This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

BlenderGym: Benchmarking Foundational Model Systems for Graphics Editing

Yunqi Gu
Stanford University
yrichard@stanford.edu

Ian Huang
Stanford University
ianhuang@stanford.edu

Jihyeon Je
Stanford University
jihyeonj@stanford.edu

Guandao Yang
Stanford University
guandao@stanford.edu

Leonidas Guibas
Stanford University
guibas@cs.stanford.edu

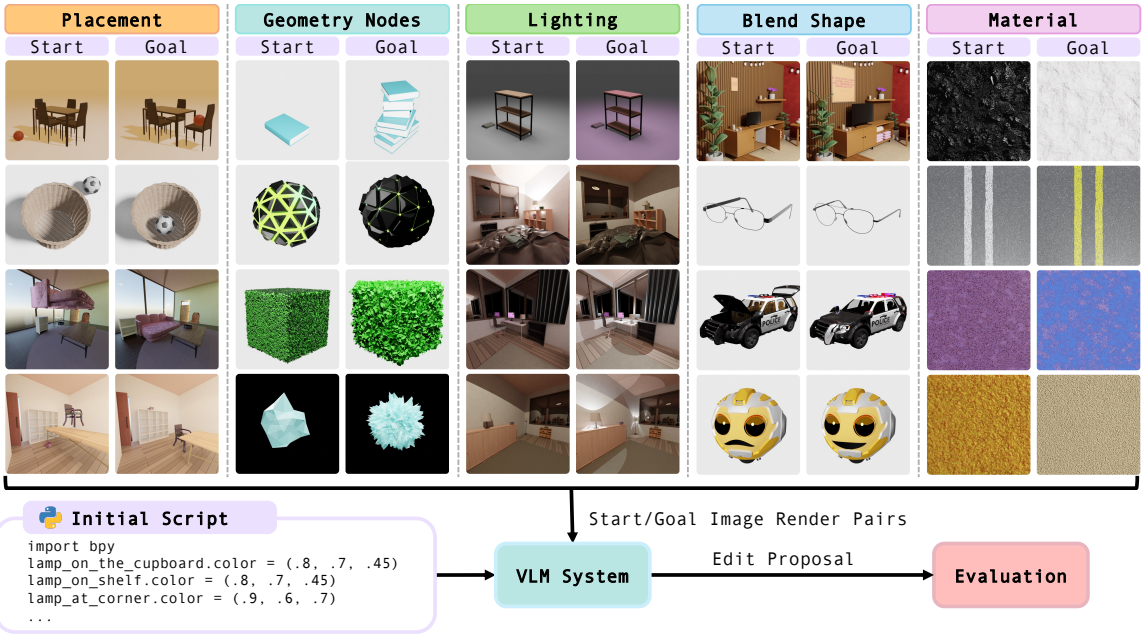


Figure 1. Example task instances of BlenderGym, a 3D graphics benchmark that tasks VLMs with 3D scene reconstruction through code editing. BlenderGym consists of 245 handcrafted start-goal scene pairs across five key graphics editing tasks: object placement, lighting adjustment, procedural material editing, blend shape manipulation, and procedural geometry editing.

Abstract

3D graphics editing is crucial in applications like movie production and game design, yet it remains a time-consuming process that demands highly specialized domain expertise. Automating this process is challenging because graphical editing requires performing a variety of tasks, each requiring distinct skill sets. Recently, vision-language models (VLMs) have emerged as a powerful framework for automating the editing process, but their development and evaluation are bottlenecked by the lack of a comprehensive benchmark that requires human-level perception and presents real-world editing complexity. In this work, we present ***BlenderGym****, the first comprehensive VLM system benchmark for 3D graphics editing. BlenderGym evaluates VLM systems through code-based 3D reconstruction tasks. We evaluate closed- and open-source VLM systems and observe that even the state-of-the-art VLM system struggles with tasks relatively easy for human Blender users. Enabled by BlenderGym, we study how inference scaling techniques impact VLM's performance on graphics editing tasks. Notably, our findings reveal that the verifier used to guide the scaling of generation can itself be improved through inference scaling, complementing recent insights on inference scaling of LLM generation in coding and math tasks. We further show that inference compute is not uniformly effec-*

tive and can be optimized by strategically distributing it between generation and verification.

1. Introduction

Producing 3D graphics for video games and movies is tedious and challenging. Beyond mastering 3D software such as Blender, Unity, Unreal, and Maya, artists must be skilled in diverse tasks, each with its own workflow—from animation to material design and geometry modeling. To address these diverse and difficult tasks, significant efforts have focused on automating graphics editing to accelerate production and reduce barriers to entry.

Prior works [11, 12, 14, 15, 18, 19, 35, 39, 42] demonstrate that large language models [22, 28, 37] can automate tasks in 3D graphics editing, such as layout generation, animation sequence creation, and texture generation. Building on this, Huang et al. [20] explore using vision-language models (VLMs) [29] to edit 3D procedural geometry, material, and lighting by leveraging their visual understanding to guide edits. As multi-modal foundation models continue advancing, they could potentially play an increasingly central role in 3D graphics editing. In contrast to the growing applications of VLMs in the field, their evaluation remains underexplored, making it difficult for the community to compare different approaches and identify where improvements are needed.

Despite the growing demand for evaluation metrics for graphics, prior attempts[11, 18, 20, 35, 39, 42] are not comprehensive enough. Some primarily rely on limited sample sets within the narrow task settings in which their method specializes and, therefore, do not allow cross-comparison between different approaches [18, 20, 35]. Others are limited by the lack of reliable, cost-effective, and quantitative metrics [7, 9, 18, 20, 23, 24, 38, 39, 42]. Unlike fields such as mathematics, graphics editing generally lacks standardized answers, making human evaluation a common choice [18, 20, 39, 42], which is often slow and expensive. Some works, therefore, have introduced the AI-as-judge paradigm [7, 9, 23, 24, 38], but this raises concerns about reliability due to models' lack of reasoning capabilities and, more critically, the bias when the same AI judge model is also involved in the pipeline being judged.

To this end, we introduce BlenderGym (Fig. 1), a 3D graphics editing benchmark for VLM systems. BlenderGym evaluates VLM systems by tasking them with reconstructing a goal Blender scene from a start Blender scene through Python program editing. It (1) covers a wide variety of task settings in 3D graphics editing, (2) allows easy plug-and-play of customized VLM systems, and (3) provides fixed start-goal scene pairs to enable quantitative evaluation of editing results, removing human or AI involvement in evaluation. BlenderGym covers five key task settings of 3D graphics editing: object placement, lighting adjustment, procedural material editing, blend shape manipulation, and procedural geometry editing. These tasks are chosen to meet the demand of downstream graphics editing applications and can therefore reflect a VLM system's abilities in the field. We also provide human baselines to contextualize the performance of VLM systems and demonstrate the solvability and validity of each task.

In addition to VLM system evaluation, BlenderGym enables us to study inference scaling to enhance VLM systems' graphics editing capabilities. As shown in [6, 34], scaling inference compute can be effective *if* a capable *verifier* exists to select among a large set of candidate answers reliably. Specifically, given that VLMs can function as verifiers that select among their own outputs based on visual comparisons [20], we demonstrate on BlenderGym that (1) VLMs used as verifiers to guide generation could also benefit from inference scaling, complementing their established scaling laws as generators [6], and that (2) inference compute is not uniformly effective and can be optimized by strategically allocating it between VLM generation and verification. When more compute is available, a higher share of verification yields greater benefits.

In summary, our contributions are as follows:

- We introduce BlenderGym, a VLM system benchmark on 3D graphics that tasks VLMs with code-based 3D scene reconstruction. It evaluates VLM systems quantitatively on various realistic editing tasks, spanning across lighting, material, blend shape, geometry, and placement.
- We provide an evaluation of state-of-the-art (SOTA) closed- and open-source VLM systems on BlenderGym, as well as a comparison to human baselines.
- We apply our benchmark to study inference time scaling for graphics editing, particularly the effectiveness of scaling the verifier in the paradigm introduced in [6].

2. Related Works

Automating graphics editing As 3D graphics editing is a time-consuming and heavily specialized task, automating it can save substantial production costs and allow faster content generation. Following the remarkable success of large foundation models, the AI and graphics community has made great efforts to adapt VLM/LLMs for 3D graphics editing in tasks such as scene generation [18, 36, 41] and material editing [8, 16, 20, 33, 40, 45]. VLMs, in particular, are well-suited for tasks that require visual reasoning and textual-visual relationship understanding. BlenderAlchemy [20], for example, has demonstrated promising results in material editing. 3D-GPT [35] shows decent language-driven 3D modeling capabilities. Both LLPlace [41] and SceneCraft [18] leverage the spatial reasoning of VLM to generate 3D layouts. While these systems show significant success in their self-designed evalua-

tion, the lack of a comprehensive benchmark makes it challenging to properly cross-compare their approaches even in the same task setting.

Benchmarks for graphics editing Following the rise of graphics editing tools, the need for a comprehensive graphics benchmark has become increasingly critical. Many existing benchmarks focus on some specific tasks of graphics editing, such as material[20], lighting [20], and the artistic quality of images [21, 25]. Other benchmarks attempt to measure semantic understanding in symbolic graphics programs [30, 43], yet they are tailored toward their domain, like SVG programs. This makes it challenging to account for the multi-faceted nature of 3D graphics editing, which includes tasks like material editing, lighting adjustments, and complex scene composition.

Benchmarks for VLMs Recent efforts have focused on creating comprehensive benchmarks to evaluate VLMs across multiple dimensions. UniBench [2], for instance, provides an evaluation framework for nearly 60 publicly available VLMs, assessing capabilities ranging from object recognition and relational understanding to robustness. Although UniBench [2] and similar frameworks [13, 17, 27, 44] provide a broader evaluation of vision-language capabilities of VLMs, they are not tailored to 3D graphics. This gap underscores the need for a specialized benchmark that addresses the unique demand of 3D graphics editing.

3. Methods

We aim to enable comprehensive comparison and analysis of VLM systems on 3D graphics editing tasks, which are currently bottlenecked by (1) the incomprehensive coverage of task settings with current evaluation approaches and (2) the absence of a robust evaluation metric due to unscalability of human evaluation and unreliability of AI-judge. To overcome these two challenges, we introduce BlenderGym—the first VLM systems benchmark tailored to 3D graphics (Fig. 1). BlenderGym features in (1) the comprehensive coverage of essential graphics editing tasks with support for a wide range of VLM systems, (2) quantitative evaluation with image and 3D metrics, eliminating the need for human or AI-based evaluations, and (3) support for inference time scaling experiments on graphics editing tasks.

3.1. Data Collection and Pipeline Design

BlenderGym comprises of 245 hand-crafted Blender scenes across five key graphics editing tasks: procedural geometry editing, lighting adjustments, procedural material design, blend shape manipulation, and object placement (50/40/40/75/40 instances, respectively). Each instance in BlenderGym presents a reconstruction task from a start scene to a goal scene, where the start scene serves as the original input for the VLM system, and the goal scene represents the desired output state that the VLM system is tasked with reconstructing. Each start-goal instance includes a base Blender file of the scene setup, a pair of Python scripts that generate the start and goal scene, rendered images for both scenes, and a language description of the differences between the two scenes. Example instances of each task are demonstrated in Fig. 1. For variety, we incorporate task instances based on Blender prototype from BlenderAlchemy [20], Infinigen [32], and Blenderkit [5].

VLM systems run on BlenderGym in the following steps, as shown in Fig. 1. To prepare the input data, we first run and render the start and goal Python scripts on the base Blender file. The VLM system then compares the start and goal rendered images, analyzes their visual differences, and modifies the Python script of the start scene to reconstruct the goal scene. Finally, we compute the distance between the goal scene and the VLM system output using a set of human-aligned distance metrics.

3.1.1 Task Settings and Evaluation Metrics

Procedural Material The VLM system is expected to perceive and modify a procedural surface material, requiring an understanding of color and texture. The VLM system should edit the Python script with Blender Python API(BPY) and the Infinigen package[32]. The results are evaluated using photometric loss (PL) and CLIP score [31].

Procedural Geometry The geometry editing task challenges VLMs in spatial and geometric reasoning capabilities, requiring them to handle variations in attributes like shape, scale, and spatial distribution. Similar to material editing, the system is expected to propose procedural edits through BPY and Infinigen. 3D geometry editing results are evaluated using Chamfer distance (CD), PL, and CLIP.

Lighting Adjustments The lighting adjustment task involves manipulating the color, intensity, location, and orientation of the light sources to assess a VLM system's reasoning abilities regarding the lighting environment. Notably, we include some scenes where light sources are not directly visible from the camera but can be inferred from the shadow. VLMs must edit both continuous parameters, such as RGB values, and discrete parameters, such as the type of light source (e.g. point light v.s. area light). We use PL and CLIP to evaluate the edits of lighting task.

Object Placement Object placement task requires VLM systems to perceive objects in the start scene and reposition them to their corresponding location in the goal scene. This task evaluates a VLM system's perception and reasoning of spatial information. In the input code script, we prepare various Python functions for VLM systems to modify the location of an object. They can directly set its 3D coordinates,

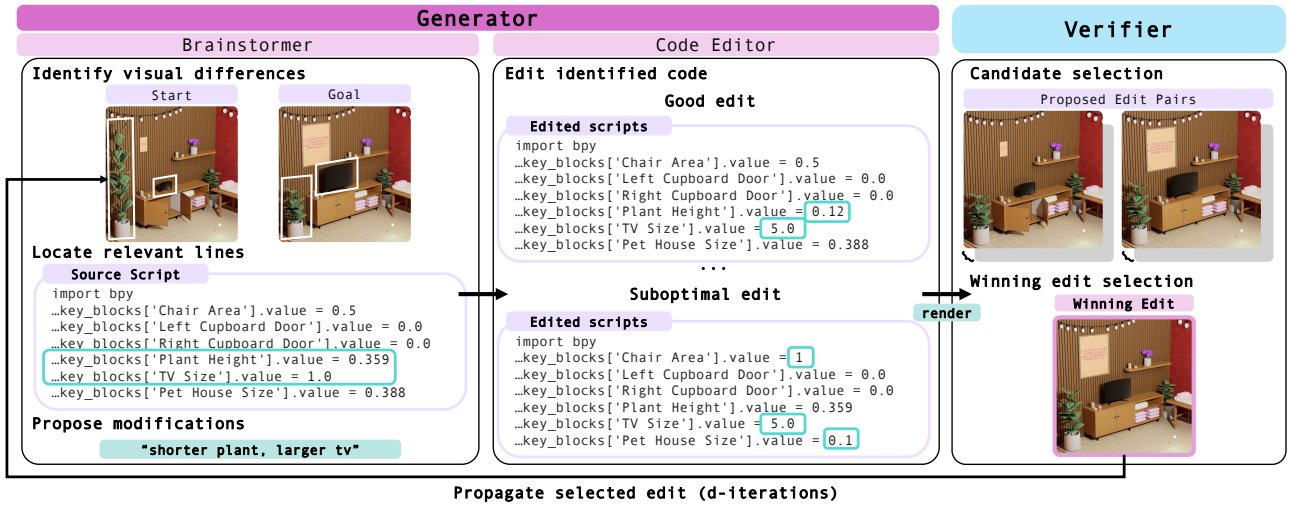


Figure 2. VLM system setup used by BlenderGym. It follows a generator-verifier structure, where the generator further contains a brainstormer and a code editor. The generator takes in start-goal image pairs along with Python script of the start scene, and then edits the Python script of the start scene to achieve the goal scene based on their visual differences. The verifier takes a pair of renders of proposed edits and selects a single best edit, which is propagated back to the generator for the next iteration.

shift its position via a vector offset, or place it relative to another object with some distance in a given direction. The distance metrics within the scene are calibrated by preserving the real-world distance scale and defining boundaries for the x, y, and z coordinates for the scene. For placement, we utilize CD, PL and CLIP to measure the quality of edits.

Blend Shapes In this task, the VLM system is responsible for shape blending – adjusting continuous variables that control some features of an object, such as facial expression, shape, or scale. Since these blend shapes are semantically annotated with language in the code scripts, the VLM system must match the name of the blend shapes to the features they refer to and tweak their value accordingly. We leverage CD, PL and CLIP to measure the resulting edits as blend shapes usually involve 3D structures.

Note that despite our choice of name for blend shapes and our usage of Infinigen[32] as wrapper functions for Blender-Python API in procedural material and geometry editing tasks, BlenderGym users can experiment with other representations of the start and goal Blender scenes.

For evaluation, we use negative CLIP similarity (N-CLIP), defined as $(1 - \text{CLIP Score})$ to clip the CLIP scores to positive values and reverse its trend, keeping it consistent with photometric loss and Chamfer distance such that a lower metric value indicates greater similarity to the goal.

3.1.2 Camera Viewpoint Selection

Occluded or incomplete render views of the 3D scene can introduce bias for VLM edits. Therefore, for all the 3D-related tasks, we manually select and fix at least three render views for each start-goal scene pair, ensuring that (1) at least one comprehensive view, in which most objects are exposed, is included and that (2) all objects are captured by at least one camera view. A more detailed definition and examples of comprehensive views can be found in Appendix S5. For VLM input, we use two camera views, with at least one being a comprehensive view. For evaluation, we use all the views. Note that since we have ensured that evaluation views cover every object in the scene, BlenderGym users can experiment with different choices of camera position or visual representation for VLM system input.

3.2. Human Baseline

We demonstrate the solvability and validity of BlenderGym and show the performance gap between VLMs and humans by engaging 10 average Blender users to complete approximately 20% of all task instances in BlenderGym. We restrict the edit time of human users to at most 8 minutes per instance, which matches the average time taken by a VLM system. We then evaluate human edits using the same metrics as for VLMs. The results of human Blender users are shown in Tab. 1 along with that of VLM systems.

4. Experiment

In this section, we provide a comprehensive analysis of VLM system performance on BlenderGym, present human baselines, and explore methods to enhance the editing capabilities of VLM systems.

4.1. VLM System Setup

We base our VLM setup on the BlenderAlchemy [20] multi-agent system, as it is the current SOTA pipeline for multiple graphics editing tasks and supports seamless integra-



Figure 3. Examples of VLM/human generated outputs on lighting and blend shape tasks. Even powerful closed-source VLMs fail to generate lighting settings with accurate colors. In the blend shape task above, only Claude 3.5 Sonnet correctly captures the overall appearance of the police car but still misses the difference of the front light. We report N-CLIP (10^{-3}) for all edits above as a calibration.

	Blend Shape			Placement			Geometry			Lighting		Material	
Model	PL ↓	N-CLIP ↓	CD ↓	PL ↓	N-CLIP ↓	CD ↓	PL ↓	N-CLIP ↓	CD ↓	PL ↓	N-CLIP ↓	PL ↓	N-CLIP ↓
GPT-4o	**9.140**	**20.47**	**0.904**	11.89	**30.38**	11.22	**6.747**	8.561	1.192	**2.410**	2.398	**3.653**	8.942
Claude-3.5-Sonnet	12.79	27.96	1.962	13.19	51.76	11.29	10.81	13.04	1.452	2.897	4.049	5.769	11.44
GPT-4-Turbo	15.21	26.15	1.927	12.21	37.57	12.80	8.160	10.92	**1.120**	2.723	3.912	5.424	**8.812**
Claude-3-Haiku	13.62	29.72	2.563	14.78	44.10	12.13	10.15	12.51	1.362	3.712	4.824	5.960	11.61
Gemini-1.5-flash	23.18	30.47	2.412	**10.94**	45.34	**8.324**	9.443	10.49	1.323	3.514	5.688	6.364	10.42
Qwen2-vl-7b	16.78	29.22	2.123	15.31	41.12	14.21	–	–	–	2.985	**2.225**	–	–
Qwen-Llama	14.32	28.23	2.012	14.65	34.93	12.41	13.97	14.13	1.673	3.173	3.998	–	–
Phi-3.5-vision	12.51	24.14	2.012	–	–	–	–	–	–	3.127	6.012	–	–
Phi-Llama	12.13	24.77	1.826	14.61	35.61	12.61	9.818	11.92	1.471	3.621	6.895	–	–
MiniCPM-V-2.6	13.86	29.92	1.997	11.99	31.69	12.62	7.127	**8.542**	1.229	3.829	6.124	–	–
MiniCPM-Llama	13.76	27.21	1.882	12.74	31.72	15.81	9.561	11.47	1.569	3.725	6.090	7.152	12.14
InternVL2-8b	12.69	29.09	1.920	14.71	35.92	17.22	–	–	–	3.920	6.825	–	–
Intern-Llama	11.80	23.83	1.861	16.15	37.23	18.22	13.70	14.44	1.578	3.825	6.152	–	–
Human	**0.934**	**9.12**	**0.399**	**0.423**	**13.34**	**1.532**	**1.269**	**2.434**	**0.334**	**1.239**	**1.632**	**0.629**	**3.043**

Table 1. Quantitative evaluation of VLM system edits. The magnitude of photometric loss(PL) and negative-clip(N-CLIP) is scaled to 10^{-3} for blend shape and placement tasks, and to 10^{-2} for the remaining tasks. We report unscaled Chamfer Distance(CD). Values marked with "-" indicate that the VLM system was unable to generate any executable code script in more than 75% of the instances for that task due to the complexity in the code scripts of these two tasks.

tion of various VLMs. Our implementation leverages a generator-verifier structure shown in Fig. 2, with two hyperparameters: depth d and breadth b.

Generator The generator has two key model components: the brainstormer and the code editor. The brainstormer identifies the most significant visual difference between the rendered images of the start and goal scenes, examines the Python script of the start scene to locate relevant lines, and proposes instructions to edit the start script. These instructions may take the form of natural language or Python code. The code editor then incorporates the instructions into an edited Python script. This brainstormer-code-editor structure is repeated independently for a breadth b times to derive a list of b parallel Python script proposals.

Verifier The verifier receives b proposals from the generator and splits them into $\frac{b}{2}$ pairs in sequential order. It then compares the render images of the candidates with the goal render and selects the one closest to the goal. The selected edits from all pairs are then aggregated into a list of $\frac{b}{2}$ winner candidates, on which we perform pair-wise selection iteratively until only one candidate remains as the winner.

One generator-verifier pair forms an edit iteration that incrementally makes the input scene more closely resemble the goal scene. For the sake of consistency, our setup uses the **same VLM** for both the generator and the verifier. The output of each iteration is propagated as the input

scene for the subsequent iteration. We repeat such iteration for a depth d times, with the final output from the last iteration representing the result of the entire pipeline. The overall structure resembles a $d \times b$ tree, where each iteration prunes suboptimal edits and progressively refines the best candidate. For this benchmark, we set $d = 3$ and $b = 4$. Prompts used for each agent are provided in Appendix S6. Note that despite our usage, BlenderGym is not limited to BlenderAlchemy, but supports easy plug-and-play of any VLM system. Other input information, such as prompt and camera views, can also be customized by the user.

4.2. Benchmark Results

We benchmark 13 closed- and open-source VLMs, including GPT-4o [28], Claude 3.5 Sonnet [3], Gemini 1.5 Flash [37], InternVL2-8B [10], and Qwen2-VL-7B-Instruct-AWQ [4]. For VLMs that do not accept text-only input, we prompt them with a random noise image to meet the format requirements. As discussed in Sec. 3.2, we also evaluate the performance of 10 human Blender users as a baseline to compare VLM performance with. We calibrate the N-CLIP metric in Fig. 3 and other metrics in Appendix S4 to illustrate these metric values visually.

Tab. 1 shows quantitative results on VLM and human performance in all tasks. Throughout the experiments, we observe a substantial performance gap between human Blender users and VLM systems (Tab. 1 & Fig. 3), indicating that graphics editing remains an unsolved challenge for VLMs. Furthermore, performance varies significantly between different VLM systems, indicating that different VLM systems might be better suited for different tasks.

4.2.1 Generation Analysis

VLM generators remain imperfect, as illustrated by the qualitative examples in Fig. 3. Among the failure cases, we observe a few factors that significantly contribute to suboptimal edits. We attach examples of each in Appendix S2.

Generator fails to capture subtle visual differences. The VLM sometimes struggles to detect visual differences between start and goal scenes, essential for fine-tuning graphical edits.

Generator fails to produce an executable Blender Python script. VLM systems perform particularly poorly in procedural material and geometry editing, as these tasks require wiring of the nodes of the shader editor and geometry editor in Blender Python API through code editing. All open-source VLM systems, with the exception of MiniCPM, fail to produce any executable code scripts for more than 75% of task instances in geometry editing task. To address this, we incorporate Llama3.1-8b[1], known for its strong performance on code editing benchmark, as the code editor model for the generator described in Sec. 4.1. Performance of Llama-enhanced VLMs is marked with a suffix Llama in Tab. 1. While Llama indeed enables Phi, Qwen, and Intern to complete more than 75% of geometry tasks, this combination does not necessarily lead to stronger performance for all task settings.

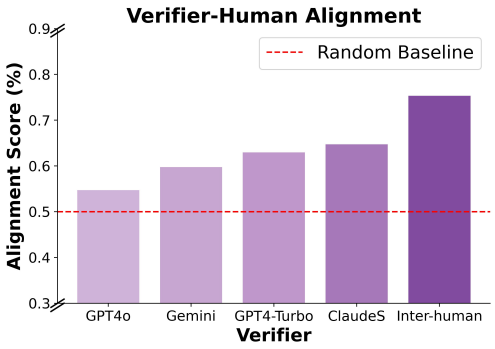


Figure 4. Human-VLM and inter-human verifier alignment rate. All models perform above the random baseline (0.5) yet differ notably, with even the highest-aligned model, Claude-3.5-Sonnet (0.66), falling short of inter-human alignment (0.79).

Generator provides irrelevant code changes to reflect the visual differences spotted. Even if the resulting proposal is executable in Blender, it may not reflect the visual differences indicated by the brainstormer. This problem is particularly common with complex Blender Python scripts such as those of procedural material or geometry editing tasks, which usually consist of more than 80 lines of code.

4.2.2 Verification Analysis

The verifier sometimes fails to distinguish between desirable and suboptimal edits. Certain VLMs, such as Qwen [4], consistently favor the second edit candidate in the pair, regardless of how we permute them. We provide example failure cases of VLM verifier in Appendix S3.

We measure human-VLM verifier alignment to ground our verifier analysis more quantitatively. For each of the five tasks, we select one instance and collect all the pair-wise selection decisions that the VLM verifier made throughout the inference. We collect 7,950 pair-wise judgments from 50 participants. For each pair-wise selection, VLM and human verifiers are asked to choose the edit render closest to the goal render. We compute the alignment rate between two verifiers by calculating the ratio of the number of aligned pair-wise selections over the total number of pairs. We compute the alignment rate between human and GPT-4o, GPT-4 Turbo, Claude 3.5 Sonnet, and Gemini 1.5 Flash, and also measure inter-human alignment. Results in Fig. 4 reveal that human verifiers achieve a significantly higher alignment rate compared to VLMs, highlighting considerable room for improvement in VLM verifier performance.

Algorithm 1: Scaled Verification

Function ScaledSelect(List of proposal candidates $\{S_1, S_2, \ldots, S_n\}$, Verifier V, Reselection parameter k):
Initialize *winner_list* as an empty list.
for $i = 1$ to k **do**
  $\{S'_1, S'_2, .., S'_n\} \leftarrow$ shuffle$(\{S_1, S_2, .., S_n\})$
  $winner \leftarrow$ SingleSelect$(\{S'_1, S'_2, .., S'_n\}, V)$
  Append $winner$ to *winner_list*.
end for
$final_winner \leftarrow$ SingleSelect(winner_list, V)
return $final_winner$

4.3. Verifier Scaling

To complement recent findings on inference scaling of VLM/LLM generation[6, 34], we explore improving the verifier that guides the generation by selecting desirable edits and pruning suboptimal ones. Brown et al. [6] demonstrates that, assuming the presence of an oracle verifier that always identifies the best answer, scaling open-source models for generation can outperform closed-source models in coding and math problems. However, when this impractical oracle verifier is replaced by practical verifiers like majority voting or reward models, the accuracy of open-source LLM outputs plateaus quickly below that of closed-source models. Another verifier choice explored is the task-specific process-based reward model (PRM), which has been shown to be effective for math problems by Lightman et al. [26] and Snell et al. [34], but it requires additional training on task-specific data. Therefore, finding cost-effective verifiers that do not require domain-specific data remains an open challenge. To fill this gap, we propose **inference time scaling for VLM verifier**, the scaling of VLM as an answer selector, as an alternative path. This quantitative experiment is enabled by BlenderGym and the scalable visual state evaluator structure within our pipeline in Fig. 2.

We find that (1) similar to generation, VLM verifiers used for guiding generation also benefit from inference scaling, and that (2) the performance of scaled open-source VLM verifiers can exceed that of closed-source ones.

4.3.1 Experiment Details

To scale up the verifier, we introduce a reselection parameter k into the verifier structure (Algorithm 1). We run the vanilla verification process k times to generate a list of k winners, each time shuffling the candidate list. Then, we select a single final winner from the winner list.

With the scaling mechanism defined, we select 16 instances of blend shape task and use Claude 3.5 Sonnet to generate 32 candidate edits (4x8 tree) for each blend shape instance. We fix these 32-candidate trees and run ScaledSelect (Algorithm 1) with InternVL2-8B to select the best edit from the trees. We vary the reselection parameter k in 1, 2, 4, 8, 16, 32, and 64. Fig. 5 presents the metric scores and qualitative examples of the finally selected edit for different levels of verification queries. We also report smaller-scale scaling results for GPT-4o and Claude 3.5 Sonnet. Three key findings emerge from this experiment. (1) As inference compute increases for verification, the values of all three metrics of the selected edits decrease, indicating improved verifier selection. (2) Scaled InternVL2-8B outperform unscaled or slightly scaled GPT-4o and Claude 3.5 Sonnet in all three metrics. (3) Closed-source VLM systems also benefit from scaling. We conclude that for graphics editing, inference scaling is effective not only for generation but also for VLM verification that guides the generation. These findings encourage further exploration of the scalability of verification.

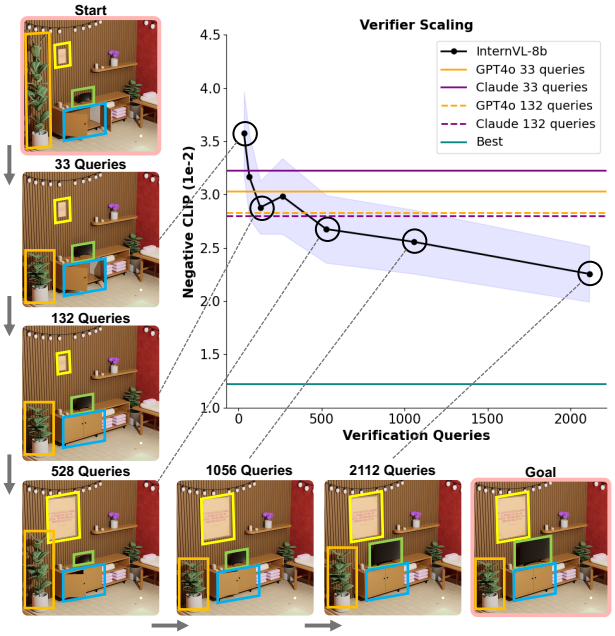


Figure 5. Verifier scaling results with InternVL2-8B, Claude3.5 Sonnet, and GPT4o. We show that increasing verifier queries brings the selected edit closer to the goal. Bounding boxes of all object instances of interest are shown.

4.4. Allocation of Compute

With the knowledge that VLM generation and verification both benefit from inference scaling, we further explore whether an optimal distribution of inference compute between generation and verification exists. We experiment with different ratios of verification queries over total queries (VeriRatio) and find that (1) the choice of VeriRatio significantly impacts the VLM performance and that (2) the optimal VeriRatio varies with the level of total compute — more total compute benefits from a higher share of verification.

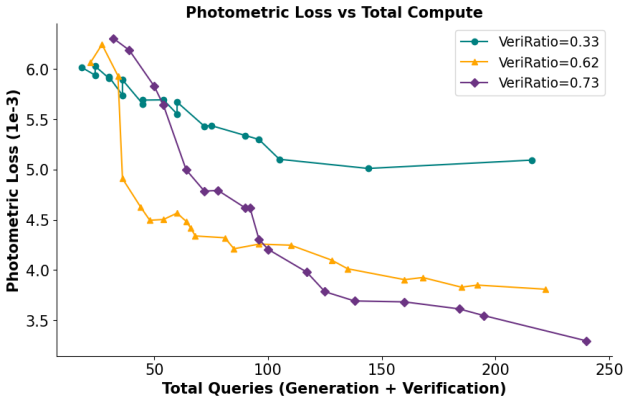


Figure 6. The impact of compute allocation on VLM system performance for 16 blend shape task instances. We show photometric loss at VeriRatios of 0.73, 0.62, and 0.33. We observe that with fewer compute, the generation process dominates the performance of the whole pipeline, while with a large compute budget, increasing verifier compute is more effective. The compute unit is query as every 100 generation/verification queries incur a similar cost of 0.45 USD.

4.4.1 Experiment Details

To measure the impact of compute allocation on VLM system performance, we fix three VeriRatios and run our setup across different total compute levels, measured by the total amount of queries. We use InternLlama with total queries varying from 12 to 245 and set reselection parameter k (defined in Sec. 4.3.1) to 1, 3, and 5, corresponding to VeriRatios of 0.33, 0.62, and 0.73, respectively.

Note that rather than selecting from a fixed candidate pool repeatedly as in Sec. 4.3, the reselection algorithm here is integrated into the pipeline (Fig. 2) to allow a scaled real-time pruning of suboptimal edits at each iteration. We experiment on 16 blend shape tasks (see Fig. 6) and 15 lighting tasks (see Appendix S1).

Fig. 6 demonstrates that the allocation of inference compute significantly impacts VLM system performance. We observe from Tab. 2 that as the total compute increases, a higher VeriRatio leads to a better performance. This suggests that when total inference compute is limited, generating a variety of candidates benefits more than stronger verification. However, when the inference budget is sufficiently high, refining a relatively smaller batch of candidate edits is more helpful than generating a larger set of potentially suboptimal candidates. This points to a broader insight about the importance of "deliberative" compute, or strategically allocated compute, in graphics editing: improvement in performance demands a shift from sheer generation capacity to a propose-verify-improve iterative workflow.

	Total Number of Queries			
	< 30	30-60	60-100	> 100
VeriRatio=0.33	**1**	2	3	3
VeriRatio=0.62	2	**1**	**1**	2
VeriRatio=0.73	3	3	2	**1**

Table 2. Performance rank of InternLlama with different VeriRatio across total compute levels. We observe that as the total compute, measured by total number of queries, increases, larger VeriRatio leads to better performance.

5. Conclusion and Discussion

In this work, we introduce BlenderGym, a comprehensive 3D graphics benchmark that tasks VLM systems with 3D scene reconstruction through code editing. Our experiments reveal that graphics editing remains unsolved for VLM systems. Furthermore, we demonstrate on BlenderGym that scaling verifiers can improve the VLM system's performance and that the more inference compute allowed, the higher proportion of it should be allocated to verification to optimize the system performance.

Limitations and future works While BlenderGym provides a robust foundation, further improvements are possible in the following areas. (1) The coverage and scale of BlenderGym can be expanded to more graphics editing tasks, such as object sculpting, camera-view adjustment, and animation creation with more data instances. (2) VLM-human verifier alignment and human Blender generator experiment can be enhanced by increasing the number of human annotators beyond the current scale. (3) More approaches can be explored to improve verifier performance, as they sometimes still fail to select the best edit despite being scaled. Exploring more advanced verifier pipelines or scaling up closed-source VLMs may offer further benefits.

Societal Impacts The automation of 3D graphics editing introduces complex socio-technological challenges, as graphics editing is a creative human-driven effort. The integration of VLMs into the system is costly and can reduce opportunities for human involvement due to the replacement of creative toolsets. Nonetheless, BlenderGym allows us to systematically study VLM systems and bridge the gap between graphics editing and the development of foundational models. Future work in automating the editing workflow must carefully consider these risks and prioritize mitigating them.

Acknowledgements We acknowledge the support of ARL grant W911NF-21-2-0104 and a Vannevar Bush Faculty Fellowship. Additionally, we would like to thank Yangjun Ruan, Haiwen (Haven) Feng, Jordan Juravsky, and all our reviewers for their feedback on the paper revisions.

References

[1] Meta AI. Llama 3.1-8b. https://ai.meta.com/llama, 2024. 6

[2] Haider Al-Tahan, Quentin Garrido, Randall Balestriero, Diane Bouchacourt, Caner Hazirbas, and Mark Ibrahim. Unibench: Visual reasoning requires rethinking vision-language beyond scaling. *Advances in Neural Information Processing Systems*, 37:82411–82437, 2025. 3

[3] Anthropic. Claude-3 model card, 2023. Accessed: 2025-01-04. 6

[4] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023. 6

[5] BlenderKit. BlenderKit: Online Asset Library for Blender. https://www.blenderkit.com/. 3

[6] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. 2, 7

[7] Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark. In *Forty-first International Conference on Machine Learning*, 2024. 2

[8] Dave Zhenyu Chen, Yawar Siddiqui, Hsin-Ying Lee, Sergey Tulyakov, and Matthias Nießner. Text2tex: Text-driven texture synthesis via diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18558–18568, 2023. 2

[9] Zhaorun Chen, Yichao Du, Zichen Wen, Yiyang Zhou, Chenhang Cui, Zhenzhen Weng, Haoqin Tu, Chaoqi Wang, Zhengwei Tong, Leria HUANG, et al. Mj-bench: Is your multimodal reward model really a good judge? In *ICML 2024 Workshop on Foundation Models in the Wild*. 2

[10] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198, 2024. 6

[11] 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 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2024. 2

[12] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[13] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, and Rongrong Ji. Mme: A comprehensive evaluation benchmark for multimodal large language models, 2024. 3

[14] Rao Fu, Zehao Wen, Zichen Liu, and Srinath Sridhar. Anyhome: Open-vocabulary generation of structured and textured 3d homes. In *European Conference on Computer Vision*, pages 52–70. Springer, 2024. 2

[15] Purvi Goel, Kuan-Chieh Wang, C Karen Liu, and Kayvon Fatahalian. Iterative motion editing with natural language. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–9, 2024. 2

[16] Paul Guerrero, Milos Hasan, Kalyan Sunkavalli, Radomír Mech, Tamy Boubekeur, and Niloy J. Mitra. Matformer: a generative model for procedural materials. *ACM Trans. Graph.*, 41(4):46:1–46:12, 2022. 2

[17] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. 3

[18] 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 *Forty-first International Conference on Machine Learning*, 2024. 2

[19] Ian Huang, Vrishab Krishna, Omoruyi Atekha, and Leonidas Guibas. Aladdin: Zero-shot hallucination of stylized 3d assets from abstract scene descriptions. *arXiv preprint arXiv:2306.06212*, 2023. 2

[20] Ian Huang, Guandao Yang, and Leonidas Guibas. Blenderalchemy: Editing 3d graphics with vision-language models. In *European Conference on Computer Vision*, pages 297–314. Springer, 2024. 2, 3, 4

[21] Yipo Huang, Quan Yuan, Xiangfei Sheng, Zhichao Yang, Haoning Wu, Pengfei Chen, Yuzhe Yang, Leida Li, and Weisi Lin. Aesbench: An expert benchmark for multimodal large language models on image aesthetics perception, 2024. 3

[22] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 2

[23] Seongyun Lee, Seungone Kim, Sue Park, Geewook Kim, and Minjoon Seo. Prometheus-vision: Vision-language model as a judge for fine-grained evaluation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11286–11315, 2024. 2

[24] Zhen Li, Xiaohan Xu, Tao Shen, Can Xu, Jia-Chen Gu, Yuxuan Lai, Chongyang Tao, and Shuai Ma. Leveraging large language models for nlg evaluation: Advances and challenges. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16028–16045, 2024. 2

[25] Peiyuan Liao, Xiuyu Li, Xihui Liu, and Kurt Keutzer. The artbench dataset: Benchmarking generative models with artworks, 2022. 3

[26] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. 7

[27] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer, 2024. 3

[28] OpenAI. Gpt-4 system card. *OpenAI*, 2023. 2, 6

[29] OpenAI. Gpt-4v(ision) system card. *OpenAI*, 2023. 2

[30] Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z. Xiao, Katherine M. Collins, Joshua B. Tenenbaum, Adrian Weller, Michael J. Black, and Bernhard Schölkopf. Can large language models understand symbolic graphics programs? In *The Thirteenth International Conference on Learning Representations*, 2025. 3

[31] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 3

[32] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12630–12641, 2023. 3, 4

[33] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes. In *ACM SIGGRAPH 2023 conference proceedings*, pages 1–11, 2023. 2

[34] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. 2, 7

[35] 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. 2

[36] Hou In Ivan Tam, Hou In Derek Pun, Austin T. Wang, Angel X. Chang, and Manolis Savva. Scenemotifcoder: Example-driven visual program learning for generating 3d object arrangements. *CoRR*, abs/2408.02211, 2024. 2

[37] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 2, 6

[38] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. *arXiv preprint arXiv:2407.19594*, 2024. 2

[39] 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. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. 2

[40] Haibo Yang, Yang Chen, Yingwei Pan, Ting Yao, Zhineng Chen, and Tao Mei. 3dstyle-diffusion: Pursuing fine-grained text-driven 3d stylization with 2d diffusion models. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 6860–6868, 2023. 2

[41] Yixuan Yang, Junru Lu, Zixiang Zhao, Zhen Luo, James J. Q. Yu, Victor Sanchez, and Feng Zheng. Llplace: The 3d indoor scene layout generation and editing via large language model, 2024. 2

[42] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, pages 20–25. IEEE/CVF, 2024. 2

[43] Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J Mitra, and Changjian Li. Cadtalk: An algorithm and benchmark for semantic commenting of cad programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3753–3762, 2024. 3

[44] Kaizhi Zheng, Xiaotong Chen, Odest Chadwicke Jenkins, and Xin Wang. Vlmbench: A compositional benchmark for vision-and-language manipulation. *Advances in Neural Information Processing Systems*, 35:665–678, 2022. 3

[45] Károly Zsolnai-Fehér, Peter Wonka, and Michael Wimmer. Gaussian material synthesis. *ACM Trans. Graph.*, 37(4):76:1–76:14, 2018. 2