Supplemental Materials - Node Graph Optimization Using Differentiable Proxies

YIWEI HU, Yale University, USA and Adobe Research, USA
PAUL GUERRERO, Adobe Research, UK
MILOŠ HAŠAN, Adobe Research, USA
HOLLY RUSHMEIER, Yale University, USA
VALENTIN DESCHAINTRE, Adobe Research, UK

ACM Reference Format:
Yiwei Hu, Paul Guerrero, Miloš Hašan, Holly Rushmeier, and Valentin Deschaintre. 2022. Supplemental Materials - Node Graph Optimization Using Differentiable Proxies. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3528233.3530733

1 EXAMPLES OF DIFFERENTIABLE PROXIES

We show in Fig. 7 examples of all our trained differentiable proxies. We see that we closely match the original generator in all cases. The generators with high stochasticity benefit from a relaxed training loss, through the addition of an adversarial term, which allows to optimize the loss without pixel perfect proxy generation, because the per-pixel accuracy tends to be less important for stochastic patterns. The following proxies where trained with the additional adversarial term:

- Scratch generator
- Tile Generator (Paraboloid)
- PPTBF [Guehl et al. 2020]

The additional adversarial term is weighed down, typically with a factor 0.1 to augment the main three losses described in the paper (L_1, L_{feat}, L_{style}) without replacing the main goal of the training — reproducing the original generator as closely as possible.

2 CHOICES OF PROXY

To find the ideal differentiable proxy with an optimal latent representation, we evaluate three network architectures: an autoencoder, a GAN and our approach.

2.1 AutoEncoder as Proxy

We tried to define the optimization space using a 2D Convolutional AutoEncoder, designed after the one used in [Gao et al. 2019]. We slightly reduce the latent space Z size to 4x4x256 as we only train for single-channel mask maps (as opposed to material maps in the original paper). We train this AutoEncoder to reconstruct the input image generated by the real generators in the same way as our method described in the main paper. The hypothesis is that by optimizing the latent space Z only, we could force the optimization to remain in the original generator manifold. During optimization to match a target appearance, we use the decoder of our pre-trained AutoEncoder network, and perform optimization in its latent space Z. The trained AutoEncoder reconstructs the input generator map well, but is not constrained enough, as shown in Fig. 1. We see that the optimized material appearance is close to the target, but the generator maps (insets) cannot be represented by the original generator. The optimization leads the latent space outside the manifold of the original generator.

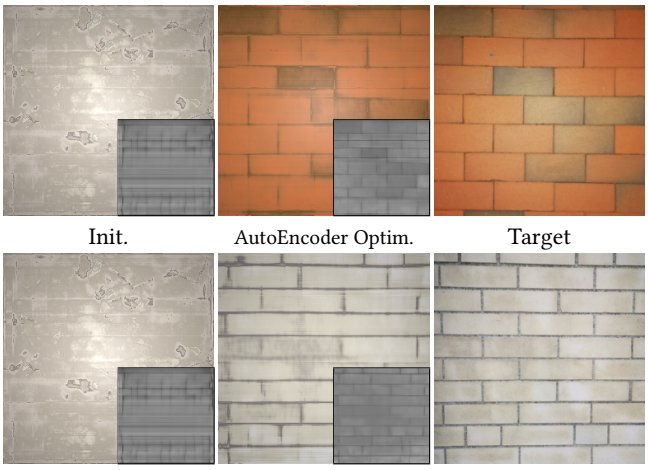

Fig. 1. Results by using an autoencoder as proxy. Inset shows mask maps outputted by the decoder before/after optimization. We initialize the latent space by randomly sampling mask maps, projecting them to the latent space by encoder and averaging the latent codes. The optimized material looks good but the mask are far from a procedural one because the latent space is less constraint.

2.2 StyleGAN as Proxy

To better constrain the optimization space, we experimented with training a generative model (StyleGAN2 [Karras et al. 2020]) as proxy as described in the main paper. In our optimization experiments, we initialize the latent code with the mean of 10,000 randomly sampled Gaussian, as proposed in [Karras et al. 2020]. We optimize the StyleGAN2's $W+$ latent space to match the structure. However, results (Fig. 2, similar to the main paper) show that the latent space of a GAN is still expressive beyond the original generator scope.

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530733

Fig. 2. Result showing the problem with using the original StyleGAN2 architecture as differentiable proxy. Insets represent generator maps synthesized by the trained proxy before/after optimization. We see that it fails to generate a good pattern to match the target.

Additionally, we note that both these solutions (AE & GAN) rely on complex latent spaces Z which would require to train an additional network per proxy to map Z to the original generator parameter space.

2.3 Our Architecture

To constrain the optimization to the original generator space, we perform three changes to the original StyleGAN.

- We replace the randomly sampled latent space Z with normalized parameters P of the original generator f . We know that a function f exists such that $f(P) = M$, with M the target generator map, making P a great input to our network $\bar{f}$, which learns to approximate f. Importantly, using P to guide $\bar{f}$ enables direct optimization of the procedural parameters.
- As we aim at a deterministic one-to-one mapping from parameters to generator maps, we remove the noise inputs at each AdaIn block which are a source of stochasticity.
- We change the loss to enforce a strong local coherency with L_1, L_{feat} and L_{style}, which we can enforce because we aim at approximating f exactly, with as little variation as possible, to make the projection from the proxy parameters to the original generator parameters as transparent as possible.

3 MATERIAL GRAPH OPTIMIZATION

3.1 Optimization Results

We show additional comparison results to Hu et al. [2022] in Fig. 6. All material optimization results are included in our supplemental HTML on three different pages.

The first page, titled "Our Optimization Results with Material Maps and Optimization Sequence" shows our results on real photographs and synthetic targets. We show the target appearance, the initialization of our optimization and the optimization of the Stage II after the random initialization described in Sec. 3.4 of the main paper. We also show our optimized results, a video illustrating the progression of the optimization Stage II and III, and final optimized material maps.

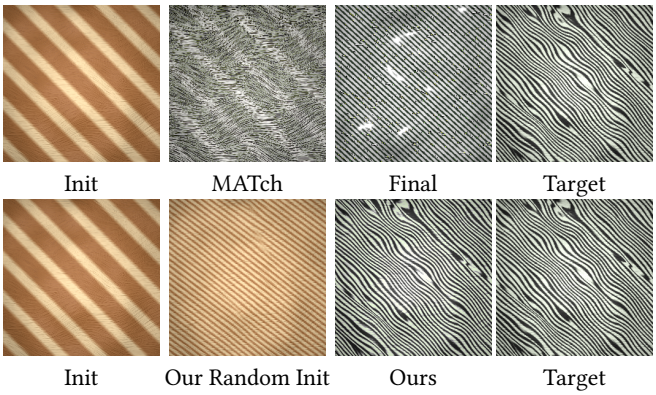

Fig. 3. Sometimes, applying MATch framework first leads the filter nodes' parameters in a local minimum, making our own optimization difficult (first row, Final). In these rare cases, we do not use the MATch output, but directly initialize the optimization with our Stage II initialization step (second row).

The second page, titled "Comparison with MATch" shows all our results with a comparison with the MATch [Shi et al. 2020] framework.

In the third page "Differentiable Proxy in Hu et al.'s Framework" we show that our differentiable proxy can be generalized to benefit the recent inverse material modeling framework by [Hu et al. 2022] and compare with their results. Their optimization of the structure requires 20 minutes, when ours requires 30 seconds.

3.2 MATch as Initialization

As mentioned in the paper, in some cases, the first stage of the optimization (MATch) doesn't help get closer to the final desired appearance, and optimizes the filter parameters to local minima from which it is difficult to get out. In such case we simply discard this step and directly use the random initialization described in Sec. 3.4 of the main paper, allowing to jointly optimize the structure, helping to avoid the local minima. A typical example of this phenomena can be seen in Fig. 3.

3.3 Generator Initialization

We show in Fig. 4 examples of results of our method without our initialization strategy of generators, described in Sec. 3.4 of the main paper. Without a proper initialization, the optimization sometimes stays trapped in a local minimum and struggles to recover.

3.4 Post Optimization

As demonstrated in the main paper, our post-optimization step helps improve the final quality. As small errors can be introduced by replacing our proxy with the real generator, this fine-tuning step helps reach the best possible match. Similar to the main paper, we show such a refinement in Fig. 5 and more results can be seen in our videos "Optimization Proc." (The second half of each video is this post optimization step).

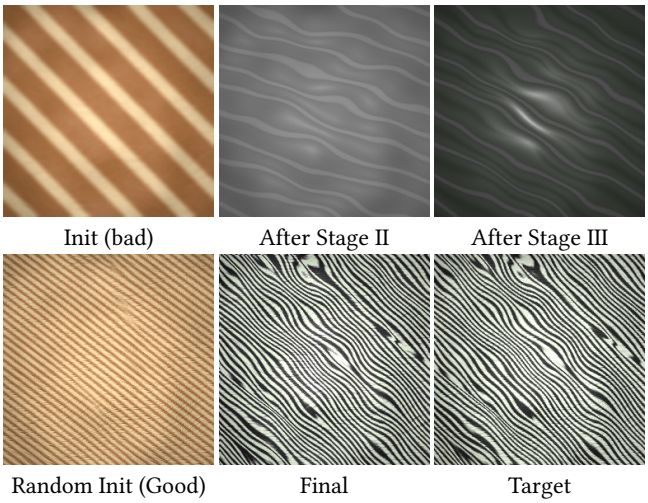

Fig. 4. We show a case where initialization of the generator is not good without random searching at the beginning of Stage II. The optimization sometimes could be trapped to a weird local minima (After Stage II), and the post-optimization step cannot rectify its appearance (After Stage III), making the whole optimization fails. While with random searching, a good initialization is found, which provides reasonable final optimized material.

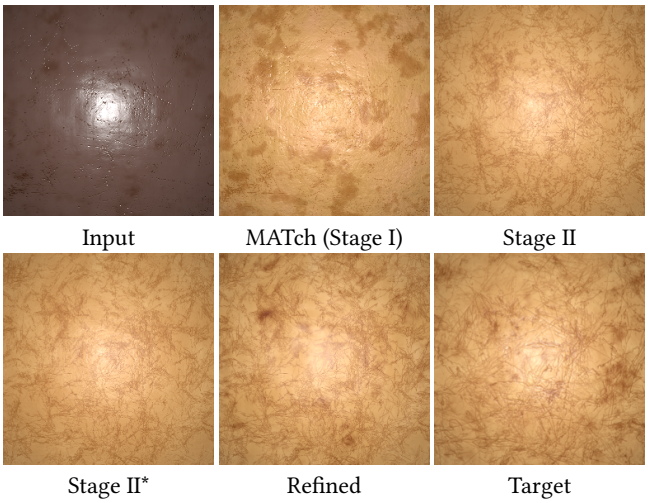

Fig. 5. We optimize a leather material (Input) to match a scratched potato skin (Target). We first match the overall material parameters such as color or roughness (MATch, Stage I). After global optimization (Stage II), we retrieve correct scratch patterns. We then replace our proxy with the real generator (Stage II*) and re-optimize the filter nodes with fixed generators and a smaller learning rate, refining the result (Refined) to best match the target.

4 DIFFERENTIATING FROM RECENT WORK

4.1 Differentiable Rendering

Previous differentiable rendering work [Bangaru et al. 2021; Li et al. 2020] focuses on estimating accurate derivatives of pixel values when image-space/path-space discontinuities are present (e.g., boundaries of a moving object), which is different from our work. We instead deal with the discontinuities in input parameters of the procedures; these can be discrete, or can have non-differentiable

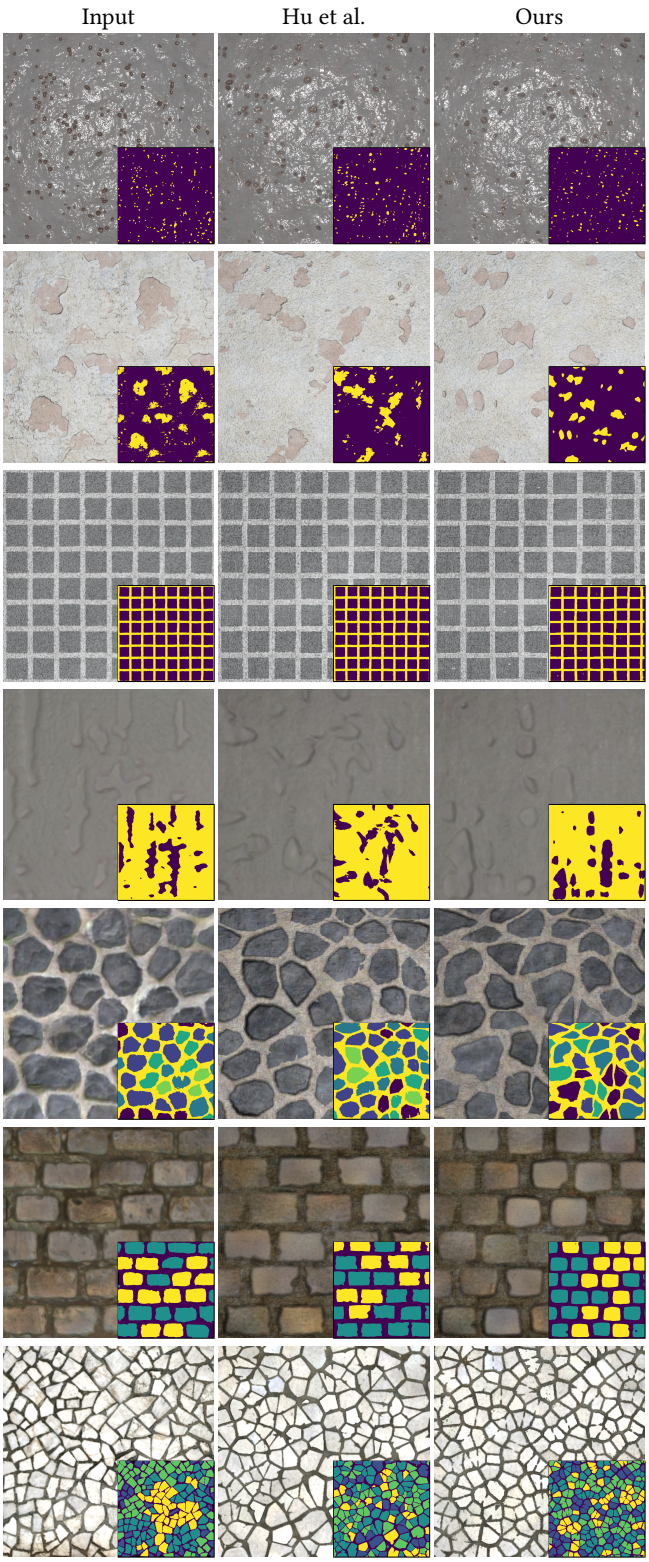

Fig. 6. We plug our differentiable PPTBF proxy in Hu et al.'s [2022] framework. Their framework optimizes parameters of a procedural PPTBF mask to match a user-segmented mask map with a gradient-free method, taking 20 minutes. Using SGD, enabled by our proxy, we achieve similar results in 30 seconds. The first three materials are synthetic while the last four are real data. See ppbtf.html for individual synthesized material maps.

effects (e.g., parameters control the level of randomness of intensity/angles/sizes of patterns).

More generally, generator nodes we consider can be arbitrary pieces of code, often without published source code. Our differentiable proxies allow us to learn a continuous space and differentiate through black box functions for which a clearly defined gradient does not exist, or cannot be determined explicitly.

4.2 StyleGAN Derivatives

Lots of recent work is derived from StyleGAN architecture [Richardson et al. 2021; Tov et al. 2021]. These methods attempt to encode an image directly to the latent code of a pre-trained StyleGAN without optimization. There are significant differences.

First, our proxy is not a generative model: there is only one single correct output for a given input, and the inputs are explicitly interpretable parameters rather than images. We chose StyleGAN because we found its architecture powerful for learning this mapping but modified it to suit our task. Second, our approach is not an encoder. One can train an encoder to directly map a material target image to parameters of nodes (similar to Hu et al. [2019] and MATch's neural initialization [Shi et al. 2020]), but the resulting quality was shown to be significantly lower than optimization approaches like ours.

REFERENCES

Sai Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically Differentiating Parametric Discontinuities. *ACM Trans. Graph.* 40, 107 (2021), 107:1–107:17.

Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. 2019. Deep Inverse Rendering for High-resolution SVBRDF Estimation from an Arbitrary Number of Images. *ACM Trans. Graph.* 38, 4, Article 134 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3323042

Pascal Guehl, Remi Allègre, Jean-Michel Dischler, Bedrich Benes, and Eric Galin. 2020. Semi-Procedural Textures Using Point Process Texture Basis Functions. *Computer Graphics Forum* 39, 4 (2020), 159–171. https://doi.org/10.1111/cgf.14061

Yiwei Hu, Julie Dorsey, and Holly Rushmeier. 2019. A Novel Framework for Inverse Procedural Texture Modeling. *ACM Trans. Graph.* 38, 6, Article 186 (Nov. 2019), 14 pages. https://doi.org/10.1145/3355089.3356516

Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. 2022. An Inverse Procedural Modeling Pipeline for SVBRDF Maps. *ACM Trans. Graph.* 41, 2, Article 18 (jan 2022), 17 pages. https://doi.org/10.1145/3502431

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. 2020. Training Generative Adversarial Networks with Limited Data. In *Proc. NeurIPS.*

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 193:1–193:15.

Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. 2021. Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. 2020. MATch: Differentiable Material Graphs for Procedural Material Capture. *ACM Trans. Graph.* 39, 6 (Dec. 2020), 1–15.

Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. 2021. Designing an Encoder for StyleGAN Image Manipulation. *ACM Trans. Graph.* 40, 4, Article 133 (jul 2021), 14 pages. https://doi.org/10.1145/3450626.3459838

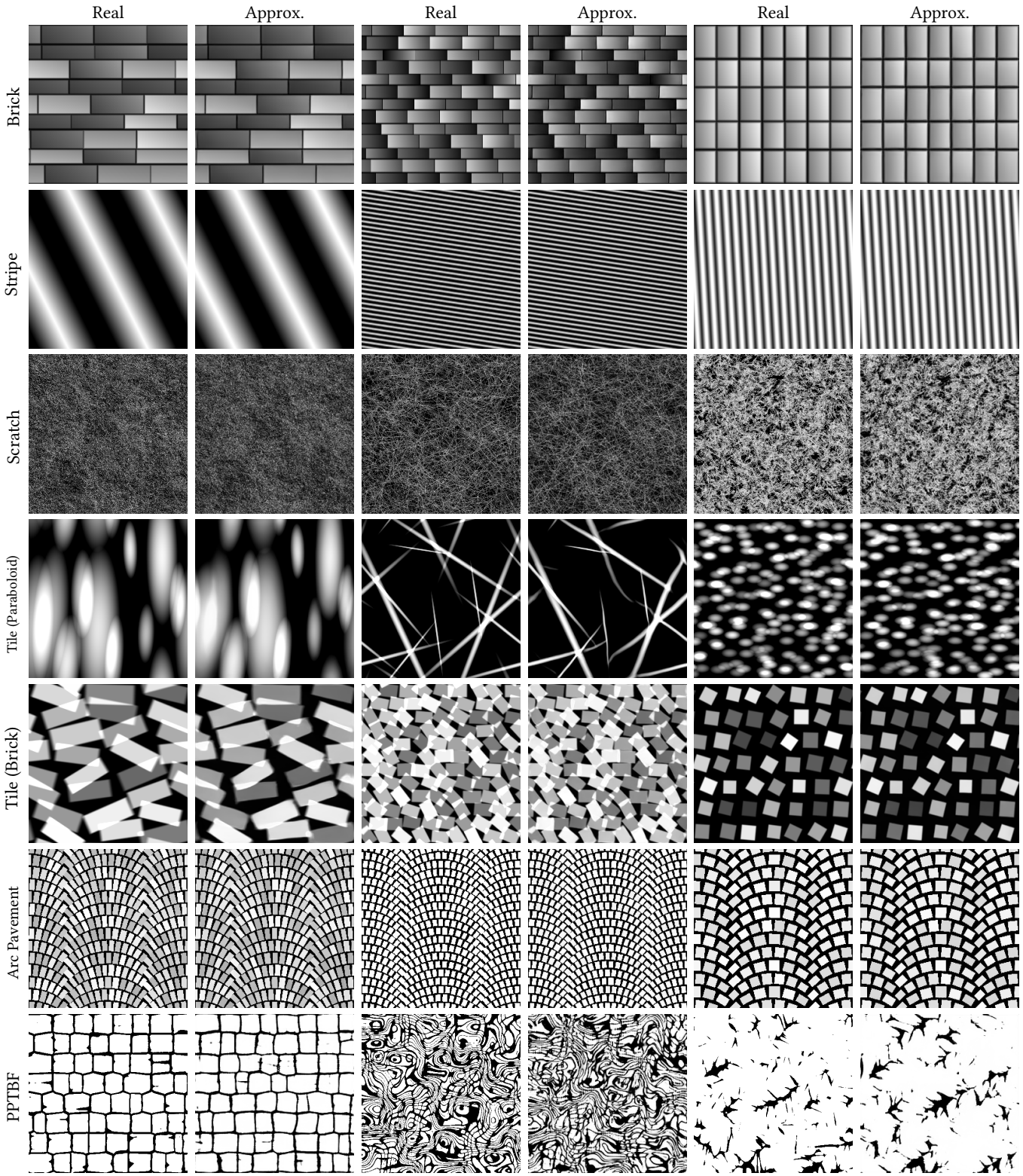

Fig. 7. We compare generator maps synthesized by our proxies (Approx.) with their original procedural counterpart (Real) via randomly sampled parameters, showing that they are very close for all generators