Scene Co-pilot: Procedural Text to Video Generation with Human in the Loop

Zhaofang Qian
University of Central Florida
zh103512@ucf.edu

Abolfazl Sharifi
University of Kashan
sharifi.abolfazl@grad.kashanu.ac.ir

Tucker Carroll
University of Central Florida
tu512807@ucf.edu

Ser-Nam Lim
University of Central Florida
sernam@ucf.edu

Abstract

Video generation has achieved impressive quality, but it still suffers from artifacts such as temporal inconsistency and violation of physical laws. Leveraging 3D scenes can fundamentally resolve these issues by providing precise control over scene entities. To facilitate the easy generation of diverse photorealistic scenes, we propose Scene Copilot, a framework combining large language models (LLMs) with a procedural 3D scene generator. Specifically, Scene Copilot consists of Scene Codex, BlenderGPT, and Human in the loop. Scene Codex is designed to translate textual user input into commands understandable by the 3D scene generator. BlenderGPT provides users with an intuitive and direct way to precisely control the generated 3D scene and the final output video. Furthermore, users can utilize Blender UI to receive instant visual feedback. Additionally, we have curated a procedural dataset of objects in code format to further enhance our system's capabilities. Each component works seamlessly together to support users in generating desired 3D scenes. Extensive experiments demonstrate the capability of our framework in customizing 3D scenes and video generation. For the best visualization, please visit our project page https://abolfazl-sh.github.io/Scene_co_pilot_site/.

1. Introduction

Video generation has emerged as a vibrant and actively investigated area of research. Current state-of-the-art models such as Open AI's Sora [5], Kling [34], and StreamingT2V [23] can produce movie-quality videos of several minutes in length. However, both industrial and research-level models suffer from numerous limitations ranging from unrealistic and incoherent motion, intermittent object disappearances, to incorrect interactions between multiple objects [56]. Two particularly noticeable and counterintuitive issues are violations of physical laws and temporal inconsistencies.

To address these challenges, one promising direction is to generate videos based on a preexisting scene. By creating 3D entities within the scene, the aforementioned limitations can be automatically avoided, and the generated videos could theoretically be infinitely long. Embodied AI and applications in autonomous driving have motivated the community to create numerous scene datasets such as WayveScenes101 [75], nuScenes [8], and EmbodiedScan [60]. However, the creation and collection of these datasets can be a time-intensive process. Infinigen, a photorealistic procedural scene generator based on Blender [12], has demonstrated the potential to alleviate such laborious work [48, 49].

Meanwhile, large language models (LLMs) have demonstrated outstanding ability to efficiently assist with various tasks across different domains [11, 41, 63]. Inspired by this, we've enhanced Infinigen's capabilities by selectively incorporating LLMs. Additionally, we've expanded upon its supporting assets by curating a dataset of procedural objects. Our dataset consists of over 300 node-based assets in code format, enabling LLMs to creatively combine assets according to user requests. We also include a backup modular text-to-3D model to generate the required elements when using geometry nodes.

While Infinigen is powerful, it remains esoteric because of its prerequisite in comprehensive understanding of generation rules and mathematical algorithms. Procedural generation with geometry nodes in Blender would remain niche without an intuitive user interface. Utilizing our dataset, we propose a framework named Scene Copilot to integrate Infinigen with LLMs, making the pipeline more controllable and user-friendly. Leveraging existing LLMs also renders our system training-free and updatable as more powerful models become available (Sec. 3.4).

Specifically, given a user's textual input, the first LLM interprets the prompt, incorporating additional related contexts from the Infinigen codebase, and generates a base scene Python command. The initial base scene with a simple and crude setting is generated, allowing users to control the details and swiftly update the object settings in the scene with the help of another LLM. Intermediate users can directly interact with the scene using Blender's graphical user interface (GUI), while beginners can also utilize advanced Blender features with the assistance of the LLM. After updating the base scene, Infinigen continues to add more de-

tails. Users can review the finished scene and make further updates before the rendering process starts. As such, with the help of LLMs, general users can benefit from the advanced features in Infinigen and Blender as these two powerful engines are updated.

Our contributions can be summarized as follows:
- We've curated a procedural asset dataset in code format.
- We've proposed a training-free framework Scene Copilot and demonstrated its controllability on generated scenes and videos.

2. Related Work

2.1. Synthesis Scene Dataset

Virtual 3D scenes have been widely deployed in fields such as simulation, computer vision, and interactive systems [44, 58, 66, 68]. However, real-world scene data is limited due to the laborious and time-consuming nature of data collection. To address the scarcity of real-world data, researchers have turned to synthetic data generation [7, 13, 16]. Hypersim [51] is an indoor synthetic scene dataset containing objects such as chairs, TVs, tables, and lamps. In contrast, GOS [64] offers outdoor models suitable for tasks that rely on outdoor scene representations. Infinigen [48] is a framework designed to procedurally generate photorealistic 3D scenes within Blender, with the generation process guided by Python APIs. The framework includes a foundational dataset, and due to the randomization of parameters in the generation process, each generated scene is unique.

2.2. Procedural Generation

Procedural Content Generation has been extensively studied and applied in industrial production [22, 40, 50, 53]. Recently, Blender introduced geometry nodes to provide an intuitive UI for artists. Procedural generation using geometry and material nodes allows for automatic manipulations of models [22, 26, 54, 70]. The authors in [27] propose a framework that creates Geometry-Graph materials based on text conditions and images. Text conditions are converted into embedded vectors, which are then mapped to corresponding images. Subsequently, procedural materials are generated from these images. The resulting materials are structured as node graphs, facilitating user-driven modifications. Additionally, a procedural material dataset is introduced to enhance the quality of generated materials.

2.3. Large Language models

Large Language Models such as GPT-4o [46] and Claude-3.5-Sonnet [1] have demonstrated exceptional capabilities in coding, reasoning, and language comprehension. Their potential in code understanding and generation can be leveraged for procedural 3D modeling, where user prompts

are converted into corresponding code by the model. LLplace [67] utilizes LLMs to transform user prompts into corresponding code to create 3D scenes, allowing users to precisely specify the location of target objects.

Fine-tuning LLMs is a prominent approach to improve the quality of generated 3D models [6, 39, 45]. Incorporating contextual information into the 3D model generation process enables LLMs to operate with greater precision. For instance, Liu et al. [38] focus on a Multi-modal Large Language Model (MLLM) for generating 3D models from user prompts, enhancing the user's input description prior to model generation. The integration of extensive linguistic knowledge facilitates the analysis of the semantics within user prompts, enabling a more accurate understanding of user intentions [9, 14, 42, 67]. Fang et al. [18] employ LLMs to enhance the flexibility of scene editing for users. In their method, the 3D model is converted into 2D space using a Hash-Atlas network. Editing models in 2D space simplifies the complexity of fusion design. Additionally, few-shot learning approaches have become a popular strategy for enhancing LLM-based methods [20, 43, 57, 74]. These approaches provide structured guidance for generating 3D models from a limited number of examples. This guidance reduces processing time while improving output model quality.

2.4. Text-to-Scene

Recent progress in Text-to-image models have promoted more precise text-to-scene generation [31, 33, 35]. In [71], the authors propose a method for converting user prompts into photorealistic scenes by transforming prompts into image priors from different angles, which are then to NeRF model. SceneScape [19], converts a given prompt into a series of images with overlapping and shared segments, which are subsequently used to generate a long-term video that depicts a walkthrough of the scene. Additionally, several methods rely on large image datasets [30, 65, 69]. These methods map text to corresponding images within the dataset and synthesize scenes based on the matched visual content.

Agent-based approaches have also been explored [28, 55, 73]. In [55], the authors present a text-to-3D model platform named 3D-GPT. The proposed method employs a multi-agent system for reasoning, planning, and tool utilization to convert user prompts into 3D models. [73] presents a similar approach based on multi-agent systems to manage scene generation. In this framework, a task-planning agent assigns various tasks to subordinate agents to accomplish the primary objective.

3. Dataset

Collecting procedural objects is a time-consuming and tedious task as assets are limited in both quantity and quality.
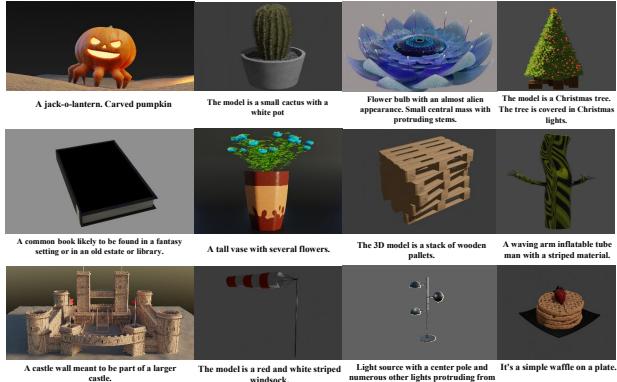
Figure 1. Dataset Samples: A subset of assets from our dataset, accompanied by corresponding descriptions. All assets are fully procedural, meaning their components can be modified either manually or automatically.

In this work, we aim to address this gap by compiling a procedural dataset to support LLM-driven code generation approaches. To achieve this, we systematically gathered Blender-based procedural geometry and materials assets from the Internet. After constructing the dataset, we utilized the Node Transpiler from Infinigen and Node2Python [4] to convert assets and materials into Python code. This conversion enables automated modification of asset features, facilitating the generation of new assets.

3.1. Dataset Details

Our dataset consists of 323 procedural assets and materials, covering a wide range of categories to provide a comprehensive resource (Table 1).

Assets Category	Num. Generator
Indoors	29
Outdoors	8
Terrain	6
Rocks	7
Plants	17
Trees	10
Weather	7
Foods	9
Scattering	24
Materials	206
Total	323

Table 1. Dataset Details: All components are procedural and have been converted to code, allowing for automated modifications.

A selection of sample objects is shown in Figure 1. In addition to utilizing our primary dataset, we integrated a supplementary dataset to generate assets exhibiting minimal similarity to existing assets within our collection.

To achieve this, we employed Shap-E [32], a text-to-3D diffusion-based model, to create a diverse array of objects. Shap-E is trained on Neural Radiance Fields (NeRF) and further refined through Signed Distance Fields (SDF) rendering, enhancing the output quality. Notably, Shap-E is a lightweight yet precise model, effectively translating user prompts into detailed 3D objects. Asset generation was guided by a probability function derived from a Gaussian distribution, which determines whether to use LLMs with our procedural dataset or Shap-E for object creation. Objects generated by Shap-E can also be used in tandem with our procedural dataset or primitive objects (Section 3.2).

We have collected 206 materials to facilitate the generation of assets with diverse textures, partially shown in Figure 2. The integration of these materials with generated assets can be performed either manually or automatically. To streamline the combination process and enable automatic modification of material features, we maintain all materials in code format.

Figure 2. Materials in our dataset can serve as textures, and they can be automatically integrated with various assets, allowing for seamless combination and enhancing visual coherence.

As shown in Figure 3, our procedural assets can be generated with a variety of material types, enhancing the diversity available for user creation. The integration of materials with procedural assets is demonstrated in Figure 4. Material selection for each asset can be determined either by user prompts or by analyzing the conceptual attributes of the asset. For instance, the leaves of trees can be rendered in green, yellow, or orange, reflecting natural variations.

3.2. Modular Objects

Combining Shap-E with our dataset enables the generation of more complex assets or entire scenes. For example, in Figure 5, Shap-E was used to generate the desk and the stack of papers, while our procedural dataset was used to generate the lamp. Objects themselves can also be modular, allowing them to be combined to inherit or exert traits of other objects, as shown in Figure 5.

Figure 3. Various types of materials are employed as textures for a table asset, facilitating the effective fulfillment of user prompts. These materials enable seamless integration and enhance the versatility of the assets to meet diverse user requirements.
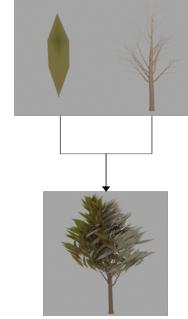

Figure 4. Materials are combined with assets automatically, with selection guided either by user prompts or by identifying materials relevant to each specific asset. Additionally, users have the option to manually apply materials to assets according to their individual requirements.
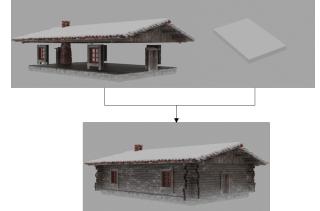
3.3. Accessibility

An essential characteristic of any dataset is its accessibility and usability for the broader research community. In this context, the dataset presented in our work represents a significant contribution to the field and constitutes one of the key innovations of this work. The dataset has been meticulously curated to support further research, development, and experimentation.

This comprehensive dataset includes procedural assets, the corresponding source code, direct links to the assets, and detailed metadata describing each component. The procedural assets are organized in a manner that allows for easy integration and use in related studies, while the descriptions provide contextual information to assist in the understanding and application of each asset. The inclusion of code further enables reproducibility and the validation of results, a core principle in contemporary research.
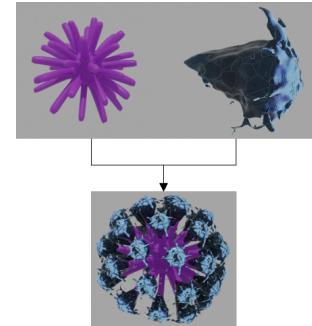
The dataset is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license, ensuring that it can be freely accessed, shared, and adapted, provided proper attribution is given. By making this dataset openly available, we aim to foster collaboration, support future research, and enhance the transparency and reproducibility of experimental work within the community.
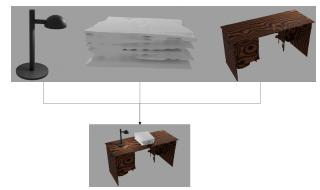

(a) A leaf from our procedural dataset (top left) combined with a tree with no leaves, also from our procedural dataset (top right) to create a fully procedural tree with leaves (bottom).

(b) A cottage generator from our procedural dataset (top left) combined with a primitive user-created cube mesh (top right) resulting in a cottage made of logs matching the shape specified by the primitive mesh object (bottom).

(c) The "FlowerBulb" object from our procedural dataset (top left) and an object generated by Shap-E with the prompt "alien fruit which would be found growing from an alien plant" (top right) combined to create an alien flower with custom fruit (bottom).

(d) A lamp from our procedural dataset (top left), a "stack of papers" generated with Shap-E (top middle), and a "desk" generated with Shap-E (top right) combined to create a more complicated and nuanced object (bottom).

Figure 5. Combining procedural assets and generative models to create diverse, modular objects.

3.4. Expanding upon the Dataset

New procedural assets are released for Blender every single day. As such, the number of procedural assets available is infinitely expanding, the vast majority of which are locked behind paywalls or subscriptions. Thus, in order to keep our dataset free and open-source, we have included scripts that can automatically integrate users' objects into our procedural dataset.

4. Scene Copilot

To better control the generation of scenes, we combine Infinigen with LLM to generate the necessary commands based on user requests (Sec 4.1). We then utilize another LLM with chain-of-thought to assist users when editing the scene in Blender (Sec 4.2). The full pipeline of Scene Copilot is shown in Figure 6.
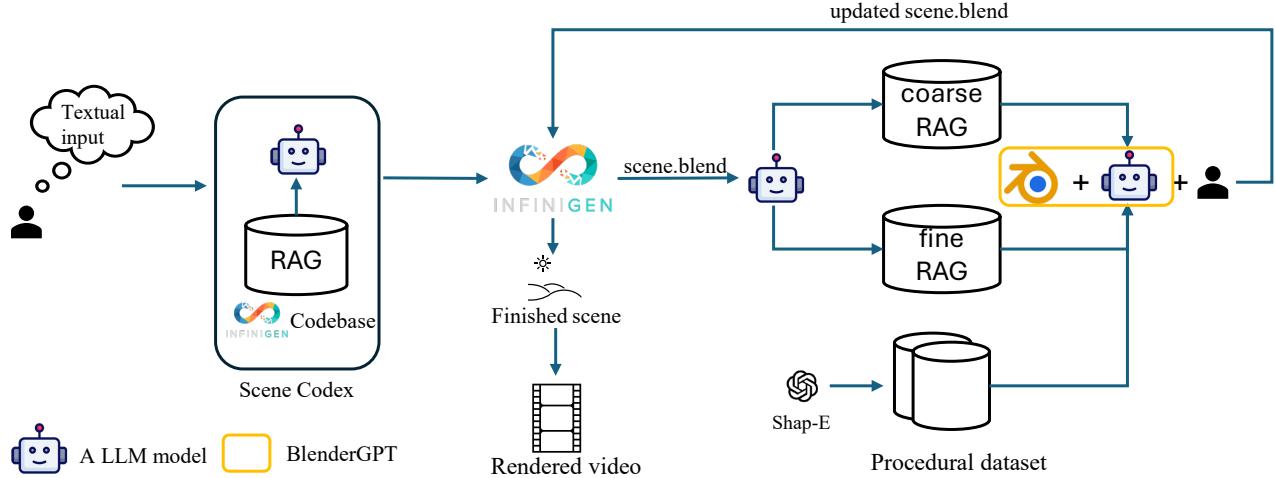
Figure 6. Overview of Scene Copilot with procedural dataset. Starting with a user textual input, Scene Codex combines an LLM with an RAG database of the Infinigen code and generates an Infinigen executable Python command. Infinigen initially creates a coarse scene, which is converted into textual format so that LLMs can comprehend objects and metadata in the scene. Such scene file is condensed into a coarse RAG database. BlenderGPT, incorporating Blender and LLM, utilizes this database to edit and modify the 3D contents in the scene with user involved through either textual or visual interaction. The updated coarse scene is fed back to Infinigen to create a fine scene. Similar to coarse scene, it is condensed into a fine RAG database, and BlenderGPT collaborates both databases while editing the final scene. Meanwhile, The procedural dataset will provide the requested procedural asset code. The finished scene is then rendered by Inifinigen and outputs the requested video.

4.1. Scene Codex

Procedural generation is powerful yet complex as countless rules and mathematical formulas are involved in each object generator. There are more than 1,000 human-interpretable parameters in Infinigen and even more for low-level external parameters [48]. This complexity hinders general designers from utilizing Infinigen to swiftly generate 3D scenes or assets. However, LLMs have shown remarkable performance in interpreting large codebase and generating high-quality code [17, 37, 52].

Similar to Codex [11], Scene Codex is designed to convert user intentions into Infinigen commands to create the base scene. To improve the accuracy and correctness of the generated commands, we power our LLM with Retrieval-augmented generation (RAG) [36] using the Infinigen codebase, given its effectiveness in LLMs answer quality [21]. Guided by few-shot *description-command* example pairs, our Scene Codex converts the user's textual input directly into an Infinigen executable Python command. We applied Claude-3.5-Sonnet as our LLM model and FAISS [15] as the RAG database. See Appendix for more details on the system prompt and few-shot examples of Scene Codex.

4.2. BlenderGPT

As shown in Figure 6, Infinigen first generates a coarse scene, and users can edit and update details with our

adapted BlenderGPT[1], a Blender add-on with LLMs to directly convert text input to Blender code. We follow the idea of chain-of-thought [62], such that BlenderGPT converts the user's prompt into initial steps, reflects on them, and forms the final procedures with objects related to the task. Subsequently, another LLM generates the executable `bpy` code closely following the procedure, consulting with the textual scene description with the coarse and fine RAG database detailed below. The edited scene is fed back to Infinigen to generate a refined scene with more objects and details. BlenderGPT repeats the previous process to allow users to make a final adjustment. The procedural dataset is integrated with BlenderGPT at both stages to provide the required code of procedural geometry or material nodes. After the user's final confirmation, Infinigen populates the final blender scene and outputs the rendered video.

Preprocessing Blender Scene BlenderGPT combines LLMs with Blender so that it can be controlled by natural languages and support multi-round conversations in direct and simple English. However, it defaults from a blank Blender project and lack the ability to understand any imported 3D scenes. Therefore, it is crucial to introduce the base scene to the BlenderGPT. As LLMs can only understand textual input, we convert Blender scenes into `.usda` files that describe each scene in textual format. Figure 7 shows an example of a camera code snippet in a `.usda`

---

[1]https://github.com/gd3kr/BlenderGPT

```
def Xform "CameraRigs_0"
{
    matrix4d xformOp:transform.timeSamples = { ▪ }
    uniform token[] xformOpOrder = ["xformOp:transform"]

    def Xform "CameraRigs_0_0"
    {
        matrix4d xformOp:transform = ( (1, 1.864466803169762e
        1.6971407612587086e-9, -0.9284766912460327, 0.3713907
        uniform token[] xformOpOrder = ["xformOp:transform"]

        def Camera "Camera_001"
        {
            float2 clippingRange = (0.1, 10000)
            float focalLength = 35.19107
            float horizontalAperture = 32
            float horizontalApertureOffset = 0
            token projection = "perspective"
            float verticalAperture = 18
            float verticalApertureOffset = 0
        }
    }
}
```

Figure 7. Code segment from converted .usda file. It shows a snippet of camera configurations, including the camera's SE(3)(xformOp:transform), focal length, and aperture. All these parameters are also available in Blender's GUI.
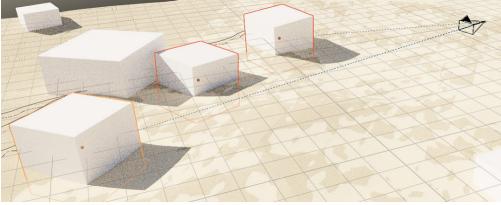

Figure 8. Image demo

file, which includes metadata ranging from the scene terrain to object animation information. After acquiring the raw .usda file that describes the whole scene, we utilize a lightweight LLM (GPT-4o mini [47]) to further convert and clean the textual contents into a dictionary format and replace the numerical data with **NUM** placeholder to concise the whole context, such as:

```
    matrix4d xformOp:transform = {...}
 -> matrix4d xformOp:transform = NUM
```

Such condensation helps LLMs focus on the interpretable parameters instead of being overwhelmed by numerical information. The cleaned scene file is then segmented into non-overlapping chunks and stored as a RAG database. Both the coarse and fine RAG databases are connected to BlenderGPT to help create and interact with the scene objects in later stages. See Appendix for more details on the prompt initialization of BlenderGPT.

Human in the Loop Blender is designed for 3D editing, and it is intuitive and efficient to interact with 3D scenes through the GUI. Therefore, to allow users to control the objects in the scene precisely, we believe it is essential to preserve the user interface. With both visual and textual interactions available, users can select an object and then prompt BlenderGPT with a textual request. Figure 8 demonstrates an example of asking the camera to follow a

snake's movement in the scene. Three snakes are shown as white cubic placeholders highlighted with orange sides in this coarse scene. The camera was originally following the snake far from the viewpoint. As there are multiple snakes in the scene, it could be challenging and inefficient to use only textual input to describe the desired object. Instead, the user can directly click on the snake object closest to the viewpoint with the textual input "follow the selected object during the whole animation".

5. Experiment

We demonstrate the capabilities of Scene Copilot by rendering videos from generated Blender scenes.
Implementation details We choose Claude-3.5-Sonnet [1] as the model in Scene Codex and GPT-4o [46] as the model in BlenderGPT. To avoid hallucinations and ensure relevance to the given prompts, we set the decoding temperature to 0. All methods are implemented using PyTorch 2.4.0 and executed on multiple NVIDIA H100 GPUs.

5.1. Qualitative Results

In Figure 9, we present five samples of eight key frames from the rendered videos. We start with a base scene and provide additional prompts to update objects and details using Scene Copilot. After direct GUI-based revision such as moving camera positions or selecting desired subjects, the final output videos closely follow the textual prompts. In Figure 10, we compare the rendered videos between the direct output videos from Infinigen and the results after editing with Scene Copilot. Note that we optimized Scene Codex specifically for two different tasks. Because of Infinigen's randomness, the direct output videos have a high probability of not focusing on the main subjects or may even fail to generate the requested assets. In contrast, with Scene Copilot, the user, acting as a "director", can have more control over the scene and the output video. For example, as illustrated in Figure 10a, since Infinigen does not include a *"graveyard"* in the asset, the camera is pointed in a random direction. However, using BlenderGPT, we generated a church and gravestones with fixed camera animation.

5.2. Quantitative Results

To quantitatively evaluate the output videos, we applied VBench [29], a comprehensive video evaluation benchmark with multiple dimensions, and targeted on 100 textual prompts in *scenery* category as Infinigen's major strength lies in creating natural photorealistic scenes. We created scenes using VBench text prompts as the only input and edited details with BlenderGPT. The average editing time was approximately 20 minutes. Table 2 summarizes our main results compared with other AI-driven (transformer/diffusion-based) video generation models including LaVie [61], ModelScope [59] CogVideo [25], and
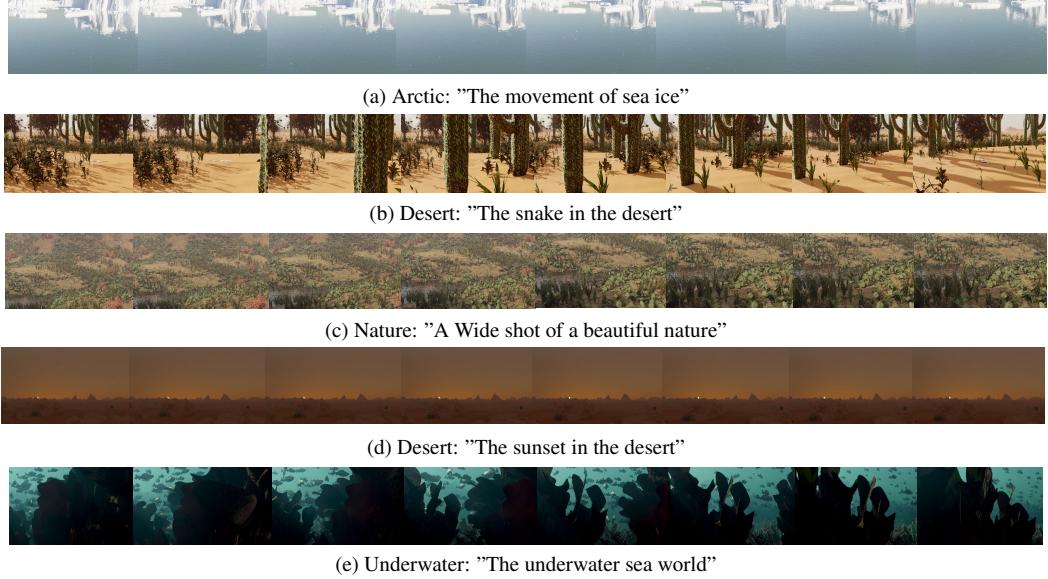
(a) Arctic: "The movement of sea ice"

(b) Desert: "The snake in the desert"

(c) Nature: "A Wide shot of a beautiful nature"

(d) Desert: "The sunset in the desert"

(e) Underwater: "The underwater sea world"

Figure 9. Five samples of eight key frames from generated videos with the corresponding textual input prompt.

(a) "graveyard at sunset"

(b) "a relaxing scenery of beach view under cloudy sky"

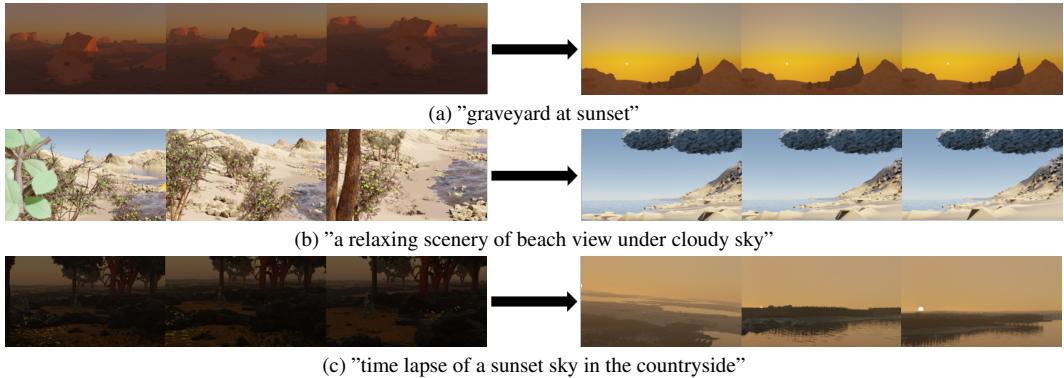(c) "time lapse of a sunset sky in the countryside"

Figure 10. **Left** Three key frames from rendered videos of the original scene output from Infinigen. **Right** Three key frames from rendered videos of the edited scene using Scene Copilot. Associated text prompts from VBench [29] are used as the direct textual input.

VideoCrafter1 [10]. Overall, our generated videos show competitive results with similar scores compared to other autoregressive text-to-video(T2V) models. Specifically, Scene Copilot achieves the best results in Motion Smoothness and Dynamic Degree and the second-best in Aesthetic Quality and Imaging Quality. Both Motion Smoothness and Dynamic Degree evaluate whether the motion in the generated video is smooth and follows the physical law of the real world [29]. Achieving the best results in these two dimensions and second-best in frame-wise quality demonstrates great potential and feasibility in generating video through Infinigen, enhanced by our Scene Copilot.

5.3. Ablation

As elaborated in Section 4.1, our Scene Codex combines RAG and few-shot examples to generate the Infinigen commands. We investigate the effect of each component on the generated commands. We used textual prompts from VBench as input and measured the Executable Rate (ER@1) by calculating the proportion of generated commands that are executable, to evaluate the usability of our Scene Codex [11, 72]. Table 11 shows the results ranging from using only the LLM to the final model with both techniques applied. It is evident that both the RAG database of Infinigen and few-shot examples are valuable to our Scene Codex. Without including any references, it is impossible to directly generate any executable Infinigen commands. With few-shot examples, Scene Codex imitated examples and produced 2.0% runnable commands. Extra context from RAG database helps the LLMs understand the code relationships, boosting the runnable commands to 20.0%. Finally, when we include both components in our Scene

Table 2. **VBench result on Scenery category.** We compare our Scene Copilot with the other four T2V AI-based models in the Scenery category across eight dimensions. The best result in each dimension is highlighted in bold, and the second-best result is highlighted in Italics.

Models	Subject Consistency	Background Consistency	Motion Smoothness	Dynamic Degree	Aesthetic Quality	Imaging Quality
LaVie	**97.27**	**97.06**	*97.58*	6.40	**51.76**	**63.86**
ModelScope	*94.88*	*95.57*	97.03	26.00	48.57	57.49
VideoCrafter1	89.67	92.86	94.17	*51.80*	43.06	58.98
CogVideo	95.27	95.46	*97.58*	13.20	46.72	40.49
Scene Copilot (Ours)	93.10	95.17	**99.36**	**66.67**	*49.24*	*62.10*

Figure 11. The evaluation result of prompting RAG database Infinigen and few-shot examples. Based on the result, Using both of them are valuable and vital for our Scene Codex.

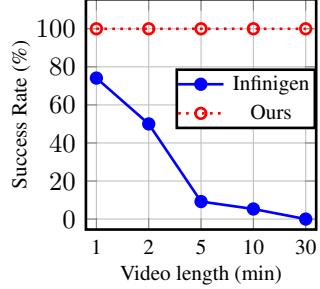|	RAG	Few-Shot	ER@1
1			0.0%
2		✓	2.0%
3	✓		20.0%
4	✓	✓	**43.2**%

Figure 12. Video generation success rate comparison between Infinigen and Ours across different video lengths. Infinigen's performance decreases as video length increases.

Codex, the ER@1 reaches 43.2%. This result demonstrates the value of including additional information when prompting LLMs.

5.4. Long-Video Generation

Many models have demonstrated the ability to generate high-quality short-duration videos. However, generating videos spanning minute-level or even hour-level durations remains a challenge, particularly with respect to maintaining object integrity and adhering to physical laws [2, 3, 24]. One advantage of generating videos from a pre-constructed scene is the consistency of the objects within that scene. This consistency allows for the creation of videos of arbitrary length without concern for the preservation of object integrity or the enforcement of the physical laws governing the generated world. We evaluated the long-video generation capability of Infinigen and Infinigen with Scene Copilot. We generated 12 scenes in each Infinigen-provided procedural scene setting and calculated the success rate at different video lengths, as shown in Figure 12. From the figure, Infinigen is capable of generating one-minute-long videos with a 74.1% success rate, but the success rate decreases as the requested video length increases. This is because Infinigen has to find an existing camera path spanning the whole video length before it can create the scene. In contrast, benefiting from generating scenes with human-

in-the-loop, users can start with a one-second-long scene and customize the camera animation for videos of arbitrary length.

6. Conclusion

In this work, we present Scene Copilot, an automatic pipeline combining procedural scene generation and human-centered editing with additional procedural assets. Scene Copilot consists of Scene Codex and BlenderGPT. Utilizing remarkable LLMs, Scene Codex converts user prompts into an Infinigen-based Python command to create the base 3D scene. BlenderGPT then helps users accurately revise and update the 3D scene with visual and textual human input. Additionally, our procedural dataset provides more than 300 extra geometric and material assets to make Scene Copilot more versatile. Supported by unprecedented LLMs and our procedural dataset, Scene Codex, and BlenderGPT could help users control the 3D scene generation with high precision. We hope Scene Copilot can lower the bound of creating high-quality, dynamic, and controllable 3D scenes.

Limitations Although Scene Copilot demonstrated promising ability in supporting users' 3D scene editing, we would like to acknowledge several assumptions and limitations: 1) Because we introduced LLMs in Scene Copilot, the hallucination of LLMs is inevitable, resulting in advanced programming knowledge from user to help correct issues and mistakes. This issue may shield the power of introducing LLMs in the framework, but it could be alleviated with advancements of more powerful LLMs. 2) The amount of procedural objects in our datasets is limited, compared with other 3D model datasets in other formats such as mesh and point cloud. Creating procedural geometry objects is time-consuming and most of them are proprietary. We plan to continuously include new assets in our datasets, and we believe it is possible to generate procedural geometry assets with multi-modal LLMs and neural networks.

References

[1] Anthropic. Claude 3.5 sonnet, 2024. 2, 6

[2] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023. 8

[3] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023. 8

[4] carls3d BrendanParmer. Nodetopython. https://github.com/BrendanParmer/NodeToPython, 2023. 3

[5] Tim Brooks, Bill Peebles, Connor Homes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Wing Yin Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. 1

[6] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 2

[7] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12*, pages 611–625. Springer, 2012. 2

[8] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving, 2020. 1

[9] Yang Cao, Zeng Yihan, Hang Xu, and Dan Xu. Coda: Collaborative novel box discovery and cross-modal alignment for open-vocabulary 3d object detection. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[10] Haoxin Chen, Menghan Xia, Yingqing He, Yong Zhang, Xiaodong Cun, Shaoshu Yang, Jinbo Xing, Yaofang Liu, Qifeng Chen, Xintao Wang, Chao Weng, and Ying Shan. Videocrafter1: Open diffusion models for high-quality video generation. *arXiv preprint arXiv:2310.19512*, 2023. 7

[11] 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. 1, 5, 7

[12] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 1

[13] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022. 2

[14] Runyu Ding, Jihan Yang, Chuhui Xue, Wenqing Zhang, Song Bai, and Xiaojuan Qi. Pla: Language-driven open-vocabulary 3d scene understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7010–7019, 2023. 2

[15] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024. 5

[16] Matteo Fabbri, Guillem Brasó, Gianluca Maugeri, Orcun Cetintas, Riccardo Gasparini, Aljoša Ošep, Simone Calderara, Laura Leal-Taixé, and Rita Cucchiara. Motsynth: How can synthetic data help pedestrian detection and tracking? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10849–10859, 2021. 2

[17] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. Large language models for code analysis: Do LLMs really do their job? In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 829–846, Philadelphia, PA, 2024. USENIX Association. 5

[18] Shuangkang Fang, Yufeng Wang, Yi-Hsuan Tsai, Yi Yang, Wenrui Ding, Shuchang Zhou, and Ming-Hsuan Yang. Chatedit-3d: Interactive 3d scene editing via text prompts. In *European Conference on Computer Vision*, pages 199–216. Springer, 2025. 2

[19] Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[20] Rao Fu, Jingyu Liu, Xilun Chen, Yixin Nie, and Wenhan Xiong. Scene-llm: Extending language model for 3d visual understanding and reasoning. *arXiv preprint arXiv:2403.11401*, 2024. 2

[21] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. 5

[22] Cristina Gasch, José Martínez Sotoca, Miguel Chover, Inmaculada Remolar, and Cristina Rebollo. Procedural modeling of plant ecosystems maximizing vegetation cover. *Multimedia Tools and Applications*, 81(12):16195–16217, 2022. 2

[23] Roberto Henschel, Levon Khachatryan, Daniil Hayrapetyan, Hayk Poghosyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Streamingt2v: Consistent, dynamic, and extendable long video generation from text. *arXiv preprint arXiv:2403.14773*, 2024. 1

[24] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022. 8

[25] Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. CogVideo: Large-scale pretraining for text-to-video generation via transformers. *arXiv preprint arXiv:2205.15868*, 2022. 6

[26] Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. An inverse procedural modeling pipeline for svbrdf maps. *ACM Transactions on Graphics (TOG)*, 41(2):1–17, 2022. 2

[27] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Generating procedural materials from text or image prompts. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023. 2

[28] 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

[29] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 6, 7

[30] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5885–5894, 2021. 2

[31] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 867–876, 2022. 2

[32] Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions, 2023. 3

[33] N Khalid, T Xie, E Belilovsky, and T Popa. *Clip-mesh: Generating textured meshes from text using pretrained image-text models*. PhD thesis, Concordia University Montréal, Québec, Canada, 2023. 2

[34] Kuaishou. Kling AI. https://klingai.kuaishou.com, 2024. 1

[35] Han-Hung Lee and Angel X Chang. Understanding pure clip guidance for voxel grid nerf models. *arXiv preprint arXiv:2209.15172*, 2022. 2

[36] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. 5

[37] Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts?, 2024. 5

[38] Dingning Liu, Xiaoshui Huang, Yuenan Hou, Zhihui Wang, Zhenfei Yin, Yongshun Gong, Peng Gao, and Wanli Ouyang. Uni3d-llm: Unifying point cloud perception, generation and editing with large language models. *arXiv preprint arXiv:2402.03327*, 2024. 2

[39] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021. 2

[40] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. *Deep learning for procedural content generation*. Springer, 2021. 2

[41] Ming Y Lu, Bowen Chen, Drew FK Williamson, Richard J Chen, Melissa Zhao, Aaron K Chow, Kenji Ikemura, Ahrong Kim, Dimitra Pouli, Ankush Patel, et al. A multimodal generative ai copilot for human pathology. *Nature*, pages 1–3, 2024. 1

[42] Shiyang Lu, Haonan Chang, Eric Pu Jing, Abdeslam Boularias, and Kostas Bekris. Ovir-3d: Open-vocabulary 3d instance retrieval without training on 3d data. In *Conference on Robot Learning*, pages 1610–1620. PMLR, 2023. 2

[43] Xianzheng Ma, Yash Bhalgat, Brandon Smart, Shuai Chen, Xinghui Li, Jian Ding, Jindong Gu, Dave Zhenyu Chen, Songyou Peng, Jia-Wang Bian, et al. When llms step into the 3d world: A survey and meta-analysis of 3d tasks via multi-modal large language models. *arXiv preprint arXiv:2405.10255*, 2024. 2

[44] Jiageng Mao, Minzhe Niu, Chenhan Jiang, Hanxue Liang, Jingheng Chen, Xiaodan Liang, Yamin Li, Chaoqiang Ye, Wei Zhang, Zhenguo Li, et al. One million scenes for autonomous driving: Once dataset. *arXiv preprint arXiv:2106.11037*, 2021. 2

[45] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022. 2

[46] OpenAI. Hello gpt-4o. *OpenAI Blog*, 2024. 2, 6

[47] OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. *OpenAI Blog*, 2024. 6

[48] 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*, pages 12630–12641, 2023. 1, 2, 5

[49] 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 (CVPR)*, pages 21783–21794, 2024. 1

[50] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2(8):428–436, 2020. 2

[51] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10912–10922, 2021. 2

[52] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt,

Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. 5

[53] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016. 2

[54] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match: Differentiable material graphs for procedural material capture. *ACM Transactions on Graphics (TOG)*, 39(6): 1–15, 2020. 2

[55] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models, 2024. 2

[56] Rui Sun, Yumin Zhang, Tejal Shah, Jiahao Sun, Shuoying Zhang, Wenqi Li, Haoran Duan, Bo Wei, and Rajiv Ranjan. From sora what we can see: A survey of text-to-video generation. *arXiv preprint arXiv:2405.10674*, 2024. 1

[57] Yuan Tang, Xu Han, Xianzhi Li, Qiao Yu, Yixue Hao, Long Hu, and Min Chen. Minigpt-3d: Efficiently aligning 3d point clouds with large language models using 2d priors. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 6617–6626, 2024. 2

[58] Madhawa Vidanapathirana, Qirui Wu, Yasutaka Furukawa, Angel X Chang, and Manolis Savva. Plan2scene: Converting floorplans to 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10733–10742, 2021. 2

[59] Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. Modelscope text-to-video technical report. *arXiv preprint arXiv:2308.06571*, 2023. 6

[60] Tai Wang, Xiaohan Mao, Chenming Zhu, Runsen Xu, Ruiyuan Lyu, Peisen Li, Xiao Chen, Wenwei Zhang, Kai Chen, Tianfan Xue, Xihui Liu, Cewu Lu, Dahua Lin, and Jiangmiao Pang. Embodiedscan: A holistic multi-modal 3d perception suite towards embodied ai. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 1

[61] Yaohui Wang, Xinyuan Chen, Xin Ma, Shangchen Zhou, Ziqi Huang, Yi Wang, Ceyuan Yang, Yinan He, Jiashuo Yu, Peiqing Yang, et al. Lavie: High-quality video generation with cascaded latent diffusion models. *arXiv preprint arXiv:2309.15103*, 2023. 6

[62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. 5

[63] Michel Wermelinger. Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 172–178, 2023. 1

[64] Mingye Xie, Ting Liu, and Yuzhuo Fu. Gos: A large-scale annotated outdoor scene synthetic dataset. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3244–3248. IEEE, 2022. 2

[65] Dejia Xu, Yifan Jiang, Peihao Wang, Zhiwen Fan, Humphrey Shi, and Zhangyang Wang. Sinnerf: Training neural radiance fields on complex scenes from a single image. In *European Conference on Computer Vision*, pages 736–753. Springer, 2022. 2

[66] Yongzhi Xu, Yonhon Ng, Yifu Wang, Inkyu Sa, Yunfei Duan, Yang Li, Pan Ji, and Hongdong Li. Sketch2scene: Automatic generation of interactive 3d game scenes from user's casual sketches. *arXiv preprint arXiv:2408.04567*, 2024. 2

[67] Jihan Yang, Runyu Ding, Weipeng Deng, Zhe Wang, and Xiaojuan Qi. Regionplc: Regional point-language contrastive learning for open-world 3d scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19823–19832, 2024. 2

[68] Yandan Yang, Baoxiong Jia, Peiyuan Zhi, and Siyuan Huang. Physcene: Physically interactable 3d scene synthesis for embodied ai. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[69] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4578–4587, 2021. 2

[70] Jian Zhang, Chang-bo Wang, Hong Qin, Yi Chen, and Yan Gao. Procedural modeling of rivers from single image toward natural scene production. *The Visual Computer*, 35: 223–237, 2019. 2

[71] Jingbo Zhang, Xiaoyu Li, Ziyu Wan, Can Wang, and Jing Liao. Text2nerf: Text-driven 3d scene generation with neural radiance fields. *IEEE Transactions on Visualization and Computer Graphics*, 2024. 2

[72] Mengqi Zhou, Yuxi Wang, Jun Hou, Chuanchen Luo, Zhaoxiang Zhang, and Junran Peng. Scenex: Procedural controllable large-scale scene generation via large-language models. *arXiv preprint arXiv:2403.15698*, 2024. 7

[73] Mengqi Zhou, Yuxi Wang, Jun Hou, Chuanchen Luo, Zhaoxiang Zhang, and Junran Peng. Scenex:procedural controllable large-scale scene generation via large-language models, 2024. 2

[74] Lanyun Zhu, Tianrun Chen, Deyi Ji, Jieping Ye, and Jun Liu. Llafs: When large language models meet few-shot segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3065–3075, 2024. 2

[75] Jannik Zürn, Paul Gladkov, Sofía Dudas, Fergal Cotter, Sofi Toteva, Jamie Shotton, Vasiliki Simaiaki, and Nikhil Mohan. Wayvescenes101: A dataset and benchmark for novel view synthesis in autonomous driving, 2024. 1

Scene Co-pilot: Procedural Text to Video Generation with Human in the Loop

Supplementary Material

A. Prompts Initialization

A.1. Initial Prompts For Scene Codex and BlenderGPT

As presented in Table 3, the initial system prompt for Scene Codex is provided. This table outlines the specific configuration and structure required for engaging with the Scene Codex framework, offering a clear overview of the input parameters that govern its functionality. Additionally, Table 4 displays the initial prompt associated with BlenderGPT. We first prompt LLM to generate a guideline to accomplish the users' request through Chain-of-Thoughts. Then, another LLM generates detailed executable Python codes based on the provided steps.

A.2. Few-shot examples

As presented in Table 5, several *description-command* pairs, used as few-shot examples for generating scenes with Infinigen, are provided. They are used as additional guidelines for Scene Codex to generate faithful, correct Infinigen commands.

B. Detailed Procedure

The detailed procedure for Scene Copilot is presented in Alg. 1. In this algorithm, we include additional details and processes of our proposed framework. The pseudo-code provides a comprehensive overview that accurately reflects the core functionalities and aids the understanding of the workflow and operational mechanics shown in Figure 6. Each step is carefully articulated to ensure that the logical flow and interactions between components are clearly conveyed.

C. Additional Video Examples

We further rendered additional video examples to demonstrate our long-video generation capabilities. We created one 10-minute video, two 2-minute videos, three 1-minute videos, and three 30-second videos. We demonstrated Human in the Loop with BlenderGPT in a short video.

We have also included more examples of outputs from Scene Copilot in Figure 13. We have included these examples to showcase the utility of our procedural dataset and the capabilities of Scene Copilot beyond the creation of generic environments and scenery. Figure 13 focuses on prompts that relate to a subject or focal point rather than the description of an environment/setting, relying on our procedural dataset to create these subjects in each scene.

D. Video Samples

We include more video samples produced by the Scene Copilot in Figure 14. We show a single key frame of each video in various scenarios, showcasing Scene Copilot's ability to handle diverse inputs and produce coherent, contextually relevant outputs. This figure serves as a visual representation of our framework's potential, providing a clear indication of its applicability to real-world tasks.

E. New Assets

We have expanded our procedural database by adding an additional 41 material nodes assets and 12 geometry nodes assets to demonstrate the ability to extend the dataset for more precise alignment with user prompts, as shown in Figure 15 and Figure 16. This enhancement improves the system's flexibility and ability to generate more tailored outputs for a wider range of user queries.

Table 3. Scene Codex System Prompt

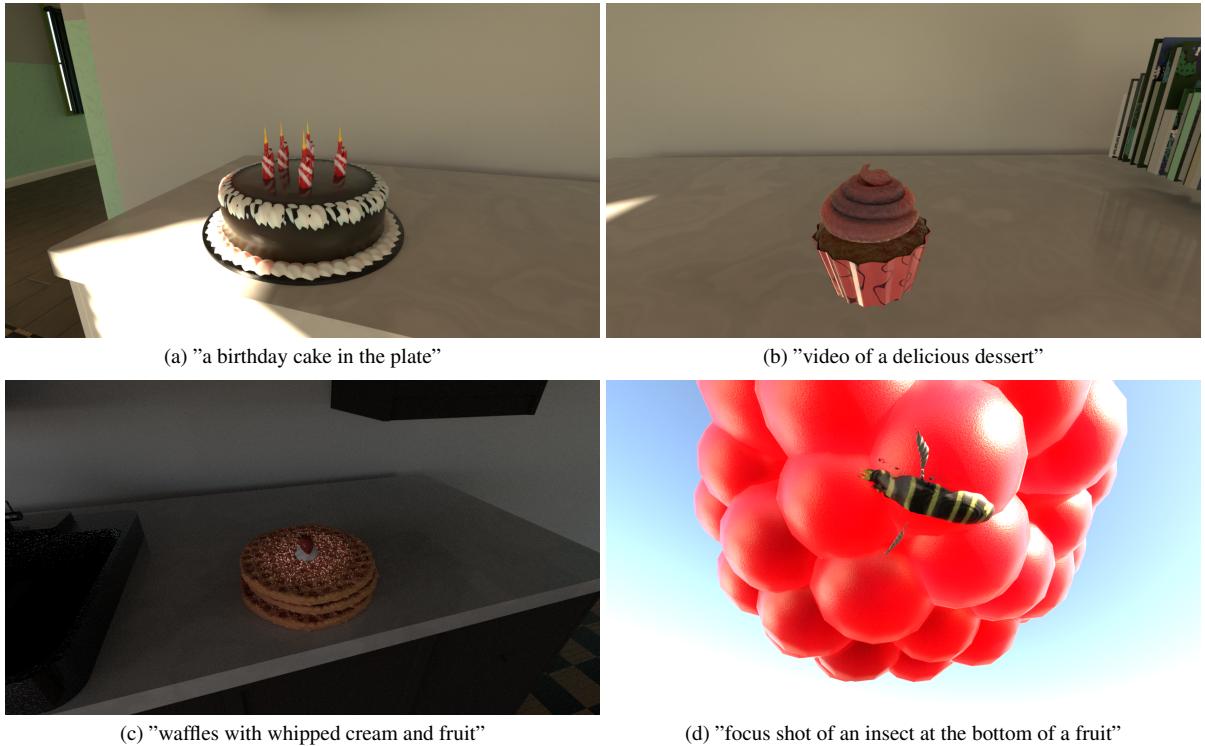|	The Prompts
Description:	You are an assistant for generating Infinigen code command based on natural language input. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Here is the helping doc for python function manage_jobs, you should only choose and include those options in your generated command after the starting code. You are an assistant for generating Infinigen code command based on natural language input. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Here is the helping doc for python function manage_jobs, you should only choose and include those options in your generated command after the starting code.
Usage:	manage_jobs.py[-h] [-o OUTPUT_FOLDER] [–num_scenes NUM_SCENES] [–meta_seed META_SEED] [–specific_seed SPECIFIC_SEED [SPECIFIC_SEED ...]] [–use_existing] [–warmup_sec WARMUP_SEC] [–cleanup (all,big_files,none,except_logs,except_crashed)] [–configs [CONFIGS ...]] [-p OVERRIDES [OVERRIDES ...]] [–wandb_mode (online,offline,disabled)] [–pipeline_configs PIPELINE_CONFIGS [PIPELINE_CONFIGS ...]] [–pipeline_overrides PIPELINE_OVERRIDES [PIPELINE_OVERRIDES ...]] [–overwrite] [-d] [-v]
Options:	-h, –help Explanation: show this help message and exit -o OUTPUT_FOLDER, –output_folder OUTPUT_FOLDER –num_scenes NUM_SCENES. Explanation: Number of scenes to attempt before terminating –meta_seed META_SEED. Explanation: What seed should be used to determine the random seeds of each scene? Leave as None unless deliberately replicating past runs –specific_seed SPECIFIC_SEED [SPECIFIC_SEED ...]. Explanation: The default, None, will choose a random seed per scene. Otherwise, all scenes will have the specified seed. Interpreted as an integer if possible. –use_existing Explanation: If set, then assume output_folder is an existing folder from a terminated run, and make a best-possible-effort to resume from where it left off –warmup_sec WARMUP_SEC Explanation: Perform a staggered start over the specified period, so that jobs don't sync up or all write to disk at similar times. –cleanup (all,big_files,none,except_logs,except_crashed) Explanation: What files should be cleaned up by the manager as it runs –configs [CONFIGS ...] Explanation: List of gin config names to pass through to all underlying scene generation jobs. Available gin files are: ['canyon.gin', 'plain.gin', 'under_water.gin', 'fast_terrain_assets.gin', 'mountain.gin', 'kelp_forest.gin', 'stereo_training.gin', 'snowy_mountain.gin', 'high_quality_terrain.gin', 'simple.gin', 'no_creatures.gin', 'no_rocks.gin', 'natural.gin', 'reuse_terrain_assets.gin', 'no_assets.gin', 'dev.gin', 'tilted_river.gin', 'arctic.gin', 'no_particles.gin', 'simulated_river.gin', 'coast.gin', 'use_cached_fire.gin', 'forest.gin', 'coral_reef.gin', 'cliff.gin', 'snake.gin', 'use_on_the_fly_fire.gin', 'tiger.gin', 'base_surface_registry.gin', 'experimental.gin', 'river.gin', 'cave.gin', 'desert.gin']. You should primarily choose files from this list. Any new ",gin" file should have detailed additional config setting. -p OVERRIDES [OVERRIDES ...], –overrides OVERRIDES [OVERRIDES ...] Explanation: List of gin overrides to pass through to all underlying scene generation jobs –wandb_mode (online,offline,disabled) Explanation: Mode kwarg for wandb.init(). Set up wandb before use. –pipeline_configs PIPELINE_CONFIGS [PIPELINE_CONFIGS ...] Explanation: List of gin config names from datagen/pipeline_configs to configure this execution. Available gin files are: ['cuda_terrain.gin', 'indoor_background_configs.gin', 'export.gin', 'base.gin', 'asset_demo.gin', 'upload.gin', 'opengl_gt.gin', 'blender_gt.gin', 'gt_test.gin', 'opengl_gt_noshortrender.gin', 'stereo.gin', 'block_terrain_experiment.gin', 'stereo_1h_jobs.gin', 'stereo_video.gin', 'monocular_video.gin', 'monocular_flow.gin', 'monocular.gin', 'slurm_cpuheavy.gin', 'slurm.gin', 'local_256GB.gin', 'local_64GB.gin', 'local_128GB.gin', 'slurm_high_memory.gin', 'slurm_1h.gin', 'local_16GB.gin'] –pipeline_overrides PIPELINE_OVERRIDES [PIPELINE_OVERRIDES ...] Explanation: List of gin overrides to configure this execution. Use command '–pipeline_overrides' rather than '–overrides' when a gin file override the 'manage_jobs.py' process, not the main 'Infinigen_examples/generate_nature.py' driver. –overwrite Explanation: Overwrite the output folder if it already exists. -d, –debug -v, –verbose
Description:	Keep the answer concise. You should always generate Infinigen commands in a single line starting with "python -m Infinigen.datagen.manage_jobs". You can provide options to user if you need based on retrieved context. And you can write your own ".gin" file from the retrieved context if needed. You can propose to change seed/including simple scene setting gin files due to insufficient GPU memory or crashed animation with max_full_retries=30 and max_step_tries=25 You should only include parameters in the context, do not hallucinate or improvise any non-existent parameters because of the request. Say you don't know how to do it if you don't know.

(a) "a birthday cake in the plate"

(b) "video of a delicious dessert"

(c) "waffles with whipped cream and fruit"

(d) "focus shot of an insect at the bottom of a fruit"

Figure 13. Screenshots from 4 different outputs from Scene Copilot and their respective prompts.

Table 4. BlenderGPT Prompt

	The Prompts
Description	You are an assistant made for the purpose of helping the user with Blender, the 3D software. - Respond with your answers in markdown ("'`"). - Preferably import entire modules instead of bits. - Do not perform destructive operations on the meshes. - Do not use cap_ends. Do not do more than what is asked (setting up render settings, adding cameras, etc) - Do not respond with anything that is not Python code. - There is a pre-generated scene.blend with objects, animations, and cameras in the scene. You should primarily use those objects in the scene and generate related code. - You will be given steps from another Chain-of-Thought agent to help accomplish the user-requested task. Generate python codes following the directions.
CoT Prompt:	Follow these steps: 1. Think through the problem step by step within the ⟨thinking⟩ tags. 2. Reflect on your thinking to check for any errors or improvements within the ⟨reflection⟩ tags. 3. Make any necessary adjustments based on your reflection. 4. Provide your final, concise answer within the ⟨output⟩ tags. Important: The ⟨thinking⟩ and ⟨reflection⟩ sections are for your internal reasoning process only. Do not include any part of the final answer in these sections. The potential related object names should be gathered from those extra context. The actual response to the query must be entirely contained within the ⟨output⟩ tags.
Format:	Use the following format for your response: ⟨thinking⟩ [Your step-by-step reasoning goes here. This is your internal thought process, not the final answer.] ⟨reflection⟩ [Your reflection on your reasoning, checking for errors or improvements] ⟨/reflection⟩ [Any adjustments to your thinking based on your reflection] ⟨/thinking⟩ ⟨output⟩ [Your final, concise answer to the query. This is the only part that will be shown to the next agent.] ⟨/output⟩

Table 5. Few-Shot Examples

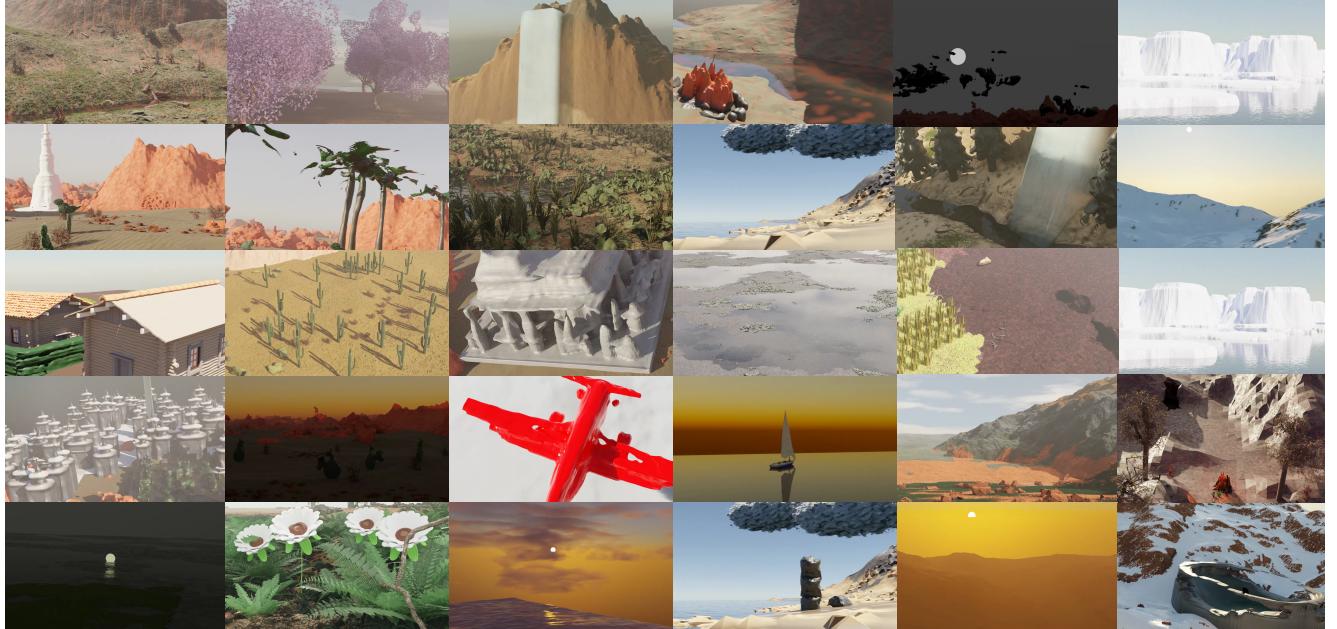|	The Examples
1	"Description": "Generate an Infinigen command that will create a low quality desert scene", "Command": "python -m Infinigen.datagen.manage_jobs output_folder outputs/hello_world –num_scenes 1 –specific_seed 0 –configs desert.gin simple.gin –pipeline_configs local_16GB.gin monocular.gin blender_gt.gin –pipeline_overrides LocalScheduleHandler.use_gpu=False"
2	"Description": "Generate an Infinigen command that will create 10000 large-scale high-quality stereo scenes", "Command": "python -m Infinigen.datagen.manage_jobs –output_folder outputs/stereo_data –num_scenes 10000 –pipeline_configs slurm.gin stereo.gin cuda_terrain.gin –cleanup big_files –warmup_sec 60000 –config high_quality_terrain"
3	"Description": "Generate an Infinigen command that will create high quality 500 videos", "Command": "python -m Infinigen.datagen.manage_jobs –output_folder outputs/my_videos –num_scenes 500 –pipeline_configs slurm.gin monocular_video.gin cuda_terrain.gin –cleanup big_files –warmup_sec 60000 –config trailer_video high_quality_terrain"
4	"Description": "Generate an Infinigen command that will create a few (50) low-resolution scenes", "Command": "python -m Infinigen.datagen.manage_jobs –output_folder outputs/dev –num_scenes 50 –pipeline_configs slurm.gin monocular.gin cuda_terrain.gin –cleanup big_files –warmup_sec 1200 –configs dev"
5	"Description": "Generate an Infinigen command that will create a image that always have rain:", "Command": "python -m Infinigen.datagen.manage_jobs –output_folder outputs/my_videos –num_scenes 500 –pipeline_configs slurm.gin monocular.gin cuda_terrain.gin –cleanup big_files –warmup_sec 30000 –overrides compose_nature.rain_particles_chance=1.0"
6	"Description": "Generate an Infinigen command that will create a high quality arctic scene with windy weather and output a 20s video", "Command": "python -m Infinigen.datagen.manage_jobs –num_scenes 1 –pipeline_configs monocular_video.gin cuda_terrain.gin local_64GB.gin –configs arctic.gin high_quality_terrain –pipeline_overrides iterate_scene_tasks.frame_range=[1,480] –overrides compose_nature.wind_chance=1.0 –output_folder outputs/windy_arctic'"

Figure 14. Thirty samples of videos generated by the framework are presented, showcasing the system's capability to produce diverse and high-quality outputs. Each video is represented by a key frame from the rendered video.

Algorithm 1: Generate and Enhance Code with Human Feedback

Input: *user_input* (text input from the user, describing the task)
Output: *final_video* (the final rendered video after code enhancement)

1 **Step 1:** Receive the user's input

2 *user_input* ← *get_input()* `// The user provides a textual prompt, which will be used for code generation and scene creation`

3 **Step 2:** Generate initial code based on user input using the Scene Codex with RAG (Retrieval-Augmented Generation)

4 *init_code* ← *Codex*(*user_input*, *RAG*) `// Scene Codex generates code based on the user's prompt, leveraging RAG for relevant data retrieval`

5 **Step 3:** Generate the initial coarse scene from Infinigen with the initial command

6 *coarse_scene* ← *Infinigen*(*init_code*) `// Infinigen creates the coarse scene from generated initial command`

7 **Step 4:** Format the generated scene for readability and structure using LLMs (Large Language Models)

8 *formatted_scene* ← *format*(*coarse_scene*, *LLMs*) `// LLMs are used to refine and structure the generated scene for better clarity and usability`

9 **Step 5:** Store the formatted scene in the coarse-specific code repository

10 *coarse_RAG.append*(*USDA_file*(*formatted_scene*)) `// The formatted scene is converted to coarse RAG database for future reference and use`

11 **Step 6:** Edit the 3D scene based on the user input using BlenderGPT with multiple tools

12 *edited_scene* ← *BlenderGPT*(*user_input*, *Procedural Dataset*, *Shap-e*, *coarse_RAG*, *Human*) `// BlenderGPT revise and edit the 3D scene by combining the user input with LLM, procedural dataset, Shap-e, and coarse_RAG`

13 **Step 7:** Update the Infinigen scene based on the edited_scene from step 6

14 *fine_scene* ← *Infinigen*(*edited_scene*) `// Infinigen updates the scene from the edited scene and outputs a fine-detailed scene`

15 **Step 8:** Format the improved code for readability and structure using LLMs

16 *imp_formatted_code* ← *format*(*fine_scene*, *LLMs*) `// The improved scene is formatted using LLMs to ensure it is well-structured and clear`

17 **Step 9:** Store the improved code in the refined code repository for further processing

18 *refined_RAG.append*(*USDA_file*(*imp_formatted_code*)) `// The improved code is added to the refined RAG, which holds optimized code for the project`

19 **Step 10:** Render the final video using Infinigen, combining the coarse and refined code repositories with BlenderGPT and Human

20 *final_video* ← *Infinigen*(*user_input*, *BlenderGPT*, *Procedural Dataset*, *Shap-E*, *coarse_RAG*, *refined_RAG*, *Human*) `// Infinigen creates the final video by merging the coarse RAG and the refined RAG, using BlenderGPT and Procedural Dataset for 3D rendering`

21 **Step 11:** Return the final rendered video to the user

22 **return** *final_video* `// The final video, which has been generated and enhanced based on user input and feedback, is returned to the user`
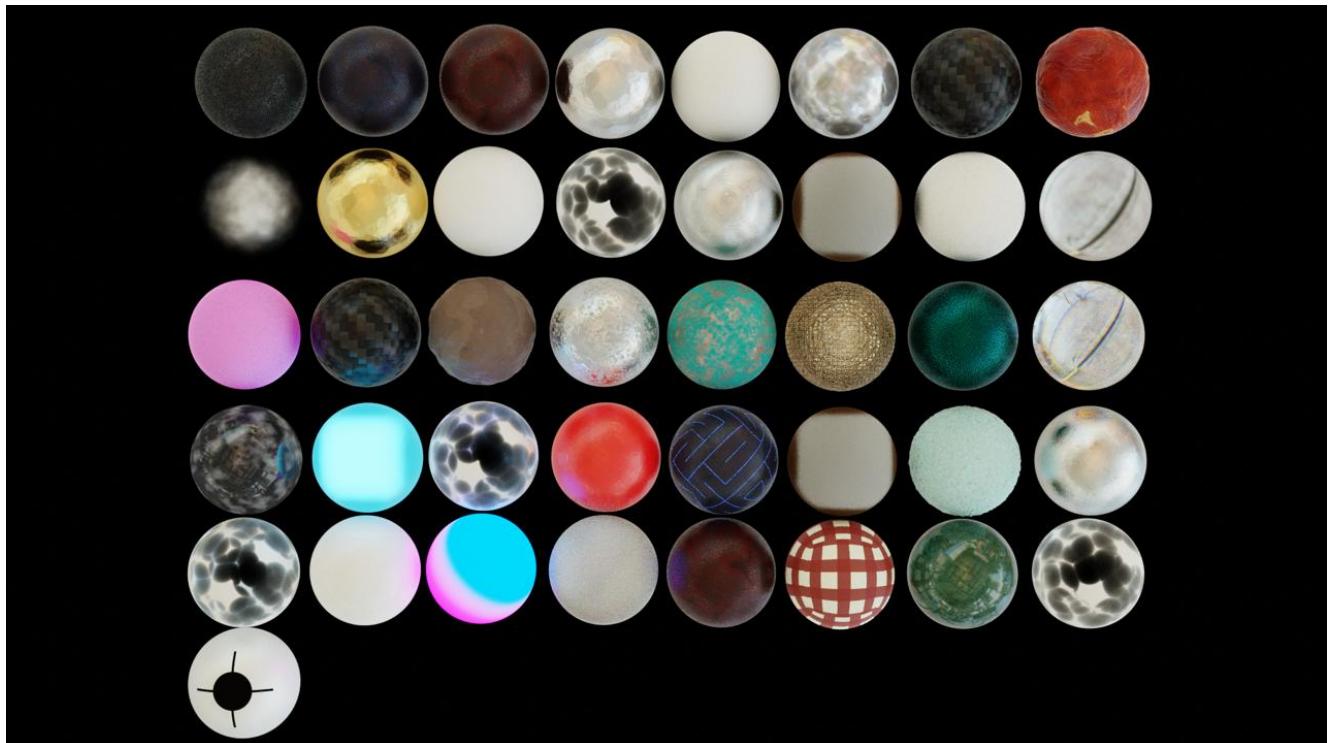
Figure 15. The dataset has been updated with 41 new materials. This addition enhances the diversity and scope of the dataset, enabling it to better accommodate a wider range of inputs and user requirements.

(a) Sugar coated candy

(b) Procedural bridge generator

(c) Apartment building

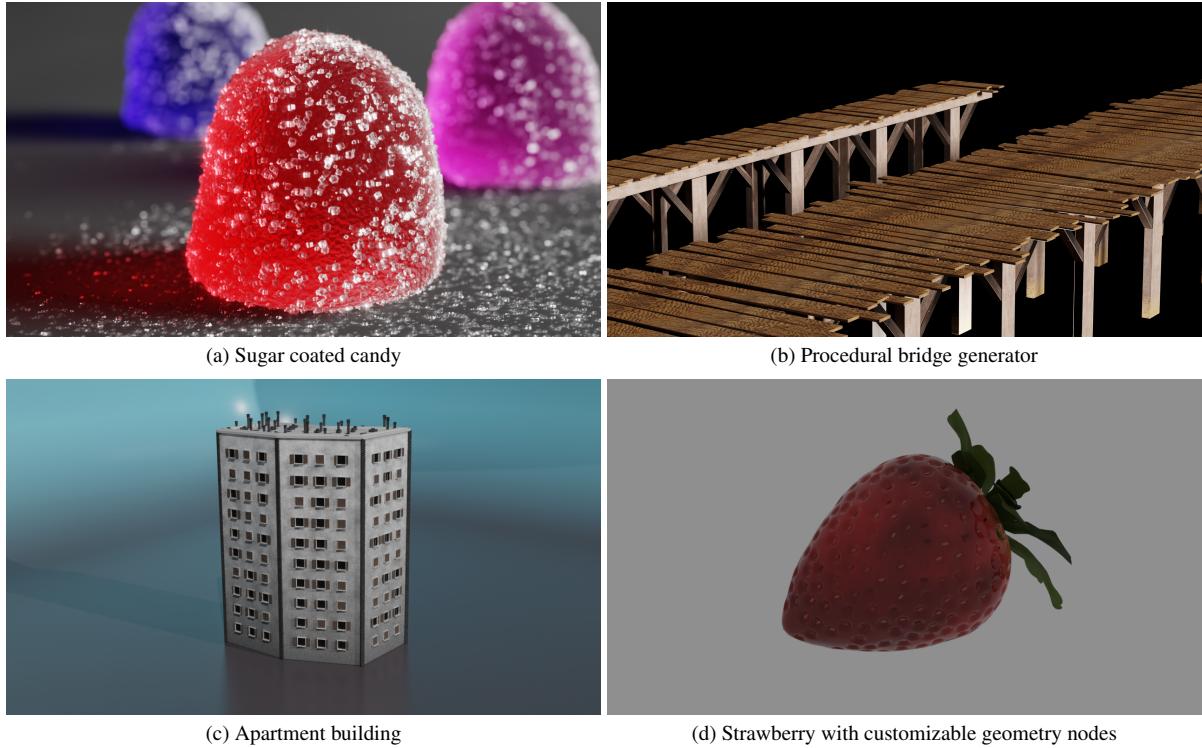
(d) Strawberry with customizable geometry nodes

Figure 16. Four of the new assets added to the dataset from the internet. Each image was rendered inside of its original blender scene. Note: (d) was used in Fig. 13c, demonstrating the expansion of the dataset's scope and consequently Scene Copilot's capabilities as new objects are integrated into the dataset.