The Visual Computer
https://doi.org/10.1007/s00371-019-01701-x

ORIGINAL ARTICLE

CityCraft: 3D virtual city creation from a single image

Suzi Kim[1] · **Dodam Kim**[2] · **Sunghee Choi**[1]

Abstract
This paper introduces a method to generate a three-dimensional (3D) virtual model of an imaginary city from a single street-view image to represent the appearance of the city in a given input photograph. The proposed approach differs from reconstruction approaches, which generate a city model by guessing the city name from the input photograph. In contrast, we use machine learning to identify where to generate the city, what to allocate in the city, and how to arrange the components. We employ generative adversarial networks and convolutional neural networks to create a terrain map and identify the components and styles that represent the virtual city appearance. We demonstrate that our system creates 3D virtual cities that are visually similar in terms of plausibility and naturalness to actual cities corresponding to input photographs from around the world. To the best of our knowledge, this is the first work to generate a city model including all general city components, including streets, buildings, and vegetation, to match the style of a single input image.

Keywords Image-based modeling · City modeling · Urban construction · Building modeling · Inverse procedural modeling

1 Introduction

With the rapid development of virtual reality, there is a growing need for innovative 3D modeling technologies that enable non-experts to easily create their own 3D models. We introduce a method for creating a 3D virtual city model from a single street-view photograph. Our goal is not to reconstruct an existing city by guessing the name of the city from the input photograph, but to create a new city that represents the appearance of the street in the input photograph.

Virtual city modeling is an important problem that has applications in many areas, such as gaming and movie industries, and civil engineering. City modeling considers various constraints and requires substantial time and effort, even for experts. There have been several previous attempts to generate 3D city models, classified into procedural and image-based modeling. Procedural modeling [8,32,40,52,60] generates city components based on the grammar, whereas image-based modeling [3,12,23,56] generates the components based on available images, such as street-level or aerial images. Both techniques focus on reconstruction of a real city that with identical grammar or input image, and three-dimensional (3D) components are only constructed when the input image contains a corresponding component. As the 3D city model becomes more complicated, it requires enormous amounts of data, which is problematic when constructing a large city model. We overcome these shortcomings by combining inverse procedural modeling [1,2,53] and procedural modeling.

City appearance is determined by two factors: *city properties*, such as street patterns, density, and building type; and distribution of various *city components*, such as roads, vegetation, and water. Thus, the city modeling problem is reduced to where to generate the city, what to allocate in the city, and how to arrange the components. We develop the proposed system based on correlations [34] between components in an image. It starts from two observations [1,11]. (i) If some components appear simultaneously in an image, they are likely to exist together in the real world; (ii) city spatial properties correlate to city components. For example, radial street patterns often occur in older cities, with streets extending outward from a center, whereas grid street patterns are more

✉ Sunghee Choi
sunghee@kaist.edu

Suzi Kim
kimsuzi@kaist.ac.kr

Dodam Kim
dodam.kim@samsung.com

[1] School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea

[2] Samsung Research of Samsung Electronics Co., Ltd., Suwon, South Korea

Published online: 20 May 2019

commonly seen in modern cities designed to expand quickly and unsophisticatedly in a short amount of time, and the latter type cities usually have skyscrapers.

The proposed system is trained from a large amount of online street-level images [11]. Component co-occurrences are extracted from scene segmentation using Dilated Net [61]. To find the solutions to where to generate a city, and what to allocate in the city, we create terrain and height maps from a city component vector using generative adversarial networks (GANs) [19]. The city component vector is constructed by extracting labels directly associated with city modeling from the parsed scene. We also use convolutional neural networks (CNNs) to identify spatial properties of component arrangements. The CNNs receive the input city images to identify the correlation between city properties and components.

To the best of our knowledge, this is the first work to create a city model including all general city components to match the style of a single input image. We demonstrate the strength of our approach at modeling 3D virtual cities that are visually similar in terms of plausibility and naturalness to actual cities corresponding to input photographs from around the world in Sect. 7.

2 Related work

Previous city-scale modeling studies can be classified into procedural and image-based modeling. Since the proposed approach is closer to inverse procedural modeling, we briefly discuss these three domains.

2.1 Procedural city modeling

Procedural city modeling constructs models with grammar-based rules for the city components. City-scale procedural modeling is more massive and complex than typical procedural modeling. There are various methods to achieve city-scale procedural modeling.

Talton et al. [52] made a city model fit on the given grammar and high-level specification. This is difficult to generate a complex city which has an intricate relationship between multiple components. Lipp et al. [32] introduced interactive city layout modeling, which facilitates manual modeling. This modeling combines procedural modeling and manual modeling methodologies to produce a valid urban layout. We simplify the process of city generation to skip the manual modeling, which was traditionally required for detailed components, such as public telephone boxes or traffic lights.

Chen et al. [8] proposed a large street network modeling technique using tensor fields. However, the system required a semi-generated map as input, and the city model focused on just the street arrangement, which is insufficient to create a city. Yang et al. [60] generated a city using streets and parcels as units. However, since this approach did not consider street-level detailed component allocation, only a semi-virtual city is produced. Parish and Müller [40] aimed to create a virtual city model, and the developed approach was the most similar previous work to the current proposed approach. However, they only utilized aerial imagery to extract building and street details, which produce less plausible detail than the proposed approach due to the lack of suitable street-level probabilistic model.

Recent urban procedural modeling researches [39,46,47,54,59] have focused more on interaction with users, but it is still difficult to create plausible models quickly and easily by non-experts.

2.2 Image-based city modeling

Image-based city modeling is indispensable for generating a realistic and detailed modeling. Image sources can be classified into two types: street-level and aerial. Fan et al. [12] extracted facade textures from an input image to achieve building reconstruction, and Vezhnevets et al. [56] also modeled buildings from the image texture. Hou et al. [24] and Wolberg and Zokai [58] reconstructed buildings procedurally using extraction of architectural patterns from the images. However, they only considered facade construction using existing building texture or architectural pattern, which is difficult to apply to city-level modeling.

Aliaga et al. [3] synthesized a city layout using street networks and parcels extracted from aerial fragments, and Henricsson et al. [23] generated 3D models using automatic detection from aerial images. However, these approaches only considered man-made objects and required an enormous number of aerial images to generate a large-scale virtual city.

2.3 Inverse procedural modeling

Inverse procedural modeling extracts rules or parameters to provide a generative system from given data, such as buildings, plants, and cities [2]. Procedural modeling is powerful for generating huge models, but also requires a huge grammar, which cannot be easily generated. Inverse procedural modeling compensates for these shortcomings by automatically converting existing models into procedural representations. Therefore, the proposed system performs inverse procedural modeling before constructing a city using procedural modeling.

Vanegas et al. [53] proposed a method to obtain the desired model by controlling parameters of a procedural model, such as shadows and distance from a park. AlHalawani et al. [1] proposed street network descriptors, which included topological and geometric features, to distinguish existing cities.

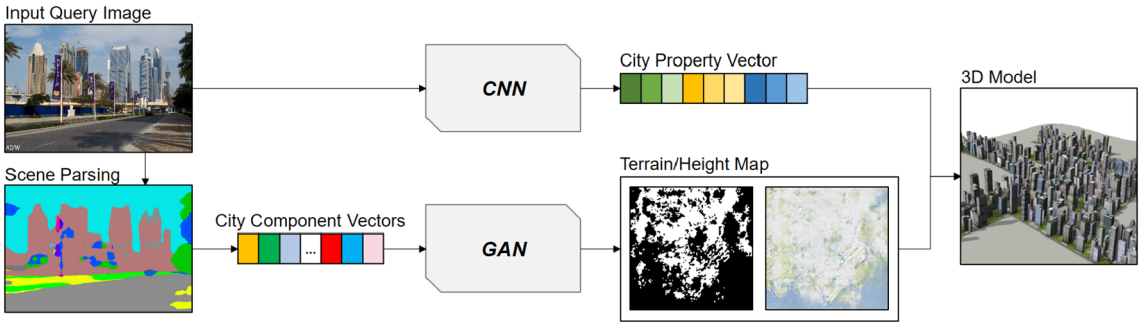


Fig. 1 System overview. Our system performs two sub-tasks: (1) CNN to generate a city property vector from the input query image and (2) GAN to generate terrain and height maps from the segmentation result of the input query image. We construct a 3D city model by combining two results of the learning model

Our system adopts some of these features to describe city characteristics.

3 Overview

We first introduce the notations used in our system and present an overview of two training types to generate 3D city models. Figure 1 shows an overview of the proposed system.

3.1 Scene parsing

The first step of the training procedure is to collect a large number of street-level images, each of which describes the co-occurrences of components that comprise the city. We download a large number of online street-level images and perform scene parsing to extract city components. Scene parsing segments an image into different regions associated with semantic categories, such as sky, road, and person. Since street-level images may contain objects that are irrelevant to land-use components, such as human and sky, we build a city component vector using a segmentation result instead of the input image itself to ignore those regions which are not related to land-use categories.

3.2 Terrain and height maps

Since terrain affects land-use upon which the city components are built, terrain modeling is a basic task for city modeling. A terrain map is composed of multiple layers with different properties, such as vegetation, water body, and landscape [17,48]. Previous studies regarding terrain generation have been largely based on procedural modeling techniques [16,20,25,27,45,46,48] and Geographic Information System footprints [28] to automate laborious and repetitious tasks. However, procedural modeling creates models only as defined in the prescribed rules; hence, it is impossible to create models that deviate from predefined guidelines [22]. To address this shortcoming, we develop a data-driven inverse procedural modeling to derive rules from the real world.

Height maps are the most common representation for terrain elevation [46]. Although the height map is associated with the terrain map, it cannot be easily generated by linear mapping from the terrain map. Thus, we combine the height and 2D terrain maps into a 3D terrain map, to perform 3D city modeling. All terrain and height maps are obtained from Terrain.party[1] and Stamen Maps,[2] respectively, both of which are based on OpenStreetMap. The downloaded maps describe basic terrain elements, such as mountains, rivers, sea, and dessert.

3.3 City component vectors

City analysis is a widely discussed topic in civil engineering. City components refer to the physical elements that make up a city. We define city components using CityGML [30], which describes 3D city models as a markup language. The components are divided into five land-use types: building, road, water, land, and vegetation, and the city component vector indicates the existence of each component in a city. The city component vector is obtained from online street-level images through Dilated Net [61], and forms the input to generate a terrain map of the target city.

3.4 City property vectors

Two city property types are used in component allocation: common and selective properties. Common properties are based on the observation that components tend to be densely

[1] https://terrain.party/.

[2] http://maps.stamen.com.

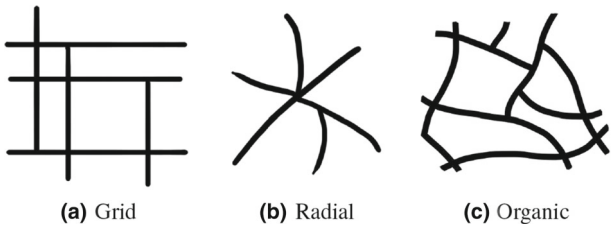


Fig. 2 Street patterns of cities. **a** Grid pattern has orthogonally crossed straight streets. It mainly occurs in planned city. **b** Radial pattern has a single civic center and extends streets outward from the center. **c** Organic pattern has an irregular shape. This pattern is evolved naturally as cities have developed over time

distributed at civic centers and more sparsely distributed as the distance from civic center increases. Without loss of generality, we assume that all cities satisfy these common properties. Selective properties are intended to distinguish each city by its unique characteristic. They are represented with city features, which are essential factors that determine the city appearance and pattern. We represent selective properties using a simple *city property vector*.

A major factor affecting city appearance is the street pattern. We analyze street networks to understand the component distributions in real cities. Since the street network is created before other components [40], component distribution is dependent on the shape and characteristics of the street network. Bauer [6] showed that the street patterns are divided into three types: grid, radial, and organic, as shown in Fig. 2. We adopt various common properties for city modeling, as shown in Table 1; hence, the property vector can be expanded adding extra properties to describe a city model in more detail.

3.5 System overview

Figure 1 shows that the proposed system consists of four stages: scene parsing, generation of city property vectors, generation of terrain and height map, and 3D model construction. First, we parse the input query image and extract the city component vector by filtering segmentation labels associated with city modeling. The city component vector is passed through the trained GAN model to obtain terrain and height maps. CNN model takes the original image as input to obtain a city property vector. We synthesize a 3D city model based on the obtained terrain and height maps by applying parameters collected from the city property vector.

Figure 3 shows the paired training data of CNN and GAN models. We train the CNN model using pairs of street-level city images and city property vectors, as described in Sect. 4. The segmented city images are converted to city component vectors by filtering labels associated with city modeling. Pairs of city component vectors and terrain and height maps are used to train the GAN model, as described in Sect. 5.

4 City property vector generation

CNNs have become commonly used for computer vision tasks, such as image classification [31], detection [18], and parameter estimation [38] in recent years. Our model is constructed based on AlexNet [31], a widely adopted CNN introduced by Krizhevsky et al. Five convolutional layers and three fully connected layers are used to learn the relationship between street-level images and the city property vectors. We perform batch normalization and max-pooling at various stages, as shown in Fig. 4.

Table 1 City properties describes city appearance and pattern

Notation	Property	Sub-properties
$\mathbf{p_{sd}}$	Street density	High, mid, low
$\mathbf{p_{sp}}$	Street pattern	Radial, grid, organic
$\mathbf{p_{st}}$	Street texture	Asphalt, concrete, cobblestone, unpaved
$\mathbf{p_{bd}}$	Building density	High, mid, low
$\mathbf{p_{bt}}$	Building type	Skyscraper, mid-level, low-level
$\mathbf{p_{bls}}$	Building level (skyscraper)	High(36~), mid(29~35), low(22~28)
$\mathbf{p_{blm}}$	Building level (mid-level)	High(16~21), mid(10~15), low(4~9)
$\mathbf{p_{bll}}$	Building level (low-level)	High(3), mid(2), low(1)
$\mathbf{p_{br}}$	Building roof type	Flat, gabled, hipped, pyramidal, mansard
$\mathbf{p_{vd}}$	Vegetation density	High, mid, low
$\mathbf{p_{vt}}$	Vegetation type	Broad-leaved, needle-leaved, leafless

A vector serialized with these properties represents the 3D city model style. Properties are divided into two types according to how they are activated in the 3D modeling process: (1) one sub-property with the highest value is activated ($\mathbf{p_{sd}}$, $\mathbf{p_{sp}}$, $\mathbf{p_{bd}}$, $\mathbf{p_{bls}}$, $\mathbf{p_{blm}}$, $\mathbf{p_{bll}}$, $\mathbf{p_{vd}}$, $\mathbf{p_{vt}}$) or (2) every sub-property is activated and each sub-property indicates the proportions of sub-properties within the property ($\mathbf{p_{st}}$, $\mathbf{p_{bt}}$, $\mathbf{p_{br}}$). Sub-properties of building levels indicate the range of actual building levels

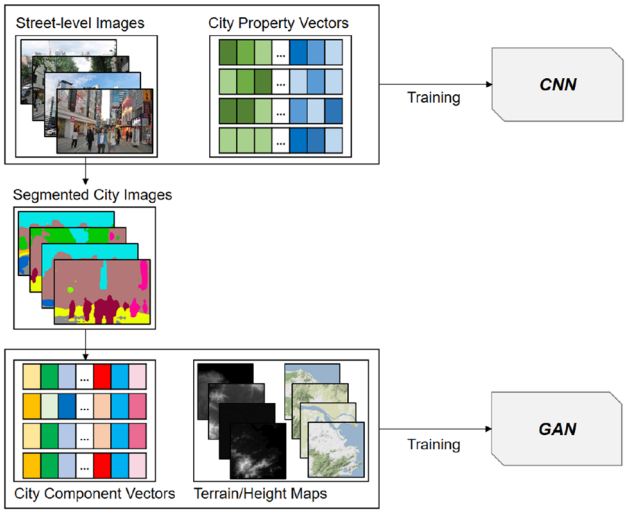


Fig. 3 Paired training data. Both learning models require paired training data. We obtain pairs of a street-level image and a city property vector to train our CNN model. We generate city component vectors from the segmentation results of the street-level images used in CNN training and make pairs of the city component vector and terrain/height maps to train our GAN model

With the proliferation of data-driven approaches incorporating large geotagged image datasets available online, techniques for utilizing online images have attracted great interest [10,11,49,50]. Doersch et al. [11] proposed a set of stylistic elements to determine the impression of the city model, and these visual elements can be acquired from online images. We choose 109 cities to reflect various city characteristics and download 150 street-level images per city. The fully connected layers terminate in an output layer with 36 output neurons corresponding to the city property vector $\mathbf{u} = \{\mathbf{p_i^j} \mid \mathbf{p_i} \in P, j \in E_{\mathbf{p_i}}\}$, where $P = \{\mathbf{p_{sd}}, \mathbf{p_{sp}}, \mathbf{p_{st}}, \mathbf{p_{bd}}, \mathbf{p_{bt}}, \mathbf{p_{bls}}, \mathbf{p_{blm}}, \mathbf{p_{bll}}, \mathbf{p_{br}}, \mathbf{p_{vd}}, \mathbf{p_{vt}}\}$ and $E_{\mathbf{p_i}}$ are the set of all the sub-property p_i, as described in Table 1.

The ground truth city property vectors are obtained from OpenStreetMap. We extract property information from OpenStreetMap across various cities, where the property vector indicates how much a given city holds corresponding properties. We obtained the number of entities in the city through the Overpass API query. For the density calculation, additional step was taken to divide the number of entities by the official area. In the case of street pattern, we manually labeled using the result maps of the query that highlighted the primary road. A city can manifest multiple property elements, e.g. older district streets might be covered with cobblestone, whereas main streets are covered with asphalt. We serialize the property values into a linear vector of length 36, and a set of vectors becomes the CNN input to define a style that represents the virtual city appearance.

5 Terrain and height maps generation

Virtual terrains are important components to represent natural environments, and have a wide range of applications, such as film, computer games, landscape design, and simulations. Various approaches have been suggested to generate the terrain, such as procedural [16,20], physics-based [9,55], sketch-based [13], and example-based [4,21] modeling.

Guérin et al. [21] generate a terrain model using the Conditional GANs (cGANs) [36]. However, it requires a user-guided sketch of the main terrain features, which corresponds to the output terrain map. We adopt deep convolutional generative adversarial networks (DCGANs) [41] to generate a virtual terrain map inferred from a component vector extracted from a street-level image, and it is a vector-to-image learning, not an image-to-image transfer with the same size of input and output (Fig. 5).

GANs [19] train a generator model G by deceiving a discriminator model, D. From a random noise vector, z, G tries to generate a realistic output, y, that cannot be distinguished from real images, i.e., $G : z \rightarrow y$. The discriminator is trained to identify whether a given image is real or generated by G. Conditional generative adversarial networks (cGANs) [36] feed additional input information to both G and D to condition the model. The cGANs learn mapping using this additional information, x, and z onto y, i.e., $G : \{x, z\} \rightarrow y$.

In this paper, we extend the cGANs to generate terrain and height maps from a given city component vector. We follow the approach of Reed et al. [42] to embed a city component vector. Both G and D are based on the deep convolutional

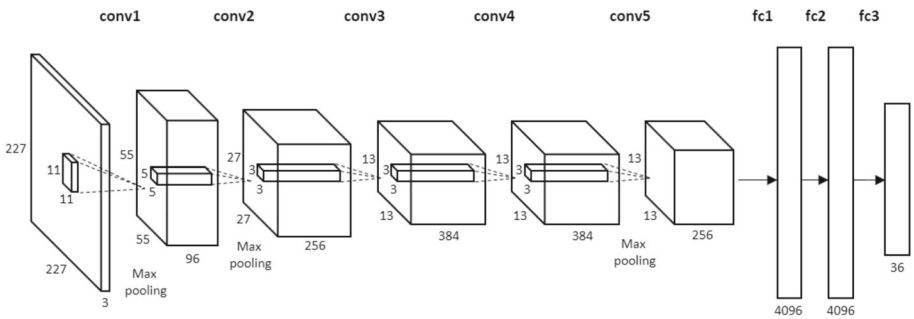


Fig. 4 Architecture of our CNN model to generate city property vectors

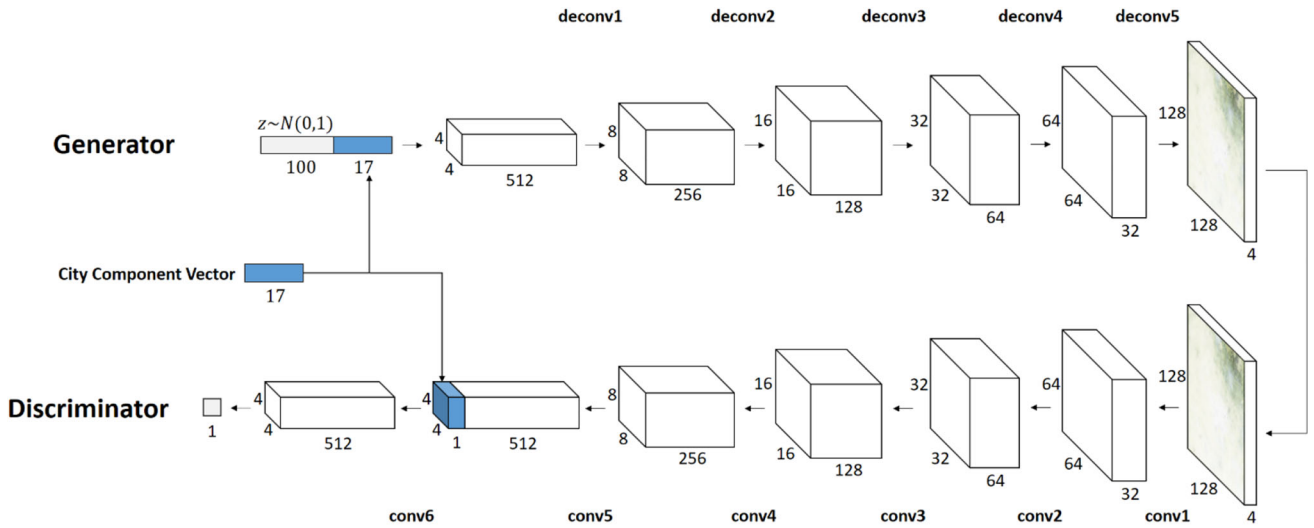


Fig. 5 Architecture of our generator and discriminator models to generate terrain and height maps

generative adversarial networks (DCGANs) [41], employing batch normalization and ReLU immediately after each convolution [26]. To reduce mode collapse during training, we adopt unrolling mechanism [35] using the generator objective with respect to unrolled optimization of the discriminator.

Every deconvolutional layer in the generator, except the last layer, is preceded by an ReLU and followed by batch normalization. Every convolutional layer in the discriminator is preceded by a Leaky ReLU and followed by batch normalization. Every layer has kernel size of 5x5 and stride of 2, except conv6 which has kernel size of 1x1 and stride of 1. The generator input is a city component vector of length 17, and output is a four-dimensional image to contain the terrain map in RGB channels and a height map in the alpha channel. The city component vector is generated by reusing the segmented images discussed in Sect. 4.

We calculate the city component vectors for 109 cities using the ratio of city components from 150 images per city. Each element of a city component vector $\mathbf{v} = \{e_1, \ldots, e_n\}$ is defined as:

$$e_i(c) = \frac{\sum_{w \in W(c)} \sum_{p \in w} g(p, \Phi_i)}{\sum_{w \in W(c)} \sum_{p \in w} g(p, \Phi_*)}, \qquad g(r, \Omega) = \begin{cases} 1 & \text{if } label(r) \in \Omega \\ 0 & \text{otherwise} \end{cases}. \tag{1}$$

We calculate the sum of pixels p in image w for $W(c)$, an image set of city. If the segmented label of pixel r is included in Ω, the $g(r, \Omega)$ becomes 1, and 0 otherwise. Φ_i is the set of the i-th component, and Φ_* is the set of all components. Since the segmentation result may have irrelevant labels, e.g. sky and people, we filter the components related to city modeling and combine them into Φ_*. Thus, we use 17 components to represent city land-use, divided into five types, as shown in Table 2.

Table 2 City components

Land-use	Components
Building	Building, skyscraper, tower
Vegetation	Tree, palm tree, grass, bush
Water	Sea, river, lake
Road	Road, sidewalk, path
Land	Soil, sand, mountain, hill

City components refer to the physical elements that make up a city. We define the components of city based on CityGML [30], which describes 3D city models as a markup language

The discriminator goal is to distinguish generated from real maps. We use pairs of the city component vector and terrain/height maps to train the GAN model. The terrain and height maps are downloaded from Terrain.party and Stamen Maps, at sufficient scale to cover the whole city area, and the height map is added as the alpha channel of the terrain map, which uses RGB channels. Each 4-channel map is resized to 128x128 pixels and input to the discriminator. The discriminator output is a sigmoid function, where 0 indicates fake and 1 indicates real maps.

6 3D city model generation

The remainder of our system constructs a 3D city model using the obtained information. Since the road network has a greater influence on city shape than vegetation or buildings, we generate the road model before the other components. Thus, we construct 3D models in the sequence: roads, build-

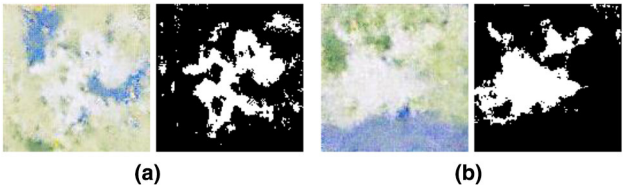


Fig. 6 Examples of obstacle maps. The obstacle map describes the regions where 3D models can be constructed. We generate the obstacle map (right) by masking the terrain map (left), produced through GAN, greater than $K = (215, 215, 215)$ to white

ings, and vegetation on a 3D terrain map by modifying the city generation workflow, as suggested by Smelik et al. [48].

First, we upscale the terrain map, smooth it to eliminate aliasing, and elevate the terrain based on the height map. An obstacle map shows the areas where the 3D model cannot be constructed, with obstacle regions marked in a dark color to avoid attempted generation of 3D models upon it. Residential areas are shown by brighter color than other regions, such as water and mountain. We follow the color scheme of Stamen Maps, which filled the urban area with the gray color (242, 242, 234). Since the land-use can be overlapped as map scale becomes smaller, the color of the urban area becomes darker than (242, 242, 234). We have gradually found an appropriate value for RGB threshold K at a sufficient map scale to cover the whole city in a single bounding box, and generate the obstacle map by masking regions lower than RGB threshold $K = (215, 215, 215)$ to black, as shown in Fig. 6.

6.1 Road network

There are several generation techniques to automatically generate a road network based on a set of input parameters and rules [8,15,32,51]. We adopt the generation of roads and urban areas from Smelik et al. [48] to perform the parcel generation, and the building arrangement in a parcel.

We create a simple road network model with parameters of street pattern ($\mathbf{p_{sp}}$), street density ($\mathbf{p_{sd}}$), and street texture ($\mathbf{p_{st}}$). These parameters are the key features of road network modeling, and provide the impression of the city model. A simple road model is sufficient to represent a virtual city model with similar appearance to the input image, which is the purpose of this paper, and attempting to create an elaborate road network becomes a significant computational overhead.

The road network is hierarchical according to functions and capacities [14,48,51]. We categorize the road network into primary and secondary roads. Primary roads divide large parcels, and secondary roads subdivide these parcels into small parcels with multiple lots. Primary road pattern is decided by $\mathbf{p_{sp}}$. City property vectors, $\mathbf{u}$, include 3 elements related to street pattern: grid ($\mathbf{p_{sp}}^{grd}$), radial ($\mathbf{p_{sp}}^{rad}$), and organic($\mathbf{p_{sp}}^{org}$). The largest element, $\text{argmax}_j\ \mathbf{p_{sp}}^j$, is adopted as the primary road pattern.

Secondary roads are generally checker shape [33,40,43]; hence, we apply that grid pattern to generate the secondary roads. We control the road network total length or density by reflecting a street density ($\mathbf{p_{sd}}$). Similar to the primary street pattern, we use the largest element, $\text{argmax}_j\ \mathbf{p_{sd}}^j$, to decide the road network density. Street texture ($\mathbf{p_{st}}$) affects the road network texture, as explained detail later.

6.2 Buildings

To construct 3D building models, we adapt the concept of *architectonic likelihood* suggested by Bellotti et al. [7]. This concept originated from the observation that a tourist usually remembers general building features after visiting a city and does not aim to construct a detail modeling for every individual building, but just to represent the sense of being in a particular city with a few meaningful styles. Thus, we generate a set of simple 3D building models to represent similar features from an input image.

First, we generate a 3D building model by simple extrusion of footprint shape from subdividing the small parcels. The building lots are chosen randomly according to the building density ($\mathbf{p_{bd}}$), aside from the amount to be assigned to vegetation according to the vegetation density ($\mathbf{p_{vd}}$). Other land-uses, such as water and land, are already marked in the terrain map, so we only consider building and vegetation areas during lot allocation.

A building is extruded in the shape of the assigned lot. If the lot is not placed on the corner of parcel, the building shape is cuboid; otherwise, the shape is a prism identical to the shape of the lot. The building type ($\mathbf{p_{bt}}$), building levels ($\mathbf{p_{bls}}$, $\mathbf{p_{blm}}$, $\mathbf{p_{bll}}$) and roof type ($\mathbf{p_{br}}$), from the property vector, decide the particular 3D building model style. In contrast to road network generation, which chooses the largest element, we use all property elements to construct the building model: building type (skyscraper, mid-level, low-level) and roof type (flat, gabled, hipped, pyramidal, mansard). We split the lots according to the proportional amount λ of element k for building property θ,

$$\lambda_\theta^k = \mathbf{p}_{\cdot k} / \sum\nolimits_{j \in E_{\mathbf{p}_\cdot}} \mathbf{p}_{\cdot j}. \tag{2}$$

This assignment process is repeated for each $\mathbf{p_{bt}}$ element. For $\mathbf{p_{br}}$, we do not apply the allocation directly, since skyscrapers usually have flat roofs [5,40]. Therefore, we assign flat roofs to every skyscraper model, and reassign the new ratio between roof types $\widetilde{\lambda}_{br}$ to the remaining mid-level, and low-level buildings as follows:

$$\tilde{\lambda}_{br}^{flat} = max\left(\lambda_{br}^{flat} - \lambda_{bt}^{skysc}, 0\right),$$
$$\tilde{\lambda}_{br}^{k} = \frac{\lambda_{br}^{k}}{\lambda_{br}^{gable} + \lambda_{br}^{hip} + \lambda_{br}^{pyra} + \lambda_{br}^{mans} + \tilde{\lambda}_{br}^{flat}}, \quad (3)$$

where k is one of the roof types except flat one. The building levels ($\mathbf{p_{bls}}$, $\mathbf{p_{blm}}$, $\mathbf{p_{bll}}$) affect to the average level of each building type, and one sub-property with the highest value is activated.

6.3 Vegetation

After generating the building models, the remaining lots are chosen randomly according to the vegetation density ($\mathbf{p_{vd}}$). Every vegetation lot is covered with grass texture, and we add 3D vegetation models according to the vegetation type ($\mathbf{p_{vt}}$). Since vegetation is largely dependent on climate and a city generally has relatively constant climate, we choose the largest, $\text{argmax}_j\ \mathbf{p_{vt}}_j$, among the vegetation type elements.

6.4 Textures

The 3D city model components, buildings and roads, require textures to provide a plausible impression similar to the input photographs. For the road network, road materials are affected by street texture parameters from the city property vector. However, since wide roads are generally paved with asphalt, we apply asphalt texture to primary roads regardless of the parameters. Secondary roads are paved with a variety of materials, since they frequently appear on street-view images with unique characteristics. We apply a specific texture to each secondary road following the ratio between the street texture parameters (asphalt, concrete, cobblestone, unpaved), as described in Eq. 2.

For each building model, we use building textures inferred from the input image. We obtain facade texture by incorporating concepts from texture perception methods, which are frequently used in 3D reconstruction of buildings from images [7,12,44]. However, we do not integrate texture synthesis methods, which analyze shape grammar from facade images to reconstruct or synthesize facade textures [37].

In contrast, we define a set of texture clusters, and the i-th cluster with $\text{argmax}_i\ (Sim(S_q, S_i))$ is used as the building model texture. We extract a set of textures, S_q, from an input query image, and measure the similarity $Sim(S_q, S_x)$ between S_q and each predefined texture cluster to select the most similar cluster to apply for the current building texture. The similarity between two clusters M and N is:

$$Sim\,(M, N) = \frac{1}{|M|\,|N|} \sum_{m \in M} \sum_{n \in N} \delta(m, n), \quad (4)$$

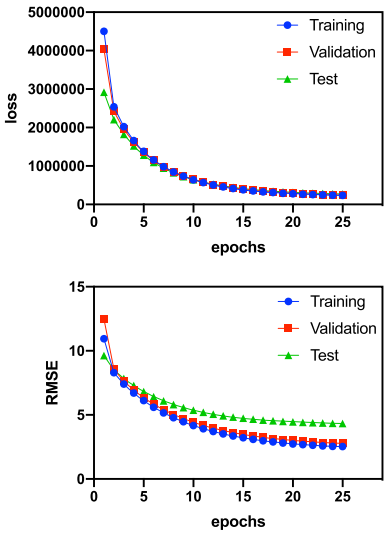


Fig. 7 Loss and RMSE of the training, validation and test data

where $|M|$ is the number of textures contained in cluster M; and

$$\delta(m, n) = \frac{1}{W} \sum_{i=1}^{W} l_i(x, y)^{\alpha} c_i(x, y)^{\beta} s_i(x, y)^{\gamma} \quad (5)$$

is the structural similarity [57] between two textures, m and n. We only compare textures within the same category, such as plaster, roof, window, and door, and calculate the structural similarity by averaging over 11x11 pixel sliding windows. For two image windows x and y, we calculate the luminance, $l(x, y)$; contrast, $c(x, y)$; and structure $s(x, y)$:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \quad (6)$$
$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \quad (7)$$
$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \quad (8)$$

where μ_x and μ_y, and σ_x^2 and σ_y^2 are the window means and variances, respectively; σ_{xy} is the covariance between the two windows; C_1, C_2, and C_3 are small positive constants that stabilize each term; and α, β, and γ are positive constants that control the weight of each term. We set $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$ in our experiments.

We do not define clusters by architectural styles, such as medieval, renaissance, high rise, neoclassic, since this causes problems where a texture is applied without considering the building shape. For example, if the selected cluster is renaissance style, this would produce inappropriate texture matching for a building cluster containing a skyscraper. Therefore, we form clusters with similar style and shape

Table 3 Partial outcomes from the CNN model

	Street texture				Building type			Building roof type				
	Asphalt	Concrete	Cobblestone	Unpaved	Skyscraper	Mid-level	Low-level	Flat	Gabled	Hipped	Pyramidal	Mansard
Figure 9a	45.54	10.95	14.78	28.74	28.97	63.18	7.85	25.36	25.39	30.01	11.11	8.14
Figure 9b	42.20	22.66	5.11	30.03	17.60	43.77	38.64	20.76	38.78	31.50	6.25	2.71
Figure 9c	56.97	3.16	39.47	0.39	53.78	37.66	8.56	77.61	8.61	7.69	3.65	2.45
Figure 9d	29.56	20.30	11.58	38.57	5.95	71.69	22.35	11.29	15.80	30.31	29.03	13.56
Figure 9e	46.58	16.89	24.38	12.16	52.26	36.68	11.06	57.93	11.98	13.92	10.40	5.77
Figure 10a	57.60	9.63	17.41	15.37	57.17	35.30	7.53	63.11	12.99	8.99	8.60	6.31
Figure 10b	39.36	30.26	17.67	12.71	26.65	40.17	33.19	25.48	20.33	24.88	17.48	11.83

Our CNN model generates a 1×36 city property vector. The result indicates the percentage of elements in each category by normalizing the learning results

regardless of architectural style, using k-means clustering on texture images to form clusters with similar shapes. Structural similarity is used to calculate the distance between texture images during k-means clustering. We separate roof and facade textures to form clusters, and choose the most similar cluster for each roof and facade. To allow for texture flexibility according to building type, we split facade textures into two layers, ground and upper floors, and synthesize the layers separately according to the number of floors.

If we add the facade texture property to the city property vector and set each style cluster as sub-property, we should choose one of two strategies to determine activated textures: selecting top-1 cluster; or selecting several top clusters. Selecting a top-1 cluster means that sub-property with the highest value is activated. It requires more complex training but shows the same results as the current cluster matching method. There are about 50 textures in one cluster, and selecting several top clusters may generate a confusing model with the use of too many unnecessary textures. Therefore, unlike other city properties, facade texture is excluded from the city property vector since the linear matching result from the facade texture of the input photograph to the output model seems plausible.

7 Results

We downloaded 150 street-level images from Google Images for each of 109 cities with various combinations of terrains and city styles, including Amsterdam, Da Nang, Cape Town, Chicago, Beijing, Helsinki, London, Lisbon, Nairobi, Montreal, Hong Kong, Kyoto, Vancouver, Seattle, Wellington, and Lima, to name just a few. The ground truth city property vectors were obtained from OpenStreetMap by counting the number of corresponding entities through the Overpass API query. The calculation of density properties took an additional step to divide the number of entities by the official area. The street pattern was manually labeled using the result maps of the query that highlighted the primary road. All experiments were performed using a single machine running Ubuntu 16.04.1 LTS with i7-6800K Intel Core CPU, 32GB of memory, and NVIDIA GTX 1080 graphics card. Models were exported and rendered to CityEngine,[3] a state of the art system for 3D city modeling. The textures used in this paper are provided by CityEngine or downloaded from Google Images.

7.1 City property vector generation

We fine-tuned the AlexNet, which was pre-trained on the ImageNet, to obtain better performance. We used a learning rate of 0.00001 and applied dropout with a probability of 0.5 in the hidden layers between fully connected layers for 25 epochs. Our regression model was trained with L2 loss function. Training images were augmented with probability 0.5 by rotations ($-30°$ to $30°$) and horizontal flipping.

Figure 7 shows the sum-squared error loss and root-mean-square error (RMSE) of fivefold cross-validation. Table 3 shows example results of generated city property vectors using the proposed model. Asphalt tends to be higher than other street textures since the city primary road networks comprise a relatively large area compared to secondary roads, and are all paved with asphalt. European style cities have high cobblestone ratio of secondary roads, which is consistent with most older European cities actual street patterns.

7.2 Terrain and height maps generation

We used a learning rate of 0.0002 for the ADAM optimizer [29] with momentum 0.5 and batch size 64 for 900 epochs. We split the 16,350 pairs into a training and testing sets with 75% and 25%, respectively. Training images were augmented with probability 0.5 by rotations ($-30°$ to $30°$) and horizon-

[3] https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview.

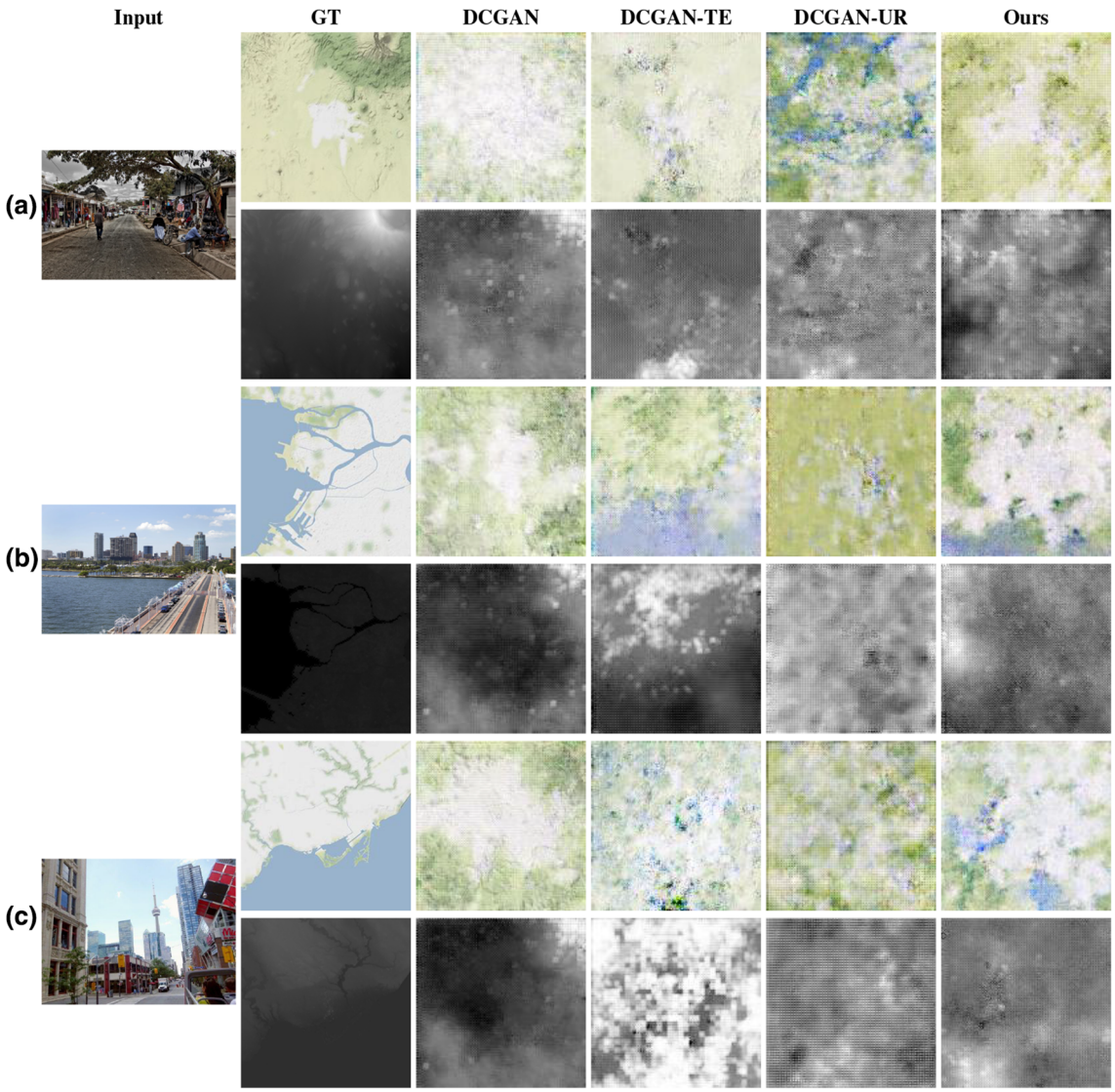


Fig. 8 Terrain and height map generations using DCGAN, DCGAN-TE, DCGAN-UR, and ours. Input images are chosen from unseen test set. Each example is an street-level image of **a** Arusha, **b** Saint Petersburg and **c** Toronto

tal flipping. The output is 4-channel images, incorporating the terrain map in RGB channels and height map in the alpha channel.

Figure 8 compares the DCGAN baseline, DCGAN with text embedding (DCGAN-TE) [42], DCGAN with unrolled optimization (DCGAN-UR) [35] and ours which combines both text embedding and unrolled optimization. DCGAN-TE generates a map including plausible city components, but the maps do not look good. DCGAN-UR generates a plausible form of the map, but do not include proper city components. Ours combines the strengths of the two models and produces a plausible form of the map including proper city components.

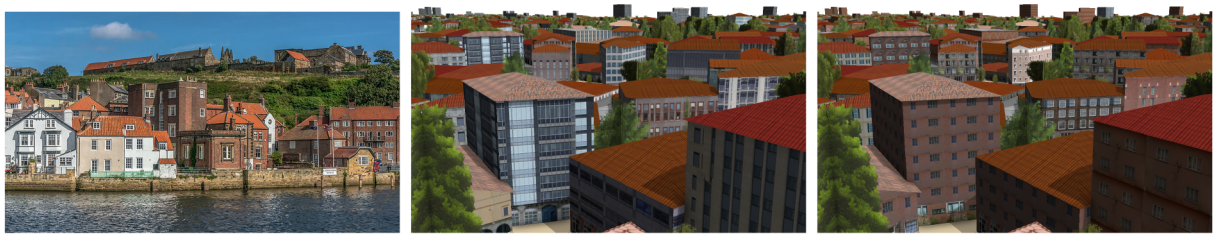
Figure 8a shows example maps that represent only components within the query image. However, we do not limit the maps to consist only of components within the input image. The proposed model infers some components, such as sea and river, which is not in the input query images, but actually existed in the city.

7.3 3D city model generation

Figure 9 shows some examples of 3D virtual city models generated by the proposed method. The virtual city models not only reflect the visible components of the input image, but also include some estimated components and properties from that image. For example, Fig. 9c shows that although no water component is visible in the input image, the 3D model contains waterbody, since the real city (Yokohama, Japan) is by the sea. Thus, even in the absence of information

Fig. 9 Example outcomes for 3D virtual city modeling: (column 1) input query image; (column 2) aerial view, (column 3) mid-level view, and (column 4) street-view of virtual cities

Fig. 10 3D virtual city models reconstructed from different query images of Dubai



(a) Input **(b)** Clustered with architectural style **(c)** Clustered by our method

Fig. 11 Texture differences from different clustering methods

regarding the presence of some components, we can guess their occurrence plausibly through the proposed approach.

If we were to guess actual city identity from the input image, and then generate a 3D model similar to the identified city regardless of the input image, many anomalies would arise. For example, Dubai, UAE center is filled with skyscrapers, whereas the outskirts are surrounded by desert. The resultant models should, and do, vary depending on the input image focus, as shown in Fig. 10. Thus, the proposed method generates a model faithfully reflecting the actual input image.

We performed k-means clustering on 1,044 facade textures with $k = 20$, and compared textures for clustering using normal architectural style and the proposed method (Fig. 11b, c, respectively). The skyscraper texture in Fig. 11b is less suitable than the texture provided by the proposed method, because the architectural style texture in is derived to cover only mid-level buildings.

8 Conclusion

We proposed a method to generate a 3D city model, which represents the appearance of the street in a given street-level photograph, by guessing unknown factors affecting appearance of the city. We categorized the factors into city components and city properties, where the city components refer to the physical elements that make up a city, such as roads, vegetation, and water, and the city properties refers to intangible characteristics, such as street patterns, density, and building type. We use GAN and CNN models to infer the unknown components and properties, and the models are trained from the real data of existing cities. We showed how segmentation affects perceived city properties and terrain and height maps through the proposed learning models, and generated 3D city models by plausibly combining the derived city properties on the height and terrain maps. We provided example model outcomes for a variety of input query images. The proposed method is appropriate for the film or game industries, providing city-scale models to be created relatively quickly with minimal effort by non-experts.

Funding This study was funded by National Research Foundation of Korea (NRF) (Grant Number 2015R1D1A1A09060399).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. AlHalawani, S., Yang, Y.L., Wonka, P., Mitra, N.J.: What makes London work like London? Comput Graph Forum **33**(5), 157–165. https://doi.org/10.1111/cgf.12441
2. Aliaga, D.G., Demir, İ., Benes, B., Wand, M.: Inverse procedural modeling of 3D models for virtual worlds. In: ACM SIGGRAPH 2016 Courses, p. 16. ACM (2016)
3. Aliaga, D.G., Vanegas, C.A., Beneš, B.: Interactive example-based urban layout synthesis.ACM Trans. Graph. **27**(5), 160:1–160:10 (2008). https://doi.org/10.1145/1409060.1409113
4. Argudo, O., Andujar, C., Chica, A., Guérin, E., Digne, J., Peytavie, A., Galin, E.: Coherent multi-layer landscape synthesis. Vis. Comput. **33**(6–8), 1005–1015 (2017). https://doi.org/10.1007/s00371-017-1393-6
5. Bao, F., Schwarz, M., Wonka, P.: Procedural facade variations from a single layout. ACM Trans. Graph. (TOG) **32**(1), 8 (2013)
6. Bauer, K.W.: City Planning for Civil Engineers, Environmental Engineers, and Surveyors. CRC Press, Boca Raton (2009)
7. Bellotti, F., Berta, R., Cardona, R., De Gloria, A.: An architectural approach to efficient 3D urban modeling. Comput. Graph. **35**(5), 1001–1012 (2011)
8. Chen, G., Esch, G., Wonka, P., Müller, P., Zhang, E.: Interactive procedural street modeling. ACM Trans. Graph. **27**(3), 103:1–103:10 (2008). https://doi.org/10.1145/1360612.1360702
9. Cordonnier, G., Braun, J., Cani, M.P., Benes, B., Galin, E., Peytavie, A., Guérin, E.: Large scale terrain generation from tectonic uplift and fluvial erosion. Comput. Graph. Forum **35**(2), 165–175 (2016). https://doi.org/10.1111/cgf.12820
10. Crandall, D.J., Backstrom, L., Huttenlocher, D., Kleinberg, J.: Mapping the world's photos. In: Proceedings of the 18th International Conference on World Wide Web, pp. 761–770. ACM (2009)
11. Doersch, C., Singh, S., Gupta, A., Sivic, J., Efros, A.A.: What makes Paris look like Paris? ACM Trans. Graph. **31**(4), 101:1–101:9 (2012). https://doi.org/10.1145/2185520.2185597
12. Fan, L., Musialski, P., Liu, L., Wonka, P.: Structure completion for facade layouts. ACM Trans. Graph. **33**(6), 210:1–210:11 (2014). https://doi.org/10.1145/2661229.2661265

13. Gain, J., Long, H., Cordonnier, G., Cani, M.P.:EcoBrush: interactive control of visually consistent large-scale ecosystems. Comput. Graph. Forum **36**(2), 63–73 (2017). https://doi.org/10.1111/cgf.13107
14. Galin, E., Peytavie, A., Guérin, E., Beneš, B.: Authoring hierarchical road networks. Comput. Graph. Forum **30**(7), 2021–2030 (2011). https://doi.org/10.1111/j.1467-8659.2011.02055.x
15. Galin, E., Peytavie, A., Maréchal, N., Guérin, E.: Procedural generation of roads. Comput. Graph. Forum **29**(2), 429–438 (2010). https://doi.org/10.1111/j.1467-8659.2009.01612.x
16. Génevaux, J.D., Galin, É., Guérin, E., Peytavie, A., Benes, B.: Terrain generation using procedural models based on hydrology. ACM Trans. Graph. (TOG) **32**(4), 143 (2013)
17. Génevaux, J.-D., Galin, E., Peytavie, A., Guérin, E., Briquet, C., Grosbellet, F., Benes, B.: Terrain modelling from feature primitives. Comput. Graph. Forum **34**(6), 198–210 (2015). https://doi.org/10.1111/cgf.12530
18. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
19. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
20. Guérin, E., Digne, J., Galin, E., Peytavie, A.: Sparse representation of terrains for procedural modeling. Comput. Graph. Forum **35**(2), 177–187 (2016). https://doi.org/10.1111/cgf.12821
21. Guérin, E., Digne, J., Galin, E., Peytavie, A., Wolf, C., Benes, B., Martinez, B.: Interactive example-based terrain authoring with conditional generative adversarial networks. ACM Trans. Graph. (TOG) **36**(6), 228 (2017)
22. Guo, J., Cheng, Z., Xu, S., Zhang, X.: Realistic procedural plant modeling guided by 3D point cloud. In: ACM SIGGRAPH 2017 Posters, pp. 85:1–85:2. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3102163.3102193
23. Henricsson, O., Streilein, A., Gruen, A.: Automated 3-D reconstruction of buildings and visualization of city models (1996)
24. Hou, F., Qin, H., Qi, Y.: Procedure-based component and architecture modeling from a single image. Vis. Comput. **32**(2), 151–166 (2016)
25. Huijser, R., Dobbe, J., Bronsvoort, W.F., Bidarra, R.: Procedural natural systems for game level design. In: 2010 Brazilian Symposium on Games and Digital Entertainment (SBGAMES), pp. 189–198. IEEE (2010)
26. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
27. Kaňuk, J., Gallay, M., Hofierka, J.: Generating time series of virtual 3-D city models using a retrospective approach. Landsc. Urban Plan. **139**, 40–53 (2015)
28. Kelly, T., Femiani, J., Wonka, P., Mitra, N.J.: BigSUR: large-scale structured urban reconstruction. ACM Trans. Graph. **36**(6), 204:1–204:16 (2017). https://doi.org/10.1145/3130800.3130823
29. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv:1412.6980 (2014)
30. Kolbe, T.H., Gröger, G., CityGML: interoperable access to 3D city models. In: van Oosterom, P., Zlatanova, S., Fendel, E.M. (eds.) Geo-Information for Disaster Management. Springer, Berlin, Heidelberg (2005)
31. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Advances Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
32. Lipp, M., Scherzer, D., Wonka, P., Wimmer, M.: Interactive modeling of city layouts using layers of procedural content. Comput. Graph. Forum **30**(2), 345–354 (2011). https://doi.org/10.1111/j.1467-8659.2011.01865.x
33. Lyu, X., Han, Q., de Vries, B.: Procedural modeling of urban layout: population, land use, and road network. Transp. Res. Procedia **25**, 3333–3342 (2017)
34. Ma, R., Li, H., Zou, C., Liao, Z., Tong, X., Zhang, H.: Action-driven 3D indoor scene evolution. ACM Trans. Graph. **35**(6), 173-1 (2016)
35. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J.: Unrolled Generative Adversarial Networks. arXiv:1611.02163 (2016)
36. Mirza, M., Osindero, S.: Conditional Generative Adversarial Nets. arXiv:1411.1784 (2014)
37. Müller, P., Zeng, G., Wonka, P., Van Gool, L.: Image-based procedural modeling of facades. ACM Trans. Graph. (TOG) **26**(3), 85 (2007)
38. Nishida, G., Bousseau, A., Aliaga, D.G.: Procedural modeling of a building from a single image. Comput. Graph. Forum **37**(2), 415–429 (2018). https://doi.org/10.1111/cgf.13372
39. Nishida, G., Garcia-Dorado, I., Aliaga, D.G., Benes, B., Bousseau, A.: Interactive sketching of urban procedural models. ACM Trans. Graph. (TOG) **35**(4), 130 (2016)
40. Parish, Y.I., Müller, P.: Procedural modeling of cities. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 301–308. ACM (2001)
41. Radford, A., Metz, L., Chintala, S.: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 (2015)
42. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text-to-image synthesis. In: Proceedings of The 33rd International Conference on Machine Learning (2016)
43. Sharma, R.: Procedural city generator. In: International Conference System Modeling & Advancement in Research Trends (SMART), pp. 213–217. IEEE (2016)
44. Sinha, S.N., Steedly, D., Szeliski, R., Agrawala, M., Pollefeys, M.: Interactive 3D architectural modeling from unordered photo collections. ACM Trans. Graph. **27**(5), 159:1–159:10 (2008). https://doi.org/10.1145/1409060.1409112
45. Smelik, R., Galka, K., De Kraker, K.J., Kuijper, F., Bidarra, R.: Semantic constraints for procedural generation of virtual worlds. In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, p. 9. ACM (2011)
46. Smelik, R.M., Tutenel, T., Bidarra, R., Benes, B.: A survey on procedural modelling for virtual worlds. Comput. Graph. Forum **33**(6), 31–50 (2014). https://doi.org/10.1111/cgf.12276
47. Smelik, R.M., Tutenel, T., de Kraker, K.J., Bidarra, R.: Interactive creation of virtual worlds using procedural sketching. In: Lensch, H.P.A., Seipel, S. (eds.) Eurographics 2010 - Short Papers, Norrköping, Sweden, May 3–7, 2010. Eurographics Association (2010)
48. Smelik, R.M., Tutenel, T., de Kraker, K.J., Bidarra, R.: A declarative approach to procedural modeling of virtual worlds. Comput. Graph. **35**(2), 352–363 (2011)
49. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3D. ACM Trans. Graph. **25**(3), 835–846 (2006). https://doi.org/10.1145/1141911.1141964
50. Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from internet photo collections. Int. J. Comput. Vis. **80**(2), 189–210 (2008)
51. Sun, J., Yu, X., Baciu, G., Green, M.: Template-based generation of road networks for virtual city modeling. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, pp. 33–40. ACM (2002)
52. Talton, J.O., Lou, Y., Lesser, S., Duke, J., Měch, R., Koltun, V.: Metropolis procedural modeling. ACM Trans. Graph. (TOG) **30**(2), 11 (2011)

53. Vanegas, C.A., Garcia-Dorado, I., Aliaga, D.G., Benes, B., Waddell, P.: Inverse design of urban procedural models. ACM Trans. Graph. (TOG) **31**(6), 168 (2012)
54. Vanegas, C.A., Kelly, T., Weber, B., Halatsch, J., Aliaga, D.G., Müller, P.: Procedural generation of parcels in urban modeling. Comput. Graph. Forum **31**(2pt3), 681–690 (2012). https://doi.org/10.1111/j.1467-8659.2012.03047.x
55. Vanek, J., Benes, B., Herout, A., Stava, O.: Large-scale physics-based terrain editing using adaptive tiles on the GPU. IEEE Comput. Graph. Appl. **31**(6), 35–44 (2011)
56. Vezhnevets, V., Konushin, A., Ignatenko, A.: Interactive image-based urban modeling. In: Proceedings of PIA, pp. 63–68 (2007)
57. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
58. Wolberg, G., Zokai, S.: PhotoSketch: a photocentric urban 3D modeling system. Vis. Comput. **34**(5), 605–616 (2018). https://doi.org/10.1007/s00371-017-1365-x
59. Wonka, P., Aliaga, D., Müller, P., Vanegas, C.: Modeling 3D urban spaces using procedural and simulation-based techniques. In: ACM SIGGRAPH 2011 Courses, p. 9. ACM (2011)
60. Yang, Y.L., Wang, J., Vouga, E., Wonka, P.: Urban pattern: layout design by hierarchical domain splitting. ACM Trans. Graph. (TOG) **32**(6), 181 (2013)
61. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ade20k dataset. In: Proceedings of CVPR (2017)

Suzi Kim received the B.S. and M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 2012 and 2016. She worked for the Daewoo Shipbuilding Marine Engineering (DSME) from January 2012 to March 2013 and Prezi from April 2013 to August 2014. She is currently working toward the Ph.D. degree in School of Computing at KAIST. Her research interests include computer graphics such as procedural and inverse procedural modeling and geometry processing.

Dodam Kim received the B.S. degree in Electronic and Electrical Engineering from Sungkyunkwan University in 2014, and the M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 2016. She has been working in Samsung Research of Samsung Electronics Co., Ltd. since 2017.

Sunghee Choi received the B.S. degree in computer engineering from Seoul National University in 1995, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Austin, in 1997 and 2003, respectively. She has been working as a professor in School of Computing at Korea Advanced Institute of Science and Technology (KAIST) Daejeon, Korea since 2004. Her research interests include computational geometry, computer graphics and geometric problems in wireless sensor networks.