BLENDERLORE: LEARNING 3D CODING FROM INTERNET TUTORIAL VIDEOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models can create 3D assets through code, but still struggle with complex geometry, rigging, and simulation. Blender tutorial videos contain expert workflows that can help address these challenges, but these workflows are difficult to extract and reuse. We introduce **BlenderLore**, a dataset and pipeline for acquiring and reusing procedural knowledge from tutorial videos for 3D coding. We convert videos into illustrated tutorials, reconstruct assets in Blender Python, and extract reusable procedural knowledge. Adapting and combining methods across tutorials expands the collection to about 22K project instances associated with over 3K videos. We package the reconstruction workflow and procedural knowledge library as **BlenderLore Skill** for use in existing coding agents. We also introduce **BlenderLore-Bench**, a 100-task benchmark covering eight Blender capabilities. Our experiments show that BlenderLore Skill improves the scores of leading large language models and reduces their runtime. Expanding the procedural knowledge library further improves overall performance and time efficiency.

1 INTRODUCTION

3D coding turns a creative goal into an executable program and an editable 3D asset. Recent large language models (LLMs), including GPT-6 Astra (OpenAI, 2026b), have advanced rapidly on this task and are used in architecture, interior modeling, industrial design, and games (Yang et al., 2026b; Barkley et al., 2026; Chi et al., 2026). They can generate hard-surface geometry, low-poly objects, procedural materials, and scene layouts. Yet their abilities remain uneven: detailed organic shapes, complex local topology, character skinning, and natural deformation still require fine-grained edits and repeated visual adjustment that are difficult to express as reliable code.

Tutorial videos are worked examples of 3D creation: finished assets show what was achieved, while tutorials expose intermediate states, operations, and local decisions along the way. Worked examples can support the extraction of knowledge that applies beyond the original problem (Chi et al., 1989). Blender tutorials document demanding techniques, but their instructions are scattered across narration, frames, changing interface states, and brief edits to node graphs. Turning these demonstrations into executable code requires recovering both the creator's intent and the operations that produced each intermediate result. We introduce **BlenderLore** to learn from these tutorials.

Executable reconstruction bridges the gap between watching a tutorial and acquiring a method that an agent can use. BlenderLore converts videos into illustrated tutorials, then reconstructs the demonstrated assets with Blender Python. Executing the code and inspecting the resulting assets test whether the inferred operations reproduce the intended geometry, materials, and motion. Figure 1 illustrates this path from video evidence to an illustrated tutorial, an executable reconstruction, and reviewed procedural knowledge.

The value of a reconstructed tutorial lies in whether its method can be adapted and combined with methods from other tutorials to create something new. Successful program solutions can yield reusable abstractions for later problems (Ellis et al., 2021), and using familiar components in unfamiliar combinations is a demanding form of generalization (Lake & Baroni, 2018). BlenderLore reviews methods from reconstructions and organizes them in a **procedural knowledge library**, with each entry recording its source and scope of use. An agent can retrieve relevant entries, adapt them to a new goal, and combine them when no single tutorial provides a complete solution. Starting

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

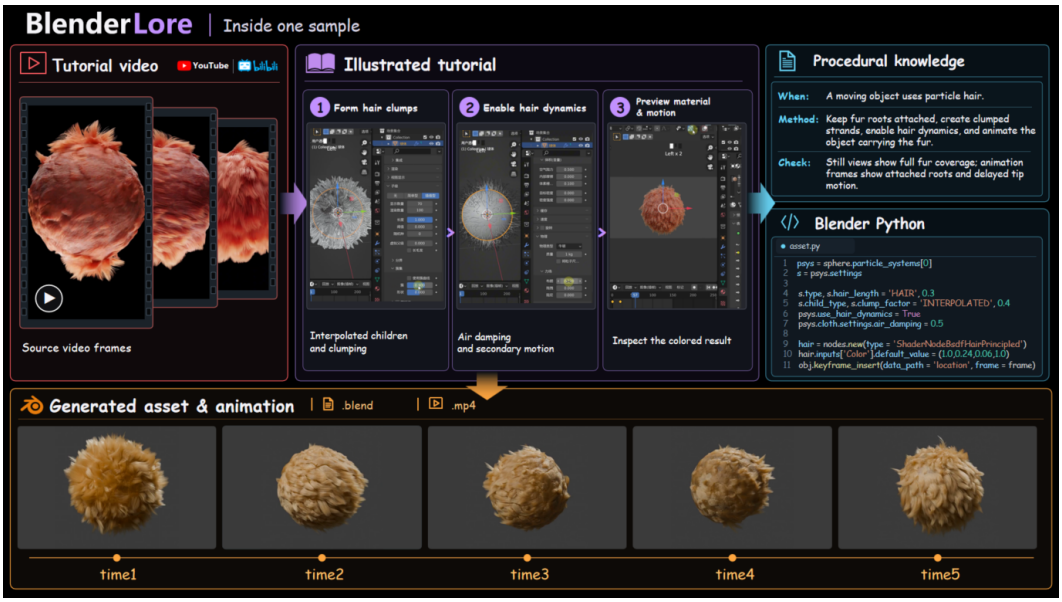


Figure 1: **A complete BlenderLore example: dynamic fur.** We extract illustrated steps from a tutorial, reconstruct the fur in Blender Python, and add the method to the procedural knowledge library.

from 8,918 collected or reconstructed projects, this reuse and combination expands BlenderLore to about 22K project instances associated with over 3K tutorial videos.

BlenderLore-Bench tests both where current coding agents struggle and whether BlenderLore helps them solve new 3D creation tasks. Its 100 tasks cover eight Blender capabilities and assess the resulting assets' appearance, structure, and behavior. We compare native coding agents with the same agents equipped with the lightweight BlenderLore skill, testing whether experience distilled from tutorials transfers to new tasks.

Our contributions are threefold:

- We introduce **BlenderLore**, a dataset of about 22K project instances associated with over 3K tutorial videos, and a pipeline that turns videos into illustrated tutorials and executable reconstructions.
- We build a **procedural knowledge library** from reviewed reconstructions, enabling agents to adapt and combine methods from different tutorials for new 3D creation tasks.
- We introduce **BlenderLore-Bench**, a 100-task benchmark across eight Blender capabilities that measures current agents' limitations and the transfer of tutorial-derived knowledge through a lightweight BlenderLore skill.

2 From Tutorial Videos to Procedural 3D Knowledge

2.1 3D Coding Requires Procedural Expertise

Coding agents and expert workflows. 3D coding translates creative specifications into executable programs and editable assets (Sun et al., 2023; Hu et al., 2024; Lu et al., 2025). Despite progress in geometry, materials, and scene layout (Gao et al., 2026; Li et al., 2025a; Yang et al., 2024), complex geometry, rigging, animation, and simulation remain challenging (Yang et al., 2026a; Zhou et al., 2026; Chi et al., 2026). These tasks require coordinating interdependent operations, not merely invoking individual APIs. We call knowledge of technique selection, operation order, parameter coordination, and intermediate checks *procedural expertise*.

Existing 3D resources. Existing resources provide finished assets (Chang et al., 2015; Deitke et al., 2023b), procedural generators (Raistrick et al., 2023; Denninger et al., 2019), asset–program pairs (Dai et al., 2025; Gao et al., 2026), and human CAD design sequences (Willis et al., 2021; Wu et al., 2021). However, they do not necessarily preserve the intermediate observations and explanations that guide construction. BlenderLore complements these resources with demonstrated workflows linked to executable code and reusable methods (Table 1).

Resource	Construction evidence	Scope
ShapeNet / Objaverse	Finished assets	3D objects
Infinigen / BlenderProc	Procedural generators	Generated scenes
Fusion 360 Gallery	Human design sequences	CAD construction
MeshCoder / 3DCodeBench	Asset–program pairs	Object construction
Material Apprentice	Tutorial-derived process and code	Procedural materials
BlenderLore	Illustrated workflows, code, and reusable methods	Multiple 3D capabilities

Table 1: Construction evidence and scope of representative 3D resources.

2.2 Methodology: Learn from Internet Tutorial Videos

Learning from internet tutorial videos aims to acquire procedural knowledge from recorded human demonstrations.[1] By combining narration, on-screen operations, and intermediate results, these videos reveal how tasks are performed rather than merely showing their final outcomes. For 3D coding, the challenge is to translate these demonstrations into executable programs and reusable construction methods, rather than simply replay interface actions.

BlenderLore converts videos into illustrated workflows and uses them, together with associated source projects when available, to reconstruct editable assets in Blender Python. Running the code and inspecting the resulting geometry, appearance, and motion help assess whether the inferred operations reproduce the intended effects. This *executable reconstruction* links tutorial evidence to concrete implementations from which reusable methods can be extracted (Ellis et al., 2021).

We organize reviewed methods into a procedural knowledge library, recording their source, applicability, implementation, and checks. Agents can retrieve, adapt, and combine these methods for new goals. The unit of reuse is thus a *construction method*, not an entire source asset: reconstruction is an intermediate step toward generalization beyond the demonstrations. BlenderLore Skill supplies the library and workflow guidance to existing coding agents, allowing knowledge to accumulate externally without updating model parameters. The library remains frozen during evaluation, where we assess its value for new 3D tasks. Section 3 details dataset construction and knowledge extraction.

2.3 Benefits to Learn from Internet Tutorial Videos

Three considerations motivate this approach.

Reason I: Expert Workflows beyond LLMs. Tutorials provide worked examples from which procedural knowledge can be extracted (Chi et al., 1989). Narration, on-screen settings, and intermediate results connect artists' operations and parameter choices to their visible effects. This process evidence offers guidance on how to organize a construction workflow, complementing general programming ability.

Reason II: Generalization beyond Demonstrated Assets. The goal is to acquire transferable methods, not merely reproduce tutorial assets. Translating demonstrated steps into Blender Python and inspecting the resulting assets ground inferred methods in executable implementations. We then extract their applicability, operations, and checks as reusable knowledge (Ellis et al., 2021). The intended generalization is through *adaptation* to new objects and *composition* across tutorials (Lake & Baroni, 2018). For example, a rigging method can be adapted to another character and combined with a separately learned material. Reconstruction is therefore an intermediate step toward method reuse, rather than the final learning objective.

[1] Prior work has used instructional videos to guide computer-use agents (Lin et al., 2024; Song et al., 2025; Liu et al., 2026b) and to extract procedural knowledge for material generation (Gupta et al., 2026).

Reason III: Scalability with Diverse Demonstrations. Internet tutorials cover modeling, materials, rigging, animation, and simulation. While Material Apprentice studies tutorial-derived knowledge for procedural materials (Gupta et al., 2026), BlenderLore draws on over 3K videos from YouTube and Bilibili across multiple Blender capabilities. This diversity provides complementary methods for more complex creation workflows.

Here, learning means accumulating an external procedural knowledge library rather than updating model parameters. The library remains frozen during evaluation, where we test its value for new 3D tasks. Section 3 describes its construction.

3 BUILDING BLENDERLORE

3.1 DATA CONSTRUCTION

Illustrated Tutorials. We collect over 3K Blender tutorial videos from YouTube and Bilibili. We align video frames, narration, on-screen text, and interface changes into ordered steps. Each step records the inputs, operations, parameters, and intermediate result. Figure 2 summarizes the full workflow, from illustrated tutorial steps to executable asset reconstruction and reusable procedural knowledge.

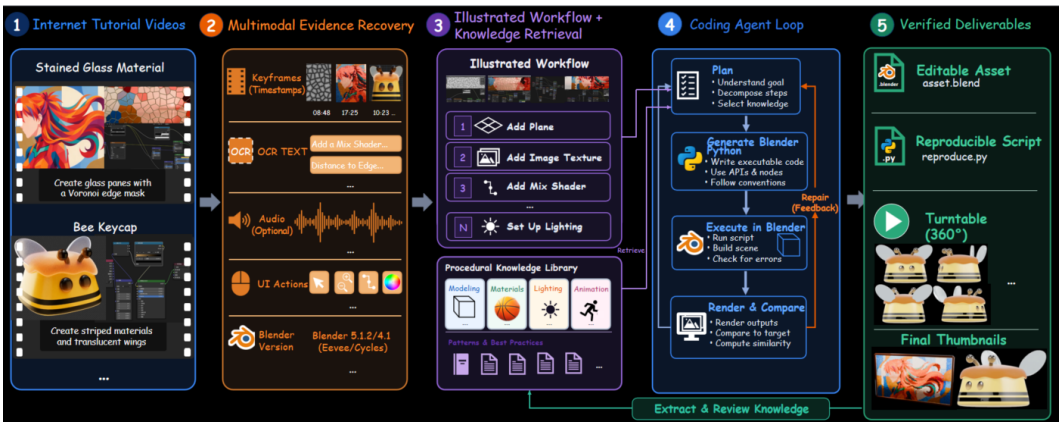


Figure 2: BlenderLore turns video evidence into illustrated steps, reconstructs editable assets through a coding agent loop, and retains procedural knowledge for future tasks.

Procedural Knowledge Library. Some tutorial creators release the assets used in their videos. We collect 1,017 such videos, linked to 3,732 open-source assets. About half of the projects can be used directly; the rest have missing components that require repair. We compare the asset files with the illustrated tutorials and match key implementation steps to the corresponding code. We organize these matches into a procedural knowledge library. Each entry records its source, retrieval keywords, when and how the method applies, an implementation excerpt, and checks for the intended result. Appendix C shows a concrete entry and how an agent retrieves and adapts it.

Asset Reconstruction. We first repair the incomplete source projects. The agent then uses the illustrated tutorials and procedural knowledge library to reconstruct further assets from scratch. This brings the collection to 8,918 projects. As the collection grows, we also extract new methods and update the library.

3.2 GENERALIZING BEYOND TUTORIALS

Creating New Assets. We combine design ideas and methods from different tutorials to create new assets. Figure 3 shows an example: we model a static Duck GiGi using one tutorial, transfer a rigging and walking method from a baby crocodile tutorial, and apply an iridescent material from a third tutorial. This process expands BlenderLore to about 22K project instances. It also contributes high-quality data for complex 3D modeling to the community.

Implementation Details. We package the reconstruction workflow and procedural knowledge library as BlenderLore skill. Codex, using GPT-5.6 Sol (OpenAI, 2026a), extracts illustrated tutorials, repairs or reconstructs assets, and then extracts procedural knowledge. Human experts check the resulting assets.

Experts in the Loop. Four human experts with an average of 6 years of experience in procedural 3D modeling take part throughout dataset construction and validation. Over three months, they help refine the workflow, propose combinations of existing design ideas for new assets, and manually check every final asset file.

Solving New 3D Tasks. Beyond reconstructing assets, BlenderLore Skill can help coding agents create new assets by retrieving, adapting, and combining methods from the procedural knowledge library. Section 4 introduces BlenderLore-Bench to evaluate this capability on new 3D tasks.

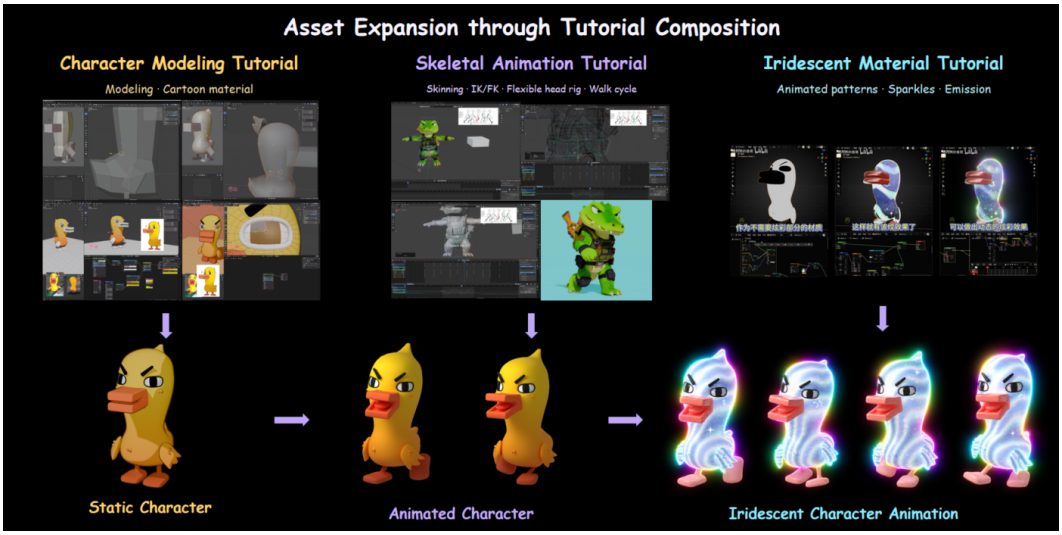


Figure 3: **BlenderLore creates a walking iridescent Duck GiGi.** It builds the yellow duck from a static modeling tutorial, transfers a crocodile rigging and walking method to it, and applies an iridescent material learned from a separate tutorial.

4 BLENDERLORE-BENCH

To test where current coding agents struggle and whether BlenderLore helps them solve new 3D creation tasks, we introduce BlenderLore-Bench.

4.1 DESIGN PRINCIPLES

Our benchmark has four design principles: **real tasks, clear descriptions, robust verification, and appropriate difficulty** (Bercovich, 2026).

- **Start from real work.** We derive tasks from real Blender tutorials. These tutorials capture the workflows of human creators.
- **Straightforward and unambiguous descriptions.** We organize each task into clear steps. Each step includes a text description and reference images that specify the desired result.
- **Robust verification.** We extract reference implementation methods from the tutorials, including operations and parameters. Human experts define the score allocation and scoring criteria for each step. These references support the evaluation of submitted assets. The task score is the sum of its step scores.
- **Appropriate difficulty.** We aim for tasks that challenge current large models while remaining solvable with the provided descriptions and inputs.

4.2 Task Construction and Capability Coverage

BlenderLore-Bench covers diverse content themes and eight capability dimensions. We define these dimensions with reference to Blender's documented tools and workflows (Blender Foundation, 2026b), and summarize them in Table 2. Based on this scope, we assemble a pool of 200 tutorial videos.

Code	Capability	Scope
GEO	Geometry	Geometry, topology, hard-surface modeling, organic modeling, curves, sculpting
PROC	Procedural construction	Modifiers, instancing, Geometry Nodes, procedural relationships
SURF	Materials and appearance	Materials, UVs, textures, procedural textures, displacement, transparency, volumes
SCN	Scene and rendering	Scene assembly, lighting, cameras, World, rendering, compositing
RIG	Rigging	Armatures, skinning, constraints, shape keys, drivers
ANM	Animation	Keyframes, F-curves, NLA, procedural animation, camera animation
SIM	Simulation	Rigid bodies, cloth, fluids, soft bodies, particles, caches
PIPE	Asset delivery	Project structures, preservation under editing, self-contained dependencies, import/export, script replay

Table 2: Blender capability categories.

Four human experts watch the original videos and develop task packages. Each package contains a task description, stepwise rubrics, scoring criteria, reference implementation methods, and reference images. All four experts cross-check each task, discuss disagreements, and revise it before finalization. We select 100 tasks for the final benchmark. To guard against benchmark leakage, human experts screened candidate tutorials and target assets against the existing BlenderLore data and excluded substantially similar cases (Appendix F). Figure 4 shows the content themes of the 100 BlenderLore-Bench tasks and the distribution of rubric points across the eight capability dimensions.

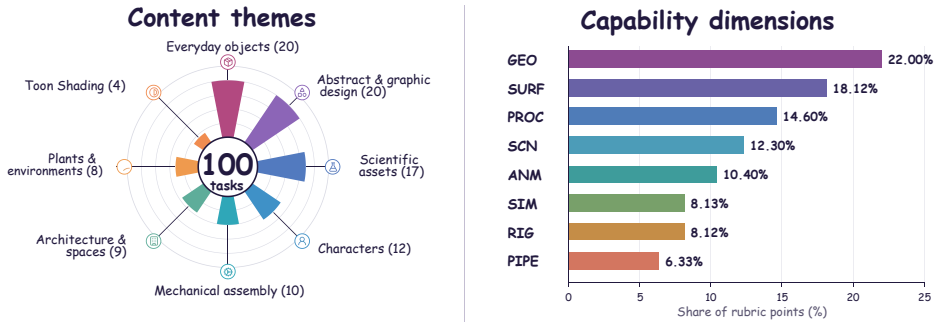


Figure 4: Content themes and capability coverage of the 100 BlenderLore-Bench tasks.

Figure 5 shows a glass honey jar task, with visual references, stepwise requirements, and an example criterion for scoring the viscous dripping effect.

4.3 Evaluation

We use Gemini 3.8 Flash (Google, 2026) in Gemini CLI as the evaluator. It receives the agent-generated preview images or videos, the `.blend` file, and `build.py`. It also receives the expert-authored scoring criteria, reference implementation methods, and reference images. A tool-equipped evaluation harness can inspect renders, examine scene structure, and test animation or simulation behavior (Chen et al., 2026b). These checks make it better suited to our tasks than API-only scoring. Reference implementation methods guide the evaluation, but alternative implementations receive

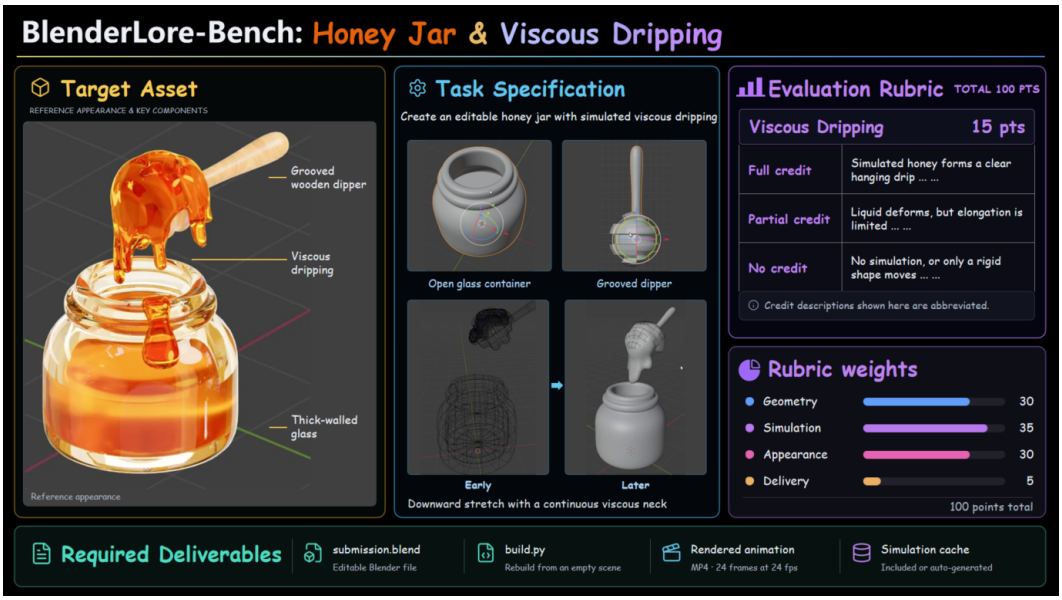


Figure 5: **A BlenderLore-Bench example: honey jar with viscous dripping.** The task provides visual references and requires the model to construct the geometry, reproduce the honey's material appearance, and simulate viscous dripping through stepwise instructions.

credit when they satisfy the task requirements. The evaluator scores each step under the expert-defined criteria and sums the step scores. Appendix G reports judge–human agreement.

5 Experiments

We evaluate coding agents on the 100 BlenderLore-Bench tasks. Table 3 compares overall performance and resource use. Table 4 breaks down performance by Blender capability.

5.1 Experimental Setup

Models and harnesses. We evaluate seven native coding-agent configurations on all 100 tasks: Codex with GPT-6 Astra (OpenAI, 2026b), GPT-5.6 Sol (OpenAI, 2026a), and GLM-5.3-Flash (Zhipu AI, 2026); Claude Code with Claude Opus 5 (Anthropic, 2026a), Kimi K3 (Moonshot AI, 2026), and DeepSeek-V4.1-Flash (DeepSeek, 2026); and Gemini CLI with Gemini 3.8 Flash (Google, 2026). We also run BlenderLore Skill in Codex with GPT-6 Astra, GPT-5.6 Sol, and GLM-5.3-Flash. The skill provides the reconstruction workflow and access to the procedural knowledge library. Generation used each harness's default reasoning settings.

Runtime Environment. BlenderLore-Bench is implemented on Harbor and executes agent trials as local subprocesses with separate task workspaces. The reference environment uses Ubuntu 22.04, Python 3.12, and Blender 5.1.2, with EEVEE and Cycles available for rendering. FFmpeg and FFprobe support video inspection and frame extraction for dynamic tasks. Each agent must save build.py, submission.blend, and the required image or video evidence in its task workspace. Each generation trial has a wall-clock timeout of 4 hours. Offline evaluation uses Gemini CLI v0.61.0 with gemini-3.8-flash at LOW thinking level.

Resource use. Time is the mean agent runtime per task, including code execution, rendering, and revisions. Token counts include all model calls during a run. Token counts across providers should be read with care because their tokenizers and reporting conventions differ.

5.2 Main Results

Table 3 reports results on the 100 BlenderLore-Bench tasks. GPT-6 Astra is the strongest native agent, with a score of 85.80. BlenderLore Skill raises its score to 90.94. GPT-5.6 Sol and GLM-5.3-Flash gain more: 14.41 and 14.82 points, respectively. With the skill, both achieve scores comparable to native GPT-6 Astra. All three models also finish faster when tested with the skill. Claude Opus 5 has the lowest score because 43 of its tasks timed out and were scored as failures.

Model	Harness	Score ↑ /100	Δ Score ↑ points	Time ↓ min	Tokens ↓ In / Out (k)
A. Native Harness					
GPT-6 Astra	Codex	85.80	–	16.97	862.19 / 13.80
GPT-5.6 Sol	Codex	73.32	–	13.26	**802.69 / 13.12**
Gemini 3.8 Flash	Gemini CLI	70.65	–	17.23	5514.72 / 85.32
GLM-5.3-Flash	Codex	67.75	–	32.40	5711.67 / 35.68
Kimi K3	Claude Code	64.82	–	48.24	4845.28 / 56.57
DeepSeek-V4.1-Flash	Claude Code	49.54	–	42.08	9606.56 / 171.04
Claude Opus 5	Claude Code	30.81	–	93.46	3112.01 / 113.78
B. BlenderLore Skill (Ours)					
GPT-6 Astra	Codex	**90.94**	+5.14	10.58	1165.57 / 15.55
GPT-5.6 Sol	Codex	87.73	+14.41	**9.39**	1435.24 / 16.92
GLM-5.3-Flash	Codex	82.57	+14.82	30.29	5249.71 / 36.27

Table 3: Overall performance on 100 BlenderLore-Bench tasks. Score is the mean task score out of 100. Δ compares BlenderLore Skill with the same model in its native harness. Time is mean minutes per task; tokens are mean input/output tokens in thousands. Bold and underline mark the best and second-best values.

Key Finding. BlenderLore Skill improves coding agents' 3D creation performance and reduces their runtime.

6 Analysis

We analyze where coding agents struggle, which capabilities benefit from BlenderLore Skill, and how the skill affects runtime.

6.1 Capability-wise Performance and Failure Modes

Table 4 shows substantial differences across Blender capabilities. Strong native agents score well on geometry, materials, and procedural construction. Animation scores show a correlation with multimodal ability: GPT-6 Astra, Gemini 3.8 Flash, and GLM-5.3-Flash score higher than the other models. Rigging and simulation are common weaknesses for coding agents.

Model	Harness	GEO	PROC	SURF	SCN	RIG	ANM	SIM	PIPE	Overall
A. Native Harness										
GPT-6 Astra	Codex	92.23	90.41	91.94	76.02	55.61	90.00	74.92	**100.00**	85.80
GPT-5.6 Sol	Codex	84.63	70.19	74.37	62.44	58.41	56.06	22.49	79.33	73.32
Gemini 3.8 Flash	Gemini CLI	79.80	55.85	70.74	74.47	55.14	77.50	41.71	59.22	70.65
GLM-5.3-Flash	Codex	76.60	53.78	71.42	72.08	55.69	73.72	48.33	53.87	67.75
Kimi K3	Claude Code	75.10	56.84	68.54	62.48	63.63	58.91	39.85	48.08	64.82
DeepSeek-V4.1-Flash	Claude Code	64.65	36.25	53.42	53.84	27.01	44.89	26.94	41.89	49.54
Claude Opus 5	Claude Code	33.33	41.67	27.35	24.21	25.43	44.44	17.65	21.09	30.81
B. BlenderLore Skill (Ours)										
GPT-6 Astra	Codex	96.11$_{+3.9}$	**91.64**$_{+1.2}$	**93.88**$_{+1.9}$	90.98$_{+15.0}$	71.29$_{+15.7}$	**90.48**$_{+0.5}$	**84.44**$_{+9.5}$	97.15$_{-2.9}$	**90.94**$_{+5.1}$
GPT-5.6 Sol	Codex	**96.31**$_{+11.7}$	76.78$_{+6.6}$	87.01$_{+12.6}$	89.05$_{+26.6}$	**73.21**$_{+14.8}$	83.51$_{+27.5}$	73.22$_{+50.7}$	83.97$_{+4.6}$	87.73$_{+14.4}$
GLM-5.3-Flash	Codex	94.86$_{+18.3}$	90.31$_{+36.5}$	82.56$_{+11.1}$	**93.11**$_{+21.0}$	72.88$_{+17.2}$	66.18$_{-7.5}$	72.51$_{+24.2}$	53.87$_{0.0}$	82.57$_{+14.8}$

Table 4: Scores by Blender capability on applicable tasks (0–100). Subscripts show the change from the same model in its native harness, rounded to one decimal place; green indicates a gain and red a loss. Overall is the mean score across all 100 tasks. Bold and underline mark the best and second-best scores in each column.

Completing a scene does not guarantee a complete submission. Except for GPT-6 Astra, all agents lose points on asset delivery. Common problems include missing required preview files, unfinished assets, and reconstruction scripts that fail to run.

Key Finding. Coding agents struggle with rigging and simulation.

6.2 What Does BlenderLore Skill Improve?

Gains and Exceptions. BlenderLore Skill improves most capability dimensions for all three tested models. The gains are particularly large on their initially weaker capabilities. GPT-5.6 Sol improves from 22.49 to 73.22 on simulation, a gain of 50.73 points. GLM-5.3-Flash improves from 53.78 to 90.31 on procedural construction, a gain of 36.53 points. GPT-6 Astra improves from 55.61 to 71.29 on rigging, a gain of 15.68 points. Some scores also decrease. For example, GLM-5.3-Flash drops from 73.72 to 66.18 on animation.

Key Finding. BlenderLore Skill improves most capabilities, with larger gains on initially weaker ones.

Runtime and Token Use. BlenderLore Skill substantially reduces runtime for GPT-6 Astra and GPT-5.6 Sol. Their mean runtime falls from 16.97 to 10.58 minutes and from 13.26 to 9.39 minutes, respectively. Our analysis of GPT-6 Astra's traces shows that the time savings come mainly from reduced thinking time outside tool calls. Meanwhile, the mean number of image-inspection calls increases from 8.14 to 10.69 per task. The agent therefore performs more visual checks despite finishing sooner. These additional multimodal interactions may contribute to the higher token use.

Key Finding. BlenderLore improves time efficiency by reducing model thinking time, but slightly increases token consumption.

6.3 Performance as the Procedural Knowledge Library Grows

We examine how knowledge library size affects the performance of BlenderLore Skill. We sort the procedural knowledge chronologically and retain the first 0%, 25%, 50%, 75%, and 100%. We evaluate performance at each library size using GPT-6 Astra in Codex.

Knowledge library size	0%	25%	50%	75%	100%
Overall score	85.80	88.10	90.10	90.09	90.94
Time (min)	16.97	13.42	11.64	10.89	10.58

Table 5: Trends in GPT-6 Astra's overall score and runtime across different knowledge library sizes.

More Reconstructed Assets Improve Performance. As shown in Table 5, expanding the knowledge library generally improves the overall score while reducing runtime. This indicates that BlenderLore Skill becomes stronger as more 3D assets are successfully reconstructed.

7 Conclusion

We presented BlenderLore, which turns Blender tutorial videos into executable 3D assets and reusable procedural knowledge. By reconstructing assets and combining methods across tutorials, we build a dataset of about 22K project instances. We package the reconstruction workflow and procedural knowledge library as BlenderLore Skill to help coding agents apply this knowledge to new tasks. Experiments on BlenderLore-Bench show that the skill improves overall scores and reduces runtime across three models. Expanding the procedural knowledge library further improves performance and time efficiency. These results demonstrate that knowledge accumulated through asset reconstruction can strengthen coding agents' ability to create new 3D assets.

AI USE STATEMENT

Generative AI was used to interpret Blender tutorial videos, assist with the reconstruction and creation of 3D assets, generate and revise Blender Python code, and extract reusable procedural knowledge. We also evaluated generative AI systems on BlenderLore-Bench. Generative AI assisted with code and data processing and with drafting tables and figures. The expert team reviewed all generated 3D assets and a subset of BlenderLore-Bench outputs. The authors made the final edits to AI-drafted tables and figures, checked the reported results against the underlying experiments, and take responsibility for the final content of this paper.

REFERENCES

Siddharth Ahuja. Blendermcp: Blender model context protocol integration for claude and other llm agents. https://github.com/ahujasid/blender-mcp, 2025.

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating cad code with vision-language models for 3d designs. *arXiv preprint arXiv:2410.05340*, 2024.

Anthropic. Introducing claude opus 5. https://www.anthropic.com/news/claude-opus-5, 2026a.

Anthropic. Claude for creative work: Connecting claude to blender, adobe, ableton, autodesk fusion, and sketchup. https://www.anthropic.com/news/claude-for-creative-work, 2026b.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, Jakob Engel, Edward Miller, Richard Newcombe, and Vasileios Balntas. Scenescript: Reconstructing scenes with an autoregressive structured language model. In *European Conference on Computer Vision*, pp. 247–263, 2024.

Jesse Barkley, Rumi Loghmani, and Amir Barati Farimani. Cadsmith: Multi-agent cad generation with programmatic geometric validation. *arXiv preprint arXiv:2603.26512*, 2026.

Ivan Bercovich. Good benchmarks. *arXiv preprint arXiv:2607.12217*, 2026. URL https://arxiv.org/abs/2607.12217.

Blender Foundation. Blender mcp server. https://www.blender.org/lab/mcp-server/, 2026a.

Blender Foundation. Blender manual. https://docs.blender.org/manual/en/latest/, 2026b. Accessed September 2026.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Jingyuan Chen, Sheng Jin, Haopeng Sun, Wentao Liu, and Chen Qian. Unicad: A unified benchmark and universal model for multi-modal multi-task cad. *arXiv preprint arXiv:2606.05058*, 2026a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Weiliang Chen, Haowen Sun, Jun Gao, Jiawei Chi, Hanyang Wang, Qiyu Dai, Yihao Li, Hao Li, Jingnan Gao, Yi-Hsin Hung, et al. Harnesseval-w: Agentifying the evaluation of visual worlds. *arXiv preprint arXiv:2608.16859*, 2026b.

Michelene T. H. Chi, Miriam Bassok, Matthew W. Lewis, Peter Reimann, and Robert Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13(2):145–182, 1989. doi: 10.1207/s15516709cog1302_1.

Wayne Chi, Yixiong Fang, Arnav Yayavaram, Siddharth Yayavaram, Seth Karten, Qiuhong Anna Wei, Runkun Chen, Alexander Wang, Valerie Chen, Ameet Talwalkar, and Chris Donahue. Gamedevbench: Evaluating agentic capabilities through game development. In *International Conference on Machine Learning*, 2026. arXiv:2602.11103.

Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F. Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. Abo: Dataset and benchmarks for real-world 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21126–21136, 2022.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5828–5839, 2017.

Bingquan Dai, Li Ray Luo, Qihong Tang, Jie Wang, Xinyu Lian, Hao Xu, Minghan Qin, Xudong Xu, Bo Dai, Haoqian Wang, Zhaoyang Lyu, and Jiangmiao Pang. Meshcoder: Llm-powered structured mesh code generation from point clouds. *arXiv preprint arXiv:2508.14879*, 2025.

Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. Llmr: Real-time prompting of interactive worlds using large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024.

DeepSeek. Introducing deepseek-v4.1-flash. `https://www.deepseek.com/en/news/deepseek-v4-1-flash/`, 2026.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. In *Advances in Neural Information Processing Systems*, 2022.

Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-xl: A universe of 10m+ 3d objects. In *Advances in Neural Information Processing Systems*, 2023a.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13142–13153, 2023b.

Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc. *arXiv preprint arXiv:1911.01911*, 2019.

Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. Muse: Benchmarking manufacturable, functional, and assemblable text-to-cad generation. *arXiv preprint arXiv:2605.28579*, 2026.

Anna C. Doris, Md Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. Cad-coder: An open-source vision-language model for computer-aided design code generation. *arXiv preprint arXiv:2505.14646*, 2025.

Anna C. Doris, Jacob Thomas Sony, Ghadi Nehme, Era Syla, Amin Heyrani Nobari, and Faez Ahmed. Cadbench: A multimodal benchmark for ai-assisted cad program generation. *arXiv preprint arXiv:2605.10873*, 2026.

Yuhao Du, Shunian Chen, Wenbo Zan, Peizhao Li, Mingxuan Wang, Dingjie Song, Bo Li, Yan Hu, and Benyou Wang. Blenderllm: Training large language models for computer-aided design with self-improvement. *arXiv preprint arXiv:2412.14203*, 2024.

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*, pp. 835–850, 2021.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems*, 2022.

Yipeng Gao, Lei Shu, Genzhi Ye, Xi Xiong, Ameesh Makadia, Meiqi Guo, Laurent Itti, and Jindong Chen. 3dcodebench: Benchmarking agentic procedural 3d modeling via code. *arXiv preprint arXiv:2606.01057*, 2026.

Google. Gemini 3.8 flash. `https://ai.google.dev/gemini-api/docs/models/gemini-3.8-flash`, 2026.

Prashanth Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. Cadmium: Fine-tuning code language models for text-driven sequential cad design. *Transactions on Machine Learning Research*, 2026.

Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J. Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3739–3751, 2022.

Yunqi Gu, Ian Huang, Jihyeon Je, Guandao Yang, and Leonidas Guibas. Blendergym: Benchmarking foundational model systems for graphics editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18574–18583, 2025.

Chunchao Guo, Jinpeng Li, Yang Li, and Zilong Huang. Worldclaw: Agentic 3d open-world generation at scale. *arXiv preprint arXiv:2608.05248*, 2026.

Kunal Gupta, Gaurav Joshi, Yen-Ru Chen, Seemandhar Jain, Ishit Mehta, and Manmohan Chandraker. Reflecting process expertise in procedural material generation. *arXiv preprint arXiv:2607.13318*, 2026.

Guangzhao He, Rundong Luo, Wei-Chiu Ma, and Hadar Averbuch-Elor. Thinking in blender: Staged executable inverse graphics with vision-language models. *arXiv preprint arXiv:2606.02580*, 2026.

Liu He, Yizhi Song, Hejun Huang, Daniel Aliaga, and Xin Zhou. Kubrick: Multimodal agent collaborations for synthetic video generation. *arXiv preprint arXiv:2408.10453*, 2024.

Yuze He, Yishun Bai, Matthieu Lin, Wang Zhao, Yubin Hu, Jenny Sheng, Ran Yi, Juanzi Li, and Yong-Jin Liu. T^3bench: Benchmarking current progress in text-to-3d generation. *arXiv preprint arXiv:2310.02977*, 2023.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *International Conference on Machine Learning*, pp. 19252–19282, 2024.

Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models. In *European Conference on Computer Vision*, 2024.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics*, 39(6):1–20, 2020.

R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics*, 42 (4):1–17, 2023.

Abhishek Joshi, Beining Han, Jack Nugent, Max G. Saez-Diez, Yiming Zuo, Jonathan Liu, Hongyu Wen, Stamatis Alexandropoulos, Karhan Kayan, Alessandro Calveri, Tao Sun, Grace Liu, Yiming Shao, Alexander Raistrick, and Jia Deng. Procedural generation of articulated simulation-ready assets. *arXiv preprint arXiv:2505.10755*, 2025.

Mohammad Sadil Khan, Sankalp Sinha, Sheikh Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems*, 2024.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9601–9611, 2019.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multimodal cad reconstruction with online reinforcement learning. *arXiv preprint arXiv:2505.22914*, 2025.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 2873–2882, 2018.

Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. In *International Conference on Learning Representations*, pp. 17578–17602, 2025.

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 *International Conference on Learning Representations*, 2025a.

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-cad: A self-refined generative modeling for 3d parametric cad using local inference via deepseek. *arXiv preprint arXiv:2505.17702*, 2025b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Wu, Mingyi Yan, Zhengyuan Yang, Lijuan Wang, and Mike Zheng Shou. Videogui: A benchmark for gui automation from instructional videos. In *Advances in Neural Information Processing Systems*, 2024.

Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. Scenethesis: A language and vision agentic framework for 3d scene generation. *arXiv preprint arXiv:2505.02836*, 2025.

Ling Liu, Bingchen Gong, Amal Dev Parakkat, and Maks Ovsjanikov. 3dharnessbench: Probing agentic 3d-to-code capabilities of frontier vision-language models. *arXiv preprint arXiv:2609.06535*, 2026a.

Yujian Liu, Ze Wang, Hao Chen, Ximeng Sun, Xiaodong Yu, Jialian Wu, Jiang Liu, Emad Barsoum, Zicheng Liu, and Shiyu Chang. Learning from online videos at inference time for computer-use agents. *Transactions on Machine Learning Research*, 2026b. arXiv:2511.04137.

Shuo Lu, Yinuo Xu, Kecheng Yu, Siru Jiang, Yongcan Yu, Yubin Wang, Haitao Yang, Yuxiang Zhang, Bin Wang, Ran He, and Jian Liang. Worldcoder-bench: Benchmarking physically grounded 3d world synthesis. *arXiv preprint arXiv:2606.01869*, 2026.

Sining Lu, Guan Chen, Nam Anh Dinh, Itai Lang, Ari Holtzman, and Rana Hanocka. Ll3m: Large language 3d modelers. *arXiv preprint arXiv:2508.08228*, 2025.

Tiange Luo, Chris Rockwell, Honglak Lee, and Justin Johnson. Scalable 3d captioning with pretrained models. In *Advances in Neural Information Processing Systems*, pp. 75307–75337, 2023.

Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-assistant: Tool-augmented vllms as generic cad task solvers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7284–7294, 2025.

Moonshot AI. Kimi k3. `https://platform.kimi.com/docs/guide/kimi-k3-quickstart`, 2026.

OpenAI. Gpt-5.6 sol model. `https://developers.openai.com/api/docs/models/gpt-5.6-sol`, 2026a.

OpenAI. Gpt-6 astra model. `https://developers.openai.com/api/docs/models/gpt-6-astra`, 2026b.

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12630–12641, 2023.

Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21783–21794, 2024.

F. Raji, Stefano Petrangeli, Matheus Gadelha, Y. Shen, Uttaran Bhattacharya, and Gang Wu. Proc3d: Procedural 3d generation and parametric editing of 3d shapes with large language models. *arXiv preprint arXiv:2601.12234*, 2026.

Massimo Rondelli, Francesco Pivi, and Maurizio Gabbrielli. Blenderrag: High-fidelity 3d object generation via retrieval-augmented code synthesis. *arXiv preprint arXiv:2605.00632*, 2026.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-recode: Reverse engineering cad code from point clouds. *arXiv preprint arXiv:2412.14042*, 2024.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5515–5523, 2018.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

Chan Hee Song, Yiwen Song, Palash Goyal, Yu Su, Oriana Riva, Hamid Palangi, and Tomas Pfister. Watch and learn: Learning to use computers from online videos. *arXiv preprint arXiv:2510.04673*, 2025.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models. *arXiv preprint arXiv:2310.12945*, 2023.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 29469–29478, 2025a.

Fan-Yun Sun, Shengguang Wu, Christian Jacobsen, Thomas Yim, Haoming Zou, Alexander Zook, Shangru Li, Yu-Hsin Chou, Ethem Can, Xunlei Wu, Clemens Eppner, Valts Blukis, Jonathan Tremblay, Jiajun Wu, Stan Birchfield, and Nick Haber. 3d-generalist: Self-improving vision-language-action models for crafting 3d worlds. *arXiv preprint arXiv:2507.06484*, 2025b.

Yolo Y. Tang, Daiki Shimada, Jiayue Meng, Jing Bi, Pinxin Liu, Yicheng Wang, Yunzhong Xiao, Zhangyun Tan, Zeliang Zhang, Chao Huang, Susan Liang, Qianxiang Shen, Luchuan Song, Ali Vosoughi, Mingqian Feng, Melika Filvantorkaman, and Chenliang Xu. Bvb: Benchmarking agentic video understanding via programmatic reconstruction in blender. *arXiv preprint arXiv:2609.15478*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Liang Wang, Heng Meng, Zekai Xiang, Jin Liu, Pingyi Zhou, Litao Chen, and Yongqiang Tang. Text2cad-bench: A benchmark for llm-based text-to-parametric cad generation. *arXiv preprint arXiv:2605.18430*, 2026.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-cad generation through infusing visual feedback in large language models. *arXiv preprint arXiv:2501.19054*, 2025.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics*, 40 (4), 2021.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6772–6782, 2021.

Tong Wu, Jiarui Zhang, Xiao Fu, Yuxin Wang, Jiawei Ren, Liang Pan, Wayne Wu, Lei Yang, Jiaqi Wang, Chen Qian, Dahua Lin, and Ziwei Liu. Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 803–814, 2023.

Haoyang Xie and Feng Ju. Text-to-cadquery: A new paradigm for cad generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm. *arXiv preprint arXiv:2411.04954*, 2024.

Yutaro Yamada, Khyathi Chandu, Bill Yuchen Lin, Jack Hessel, Ilker Yildirim, and Yejin Choi. L3go: Language agents with chain-of-3d-thoughts for generating unconventional objects. *arXiv preprint arXiv:2402.09052*, 2024.

Yikang Yang, Zhanpeng Hu, Youtian Lin, Mengqi Zhou, Jingxi Xu, Feihu Zhang, Jiaheng Liu, and Yao Yao. P3d-bench: Benchmarking mllms for parametric 3d generation and structural reasoning. *arXiv preprint arXiv:2606.11152*, 2026a.

Yixuan Yang, Zhen Luo, Wanshui Gan, Jinkun Hao, Junru Lu, Jinghao Yan, Zhaoyang Lyu, and Xudong Xu. Code-as-room: Generating 3d rooms from top-down view images via agentic code synthesis. *arXiv preprint arXiv:2605.18451*, 2026b.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16227–16237, 2024.

Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, C. Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Viga: Vision-as-inverse-graphics agent via interleaved multimodal reasoning. *arXiv preprint arXiv:2601.11109*, 2026.

Haozhe Zhang, Kaichen Liu, Miaomiao Chen, Lei Li, Shaojie Yang, Cheng Peng, and Hanjie Chen. Benchcad: A comprehensive, industry-standard benchmark for programmatic cad. *arXiv preprint arXiv:2605.10865*, 2026.

Yuhan Zhang, Mengchen Zhang, Tong Wu, Tengfei Wang, Gordon Wetzstein, Dahua Lin, and Ziwei Liu. 3dgen-bench: Comprehensive benchmark suite for 3d generative models. *arXiv preprint arXiv:2503.21745*, 2025.

Yifei Zhao, Xiangxin Zhou, Wenhao Yang, Jiaqi Tang, Pu Jian, Huanjin Yao, Jiarui Yao, Haowei Lin, Chunchao Guo, Zhuo Chen, Wenkai Lyu, Jianzhu Ma, Xueqian Wang, and Wenxi Zhu. Sceneactbench: Can agents act on the 3d scenes they see? *arXiv preprint arXiv:2607.22393*, 2026.

Yan Zheng and Florian Bordes. Voxelcodebench: Benchmarking 3d world modeling through code generation. *arXiv preprint arXiv:2604.02580*, 2026.

Zhipu AI. Glm-5.3-flash/flashx. `https://docs.bigmodel.cn/cn/guide/models/vlm/glm-5.3-flash`, 2026.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation. *arXiv preprint arXiv:2605.15187*, 2026.

Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

A Related Work

3D datasets. Open 3D data has progressed from static repositories such as ShapeNet (Chang et al., 2015), ABO (Collins et al., 2022), Thingi10K (Zhou & Jacobson, 2016), Objaverse (Deitke et al., 2023b;a), OmniObject3D (Wu et al., 2023), ScanNet (Dai et al., 2017), and Cap3D (Luo et al., 2023) to procedural pipelines such as Infinigen (Raistrick et al., 2023; 2024; Joshi et al., 2025), BlenderProc (Denninger et al., 2019), Kubric (Greff et al., 2022), and ProcTHOR (Deitke et al., 2022), whose assets are produced by hand-written generators. A newer line pairs assets with the programs that build them: 3DCodeBench (Gao et al., 2026) and MeshCoder (Dai et al., 2025) distill standalone Blender scripts from procedural generators, BlendNet (Du et al., 2024) and BlenderRAG (Rondelli et al., 2026) collect text–script pairs, VLMaterial (Li et al., 2025a) collects procedural materials as Python, and the CAD community pairs models with sketch-and-extrude sequences or CadQuery programs (Koch et al., 2019; Willis et al., 2021; Wu et al., 2021; Khan et al., 2024; Rukhovich et al., 2024; Zhang et al., 2026). These corpora record finished assets or programs written by engineers rather than the interactive process of a human creator.

Benchmarks. General-purpose code benchmarks such as HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and CodeContests (Li et al., 2022) verify programs with unit tests; 3D generation has no such oracle, so benchmarks compare outputs with references. BlenderGym (Gu et al., 2025) and BlenderBench (Yin et al., 2026) score edits to existing scenes rather than construction from scratch. 3DCodeBench (Gao et al., 2026) scores from-scratch objects against Infinigen references with view similarity, shape distance, and a human-preference arena, and 3DHarnessBench (Liu et al., 2026a) reuses its targets to measure how progressively richer agent access through Blender MCP changes the outcome. SceneActBench (Zhao et al., 2026) evaluates agents acting on multi-object scenes, and BVB (Tang et al., 2026) asks agents to reconstruct real videos as animated Blender scenes. Beyond Blender, VoxelCodeBench (Zheng & Bordes, 2026) targets an Unreal voxel API, WorldCoder-Bench (Lu et al., 2026) checks Three.js worlds against hidden behavioral contracts, P3D-Bench (Yang et al., 2026a) scores parametric programs for structural correctness, and CAD suites measure dimensional, functional, and assembly correctness (Du et al., 2024; Zhang et al., 2026; Wang et al., 2026; Dong et al., 2026; Chen et al., 2026a; Doris et al., 2026). SceneScript (Avetisyan et al., 2024) predicts layouts instead of executable assets, and T^3Bench (He et al., 2023) and 3DGen-Bench (Zhang et al., 2025) evaluate neural text-to-3D without a code modality. Closest to our construction, GameDevBench (Chi et al., 2026) derives game-development tasks from web and video tutorials in Godot. These benchmarks mostly measure executability and similarity to a reference mesh or render.

Agents that write 3D programs. Early methods learned shape programs in restricted domain-specific languages (Sharma et al., 2018; Jones et al., 2020; 2023). With large language models, 3D-GPT (Sun et al., 2023), SceneCraft (Hu et al., 2024), L3GO (Yamada et al., 2024), and LL3M (Lu et al., 2025) generate Blender Python directly from text, and later systems strengthen such agents in several ways. BlenderAlchemy (Huang et al., 2024), VIGA (Yin et al., 2026), SEIG (He et al., 2026), and 3D-Generalist (Sun et al., 2025b) close the loop with rendered feedback for editing, single-image inverse graphics, and sequential decision making; BlenderLLM (Du et al., 2024) and MeshCoder (Dai et al., 2025) fine-tune models on asset–code pairs; and Proc3D (Raji et al., 2026) replaces the raw API with simplified procedural graphs. A further line supplies the agent with retrieved knowledge, in the manner of retrieval-augmented generation: SceneCraft compiles reusable functions into a library, LL3M and CADSmith (Barkley et al., 2026) retrieve API documentation, BlenderRAG (Rondelli et al., 2026) retrieves worked examples, Kubrick (He et al., 2024) grounds Blender scripting in retrieved documentation, 3DCodeBench (Gao et al., 2026) and ArtiCAD (Shui et al., 2026) maintain experience stores that accumulate resolved issues and design knowledge, and MaterialApprentice (Gupta et al., 2026) extracts process traces from material tutorial videos and retrieves them at generation time, echoing skill libraries in embodied agents (Wang et al., 2023; Fan et al., 2022) and video-derived guidance for computer-use agents (Lin et al., 2024; Song et al., 2025; Liu et al., 2026b). Related agents generate material node graphs (Li et al., 2025a), articulated assets (Le et al., 2025; Joshi et al., 2025; Zhou et al., 2026), scene layouts (Yang et al., 2024; Ling et al., 2025; Sun et al., 2025a), and open worlds (Guo et al., 2026). Outside Blender, a large body of work generates parametric CAD as CadQuery or Python code, from fine-tuned and reinforcement-learned models (Xu et al., 2024; Wang et al., 2025; Xie & Ju, 2025; Doris et al., 2025; Rukhovich et al., 2024; Kolodiazhnyi et al., 2025; Govindarajan et al., 2026; Li et al., 2025b) to agents that verify code visually or geometrically (Alrashedy et al., 2024; Mallis et al., 2025; Barkley et al.,

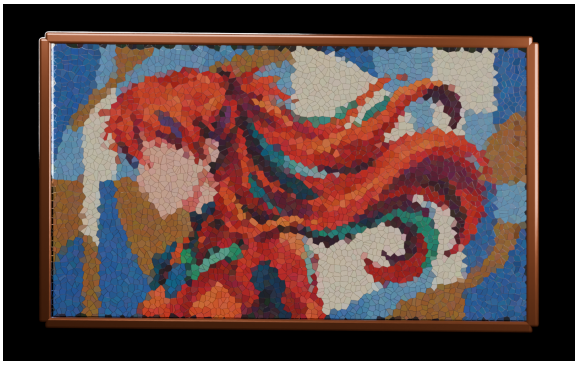
2026; Shui et al., 2026); LLMR (De La Torre et al., 2024) drives Unity, and MCP servers connect frontier models to Blender and other packages (Ahuja, 2025; Blender Foundation, 2026a; Anthropic, 2026b). Across these systems, execution feedback largely resolves executability, whereas structural correctness and physical plausibility remain the bottleneck (Gao et al., 2026; Yang et al., 2026a), and the knowledge an agent can retrieve is limited to API documentation, fixed examples, experience kept inside a single pipeline, or process traces for a single capability.

B Illustrated Tutorial and Executable Reconstruction: An Example

The Asuka stained-glass example takes a tutorial video (BV18xqdBYEEv) and a source illustration as input. Its outputs are an illustrated Markdown tutorial, an editable Blender project, a rendered preview, and a four-second animation. Figure 6 shows the source illustration and the rendered output.



(a) Input: source illustration



(b) Output: Blender render

Figure 6: The source illustration and the resulting stained-glass window.

B.1 Tutorial Markdown

The tutorial records the operations, parameters, and intermediate results of ten stages covering material construction, scene setup, and animation. Figure 7 shows the reference images accompanying these steps.

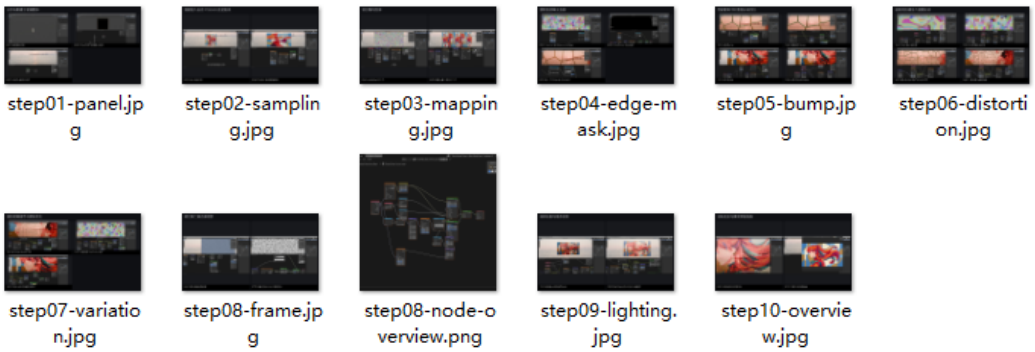


Figure 7: Reference images used in the illustrated tutorial Markdown.

```
# Stained glass: a Voronoi material from a 2D illustration

Source video: [Blender stained-glass material](https://www.bilibili.com/video/BV18xqdBYEEv/)
```

```
## Goal

Turn the input illustration into a stained-glass window with real thickness. Use shared 2D Voronoi
     coordinates for mosaic color sampling, continuous lead seams, edge relief, subtle distortion,
     and color variation. Add a copper frame and a four-second presentation animation. Preserve the
      red-pink subject and blue-cyan and cream background in a horizontal frame. Continuous dark
     seams enclose irregular glass cells; an oblique view reveals the thin glass, metallic
     highlights, and fine surface relief.

## Input

| File | Location | Use |
|---|---|---|
| Source illustration | `input.png` | The illustration introduced after the glass plane in the video
     ; supplies the colors of the Voronoi mosaic cells. |

![Source illustration](input.png)

*Source illustration.*

## Procedure

### 1. Create a horizontal glass panel with thickness

1. Add a Plane, rotate it 90 degrees about X to stand upright, scale it to a 1.75:1 aspect ratio,
     and apply Scale.
2. Add a Solidify modifier with Thickness=0.003 m and Offset=-1.0.
3. Create a material named Stained_Glass and connect Glass BSDF to Material Output.Surface.

**Key parameters:**
- **Panel aspect ratio:** X:Y=1.75:1.
- **Solidify:** Thickness=0.003 m; Offset=-1.0.
- **Base shading:** Glass BSDF -> Material Output.Surface.

![Upright plane, Glass BSDF, and 0.003 m thickness.](images/step01-panel.jpg)

*Upright plane, Glass BSDF, and 0.003 m thickness.*

The panel now has a visible thin edge without moving its front surface.

### 2. Resample the illustration with Voronoi cells

1. Add Texture Coordinate, Mapping, Voronoi Texture, and Image Texture nodes in the Shader Editor.
     Select input.png in Image Texture.
2. Connect Generated -> Mapping.Vector -> Voronoi.Vector. Set Voronoi to 2D, F1, and Euclidean.
3. Connect Voronoi.Position -> Image Texture.Vector, then Image Texture.Color -> Glass BSDF.Color.
4. Set Image Texture.Extension to Extend.

**Key parameters:**
- **Sampling:** Generated -> Mapping -> Voronoi.Vector; Voronoi.Position -> Image Texture.Vector.
- **Voronoi:** 2D; Feature=F1; Distance=Euclidean.
- **Image boundary:** Extension=Extend.

![Voronoi Position resamples the illustration into a mosaic.](images/step02-sampling.jpg)

*Voronoi Position resamples the illustration into a mosaic.*

The continuous illustration becomes irregular colored glass cells while retaining the subject's
     silhouette.

### 3. Correct the cell and illustration proportions

1. Set the Mapping Scale before Voronoi to (1.75,1,1), restoring approximately equal cell
     proportions on the horizontal panel.
2. Insert Vector Math/Multiply between Voronoi.Position and Image Texture.Vector; set the
     multiplier to (0.5714286,1,1).

**Key parameters:**
- **Mapping Scale:** (1.75,1,1).
- **Inverse correction:** Vector Math Multiply=(1/1.75,1,1)=(0.5714286,1,1).

![Horizontal Mapping correction and its reciprocal preserve image proportions.](images/step03-
     mapping.jpg)

*Horizontal Mapping correction and its reciprocal preserve image proportions.*
```

```
The cells are no longer stretched horizontally, and the illustration retains its original
    proportions.

### 4. Extract a continuous, sharp lead mask

1. Duplicate Voronoi, connect the same Mapping.Vector, and change Feature to Distance to Edge. Keep
     dimensions, Scale, and Randomness identical.
2. Connect Distance to a first ColorRamp; use its stop positions to control seam width.
3. Connect this ColorRamp to a second ColorRamp with Constant interpolation to obtain a sharp black
    -and-white edge mask.

**Key parameters:**
- **Edge Voronoi:** 2D; Feature=Distance to Edge; shared Scale/Randomness with the sampling Voronoi
    .
- **Hard edge:** second ColorRamp Interpolation=Constant.
- **Web-example thresholds:** 0.018/0.034.

![Distance to Edge and two ColorRamps form continuous, sharp seams.](images/step04-edge-mask.jpg)

*Distance to Edge and two ColorRamps form continuous, sharp seams.*

Each glass cell is enclosed by a continuous, clearly defined lead mask.

### 5. Mix glass and metal seams and add opposing relief

1. Add a Principled BSDF with a dark, warm metallic color. Use the sharp seam mask as Mix Shader.
    Factor to mix Glass BSDF and the metal shader.
2. Connect the first continuous ColorRamp to Bump A.Height and Bump A.Normal to the metal shader,
    raising the seams slightly.
3. Duplicate Bump as Bump B, enable Invert, and connect its Normal to the glass shader, recessing
    the glass beside the seams.
4. Use two Value nodes to control Scale and Randomness jointly for both Voronoi nodes.

**Key parameters:**
- **Metal:** Metallic=0.72; Roughness=0.28.
- **Bump:** Strength=0.32; Distance=0.035; Invert on the glass branch.
- **Shared controls:** Scale=50; Randomness=1.0.

![One edge signal controls raised metal, recessed glass, and shader mixing.](images/step05-bump.jpg
    )

*One edge signal controls raised metal, recessed glass, and shader mixing.*

The seams rise above the glass, the glass edges recede, and sampling and seams share the same cells
    .

### 6. Add subtle, natural edge distortion

1. Add a 2D Noise Texture after Mapping with Scale=50.
2. Use Mix Color to blend the Mapping vector and Noise color with Factor=0.995. Connect the result
    to both Voronoi.Vector inputs.
3. Use exactly the same distorted coordinates for the sampling and Distance to Edge Voronoi nodes.

**Key parameters:**
- **Noise:** 2D; Scale=50.
- **Mix Color:** Factor=0.995.
- **Connections:** Mix Result -> both Voronoi.Vector inputs.

![A small 2D Noise contribution introduces irregular cell edges.](images/step06-distortion.jpg)

*A small 2D Noise contribution introduces irregular cell edges.*

The seams become subtly irregular while the subject's outline and cell alignment remain intact.

### 7. Add fine glass relief and color variation

1. Add a second 2D Noise Texture with Scale=100 and blend it into the glass Bump.Height at low
    strength.
2. Add Mix Color after Image Texture. Connect the image color to A and the sampling Voronoi.Color
    to B; choose Hard Light.
3. Use a small Factor for the Voronoi.Color contribution to introduce subtle brightness and hue
    variation within the cells.

**Key parameters:**
```

```
- **Fine grain:** Noise Scale=100; low-strength contribution to glass Bump.
- **Color variation:** Image Color + Voronoi.Color; Hard Light; small Factor.
- **Final brightness:** Brightness approximately 0.15; Contrast=0.00.

![Fine Noise and Hard Light add texture to the glass.](images/step07-variation.jpg)

*Fine Noise and Hard Light add texture to the glass.*

Fine grain and subtle color variation enrich the glass while preserving the illustration's red,
     blue, and cream regions.

### 8. Build the copper frame

1. Place four narrow Cubes along the panel edges and add Bevel modifiers for rounded highlight
     edges.
2. Create a warm copper Principled material with Metallic=0.82 and Roughness=0.24.
3. Parent the frame and panel to an Empty named Window_Root.

**Key parameters:**
- **Copper:** Metallic=0.82; Roughness=0.24.
- **Panel:** 6.0 x 3.43; glass thickness=0.003 m.
- **Parenting:** panel and four frame bars -> Window_Root.

![Glass panel and warm metal frame.](images/step08-frame.jpg)

*Glass panel and warm metal frame.*

![Final glass, lead, and bump node overview.](images/step08-node-overview.png)

*Final glass, lead, and bump node overview.*

A continuous copper frame surrounds the window, and the panel and frame can be transformed together
     .

### 9. Set up a black-background scene and four-second sway

1. In Eevee Next, set World Background to black. Add a front key light, side fill light, and rim
     Area Light.
2. Use an orthographic camera that includes the complete horizontal frame, with the window
     occupying most of the image.
3. Keyframe Window_Root local Y Rotation at frames 1, 49, and 97 to -8, 8, and -8 degrees. Use
     Bezier interpolation and 24 fps.

**Key parameters:**
- **Animation:** frames 1-96; 24 fps; 4 s; Y Rotation=-8 -> 8 -> -8 degrees.
- **Camera:** orthographic; horizontally centered; complete copper frame visible.
- **Render engine:** Eevee Next.

![Material brightness and surface detail in Eevee.](images/step09-lighting.jpg)

*Material brightness and surface detail in Eevee.*

The window sways gently against black, revealing thickness, reflections, and seam highlights as the
      angle changes.

### 10. Export the project, still, and animation

1. Run File > External Data > Pack Resources and save the project as asset.blend.
2. Render a front-view still at 1280 x 720 or higher to preview.png.
3. Render frames 1-96 and encode the four-second animation as H.264/yuv420p in final_effect.mp4.

**Key parameters:**
- **Still:** at least 1280 x 720; PNG.
- **Animation:** 96 frames; 24 fps; 4 s; H.264; yuv420p.

![Illustration input and completed stained-glass window.](images/step10-overview.jpg)

*Illustration input and completed stained-glass window.*

The outputs are a packed, editable project, a final still, and a four-second presentation animation
     .

## Final outputs
```

```
The window includes editable glass cells, continuous lead seams, opposing bump relief, a copper
     frame, lighting, and a presentation animation.

- ‘asset.blend‘
- ‘preview.png‘
- ‘final_effect.mp4‘
```

B.2 PYTHON IMPLEMENTATION

The reconstruction script builds a glass panel, Voronoi seams, copper frame, camera, and lights. It reads `input.png` from the script directory and saves `asset.blend`, `preview.png`, and `final_effect.mp4`. A diffuse backlight illuminates the glass from behind. Blender renders the frames, and FFmpeg encodes the animation. The script runs in background mode:

```
blender --background --factory-startup --python reproduce.py
```

```python
"""Reconstruct the stained-glass tutorial in Blender 4.2+.

Run: blender --background --factory-startup --python reproduce.py
Use -- --still-only to skip animation rendering. MP4 encoding requires ffmpeg.
"""
import math
from pathlib import Path
import shutil
import subprocess
import sys
import bpy
from mathutils import Vector

OUT = Path(__file__).resolve().parent
WIDTH, HEIGHT = 6.0, 6.0 / 1.75
FPS, FRAMES = 24, 96


def node(nodes, kind, name, position):
    n = nodes.new(kind)
    n.name = n.label = name
    n.location = position
    return n


def stained_glass():
    material = bpy.data.materials.new('Stained_Glass')
    material.use_nodes = True
    nodes, links = material.node_tree.nodes, material.node_tree.links
    nodes.clear()
    tex = node(nodes, 'ShaderNodeTexCoord', 'Panel coordinates', (-1500, 300))
    mapping = node(nodes, 'ShaderNodeMapping', 'Aspect ratio 1.75', (-1300, 300))
    mapping.inputs['Scale'].default_value = (1.75, 1, 1)
    links.new(tex.outputs['Generated'], mapping.inputs['Vector'])
    noise = node(nodes, 'ShaderNodeTexNoise', 'Edge distortion', (-1300, -50))
    noise.noise_dimensions = '2D'
    noise.inputs['Scale'].default_value = 50
    links.new(mapping.outputs['Vector'], noise.inputs['Vector'])
    warp = node(nodes, 'ShaderNodeMixRGB', 'Shared distorted coordinates', (-1050, 300))
    warp.inputs[0].default_value = 0.995
    links.new(noise.outputs['Color'], warp.inputs[1])
    links.new(mapping.outputs['Vector'], warp.inputs[2])
    scale = node(nodes, 'ShaderNodeValue', 'Cell scale', (-1050, -120))
    scale.outputs[0].default_value = 50
    random = node(nodes, 'ShaderNodeValue', 'Cell randomness', (-1050, -230))
    random.outputs[0].default_value = 1
    cells = node(nodes, 'ShaderNodeTexVoronoi', 'Mosaic sampling', (-800, 450))
    edge = node(nodes, 'ShaderNodeTexVoronoi', 'Distance to Edge', (-800, 0))
    for n in (cells, edge):
        n.voronoi_dimensions = '2D'
        n.distance = 'EUCLIDEAN'
        links.new(warp.outputs[0], n.inputs['Vector'])
        links.new(scale.outputs[0], n.inputs['Scale'])
        links.new(random.outputs[0], n.inputs['Randomness'])
    cells.feature = 'F1'
    edge.feature = 'DISTANCE_TO_EDGE'
    inverse = node(nodes, 'ShaderNodeVectorMath', 'Inverse aspect ratio', (-550, 500))
```

```
    inverse.operation = 'MULTIPLY'
    inverse.inputs[1].default_value = (1 / 1.75, 1, 1)
    links.new(cells.outputs['Position'], inverse.inputs[0])
    image = node(nodes, 'ShaderNodeTexImage', 'Source illustration', (-320, 550))
    image.image = bpy.data.images.load(str(OUT / 'input.png'), check_existing=True)
    image.image.pack()
    image.extension = 'EXTEND'
    links.new(inverse.outputs[0], image.inputs['Vector'])
    # Hard Light(A, B) is Overlay(B, A); blend with A at 0.07.
    hard = node(nodes, 'ShaderNodeMixRGB', 'Hard Light', (-60, 750))
    hard.blend_type = 'OVERLAY'
    hard.inputs[0].default_value = 1
    links.new(cells.outputs['Color'], hard.inputs[1])
    links.new(image.outputs['Color'], hard.inputs[2])
    color = node(nodes, 'ShaderNodeMixRGB', 'Cell color variation', (-60, 550))
    color.inputs[0].default_value = 0.07
    links.new(image.outputs['Color'], color.inputs[1])
    links.new(hard.outputs[0], color.inputs[2])
    bright = node(nodes, 'ShaderNodeBrightContrast', 'Brightness', (160, 550))
    bright.inputs['Bright'].default_value = 0.15
    bright.inputs['Contrast'].default_value = 0
    links.new(color.outputs[0], bright.inputs['Color'])
    ramp = node(nodes, 'ShaderNodeValToRGB', 'Continuous seam profile', (-540, 0))
    for e, pos, rgba in zip(ramp.color_ramp.elements, (0.018, 0.034),
                            ((0, 0, 0, 1), (1, 1, 1, 1))):
        e.position, e.color = pos, rgba
    links.new(edge.outputs['Distance'], ramp.inputs[0])
    mask = node(nodes, 'ShaderNodeValToRGB', 'Sharp glass mask', (-300, 60))
    mask.color_ramp.interpolation = 'CONSTANT'
    mask.color_ramp.elements[1].position = 0.5
    links.new(ramp.outputs['Color'], mask.inputs[0])
    height = node(nodes, 'ShaderNodeMath', 'Raised seam height', (-300, -200))
    height.operation = 'SUBTRACT'
    height.inputs[0].default_value = 1
    links.new(ramp.outputs['Color'], height.inputs[1])
    lead_bump = node(nodes, 'ShaderNodeBump', 'Raised lead', (160, -100))
    lead_bump.inputs['Strength'].default_value = 0.32
    lead_bump.inputs['Distance'].default_value = 0.035
    links.new(height.outputs[0], lead_bump.inputs['Height'])
    fine = node(nodes, 'ShaderNodeTexNoise', 'Fine glass grain', (-540, -450))
    fine.noise_dimensions = '2D'
    fine.inputs['Scale'].default_value = 100
    links.new(mapping.outputs['Vector'], fine.inputs['Vector'])
    grain = node(nodes, 'ShaderNodeMath', 'Subtle grain amplitude', (-300, -450))
    grain.operation = 'MULTIPLY'
    grain.inputs[1].default_value = 0.03
    links.new(fine.outputs['Fac'], grain.inputs[0])
    combine = node(nodes, 'ShaderNodeMath', 'Seams and grain', (-50, -350))
    combine.operation = 'ADD'
    links.new(height.outputs[0], combine.inputs[0])
    links.new(grain.outputs[0], combine.inputs[1])
    glass_bump = node(nodes, 'ShaderNodeBump', 'Recessed glass', (160, 200))
    glass_bump.invert = True
    glass_bump.inputs['Strength'].default_value = 0.32
    glass_bump.inputs['Distance'].default_value = 0.035
    links.new(combine.outputs[0], glass_bump.inputs['Height'])
    glass = node(nodes, 'ShaderNodeBsdfGlass', 'Colored glass', (400, 420))
    glass.inputs['Roughness'].default_value = 0.12
    glass.inputs['IOR'].default_value = 1.5
    links.new(bright.outputs[0], glass.inputs['Color'])
    links.new(glass_bump.outputs['Normal'], glass.inputs['Normal'])
    lead = node(nodes, 'ShaderNodeBsdfPrincipled', 'Metal lead', (400, -150))
    lead.inputs['Base Color'].default_value = (0.018, 0.012, 0.016, 1)
    lead.inputs['Metallic'].default_value = 0.72
    lead.inputs['Roughness'].default_value = 0.28
    links.new(lead_bump.outputs['Normal'], lead.inputs['Normal'])
    mix = node(nodes, 'ShaderNodeMixShader', 'Lead and glass', (720, 300))
    links.new(mask.outputs['Color'], mix.inputs[0])
    links.new(lead.outputs[0], mix.inputs[1])
    links.new(glass.outputs[0], mix.inputs[2])
    output = node(nodes, 'ShaderNodeOutputMaterial', 'Surface', (940, 300))
    links.new(mix.outputs[0], output.inputs['Surface'])
    return material


def frame_bar(name, location, scale, material, parent):
```

```
    bpy.ops.mesh.primitive_cube_add()
    obj = bpy.context.object
    obj.name, obj.location, obj.scale = name, location, scale
    bpy.ops.object.transform_apply(location=False, rotation=False, scale=True)
    obj.parent = parent
    bevel = obj.modifiers.new('Rounded frame edges', 'BEVEL')
    bevel.width, bevel.segments = 0.035, 3
    obj.data.materials.append(material)


def aim(obj, point=(0, 0, 0)):
    obj.rotation_euler = (Vector(point) - obj.location).to_track_quat('-Z', 'Y').to_euler()


def build_scene():
    bpy.ops.object.select_all(action='SELECT')
    bpy.ops.object.delete(use_global=False)
    # The parent makes the panel upright; Window_Root keeps panel-local axes.
    orientation = bpy.data.objects.new('Upright_Orientation', None)
    bpy.context.collection.objects.link(orientation)
    orientation.rotation_euler.x = math.pi / 2
    root = bpy.data.objects.new('Window_Root', None)
    bpy.context.collection.objects.link(root)
    root.parent = orientation
    bpy.ops.mesh.primitive_plane_add()
    panel = bpy.context.object
    panel.name, panel.parent = 'Glass_Panel', root
    panel.scale = (WIDTH / 2, HEIGHT / 2, 1)
    bpy.ops.object.transform_apply(location=False, rotation=False, scale=True)
    solid = panel.modifiers.new('Glass thickness', 'SOLIDIFY')
    solid.thickness, solid.offset = 0.003, -1
    panel.data.materials.append(stained_glass())
    copper = bpy.data.materials.new('Copper_Frame')
    copper.use_nodes = True
    bsdf = copper.node_tree.nodes.get('Principled BSDF')
    bsdf.inputs['Base Color'].default_value = (0.19, 0.055, 0.022, 1)
    bsdf.inputs['Metallic'].default_value = 0.82
    bsdf.inputs['Roughness'].default_value = 0.24
    w, h, bar, depth = WIDTH / 2, HEIGHT / 2, 0.075, 0.065
    for sign in (-1, 1):
        frame_bar('Frame_Horizontal', (0, sign * (h + bar), 0),
                  (w + bar, bar, depth), copper, root)
        frame_bar('Frame_Vertical', (sign * (w + bar), 0, 0),
                  (bar, h + bar, depth), copper, root)
    for frame, degrees in ((1, -8), (49, 8), (97, -8)):
        root.rotation_euler.y = math.radians(degrees)
        root.keyframe_insert(data_path='rotation_euler', index=1, frame=frame)
    # Blender's default keyframe interpolation is Bezier.
    scene = bpy.context.scene
    scene.frame_start, scene.frame_end = 1, FRAMES
    scene.render.fps = FPS
    scene.frame_set(1)
    scene.world = bpy.data.worlds.new('Black_Background')
    scene.world.use_nodes = True
    scene.world.node_tree.nodes['Background'].inputs['Color'].default_value = (0, 0, 0, 1)
    # A diffuse backlight makes transmitted colors visible against the black world.
    bpy.ops.mesh.primitive_plane_add()
    backlight = bpy.context.object
    backlight.name, backlight.parent = 'Diffuse_Backlight', root
    backlight.location = (0, 0, -0.18)
    backlight.scale = (WIDTH / 2, HEIGHT / 2, 1)
    light_material = bpy.data.materials.new('Backlight')
    light_material.use_nodes = True
    light_nodes = light_material.node_tree.nodes
    light_nodes.clear()
    emission = light_nodes.new('ShaderNodeEmission')
    emission.inputs['Strength'].default_value = 0.8
    light_out = light_nodes.new('ShaderNodeOutputMaterial')
    light_material.node_tree.links.new(emission.outputs[0], light_out.inputs['Surface'])
    backlight.data.materials.append(light_material)
    for name, location, energy, size in (
        ('Key', (-3, -4, 4), 1300, 5),
        ('Fill', (4, -2, 1), 800, 4),
        ('Rim', (-2, 3, 2), 1700, 5),
    ):
        data = bpy.data.lights.new(name, 'AREA')
```

```
        data.energy, data.shape, data.size = energy, 'DISK', size
        light = bpy.data.objects.new(name, data)
        scene.collection.objects.link(light)
        light.location = location
        aim(light)
    camera = bpy.data.objects.new('Camera', bpy.data.cameras.new('Camera'))
    scene.collection.objects.link(camera)
    camera.location = (0, -12, 0.8)
    aim(camera)
    camera.data.type, camera.data.ortho_scale = 'ORTHO', 7.15
    scene.camera = camera
    scene.render.engine = 'CYCLES'
    scene.cycles.samples = 32
    scene.cycles.use_denoising = True
    preferences = bpy.context.preferences.addons['cycles'].preferences
    for backend in ('OPTIX', 'CUDA', 'HIP', 'METAL', 'ONEAPI'):
        try:
            preferences.compute_device_type = backend
            preferences.get_devices()
        except (TypeError, RuntimeError):
            continue
        if any(d.type == backend for d in preferences.devices):
            for device in preferences.devices:
                device.use = device.type == backend
            scene.cycles.device = 'GPU'
            break
    scene.render.resolution_x, scene.render.resolution_y = 1440, 900
    scene.render.resolution_percentage = 100
    scene.render.image_settings.file_format = 'PNG'
    scene.render.film_transparent = False
    scene.view_settings.view_transform = 'AgX'
    return scene


def main():
    scene = build_scene()
    bpy.ops.file.pack_all()
    bpy.ops.wm.save_as_mainfile(filepath=str(OUT / 'asset.blend'))
    scene.render.filepath = str(OUT / 'preview.png')
    bpy.ops.render.render(write_still=True)
    if '--still-only' in sys.argv:
        return
    encoder = shutil.which('ffmpeg')
    if not encoder:
        raise RuntimeError('ffmpeg is required to encode final_effect.mp4')
    frames = OUT / 'frames'
    frames.mkdir(exist_ok=True)
    scene.render.filepath = str(frames / 'frame_')
    bpy.ops.render.render(animation=True)
    subprocess.run([encoder, '-y', '-framerate', str(FPS), '-i',
                    str(frames / 'frame_%04d.png'), '-c:v', 'libx264',
                    '-pix_fmt', 'yuv420p', '-crf', '18', '-movflags', '+faststart',
                    str(OUT / 'final_effect.mp4')], check=True)


if __name__ == '__main__':
    main()
```

C PROCEDURAL KNOWLEDGE EXAMPLE

Figure 8 shows the progression from the original Duck GiGi to its animated iridescent appearance. Four reusable procedural knowledge units contribute to the result. Each unit describes when to use the method, how to apply it, and what to check, followed by a short code excerpt.

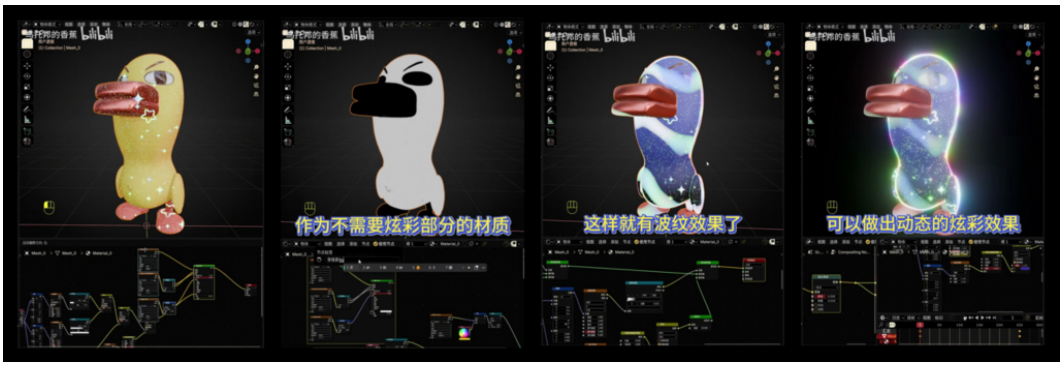


Figure 8: Layered construction of Duck GiGi's iridescent material. From left to right, the tutorial frames show the starting duck, separate materials for regions that remain non-iridescent, wave-like color variation, and the final setup for animated iridescence with a glowing outline. The Chinese subtitles in the source frames describe these material steps.

1. Iridescent base color. **When:** A surface needs broad, irregular color regions that can change over time.
Method: Use 4D Noise to drive a blue-to-cyan color ramp; animate `W` to shift the pattern.
Check: Color varies continuously across the body and changes as `W` changes.

```
noise = nodes.new('ShaderNodeTexNoise')
noise.noise_dimensions = '4D'
noise.inputs['Scale'].default_value = 2.0
noise.inputs['Detail'].default_value = 12.0
noise.inputs['Roughness'].default_value = 0.55
# Noise.Fac -> ColorRamp.Fac -> Principled.Base Color
```

2. Flowing iridescent bands. **When:** A base material needs directional color variation rather than a static mottled pattern.
Method: Pass mapped coordinates through a Wave Texture and a second color ramp, then combine its colors with the noise-based base using Screen blending. Animate the wave phase for motion.
Check: Bright bands cross the body diagonally and move smoothly without breaking the underlying color pattern.

```
wave = nodes.new('ShaderNodeTexWave')
wave.wave_type = 'RINGS'
wave.inputs['Scale'].default_value = 1.10
wave.inputs['Distortion'].default_value = 4.0
# Wave.Fac -> second ColorRamp
# ScreenMix(Noise colors, Wave colors, factor=0.42)
# -> Principled.Base Color
```

3. Emissive star details. **When:** A surface needs many small highlights without placing each one by hand.
Method: Threshold a high-scale 4D Voronoi texture to isolate sparse points, feed them into an Emission shader, and add that shader to the body material. Attach larger star shapes separately for distinct highlights.
Check: Fine points remain sparse and bright, while the larger stars stay attached to the moving body.

```
stars = nodes.new('ShaderNodeTexVoronoi')
stars.voronoi_dimensions = '4D'
stars.inputs['Scale'].default_value = 85.0
stars.inputs['Randomness'].default_value = 0.70
# Voronoi.Distance -> threshold ColorRamp
# -> Emission -> Add Shader
```

4. Luminous silhouette. **When:** A character needs a clear luminous outline against a dark background.
Method: Assign an Emission material to separate outline geometry around the body and wings, then use compositor glow to soften the light.
Check: The outline follows the moving character without obscuring its shape or facial features.

```
rim = nodes.new('ShaderNodeEmission')
rim.inputs['Color'].default_value = (0.75, 1.00, 0.82, 1.0)
rim.inputs['Strength'].default_value = 1.0
# Emission -> Material Output.Surface
# Assign this material to the outline geometry.
```

These units can be reused independently: an agent retrieves an applicable method, then adapts its material regions, color ramps, pattern scale, and animation timing to a new asset.

D BLENDERLORE SKILL: WORKFLOW AND EVALUATION CONFIGURATION

BlenderLore Skill routes the input to the appropriate stage of the asset-creation workflow. Given a video URL or file, it extracts an illustrated tutorial before retrieving relevant procedural knowledge, generating a Blender asset, and checking the result. Given an existing illustrated tutorial, it skips video extraction and proceeds to knowledge-guided generation and verification. Given a specified creation task, such as a task Markdown file with its inputs and references, it starts from that task and uses relevant knowledge to create and verify a new asset. Thus, tutorial extraction is conditional on the input; the skill can also be used to create assets for goals that are already specified.

COMPLETE WORKFLOW ENTRY POINT

The full Skill entry point below shows these routes. Its referenced workflow modules and knowledge documents provide the corresponding implementation and guidance.

```
---
name: blender-pipeline
description: Extract a tutorial or reconstruct a Blender result from video links, local videos with
      supporting assets, or Markdown tutorials in Codex chat. Clarify the intended output when
     unspecified; also route explicit showcase reproduction and independent asset edits.
---

# Blender Pipeline

## Start from the Codex client

This is the user-facing Codex launcher. If the requested output is unclear, ask
whether the user wants only a tutorial or a complete Blender reconstruction;
wait for their choice before launching. Do not ask again when the goal is explicit.

Read [references/codex-launch.md](references/codex-launch.md), interpret the
user's natural-language inputs, and invoke this skill's
`scripts/launch_from_codex.py` yourself. It calls the same maintained pipeline
as the API entrypoint. Tutorial-only stops after preparation; full reconstruction
continues through knowledge retrieval → Blender generation → render and review.

A video with an accompanying `.blend`, preview, texture, or dependency bundle
retains those inputs in either selected route; it is not an independent edit.
A supplied Markdown tutorial skips video extraction: prepare it for tutorial-only,
or continue knowledge-backed generation for reconstruction. No RW1 is required.

Use the user's requested output location or established data root. Ask only for
missing information needed to run, not command-line flags or API credentials.
The Codex path uses the signed-in Codex CLI; the user need not run a Python
launcher manually. `run_codex.py` remains a compatibility entrypoint, not the
client interaction. Requests to explain, inspect, or dry-run do not authorize
an actual reconstruction.

## Extract a tutorial from video

For tutorial-only, pass `--extract-only` and do not start reconstruction. Read
[tutorial-extraction/SKILL.md](tutorial-extraction/SKILL.md) for illustrated
Markdown, learner inputs and the separate rubric; both replay routes reuse it.
```

```
## Reproduce a maintained showcase

Read [reproduction/SKILL.md](reproduction/SKILL.md) only for an explicitly
requested catalog recipe or hybrid-delivery workflow. A tutorial's subject
also appearing on the showcase does not change ordinary replay routing.
Public previews do not authorize downloading the underlying assets.

## Generate from zero

For ordinary video/tutorial requests use the client launcher above. Use the
specialized generation mode for an explicit from-zero reference/text task or
low-level generation work, with no authoritative starting project.

Read [generation/SKILL.md](generation/SKILL.md), then load
[knowledge/generation.md](knowledge/generation.md),
[knowledge/evidence-and-routing.md](knowledge/evidence-and-routing.md), and only
the relevant sections of
[knowledge/rendering-and-dynamics.md](knowledge/rendering-and-dynamics.md).

## Edit an existing asset

Use editing for a standalone change request on an existing asset, without a
video/tutorial reconstruction workflow. A generated replacement object within
that scoped change remains part of the edit.

Read [editing/SKILL.md](editing/SKILL.md), then load
[knowledge/editing.md](knowledge/editing.md) and only the relevant sections of
[knowledge/rendering-and-dynamics.md](knowledge/rendering-and-dynamics.md).

## Shared rules

- Keep mode-specific code, tests, manifests, and outputs separate. Share only
  stable utilities and contracts.
- Preserve verified source intent and non-target dimensions. A successful file
  write is not a visual or semantic pass.
- Use explicit input/output paths and environment-backed credentials; never
  embed workstation, mounted-volume, or server paths in committed code.
- Load [knowledge/operations-and-knowledge.md](knowledge/operations-and-knowledge.md)
  only for orchestration, paid calls, knowledge maintenance, or publication.
- Start knowledge lookup at [knowledge/index.md](knowledge/index.md); do not
  load every reference for an ordinary single-mode task.

Package maintenance uses `scripts/validate_package.py`, not every ordinary launch.
```

CONFIGURATION USED IN EVALUATION

BlenderLore-Bench supplies each creation task and its inputs, so the evaluation does not require video extraction. The Codex +Knowledge configurations use the same task-facing Skill entry point below to retrieve and apply a frozen procedural knowledge library while the coding agent creates the asset. This configuration does not invoke the dataset-construction pipeline. Its directory structure is:

```
blender-pipeline/
  SKILL.md
  knowledge/editing.md
  knowledge/evidence-and-routing.md
  knowledge/generation.md
  knowledge/index.md
  knowledge/manifest.json
  knowledge/manifest.schema.json
  knowledge/operations-and-knowledge.md
  knowledge/rendering-and-dynamics.md
```

The manifest catalogs documents using `path`, `scope`, `title`, and `keywords`. Each knowledge rule is a Markdown section with four fields: *Applies when*, *Rule*, *Verify*, and *Search terms*. The agent selects a relevant document, searches for task concepts, and reads the matching sections. If no applicable knowledge is found, it continues with its native capabilities.

```
---
name: blender-pipeline
description: Use BlenderLore's Knowledge library to help complete Blender task files, including
     generation, editing, materials, rendering and animation.
```

```
---

# Blender Knowledge

Read the user's task file and use relevant knowledge as implementation guidance.
The task defines the goal, inputs and outputs; no tutorial format is required.

## Data structure

- `knowledge/manifest.json`: document catalog; each `documents[]` entry has
  `path`, `scope`, `title`, `keywords`. Paths are relative to `knowledge/`.
- Documents are Markdown. Each `##` section is one rule with `Applies when`,
  `Rule`, `Verify` and `Search terms`.
- [knowledge/index.md](knowledge/index.md) is a short topic-to-file map.

## Retrieval

Select a relevant file from the map or catalog. Search it using task concepts,
Blender feature names and their English/Chinese equivalents, for example:

```sh
rg -n -i 'material graph|roughness|材质' "<skill-dir>/knowledge/editing.md"
```

Read the matching `##` section through the next heading, not the whole library.
Expand the search only when needed. Apply rules only within their stated scope;
unrelated pipeline/harness protocols are not default requirements for a task.
If nothing relevant is found, continue with Codex's normal Blender capabilities.

This is a reference library, not an execution harness. Use the current session;
do not build an index, start a nested pipeline, or create extra reports merely
to use Knowledge.
```

E GEMINI CLI EVALUATION PROTOCOL

We use Gemini CLI v0.61.0 with `gemini-3.8-flash` at LOW thinking level as the evaluator. For each task, it receives the task description, expert-authored rubric, reference images, the submitted `.blend` file and `build.py`, and available preview images or animation frames. It scores each rubric item according to its original PASS, PARTIAL, and FAIL conditions, assigning a normalized score of 1, 0.5, or 0. Alternative implementations receive credit when they satisfy the same requirements.

The following is a representative judging input; placeholders stand for task-specific text and evidence files. We deliberately instruct the judge to require observable support for each score. In particular, source-code intent, object names, and unsupported claims do not establish that a functional effect occurred.

```
You are the strict judge for one BlenderLore task. Evaluate the
submitted images, build.py, and submission.blend against the
original rubric. Inspect the submitted artifacts and perform
the required functional checks when visual evidence alone is
insufficient. Do not modify or improve the submitted asset.

Award credit only for requirements directly demonstrated by
the evidence or verified in the submitted artifacts. Do not
infer hidden quality, intended behavior, or implementation
from filenames, code declarations, or claims. If evidence is
ambiguous, choose the lower applicable score. If a required
check has not been performed, mark that condition as
unverified; use FAIL only when an observed result satisfies
the rubric's FAIL condition. Apply the original PASS, PARTIAL,
and FAIL rules without adding new thresholds.

Return one JSON object with "scores" and "rationales" keyed
by every rubric ID. Each score must be 0, 0.5, or 1. Explain
the observed evidence supporting each decision.

Task instructions:
```

```
<TASK_MARKDOWN>

Rubric:
<RUBRIC_JSON>

Submitted artifacts:
@build.py
submission.blend

Reference images and submitted visual evidence:
<IMAGE_FILES>
```

F BENCHMARK SEPARATION FROM THE KNOWLEDGE LIBRARY

We constructed BlenderLore-Bench tasks from tutorials first published after the final update of the BlenderLore knowledge library. Human experts checked each candidate creator's earlier publications for substantially similar tutorials. They also inspected related videos recommended by the hosting platform and searched for similar tutorials using relevant keywords. Candidates with closely matching earlier tutorials were excluded.

The experts had manually reviewed every asset in the BlenderLore dataset. They additionally screened candidate tasks against those assets and excluded tasks whose target assets were highly similar to existing ones. Together, the publication-date filter and manual checks substantially reduce the risk that benchmark tasks are already represented in the procedural knowledge library.

G JUDGE–HUMAN AGREEMENT

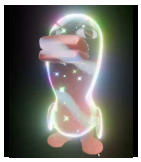
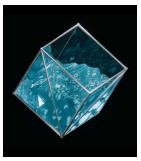
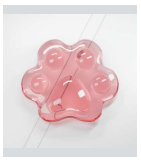
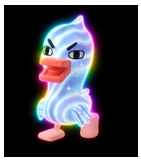
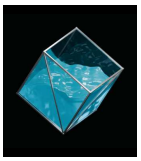
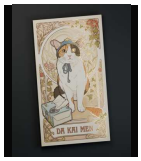
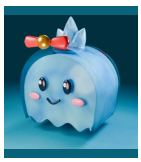
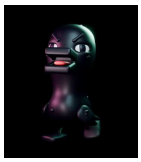
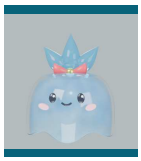
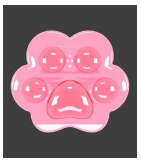
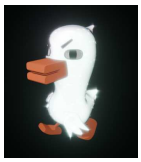
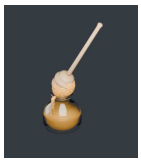
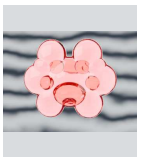
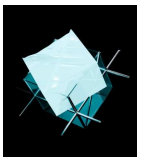
We randomly sampled 100 evaluated outputs from different models. Three human experts scored each output independently using the BlenderLore-Bench rubrics. We also evaluated each output in three independent Gemini CLI runs with `gemini-3.8-flash`. Table 6 reports intraclass correlation coefficients (ICC) for human scores, Gemini CLI scores, and agreement between Gemini CLI and the mean human score.

Score comparison	ICC
Among three human experts	0.75
Gemini CLI vs. mean human score	0.77
Among three independent Gemini CLI runs	0.92

Table 6: Scoring agreement on 100 randomly sampled model outputs.

The agreement among human experts suggests that the rubrics support reasonably consistent judgments. Gemini CLI's agreement with the mean human score is of a similar magnitude, supporting its use as an automated judge for these tasks. Its higher agreement across repeated runs indicates that its scores are comparatively stable.

H QUALITATIVE EVALUATION EXAMPLES

Model Name	Honey Jar B03	Wave Display B97	Water Character B127	Glass Paw B36	Iridescent Material B54	Foil Card B18
Task Reference						
Ours (GPT-6-Astra + BlenderLore)						
GPT-6-Astra						
GPT-5.6-Sol						
Gemini-3.8-Flash						
GLM-5.3-Flash		Task failed				
Kimi K3						
DeepSeek-V4.1-Flash						
Claude Opus 5	Task failed					