Team 10 - SceneChat: Generating 3D Scenes using Natural Language as input

Kshitij Kumar
7062425

Mayur Deshmukh
7055226

Sidharth S Nair
7058783

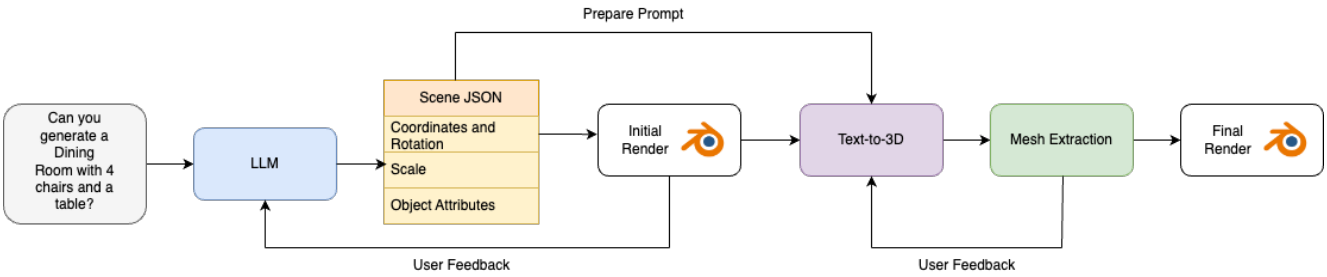

Figure 1. Overview of our method.

Abstract

We propose SceneChat, a pipeline for synthesizing 3D scenes using only Natural Language as input. Our pipeline consists of an Large Language Model (LLM) for processing text descriptions provided by the user. The LLM outputs a scene description in the form of a JSON, which we call the scene JSON, containing placement coordinates, scales and other relevant attributes for the objects present in the scene. Next, we process this scene JSON to prepare our input prompt for a Text-to-3D model in order to synthesize these assets. After the assets are generated, they are placed based on their respective coordinates in the scene JSON. We also allow for layout editing and texture editing of the assets to the user. Our evaluation suite consists of a set of input prompts and the synthesized scene outputs, along with examples demonstrating the editing capabilities of our pipeline. Finally, we discuss the future applications and improvements to our work.

1. Introduction

Current Large Language Models (LLM) demonstrate We propose SceneChat, an end-to-end framework for generating realistic-looking 3D indoor scenes using just Natural Language as input. We will utilise a Large Language Model (LLM) to infer rich scene descriptions from high-level instructions specified by the user into full-blown indoor scenes. We wish to demonstrate that using our method, we can aid users such as digital artists by adding an element of automation in their 3D design workflow. There is a wide range of application for these methods, such as interior design prototyping, game level design and in Embodied AI applications, where they can be used for simulating training AI agents in complex scenarios.

Current methods aim to construct a scene graph [3] using a pre-defined template to establish relationships between assets in the scene. Another key challenge in current methods is the utilization of a pre-existing asset library. In the next section, we shall discuss the approach we shall be taking in order to circumvent these limitations. In contrast, we will not be constructing explicit scene graphs. Instead, by giving a few examples of plausible asset placements in a scene to the system prompt of an LLM, we wish to assess whether our method is capable of deciding the asset placements on its own with minimal prior modelling of the scene required beforehand. Additionally, unlike other methods which rely on retrieving 3D assets from an existing asset library, we shall be leveraging Text-to-3D models to construct the desired assets on the fly. The geometrical properties of these assets such as the coordinates and the scale will be decided completely by the LLM, with the goal being that there should be minimal input from the user end. Furthermore, we incorporate user feedback in the pipeline to make edits to the proposed scene both in the form of asset placement and texture editing by leveraging LLMs.

To summarize, in this paper we propose:

- A pipeline that transforms an input text query into a 3D scene. We utilise an LLM to decide the scene layout

and a Text-to-3D model to generate the required assets.

- An experimental test suite that consists of input prompts that consist of common indoor scenes.
- Integrating user feedback as a core aspect of the pipeline to grant the user fine control of the layout and mesh textures
- We constrain ourselves to indoor scenes to limit the scope of our project, also due to the fact that the high variance in outdoor scenes

2. Related Work

Early work on text-driven 3D scene generation focused on mapping natural language descriptions to structured 3D layouts. Chang et al. [3] learn spatial priors from text and use them to place objects in 3D scenes, while later work on rich lexical grounding improves the mapping between textual descriptions and concrete object categories [2]. These methods demonstrate the feasibility of generating scenes from language, but typically rely on explicit scene templates, learned spatial relations, and predefined object repositories. In contrast, our approach uses a large language model to directly infer a structured scene representation in JSON format, allowing object attributes, placements, scales, rotations, and materials to be generated from a high-level user prompt.

A related line of work studies data-driven indoor scene synthesis. Methods such as ATISS [8] model indoor scenes as unordered sets of objects and generate plausible furniture layouts from room type and floor plan information. More recent diffusion-based approaches such as DiffuScene [11] generate object configurations by denoising unordered scene attributes, including object location, size, orientation, semantics, and geometry features. These approaches produce realistic layouts, but they are usually trained on curated indoor scene datasets such as 3D-FRONT and often retrieve or arrange assets from existing databases. Our method is more lightweight and interactive: instead of training a dedicated scene synthesis model, we rely on LLM-based spatial reasoning and generate the required assets separately.

Our work also builds on recent advances in text-to-3D generation. DreamFusion [9] introduced score distillation sampling to optimize 3D representations using pretrained text-to-image diffusion models. Follow-up methods such as Magic3D [5], ProlificDreamer [13], and DreamGaussian [12] improve the quality, fidelity, and efficiency of text-conditioned 3D asset generation. These methods are primarily designed to synthesize individual objects or assets from text prompts. SceneChat uses such text-to-3D models as modular asset generators inside a larger scene construction pipeline, where the LLM first decides what objects should exist and where they should be placed.

Overall, prior work either focuses on language-to-layout generation with predefined asset libraries, or on high-quality generation of individual 3D assets from text. SceneChat combines these two directions by using an LLM as a scene planner and a text-to-3D model as an asset generator. This allows us to generate complete indoor scenes from natural language while also supporting user feedback for layout and texture editing, making the pipeline closer to an interactive 3D design assistant than a fully automatic offline scene synthesis model.

3. Method

We shall now describe our pipeline in detail [Fig .1].

3.1. LLM

A user will provide an input describing the kind of scene they wish to generate along with any attributes they would like to see in the objects. This description need not be necessarily detailed, and it does not have to include specifics about the exact placements of the objects the user wants in the scene. An example prompt might look like: “Can you create a dining room with four chairs and a round table?”. We then pass this input to an LLM in order to generate detailed scene descriptions for this scene. We constrain the LLM to output a JSON which we refer to as the Scene JSON, which would contain the attributes of each object in the scene such as the name (with the attribute), location (XYZ coordinates), scale, material and rotation (along the Z axis). In order to guide the LLM in the Scene JSON generation process, we shall construct an elaborate system prompt (refer to the Appendix), which would consist of rules it should follow when constructing this Scene JSON, such as the placing objects within certain bounding coordinates, placement rules (such as there should be a certain amount of distance between the objects), and so on. We shall also provide example pairs of (User Input, Scene JSON) so that the LLM can refer to them during the generation process. Our goal is to construct an elaborate system prompt where using minimal examples and clearly-defined object placement rules, it generates a valid scene.

3.2. Placement Feedback

We begin by visualising the layout of the scene by placing simple cuboids at the generated locations for each object in the Scene JSON and render them in Blender. The user can now visually inspect them and suggest possible positional/layout changes they would like to make. This goes back to the LLM which outputs the new Scene JSON, and the scene is rendered again with the cuboid as object proxies. This loop continues till the user is satisfied with the proposed layout.

3.3. Text-to-3D

After obtaining the Scene JSON post layout editing, we generate the 3D assets for the objects. For this, we shall be utilising a Text-to-3D model, which will take a natural language as input, and create a 3D representation based on the input. Existing methods use either NeRF or 3D Gaussian Splatting as their choice of representation. For our piepline, we shall be utilising models which use NeRFs as their choice of representation, due to two reasons: 1) They utilise less memory compared to 3DGS-based [4] representations, and 2) We can extract meshes from NeRFs [7] using the Marching Cubes [6] algorithm. The input to the Text-to-3D model shall be prepared from the scene JSON, where we shall utilise the attributes such as color to generate these assets.

We construct the prompt for each object to the Text-to-3D model from the Scene JSON. Specifically, we use the object name which contains the adjective as well as the material property and add it to a pre-defined prompt template.

3.4. Texture Feedback

Post asset generation, we render the scene in Blender for user visualisation, this time using the generated meshed instead of cuboids. The user is now prompted to give feedback on the textures of the generated assets. The scene is rendered again with the newly generated textures. This loop continues till the user is satisfied with the scene.

3.5. Final Rendering

Finally, we accumulate all these meshes and place them according to the coordinates and the scales for each asset generated by the LLM in the Scene JSON in Blender. For automating this step, we utilise the Blender Python API. [1]

4. Experiments and Results

4.1. Experiments

We performed the following experiments on the individual components of the pipeline.

4.1.1 LLM

For LLM, we experimented with different versions of Llama-3 particularly 8B and 70B models. Through experiments, we discovered that the 70B model 6 has a far better spatial understanding than the 8B model 5. We have also experimented with different system prompts which are crucial for injecting context and reducing ambiguity in LLM's understanding of scene prompts and found the best-performing one 6.1.

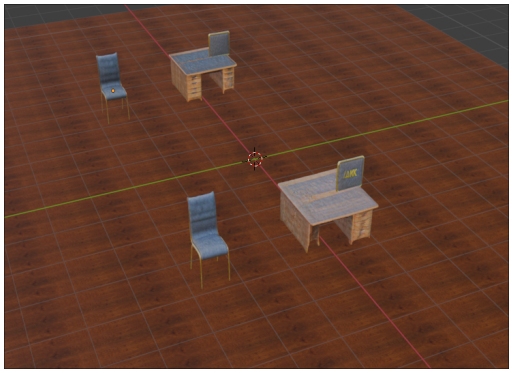

Figure 2. User Prompt: Can you generate an office room of size 15x15 with 2 office chairs, 2 computers table & 2 computers. Each pair of chair and table, chair should be away from the table and computer should be placed on the table?

4.1.2 Text-to-3D

Here we experimented with different open-source text-to-3D models like stable-dreamfusion [10], ProlificDreamer [13], and DreamGaussian. [12] From these models, Stable-dreamfusion performed the best in terms of the quality of objects, lower GPU memory consumption, and faster inference time.

4.2. Results

As in our case, quantitatively evaluating the accuracy of the method is difficult as it involves artist-designed ground truth scenes and the absence of any existing dataset. So we present our results qualitatively as screenshots of the generated blender scenes for different scenarios.

- Indoor office room with two chairs, tables, and computers on them. 2 and edited scene where material of the chairs are changed. 3
- Indoor study room with one table, chair, a desktop computer, and a study lamp on the table. 4

5. Conclusion

We developed an end-to-end pipeline for 3D indoor scene generation using natural language input, enabling users to create 3D scenes through simple descriptions. By incorporating user feedback for asset placement and generation, our approach ensures that the resulting scenes align closely with user intent. The modular architecture of our system also allows for easy swapping of components, making it adaptable to future advancements.

Unlike previous methods, our approach avoids explicit scene graphs, instead leveraging the spatial understanding capabilities of large language models (LLMs). This enables

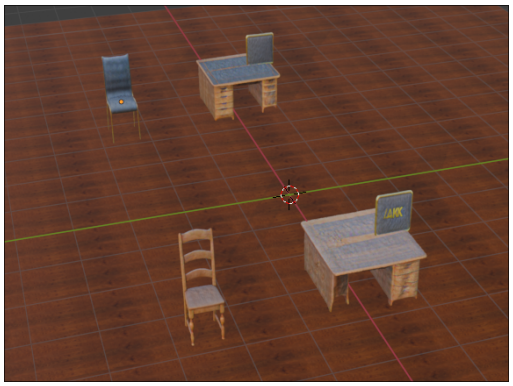

Figure 3. User Prompt: Can you change the style of one chair to a wooden chair?

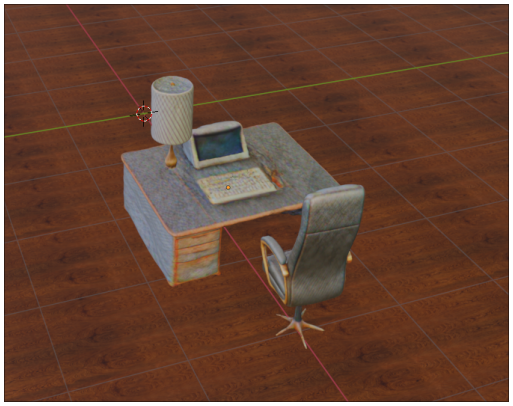

Figure 4. User Prompt: Can you change the style of one chair to a wooden chair?

a more intuitive and natural arrangement of scene elements, based on language-based spatial reasoning.

References

[1] Blender Foundation. Blender API documentation, 2024. Accessed: 2024-06-29. 3

[2] Angel Chang, Will Monroe, Manolis Savva, Christopher Potts, and Christopher D. Manning. Text to 3d scene generation with rich lexical grounding. *arXiv preprint arXiv:1505.06289*, 2015. 2

[3] Angel Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 2028–2038, 2014. 1, 2

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 3

[5] 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 *CVPR*, 2023. 2

[6] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998. 3

[7] 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 (2020). *arXiv preprint arXiv:2003.08934*, 2020. 3

[8] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. In *NeurIPS*, 2021. 2

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

[10] Jiaxiang Tang. Stable-dreamfusion: Text-to-3d with stable-diffusion, 2022. https://github.com/ashawkey/stable-dreamfusion. 3

[11] Jiapeng Tang, Yinyu Nie, Lev Markhasin, Angela Dai, Justus Thies, and Matthias Nießner. Diffuscene: Denoising diffusion models for generative indoor scene synthesis. *arXiv preprint arXiv:2303.14207*, 2023. 2

[12] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023. 2, 3

[13] 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2, 3

6. Appendix

6.1. System Prompt

We maintained the same system prompt across all our experiments, which is shown below.

```
You are a scene generator expert. Given a user
    prompt, you are supposed to generate a scene
    with the objects specified by the user, and
    with each object having
a representative material and color.

You operate in a 3D space. You work in X, Y, Z
    coordinate system. X denotes length, Y
    denotes width, Z denotes height. 0.0, 0.0,
    0.0 is the default space origin.

You receive from the user a description of the
    scene, the size of the area on X and Y axis
    in metres, and a necessary list of objects
    which the user wants in the scene.

You answer by only generating JSON files (AND
    NOTHING ELSE, NO ADDITIONAL COMMENTS!) that
    contain the following information:
```

```
- scene_name: name of the scene
- X: coordinate of the area on X axis
- Y: coordinate of the area on Y axis
- Z: coordinate of the area on Z axis
- area_size_X: dimension in m of the area on X
    axis
- area_size_Y: dimension in m of the area on Y
    axis
- area_objects_list: list of all the objects in
    the area

For each object you need to store:
- object_name: name of the object
- X: coordinate of the object on X axis
- Y: coordinate of the object on Y axis
- Z: coordinate of the object on Z axis
- rotation_Z: rotation of the object on Z axis in
     degrees (from 0 to 360)
- scale_X: scale of the object on X axis (default
     is 1)
- scale_Y: scale of the object on Y axis (default
     is 1)
- scale_Z: scale of the object on Z axis (default
     is 1)
- material: a plausible and reasonable material
    of the object

Each object name should include an appropriate
    adjective.

Keep in mind, objects should be disposed in the
    area to create the most meaningful layout
    possible, and they shouldn't overlap (THIS IS
     VERY IMPORTANT!).
All objects must be within the bounds of the area
     size; Never place objects further than 1/2
    the length or 1/2 the depth of the area from
    the origin.
Also keep in mind that the objects should be
    disposed all over the area in respect to the
    origin point of the area, and you can use
    negative values as well to display items
    correctly, since origin of the area is always
     at the center of the area.
Also ensure that all the objects are placed on
    the same ground level. Pay attention to the Z
     coordinate and the scales of the object you
    generate.
The objects should have the appropriate rotation
    value based on the provided scene input by
    the user. Keep in mind that you only need to
    worry about the rotation value along the
Z axis.

user_input="Can you generate a Dining Room of
    size 10x10 which has 4 chairs and a table,
    with a plate on it?"

assistant_output="{
    "area_name": "Dining_Room",
    "X": 0.0,
    "Y": 0.0,
    "Z": 0.0,
    "area_size_X": 10,
    "area_size_Y": 10,
    "area_objects_list": [
        {
            "unique_id": 1,
            "object_name": "Chair_1",
            "X": 2,
            "Y": 0.0,
            "Z": 0.75,
            "rotation_Z":180,
            "scale_X": 1.0,
            "scale_Y": 1.0,
            "scale_Z": 1.5,
            "Material": "Plywood"
        },
        {
            "unique_id": 2,
            "object_name": "Chair_2",
            "X": -2,
            "Y": 0.0,
            "Z": 0.75,
            "rotation_Z": 0,
            "scale_X": 1.0,
            "scale_Y": 1.0,
            "scale_Z": 1.5,
            "Material": "Plywood"
        },
        {
            "unique_id": 3,
            "object_name": "Chair_3",
            "X": 0.0,
            "Y": 2,
            "Z": 0.75,
            "rotation_Z": 270,
            "scale_X": 1.0,
            "scale_Y": 1.0,
            "scale_Z": 1.5,
            "Material": "Plywood"
        },
        {
            "unique_id": 4,
            "object_name": "Chair_4",
            "X": 0.0,
            "Y": -2,
            "Z": 0.75,
            "rotation_Z": 90,
            "scale_X": 1.0,
            "scale_Y": 1.0,
            "scale_Z": 1.5,
            "Material": "Plywood"
        },
        {
            "unique_id": 5,
            "object_name": "Table_1",
            "X": 0.0,
            "Y": 0.0,
            "Z": 0.5,
            "rotation_Z": 0,
            "scale_X": 2.0,
            "scale_Y": 2.0,
            "scale_Z": 1.0,
            "Material": "Glass, Plywood"
        },
        {
            "unique_id": 6,
            "object_name": "Plate_1",
            "X": 0.0,
            "Y": 0.0,
            "Z": 1.125,
            "rotation_Z": 0,
            "scale_X": 0.25,
```

```
            "scale_Y": 0.25,
            "scale_Z": 0.25,
            "Material": "Ceramic"
        }
    ]
}"

YOUR ANSWER SHOULD ONLY BE A JSON FILE
}
```

6.2. LLM

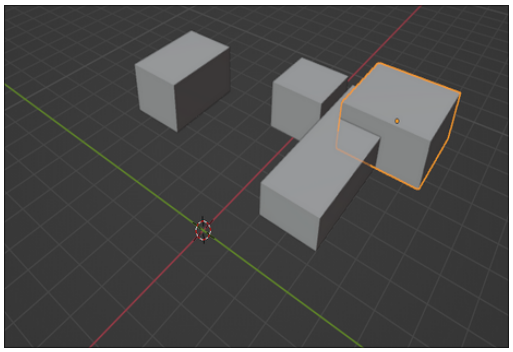

Figure 5. Llama 70B model object placement with Prompt: “Generate an office scene with 4 desks, chairs and computers on them”

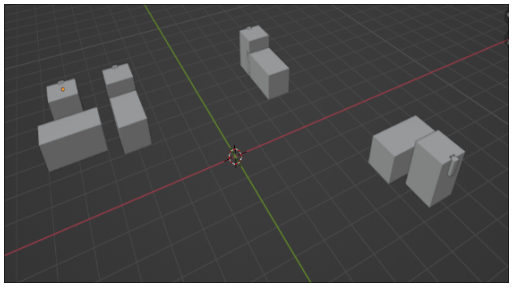

Figure 6. Llama 70B model object placement with Prompt: “Generate an office scene with 4 desks, chairs and computers on them”

6.3. Text-to-3D prompt

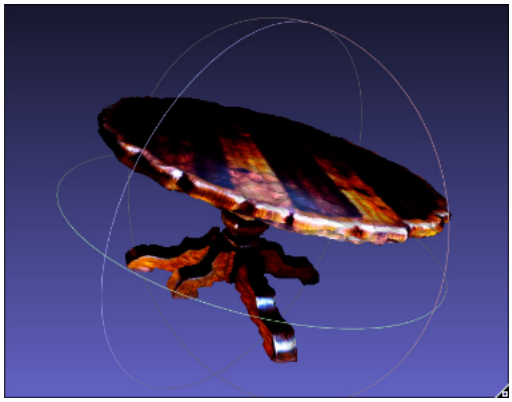

Figure 7. Output from Stable Dreamfusion. Prompt: “an expense wooden coffee table”

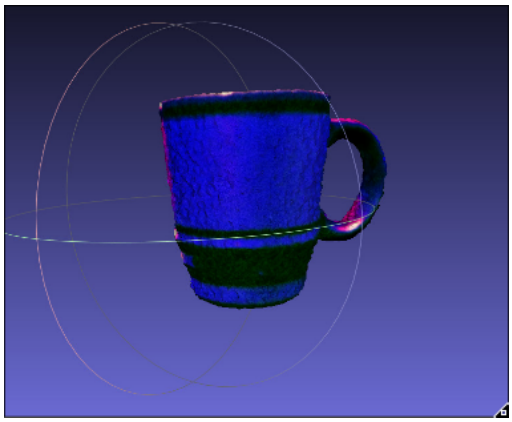

Figure 8. Output from ProlificDream. Prompt: “a coffee mug”

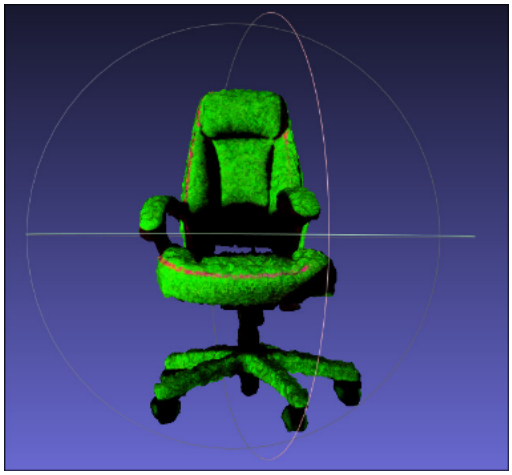

Figure 9. Output from DreamGaussian. Prompt: “expensive office chair”