Structure Preserving Exemplar-Based 3D Texture Synthesis

Andrew Babichev [1] and Vladimir Frolov [1,2]

[1] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[2] *Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow, 125047, Russia*

Abstract

In this paper we propose exemplar-based 3D texture synthesis method which unlike existing neural network approaches preserve structural elements in texture. The proposed approach does this by accounting additional image properties which stand for the preservation of the structure with the help of a specially constructed error function used for training neural networks. Thanks to the proposed solution we can apply 2D texture to any 3D model (even without texture coordinates) by synthesizing high quality 3D texture and using local or world space position of surface instead 2D texture coordinates (fig. 1). Our solution is based on introducing 3 different error components in to the process of neural network fitting which helps to preserve desired properties of generated texture. The first component is for structuredness of the generated texture and the sample, the second component increases the diversity of the generated textures and the third one prevents abrupt transitions between individual pixels.

Keywords

3D texture synthesis, neural network, exemplar-based texture synthesis, structure preserving.

1. Introduction

Textures are one of the main components of realistic image synthesis. Exemplar based texture synthesis is used for generating new textures of a desired resolution which has similar appearance with the input texture but different pixels (like different parts of the road of the wall).

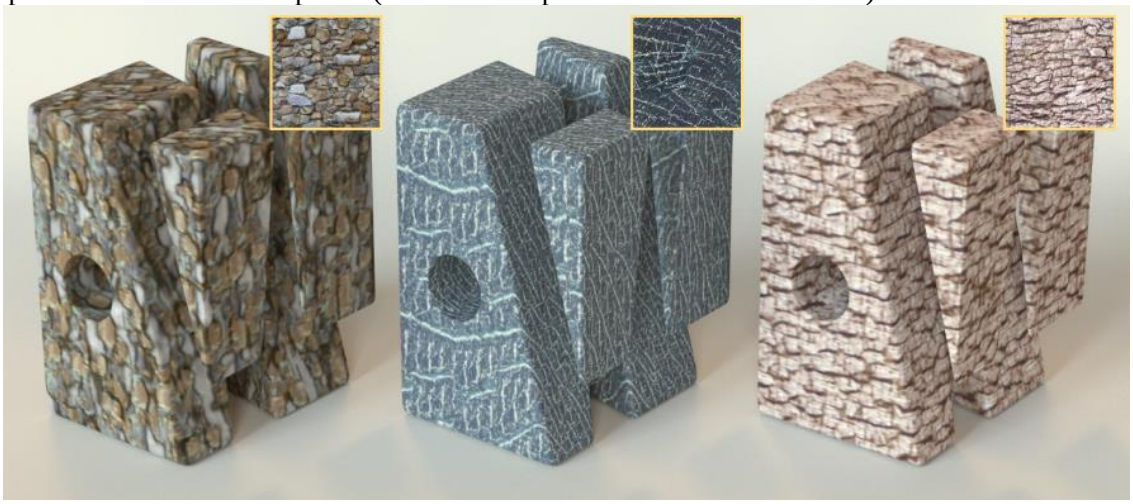

Figure 1: Examples of input images (in yellow squares) and synthesized 3D textures applied to 3D model

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: andrey.babichev@graphics.cs.msu.ru (A. Babichev); vfrolov@graphics.cs.msu.ru (V. Frolov)
ORCID: 0000-0003-4371-8066 (A. Babichev); 0000-0001-8829-9884 (V. Frolov)

In this work we propose new method for exemplar-based 3D texture synthesis. A 3D texture can be thought of as a voxel grid, where each voxel contains a certain color and each plane cut in any direction of which is a regular 2D texture (fig. 1).

2. Related work

When studying existing methods of texture synthesis it is needed to consider two principal problems: (1) how to define similarity between exemplar and generated images and (2) how to generate the new texture. Most existing methods for exemplar-based texture synthesis can be divided into two categories.

The *first group* of methods defines a function of the properties of the exemplar texture, and then tries to modify a random noise texture until property functions of generated and exemplar images become similar to the function of the properties of the exemplar texture. Either neural network based approaches and pyramidal synthesis can be attributed to that group [15]. As the pronounced properties of this group of methods, one can note high variability of the synthesized textures, low speed, as well as in many cases poor results when working with textures that have high resolution or complex structure. This methods ignore large details and poorly preserve structural features. Some of these shortcomings have been eliminated in recent works by consistently doubling the resolution of the generated textures [6,8].

The *second group* of methods extracts some patches from original texture and reorders them to generate the new texture. Typical representatives of this group of methods are per-pixel [16] and per-patch synthesis [17, 18]. As a characteristic of this group, one can designate the high speed and acceptable results on texture sampling. The disadvantage of this group of methods is the low variability of the synthesized textures.

2.1. Neural network sythesis

Most existing neural-network based texture synthesis methods [1-5] uses VGG-19 [9]. Neural network generators first define the image property function. For this, an exemplar of texture is fed into the neural network, and the activation function is calculated for each layer l of the neural network. Each activation function generates a certain set of filtered images - the so-called *feature maps*. Each layer with N_l filers will have same amount of feature maps, each of which has a total dimension M_l. Therefore, all feature maps can be stored in matrix $F^l \in R^{N_l \times M_l}$, where F^l_{jk} – is an activation for j filter at spatial position k inside layer l.

After finding all feature maps, for each of them we can calculate the Gram matrix $G^l \in R^{N_l \times N_l}$:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}. \quad (1)$$

A set of Gram matrices $\{G^1, G^2 \ldots, G^L\}$ of neural network layers *1, 2, ..., L* is a possible way to describe the properties of an image. Then, to define the *similarity function* of the exemplar texture and the synthesized texture, we can use the following error L_G:

$$L_G(g, s) = \sum_l \omega^l_G \frac{1}{N_l M_l} \left\| G^l(g) - G^l(s) \right\|^2_F, \quad (2)$$

where g и s – generated texture and source texture respectively; ω^l_G – contribution of each layer l of neural network to the error L_G, $\| \quad \|_F$ – Frobenius norm.

Thus, to synthesize a new texture, we need to minimize the specified error L_G between exemplar texture and the random noise texture. This can be achieved by applying gradient descent to the noise texture while calculating the gradients using backpropagation of errors.

2.2. 3D texture synthesis

2.2.1. 3D texture generator concept

A similar approach to the two-dimensional case is used in the synthesis of three-dimensional textures [10-14]. The algorithm can be described as follows: we have a certain texture generator based on convolutional neural networks and which depends on a fixed set of parameters θ. Three-dimensional white noise of several different resolutions K is fed to this generator, and the resolution of the synthesized texture will depend on the resolution of the supplied noise. Passing noise through itself, the generator will produce a three-dimensional texture of the specified resolution.

During training the generated texture will be divided into all possible planes along the three directions of the coordinate axes, and using some function that describes the properties of the image, it will calculate the error between the planes representing generated 3D texture and the exemplar textures we have. Setting the necessary parameters of the generator θ to generate a high-quality texture will be done using gradient descent and back propagation of the error similar to the two-dimensional case.

2.2.2. Generator architecture

The generator architecture is shown in fig. 2. It is a sequence of convolutional blocks, upscaling blocks, and concatenation blocks:

- A convolution block is a sequence of 3 convolutional layers, the first two of which have 3x3x3 kernels, and the last - 1x1x1. This is followed by the batch normalization layer [20] and the Relu activation function [21]. Thus, after this layer, the size of the texture is reduced by 4.
- The upscale block doubles 3D texture resolution (each voxel will be copied 8 times).
- The concatenation block performs batch normalization and then concatenates our textures into channels. If the textures are Are different in size then they are cropped to a smallest size.

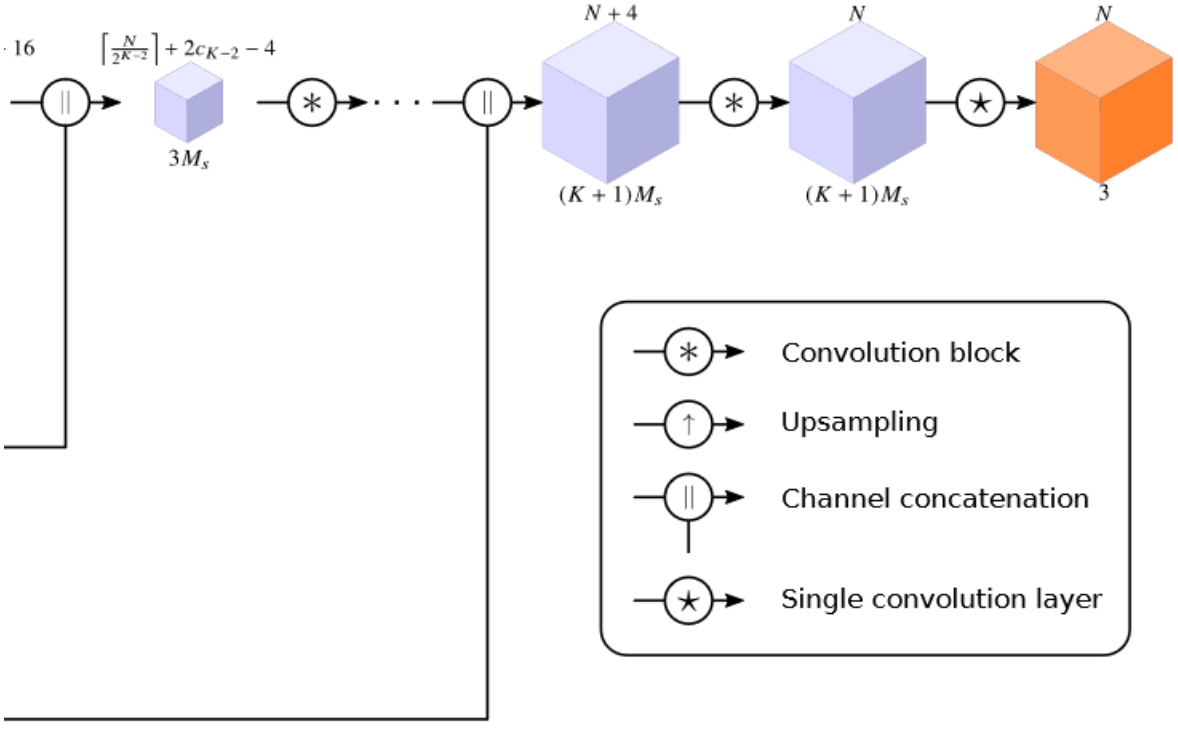

Figure 2: Architecture of a neural network generator of three-dimensional textures: ↑ – upscale block, - convolutional block, V – concatenation block, ⋆ – convolutional layer. At the top of each cube - its spatial size, at the bottom - the number of channels. Our scheme is similar to [10]. Please refer to fig.2 in [10] to see original image in full resolution.

Starting at the smallest scale, the input noise is processed by a set of convolutions, followed by an upscaling block to reach the next scale. It is then combined with independent noise of the same size, which is also pre-passed through the convolutional block. This process is repeated K times before the final single convolution layer, which yields three channels to obtain a colored texture. The parameters that the generator will learn weights of convolutional kernels and their biases, as well as weights, biases, mean-variance of batch normalization layers.

2.2.3. Parameters fitting

As already mentioned, to calculate the similarity between the three-dimensional texture obtained by the generator and the exemplar texture we have, the first one will be divided into all possible planes in all directions of the coordinate axes. Then the error can be written as:

$$L(g,s) = \sum_{d=1}^{D} \frac{1}{N_d} \sum_{n=0}^{N_d-1} L_2(g_{d,n}, s); \tag{3}$$

$$L_2(g_{d,n}, s) = L_G(g_{d,n}, s), \tag{4}$$

where d – means axis (x, y or z) in which 2D texture was split, N_d – number of planes for axis d in which the texture was split, $g_{d,n}$ – n-*th* plane in the d direction of the coordinate axis of the synthesized three-dimensional texture and L_2 – is an error implying the difference between two-dimensional texture exemplars. For calculation of activation maps that are required by our error function, we use the same neural network as in VGG-19 paper.

3. Proposed method

Using just a single Gram matrix to calculate the similarity between two-dimensional samples of textures is not an optimal approach since the Gram matrix itself poorly represents such aspects of the texture as the presence of structure, smooth inter-pixel transitions, and many others. For this reason, the existing exemplar-based methods [10, 11] of three-dimensional synthesis poorly work for input textures with complex structure: the resulting texture looks “broken”.

To solve these problems of the two-dimensional synthesis, in [2] it was proposed to generate a texture only not using the Gram matrix, but also a combination of several error components; moreover, an approach that allows preserving the structure of synthesized textures was presented in [3, 4, 8]. We have further developed idea from [2] to solve 3D synthesis problems and *added three additional error components* to train the generator proposed in [10]: using the *first component*, we have compared the structuredness of the generated texture and the sample. In fact, this error is the autocorrelation of feature maps of the given layers of the neural network, multiplied by the coefficient:

$$L_C(g_{d,n}, s) = \sum_{l} \frac{1}{H_l W_l} \left\| R^l(g_{d,n}) - R^l(s) \right\|_F^2; \tag{5}$$

$$R_{i,j}^{l,n} = \sum_{q,m} w_{i,j}^{l} F_{n,(q,m)}^{l} F_{n,(q-i,m-j)}^{l}; \tag{6}$$

$$w_{i,j}^{l} = \frac{1}{(H_l - |i|)(W_l - |j|)}; i \in \left[\frac{-H_l}{2}, \frac{H_l}{2}\right]; j \in \left[\frac{-W_l}{2}, \frac{W_l}{2}\right], \tag{7}$$

where H_l и W_l – spatial sizes of feature maps F^l ($H_l * W_l = M_l$. Also for coefficient w^l Gauss window could be used. The *second introduced* component of the error function increases the diversity of the generated textures and is the usual difference between feature maps:

$$L_D(g_{d,n}, s) = \sum_l \left\| F^l(g_{d,n}) - F^l(s) \right\|_F^2. \quad (8)$$

The *third component* prevents abrupt transitions between individual pixels and is some kind of anti-aliasing:

$$L_S(g_{d,n}, s) = \sum_l \left\| S^l(g_{d,n}) - S^l(s) \right\|_F^2; \quad (9)$$

$$S_{i,j}^{l,n} = \frac{1}{2\sigma} log \sum_{\delta i, \delta j} exp\left(-\sigma\left(F_{n,(i,j)}^l - F_{n,(\delta i, \delta j)}^l\right)^2\right), \quad (10)$$

where σ – is a parameter of and algorithm.

Therefore, final difference between two 2D exemplars is:

$$L_2(g_{d,n}, s) = \alpha L_G(g_{d,n}, s) + \beta L_C(g_{d,n}, s) + \eta L_D(g_{d,n}, s) + \gamma L_S(g_{d,n}, s), \quad (11)$$

The values of the coefficients for the weights of the individual elements of the error function are given in Table 1.

Table 1
Parameter values, used by us for texture synthesis

Loss type	Layer names	Layer weights	Loss weight	Miscellaneous
L_G	Relu11, Relu21, Relu31, Relu41, Relu51	0.2, 0.2, 0.2, 0.2, 0.2	α=0.5	-
L_C	Pool2	1	β=0.5*1e-6	-
L_D	Pool2	1	η=-1*1e-4	-
L_S	Relu11	1	γ=-1*1e-3	σ=1e-3

4. Experimental evaluation and comparison

To test and compare the proposed method, the following two groups of textures were used:

1. Textures were selected in *the first group* (Fig. 3,4) in order to find out how high-quality and original the textures synthesized by the generator, and also whether a proposed error components interferes with the synthesis. The textures were chosen, consisting of a monochrome background with some chaotic picture on top of it. These textures were chosen with the idea that the generator should work reasonably on this type of textures due to its simplicity, but the result may be too similar to the original texture or differ in random outbursts of colors atypical for this texture.
2. Textures were selected in *the second group* (Fig. 5,6) in order to test the generator's ability to preserve structure. We take textures with repeating patterns with a small number of colors used and without sharp gradient transitions. Such textures were chosen in order to exclude the influence on the synthesis of characteristics not related to the structural organization of the texture.

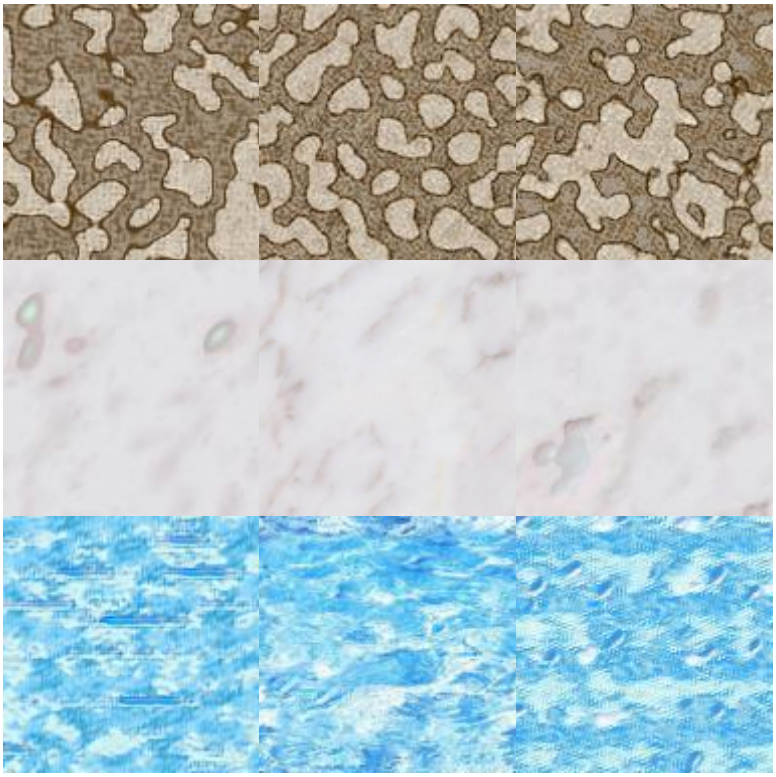

Figure 3: The first group of textures for comparison: from left to right - base method [10] (central plane from generated 3D texture), sample texture, proposed method (central plane from generated 3D texture)

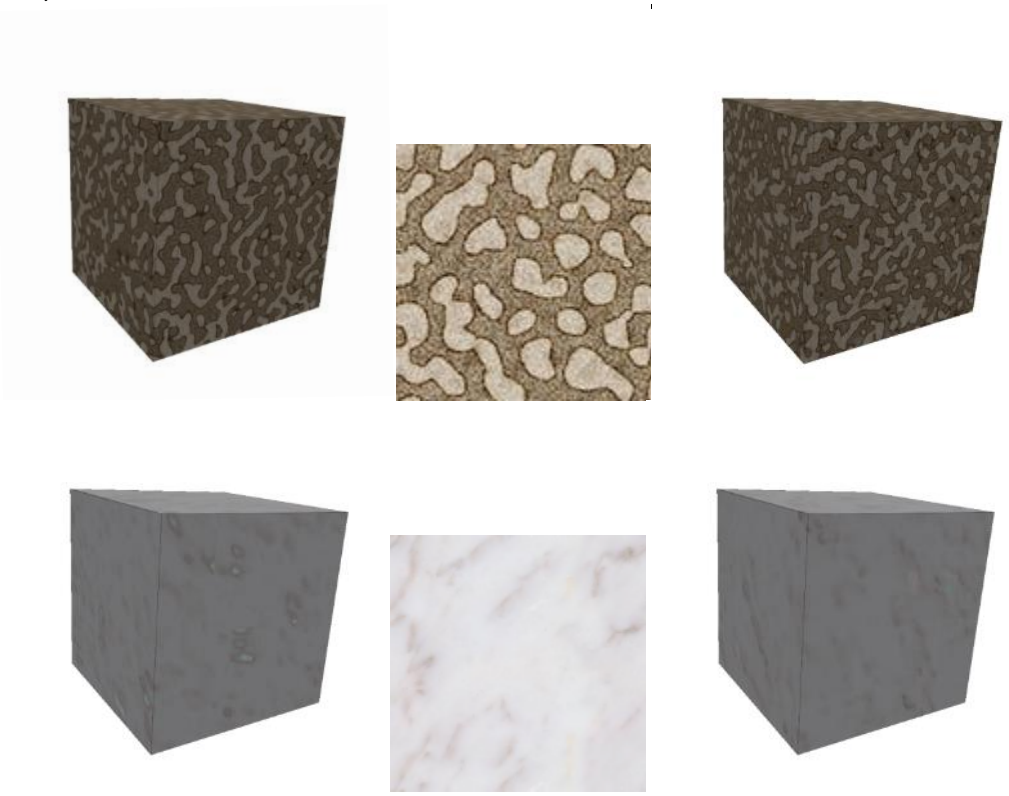

Figure 4: The first group of textures for comparison: from left to right - base method [10] (generated 3D texture), sample texture, proposed method (generated 3D texture)

Figure 5: The second group of textures for comparison: from left to right - base method [10] (central plane from generated 3D texture), sample texture, proposed method (central plane from generated 3D texture)

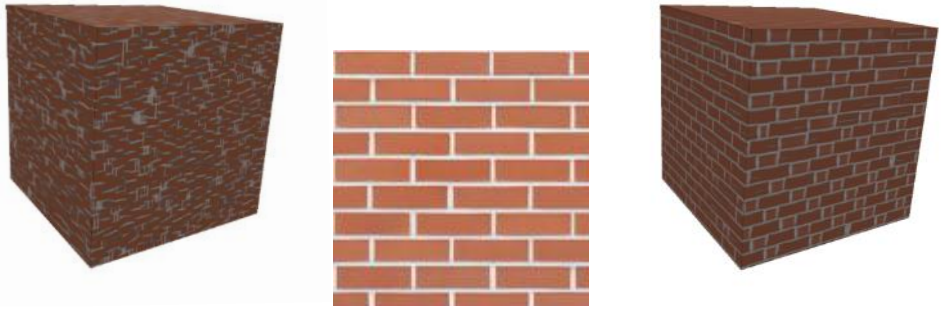

Figure 6: The second group of textures for comparison: from left to right - base method [10] (generated 3D texture), sample texture, proposed method (generated 3D texture)

Finally, we have performed visual comparison of our method to [11] at figures 7 and 8. Unfortunately, we were not able to run their implementation due to version conflicts of libraries, either we didn't find original texture in desired resolution. Therefore, our comparison is not strict but shows that both methods preserve details well.

Figure 7: Visual comparison of [11] (left) and our method (right). Input texture is shown in bottom left corners in white box. In both cases texture structure is preserved well. Input texture size is 256x256.

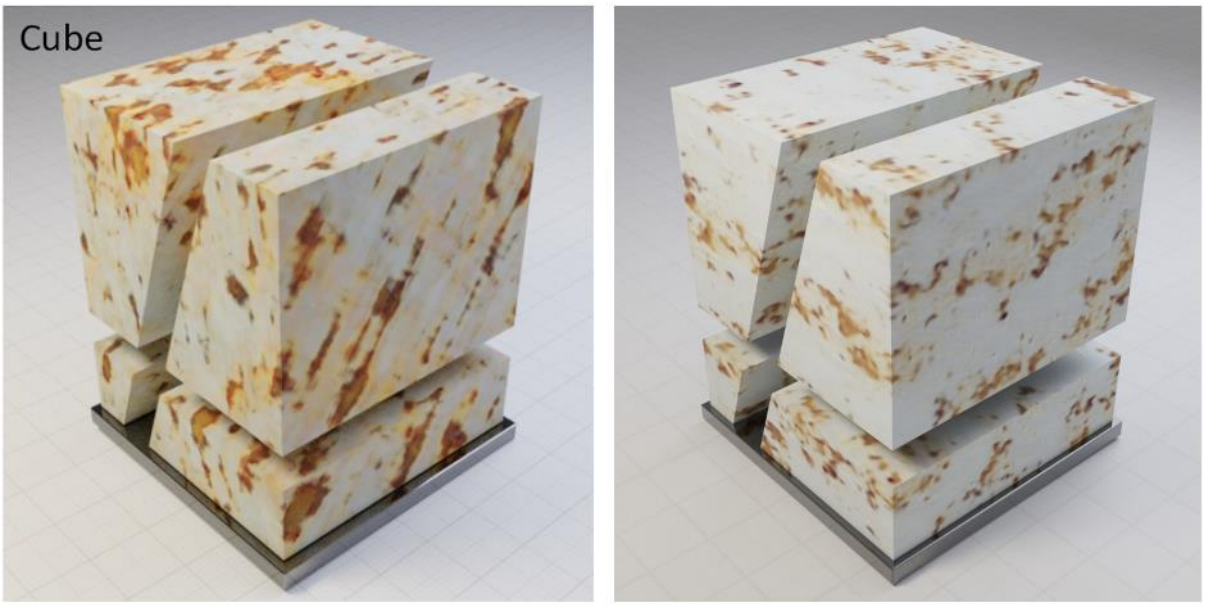

Figure 8: Visual comparison of [11] (left) and our method (right). In both cases texture structure is preserved well. Input texture size is 256x256.

5. Conclusions

It can be seen from fig. 3-6 that the proposed method was able to show better results than it is base version [10]. Like [10], it works well with textures from the first group - the images obtained by the generator are new textures without any color outliers. Additional error components not only did preserve the stability of the synthesis, but also helped to broaden the already wide variety of generated textures, as well as improved the sharpness of pixel transitions (fig. 3, 4). If we look at the results of the second group, it can be seen that the proposed method preserves the structure better in textures containing structural patterns (fig. 5, 6). Thus, introducing additional components to the error helped to overcome the fundamental problem of poor preservation of structuredness inherent to the group of methods to which neural network synthesis belongs, and in the problem of synthesizing three-dimensional textures.

6. References

[1] L. A. Gatys, A. S. Ecker, M. Bethge, Texture synthesis using convolutional neural networks, in: C. Cortes, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15), Vol. 1. MIT Press, Cambridge, MA, USA, 2015, pp. 262–270.

[2] O. Sendik, D. Cohen-Or, Deep Correlations for Texture Synthesis. ACM Trans. Graphics 36(5) (2017) Article 161. doi:10.1145/3015461

[3] C. Li, M. Wand, Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2479–2486.

[4] G. Liu, Y. Gousseau, G.-S. Xia, Texture synthesis through convolutional neural networks and spectrum constraints, in: Proceings of the 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 3234-3239. doi:10.1109/ICPR.2016.7900133.

[5] D. Ulyanov, A. Vedaldi, V. Lempitsky, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6924–6932.

[6] M. Tesfaldet, M. A. Brubaker, K. G. Derpanis, Two-Stream Convolutional Networks for Dynamic Texture Synthesis, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6703–6712.

[7] R. P´eteri, S. Fazekas, M. J. Huiskes, DynTex: A Comprehensive Database of Dynamic Textures, 2010. URL: http://dyntex.univ-lr.fr/database.html
[8] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, H. Huang, Non-Stationary Texture Synthesis by Adversarial Expansion, arXiv preprint (2018) arXiv:1805.04487
[9] A. Frühstück, I. Alhashim, P. Wonka, TileGAN: synthesis of large-scale non-homogeneous textures. ACM Trans. Graph. 38(4) (2019) Article 58. doi: 10.1145/3306346.3322993
[10] J. Gutierrez, J. Rabin, B. Galerne, T. Hurtut, On Demand Solid Texture Synthesis Using Deep 3D Networks, Computer Graphics Forum 36 (2019). doi:10.1111/cgf.13889
[11] P. Henzler, N. J. Mitra, T. Ritschel, Learning a Neural 3D Texture Space From 2D Exemplars, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 8356–8364.
[12] D. J. Rezende, S. Eslami, S. Mohamed, P. Battaglia, M. Jaderberg, N. Heess, Unsupervised Learning of 3D Structure from Images, Advances in neural information processing systems 29 (2016) 4996–5004.
[13] J. Gwak, C. B. Choy, M. Chandraker, A. Garg, S. Savarese, Weakly supervised 3d reconstruction with adversarial constraint, in: International Conference on 3D Vision, 2017, pp. 263–272. doi:10.1109/3DV.2017.00038
[14] X. Yan, J. Yang, E. Yumer, Y. Guo, H. Lee, Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision, arXiv preprint (2016) arXiv:1612.00814.
[15] J. S. De Bonet, Multiresolution sampling procedure for analysis and synthesis of texture images, in: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pp. 361-368. doi:10.1145/258734.258882
[16] A. A. Efros, T. K. Leung, Texture Synthesis by Non-Parametric Sampling, in: Proceedings of the seventh IEEE international conference on computer vision, 1999, pp. 1033–1038.
[17] A. A. Efros, W. T. Freeman, Image quilting for texture synthesis and transfer, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, New York, NY, USA, 2001, pp. 341-346. doi: 10.1145/383259.383296
[18] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, H.-Y. Shum, Real-time texture synthesis by patch-based sampling, ACM Trans. Graph. 20(3) (2001) 127–150. doi:10.1145/501786.501787
[19] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv preprint (2014) arXiv:1409.1556.
[20] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning, 2015, pp. 448–456.
[21] X. Glorot, A. Bordes, Y. Bengio, Deep Sparse Rectifier Neural Networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.