

Reflecting Process Expertise in Procedural Material Generation

Kunal Gupta, Gaurav Joshi, Yen-Ru Chen, Seemandhar Jain,
Ishit Mehta and Manmohan Chandraker

University of California San Diego, La Jolla, CA, USA

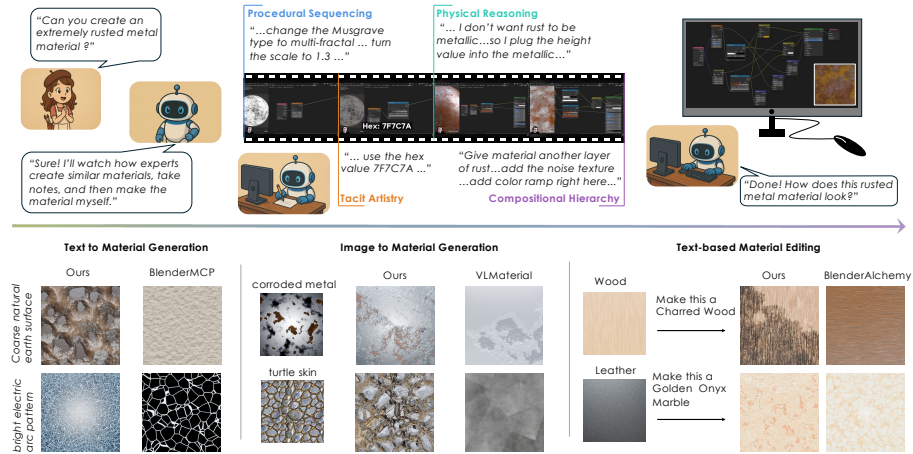


Fig. 1: Retrieval-time process reasoning for procedural materials. (Top) Rather than synthesizing material graphs directly, our approach reasons over *process traces* derived from expert demonstrations (e.g., tutorial videos), capturing procedural sequencing, physical intuition, and compositional design strategies used by artists. (Bottom) This process-centric reasoning enables diverse material creation tasks, including text-to-material generation (left), image-conditioned synthesis (center), and text-based editing (right). Across tasks, materials produced through process reasoning align more closely with user intent and exhibit stronger visual fidelity than prior graph-only methods such as BlenderMCP [2], VLMaterial [15], and BlenderAlchemy [11].

Abstract. Procedural material creation underpins applications in digital content creation, visual effects, and 3D asset design. Achieving high-quality results requires more than reproducing node graphs—it demands understanding the process by which experts construct materials. We formulate procedural material generation as retrieval-time process reasoning over expert demonstrations, elevating process to a first-class representation beyond graph-only synthesis. Concretely, we represent expert workflows as *process traces*: textual records of construction steps, parameters, and design intent. To instantiate this idea, we use a pretrained LLM-based PROCESSSYNTHESIZER to synthesize a process trace aligned with a user’s intent and a pretrained LLM-based COMPILER to ground

the process trace into an executable Blender material graph. Because procedural expertise is most naturally conveyed through demonstrations, we leverage tutorial videos as a source of process knowledge and extract textual, LLM-compatible traces using automated video analysis tools. In an expert study with five Blender artists (avg. 7.5 years of experience), materials generated by reflecting expert demonstrations were found to produce workflows requiring fewer edits, and more closely match professional design strategies than methods operating solely on static artifacts. A user study with 150 participants further shows that our approach achieves superior generation and editing performance compared to prior procedural systems. All code, models, and data will be available at <https://materialapprentice.github.io>.

Keywords: Procedural Generation · Materials · Process Modeling

1 Introduction

Procedural materials power the visual surfaces of films, games, product design, and virtual worlds by encoding appearance as parameterized node graphs rather than fixed bitmaps. Their flexibility makes them essential for scalable, editable, and physically consistent asset creation. Yet crafting these materials remains specialist work: experienced artists rely on visual intuition, procedural reasoning, and domain-specific workflows. Both aspiring and established creators continually refine their craft by observing how experts layer textures, tune parameters, and reason about physical effects. Such demonstrations convey process knowledge that extends far beyond the final graph. Figure 1 illustrates our central premise: material generation should reflect the process expertise artists use when constructing procedural materials. Current methods, even when trained on large procedural material datasets, operate in isolation at inference time: they generate materials but do not incorporate such expertise into the generation.

If process knowledge is central to high-quality material design, how can it be reflected automatically during generation? We address this by framing *procedural material generation as retrieval-time process reasoning*. We introduce MATERIALAPPRENTICE, a multimodal framework that integrates expert-derived process traces into the generative loop. Rather than operating solely in graph space, MATERIALAPPRENTICE retrieves relevant traces, synthesizes a target trace aligned with the user’s intent, and grounds it into an executable material graph. As illustrated in Fig. 2, MATERIALAPPRENTICE distinguishes three representations: a *process trace*, which records construction steps, parameters, and design intent; a *material graph*, the executable shader-node layout compiled from the trace; and a *procedural material*, the resulting editable shader asset. Treating process traces as a first-class representation provides a foundation for higher-quality, more editable material generation.

Expert procedural knowledge is typically acquired by observing skilled artists as they construct materials. In practice, this expertise is often externalized through tutorial videos, which provide a temporally ordered and multimodal record of the creation process: the evolving node graph, parameter adjustments, synchronized

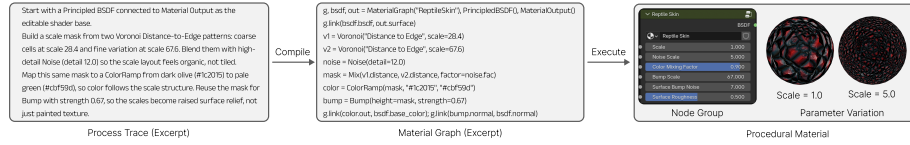


Fig. 2: Three representations used in MATERIALAPPRENTICE. A *process trace* gives artist-like construction steps with concrete parameters and design intent. The trace is compiled into a *material graph*, an executable shader structure. Executing the graph instantiates a *procedural material* with editable controls and rendered appearances.

narration explaining intent, and visual focus on individual operations. These signals indicate not only the final artifact but the reasoning trajectory behind it.

In our setting, we approximate this observational learning by analyzing Blender tutorial videos and extracting structured, agent-friendly process traces using automated video analysis tools. Rather than forming a training dataset, these traces serve as demonstrations retrieved at inference time. In practice, only a small number of relevant exemplars is sufficient to guide synthesis, forming a lightweight knowledge base for retrieval-time reasoning.

Technically, we instantiate retrieval-time process reasoning through a retrieval-augmented large language model (LLM) framework. Given a user query, MATERIALAPPRENTICE retrieves relevant process traces and conditions a pretrained LLM-based PROCESSSYNTHESIZER to synthesize a process trace aligned with the intended material. This trace is then grounded into an executable representation with a pretrained LLM-based COMPILER (differing only in prompting), which translates it into a structured material graph implemented using Blender shader nodes. This design separates reasoning from execution while integrating process-level conditioning directly into the generative loop. Importantly, beyond one-shot synthesis, MATERIALAPPRENTICE can also *demonstrate the process*, producing step-by-step constructions that mirror expert workflows and support structured editing. In contrast, prior systems such as VLMaterial [15], BlenderAlchemy [11], and Blender-MCP [2] are limited to operation in graph space.

We evaluate MATERIALAPPRENTICE on a suite of Blender material creation tasks, including multimodal material generation and text-driven editing. Across quantitative metrics and perceptual studies, MATERIALAPPRENTICE consistently outperforms VLMaterial [15], BlenderAlchemy [11], and Blender-MCP [2]. In addition, a study with experienced Blender artists shows that process traces generated by MATERIALAPPRENTICE more closely follow expert constructions and require fewer manual edits than alternatives.

In summary, we frame procedural material generation as retrieval-time process reasoning over expert demonstrations. Our contributions are:

- A formulation of procedural material generation as retrieval-time process reasoning, elevating process to a first-class representation beyond graphs.
- A retrieval-augmented architecture with an explicit process-trace representation and compiler that separate reasoning from execution and ground generated traces into executable Blender shader graphs.

- A curated corpus of Blender tutorial videos and video analysis tools for extracting agent-friendly process traces that enable inference-time conditioning.
- Comprehensive evaluation, including with Blender artists, showing process-based synthesis better aligns with expert workflows than graph-based systems.

2 Related Work

Generative Materials Recent work directly synthesizes material appearances. MaterialGAN [8] extends StyleGAN2 [13] to generate SVBRDF textures, but operates in an opaque latent space. Diffusion-based methods improve visual quality: ControlMat [19] uses ControlNet-guided diffusion for SVBRDF generation with limited structural control, DreamMat [24] maps text prompts to point-wise BRDF parameters in a hash grid, and MaterialPicker [16] fine-tunes a video generator for text- and image-conditioned material textures. Huang et al. [12] generate textures directly in UV space, improving surface consistency. However, these methods produce non-procedural, sample-based or latent representations and do not expose editable procedural graphs.

Generative Procedural Materials Closest to our work are methods that synthesize procedural material graphs. Hu et al. [9] generate node-based materials conditioned on text prompts, example images, or partial node graphs, using node graphs as defined in Adobe Substance 3D [1] and the Matformer representation [7]. Li et al. [14] build on these ideas, using synthetic data to train a transformer model for material graph generation and applying RL post-training on both real and synthetic data to improve realism and stability. While these methods produce procedural representations that are more editable and interpretable than purely texture-based outputs, they rely on large-scale training and custom models.

LLMs for Program Synthesis Large language models (LLMs) have demonstrated strong capabilities in program synthesis [4], motivating their use as engines for 3D content creation. Early systems directly generate Python code for Blender, such as 3DGPT [17]. However, operating in a general-purpose, low-level language makes it difficult to maintain structure, guarantee validity, and scale to complex scenes. To address these limitations, several works introduce domain-specific languages (DSLs) tailored for 3D generation. Holodeck [21] and L3GO [20] use LLMs to author scenes in compact DSLs, simplifying high-level planning and constraint reasoning. While DSLs can make LLM outputs more structured, they typically lack large-scale training corpus, which can lead to unexpected behavior [3].

LLMs as Planners Another line of work treats LLMs as planners that orchestrate external tools for 3D creation. LayoutGPT [6] uses LLMs to generate layouts, while SceneCraft [10] and LayoutVLM [18] adopt an analysis-by-synthesis loop in which a vision-language critic evaluates rendered scenes and guides refinement. When combined with tool-use frameworks like ReAct [23], LLM agents can scale to interactive, photorealistic environments [22].

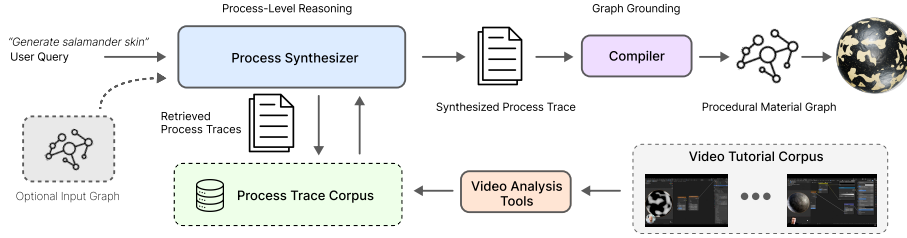


Fig. 3: Overview: Given a user query and optionally an input graph, the PROCESSSYNTHESIZER retrieves relevant process traces from a corpus of expert traces and synthesizes a target process trace aligned with the query. These traces are extracted automatically from tutorial videos using video analysis tools that convert demonstrations into textual records of construction steps, parameters, and design intent. The COMPILER then grounds the synthesized trace into an executable Blender procedural material graph.

VLMs for Materials LLMs and vision-language models have also been applied directly to material synthesis. VLMaterial [15] fine-tunes a VLM to generate procedural materials represented as node graphs and BlenderAlchemy [11] iteratively samples and refines node graphs using VLM-based evaluation of rendered outputs, demonstrating that large models can reason about procedural material programs and visual feedback. In contrast, our method is training-free, leverages process-trace reasoning, and uses in-context learning from expert material generation workflows to achieve high-quality procedural material synthesis and editing.

3 Method

Procedural material creation is not merely graph assembly, but the execution of a structured reasoning trajectory. Skilled artists iteratively refine structure, adjust parameters, and articulate intent; the final node graph is only the endpoint of this process. We therefore distinguish the three representations used by MATERIALAPPRENTICE: a *process trace*, which records construction steps, parameters, and design intent; a *material graph*, the executable shader-node layout compiled from the trace; and a *procedural material*, the resulting editable shader asset. While prior systems operate directly in graph space, we perform generation in process space.

To operationalize this formulation, MATERIALAPPRENTICE instantiates retrieval-time process reasoning. As summarized in Figure 3, given a user query, it retrieves relevant process traces and synthesizes a process trace via PROCESSSYNTHESIZER aligned with the intended material. This trace is then grounded into an executable Blender shader graph through the COMPILER, explicitly separating reasoning from execution. Process traces are derived from expert video demonstrations using video analysis tools which convert raw tutorials into structured, agent-friendly textual representations. These traces form the knowledge base for retrieval-time reasoning. Figure 4 details the PROCESSSYNTHESIZER, DECOMPILER, and video analysis tools, which are described in the following subsections.

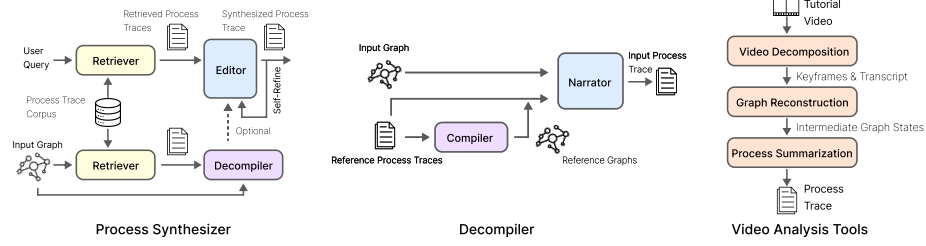


Fig. 4: Components of MATERIALAPPRENTICE. (Left) **PROCESSSYNTHESIZER**: Given a user query, the retriever selects relevant process traces, which condition the EDITOR to synthesize a target trace. For editing, an input graph is first converted into a process trace by the **DECOMPILER**. (Middle) **DECOMPILER**: An input material graph is converted into a process trace using exemplar graph–trace pairs; the **NARRATOR** matches graph substructures to similar patterns in the exemplars and transfers their associated construction steps. (Right) **Video analysis tools**: Tutorials are decomposed into keyframes and transcripts, intermediate graph states are reconstructed, and multimodal signals are summarized into process traces for the **PROCESSSYNTHESIZER**.

3.1 Process Synthesizer

The **PROCESSSYNTHESIZER** is the core reasoning module of **MATERIALAPPRENTICE**, synthesizing process traces aligned with user intent. Given a query q , a **RETRIEVER** selects relevant reference traces P from the demonstration corpus C_P . Conditioned on q and P , the **EDITOR** synthesizes a target trace p describing how to construct the desired material.

For editing, **PROCESSSYNTHESIZER** additionally accepts an input graph g_i . In this setting, the **RETRIEVER** selects process traces relevant to g_i , and a **DECOMPILER** module reconstructs a corresponding process trace p_i . This inferred process trace, together with the user’s edit instruction, conditions the **EDITOR** to synthesize an updated process trace consistent with the requested modification.

Retriever. The **RETRIEVER** selects relevant demonstrations for both generation and editing. Rather than ranking examples by surface appearance alone, it uses the pretrained LLM to compare how materials are constructed. Given material names and descriptions in C_P , it predicts which expert demonstrations best match the user’s intent, desired material properties, and construction strategy. The top- K corresponding process traces are selected to condition synthesis.

Editor. The **EDITOR** is a pretrained LLM prompted to perform in-context synthesis in the *process* domain. Conditioned on P and q , it generates a target process trace p specifying the construction sequence, parameter choices, and compositional structure required to realize the requested material. As illustrated in Fig. 5, the **EDITOR** combines construction patterns from retrieved traces into a target trace before compilation. Here, lowercase p denotes a single trace, while P refers to sets of retrieved process traces from C_P . To improve reliability, the **EDITOR** employs a lightweight self-refinement step, critiquing and revising its draft before finalizing p , enhancing consistency and adherence to the query. Because the

explicit parameter explanation, and structured workflow progression). It spans 29 material categories (e.g., metals, fabrics, organic surfaces), with an average of 5.5 tutorials for each. Tutorials average 12.6 minutes in duration, with materials containing on average 14.3 shader nodes, providing rich signals about compositional structure, parameter tuning, and procedural reasoning.

In addition, we create 10 original tutorials for which we hold full rights and will release publicly to ensure long-term reproducibility. Publicly sourced tutorials are used exclusively as a retrieval-time knowledge source; no model training is performed on raw video content. We release YouTube IDs and category annotations to enable reconstruction of the corpus while preserving attribution to original creators. Detailed statistics are provided in the supplementary.

3.4 Process Extraction via Video Analysis Tools

To derive process traces from expert demonstrations, MATERIALAPPRENTICE uses automatic video analysis tools to convert tutorials into structured text that LLM agents can directly retrieve and reason over. This component is primarily an engineering step rather than a core research contribution: the goal is simply to automatically extract procedural signals from demonstrations so that they can be used for retrieval-time reasoning. In practice, Blender tutorials exhibit a simple box-and-wire interface, allowing standard detection and segmentation models to reliably recover graph structure. Given a tutorial video v , the pipeline extracts keyframes, reconstructs intermediate graph states, and aligns visual edits with narrated explanations.

Video Decomposition. Videos are first decomposed into frames and audio narration. Informative keyframes are selected to capture meaningful transitions in the material construction process while removing redundant views. The audio stream is transcribed to preserve the artist’s verbal explanations and design rationale.

Graph Reconstruction. The visual stream is analyzed to recover the evolving node graph. From selected keyframes, the system identifies node layouts, connectivity patterns, and parameter states using standard detection and segmentation models fine-tuned for the Blender shader editor. This produces a sequence of intermediate graph representations reflecting the progressive assembly of the material.

Process Summarization. A SUMMARIZER module integrates the reconstructed graph states with the transcribed narration to produce a compact textual process trace p_v , capturing both the sequence of construction steps and the accompanying procedural reasoning expressed by the artist. These traces populate the retrieval corpus used by the PROCESSSYNTHESIZER at inference time.

Robustness of Process Extraction. Three design choices make extraction reliable. (1) *Grounding*: frame understanding relies on explicit vision tasks – node, socket, and wire detection – rather than direct LLM inference. Given the structured box-and-wire layout of the Blender shader editor, these detectors achieve $\text{mAP}_{50} = 0.98$ and $\text{mIoU} = 0.87$. (2) *Selection*: a fine-tuned frame-selection model retains only visually parseable frames robust to zoom, pan, and occlusion. (3) *Redundancy*: frames are parsed at 0.2 Hz and combined with narration through

an LLM summarizer, mitigating localized perception errors and producing stable process traces. Further, while the pipeline contains multiple stages, errors in early stages are mitigated by the process abstraction. The `PROCESSSYNTHESIZER` reasons over construction patterns rather than exact node configurations, allowing it to tolerate minor inaccuracies in reconstructed graph states or narration alignment. In practice, we observe that small parsing errors rarely propagate to catastrophic failures in the final material graph. Further details are provided in the supplementary.

3.5 Implementation Details

All LLM-based components—`RETRIEVER`, `EDITOR`, `DECOMPILER`, `COMPILER`, and `SUMMARIZER`—use the same pretrained GPT-5 model, differing only in prompt structure and contextual conditioning. The `RETRIEVER` selects the top $K = 3$ process traces per query, balancing contextual coverage and computational efficiency. The compiler generates $N = 10$ programs. While processes synthesized by the `PROCESSSYNTHESIZER` are typically coherent and faithful to the query, final material quality may be further refined through established procedural optimization techniques [11]. A single process trace is $\approx 8k$ tokens, a single run costs $\approx 2\text{USD}$ and runs for 30 mins.

4 Experiments

Our experiments probe the implications of viewing procedural material generation as *retrieval-time process reasoning*. Section 4.1 evaluates the key design choices implied by this formulation—reasoning in process space rather than graph space and grounding execution through the `COMPILER`. Section 4.2 then situates `MATERIALAPPRENTICE` within existing procedural material systems through comparisons on established generation and editing benchmarks. Finally, Section 4.3 presents ablations analyzing generalization, retrieval behavior, the impact of video analysis tools, and the reproducibility of our framework through experiments on in-house tutorial videos.

4.1 Reasoning in the Process Domain

To evaluate process-space reasoning, we isolate the representation used for retrieval while keeping the output modality fixed. Our method retrieves *process traces*, while the baseline retrieves *material graphs*; both variants synthesize process traces that artists follow to construct and refine material graphs. This isolates the effect of structured procedural knowledge relative to raw graph structure.

Evaluation setup. We recruit five Blender artists (avg. 7.5 years of procedural-material experience) and evaluate 60 synthesized process traces from 30 prompts under two retrieval conditions. For each prompt, artists see anonymized, randomly ordered process traces from the two variants. The study has two stages: artists first construct a material graph by following each trace, then refine the graph for up to ten minutes to better match the input prompt. We record atomic graph

Edit Effort			Expert Preference		Editing Sentiment	
Metric	Proc.	Graph	Criterion	Pref.	Signal	$\Delta(\text{P-G})$
Node edits ↓	2.42	3.42	Impl. clarity ↑	100%	Confusion ↓	-0.15
Conn. edits ↓	5.42	7.42	Proc. strategy ↑	83%	Hesitation ↓	-0.10
Param. tweaks ↓	5.75	4.83	Outcome quality ↑	92%	Satisfaction ↑	+0.40
Struct. edits ↓	7.84	10.84	Param. control ↑	83%	Completion ↑	+0.25
Total edits ↓	13.58	15.67	Pedagogy ↑	84%		
			Final render ↑	83%		

Table 1: Expert Study. (Left) Average refinement edits. (Middle) LLM-assigned preferences from artists’ think-aloud commentary; final render preference is provided by artists after refinement. (Right) Process–Graph sentiment deltas after mapping low/medium/high to 1/2/3. Process-derived graphs require fewer structural/total edits, are preferred on key criteria, and reduce editing friction.

edits—node edits, connection edits, and parameter refinements—as a direct measure of corrective editing effort.

Artists also follow a think-aloud protocol [5] during construction and refinement, verbalizing their understanding of each trace, the rationale behind their edits, and their assessment of the resulting graph. We ask them to comment on implementation clarity, procedural strategy, expected outcome quality, parameter control, and pedagogical value. An LLM then analyzes the recorded narrations with a fixed rubric to assign structured pairwise preferences and editing-friction labels. Final render preference is collected separately after refinement. Table 1 summarizes the study along three axes: edit effort, expert preference, and think-aloud sentiment. We provide the GPT-5 analysis prompts in the supplement.

Editing effort. The left block of Table 1 reports edits required to refine the generated graphs. Node and connection edits modify graph structure and therefore reflect changes to the underlying procedural strategy, while parameter refinements tune existing controls such as sliders and color ramps. Process-conditioned traces require fewer structural edits than graph-conditioned traces (7.84 vs. 10.84) and fewer total edits overall (13.58 vs. 15.67), indicating that they produce graphs closer to the intended procedural structure. They require slightly more parameter tweaks (5.75 vs. 4.83), which we interpret as a favorable shift: artists spend less effort repairing structure and more effort tuning exposed controls.

Expert preference. The middle block reports LLM-assigned preferences from artists’ think-aloud commentary; final render preference is provided directly by artists after refinement. Process-conditioned workflows are favored across all criteria, especially implementation clarity (100%) and outcome quality (92%), with strong preferences for procedural strategy (83%), parameter control (83%), pedagogical value (84%), and final render quality (83%). This indicates that process traces provide clearer construction guidance beyond final appearance.

Editing sentiment. The right block analyzes editing friction from think-aloud narrations. An LLM assigns low/medium/high labels to four signals: *confusion* and *hesitation*, which indicate difficulty understanding the graph or choosing the

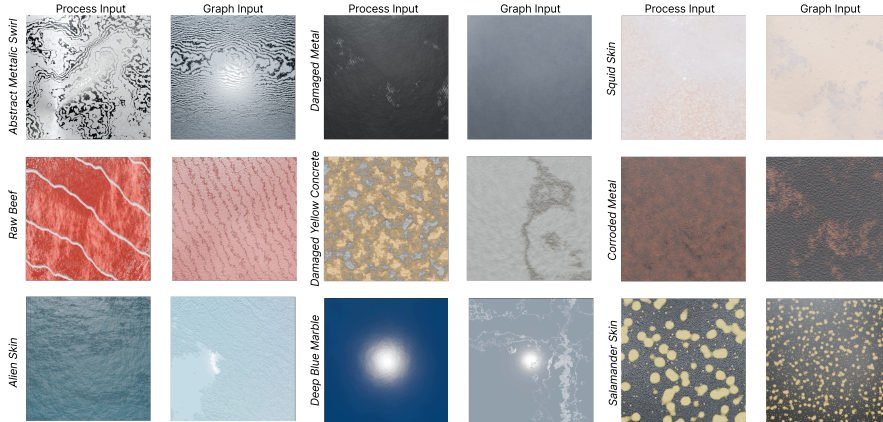


Fig. 6: Qualitative comparison of fully automatic generation using process-conditioned and graph-conditioned retrieval. Each pair shows outputs from the complete pipeline without manual editing. Process-conditioned generation better captures the intended procedural structure and visual characteristics, while graph-conditioned retrieval often produces oversimplified or structurally inconsistent patterns.

next edit, and *satisfaction* and *completion*, which indicate successful refinement and confidence in the result. After mapping labels to 1/2/3, process-conditioned workflows reduce confusion (-0.15) and hesitation (-0.10), while increasing satisfaction ($+0.40$) and completion ($+0.25$). This suggests that process-derived graphs are easier for artists to understand, modify, and finalize.

Fully automatic generation. Figure 6 evaluates the complete pipeline without manual editing. For 50 diverse material prompts, we compare paired outputs from process-conditioned and graph-conditioned retrieval. Three experts judge each pair against the input prompt, with majority vote determining preference. Process-conditioned outputs are preferred in 72% of comparisons, showing that process retrieval improves not only artist-facing editability but also automatic prompt alignment. Qualitatively, process-conditioned retrieval better captures hierarchical procedural structure, including directional fibers, layered corrosion, and structured surface noise, whereas graph-conditioned retrieval often yields simpler or structurally incorrect patterns.

Need for Compiler. We evaluate a variant where the EDITOR directly generates material graphs instead of processes. We measure reliability by *execution rate*: the fraction of prompts for which at least one candidate program runs successfully in Blender. Across 100 prompts, direct graph generation fails in 30% of cases, whereas our full process-to-compiler pipeline achieves a 100% execution rate.

4.2 Comparison with Prior Procedural Material Systems

We compare MATERIALAPPRENTICE against three state-of-the-art procedural material systems: BlenderAlchemy [11], BlenderMCP [2], and VLMaterial [15]. We also report a Nearest-Neighbor baseline (the closest retrieved process) to verify

Image-to-Material Generation							Text-based Material Editing				
Dataset	Method	CLIP ↓	Style Loss ↓	SWD ↓	GPT-5 Pref. ↑	User Pref. (%) ↑	Dataset	Method	CLIP ↓	GPT-5 Pref. ↑	User Pref. (%) ↑
BlenderKit	VLMaterial	0.856	0.040	2.44	0.660	80	Coarse	BlenderAlchemy	0.244	0.611	85
	BlenderMCP	0.781	0.061	4.428	0.835	86		BlenderMCP	0.237	0.611	80
	Nearest Neighbour	0.791	0.059	4.326	0.874	84		Nearest Neighbour	0.238	0.722	100
	Ours	0.852	0.043	2.567	-	-		Ours	0.260	-	-
	VLMaterial	0.764	0.066	8.973	0.771	89		BlenderAlchemy	0.240	0.643	81
In the Wild	BlenderMCP	0.737	0.080	8.227	0.800	85	Fine-grained	BlenderMCP	0.215	0.929	100
	Nearest Neighbour	0.748	0.071	7.373	0.714	87		Nearest Neighbour	0.228	0.786	100
	Ours	0.768	0.065	5.949	-	-		Ours	0.253	-	-

Table 2: Quantitative comparison of image-to-material generation (Left) and text-based material editing (Right) across datasets. User preference (%) reports the fraction of comparisons where users preferred **Ours** over each baseline. GPT-5 and Users prefer our editing over baselines.

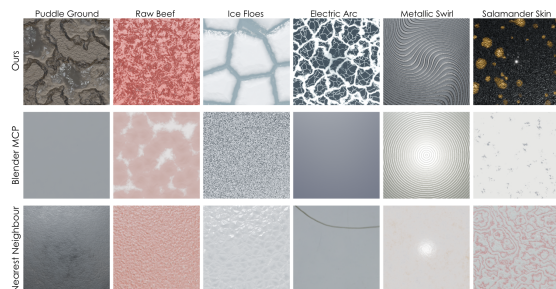


Fig. 7: Text-conditioned Material Generation. Our method produces visually diverse materials across a wide range of text prompts. Unlike BlenderMCP, our outputs align more closely with the user prompt while remaining distinct from the nearest retrieved example.

that MATERIALAPPRENTICE isn’t simply copying demonstrations. All baselines and MATERIALAPPRENTICE modules use a GPT-5 backbone for fairness.

For generation, perceptual similarity is measured following [15] using CLIP cosine similarity, style loss, and sliced Wasserstein distance (SWD). For editing, where no reference render exists, we follow [11] and compute CLIP similarity between the edit instruction and the generated output. We additionally conduct GPT-5 and user forced-choice studies: for generation, evaluators receive the target and two outputs; for editing, they receive the initial material, the edit instruction, and two candidate results. We recruit 150 users on Amazon MT, with comparisons rated by three annotators and aggregated by majority vote.

Procedural Material Generation. We evaluate all methods, excluding BlenderAlchemy (which does not support prompt-based generation), on two benchmarks: (i) 110 rendered materials from BlenderKit and (ii) 35 in-the-wild close-up photographs. Each prompt includes a short text description (e.g., “banana skin”) and a reference image. Figure 7 shows text-to-material generation results. MATERIALAPPRENTICE produces materials more closely aligned with textual prompts, achieving higher CLIP similarity (0.806 vs. 0.791 for nearest neighbor and 0.781 for BlenderMCP). Users prefer our results in 86% and 92% of comparisons against nearest neighbor and BlenderMCP, respectively. Table 2 reports quantitative results. MATERIALAPPRENTICE achieves the strongest performance on the challenging in-the-wild benchmark and is competitive with VLMaterial on BlenderKit, where VLMaterial is fine-tuned. CLIP, Style Loss, and SWD capture image-level similarity and are sensitive to scale, alignment, and rendering, while GPT-5 and user preferences reflect prompt faithfulness and procedural plausibility; these metrics therefore provide complementary views of quality. Across both preference studies, MATERIALAPPRENTICE is consistently favored over competing methods.

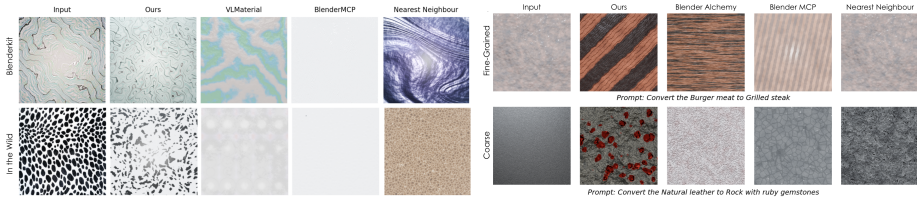


Fig. 8: Image conditioned Material Generation. (Left) Images include nacre and dalmatian material. Our method better captures micro and macro details specified via the image reference. **Text conditioned Material Editing.** (Right) Our method enables more precise and reliable manipulation of material graphs. For coarse edits, it makes the substantial structural changes required to achieve the intended transformation, while for fine edits, it can introduce subtle, localized modifications—producing nuanced effects that competing baselines fail to capture.

Figure 8 (left) shows qualitative comparisons. MATERIALAPPRENTICE reproduces both macro- and micro-scale structures—from nacre patterns to dalmatian textures—while VLMaterial often produces coarse approximations and BlenderMCP drifts from the target appearance. Our outputs remain clearly distinct from nearest-neighbor examples, indicating genuine synthesis rather than retrieval.

Procedural Material Editing. We evaluate BlenderAlchemy [11], BlenderMCP [2], and MATERIALAPPRENTICE on two editing tasks: 25 coarse edits (e.g., wood → marble) and 25 fine-grained edits (e.g., adding moss to brick), using base materials from BlenderKit. Coarse edits require substantial restructuring of the shader graph, while fine edits involve localized modifications.

As shown in Table 2, GPT-5 consistently prefers edits produced by MATERIALAPPRENTICE over BlenderMCP and nearest-neighbor baselines. Qualitative examples (Fig. 8, right) show that baselines often fail to incorporate key prompt attributes—for example, missing structural features—while MATERIALAPPRENTICE introduces both large-scale structural changes and subtle surface details.

4.3 Ablations

We now analyze several aspects of MATERIALAPPRENTICE including its ability to generalize beyond the materials present in the process corpus and the effect of the number of retrieved demonstrations used during synthesis.

Generalization beyond the process corpus. To test whether MATERIALAPPRENTICE relies on category-level cues from the corpus, we remove entire material categories corresponding to the evaluation prompts—Organic (blue jay egg), Concrete (cracked concrete), and Marble (midnight gold marble). Figure 9 shows that even under this setting, MATERIALAPPRENTICE continues to synthesize plausible procedural structures that capture the essential characteristics of the input images, indicating that the model generalizes procedural strategies rather than memorizing category-specific examples.

Effect of the number of retrieved demonstrations. We next vary the number of demonstrations provided as context during synthesis (Fig. 9). Even a single retrieved process trace substantially improves generation quality compared to using no demonstrations. However, the benefit saturates quickly, with limited

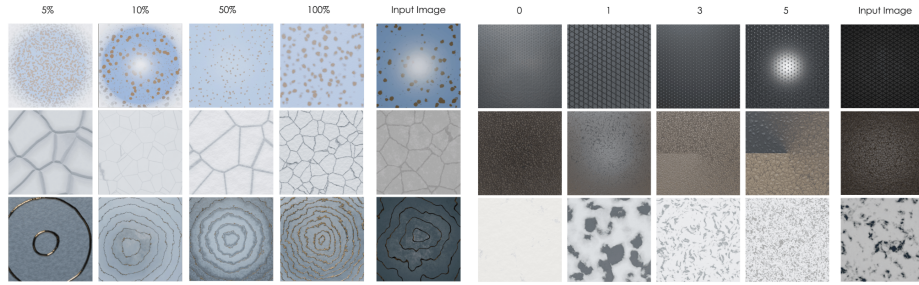


Fig. 9: Effect of Video Corpus Size on Generation Quality. (Left) Our retrieval-based approach maintains strong generation quality even when given only a small fraction of the tutorial corpus—evaluated at 5% to 100% availability. This demonstrates that MATERIALAPPRENTICE can synthesize novel, high-quality materials by reflecting on only a handful of expert demonstrations. **Effect of Number of Retrieved Demonstrations.** (Right) We evaluate MATERIALAPPRENTICE using 0 (no retrieval), 1, 3, and 5 retrieved demonstrations. Quality improves from 0 to 3 retrieved process traces, with the largest gain from the first example; beyond three, improvements are marginal relative to the token cost.

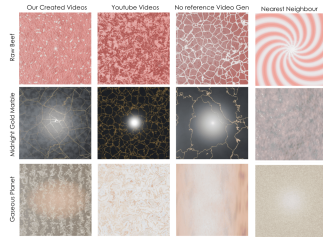
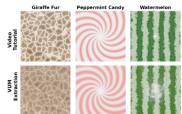


Fig. 10: Generation using our curated video set. Materials generated by MATERIALAPPRENTICE using only our released tutorial videos as references (left column). Results remain strong across diverse prompts and differ clearly from YouTube-based generation, generation without references, and the nearest neighbor baseline.

improvement beyond three retrieved examples, suggesting that a small number of relevant demonstrations is sufficient to guide effective process synthesis.

Reproducibility with a curated video set. To further demonstrate reproducibility, we evaluate MATERIALAPPRENTICE using a curated set of tutorial videos that we created ourselves and release publicly with full rights. Figure 10 shows text-conditioned generation results when the reference pool is restricted to this small set. Despite the limited corpus, MATERIALAPPRENTICE continues to synthesize diverse materials that differ clearly from both nearest-neighbor retrieval and generation without demonstrations, indicating that even a handful of shareable tutorials provides sufficient process-level cues for effective material synthesis.

Impact of Video Analysis Tools. We analyze the contribution of our video analysis pipeline by constructing three variants: (1) *Audio-only*, which ignores the visual stream and relies solely on narration; (2) *GPT-5 Frame Parsing*, which replaces our specialized detectors with GPT-5-based frame interpretation; and (3) *Random Keyframe Selection*, which removes our informativeness-based keyframe filtering. We evaluate all variants on 100 tutorial videos by reconstructing material graphs from extracted process traces and comparing them with ground-truth graphs. Node detection and topology reconstruction are evaluated using precision, recall, and F1 scores. Parameter fidelity is measured as the normalized absolute difference



Method	N-Prec $\uparrow$	N-Rec $\uparrow$	N-F1 $\uparrow$	T-Prec $\uparrow$	T-Rec $\uparrow$	T-F1 $\uparrow$	Params $\uparrow$	SWD $\downarrow$	Edits $\downarrow$
Ours (Full)	98.60	98.72	98.65	83.16	83.01	83.07	85.96	0.0304	2.66
Audio-Only	88.24	93.13	90.37	60.58	64.64	62.25	78.32	0.0528	4.02
GPT-5 Frame Parsing	77.98	83.42	79.99	53.98	59.74	56.23	69.84	0.1572	4.66
Random Keyframe Selection	89.21	91.77	90.21	62.58	65.51	63.71	76.92	0.0585	3.78

Fig. 11: Video analysis tools ablation. (Left) Materials reconstructed from VAT-extracted process traces closely match those recreated by humans from the original tutorials. (Right) Quantitative comparison with three ablations: audio-only extraction, GPT-5 frame parsing, and random keyframe selection. Our full pipeline achieves the best node accuracy, topology reconstruction, parameter fidelity, perceptual similarity, and lowest edit distance.

between predicted and ground-truth parameters, while perceptual similarity is computed using sliced Wasserstein distance (SWD). We additionally report graph edit distance, defined as the number of atomic operations required to transform the reconstructed graph into the ground-truth graph (node additions/removals, connection changes, and parameter adjustments), estimated by prompting an LLM to produce the minimal edit sequence. As shown in Fig. 11, our full pipeline consistently achieves the best performance across all metrics and produces graphs that most closely match the original materials. These results highlight the importance of jointly modeling visual structure, narration, and keyframe selection for reliable process extraction from tutorials.

5 Conclusion

Procedural material design is inherently process-driven: artists construct materials through layered operations, parameter tuning, and physical reasoning. We frame procedural material generation as retrieval-time process reasoning over expert demonstrations, elevating process to a first-class representation beyond graph-only synthesis. By retrieving process traces and grounding them into executable shader graphs, MATERIALAPPRENTICE produces materials that better reflect expert workflows while remaining interpretable and editable. More broadly, this perspective suggests generative systems that learn directly from creative practice rather than static artifacts alone. Extending process reasoning to multi-step design workflows, additional authoring tools beyond Blender, and larger corpora of demonstrations are promising directions for future work. Failure cases are discussed in the supplementary. Code, models, and data will be publicly released.

6 Acknowledgments

This work was supported in part by NSF grant 2402583 and gifts from Qualcomm, Google, and Adobe. We are grateful to the Blender experts who participated in our expert study: Barnabas Thomas, John Arquero, Benjamin Olawuni, Ricardo Raimundo, and Tanushree Roychoudhury. We also thank Krishna Chaitanya, Harsh Sinha, Kanav, and Diptimayee Gupta for their inspiration, support, and helpful discussions throughout this work.

References

1. Adobe: Substance designer (2023)
2. Ahuja, S.: Blender-mcp. <https://github.com/ahujasid/blender-mcp> (2025), accessed: 2025-11-06
3. Barbero Jiménez, Á.: An evaluation of llm code generation capabilities through graded exercises. arXiv preprint arXiv:2410.16292 (2024)
4. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Yedkar, Y.B., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. In: NeurIPS (2021)
5. Ericsson, K.A., Simon, H.A.: Verbal reports as data. *Psychological review* **87**(3), 215 (1980)
6. Feng, W., Zhu, W., Fu, T.j., Jampani, V., Akula, A., He, X., Basu, S., Wang, X.E., Wang, W.Y.: Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems* **36** (2024)
7. Guerrero, P., Hasan, M., Sunkavalli, K., Mech, R., Boubekeur, T., Mitra, N.: Matformer: A generative model for procedural materials. *ACM Transactions on Graphics* **41**(4), 46:1–46:12 (2022)
8. Guo, Y., Smith, C., Hašan, M., Sunkavalli, K., Zhao, S.: Materialgan: Reflectance capture using a generative svbrdf model. *ACM TOG* **39**(6), 254:1–254:13 (2020). <https://doi.org/10.1145/3414685.3417779>
9. Hu, Y., Guerrero, P., Hasan, M., Rushmeier, H., Deschaintre, V.: Generating procedural materials from text or image prompts. In: ACM SIGGRAPH 2023 conference proceedings. pp. 1–11 (2023)
10. Hu, Z., Iscen, A., Jain, A., Kipf, T., Yue, Y., Ross, D.A., Schmid, C., Fathi, A.: Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In: Forty-first International Conference on Machine Learning (2024)
11. Huang, I., Yang, G., Guibas, L.: Blenderalchemy: Editing 3d graphics with vision-language models. In: ECCV. pp. 297–314. Springer (2024)
12. Huang, X., Wang, T., Liu, Z., Wang, Q.: Material anything: Generating materials for any 3d object via diffusion. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 26556–26565 (2025)
13. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of StyleGAN. In: CVPR (2020)
14. Li, B., Hu, Y., Guerrero, P., Hasan, M., Shi, L., Deschaintre, V., Matusik, W.: Procedural material generation with reinforcement learning. *ACM TOG* **43**(6), 1–14 (2024)
15. Li, B., Wu, R., Solar-Lezama, A., Zheng, C., Shi, L., Bickel, B., Matusik, W.: Vlmaterial: Procedural material generation with large vision-language models. In: ICLR
16. Ma, X., Deschaintre, V., Hašan, M., Luan, F., Zhou, K., Wu, H., Hu, Y.: Materialpicker: Multi-modal dit-based material generation. *ACM Transactions on Graphics (TOG)* **44**(4), 1–12 (2025)
17. Sun, C., Sun, C., Chen, H.J.: 3d-gpt: Procedural 3d modeling with large language models. arXiv preprint arXiv:2310.12945 (2023)
18. Sun, F.Y., Liu, W., Gu, S., Lim, D., Bhat, G., Tombari, F., Li, M., Haber, N., Wu, J.: Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 29469–29478 (2025)

19. Vecchio, G., Martin, R., Roullier, A., Kaiser, A., Rouffet, R., Deschaintre, V., Boubekeur, T.: Controlmat: a controlled generative approach to material capture. *ACM Transactions on Graphics* **43**(5), 1–17 (2024)
20. Yamada, Y., Chandu, K., Lin, B.Y., Hessel, J., Yildirim, I., Choi, Y.: L3go: Language agents with chain-of-3d-thoughts for generating unconventional objects. In: *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (System Demonstrations)*. pp. 456–469. Association for Computational Linguistics (2025). <https://doi.org/10.18653/v1/2025.naacl-demo.37>
21. Yang, Y., Sun, F.Y., Weihs, L., Vanderbilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., Callison-Burch, C., Yatskar, M., Kembhavi, A., Clark, C.: Holodeck: Language guided generation of 3d embodied ai environments. In: *CVPR*. pp. 16227–16237 (2024)
22. Yao, K., Zhang, L., Yan, X., Zeng, Y., Zhang, Q., Yang, W., Xu, L., Gu, J., Yu, J.: Cast: Component-aligned 3d scene reconstruction from an rgb image. *ACM TOG* **44**(4), 1–19 (2025)
23. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022)
24. Zhang, Y., Liu, Y., Xie, Z., Yang, L., Liu, Z., Yang, M., Zhang, R., Kou, Q., Lin, C., Wang, W., et al.: Dreammat: High-quality pbr material generation with geometry-and light-aware diffusion models. *ACM Transactions on Graphics (TOG)* **43**(4), 1–18 (2024)