

End-to-End Procedural Material Capture with Proxy-Free Mixed-Integer Optimization

BEICHEN LI, MIT CSAIL, USA
 LIANG SHI*, MIT CSAIL, USA
 WOJCIECH MATUSIK, MIT CSAIL, USA

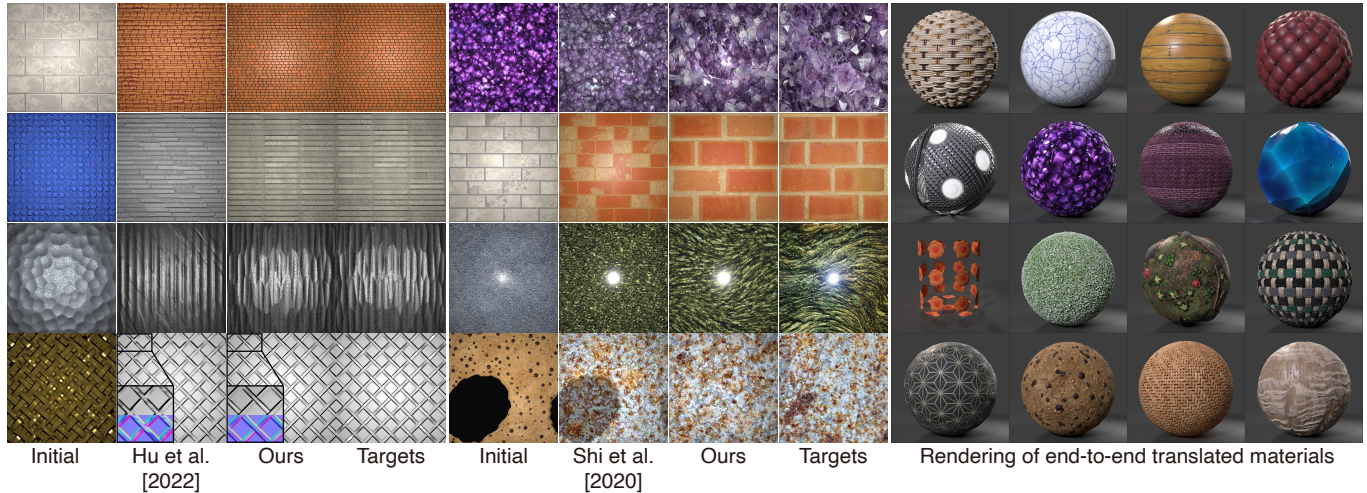


Fig. 1. Comparison of single-image procedural material capture with Neural Proxies [Hu et al. 2022a] (left) and MATCh [Shi et al. 2020] (middle). See Sec. 4.2 for a detailed discussion. (Right) Our new differentiable procedural material library, Diffmat v2, is able to translate artist-created production-grade procedural materials into differentiable programs without relying on proprietary automation tools [Shi et al. 2020]. See Appendix Fig. 10 for additional examples.

Node-graph-based procedural materials are vital to 3D content creation within the computer graphics industry. Leveraging the expressive representation of procedural materials, artists can effortlessly generate diverse appearances by altering the graph structure or node parameters. However, manually reproducing a specific appearance is a challenging task that demands extensive domain knowledge and labor. Previous research has sought to automate this process by converting artist-created material graphs into differentiable programs and optimizing node parameters against a photographed material appearance using gradient descent. These methods involve implementing differentiable filter nodes [Shi et al. 2020] and training differentiable neural proxies for generator nodes to optimize continuous and discrete node parameters [Hu et al. 2022a] jointly. Nevertheless, Neural Proxies exhibits critical limitations, such as long training times, inaccuracies, fixed resolutions, and confined parameter ranges, which hinder their scalability towards the broad spectrum of production-grade material graphs.

*Corresponding author: liangs@mit.edu

Authors' addresses: Beichen Li, MIT CSAIL, USA, beichen@mit.edu; Liang Shi, MIT CSAIL, USA, liangs@mit.edu; Wojciech Matusik, MIT CSAIL, USA, wojciech@csail.mit.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

0730-0301/2023/8-ART153 \$15.00

<https://doi.org/10.1145/3592132>

These constraints fundamentally stem from the absence of faithful and efficient implementations of generic noise and pattern generator nodes, both differentiable and non-differentiable. Such deficiency prevents the direct optimization of continuous and discrete generator node parameters without relying on surrogate models.

We present *Diffmat v2*, an improved differentiable procedural material library, along with a fully-automated, end-to-end procedural material capture framework that combines gradient-based optimization and gradient-free parameter search to match existing production-grade procedural materials against user-taken flash photos. *Diffmat v2* expands the range of differentiable material graph nodes in *Diffmat* [Shi et al. 2020] by adding generic noise/pattern generator nodes and user-customizable per-pixel filter nodes. This allows for the complete translation and optimization of procedural materials across various categories without the need for external proprietary tools or pre-cached noise patterns. Consequently, our method can capture a considerably broader array of materials, encompassing those with highly regular or stochastic geometries. We demonstrate that our end-to-end approach yields a closer match to the target than MATCh [Shi et al. 2020] and Neural Proxies [Hu et al. 2022a] when starting from initially unmatched continuous and discrete parameters.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: procedural materials, material acquisition

ACM Reference Format:

Beichen Li, Liang Shi, and Wojciech Matusik. 2023. End-to-End Procedural Material Capture with Proxy-Free Mixed-Integer Optimization. *ACM Trans. Graph.* 42, 4, Article 153 (August 2023), 15 pages. <https://doi.org/10.1145/3592132>

1 INTRODUCTION

The pursuit of photorealistic rendering has significantly increased in recent years, driven by emerging applications such as constructing digital twins of real-world environments for virtual and augmented reality, transferring robotic control policies learned in simulations to physical tasks, and training autonomous vehicles to navigate hazardous situations. In each of these scenarios, incorporating high-quality, detailed materials on 3D assets substantially enhances the perception of realism, which in turn improves task performance or user experience in downstream applications. Typically, materials in physics-based rendering are modeled by spatially varying bi-directional reflectance distribution functions (SVBRDFs) and encoded as pixel maps. These pixel maps can be acquired by scanning real objects or created using image processing tools like Adobe Photoshop. However, this conventional authoring workflow presents challenges in creating meaningful appearance variations, adjusting resolutions, and ensuring seamless tiling.

Recently, procedural textures have emerged as a popular alternative to pixel maps for representing materials. A procedural texture is a computer-generated image produced through an algorithm. In the case of current production-grade materials, this algorithm is embodied by a functional node graph consisting of noise/pattern generator nodes and filter nodes. Procedural material graphs offer mathematical abstractions for abundant material appearances based on diverse graph structures. These appearances can be generated on-demand at any resolution, eliminating the need for explicit storage in raw pixels [Ebert et al. 2002]. Furthermore, the appearance of procedural materials can be semantically controlled via meta-parameters, which connect groups of low-level node parameters to high-level, intuitive meanings (e.g., the number of annual rings in a tree trunk). This allows artists to simply tweak a few exposed parameters rather than tediously manipulating low-level parameters.

Procedural textures are notably thriving due to the maturity of industry-standard authoring tools such as Adobe Substance 3D Designer (S3D) [Adobe 2022a]. However, mastering these tools demands considerable training. Professional artists may spend hours or even days creating a production-grade material graph and fine-tuning a multitude of parameters to adjust intricate details. In order to alleviate the burden of extensive parameter tweaking for artists, inverse procedural material modeling methods have been developed, which automatically predict or optimize the node parameters of an artist-created material graph to match a photographed appearance [Hu et al. 2019a; Shi et al. 2020]. The MATch framework, developed by Shi et al. [2020], translates material graphs into differentiable programs for optimizing continuous filter node parameters using gradient descent. Hu et al. [2022a] further extends this approach by training differentiable neural network approximations for generator nodes, enabling the joint optimization of continuous and discrete parameters.

Despite some successful examples, these inverse methods struggle to scale well against the vast spectrum of production-grade procedural materials due to the absence of accurate, computationally efficient noise/pattern generator nodes. MATch relies on pre-cached generator node outputs using S3D's proprietary tools, which leads

to failure when initial and target textures do not match in spatial scaling. Neural Proxies [2022a] can optimize generator nodes but suffers from several fundamental limitations, including long training times, inaccuracies, fixed resolutions, and poor generalization outside training data. Therefore, their method has only been demonstrated on a narrow range of materials containing one or two supported generator nodes.

To address the inadequacies of current differentiable procedural material capture approaches, we introduce Diffmat v2, an advanced differentiable procedural material modeling library with three key additions to Diffmat [Shi et al. 2020]: generic differentiable noise generators, stochastic pattern generators, and custom differentiable pixel shading functions. Diffmat v2's dedicated node graph system for noise and pattern generation accurately reproduces the FX-Map node in S3D, a complex generator node that serves as the core building block for almost all procedural noises and patterns. Our reproduction achieves near-per-pixel accuracy, where the only difference lies in random number generation. The underlying implementation is carefully engineered to provide runtimes comparable to S3D (see Table 1). Moreover, Diffmat v2 includes a specialized node graph system for creating custom differentiable pixel shaders, corresponding to S3D's Pixel Processor node. Thanks to its functional enhancements, Diffmat v2 stands as the first self-contained library capable of automatically translating production-grade procedural materials into equivalent differentiable programs without relying on S3D's proprietary tools.

Diffmat v2 substantially enriches the collection of existing materials that can be fully translated and optimized. More importantly, it enables the co-optimization of continuous and discrete (integer) generator node parameters without neural network approximations. In contrast, Neural Proxies requires training a new model for any change in texture resolution, categorical node parameters, or default parameter ranges. On the other hand, our analytical implementation produces pixel-matching results irrespective of resolution and parameter settings while computing accurate gradients during backpropagation. Harnessing these advantages, we propose an end-to-end procedural material capture pipeline that combines gradient-based and gradient-free approaches to automatically optimize the entire set of node parameters in a prespecified material graph (e.g., obtained from nearest neighbor search [Shi et al. 2020]).

Compared to previous works [Hu et al. 2022a; Shi et al. 2020], our method captures a perceptually closer appearance to the target photo across diverse material categories. Additionally, we demonstrate the translation of high-level exposed parameters from a reference graph to a target graph for streamlined post-editing. In summary, our work incorporates new infrastructure and algorithms to enable more robust inverse procedure material capture and intuitive post-editing, offering artists enhanced support throughout the entire creative process.

Diffmat v2 is hosted on GitHub (<https://github.com/mit-gfx/diffmat>). Additional data is available at <http://diffmatv2.csail.mit.edu>.

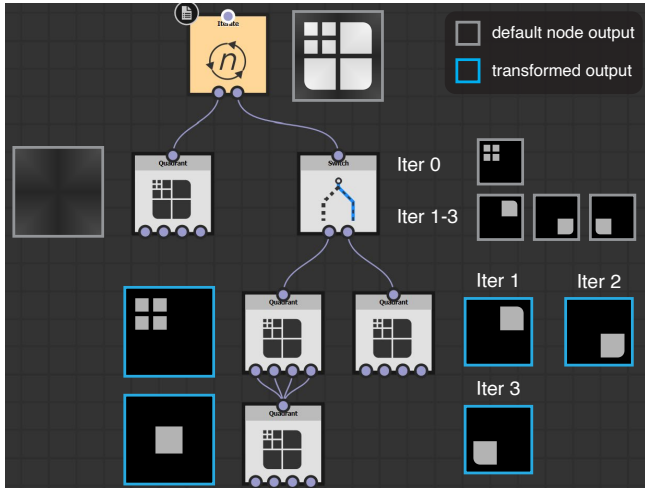


Fig. 2. Visualization of an example FX-Map graph that combines Quadrant, Iterate, and Switch nodes to generate an approximated FX-Map icon, where the output pattern of each node is shown next to its position. A Quadrant node can apply an optional 2D transformation to its output (in blue frames). In addition, some nodes generate patterns that vary between iterations since their behaviors are contingent on the iteration number.

2 RELATED WORK

Procedural noise is a fundamental tool for generating realistic textures for natural materials with rich visual details [Ebert et al. 2002]. It is largely classified into lattice, explicit, and sparse convolution noises [Lagae et al. 2010a]. Lattice noises are generated by interpolating random values/gradients defined at the grid points of an integer lattice. The most famous example is Perlin noise [Perlin 1985, 2002]. Meanwhile, alternatives to integer lattices were proposed to mitigate axis-aligned artifacts and reduce the computational cost, such as lattice convolution noise [Wyvill and Novins 1999] and simplex noise [Olano et al. 2002]. Explicit noises are generated in a more deterministic process [Cook and DeRose 2005; Fournier et al. 1982; Lewis 1986, 1987]. A prominent category of explicit noises is generated through Fourier spectral synthesis, which creates noises with distinct power spectra by filtering white noise in the frequency domain [Anjyo 1988; Gardner 1984; Goldberg et al. 2008; Saupe 1988; van Wijk 1991; Voss 1988]. Finally, sparse convolution noises are produced as the sum of randomly positioned and weighted kernel functions [Lagae et al. 2009; Lewis 1984, 1989; Praun et al. 2000]. We refer readers to the survey by Lagae et al. [2010a] for a more detailed discussion.

Inverse procedural appearance capture optimizes a procedural appearance model to match materials against user-specified examples. Early attempts used hand-crafted templates and rules to extract structural and local information from the source examples [Dischler et al. 2002; Eisenacher et al. 2010; Lefebvre and Poulin 2000; Lefebvre et al. 2010]. The effectiveness of these methods primarily depends on the quality of the templates, and their applications are often restricted to certain material categories. For more stochastic and isotropic texture examples, procedural noises are employed to directly match their appearances by optimizing the power spectrum

in both the image and frequency domains [Galerne et al. 2012; Guo et al. 2019; Lagae et al. 2010b]. Example-based noises such as local random phase noises [Gilet et al. 2014], Texton noise [Galerne et al. 2017], and histogram-preserving blending operator [Heitz and Neyret 2018] were proposed to accommodate more appearance variations. Hu et al. [2022b] expands upon this concept by modeling procedural noises using hierarchical, multi-frequency texture decomposition. However, the expressiveness and general usability of the resulting materials are limited due to an insufficient set of noise patterns, filters, and decomposition models [Hu et al. 2022b].

The advances in node-based procedural material authoring tools (e.g., Adobe Substance 3D Designer, Blender) facilitated the creation of complex procedural materials. Simultaneously, these tools fostered an online community where artists can freely share and reuse high-quality assets. Such a phenomenon has inspired a new line of methods that leverage artist-designed production-grade graphs to match photographed material appearances. Hu et al. [2019a] predicts the exposed parameters (artist-defined high-level meta-parameters) in a material graph using neural networks. Shi et al. [2020] employs gradient descent to optimize both exposed and low-level node parameters based on a differentiable version of S3D (Diffmat). As an improvement, Hu et al. [2022a] further develops differentiable neural proxies to approximate generator nodes that are non-differentiable or challenging to implement, enabling the joint optimization of continuous and discrete parameters. Yeh et al. [2022] applies Diffmat to procedural material capture from indoor 3D photos. Last but not least, Guerrero et al. [2022] establishes a new research direction by training a transformer network to generate new material graphs unconditionally. Their work paves the way for learning-based procedural material synthesis from image prompts. Nonetheless, we continue to focus on node parameter optimization for existing graph structures, substantially improving its capabilities and practicality compared to previous works.

For a review of non-procedural inverse appearance capture methods, we refer readers to Appendix A.

3 METHOD

This section introduces the new computational infrastructure, Diffmat v2, and our end-to-end inverse procedural material capture pipeline. We first describe the theory and differentiable implementation of FX-Map (Sec. 3.1) and Pixel Processor (Sec. 3.2). They are fundamental to replicating sophisticated generator and filter nodes in S3D. Next, we discuss how to update continuous and discrete node parameters without neural network approximations by alternating between gradient-based and gradient-free optimization (Sec. 3.3). Finally, we introduce an efficient approach to translating exposed parameters from a reference material to a target material (Sec. 3.4), providing artists knobs for high-level semantic material editing.

3.1 FX-Map Theory and Implementation

The FX-Map node, according to S3D, is the most powerful and complex node that enables the procedural creation of texture images. Functionally, FX-Map represents a generalized noise generation procedure that recursively subdivides the texture canvas into smaller regions and renders atomic texture patterns in these regions. The

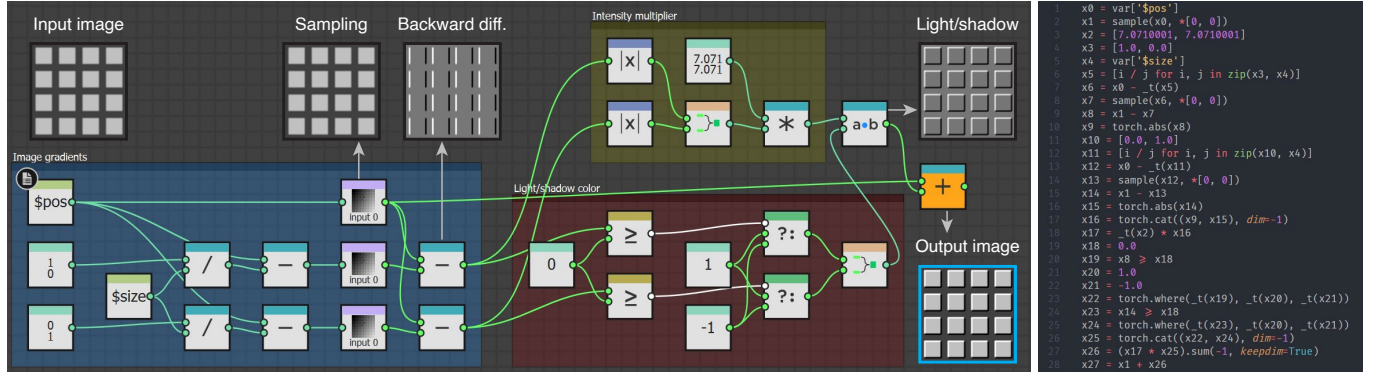


Fig. 3. A simplified directional emboss filter implemented using Pixel Processor, which applies a simulated lighting effect to a grayscale height map. The function node graph is organized by colored frames to showcase groups of nodes that constitute an integral functionality. Intermediate results at crucial nodes are visualized, where images with negative values are normalized to $[0, 1]$. Diffmat v2 compiles the function graph into Python code sequence on the right side of the figure, where per-pixel instructions are automatically generated by referencing PyTorch API.

subdivision follows a quadtree structure with a canvas area evenly split into four squares in the next quadtree layer. The 2^{2d} squares at the d -th quadtree layer constitute an octave of depth d . The rendering step draws one or multiple atomic patterns at the center of each square (d, i, j) ($i, j \in \{0, 1, \dots, 2^d - 1\}$). Each atomic pattern $F_{d,i,j,k}(x, y)$ (k is the pattern index; x, y are pixel coordinates) undergoes a user-defined spatial transformation $\xi_{d,i,j,k}[\cdot]$ that can be either deterministic or random. Deguy and Benassi [2001] showed that the FX-Map procedure achieves a close approximation of intermittent locally self-similar Gaussian processes (ILSSGP), of which fractional Brownian motion (fBM) is a special case. On a $2^M \times 2^M$ canvas $\mathbb{C}$ with a max octave depth D , the FX-Map output $I(x, y)$ is defined by the following recursive formula (x, y are omitted in image-level calculation for simplicity)

$$T_{d,i,j}(x, y) = \sum_{k=1}^{R_{d,i,j}} \alpha_{d,i,j,k} \cdot \xi_{d,i,j,k}[F_{d,i,j,k}](2^d x - 2^{M-d} i, 2^d y - 2^{M-d} j) \quad (1)$$

$$S_{d,i,j} = \begin{cases} T_{d,i,j} & d = D \\ T_{d,i,j} + \sum_{u,v \in \{0,1\}} T_{d+1,2i+u,2j+v} & 0 \leq d < D \end{cases} \quad (2)$$

$$I = S_{0,0,0} \quad (3)$$

Here, $T_{d,i,j}(x, y)$ blends the $R_{d,i,j}$ rendered atomic patterns at square (d, i, j) , where the intensity of each pattern is parameterized by $\alpha_{d,i,j,k}$. $S_{d,i,j}$ denotes the partial texture after blending all atomic patterns in the sub-quadtree of square (d, i, j) . Under this formulation, FX-Map's construction process is conceptually equivalent to the depth-first traversal of the quadtree:

- (1) Starting from the root ($d = 0$), every quadtree node generates $R_{d,i,j}$ atomic patterns $F_{d,i,j,k}$, each transformed by $\xi_{d,i,j,k}[\cdot]$ and rendered on the canvas with intensity $\alpha_{d,i,j,k}$.
- (2) Enumerate four child nodes and traverse their subtrees recursively. Return to the parent node upon completion.
- (3) Repeat (2) until returning to the root node.

If the same pattern function F is used for all atomic patterns and $R_{d,i,j} \equiv 1$, FX-Map effectively constructs an fBM noise by reusing the same noise map (with scaling and shifting) in all octaves instead of resampling the noise with different parameters at each octave.

The FX-Map node in Adobe Substance 3D Designer is designed using a dedicated quadtree graph system comprising three specialized nodes: *Quadrant*, *Iterate*, and *Switch*. A *Quadrant* node generates an atomic pattern and renders the pattern onto the canvas under a user-defined transformation function. This node also subdivides the rendering region into four identical quadrants, each corresponding to one child node at the next quadtree layer and accessible via an input connection. Moreover, S3D can construct a directed multigraph from the FX-Map quadtree by collapsing identical *Quadrant* nodes at the same depth into a single node. As a result, a chained *Quadrant* node sequence is equivalent to a full quadtree during top-down traversal, allowing for the generation of highly intricate texture maps. Building on quadtree structure defined by *Quadrant* nodes, *Iterate* and *Switch* nodes specify repetition and branching behaviors during quadtree traversal, respectively. An *Iterate* node accepts two inputs, first drawing the left subtree on the canvas, then rendering the right subtree repeatedly for a user-specified number of iterations. Introducing randomness to the right subtree further increases texture complexity since the randomness applies to each iteration individually. In contrast, A *Switch* node renders either subtree based on a custom Boolean decision function. This node is commonly used to prune unwanted subtrees or alternate between branches during quadtree traversal. In Fig. 2, we show a simple FX-Map instance consisting of all three node types. Intermediate results produced by each node are placed aside to visualize node behaviors.

Without access to S3D's proprietary codebase, we developed an in-house implementation of the FX-Map graph system using PyTorch [Paszke et al. 2019] and the Taichi programming language [Hu et al. 2019b] after thoroughly comprehending its functionalities. As with S3D, our implementation includes full-featured versions of *Quadrant*, *Iterate*, and *Switch* nodes. Diffmat v2 accurately reproduces S3D's output for FX-Map and other noise/pattern generator nodes built upon various FX-Map instances, with only minor

differences in random number generation (see Fig. 4). Our implementation reorganizes the FX-Map construction procedure into two phases: pattern generation and rendering. In pattern generation, the FX-Map quadtree is traversed depth-first through all three types of nodes. Upon visiting each node, we first compute the node parameters by fetching constant values from a look-up dictionary or evaluating Value Processors associated with dynamically-valued parameters. Next, we execute the node's function based on its type. Each Quadrant node produces a pattern rendering *job* parameterized by spatial transformation, color intensity, and blending option. Rendering jobs are collected in a global job queue in the order of their timestamps upon creation. The rendering phase begins once the traversal is complete. During this phase, an FX-Map renderer sequentially transforms atomic patterns in the job queue and draws them on the canvas. The result is an FX-Map image. Besides adhering to the default FX-Map algorithm, our system implements various FX-Map nodes using different acceleration methods. This allows us to avoid the exponential scaling in computational cost due to the sequential nature of FX-Map traversal and rendering (see more technical details in the Appendix).

3.2 Graph System for Pixel Processor

Pixel Processor is a generic node-graph-based per-pixel shader in S3D, where the user defines a function graph that computes a numerical output for each pixel. Unlike most atomic filter nodes that only conduct prescribed operations, a Pixel Processor node can execute mathematical functions of arbitrary complexities based on rudimentary numeric and logical operators. Therefore, many advanced filter nodes are represented by Pixel Processors, such as the Sobel operator, geometric warping, Cartesian-to-polar coordinates conversion, and even ray tracing. Fig. 3 demonstrates a Pixel Processor that computes a pixel-wise scalar function to emboss a grayscale image. S3D defines diverse function graph nodes for value definition and manipulation, and we refer readers to the official documentation for a full catalog [Adobe 2022b].

Despite the ability to parse function graphs in S3D, Diffmat can not execute them in parallel per pixel. Thus, Diffmat v2 compiles S3D's function graph representation into an equivalent PyTorch program with unified support of scalar and per-pixel execution. The generated Python program is automatically differentiable whenever it references a continuous parameter or an intermediate texture map that receives gradient. Diffmat v2 also implements many missing function nodes in Diffmat, such as image sampling nodes and Set/Sequence nodes (see technical details in the Appendix). These additions are a cornerstone for successfully reproducing many noise generators in S3D (see Fig. 5).

3.3 End-to-end Inverse Procedural Material Capture

Our material capture pipeline augments MATch with integer parameter optimization. Given a user-provided or algorithm-selected material graph, let its continuous parameters be $\theta = (\theta_f, \theta_g, \bar{\theta}_g)$, where θ_f , θ_g , $\bar{\theta}_g$ are the vectors of differentiable continuous parameters in filter nodes, differentiable continuous parameters in generator nodes, and non-differentiable continuous parameters in generator nodes, respectively. Similarly, let its discrete (integer)

parameters be $\phi = (\phi_f, \phi_g)$, where ϕ_f and ϕ_g are the vectors of discrete parameters in filter nodes and generator nodes, respectively. Finally, we set $e = (e_\theta, \bar{e}_\theta, e_\phi)$, where e_θ represents exposed, differentiable continuous parameters, $\bar{e}_\theta$ represents exposed, non-differentiable continuous parameters, and e_ϕ represents exposed discrete parameters. Similar to Hu et al. [2022a], our optimization consists of three stages.

Stage 1: Gradient-based initialization. In this stage, we adjust the continuous parameters to approximately match the global albedo and align the geometries if the graph includes affine transformation nodes. When the initial material appearance deviates significantly from the target, optimizing integer parameters directly may not yield an ideal solution. This step aims to reduce the appearance gap and facilitate convergence to a better solution in the next stage. Our framework allows the user to determine the granularity of optimization by stopping the gradient for θ_f , θ_g , and e_θ , respectively. This is crucial for maintaining predefined material structures (e.g., preventing undesired randomness in uniform tile or weave patterns) or preserving the interpretability of optimized parameters (e.g., confining the optimization to exposed parameters). Without loss of generality, we assume all differentiable continuous parameters are optimizable. The optimization is given by

$$\theta_f^{(1)}, \theta_g^{(1)}, e_\theta^{(1)} \leftarrow \arg \min_{\theta_f, \theta_g, e_\theta} \| \mathcal{T}(I(\theta, \phi, e)) - \mathcal{T}(I) \|, \quad (4)$$

$\mathcal{T}$ is a texture descriptor operator that computes the Gram matrix of extracted VGG19 features [Gatys et al. 2015]. $I(\cdot)$ denotes the rendered texture image under an input node parameter configuration. I refers to the target photo. $\|\cdot\|$ represents L1 loss. We run no more than 2000 iterations of gradient descent.

Stage 2: Gradient-free mixed integer optimization. After gradient-based initialization, we employ gradient-free methods to optimize integer parameters ϕ , continuous generator node parameters $(\theta_g, \bar{\theta}_g)$, and exposed parameters e . We optionally include a portion of differentiable continuous parameters here because they either control input noises/patterns alongside discrete parameters (i.e., θ_g) or specify semantic constraints among node parameters (i.e., e). The user may choose to ignore them to reduce the granularity of optimization.

Mathematically, this stage constitutes a mixed-integer non-linear programming (MINLP) problem that entails NP-hard combinatorial optimization [Belotti et al. 2013]. Since the concatenation of a procedural material graph, a differentiable rendering layer, and a neural network texture descriptor creates an exceedingly sophisticated objective landscape that precludes analytical simplifications [Lee and Leyffer 2011], we turn to generic mixed-integer optimizers such as Bayesian optimization (BO), simulated annealing (SA), and genetic algorithm (GA). BO is computationally expensive when optimizing numerous continuous and integer parameters in a production-grade material graph. SA and GA are hindered by poor sample efficiency and may produce implausible material appearances due to aggressive random exploration. To balance exploration, stability, and computational cost, we ultimately opt for a coordinate descent approach based on the intuition that most node parameters affect relatively orthogonal properties (e.g., color, size, and offset) and can

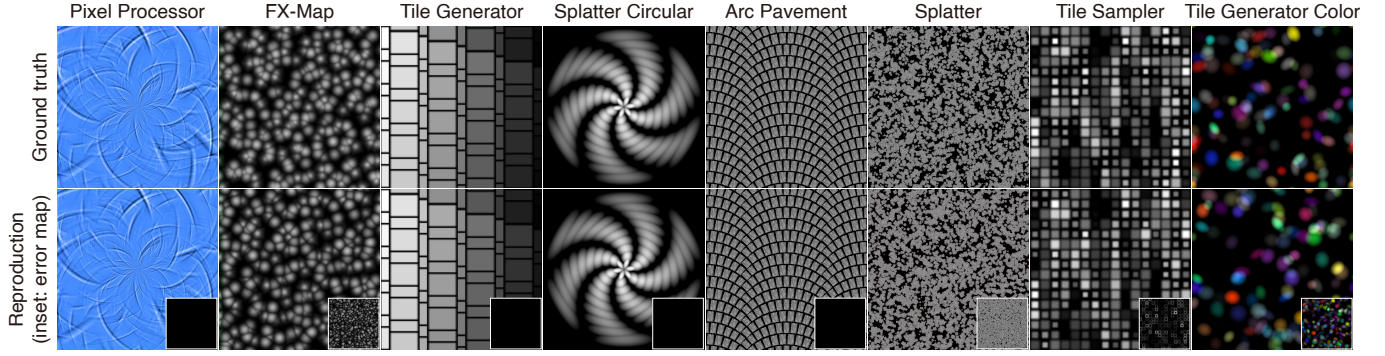


Fig. 4. omparison of Diffmat v2's reproduced FX-Map, Pixel Processor, and other pattern generator node outputs with S3D (ground truth). An error map is visualized at the bottom right corner of each reproduction. All pattern generator nodes (Column 3-8) are created from various FX-Map instances. Diffmat v2 achieves per-pixel match in nodes that do not implement any randomness or whose randomness is disabled by node parameters (Column 1, 3-5). Otherwise, our reproduction is perceptually identical but pixel-wise different (Column 2, 6-8) because S3D's pseudo-random number generation algorithm is non-public.

be optimized independently. Our method alternates between individual parameter optimization using linear grid search. Specifically, we cycle through all active parameters in each epoch and update each parameter while holding others constant. We use a search step of 1 for integer parameters and 0.01 for normalized continuous parameters within their default parameter ranges. Unlike Neural Proxies, which generalizes poorly beyond training data, our method allows the user to reconfigure the search ranges arbitrarily. The gradient-free optimization is guided by a weighted loss function

$$\begin{aligned} \phi^{(2)}, \theta_g^{(2)}, \bar{\theta}_g^{(2)}, e^{(2)} \leftarrow \arg \min_{\phi, \theta_g, \bar{\theta}_g, e} & \| \mathcal{T}(\mathcal{I}(\theta, \phi, e)) - \mathcal{T}(I) \| \\ & + w_{\mathcal{F}} \| \mathcal{F}(\mathcal{I}(\theta, \phi, e)) - \mathcal{F}(I) \| \end{aligned} \quad (5)$$

where $\mathcal{F}$ is the 2D Fast Fourier Transform (FFT) operator, and $w_{\mathcal{F}}$ is the corresponding weight. The FFT loss is sensitive to structural changes and helps identify desired values for integer parameters. The optimization typically converges within five epochs.

Stage 3: Gradient-based fine-tuning. With a closer estimation for integer parameters from the second stage, we revert to gradient-based optimization (using the same formula as Eq. (4)) to fine-tune the differentiable continuous parameters. We optionally repeat stages 2 and 3 to improve the result further, but in most cases, the process converges in no more than two repetitions.

3.4 Exposed Parameter Translation

Procedural materials are not always created with labeled subgraphs and exposed high-level parameters. These graphs are challenging to reuse because artists have to deduce the purpose of low-level nodes and manually modify their parameters. However, many other graphs within the same material category may have well-defined exposed parameters. We present a straightforward yet effective method for distilling and translating continuous exposed parameters from a reference graph to a target graph. Our method stems from a fundamental insight—the semantics of an exposed parameter are characterized by its impact on material appearance. Therefore, an exposed parameter e_r in a reference graph G_r should induce the same perceptual change in the output texture after being migrated

to a target graph $G_t \neq G_r$. This premise enables us to pinpoint the node parameters in G_t that correlate the most with e_r by matching texture descriptor changes.

Let $\theta_r(e_r)$ denote the vector of continuous node parameters in the reference graph controlled by e_r . We first grid-sample e_r in G_r to compute a set of Gatys texture descriptors (the texture rendering function is denoted by $\mathcal{I}_{G_r}(\cdot)$). Assuming that all continuous parameters have a normalized domain $[0, 1]$, we use a step size of $\delta = 1/N$ and compute the i -th texture descriptor $\mathcal{T}_{G_r}^{(i)}$ using

$$\mathcal{T}_{G_r}^{(i)} = \mathcal{T}(\mathcal{I}_{G_r}(\theta_r(e_r = i\delta))) \quad (i = 0, 1, \dots, N) \quad (6)$$

The appearance change in the reference graph from $e_r = 0$ to $e_r = i\delta$ is then derived by

$$\Delta \mathcal{T}_{G_r}^{(i)} = \mathcal{T}_{G_r}^{(i)} - \mathcal{T}_{G_r}^{(0)} \quad (7)$$

Our method transfers these texture descriptor changes to G_t through node parameter optimization. Here, we reuse the notations above to define θ_t and $\mathcal{I}_{G_t}(\cdot)$ for G_t . Additionally, θ_t^* denotes G_t 's default parameter values. The following problems are solved to match the impact of e_r in G_r at each sampled value ($i = 0, 1, \dots, N$):

$$\hat{\theta}_t^{(i)} = \arg \min_{\theta_t} \left(\left\| \mathcal{T}(\mathcal{I}_{G_t}(\theta_t)) - \mathcal{T}(\mathcal{I}_{G_t}(\theta_t^*)) - s \Delta \mathcal{T}_{G_r}^{(i)} \right\| + \mathcal{L}_{\text{reg}}^{(i)} \right) \quad (8)$$

where s is a hyperparameter amplifying (or attenuating) the transferred texture descriptor difference. $\mathcal{L}_{\text{reg}}^{(i)}$ is a regularization term that compares θ_t against the previous best solution $\hat{\theta}_t^{(i-1)}$ to encourage a smooth transition in material appearance

$$\begin{aligned} \mathcal{L}_{\text{reg}}^{(i)}(\theta_t) &= 0 & (i = 0) \\ \mathcal{L}_{\text{reg}}^{(i)}(\theta_t) &= \lambda \left\| \mathcal{I}_{G_t}(\theta_t) - \mathcal{I}_{G_t}(\hat{\theta}_t^{(i-1)}) \right\| \\ &\quad + \eta \left\| \mathcal{T}(\mathcal{I}_{G_t}(\theta_t)) - \mathcal{T}(\mathcal{I}_{G_t}(\hat{\theta}_t^{(i-1)})) \right\| & (i > 0) \end{aligned} \quad (9)$$

where λ and η are the weights for texture image and texture descriptor losses, respectively.

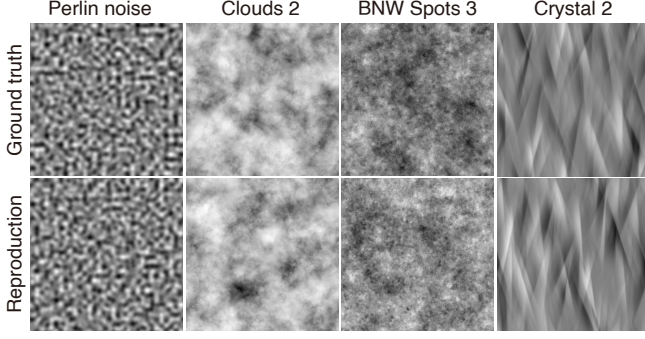


Fig. 5. Comparison of noise generator outputs by Diffmat v2 and S3D (ground truth). We analyze the statistics of noise usage in the Adobe Substance 3D Assets Library and visualize four representative noises as examples. Owing to the utilization of a non-public pseudo-random number generator in S3D, our reproduction is statistically indistinguishable from the ground truth, though not pixel-perfect.

Finally, we select the top K node parameters in G_t based on the magnitude of texture descriptor difference (between $\hat{\theta}_t^{(N)}$ and θ_t^*) contributed by each node parameter. This step discerns the essential node parameters underlying the transferred exposed parameter e_t . For each selected parameter k , we extract its optimized value sequence from $\hat{\theta}_t^{(i)}$ ($i = 0, 1, \dots, N$) and fit an M -th degree polynomial function $\mathcal{P}_k(e_t) = \sum_{m=0}^M \beta_{k,m} e_t^m$ from e_t using least squares, where $\beta_k = \{\beta_{k,0}, \dots, \beta_{k,M}\}$ is the set of polynomial coefficients for the k -th selected parameter. These polynomial functions can be easily constructed using S3D’s function graphs.

4 RESULTS

4.1 Validation and Benchmarking of Diffmat v2

We validate the reproduction quality of Diffmat v2’s new modalities by comparing them side-by-side with S3D results. Fig. 4 compares the output of custom FX-Map and Pixel Processor instances and other pattern generator nodes; Fig. 5 compares a set of frequently used noise generators. Diffmat v2 matches the ground truth precisely in nodes that do not implement randomness or whose randomness is disabled by node parameters. The reproductions of nodes with internal randomness are statistically identical to the ground truth but not pixel-perfect since S3D uses non-public pseudo-random number generators.

We use Diffmat v2 to translate artist-created production-grade procedural materials across various categories without using S3D’s proprietary automation tools. Fig. 1 (right) displays a selection of 16 examples (see Fig. 10 for the full collection). Most of them contain highly structured or complex stochastic geometries enabled by Diffmat v2’s functional additions. The removal of external dependency saves a substantial amount of disk storage since input noises and patterns are generated on the fly rather than pre-cached into images.

Furthermore, fast forward and backward evaluations are necessary for interactive editing and rapid inverse capture. Through extensive performance engineering (see Appendix Sec. B), we ultimately achieve runtimes comparable to S3D for every supported

generator node in Diffmat v2 (see Table 1 for a comparison using selected generator nodes). Forward and backward evaluation speeds at the graph level are reported in Appendix Table 2.

4.2 Material Capture from Synthetic and Real Images

We demonstrate the effectiveness and superiority of our method by comparing it with state-of-the-art solutions on both synthetic and real-world targets.

Comparison with MATch [2020]. To ensure a fair comparison, we preprocess material graphs that cannot be fully translated by Diffmat using the following steps: 1) we replace unsupported generator nodes with pre-cached bitmaps using S3D; 2) we also pre-cache unsupported filter nodes (including FX-Map s and Pixel Processor s) and freeze their predecessor subgraphs during optimization. The results are shown in Fig. 1 (middle) and Fig. 6. In Fig. 1 (middle), the top three results demonstrate our ability to jointly optimize continuous and discrete parameters. Although the initial integer parameters result in mismatched texture scaling, our method manages to recover the correct scales. The last example contains a binary switch that controls a cookie material’s “bites” feature. When re-targeting the material at a rusty metal image, MATch can only lighten the bitten area using filter nodes since the hole is encoded in a pre-cached bitmap. Instead, our method directly turns off the “bites” switch during integer parameter optimization, yielding a solid rusty metal texture. A similar example is the removal of a forest ground material’s “skull” feature at Fig. 6 Row 3 Column 3 (abbreviated as R3C3 hereafter). Additional examples of scale correction via integer parameter optimization can be found in Fig. 6 R5/R6/R7C1/R8C2. Fig. 6 R1 shows that our approach is able to match two distinct fabric photos with the same source material. MATch merely adjusts the color and slightly tweaks the yarn substrate and embroidery threads. In contrast, our method completely alters the fabric geometry by 1) R1C1: changing the substrate yarns into reflective brown-green wool swirls and completely removing the horizontal threads; 2) R1C2: transforming sector-shaped thread patterns into square shapes and tinting them with blue and gold colors. Meanwhile, our method flattens the rounded-shaped wood plank at R2C1 to resemble the target, and reconstructs the normal map of the porous rocks at R2C2 more accurately. Other relevant cases include R4C2 (the intricate microfiber detail of the carpet) and R8C1 (removed scattered small dents on the crystal surface). Besides geometries, our method also more closely aligns with the target image in other material properties, such as roughness (see Fig. 6 R3C1/R6C2/R7C2). In summary, our method’s improvements are thoroughly assessed among a wide range of material categories, including fabric, leather, concrete, ground, wood, stone, porcelain, plastic, as well as organic and translucent materials.

Comparison with Neural Proxies [2022a]. Our method captures the target more faithfully without being restricted to a predetermined resolution or parameter range (see Fig. 1 (left) and Fig. 7). Before delving into the comparison, it is important to note that most material graphs used in the MATch comparison contain over ten generator nodes not provided by Hu et al. [2022a]. Consequently,

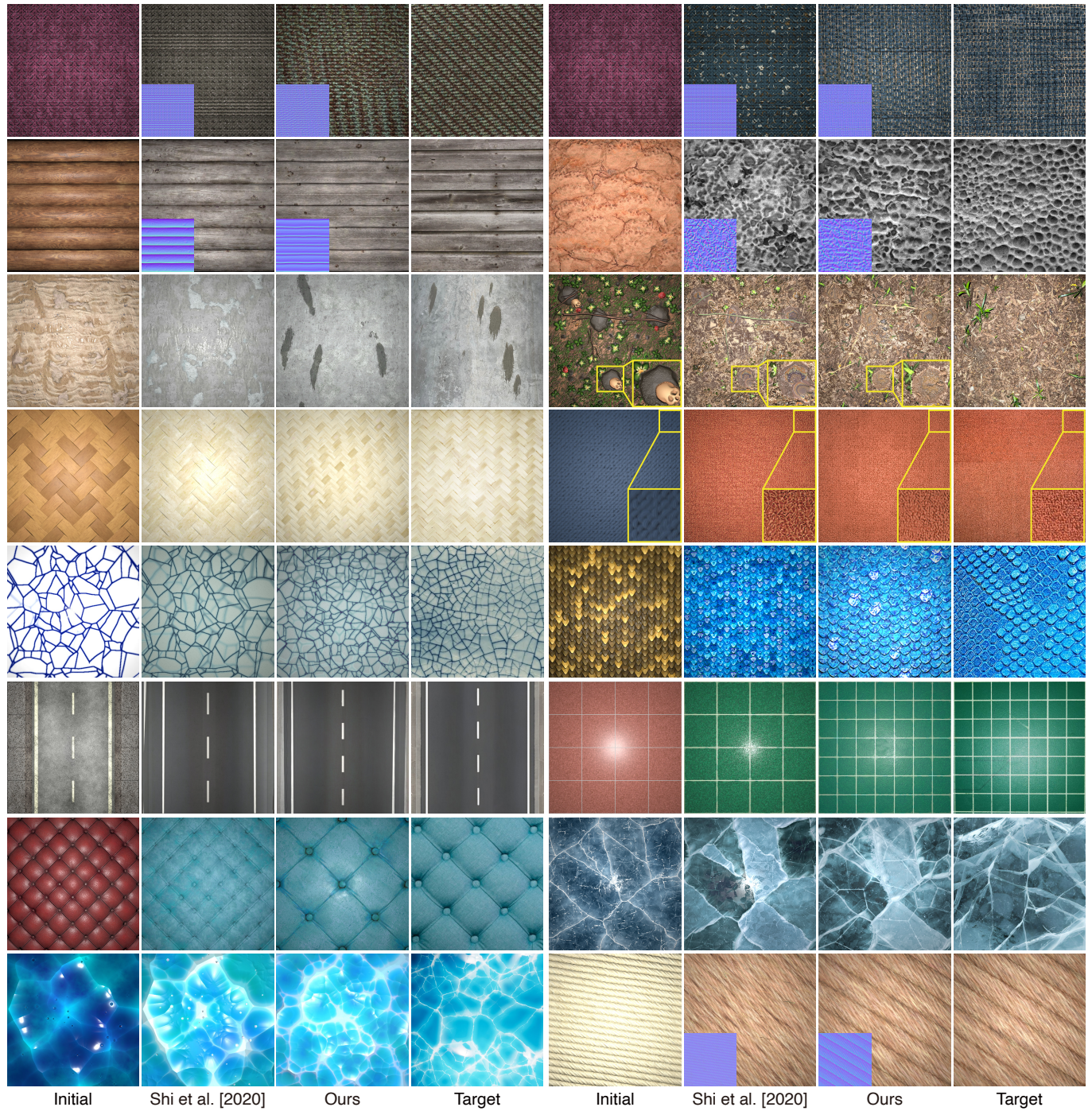


Fig. 6. Additional comparisons with MATch [Shi et al. 2020]. For each example, we optionally display the optimized normal map at the bottom left or provide a zoomed-in view of a specific region of interest. Please refer to Sec. 4.2 Paragraph 1 for a detailed discussion.

Table 1. Runtime comparison of generator nodes in Diffmat v2 and S3D at different texture resolutions. Our benchmark includes forward and backward evaluation times for several frequently used generator nodes, excluding backward evaluations for nodes that do not require gradient backpropagation (i.e., those defined without differentiable continuous parameters). In all cases, Diffmat v2 demonstrates comparable or even faster runtimes than S3D, enabling seamless integration into commercial-grade software without sacrificing performance. For the Make It Tile Patch node, Diffmat v2’s FX-Map supersedes Diffmat’s hard-coded implementation, resulting in a 300× speedup. Due to the substantial computational cost associated with the StyleGAN2 (> 25M parameters) network, a neural proxy takes 11 ms to evaluate at a 256×256 resolution, with the runtime scaling linearly (i.e., 44 ms at 512×512). In contrast, Diffmat v2 has a fixed kernel launch cost from PyTorch and Taichi, with the runtime scaling sub-linearly.

Node Name	Requires Gradient	256×256		512×512		1024×1024	
		Diffmat v2 (F/B)	S3D	Diffmat v2 (F/B)	S3D	Diffmat v2 (F/B)	S3D
Make It Tile Patch	Y	8 ms / 5 ms	11 ms	10 ms / 6 ms	13 ms	22 ms / 19 ms	13 ms
Tile Generator	Y	8 ms / 6 ms	2 ms	8 ms / 9 ms	2 ms	16 ms / 16 ms	4 ms
Brick Generator	Y	5 ms / 4 ms	16 ms	5 ms / 4 ms	16 ms	6 ms / 5 ms	18 ms
Stripes	Y	3 ms / 1 ms	6 ms	3 ms / 2 ms	6 ms	4 ms / 5 ms	7 ms
Arc Pavement	Y	6 ms / 3 ms	2 ms	6 ms / 4 ms	2 ms	8 ms / 5 ms	2 ms
Splatter	Y	7 ms / 4 ms	6 ms	7 ms / 4 ms	6 ms	12 ms / 8 ms	5 ms
Perlin Noise	N	5 ms / –	8 ms	6 ms / –	9 ms	7 ms / –	13 ms
Crystal 2	N	6 ms / –	3 ms	6 ms / –	5 ms	8 ms / –	15 ms
Clouds 2	N	11 ms / –	16 ms	13 ms / –	16 ms	15 ms / –	20 ms
BnW Spots 3	N	15 ms / –	24 ms	17 ms / –	25 ms	20 ms / –	26 ms

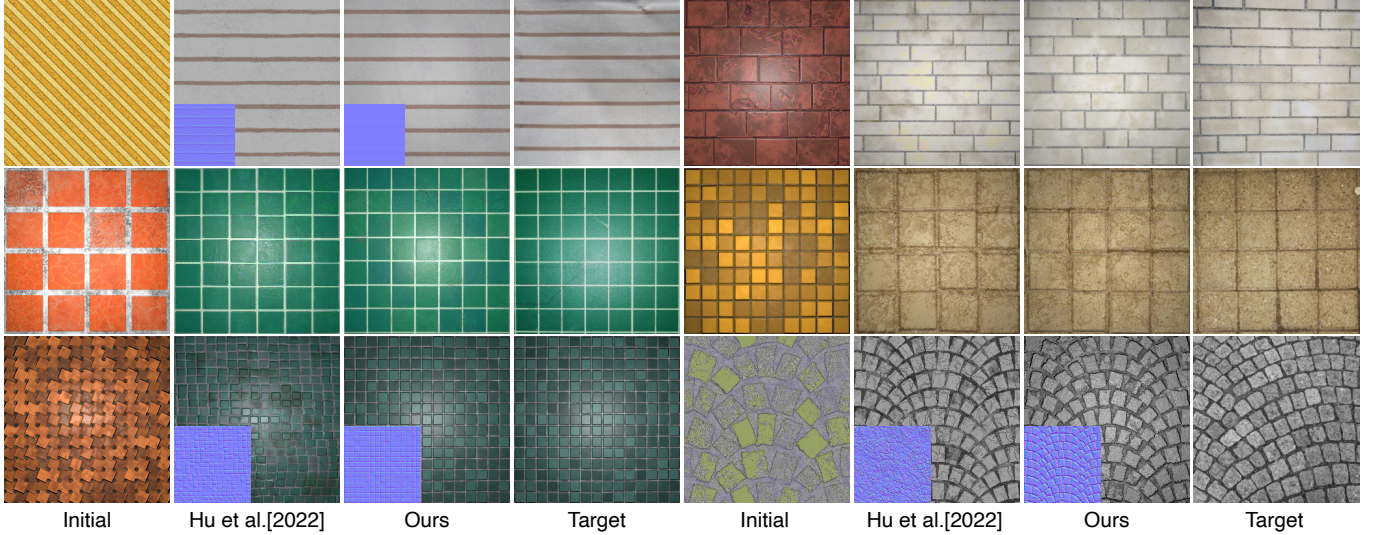


Fig. 7. Additional comparisons with Neural Proxies [Hu et al. 2022a]. The optimized normal maps are visualized at the bottom left for some examples. Sec. 4.2 Paragraph 2 provides a detailed discussion.

their approach is currently inapplicable to these examples. Furthermore, testing these examples poses out-of-memory risks, as each neural proxy already occupies ~1500MB GPU memory during forward evaluation at a 256×256 resolution. As such, we reuse the original graphs and results in Hu et al.[2022a] as a baseline. In Fig. 1 (left) R1, the target brick wall has a greater number of rows and columns than the training data of the brick generator network. Neural Proxies thus fails to converge to the correct numbers. Our analytical brick generator node implementation accommodates arbitrary parameter combinations, allowing the grid search to find a satisfactory match in a mere two epochs. Neural Proxies is also susceptible to local minima as the deep neural network approximation

creates a complicated loss landscape. The reproduction quality is often compromised when the initial guess is visually distant from the target (Fig. 1 (left) R2 and R3). Our method, on the contrary, conducts gradient-free optimization to escape such local minima and find better solutions. Additionally, the approximative nature of Neural Proxies introduces further errors when exporting optimized parameters back to S3D. These errors typically materialize as structural artifacts (e.g., incorrect fluctuations in the normal at Fig. 1 (left) R4 bottom left) or inconsistent SVBRDF maps (e.g., misaligned normal and albedo maps at Fig. 7 R3C2). Of equal importance, optimizing procedural materials at a high image resolution is often required to capture fine details in the target photo. Nevertheless,

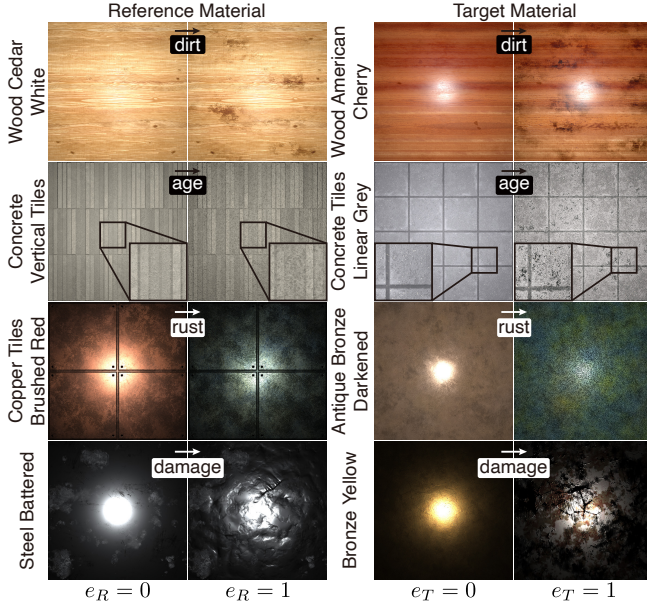


Fig. 8. Results of exposed parameter transfer between material graphs. The labeled arrow between each image pair indicates the name of the exposed parameter. We direct readers to the Supplementary Videos for transition animations when varying exposed parameters from 0 to 1. A detailed analysis of the results can be found in Sec. 4.3.

Neural Proxies can only operate in predetermined resolutions once trained. Diffmat v2’s resolution-agnostic implementation ensures the accurate reproduction of the finest details (as illustrated by the dirt details at Fig. 7 R2C2). Throughout the comparison experiment, we observed that the limitations of Hu et al. [2022a] frequently result in suboptimal solutions (e.g., incorrect integer parameters in Fig. 7 R1C2/R2C1/R3C1), while our method exhibits superior robustness.

4.3 Exposed Parameter Translation

Fig. 8 illustrates the translation of various exposed parameters from a reference material graph to a target material graph. These exposed parameters alter details in albedo (R1), normal (R2), color tone (R3), and material structure (R4). We note that the example in R4 is feasible entirely due to Diffmat v2’s differentiable generator nodes, which modify the geometry of the steel material. Our pixel-matching implementation effectively transfers these geometric changes to the target graph. The Supplementary Videos provide a visualization of material transitions in the reference and target materials under the control of exposed parameters.

4.4 Failure Cases

Fig. 9 showcases two instances where Diffmat v2 fails to match procedural materials against synthetic or real photographs successfully. In the first example, we attempt to optimize a complex, overparameterized mixed weave material whose node graph contains a large number of integer parameters. Our method is unaware that many integer parameters are implicitly correlated to preserve the weave

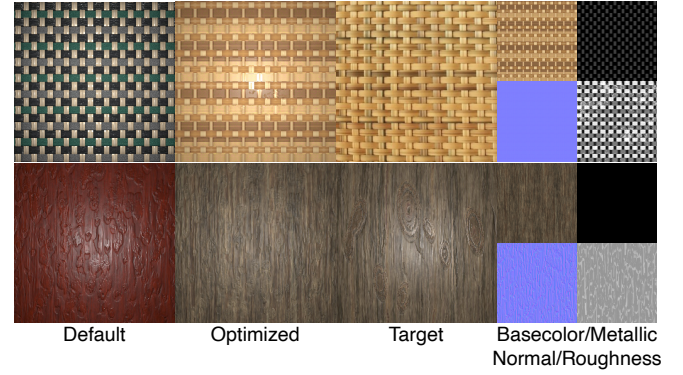


Fig. 9. Demonstration of failure cases when optimizing procedural materials against real-world (top) and synthetic (bottom) images.

structure. This material graph does not define any exposed parameter as explicit constraints, so it is incorrect to assume that their effects are orthogonal and optimize each parameter independently. As a result, our coordinate descent method gets trapped in local minima and disrupts the weave structure. A workaround for similar overparameterized cases is to define exposed parameters that specify underlying constraints, thereby reducing the search space. However, this requires an artist’s effort as Diffmat v2 can only automatically transfer continuous exposed parameters from similar graphs. The second failure case, also reported by Hu et al. [2022a], involves the Gatys texture descriptor only capturing global image features and overlooking subtle details (i.e., wood knots). To address this issue, the image feature extraction network could be enhanced to focus more on modeling textures than generic images. Alternatively, our method could be extended to capture material appearances from multiple angles and distances, providing stronger supervision of surface properties.

Like previous work in differentiable procedural material capture, our method does not infer the procedural material graph structure from the input image. Optimization will fail if the prespecified graph lacks crucial elements of the target appearance. Guerrero et al. [2022] tried to overcome this limitation by synthesizing novel graph structures using a deep generative model. However, it remains a challenge to accurately reconstruct production-grade procedural material graphs from target photos while ensuring that the output graph is easily comprehensible to artists.

5 CONCLUSION

We present an end-to-end pipeline for inverse procedure material capture based on Diffmat v2, a significantly enhanced computational infrastructure derived from the original Diffmat. Diffmat v2 fully translates and optimizes a vast array of artist-created, production-grade material graphs, which was previously infeasible due to the lack of differentiable noise and pattern generators. Our approach alternates between gradient-based and gradient-free optimization techniques to capture continuous and discrete parameters, resulting in higher-fidelity appearance reproductions than prior work. In addition, we introduce a novel method for example-based exposed parameter transfer, which simplifies the post-editing of materials

without high-level meta-parameters. This allows for easy reuse and retargeting of materials. Although our work does not directly address graph structure synthesis, we provide advanced tools to alleviate the burden on artists by automating low-level parameter tuning and high-level parameter translation.

The applicability of Diffmat v2 can be further expanded with some minor improvements. Firstly, it is currently unable to parse procedural graphs with hierarchical dependencies. To create intricate procedural materials, artists typically package subgraphs into custom nodes and incorporate them into the main material graph. Our parser, however, is unable to process such subgraphs, requiring users to manually extract and position them in the main graph for end-to-end translation. Secondly, Diffmat v2 does not support non-square resolutions. Artists frequently create non-square textures to accommodate specific object shapes or reduce computational expenses for intermediate maps (e.g., when producing 1D gradient maps). Although Diffmat v2 automatically extends the canvas to a square in these scenarios, supporting non-square textures will improve optimization efficiency, especially at very high resolutions. Thirdly, Diffmat v2 does not support SVG or Text nodes, as they are less commonly used in natural materials. Moving forward, we plan to provide regular updates and maintain active technical support for Diffmat v2 to refine its features and encourage its use in various downstream tasks. Finally, it will be beneficial to develop a versatile procedural material capture method for non-flat surfaces under random global illumination conditions. The wealth of recent research on differentiable rendering theory and software will potentially facilitate the development of such a system [Bangaru et al. 2020; Kato et al. 2020; Li et al. 2018a; Loubet et al. 2019; Nimier-David et al. 2019; Zhang et al. 2020; Zhao et al. 2020].

REFERENCES

- Substance 3D Designer Adobe. 2022a. Adobe. <https://substance3d.adobe.com/documentation/sddoc/substance-3d-designer-102400008.html>.
- Substance 3D Designer Function Nodes Overview Adobe. 2022b. Adobe. <https://substance3d.adobe.com/documentation/sddoc/function-nodes-overview-102400052.html>.
- Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2013. Practical SVBRDF capture in the frequency domain. *ACM Trans. Graph.* 32, 4 (July 2013), 1–12.
- Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2015. Two-shot SVBRDF capture for stationary materials. *ACM Trans. Graph.* 34, 4 (July 2015), 1–13.
- Ken-ichi Anjyo. 1988. A Simple Spectral Approach to Stochastic Modelling for Natural Objects. In *EG 1988-Technical Papers*. Eurographics Association. <https://doi.org/10.2312/egtp.19881023>
- Louis-Philippe Asselin, Denis Laurendeau, and Jean-Francois Lalonde. 2020. Deep SVBRDF Estimation on Real Materials.
- Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph.* 39, 6 (Nov. 2020), 1–18.
- Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan. 2013. Mixed-integer nonlinear optimization. *Acta Numerica* 22 (2013), 1–131.
- Robert L Cook and Tony DeRose. 2005. Wavelet noise. *ACM Trans. Graph.* 24, 3 (July 2005), 803–811.
- Kristin J Dana, Bram van Ginneken, Shree K Nayar, and Jan J Koenderink. 1999. Reflectance and texture of real-world surfaces. *ACM Trans. Graph.* 18, 1 (Jan. 1999), 1–34.
- Sebastien Deguy, Albert Benassi, and Université Blaise Pascal. 2001. A Flexible Noise Model For Designing Maps. <http://citeseerx.ist.psu.edu/viewdoc/summary?http://citeseerx.ist.psu.edu/viewdoc/summary> (2001).
- Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. 2018. Single-image SVBRDF capture with a rendering-aware deep network. *ACM Trans. Graph.* 37, 4 (Aug. 2018), 1–15.
- Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. 2019. Flexible SVBRDF capture with a multi-image deep network. *Comput. Graph. Forum* 38, 4 (July 2019), 1–13.
- J-M Dischler, K Maritaud, B Lévy, and D Ghazanfarpour. 2002. Texture particles. *Comput. Graph. Forum* 21, 3 (Sept. 2002), 401–410.
- Yue Dong, Guojun Chen, Pieter Peers, Jiawan Zhang, and Xin Tong. 2014. Appearance-from-motion: recovering spatially varying surface reflectance under unknown lighting. *ACM Trans. Graph.* 33, 6 (Nov. 2014), 1–12.
- Yue Dong, Jiaping Wang, Xin Tong, John Snyder, Yanxiang Lan, Moshe Ben-Ezra, and Baining Guo. 2010. Manifold bootstrapping for SVBRDF capture. In *ACM SIGGRAPH 2010 papers* (Los Angeles, California) (SIGGRAPH '10, Article 98). Association for Computing Machinery, New York, NY, USA, 1–10.
- David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2002. *Texturing and Modeling: A Procedural Approach* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Christian Eisenacher, Chuck Tappan, Brent Burley, Daniel Teece, and Arthur Shek. 2010. Example-based texture synthesis on Disney's Tangled. In *ACM SIGGRAPH 2010 Talks* (Los Angeles, California) (SIGGRAPH '10, Article 32). Association for Computing Machinery, New York, NY, USA, 1.
- Alain Fournier, Don Fussell, and Loren Carpenter. 1982. Computer rendering of stochastic models. *Commun. ACM* 25, 6 (June 1982), 371–384.
- Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor noise by example. *ACM Trans. Graph.* 31, 4 (July 2012), 1–9.
- B Galerne, A Leclaire, and L Moisan. 2017. Texton noise. *Comput. Graph. Forum* 36, 8 (Dec. 2017), 205–218.
- 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 (July 2019), 1–15.
- Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. 2003. Linear light source reflectometry. *ACM Trans. Graph.* 22, 3 (July 2003), 749–758.
- Geoffrey Y Gardner. 1984. Simulation of natural scenes using textured quadric surfaces. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 11–20.
- Leon Gatys, Alexander S Ecker, and Matthias Bethge. 2015. Texture Synthesis Using Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, C Cortes, N Lawrence, D Lee, M Sugiyama, and R Garnett (Eds.), Vol. 28. Curran Associates, Inc.
- Abhijeet Ghosh, Tongbo Chen, Pieter Peers, Cyrus A Wilson, and Paul Debevec. 2009. Estimating specular roughness and anisotropy from second order spherical gradient illumination. *Comput. Graph. Forum* 28, 4 (June 2009), 1161–1170.
- Abhijeet Ghosh, Tongbo Chen, Pieter Peers, Cyrus A Wilson, and Paul Debevec. 2010. Circularly polarized spherical illumination reflectometry. In *ACM SIGGRAPH Asia 2010 papers* (Seoul, South Korea) (SIGGRAPH ASIA '10, Article 162). Association for Computing Machinery, New York, NY, USA, 1–12.
- Abhijeet Ghosh, Tim Hawkins, Pieter Peers, Sune Frederiksen, and Paul Debevec. 2008. Practical modeling and acquisition of layered facial reflectance.
- Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. 2014. Local random-phase noise for procedural texturing. *ACM Trans. Graph.* 33, 6 (Nov. 2014), 1–11.
- Alexander Goldberg, Matthias Zwicker, and Frédo Durand. 2008. Anisotropic noise. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–8.
- Dan B Goldman, Brian Curless, Aaron Hertzmann, and Steven M Seitz. 2010. Shape and spatially-varying BRDFs from photometric stereo. *IEEE Trans. Pattern Anal. Mach. Intell.* 32, 6 (June 2010), 1060–1071.
- Paul Guerrero, Miloš Hašan, Kalyan Sunkavalli, Radomir Měch, Tamy Boubekeur, and Niloy J Mitra. 2022. MatFormer: a generative model for procedural materials. *ACM Trans. Graph.* 41, 4 (July 2022), 1–12.
- 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 (July 2021), 1–14.
- Yu Guo, Milos Hasan, Lingqi Yan, and Shuang Zhao. 2019. A Bayesian Inference Framework for Procedural Material Parameter Estimation. (Dec. 2019). arXiv:1912.01067 [cs.GR]
- Yu Guo, Cameron Smith, Miloš Hašan, Kalyan Sunkavalli, and Shuang Zhao. 2020. MaterialGAN: reflectance capture using a generative SVBRDF model. *ACM Trans. Graph.* 39, 6 (Nov. 2020), 1–13.
- Eric Heitz and Fabrice Neyret. 2018. High-Performance By-Example Noise using a Histogram-Preserving Blending Operator. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2 (Aug. 2018), 1–25.
- Philipp Henzler, Valentin Deschaintre, Niloy J Mitra, and Tobias Ritschel. 2021. Generative modelling of BRDF textures from flash images. *ACM Trans. Graph.* 40, 6 (Dec. 2021), 1–13.
- Michael Holroyd, Jason Lawrence, and Todd Zickler. 2010. A coaxial optical scanner for synchronous acquisition of 3D geometry and surface reflectance. *ACM Trans. Graph.* 29, 4 (July 2010), 1–12.
- Yiwei Hu, Julie Dorsey, and Holly Rushmeier. 2019a. A novel framework for inverse procedural texture modeling. *ACM Trans. Graph.* 38, 6 (Nov. 2019), 1–14.
- Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. 2022a. Node Graph Optimization Using Differentiable Proxies. In *ACM SIGGRAPH*

- 2022 *Conference Proceedings* (Vancouver, BC, Canada) (*SIGGRAPH '22, Article 5*). Association for Computing Machinery, New York, NY, USA, 1–9.
- Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. 2022b. An Inverse Procedural Modeling Pipeline for SVBRDF Maps. *ACM Trans. Graph.* 41, 2 (Jan. 2022), 1–17.
- Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019b. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.
- Zhuo Hui, Kalyan Sunkavalli, Joon-Young Lee, Sunil Hadap, Jian Wang, and Aswin C Sankaranarayanan. 2017. Reflectance Capture Using Univariate Sampling of BRDFs. Kaizhang Kang, Cihui Xie, Chengan He, Mingqi Yi, Minyi Gu, Zimin Chen, Kun Zhou, and Hongzhi Wu. 2019. Learning efficient illumination multiplexing for joint capture of reflectance and shape. *ACM Trans. Graph.* 38, 6 (Dec. 2019), 1–12.
- Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. Differentiable Rendering: A Survey. (June 2020). arXiv:2006.12057 [cs.CV]
- A Lagae, S Lefebvre, R Cook, T DeRose, G Drettakis, D S Ebert, J P Lewis, K Perlin, and M Zwicker. 2010a. A survey of procedural noise functions. *Comput. Graph. Forum* 29, 8 (Dec. 2010), 2579–2600.
- Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. 2009. Procedural noise using sparse Gabor convolution. In *ACM SIGGRAPH 2009 papers* (New Orleans, Louisiana) (*SIGGRAPH '09, Article 54*). Association for Computing Machinery, New York, NY, USA, 1–10.
- Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. 2010b. Procedural isotropic stochastic textures by example. *Comput. Graph.* 34, 4 (Aug. 2010), 312–321.
- Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. 2006. Inverse shade trees for non-parametric material representation and editing. *ACM Trans. Graph.* 25, 3 (July 2006), 735–745.
- Jon Lee and Sven Leyffer. 2011. *Mixed integer nonlinear programming*. Vol. 154. Springer Science & Business Media.
- L Lefebvre and Pierre Poulin. 2000. Analysis and Synthesis of Structural Textures. *Graphics Interface* (2000).
- Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. 2010. By-example synthesis of architectural textures. In *ACM SIGGRAPH 2010 papers* (Los Angeles, California) (*SIGGRAPH '10, Article 84*). Association for Computing Machinery, New York, NY, USA, 1–8.
- John-Peter Lewis. 1984. Texture synthesis for digital painting. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 245–252.
- J P Lewis. 1986. Methods for stochastic spectral synthesis. In *Proceedings on Graphics Interface '86/Vision Interface '86* (Vancouver, British Columbia, Canada). Canadian Information Processing Society, CAN, 173–179.
- J P Lewis. 1987. Generalized stochastic subdivision. *ACM Trans. Graph.* 6, 3 (July 1987), 167–190.
- J P Lewis. 1989. Algorithms for solid noise synthesis. *SIGGRAPH Comput. Graph.* 23, 3 (July 1989), 263–270.
- T M Li, M Aittala, F Durand, and J Lehtinen. 2018a. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.* (2018).
- 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 (July 2017), 1–11.
- Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. 2018b. Materials for masses: SVBRDF acquisition with a single mobile phone image. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 72–87.
- Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. 2018c. Learning to reconstruct shape and spatially-varying reflectance from a single image. *ACM Trans. Graph.* 37, 6 (Dec. 2018), 1–11.
- Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. 2019. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Trans. Graph.* 38, 6 (Nov. 2019), 1–14.
- Giljoon Nam, Joo Ho Lee, Diego Gutierrez, and Min H Kim. 2018. Practical SVBRDF acquisition of 3D objects with unstructured flash photography. *ACM Trans. Graph.* 37, 6 (Dec. 2018), 1–12.
- Giljoon Nam, Joo Ho Lee, Hongzhi Wu, Diego Gutierrez, and Min H Kim. 2016. Simultaneous acquisition of microscale reflectance and normals. *ACM Trans. Graph.* 35, 6 (Nov. 2016), 1–11.
- M Nimier-David, D Vicini, T Zeltner, and others. 2019. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on* (2019).
- M Olano, Hartj. C. W Heidrich, B Mark, and K Perlin. 2002. Real-time shading languages. *ACM SIGGRAPH 2002 Course* (2002).
- 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. *Adv. Neural Inf. Process. Syst.* 32 (2019).
- K Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* (1985).
- K Perlin. 2002. Improving noise. *Proceedings of the 29th annual conference on* (2002).
- Emil Praun, Adam Finkelstein, and Hugues Hoppe. 2000. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 465–470.
- Peiran Ren, Jiaping Wang, John Snyder, Xin Tong, and Baining Guo. 2011. Pocket reflectometry. *ACM Trans. Graph.* 30, 4 (July 2011), 1–10.
- J Riviere, P Peers, and A Ghosh. 2016. Mobile surface reflectometry. *Comput. Graph. Forum* 35, 1 (Feb. 2016), 191–202.
- Dietmar Saupe. 1988. Algorithms for random fractals. In *The Science of Fractal Images*, Michael F Barnsley, Robert L Devaney, Benoit B Mandelbrot, Heinz-Otto Peitgen, Dietmar Saupe, Richard F Voss, Heinz-Otto Peitgen, and Dietmar Saupe (Eds.). Springer New York, New York, NY, 71–136.
- 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 (Nov. 2020), 1–15.
- Borom Tunwattananpong, Graham Fyffe, Paul Graham, Jay Busch, Xueming Yu, Abhijeet Ghosh, and Paul Debevec. 2013. Acquiring reflectance and shape from continuous spherical harmonic illumination. *ACM Trans. Graph.* 32, 4 (July 2013), 1–12.
- Jarke J van Wijk. 1991. Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph.* 25, 4 (July 1991), 309–318.
- Richard F Voss. 1988. Fractals in nature: From characterization to simulation. In *The Science of Fractal Images*, Michael F Barnsley, Robert L Devaney, Benoit B Mandelbrot, Heinz-Otto Peitgen, Dietmar Saupe, Richard F Voss, Heinz-Otto Peitgen, and Dietmar Saupe (Eds.). Springer New York, New York, NY, 21–70.
- Jiaping Wang, Shuang Zhao, Xin Tong, John Snyder, and Baining Guo. 2008. Modeling anisotropic surface reflectance with example-based microfacet synthesis. *ACM Trans. Graph.* 27, 3 (Aug. 2008), 1–9.
- Hongzhi Wu, Zhaotian Wang, and Kun Zhou. 2016. Simultaneous Localization and Appearance Estimation with a Consumer RGB-D Camera. *IEEE Trans. Vis. Comput. Graph.* 22, 8 (Aug. 2016), 2012–2023.
- Geoff Wyvill and Kevin Novins. 1999. Filtered noise and the fourth dimension. In *ACM SIGGRAPH 99 Conference abstracts and applications* (Los Angeles, California, USA) (*SIGGRAPH '99*). Association for Computing Machinery, New York, NY, USA, 242.
- Zexiang Xu, Jannik Boll Nielsen, Jiyang Yu, Henrik Wann Jensen, and Ravi Ramamoorthi. 2016. Minimal BRDF sampling for two-shot near-field reflectance acquisition. *ACM Trans. Graph.* 35, 6 (Nov. 2016), 1–12.
- Wenjie Ye, Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. 2018. Single image surface appearance modeling with self-augmented CNNs and inexact supervision. *Comput. Graph. Forum* 37, 7 (Oct. 2018), 201–211.
- Yu-Ying Yeh, Zhengqin Li, Yannick Hold-Geoffroy, Rui Zhu, Zexiang Xu, Milos Hasan, Kalyan Sunkavalli, and Manmohan Chandraker. 2022. PhotoScene: Photorealistic Material and Lighting Transfer for Indoor Scenes. (June 2022).
- Jiyang Yu, Zexiang Xu, Matteo Mannino, Henrik Wann Jensen, and Ravi Ramamoorthi. 2016. Sparse sampling for image-based SVBRDF acquisition.
- C Zhang, B Miller, K Yan, I Gkioulekas, and others. 2020. Path-space differentiable rendering. *ACM transactions on* (2020).
- S Zhao, W Jakob, and T M Li. 2020. Physics-based differentiable rendering: from theory to implementation. *ACM SIGGRAPH 2020 Courses* (2020).
- Xilong Zhou and Nima Khademi Kalantari. 2021. Adversarial single-image SVBRDF estimation with hybrid training. *Comput. Graph. Forum* 40, 2 (May 2021), 315–325.
- Zhiming Zhou, Guojun Chen, Yue Dong, David Wipf, Yong Yu, John Snyder, and Xin Tong. 2016. Sparse-as-possible SVBRDF acquisition. *ACM Trans. Graph.* 35, 6 (Nov. 2016), 1–12.

A A REVIEW OF NON-PROCEDURAL INVERSE APPEARANCE CAPTURE

A large body of works for inverse appearance (SVBRDF) modeling uses non-procedural material representations. We summarize them based on the capture and reconstruction method.

Appearance Capture with Specialized Hardware. Physically accurate SVBRDF capture requires dense spatial and angular sampling to disambiguate material parameters. Achieving adequate sampling at large scales often necessitates a specialized captured environment that includes multiple lights and cameras. Many prior works have devised custom capture setups, including the use of a robot arm [Dana et al. 1999], matrix light pattern (LCD, DLP projector) [Aittala et al. 2013], spherical gantry [Holroyd et al. 2010; Lawrence et al. 2006; Yu et al. 2016], light dome [Ghosh et al. 2010, 2008; Nam et al. 2016], light cube [Kang et al. 2019], spherical harmonic illumination [Ghosh et al. 2009; Tunwattanapong et al. 2013], LED array [Asselin et al. 2020; Goldman et al. 2010; Wang et al. 2008], or moving linear light source [Gardner et al. 2003; Ren et al. 2011]. While these methods generally exhibit satisfactory reconstruction accuracy, the entailed hardware costs, human labor in environment setup, operation complexity, and acquisition time can be prohibitive for amateur users and casual purposes.

Prior-based Mobile Appearance Capture. The rapid advancement in cell phone imaging has ushered in a new era of mobile appearance capture solely based on casually captured flash photographs. Aittala et al. [2015] first demonstrated such a system using a pair of flash/no-flash photographs, while a contemporary work by Riviere et al. [2016] used a flash video captured over a hemisphere. However, the scarcity of captured views and lighting variations in these methods demands domain-specific priors to constrain the results within a plausible solution space. These priors may require the results to be a linear combination of certain basis materials [Hui et al. 2017; Nam et al. 2018; Ren et al. 2011; Wu et al. 2016; Xu et al. 2016; Zhou et al. 2016], exhibit self-similarity (containing stationary spatial structures) [Aittala et al. 2015], be illuminated under sparse incident lighting [Dong et al. 2014], or exist within a low-dimensional manifold [Dong et al. 2010]. Unfortunately, these assumptions do not always hold true for various material families, which often compromises the complexity and fidelity of the reconstructed SVBRDFs.

Learning-based Mobile Appearance Capture. The revolutionary knowledge distillation and generalization capabilities of convolutional neural networks (CNNs) make them an attractive alternative to hand-crafted priors. Li et al. [2017] trained a CNN-based SVBRDF predictor utilizing a small labeled dataset of measured SVBRDFs with self-augmentation. Ye et al. [2018] extended this method to enable training with unlabelled data. Later on, the introduction of differentiable rendering layers and large-scale synthetic SVBRDF datasets allowed for per-pixel albedo, normal, and roughness predictions [Deschaintre et al. 2018, 2019; Li et al. 2018b], as well as the joint recovery of 3D shape [Li et al. 2018c]. Alternatively, CNNs were trained to generate a low-dimensional latent space as a neural prior, enabling the optimization of more plausible SVBRDFs [Gao et al. 2019; Guo et al. 2020; Henzler et al. 2021]. Zhou et al. [2021]

employed adversarial training to enhance the realism of the reconstructed SVBRDFs. Guo et al. [2021] introduced highlight-aware convolution to handle saturated specular highlights.

B IMPLEMENTATION OF FX-MAP

The FX-Map graph system in Diffmat v2 follows the construction procedure in S3D, conducting a top-down depth-first traversal of the FX-Map graph under the control of Quadrant, Iterate, and Switch nodes. This process enumerates atomic patterns generated from Quadrant nodes and sequentially applies them to a background canvas under user-specified transformations and blending options.

Unlike S3D, Diffmat v2 does not have access to commercially available, high-performance pixel shaders. As such, the implementation strategy must carefully balance versatility and performance, prioritizing performance optimization while ensuring differentiability. Unfortunately, the computational cost of FX-Map nodes scales exponentially with the quadtree depth. In particular, commonly-used multi-frequency noise generator nodes can comprise nine or ten densely-chained quadrant nodes, totaling over a million atomic patterns. Generating and rendering these many atomic patterns sequentially is consequently impractical.

Our implementation classifies FX-Map nodes into *generic* and *specialized* nodes. Generic nodes are custom, user-defined FX-Maps that potentially enact any supported behavior. There is little room for parallel optimization since these nodes may specify a complicated random process that only allows for sequential execution, especially if Set/Sequence nodes are extensively used to introduce data dependency. Thus, we follow the two-phase sequential approach introduced in Sec. 3.1. In the pattern generation phase, the FX-Map graph is traversed depth-first with a manual stack to track the state of each node, e.g., how many child nodes have been visited. Pushing a node into the stack represents recursion to the next octave. Popping the top node from the stack is equivalent to returning to the octave above. We also update the cumulative branch offset (a displacement effective to all atomic patterns in the subtree) and a variable dictionary containing all named variables accessible via *Get* function nodes (for referencing external parameters) in the process. Each Quadrant node submits a pattern generation job to a global rendering job queue, where the atomic patterns are sorted by types and timestamps. In the rendering phase, we invoke the FX-Map renderer to batch individual pattern generation jobs into a job array. The renderer transforms these patterns in parallel and draws them onto respective regions of the canvas.

In contrast, most predefined noise/pattern generator nodes in S3D comprise specialized FX-Map nodes, such as Tile Generator, Splatter, and Make It Tile Patch nodes. We devise corresponding acceleration strategies from their prescribed implementations to maximize data parallelism. For example, the *Make It Tile Patch* node (abbreviated as MITP) makes an arbitrary input image tileable by stamping it around repetitively on an FX-Map canvas with deterministic or stochastic transformations. While a MITP node can generate thousands of atomic patterns, it only renders a single FX-Map octave, with several node parameters controlling randomness in pattern transformations. As there is no dependency among consecutive patterns, we compute all atomic patterns in parallel rather than using

Table 2. Optimization runtimes of production-grade material graphs. We show the time consumption per forward and backward iteration at various image resolutions. As a reference, Diffmat takes 1,659 ms to run forward evaluation of the Office Carpet material (the last row in the table) at a 512×512 resolution, and 1,853 ms for backward evaluation. These runtimes are 17× slower than Diffmat v2.

Graph Name (Position)	256×256		512×512		1024×1024	
	Forward	Backward	Forward	Backward	Forward	Backward
Brick Wall (Fig. 1, Row 1 Col 1)	105 ms	51 ms	108 ms	54 ms	117 ms	89 ms
Concrete Tiles (Fig. 1, Row 2 Col 1)	21 ms	13 ms	18 ms	20 ms	24 ms	60 ms
Aluminium Cut (Fig. 1, Row 3 Col 1)	28 ms	16 ms	28 ms	25 ms	41 ms	50 ms
Metal Grill (Fig. 1, Row 4 Col 1)	22 ms	14 ms	20 ms	25 ms	40 ms	168 ms
Amethyst (Fig. 1, Row 1 Col 2)	52 ms	40 ms	60 ms	52 ms	102 ms	129 ms
Marble (Fig. 1, Row 3 Col 2)	44 ms	15 ms	46 ms	18 ms	53 ms	68 ms
Bitten Cookie (Fig. 1, Row 4 Col 2)	150 ms	84 ms	150 ms	87 ms	188 ms	214 ms
Weave and Knit (Fig. 6, Row 1 Col 1)	160 ms	114 ms	199 ms	144 ms	412 ms	609 ms
Nature Ground (Fig. 6, Row 3 Col 2)	264 ms	186 ms	299 ms	235 ms	479 ms	758 ms
Office Carpet (Fig. 6, Row 4 Col 2)	68 ms	51 ms	95 ms	106 ms	227 ms	434 ms

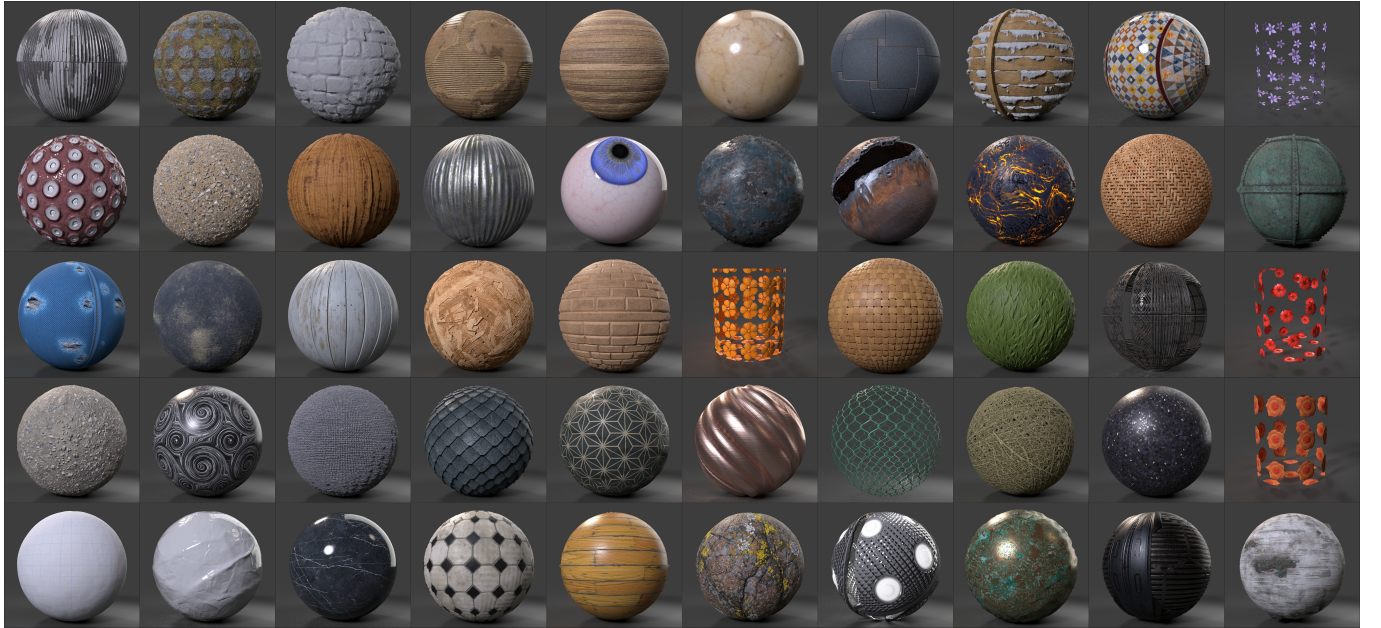


Fig. 10. A gallery of 50 differentiable procedural materials end-to-end translated from artist-authored material graphs in Adobe Substance 3D Designer using Diffmat v2. Most materials exhibit regular structures (e.g., bricks, tiles, waves, and polygons) or complex, stochastic patterns (e.g., foliage, flower, and irregular geometries). All images are rendered using SVBRDF texture maps generated by Diffmat v2 at a 1024×1024 resolution.

the job queue mechanism for generic FX-Maps. This eliminates the performance bottleneck of traversing the FX-Map quadtree. Our engineered implementation using PyTorch and Taichi [Hu et al. 2019b] achieves a **300× speedup** over Diffmat. This significant improvement accelerates material graph optimization considerably. The Office Carpet material, for instance, primarily uses MITP to render dense, carpet-like patterns. As shown in Appendix Table 2, Diffmat v2 runs **17× faster** than Diffmat in each forward and backward evaluation of the material graph. Additionally, many computation-heavy stochastic noise generators in S3D, like BnW Spots 3 and Clouds 2, only define an integer “scale” parameter and a continuous

“disorder” parameter that functions like a random seed. Since neither parameter is optimizable by gradient descent, these nodes do not require differentiable implementations. Diffmat v2 implements non-differentiable versions of these specialized FX-Map nodes using Taichi, leading to comparable or superior performance to S3D (Table 1).

C IMPLEMENTATION OF PIXEL PROCESSOR

As briefed in Sec. 3.2, Diffmat v2 implements full-fledged Pixel Processor nodes by compiling function graph representations in S3D into PyTorch programs. Specifically, the compiler automatically

translates each function node into a standard Python operation or a PyTorch routine using statically inferred operand data types (Fig. 3). If a function node has mismatched input data types (e.g., a tensor and a scalar/vector), the compiler performs explicit type casting so that the operands are compatible with the translated operation. Therefore, our function graph parser can translate Value Processors and Pixel Processors in a unified framework.

Besides the added functionality of per-pixel function translation, Diffmat v2 also introduces support for image sampling nodes and Set/Sequence nodes in function graphs. Sampling nodes are an essential part of Pixel Processor since they allow access to input images, without which Pixel Processor nodes can only be used as a per-pixel value generator and loses filtering capabilities. Furthermore, Set/Sequence nodes are frequently used to export variables between function graphs, similar to Python global variables. They are indispensable in FX-Map nodes for implementing sequential control like finite-state machines. Consequently, various structured patterns can be generated with a simple Iterate node, such as tiles, weaves, and circular splatters.