MultiMat: Multimodal Program Synthesis for Procedural Materials using Large Multimodal Models

Anonymous authors
Paper under double-blind review

Abstract

Material node graphs are programs that generate the 2D channels of procedural materials, including geometry such as roughness and displacement maps, and reflectance such as albedo and conductivity maps. They are essential in computer graphics for representing the appearance of virtual 3D objects parametrically and at arbitrary resolution. In particular, their directed acyclic graph structures and intermediate states provide an intuitive understanding and workflow for interactive appearance modeling. Creating such graphs is a challenging task and typically requires professional training. While recent neural program synthesis approaches attempt to simplify this process, they solely represent graphs as *textual* programs, failing to capture the inherently visual-spatial nature of node graphs that makes them accessible to humans. To address this gap, we present MultiMat, a *multimodal* program synthesis framework that leverages large multimodal models to process both visual and textual graph representations for improved generation of procedural material graphs. We train our models on a new dataset of production-quality procedural materials and combine them with a constrained tree search inference algorithm that ensures syntactic validity while efficiently navigating the program space. Our experimental results show that our multimodal program synthesis method is more efficient in both unconditional and conditional graph synthesis with higher visual quality and fidelity than text-only baselines, establishing new state-of-the-art performance.

1 Introduction

Procedural materials have become increasingly important in modern 3D content creation, offering artists greater control and flexibility in designing surface appearances for digital assets. Unlike traditional image-based textures, which are constrained by fixed resolutions and limited editability, procedural material modeling tools like Adobe Substance Designer (Adobe, 2025c) or Blender (Blender, 2025) leverage node-based graphs to generate textures programmatically. This enables resolution-independent execution, high-level parametric control, and non-destructive editing workflows that have proven valuable in industries such as game development, film production, and VR/AR applications (Ebert et al., 2003). More specifically, a procedural material is defined as a directed graph where nodes represent texture generators (e.g., noise functions, patterns) or filtering operations (e.g., blurs, color adjustments), and edges encode the flow of data between these operations, ultimately producing the texture maps required by physically-based rendering (PBR) models (Pharr et al., 2016) (cf. Figure 1). However, the complexity of crafting these procedural material graphs presents a substantial barrier to entry, creating a pressing need for automated and semi-automated approaches to support material artists at all levels of proficiency.

With recent advances in neural program synthesis (Huynh & Lin, 2025), procedural material synthesis has become increasingly feasible. MatFormer pioneered this direction with a multi-stage transformer-based model for unconditional generation with Adobe Substance Designer (Guerrero et al., 2022). Building on this foundation, Hu et al. (2023) extended the approach to support conditional synthesis, enabling applications such as inverse rendering (Patow & Pueyo, 2003), i.e., generating procedural materials that match the appearance of captured or rendered images. More recently, VLMaterial demonstrated that large language models (Zhao et al., 2025) can effectively perform end-to-end procedural material synthesis (Li et al., 2025a). However, these approaches share a fundamental limitation: they generate node graphs as text-only programs without access to visual feedback during

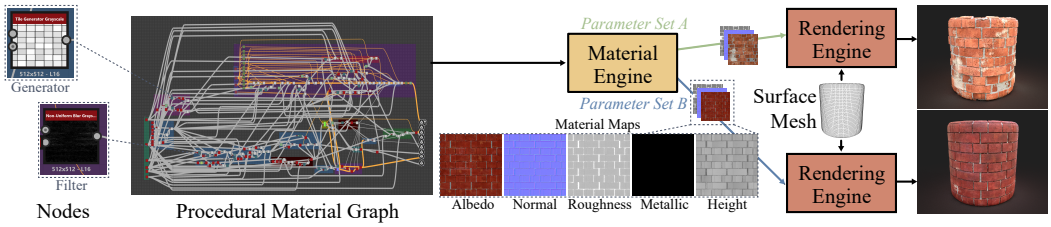


Figure 1: Procedural materials offer powerful control over the appearance of 3D objects through a few high-level parameters. Here, a production-grade example (left) with the images obtained using two distinct parameter sets A and B (right).

synthesis. This contrasts sharply with how human artists work, who create procedural materials by manipulating node graphs through an arguably more intuitive visual interface, as illustrated in Figure 1 (left). Without visual feedback, models must rely solely on textual representations to reason about complex spatial relationships and visual outcomes, a task that becomes increasingly difficult as material complexity grows. To address this limitation, we propose a novel *multimodal program synthesis* paradigm based on large multimodal models (Yin et al., 2024) that incorporates visual feedback throughout the generation process, more closely mirroring human creative workflows. We demonstrate that this approach, to which we refer as MultiMat, outperforms previous state-of-the-art methods (cf. §6). Our key contributions are as follows:

1. We introduce MultiMat, a novel procedural material synthesis approach that incorporates visualizations of intermediate graphs, including node states, into its context. This multimodal feedback loop improves material quality substantially compared to text-only baselines.
2. Investigating intermediate states enables real-time validation of each generated node. This allows us to develop a tree search algorithm that backtracks upon encountering invalid states, enabling more efficient inference than prior methods, which often produce invalid graphs.
3. We implement a transpiler that converts between Adobe Substance Designer formats and a compact representation suitable for language modeling while supporting the complete feature set. This enables training on larger datasets and the generation of more complex materials than previous approaches, which examined only limited subsets of Designer's capabilities.

2 Related Work

Large Language Models for Program Synthesis Our work builds upon recent advances in neural program synthesis (Parisotto et al., 2017; Devlin et al., 2017; Ellis et al., 2021). Traditional program synthesizers require formal specifications and employ search or logical derivation to produce programs that provably satisfy these specifications (Alur et al., 2013). Recently, large language models have demonstrated impressive capabilities in this domain (Huynh & Lin, 2025; Li et al., 2025b; Lozhkov et al., 2024; Li et al., 2023b; Rozière et al., 2023; Fried et al., 2023; Li et al., 2022; Chen et al., 2021). However, current research predominantly targets high-resource programming languages such as Python, Java, and JavaScript (Zan et al., 2023; Huynh & Lin, 2025). In contrast, our work synthesizes *graphics* programs, which pose unique challenges due to domain-specific requirements and considerable data scarcity, establishing it as a distinct research area.

Graphics Program Synthesis Deep learning approaches have shown strong performance in synthesizing graphics programs that compile to visual outputs (Ellis et al., 2018; 2019; Ganin et al., 2018). This progress has been accelerated by the emergence of large multimodal models, particularly vision-language models that bridge visual and textual domains (Alayrac et al., 2022; Liu et al., 2023; Belouadi et al., 2024b; Kulits et al., 2024; Li & Ellis, 2024; Kapur et al., 2025). The field encompasses both controlled experimental settings using domain-specific languages (Ellis et al., 2018; Tian et al., 2019; Sharma et al., 2018; Cámara et al., 2023; Kulits et al., 2024; Kapur et al., 2025) and practical applications. Notable examples include systems for generating scientific figures using Ti*k*Z (Belouadi et al., 2024a;b; 2025; Laurençon et al., 2024; Laurençon et al., 2024; Tong et al., 2024; Zhang et al.,

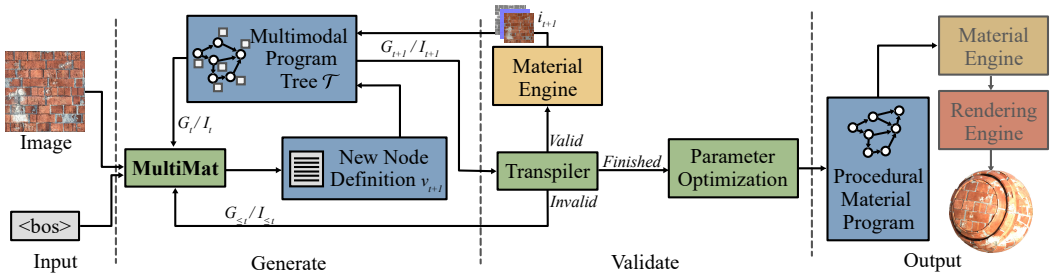


Figure 2: Architecture overview of MultiMat during inference. The system constructs a multimodal program tree $\mathcal{T}$ by iteratively generating node definitions. At each step t, the system derives a graph G_t of valid nodes along with corresponding intermediate outputs I_t by traversing $\mathcal{T}$, which may contain both valid and invalid nodes, to generate the next node v_{t+1}. When transpilation and execution succeed, the system advances with updated graph G_{t+1} and outputs I_{t+1}. If errors occur, it reverts to a previous state $(G_{\leq t}, I_{\leq t})$. The generation process initiates from either an input image or unconditionally using a beginning-of-sequence token (<bos>). Following optional parameter optimization (cf. §6.2), the final procedural material can be applied to any target geometry for rendering.

2025) and automating data visualization (Mackinlay, 1986; Roth et al., 1994; Luo et al., 2021; Wu et al., 2024; Voigt et al., 2024). However, these approaches generate code designed for text-based editing and therefore do not face the unique circumstances of node graphs in procedural material synthesis that our work addresses.

Procedural Material Synthesis Procedural material modeling is one of the most challenging domains in graphics program synthesis. The combination of lengthy, complex material programs and severe data scarcity creates unique obstacles for learning-based approaches (Li et al., 2025a; 2024). Existing methods primarily focus on inverse procedural material modeling by synthesizing graphs that reproduce a given target appearance (Hu et al., 2023) or unconditional generation to create diverse, novel materials without specific targets (Guerrero et al., 2022). A related line of work optimizes parameters of existing material graphs to match image targets by transpiling them into differentiable programs (Shi et al., 2020; Hu et al., 2022; Li et al., 2023a). As discussed in §1, previous generative approaches are limited to text-only representations, a limitation we address in this work.

3 Background on Procedural Materials

As indicated in §1, procedural materials are directed acyclic graphs G, executed by a material engine to produce raster images representing the physical properties of materials. These so-called material maps define surface characteristics e.g., albedo, roughness or normal (tangent space orientation), that enable photorealistic rendering when applied to 3D objects, with their appearance controlled through a small set of high-level parameters (cf. Figure 1). The internal structure of a material graph G comprises nodes $\{v_1, v_2, \ldots, v_N\}$ connected by edges that define the flow of image data. Each node v_i functions as either a generator that creates new image content or a filter that transforms existing images from upstream nodes. Common node operations include noise generation, blending, and mathematical transformations, which collectively produce intermediate image outputs $I = \{i_1, i_2, \ldots, i_N\}$. The behavior of each node is governed by parameters that may be discrete or continuous scalars or vectors, providing fine-grained control over the final material appearance.

Professional material authoring tools such as Blender and Adobe Substance Designer enable artists to construct and modify procedural material graphs through visual interfaces (cf. Figure 1). Users can interactively add or remove nodes and edges while adjusting node parameters to achieve desired visual effects. Among these tools, Adobe Substance Designer stands out for its particularly expressive node graph system, which MultiMat specifically targets. It offers advanced capabilities for creating complex material appearances through features like function graphs and pixel processors. Function graphs allow parameters to be controlled through custom operations on input values, while pixel processors enable users to define specialized computational graphs that operate on individual pixels

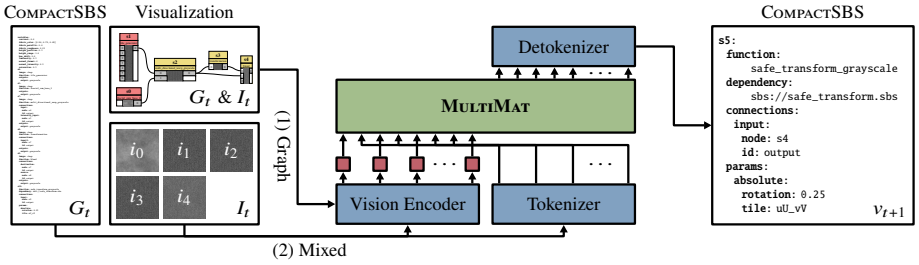


Figure 3: Visualization of the two conditioning approaches used by MultiMat for generating node definition v_{t+1}. In the graph-conditioned approach (1), MultiMat processes the graph G_t as a visual representation similar to human perception. In the mixed-conditioned approach (2), MultiMat receives G_t as a multimodal program where <img> tokens are replaced with their corresponding vision encoder representations from I_t.

using sequences of atomic mathematical operations. These sophisticated capabilities make automated procedural material synthesis a particularly challenging problem in this domain.

4 The MultiMat Model & Architecture

Figure 2 illustrates our complete model pipeline. At its core, MultiMat is a vision-language model, trained for synthesizing procedural material graphs. It accepts images as input for inverse procedural material synthesis and supports unconditional generation. Unlike previous approaches, MultiMat generates nodes *topologically*, ensuring each node precedes all nodes it connects to. This enables an iterative generation process detailed below that can provide continuous visual feedback to the model, verify the validity of intermediate outputs, and recover from errors automatically in certain cases.

4.1 Multimodal Program Synthesis

Given a partially generated material graph $G_t = \{v_1, v_2, \ldots, v_t\}$ with nodes v_i at generation step t, the topological ordering of nodes allows to visualize intermediate node states, similar to visual editing environments that target humans. This enables an iterative generation loop where MultiMat generates one node definition—including node parameters and connections to previous nodes—at a time that is processed accordingly before the generation continues. After generating node v_{t+1} in an intermediate text format (cf. §5), we combine it with the existing node definitions $\{v_1, \ldots, v_t\}$ and feed them to a transpiler, which compiles the intermediate representations back to a format the material engine understands. We then use the material engine to visualize the state of node v_t. Upon successful transpilation and execution, v_{t+1} is appended to the graph G_{t+1}. This updated state, including the visualized intermediate outputs I_t, is fed back to the model to generate the subsequent node v_{t+2} (cf. Figure 2). If execution or transpilation fails, we discard the current v_t and resample, or backtrack further in case of repeated errors (cf. §4.2). We explore two complementary approaches for representing G_t and I_t as *multimodal programs* to the model, as visualized in Figure 3:

Mixed Conditioning Starting with a textual representation of G_t (cf. §5), we enhance each node v_i with an additional field containing its visualized intermediate state. This creates a multimodal program where the model processes textual tokens interleaved with image patch embeddings (cf. Figure 3). To manage the increased context size from image embeddings, we omit node parameters (which are implicitly encoded in the visualizations) but explicitly include node output type information (e.g., grayscale or color) that the model cannot infer from the visualization alone.

Graph Conditioning This approach more closely mirrors human visual experience by conditioning MultiMat solely on a visualization of the entire graph G_t with embedded intermediate visual outputs I_t, as shown in Figure 3. The model generates subsequent node v_{t+1} using only this complete visual context, without explicit access to underlying textual node definitions.

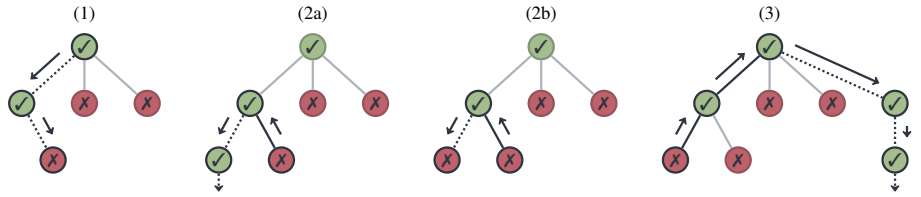


Figure 4: Visualization of our inference algorithm as a tree search. Tree nodes represent generated node definitions, and edges represent possible continuations. The algorithm proceeds as follows: generation continues until an invalid state (✗) is encountered (1), triggering backtracking to the previous node; from this point, if a valid node (✓) is generated, normal generation resumes (2a), but if invalid outputs persist (2b), the algorithm backtracks further until a valid path is found (3).

4.2 Incremental Tree Search

Another advantage of topological node ordering is the ability to validate node definitions incrementally during generation. By invoking our transpiler and material engine at each step, we can detect syntactic and semantic errors immediately rather than waiting until the entire graph is complete. When an erroneous node definition is encountered, we execute an adaptive backtracking strategy: first discarding and resampling the problematic node, and if errors persist, inferring deeper structural issues by reversing further back in the generation sequence. Specifically, we discard the $2^{(i-1)}$ most recently generated nodes, where i represents the current backtracking iteration. This approach effectively transforms our generation process into an *incremental tree search* on a tree $\mathcal{T}$ of valid and invalid nodes (cf. Figure 4), systematically exploring the solution space to discover valid programs. This incremental validation approach identifies invalid outputs much faster than previous approaches, which require sampling complete programs before validation can commence.

4.3 Automatic Error Repair

Through systematic analysis of failure cases, we identified recurring error patterns that could be repaired automatically: (1) removal of extraneous parameters that are specified for node types that do not support them, and (2) automatic insertion of conversion nodes to resolve type mismatches between connected nodes. For instance, when a color output is erroneously connected to a grayscale input, we automatically insert an appropriate grayscale conversion node. Conversely, when a grayscale output feeds into a color input, we insert a gradient map node to perform type conversion. These repair mechanisms increase the proportion of valid generations without requiring additional sampling steps.

5 Dataset

To support the training and evaluation of MultiMat, we collect procedural materials from Adobe's Substance 3D Assets Repository (Adobe, 2025a). Unlike previous work that either focuses on basic graphs utilizing only a subset of Substance Designer features (e.g., lacking complex nodes such as pixel processors or function graphs; Guerrero et al., 2022; Hu et al., 2023) or targets other tools with more limited capabilities (Li et al., 2025a), our approach supports the complete feature set. This comprehensive coverage enables us to collect over 6 000 unique materials, substantially more than existing datasets. Table 1 summarizes key characteristics of our dataset compared to prior work.

Models	Size	Max Nodes	Feature Set	Program
MatFormer	2 820	≤ 400[1]	Subset	Designer
Mat. (Cond)	4 667	≤ 80[1]	Subset	Designer
VLMaterial	3 663	30	Limited	Blender
MultiMat	6 878	128	Complete	Designer

[1] Upper bound in complex filtering pipeline, actual could be less.

Table 1: Comparison of training data of MatFormer (Guerrero et al., 2022), conditional MatFormer (Hu et al., 2023), VLMaterial (Li et al., 2025a), and MultiMat (ours). We procure the largest dataset with the most comprehensive set of features.

Human-Readable Graph Representation Substance Designer's native file format (SBS) has not been designed for human readability, containing verbose XML structures, embedded binary data, legacy metadata, and other implementation details, which makes direct language modeling impractical. To address this, we develop a bidirectional transpiler that converts between SBS and a compact, human-readable YAML-based representation with topological node order, which we call *CompactSBS*. Unlike previous approaches that support only partial feature sets (Guerrero et al., 2022; Hu et al., 2023), our transpiler preserves the complete functionality of Substance graphs with programs that are, on average, over 80% shorter. Models operate exclusively in CompactSBS, with outputs transpiled back to SBS for execution. We provide representative examples in Figure 3 and complete program listings in Appendix A.

Graph Preprocessing Our preprocessing pipeline standardizes graphs for the PBR workflow, focusing on five essential texture maps: base color, normal, roughness, metallic, and height. We trace backwards from these outputs to identify all contributing nodes, pruning unconnected components and other output maps. Graphs containing embedded bitmap graphics and SVGs are excluded to keep graphs fully procedural. We further filter out graphs exceeding 128 nodes and flatten hierarchical structures by inlining nested subgraphs and custom author dependencies into the main graph. Non-atomic nodes from the standard Substance Designer library remain as external references.

6 Experiments

We build MultiMat models upon the QWen2.5$_{\text{VL}}$ (7B) foundation model (Bai et al., 2025). We train and evaluate separate models for unconditional generation (cf. §6.1) and inverse procedural material synthesis (cf. §6.2). Across all model variants, we maintain a consistent maximum sequence length of 8 192 tokens. The training setup consists of 5 epochs using a learning rate of 5e−5 and a batch size of 128. To ensure diversity in our generated outputs, we set the inference sampling parameters to a temperature of 0.8 and a top-p value of 0.95. We provide examples in Figure 5 and Appendix A.

6.1 Evaluation of Unconditional Generation

For unconditional generation, the mixed conditioning variant, *MultiMat (Mixed)*, embeds node previews at 140×140 resolution, resulting in 25 patch embeddings per image. For the graph conditioning variant, *MultiMat (Graph)*, graph visualizations can utilize up to 6 144 tokens, with larger images downscaled to accommodate this limit. We generate 100 outputs per model for evaluation.

Baselines For text-only procedural material synthesis, VLMaterial represents the current state-of-the-art approach. However, its Blender-specific training makes direct comparison with our method difficult. We therefore create VLMaterial (SBS) by retraining a VLMaterial-style model on our dataset for fair comparison. Since VLMaterial (SBS) does not receive any images in the unconditional setting, we base it on the larger and more powerful text-only model QWen3 (8B; Yang et al., 2025a), giving it a slight advantage over our models. While graphics program synthesis research typically also benchmarks against proprietary large language models such as GPT-4o (OpenAI et al., 2024) or Claude 4 (Anthropic, 2025), which have demonstrated competitive performance in related domains (Belouadi et al., 2024a;b; 2025; Rodriguez et al., 2025), these models' unfamiliarity with CompactSBS and inability to produce valid SBS output preclude their inclusion as baselines.

Metrics Our multimodal task permits diverse evaluation schemes for automatic evaluation. To evaluate the *visual quality* of generated materials, we compute the Kernel Inception Distance (**KID**; Bińkowski et al., 2018), which compares the distribution of generated material maps with material maps from our dataset. To detect degenerate low KID scores due to *memorization* of training data (a legitimate concern given our relatively small dataset), we also calculate **ROUGE-L** scores (Lin, 2004) between the CompactSBS representation of our generated materials and the training set (with masked parameters). This metric computes the longest common subsequence and serves as an effective memorization indicator (Hans et al., 2024). Notably, we specifically require *consecutive* subsequences due to CompactSBS's limited syntactic diversity, which could otherwise produce misleading matches. To measure *efficiency*, we introduce the Node Error Ratio (**NER**), defined as the average ratio between discarded nodes and the total number of generated nodes.

Models	DSim$_\uparrow$	CLIP$_\uparrow$	Style$_\downarrow$	KID$_\downarrow$	ROUGE-L$_\downarrow$	NER$_\downarrow$
VLMaterial (SBS)	31.344	65.678	3.211	14.976	**1.621**	16.933
MultiMat (Mixed)	34.922	66.737	3.199	3.675	2.194	**12.388**
MultiMat (Graph)	**36.609**	**67.907**	**3.178**	**2.801**	2.037	17.046
VLMaterial$^+$ (SBS)	31.348	65.867	3.126	27.862		
MultiMat$^+$ (Mixed)	40.258	69.687	3.093	17.792		
MultiMat$^+$ (Graph)	**40.367**	**70.114**	**3.046**	**14.886**		

Table 3: System-level scores $\times$ 100 for conditional (inverse) generation, without (top) and with (bottom) parameter optimization. Bold and underlined values indicate the best and second-best scores for each metric column, respectively. Arrows indicate metric directionality. ROUGE-L and NER scores remain unchanged by parameter optimization and are shown only once. MultiMat (Graph) and MultiMat$^+$ (Graph) achieve the best overall performance.

Results Table 2 presents the system-level metric scores for our evaluation. MultiMat (Graph) leads in visual quality with the lowest KID score, outperforming MultiMat (Mixed) by over 4pp (percentage points) and VLMaterial (SBS) by more than 11pp. This considerable gap in performance suggests that the better the visual representations are aligned with human creative workflows, the better the results—an intuitive but important finding. All models exhibit minimal memorization, with ROUGE-L scores showing that no more than 4% of any generated sequence matches a contiguous segment from the training data. Nonetheless, both MultiMat variants demonstrate approximately 1.5pp lower copying rates compared to VLMaterial (SBS), suggesting slightly better generalization. Regarding efficiency, MultiMat (Mixed) excels with the lowest NER, achieving a 6pp improvement over the other models. Both MultiMat (Graph) and VLMaterial (SBS) show comparable NER scores around 15%. For MultiMat (Graph), these errors are primarily due to OCR-like errors in reading node names and function types embedded as text in graph images. In contrast, we attribute the errors in VLMaterial (SBS) to fundamental difficulties in understanding graph structures. Despite these limitations, the error rates remain within acceptable bounds for practical applications, and MultiMat (Graph) emerges as the best overall model.

Models	KID$_\downarrow$	ROUGE-L$_\downarrow$	NER$_\downarrow$
VLMaterial (SBS)	14.155	3.641	14.846
MultiMat (Mixed)	6.752	2.195	**8.923**
MultiMat (Graph)	**2.365**	**1.915**	15.024

Table 2: System-level scores $\times$ 100 for unconditional generation. Bold and underlined values indicate the best and second-best scores for each metric column, respectively. Arrows indicate metric directionality. MultiMat (Graph) achieves the best overall performance.

6.2 Evaluation of Conditional Generation

As in prior work (Hu et al., 2023; Li et al., 2025a), we train inverse MultiMat variants that learn to generate procedural materials from rendered images. These models follow the same training procedure as their unconditional counterparts, with one key modification: each training example is preceded by a 512×512 rendering of itself, which adds 324 additional image patches to the model context. During inference, the model takes an image as input and generates a corresponding procedural material. We reserve 100 examples from our data as held-out test data for evaluation.

Baselines Analogously to §6.1, we adapt VLMaterial for inverse rendering with SBS and use it as a baseline. Since an image input is now required for VLMaterial (SBS), we also base it on Qwen2.5$_{\text{VL}}$ (7B) instead of Qwen3 (8B) and train it using the same method as MultiMat.

Parameter Optimization To further refine generated materials, we apply gradient-based optimization using differentiable rendering. This approach has proven effective for optimal parameter estimation (Shi et al., 2020; Hu et al., 2022; Li et al., 2023a; Hu et al., 2023). We employ DiffMat (Shi et al., 2020; Li et al., 2023a), a widely adopted differentiable renderer for Designer materials, to optimize the generated graphs against the input images. Models using this refinement step are denoted as MultiMat$^+$ and VLMaterial$^+$, respectively.

Metrics In addition to the metrics from §6.1, we evaluate reconstruction quality by rendering the generated materials and comparing them to the input images using perceptual similarity metrics. Specifically, we measure cosine similarity between **CLIP** image embeddings (Radford et al., 2021; Hessel et al., 2021), compute Style Loss loss (**Style**; Gatys et al., 2016) as the L1 distance between Gram matrices of VGG features, and calculate DreamSim (**DSim**; Fu et al., 2023), a learned perceptual similarity metric designed to align with human judgments.

Results Table 3 presents the system-level metric scores for conditional evaluation. The perceptual similarity metrics consistently demonstrate that MultiMat (Graph) achieves the highest fidelity to input images, with MultiMat (Mixed) performing second-best and VLMaterial (SBS) ranking last. For example, DreamSim scores are 36.609, 34.922, and 31.344, respectively, a ranking that mirrors our unconditional evaluation results. Parameter optimization yields substantial improvements in perceptual similarity, with MultiMat$^+$ (Graph) and MultiMat$^+$ (Mixed) showing average gains of 6% and 8%, respectively. In contrast, VLMaterial$^+$ (SBS) exhibits minimal improvement (only 1%), suggesting its outputs deviate too far from the input for parameter optimization to be effective. Interestingly, while parameter optimization improves perceptual similarity, KID scores increase. This could occur because optimization aligns outputs more closely with the test set, which represents only a subset of the training distribution, potentially increasing distance from the full distribution. Nevertheless, both MultiMat and MultiMat$^+$ variants outperform VLMaterial (SBS) and VLMaterial$^+$ (SBS) on KID by over 10pp, respectively. The remaining metrics reinforce trends from unconditional evaluation. ROUGE-L scores do not exceed 2% (indicating minimal memorization), and MultiMat (Mixed) produces the fewest errors. Overall, MultiMat (Graph) and its optimized variant, MultiMat$^+$ (Graph), deliver the strongest performance across metrics.

7 Analysis & Discussion

Our results show that model performance steadily improves as the degree of visualization of graphs increases, with MultiMat (Graph) achieving the best results overall. This finding aligns with how humans interact with procedural materials—through visual node graph interfaces—and validates established UX design principles in this domain. The qualitative examples in Figure 5 further illustrate this trend, with VLMaterial$^+$ (SBS) struggling to generate faithful outputs, indicating that purely text-based approaches are not ideal for expressive node graph systems like Designer. This limitation persists even with more powerful base models, as our unconditional generation experiments confirm. Beyond architectural improvements, our tree search algorithm enables more efficient graph generation; without it, models may have to resort to sampling complete outputs for validation (the inference approach used by previous methods), which is expensive. For instance, disabling tree search causes NER of VLMaterial (SBS) to deteriorate further from 14.846 to 33.953, highlighting how our search strategy can improve inference without further training.

8 Conclusion

We present MultiMat, a multimodal program synthesis framework and model suite that generates procedural materials by incorporating visual feedback throughout the generation process. Our key insight is that procedural material graphs are inherently visual-spatial programs, and treating them as such leads to substantial improvements over text-only approaches. By conditioning on visual intermediate states—either interleaved with text (mixed conditioning) or as complete graph visualizations (graph conditioning)—our models achieve consistent improvements over text-only baselines. Our incremental tree search algorithm further enhances generation efficiency by validating nodes as they are created and backtracking upon errors. While we demonstrate MultiMat specifically for procedural material synthesis, we hope its general principles will inspire further research at the intersection of computer graphics, program synthesis, and multimodal AI.

Future Work The development of procedural material graph synthesis approaches is currently constrained by limited training data availability. We plan to address this challenge through self-learning techniques (He et al., 2020; Wei et al., 2021) that leverage our unconditional models to generate synthetic supervised training data by rendering outputs and subsequently training conditional models on this expanded data. Additionally, we aim to develop a unified model trained across multiple

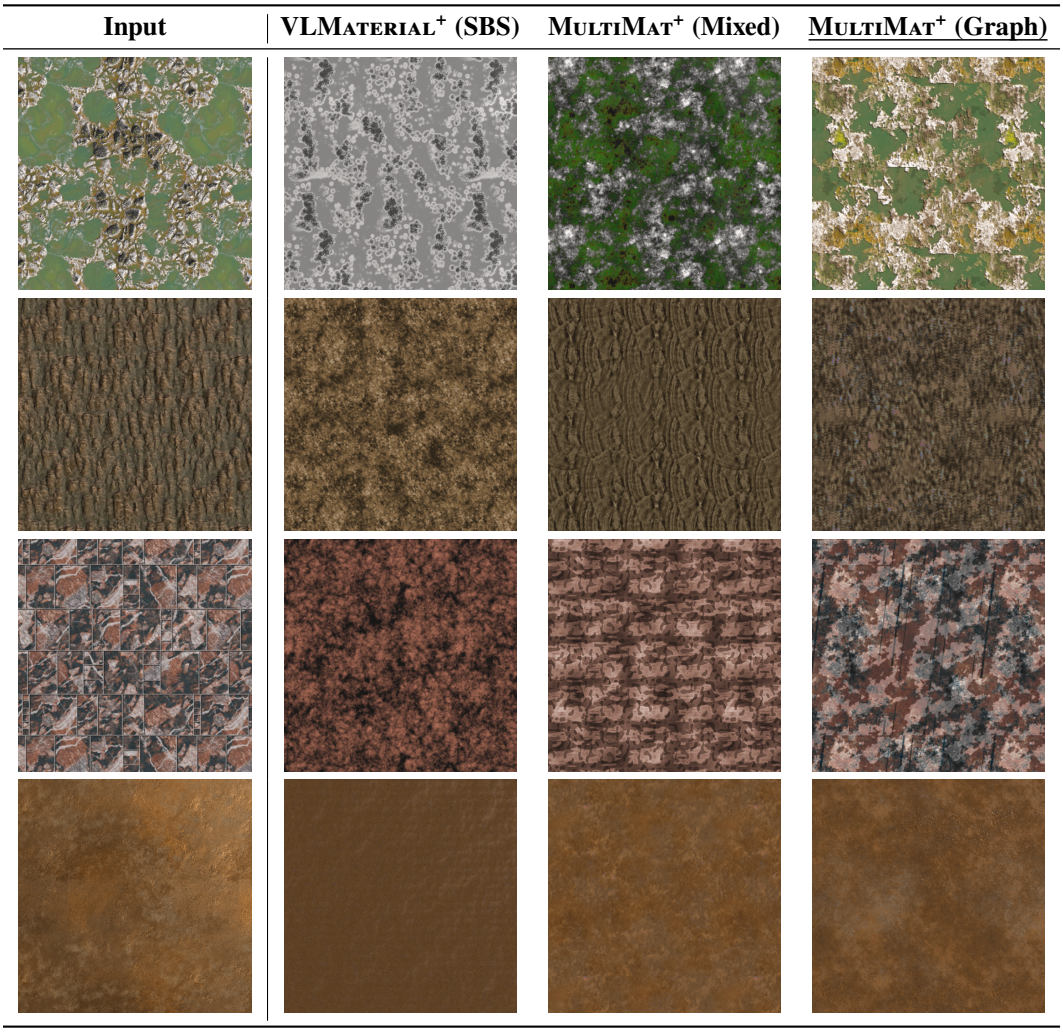


Figure 5: Qualitative results for inverse procedural material modeling. The leftmost column shows input materials from graphs filtered during preprocessing (e.g., due to excessive length), making these particularly challenging test cases. Following Hu et al. (2023); Li et al. (2025a), we generate multiple programs ($N = 40$) per model and select the result with the highest DreamSim score. MultiMat+ (Mixed) consistently outperforms VLMaterial+ (SBS), while MultiMat+ (Graph) achieves the best results overall. Additional examples, including failure cases, are provided in Appendix A.

node graph systems to investigate potential transfer learning benefits (Pan & Yang, 2010). Beyond methodological advances, our models offer promising practical applications: conditional models could extract material graphs directly from photographic regions, while unconditional models could power intelligent auto-completion features in user interfaces. Furthermore, our methodology naturally extends to related domains such as vector graphics synthesis (Wu et al., 2023; Polaczek et al., 2025; Rodriguez et al., 2025; Yang et al., 2025b), where visual editing interfaces are similarly prevalent.

Limitations Although our models and baselines use the same or similar base models, they generate graphs in fundamentally different ways, resulting in considerable differences in training efficiency. Text-only models like VLMaterial can process entire graphs as single training examples, whereas MultiMat must adapt the visual context for each individual node, effectively processing training examples one node at a time. This difference leads to much longer training times: while VLMaterial completes training in a few hours on 8 × A100 80GB GPUs, MultiMat models require several days on the same hardware despite being trained on a comparable number of tokens. However, this training inefficiency does not affect inference, where both approaches achieve comparable generation speeds.

Ethics Statement

We ensure that all procedural materials collected for model training are properly licensed and explicitly permit such usage, thereby preventing any copyright infringement. In adherence to this principle, we specifically exclude Substance 3D Community Assets (Adobe, 2025b) from our training data due to licensing restrictions. While we acknowledge the use of generative models in preparing this manuscript, their application is strictly limited to writing assistance, such as paraphrasing, spell checking, and synonym suggestions.

References

Adobe. Substance 3D Assets. https://substance3d.adobe.com/assets, 2025a.

Adobe. Substance 3D Community Assets. https://substance3d.adobe.com/community-assets, 2025b.

Adobe. Substance 3D Designer. https://www.adobe.com/products/substance3d.html, 2025c.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, and 8 others. Flamingo: a visual language model for few-shot learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=EbMuimAbPbs.

Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pp. 1–8, 2013. doi: 10.1109/FMCAD.2013.6679385.

Anthropic. System card: Claude Opus 4 & Claude Sonnet 4, 2025. URL https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. Qwen2.5-VL technical report, 2025. URL https://arxiv.org/abs/2502.13923.

Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *The Twelfth International Conference on Learning Representations*, Vienna, Austria, May 2024a. URL https://openreview.net/forum?id=v3K5TVP8kZ.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. DeTikZify: Synthesizing graphics programs for scientific figures and sketches with TikZ. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, Vancouver, Canada, December 2024b. URL https://openreview.net/forum?id=bcVLFQCOjc.

Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen Eger, and Simone Paolo Ponzetto. TikZero: Zero-shot text-guided graphics program synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Honolulu, Hawaii, October 2025.

Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1lUOzWCW.

Blender. Blender. https://www.blender.org/, 2025.

Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. On the assessment of generative AI in modeling tasks: an experience report with chatgpt and UML. *Softw. Syst. Model.*, 22 (3):781–793, 2023. doi: 10.1007/S10270-023-01105-5. URL https://doi.org/10.1007/s10270-023-01105-5.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. Evaluating large language models trained on code, 2021.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural program learning under noisy I/O. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 990–998. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/devlin17a.html.

David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling - a procedural approach, Third Edition*. Morgan Kaufmann series in computer graphics and geometric modeling. Elsevier, 2003. ISBN 978-1-55860-848-1. doi: https://doi.org/10.5860/choice.32-5129.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Thirty-second Conference on Neural Information Processing Systems*, pp. 6062–6071, 2018. URL http://papers.nips.cc/paper/7845-learning-to-infer-graphics-programs-from-hand-drawn-images.

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

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, pp. 835–850, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383912. doi: 10.1145/3453483.3454080. URL https://doi.org/10.1145/3453483.3454080.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hQwb-lbM6EL.

Stephanie Fu, Netanel Yakir Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. DreamSim: Learning new dimensions of human visual similarity using synthetic data. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=DEiNSfh1k7.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1666–1675. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/ganin18a.html.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Paul Guerrero, Miloš Hašan, Kalyan Sunkavalli, Radomír Měch, Tamy Boubekeur, and Niloy J. Mitra. MatFormer: a generative model for procedural materials. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530173. URL https://doi.org/10.1145/3528223.3530173.

Abhimanyu Hans, John Kirchenbauer, Yuxin Wen, Neel Jain, Hamid Kazemi, Prajwal Singhania, Siddharth Singh, Gowthami Somepalli, Jonas Geiping, Abhinav Bhatele, and Tom Goldstein. Be like a goldfish, don't memorize! mitigating memorization in generative LLMs. In *The*

Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://openreview.net/forum?id=DylSyAfmWs.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. Revisiting self-training for neural sequence generation. In *Proceedings of ICLR*, 2020. URL https://openreview.net/forum?id=SJgdnAVKDH.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7514–7528, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.595. URL https://aclanthology.org/2021.emnlp-main.595.

Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393379. doi: 10.1145/3528233.3530733. URL https://doi.org/10.1145/3528233.3530733.

Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Generating procedural materials from text or image prompts. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi: 10.1145/3588432.3591520. URL https://doi.org/10.1145/3588432.3591520.

Nam Huynh and Beiyu Lin. Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications, 2025. URL https://arxiv.org/abs/2503.01245.

Shreyas Kapur, Erik Jenner, and Stuart Russell. Diffusion on syntax trees for program synthesis. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=wN3KaUXA5X.

Peter Kulits, Haiwen Feng, Weiyang Liu, Victoria Fernandez Abrevaya, and Michael J. Black. Re-thinking inverse graphics with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=u0eiu1MTS7.

Hugo Laurençon, Leo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=dtvJF1Vy2i.

Hugo Laurençon, Andrés Marafioti, Victor Sanh, and Léo Tronchon. Building and better understanding vision-language models: insights and future directions, 2024. URL https://arxiv.org/abs/2408.12637.

Beichen Li, Liang Shi, and Wojciech Matusik. End-to-end procedural material capture with proxy-free mixed-integer optimization. *ACM Transactions on Graphics (TOG)*, 42(4):1–15, 2023a.

Beichen Li, Yiwei Hu, Paul Guerrero, Milos Hasan, Liang Shi, Valentin Deschaintre, and Wojciech Matusik. Procedural material generation with reinforcement learning. *ACM Trans. Graph.*, 43(6), November 2024. ISSN 0730-0301. doi: 10.1145/3687979. URL https://doi.org/10.1145/3687979.

Beichen Li, Rundi Wu, Armando Solar-Lezama, Changxi Zheng, Liang Shi, Bernd Bickel, and Wojciech Matusik. VLMaterial: Procedural material generation with large vision-language models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=wHebuIb6IH.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, and 39 others. StarCoder: may the source be with you! *Transactions on Machine Learning*

Research, 2023b. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

Wen-Ding Li and Kevin Ellis. Is programming by example solved by LLMs? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=xqc8yyhScL.

Wen-Ding Li, Darren Yan Key, and Kevin Ellis. Toward trustworthy neural program synthesis. In *ICLR 2025 Workshop on Human-AI Coevolution*, 2025b. URL https://openreview.net/forum?id=HPlvbIJGWy.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Ré mi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, dec 2022. doi: 10.1126/science.abq1158. URL https://doi.org/10.1126%2Fscience.abq1158.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013/.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=w0H2xGHlkw.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, and 47 others. StarCoder 2 and The Stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pp. 1235–1247, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383431. doi: 10.1145/3448016.3457261. URL https://doi.org/10.1145/3448016.3457261.

Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, April 1986. ISSN 0730-0301. doi: 10.1145/22949.22950. URL https://doi.org/10.1145/22949.22950.

OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, and 400 others. GPT-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJ0JwFcex.

Gustavo Patow and Xavier Pueyo. A survey of inverse rendering problems. *Computer Graphics Forum*, 22(4):663–687, 2003. doi: https://doi.org/10.1111/j.1467-8659.2003.00716.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2003.00716.x.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, November 2016. ISBN 9780128006450.

Sagi Polaczek, Yuval Alaluf, Elad Richardson, Yael Vinker, and Daniel Cohen-Or. NeuralSVG: An implicit representation for text-to-vector generation, 2025. URL https://arxiv.org/abs/2501.03992.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. StarVector: Generating scalable vector graphics code from images and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16175–16186, June 2025.

Steven F. Roth, John Kolojejchick, Joe Mattis, and Jade Goldstein. Interactive graphic design using automatic presentation knowledge. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pp. 112–117, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916506. doi: 10.1145/191666.191719. URL https://doi.org/10.1145/191666.191719.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. Code LLaMA: Open foundation models for code, 2023.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 5515–5523. Computer Vision Foundation / IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00578. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Sharma_CSGNet_Neural_Shape_CVPR_2018_paper.html.

Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: Differentiable material graphs for procedural material capture. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to infer and execute 3D shape programs. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylNH20qFQ.

Shengbang Tong, Ellis L Brown II, Penghao Wu, Sanghyun Woo, Adithya Jairam Iyer, Sai Charitha Akula, Shusheng Yang, Jihan Yang, Manoj Middepogu, Ziteng Wang, Xichen Pan, Rob Fergus, Yann LeCun, and Saining Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Vi8AepAXGy.

Henrik Voigt, Kai Lawonn, and Sina Zarrieß. Plots made quickly: An efficient approach for generating visualizations from natural language queries. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 12787–12793, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.1119/.

Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. Theoretical analysis of self-training with deep networks on unlabeled data. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rC8sJ4i6kaH.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. IconShop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Trans. Graph.*, 42(6), December 2023. ISSN 0730-0301. doi: 10.1145/3618364. URL https://doi.org/10.1145/3618364.

Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. Automated data visualization from natural language via large language models: An exploratory study. *Proc. ACM Manag. Data*, 2(3), May 2024. doi: 10.1145/3654992. URL https://doi.org/10.1145/3654992.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. OmniSVG: A unified scalable vector graphics generation model, 2025b. URL https://arxiv.org/abs/2504.06263.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 11 2024. ISSN 2095-5138. doi: 10.1093/nsr/nwae403. URL https://doi.org/10.1093/nsr/nwae403.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7443–7464, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.411. URL https://aclanthology.org/2023.acl-long.411.

Haotian Zhang, Mingfei Gao, Zhe Gan, Philipp Dufter, Nina Wenzel, Forrest Huang, Dhruti Shah, Xianzhi Du, Bowen Zhang, Yanghao Li, Sam Dodge, Keen You, Zhen Yang, Aleksei Timofeev, Mingze Xu, Hong-You Chen, Jean-Philippe Fauconnier, Zhengfeng Lai, Haoxuan You, and 4 others. MM1.5: Methods, analysis & insights from multimodal LLM fine-tuning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HVtu26XDAA.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. A survey of large language models, 2025. URL https://arxiv.org/abs/2303.18223.

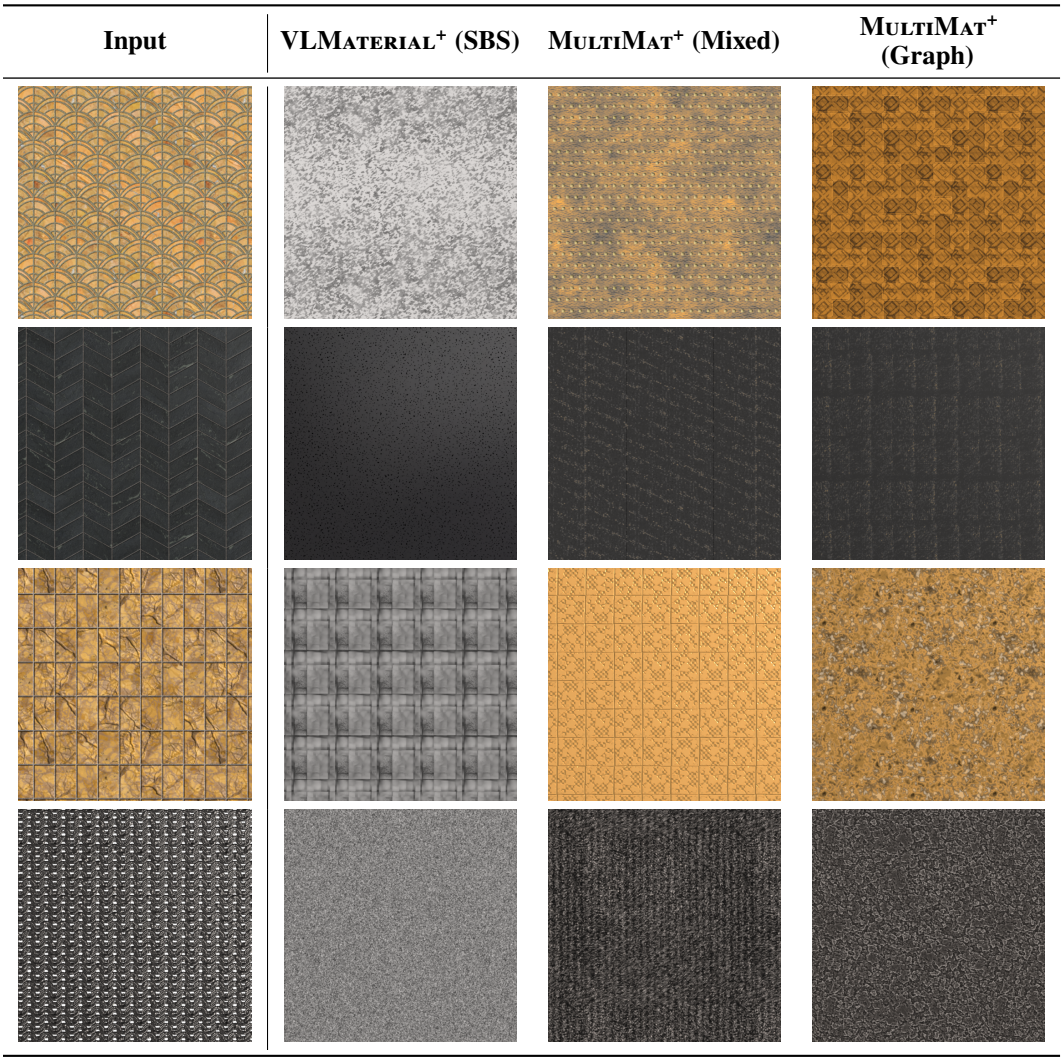


Figure 6: Representative failure cases from the same challenging subset in Figure 5. All models struggle to reproduce the intricate patterns in these examples, though MultiMat+ (Graph) and MultiMat+ (Mixed) still outperform VLMaterial+ (SBS).

A Additional Examples

Figure 6 complements Figure 5 by showcasing failure cases where our models struggle to produce faithful outputs, though notably, the outputs from MultiMat+ (Graph) and MultiMat+ (Mixed) still demonstrate superior representation of the input compared to VLMaterial (SBS). Beyond these conditional generation examples, Figure 7 presents unconditional samples generated by MultiMat (Graph), which exhibit high visual quality with realistic material properties. Adjacent to these rendered materials, we visualize their underlying material graphs in the same format used as model input. In Figure 8, we show a graph in CompactSBS representation to give an impression of the structure of our format.

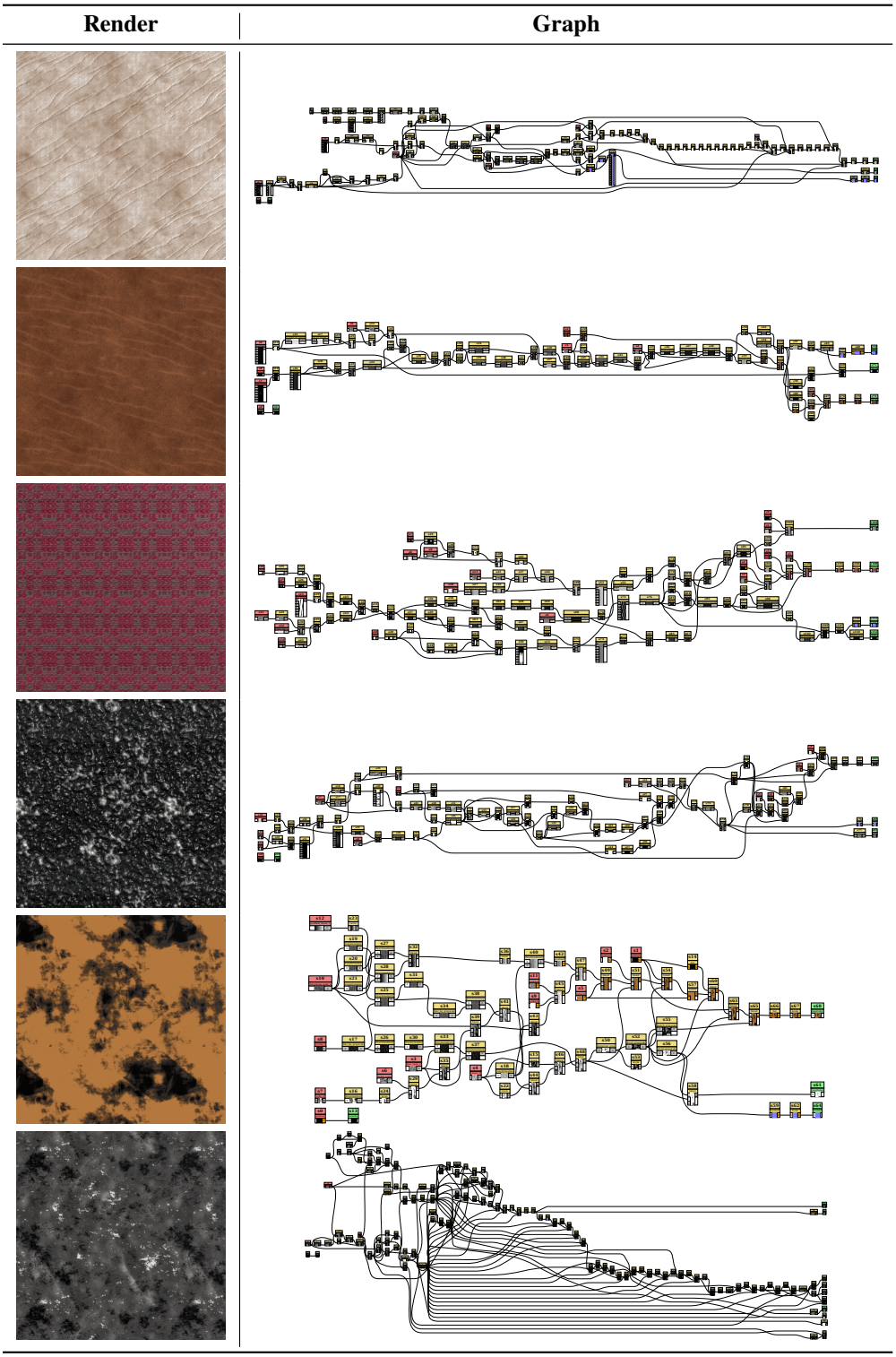
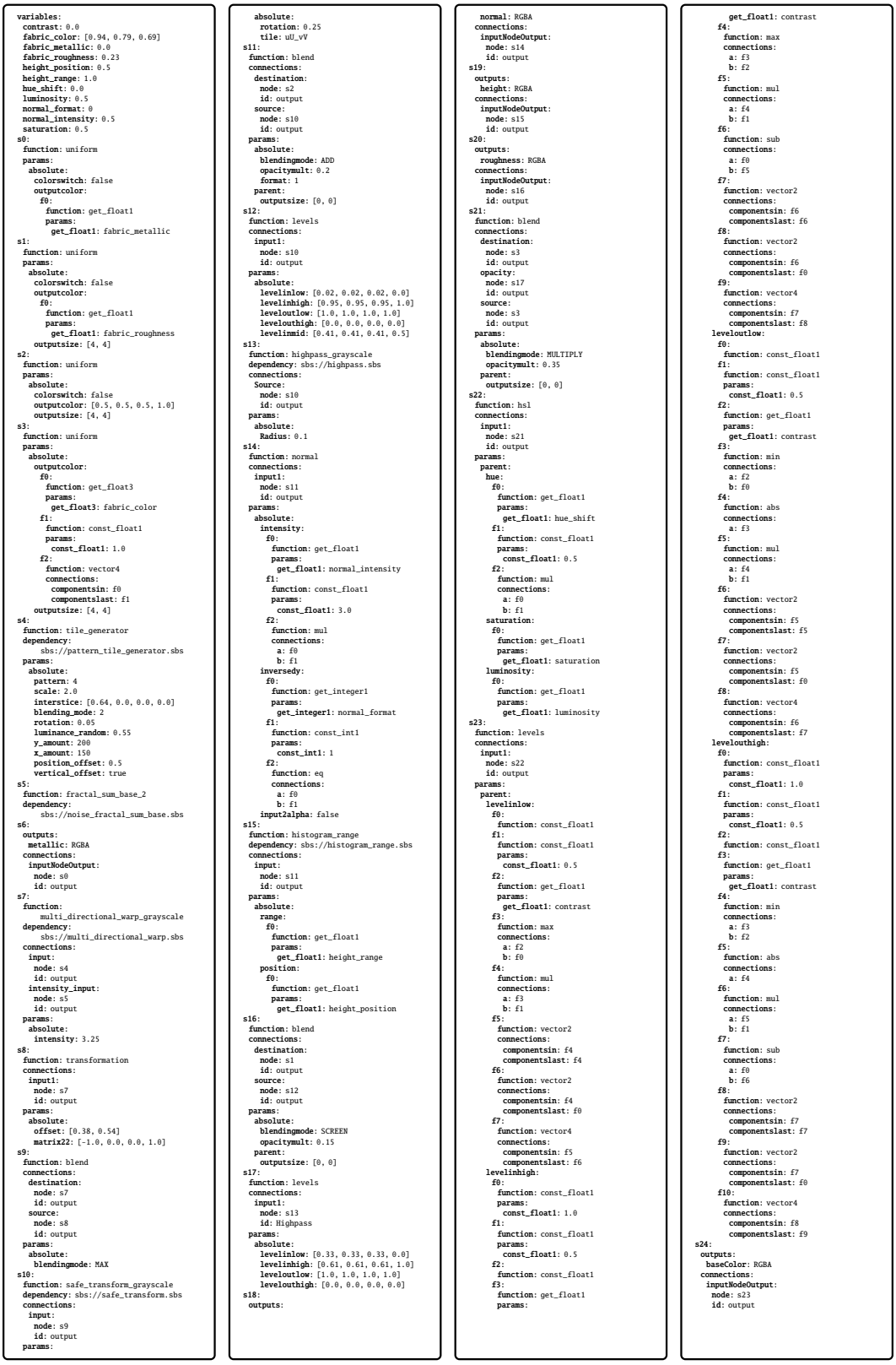


Figure 7: Example materials generated unconditionally by MULTIMAT (Graph), shown alongside their corresponding procedural graphs.



```
variables:
 contrast: 0.0
 fabric_color: [0.94, 0.79, 0.69]
 fabric_metallic: 0.0
 fabric_roughness: 0.23
 height_position: 0.5
 height_range: 1.0
 hue_shift: 0.0
 luminosity: 0.5
 normal_format: 0
 normal_intensity: 0.5
 saturation: 0.5
s0:
 function: uniform
 params:
  absolute:
   colorswitch: false
   outputcolor:
    f0:
     function: get_float1
     params:
      get_float1: fabric_metallic
s1:
 function: uniform
 params:
  absolute:
   colorswitch: false
   outputcolor:
    f0:
     function: get_float1
     params:
      get_float1: fabric_roughness
   outputsize: [4, 4]
s2:
 function: uniform
 params:
  absolute:
   colorswitch: false
   outputcolor: [0.5, 0.5, 0.5, 1.0]
   outputsize: [4, 4]
s3:
 function: uniform
 params:
  absolute:
   outputcolor:
    f0:
     function: get_float3
     params:
      get_float3: fabric_color
    f1:
     function: const_float1
     params:
      const_float1: 1.0
    f2:
     function: vector4
     connections:
      componentsin: f0
      componentslast: f1
   outputsize: [4, 4]
s4:
 function: tile_generator
 dependency:
    sbs://pattern_tile_generator.sbs
 params:
  absolute:
   pattern: 4
   scale: 2.0
   interstice: [0.64, 0.0, 0.0, 0.0]
   blending_mode: 2
   rotation: 0.05
   luminance_random: 0.55
   y_amount: 200
   x_amount: 150
   position_offset: 0.5
   vertical_offset: true
s5:
 function: fractal_sum_base_2
 dependency:
    sbs://noise_fractal_sum_base.sbs
s6:
 outputs:
  metallic: RGBA
 connections:
  inputNodeOutput:
   node: s0
   id: output
s7:
 function:
    multi_directional_warp_grayscale
 dependency:
    sbs://multi_directional_warp.sbs
 connections:
  input:
   node: s4
   id: output
  intensity_input:
   node: s5
   id: output
 params:
  absolute:
   intensity: 3.25
s8:
 function: transformation
 connections:
  input1:
   node: s7
   id: output
 params:
  absolute:
   offset: [0.38, 0.54]
   matrix22: [-1.0, 0.0, 0.0, 1.0]
s9:
 function: blend
 connections:
  destination:
   node: s7
   id: output
  source:
   node: s8
   id: output
 params:
  absolute:
   blendingmode: MAX
s10:
 function: safe_transform_grayscale
 dependency: sbs://safe_transform.sbs
 connections:
  input:
   node: s9
   id: output
 params:
  absolute:
   rotation: 0.25
   tile: uU_vV
s11:
 function: blend
 connections:
  destination:
   node: s2
   id: output
  source:
   node: s10
   id: output
 params:
  absolute:
   blendingmode: ADD
   opacitymult: 0.2
   format: 1
  parent:
   outputsize: [0, 0]
s12:
 function: levels
 connections:
  input1:
   node: s10
   id: output
 params:
  absolute:
   levelinlow: [0.02, 0.02, 0.02, 0.0]
   levelinhigh: [0.95, 0.95, 0.95, 1.0]
   leveloutlow: [1.0, 1.0, 1.0, 1.0]
   levelouthigh: [0.0, 0.0, 0.0, 0.0]
   levelinmid: [0.41, 0.41, 0.41, 0.5]
s13:
 function: highpass_grayscale
 dependency: sbs://highpass.sbs
 connections:
  Source:
   node: s10
   id: output
 params:
  absolute:
   Radius: 0.1
s14:
 function: normal
 connections:
  input1:
   node: s11
   id: output
 params:
  absolute:
   intensity:
    f0:
     function: get_float1
     params:
      get_float1: normal_intensity
    f1:
     function: const_float1
     params:
      const_float1: 3.0
    f2:
     function: mul
     connections:
      a: f0
      b: f1
   inversedy:
    f0:
     function: get_integer1
     params:
      get_integer1: normal_format
    f1:
     function: const_int1
     params:
      const_int1: 1
    f2:
     function: eq
     connections:
      a: f0
      b: f1
   input2alpha: false
s15:
 function: histogram_range
 dependency: sbs://histogram_range.sbs
 connections:
  input:
   node: s11
   id: output
 params:
  absolute:
   range:
    f0:
     function: get_float1
     params:
      get_float1: height_range
   position:
    f0:
     function: get_float1
     params:
      get_float1: height_position
s16:
 function: blend
 connections:
  destination:
   node: s1
   id: output
  source:
   node: s12
   id: output
 params:
  absolute:
   blendingmode: SCREEN
   opacitymult: 0.15
  parent:
   outputsize: [0, 0]
s17:
 function: levels
 connections:
  input1:
   node: s13
   id: Highpass
 params:
  absolute:
   levelinlow: [0.33, 0.33, 0.33, 0.0]
   levelinhigh: [0.61, 0.61, 0.61, 1.0]
   leveloutlow: [1.0, 1.0, 1.0, 1.0]
   levelouthigh: [0.0, 0.0, 0.0, 0.0]
s18:
 outputs:
  normal: RGBA
 connections:
  inputNodeOutput:
   node: s14
   id: output
s19:
 outputs:
  height: RGBA
 connections:
  inputNodeOutput:
   node: s15
   id: output
s20:
 outputs:
  roughness: RGBA
 connections:
  inputNodeOutput:
   node: s16
   id: output
s21:
 function: blend
 connections:
  destination:
   node: s3
   id: output
  opacity:
   node: s17
   id: output
  source:
   node: s3
   id: output
 params:
  absolute:
   blendingmode: MULTIPLY
   opacitymult: 0.35
  parent:
   outputsize: [0, 0]
s22:
 function: hsl
 connections:
  input1:
   node: s21
   id: output
 params:
  parent:
   hue:
    f0:
     function: get_float1
     params:
      get_float1: hue_shift
    f1:
     function: const_float1
     params:
      const_float1: 0.5
    f2:
     function: mul
     connections:
      a: f0
      b: f1
   saturation:
    f0:
     function: get_float1
     params:
      get_float1: saturation
   luminosity:
    f0:
     function: get_float1
     params:
      get_float1: luminosity
s23:
 function: levels
 connections:
  input1:
   node: s22
   id: output
 params:
  parent:
   levelinlow:
    f0:
     function: const_float1
    f1:
     function: const_float1
     params:
      const_float1: 0.5
    f2:
     function: get_float1
     params:
      get_float1: contrast
    f3:
     function: max
     connections:
      a: f2
      b: f0
    f4:
     function: mul
     connections:
      a: f3
      b: f1
    f5:
     function: vector2
     connections:
      componentsin: f4
      componentslast: f4
    f6:
     function: vector2
     connections:
      componentsin: f4
      componentslast: f0
    f7:
     function: vector4
     connections:
      componentsin: f5
      componentslast: f6
   levelinhigh:
    f0:
     function: const_float1
     params:
      const_float1: 1.0
    f1:
     function: const_float1
     params:
      const_float1: 0.5
    f2:
     function: const_float1
    f3:
     function: get_float1
     params:
      get_float1: contrast
    f4:
     function: max
     connections:
      a: f3
      b: f2
    f5:
     function: mul
     connections:
      a: f4
      b: f1
    f6:
     function: sub
     connections:
      a: f0
      b: f5
    f7:
     function: vector2
     connections:
      componentsin: f6
      componentslast: f6
    f8:
     function: vector2
     connections:
      componentsin: f6
      componentslast: f0
    f9:
     function: vector4
     connections:
      componentsin: f7
      componentslast: f8
   leveloutlow:
    f0:
     function: const_float1
    f1:
     function: const_float1
     params:
      const_float1: 0.5
    f2:
     function: get_float1
     params:
      get_float1: contrast
    f3:
     function: min
     connections:
      a: f2
      b: f0
    f4:
     function: abs
     connections:
      a: f3
    f5:
     function: mul
     connections:
      a: f4
      b: f1
    f6:
     function: vector2
     connections:
      componentsin: f5
      componentslast: f5
    f7:
     function: vector2
     connections:
      componentsin: f5
      componentslast: f0
    f8:
     function: vector4
     connections:
      componentsin: f6
      componentslast: f7
   levelouthigh:
    f0:
     function: const_float1
     params:
      const_float1: 1.0
    f1:
     function: const_float1
     params:
      const_float1: 0.5
    f2:
     function: const_float1
    f3:
     function: get_float1
     params:
      get_float1: contrast
    f4:
     function: min
     connections:
      a: f3
      b: f2
    f5:
     function: abs
     connections:
      a: f4
    f6:
     function: mul
     connections:
      a: f5
      b: f1
    f7:
     function: sub
     connections:
      a: f0
      b: f6
    f8:
     function: vector2
     connections:
      componentsin: f7
      componentslast: f7
    f9:
     function: vector2
     connections:
      componentsin: f7
      componentslast: f0
    f10:
     function: vector4
     connections:
      componentsin: f8
      componentslast: f9
s24:
 outputs:
  baseColor: RGBA
 connections:
  inputNodeOutput:
   node: s23
   id: output
```

Figure 8: Complete example of a graph in CompactSBS format. This listing shows the full representation of the material partially illustrated in Figure 3.