On Demand Solid Texture Synthesis Using Deep 3D Networks

Jorge Gutierrez, Julien Rabin, Bruno Galerne, Thomas Hurtut

SOLID TEXTURE GENERATIVE NETWORK FROM A SINGLE IMAGE

Jorge Gutierrez[⋆] *Julien Rabin*[◇] *Bruno Galerne*[†] *Thomas Hurtut*[⋆]

[⋆]Polytechnique Montréal, Montréal, Canada
[◇] Normandie Univ., UniCaen, ENSICAEN, CNRS, GREYC, Caen, France
[†]Laboratoire MAP5, Université Paris Descartes and CNRS, Sorbonne Paris Cité

ABSTRACT

This paper addresses the problem of generating volumetric data for solid texture synthesis. We propose a compact, memory efficient, convolutional neural network (CNN) which is trained from an example image. The features to be reproduced are analyzed from the example using deep CNN filters that are previously learnt from a dataset. After training the generator is capable of synthesizing solid textures of arbitrary sizes controlled by the user. The main advantage of the proposed approach in comparison to previous patch-based methods is that the creation of a new volume of data can be done in interactive rates.

***Index Terms*—** Texture synthesis, convolutional neural networks, volumetric data generation, solid textures.

1. INTRODUCTION

Motivation and context Besides imaging problems which deal with volumetric data (such as MRI in medical imaging, or 3D seismic geological data), the main application of solid texture synthesis originates from computer graphics. Since [1, 2], this approach has been proposed to create 3D objects as an alternative to texture mapping or synthesis *onto* 3D meshes. It is an efficient and elegant way of dealing with many problems that arise with the later approaches, such as time consuming artist design, mapping distorsion and seams, *etc*. Once the volumetric information is generated, surfaces can be carved inside without further parametrization no matter how intricate they are. However, several reasons still limit the use of such techniques in practice: *i)* storing volumetric data is prohibitive (HD quality requires several GB of memory); *ii)* learning to generate such data is a difficult task as there is no 3D examples; *iii)* previous synthesis approaches based on optimizing a matching criterion do not achieve real-time performance.

In this work, we investigate the use of Convolutional Neural Networks (CNN) for solid texture synthesis. The general principle of CNN is first to off-line train a network to achieve a specific task, usually by optimizing its parameters given a loss function. Then, at run-time, the network is applied to new images to perform the task which is often achieved in real-time.

Contributions We introduce a solid texture generator based on convolutional neural networks that is capable of synthesizing solid textures from a random input at interactive rates. Taking inspiration from previous work on image texture synthesis with CNN, a framework is proposed for the training of this solid texture generator. A volumetric loss function based on deep CNN filters is defined by comparing multiple viewpoint slices of the generated textures to a single example image. This model ensures that each cross-section of the generated solid shares the same visual characteristics as the input example. To the best of our knowledge, there is no such architecture proposed in the literature to generate volumetric color data.

Outline The paper is organized as follows: Section 2 gives a short overview of the literature about solid texture synthesis methods and 2D texture synthesis using neural networks, Section 3 describes the proposed generative model and Section 4 gives the implementation details and some visual results.

2. RELATED WORKS ON TEXTURE SYNTHESIS

Let us give a quick review on example-based texture synthesis, first with a focus on solid texture synthesis and second on convolutional networks for image generation.

2.1. Solid texture synthesis

Procedural texture synthesis Procedural methods are a very special category which consists of evaluating a specific noise function depending on the spatial coordinates to achieve real-time synthesis with minimal memory requirements [2, 3]. Unfortunately, these methods often fail at reproducing real-world examples with structured features.

Example-based solid texture synthesis Most successful methods focus on generating volumetric color data (grids of voxels) whose cross-sections are visually similar to a respective input texture example.

This work was partially funded by CONACYT and I2T2 Mexico

In [4, 5] an analog "slicing" strategy is used, where the texture is generated iteratively, alternating between independent 2D synthesis of the slices along each of the three orthogonal axes of the cartesian grid, and 3D aggregation by averaging. The main difference between the two approaches is the features to be considered: steerable filters coefficients and laplacian pyramid in [4] and spectral information in [5].

Generalizing some previous work on texture synthesis, Wei [6] designed an algorithm which consists of modifying the voxels of a solid texture initialized randomly by copying the average color values from an image in a raster scan order based on the most similar neighborhood extracted along each axis. This strategy has been refined and accelerated by Dong et al. [7] by pre-processing compatible 3D neighborhoods accordingly to the examples given along some axis.

Closely related to these methods, patch-based approaches [8, 9] iteratively modify the local 2D neighborhood of a voxel to match similar neighborhoods in examples, under statistical constraints on colors or spatial coordinates.

While recent methods achieve state-of-the-art results, their main limitation is the required computation time due to iterative update of each voxel, (several minutes except for [7]) thus prohibiting real-time applications.

2.2. Neural networks on texture synthesis

Several methods based on deep CNN have been recently proposed for 2D texture synthesis that outperform previous ones based on more "shallow" representation (wavelets, neighborhood-based representations, *etc*). In the following, we present and discuss the most successful of such architectures.

Image optimization Gatys et al. [10] proposed a variational framework inspired from [11] to generate an image which aims at matching the features of an example image. The most important aspect of this framework is that the discrepancy is defined as the distance between Gram matrices of the input and the example images; these matrices encode the filters' responses of an image at some layers of a deep convolutional neural network which is pretrained on a large image dataset [12]. Starting from a random initialization, the input image is then optimized using back-propagation and a stochastic gradient descent algorithm.

Since then, several variants have been using this framework to improve the quality of the synthesis: for structured textures by adding a Fourier spectrum discrepancy [13]; for non-local patterns by considering spatial correlation and smoothing [14]; for stability by considering a histogram matching loss and smoothing[15].

The main drawback of such strategy comes from the optimization process itself, as it requires several minutes to generate one image.

Feed-forward texture networks The principle introduced by Ulyanov et al. [16] to circumvent this issue is to train a network to generate texture samples from random inputs that produce the same visual features as the texture example. As in the previous framework, the objective loss function to be minimized relies on the comparison between feature activations in a pre-trained discriminative CNN from the output and the texture example. However, this time, instead of optimizing the generated output image the training aims at tuning the parameters of the generative network. Such optimization is more intensive as much more variables are now involved, but it only needs to be done once and for all. This is achieved in practice using back propagation and a gradient-based optimization algorithm using batches of random inputs. Once trained, the generator is able to quickly generate samples similar to the input example by forward evaluation. Feed-forward texture networks methods generally produce results with slightly lower quality than the image optimization using CNN ones but it exhibits computation times extremely shorter.

Originally these methods train one network per texture sample. Li et al. [17] proposed a training scheme to allow one network to have the capacity to generate and mix several different textures. In order to prevent the generator from producing identical images, they also incorporate in the objective function a term that encourages diversity in synthesized batches, similarly to [18].

Aside from texture synthesis, other computer vision applications dealing with volumetric benefit from CNN. As an exemple, [19] proposed recently a GAN for 3D shape inpainting and [20] proposed Octree Networks for 3D shapes generation.

In the more specific problem of generating 3D objects from several 2D view images which is close to the problem considered in this work, deep learning methods have also been recently introduced, including [21] for dynamic facial texture, and [22, 23, 24] for 3D object generation from 2D views.

3. SOLID TEXTURE NETWORKS

Similar to the 2D feed-forward texture networks, our method consists of a solid texture generator and a training framework incorporating a descriptor network and a 3D loss. One generator is to be trained for each input example, with most of the computations occurring in that process. Once the generator is trained it can be used to quickly generate new solid samples of arbitrary sizes (only limited by the available memory). Here we describe each element of the framework.

3.1. Generator

The generator is a feed-forward network that takes a noise input $z \in \mathbb{R}^{\frac{N_x}{2^k} \times \frac{N_y}{2^k} \times \frac{N_z}{2^k} \times N_c}$, and produces a solid texture sample $g(z) \in \mathbb{R}^{N_x \times N_y \times N_z \times 3}$. The proposed architecture is illustrated in Figure 1, it consists of $k+1$ *filter blocks*, k up-sampling operators and a final single volumetric convolutional layer that maps the number of channels to three.

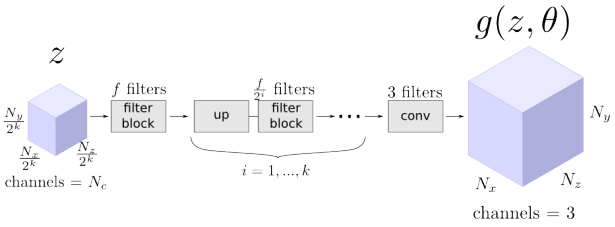

Fig. 1. Architecture of the generator network.

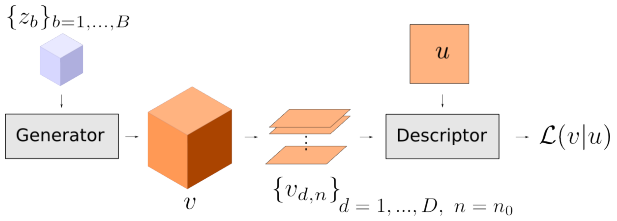

Fig. 2. Architecture of the training framework.

The *filter blocks* depict three volumetric convolutional layers, each one of them followed by a batch normalization layer and a ReLU activation function. The sizes of the filters are $3 \times 3 \times 3$, $3 \times 3 \times 3$ and $1 \times 1 \times 1$ similar to what previous methods do for 2D textures. *up* represents an up-sampling of the solid by a factor of 2 using a nearest neighbor interpolation. The first *filter block* uses f filters and the central pair *filter block - up* is applied k times with $\frac{f}{2^i}$, $i = 1, ..., k$ filters.

The architecture of the generator is particularly useful for texture synthesis because the size of the output only depends on the size of the noise input used. This allows us to train a generator with a given size of input image and afterwards generate solids of different sizes.

3.2. 3D slice-based loss

As each *slice* (cross-section) of the generated solid should capture the visual characteristics of the 2D input example it is straightforward to define a slice-based 3D loss. The proposed loss simply aggregates the 2D losses produced by each one of the slices of the solid. For a color solid $v \in \mathbb{R}^{N_x \times N_y \times N_z \times 3}$ we consider D slicing directions with the corresponding N_d number of slices. We then define a slice of the solid $v_{d,n}$ as the 2D slice n of v orthogonal to the direction d. Given the input image u we propose the following slice based loss:

$$\mathcal{L}(v|u) = \sum_{d \in \{1..D\}} \frac{1}{N_d} \sum_{n=1}^{N_d} \mathcal{L}_2(v_{d,n}, u) \tag{1}$$

where $\mathcal{L}_2(\cdot, u)$ is a 2D loss that computes the similarity between an image and the input image u.

3.3. Descriptor network

We reproduce the strategy of previous methods of using a pre-trained neural network to compute and back-propagate a 2D loss that successfully compares the image statistics. A popular way is to consider the Gram matrix loss computed on a descriptor network between the input example u and a slice $v_{d,n}$ is defined as:

$$\mathcal{L}_2(v_{d,n}, u) = \sum_{l \in L} \left\| G^l(v_{d,n}) - G^l(u) \right\|_F^2 \tag{2}$$

where G^l is the Gram matrix computed as the scalar product $G^l_{i,j}(x) = \langle F^l_i(x), F^l_j(x) \rangle_{\mathbb{R}^{\Omega_\ell}}$ between the i^{th} and j^{th} feature maps, and $F : x \in \mathbb{R}^{\Omega \times 3} \mapsto \{F^\ell_i(x) \in \mathbb{R}^{\Omega_\ell}\}_{\ell \in L, i \in W_\ell}$ is the feature map associated to x and L is a set of layers on the descriptor network that are taken in account for the loss.

3.4. Training

We train the generator network g to generate solid textures whose cross-sections are similar to the corresponding input u. The training of the generator $g(z, \theta)$ with parameters θ consists in minimizing the expectation of the loss in Equation 1 given the example u :

$$\theta_u \in \underset{\theta}{\operatorname{argmin}} \quad \mathbb{E}_{Z \sim \mathcal{Z}} \left[\mathcal{L}(g(Z, \theta), u) \right] \tag{3}$$

where Z is a white noise sample from a certain distribution $\mathcal{Z}$. However, since the distribution of $\mathcal{Z}$ is invariant by translation and the generative network is (approximatively) invariant by translation (disregarding border effects and the non strict stationarity of the upscaling), for each slicing direction $d \in \{1..D\}$, the N_d slices $g(Z, \theta)_{d,n}$ have the same distribution. Hence, for all fixed slice index $n_0 \in \{1..N_d\}$,

$$\begin{aligned} &\mathbb{E}_{Z \sim \mathcal{Z}} \left[\mathcal{L}(g(Z, \theta), u) \right] \\ &\quad = \sum_{d \in \{1..D\}} \frac{1}{N_d} \sum_{n=1}^{N_d} \mathbb{E}_{Z \sim \mathcal{Z}} \left[\mathcal{L}_2(g(Z, \theta)_{d,n}, u) \right] \\ &\quad = \sum_{d \in \{1..D\}} \mathbb{E}_{Z \sim \mathcal{Z}} \left[\mathcal{L}_2(g(Z, \theta)_{d,n_0}, u) \right]. \end{aligned} \tag{4}$$

This means that thanks to the stationarity of the generative model, minimizing in expectation the loss of Equation 1 that has $D \times N_d$ terms can be done by only computing one loss term per slicing direction. In practice we use the central slice index $n_0 = N_d/2$ to avoid boundary issues. Note that computing the loss for all slices requires more memory.

The training framework is illustrated in Figure 2. In short it consists of extracting the central slice of a generated solid $v = g(Z, \theta)$ and sum the D 2-D losses to compute a stochastic gradient estimation of the full 3-D loss of Equation 1. This framework allows the simple back propagation of the gradient to the generator network, and each slice of the volume model is impacted since the convolution weights are shared by all slices.

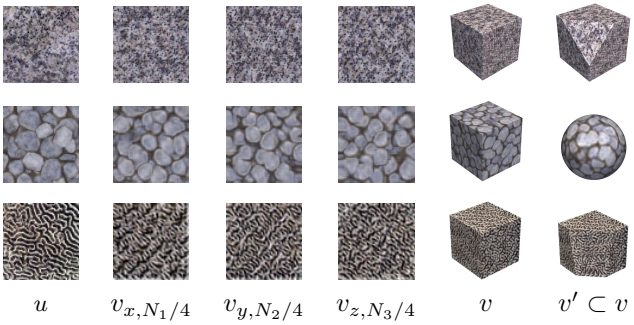

Fig. 3. Three examples of solid generation. From left to right: example, slices of the solid along each axis at $n = N_d/4$, full solid cube v and a subset v' of v. From top to bottom: $f = 64, 48, 64$. The sizes are 128^3 for the first row and then 64^3 for the others.

4. RESULTS AND DISCUSSION

4.1. Experimental setting

Our framework is based on the code for [16], using *Torch7*. For our experiments, we restrict the number of slicing directions to three orthogonal axes ($D = 3$ for each example) and for simplicity we set $N_x = N_y = N_z$. We train our generator with textures of 64^2 and 128^2 pixels. During training we generate cubic solids with the same width as the example.

Generator We use a network with two up-samplings ($k = 2$). Each convolution is applied with a step of 1 and when the filter is bigger than $1 \times 1 \times 1$ we use replicating padding to preserve the size of the sample and lessen boundary artifacts. We sample the noise inputs Z from a uniform distribution with 32^3 values, therefore the number of channels N_c is adjusted to the training size $N_x \times N_y \times N_z$ of the solid and k.

Discriminator We use the VGG19 [12] as discriminator network with the set of loss layers $L = \{$relu1_1, relu2_1, relu3_1, relu4_1, relu5_1$\}$ similar to [16]. Before feeding the input example to the discriminator it has to be pre-processed by subtracting the mean of the dataset the discriminator was trained with. Likewise, we need to post-process the generated samples by adding the same mean values.

Training We train the generator using stochastic gradient descent, at each iteration sampling a batch $\{z_b\}_{b=1..B}$ with $B = 2$. We use 3000 iterations with a learning rate starting at 0.01 and decreasing by $\times 0.8$ each 200 iterations after 500 iterations.

4.2. Results

Figure 3 shows some examples of solid texture generation using our method. We show one slice of the generated solid for each orthogonal axis, the whole generated cube and a subset of the cube showing an oblique cut. The generator is able to synthesize color information that is coherent through the whole volume. The structure elements of the generated solids take a plausible shape. However note that for some cases it might not exist a *good* structure given the input example. The computation time is respectively around 0.1, 1 and 5 seconds for a 64^3, 128^3, and 200^3 texture with a generator of $f = 64$ using a GPU Tesla K40M 2880 Cores 12G RAM. While some of the aforementionned methods report having issues with generating samples that are different from each other [17, 18], our model generates output with satisfying diversity (see Figure 4).

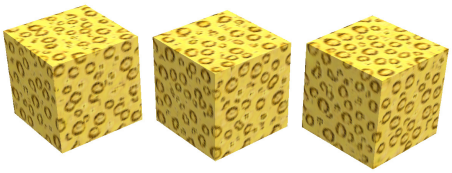

Fig. 4. Diversity of the generated solids with the same generator (size 64^3) and different inputs. This time, the generator is learnt using odd slices ($n \in \{1, 3..63\}$ instead of only $n_0 = 32$), showing that the model does not rely on the number of slices, as demonstrated in Eq. 4.

As mentioned before, once the generator is trained, the size of the sample can be controlled by the size of the input noise. In Figure 5 we show two samples of different size generated with the same network.

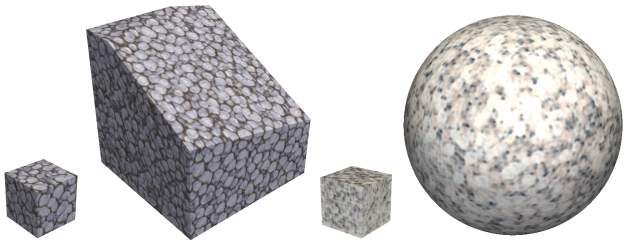

Fig. 5. Solid textures synthesized with the same trained generator ($f = 48$) with size 64^3 or 200^3.

4.3. Discussion

We have proposed a new approach to solid texture generation that may become an important tool for computer graphics applications requiring fast processing. Our approach moves the computational burden to an off-line learning stage to provide a compact network containing only the information required to generate a volumetric data of an arbitrary size having the desired visual features. Moreover, contrarily to patch-based approaches, the synthesized texture is not based on local copies of the input. Finally, the quality of the generated textures depends mostly on the computing resources (*e.g* limiting the number of filters stored in memory).

5. REFERENCES

[1] D. R. Peachey, "Solid texturing of complex surfaces," in *SIGGRAPH Computer Graphics*. ACM, 1985, vol. 19, pp. 279–286. 1

[2] K. Perlin, "An image synthesizer," *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985. 1

[3] N. Pavie, G. Gilet, J-M. Dischler, E. Galin, and D. Ghazanfarpour, "Volumetric spot noise for procedural 3d shell texture synthesis," in *Computer Graphics & Visual Computing*. Eurographics Association, 2016, pp. 33–40. 1

[4] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *SIGGRAPH*. 1995, pp. 229–238, ACM. 2

[5] J.M. Dischler, D. Ghazanfarpour, and R. Freydier, "Anisotropic solid texture synthesis using orthogonal 2d views," *Computer Graphics Forum*, vol. 17, no. 3, pp. 87–95, 1998. 2

[6] L.-Y. Wei, "Texture synthesis from multiple sources," in *SIGGRAPH*, New York, NY, USA, 2003, SIGGRAPH '03, pp. 1–1, ACM. 2

[7] Y. Dong, S. Lefebvre, X. Tong, and G. Drettakis, "Lazy solid texture synthesis," in *Eurographics Conference on Rendering*. 2008, EGSR, pp. 1165–1174, Eurographics Association. 2

[8] J. Kopf, C. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T. Wong, "Solid texture synthesis from 2d exemplars," in *SIGGRAPH*. 2007, ACM. 2

[9] J. Chen and B. Wang, "High quality solid texture synthesis using position and index histogram matching," *Vis. Comput.*, vol. 26, no. 4, pp. 253–262, 2010. 2

[10] L. Gatys, A. S Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *NIPS*, pp. 262 – 270. 2015. 2

[11] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *IJCV*, vol. 40, no. 1, pp. 49–71, 2000. 2

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. 2, 4

[13] G. Liu, Y. Gousseau, and G.-S. Xia, "Texture synthesis through convolutional neural networks and spectrum constraints," in *ICPR*. IEEE, 2016, pp. 3234–3239. 2

[14] O. Sendik and D. Cohen-Or, "Deep correlations for texture synthesis," *ACM Trans. Graph.*, vol. 36, no. 5, pp. 161:1–161:15, 2017. 2

[15] P. Wilmot, E. Risser, and C. Barnes, "Stable and controllable neural texture synthesis and style transfer using histogram losses," *CoRR*, vol. abs/1701.08893, 2017. 2

[16] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images," in *ICML*, 2016, pp. 1349–1357. 2, 4

[17] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," *CoRR*, vol. abs/1703.01664, 2017. 2, 4

[18] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," *CoRR*, vol. abs/1701.02096, 2017. 2, 4

[19] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann, "Shape inpainting using 3d generative adversarial network and recurrent convolutional networks," in *ICCV*. IEEE, 2017, pp. 2298–2306. 2

[20] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," *arXiv preprint arXiv:1703.09438*, 2017. 2

[21] K. Olszewski, Z. Li, C. Yang, Y. Zhou, R. Yu, Z. Huang, S. Xiang, S. Saito, P. Kohli, and H. Li, "Realistic dynamic facial textures from a single image using gans," in *ICCV*, 2017, pp. 5429–5438. 2

[22] J. Gwak, C. B. Choy, A. Garg, M. Chandraker, and S. Savarese, "Weakly supervised generative adversarial networks for 3d reconstruction," *CoRR*, vol. abs/1705.10904, 2017. 2

[23] D. Jimenez Rezende, S. M. A. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, and N. Heess, "Unsupervised learning of 3d structure from images," in *NIPS*, pp. 4996–5004. 2016. 2

[24] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, "Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision," in *NIPS*, pp. 1696–1704. 2016. 2