UnrealLLM: Towards Highly Controllable and Interactable 3D Scene Generation by LLM-powered Procedural Content Generation

Song Tang[1], **Kaiyong Zhao**[2], **Lei Wang**[2], **Yuliang Li**[2]
Xuebo Liu[3], **Junyi Zou**[2], **Qiang Wang**[3]*, **and Xiaowen Chu**[1]*
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]XGRIDS
[3]Harbin Institute of Technology (Shenzhen)
stang428@connect.hkust-gz.edu.cn

Abstract

The creation of high-quality 3D scenes is essential for applications like video games and simulations, yet automating this process while retaining the benefits of Procedural Content Generation (PCG) remains challenging. In this paper, we introduce UnrealLLM, a novel multi-agent framework that connects natural language descriptions with the professional PCG system (Unreal Engine 5) to automate scene generation. UnrealLLM constructs a comprehensive knowledge base to translate text into executable PCG blueprints and a diverse asset library that guarantees high-quality scene generation. Additionally, it also introduces a text-based blueprint system with a spline-based control mechanism for geometric arrangement, enabling natural language interaction and enhancing interactivity in 3D environments using UE5's advanced capabilities. Through extensive experiments, we show that UnrealLLM achieves competitive performance in technical metrics and aesthetic quality, offering unique advantages in generation scale and interactivity. This work makes a valuable contribution to automated 3D content creation, benefiting both novice users and professional designers.

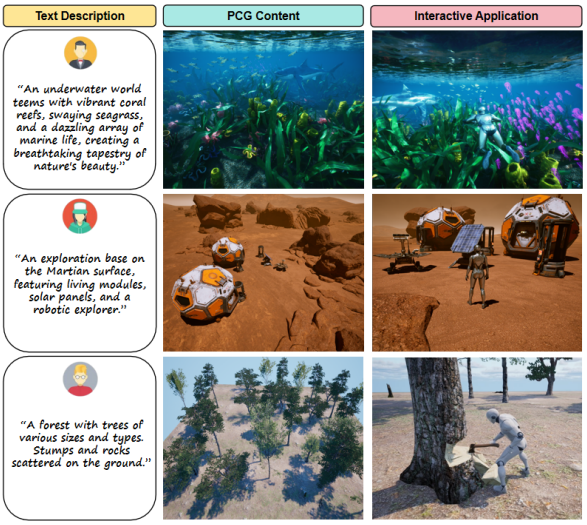

Figure 1: Scene demonstration of UnrealLLM with high controllability and interactivity.

1 Introduction

The creation of high-quality 3D scenes plays a crucial role in various applications, including video games, simulation, and visual effects production. These applications demand not only visually appealing scenes but also functionally complete 3D environments that can support interaction, simulation, and real-world deployment. While recent advances in AI-driven content generation have shown promising results, creating complex, production-ready 3D scenes still faces various challenges. To be concrete, current approaches to 3D scene generation broadly fall into two categories, explicit and implicit methods. Explicit 3D generation directly generates 3D meshes from descriptions (Chen et al., 2025; Zhang et al., 2024; Gao et al., 2022; Fu et al., 2022; Chen et al., 2019; Poole et al., 2022). Although showing promise in generating individual 3D objects, they face significant challenges in terms of quality and scalability when dealing with scenes comprising multiple assets. On the other hand, implicit scene generation methods (Wu et al., 2024; Razzhigaev et al., 2023; Cao et al., 2023; Saharia et al., 2022; Zhang et al., 2023), despite recent advances in interactivity demonstrated by approaches like GameNGen (Valevski et al., 2025) and GameGenX (Che et al., 2025), primarily focus on 2D visual rendering. While these methods can produce visually impressive results, the lack of explicit 3D representations renders the objects indecomposable, leading to reduced flexibility and applicability.

Given these limitations in current approaches, Procedural Content Generation (PCG) offers a promising solution by leveraging rule-based spatial assembly mechanisms and existing high-quality

* Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2025, pages 19417–19435

assets. These systems excel at creating complex, realistic environments through sophisticated procedural rules, from natural landscapes with realistic ecological patterns to structured urban environments with precise spatial arrangements. However, PCG requires a deep understanding of generation rules and professional software knowledge, making it challenging for beginners and time-consuming to use. To tackle the aforementioned challenges, recent works like 3D-GPT (Sun et al., 2023) and SceneX (Zhou et al., 2024) have proposed an instruction-driven approach to 3D scene generation by combining a large language model (LLM) agent with procedural generation tools in Blender. While current methods have effectively collaborated with human designers to establish procedural generation frameworks, they encounter some limitations. Firstly, they depend on Infinigen (Raistrick et al., 2023), which offers a restricted variety of scene types and limited styles of 3D resources. Additionally, their approach of using LLMs to primarily orchestrate pre-defined Infinigen APIs places inherent limitations on creative potential.

In contrast, Unreal Engine 5 (UE5)[1] offers a powerful and flexible ecosystem for procedural content generation (PCG), enabling users to utilize a diverse range of professional assets from the Marketplace and leverage advanced technologies like Nanite, Lumen, and Chaos for high-quality visuals. Moreover, UE5's native PCG system facilitates direct access to node-level scene composition, presenting opportunities for more dynamic and innovative content creation. However, how LLMs can interface with node-level scene generation in UE5 also raises challenges. First, constructing the UE5-specific blueprint for scene generation requires extensive expert knowledge, needing a semantic translation mechanism between natural language and node-level PCG workflows. Second, the fine granularity and high diversity of assets in UE5 present efficiency challenges for context-aware retrieval across various artistic styles. Finally, ensuring stability and quality in generation demands the systematic integration of expert knowledge.

The preceding analysis motivates our design of a UE5-native generation framework, named UnrealLLM, which redefines the interaction between large language models (LLMs) and procedural content generation (PCG) processes. This novel multi-agent framework bridges the gap between natural language interaction and PCG-based scene generation. As demonstrated in Figure 1, our framework not only facilitates text-to-scene generation but also fully interactive environments that support complex gameplay mechanics like character walking, swimming, and object destruction, leveraging UE5's real-time physics and gameplay systems. A key innovation is the development of a text-based blueprint representation that allows AI agents to systematically process and manipulate PCG graphs, capturing both structural composition and parametric configurations in a machine-processable and human-interpretable format. To enhance asset utilization, we construct a rich 3D asset library with multi-modal representations for context-aware retrieval, and we develop a comprehensive PCG knowledge base encompassing node operations, parameter configurations, and proven generation strategies to ensure reliable, high-quality scene generation. Additionally, we introduce a spline-based control enhancement that provides precise spatial control over the generation process, enabling users to achieve specific aesthetic goals while maintaining procedural efficiency. Our main contributions can be summarized as follows:

- **Reliability**: A specialized multi-agent framework where agents collaborate to handle node-level PCG tasks, supported by a comprehensive knowledge base of PCG expertise to ensure generation reliability and quality.
- **Diversity**: A multi-modal asset library and context-aware retrieval system that empowers our framework to generate virtually any type of 3D scene while maintaining consistent artistic styles and high visual quality.
- **Interactivity**: A novel text-based blueprint representation that enables seamless interaction between natural language and professional PCG operations, while supporting rich gameplay interactivity in generated scenes through UE5's physics and gameplay systems.

2 Related Works

Procedural 3D Content Generation. Procedural Content Generation (PCG) has emerged as a powerful paradigm for automated 3D content creation, with applications spanning both natural and urban environments. Early works established strong foundations in natural scene generation (Zhang et al., 2019; Gasch et al., 2022), while

[1] https://www.unrealengine.com

parallel efforts developed sophisticated urban planning frameworks (Talton et al., 2011; Lipp et al., 2011; Vanegas et al., 2012; Yang et al., 2013). These approaches typically rely on carefully crafted mathematical rules and parameters to guide the generation process. Procedural modeling of cities (Parish and Müller, 2001) pioneered city generation through boundary-aware procedural rules, effectively handling street layouts and building arrangements via L-system-based algorithms that incorporate environmental constraints and procedural generation patterns. Recent advances like Infinigen (Raistrick et al., 2023) have pushed the boundaries of procedural generation, enabling the creation of diverse, realistic 3D natural objects and scenes through sophisticated algorithmic techniques.

However, these systems generally require extensive technical expertise to operate effectively, limiting their accessibility to non-expert users. Our work addresses this challenge by introducing a natural language interface to PCG systems while maintaining their powerful generation capabilities.

Text-to-3D Scene Generation Recent advances in text-to-image synthesis have catalyzed significant developments in text-guided 3D content creation. Early approaches explored vision-language models for generation guidance (Jain et al., 2022), followed by the development of Score Distillation Sampling techniques that enabled the adaptation of 2D generative models for 3D content synthesis (Poole et al., 2022; Wang et al., 2023a). The field has witnessed diverse explorations in representation formats, ranging from neural implicit representations (Metzer et al., 2023; Mildenhall et al., 2021) to optimized mesh-based approaches (Lin et al., 2023; Chen et al., 2023). Researchers have addressed generation consistency through various strategies, including multi-view synthesis methods (Shi et al., 2023; Ye et al., 2024) and advanced distillation techniques (Wang et al., 2023b). Parallel efforts have focused on single-view reconstruction (Huang et al., 2023; Liu et al., 2023) and specialized applications in shape synthesis (Yu et al., 2024b; Erkoç et al., 2023) and domain-specific generation (Sarafianos et al., 2025). However, existing approaches face challenges in creating production-quality 3D scenes, particularly in terms of structural organization and asset quality. Our work addresses these limitations by integrating natural language understanding with established PCG systems and high-fidelity asset libraries.

3 Method

3.1 Rule-based Procedural Generation

Procedural content generation (PCG) systems offer a powerful approach to 3D content creation through their sophisticated rule-based spatial assembly mechanisms. As illustrated in Figure 2, these systems fundamentally operate through sequential steps: target space sampling, 3D pose transformation, composition, filtering, and asset selection. The core workflow involves iterative interactions between these operations, which collectively define the spatial rules for scene generation. To enable AI agents to effectively understand and manipulate these complex PCG workflows, we need a systematic way to represent them in a machine-processable format.

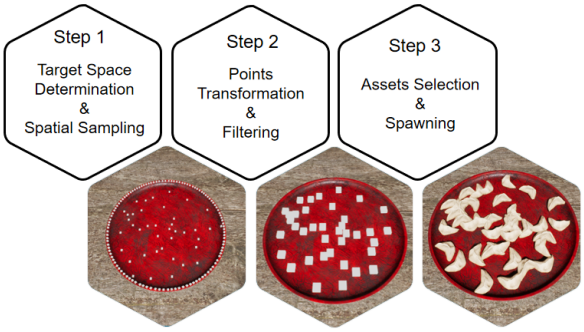

Figure 2: A simple workflow demonstration of PCG (Procedural Content Generation). This image showcases how a realistic plate of dumplings can be generated through simple steps.

3.2 Text-based PCG Blueprint Representation

The effective automation of PCG systems requires a standardized and efficient representation of procedural generation logic. While traditional node-based PCG systems rely on visual programming interfaces, we propose a text-based blueprint representation that enables systematic processing and manipulation by AI agents. This representation captures the structural composition and parametric configurations of PCG graphs in a format that is both machine-processable and human-interpretable, and critically, enables generation and manipulation by Large Language Models (LLMs).

Our blueprint representation maintains a hierarchical structure mirroring PCG graphs. The key innovation is a structured, JSON-based Domain-Specific Language (DSL). This DSL bridges natural language to technical PCG configurations by precisely defining core Unreal Engine Blueprint

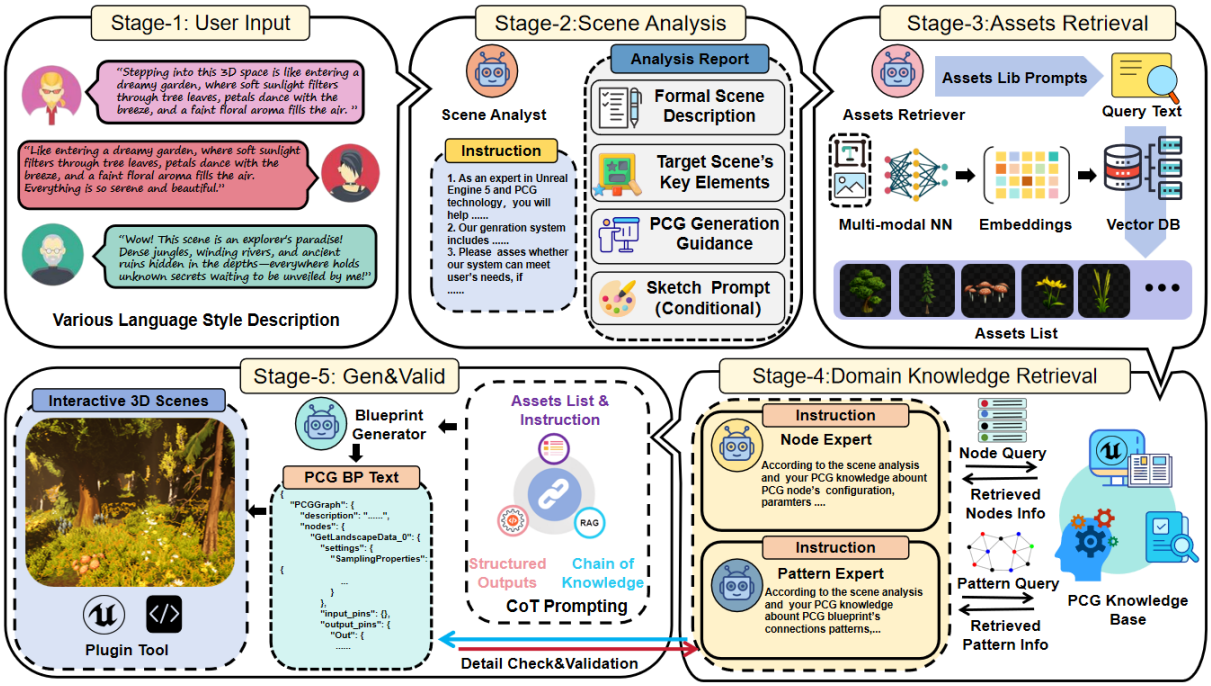

Figure 3: Overview of our multi-agent PCG automation framework. The system comprises five specialized agents working in coordination: the Scene Analysis agent processes natural language inputs into structured specifications, the Asset Manager handles 3D content retrieval through cross-modal matching, while two Expert agents (Node Expert and Pattern Expert) provide technical guidance through retrieval-augmented generation. The Blueprint Generator integrates all inputs to construct the final PCG graph, which is then converted to executable Unreal Engine blueprints.

elements (e.g., nodes, connections, parameters) and allows for bidirectional conversion with UE's native Blueprint format. This design facilitates efficient processing and automated generation of blueprints by AI agents, supporting template-based approaches and enabling LLMs to effectively manipulate PCG logic.

3.3 Multi-agent Framework for PCG Automation

Based on our text-based blueprint representation, we develop a multi-agent framework to automate the PCG graph construction process. The multi-agent framework decomposes complex PCG expertise into specialized domains, enabling reliable scene generation through collaborative problem-solving. Our framework orchestrates five agents in a coordinated pipeline to transform natural language descriptions into executable PCG graphs. They are Scene Analyst, Asset Manager, Node Expert, Pattern Expert and Blueprint Generator.

Scene Analyst At the entry point, the Scene Analyst processes user inputs into technically precise specifications, addressing the challenge of varying user expression styles by producing standardized scene descriptions, structured element lists, and strategic generation guidance. This standardized output ensures consistent interpretation of scene requirements while preserving creative intent.

Asset Manager The Asset Manager operates on a comprehensive 3D asset library that we constructed from MegaScan's free assets and Objaverse (Deitke et al., 2023) dataset. For each asset in our library, we first establish a rich multi-modal representation through automated annotation and encoding. The annotation process leverages a multi-modal large language model to generate detailed descriptions:

$$T(A) = \text{LLM}(V(A), M(A)) \tag{1}$$

where $V(A) = \{v_1, \ldots, v_n\}$ represents a set of rendered views from different angles, and $M(A)$ contains basic metadata including technical specifications, material properties, and usage contexts. These multi-modal features are encoded into visual E_v and textual E_t embedding spaces:

$$[E_\text{v}(A), E_\text{t}(A)] = [\text{CLIP}(V(A)), \text{CLIP}(T(A))] \tag{2}$$

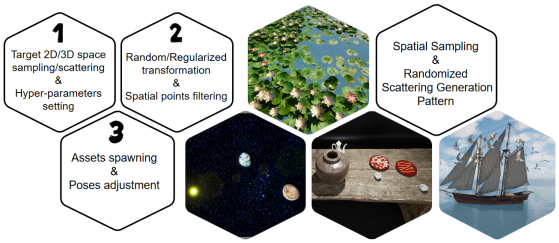

Figure 4: Demonstration of our multi-agent system's generation pattern based on spatial sampling and random distribution.

The Asset Manager follows a sophisticated workflow to ensure optimal asset selection. Initially, through prompt engineering, the agent develops a comprehensive understanding of the asset library's scope and categories. When receiving input from the Scene Analysis agent, including scene descriptions and element requirements, it leverages this knowledge to formulate effective natural language queries. For a given query q, the retrieval score is computed as:

$$S(A, q) = \lambda_{\text{t}} \cos(E_q, E_{\text{t}}(A)) + \lambda_{\text{v}} \cos(E_q, E_{\text{v}}(A)) \quad (3)$$

where λ_{t} and λ_{v} are weights balancing the contribution of visual and text modality. For each query, the vector database returns top-k most similar assets. The agent then evaluates these candidates against the original scene description, considering both semantic appropriateness and technical constraints, to select the most suitable assets. Finally, it organizes the selected assets into a structured text for other agents in the scene generation process.

Node Expert and Pattern Expert Our framework incorporates two specialized expert agents that leverage comprehensive PCG knowledge bases. These knowledge bases, carefully curated by experienced PCG specialists and UE experts, contain proven generation strategies and technical specifications. The expert agents employ retrieval-augmented generation (RAG) (Lewis et al., 2020) to bridge the gap between natural language scene descriptions and domain-specific PCG knowledge, ensuring reliable generation guidance.

The Node Expert agent maintains a comprehensive knowledge base of PCG nodes and their applications, documenting node behaviors, parameter spaces, and connection patterns. This knowledge is encoded into a dense vector space using specialized technical embeddings:

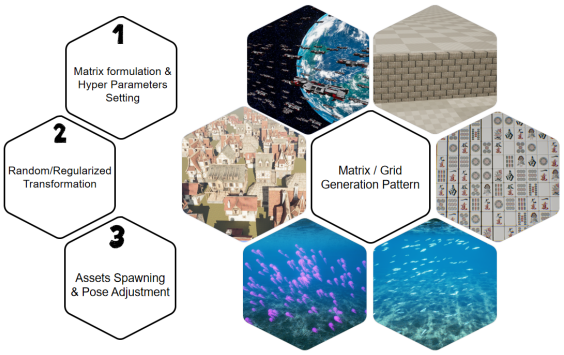

Figure 5: Demonstration of our multi-agent system's generation patterns based on matrices and regularized spatial grids.

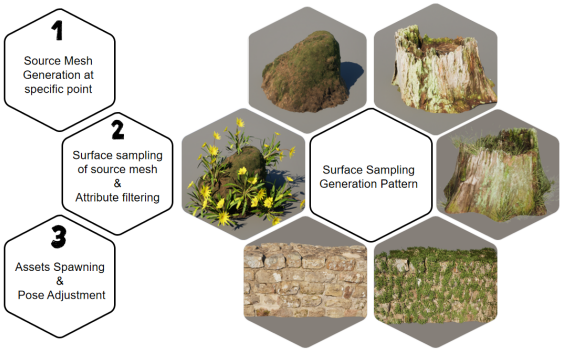

Figure 6: Demonstration of our multi-agent system's generation patterns based on mesh surface sampling.

$$K_{\text{node}}(n) = \text{Encode}(D(n), P(n), C(n)) \quad (4)$$

Each PCG node n's knowledge representation $K_{\text{node}}(n)$ is computed by encoding three key components: $D(n)$ representing the node's technical documentation and usage guidelines, $P(n)$ containing valid parameter ranges and recommended configurations, and $C(n)$ describing possible connection patterns with other nodes. When processing scene requirements, the agent employs semantic retrieval to identify relevant node configurations:

$$R(q) = \text{TopK}(\text{Sim}(E(q), K_{\text{node}}), k) \quad (5)$$

For a given query q derived from scene requirements, this equation retrieves the top k most relevant node configurations by computing similarity scores between the query embedding $E(q)$ and the encoded node knowledge K_{node}.

Working in parallel, the Pattern Expert agent manages a library of proven PCG strategies and architectural templates, focusing on higher-level generation patterns that have demonstrated effectiveness in various scene types. Through careful curation of successful PCG implementations, we have

Figure 7: Demonstration of spline-based scene generation control through a medieval town example. The system utilizes boundary splines for forest and town areas, and a path spline for the dirt road. These splines, generated by SketchAgent with semantic labels, serve as geometric controls for PCG blueprint generation.

established a comprehensive pattern library that enables diverse generation capabilities. As shown in Figure 4, our system supports spatial sampling with randomized distribution, ideal for natural scattered elements like lotus on water or birds in sky. Figure 5 demonstrates our matrix-based generation patterns, enabling precise structural arrangements such as organized space fleets and gridded town layouts. For complex environmental detailing, Figure 6 showcases our surface-based sampling patterns, allowing sophisticated combinations like flowers on terrain or vines on walls. The Pattern Expert leverages these established patterns through a structured retrieval and composition process, where the pattern selection score $S_{\text{pattern}}(q)$ for a given query q is computed as:

$$S_{\text{pattern}}(q) = \text{Retrieve}(q, L_{\text{pattern}}) \cdot \text{Compose}(G_{\text{scene}}) \quad (6)$$

where L_{pattern} is the expert-curated pattern library, and G_{scene} represents the scene generation goals derived from the Scene Analysis agent's output. The Retrieve function identifies relevant patterns from the library, while Compose combines these patterns according to the specific scene requirements. Based on this score, the Pattern Expert selects the most suitable generation patterns and determines how to combine them for optimal scene composition. The collaborative interaction between these expert agents ensures both technical correctness and generation intent preservation. The Node Expert validates technical specifications while the Pattern Expert guides the overall generation strategy.

Blueprint Generator Given the combined expertise flows from both the Node Expert and the Pattern Expert, the Blueprint Generator integrates their recommendations to construct the final PCG graph. The Blueprint Generator leverages advanced large language models (LLMs) like GPT-4o and Claude 3.5, known for their enhanced capabilities in handling complex information streams and generating structured outputs, to effectively process multiple information streams from preceding agents: the structured asset catalog from Asset Manager, technical specifications from Node Expert, and generation strategies from Pattern Expert. During the construction process, it maintains a feedback loop with Expert agents for iterative refinement. The final output is encoded in our text-based representation format and converted to executable Unreal Engine PCG blueprints through our custom plugin.

3.4 Spline-based Control Enhancement

While our multi-agent framework enables diverse scene generation through PCG rules, achieving precise spatial control requires additional guidance mechanisms. We address this by incorporating spline-based control into our system, leveraging the SketchAgent approach (Vinker et al., 2024) to automatically generate control splines from natural language descriptions. As illustrated in Figure 7, this approach enables intuitive geometric control over complex scene layouts through various spline primitives: distribution paths for asset placement, terrain feature shaping, density variation control, and environmental zone boundaries. For example, our medieval town scene demonstrates the power

Figure 8: Comparison of scene generation capabilities. From left to right: SceneX's photorealistic results and our method's results in photorealistic, stylized, and low-poly styles. Each row shows a different scene type (forest, desert, town). Our approach demonstrates enhanced diversity by supporting multiple artistic styles.

Instruction	Node KB	Pattern KB	Example	ER@1	SR@1
✓				19.5	12.2
	✓			39.0	36.6
✓	✓			56.1	49.0
✓	✓	✓		63.4	53.6
✓	✓	✓	✓	**75.6**	**73.2**

Table 1: Ablation study of different prompt components for 3D scene generation.

Method	ER@1	SR@1
gpt-3.5(Brown et al., 2020)	58.6	41
gpt-4o(Achiam et al., 2023)	68.2	63.4
o1-mini(Jaech et al., 2024)	83.0	78
o1(Jaech et al., 2024)	75.6	73.2
Claude3.5 Sonnet(Anthropic, 2024)	**80.1**	**70.8**

Table 2: Comparison of different models' ER and SR as the Blueprint Generator

of this approach through the orchestration of multiple spline controls: boundary splines define forest and town areas, while path splines determine road trajectories, collectively creating a cohesive environment. The system represents splines in a hierarchical format that captures both geometric and semantic information, allowing users to fine-tune generated scenes through control point adjustments while maintaining the intended scene structure.

4 Experiments

Metrics. We use multiple metrics to evaluate the performance of our system. For technical assessment, we adopt Executability Rate (ER@1) and Success Rate (SR@1), where ER@1 measures the proportion of proposed actions that can be executed in the PCG system, and SR@1 evaluates the correctness of these actions (Chen et al., 2021). To quantify aesthetic quality, we employ the GPT Aesthetic Score (GAS), extending the aesthetic evaluation methodology from SceneX (Zhou et al., 2024). While SceneX relies on human assessors for aesthetic scoring, we leverage large language models to provide consistent evaluation across different methods. Additionally, we employ CLIP (Radford et al., 2021) similarity to evaluate the semantic alignment between generated scenes and input text descriptions. Following established practices (Lin et al., 2023; Zhou et al., 2024), the CLIP similarity is computed as the cosine similarity between CLIP embeddings of rendered scene views and the input text descriptions, providing a quantitative measure of how well the generated content matches the intended design.

4.1 Experimental Setup

Dataset. To evaluate our system, we curated a test set of over 40 diverse scene generation scenarios, covering natural landscapes (forests, mountains, coastlines), towns (buildings, villages), and fantasy scenes (magical forests, alien landscapes). Each

Method	GAS↑
DreamFusion (Poole et al., 2022)	4.83
Magic 3D (Lin et al., 2023)	6.39
WonderJ (Yu et al., 2024a)	7.38
Infinigen (Raistrick et al., 2023)	6.61
3D-GPT (Sun et al., 2023)	6.76
SceneX (Zhou et al., 2024)	7.31
Ours	**7.71**

Table 3: Comparative analysis of GPT Aesthetic Scores.

Method	ViT-L14	ViT-B/16↑	ViT-B/32↑
Magic 3D (Lin et al., 2023)	27.86	31.78	**31.94**
DreamFusion (Poole et al., 2022)	**29.40**	**35.37**	31.60
Text2Room (Höllein et al., 2023)	23.51	30.10	29.29
WonderJ (Yu et al., 2024a)	18.78	25.70	25.45
BlenderGPT	21.23	25.65	26.19
3D-GPT (Sun et al., 2023)	18.67	25.80	25.59
SceneCraft (Hu et al., 2024)	22.04	25.82	25.30
SceneX (Zhou et al., 2024)	22.82	27.82	26.89
Ours	**26.34**	**32.49**	**31.41**

Table 4: Comparative analysis of text-image similarity predicted by different CLIP models.

scenario is designed to test different aspects of our system's generation capabilities.

4.2 Results and Analysis

Ablation Studies. We conduct ablation studies to evaluate our prompt template's different components. As shown in Table 1, each component contributes positively to the system's performance. The components include: Instruction (basic task description and requirements), Node KB (technical knowledge about PCG nodes and operations), Pattern KB (proven generation strategies and patterns), and Example (reference scenes with similar generation logic). Using instruction alone yields modest results (ER@1: 19.5%, SR@1: 12.2%), while incorporating Node KB significantly improves performance (ER@1: 39.0%, SR@1: 36.6%). The full system with all components achieves the best results (ER@1: 75.6%, SR@1: 73.2%).

Model Comparison. We evaluate different language models as the Blueprint Generator, as shown in Table 2. While GPT-3.5 shows moderate performance (ER@1: 58.6%, SR@1: 41.0%), GPT-4o achieves better results (ER@1: 68.2%, SR@1: 63.4%). More recent models demonstrate even stronger capabilities, with o1-mini achieving the best overall performance (ER@1: 83.0%, SR@1: 78.0%), followed by Claude 3.5-Sonnet (ER@1: 80.1%, SR@1: 70.8%) and o1 (ER@1: 75.6%, SR@1: 73.2%).

Generation Diversity. As shown in Figure 8, our approach demonstrates superior flexibility in scene generation compared to existing methods. While SceneX can only generate photorealistic scenes due to its reliance on Infinigen's fixed asset pipeline, our framework supports multiple artistic styles including photorealistic, stylized, low-poly, and other styles. This enhanced diversity is enabled by our comprehensive asset library and flexible PCG blueprint system, allowing for versatile scene creation across different environment types.

	Magic3D	DreamFusion	InfiniGen	SceneX	3D-GPT	**Ours**
Text Guidance	✓	✓	×	✓	✓	✓
Parameter Control	×	×	✓	✓	✓	✓
Spline Control	×	×	✓	×	×	✓
Game Interaction	×	×	×	×	×	✓
Generation Scale	limited	limited	unlimited	unlimited	unlimited	unlimited

Table 5: Feature comparison of different methods (✓: supported, ×: unsupported).

Feature Comparison. We further analyze different methods' capabilities in Table 5. While most methods support text guidance, our approach uniquely combines parameter control, spline-based control, and game interaction capabilities. This comprehensive feature set, coupled with unlimited generation scale, enables more flexible and practical scene creation. Notably, we are the first to support direct game interaction with generated scenes, allowing for dynamic gameplay mechanics like character movement and object interaction.

Aesthetic Quality and Comparative Analysis. We compare our method with state-of-the-art approaches using both aesthetic and technical metrics, as shown in Tables 3 and 4. Our method achieves a GAS of 7.71, surpassing previous methods including SceneX (7.31), 3D-GPT (6.76), and Infinigen (6.61). While methods like Magic3D and DreamFusion show higher CLIP similarity scores across different ViT models, this is primarily because these methods incorporate text-to-image alignment in their training or optimization process. In contrast, our approach focuses on generating functionally complete 3D scenes with diverse artistic styles and interactive capabilities, while maintaining competitive semantic alignment with input descriptions. This comprehensive capability set, coupled with strong aesthetic scores, demonstrates our system's effectiveness in creating both visually appealing and functionally complete 3D scenes.

5 Conclusion

In this paper, we present a novel multi-agent framework that bridges natural language interaction with professional PCG systems in Unreal Engine 5. Our approach combines large language models with pro-

cedural generation techniques through text-based blueprint representation and comprehensive knowledge bases, enabling intuitive control over scene generation. Experimental results demonstrate our system's effectiveness in technical reliability, aesthetic quality, and gameplay interactivity. We believe our work represents a significant step toward democratizing professional PCG systems, benefiting both novice users and professional designers in automated 3D content creation.

Limitations

While our framework demonstrates promising results, there are several limitations worth noting. First, the system's performance heavily relies on the quality and coverage of our PCG knowledge base. For novel or unconventional scene types not well-represented in our knowledge base, the generation quality may be compromised. Second, although our multi-modal asset library supports diverse scene generation, the system is still bounded by the available assets. Creating scenes with highly specific or unique artistic requirements might be limited by the asset collection. Additionally, while our framework supports basic gameplay interactions, implementing complex, multi-step interaction patterns or specialized gameplay mechanics may require manual adjustments to the generated scenes. Finally, the multi-agent architecture and comprehensive knowledge retrieval process can be computationally intensive, potentially affecting real-time performance in resource-constrained environments.

Ethics Statement

Our research aims to democratize professional 3D content creation while maintaining high-quality standards. We recognize the computational intensity of our system and its potential environmental impact, and future work should focus on optimizing resource usage and reducing energy consumption. While our system makes PCG more accessible, we ensure transparency about its capabilities and limitations to prevent misuse or unrealistic expectations. Regarding content usage, our asset library is built using a combination of properly licensed content from MegaScan's free assets, the Objaverse dataset, and commercially licensed assets from the Unreal Engine Marketplace, with all assets being used in accordance with their respective license agreements and attribution requirements. Furthermore, we actively work to ensure our knowledge base and asset library represent diverse artistic styles and cultural perspectives, avoiding potential biases in scene generation. Through these considerations, we strive to develop a system that not only advances technical capabilities but also adheres to ethical principles in AI-driven content creation.

Acknowledgements

This work was partially supported by National Natural Science Foundation of China under Grant No. 62272122, the Guangzhou Municipal Joint Funding Project with Universities and Enterprises under Grant No. 2024A03J0616, Guangzhou Municipality Big Data Intelligence Key Lab (2023A03J0012), and Hong Kong CRF grants under Grant No. C7004-22G and C6015-23G.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv:2303.08774*.

AI Anthropic. 2024. Claude 3.5 sonnet model card addendum. *Claude-3.5 Model Card*, 3(6).

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Proc. of NIPS*, pages 1877–1901.

Tingfeng Cao, Chengyu Wang, Bingyan Liu, Ziheng Wu, Jinhui Zhu, and Jun Huang. 2023. BeautifulPrompt: Towards automatic prompt engineering for text-to-image synthesis. In *Proc. of EMNLP*, pages 1–11.

Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. 2025. Gamegen-x: Interactive open-world game video generation. In *Proc. of ICLR*.

Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. 2019. Text2shape: Generating shapes from natural language by learning joint embeddings. In *Proc. of ACCV*, pages 100–116.

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. 2021. Evaluating large language models trained on code. *arXiv:2107.03374*.

Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. 2023. Fantasia3d: Disentangling geometry and appearance for high-quality text-to-3d content creation. *arXiv:2303.13873*.

Zhaoxi Chen, Jiaxiang Tang, Yuhao Dong, Ziang Cao, Fangzhou Hong, Yushi Lan, Tengfei Wang, Haozhe Xie, Tong Wu, Shunsuke Saito, et al. 2025. 3dtopia-xl: Scaling high-quality 3d asset generation via primitive diffusion. *arXiv:2409.12957*.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. 2023. Objaverse: A universe of annotated 3d objects. In *Proc. of CVPR*, pages 13142–13153.

Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. 2023. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proc. of ICCV*, pages 14300–14310.

Rao Fu, Xiao Zhan, Yiwen Chen, Daniel Ritchie, and Srinath Sridhar. 2022. Shapecrafter: A recursive text-conditioned 3d shape generation model. In *Proc. of NIPS*, pages 8882–8895.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. 2022. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Proc. of NIPS*, pages 31841–31854.

Cristina Gasch, José Sotoca, Miguel Chover, Inmaculada Remolar, and Cristina Rebollo. 2022. Procedural modeling of plant ecosystems maximizing vegetation cover. *Multimedia Tools and Applications*, 81:16195–16217.

Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. 2023. Text2room: Extracting textured 3d meshes from 2d text-to-image models. In *Proc. of ICCV*, pages 7909–7920.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. 2024. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *Proc. of ICML*, pages 19252–19282.

Yangyi Huang, Hongwei Yi, Yuliang Xiu, Tingting Liao, Jiaxiang Tang, Deng Cai, and Justus Thies. 2023. Tech: Text-guided reconstruction of lifelike clothed humans. In *Proc. of 3DV*, pages 1531–1542.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. 2022. Zero-shot text-guided object generation with dream fields. In *Proc. of CVPR*, pages 867–876.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proc. of NIPS*, pages 9459–9474.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2023. Magic3d: High-resolution text-to-3d content creation. In *Proc. of CVPR*, pages 300–309.

Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. 2011. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum*, 30:345–354.

Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Zexiang Xu, Hao Su, et al. 2023. One-2-3-45: Any single image to 3d mesh in 45 seconds without per-shape optimization. In *Proc. of NIPS*, pages 22226–22246.

Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. 2023. Latent-nerf for shape-guided generation of 3d shapes and textures. In *Proc. of CVPR*, pages 12663–12673.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65:99–106.

Yoav Parish and Pascal Müller. 2001. Procedural modeling of cities. In *Proc. of SIGGRAPH*, volume 2001, pages 301–308.

Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. 2022. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv:2209.14988*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *Proc. of ICML*, pages 8748–8763.

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. 2023. Infinite photorealistic worlds using procedural generation. In *Proc. of CVPR*, pages 12630–12641.

Anton Razzhigaev, Arseniy Shakhmatov, Anastasia Maltseva, Vladimir Arkhipkin, Igor Pavlov, Ilya Ryabov, Angelina Kuts, Alexander Panchenko, Andrey Kuznetsov, and Denis Dimitrov. 2023. Kandinsky: An improved text-to-image synthesis with image prior and latent diffusion. In *Proc. of EMNLP*, pages 286–295.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. In *Proc. of NIPS*, pages 36479–36494.

Nikolaos Sarafianos, Tuur Stuyck, Xiaoyu Xiang, Yilei Li, Jovan Popovic, and Rakesh Ranjan. 2025. Garment3dgen: 3d garment stylization and texture generation. *arXiv:2403.18816*.

Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. 2023. Mvdream: Multi-view diffusion for 3d generation. In *Proc. of ICLR*.

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

Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Mech, and Vladlen Koltun. 2011. Metropolis procedural modeling. *ACM Transactions on Graphics*, 30:1–14.

Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. 2025. Diffusion models are real-time game engines. In *Proc. of ICLR*.

Carlos Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel Aliaga, and Pascal Müller. 2012. Procedural generation of parcels in urban modeling. *Computer Graphics Forum*, 31:681–690.

Yael Vinker, Tamar Rott Shaham, Kristine Zheng, Alex Zhao, Judith E Fan, and Antonio Torralba. 2024. Sketchagent: Language-driven sequential sketch generation. *arXiv preprint arXiv:2411.17673*.

Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A Yeh, and Greg Shakhnarovich. 2023a. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation. In *Proc. of CVPR*, pages 12619–12629.

Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. 2023b. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. In *Proc. of NIPS*, pages 8406–8441.

Zongyu Wu, Hongcheng Gao, Yueze Wang, Xiang Zhang, and Suhang Wang. 2024. Universal prompt optimizer for safe text-to-image generation. In *Proc. of NAACL*, pages 6340–6354.

Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. 2013. Urban pattern: layout design by hierarchical domain splitting. *ACM Transactions on Graphics*, 32:1–12.

Jianglong Ye, Peng Wang, Kejie Li, Yichun Shi, and Heng Wang. 2024. Consistent-1-to-3: Consistent image to 3d view synthesis via geometry-aware diffusion models. In *Proc. of 3DV*, pages 664–674.

Hong-Xing Yu, Haoyi Duan, Junhwa Hur, Kyle Sargent, Michael Rubinstein, William T Freeman, Forrester Cole, Deqing Sun, Noah Snavely, Jiajun Wu, et al. 2024a. Wonderjourney: Going from anywhere to everywhere. In *Proc. of CVPR*, pages 6658–6667.

Zhengming Yu, Zhiyang Dou, Xiaoxiao Long, Cheng Lin, Zekun Li, Yuan Liu, Norman Müller, Taku Komura, Marc Habermann, Christian Theobalt, et al. 2024b. Surf-d: High-quality surface generation for arbitrary topologies using diffusion models. In *Proc. of ECCV*, pages 419–438.

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

Longwen Zhang, Ziyu Wang, Qixuan Zhang, Qiwei Qiu, Anqi Pang, Haoran Jiang, Wei Yang, Lan Xu, and Jingyi Yu. 2024. Clay: A controllable large-scale generative model for creating high-quality 3d assets. *ACM Transactions on Graphics*, 43:1–20.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *Proc. of ICCV*, pages 3836–3847.

Mengqi Zhou, Yuxi Wang, Jun Hou, Chuanchen Luo, Zhaoxiang Zhang, and Junran Peng. 2024. Scenex: Procedural controllable large-scale scene generation via large-language models. In *Proc. of AAAI*, pages 10806–10814.

A PCG Fundamentals

To better understand our system's foundation, we introduce the basic principles and workflow of Procedural Content Generation (PCG). PCG systems operate through a series of well-defined steps that transform simple primitives into complex, organized content. At its core, PCG in Unreal Engine is implemented as a node-based system, as demonstrated in Figure 9. Each node in the system represents a specific operation or rule, with parameters that can be fine-tuned to control the generation process. The workflow proceeds from left to right, with nodes processing and passing data to subsequent nodes, allowing for intuitive creation and modification of generation logic. This visual programming interface enables developers to construct sophisticated generation systems while maintaining precise control over each step of the process.

The power of PCG systems lies in their ability to handle content generation at various scales of complexity. As illustrated in Figure 10, PCG can generate content ranging from simple individual elements to complex, hierarchically structured scenes. The figure demonstrates this scalability through a progression from a single tree to a cluster of trees, and finally to a structured forest with multiple layers of organization. As the target scene's complexity increases, the generation logic can be accordingly expanded and refined. This flexibility allows developers to construct appropriate generation rules based on the specific requirements of their target scenes, whether simple or complex.

To provide a concrete understanding of the PCG workflow, Figure 11 breaks down the generation process using a simple yet illustrative example of creating a plate of dumplings. This step-by-step demonstration reveals the typical components of a PCG pipeline: initial sampling and projection of the target space to establish the basic layout, 3D pose transformation to create natural variation, composition rules to ensure proper arrangement, filtering to maintain realistic constraints, and finally, asset selection and generation to produce the final result. Each step builds upon the previous ones, demonstrating how simple geometric operations can progressively create realistic and visually appealing arrangements.

Through these examples, we can see that PCG systems provide a powerful and flexible framework for automated content generation. The combination of node-based visual programming, scalable complexity handling, and well-defined generation steps enables the creation of diverse and sophisticated 3D content while maintaining precise control over the generation process. This foundation is crucial for understanding how our system leverages PCG capabilities to transform natural language descriptions into interactive 3D environments.

B Evaluation Process

Our evaluation process consists of two main components: GPT Aesthetic Score (GAS) assessment and CLIP similarity computation. For GAS evaluation, we employ a carefully designed prompt template to ensure consistent aesthetic assessment:

```
You are an art critic evaluating the
aesthetic quality of a rendered image.
Please analyze the image and provide
a score from 1 to 10, where:

1-2 (Poor): Serious issues with
composition, colors, or details
3-4 (Below Average): Notable problems
affecting viewing experience
5-6 (Average): Basic aesthetic quality,
balanced but not unique
7-8 (Good): High quality with
sophisticated composition
9-10 (Excellent): Outstanding artistic
quality and impact

You MUST provide your response in
exactly this format:
Score: [X]/10
Analysis: [Your detailed analysis
covering composition, colors, technical
quality, and artistic impact]
```

The GAS evaluation process is designed to assess multiple aspects of scene quality, including composition, lighting, material quality, and overall artistic impact. As shown in Figure 13, our system consistently achieves high scores across different scene types, with particularly strong performance in natural environments and architectural scenes. The evaluation results demonstrate that our system can maintain high aesthetic standards while handling diverse generation requirements.

For semantic alignment evaluation, we implement a comprehensive CLIP-based similarity computation process, as detailed in Figure 14. Our implementation utilizes different clip models to encode both rendered images and text descriptions

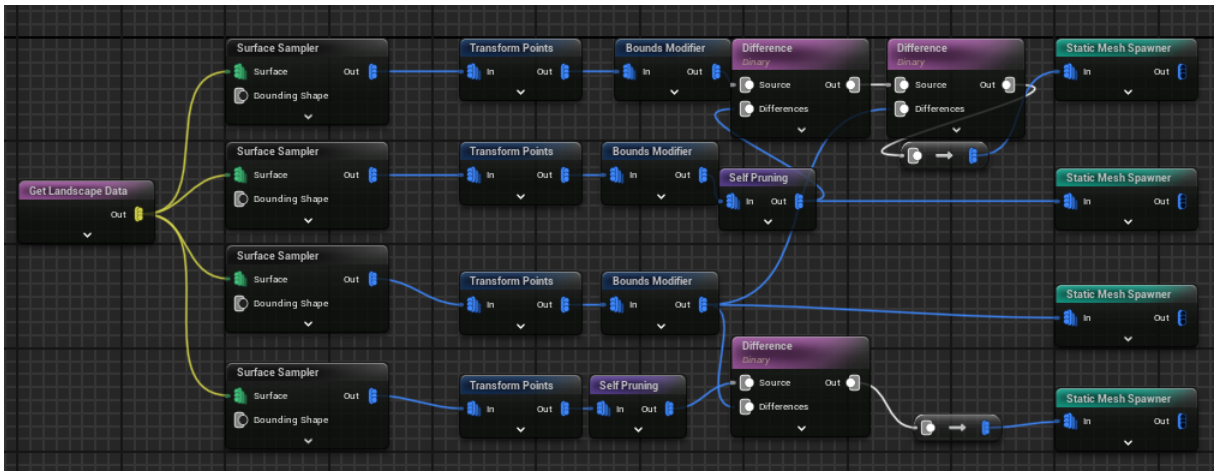

Figure 9: Example of a PCG system implementation in Unreal Engine, showing the node-based workflow that enables procedural scene generation. The visual programming interface allows for intuitive creation and modification of generation rules.

Figure 10: Demonstration of PCG's capability in handling varying complexity: from a single tree (left) to a cluster of trees (middle), and finally to a hierarchically structured forest (right), showing how PCG can scale from simple elements to complex scene compositions.

into a shared embedding space. The computation process includes several key steps: image preprocessing to ensure consistent input format, text prompt normalization, and cosine similarity calculation between the resulting embeddings. To handle multiple viewpoints of each scene, we compute similarities for multiple rendered views and aggregate the results.

Figure 12 presents a detailed analysis of CLIP similarity scores across our test scenarios. The results show consistently high alignment between generated scenes and their corresponding text descriptions. This strong performance is maintained across different scene categories, from simple natural environments to complex architectural compositions. Notably, our system achieves higher scores for scenes with distinct visual elements and clear spatial relationships, demonstrating its ability to accurately interpret and implement spatial and stylistic requirements from natural language descriptions.

C Text Representation of PCG Scenes

Following the discussion of PCG fundamentals and our evaluation process, this section further elaborates on the text-based PCG blueprint representation, a core mechanism of UnrealLLM introduced in Section 3.2. This representation method is pivotal in bridging the natural language understanding capabilities of Large Language Models (LLMs) with the complex Procedural Content Generation (PCG) system in Unreal Engine 5 (UE5). Figure 15 visually presents several excerpts of such textual representations alongside the corresponding scenes generated in UE5, offering a concrete look at its structure and output.

As detailed in Section 3.2, our representation employs a JSON-based Domain-Specific Language (DSL) for the textual depiction of PCG blueprints. The choice of JSON is primarily motivated by two key considerations.

Firstly, the structured nature of JSON allows for precise programmatic parsing, facilitating reliable bidirectional conversion with UE5 blueprints. Our custom UE5 plugin is designed to parse information defined in the JSON text, such as nodes, their parameters (or `settings`), and connections. This information is then accurately mapped and transformed into the internal PCG blueprint structures and corresponding parameter configurations within the UE5 engine. Significantly, this design supports the bidirectional conversion between our textual representation and UE's native blueprint format, as

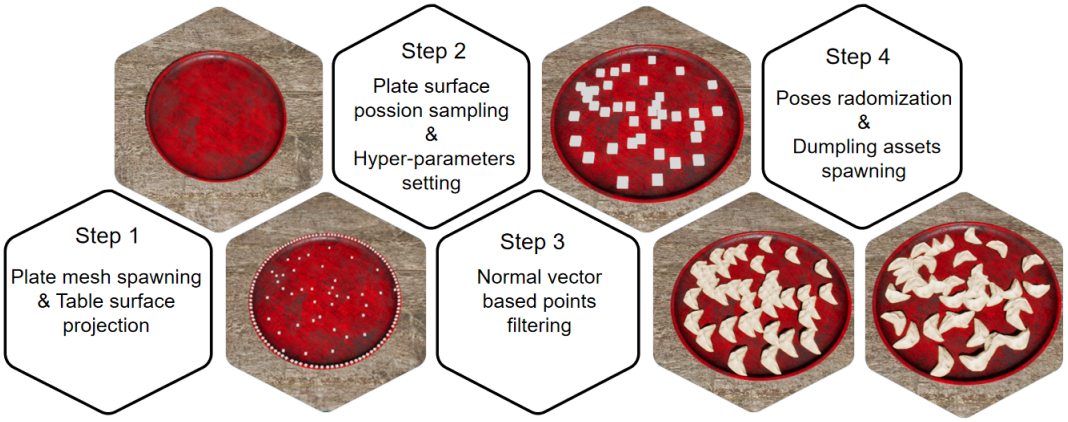

Figure 11: A simple workflow demonstration of PCG (Procedural Content Generation). Typically, PCG steps include: sampling and projection of the target space, 3D pose transformation, composition, filtering, and asset selection and generation. This image showcases how a realistic plate of dumplings can be generated through simple steps.

```
UnderWater7-JellyFish.png:
Prompt: A lively underwater scene with vibrant pink jellyfish gliding among lu
sh green seaweed, bathed in blue light with diffused sunlight and coral textur
es.
Similarity: 0.2556

UnderWater8-JellyFish.png:
Prompt: A group of translucent pink jellyfish float gracefully in deep blue wa
ter, illuminated by soft ambient light, with realistic aquatic textures and ge
ntle surface ripples.
Similarity: 0.3372

UnderWater9-Fishes.png:
Prompt: A clear underwater scene features a school of small fish in vibrant bl
ue water, with sunlight dappling through and casting soft shadows on the sandy
 seabed.
Similarity: 0.3285


Average similarity: 0.3141
Highest similarity: 0.3882
Lowest similarity: 0.2556
Standard deviation: 0.0278
```

Figure 12: CLIP similarity scores across test scenarios, demonstrating strong alignment between generated scenes and input descriptions.

```
Evaluating image 2: BrickWall2-Vines.png
Evaluating image...

Score: 7/10
Analysis:
The composition of the rendered image depicts a stone wall that is skillfully entwined with greenery, creating a visually pleas
ing contrast between the rugged texture of the stones and the softness of the foliage. The use of natural elements enhances the
 overall appeal, giving a sense of life and organic growth.

The colors are well-balanced, with earthy tones in the stone contrasting effectively with the vibrant green of the plants. This
 color scheme adds warmth and invites the viewer to appreciate the natural beauty depicted.

In terms of technical quality, the detailing on both the stones and leaves is commendable, lending a realistic look to the elem
ents. However, there could be improved variation in the shades of green and stone textures to add depth.

Artistically, while the image is high quality and evokes a sense of tranquility, it may lack a unique perspective or dramatic f
lair that would elevate it further. Some additional dynamics or narrative elements could enhance its impact. Overall, a solid r
endering with good aesthetic qualities.
Score: 7.0/10.0

Evaluating image 3: Desert1-Realistic.png
Evaluating image...

Score: 6/10
Analysis:
The rendered image captures a desert landscape with a defined sense of environment, showcasing sandy dunes and sparse vegetatio
n. The composition is relatively balanced, with the placement of rocks and dry shrubs contributing to the overall scene. Howeve
r, the image lacks dynamic elements or focal points, which could enhance its visual interest.

Color-wise, the warm tones of orange and beige effectively convey a dry and arid atmosphere, though they can feel one-dimension
al. There's a subtle variation in shading and texture that adds depth, but the overall palette could benefit from more contrast
ing colors to evoke a stronger emotional response.

Technically, the details in the vegetation and dunes are commendable, and the serene nature of the landscape is conveyed well.
However, the artistic impact is diminished due to the lack of a narrative or vivid subject matter. Overall, it presents a decen
t yet average aesthetic quality that could shine with more inventive composition and color interplay.
Score: 6.0/10.0
```

Figure 13: Example GAS evaluation results across different scene types, showing consistent aesthetic quality assessment.

```python
def load_prompts(json_file="human_prompts_inputs.json"):
    with open(json_file, 'r', encoding='utf-8') as f:
        return json.load(f)

def compute_similarities():
    print("Loading CLIP model...")
    model = SentenceTransformer('clip-ViT-B-32')
    print("Loading prompt data...")
    prompts = load_prompts()
    similarities = {}
    scores = []
    image_folder = "RendersForEvaluation"
    print("\nStarting to compute similarities...")
    for image_name, prompt in prompts.items():
        try:
            image_path = os.path.join(image_folder, image_name)
            img_emb = model.encode(Image.open(image_path))
            text_emb = model.encode([prompt])
            cos_score = float(util.cos_sim(img_emb, text_emb)[0][0])
            similarities[image_name] = cos_score
            scores.append(cos_score)
            print(f"{image_name}:")
            print(f"Prompt: {prompt}")
            print(f"Similarity: {cos_score:.4f}\n")
        except Exception as e:
            print(f"Error processing {image_name}: {str(e)}\n")
            continue
    if scores:
        average_score = np.mean(scores)
        print(f"\nAverage similarity: {average_score:.4f}")
        print(f"Highest similarity: {max(scores):.4f}")
        print(f"Lowest similarity: {min(scores):.4f}")
        print(f"Standard deviation: {np.std(scores):.4f}")
    else:
        print("No similarity scores were computed")
```

Figure 14: Implementation of CLIP similarity computation, showing the process of encoding and comparing scene renders with text descriptions.

mentioned in Section 3.2. We have implemented this bidirectional capability using custom parsers that extract and preserve the hierarchical structure of nodes, execution flows, and data dependencies. This opens up possibilities for LLMs to understand and edit existing PCG assets, or for visually constructed blueprints in UE5 to be exported into a textual format for further processing by LLMs.

Secondly, the JSON format is highly compatible with current Large Language Models. As a widely adopted standardized data interchange format, JSON's concise syntax and clear hierarchical structure are well-suited for LLM processing and generation. Many advanced LLMs, such as GPT-4o and Claude 3.5 as utilized in our framework (see Section 3.3), are capable of directly producing structured JSON output. Even smaller models can achieve high accuracy and conformance in JSON generation through techniques like constrained decoding. This LLM-friendliness greatly facilitates the task of the Blueprint Generator agent within the UnrealLLM framework, enabling it to construct and modify PCG logic textually based on analytical and planning outcomes. This structured intermediate format is also crucial for effectively constraining and validating LLM outputs, thereby enhancing the accuracy and reliability of the generated content.

Typically, this JSON text includes a top-level object describing the entire PCG graph. Its core content generally comprises an array of `nodes`, where each node object defines its type (e.g., *SurfaceSampler*, *StaticMeshSpawner*), a unique identifier within the graph (`NodeID`), and a `parameters` (or `settings`) object containing parameters specific to that node type. Examples include model asset paths (e.g., `MeshEntries`) for a static mesh spawner, or point density (e.g., `PointsPerSquareMeter`) for a surface sampler, as illustrated in Figure 15 (left column). Additionally, an array of `connections` is included to define the input-output relationships between nodes, thereby establishing the data flow and logical execution order of the PCG graph.

This carefully designed textual representation enables LLM agents to systematically define, modify, and iterate on PCG logic without needing to directly interact with UE5's complex visual programming interface. This lowers the barrier for LLMs to drive professional-grade PCG tools and provides a solid foundation for achieving more fine-grained and flexible automation of procedural content generation. Once the LLM has generated or modified the blueprint text in JSON format, our custom plugin converts it into executable PCG blueprints within the UE5 engine (as described in Section 3.3), which subsequently generate the desired 3D scenes.

D More Generations

To demonstrate the versatility and capabilities of our UnrealLLM system, we present an extensive collection of generated scenes across various environments and styles. Figure 16 showcases our system's ability to interpret diverse natural language inputs, from underwater environments with detailed marine vegetation to fairytale villages with atmospheric lighting. Each scene demonstrates not only visual fidelity but also complete interactivity, allowing users to explore and interact with the generated environments.

Figure 17 and Figure 18 extends this demonstration with specialized environments that highlight our system's technical capabilities. These include challenging scenarios such as: underwater ecosystems with complex caustic lighting, large-scale space fleet formations in cosmic settings, stylized forest environments requiring coordinated visual aesthetics, and maritime scenes with realistic water surface interactions. Notably, all generated environments support rich gameplay interactions ranging from basic character navigation to complex physics-based responses, particularly evident in spacecraft formation controls and water surface physics simulations.

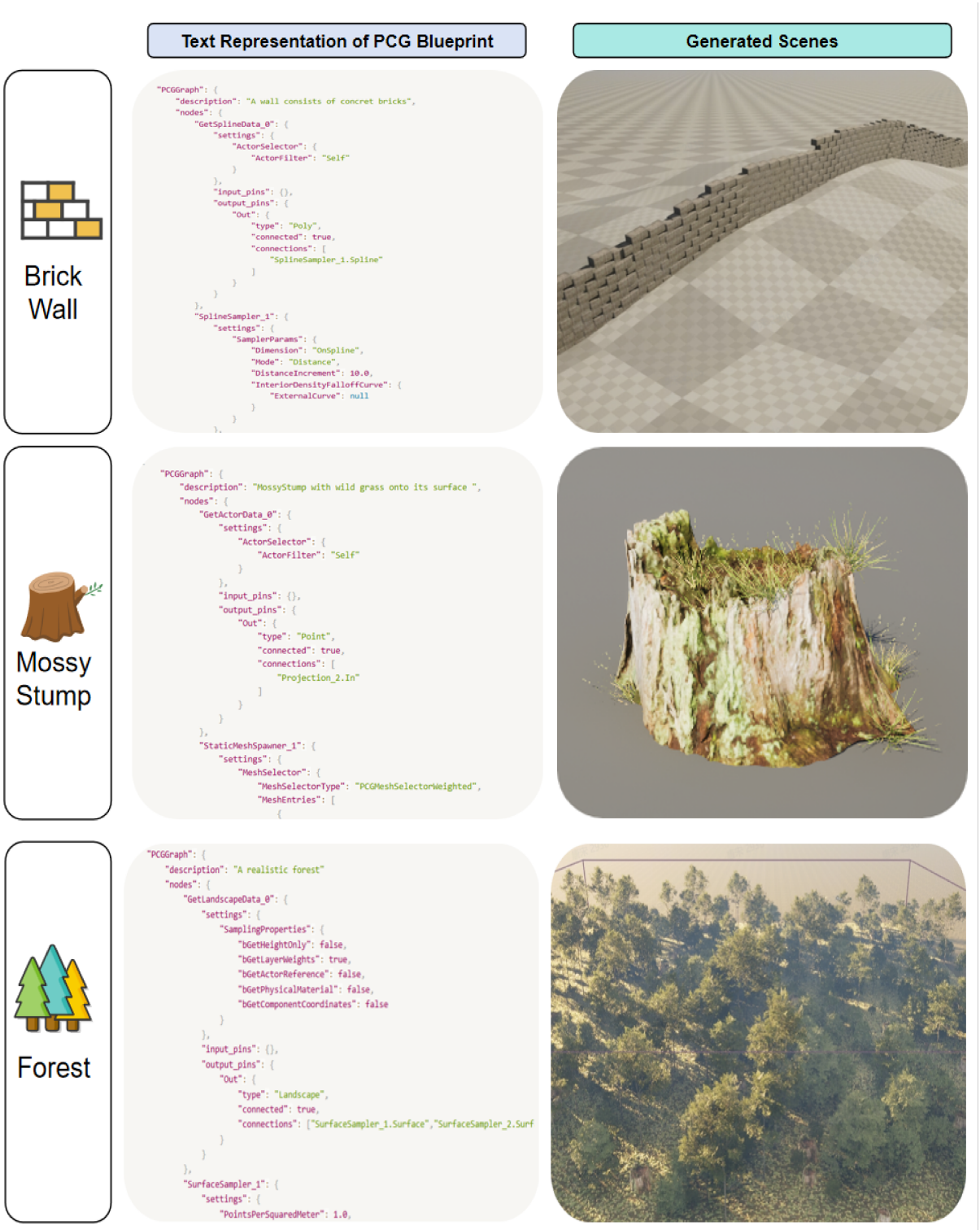

Figure 15: Examples of our JSON-based text for PCG blueprints (left) and the 3D scenes they generate in Unreal Engine 5 (right). These snippets show how PCG nodes, parameters, and connections are structured.

Figure 16: Our *UnrealLLM* converts natural language input from users into a variety of interactive 3D scenes, enhancing the ease of use of programmatic content generation technology

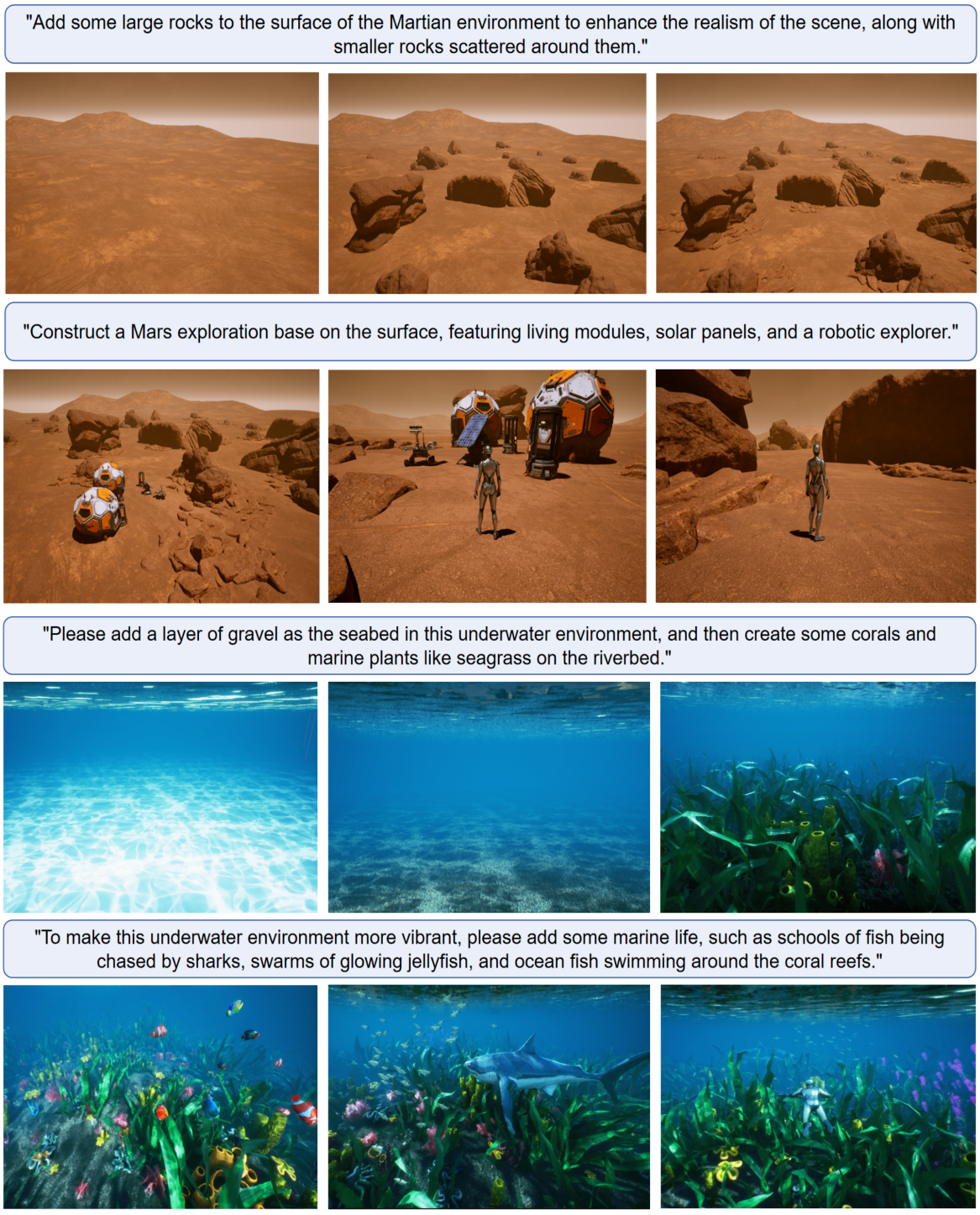

Figure 17: Demonstration of text to 3D scenes.

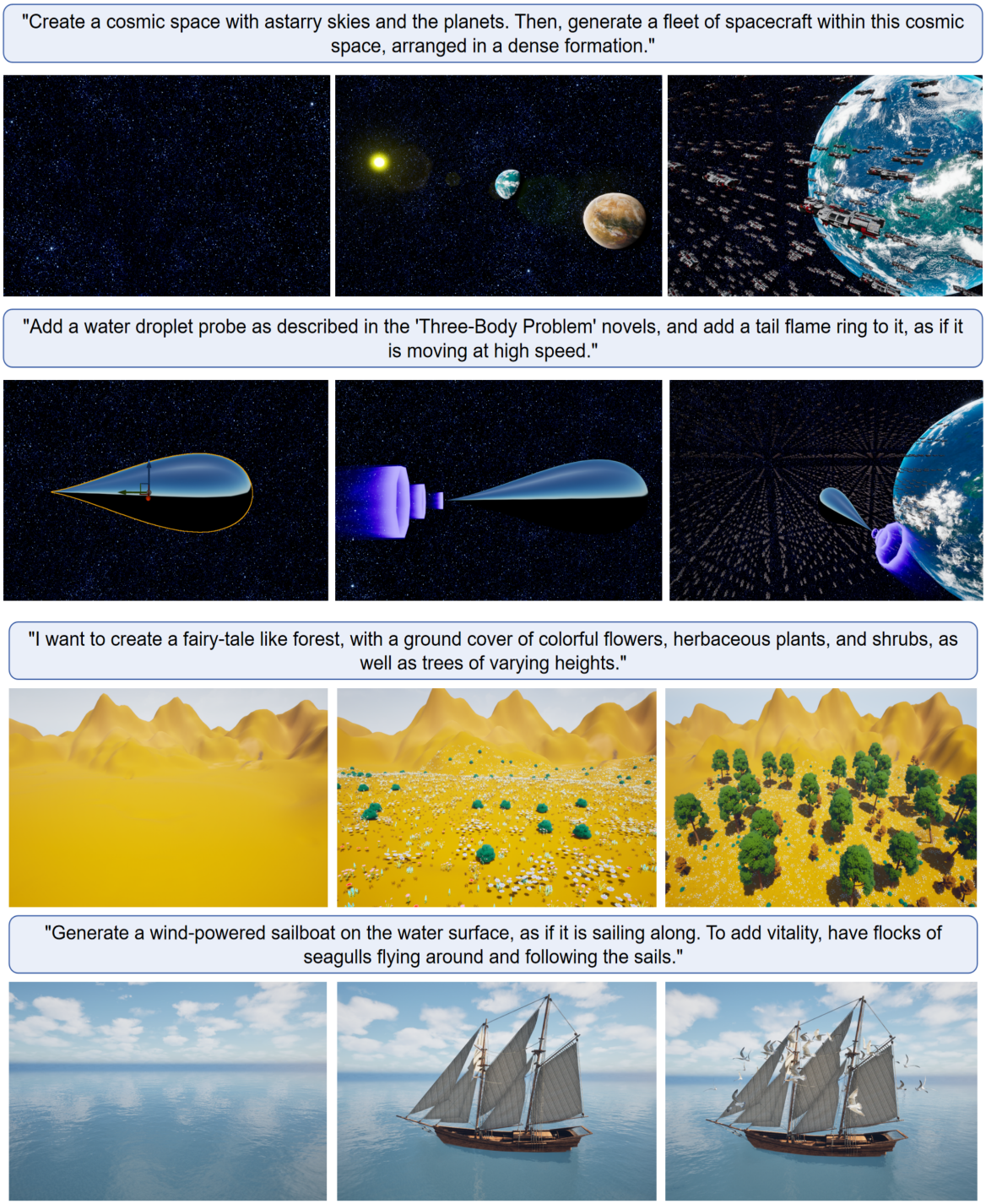

Figure 18: Demonstration of text to 3D scenes.