Node Graph Optimization Using Differentiable Proxies

Yiwei Hu
Yale University
New Haven, CT, USA
Adobe Research
San Jose, CA, USA
yiwei.hu@yale.edu

Paul Guerrero
Adobe Research
London, UK
guerrero@adobe.com

Miloš Hašan
Adobe Research
San Jose, CA, USA
mihasan@adobe.com

Holly Rushmeier
Yale University
New Haven, CT, USA
holly.rushmeier@yale.edu

Valentin Deschaintre
Adobe Research
London, UK
deschain@adobe.com

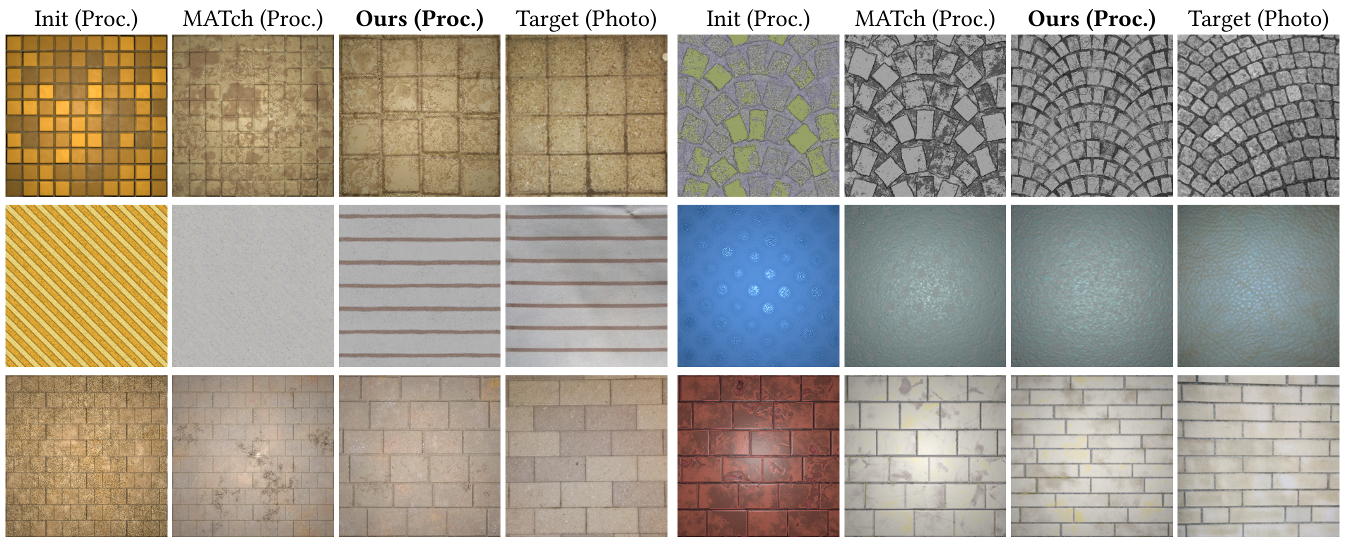

Figure 1: Given a user or classifier provided procedural graph, our method enables and performs end-to-end optimization of graph parameters towards the photograph of a surface. We show here results of our method, against previous work (MATch [Shi et al. 2020]). In particular, compared to previous work, we enable gradient-based optimization of structure and scale of procedural materials. Images which are renderings of procedural models are marked with (Proc.)

ABSTRACT

Graph-based procedural materials are ubiquitous in content production industries. Procedural models allow the creation of photorealistic materials with parametric control for flexible editing of appearance. However, designing a specific material is a time-consuming process in terms of building a model and fine-tuning parameters. Previous work [Hu et al. 2022; Shi et al. 2020] introduced material graph optimization frameworks for matching target material samples. However, these previous methods were limited to optimizing differentiable functions in the graphs. In this paper, we propose a fully differentiable framework which enables end-to-end gradient-based optimization of material graphs, even if some functions of the graph are non-differentiable. We leverage the Differentiable Proxy, a differentiable approximator of a non-differentiable black-box function. We use our framework to match structure and appearance of an output material to a target material, through a multi-stage differentiable optimization. Differentiable Proxies offer a more general optimization solution to material appearance matching than previous work.

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530733

CCS CONCEPTS

• **Computing methodologies → Rendering**.

KEYWORDS

procedural materials, inverse material modeling

ACM Reference Format:
Yiwei Hu, Paul Guerrero, Miloš Hašan, Holly Rushmeier, and Valentin Deschaintre. 2022. 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, 9 pages. https://doi.org/10.1145/3528233.3530733

1 INTRODUCTION

Virtual environment creation relies on a number of artist-created assets. Geometries, lighting, and materials constitute the core of complex environments in movies, video games, or architectural design. Graph-based modeling is commonly used in industry to design sophisticated effects. In this paper, we focus on graph-based procedural representations of materials, e.g., Spatially-varying Bidirectional Reflectance Distribution Functions (SVBRDFs), using parametric operators organized as a computational graph.

SVBRDFs can be represented by pixel maps encoding spatially varying material parameters of an analytical material model (e.g., Cook-Torrance [1982]). Pixel map representations are orders of magnitude more compact than tabulated BRDFs, but are limited to a fixed resolution and require inconvenient per-pixel operations to edit. Procedural graph models, on the other hand, provide parametric flexible control of materials and allow the generation of arbitrary-resolution material maps. The main challenge of procedural materials is the time-consuming modeling process [Adobe 2022]. To achieve a specific material appearance, trained artists need to carefully design the graph and hand-tune a number of parameters. Previous work [Hu et al. 2019, 2022; Shi et al. 2020] focused on reducing the design effort required by using various inverse modeling frameworks, but with limitations e.g., only considering parameters in differentiable nodes in a graph. In this paper, we propose using Differentiable Proxies for non-differentiable nodes to allow end-to-end optimization of entire material graphs.

Nodes in a material graph can typically be classified into two types [Adobe 2022; Hu et al. 2022; Shi et al. 2020]: 1) *Generators*, generating patterns and noises which usually require specifying discrete parameters; 2) *Filters*, which are mostly smooth functions manipulating the generated patterns to reach the envisioned appearance. Fig. 2 shows a few commonly used generators and filters. In previous work, Shi et al. [2020] implemented a library of differentiable filter nodes in the MATch system to enable a gradient-based optimization of existing procedural graphs to match a target appearance. MATch is however limited to differentiable operations, such as *Filter* nodes and cannot optimize non-differentiable operations such as most *Generator* nodes, which rely on discrete parameters with non-differentiable effects e.g., level of randomness of intensity/angles/sizes of patterns. This inherently prevents MATch from adjusting for structural differences between target materials and existing procedural graph output, limiting the generality of this solution, as shown in Figs. 1&8.

To overcome this non-differentiability limitation, we propose leveraging deep learning to enable end-to-end gradient-based optimization of a procedural material. Our method allows joint optimization of all procedural graph parameters to match a material sample in terms of both structure and appearance.

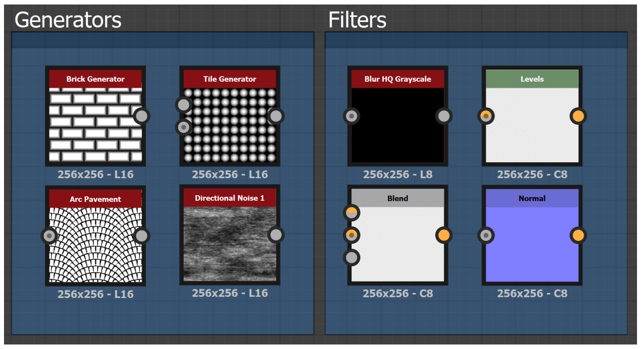

Figure 2: Typical generators and filters used in material graphs such as Substance Graphs where generators model elemental patterns, while filters are mostly smooth functions, modifying the input image appearance.

The core idea is to use a *Differentiable Proxy*, implemented as a differentiable neural network, to approximate a non-differentiable procedure. We adopt the state-of-the-art generative-model-like architecture StyleGAN2 and adapt its inputs and loss to our application. By replacing the original non-differentiable procedures with differentiable proxies, we create a differentiable space to optimize all parameters, including discrete ones. More generally, such proxies allow differentiation of black-box functions for which clearly defined gradients do not exist or cannot be determined explicitly.

We propose a three-stage strategy to smooth the optimization and avoid local minima. The pre-optimization step calibrates the material appearance for better structure initialization. The global optimization starts by efficiently finding a good generator initialization and then jointly optimizes all parameters in the graph using our differentiable Proxy. The post-optimization further refines the material by switching back to the original non-differentiable generator, using the optimal generator parameters found in the last step, and refining only differentiable parameters.

We experiment with different differentiable proxies with both regular and stochastic generation patterns. We show that our neural approximation well reproduces visual patterns created by the original generators. Integrating our neural proxy, we analyze our novel optimization routine and show that we can match the appearance of challenging materials without manual hand-tuning.

In summary, we propose a more general inverse modeling framework for material graphs through the following contributions:

- Neural differentiable proxies for non-differentiable procedures.
- A multi-stage optimization strategy, achieving a high-quality match of a procedural material to a target appearance.
- An end-to-end fully differentiable pipeline that is more general than previous approaches, allowing optimization of all graph parameters without manual tuning.

The code is available at https://github.com/yiwei-hu/DiffProxy.

2 RELATED WORK

2.1 Material Procedural Modeling

Procedural modeling of materials aims at representing analytic materials as procedures [Adobe 2022; Guehl et al. 2020; Guo et al.

2020a; Hu et al. 2019, 2022; Liu et al. 2016; Shi et al. 2020]. These methods generate procedural content from an image, materials or simply imagination. Closest to our method are works by Hu et al. [2019; 2022] and Shi et al. [2020] which estimate parameters of a given procedural graph to match an input material photo.

Hu et al. [2019] train a neural network for each procedural materials, only learning to predict their artist-exposed parameters as opposed to all node graph parameters.

Shi et al. [2020] (MATch) implemented a differentiable version of filter nodes in the Substance Engine [2022] to optimize their continuous parameters to match a target material appearance. Being limited to filter nodes, MATch can only optimize the material appearance (e.g., albedo, roughness) and fails to match material structure.

Hu et al. [2022] present a semi-automatic pipeline for creating a material graph given material maps, requiring artists to segment the material they want to proceduralize. Hu et al.'s non-differentiable structure matching is disconnected from material property optimization, preventing joint optimization. Additionally, their structure matching step requires a time-consuming (~20 min) gradient-free optimization, dominating their runtime.

Our fully differentiable framework addresses the non-differentiability of generator nodes and can be directly plugged into MATch [Shi et al. 2020] and Hu et al. [2022] method, enabling end-to-end global optimization with better and faster appearance matching.

2.2 Material Acquisition

Material acquisition targets the recovery of material properties based on one or more images. Traditionally, dozens to thousands of images were required to sample the light-view space as described in the extensive survey by Guarnera et al. [2016]. More recently, deep neural networks were used to improve reconstruction from a single image [Deschaintre et al. 2018; Guo et al. 2021; Henzler et al. 2021; Li et al. 2017; Zhou and Kalantari 2021] and from a small number of images [Deschaintre et al. 2019, 2020; Gao et al. 2019; Guo et al. 2020b; Ye et al. 2021]. These methods can be separated into two categories. The first category relies on a single forward inference to recover the material parameters using an encoder/decoder architecture [Deschaintre et al. 2018, 2019; Guo et al. 2021; Li et al. 2017; Ye et al. 2021; Zhou and Kalantari 2021], while the second optimizes the latent space of a pre-trained decoder network [Gao et al. 2019; Guo et al. 2020b; Henzler et al. 2021]. While these methods can recover material parameters, they primarily focus on the reconstruction of material parameter pixel maps with their limited resolution and editability. In contrast to pixel maps, we focus on procedural representations of materials which allow users to parametrically control, edit and synthesize materials at any resolution and scale.

2.3 Program Generation

Since a material graph can be interpreted as a program, approaches that generate or infer programs from a given input are relevant to our work. Most of the research in program generation and inference has focused on the 2D or 3D shape domain, with a few notable exceptions [Ganin et al. 2018; Hu et al. 2018] which generate sequences of image edits using reinforcement learning. Early works focus on inverse proceduralization of given 3D shapes [Demir et al. 2016; Štava et al. 2010], while more recent work uses data-driven methods to learn a prior over shape programs that can either be used for program generation or program induction from a given shape [Du et al. 2018; Ellis et al. 2019, 2018; Johnson et al. 2017; Jones et al. 2020; Kania et al. 2020; Lu et al. 2019; Sharma et al. 2018; Tian et al. 2019; Walke et al. 2020; Wu et al. 2019; Xu et al. 2021].

These methods operate in a different domain than our approach and also differ in their problem setup. Our goal is to modify an existing program, i.e., a computational graph, to more closely match a target material image rather than generating a program from scratch. We focus on inferring parameter configurations of a graph.

3 METHOD

Procedural material graphs are acyclic computational graphs generating spatially-varying (SV) material maps. The starting point of a material graph is a set of *generators* producing patterns at multiple scales which serve as building blocks of a material graph. These generators typically define structures and multi-level texture features of the material, like a brick or tile pattern. Starting from the generator outputs, a variety of different *filters* modify their appearance and combine them in multiple steps to achieve a final realistic procedural material. Although the generators play an important role in material graph design, previous graph optimization frameworks assume that the generators are pre-calibrated and fixed during graph optimization [Hu et al. 2022; Shi et al. 2020], thus they cannot optimize some features of the material. In contrast, we propose a fully end-to-end optimization framework that allows global and joint optimization of both material structures/features and material appearance.

The main challenge for optimizing generator nodes is their non-differentiablity. Our key idea is to avoid direct optimization of the original generator by optimizing a differentiable proxy.

3.1 Differentiable Proxy

Given an arbitrary, non-differentiable, 2D image generator $G(\theta)$, where θ represents its procedural parameters, we create a differentiable proxy. To do so, we approximate $G(\theta)$ with a convolutional neural network (CNN) $\hat{G}(\theta)$ trained to reproduce the behaviour of $G(\theta)$: given a set of parameters θ_i we want $G(\theta_i) = \hat{G}(\theta_i)$.

A natural solution would be to train a generative model (e.g., StyleGAN2). However, we notice that generative models' optimizable latent spaces (e.g., $W+$ in StyleGAN2) can be expressive beyond the original generator scope. This allows the proxy to generate maps that do not exist in the original generator space, leading to (1) poor appearance matching and (2) complicating the process of mapping back to the original generator parameters. The problem (1) is illustrated in Fig. 3. We trained the original StyleGAN2 with the data sampled using the process described in Sec. 4. The insets of the second column of Fig. 3 represent the direct outputs of the trained StyleGAN2 which fail to match the targets. Solving problem (2) in this context would require training an additional network mapping from $W+$ to the original generator parameter space. A more detailed discussion of the limits of using StyleGAN2 or AutoEncoder [Gao et al. 2019] is available in the supplemental materials.

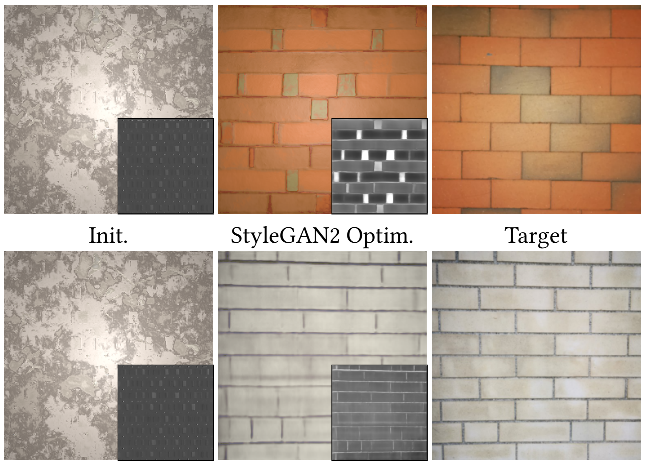

Figure 3: Result using the original StyleGAN2 architecture as differentiable proxy. Insets represent generator maps synthesized by the trained proxy before/after optimization. We follow the procedure described in Sec. 3.3 to optimize the structure using StyleGAN2's $W+$ latent space. We see that it fails to generate a good pattern to match the target. Our proxy solves these issues as can be seen in Fig. 1 and supplemental materials.

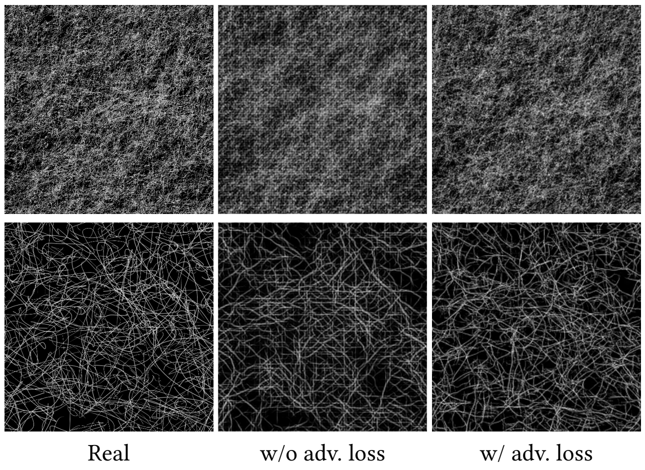

Figure 4: Adding an adversarial loss helps reduce artifacts when patterns are stochastic. We show here a typical case with the scratches generator which generates highly stochastic patterns.

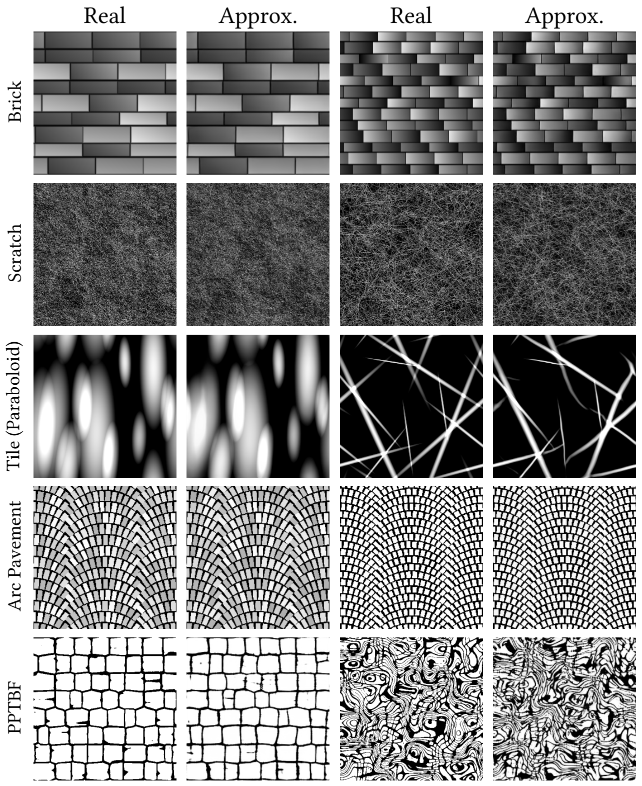

Figure 5: We compare generator maps synthesized by our proxies (Approx.) with their original procedural counterpart (Real) via randomly sampled parameters, showing that they are very close. See supplemental documents for results generated from all our trained proxies.

To overcome this problem, we use a modified StyleGAN2 [Karras et al. 2020] architecture. We modify the architecture input and train our network to approximate the mapping between the parameters and the output of a generator node. Specifically, rather than starting from a random latent vector Z, we encode parameters θ into an intermediate latent space W, through a set of fully connected layers. We then feed W to AdaIn layers, similar to the original StyleGAN.

We further modify the architecture to use θ as its sole input: we remove the noise added in each block of the original StyleGAN2, making our network deterministic, allowing it to learn the one-to-one mapping as we need.

We train our neural proxy using synthetic data, directly generated by carefully sampling (Sec. 4) the procedural generator. With our modified architecture, we can train our differentiable proxy $\hat{G}$ by directly sampling a data pair (θ, I) and minimizing the per-pixel difference between the approximated generator map $\hat{G}(\theta)$ and the ground-truth I. We design a weighted loss function combining L_1 loss, deep feature loss, style loss [Gatys et al. 2015], and an optional adversarial loss L_{Adv}:

$$L = \lambda_0 L_1 + \lambda_1 L_{\text{feat}} + \lambda_2 L_{\text{style}} (+\lambda_3 L_{\text{Adv}}) \tag{1}$$

Deep feature loss L_{feat} is defined by the L_1 difference between deep feature maps extracted from a pre-trained VGG19 ([Simonyan and Zisserman 2015] neural network. Style loss L_{style} is defined by the L_1 difference between the Gram Matrices of extracted deep feature maps. We empirically assign $\lambda_0 = \lambda_2 = 1$ and $\lambda_1 = 10$.

In our experiments, the first three loss terms are generally enough to guide the neural network to learn a one-to-one mapping for procedural generators ($\lambda_3 = 0$). However, for highly random patterns, such as scratch generators, this combination of losses results in small artifacts and struggles to reproduce stochastic behavior. To solve this, we add an adversarial critic to improve the fitting quality,

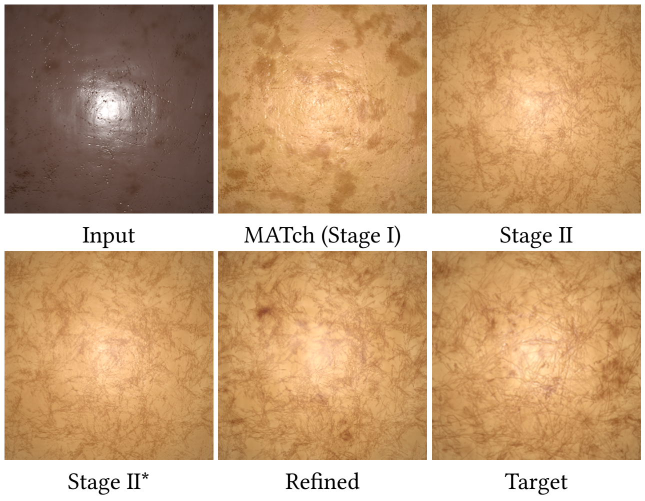

Figure 6: We optimize a leather material (Input) to match a scratched potato skin (Target). We first match the overall material parameters such as color or roughness (MATch, Stage I). After global optimization (Stage II), we retrieve correct scratch patterns. We then replace our proxy with the real generator (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.

as shown in Fig. 4. For these stochastic generators, we adjust $\lambda_3 > 0$ to ensure that the loss is still dominated by the first three loss terms, relaxing the one-on-one mapping goal, without replacing it. L_{Adv} is defined as the cross-entropy loss with $R1$ regularization [Mescheder et al. 2018]. The critic takes both parameter θ and generator map I as input and evaluates whether it is a real or fake data pair.

3.2 Our Differentiable Proxies

In our experiments, we train differentiable proxies for generators in both MATch and the more recent system by Hu et al. [2022]. In the case of generators for MATch we train proxies for the following generators, which represent the majority of generators used across 100 analyzed Substances: *Brick Generator, Stripe Generator, Scratches Generator, Tile Generator (Paraboloid), Tile Generator (Brick), Arc Pavement Generator*. In the case of Hu et al. [2022] we train a proxy for the generator they used: *Point Process Texture Basis Functions (PPTBF)* [Guehl et al. 2020]).

We compare in Fig. 5 generator maps synthesized with our trained differentiable proxy $\hat{G}$ to those generated with the original procedural generator G. Each result shows a randomly sampled set of parameters, showing that our proxies can generate mask maps which closely approximate those generated by G. This ensures that we remain in the space of masks that can be generated by G during proxy optimization, allowing us to project back the parameters to the original non-differentiable procedural material graph.

3.3 Fully Differentiable Optimization

Given a user (or classifier [Shi et al. 2020]) chosen material graph $\mathcal{G}$, we convert it to its differentiable counterpart by replacing the procedural generator nodes with our proxies and the filters with differentiable filter nodes from the DiffMat library [Shi et al. 2020]. The optimizable parameters are $\theta = (\theta_g, \theta_f)$ where θ_g and θ_f drive our differentiable proxies and the differentiable filters respectively.

The material graph $\mathcal{G}$ outputs a set of spatially-varying 2D material maps $M = \mathcal{G}(\theta)$ (e.g., albedo, normal, roughness, and metallic). A rendering operator R can be applied to synthesize an image $I = R(M)$. Our optimization process recovers an optimal θ^* that minimizes the difference $d(I, I^*)$ between our rendered image $I = R(M)$ and a user provided target image I^*:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L_\theta = \underset{\theta}{\operatorname{argmin}}\, d(I, I^*) \tag{2}$$

Leveraging $\mathcal{G}$ differentiability, we optimize θ^* with gradient descent using PyTorch [Paszke et al. 2019] as a general auto-differentiation framework.

A key challenge is the existence of local minima in the joint optimization of both generators and filters. While we relax the generator differentiability, the original discrete attribute variation tends to form local minima, making the optimization non-convex. To stabilize it, we propose a multi-stage optimization strategy.

3.3.1 Stage I: Pre-optimization. We start by only matching the overall material appearance, i.e., only optimizing θ_f with a fixed θ_g. This is a pre-optimization step. We use the MATch framework. The loss function we defined is a multi-scale style loss:

$$L_{\theta_f} = ||GM(I) - GM(I^*)||_1 \tag{3}$$

where GM is an operator that computes Gram Matrices of extracted deep features [Gatys et al. 2015]. We compute and aggregate the loss function at multiple resolutions (256x256, 128x128 and 64x64). This pre-optimization step calibrates basic material properties (e.g., albedos and roughness). In few cases MATch fails to improve the initialization, in which case, we directly move to Stage II.

3.3.2 Stage II: Global Optimization. We use the optimization results from Stage I to initialize this step. We now optimize the entire set of parameters $\theta = (\theta_f, \theta_g)$. To minimize the impact of local minima, we find a good initialization. We randomly sample possible generator parameters and initialize our optimization with the parameters which generate the closest appearance to the targeted image:

$$L_{\theta_g} = ||F(I_g) - F(I_g^*)||_1 \tag{4}$$

where F denotes the extracted deep features from a pre-trained neural network [Simonyan and Zisserman 2015]. I_g and I_g^* are grayscale version of I and I^* respectively. We can efficiently compare the grayscale images as color and roughness values were already optimized. In practice, we sample 500 possible parameters initialization for a differentiable graph $\mathcal{G}$ in less than a minute.

Using the selected initialization of θ_g, we optimize all parameters θ with a combination of feature and style loss:

$$L_\theta = ||F(I) - F(I^*)||_1 + \alpha||GM(I) - GM(I^*)||_1 \tag{5}$$

where α is a weighting variable. We use feature loss as a main loss term to measure the structure and appearance similarity. We also add a small style loss component as a flexible component that matches the overall statistics of material appearance as the procedural material may not always achieve pixel-perfect matches of real material pictures. We empirically choose $\alpha = 0.05$ and, similar to Eq. 3, we evaluate the loss function using a multi-resolution

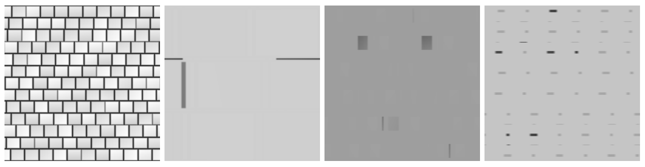

Figure 7: We sample the data to train our proxies by sampling in the parameter spaces used by artists. On the left is an example of pattern generated from this space, while on the right are three examples of (undesirable) patterns that can be generated with the same generator, with uniform sampling of the entire parameter space.

approach. To further reduce the risk of encountering local minima, we set a larger learning rate, decaying over the optimization.

3.3.3 Stage III: Post-Optimization. As our differentiable proxies may generate slightly different images than the original generators, we fine-tune the results of our Global Optimization step. In this step, we switch our optimized proxies back to their original procedural counterparts with the optimized parameters θ_g. As the generators are now non-differentiable, we fine-tune the filter nodes parameters θ_f. This fine-tuning step is the same as Stage I, using a multi-scale style loss as described in Eq. 3, with a closer material structure and a finer learning rate, refining the appearance to better match the target. We show the result of each of these step in Fig. 6, and videos of the optimization process in the supplemental material.

4 IMPLEMENTATION

To train our models, we build a dataset for a chosen generator by randomly sampling its parameter space. However, a fully random-sampling strategy leads to unrealistic or invalid patterns. For instance, some parameter combinations for a brick pattern generator may not lead to a brick-like pattern, as shown in Fig. 7.

To build a more representative dataset, we use a heuristic sampling approach. For popular systems, we leverage existing collections of node graphs. For example when training proxies compatible with MATch, (based on Substance), we analyze the parameter distribution from Substance Source, a database containing 7000+ artist-designed material graphs [Adobe 2022]. We compute the range of parameters used by artists, as well as their mean and standard derivations, and independently sample each parameter. We generate a 300,000 256x256 images dataset for each differentiable proxy $\hat{G}$, which, depending on the procedural generator complexity takes 5 ∼ 10 hours. Once trained, each $\hat{G}$ requires less than 100 MB.

To train the PPTBF proxy for the recent Hu et al. [2022] framework, we use the PPTBF released dataset [Guehl et al. 2020] and resample the parameters based on the parameter distributions analyzed from that dataset. We sample an additional 500,000 mask maps to increase the sample density.

We implement our differentiable proxies and optimization pipeline in PyTorch. We train our proxy using a Nvidia RTX 3090 with a CPU of Intel i9 10900K. To reach convergence, 60 epochs of training usually takes 2 ∼ 3 days on a single GPU, using Adam [Kingma and Ba 2015] with a learning rate of 0.0025, a batch size of 32 and normalizing the parameters between 0 and 1 to stabilize the training. When applying adversarial training, we set $\gamma = 10$ for $R1$ and train the discriminator with Adam and a learning rate of 10^{-3}.

For the parameter optimization itself, we use the PyTorch auto-differentiation framework and use Adam($\beta = (0.9, 0.999)$ with a learning rate decaying strategy (lr is halved every 200 steps) to optimize for our target appearance.

We set a smaller learning rate for pre- and post- optimization steps (0.002) because the optimization space of continuous parameters of filter nodes are more convex. We set a larger initial learning rate for global optimization (0.02) as we observe that a larger early step size helps avoid local minima. The full optimization takes around 3 ∼ 5 minutes for 1000 steps, depending on the complexity (numbers of nodes and connections) of the material graphs. For reference, MATch optimization takes 2 ∼ 3 min.

5 RESULTS AND COMPARISONS

In this section, we show our end-to-end material optimization results and compare to previous work. We use our differentiable proxies to enable material graph optimization in two frameworks [Hu et al. 2022; Shi et al. 2020]. We show our material graph optimization results in the MATch framework, using the MATch graph selection step and compare to it in Figs. 1 & 8. Our approach better matches the target appearance thanks to our optimization of the material structure and scale. We show results with synthetic data in Fig. 8 and with real-world photographs in Fig. 1, showing that our approach can match a variety of generators and appearances. Please see our supplemental materials for additional results. In a quantitative evaluation, the average of feature/style loss of all materials we optimized is 0.408/0.261 for ours against 0.548/1.038 for MATch.

In Fig. 9, we demonstrate the generality of our differentiable proxy in another inverse modeling framework by Hu et al. [2022] where we train a PPTBF [Guehl et al. 2020] proxy which can simply be plugged into the proposed pipeline to replace their time-consuming structure matching process. Following their approach, we optimize our proxy towards a user-segmented mask map. Our proxy enables gradient-based optimization, reaching similar material appearance to their method, with a 40x speedup (their optimization requires 20 minutes, while ours converges in 30 seconds). Please see our supplemental materials for additional examples.

We highlight that despite the optimization running on fixed resolution, we recover procedural model properties, making our results entirely procedural. Our results can therefore be generated with arbitrary resolution, without a costly high-resolution optimization, and preserve the editing possibilities inherent to procedural models, allowing artists to use our results as a base to kickstart their final vision. We also preserve the original generator properties, such as tileability for the Substance generators, making our results tileable. We demonstrate editability, high resolution material synthesis, and tileability in Fig. 10.

6 LIMITATIONS

We show limitations of our method in Fig. 11. In a Substance we experimented with, the graph structure was such that the optimization was more prone to local minima. This could however be solved by simple manual edits of the graph to simplify the gradient flow.

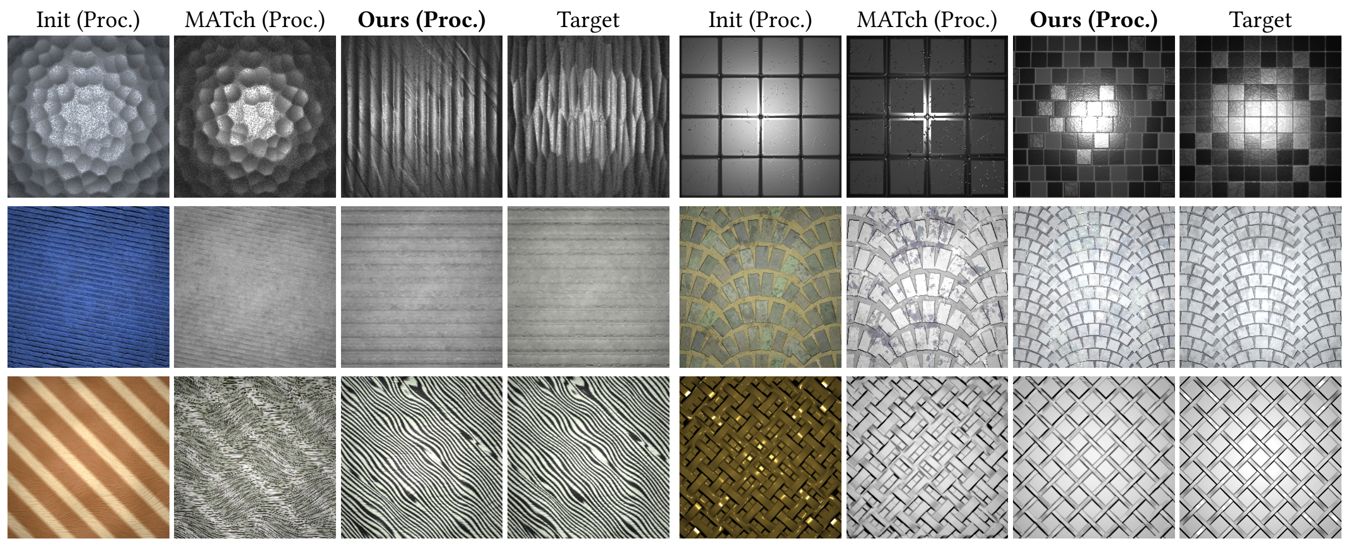

Figure 8: Results on synthetic materials. We sample random parameters in a graph as the target appearance and show that our optimization is able to recover these parameters even with a distant initialization. See supplemental materials for additional results.

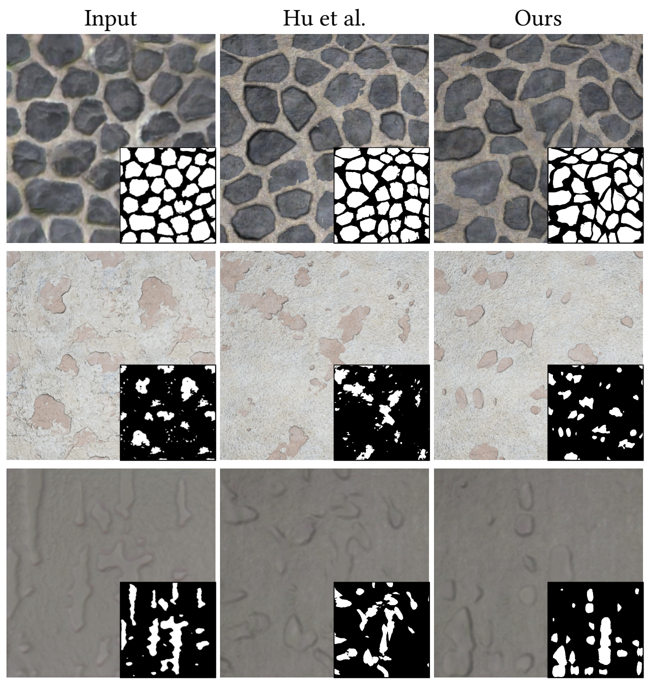

Figure 9: We plug our differentiable PPTBF proxy in Hu et al.'s [2022] framework. Their framework optimizes parameters of a procedural PPTBF mask to match a user-segmented mask map with a gradient-free method, taking 20 minutes. Using SGD, enabled by our proxy, we achieve similar results in 30 seconds. See supplemental materials for additional comparisons.

Another limitation comes from our loss which does not match elements per pixel, missing local details (see wood node in the second row of Fig. 11), or not precisely match the scale (see the first result in Fig. 10 which has a 7x7 tiles target, but a 6x7 tiles result).

Additionally, due to the importance-sampled parameter space, patterns out of the parameter range used by the artist covered in the training of a proxy are not well reproduced. For instance, when the number of bricks is outside of the distribution, the differentiable proxy cannot reproduce the desired patterns (third row).

Similar to recent inverse procedural material modeling methods [Hu et al. 2019; Shi et al. 2020], our approach fails if the selected initial graph is not expressive enough to match the target appearance, which is illustrated in the last row of Fig. 11.

Finally, although most filters in Substance are differentiable, a few complex filter nodes like FX-map can also have discrete and random behaviors that remain difficult to differentiate. Creating an image-conditioned differentiable proxy for these filter nodes is an interesting future challenge.

7 CONCLUSION

We present a general differentiable solution that optimizes both generator and filter nodes in a material graph. We introduce the Differentiable Proxy, a neural-network-based universal approximator to establish a differentiable parameter space for non-differentiable generators. We demonstrate, with a multi-stage optimization pipeline, that the proxies enable end-to-end optimization of both structures and appearance to match material photographs or SVBRDF maps. We apply gradient-based optimization, supported by auto-differentiation, on a wide range of material graphs, showing that our framework can achieve high-quality procedural materials from various exemplars. We believe our proxies will enable better material proceduralization and improve differentiability of complex black box functions.

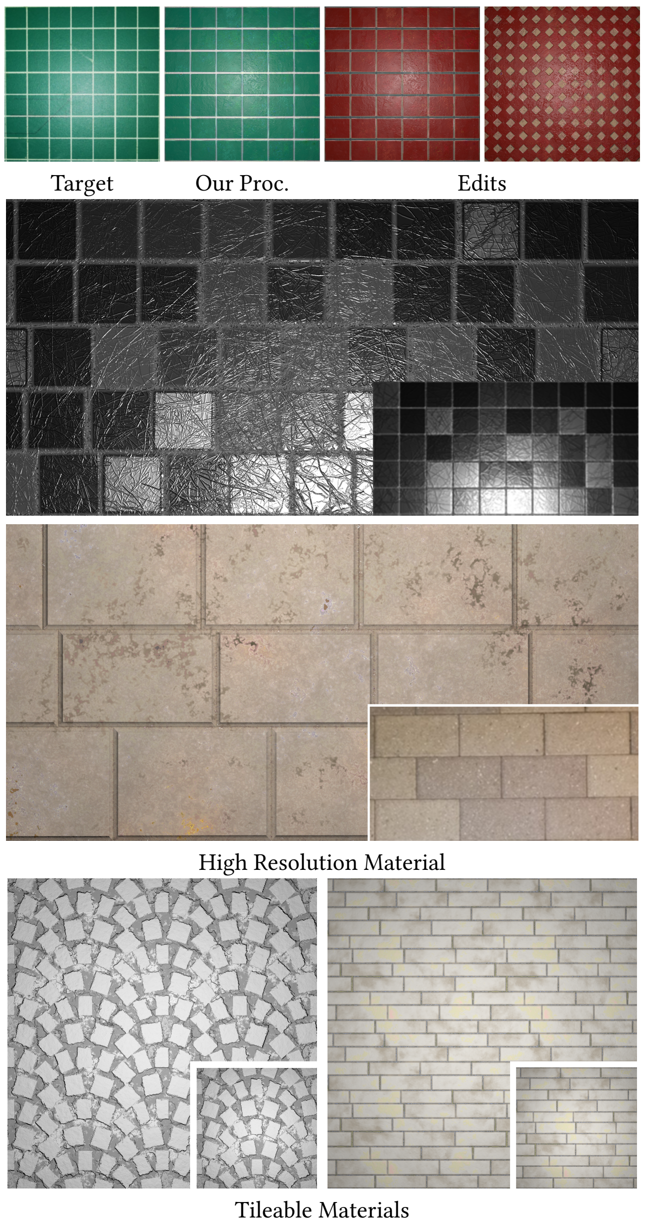

Figure 10: Our results are procedural and can easily be edited (First row). After optimization, our results can be synthesized in arbitrary resolution (cropped 2K, cropped optimization target image as inset, 2nd row). Finally, we preserve the generators' tileability, showing here 2x2 tiled results.

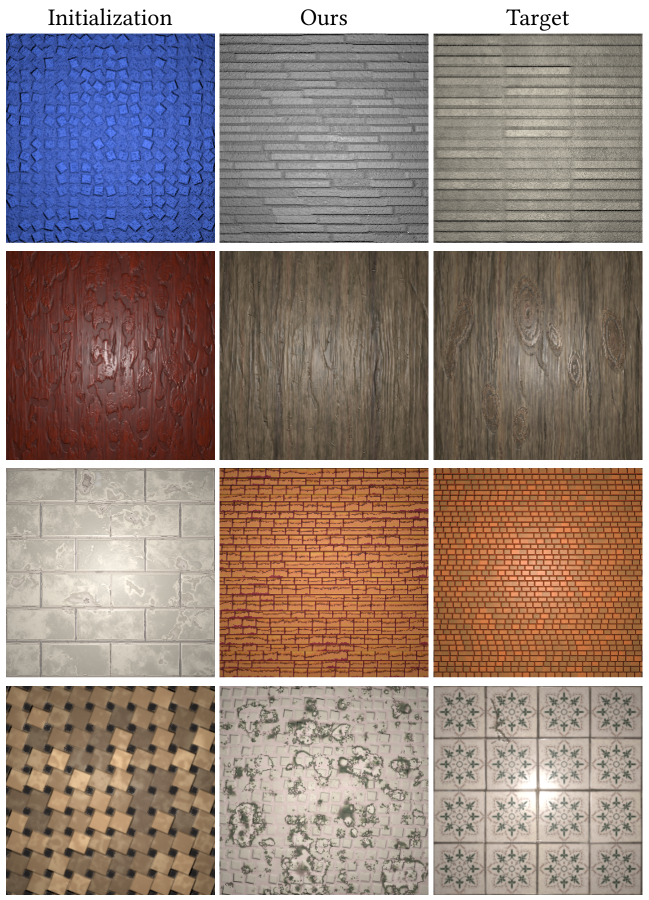

Figure 11: Limitations. First row: On complex material graphs, in a few cases, our method can lead to a local minimum, despite our initialization approach. Second row: our loss function targets global appearance matching, here it fails to reproduce the knots details in the wood plank. Third row: our proxy fails to reproduce patterns when parameters fall outside of the range used by artists. Here, the number of bricks are beyond our defined sample ranges. Last row: a common limitation in procedural modeling, optimizing a less expressive material graph which does not model specific targeted patterns cannot represent well the target.

ACKNOWLEDGMENTS

This work was supported in part by NSF Grant No. IIS-2007283.

REFERENCES

Adobe. 2022. Substance Designer. https://substance3d.adobe.com.

R. L. Cook and K. E. Torrance. 1982. A Reflectance Model for Computer Graphics. *ACM Trans. Graph.* 1, 1 (Jan. 1982), 7–24. https://doi.org/10.1145/357290.357293

Ilke Demir, Daniel G Aliaga, and Bedrich Benes. 2016. Proceduralization for editing 3d architectural models. In *2016 Fourth International Conference on 3D Vision (3DV).* IEEE, 194–202.

Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. 2018. Single-Image SVBRDF Capture with a Rendering-Aware Deep Network. *ACM Trans. Graph.* 37, 4, Article 128 (Aug 2018), 15 pages.

Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. 2019. Flexible SVBRDF Capture with a Multi-Image Deep Network. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 38, 4 (July 2019), 1–13.

Valentin Deschaintre, George Drettakis, and Adrien Bousseau. 2020. Guided Fine-Tuning for Large-Scale Material Transfer. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 39, 4 (2020), 91–105. http://www-sop.inria.fr/reves/Basilic/2020/DDB20

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–16.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. 2019. Write, Execute, Assess: Program Synthesis with a REPL. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/50d2d2262762648589b1943078712aa6-Paper.pdf

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to Infer Graphics Programs from Hand-Drawn Images. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/6788076842014c83cedadbe6b0ba0314-Paper.pdf

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. 2018. Synthesizing programs for images using reinforced adversarial learning. In *International Conference on Machine Learning*. PMLR, 1666–1675.

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

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. arXiv:1508.06576 [cs.CV]

Dar'ya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. 2016. BRDF Representation and Acquisition. *Computer Graphics Forum* 35, 2 (2016), 625–650.

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

Jie Guo, Shuichang Lai, Chengzhi Tao, Yuelong Cai, Lei Wang, Yanwen Guo, and Ling-Qi Yan. 2021. Highlight-Aware Two-Stream Network for Single-Image SVBRDF Acquisition. *ACM Trans. Graph.* 40, 4, Article 123 (July 2021), 14 pages. https://doi.org/10.1145/3450626.3459854

Yu Guo, Miloš Hašan, Lingqi Yan, and Shuang Zhao. 2020a. A Bayesian Inference Framework for Procedural Material Parameter Estimation. *Computer Graphics Forum* 39, 7 (2020), 255 – 266.

Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. 2020b. MaterialGAN: Reflectance Capture Using a Generative SVBRDF Model. *ACM Trans. Graph.* 39, 6, Article 254 (Nov. 2020), 13 pages. https://doi.org/10.1145/3414685.3417779

Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. 2021. Generative Modelling of BRDF Textures from Flash Images. *ACM Trans Graph (Proc. SIGGRAPH Asia)* 40, 6 (2021).

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

Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. 2018. Exposure: A White-Box Photo Post-Processing Framework. *ACM Transactions on Graphics (TOG)* 37, 2 (2018), 26.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*. 2989–2998.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis. *ACM Transactions on Graphics (TOG), Siggraph Asia 2020* 39, 6 (2020), Article 234.

Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. 2020. UCSG-Net – Unsupervised Discovering of Constructive Solid Geometry Tree. In *arXiv*.

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. 2017. Modeling Surface Appearance from a Single Photograph using Self-augmented Convolutional Neural Networks. *ACM Trans. Graph.* 36, 4, Article 45 (2017), 11 pages.

Albert Julius Liu, Zhao Dong, Miloš Hašan, and Steve Marschner. 2016. Simulating the Structure and Texture of Solid Wood. *ACM Trans. Graph.* 35, 6, Article 170 (Nov. 2016), 11 pages. https://doi.org/10.1145/2980179.2980255

Sidi Lu, Jiayuan Mao, Joshua Tenenbaum, and Jiajun Wu. 2019. Neurally-guided structure inference. In *International Conference on Machine Learning*. PMLR, 4144–4153.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. 2018. Which Training Methods for GANs do actually Converge?. In *International Conference on Machine Learning (ICML)*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. 2018. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *The IEEE 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.

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

Ondrej Šťava, Bedrich Beneš, Radomír Měch, Daniel G Aliaga, and Peter Krištof. 2010. Inverse procedural modeling by automatic generation of L-systems. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 665–674.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. 2019. Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations*.

Homer Walke, R Kenny Jones, and Daniel Ritchie. 2020. Learning to infer shape programs using latent execution self training. *arXiv preprint arXiv:2011.13045* (2020).

Chenming Wu, Haisen Zhao, Chandrakana Nandi, Jeffrey I Lipton, Zachary Tatlock, and Adriana Schulz. 2019. Carpentry compiler. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl DD Willis, and Daniel Ritchie. 2021. Inferring cad modeling sequences using zone graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6062–6070.

Wenjie Ye, Yue Dong, Pieter Peers, and Baining Guo. 2021. Deep Reflectance Scanning: Recovering Spatially-varying Material Appearance from a Flash-lit Video Sequence. *Computer Graphics Forum* (2021). https://doi.org/10.1111/cgf.14387 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14387

Xilong Zhou and Nima Khademi Kalantari. 2021. Adversarial Single-Image SVBRDF Estimation with Hybrid Training. *Computer Graphics Forum* 40, 2 (2021), 315–325. https://doi.org/10.1111/cgf.142635 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.142635