The Visual Computer (2019) 35:1181–1191
https://doi.org/10.1007/s00371-019-01667-w

ORIGINAL ARTICLE

Example-based rapid generation of vegetation on terrain via CNN-based distribution learning

Jian Zhang[1] · Changbo Wang[1] · Chen Li[1] · Hong Qin[2]

Published online: 7 May 2019

Abstract
Modeling large-scale vegetation on terrain is an important and challenging task in computer games, movie production and other digital entertainment applications. In this work, we propose a novel example-based method for rapid generation of vegetation in outdoor natural environments. Its central idea is to learn the vegetation distribution on terrain via deep convolution neural networks. We first use a pre-trained deep neural network to extract rich local information from the terrain pertinent to vegetation distribution. Second, we produce the initial features of the target vegetation distribution based on patch matching and further introduce a network that generates a vegetation density map based on the initial features. Third, during the synthesis stage, we propose a procedural method to generate the vegetation distribution data corresponding to the terrain data. Our research work confirms that the image features extracted by the pre-trained deep neural network could be utilized to explore the connection between vegetation and terrain. We validate our new method over various outdoor scenes, including procedural generated scenes and scenes with manual control on tree patterns. The experimental results demonstrate that our method can rapidly produce new realistic scenes for outdoor natural environments, which relies on the mechanism of learning correlationship between vegetation distribution and terrain data.

Keywords Vegetation modeling · Style transfer · Natural phenomena

1 Introduction and motivation

Planting vegetation on existing terrain is a commonly encountered task in 3D outdoor scene modeling for natural phenomena. However, modeling large-scale vegetation with classical popularly used modeling software demands a lot of time and manpower. A procedural modeling method is oftentimes the first choice for natural scene modeling. Over the past decades, researchers have successfully applied procedural modeling techniques to many scenarios, such as terrains [36], cities [28], trees [37] and so on. Nonetheless, exploring the rapid generation of vegetation distribution on terrains has not been well studied with perfect solutions.

Typically, procedural techniques define a model controlled by parameters, which allow effectively generating complex details. However, users will be facing difficulties during parameter tuning in any procedural model, especially for novice users. Existing game engines, such as Unity®, usually provide vegetation brush for users to plant trees in an interactive manner. But this interactive approach tends to be time-consuming for large-scale scenes. Considering existing difficulties, we seek a new data-driven approach for vegetation generation in this paper. Data-driven or example-based methods employ data as input and are generally capable of producing similar results rather rapidly with certain constraints (from a physical or artistic standpoint). In our new method, the input data is a scene that has already been planted; then, we can reasonably plant vegetation on the new terrain. But how to measure 'being-reasonable' (i.e., the reasonability) for any output result would need rigorously mathematical tools in a quantitative fashion. From the biological perspective, the distribution of vegetation is usually related to water, altitude, temperature, light, slope, etc. Here, we consider the effect of terrain on vegetation distribution, which is frequently the case for games or simulations. We do not intend to enumerate all possible constraints (such as slope or height) of terrain on vegetation distribution, because it is

✉ Changbo Wang
cbwang@sei.ecnu.edu.cn; cbwangcg@gmail.com

[1] School of Computer Science and Software Engineering, East China Normal University, Shanghai, China

[2] Department of Computer Science, Stony Brook University, Stony Brook, New York 11794-4400, USA

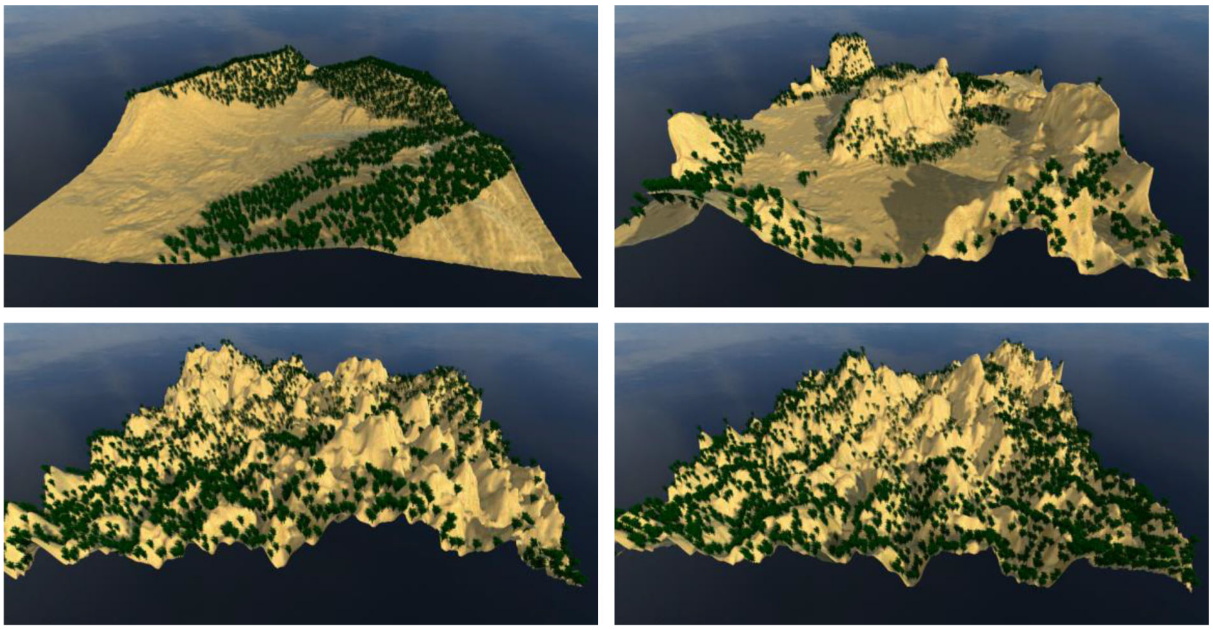


Fig. 1 An exemplar scene (top left) followed by three generated scenes. Given a simple terrain with manually planted trees, our method can catch the correlation between the terrain and the trees and apply it on new terrains

almost impossible to guarantee that there are no omissions. Inspired by the recent successes of deep learning techniques on feature extraction, we choose to adopt neural networks for extracting information from 3D scenes and then generating new scenes.

In this paper, we present a novel example-based method that takes exemplar scene as input and generates new vegetation scenes via deep neural networks (Fig. 1). To make full use of the functionalities of neural networks, we treat a 3D scene as a combination of height maps and vegetation density maps. We observe that the low-level features extracted by the VGG-Network are very relevant to our task. The correlation between terrain and vegetation distribution can be optimized with a Gram matrix of these features. Nonetheless, the optimization method is time-consuming and the direct use of available networks cannot produce precise local details. We further devise a forward neural network to predict the result with one feed-forward calculation. The key to this approach is to fuse the features of the input images via patch matching. After that we use a convolutional neural network (CNN) to extract the target image from the fused features (see Fig. 2). Training data would require special preparation for such a special functional neural network, since terrain and vegetation always come in pairs. We feed height maps from NASA SRTM as terrain training data. To obtain vegetation density data corresponding to terrain data, we could rely on an algorithm based on multiple random transfer functions. Our main contributions are highlighted as follows:

– We use the Gram matrix to characterize the correlation between terrain and its vegetation distribution base on the VGG-Network.
– We develop an end-to-end network to generate vegetation distribution rapidly.
– To train the network, we propose a method to dynamically generate training data, which can be also used for model evaluation and validation.

Directly benefiting from the aforementioned contributions, our method could quickly generate new scenes based on the given prototype scene, which can be used either in games or simulations or as initial settings to be further refined.

2 Related works

Our work is closely related to procedural modeling and inverse procedural modeling methods for vegetation generation. We also briefly review texture synthesis and style transfer techniques, especially approaches based on deep learning in recent years because of their great relevance to this paper.

Procedural modeling Procedural modeling techniques [4] offer rapid generation of vegetation and ecosystem. Deussen et al. [10] described a framework to produce the vegetation density map which is determined by user editing or ecosystem simulation. Their procedural model is driven by L-system and contains ecological parameters such as shade tolerance. Lane et al. [22] extended [10] to simulate the

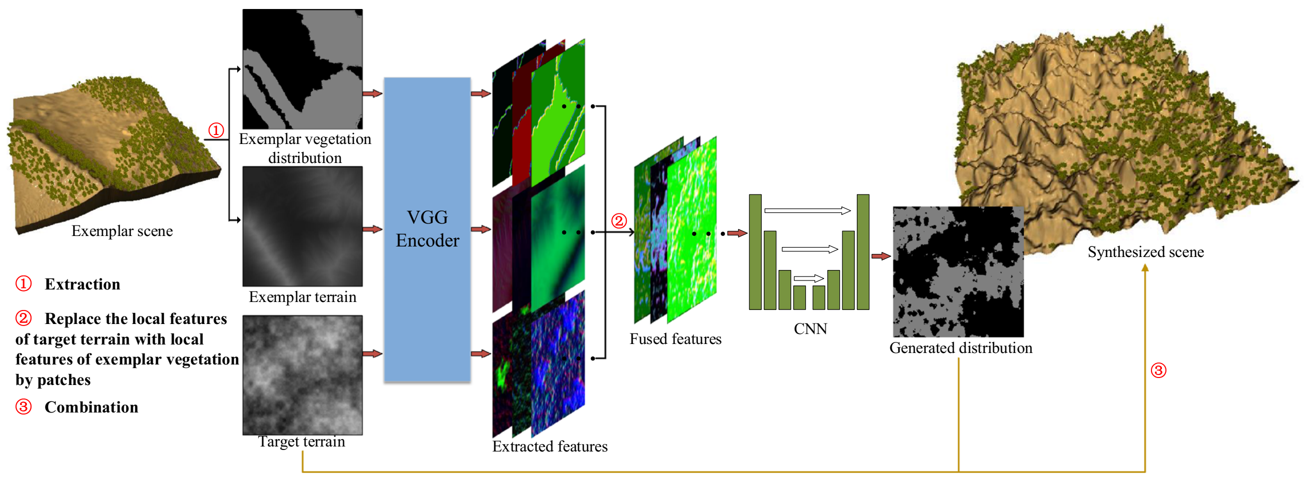


Fig. 2 The framework of the proposed method

collective ecological behavior of vegetation, for example succession and clustering of plants. Modeling the interactive behavior of plants with intuitive parameters was described in [3]. Recently, ecosystem modeling techniques that consider landscape elements such as water bodies and forests were presented in [2,8,33]. Kohek et al. [20] proposed a framework for large-scale forest construction and visualization which combines volumetric modeling and level of detail based on GPU.

Inverse procedural modeling Inverse procedural modeling has received more and more attention in computer graphics community [5,24,29,30,38]. Inverse procedural modeling, deducing the rules and/or parameters needed by traditional procedural modeling from the example inputs, has been successfully applied to objects including trees [37], buildings [43], cities [39], etc. They typically infer the composition of a procedural model from the attributes of the input data or run an existing model to produce expected results that satisfy some criteria. Machine learning techniques have also been used as components of procedural models [15,40,44]. Nishida et al. [27] proposed a set of regression CNN to estimate parameters and executed procedural models to match the user's building sketch. Ritchie et al. [31] uses CNN to help the procedural model making random choices and generate outputs with image constraints. Conditional generative adversarial networks that take terrain features as input can output convincing results [14]. The framework proposed in [11], which is similar to us, synthesizes virtual worlds from example scenes. They [11] focused on spatial statistics of natural features, while we mainly consider the correlation between local features.

Texture synthesis and style transfer Example-based texture synthesis [9,42] produces high-resolution images similar to examples. Gatys et al. [12] showed the possibility of using deep neural networks (VGG-19 [35]) to extract multi-scale features for texture synthesis. Follow-up study [13] has successfully applied it to image style transfer. The Gram-based optimization method [13] presented by Gatys et al. inspired many subsequent studies [18,34,41] on style transfer, which mainly focused on replacing the iterative optimization to one feed-forward calculation. Recently, Chen et al. [6] proposed a fast algorithm to transfer image style. They extracted patches from the example images and matched them to the result images, where the patches are computed with pre-trained networks. Huang et al. [16] believed that style can be represented by the normalized features. They successfully mapped the target image features to a new space with parameters of the normalized style image and then trained a network for prediction. These studies have motivated us because we can regard the vegetation distribution as a kind of 'style.' The most obvious difference between our method and theirs is that we need to find the inter-features between two images and they only capture intra-features of one image.

3 Problem statement and method overview

Our goal is to obtain a model that generates a reasonable vegetation distribution on a terrain from an exemplar scene with planted vegetation, where the new distribution should satisfy the correlation of the exemplar scene. In this article, we present an end-to-end network, which is successfully trained with online generated data, to solve the problem.

Here, we give the definition of 3D scene S as follows:

$$S = \{T, V\}, V = \{v_1, v_2, ..., v_n\}, \qquad (1)$$

where T is the terrain height map and V is the collection of distribution density map(s) of n type(s) of vegetation(s). Given a 3D scene $S_1 = \{T_1, V_1\}$ and height map T_2, we hope to generate new scene $S_2 = \{T_2, V_2\}$ similar to S_1. That is to say, the distribution of vegetation(s) on T_2 should be as reasonable as S_1.

As shown in Fig. 2, we extract the height map and the vegetation map(s) from the given 3D scene. Typically there will be more than one species of vegetation on the terrain; here, we only look into one vegetation for simplicity. The vegetation map is generated based on the density of the local distribution of plants on the terrain. To measure the correlation between terrain and vegetation distribution, we try to calculate the Gram matrix of these feature extracted by pre-trained network (Sect. 4.1). Then, we can produce new distribution through iteration optimization. We test it on different layers of VGG-Network and find that low-level features are useful for our task. However, Gram matrix does not ensure precise details for generated results and the iterative optimization method is slow, so we further seek a one feed-forward network to compute the results. In Sect. 4.2, we propose an U-Net like structure with an added preprocessing layer. The preprocessing layer, taking the characteristics of the task, swaps the input features locally to simplify the training task of the neural network. The skip connection in our network ensures that the local details are not lost during feed-forward. Training data is specially prepared. For vegetation distribution images, we propose a procedural generation algorithm to produce data correlated with the terrain data (detailed in Sect. 4.3). In this way, we can obtain the model after iterative training. To sum up, our model makes full use of the features extracted by the pre-trained network and uses patch matching to obtain appropriate features, then compresses these features into the domain of the density map through heuristic training.

4 CNN-driven vegetation generation

In this section, we will present how to produce vegetation layer on terrain from exemplar scenes. We use pre-trained VGG-Network to extract rich information of the terrain height map and vegetation distribution map. We further propose an end-to-end model to rapidly produce the vegetation distribution. During the training stage, we rely on a proposed algorithm to procedurally generate vegetation data.

4.1 Correlationship between terrain and vegetation

In general, the distribution of trees is always correlated with the elevation and slope of topography in nature, which we think is a kind of ‘style.’ Therefore, we treat the generation of new scenes with similar styles from the exemplar scenes as a problem of style transfer. Compared with the classical method for style transfer, the most obvious difference is that the style here cannot be described in only one image. Thus, our first priority is to learn the style from a 3D scene. CNN has been proven to have powerful ability to extract image features [21], including both low-level and high-level information. Here, we use height maps to represent terrains, and density maps to represent vegetation distributions on terrains (Fig. 2). Inspired by [13], we adopt VGG-Network [35] to extract information of terrains and vegetation distributions. Here, the feature correlationship between terrain and vegetation is denoted by Gram matrix $G = [G_{ij}]$, where G_{ij} is the inner product between terrain feature map i and vegetation feature map j:

$$G_{ij} = \sum_k F^t_{ik} \cdot F^v_{jk}. \tag{2}$$

Unlike in [13], here F^t and F^v are feature maps extracted from two images by the VGG-Network, where F^t denotes terrain features and F^v denotes vegetation features. VGG-Network, composed of sequential convolutional layers and pooling layers, is able to extract rich features from image. As the neural layer gets deeper, the receptive field becomes bigger and the features extracted by the VGG-Network get more abstract. We use G to calculate the correlation between terrain features and vegetation features. Meanwhile, we focus on topography information such as height or slope, namely low-level features that extracted with small receptive fields. Therefore, we select the front part of the neural network to learn the scene style. Experiments also show that it works on the generation problem of terrain scene (Fig. 3).

Obtaining the abstract representation of the style, we can use an iterative optimization approach to produce new scenes. Let G_1 and S_2 denote the Gram matrix of the features of exemplar scene S_1 and the target scene G_2, respectively. The objective is calculated with mean square error:

$$L_{\text{global}}(S_1, S_2) = \frac{1}{(2MN)^2}(G_1 - G_2)^2, \tag{3}$$

where M and N denote the dimensions of the feature maps. By minimizing Eq. 3 with L-BFGS [46], we can produce vegetation distribution in a similar style of S_1. As mentioned above, deep neural network learns multi-scale features from image. Here, we need low-level features, so we select layer ‘conv1_2’ (see Fig. 3 for a comparison of optimization results with different layers).

4.2 End-to-end model structure

The optimization method cannot produce precise local details because the objective (Eq. 3) based on Gram matrix mainly captures the overall correlation between terrain and vegetation distribution (Fig. 3). We propose to use a deep neural network Q to catch and produce the correlationship between

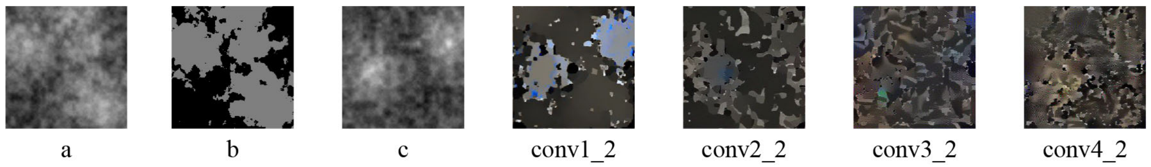


Fig. 3 In the given exemplar scene (**a**, **b**), the trees grow above a certain altitude. To generate distribution map on new terrain c, The optimization method performs best by using layer 'conv1_2'

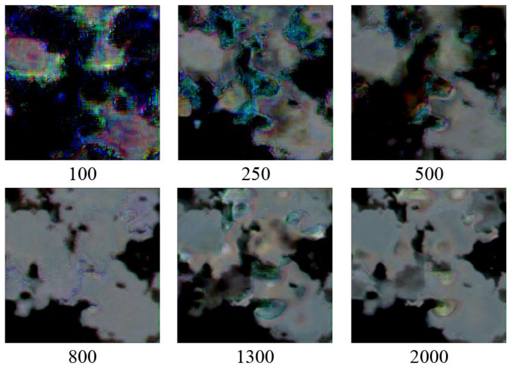


Fig. 4 Generation results of several iterations during training a direct network. The test input is the same as Fig. 3. We use the U-Net [32] structure to build the model and concatenate three images (i.e., the exemplar terrain and vegetation and the target terrain) as input. As shown from the figure, the model has a tendency to produce results similar to the exemplar vegetation input (Fig. 3b)

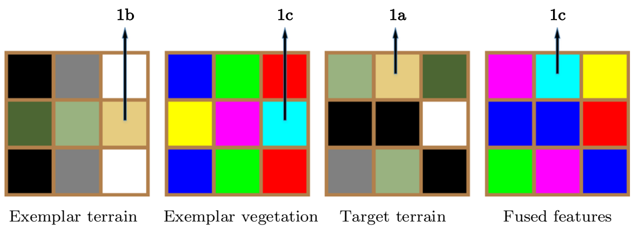


Fig. 5 Illustration of the function of the preprocessing layer. (1b) is the nearest patch of (1a). Then, (1a) is replaced by (1c) (patch in the same place as (1b))

terrain and vegetation distribution. To simplify the illustration, we consider only one type of vegetation on terrain. Then, $\mathcal{Q}$ takes T_1, V_1 and T_2 as input and outputs the vegetation distribution layer V_2 (see Sect. 3 for notations), which is a unified end-to-end network.

An approach easy to fetch is introducing a straightforward neural network and adopting Eq. 3 as the objective function. However, we find that Eq. 3 is hard to optimize to an acceptable situation directly and the training stage may get stuck (Fig. 4). This network tends to output a vegetation distribution similar to V_1.

In order to avoid the collapse of the model, we recommend using a preprocessing layer L_{pre} over the model input, which should facilitate the training of the model. As shown in Sect. 4.1, the low-level features learned by the VGG-Network

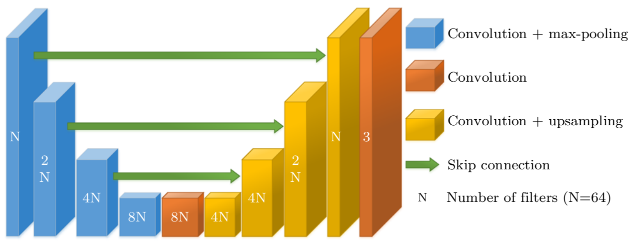


Fig. 6 The proposed end-to-end architecture of CNN

is very helpful to solve the problem, thus preprocessing is performed on the features extracted by the VGG-Network instead of on original input images. We propose a patch based method to replace the features of the target terrain (see Fig. 5 for an intuitive illustration). This approach can be implemented in parallel by using the convolution and the transposed convolution. The procedure of L_{pre} is as follows:

- Let $F_1^t(N \times M \times C)$ denotes the extracted features of the exemplar terrain, where N, M, C are the image width, the image height and the number of filters. Extract a set of patches $F_{1,p}^t(k \times k \times C \times (\frac{N}{k} \times \frac{M}{k}))$ from F_1^t, where k is the patch size (we set $k = 3$ for our experiments).
- To compute the similarity between the patches and target terrain, we apply a convolution on features of target terrain F_2^t with $F_{1,p}^t$ as filters:

$$\eta(F_2^t, F_1^t) = F_2^t \otimes F_{1,p}^t. \tag{4}$$

To obtain the best matching patches, an argmax operation is applied on the last axis of $\eta(F_2^t, F_1^t)$:

$$\eta(F_2^t, F_1^t)' = \text{argmax}(\eta(F_2^t, F_1^t)). \tag{5}$$

- Extract the patches $F_{1,p}^v$ from the features of the exemplar vegetation F_1^v in the same way as extracting $F_{1,p}^t$. Then, we can get the reconstructed features by applying a transposed convolution on $\eta(F_2^t, F_1^t)'$ with $F_{1,p}^v$ as filters.

The following part of $\mathcal{Q}$ is an U-Net [32] like structure (Fig. 6). The goal of adopting the deep neural network is to generate the vegetation map and to optimize the objective. The structure of deep CNN mainly consists of three parts:

– *Convolution + max-pooling* In the first half of the network, each convolution layer is followed by a pooling layer in order to reduce the dimensions of the features. To enhance the nonlinear capability of the network, we choose the max-pooling layer instead of the average-pooling. We also use Relu [26] activation function after each convolution layer for its strong nonlinearity and fast calculation.
– *Convolution + upsampling* In the second half of the network, each convolution layer is followed by a upsampling layer in order to enlarge the size of the features. Typically a convolution layer followed by an upsampling layer can be replaced by a transposed convolution layer [45] with stride of 2, but we find that our structure is already producing fairly good results.
– *Skip connection* The use of pooling layers that keep the number of filters unchanged leads to inevitable loss of information, which is harmful to the image synthesis problem [32]. Thus, we bring in skip connections to hold the multi-scale features.

As shown in Fig. 2, after connecting a VGG-Network, the preprocessing layer L_{pre} and the U-Net like network, $\mathcal{Q}$ becomes a purely end-to-end structure. In Fig. 6, every convolution layer is connected to a batch normalization layer [17] except for the last layer. In order to make the local features of the generated distribution conform to the output of L_{pre}, here we bring another objective L_{local} as below:

$$L_{\text{local}} = \left\| L_{\text{pre}}(F_1^t, F_1^v, F_2^t) - \hat{F}_2^v \right\|_2, \tag{6}$$

where $\hat{F}_2^v$ defines the features extracted by VGG-Network of $\mathcal{Q}$'s output. Thus, the loss function that considers both global similarity and local similarity is as below:

$$L_{\text{total}} = L_{\text{local}} + \lambda L_{\text{global}}, \tag{7}$$

where λ is a hyperparameter that balances L_{local} and L_{global}. At training stage, we pass the input triplet (T_1, V_1, T_2) into $\mathcal{Q}$ and get the prediction $\widehat{V_2}$; then, we can calculate the loss (Eq. 7) through pre-trained VGG-Network. The VGG-Network model is always fixed during training stage or testing stage. Since we do not use any fully connected layer and our preprocessing layer is composed of convolution operation and transposed convolution operation, $\mathcal{Q}$ can be applied to images of arbitrary size at the testing stage.

4.3 Datasets and vegetation density data generation

We use about 500 km × 500 km of real terrains from NASA SRTM for model training. Now given a terrain image T_{img}, we need a corresponding vegetation image V_{img} for our training. A choice is to randomly choose another height map from the terrain dataset as the vegetation image V_{img}. However, actually we hope that the model can learn the ability to extract the correlationship between terrain images and vegetation images and the ability to generate new conformable vegetation images. There is obviously no reasonable connection between the randomly selected T_{img} and V_{img}. Using such input data will make model optimization very difficult. Here, we propose a method of procedurally generating vegetation images to help model training.

Algorithm 1: Vegetation Density Data Generation

```
   Input: the input terrain $T_{N\times M}$
   Output: generated vegetation density map $V_{N\times M}$
1  $V_{N\times M} \leftarrow 0_{N\times M}$
2  Randomly select $n (0 \le n \le \delta)$ disjoint intervals
   $\mathbb{R} : \{r : [r_a, r_b]\}_n, 0 \le r_a < r_b \le 255$
3  Randomly generate $n$ functions $\mathbb{F} : \{f(x)\}_n$ corresponding to $\mathbb{R}$
4  foreach point $P_{i,j}$ in $T_{N\times M}$ do
5      if $height(P_{i,j})$ is in $r_i \in \mathbb{R}$ then
6          $V_{i,j} = f_i(P_{i,j})$
7      end
8  end
9  return $V_{N\times M}$
```

Examine Algorithm 1, the main idea is to apply different transformation functions to the different parts of the terrain. We define the form of the transformation functions in Algorithm 1 as follows:

$$f(x) = \kappa(\mu_1 \cdot \eta(x) + \mu_2), \tag{8}$$

where μ_1 and μ_2 are random numbers for perturbation purpose. $\eta(x)$ returns certain information at position x in terrain, and $\kappa(x)$ is the transfer function. In our implementation, $\eta(x)$ returns the normalized height or slope value at training stage, and we set 6 alternative transfer functions (Table 1) for Algorithm 1. During training, every time we randomly select an exemplar terrain and a target terrain, and dynamically generate an exemplar vegetation image at the same time (see Fig. 7 for a intuitive illustration). In Algorithm 1, δ is a parameter that limits the number of the functions and we set it to a small value (5) for the consideration of performance.

Table 1 The alternative transfer functions used in Eq. 8

$\kappa_1(x)$	$\kappa_2(x)$	$\kappa_3(x)$	$\kappa_4(x)$	$\kappa_5(x)$	$\kappa_6(x)$
x	x^2	x^3	$\sin(x \cdot \frac{\pi}{2})$	$e^x - 1$	c

$\kappa_6(x)$ is a constant function and c is a random number from 0 to 1 (for instance $\kappa_6(x) = 0.5$)

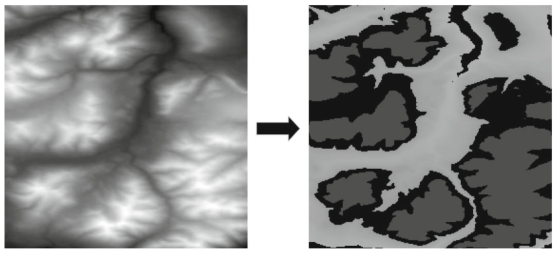


Fig. 7 The source terrain (left) and the generated density map (right). There are two transformation functions: $f_1(x) = e^{-0.2 \cdot \text{height}(x)+0.5} - 1(20 \leq \text{height}(x) \leq 100)$ and $f_2(x) = 0.2(150 \leq \text{height}(x))$

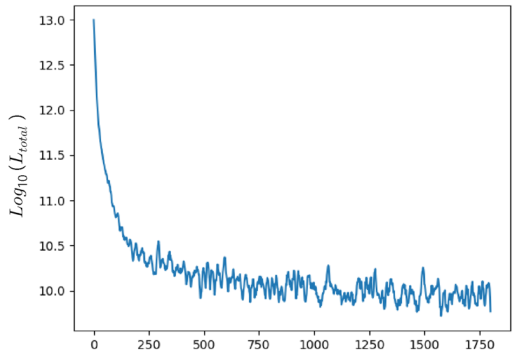


Fig. 8 Training curves of the loss

5 Implementation details and experimental results

In this section, we detail the training setup and evaluate our model with several scenes. We also perform quantitative performance statistics with procedurally generated data.

5.1 Network training

We use keras [7] library, with TensorFlow [1] as backend, as the framework to train our model. The training stage is performed on a PC with NVIDIA GeForce GTX 1060 (6 GB) and Intel Core i7 CPU, running at 2.8GHz with 16 GB RAM. We use Adam optimizer [19] with a learning rate of 10^{-4}, and both the parameters β_1 and β_2 are set to 0.9. The batch size is set to 8, i.e., at each iteration 24 input images (128×128) are fed into the model. λ in Eq. 7 is set to 1×10^{-3}. In our implementation, the training takes roughly 4 hours for 1800 iterations. We always perform model testing and parameters saving to disk after several iterations, which slightly affects the time consumption. The convergence of the training process is shown in Fig. 8.

5.2 Results and analysis

Experiments and Evaluation An obvious fact is that there exists no 'ground truth' for the scene generation task raised

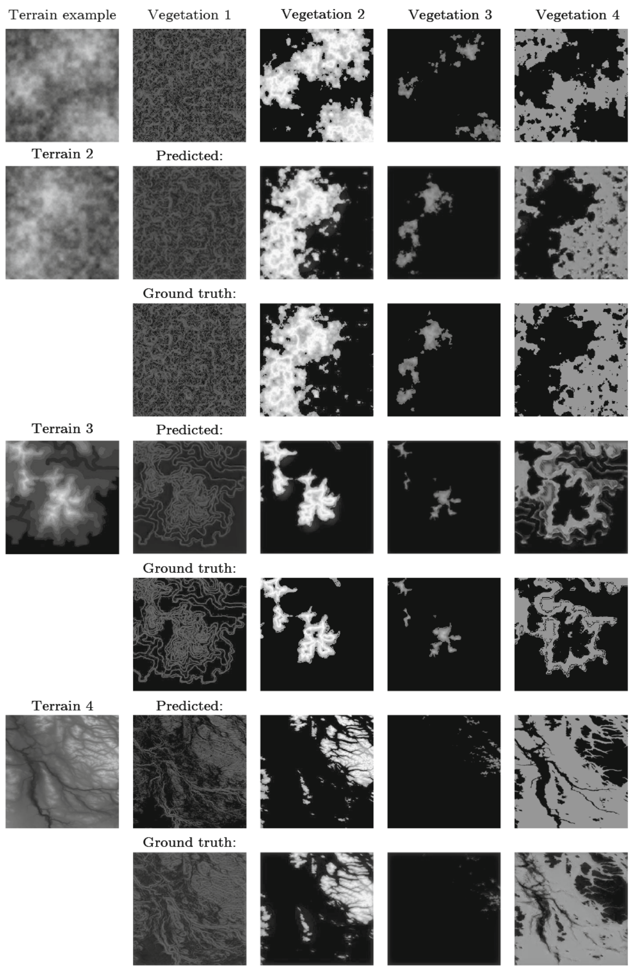


Fig. 9 Producing 4 plantings on the new 3 terrains. For better display of robustness, we choose 3 terrains that look different

in this article, and it is hard to produce it even if there exists. Although we have proposed an generation method in Algorithm 1, it cannot cover all conditions. However, we can use it to evaluate the capability of the model. In Fig. 9, we show 12 results generated by our model. The provided 4 vegetation density maps are generated by Algorithm 1, and the expected images are generated through the same functions and parameters. Our model is not exposed to these terrains during training stage, but Fig. 9 has shown the power of our example-based generation model. Examine Fig. 9, most of the images are very similar to ground truth, except for Vegetation 4 over Terrain 3, which is slightly different.

To further test the model, we seek terrain properties that the model has not touched during the training stage. In nature, the distribution of vegetation is usually closely related to light. Figure 10 shows the test of the model in this regard. We planted some trees in one direction on a simple terrain (Fig. 10a), which can be done in a few minutes. Figure 10c, e shows the results generated by the model on more complex terrains. Figure 10a, c, e is all rendered in the same

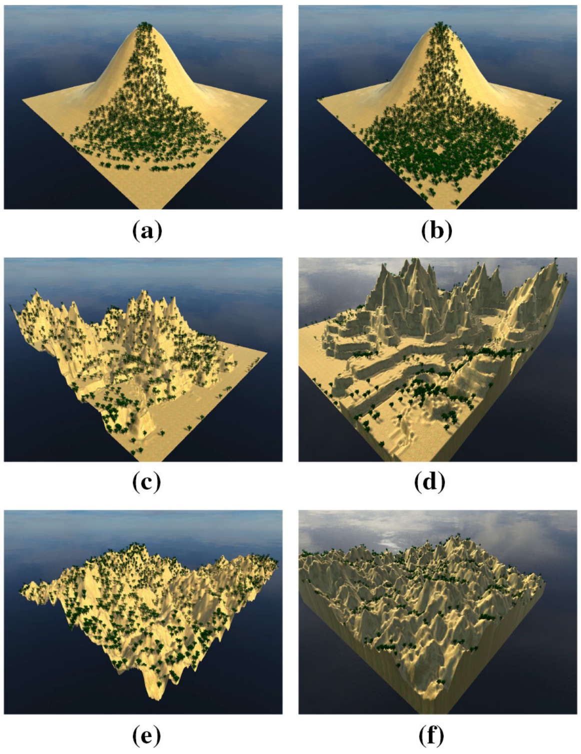


Fig. 10 The test over anisotropic features. One exemplar scene a followed by 3 generated scene, where **c** and **d** are different perspectives on the same scene (so does **e** and **f**)

direction, while Fig. 10d, f is in opposite directions for a clearer comparison. The two scenes (second and third rows in Fig. 10) showcase that our model is able to handle the anisotropic features. Figure 10 also showcases that the user can plant vegetation on a simple terrain and then use our model to rapidly generate scenes on complex terrain, which can significantly save manpower. We also test the effects of reproducing on the same terrain (Fig. 10b).

Hydraulic erosion simulation can produce very convincing ridges with terrain. However, simulation with different parameters and iteration numbers will result in various behavior. We generate flow map by using the hydraulic erosion method described in [25] and use our model to reproduce the flow map. Figure 11 showcases a comparison between the physically based method and our model. Although our model is not designed and trained to simulate hydraulic erosion, its results are generally consistent with the physically based method.

Ultimately, the approach in this article is for better and faster generation of new scenarios. In Fig. 12, there are five different types of vegetation in the exemplar scene (first row). We plant these vegetation with consideration of heights, slopes, sunlight and different distribution densities. The two new scenes, produced in seconds, show sufficient fidelity compared to the sample scene.

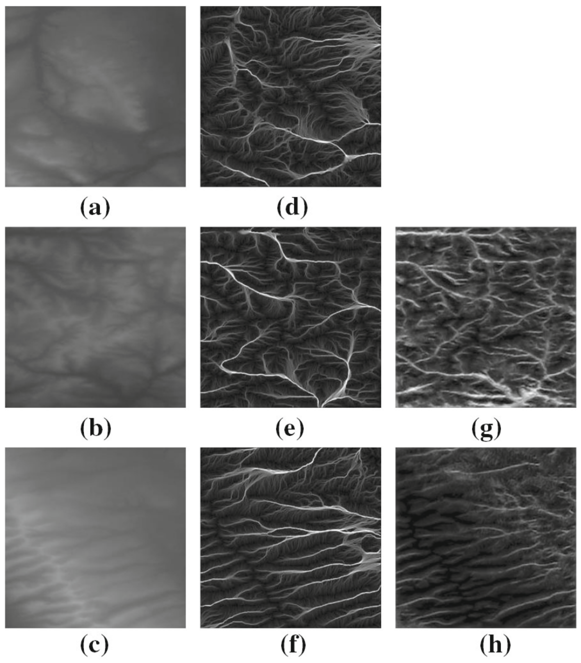


Fig. 11 The test over hydraulic erosion simulation. Given the terrain (**a**) and flow map (**d**), we generate new flow maps (**g**) and (**h**) with the terrain (**b**) and (**c**). **e** and **f** are produced with physically based method

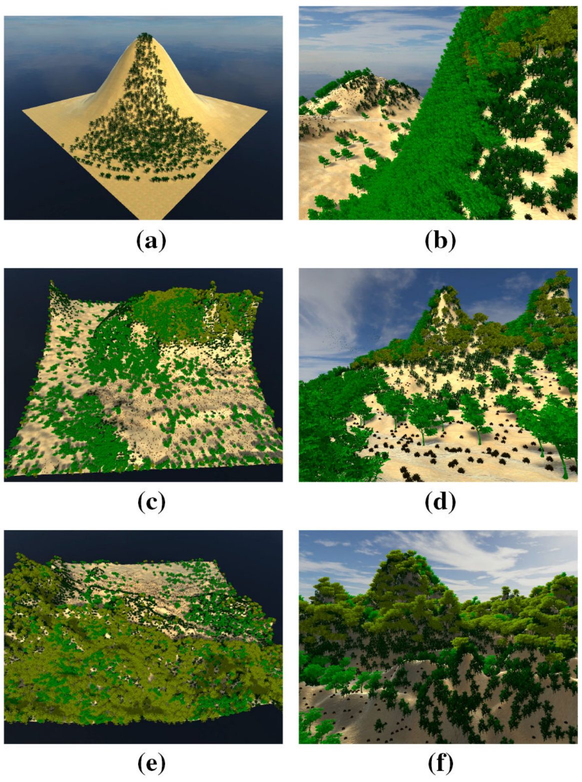


Fig. 12 One exemplar scene followed by 2 generated scenes. The first column is the aerial view, and the second column shows local details

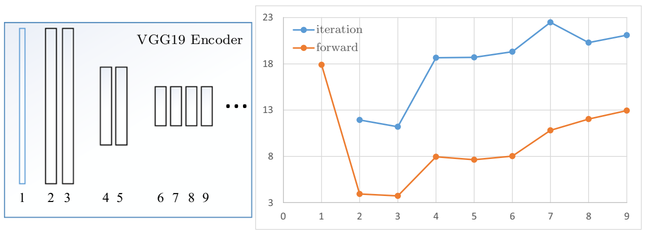


Fig. 13 The comparison of the iterative optimization method (where we only consider Eq. 3 as objective) and feed-forward method over different layers of VGG-Network (the first three convolution blocks). The ordinate of the line chart represents the MAE statistics ($\times 100$). 1 to 9 on the abscissa represent input image, 'conv1_1,' 'conv1_2,' 'conv2_1,' 'conv2_1,' 'conv3_1,' 'conv3_2,' 'conv3_3,' and 'conv3_4,' respectively

Statistics In general because of the unquantifiable human intention, it is difficult to measure an example-based generative model quantitatively. However, based on the data generation method proposed in Algorithm 1, we can partly test the model quantitatively. We simply use the mean absolute error (MAE) to measure the model base on some precomputed scenes, where the vegetation density ranges from 0 to 1. We prepare 100 scenes by using Algorithm 1. The terrains in these scenes are never seen in our training data set. As showcased in Fig. 13, the end-to-end network is comprehensively superior to the iterative optimization approach. The statistical results also show that low-level features are more useful to our task than high-level features. That is intuitive as high-level features with bigger receptive fields are not essential for the task in this paper. We also test the direct use of the preprocessing layer to manipulate the image itself. The result is worse than using VGG-Network, which also demonstrates the robustness of VGG-Network. Figure 14 visually shows the comparison of different layers.

We also document the performance of our model with different patch size (Table 2). As presented in Sect. 4.2, patch size directly affects the shapes of $F^t_{1,p}$, $\eta(F^t_2, F^t_1)$ and other tensors in the preprocessing layer. $\eta(F^t_2, F^t_1)$ takes heaviest memory footprint with a shape of $(N \times M \times (\frac{N}{k} \times \frac{M}{k}))$. As the preprocessing layer is based on convolution operation, patch size here works like filter size in convolutional layers, and it also affects MAE performance. Our model performs not well when patch is set to 1, where the similarity computation is based on pixel level.

Table 2 The model statistics on memory usage and MAE with different patch size

Patch size	1	2	3	4	5	6
Memory (MB)	1085	276	124	76	51	39
MAE (%)	29.2	18.2	3.8	9.6	7.9	8.7

For better comparison, the memory cost is computed with tensors in the preprocessing layer rather than the entire model

Limitations Our approach has some limitations. Firstly, our method considers terrain constraints on vegetation distribution but ignores other constraints such as temperature, latitude, etc. This is because terrain is the most commonly considered factor for vegetation distribution and our method can be easily applied to solving other local constraints. However, our method is based on low-level features extracted by pre-trained networks so we cannot take global constraints into account. For example, we cannot solve the spatial radiation effect of lakes or human buildings, and this may require the introduction of new anisotropic objective functions for the model. Another limitation is that the input terrain to model cannot be too monotonous. If the user wants the model to automatically plant trees at altitudes above 500 m, the user cannot provide exemplar terrain below 500 m. Moreover, our model requires more memory in the preprocessing layer than general CNN model. The preprocessing layer splits the input features into many patches and recombines them into new convolution kernel, which takes up a lot of memory and restricts model usage.

6 Conclusion and future works

This paper has introduced an example-based generation method for the vegetation distribution on terrain. We proposed a unified CNN-based model for producing reasonable distribution with exemplar scenes, with a goal of catching the correlationship between terrains and vegetation. Our model utilizes the rich local features extracted by pre-trained CNN and fuse input scene features to simplify model training. During the training stage, we procedurally generated vegetation distribution data based on multiple random transfer functions. The data generation algorithm was also used for model

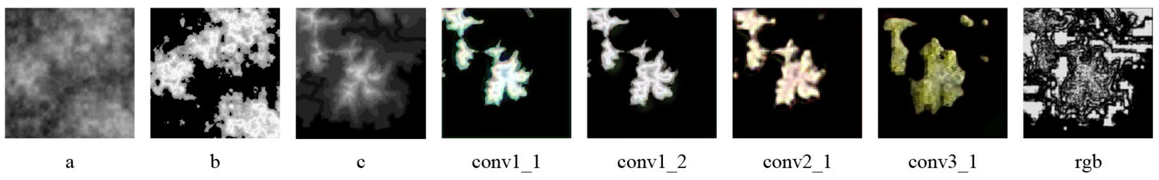


Fig. 14 The comparison of results that generated with different layers of VGG-Network. Given the exemplar scene (**a**, **b**), our model generates new distributions on terrain c with layer 'conv1_1,' 'conv1_2,' 'conv2_1,' 'conv3_1,' and original input ('rgb')

evaluation and validation, where it employed different terrain attributes for training and testing. The experimental results show that our method could learn the correlation between vegetation distribution and terrain and rapidly produce new realistic scenes. Our work also suggested that features learned by classical classification networks (e.g., VGG-19 in this paper) might work well for other types of tasks.

In the near future, we shall focus on considering more constraints on vegetation distribution such as light, water, latitude, etc. Moreover, we intend to explore more diversified natural information including discrete or vectorial features, for instance, buildings, rivers, lakes and road network. The generation approach for vegetation distribution could also be extended to generation of terrain textures or urban landscape planning.

Acknowledgements The authors would like to especially thank all reviewers for their sincere and thoughtful suggestions. This paper is partially supported by National Natural Science Foundation of China (No. 61532002, 61672237) and National Science Foundation of USA (IIS-1715985, IIS-1812606). The authors wish to thank the financial support from the program of China Scholarship Council No. 201806140115.

Funding This study was funded by National Natural Science Foundation of China (Nos. 61532002, 61672237), National Science Foundation of USA (IIS-1715985, IIS-1812606) and China Scholarship Council (No. 201806140115).

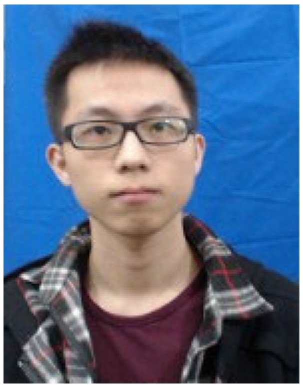
Compliance with ethical standards

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

References

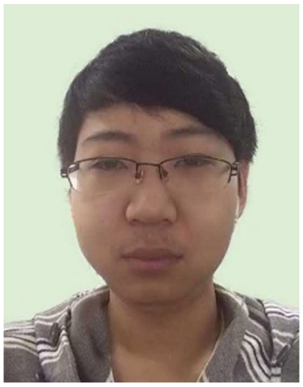
1. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). URL https://www.tensorflow.org/. Accessed 25 Aug 2018. Software available from tensorflow.org
2. Argudo, O., Andujar, C., Chica, A., Guérin, E., Digne, J., Peytavie, A., Galin, E.: Coherent multi-layer landscape synthesis. Vis. Comput. **33**(6–8), 1005–1015 (2017)
3. Beneš, B., Andrysco, N., Štava, O.: Interactive modeling of virtual ecosystems. In: Proceedings of the Fifth Eurographics conference on Natural Phenomena, pp. 9–16. Citeseer (2009)
4. Benes, B., Deussen, O., Pirk, S., Chen, B., Mech, R., Ijiri, T.: Modeling plant life in computer graphics. In: ACM SIGGRAPH 2016 Courses, SIGGRAPH '16, pp. 18:1–18:180. ACM (2016)
5. Beneš, B., Št'ava, O., Měch, R., Miller, G.: Guided procedural modeling. Comput. Graph. Forum **30**(2), 325–334 (2011)
6. Chen, T.Q., Schmidt, M.: Fast patch-based style transfer of arbitrary style. CoRR arXiv:1612.04337 (2016)
7. Chollet, F., et al.: Keras. (2015). https://keras.io. Accessed 25 Aug 2018
8. Cordonnier, G., Galin, E., Gain, J., Benes, B., Guérin, E., Peytavie, A., Cani, M.P.: Authoring landscapes by combining ecosystem and terrain erosion simulation. ACM Trans. Graph. **36**(4), 134 (2017)
9. Darabi, S., Shechtman, E., Barnes, C., Goldman, D.B., Sen, P.: Image melding: combining inconsistent images using patch-based synthesis. ACM Trans. Graph. **31**(4), 82:1–82:10 (2012)
10. Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., Prusinkiewicz, P.: Realistic modeling and rendering of plant ecosystems. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '98, pp. 275–286. ACM (1998)
11. Emilien, A., Vimont, U., Cani, M.P., Poulin, P., Benes, B.: Worldbrush: Interactive example-based synthesis of procedural virtual worlds. ACM Trans. Graph. **34**(4), 106 (2015)
12. Gatys, L.A., Ecker, A.S., Bethge, M.: Texture synthesis using convolutional neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS'15, pp. 262–270. MIT Press (2015). URL http://dl.acm.org/citation.cfm?id=2969239.2969269. Accessed 11 Oct 2018
13. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2414–2423 (2016)
14. 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. **36**(6), 228:1–228:13 (2017)
15. Huang, H., Kalogerakis, E., Yumer, E., Mech, R.: Shape synthesis from sketches via procedural models and convolutional networks. IEEE Trans. Vis. Comput. Graph. **23**(8), 2003–2013 (2017)
16. Huang, X., Belongie, S.J.: Arbitrary style transfer in real-time with adaptive instance normalization. CoRR arXiv:1703.06868 (2017)
17. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, pp. 448–456. JMLR.org (2015)
18. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision, pp. 694–711. Springer (2016)
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Kohek, Š., Strnad, D.: Interactive large-scale procedural forest construction and visualization based on particle flow simulation. Comput. Graph. Forum **37**(1), 389–402 (2018)
21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
22. Lane, B., Prusinkiewicz, P.: Generating spatial distributions for multilevel models of plant communities. In: Graphics interface, vol. 2002, pp. 69–87. Citeseer (2002)
23. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European Conference on Computer Vision, pp. 740–755. Springer (2014)
24. Martinovic, A., Van Gool, L.: Bayesian grammar learning for inverse procedural modeling. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 201–208 (2013)
25. Mei, X., Decaudin, P., Hu, B.: Fast hydraulic erosion simulation and visualization on gpu. In: 15th Pacific Conference on Computer Graphics and Applications, pp. 47–56. IEEE (2007). https://doi.org/10.1109/PG.2007.15
26. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning, pp. 807–814 (2010)
27. Nishida, G., Garcia-Dorado, I., Aliaga, D.G., Benes, B., Bousseau, A.: Interactive sketching of urban procedural models. ACM Trans. Graph. **35**(4), 130 (2016)

28. 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)
29. Ritchie, D., Jobalia, S., Thomas, A.: Example-based authoring of procedural modeling programs with structural and continuous variability. Comput. Graph. Forum **37**(2), 401–413 (2018)
30. Ritchie, D., Mildenhall, B., Goodman, N.D., Hanrahan, P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. ACM Trans. Graph. **34**(4), 105 (2015)
31. Ritchie, D., Thomas, A., Hanrahan, P., Goodman, N.: Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In: Advances in Neural Information Processing Systems, pp. 622–630 (2016)
32. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234–241. Springer (2015)
33. Samavati, F., Runions, A.: Interactive 3D content modeling for digital earth. Vis. Comput. **32**(10), 1293–1309 (2016)
34. Sendik, O., Cohen-Or, D.: Deep correlations for texture synthesis. ACM Trans. Graph. **36**(4), 161 (2017)
35. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
36. Smelik, R.M., De Kraker, K.J., Tutenel, T., Bidarra, R., Groenewegen, S.A.: A survey of procedural methods for terrain modelling. In: Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation, pp. 25–34 (2009)
37. Stava, O., Pirk, S., Kratt, J., Chen, B., Měch, R., Deussen, O., Benes, B.: Inverse procedural modelling of trees. Comput. Graph. Forum **33**(6), 118–131 (2014)
38. Talton, J.O., Lou, Y., Lesser, S., Duke, J., Měch, R., Koltun, V.: Metropolis procedural modeling. ACM Trans. Graph. **30**(2), 11 (2011)
39. Vanegas, C.A., Garcia-Dorado, I., Aliaga, D.G., Benes, B., Waddell, P.: Inverse design of urban procedural models. ACM Trans. Graph. **31**(6), 168 (2012)
40. Wang, K., Savva, M., Chang, A.X., Ritchie, D.: Deep convolutional priors for indoor scene synthesis. ACM Trans. Graph. **37**(4), 70 (2018)
41. Wang, L., Wang, Z., Yang, X., Hu, S.M., Zhang, J.: Photographic style transfer. Vis. Comput. pp. 1–15 (2018)
42. Wei, L.Y., Lefebvre, S., Kwatra, V., Turk, G.: State of the art in example-based texture synthesis. In: Eurographics 2009, State of the Art Report, EG-STAR, pp. 93–117. Eurographics Association (2009)
43. Wu, F., Yan, D.M., Dong, W., Zhang, X., Wonka, P.: Inverse procedural modeling of facade layouts. ACM Trans. Graph. **33**(4), 121 (2014)
44. Yumer, M.E., Asente, P., Mech, R., Kara, L.B.: Procedural modeling using autoencoder networks. In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, pp. 109–118. ACM (2015)
45. Zeiler, M.D., Taylor, G.W., Fergus, R.: Adaptive deconvolutional networks for mid and high level feature learning. In: Proceedings of the 2011 International Conference on Computer Vision, pp. 2018–2025. IEEE Computer Society (2011)
46. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Trans. Math. Softw. **23**(4), 550–560 (1997)



Jian Zhang is currently a Ph.D. student of School of Computer Science and Software Engineering, East China Normal University, China. He received his BE degree in software engineering from East China Normal University in 2014. His research interests include procedural modeling and sketch-based modeling.

Changbo Wang is a professor of School of Computer Science and Software Engineering, East China Normal University, China. He received his Ph.D. degree at the State Key Lab. Of CAD & CG, Zhejiang University in 2006, and received BE degree in 1998 and ME degree in civil engineering in 2002, respectively, both from Wuhan University of Technology. His research interests include physically based modeling and rendering, computer animation and realistic image synthesis, information visualization and others.



Chen Li is currently a Ph.D. student of School of Computer Science and Software Engineering, East China Normal University, China. He received his BE degree in computer science from Tianjin University in 2013. His research interests include physically based animation and virtual reality.

Hong Qin is a Full Professor of Computer Science in Department of Computer Science at State University of New York at Stony Brook (Stony Brook University). He received his BS (1986) degree and his MS degree (1989) in Computer Science from Peking University in Beijing, China. He received his Ph.D. (1995) degree in Computer Science from the University of Toronto. His research interests include computer graphics, geometric and physics-based modeling, computer-aided design, computer-aided geometric design, computer animation and simulation, virtual environments and virtual engineering, and others.