Inverse Design of Urban Procedural Models

Carlos A. Vanegas
Purdue University
U.C. Berkeley

Ignacio Garcia-Dorado
Purdue University

Daniel G. Aliaga
Purdue University

Bedrich Benes
Purdue University

Paul Waddell
U.C. Berkeley

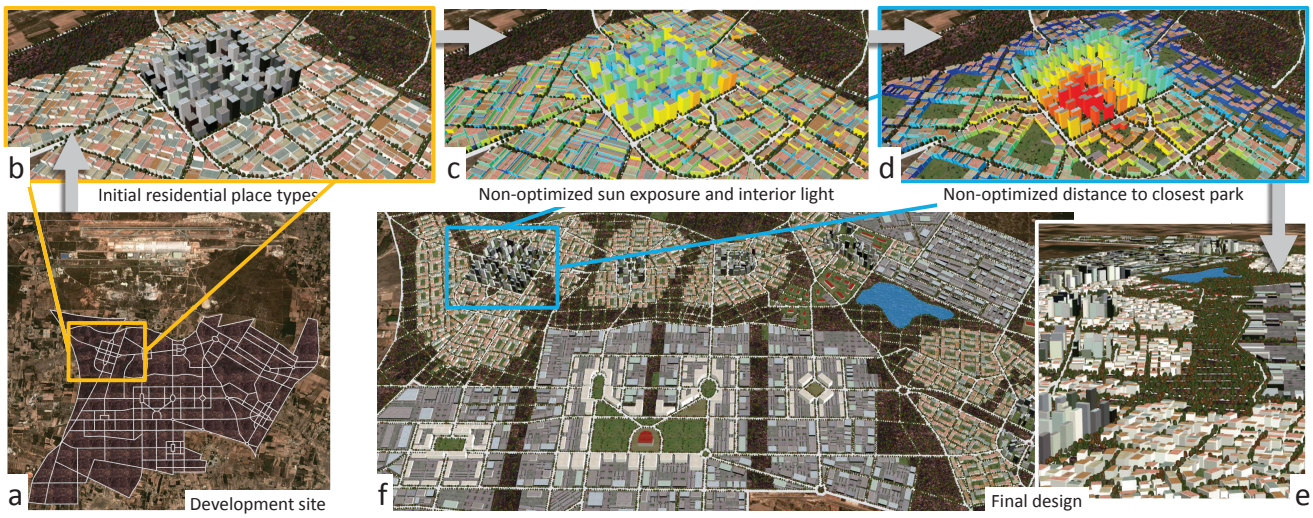

Figure 1: ***Example Urban Design Process.*** *The user interactively controls a 3D urban model by altering the parameters of an underlying procedural model (forward modeling) or by changing the values of arbitrary target indicator functions (inverse modeling). a) Starting with a development site, b) the user selects an initial urban layout (using templates or "place types"). The layout now has parcel egress but has c) initial undesired values of building sun-exposure and natural interior light, and d) unwanted average distance from buildings to closest parks. The model alterations needed to obtain user-specified local or global values for these indicator values are computed "inversely" resulting in a final design, which is e) seen up close or f) from a distance. In contrast, accomplishing such an output with traditional forward modeling would require either specifically writing a procedural model with the desired parameters or very careful parameter tuning by an expert to obtain the intended results.*

Abstract

We propose a framework that enables adding intuitive high level control to an existing urban procedural model. In particular, we provide a mechanism to interactively edit urban models, a task which is important to stakeholders in gaming, urban planning, mapping, and navigation services. Procedural modeling allows a quick creation of large complex 3D models, but controlling the output is a well-known open problem. Thus, while forward procedural modeling has thrived, in this paper we add to the arsenal an inverse modeling tool. Users, unaware of the rules of the underlying urban procedural model, can alternatively specify arbitrary target indicators to control the modeling process. The system itself will discover how to alter the parameters of the urban procedural model so as to produce the desired 3D output. We label this process inverse design.

CR Categories: I.3 [Computer Graphics] I.3.3 [Computer Graphics]: Picture/Image Generation I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.6 [Computer Graphics]: Methodology and Techniques

Keywords: interactive, procedural modeling, inverse procedural modeling, urban models

Links: DL PDF

1 Introduction

Urban procedural modeling is becoming increasingly popular in computer graphics and urban planning applications. A key basis for the popularity of city-scale urban procedural modeling is that once the procedural model is defined, it encapsulates the complex interdependencies within realistic urban spaces [Batty 2007] and enables users, who need not be aware of the internal details of the procedural model, to quickly create large complex 3D city models (e.g., [Parish and Müller 2001; Honda et al. 2004; Weber et al. 2009; Vanegas et al. 2009]). Effectively, the detail amplification inherently provided by procedural modeling is exploited: a small set of succinct *input rules* and *input parameters* can yield very complex and coherent outputs. However, the succinctness of urban pro-

cedural modeling is also its Achilles' heel: obtaining a 3D urban model with complex user requirements is a challenging task that requires experience and in-depth knowledge of the underlying procedural model. An expert user with programming skill must set the values for the input parameters, implement the procedural rules in software, and iterate between code, parameters and examination of the output in order to achieve the desired model. In short, what is needed is a means to efficiently learn the parameters and rules required to produce a desired 3D urban model, without requiring the end user to write complex software programs. Urban planners and content designers often have a clear vision for the target urban model, but lack the computational tools to rapidly create models that meet their design requirements.

To address this challenge, previous papers have proposed solutions for locally changing the 3D output of a procedural model to produce the desired model (e.g., [Lipp et al. 2011]), for stochastically driving a procedural model so as to obtain an output following a desired global shape (e.g., [Talton et al. 2011]), or for guiding procedural modeling by dividing the model into smaller parts that are easier to describe and can inter-communicate (e.g., [Beneš et al. 2011]). An alternative strategy focuses directly on the inverse modeling problem: given a desired 3D output, estimate the procedural parameters and rules needed to generate the provided output. This strategy is particularly promising, but so far has only been applied to selected elements of our problem, such as to generate 2D vector geometry [Šťava et al. 2010], to complete or edit input models exhibiting certain symmetries [Bokeloh et al. 2010], or to recreate animated sequences [Park et al. 2011].

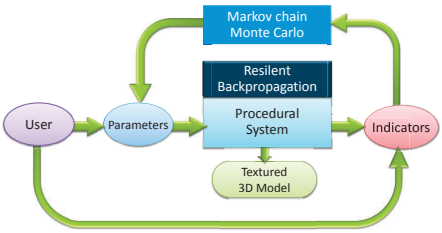

Figure 2: ***System Pipeline.*** *A summary of our coupled forward and inverse modeling pipeline.*

We propose coupling a forward procedural urban modeling process with an inverse urban modeling algorithm that optimizes the input parameters to satisfy user-specified goals (Figure 1). We are closing the loop between forward and inverse modeling strategies (as depicted in Figure 2) and provide both increased control and higher flexibility, enabling the user to be much more efficient in generating a model that satisfies their requirements. Both forward and inverse modeling strategies can be applied during an interactive editing session for either local or global model modifications. The key advantage of supporting an inverse modeling methodology is that the procedural model can be controlled in new ways without having to re-program the forward engine or explicitly re-write the procedural model rules. Our work abstracts the urban procedural model into a general parameterized form and adds a component that is able to discover how to control the procedural model so as to obtain the desired values for user-specified indicator values derived from the resulting model.

To ensure we have an expressive forward system, we have developed a versatile parameterized urban procedural modeling engine. Similar to Parish and Müller [2001] and to CityEngine [http://www.esri.com/software/cityengine], our system allows for an interactive editing at levels of detail ranging from buildings to cities. A key innovation in the design of the procedural model exploits the place types concept used by urban designers to represent a coherent design pattern of buildings and streets (e.g., [Calthorpe 2010]). Our system has operationalized the concept of place types into an interface that allows urban planners and designers to interact with the design task at a high level of abstraction from the details. Internally, we use place types as a hierarchical means of organizing the geometric elements for spaces much larger than individual buildings, and encoding these efficiently as procedures. Akin to example-based methods (e.g., [Kwatra et al. 2005; Merrell and Manocha 2008; Aliaga et al. 2008]), place types effectively provide a succinct way to assemble a complex and realistic urban space; however, unlike example-based methods, the generation process is fully parameterized.

Our *inverse system* enables users to directly alter the 3D model via a set of user-specified local or global target indicators. The concept of indicators is well used by the urban design and planning community (e.g., [Murphy 1980; Wong 2006]) and the work presented here is strongly motivated by our urban design and planning collaborators. Indicators can be thought of as an additional output layer that can vary from simple evaluation of geometric properties that can be used to alter parameters in a straightforward way (e.g., average distance of a house from the street, floor-to-area ratio) to complex semantic metrics (e.g., landmark visibility, amount of sun exposure per building, suitability of roofs for solar energy panels) that do not have an obvious relationship to input parameters. Nevertheless, indicators provide an intuitive means for the user to evaluate a 3D urban model, and are easy to compute and express. To our knowledge, indicators have not been used to control modeling because the exact relationship between the input parameters and target indicators is in general unknown, complex, and highly nonlinear.

Our inverse design strategy has two main advantages:

- *Abstraction* - we enable urban planners and designers to work on design tasks at a high level of abstraction, enabling users to manipulate place types, parameters, and indicators in order to generate and evaluate the 3D urban model interactively. This design approach enables users to focus on the design task and not be distracted with implementation details of the procedural model, thus supporting more creativity and productivity. In particular, an urban procedural model can be created/defined once by an expert and then used by others who do not need to be aware of its internal structure or of its precise response to the input parameters.

- *Interactivity* - our approach enables interactively manipulating indicator targets, while regenerating 3D geometry at high frame rates. Further, our methodology to mapping target indicators to input parameters is sophisticated enough to allow the use of complex indicators, including those for which it is not known how to reprogram the procedural model.

To enable efficiently exploring the procedural parameter space in order to find one or more solutions that produce a 3D urban model of the desired parameter and indicator behavior, we use Monte Carlo Markov Chains (MCMC) [Gilks et al. 1995] and resilient back propagation [Riedmiller and Braun 1993]. We tailor our use of MCMC and back-propagation to find several solutions that are as different as possible yet comply with the desired values. Unlike previous systems using MCMC within the context of architectural or procedural modeling (e.g., [Merrell et al. 2011; Talton et al. 2011; Yu et al. 2011]), i) we support complex indicators, thus enabling control beyond global shape, such as by high-level semantics and indicators, ii) we consider the procedural model as a black box and thus support context-free and context-sensitive stochastic grammars, and in fact impose no grammar limitation – this is advantageous for urban procedural modeling because cities are hard to express with a strict grammar and even harder using the context-free grammars of Talton et al. [2011], and iii) our system is inter-

active and able to alter models consisting of over 1500 buildings as well as geometry for roads, in only fractions of a second on a desktop computer. Our use of a resilient back-propagation engine is to imitate the indicator behavior of the urban procedural model without having to explicitly produce a 3D model and thus allow the MCMC engine to evaluate the target indicators for a large number of iterations and instances of urban models. Since there is no guarantee that a 3D model is possible for arbitrary indicator values, our system also provides an analysis tool which informs the user if a particular indicator value is feasible. Using our system we can interactively modify 3D models for urban areas spanning over 100 km^2 and containing up to 10,000 parcels. Our typical frame rate while changing indicator values interactively and generating new 3D models is 2.5 to 10 frames per second. The back-propagation engine is able to compute indicator values that are typically within 5% of using the actual procedural model and require only a small fraction of the compute time (e.g., 0.01 ms as opposed to a typical 250 ms using a procedural modeling engine, thus an effective speedup of over 25,000x). The user can choose from a proposed best set of 3D models. Subsequently, more edits can be made, the model can be saved to disk, or the geometry exported to commercial rendering engines.

2 Related Work

Procedural modeling has been used in several principal domains: plant generation, fractal noise and terrain generation, and urban modeling. The seminal Paris and Müller [2001] defines a forward procedural process for generating cities using L-systems. Vanegas et al. [2010a] and Watson et al. [2008] provide comprehensive surveys of subsequent improvements in forward-generating urban procedural modeling. Some procedural modeling approaches have incorporated domain specific knowledge in order to generate expressive, yet controlled, output as well (e.g., vegetation [Hart et al. 2003], architectural design [Whiting et al. 2009; Turrin et al. 2011; Merrell et al. 2011], and textures [Lefebvre and Poulin 2000; Bourque and Dudek 2004]). A common challenge to all these approaches is providing a succinct control from a compact set of rules.

Our core inverse engine attempts to control modeling by discovering how to alter the input parameters values in order to yield a desired set of user-defined target indicator values while treating the urban procedural model as a black box for the purpose of generality. Many classical inverse methods are based on regularization theory (e.g., [Bertero et al. 1988]). These methods attempt to *regularize* ill-posed inverse problems; e.g., the inverse problem can be defined as trying to find the underlying weights relating input parameters to target indicators. While standard regularization can improve the ability to compute a plausible solution to a generic inverse problem (e.g., Tikhonov-based regularization/regression), the typical emphasis is noise removal and/or an assumed unknown formation process of the observed indicators - neither is the case in our work.

More specific methods to inverse procedural modeling in graphics have also been published recently. For instance, Šťava et al. [2010] proposed inverse procedural modeling of rules and parameter values. However they focus on 2D content, terminal symbols are known, and they generate context-free rules for linear structures using L-systems. Bokeloh et al. [2010] explore partial symmetries and use inverse procedural modeling in order to complete the geometry of ill-specified input models exhibiting certain symmetries. Park et al. [2011] use inverse procedural modeling to find a grammar for animated sequences and use them to generate new animations. Aliaga et al. [2007] and Vanegas et al. [2010b] have focused on determining the parameter values for pre-specified classes of procedural building models. Several facade-level works have also proposed methods to determine procedural parameter values for individual facades (e.g., [Müller et al. 2007; Xiao et al. 2008]). Another related set of approaches are those by Honda et al. [2004], Weber et al. [2009], and Vanegas et al. [2009] which integrate urban simulation and urban geometric modeling. With a user-guided process, they define an inverse-like simulation to produce realistic 3D urban models. Most related to our work are other MCMC/Metropolis-based methods and guided procedural modeling of Beneš et al. [2011]. MCMC-based methods have been used in many vision and robotics applications (e.g., see work in [Dellaert 2003]). MCMC and Metropolis-based sampling has been used as well in computer graphics (e.g., light transport [Veach and Guibas 1997], rigid-body simulations [Chenney and Forsyth 2000], and facade modeling [Alegre and Dellaert 2004]). Recently, MCMC and stochastic optimization were also used for furniture layout distribution by searching in a parameterized design space (e.g., [Merrell et al. 2011; Yu et al. 2011]). Beneš et al. [2011] generalizes the concept of environment by spatially dividing the rules (and productions) into small guided procedural models that can communicate by parameter exchange. Talton et al. [2011] take as input a stochastic context-free grammar and explore the possible strings generated by the grammar to find a production sequence that yields a target 3D shape. They render/voxelize the model to evaluate a cost function, yielding a multi-building example result in about 14 minutes. In contrast, our method does not impose any limitations on the underlying grammar and enables editing urban areas of over a 1000 buildings in only a fraction of a second.

3 Overview

As shown in Figure 2, a procedural system P produces a geometry G using a set $\Phi = \{\phi_1, \ldots, \phi_m\}$ of m input parameter values. The geometry is evaluated by an indicator measurement system I which produces a set $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ of n indicator values. Each parameter value must be within a range $[\phi_{min_i}, \phi_{\max_i}]$ and each indicator value within a range $[\gamma_{min_i}, \gamma_{\max_i}]$. Symbolically, we can write the relationship of the input parameters to the target indicators:

$$\Gamma = I(G) = I(P(\Phi)) = (I \circ P)(\Phi) \quad (1)$$

Our objective is to let the user change the parameter values Φ (i.e., forward modeling) or the indicator values Γ (i.e., inverse modeling) and then to automatically (and interactively) compute geometry G whose indicator values are within the desired range. We define the set of tuples $\Gamma^* = \left\{(\gamma_1^*, \gamma_{\sigma_1}^*), \ldots, (\gamma_n^*, \phi_{\gamma_n}^*)\right\}$ to be the user-specified target indicator values, such that measurements of each indicator are desired to have an approximate Gaussian distribution with mean γ_i^* and standard deviation $\gamma_{\sigma_i}^*$. Similarly, we define the target parameter values to be $\Phi^* = \{(\phi_1^*, \phi_{\sigma_1}^*), \ldots, (\phi_n^*, \phi_{\sigma_n}^*)\}$. However, the specification of Φ^* is optional – if not specified, the parameters can be freely altered by the system. While the standard task of procedural modeling is to generate geometry from a set of input parameters, our methodology computes a set of parameter values Φ' that leads the procedural system to generate a geometric model G exhibiting indicators close to the target values Γ^* and, optionally, close to parameter values Φ^*. In other words, we seek a set of parameter values Φ' such that $(I \circ P)(\Phi') \rightarrow \Gamma^*$ while possibly also ensuring $\Phi' \rightarrow \Phi^*$. The overall error is encapsulated in the function $E(\Phi', \Phi^*, \Gamma^*)$ and the computational goal using MCMC is to find the values of Φ' that minimize E.

Our inverse method requires calculating $(I \circ P)(\Phi)$ for a large number (e.g., $O\left(10^4\right)$) of different values for Φ. Since standard procedural systems are unlikely to generate such a number of productions at interactive rates, we require a function $f(\Phi)$ that approximates $(I \circ P)$ and can be computed significantly faster. The function does not need to produce geometry but just the correspond-

ing indicator values. We use a resilient back propagation engine to define a function that outputs the set $\widehat{\Gamma} = \{\widehat{\gamma_1}, \ldots, \widehat{\gamma_n}\}$ of values that approximate the indicators. The function f is a good approximation of $(I \circ P)$ if $\left|\Gamma - \widehat{\Gamma}\right| < \epsilon$, where ϵ is a small positive constant. Thus, given a good approximation f of the procedural generation and indicator measurement system $(I \circ P)$, our optimization goal is to find sets of parameter values Φ' such that $f(\Phi') \rightarrow \Gamma^*$ (and optionally $\Phi' \rightarrow \Phi$).

4 Inverse Design

Given the sets Γ^* and Φ^*, our optimization explores the large multimodal space of parameters in order to propose one or more sets Φ'. In the following, we describe MCMC in the context of our pipeline including how both local and global moves are made, how we select amongst the best solutions so as to encourage diversity in the answers, and summarize our resilient back propagation engine.

4.1 MCMC-based Parameter Searching

The optimization consists in simultaneously running many sets of Markov Chains over a large number of iterations and choosing the best solution states (Figure 3). Our MCMC-based optimization uses the Metropolis-Hasting algorithm [Metropolis et al. 1953], [Hastings 1970] to seek Φ' by performing n_I attempted state changes $\Phi_t \rightarrow \Phi_{t+1}$ starting from n_P different initial parameter sets and using n_β different temperatures - each temperature corresponds to a different magnitude of incremental change per iteration. Next, we describe the computation per parameter (or per indicator) and for brevity we drop the i subindex from ϕ_i (or γ_i).

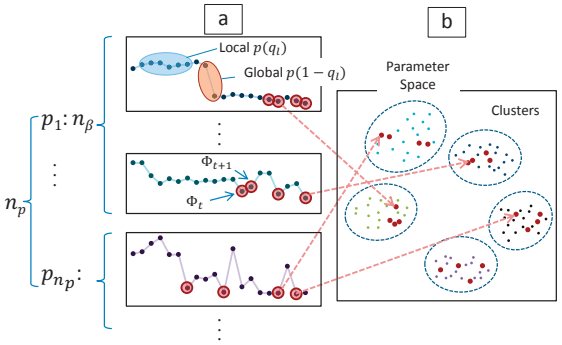

Figure 3: ***Parameter Searching.*** *We provide a diagrammatic summary of our parameter searching process. Given an initial set of n_p parameters, a) we simultaneously pursue many Markov chains for each of n_β different temperatures. Within each chain both local and global moves are performed. b) At the end, the best found solutions from all chains are clustered and filtered. Then, one solution from each cluster is shown to the user in order to increase variability in the computed 3D models.*

4.1.1 Error Function

Given a configuration Φ_t and a set of indicators $\Gamma = f(\Phi)$, the optimization error function $E(\Phi_t, \Phi^*, \Gamma^*)$ takes into consideration the proximity of $f(\Phi_t)$ to Γ^* and (optionally) the proximity of Φ_t to Φ^*. The function can be written as

$$E(\Phi_t, \Phi^*, \Gamma^*) = w_\Gamma \frac{1}{n} \sum_{\gamma \in \Gamma} \left(\frac{|\gamma - \gamma^*|}{\gamma_w^*} \right) + w_\Phi \frac{1}{m} \sum_{\phi \in \Phi} \left(\frac{|\phi - \phi^*|}{\phi_w^*} \right)$$

The user interactively sets the values of the weights $w_\Gamma \in [0, 1]$ and $w_\Phi = (1 - w_\Gamma)$ to control the tradeoff of complying with the target indicators or with the preferred parameter values, respectively.

4.1.2 Starting Points

A chain's starting point is chosen by a sampling process. The target value for a parameter may be specified as an interval with mean $\phi^* \in \Re$ and with standard deviation ϕ_σ^*. The values w_Φ and w_Γ used in the above error function are now used as probabilities (and are already in the range [0,1]). Chain starting points are chosen i) with probability w_Φ to be in the sampling neighborhood of the specified target parameter value ϕ^* (i.e., within the Gaussian distribution $N(\phi^*, \phi_\sigma^*/2)$ for each $\phi \in \Phi$ and bound by $[\phi_{\min}, \phi_{\max}]$) and ii) with probability w_Γ to be uniformly sampled within its extent $[\phi_{\min}, \phi_{\max}]$. If the parameters can have any valid value, then $w_\Phi = 0$ which implies the right hand term of equation 2 is unused and starting points are obtained by a uniform sampling of the entire parameter space.

Each chain performs, per attempted state change, either a local or a global move with probability q_l and $(1 - q_l)$, respectively. Local moves are proposed by sampling from Gaussian distributions for each parameter ϕ. Global moves are proposed by sampling points using precomputed frequency distributions of the indicators. All local and global moves are bounded by the predetermined span for each parameter in order to guarantee that the generated solutions remain within feasible limits.

4.1.3 Local Moves

Given the current state Φ_t, a candidate state change is computed by sampling for each parameter $\phi \in \Phi$ a value from the Gaussian distribution $N(0, \sigma_\phi)$, where $\sigma_\phi = \min\left(\frac{\phi_w^*}{2}, \alpha\left|\phi_{\min} - \phi_{\max}\right|\right)$, and typically $\alpha = 0.05$ in our system. Then, since the distribution of proposed moves is symmetric (i.e., it consists of a sum of weighted Gaussians – thus, the probability of moving from state t to $t+1$ is the same as from state $t+1$ to t), the acceptance probability a of a move from a current state Φ_t to a candidate state $\widetilde{\Phi}_{t+1}$ is given by the Metropolis ratio:

$$a(\Phi_t \rightarrow \widetilde{\Phi}_{t+1}) = \min\left(1, \frac{\exp\left(-\beta E\left(\widetilde{\Phi}_{t+1}\right)\right)}{\exp\left(-\beta E\left(\Phi_t\right)\right)}\right) \qquad (2)$$

where β is the chain's temperature. Hence, $\Phi_{t+1} = \widetilde{\Phi}_{t+1}$ with probability a, and $\Phi_{t+1} = \Phi_t$ with probability $(1 - a)$.

4.1.4 Global Moves

Global moves are computed by inspecting frequency distributions of the indicators. The frequency distribution for each indicator γ is computed after any user change of Γ^* as follows: (i) the parameter space is uniformly sampled within the extent of each parameter using many samples (e.g., 10^5); (ii) the value of the indicator γ is computed for each sample using f; (iii) the ranges $[\gamma_{\min}, \gamma_{\max}]$ of γ are computed from the evaluations of f and uniformly divided into n_f bins; and (iv) a frequency histogram $H_k(\gamma)$ for $k = [1, n_f]$ is computed which stores in each of the bins both the frequency (i.e., the number of sampled parameter sets that fall within the bin) and the sampled parameter sets themselves.

To perform a global move, we randomly select an indicator γ, define $b_t \in [1, n_f]$ as the index of the bin containing the target indicator value, and randomly choose one of the sets Φ from the bin $H_\rho(\gamma)$ where ρ is chosen with probability h_m as being from the bin with the highest frequency (e.g., most solutions) and with probability $(1 - h_m)$ as being from the bin containing the target indicator

value. Symbolically:

$$\rho = \begin{cases} b_m : & H_m = \max_k \left(H_k\left(\gamma\right)\right), k \in [1, n_f] \text{ with } p\left(h_m\right) \\ b_t : & \text{with } p\left(1 - h_m\right) \end{cases} \tag{3}$$

In contrast to a random local move in the parameter space, this approach for global moves tends to lead the chain towards distant configurations that are more likely to reduce the error function for at least one indicator.

4.2 Solution Selection: Ensuring Variability

The process described in the previous section explores $n_\beta * n_P * n_I$ parameter configurations from which only a few are eventually shown to the user. The selection is done by choosing the top solutions for each temperature value, clustering the resulting configurations to guarantee variability in the solutions, and filtering out solutions for which the fit between the estimated indicator values (i.e., using the back-propagation engine) and the actual values (i.e., using the procedural engine) is low.

In our implementation, for each temperature we choose among all the configurations generated the 25 solution states that returned the smallest error E, thus narrowing down the number of possible solutions to $25n_\beta$. In order to increase the variability in the solutions shown to the user (i.e., to obtain representative solutions from different regions of the parameter space), we use k-means clustering to partition the set of solutions. The parameter values of each solution are normalized before the distance between two points in parameter space is computed by the clustering algorithm. The value k is determined by the user (e.g., it equals the number of solutions that the user wants to see).

The top solutions best minimize the error expression $|(I \circ P)(\Phi') - \Gamma^*|$. Notice that this error term uses the actual indicator values computed by the procedural and indicator measurement system, as opposed to the estimated indicator values. Effectively, we use the approximation function $f(\Phi)$ to very quickly sample the parameter space and to select a set of several dozen solutions that minimize the indicator error - hence the approximation function does not need to be extremely precise. The resulting solution set is small enough to have the system interactively compute the actual indicators, thus reducing the effect of the approximation error of f.

4.3 Resilient Back Propagation Engine

Given the complexity of the procedural engine and the thousands of experiments MCMC requires per small change, we obtain interactivity by replacing the procedural and indicator measurement system with a neural network. This choice is based on the fact that neural networks are trained universal functions (e.g., they have been used successfully to replace many kinds of functions: linear/non-linear functions even periodic, exponential, and piecewise continuous) and can quickly estimate indicator measurements without generating the 3D model and explicitly evaluating the indicator functions.

We implement a multilayer feed-forward network (i.e., information moves in only one direction: from parameters to indicators). The network is set up to have m neurons in the first layer and n neurons in the last layer with the number of layers typically being $m-n+1$. Each layer is fully connected to the next layer. When evaluating the learned function, the weight stored in each neuron is applied to either a sigmoid function or a Gaussian function. The back propagation supervised learning algorithm trains weights starting with the last layer. Each weight w_{ij} is updated by

$$w_{ij}(t+1) = w_{ij}(t) - l_r \frac{\partial\left(I \circ P\left(\Phi\right)\right)}{\partial w_{ij}}\left(t\right) \tag{4}$$

where w_{ij}'s first index (i) is for a weight in the current layer and the second index (j) is for a weight in the next layer closer to the input parameters, t is the iteration index, l_r is the learning rate, and the partial derivative of $I \circ P\left(\Phi\right)$ is estimated by finite differences. We implemented an algorithm that searches for the optimal configuration of a multilayer feed-forward network by altering the number of neurons per layer and the neuron function per neuron (e.g., sigmoid or Gaussian). We found the aforementioned configuration, with Gaussians, to yield near optimal results.

Our training process uses as input a set of parameters (e.g., those the user allows the system to change), their valid ranges, and an initial urban scenario. The parameters are sampled within those ranges using a normal distribution and the corresponding indicator values are calculated. Both are then used to train the neural network. The number of samples to do the training depends on the complexity of the urban scenario; however we found empirically that using 200-500 samples was enough. It is worth noting that the neural network is trained for a specific scenario. However global indicators (i.e., those that do not depend directly on the existence of different place types) do not require any re-training. Local indicators (e.g., landmark visibility) do need to be retrained when major city changes are made. Nevertheless, in all of our shown examples, we train the neural network only once.

4.4 Feasibility

Finding a set of parameter values able to achieve an arbitrarily specified set of indicator values is not always possible. The existence of a geometrical model G that satisfies both the parameter and indicator target values depends on the expressivity and controllability of the procedural model. Thus, while we attempt to provide a flexible procedural model (see Section 5), there is no guarantee of invertability. Nevertheless, we do provide an interactive analysis tool based on the histograms $H_k(\gamma)$ for $k = [1, n_f]$ of Section 4.1.4. The histogram informs the users how feasible are particular ranges of indicator values; e.g., if all bins are near empty, the indicator cannot be generally achieved for the given procedural model.

5 Urban Procedural Model

We have implemented an urban procedural engine similar to previous city-level procedural modeling work (e.g., [Parish and Müller 2001; Weber et al. 2009; Vanegas et al. 2009]). However, we provide a broad range of urban geometrical configurations with a reasonable degree of succinctness and high-level control. Our procedural engine is inspired by urban planners. For instance, our place-type categories and initial parameter values were obtained with the assistance of our urban planning collaborators. In the following text we summarize the structure of our procedural engine, list the parameters, and describe our initial set of indicators.

5.1 Procedural Model

A key concept used in urban planning and modeling centers (e.g., [TRANSECT 2012; CNU 2012; CTOD 2012]) are place-types (e.g., as described in SmartMobility [2010]). A city consists of several instances of one or more place-type categories. All instances of the same place-type category are regions – ranging from a few blocks to an entire neighborhood – that have contained roads, parcels, parks, and buildings with similar geometric attributes (e.g.,

width, height, density, shape outlines). Similar to the urban layout editor of [Lipp et al. 2011], place-types allow defining, moving, rotating, and resizing large subsets of the city at once and can be used to very quickly produce a sketch of the urban model. The underlying road network, subdivision into parcels, placement of parks, and definition of building envelopes per place-type instance is generated with a fully parameterized approach.

The interactive session consists in the user sketching the global configuration of the urban area and then directly changing m parameter values or n indicator values. First, an arbitrary shape is defined to separate the modeled urban area from surrounding water and terrain bodies. Second, the user creates $z > 0$ place-type instances of one or more categories, each ranging in size from a few blocks to an entire city. In our implementation, we define a place-type category as a template that provides specific (initial) values for parameters used by the procedural modeling engine. Once an instance of a place-type is positioned by the user, the geometry of the contained roads, parcels, and building is automatically created and joined with the neighboring geometry. While land-use and zoning regulations are not explicitly enforced, they can be indirectly maintained by careful selection, sizing, and placement of place type instances. The expandable set of place-type categories supported by our current implementation includes regional/town/suburban center, low/medium/high-density industrial, and residential, retail, park, and institutional areas. Third, the actual city model is altered by our inverse design based on user-specified changes to input parameter or target indicator values, performed individually per place-type or done collectively for groups of place-types.

5.2 Parameters

The entire 3D urban model has $m = zm_p$ parameters controlling its generation, with $m_p = 16$ being the number of per place-type instance parameters. The per-place type parameters generate a road network with two levels of street hierarchy (i.e., arterials and local), extract city blocks from the road network, subdivide the resulting blocks into parcels, define parks, and instantiate a 3D building envelope inside each parcel (Figure 4).

Roads within a place-type instance are generated outwards from an initial seed location. The initial radially-outward direction is called the u direction and its perpendicular the v direction. The following road parameter values are specified separately i) for arterial roads and ii) for local roads, both within a single place-type instance:

- r_{u_d}, r_{v_d} : distance between two adjacent intersections in u-direction and in v-direction,
- r_i : maximum road length irregularity randomly added to a road segment between two adjacent interactions,
- r_c : maximum random rotation (or curving angle) of a road segment when it passes through an intersection,
- r_w : road width, and
- r_n : number of departing radial streets from initial seed.

Parcels are the result of subdividing an area enclosed by roads using recursive subdivision of oriented bounding boxes (OBB) - similar to that of Parish and Müller [2001] or Vanegas et al. [2009]. The parcel parameters values are:

- p_m, p_σ : mean and standard deviation of the randomly-determined target parcel areas during OBB subdivision,
- p_s : maximum random offset of the OBB subdivision split line from the center of the block, and

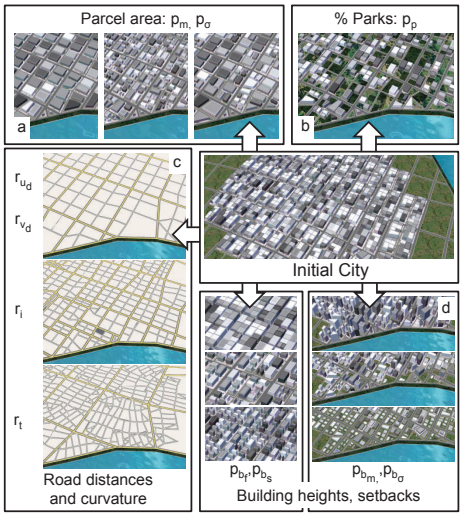

Figure 4: ***Urban Procedural Parameters.*** *The set defining an initial city model includes the following parameters which define constraints or control underlying stochastic processes: a) target parcel area range, b) percentage of parks, c) desired distance between road intersections and intersection angle parameters, and d) building height and setback guidelines.*

- p_p : percentage of parcels randomly selected as parks.

Building envelopes are controlled by the following values:

- p_{b_m}, p_{b_ρ} : mean and standard deviation of height,
- p_{b_f}, p_{b_s} : front and side setback distance from road, and
- p_{b_w}, p_{b_d} : maximum front width and maximum depth.

5.3 Indicators

To demonstrate inverse design support, the entire urban model has $n = zn_t$ indicators with $n_t = 7$ indicators per place-type being divided into three classes of indicators that range from straightforward measurements to purposefully high-level abstractions (Figure 5).

Intrinsic indicators measure attributes that are mostly independent from all other buildings and parcels:

- i_{FAR} : measures the concept of floor-to-area ratio which is often used in urban planning.

Distance-based indicators are mostly concerned with accessibility measurements and involve computing the distance between two entities:

- i_p : distance from a parcel center to the closest park, and
- i_x : distance from a parcel center to a user-specified location "x" (e.g., town center, a landmark, etc.).

Visibility-based indicators involve the 3D geometry of the city and typically perform visibility/occlusion calculations. For instance, the below indicators can be used to quantify the appropriateness of a city model for reduced energy consumption and/or for solar panels. Our implemented indicators are:

- i_L : percentage of buildings (inside the place-type) where at least one point on one landmark is visible from at least one point on the facade of the building, and

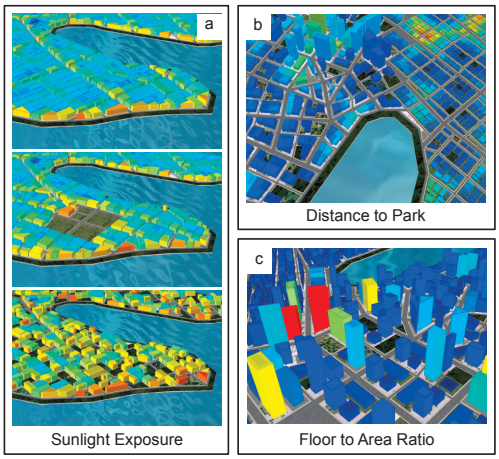

Figure 5: ***Urban Indicators.*** *We demonstrate a variety of diverse urban indicators, including: a) by measuring sunlight exposure per faade, we show city models with increasing exposure (bottom result has most exposure), b) we show a color-coded distance to park metric (red implies close to park), and c) we show the floor-to-area ratio of several buildings.*

- i_R, i_F : percentage of time during the course of a day, averaged over all days of the year and for a specified latitude/longitude, that rooftops or facades, respectively, are directly exposed to the sun (i.e., self-occlusion and occlusions by other building structures are considered).

- i_I : the natural interior light ratio, obtained from dividing the sun exposure of the facades by the average minimum distance from an interior building point to a faade.

6 Implementation Details

Our system runs on a desktop PC equipped with Intel Xeon clocked at 3.53GHz, Windows 7, and a NVIDIA GTX 580 graphics card. Our implementation is single core without GPU acceleration, yet it provides interactive feedback for all examples in the paper. The frame rate while editing indicator values was from 2.5 to 10 frames per second and during parameter editing was from 15 to 30 frames per second. Back-propagation re-training, which is only needed whenever new parameters/rules/indicators are defined, takes about 4 minutes for a training set of 1000 examples. Both MCMC processing for indicator editing and back-propagation re-training can be easily parallelized to multiple cores. Our prototype uses OpenCV for k-means clustering and for an implementation of the back-propagation engine. We set the number of temperatures $n_\beta = 4$, the number of iterations $n_I = 5000$, and the number of starting points $n_P = 20$. Further, we set $q_l = 0.9$ which implies a 10% probability of performing global state changes. For global state changes, the number of bins per indicator range is $n_f = 10$ and the probability of choosing the bin with the maximum number of sampled parameter space points is $h_m = 0.9$.

Models can be exported to CityEngine [http://www.esri.com/ software/cityengine], for example. The road and parcel network, including parameter values, are saved to OSM and OBJ files. By using multiple CityEngine rule files, together with some randomized effects, we can quickly create novel and compelling urban models. This was done for Figures 10, 11b-d, and 12e.

7 Results and Discussion

We have used our framework to create and edit a variety of city-scale 3D models. All editing and rendering is done interactively using sliders to alter parameter values (forward modeling) or indicator values (inverse modeling). All example sessions were completed in under 5 minutes and most took less than one minute. In the following, we show several analysis and case-study results.

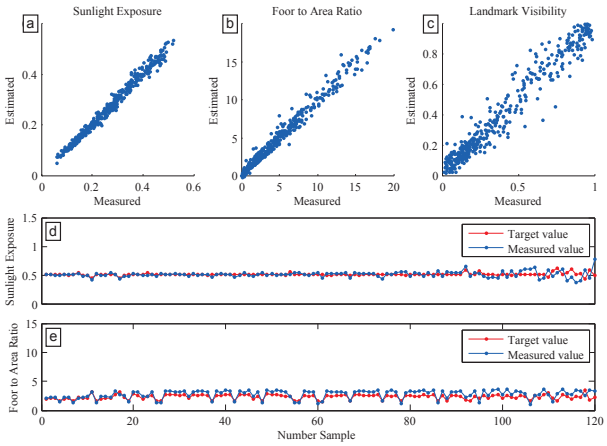

Figure 6: ***Back Propagation.*** *a-c) Error plots for three indicators comparing measured to estimated indicator values. d-e) Comparison of top 120 target and measured indicator values.*

7.1 Analysis

Figures 6-8 show results from several analyses. Figure 6 presents a visualization of the fitting error between the indicators values estimated by back propagation and the values measured by using the actual procedural model. Figures 6a-c show error plots for three indicators in an example city: sun exposure, floor-to-area ratio, and landmark visibility. The x-axis shows the measured indicator values and the y-axis shows the back propagation estimated indicator values. The graph plots 200 sampled parameter value vectors different than the 200 samples used to train the back propagation engine. Figures 6d-e show the target and measured values for two indicators using the top 120 solutions of the inverse computation. The samples are sorted by increasing fitting error: 90% of the samples have a fitting error below 10%. Solutions with a fitting error higher than 15% are not shown to the user.

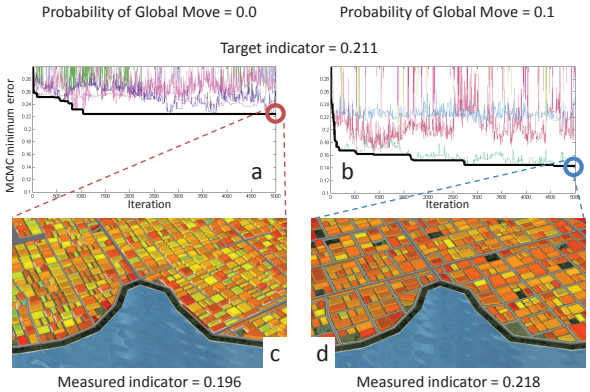

Figure 7: ***Global vs. Local Moves.*** *a) Not including global moves during optimization results in more error than b) including global moves over the same number of iterations.*

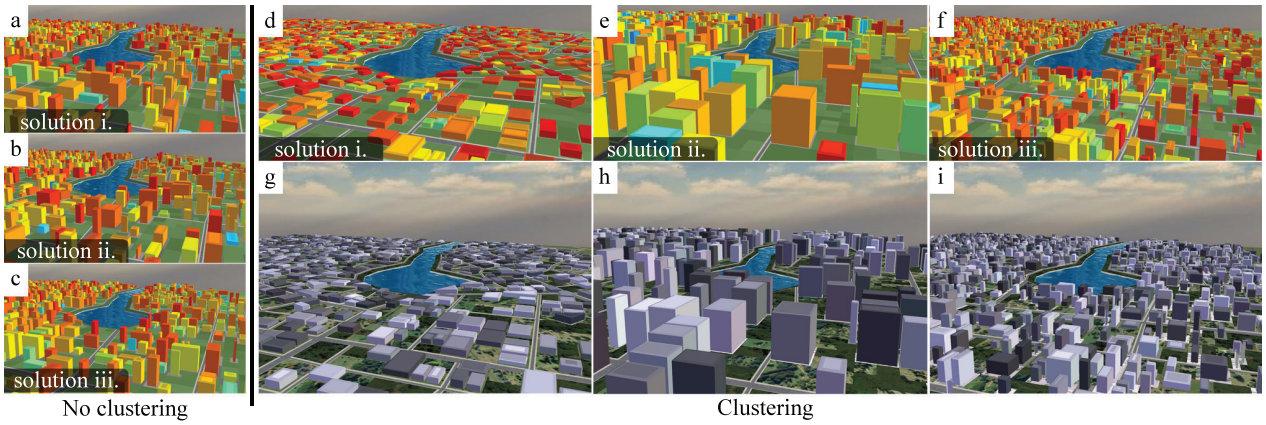

Figure 8: ***Variability.*** *a-c) We show the three top solutions generated by our system, with variability disabled, for a desired sun exposure indicator value. d-f) Next, we enable our solution to increase solution variability and obtain three clearly non-similar top solutions. g-i) Show the corresponding models of d-f but using our interactive rendering engine.*

Altogether, our back propagation engine is a good approximation of our procedural generation and indicator measurement system. However, the accuracy of the local indicator approximation could decrease with the number of input parameters due to the parameter interdependency and the stochastic nature of the indicator. To partially mitigate this effect, the number of hidden layers and the number of training samples should be increased but it will fail when the indicator function cannot be expressed through a neural network. However, the accuracy of the local indicator approximation could decrease as the number of input parameters increases because of the potential additional parameter interdependency and system complexity. To partially mitigate this effect, the number of hidden layers and the number of training samples could be increased but it will fail when the indicator function cannot be expressed sufficiently accurately. Figure 7 demonstrates the advantage of including global moves, as well as local moves, while searching through the parameter space. Figures 7a-b show the value of $E(\Phi_t, \Phi^*, \Gamma^*)$ for one randomly selected Markov chain running at each temperature β. The best solution states found so far are shown as a thick black line. These results demonstrate that our approach for including global moves yields faster convergence and avoids converging to undesired local minima. In this example, MCMC exploration with global moves resulted in a measured sun exposure value (0.218) that was closer to the target value (0.211) than when no global moves were used (0.196). The resulting 3D models are shown in Figures 7c-d. The model found by using global moves achieves higher sun exposure by using larger building setbacks and smaller building heights.

Figure 8 shows the best solutions found by our system with and without enforcing variability (Section 4.2). The user required the city to exhibit a target average sun exposure value of 0.3. When variability is not enforced, three solutions are generated with measured indicator values close to the target values (Figure 8a-c). The solutions have similar parameter values and hence produce 3D models with comparable styles (i.e., all are found in the same parameter space neighborhood). When variability is enforced, three solutions are computed with measured indicator values still close to the target values, but now show different styles (Figures 8d-i). These three solutions come from different neighborhoods (i.e., clusters) in the parameter space. The main advantage of enforcing variability is that the user can be presented with several solutions all of which closely match the target indicator values but also exhibit different urban configurations. The user can then choose amongst the top solutions using more subjective style preferences.

7.2 Example Designs

Figures 1, 9, and 10 show the results from several example design sessions using our system. A design session consists of several interactive iterations of forward and inverse editing of an urban layout that yields a model satisfying the intended goals. Figures 1 and 9 contain an example built upon our interactions with a well-known architectural and urban design company, Calthorpe Associates. In this experiment, the goal was to design a new technology park for Bangalore, India, which includes high-tech industrial buildings of various densities, campuses for institutional buildings, and five residential clusters, each containing high- and medium-density buildings. In our shown example, we focus on one of the residential clusters. Similar design processes were followed for the other areas. Each residential cluster is desired to have a prescribed amount of sun-exposure per building, a minimum of natural interior light, and a small distance from residential buildings to public parks. While such a set of goals could be achieved by a manually-created 3D model or by a customized procedural model, our system uses the ability to support arbitrary indicators, to facilitate quick editing, and to produce a model of the intended properties. First, the user sets up an initial model using place types (Figure 1a-b). The initial spatially-varying indicator values for sun-exposure and natural interior light are shown in Figures 9a and 9c (as well Figure 1c). The system automatically determines an alternative 3D model that has parcel egress, meets the desired sun-exposure and interior lighting values, and best follows the intended place-type constraints (Figures 9b and 9d). In Figure 9e (as well Figure 1d), the initial value of the distances to closest park indicator is represented. To show system flexibility, we reduce building-to-park distance using inverse indicator based modeling (Figure 9f) or forward parameter modeling (Figure 9g). Finally, Figures 1e and 1f have views of the newly produced 3D urban model. Figure 10 shows a content-design example. The modeler is seeking to alter the amount of building-produced shadows (e.g., a low value of the sun exposure indicator). Instead of the user having to comprehend the subtle interdependencies of the needed parameter changes (shown in the insets in Figure 10), we enable using a single slider to produce three results: high shadowing (Figure 10a), medium shadowing (Figure 10b), and low shadowing (Figure 10c). Our system automatically learns how to alter the parameters in order to obtain the intended result.

7.3 Urban Model Interdependencies

Figures 11 and 12 focus on showing editing flexibility whereby hypothetical urban design changes are constrained to certain ar-

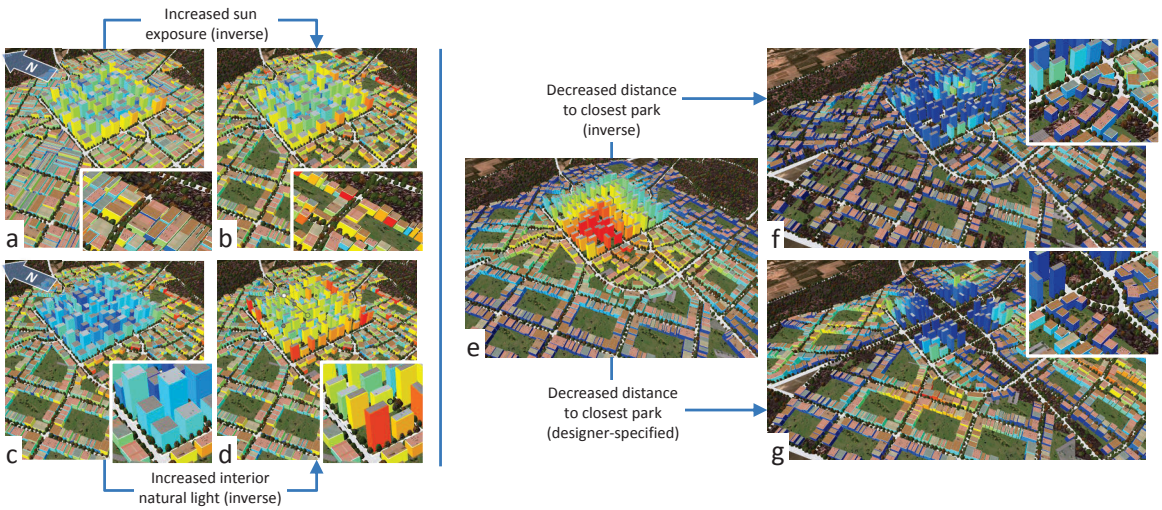

Figure 9: ***Urban Design Example (Extended).*** *We show additional imagery for the urban design session of Figure 1. The sequences (a-b) and (c-d) show the before and after of our system automatically improving the sun exposure or interior natural light of the development site of Figure 1a-b. Next, the sites' average distance to closest park is reduced either with (f) automatic inverse modeling based on the distance to park indicator or with (g) several manual forward modeling edits. The final model is shown in Figures 1e-f.*

Figure 10: ***Content Design Example.*** *We show a content design example producing 3 results: a) high shadowing, b) medium shadowing, and c) low shadowing. Instead of the user having to determine how to alter the procedural parameters (shown in the insets), the system automatically learns how to alter the parameters to obtain the intended result.*

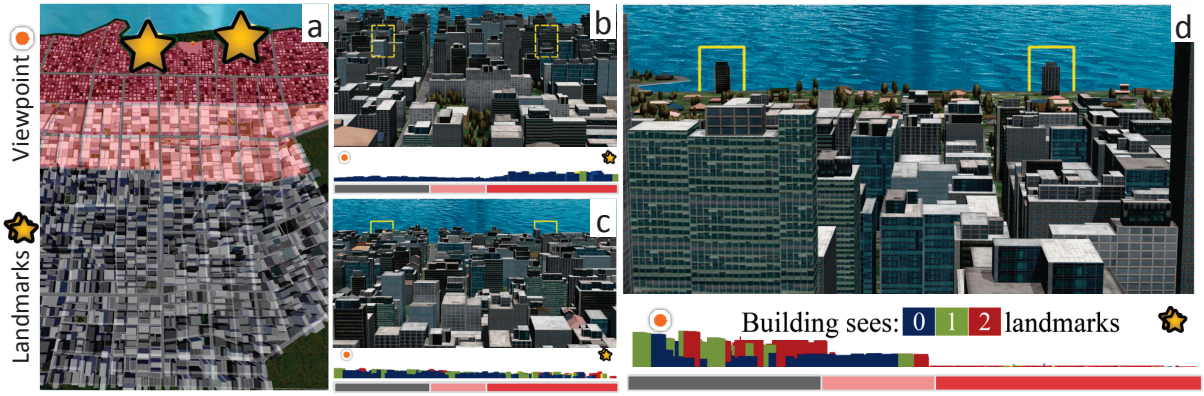

Figure 11: ***Global Indicator Control.*** *This example focuses on a global landmark visibility indicator. a) Top-down view of the city model and user-selected landmarks. b) Initial 3D city model configuration where the landmarks are not visible from most buildings (yellow boxes). c-d) User increases the desired amount of landmark visibility and the system interactively alters the city model. Below images b-d is a color coded profile of the city showing how many landmarks are visible.*

eas and/or editing in one part of a city can affect another (distant) part. Figure 11 shows a city for which our system computes new parameter values for the entire (multi-place-type) city so that a desired indicator value is met. The indicator of interest is the percentage of buildings from which at least one of two landmarks is visible. This example exhibits interdependencies between place-type instances and requires that several place-types be changed in order to achieve the desired outcome (in Figure 11a, each place type is color-coded). The user specified three different target visibility values for the city: 15%, 30% and 75%, for which configurations with 12%, 27% and 78% were computed by our system. The system automatically modified mostly the mean and standard deviation of the

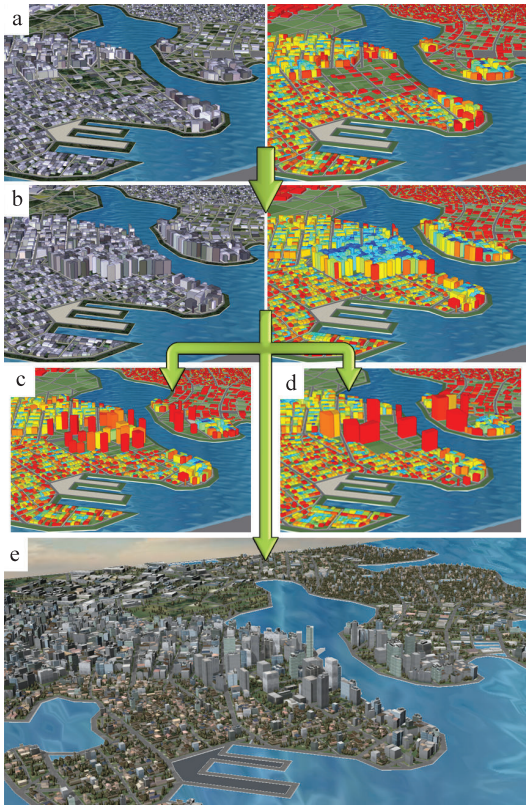

Figure 12: ***Local Changes for Global Indicator Control.*** *a) One of the nine neighborhoods of a city is redeveloped so that the average floor-area ratio of the entire city increases. b) The system proposes a solution that satisfies the target floor-area ratio but that reduces the sun exposure of the area. The user then requires the system to find a solution that maintains a high sun exposure. Three different solutions are produced (c, d, e) that exhibit different styles but satisfy the constraints on both indicator values.*

building heights and the percentage of parks in each one of the three place-type instances. In particular, low visibility was achieved by increasing the height of the buildings near the coast and assigning low heights to the buildings in the inland place-types. The system produced a high landmark visibility configuration by assigning low building heights (4 floors) to the place-type instance near the coast, mid-rise buildings (14 floors) to the middle place-type, and high-rise buildings (31 floors) to the most inland place-type.

Figure 12 shows the complementary case of Figure 11 where a user wishes to redevelop one of the nine place-type instances of an existing city (Figure 12a) so that the overall indicators of the city achieve the target goal, in this case, of bringing the average floor-to-area ratio of the entire city to 5.2 in order to meet new local development needs. The system finds a solution yielding an overall floor-to-area ratio of 5.4, but the user then realizes that the building sun exposure of the modified place-type is now too low. As a result, the user sets a target interval for the sun exposure value, and new solutions are presented that remain close to the target floor-to-area ratio while also keeping sun exposure within the specified interval (Figure 12b-c). Among these solutions, the user chooses one based on style preferences (Figure 12d) and exports it to CityEngine (Figure 12e).

In general, these figures demonstrate the use of our approach to automatically compute the parameters of a procedural urban model, either for a selected local area or for the entire city, producing a 3D model such that all or part of the model exhibits the desired indicator values. In practice, supporting local controllability of both parameters and indicators is essential to make our approach useful to content designers and urban planners.

8 Conclusions and Future Work

We have coupled an automatic inverse design approach for urban procedural modeling with forward procedural modeling. Urban indicators are intuitive metrics for measuring the desirability of urban areas, and we have incorporated this as a key method for designers to efficiently generate optimized 3D urban models that meet their target criteria. The relationship of indicators to the procedural model is in general unknown and complex which has until now hindered their direct specification. We tackle the well-known open problem of controlling procedural modeling by providing a generalized mechanism that allows users to specify arbitrary target indicators and automatically compute the optimal parameters to obtain the desired output. Our methodology uses MCMC and back propagation, including algorithms to search both local and global state changes, and presents multiple distinct 3D model options to the user.

For our current framework, we have identified several limitations and future work items. First, our method explores a parameter space of roughly the same size as other recent MCMC-based methods in computer graphics. However, the accuracy of our back propagation engine decreases as the number of parameters increases (e.g., when simultaneously optimizing for a large number of place-type instances), especially when a large number of dependent parameters are included. While we could replace the back propagation with the procedural engine itself, performance would be severely affected. We will explore alternative means to support scaling to a much larger number of parameters. Second, we will explore additional indicators, including feeding indicator values back to the model so as to, for instance, alter window sizes and wall materials based on the result of sun light exposure. Third, although applied to the concept of urban procedural modeling, there is virtually no limit to generalize the concept of target indicators to other procedural models such as buildings, trees, and furniture and to use our inverse modeling approach for these models as well.

Acknowledgements

We are grateful to our sponsors who enabled this work including NSF IIS 0964302, NSF OCI 0753116, and a Google Research gift.

References

ALEGRE, F., AND DELLAERT, F. 2004. A probabilistic approach to the semantic interpretation of building facades. *Workshop on Vision Techniques Applied to the Rehabilitation of City Centres*.

ALIAGA, D. G., ROSEN, P. A., AND BEKINS, D. R. 2007. Style grammars for interactive visualization of architecture. *IEEE Trans. on Visualization and Comp. Graph. 13*, 4 (July), 786–797.

ALIAGA, D. G., VANEGAS, C. A., AND BENEŠ, B. 2008. Interactive example-based urban layout synthesis. *ACM Trans. Graph. 27*, 5 (Dec.), 160:1–160:10.

BATTY, M. 2007. *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*. MIT Press.

BENEŠ, B., ŠŤAVA, O., MĚCH, R., AND MILLER, G. 2011. Guided Procedural Modeling. *Comp. Graph. Forum*, 325–334.

BERTERO, M., POGGIO, T., AND TORRE, V. 1988. Ill-posed problems in early vision. *Proceedings of the IEEE 76*, 8 (Aug.), 869 –889.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. 29* (July), 104:1–104:10.

BOURQUE, E., AND DUDEK, G. 2004. Procedural texture matching and transformation. *Comp. Graph. Forum 23*, 3, 461–468.

CALTHORPE, P. 2010. *Urbanism in the Age of Climate Change*. Island Press.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of SIGGRAPH '00*, ACM, 219–228.

CNU, 2012. Congress for the new urbanism. www.cnu.org.

CTOD, 2012. Center for transit-oriented development. www.ctod.org.

DELLAERT, F. 2003. A sample of monte carlo methods in robotics and vision. *Proceedings of the ISM Int. Symposium on the Science of Modeling*, 12.

DEPARTMENT OF TRANSPORTATION, C., 2010. Smart mobility 2010: A call to action for the new decade.

GILKS, W., RICHARDSON, S., AND SPIEGELHALTER, D. 1995. *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC.

HART, J. C., BAKER, B., AND MICHAELRAJ, J. 2003. Structural simulation of tree growth and response. *The Visual Computer 19*, 151–163.

HASTINGS, W. 1970. Monte carlo samping methods using markov chains and their applications. *Biometrika*, 97–109.

HONDA, M., MIZUNO, K., FUKUI, Y., AND NISHIHARA, S. 2004. Generating autonomous time-varying virtual cities. In *Proceedings of the Int. Conference on Cyberworlds*, IEEE Computer Society, 45–52.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (July), 795–802.

LEFEBVRE, L., AND POULIN, P. 2000. Analysis and synthesis of structural textures. *Graphics Interface*, 77–86.

LIPP, M., SCHERZER, D., WONKA, P., AND WIMMER, M. 2011. Interactive modeling of city layouts using layers of procedural content. *Comp. Graph. Forum 30*, 2, 345–354.

MERRELL, P., AND MANOCHA, D. 2008. Continuous model synthesis. *ACM Trans. Graph. 27*, 5 (Dec.), 158:1–158:7.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph. 30*, 4 (Jul.), 87:1–87:10.

METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., AND TELLER, E. 1953. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics 21*, 6, 1087–1092.

MÜLLER, P., ZENG, G., WONKA, P., AND VAN GOOL, L. 2007. Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3 (July).

MURPHY, T. P. 1980. *Urban indicators: a guide to information sources*. Gale Research Co.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of SIGGRAPH '01*, ACM, 301–308.

PARK, J. P., LEE, K. H., AND LEE, J. 2011. Finding syntactic structures from human motion data. *Comp. Graph. Forum 30*, 8, 2183–2193.

RIEDMILLER, M., AND BRAUN, H. 1993. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE Int. Conf. on Neural Networks*, vol. 1, 586 –591.

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

TRANSECT. 2012. *Center for Applied Transect Studies*. www.transect.org.

TURRIN, M., VON BUELOW, P., AND STOUFFS, R. 2011. Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Adv. Eng. Inform. 25*, 4 (Oct.), 656–675.

VANEGAS, C. A., ALIAGA, D. G., BENEŠ, B., AND WADDELL, P. A. 2009. Interactive design of urban spaces using geometrical and behavioral modeling. *ACM Trans. Graph. 28*, 5 (Dec.), 111:1–111:10.

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., ALIAGA, D. G., AND BENEŠ, B. 2010. Building reconstruction using manhattan-world grammars. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 358–365.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH '97*, ACM Press/Addison-Wesley Publishing Co., 65–76.

ŠŤAVA, O., BENEŠ, B., MĚCH, R., ALIAGA, D. G., AND KRIŠTOF, P. 2010. Inverse Procedural Modeling by Automatic Generation of L-systems. *Comp. Graph. Forum 29*, 2, 665–674.

WATSON, B., MÜLLER, P., WONKA, P., SEXTON, C., VERYOVKA, O., AND FULLER, A. 2008. Procedural urban modeling in practice. *IEEE Computer Graphics and Applications 28*, 3 (June), 18 –26.

WEBER, B., MÜLLER, P., WONKA, P., AND GROSS, M. 2009. Interactive Geometric Simulation of 4D Cities. *Comp. Graph. Forum 28*, 2, 481–492.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Trans. Graph. 28*, 5 (Dec.), 112:1–112:9.

WONG, C. 2006. *Indicators for Urban and Regional Planning: The Interplay of Policy and Methods*. Routledge.

XIAO, J., FANG, T., TAN, P., ZHAO, P., OFEK, E., AND QUAN, L. 2008. Image-based facade modeling. *ACM Trans. Graph. 27*, 5 (Dec.), 161:1–161:10.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph. 30*, 4 (July), 86:1–86:12.