AGENTIC3D: ITERATIVE 3D MODELING USING LLM AGENTS WITH VISION

Moritz Rietschel (3039632507), Reet Mishra (3034728048),
Sammie Shenon (3036698214), Frederik Stihler (3039672937)
Application Track
CS 194/294 Large Language Model Agents
University of California, Berkeley
Instructors: Dawn Song, Xinyun Chen
GSIs: Alex Pan, Sehoon Kim; Readers: Tara Pande, Ashwin Dara

ABSTRACT

This project investigates the application of Large Language Models (LLMs) as agents to bridge textual and visual modalities, focusing on the generation and refinement of 3D models from textual descriptions. Using OpenSCAD, a code-based 3D modeling platform, the system translates text into code, which is then rendered into visual representations for evaluation. A modality-switching feedback loop integrates Vision-Language Models (VLMs) to analyze these renderings, enabling iterative refinement of the generated models. The workflow is orchestrated by a Commander Agent, coordinating multiple specialized agents to ensure seamless interaction between components. Performance evaluation across three object complexity levels (low, medium, high) demonstrates that the system excels at generating and refining low-complexity models, with diminishing performance as object complexity increases. Metrics such as the mean maximum quality score and the mean improvement score highlight the system's effectiveness in refining simpler models. These findings underscore the potential of LLMs as versatile agents for bridging modalities and generating 3D content.

1 INTRODUCTION

This project explores the potential of Large Language Models (LLMs) as agents to bridge modalities, specifically focusing on generating 3D models from textual descriptions. By translating LLM-generated code into 3D models, which are then rendered as images for evaluation, we demonstrate how LLMs can extend their utility beyond text-based outputs. Given that OpenSCAD, the platform used for 3D modeling in this project, is code-based, its outputs are inherently textual and not easily interpretable in their raw form. Visual inspection of 3D shapes, on the other hand, is more intuitive and effective. To address this, we implement a modality switch in the feedback system: visual renderings of the generated code are analyzed using Vision-Language Models (VLMs), enabling the feedback loop to cycle between visual and textual modalities. This iterative process allows for continual refinement of the generated 3D models.

2 BACKGROUND (RELATED WORK)

2.1 TEXT TO MESH GENERATION

The generation of 3D meshes from textual queries using diffusion models has been widely explored Siddiqui et al. (2024). While some methods involve training directly on mesh data, the availability of such datasets is limited, and their annotations are often sparse. Consequently, research has shifted toward adapting pretrained image generation models to 3D data, leveraging larger image datasets. A notable early example is Poole et al. (2022), which generates 2D images from various viewpoints and subsequently reconstructs the 3D mesh in a second step. Commercial implementations, such as Luma (2024), demonstrate the viability of such text-to-mesh approaches.

Although mesh diffusion remains the most popular method for 3D asset generation, it has inherent limitations. The diffusion process learns to approximate mesh representations from data, prioritizing visual fidelity over precise geometric dimensions. This makes it suitable for applications requiring visually recognizable 3D objects, such as 3D animation, which predominantly uses mesh-based polygonal models. However, workflows demanding high-precision 3D assets—common in computer-aided design (CAD) for fabrication purposes—rely on mathematically accurate representations, such as NURBS and vector-based modeling. These are essential for domains like 3D printing, CNC machining, construction, and simulation.

Recent advancements by Hao et al. (2024) address these challenges by modeling meshes as sequences of tokens, similar to to next-token prediction in large language models (LLMs). Unlike image-based approaches, NVIDIA's method operates directly in coordinate space, learning sequential, tokenized representations rather than grid-based pixel representations. Further, while not explicitly a mesh-generation technique, Xu et al. (2024) introduced a transformer-based diffusion model for boundary representation (B-rep) generation, extending the mesh-based paradigm to precisely dimensioned CAD models.

2.2 Code Generation

The use of LLMs for code generation is a well-established area in research and industry. Commercial applications include OpenAI (2022), and specialized tools like GitHub (2021), Cursor (2024), and Windsurf (2024). These tools enable the automatic generation of programming code, simplifying tasks for developers. Recently, the emergence of agentic software developers like CognitionLabs (2024) has further expanded the accessibility of automated software development.

2.3 Code Generation for 3D Assets

Using LLMs to generate executable code for 3D model creation is a more recent development, particularly in text-to-CAD generation. For example, Li et al. (2024) employs LLMs to generate executable CADQuery code to produce simple CAD models. Similarly, Yuan et al. (2024) uses language models to generate Blender Python (bpy) scripts, allowing the creation of Blender-compatible models. This approach also integrates image feedback, leveraging VLMs to iteratively improve generated results. A similar approach is used by Yamada et al. (2024)—though their research goal is the image generation of unconventional objects—as they apply a text-to-CAD code logic and then render the result.

Commercial solutions have also emerged in this space. For instance, Zoo.dev (2024) offers a text-to-CAD API that generates code in a proprietary CAD programming language, producing precisely dimensioned models tailored for specific mechanical components like gears and pipe fittings. Beyond programming-centric methods, research efforts such as Ganin et al. (2021) and Rietschel et al. (2024) propose alternative approaches. These methods leverage language models to generate token-based representations of CAD geometry, which can subsequently be parsed into corresponding CAD elements and commands.

2.4 LLM Agents and Frameworks

LLM agents are systems that perceive and interact with environments through outputs generated by LLMs. This research domain is evolving rapidly. For example, Park et al. (2023) can simulate social interactions in a virtual village using LLM agents, while the Benchmark Yao et al. (2022) evaluates agent performance on internet-based workflows.

The growing popularity of LLM agents has led to the development of frameworks that facilitate their implementation. One such tool is Microsoft (2024), a Python framework designed to manage conversations between multiple instantiations of LLMs.

3 Methodology

Our system (see figure) is designed to iteratively generate 3D models based on textual descriptions. It comprises several specialized agents working together to bridge textual and visual modalities. The

workflow is managed by a central Commander Agent, ensuring smooth execution and interaction among all components.

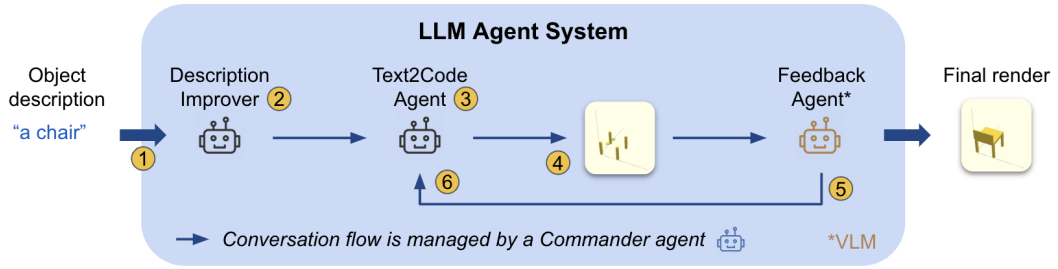

Figure 1: Schematic overview of the agentic system architecture. This diagram illustrates the components and interactions involved in the iterative 3D modeling process, including feedback loops and agent workflows.

3.1 Workflow Design

1. **User Input:** The process begins with the user providing a textual description of the desired object (e.g., "a chair").
2. **Description Improver Agent:** The initial description is passed to a Description Improver Agent, which refines it into a more detailed, actionable prompt. The goal is to reduce ambiguity and provide clear guidelines for the Coding Agent (e.g., "a four-legged chair with a straight backrest").
3. **Coding Agent (Initial):** Based on the improved description, the Coding Agent generates OpenSCAD code that serves as the blueprint for the 3D model. This initial code forms the foundation of the iterative refinement process.
4. **Rendering:** The generated OpenSCAD code is rendered into a 3D model, which is visualized as a 2D image. This visual representation allows for easier evaluation of the model's shape and structure.
5. **Feedback Agent (VLM):** The rendered image, along with the refined textual description, are sent to a Feedback Agent powered by a Vision-Language Model. The agent compares the visual output to the textual description and provides structured feedback, identifying discrepancies and offering improvement suggestions (e.g., "The legs are not evenly spaced. Adjust the positions of the legs to be equidistant.").
6. **Code Refinement:** The Coding Agent incorporates the structured feedback into the OpenSCAD script, modifying the code to better align the model with the description.

Steps 4 through 6 are repeated for a fixed number of iterations. This iterative approach should ensure progressive alignment between the textual description and the generated model. The final output is the result after the last iteration.

Throughout the process, the Commander Agent orchestrates the workflow, ensuring seamless communication between agents, carrying over information from one step to the next, and utilizing tools to render the OpenSCAD code into images for evaluation.

3.2 Implementation Details

The implementation of our system leverages the Microsoft AutoGen Framework, a robust toolkit for creating and managing multi-agent systems. AutoGen provides a structured environment that enables agents to be defined and interact as modular components. Within this framework, the agents in our system are initialized as classes of `ConversableAgent`, allowing seamless communication and collaboration.

To tailor the agents for specific tasks, we use subclasses of `ConversableAgent`:

- **UserProxyAgent:** Subclassed to define agents such as the Commander Agent and Coding Agent. The Commander Agent orchestrates the overall workflow, while the Coding Agent handles the generation and refinement of OpenSCAD code.
- **MultimodalConversableAgent:** Subclassed to implement the Feedback Agent, enabling it to process and analyze both textual and visual data.

In addition to facilitating agent interactions, we utilize AutoGen's **Tool Use Functionality**, which allows agents to perform actions by invoking predefined functions. This approach eliminates the need for agents to write arbitrary code and promotes modularity and consistency. For example, the Commander Agent can call a rendering tool to convert the OpenSCAD code generated by the Coding Agent into an image for evaluation. This tool-based approach ensures consistency, modularity, and reduced error rates during the execution process. Furthermore, by enabling multi-agent conversations through AutoGen's framework, our workflow seamlessly transitions from one iteration to the next, each of whose feedback builds upon each other.

For users who want to use agentic3D, we have developed a **Python package** that can be implemented in Jupyter notebooks for ease-of-use and interactivity linked at our Github above. This package contains an AgentBuilder class to handle the creation of all our agents with the AutoGen framework, a Prompter and Strategy class to create different styled prompts to experiment within our framework, and a Workflow class which orchestrates the flow of our agents to iteratively develop the user's 3D model description. Users can follow the instructions on Github for more details about how to use this package for their own 3D model development. For any other specific implementation details regarding our metric evaluations, please refer to the Github linked here instead.

3.3 Evaluation Setup

Our evaluation framework systematically assesses the performance of the system across three object complexity levels: low, medium, and high. The evaluation process measures the iterative refinement capability of the system in generating 3D models that match their initial textual descriptions.

We evaluate the system on five distinct objects per complexity level, resulting in an evaluation dataset of 15 objects in total.

Table 1: Evaluation dataset. Objects evaluated across different complexity levels.

Complexity Level	**Object 1**	**Object 2**	**Object 3**	**Object 4**	**Object 5**
Low	chair	table	pyramid	bowl	bottle
Medium	vase	candle	mug	desk lamp	wine glass
High	car	cathedral	eyeglasses	wristwatch	spiral staircase

For each object, our system generates 10 renders iteratively, starting with an initial render (iteration 1) and refining the output through 10 feedback loops. This process is fixed to 10 iterations to ensure consistent evaluation across objects. Additionally, the entire process is repeated three times (replicates), yielding 30 renders per object. Replicates are fixed at three to account for variability and ensure robustness in the evaluation, as they allow us to observe how the system performs across multiple runs with the same setup.

To assess performance, each render is assigned a score on a scale from 1 to 10, reflecting its alignment with the initial textual description of the object. Renders with a score of 5 or above are considered successful matches with the description.

The scores for each render are generated using a separate VLM, which assesses how well the render aligns with the intended scene description. The scoring system ranges from 1 to 10, where lower scores indicate poor alignment and higher scores represent renders that increasingly match the description, with a perfect score signifying no need for further refinement.

This scoring method, while systematic, has limitations. It depends on the VLM's interpretive capabilities, which may introduce biases or fail to fully grasp complex geometric details. Additionally, the scoring scale, though straightforward, might oversimplify subtle differences in quality. Discrepancies between the model's evaluations and human judgments can also affect the overall reliability and applicability of the scoring system.

3.4 EVALUATION METRICS

To evaluate the performance of the system, we define key metrics to quantify its iterative refinement capabilities, efficiency, and overall alignment with the textual descriptions. These metrics are calculated at three levels: per object, per complexity level, and for the overall system.

The following key evaluation metrics were selected to provide a comprehensive understanding of the system's performance:

Mean Improvement Score (MIS): This metric evaluates the system's ability to improve iteratively by measuring the difference between the initial quality score (iteration 1) and the highest quality score achieved across the 10 iterations. It is mathematically defined for object i as:

$$\text{MIS}_i = \frac{1}{R} \sum_{r=1}^{R} \left(\max_{j \in [1,10]} Q_{i,j,r} - Q_{i,1,r} \right),$$

where $Q_{i,j,r}$ represents the quality score of object i at iteration j in replicate r, and R is the number of replicates (fixed at $R = 3$).

Mean Quality Score (MQS): This metric captures the average quality score across all iterations and replicates for an object. It is defined as:

$$\text{MQS}_i = \frac{1}{R \cdot J} \sum_{r=1}^{R} \sum_{j=1}^{J} Q_{i,j,r},$$

where J is the total number of iterations (fixed at $J = 10$).

Mean First Match (MFM): This metric measures the average iteration number at which a quality score above 4 is first achieved, indicating the efficiency of the system in achieving an acceptable match. It is defined as:

$$\text{MFM}_i = \frac{1}{R} \sum_{r=1}^{R} \min\{j \mid Q_{i,j,r} > 4,\ j \in [1, 10]\}.$$

These metrics were chosen to assess the system's iterative refinement capabilities (MIS), overall alignment (MQS), and efficiency (MFM).

To further analyze performance, we compute the following additional metrics for each object:

- **Mean Initial Quality Score:** The mean quality score for iteration 1 across the 3 replicates.
- **Mean Final Quality Score:** The mean quality score for iteration 10 across the 3 replicates.
- **Mean Maximum Quality Score:** The mean of the highest score achieved across the 10 iterations for the 3 replicates.

The key metrics—Mean Improvement Score, Mean Quality Score, and Mean First Match—are also computed at the complexity level (low, medium, high) and for the overall system:

- **Per Complexity Level:** Metrics are aggregated across the five objects within each complexity level. For example:

$$\text{MIS}_{\text{complexity}} = \frac{1}{5} \sum_{i=1}^{5} \text{MIS}_i.$$

- **Overall System Evaluation:** Metrics are aggregated across all objects. For example:

$$\text{MIS}_{\text{overall}} = \frac{1}{15} \sum_{i=1}^{15} \text{MIS}_i.$$

This structured approach enables us to quantify the effectiveness of the feedback loop, assess the system's adaptability to increasing complexity, and identify areas for further refinement. By capturing key aspects of performance, it evaluates the system's initial capability through the Mean Initial

Quality Score, its ability to iteratively refine models with the Mean Improvement Score (MIS), and its efficiency in achieving a match using the Mean First Match (MFM) metric. Overall alignment and reliability are further assessed through the Mean Quality Score (MQS). This comprehensive analysis, applied across individual objects, complexity levels, and the entire system, provides a detailed evaluation of the system's precision and capacity for iterative improvement in 3D model generation.

4 Experiments/Evaluation

4.1 Results per Object

We evaluate the model on our evaluation dataset using the base architecture depicted in Figure 1, featuring OpenAI's GPT-4o Mini model. Figure 2 shows example renders for the description *"a cylindrical bottle"*. These renders demonstrate the system's iterative refinement process, where the initial output is progressively improved based on visual feedback.

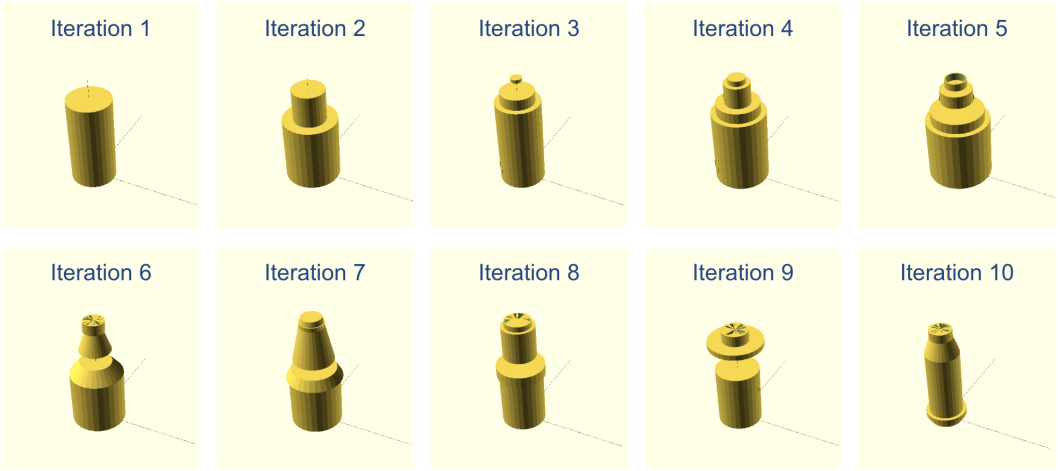

Figure 2: Ten iterative renders for the description *"a cylindrical bottle"*. The outputs demonstrate the model's refinement process over 10 iterations.

Figure 3 presents the Mean Maximum Quality Scores for the 15 objects in the dataset. The Mean Maximum Quality Score reflects the model's overall ability to generate renders that accurately depict the desired objects. The Mean Maximum Quality Score across all objects was 5.133. The highest Mean Maximum Quality Score was achieved for the object "a chair with four legs", receiving a perfect rating of 10/10. However, the model struggled to represent certain objects, achieving scores below 5 for seven of the fifteen objects. See Table 5 in Appendix for full results.

The model achieved an average mean initial score of 3.533 across all objects, highlighting its baseline performance. The average Mean Improvement Score (MIS) of 3.089 demonstrates the system's ability to improve upon one-shot attempts. However, the Mean Final Quality Score of 3.467 indicates that the model often fails to achieve the maximum score in its final iteration, even after completing the feedback loop. This discrepancy suggests possible areas for improvement. One hypothesis is that as renders improve, feedback becomes more detailed and challenging to implement, potentially leading to lower-quality results in subsequent iterations. Another explanation could be that the model occasionally achieves higher scores through random sampling rather than systematic improvement. To investigate these possibilities, future research should compare the system's performance against a baseline of 10 one-shot samples without feedback.

4.2 Results by Complexity Level

We evaluate the system's performance across three complexity levels: low, medium, and high. The results are summarized in Table 4.2, showing the key metrics for each complexity level.

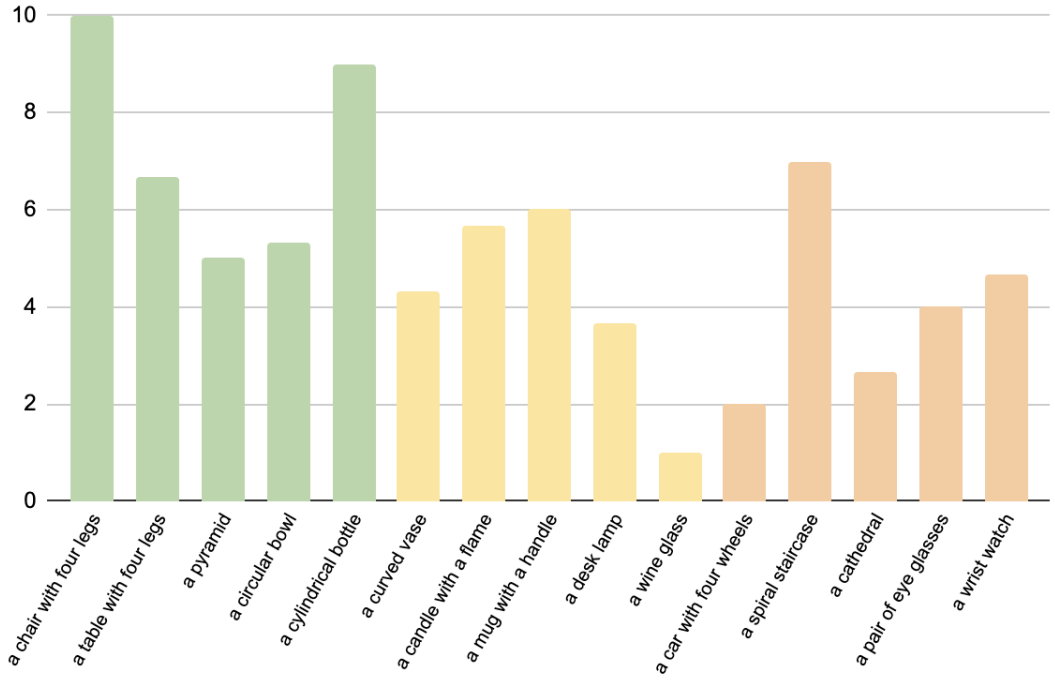

Figure 3: Mean Maximum Quality Scores for the 15 objects in the evaluation dataset. Green bars indicate low-complexity objects, yellow bars represent medium-complexity objects, and orange bars denote high-complexity objects.

Table 2: Results by Complexity Level (Base Configuration - GPT-4o Mini, with Description Improver Agent)

Level	**Mean Initial Score**	**Mean Final Score**	**Mean Max Score**	**MIS**	**MFM**	**MQS**
Low	4.800	4.533	7.200	4.600	2.667	4.973
Medium	3.133	2.867	4.133	1.933	6.000	3.060
High	2.667	3.000	4.067	2.733	6.267	2.667

The results indicate that the system performs significantly better on low-complexity objects compared to medium- and high-complexity ones. This is evident across multiple metrics, including the Mean Initial Quality Score, Mean Final Quality Score, Mean Maximum Quality Score, and Mean Quality Score (MQS). For instance, the Mean Maximum Quality Score for low-complexity objects is 7.200, whereas it drops to 4.133 and 4.067 for medium- and high-complexity objects, respectively. Additionally, the Mean Improvement Score (MIS) is highest for low-complexity objects, reflecting the system's stronger ability to refine and improve simpler models over iterative feedback loops.

4.3 Testing Different Models

Table 3: Results by Model (with Description Improver Agent)

Model	**Mean Initial Score**	**Mean Final Score**	**Mean Max Score**	**MIS**	**MFM**	**MQS**
GPT-4o Mini	3.533	3.467	5.133	3.089	4.978	3.567
GPT-4o	4.689	5.533	7.000	3.533	3.600	5.398

We evaluated our system using OpenAI's GPT-4o and GPT-4o Mini models to assess whether a larger model with greater computational power improves system performance. The results indicate that GPT-4o outperformed GPT-4o Mini across all metrics. Notably, the Mean Initial Quality Score increased by 1.156 (25%), the Mean Quality Score (MQS) improved by 1.831 (34%), and the Mean First Match (MFM) decreased by 1.378 iterations (-28%). For medium- and high-complexity objects, GPT-4o consistently improved performance across all metrics. On low-complexity objects, GPT-4o achieved improvements on all metrics except for the Mean Improvement Score. However, this exception is likely due to random variation and the nondeterministic behavior of LLMs, as the difference between the two models on this metric remained minimal. Overall, our findings suggest that increasing model size and computational capacity leads to improved system performance.

4.4 Testing the Effectiveness of the Description Improver Agent

We conducted an experiment to test the effectiveness of the Description Improver Agent by comparing system performance with and without it. The results demonstrate the Improver Agent's ability to enhance performance across all evaluated metrics. By refining the initial textual descriptions into more specific and precise prompts, the Improver Agent simplifies the task for the Coding Agent, resulting in a significant improvement in the Mean Initial Quality Score (+0.8 points on average). This improvement also translates into a 0.69-point higher Mean Maximum Quality Score, indicating better overall render quality. Additionally, the Mean First Match (MFM) is reduced, suggesting faster convergence toward satisfactory outputs. The Mean Improvement Score (MIS) is also higher when using the Description Improver Agent, reflecting the system's stronger capability to iteratively refine the 3D model. A likely explanation is that the Feedback Agent benefits from the enhanced descriptions, allowing it to provide more targeted and actionable feedback for further refinement.

Table 4: Comparison of Results With and Without Description Improver Agent (GPT-4o Mini)

Configuration (Scores are averaged across all objects)	**Mean Initial Score**	**Mean Final Score**	**Mean Max Score**	**MIS**	**MFM**	**MQS**
Without Description Improver Agent	2.733	2.999	4.444	2.711	6.178	2.942
Δ with Description Improver Agent	+0.80	+0.47	+0.69	+0.38	-1.20	+0.62

5 Conclusion

Our model demonstrates promising 3D modeling capabilities by leveraging LLM Agents with Vision. The system performs best on lower-complexity objects, where it achieves high-quality outputs and significant improvements through iterative refinement. While performance decreases for higher-complexity models, the results highlight the model's foundational strengths and potential for further enhancement.

There are several promising avenues for improving and extending the system. First, instead of using the render from the final iteration as the output, the system could be optimized to select the maximum score render as the final output, ensuring the best result is utilized. Second, we observed that when the initial render fails, the system sometimes struggles to recover despite feedback. System recovery ability improved with the use of OpenAI's GPT-4o model over 4o Mini in testing, but remains an area for future work. Potential solutions include providing render failure information back to the Coding Agent or incorporating a mechanism to select from a range of initial renders, thereby improving the starting point for the feedback loop.

Additionally, evaluating the system with alternative models, such as Anthropic's Claude 3.5 Sonnet, which are known for stronger coding capabilities, may further enhance performance. Lastly, our experiments highlight the value of improved prompting through the Description Improver Agent. Future work could explore tuning system messages and refining user descriptions to further increase performance and consistency.

REFERENCES

CognitionLabs. Devin, 2024. URL https://devin.ai/.

Cursor. Cursor - The AI Code Editor, 2024. URL https://www.cursor.com/.

Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-Aided Design as Language. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5885–5897. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/2e92962c0b6996add9517e4242ea9bdc-Abstract.html.

GitHub. GitHub Copilot · Your AI pair programmer · GitHub, 2021. URL https://github.com/features/copilot.

Zekun Hao, David W. Romero, Tsung-Yi Lin, and Ming-Yu Liu. Meshtron: High-Fidelity, Artist-Like 3D Mesh Generation at Scale, December 2024. URL http://arxiv.org/abs/2412.09548. arXiv:2412.09548 [cs].

Xingang Li, Yuewan Sun, and Zhenghui Sha. LLM4CAD: Multi-Modal Large Language Models for 3D Computer-Aided Design Generation. In *Volume 6: 36th International Conference on Design Theory and Methodology (DTM)*, pp. V006T06A015, Washington, DC, USA, August 2024. American Society of Mechanical Engineers. ISBN 978-0-7918-8840-7. doi: 10.1115/DETC2024-143740. URL https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings/IDETC-CIE2024/88407/V006T06A015/1208976.

Luma. Luma AI - Genie, 2024. URL https://lumalabs.ai/genie.

Microsoft. AutoGen | AutoGen, 2024. URL https://microsoft.github.io/autogen/0.2/.

OpenAI. Introducing ChatGPT, 2022. URL https://openai.com/index/chatgpt/.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pp. 1–22, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606763. URL https://dl.acm.org/doi/10.1145/3586183.3606763.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D Diffusion, September 2022. URL http://arxiv.org/abs/2209.14988. arXiv:2209.14988 [cs, stat].

Moritz Rietschel, Fang Guo, and Kyle Steinfeld. Mediating Modes of Thought: LLM's for design scripting, December 2024. URL http://arxiv.org/abs/2411.14485. arXiv:2411.14485 [cs].

Yawar Siddiqui, Tom Monnier, Filippos Kokkinos, Mahendra Kariya, Yanir Kleiman, Emilien Garreau, Oran Gafni, Natalia Neverova, Andrea Vedaldi, Roman Shapovalov, and David Novotny. Meta 3D AssetGen: Text-to-Mesh Generation with High-Quality Geometry, Texture, and PBR Materials, July 2024. URL http://arxiv.org/abs/2407.02445. arXiv:2407.02445 [cs].

Windsurf. Windsurf Editor by Codeium, 2024. URL https://codeium.com/windsurf.

Xiang Xu, Joseph Lambourne, Pradeep Jayaraman, Zhengqing Wang, Karl Willis, and Yasutaka Furukawa. BrepGen: A B-rep Generative Diffusion Model with Structured Latent Geometry. *ACM Transactions on Graphics*, 43(4):1–14, July 2024. ISSN 0730-0301, 1557-7368. doi: 10.1145/3658129. URL https://dl.acm.org/doi/10.1145/3658129.

Yutaro Yamada, Khyathi Chandu, Yuchen Lin, Jack Hessel, Ilker Yildirim, and Yejin Choi. L3GO: Language Agents with Chain-of-3D-Thoughts for Generating Unconventional Objects, February 2024. URL http://arxiv.org/abs/2402.09052. arXiv:2402.09052 [cs].

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, December 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Zeqing Yuan, Haoxuan Lan, Qiang Zou, and Junbo Zhao. 3D-PreMise: Can Large Language Models Generate 3D Shapes with Sharp Features and Parametric Control?, January 2024. URL http://arxiv.org/abs/2401.06437. arXiv:2401.06437 [cs].

Zoo.dev. AI CAD Model Generator | Create CAD Files With Text, 2024. URL https://zoo.dev//text-to-cad.

A Appendix

Table 5: Results by Object (Base Configuration - GPT-4o Mini, with Description Improver Agent)

Object	**Mean Initial Score**	**Mean Final Score**	**Mean Max Score**	**MIS**	**MFM**	**MQS**
a chair with four legs	5.000	7.667	10.000	7.667	2.000	7.100
a table with four legs	4.000	3.333	6.667	4.333	1.333	4.067
a pyramid	4.667	4.000	5.000	2.000	7.000	3.867
a circular bowl	3.333	2.000	5.333	4.333	1.000	2.833
a cylindrical bottle	7.000	5.667	9.000	4.667	2.000	7.000
a curved vase	3.667	3.333	4.333	1.667	7.000	3.367
a candle with a flame	5.000	2.333	5.667	3.333	1.333	3.967
a mug with a handle	4.000	5.000	6.000	2.667	1.667	4.467
a desk lamp	2.000	2.667	3.667	2.000	10.000	2.500
a wine glass	1.000	1.000	1.000	0.000	10.000	1.000
a car with four wheels	1.000	2.000	2.000	1.000	10.000	1.533
a spiral staircase	3.333	5.667	7.000	4.667	2.000	5.267
a cathedral	1.333	2.000	2.667	1.667	10.000	1.900
a pair of eye glasses	3.000	1.667	4.000	3.000	4.667	1.433
a wrist watch	4.667	3.667	4.667	3.333	4.667	3.200
Average	3.533	3.467	5.133	3.089	4.978	3.567

Table 6: Results by Object (GPT-4o, with Description Improver Agent)

Object	Mean Initial Score	Mean Final Score	Mean Max Score	MIS	MFM	MQS
a chair with four legs	7.667	8.333	9.000	3.667	2.000	7.633
a table with four legs	9.667	9.333	10.000	1.000	2.000	9.700
a pyramid	8.333	6.667	9.000	4.333	2.000	7.600
a circular bowl	2.333	4.333	7.333	6.000	5.000	4.333
a cylindrical bottle	7.000	7.333	8.667	3.667	2.000	7.067
a curved vase	2.667	4.667	6.000	3.667	4.333	4.467
a candle with a flame	4.667	4.667	6.000	3.667	2.000	4.467
a mug with a handle	3.667	7.333	8.667	5.333	2.000	6.267
a desk lamp	4.333	5.333	7.000	4.000	2.000	5.133
a wine glass	3.333	4.333	5.333	2.667	4.333	4.167
a car with four wheels	4.333	5.333	8.000	5.333	2.000	5.267
a spiral staircase	7.000	7.333	8.000	2.000	2.000	7.167
a cathedral	1.333	2.000	2.667	1.667	10.000	1.833
a pair of eye glasses	2.000	3.000	3.333	1.667	7.667	2.233
a wrist watch	2.000	3.000	6.000	4.333	4.667	3.633
Average	4.689	5.533	7.000	3.533	3.600	5.398

Table 7: Results by Object (GPT-4o Mini, without Description Improver Agent)

Object	Mean Initial Score	Mean Final Score	Mean Max Score	MIS	MFM	MQS
a table with four legs	1.00	1.333	1.333	0.333	10.00	1.100
a pyramid	7.000	8.333	9.667	2.667	2.00	8.400
a circular bowl	2.333	1.667	5.000	4.000	2.000	2.467
a cylindrical bottle	5.000	6.667	8.667	5.667	2.000	6.367
a curved vase	1.333	3.333	5.333	4.333	1.000	2.700
a candle with a flame	4.667	4.000	6.667	5.000	2.000	3.633
a mug with a handle	2.333	3.000	4.667	3.667	4.667	2.333
a desk lamp	2.333	2.000	3.000	2.000	10.000	2.100
a wine glass	1.000	1.000	1.333	0.333	10.000	1.033
a car with four wheels	1.000	1.000	1.667	0.667	10.000	1.100
a spiral staircase	4.333	7.000	8.333	5.333	2.000	6.200
a cathedral 1.333	1.333	1.667	0.667	10.000	1.200	
a pair of eye glasses	3.000	1.000	4.000	3.000	7.000	1.933
a wrist watch	2.333	1.333	3.333	2.333	10.000	1.867
Average	2.7332	2.999	4.444	2.711	6.178	2.942

Table 8: Results by Complexity Level (GPT-4o, with Description Improver Agent)

Level	Mean Initial Score	Mean Final Score	Mean Max Score	MIS	MFM	MQS
Low	7.000	7.200	8.800	3.733	2.600	7.267
Medium	3.733	5.267	6.600	3.867	2.933	4.900
High	3.333	4.133	5.600	3.000	5.267	4.027

Table 9: Results by Complexity Level (GPT-4o Mini, without Description Improver Agent)

Level	Mean Initial Score	Mean Final Score	Mean Max Score	MIS	MFM	MQS
Low	3.4667	4.000	5.333	2.667	5.200	4.007
Medium	2.333	2.667	4.200	3.067	5.533	2.360
High	2.400	2.333	3.800	2.400	7.800	2.460