3D-GPT: Procedural 3D Modeling with Large Language Models

Chunyi Sun[1] Junlin Han[2] Weijian Deng[1] Xinlong Wang[3]
Zishan Qin[1] Stephen Gould[1]
[1]Australian National University [2] University of Oxford [3] BAAI

Abstract

In the pursuit of efficient automated content creation, procedural generation, leveraging modifiable parameters and rule-based systems, has emerged as a promising approach. Nonetheless, this can be a demanding endeavor, given its intricate nature necessitating a deep understanding of rules, algorithms, and parameters. To reduce workload, we introduce 3D-GPT, a framework utilizing large language models (LLMs) for instruction-driven 3D modeling. 3D-GPT positions LLMs as proficient problem solving agents, dissecting the procedural 3D modeling tasks into accessible segments and appointing the appropriate agent for each task. 3D-GPT integrates three core agents: the task dispatch agent, the conceptualization agent, and the modeling agent. They collaboratively achieve two objectives. First, they enhance the concise initial scene descriptions, elaborating details while dynamically adapting the text based on subsequent instructions. Second, they integrate procedural generation, extracting parameter values from enriched text to seamlessly interface with 3D software for asset creation. Our empirical investigations confirm that 3D-GPT not only correctly interprets instructions, delivering reliable results but also collaborates effectively with human designers. Furthermore, it integrates directly with Blender, unlocking expanded manipulation possibilities. Our work highlights the potential of LLMs in 3D modeling, offering a basic framework for future advancements in scene generation and animation.

1. Introduction

In the metaverse era, 3D content creation is vital for defining multimedia experiences in domains like gaming, virtual reality, and cinema with intricately crafted models. Yet, designers often grapple with a time-intensive 3D modeling process, starting from basic shapes (e.g., cubes, spheres, or cylinders) and employing software like Blender for meticulous shaping, detailing, and texturing. This demanding workflow concludes with rendering and post-processing to deliver the polished final model. While procedural generation holds promise with its efficiency in automating content creation through adjustable parameters and rule-based systems [4, 9, 10, 17, 30], it demands a comprehensive grasp of generation rules, algorithmic frameworks, and individual parameters. Furthermore, aligning these processes with the creative visions of clients, through effective communication, adds another layer of complexity. This underscores the importance of simplifying the traditional 3D modeling workflow to empower noivce creators in the metaverse era.

LLMs have shown exceptional language interpretation capabilities, including planning and tool utilization [8, 14, 40, 41]. Furthermore, LLMs demonstrate outstanding proficiency in characterizing object attributes, such as structure and texture [6, 21, 27], enabling them to enhance details from rough descriptions. Additionally, they excel at parsing concise textual information and comprehending software source code, while facilitating efficient and intuitive interactions with users. Driven by these extraordinary capabilities, we embark on exploring their innovative application to procedural 3D modeling. Our primary objective is to harness the power of LLMs to exert control over 3D creation software in accordance with natural language input from users.

In pursuit of this vision, we introduce 3D-GPT, a framework to facilitate instruction-driven 3D content synthesis. 3D-GPT enables LLMs to function as problem-solving agents, breaking down the 3D modeling task into small, manageable components, and determining how to accomplish each component. 3D-GPT comprises three key agents: the conceptualization agent, the 3D modeling agent, and the task dispatch agent. The first two agents combine to fulfill the roles of conceptualization and modeling by manipulating the 3D generation API functions. Subsequently, the third agent manages the system by taking the initial text input, handling subsequent instructions, and facilitating effective cooperation between the two aforementioned agents.

By doing so, 3D-GPT works towards two key objectives. First, it enhances initial scene descriptions, guiding them to more detailed and contextually relevant forms while adapting to subsequent instructions that refine the scene description. Second, instead of directly crafting every element of 3D content, we employ procedural generation, using adaptable parameters and rule-based systems to interface with

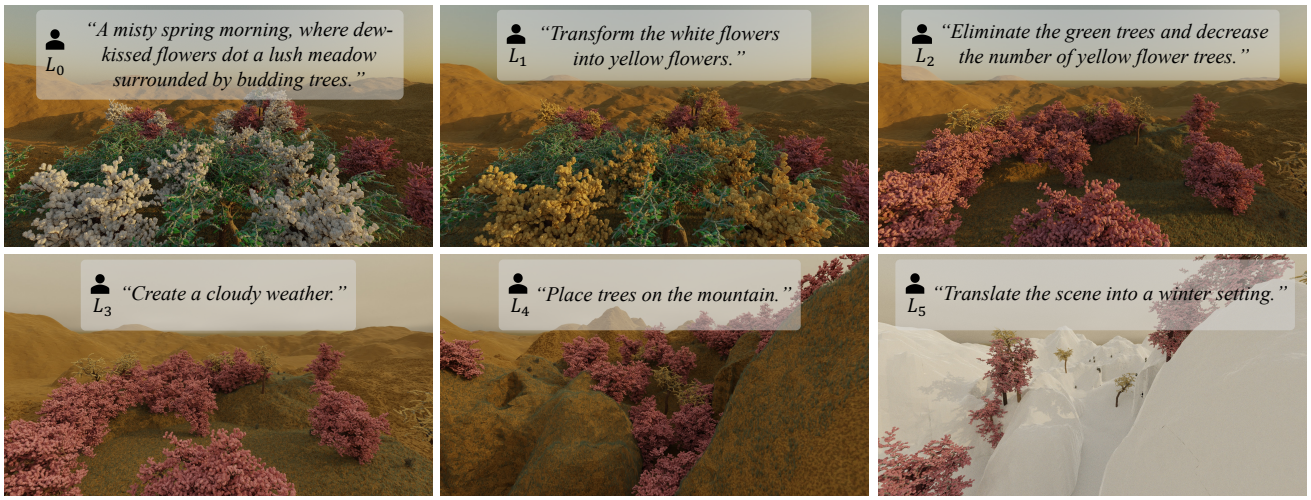

Figure 1. **Subsequence Instruction Editing Result.** L_0**:** Initial instruction-generated scene. L_1**-**L_5**:** Sequential editing steps with corresponding instructions. Our method enables controllable editing and effective user-agent communication.

3D software. Our 3D-GPT is equipped with the capability to understand procedural generation functions and extract corresponding parameter values from the enriched text.

3D-GPT offers controllable and precise 3D generation guided by users' textual descriptions. It reduces the workload of manually defining each controllable parameter in procedural generation, particularly within complex scenes that encompass diverse aspects. Moreover, 3D-GPT enhances collaboration with users, making the creative process more efficient and user-centric as demonstrated in Fig. 1. Furthermore, 3D-GPT seamlessly interfaces with Blender, granting users diverse manipulation capabilities: material adjustments, primitive additions, object animations, mesh editing, and physical motion simulations. Our contributions are summarized as follows:

- **Introducing 3D-GPT:** A zero-shot and training-free framework designed for 3D scene generation. 3D-GPT leverages the innate multimodal reasoning capabilities of LLMs, improving the efficiency of end-users engaged in procedural 3D modeling.
- **Exploration of Text-to-3D Generation:** Our 3D-GPT generates Python code to control 3D software, potentially offering increased flexibility for real-world applications.
- **Empirical Experiments:** Demonstrate the great potential of LLMs in reasoning, planning, and tool-using capabilities for 3D content generation.

2. Related Work

Text-to-3D generation. With the recent advance in text-to-image generation modeling, there has been a growing interest in text-to-3D generation [18, 22, 26, 33, 35, 37]. The common paradigm of them is to perform per-shape optimization with differentiable rendering and the guidance of the CLIP model [28] or 2D diffusion models [32]. For example, DreamFields [15] and CLIP-Mesh [24] explore zero-shot 3D content creation using only CLIP guidance. Dreamfusion [26] optimizes NeRF [23] with the guidance of a text-to-image diffusion model, achieving remarkable text-to-3D results. Further works in this direction have resulted in notable enhancements in visual quality [18, 20], subject-driven control [22, 31], and overall processing speed [15, 19]. Unlike the above approaches, our objective is not to generate neural representations as the final 3D output. Instead, we utilize LLMs to generate Python code that controls Blender's 3D modeling based on the provided instructions.

Large Language Models (LLMs). LLMs are a promising approach to capture and represent the compressed knowledge and experiences of humans, projecting them into language space [2, 3, 5, 25, 29]. LLMs exhibit the capability to address intricate tasks that were once considered the exclusive domain of specialized algorithms or human experts. These tasks encompass areas such as mathematical reasoning [14, 36], medicine [16, 38], planning [8, 12, 13, 41], and LaMDA [34] for dialogue applications. Our work explores the innovative application of LLMs in 3D modeling, employing them to control 3D procedural generation.

Recent works have used LLMs for 3D scene modeling [1, 7, 11, 39], with SceneCraft [11] enabling interaction with large asset pools and BlenderGPT [1] facilitating basic Blender operations. While they effectively use existing 3D asset libraries, they can be limited by the fixed nature of these assets, which may restrict editing flexibility. By integrating LLMs with procedural generation frameworks like Infinigen [30], our method enables dynamic creation and fine-grained manipulation of 3D assets, surpassing the constraints of static libraries. This advancement opens new possibilities for personalized content creation.

3. 3D-GPT

3.1. Task Formulation

The overall goal of this work is to generate 3D content in a conversational manner through a sequence of short, user-provided, natural language instructions, denoted as $\mathcal{L} = \langle L_i \rangle$. The initial instruction, L_0, serves as a concise description of the 3D scene, such as "*a misty spring morning, where dew-kissed flowers dot a lush meadow surrounded by budding trees*." Subsequent instructions are then employed to modify the rendered scene, e.g., "*transform the white flowers into yellow flower*" or "*translate the scene into a winter setting*" to add desried detail.

To accomplish this objective, we introduce a framework named 3D-GPT, which leverages LLMs as problem-solving agents. We note that employing LLMs to directly create every element of 3D content poses significant challenges. LLMs lack specific pre-training data for proficient 3D modeling and, as a result, may struggle to accurately determine what elements to use and how to modify them based on given instructions. To address this challenge, we employ procedural generation to control the 3D content creation. This makes use of adaptable parameters and rule-based systems to interface with 3D software (e.g., Blender) so as to efficiently conduct 3D modeling [4, 9, 10, 17, 30]. Nevertheless, there are several challenges that remain such as identifying the correct procedures to call and mapping of language to API parameters. We solve these using multiple language agents as will be discussed below.

Our approach considers the 3D procedural generation engine as a set of functions, denoted as $\mathcal{F} = \{F_j\}$, where each function F_j takes parameters P_j. For example, `add_trees(scene, density, distance_min, leaf_type, fruit_type)` will add trees to a given base scene with properties controlled by the four parameters `density`, `distance_min`, etc.

In our framework, for the initial user-provided instruction (scene description) L_0, we utilize the entire function set $\mathcal{F}$ to construct the 3D scene. Subsequently, we infer the corresponding parameters P_j for each function F_j within this set. When the user provides subsequent instructions L_i $(i > 0)$ for scene modifications, such as "*transform the white flowers into yellow flowers*," we adopt a more focused approach to prevent undesired modifications and enhance efficiency. In this example, the system should only select functions related to flowers. Therefore, it selects a subset of relevant functions, $\widehat{\mathcal{F}} \subseteq \mathcal{F}$, from the original function set $\mathcal{F}$. Then, we proceed to infer the corresponding parameters P_j for each function F_j within this subset. By addressing both function selection and parameter inference for every sub-instruction L_i, 3D-GPT generates a Python script file that allows Blender's 3D modeling environment to render high-quality scenes consistent with the instruction sequence $\mathcal{L}$.

3.2. Modeling Tool Preparation

In our framework, we use Infinigen [30], a Python-Blender-based procedural generator equipped with a rich library of generation functions. To allow LLMs to invoke the Infinigen API, we provide following crucial language prompts for each function F_j:

- **Documentation** (D_j): A comprehensive explanation of the function's purpose and clear description of its parameters P_j as one would find in standard API documentation.
- **API code** (C_j): Restructured and readable function code, ensuring that it is accessible and comprehensible to LLMs.
- **Auxiliary parameter information** (I_j): Outlines specific information required to infer the function parameters to assist LLMs in understanding the context and prerequisites of each function. For example, in the case of a flower generation function, I_j indicates the necessary visual properties for rendering, such as petal color, petal shape (e.g., size, curve, and length), and center appearance.
- **Usage examples** (E_j): Illustrative examples that demonstrate how to infer parameters P_j from the accompanying text descriptions and subsequently invoke the function. Continuing with the example above, E_j would include a practical demonstration of how to infer the parameters and call the function based on input text like "*a sunflower*."

By providing LLMs with these prompts, we enable them to leverage their generative competencies in planning, reasoning, and tool utilization. Documentation (D) elucidates functions in plain language, revealing their purpose and input data types. API Code (C) offers practical implementation, assisting the LLM in dissecting code. Auxiliary Parameter Information (I) provides the necessary details for parameter inference and can be harnessed by our agent to query and infer supplementary context. Usage Examples (E) aid the LLM in generating analogous code for novel tasks, uncovering hidden usage patterns and best practices not always evident in documentation. As a result, LLMs can effectively harness Infinigen for 3D generation based on language instructions in a seamless and efficient manner. In the context of our work, the function set $\mathcal{F}$ encompasses all functions and subfunctions within the Infinigen scene generation script, with the sole exception of the 'creatures' class. These functions play an indispensable role in our scene creation process. In the supplemental material, we present a comprehensive list of all the functions used to construct the scenes and examples for some of these functions.

3.3. Multi-agents for 3D Modeling

With the necessary tool preparation (i.e., D_j, C_j, I_j and E_j) in hand, 3D-GPT employs a multi-agent system to tackle the task of language-guided procedural 3D modeling. This system comprises three integral agents: (1) the task dispatch agent, (2) the conceptualization agent, and (3) the modeling agent, illustrated in Fig. 2. Together, these agents decom-

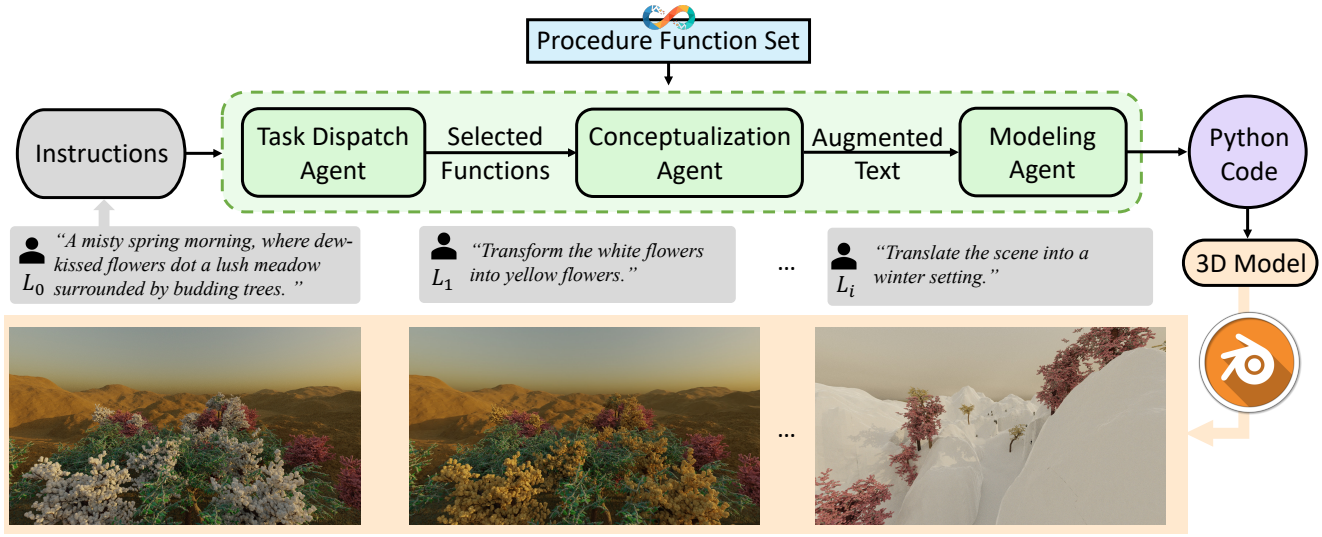

Figure 2. **3D-GPT Overview.** 3D-GPT employs LLMs as a multi-agent system with three collaborative agents for procedural 3D generation. These agents consult documents from the procedural generator, infer function parameters, and produce Python code. The generated code script interfaces with Blender's API for 3D content creation and rendering.

pose the modeling task into manageable pieces, with each agent specializing in distinct aspects: planning, 3D reasoning, and tool utilization. The task dispatch agent plays a pivotal role in the planning process. It queries function documents and subsequently selects the requisite functions for execution from natural language user instructions. Once functions are selected, the conceptualization agent engages in reasoning to enrich the user-provided text description. Building upon this, the modeling agent deduces the parameters for each selected function and generates Python code scripts to invoke Blender's API, facilitating the creation of the corresponding 3D content. From there, images and video flythroughs can be rendered directly using Blender.

Task Dispatch Agent for Planing. The task dispatch agent utilizes a comprehensive knowledge base encompassing all available functions $\mathcal{F}$ from documentation D. It also has access to an illustrative example on how to select the appropriate function subset based on the given instructions. When presented with an instruction, such as "*translate the scene into a winter setting*," the agent systematically interprets the task. It analyzes the documentation for each function, aligning requirements with function capabilities, and learn from in-context examples. Consequently, the agent efficiently identifies the requisite functions for each instructional input, such as `add_snow_layer` and `update_trees`, while excluding functions like `add_flowers` and `add_rocks`. This pivotal role played by the task dispatch agent is instrumental in facilitating efficient coordination between the conceptualization and modeling agents. Without it, the those agents would have to analyze all provided functions $\mathcal{F}$ for each given instruction. This not only increases the workload for these agents but also extends processing time and can potentially lead to undesired modifications.

The communication flow between the LLM system, the user, and the task dispatch agent is outlined as follows:

> **Message**: You are a proficient planner for selecting suitable functions based on user instructions. You are provided with the following functions: $\langle (F_j^{name}, F_j^{usage}) \rangle$. Below are a few examples of how to choose functions based on user instructions: $\langle E^{\text{task\_dispatch}} \rangle$. The user instruction is: $\langle L_i \rangle$.
> **Task Dispatch Agent**: Given the instruction $\langle L_i \rangle$, we determine the sublist of functions $\hat{\mathcal{F}}$ that need to be used for 3D modeling.

Here $\langle (F_j^{name}, F_j^{usage}) \rangle$ represents a list of function names and concise function usage descriptions for all available functions and examples $\langle E^{task\_dispatch} \rangle$ provide guided examples for prompt-based instructions. A example is provided in the supplemental material.

Conceptualization Agent for Reasoning. User instructions often lack sufficient modeling details. For instance, consider the instruction, "*a misty spring morning, where dew-kissed flowers dot a lush meadow surrounded by budding trees*". Here many required function parameters such as tree branch length, tree size, and leaf type, are not directly stated in the given text. When instructing the modeling agent to infer parameters directly, we observed that it tends to provide simplistic solutions, such as using default values or copying values from prompting examples. This reduces diversity in generation and complicates parameter inference.

To address this issue, we propose the conceptualization agent that interacts with the task dispatch agent to augment the user-provided text description (L_i) with required information list in I_j for each function. After the task dispatch agent selects the required functions, we send the user input text and the corresponding function-specific information to the conceptualization agent and request augmented text. For each function F_j, it enriches L_i to produce detailed appearance descriptions $\widehat{L_i}^j$. The communication between the system and the Conceptualization Agent for instructions $\langle L_i \rangle$ and functions $\langle F_j \rangle$ is as follows:

> **Message** : You are a skilled writer, especially when it comes to describing the appearance of objects and large scenes. Given a text $\langle L_i \rangle$, provide detailed descriptions for the following information $\langle I_j \rangle$. For terms not mentioned in the description, use your imagination to ensure they fit the text description.
> **Conceptualization Agent** : Given the $\langle L_i \rangle$ and requested information $\langle I_j \rangle$, the extended description is: $\langle \widehat{L_i}^j \rangle$.

Modeling Agent for Tool Using. After conceptualization, the 3D modeling processing is invoked to convert the detailed natural language to machine-understandable API function calls. For each function F_j and user instruction L_i, the task dispatch agent receive augmented context $\widehat{L_i}^j$ from the conceptualization agent. Moreover, for each function F_j, we have the code C_j, function documentation D_j, and one usage example E_j. The modeling agent uses this information to select appropriate functions and deduce corresponding parameters. Subsequently, the modeling agent generates Python code that calls the selected function in the right context (*e.g.*, within a loop), passing in parameters inferred from the text and of the proper data type. Communication between System and Modeling Agent is as follows:

> **Message**: You are a good 3D designer who can convert long text descriptions into parameters, and understand Python functions to manipulate 3D content. Given the text description $\langle \widehat{L_i}^f \rangle$, we have the following function codes $\langle C_j \rangle$ and the document for function $\langle D_j \rangle$. Below is an example bout how to make function calls to model the scene to fit the description: $\langle E_j^{modeling} \rangle$. Understand the function, and model the 3D scene that fits the text description by making a function call.
> **Modeling Agent**: Given the description $\langle \widehat{L_i}^j \rangle$, we model the object by calling functions: ...

We have included a comprehensive communication example between all three agents in the supplemental material.

Blender Rendering. The modeling agent ultimately constructs the Python function calls with inferred parameters, which are supplied to Blender for controlling view ports and rendering, and thereby resulting in production of the final 3D mesh and RGB results.

Implementation Detail. We implemented our system using the Infinigen [30] API. The specific function set $\mathcal{F}$ we used is available in the generation script provided in the supplemental material. Our modeling agent is developed with the OpenAI's ChatGPT API, and its code implementation can also be found in the supplemental material, showcasing the ease of system implementation.

4. Experiments

Our experimentation begins by testing the proficiency of 3D-GPT in consistently generating results that align with user instructions, encompassing scenarios involving both large scenes and individual objects. Subsequently, we delve into specific examples to illustrate how our agents effectively comprehend tool functionalities, access necessary knowledge, and employ it for precise control. As fruther analysis, we conduct an ablation study to systematically examine the contributions of each agent within our multi-agent system.

4.1. 3D Modeling

Large Scene Generation. We investigate the capability of 3D-GPT to control modeling tools based on scene descriptions *without any training*. To conduct this experiment, we generated 100 scene descriptions using ChatGPT with the following prompt: "*You are a good writer, provide* 10 *different natural scene descriptions for me*." We collected responses to this prompt 10 times to form our dataset.

In Fig. 3, we present the multi-view rendering results of 3D-GPT. These results indicate that our approach is capable of generating large 3D scenes that align well with the provided text descriptions and showcasing a noticeable degree of diversity. Importantly, all 3D outcomes are directly rendered using Blender, ensuring that all meshes are authentic, thereby enabling our method to achieve absolute 3D consistency and produce real ray-traced results.

Fine-detail Control for Single Class. Apart from generating large scenes from concise descriptions, we assess 3D-GPT's capabilities for modeling objects. We evaluate crucial factors such as curve modeling, shape control, and an in-depth understanding of object appearances. To this end, we report the results of fine-grained object control, including nuanced aspects like object curves, key appearance features,

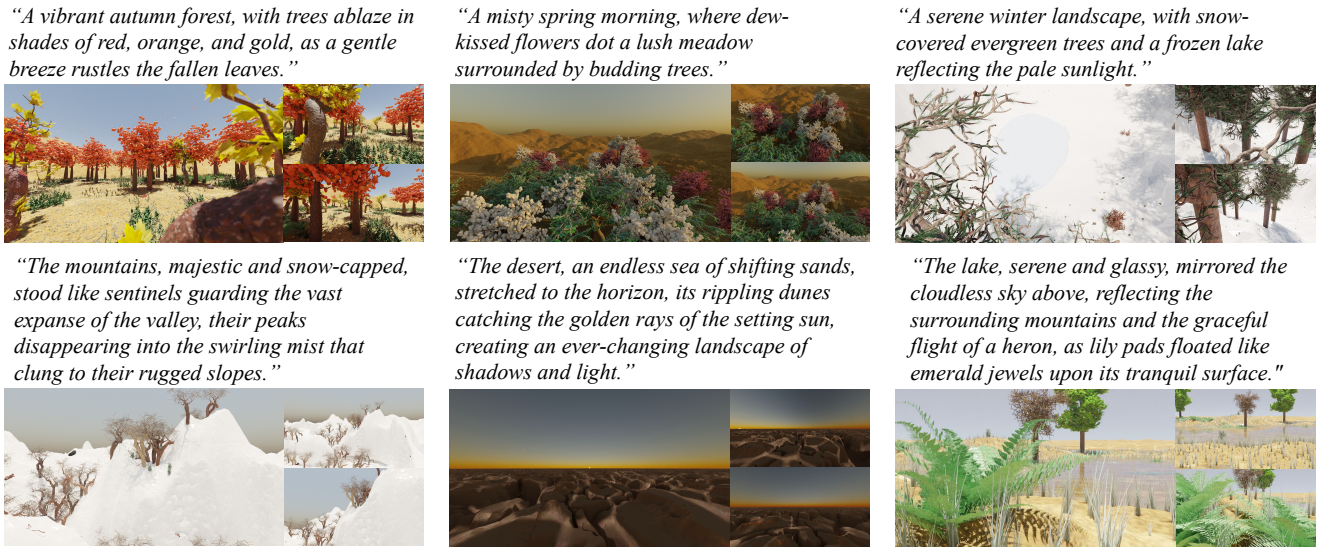

Figure 3. **Visual Examples of Instruction-Based 3D Scene Generation.** 3D-GPT can construct large 3D scenes that align with the provided initial instruction. We demonstrate that the rendered images contain various visual factors in line with the given instructions.

Figure 4. **Single Class Control Result.** Our method effectively acquires the necessary knowledge for modeling, enabling precise object control in terms of shape, curve, and key appearance capture. The generated results closely align with the given text.

and color, all derived from input text descriptions. We employ random prompts to instruct GPT for various real-world flower types. As depicted in Fig. 4, our method adeptly models each flower type, faithfully capturing their distinct appearances. This study underscores the potential of 3D-GPT in achieving precise object modeling and fine-grained attribute control of object types and visual characteristics.

Subsequence Instruction Editing. Here, we test the ability of 3D-GPT for effective human-agent communication and task manipulation. In Fig. 1, we observe that our method can comprehend subsequence instructions and make accurate decisions for scene modification. Note that, unlike the existing text-to-3D methods, 3D-GPT maintains a memory of all prior modifications, thereby facilitating the connection of new instructions with the scene's context. Furthermore, our method eliminates the need for additional networks for controllable editings [42]. This study underscores the efficiency and versatility of 3D-GPT in adeptly handling complex subsequence instructions for 3D modeling.

Individual Function Control To evaluate the effectiveness of 3D-GPT in tool utilization, we present an illustrative example that highlights our method's ability to control individual functions and infer parameters. Fig. 5 exemplifies the capability of 3D-GPT to model sky appearance based on input text descriptions. It is worth noting that the function responsible for generating the sky texture does not directly correlate color information with sky appearance. Instead, it relies on the Nishita-sky modeling method, which requires

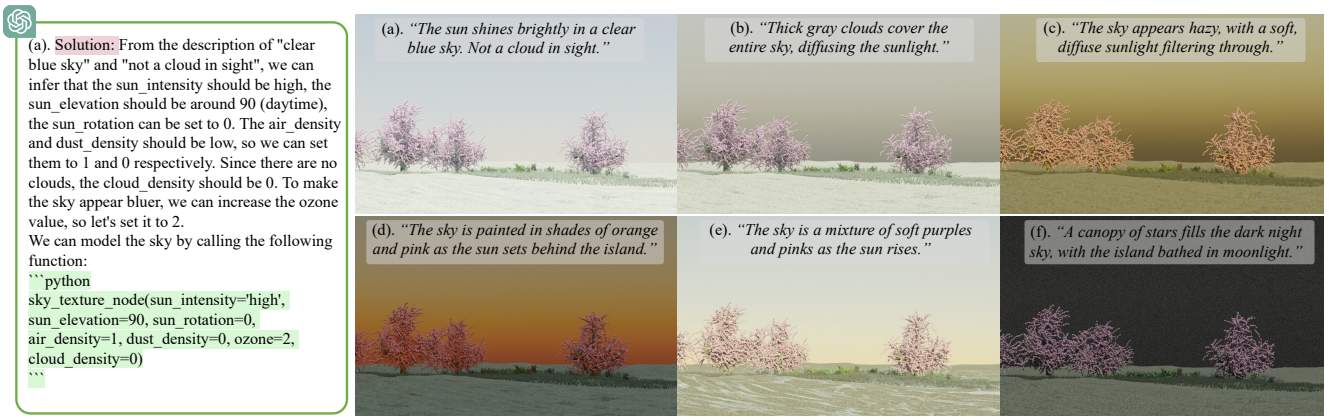

Figure 5. **Single Function Control Result.** Visual result (top) and modeling agent response example (bottom). Our method demonstrates a high degree of accuracy in inferring algorithm parameters, even when they do not possess a direct connection to visual appearance.

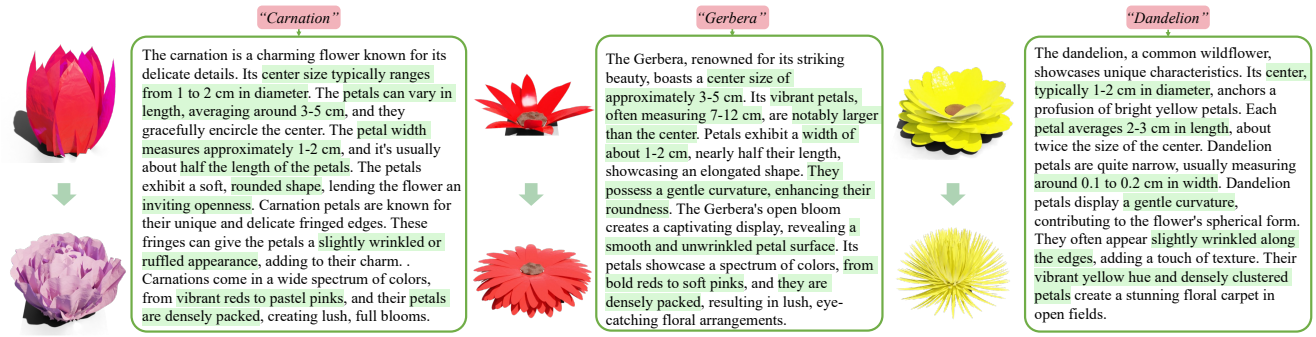

Figure 6. **Conceptualization Agent Case Study.** The enriched textual evidence demonstrates that the Conceptualization Agent provides essential knowledge for parameter inference (highlighted in green). For each subfigure, we compare the 3D model without (Top) and with (Bottom) agent. The models generated with the agent better match the text description than those without it .

a deep understanding of real-world sky and weather conditions, considering input parameters. Our method extracts relevant information from the textual input and comprehends how each parameter influences the resulting sky appearance, as evident in Fig. 5 (c) and (d). These results demonstrate that our method can effectively use individual functions as well as infer corresponding parameters.

Comparison with Text-to-3D. We provide a comparison with the state-of-the-art Text-to-3D method, Dreamfusion [26], for scene generation, as shown in Fig. 7. Our method produces visually more plausible results with outputs that are better aligned with the given text descriptions.

4.2. Ablation Study

We conduct separate ablation studies for the Conceptualization Agent, the Task Dispatch Agent, prompting components, types of LLMs, and number of examples. Our assessment focused on CLIP scores [28], failure rates, and parameter diversity, quantified using the categorical Shannon Diversity Index. The CLIP score measures the alignment between text and generated images. The failure rate represents the percentage of system failures due to issues such as incorrect datatypes, wrong response patterns, or missing parameters from the Modeling Agent. Parameter diversity aims to gauge the diversity of generated outputs.

(a) **Task Dispatch Agent.**

Method	CLIP Score
w/o TDA	22.79
Ours	29.16

(b) **Conceptualization Agent.**

Method	CLIP Score	Failure Rate	Para. Diversity
w/o CA	21.51	3.6%	6.32
Ours	30.30	0.8%	7.34

Table 1. Ablation Study on the Performance of Different Agents. (a) Impact of removing the Task Dispatch Agent (TDA) on CLIP Score. (b) Impact of removing the Conceptualization Agent (CA) on CLIP Score, Failure Rate, and Parameter Diversity.

Case Study of Task Dispatch Agent. For the Task Dispatch Agent, the CLIP score is measured using 100 initial scene descriptions, each appended with one additional subsequence instruction for each scene. Tab. 1 (a) shows that without the Task Dispatch Agent, the CLIP score dropped from 29.16 to 22.79. It is important to note that the Task Dispatch Agent primarily impacts the performance of subse-

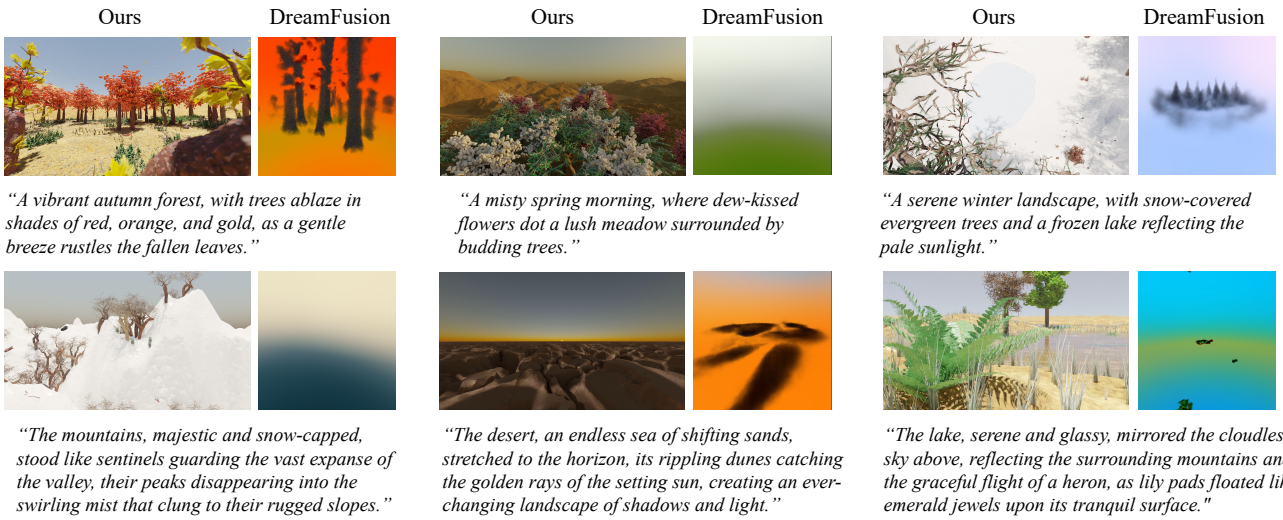

Figure 7. **Comparison with Text-to-3D Method.** This figure compares our method and DreamFusion [26], for generating various scenes.

Variants	CLIP Score	Failure Rate	Parameter Diversity
w/o D	20.7	4.2%	6.94
w/o C	28.4	1.8%	6.74
w/o I	21.6	1.4%	6.38
w/o E	24.5	3.4%	7.89
Ours	30.3	0.8%	7.34

Table 2. Ablation Study on the Impact of Prompting Components (D/C/I/E) on Model Performance.

quence instructions, as all functions are utilized for the initial instruction. These findings underscore the pivotal role of the Task Dispatch Agent in managing communication flow.

Case Study of the Conceptualization Agent. The Conceptualization Agent's effectiveness was evaluated using a CLIP score measured from 100 initial scene descriptions. As shown in Tab. 1 (b), the absence of the Conceptualization Agent led to significant declines in text alignment and parameter diversity, alongside a substantial increase in the failure rate, which negatively impacts the efficiency of the entire modeling process. Fig. 6 visually demonstrates how the Conceptualization Agent aids in acquiring essential knowledge for 3D modeling, comparing results with and without its involvement. When the Conceptualization Agent is utilized, the generated outputs closely match the intended flower type's appearance, underscoring its crucial role in enhancing the overall quality and fidelity of 3D generation.

Moreover, we conducted ablation studies examining prompting components (See Tab. 2), types of Large Language Models (See Tab. 3), and the number of examples (See Tab. 4). Key findings include: (1) each prompting component significantly contributes to the overall pipeline, (2) the type of LLM used does not notably impact 3D generation

LLMs	CLIP Score	Failure Rate	Parameter Diversity
LLAMA2	29.7	1.4%	6.97
GPT4	31.2	0.6%	7.23
GPT3.5	30.3	0.8%	7.34

Table 3. Ablation Study on the Performance of Different Large Language Models. We report CLIP Score, Failure Rate, and Parameter Diversity across LLAMA2, GPT-4, and GPT-3.5.

Number	CLIP Score	Failure Rate	Parameter Diversity
0	24.5	3.4%	7.89
1	30.3	0.8%	7.34
2	30.1	1.0%	7.23
3	30.2	0.8%	6.93

Table 4. Ablation Study on the Impact of Example Number.

results, and (3) there is no significant difference between one-shot, two-shot, and three-shot prompts, though increasing the number of examples slightly reduces parameter diversity.

5. Conclusion

We have introduced 3D-GPT, a novel framework for instruction-driven 3D procedural scene generation, leveraging pre-trained LLMs. Our approach involves coordination between three agents functioning as a cohesive 3D modeling team, ultimately yielding a 3D modeling file as output, as opposed to conventional 3D neural representations. Moreover, our method consistently delivers high-quality results, showcases adaptability to expansive scenes, ensures 3D consistency, provides material modeling and editing capabilities, and facilitates true ray tracing for achieving lifelike visualizations. Our empirical results show the potential of LLMs for reasoning, planning, and tool use in procedural 3D modeling.

References

[1] *https://github.com/gd3kr/BlenderGPT*. 2

[2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023. 2

[3] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 2

[4] 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. 1, 3

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2

[6] Lijie Fan, Dilip Krishnan, Phillip Isola, Dina Katabi, and Yonglong Tian. Improving clip training with language rewrites. *arXiv preprint arXiv:2305.20088*, 2023. 1

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

[8] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. MindAgent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023. 1, 2

[9] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3749–3761, 2022. 1, 3

[10] Ju He, Enyu Zhou, Liusheng Sun, Fei Lei, Chenyang Liu, and Wenxiu Sun. Semi-synthesis: A fast way to produce effective datasets for stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2884–2893, 2021. 1, 3

[11] 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

[12] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147, 2022. 2

[13] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023. 2

[14] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023. 1, 2

[15] 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

[16] Katharina Jeblick, Balthasar Schachtner, Jakob Dexl, Andreas Mittermeier, Anna Theresa Stüber, Johanna Topalis, Tobias Weber, Philipp Wesp, Bastian Sabel, Jens Ricke, and Michael Ingrisch. ChatGPT makes medicine easy to swallow: An exploratory case study on simplified radiology reports. *arXiv preprint arXiv:2212.14882*, 2022. 2

[17] Chenfanfu Jiang, Siyuan Qi, Yixin Zhu, Siyuan Huang, Jenny Lin, Lap-Fai Yu, Demetri Terzopoulos, and Song-Chun Zhu. Configurable 3D scene synthesis and 2D image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126:920–941, 2018. 1, 3

[18] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3D: High-resolution text-to-3D content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023. 2

[19] Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Zexiang Xu, Hao Su, et al. One-2-3-45: Any single image to 3D mesh in 45 seconds without per-shape optimization. *arXiv preprint arXiv:2306.16928*, 2023. 2

[20] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. Realfusion: 360 reconstruction of any object

from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023. 2

[21] Sachit Menon and Carl Vondrick. Visual classification via description from large language models. *arXiv preprint arXiv:2210.07183*, 2022. 1

[22] Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. Latent-NeRF for shape-guided generation of 3D shapes and textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12663–12673, 2023. 2

[23] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2

[24] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. CLIP-Mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 conference papers*, pages 1–8, 2022. 2

[25] OpenAI. GPT-4 technical report, 2023. 2

[26] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 2, 7, 8

[27] Sarah Pratt, Ian Covert, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. *arXiv preprint arXiv:2209.03320*, 2022. 1

[28] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 2, 7

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 2

[30] 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, 3, 5

[31] Amit Raj, Srinivas Kaza, Ben Poole, Michael Niemeyer, Ben Mildenhall, Nataniel Ruiz, Shiran Zada, Kfir Aberman, Michael Rubenstein, Jonathan Barron, Yuanzhen Li, and Varun Jampani. DreamBooth3D: Subject-driven text-to-3D generation. In *Proceedings of the International Conference on Computer Vision*, 2023. 2

[32] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 2

[33] Aditya Sanghi, Hang Chu, Joseph G Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. CLIP-Forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18603–18613, 2022. 2

[34] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022. 2

[35] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. ProlificDreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *arXiv preprint arXiv:2305.16213*, 2023. 2

[36] 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. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. 2

[37] Jiale Xu, Xintao Wang, Weihao Cheng, Yan-Pei Cao, Ying Shan, Xiaohu Qie, and Shenghua Gao. Dream3D: Zero-shot text-to-3D synthesis using 3D shape prior and text-to-image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20908–20918, 2023. 2

[38] Kailai Yang, Shaoxiong Ji, Tianlin Zhang, Qianqian Xie, and Sophia Ananiadou. On the evaluations of ChatGPT and emotion-enhanced prompting for mental health analysis. *arXiv preprint arXiv:2304.03347*, 2023. 2

[39] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024. 2

[40] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022. 1

[41] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao

Liang, and Yaodong Yang. Proagent: Building proactive cooperative ai with large language models. *arXiv preprint arXiv:2308.11339*, 2023. 1, 2

[42] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *IEEE International Conference on Computer Vision (ICCV)*, 2023. 6