

Generating Procedural Materials from Text or Image Prompts

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

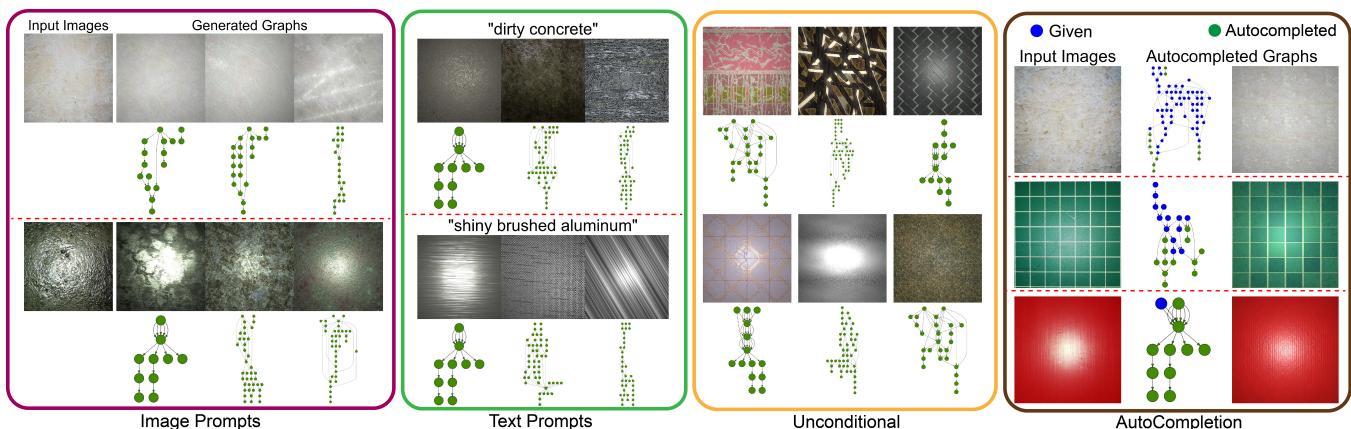


Figure 1: Procedural materials can be represented by directed computational graphs where each node represents a 2D image generator or filtering operator. The nodes are connected by unidirectional edges defining the computation flow. Our generative model can produce multiple procedural material graphs 1) from image prompts, 2) from text prompts, 3) unconditionally and 4) conditioned on partial graphs (AutoCompletion). Generated graph structures are shown in green, existing structures (in AutoCompletion) are in blue.

ABSTRACT

Node graph systems are used ubiquitously for material design in computer graphics. They allow the use of visual programming to achieve desired effects without writing code. As high-level design tools they provide convenience and flexibility, but mastering the creation of node graphs usually requires professional training. We propose an algorithm capable of generating multiple node graphs from different types of prompts, significantly lowering the bar for users to explore a specific design space. Previous work [Guerrero et al. 2022] was limited to unconditional generation of random node graphs, making the generation of an envisioned material challenging. We propose a multi-modal node graph generation neural architecture for high-quality procedural material synthesis which can be conditioned on different inputs (text or image prompts),

using a CLIP-based encoder. We also create a substantially augmented material graph dataset, key to improving the generation quality. Finally, we generate high-quality graph samples using a regularized sampling process and improve the matching quality by differentiable optimization for top-ranked samples. We compare our methods to CLIP-based database search baselines (which are themselves novel) and achieve superior or similar performance without requiring massive data storage. We further show that our model can produce a set of material graphs unconditionally, conditioned on images, text prompts or partial graphs, serving as a tool for automatic visual programming completion.

CCS CONCEPTS

• Computing methodologies → Rendering.

KEYWORDS

Node graphs, procedural materials, inverse modeling

ACM Reference Format:

Yiwei Hu, Paul Guerrero, Miloš Hašan, Holly Rushmeier, and Valentin Deschaintre. 2023. Generating Procedural Materials from Text or Image

Prompts. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 6–10, 2023, Los Angeles, CA, USA*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3588432.3591520>

1 INTRODUCTION

Node graph systems are widely adopted in the computer graphics industry, allowing artists to design various assets such as material or shader graphs. This kind of system provides a more user-friendly visual design interface than shading languages, while still offering expressive power. In this work we focus on the *material node graphs* which are commonly used in material authoring [Adobe 2023]. These graphs describe a set of material maps using a combination of noise and pattern generators and filters. Procedural materials have attractive properties like tileability, arbitrary resolution and convenient parametric editability. However, high-quality node graphs are difficult to author, requiring significant expertise and time from artists.

In recent years, procedural material fitting has seen significant progress, with parameter estimation and optimization methods [Hu et al. 2019, 2022a; Shi et al. 2020] and a segmentation-based graph fitting framework [Hu et al. 2022c]. These methods however rely on either a small fixed set of available graphs or a generic graph structure instantiated by user segmentation. Most recently, MatFormer [Guerrero et al. 2022] was proposed, enabling unconditional graph generation through multiple transformers [Vaswani et al. 2017]. While useful for random material exploration, it can not be guided by a specific target appearance. In this work, we propose a novel multi-modal conditional generative model for material graphs. The model is multi-modal in that it can work with no user input, a user provided image, a user provided text prompt, or a partial node graph as input. Our transformer model can produce a variety of graphs and enable automatic graph completion (Fig. 1).

In particular, we present a CLIP-based [Radford et al. 2021] encoder module to enable a multi-modal conditioning on either text or image prompts. Each of the transformer layers is conditioned by a CLIP embedding mapped by learnable MLPs. To train our conditional generative model we curate a new material graph dataset from Substance Source [Adobe 2023]. Given the limited available data, we augment the data and improve numerous details in the graph representation. This is particularly important not only for conditional generation to reproduce a target appearance, but also improves unconditional generation.

Further, we propose validation and regularization steps at inference time to ensure high-quality error-free graph generation. To account for the difference between the visual error and the parameter space error –on which our transformers are trained– and improve the image-space matching quality, we further improve the top-ranked generated graphs through differentiable optimization [Shi et al. 2020].

To evaluate the performance of our conditional model, we present two CLIP-search-based baselines. We show our conditional generative model achieves similar or better performance compared to retrieval from a large pre-generated database, but without the massive storage footprint. We present applications of our method for text-conditioned generation and conditional automatic graph completion, modes not supported by previous work in modeling.

In summary we present:

- A multi-modal conditional architecture for material graph generation.
- A graph dataset augmentation and cleanup strategy.
- A sampling regularization and post-sampling procedure to minimize the image/text-space distance with the generated graphs.

2 RELATED WORK

2.1 Program and Graph Generation in Graphics

Node graph systems are visual programming interfaces, making the generation of a node graph akin to program synthesis. In computer graphics, Hu et al. [2018] and Ganin et al. [2018] synthesized programs for interpretable image editing by predicting a sequence of image processing operations using reinforcement learning. Other work in program synthesis focused on 2D or 3D procedural shape generation. For example, Stava et al. [2010] proposed a framework to automatically generate L-systems while Demir et al. [2016] extracted a context-free parameterized split grammar for 3D shapes. Following the development of learning-based approaches, recent research [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] modeled priors over shape programs which can either be used for program generation or program induction from input shapes.

We build on the recent work of Guerrero et al. [2022] which presented a transformer-based unconditional generative model for material graphs. We improve its unconditional sampling to enable conditional generation. For multi-modal conditioning, we rely on the joint image/text embedding of CLIP [Radford et al. 2021] which has been shown to be effective in text-to-image [Ramesh et al. 2022; Rombach et al. 2022] and texture synthesis [Song 2022] applications.

2.2 Inverse Procedural Material Modeling

Our method is an inverse procedural material modeling approach when conditioned on an image. Procedural materials define materials or texture maps as a set of procedures [Guehl et al. 2020; Liu et al. 2016], which are typically organized as a node graph for interaction flexibility [Adobe 2023]. Given these existing procedures, different inverse methods [Guo et al. 2020a; Hu et al. 2019, 2022a; Shi et al. 2020] aim to recover parameters for a predefined procedural material model given a target image. More recently, Hu et al. [2022c] go beyond parameter regression, attempting to synthesize the structure of a node graph instead of relying on a fixed node graph fetched from a database. The method however still relies on a generic graph structure, specified by a user segmentation of the material. Our conditional generative model, on the other hand, can generate various node graph structures given different types of user input i.e., text, images or partial graphs.

Further, previous work in inverse modeling has focused on the production of a single node graph reproducing the details of a given material exemplar. Our model can generate multiple node graphs, following the prompt in terms of semantics, structures, and colors, rather than precisely matching details. The user can then select from the set of produced node graphs to continue to explore the material design space.

As we synthesize computational graphs, our method also differs from image-based inversion e.g., StyleGAN inversion [Guo et al. 2020b; Richardson et al. 2021; Tov et al. 2021]. Indeed, our results benefit from the advantages of procedural representation for controllability, tileability and arbitrary resolution.

2.3 Material Acquisition and Generation

We approach a problem similar to material acquisition that recovers material maps from images. However, material acquisition attempts to accurately estimate material properties in the form of texel values of material maps from one or more measurements (generally photographs). Classical material reconstruction [Guarnera et al. 2016] requires dense measurements, with tens to thousands of captured images to accurately digitize the material optical properties of a given target. Recent methods based on deep learning use a large amount of synthetic material training data to present various lightweight material acquisition frameworks, reducing the number of photos required to less than ten, with many requiring a single input flash photograph [Deschaintre et al. 2018, 2019; Gao et al. 2019; Guo et al. 2021, 2020b; Henzler et al. 2021; Li et al. 2017; Ye et al. 2021; Zhou et al. 2022; Zhou and Kalantari 2021].

As opposed to these methods, our goal is not to directly produce an array of texels representing the material. Rather, we generate programmable procedural node graphs. The generated node graphs can then generate material parameter maps at any resolution, with editable sub-components for convenient manipulation and variation.

3 CONDITIONAL MATERIAL GRAPH GENERATION

3.1 Overview

We present a generative model for procedural materials that can be conditioned on a given text prompt or image. Procedural materials are represented as node graphs, which are directed acyclic computation graphs (Fig. 1) that can be controlled through a set of parameters, and output a set of 2D material maps that define a material. A node graph consists of nodes that represent image operators, and edges that define the information flow between operators.

Our goal is to model the conditional distribution $p(g|y)$ of graphs g given inputs y , which may, for example, be text or image prompts. To model the probability distribution over node graphs, we follow MatFormer [Guerrero et al. 2022] and encode the graph as a set of sequences: a *node sequence* S_n , an *edge sequence* S_e and a *parameter sequence* S_p that are obtained by linearizing the graph. A probability distribution over each sequence can then be modeled using three transformer-based models [Vaswani et al. 2017], for nodes, edges, and parameters. We model correlations between the three sequences by conditioning edge generation on nodes and parameter generation on both nodes and edges. Unlike MatFormer, we also condition the generation of all three sequences on the inputs y , and make several architectural improvements to the generators.

We describe node graphs in Sec. 3.2 and give a more detailed description of our conditional generative models in Sec. 3.3. To train our conditional model, we present a new material graph dataset in Sec. 3.4, with careful preprocessing and extensive data augmentation, significantly extending the material space captured by our

model compared to MatFormer. We describe our training setup in Sec. 3.5 and discuss our regularized sampling as well as ranking-and-optimize step for image-space similarity improvement in Sec. 3.6.

3.2 Node Graphs

A material node graph $g = (N, E)$ consists of *nodes* $N = (n_1, n_2 \dots)$ and *edges* $E = (e_1, e_2, \dots)$. Nodes are instances of image operations from a predefined library. A node $n_i = (\tau_i, P_i)$ is defined by the operation type τ_i and a set of node parameters $P_i = (p_1^i, p_2^i, \dots)$ that control the operation. Individual node parameters p_j^i may have various types like integers, floats, strings, fixed-length arrays or variable-length arrays. The operation type also defines a set of input slots ($\text{in}_1^i, \text{in}_2^i, \dots$) and a set of output slots ($\text{out}_1^i, \text{out}_2^i, \dots$). Each input slot can receive one input image, and the set of all input images is transformed by the image operation into output images that are provided in the output slots. A node that has zero input slots is called a *generator node*. All nodes have at least one output slot, except special *output nodes* that define the final outputs of the graph.

Edges define the information flow in a node graph. They are unidirectional: an edge $e_i = (\text{out}_l^j, \text{in}_h^k)$ always connects an output slot of one node to an input slot of a different node. An output slot can provide images to multiple input slots of other nodes, but an input slot can only receive an image from a single output slot. Additionally, no cycles are allowed in a node graph. In the following, we denote the output slot that edge e_i starts from as e_i^{out} and the input slot that the edge ends in as e_i^{in} .

A graph is evaluated by running the operations defined by each node in a topological order. After evaluation, the output nodes of the graph provide a set of 2D material maps describing spatially varying material parameters such as diffuse color, roughness, height, or normals.

3.3 Conditional Generative Model

Our generative model for node graphs is based on MatFormer, which we improve to allow for generation *conditioned* on images or text prompts. We generate node graphs in three steps, corresponding to nodes, edges, and node parameters. For each step, we train a conditional transformer that models the probability distribution over node, edge, or parameter sequences, respectively, conditioned on the input prompt y . A transformer-based conditional generative model with parameters ξ models the conditional probability distribution of a sequence S conditioned on y as a product of conditional probabilities for each token s_i :

$$p_\xi(S|y) := \prod_i p_\xi(s_i|s_{<i}, y), \quad (1)$$

where $s_{<i} := s_1, \dots, s_{i-1}$ denotes a partial token sequence generated up to token s_i . The transformer outputs the probability distribution $p_\xi(s_i|s_{<i}, y)$ in each step, which can be sampled to obtain the next token s_i . The dependence between the sequences that represent a graph g is modeled using conditional probabilities:

$$p(g|y) := p_\theta(S_n|y) p_\phi(S_e|S_n, y) p_\psi(S_p|S_e, S_n, y), \quad (2)$$

where θ , ϕ , and ψ are parameters of the transformer models for node sequences S_n , edge sequences S_e , and parameter sequences S_p , respectively. Unlike MatFormer, our generative model accepts

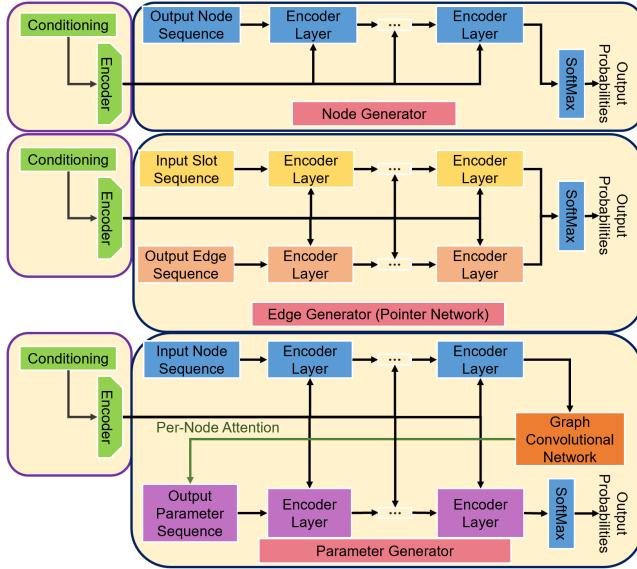


Figure 2: The architecture of our conditional generative model. We show a single next token generation step. The “Output Node/Edge/Parameter Sequence” blocks are the sequences generated so far. As an autoregressive model, the output sequences serve as inputs for next token prediction. Given a text or image prompt as conditioning, the encoder encodes it to the dimension of the hidden states of our transformer encoders. For the encoder, we use CLIP embedding with a learnable MLP. The encoded feature vector is then fused into each transformer encoder layer. In the parameter generator, the Graph Convolutional Network captures the edge connectivity of neighbor nodes and the transformed node embedding is attached as an auxiliary input sequence for the transformer to pay attention to the node that it is generating parameters for.

a condition y that guides the sampling process of the transformer. The condition y is given as a feature vector. We will describe our approach to obtain this feature vector from images or text prompts, and our transformer conditioning strategy in Sec. 3.3.1. In a further departure from MatFormer, we generate all parameters in a graph as a single sequence, instead of one sequence per node, and condition the parameter probabilities $p_\psi(s_p^i | S_e, S_n, y)$ on the edge sequence S_e in addition to the node sequence S_n (Sec. 3.3.3).

3.3.1 Conditioning on Text Prompts or Images. We consider multi-modal inputs including text prompts and images. Radford et al. (CLIP) [2021] proposed a jointly learned space for encoding both text prompts and images. To benefit from this joint encoding, we use a frozen CLIP model (ViT-L/14) to encode our inputs. As we will introduce in Sec. 3.4, we train our conditional models using image-to-graph correspondences. During training, our network is only conditioned on image space CLIP embeddings which are not exactly the same as CLIP text space embeddings [Ramesh et al. 2022]. This limits the quality of the text-conditioned generated material graphs. We therefore follow the approach of DALL-E2 and apply a prior (a specialized network) to transform CLIP text embeddings into

CLIP texture embeddings. This model was trained using 10 millions of text/texture pairs [Aggarwal et al. 2023], transforming text embedding into texture image embeddings. When providing a text prompt, we encode it as a CLIP text embedding and then transform it to a CLIP texture image embedding, compatible with our training data. The CLIP embedding is then encoded by a trainable MLP to map it to the dimension of the hidden states of our transformer encoders. The mapped embedding is then added to the output of each transformer encoder layer, after layer normalization. We show in Fig. 2 the design of our conditional generative model.

Conditioning on CLIP embedding allows our model to accept both image and text prompts as inputs. Other encoding methods are possible. We experimented with an alternative image encoding approach. While CLIP captures the high-level semantic information in an image, to encode low-level texture statistics we add VGG features statistics to capture fine-scale texture detail and a 16x16 downsampled thumbnail to summarize the main color of the input image. Detailed implementation of this encoding is presented in the supplementary material. We discuss the performance of this alternative (**Ours (VGG)**) in Sec. 5. Note that the construction of this encoding is however slightly more complicated and prevents the use of text encoding. With the exception of the comparison in Fig. 7, the image results we show in the paper are generated by our CLIP-only encoding (**Ours**).

3.3.2 Node and Edge Generation. Node and edge generation closely follows the approach proposed by MatFormer, except for the added conditioning on y .

To generate a *node sequence* $S_n = (\tau_0, \tau_1, \dots)$, we iteratively sample the node model $p_\theta(s_i^n | s_{<i}^n, y)$, where s_i^n denotes element i of sequence S_n . Each step generates the integer ID τ_i of a node.

To generate an *edge sequence* $S_e = (e_1^{\text{out}}, e_1^{\text{in}}, e_2^{\text{out}}, e_2^{\text{in}}, \dots)$, we iteratively sample the edge model $p_\phi(s_i^e | s_{<i}^e, S_n)$. Each step generates a pointer into the list of all output and input slots in the graph. Pointers are generated using a transformer with a head based on Pointer Networks [Vinyals et al. 2015]. The list of all output and input slots is derived from the node sequence S_n generated in the previous step and includes information about the operation type of the node each slot is attached to.

As usual for transformers, all sequences are extended with auxiliary tokens that mark the start and end of a sequence. Additionally, all transformer models receive auxiliary input sequences that provide information such as the index of a token in the sequence. We use the same auxiliary sequences as MatFormer, please refer to the supplementary material for details.

3.3.3 Parameter Generation. For parameter generation, we take a different approach than MatFormer. First, we condition on both the node and edge sequences S_n and S_e instead of only on the node sequence S_n . This allows us to capture the relationship between parameters and edges in the graph; these were independent in MatFormer. Second, instead of generating one parameter sequence per node, we generate all parameters of a graph in a single sequence.

To generate the *parameter sequence* $S_p = (p_1^1, p_1^2, \dots, p_2^1, p_2^2, \dots)$ of a graph, we iteratively sample the model $p_\psi(s_j^p | s_{<j}^p, S_n, S_e)$. Each step outputs a parameter, or one scalar element of a parameter in the case of vector- or array-valued parameters. Conditioning on

the node and edge sequences S_n and S_e is implemented through a context-aware embedding $\bar{\tau}_i$ of each node n_i . These node embeddings are given as an auxiliary input sequence, where each parameter token receives the embedding of the node it is being generated for (i.e., per-node attention in Fig. 2).

The embedding $\bar{\tau}_i$ includes information about both the operation type τ_i of the node and the full node sequence S_n . It is computed with a transformer encoder $\bar{\tau}'_i = g^P(S_n, i)$. We additionally include edge connectivity information from the sequence S_e in the node embedding. We use a graph convolution network (GCN) to capture the edge connectivity in the neighborhood of the node: given a node embedding $\bar{\tau}'_i$, we use a GCN h with a residual connection to capture the local edge connectivity information:

$$\bar{\tau}_i = \bar{\tau}'_i + h(S_n, S_e, i). \quad (3)$$

We use 6 layers in our GCN, allowing us to capture the edge structure up to 6 edges away from the node n_i .

The parameter sequence is extended with auxiliary start and end tokens, and with special tokens that mark the start of the parameter sub-sequences for each node. Apart from the auxiliary sequence of node embeddings $\bar{\tau}_i$, we use the same auxiliary input sequences as MatFormer, providing information such as the index and type of the parameter being generated. We also compare our graph-conditioned parameter generator with an image/text conditioned extension of the MatFormer parameter generator. We find a small performance improvement and training speed acceleration (1.5x) due to higher level of parallelism. See our supplementary material for a discussion.

3.4 Material Graph Dataset

The success of sequence generative models such as GPT3 [Brown et al. 2020] relies largely on the size of their training data. However, the data available for procedural material graphs is limited. To generate a dataset which can be used to train our conditional model, we rely on Substance Source [Adobe 2023]. However, the available material graphs were created by different material artists over multiple years, with different methodology and skills, making the data inconsistent. We therefore carefully preprocess the dataset, and perform a series of data augmentations.

3.4.1 Graph Reformatting and Simplification. Since the raw dataset contains material graphs created in different system versions, our first step is to ensure that all material graphs are properly formatted and have uniform material representation e.g., fixing precision and dependency problems (See supplementary material for a checklist).

Second, a material graph can generate different material parameter maps for different rendering workflows and can be designed to generate additional material maps such as ambient occlusion (AO) maps, anisotropic maps etc. Such parameter maps are useful for artists but make the conditional generative task particularly difficult, as not all graphs contain these outputs and they tend to increase the length of the sequence to generate. Hence, we focus on the standard physically-based-rendering (PBR) workflow, assuming a GGX microfacet [Walter et al. 2007] shading model, and generate node graphs that produce albedo, normal, roughness and metallic maps. We therefore prune the branches of the unused maps from the graph where they appear, in a back-to-front way. A node is

discarded when it is in the computation branch of a unused node, and not in the computation of a used node. This simplification step significantly reduces the size of graphs and lets our model focus on the most important sequences. To avoid the generation of biased normals sometimes observed in MatFormer, we also prune the Level nodes (the nodes that adjust the histogram of the input image) in the normal branch.

3.4.2 Graph Splitting and Filtering. In material graph design systems, switch nodes are a type of node that activates one of the multiple alternative computation branches passing this node. The branch being activated is controlled by a integer parameter specified by users. This essentially packs multiple functionalities into a single graph and is extremely difficult for our model to learn. A graph that contains switch nodes can be split into multiple smaller graphs by creating a version of the graph for each branch, further reducing its complexity. If multiple switch nodes are present, a large number of combinatorial options may exist. We set a sampling upperbound to 5. If the maximum number of branches is k_b among all the switch nodes, we sample at least $\max(k_b, 5)$ to ensure each branch is at least sampled once. Finally, we remove graphs which are too similar to others, to prevent duplicates (i.e. the average mean square difference in the material maps they generate is smaller than 0.01).

Considering the difficulty of predicting overly long sequences, we cut the graphs belonging to the tail of the length distribution. We filter node graphs which contain a large number of nodes (> 80) or edges (> 200) or slots (> 210). This graph reduction step ensures our network focuses on statistically well represented graph structures. Our filtered dataset includes 4667 (72% of our unfiltered dataset) valid graph instances, compared against the 2820 unfiltered instances in MatFormer. This filtering is particularly important to our conditional model which has to match a specific desired appearance.

3.4.3 Parameter Augmentation. The parameters of each node are crucial to control the behavior of a given graph. To increase the number of variations of materials available during the training of our generator, we sample 100 sets of parameters for each of the 4667 graphs of our dataset.

However, randomly sampling the full parameter space of a procedural model will generate incorrect parameter combinations, producing purely white or black images or material maps which are not useful to artists [Hu et al. 2022a]. We compute the statistical distribution on parameters from the Substance Source dataset to guide our sampling process. Specifically, we sample a parameter p in a material graph g based on a Gaussian distribution $G(\mu_p^g, \beta\sigma_p)$. The mean of this Gaussian is the default value μ_p^g defined in the graph g 's preset and the standard deviation is scaled by a factor β from p 's standard derivation among the whole dataset σ_p . Some parameters' statistics are not reliable due to limited observations in the dataset. In such case, we use an uniform distribution based on a scaled range i.e., $U((1 - \alpha)\mu_p^g, (1 + \alpha)\mu_p^g)$ by a scaling factor of α . The α and β are empirically selected to achieve a balance between fidelity (i.e., the sampled graphs still look like a material) and diversity. α/β are set to 0.06/0.2 when sampling float types and 0.06/0.5 for integer types.

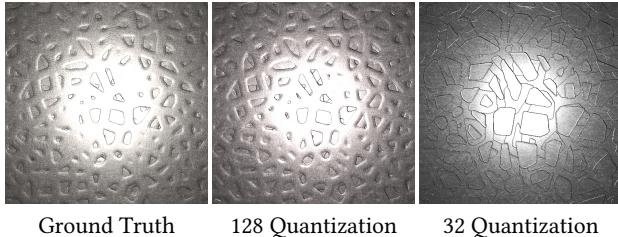


Figure 3: Using 32 bins for parameter quantization results in significant reconstruction error. A higher number of quantization bin (128 here) allow to better reconstruct the original appearance.

3.4.4 Final Dataset. Our final improved dataset contains 466,700 cleaned graphs that are more amenable to generation, paired with their associated output material maps. We render these material maps on a planar surface with a point light collocated with the camera, to synthesize an image-graph pair for training and evaluation. (We could use other lighting configurations like environment mapping, since CLIP encoding is fairly insensitive to precise lighting.) For parameter generation, the probability is modeled over quantized values. We use 128 quantization bins (compared to the 32 used by MatFormer). While more quantization bins are more challenging to learn, the result shows significantly better reconstruction quality as displayed in Fig. 3. This is particularly important for our model to match the user-provided condition.

3.5 Training

We train our conditional generative model with our new dataset. We split the dataset into a training set and a validation set before parameter augmentation, to ensure that the training set and validation set contain graphs with different topologies, not just different parameter settings.

All three models (node transformer, edge transfer and parameter transformer) are trained with the ground truth graphs as supervision, using a binary cross-entropy loss over the probabilities estimated by the transformer generators. Each transformer is trained separately using teacher forcing: that is, when generating a new token in a sequence, the ground-truth sequence is used as previously generated tokens. As we have a limited number of different graph structures in the dataset, we ensure we do not over-fit by keeping the checkpoint with the minimum validation loss.

3.6 Sampling, Ranking and Optimization

3.6.1 Sampling. Since our generative models consist of three transformers operating in succession, errors from previously generated sequences can propagate and affect the quality of sequences dependent on them. For this reason, we carefully decode the sampled sequences using semantic validity checks to ensure error-free generation.

While MatFormer ensures semantic validity as a post-processing step, once the entire sequence has been generated, we perform the validity checks during sampling, making sure to choose only among semantically valid choices for any given token. This makes sure that the chosen tokens are consistent with the semantically valid

Table 1: Statistical results expressed as style loss. Unoptimized: searched or our predicted results. Optimized: Differentiable optimized results using style loss. We searched/predicted 30 samples and optimized 10 of them. When computing statistics, we count for the top-5 samples after ranking (by style loss) in order to remove outliers. The style loss we used here as metric is the L1 difference of Gram Matrices of VGG features plus L1 difference of 16x16 down-sampled thumbnails (weighed by 0.1). We report Best-of-5: the minimum loss among the top-5 samples and Average-of-5: the average loss among the top-5 samples. Lower is better. Our model can achieve statistically better or similar performance comparable to search in a giant pre-generated image database. We also include additional quantitative comparisons to a class-conditioned generator which has a loss of 0.0544 (Best-of-5) and 0.0774 (Avg-of-5) before optimization. See our supplementary material for details.

Unoptimized	Ours	Ours (VGG)	Ours Uncond	Dataset
Best-of-5	0.0314	0.0300	0.0382	0.0384
Avg-of-5	0.0346	0.0329	0.0539	0.0499
Optimized	Ours	Ours (VGG)	Ours Uncond	Dataset
Best-of-5	0.0218	0.0199	0.0194	0.0191
Avg-of-5	0.0242	0.0221	0.0227	0.0237

choices for the previously generated tokens. See our supplementary material for the detailed validation rules we apply.

After node and edge generation, an uninitialized graph is ready for parameter prediction (Fig. 2). The graph structure serves as a condition to the parameter sequence. Considering possible prediction errors, for robust parameter generation, we regularize the generated graph by removing unconnected nodes.

3.6.2 Ranking and Optimization. During inference, we sample each token according to the probability distribution predicted by the transformer until reaching the end tokens. However, the quality of generated graphs is hard to evaluate. The cross-entropy loss of an estimated probability for a token does not directly reflect the true appearance distance to the input, since distance in the parameter space does not necessarily reflect distance in the image/text space. We would like to measure the discrepancy between the sampled graphs and the input prompts in the same space. We therefore deserialize the predicted token sequences to reconstruct a material graph. Evaluating this material graph generates material maps we render. We can then use a CLIP cosine metric to measure the difference between our graph’s results and the input text/images (alternatively, a sliced Wasserstein distance [Heitz et al. 2021; Hu et al. 2022b] can be used to measure the statistical distance to the input images). A differentiable optimization step [Hu et al. 2022a; Shi et al. 2020] can be further applied to refine the accuracy of predicted parameters to better match the given inputs. We generate multiple graph samples and rank them based on the text/image space metrics (CLIP cosine or sliced Wasserstein) and pick the top-k for further optimization (k=5 in our experiments). As node graphs are not all differentiable, we limit the refinement to the optimizable nodes only. Despite differentiable optimization limitation to adjust parameters only – it cannot modify the generated graph topology –

this step helps reduce the prediction appearance error, especially in terms of color and roughness values as shown in Fig. 5. For each image or text prompt, multiple graphs can be synthesized. The inference takes around 1.32 seconds *per graph*. If performing optimization, it takes 2 ~ 3 minutes *per graph*, measured on a NVIDIA RTX 3090.

4 IMPLEMENTATION DETAILS

Our transformer-based generative models are built upon the GPT [Brown et al. 2020] architecture. We select slightly different hidden dimensions, layers, and heads for each transformer to adjust for the size of our dataset (40/2/4 for node generator; 48/2/4 for edge generator; 96/4/4 for parameter generator). Regarding node order, we follow MatFormer to encode the node sequences in a back-to-front breadth-first traversal order π_r . For the autocompletion task, we use reversed π_r , i.e. from last to first node of π_r .

We train our models with the Adam optimizer [Kingma and Ba 2015], using a learning rate of $1e^{-5}$ and a batch size of 64. We train each transformer model in parallel on three NVIDIA V100 GPUs. The total training time for each transformer is ~22 hours for node generator; ~28 hours for edge generator and ~36 hours for parameter generator. During the sampling phase, we apply nucleus filtering with a top-p value of 0.9 and sample the top-5 candidates without adjusting the softmax temperature.

5 RESULTS

We show our conditional generative model can generate material graphs for text or image prompts. We also show improved variety for unconditional generation and demonstrate automatic conditional graph completion. Over 90% of generated graphs are valid. Not all generated graphs may fit the desired semantics, but generation is very fast, letting us quickly sample multiple graphs and automatically select the top-ranked ones.

To quantitatively evaluate the performance of our model, we compare it to two novel, challenging, CLIP-searched-based baselines:

- Searching and optimizing in a database of 102,400 pre-sampled material graphs generated using the unconditional version of our model, trained on our new augmented dataset. (**Ours Uncond**)
- Searching and optimizing directly in our new augmented dataset containing 466,700 graph variations. (**Dataset**)

We create a test image dataset containing 48 real photos. Given an input photo, we retrieve the 30 closest samples in the different databases in CLIP space and optimize (using the MATch framework [Shi et al. 2020]) the top-5 samples. We additionally optimize the top-5 (out of the 30 retrieved) samples with lowest visual statistical difference (sliced Wasserstein distance [Heitz et al. 2021]), for a total of 10 optimized samples. For our method, we first conditionally generate 150 graphs and perform the same search-and-optimization approach. Our conditional generative model achieves better or similar performance as shown in Table 1. We note that our proposed alternative conditional encoding adding VGG features (**Ours (VGG)**) produces slightly lower numerical error, but does not allow for text encoding. Visually, our results, our alternative encoding, and baselines are close, as shown in Fig. 7 (more visual

comparisons and detailed statistical analysis with precise definition of style metrics can be found in the supplemental material). However, the storage required of our model is orders of magnitude smaller than the pre-generated database: only 15 MB against 115 GB. Further, our model enables additional applications such as autocompletion (Fig. 8).

Our method can also be considered as an inverse procedural modeling framework if using images as input. We therefore show a comparison with the recent inverse material material modeling framework by Hu et al. [2022c] in Fig. 10. Compared to their method, our model doesn't require user segmentation and can produce multiple variations rather than a single output graph. More importantly, the gamut of the materials we handle is larger than this previous work, as can be seen in the failure of the previous work for the wood sample in the first row of Fig. 10.

5.1 Multi-modal Conditioning

We now evaluate our generated procedural material graphs qualitatively. **For all conditions, we show additional results in Fig. 1 and the supplementary materials.**

5.1.1 Image Conditioning. We now show conditionally generated graphs on randomly sampled synthetic images from the test set, where inputs are rendered images with co-located point light. Our generated samples are structurally and semantically close to the input image. We however observe small color mismatch due to prediction errors. Indeed, the color is controlled by very few parameters, which have very strong visual importance, making the inference of the exact color challenging. We use image-space differentiable optimization [Shi et al. 2020] to address this issue (Sec. 3.6.2).

Real photographs are more challenging to match as they can only be approximated by procedural models in most cases. In Fig. 5, we show three generated material graphs before and after optimization when given real photograph inputs. Despite the challenges, our sampled procedural graphs can reproduce the target appearance well, matching its semantics and style. Given our generated graphs, the optimization step is once more capable of adjusting the color and roughness values. We also show a comparison to a class-conditioned generation in supplemental material, illustrating the benefits of our CLIP-based conditional generation.

5.1.2 Text Conditioning. As described in Sec. 3.3, we propose to encode the given text prompt using CLIP and transform the embedding to the CLIP image embedding. We show three generated material graphs for each text prompt in Fig. 6. We can see that our model can generate diverse, semantically close material graphs.

5.1.3 Unconditional Generation. Our augmented dataset and regularized sampling process benefits unconditional material graph generation as well. We present a side-by-side comparison in our supplemental material, showing more diverse samples we generated, compared to MatFormer. We also use this unconditional version to generate the database containing 102,400 as our baseline comparison (**Ours Uncond** in Table 1 and Fig. 7)

5.1.4 Autocompletion: Conditioning on Partial Graphs. As a sequence-based model, our model can be conditioned on partial graphs inputs

and automatically predict the rest of the graph structures and parameters towards a conditional input. While MatFormer is also capable of autocompletion, it cannot be conditioned on a particular desired appearance. For example, in Fig. 8, we show examples of automatic visual programming, by generating the end of a partial graph (marked as green nodes and edges) to match target images. This application provides interesting graph modeling exploration possibilities for artists.

5.2 Failure Cases and Limitations

Despite the conditional results we show, limitations remain. The biggest limitation is in the amount of data available. Although we perform data processing and augmentation to build a material dataset $2.32x$ (before filtering) or $1.65x$ (after filtering) larger than previous state-of-the-art [Guerrero et al. 2022], the scale of our dataset is still far smaller than that of the dataset used for training large language or generative models ([Brown et al. 2020; Ramesh et al. 2022; Rombach et al. 2022]). Augmenting the number of base graphs is not trivial, as each is manually crafted by an artist. Our generative model is therefore limited to the subset of appearance and procedural material graphs which artists found useful to design.

In Figs. 9 and 11, we show failures cases of our generative model where the generated material graphs do not match well the input prompts for images/texts. While the overall structure is correct, the details do not match. Further, our model currently only supports standard PBR workflow material maps (Base Color, Normal, Roughness, Metalness) as they are the most represented in the dataset. Finally, the quantization step for token prediction is a limitation. While we choose a finer quantization than previous work, quantization errors still happen, and predicting a continuous field rather than discrete bins would be an interesting future work.

6 CONCLUSION AND FUTURE WORK

We present the first conditional generative model for procedural material node graph generation. We show that our model generates high-quality node graphs given either images or text prompts. The proposed generative model is a new tool for users to explore the design space of materials and has interesting applications such as automatic visual programming (Sec. 5.1.4).

As the dataset is a crucial component, an interesting future work would be to expand it. But more interestingly, the quality of the graphs could be improved. We take a first step in this direction with our cleanup step, but beyond cleanup, the existing material graphs designed by artists do not follow standard programming design principles. In particular, they are not necessarily intended to be easily modified or reused. An interesting future step would be to find reuseable sub-graphs i.e., sub-functions and expand the dataset, using a set of hierarchically organized modules or libraries. This would both reduce the sequence length and create more structure.

Finally, there is a gap between predicted token space error and image/text space matching error. We currently minimize this gap using an image/text-space optimization as a post-processing step, but further improvement would require finding a way to evaluate and minimize the image/text space error during training and inference.

ACKNOWLEDGMENTS

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

REFERENCES

- Adobe. 2023. Substance Designer. <https://www.substance3d.com/>.
- Pranav Aggarwal, Hareesh Ravi, Naveen Marri, Sachin Kelkar, Fengbin Chen, Vinh Khuc, Midhun Harikumar, Ritiz Tambi, Sudharshan Reddy Kakumanu, Purvak Lapsiya, Alvin Ghouas, Sarah Saber, Malavika Ramprasad, Baldo Faieta, and Ajinkya Kale. 2023. Controlled and Conditional Text to Image Generation with Diffusion Prior. arXiv:2302.11710 [cs.CV]
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- 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, Frédéric 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, Frédéric 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.
- 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/678076842014c83cedadbe6b0ba0314-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.
- 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>
- Paul Guerrero, Milos Hasan, Kalyan Sunkavalli, Radomir Mech, Tamy Boubekeur, and Niloy Mitra. 2022. MatFormer: A Generative Model for Procedural Materials. *ACM Trans. Graph.* 41, 4, Article 46 (2022). <https://doi.org/10.1145/3528223.3530173>
- 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.
- Yu Guo, Milos 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, Milos 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.
- Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. 2021. A Sliced Wasserstein Loss for Neural Texture Synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- 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.
- 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). Article 5, 9 pages.

- Yiwei Hu, Miloš Hašan, Paul Guerrero, Holly Rushmeier, and Valentin Deschaintre. 2022b. Controlling Material Appearance by Examples. *Computer Graphics Forum* 41, 4 (2022), 117–128. <https://doi.org/10.1111/cgf.14591>
- Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. 2022c. An Inverse Procedural Modeling Pipeline for SVBRDF Maps. *ACM Trans. Graph.* 41, 2 (2022), 1–17.
- Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. 2018. Exposure: A White-Box Photo Post-Processing Framework. *ACM Trans. Graph.* 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 Trans. Graph.* 39, 6 (2020), Article 234.
- Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. 2020. UCSG-Net – Unsupervised Discovering of Constructive Solid Geometry Tree. [arXiv:2006.09102](https://arxiv.org/abs/2006.09102) (2020).
- 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](https://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.
- Sidi Lu, Jiayuan Mao, Joshua Tenenbaum, and Jiajun Wu. 2019. Neurally-guided structure inference. In *International Conference on Machine Learning*. PMLR, 4144–4153.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*. PMLR, 8748–8763.
- Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. [arXiv preprint arXiv:2204.06125](https://arxiv.org/abs/2204.06125) (2022).
- Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. 2021. Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://github.com/CompVis/latent-diffusion><https://arxiv.org/abs/2112.10752>
- 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, Article 196 (Dec. 2020), 15 pages.
- Yiren Song. 2022. CLIPTexture: Text-Driven Texture Synthesis. In *Proceedings of the 30th ACM International Conference on Multimedia (Lisboa, Portugal) (MM '22)*. Association for Computing Machinery, New York, NY, USA, 5468–5476. <https://doi.org/10.1145/3503161.3548146>
- Ondřej Štava, Bedřich 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*.
- Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. 2021. Designing an Encoder for StyleGAN Image Manipulation. *ACM Trans. Graph.* 40, 4, Article 133 (jul 2021), 14 pages. <https://doi.org/10.1145/3450626.3459838>
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fb053c1c4a845aa-Paper.pdf>
- Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in neural information processing systems* 28 (2015).
- Homer Walke, R Kenny Jones, and Daniel Ritchie. 2020. Learning to infer shape programs using latent execution self training. [arXiv:2011.13045](https://arxiv.org/abs/2011.13045) (2020).
- Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet Models for Refraction Through Rough Surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques* (Grenoble, France). 195–206.
- Chenming Wu, Haisen Zhao, Chandrakana Nandi, Jeffrey I Lipton, Zachary Tatlock, and Adriana Schulz. 2019. Carpentry compiler. *ACM Trans. Graph.* 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>
- Xilong Zhou, Milos Hasan, Valentin Deschaintre, Paul Guerrero, Kalyan Sunkavalli, and Nima Khademi Kalantari. 2022. TileGen: Tileable, Controllable Material Generation and Capture. *CoRR* abs/2206.05649 (2022). [https://doi.org/10.48550/arXiv.2206.05649 arXiv:2206.05649](https://doi.org/10.48550/arXiv.2206.05649)
- Xilong Zhou and Nima Khademi Kalantari. 2021. Adversarial Single-Image SVBRDF Estimation with Hybrid Training. *Computer Graphics Forum* 40, 2 (2021), 315–325.

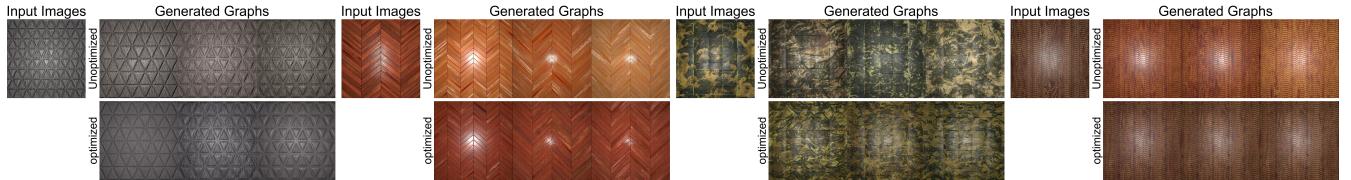


Figure 4: Synthetic Image Conditioning. We show three generated procedural material for each input. For each example, the top row shows non-optimized outputs directly predicted by our models, while the bottom row shows our final output after optimization, which easily rectifies the colors.

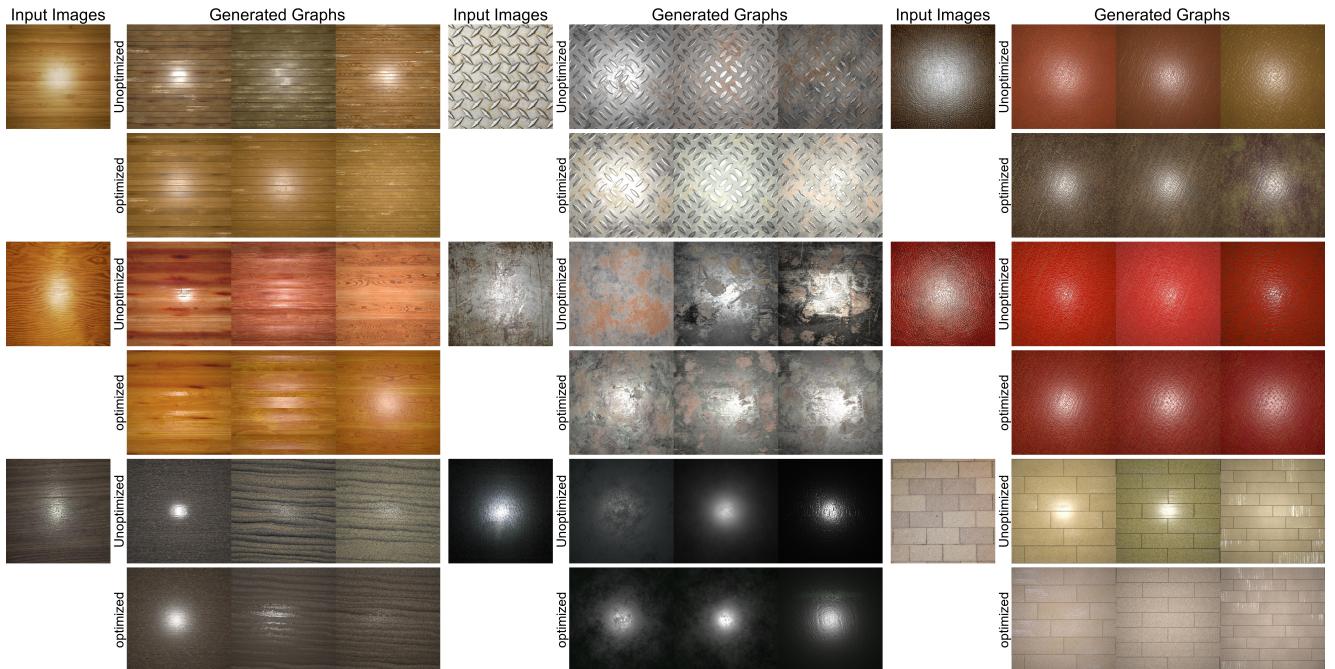


Figure 5: Real Image Conditioning. We show real photos as inputs and three of our generated material graphs. For each example, the top row shows non-optimized outputs directly predicted by our models, while the bottom row shows our final output after optimization. See supplemental material for more results.

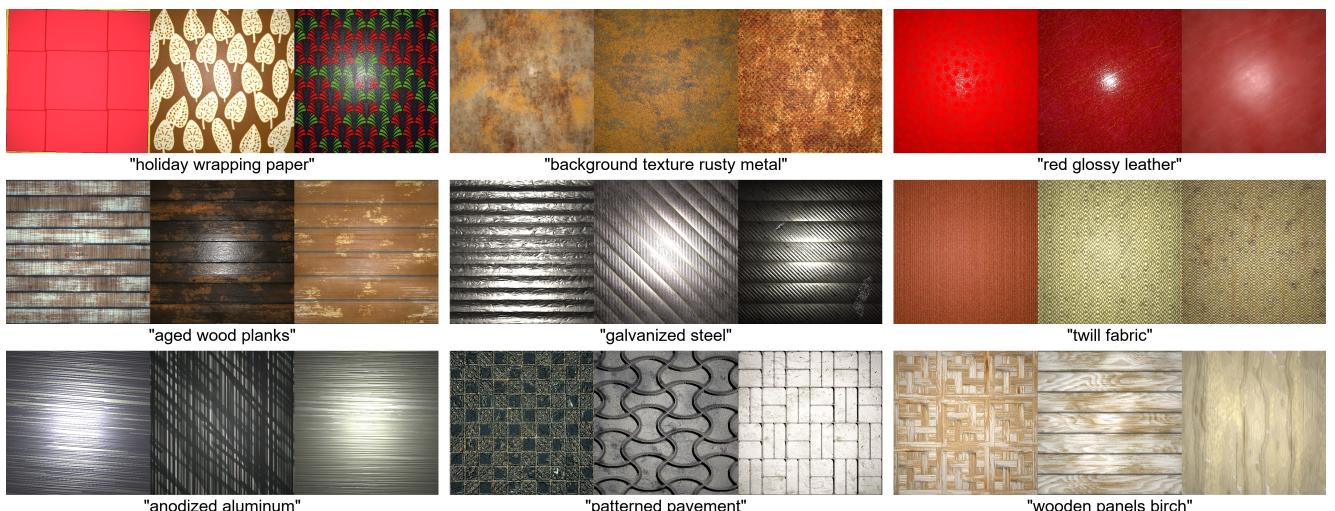


Figure 6: Text Conditioning. Our model generates multiple procedural material graphs given various text prompts. See supplemental material for more results.

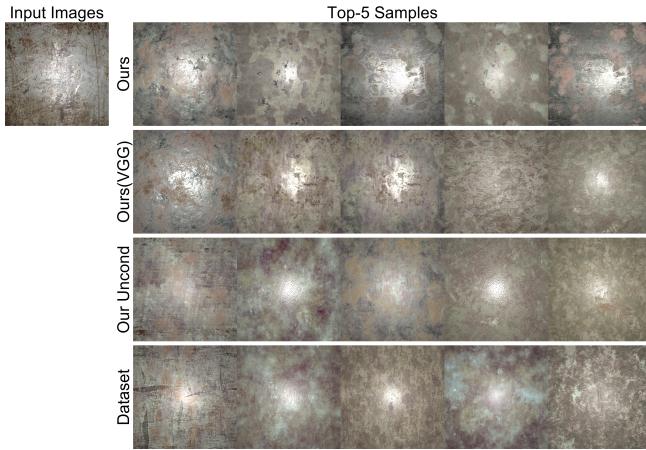


Figure 7: We show visual comparisons to our baselines. The top-5 samples are used to calculate the statistics shown in Table 1. Visually, our model generates material graphs similar to a query in a huge database.

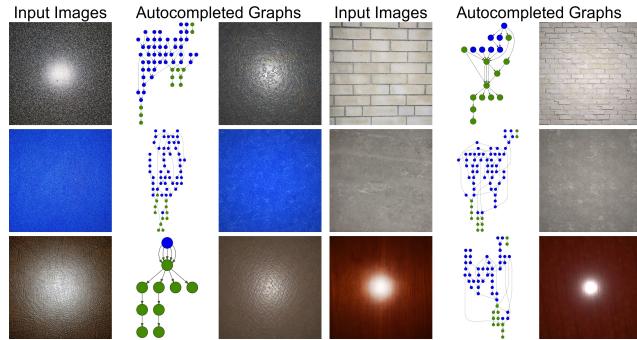


Figure 8: As a sequential model, our model can accept partial sequences (partially completed material graphs) and generate the rest of the structures and parameters toward image prompts. As in Fig. 1, existing structures are blue and our predicted are green. See supplemental for more results.

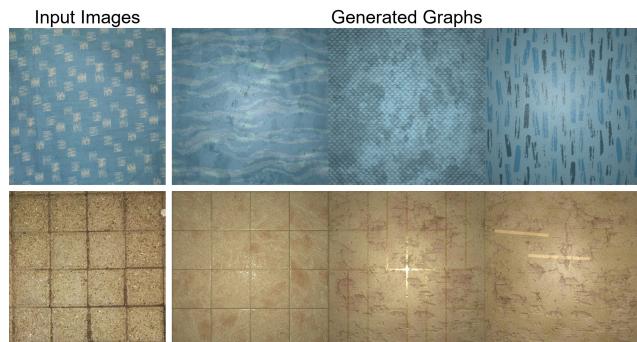


Figure 9: Our method is unable to reproduce the detailed appearance of some image inputs due to limited training data and prediction errors.

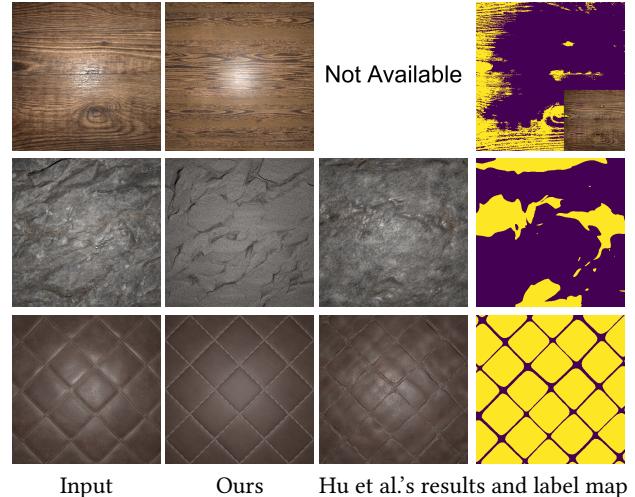


Figure 10: Comparison to the state-of-the-art inverse procedural modeling method. The semi-automatic pipeline by Hu et al. [2022c] requires segmentation as a starting point. However, not all material maps are easily segmentable e.g., the wood pattern, causing the first step of the algorithm to fail (first row). We show the scribbles used in the matting algorithm (segmentation) overlapped with the albedo map (Inset). Additionally, they model the appearance by approximating the Power Spectrum Density (PSD) of each material using a Gaussian noise, which is not expressive enough to model the appearance of some materials (second row structure and third row appearance).

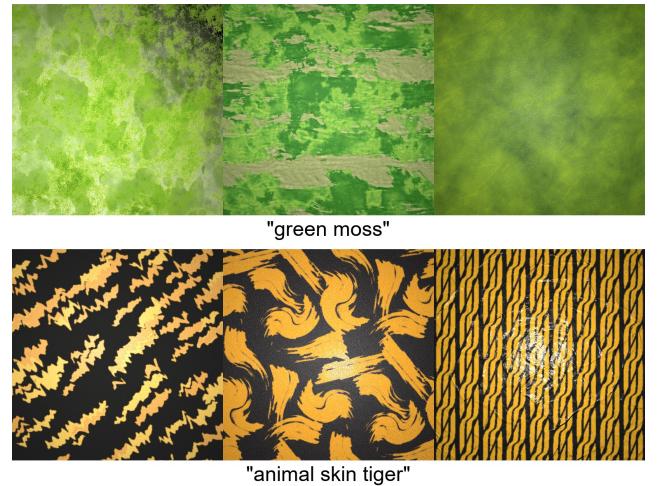


Figure 11: Material graphs sampled by text prompts do not always generate realistic materials which match the expected appearance given the text prompt.