

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

Anonymous 3DV submission

Paper ID 112

We kindly request the reader to consider visiting following website to view our high-quality 3D results:  
<https://anonymous0888.github.io/3DGPT/3dgpt.html>.

## 1. Comparison with Text-to-3D

We offer side-by-side comparisons with a state-of-the-art Text-to-3D method Dreamfusion [1] for single objects and scene, as illustrated in Figure 2 and Figure 1, respectively.

## 2. Implementation Detail

We offer an illustrative example of the prompt used for Adding Trees (Figure 6) and Sky Modeling (Figure 8), presenting the Document, Code, Information, and a Usage Example.

We provide the code that enumerates the functions within Infinigen used in constructing our 3D scene. While a similar Python file can be found at <https://github.com/princeton-vl/infinigen/blob/main/worldgen/generate.py>, our script is uniquely tailored to work with a control dictionary generated by three agents, enabling controllable scene generation.

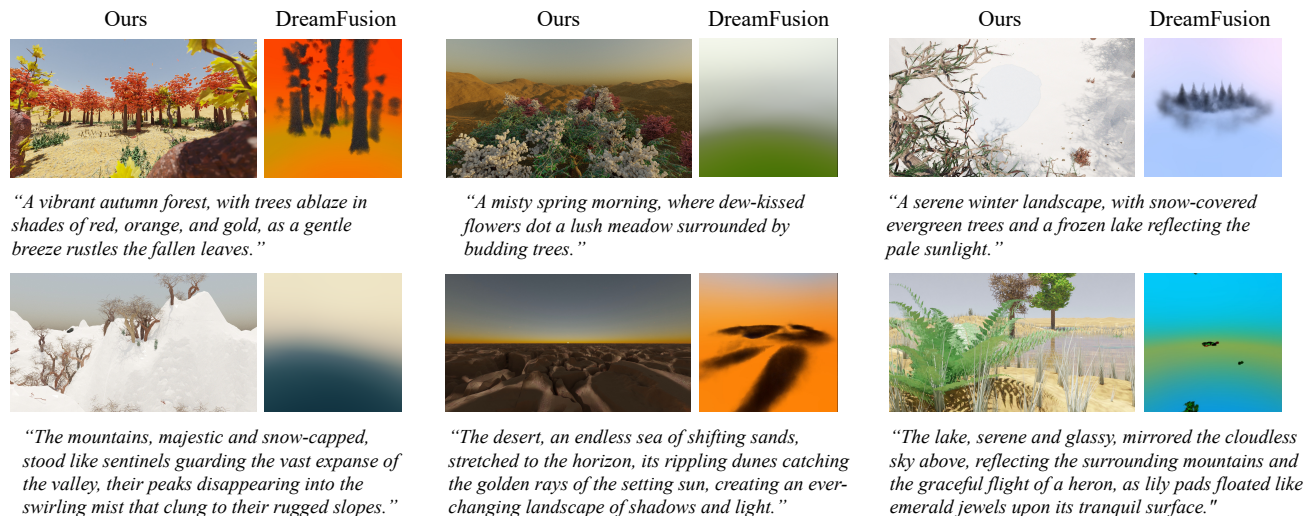
```
def compose_scene(terrain, control_dict={}):  
    """  
    Give a base terrain and control dictionary,  
    build the 3D scene.  
    -----  
    terrain: Empty mesh  
    control_dict: control dictionary generated by  
    the agents that use to control each function  
    parameters.  
    Returns  
    -----  
    """  
    add_coarse_terrain(terrain, control_dict)  
    add_trees(control_dict)  
    add_bushes(control_dict)  
    add_clouds(control_dict)  
    add_boulders(control_dict)  
    add_glowing_rocks(control_dict)  
    add_kelp(control_dict)  
    add_cactus(control_dict)  
    add_rocks(control_dict)  
    add_ground_leaves(control_dict)
```

```
add_ground_twigs(control_dict)  
add_chopped_trees(control_dict)  
add_grass(control_dict)  
add_monocots(control_dict)  
add_ferns(control_dict)  
add_flowers(control_dict)  
add_corals(control_dict)  
add_leaf_particles(control_dict)  
add_rain_particles(control_dict)  
add_dust_particles(control_dict)  
add_marine_snow_particles(control_dict)  
add_snow_particles(control_dict)  
  
return
```

Listing 1. Agent Implementation Example

We offer the code to implement the modeling agent as a demonstration of how to utilize the OpenAI API for implementing our agent.

```
def modeling_function_call(text_description,  
    function_description, function,  
    function_document,  
    example, max_tokens=2000, temperature=0.3,  
    history=[]):  
    """  
    Give a short text, call the given functions  
    to generation objects/scene to fit the given  
    text description  
    Parameters  
    -----  
    text_description: short user given text.  
    function_description: short function  
    description.  
    function: python code.  
    function_document: the detail description  
    of the function.  
    example: example of how to use the function  
    .  
    max_tokens: max tokens for the detailed  
    text.  
    temperature  
    Returns  
    -----  
    response from the agent that contains the  
    function calls.  
    """  
    if len(history) == 0:
```



```

20     history = [
089         {"role": "system", "content": "You are a
090 good 3D designer who can convert long text
091 descriptions into parameters, and is good at
092 understanding Python functions to manipulate
093 3D content. "},
094     ]
22     messages = history
23
24
25     text = f"""We have the following function
098 codes {function_description} to control
099 blender by python : {function}.
100
26     Following are the document for function:
101 {function_document}.
102
27     Below is an example bout how to make
103 function calls to model the scene to fit the
104 description: {example}.
105
28     Question: Given the text description:
106 {text_description}    analysis the function
107 parameter and call the function to {
108 function_description}"""
109
29     messages.append({"role": "user", "content":
110 text})
111
30     # define modeling agent
112     modeling_agent = openai.ChatCompletion.
113 create(
114         model = "gpt-3.5-turbo",
115         temperature = temperature,
116         max_tokens = max_tokens,
117         messages = messages
118     )
119
37     # get the response from modeling agent
120     function_call = modeling_agent.choices[0].
121 message["content"]
122
39     return function_call, messages

```

### 3. Ablation Study Details

We conduct separate ablation studies for the Conceptualization Agent and Task Dispatch Agent, evaluating their performance based on CLIP scores, failure rates, and parameter diversity.

We use the implementation of [2] to calculate CLIP score. It measures cosine similarity in the CLIP hidden space, indicating the alignment between generated images and text descriptions. For 3D scene CLIP scores, we placed a camera at the scene’s center, capturing 50 images by rotating the camera 360 degrees. In the Conceptualization Agent evaluation, we used only the initial instruction. For the Task Dispatch Agent, both the initial instruction and one subsequence instruction were used.

The failure rate reflects the Modeling Agent’s response. Failures can occur when the method cannot extract the correct pattern via the parser, generates data with an incorrect datatype, or omits/adds parameters to the function call. We evaluated five functions for each scene, recording the failure rate. We also assessed 500 parameters per scene, employing the Shannon Diversity Index to measure parameter diversity, and categorizing float parameters into 100 bins.

## 4. Agent Responses

Figure 3, Figure 4 and Figure 5 demonstrate how the system communicates with Task Dispatch Agent, Conceptualization Agent, and Modeling Agent, respectively. The Task Dispatch Agent example is deliberately simplified, featuring only four functions to ensure conciseness. In contrast, the examples for the Conceptualization Agent and Modeling Agent are derived from our comprehensive implementation.

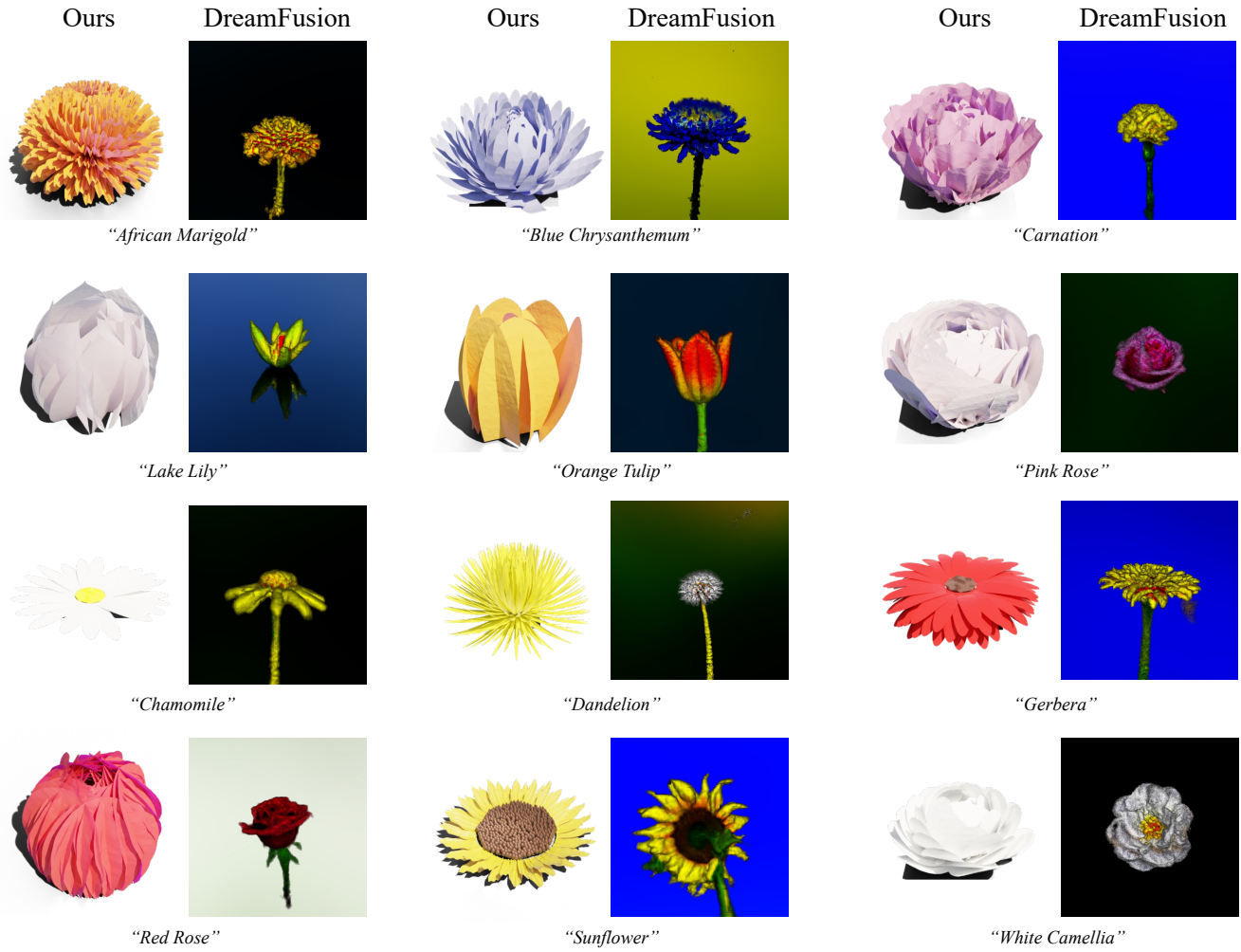


Figure 2. Comparison with Dreamfusion (Single Object).

## 5. Visual Result

Figure 6 and Figure 7 show the prompts for modelling trees and sky. The visual control result for a full scene is illustrated in Figure 8.

You are a proficient planner for selecting suitable functions based on user instructions. You are provided with the following functions:

1. `generate_terrain()`: model the 3D terrain.
2. `generate_tree()`: generate trees with parameters that controls the tree appearance and place them on the terrain surface.
3. `add_grass()`: add grass to the terrain surface()
4. `apply_snow_layer()`: add snow to the surface of all objects.

Below are examples of how to choose functions based on user instructions:

User: My instruction is: "Change weather to snowy." Answer: Based on the function provided above and the user instruction, the `apply_snow_layer()` function from above function set is suitable to call. It will add snow to all objects covered by snow and make it look like a snowy scene. Other functions (1,2,3) are not relevant to the user instruction. Given the instructions, we determine the sublist of functions that need to be used for 3D modeling: [4]

User: My instruction is: "Change tree leaf to yellow."



Based on the user instruction, "Change tree leaf to yellow," you can use the `generate_tree()` function to generate trees with yellow leaves. You can modify the parameters of the `generate_tree()` function to control the appearance of the trees, including the color of their leaves. Other functions (1, 3, 4) are not relevant to changing the color of tree leaves. Given the instructions, we determine the sublist of functions that need to be used for 3D modeling:

[2]

So, the suitable function to call in this case is `generate_tree()` with the appropriate parameters for yellow leaves.

Figure 3. Communication Between System and Task Dispatch Agent.

Given a text description "A peaceful clearing in a dense forest, blanketed in fresh, untouched snow. Tall birch trees, their white bark standing out starkly against the dark evergreens and the gray, overcast sky, form a natural cathedral.", provide detailed descriptions for the following information: "possible daytime, sun intensity, air condition, sky appearance, cloud density.".

For term not mentioned in the description, use your imagination to ensure they fit the text description.



In this peaceful clearing, the scene is bathed in the soft glow of early morning light. The sun, just beginning to rise above the horizon, casts a gentle, golden hue across the landscape. Its intensity is subdued, as if still gathering strength to break through the thick canopy of trees that surround the clearing.

The air in the clearing is crisp and cool, carrying with it a hint of winter's bite. It is still and calm, devoid of any breeze or gusts that might disturb the tranquility of the scene. The air feels pure and invigorating, as if it has been untouched by human presence.

Looking up towards the sky, the overcast conditions create a muted, gray backdrop. The clouds hang low, their edges blending seamlessly with the horizon, creating a sense of enclosure within the clearing. The sky appears vast, yet somehow intimate, as if it is a part of the natural cathedral formed by the towering birch trees.

The cloud density is moderate, with the clouds spread out across the sky in a gentle, billowing fashion. They are not heavy or oppressive, but rather wispy and ethereal, adding a touch of mystery to the scene. As the day progresses, the clouds shift and change shape, creating a dynamic and ever-evolving canvas above the clearing.

Overall, the daytime in this clearing is characterized by a serene ambiance, with the sun's gentle rays filtering through the dense forest and the sky adorned with a delicate veil of clouds. It is a scene that invites contemplation and reflection, a sanctuary of tranquility amidst the wild beauty of nature.

Figure 4. Communication Between System and Conceptualization Agent.

We have the following function codes modelling the sky appearance using nishita method to control blender by python :

```
'''python
import nishita_sky_modelling
def sky_texture_node(sun_intensity,sun_elevation,sun_rotation,air_density,dust_density,ozone,cloud_density):
    texture = nishita_sky_modelling(sun_intensity,sun_elevation,sun_rotation,air_density,dust_density,ozone,cloud_density)
    return
'''
Following are the document for function: "
input:
sun_intensity: Multiplier for sun disc lighting. (choose from 'low','median','high')
sun_elevation: Rotation of the sun from the horizon (in degrees). (0:sunset,sun rising, 90:daytime,-10:night)
sun_rotation: Rotation of the sun around the zenith (in degrees).
air_density: density of air molecules. (0 no air, 1 clear day atmosphere, 2 highly polluted day)
dust_density: density of dust and water droplets. (0 no dust, 1 clear day atmosphere, 5 city like atmosphere, 10 hazy day)
ozone: density of ozone molecules; useful to make the sky appear bluer. (0 no ozone, 1 clear day atmosphere, 2 city like atmosphere).
higher value for bluer sky.
cloud_density: density of the cloud, varying from 0 to 0.04. (0.01 very thick cloud, 0.04 very heavy cloud)
output: texture color output."
```

Below is an example bout how to make function calls to model the scene to fit the description:

Question: given the text description of the scene: "The river, reflecting the clear blue of the sky, glistened like a silver ribbon as it wound its way through the lush valley, its tranquil waters whispering secrets to the ancient trees." analysis the function parameter and call the function to generate the sky.

Solution: From the description of "clear blue sky", the sun\_intensity can not be low, let's set it as "median";As the sky is blue, the sun\_elevation cannot be very low (not morning/sunset/evening), we can set it to 50.

The sun rotation does not affect the sky appearance, let's set it to 0. To fit the description of "clear blue". The air\_density and dust\_density should be low, we can set air\_density to 1, dust\_density to 0. To make the sky more blue, we can increase the ozone value, let's make it as 2.

As the sky is very blue and clear, the cloud density should be low, we can set it as 0.0005. We can model the sky by calling the following function:

```
'''python sky_texture_node(sun_intensity = 'median',sun_elevation=50,sun_rotation=0,air_density=1,dust_density=0,
ozone=2,cloud_density=0.0005)'''
```

Question: Given the text description: "A peaceful clearing in a dense forest, blanketed in fresh, untouched snow. Tall birch trees, their white bark standing out starkly against the dark evergreens and the gray, overcast sky, form a natural cathedral. In this peaceful clearing, the scene is bathed in the soft glow of early morning light. The sun, just beginning to rise above the horizon, casts a gentle, golden hue across the landscape. Its intensity is subdued, as if still gathering strength to break through the thick canopy of trees that surround the clearing. The air in the clearing is crisp and cool, carrying with it a hint of winter's bite. It is still and calm, devoid of any breeze or gusts that might disturb the tranquility of the scene. The air feels pure and invigorating, as if it has been untouched by human presence.

Looking up towards the sky, the overcast conditions create a muted, gray backdrop. The clouds hang low, their edges blending seamlessly with the horizon, creating a sense of enclosure within the clearing. The sky appears vast, yet somehow intimate, as if it is a part of the natural cathedral formed by the towering birch trees.

The cloud density is moderate, with the clouds spread out across the sky in a gentle, billowing fashion. They are not heavy or oppressive, but rather wispy and ethereal, adding a touch of mystery to the scene. As the day progresses, the clouds shift and change shape, creating a dynamic and ever-evolving canvas above the clearing.

Overall, the daytime in this clearing is characterized by a serene ambiance, with the sun's gentle rays filtering through the dense forest and the sky adorned with a delicate veil of clouds. It is a scene that invites contemplation and reflection, a sanctuary of tranquility amidst the wild beauty of nature." analysis the function parameter and call the function to modelling the sky appearance using nishita method.



Based on the description, we can analyze the function parameters as follows:

- sun\_intensity: The description mentions a "soft glow of early morning light" and the sun's intensity is described as "subdued". Let's set the sun\_intensity to 'low'.
- sun\_elevation: The sun is described as "just beginning to rise above the horizon". Let's set the sun\_elevation to -5.
- sun\_rotation: The description does not provide any specific information about the sun's rotation, so we can set it to 0.
- air\_density: The air is described as "crisp and cool" and "untouched by human presence". Let's set the air\_density to 0.
- dust\_density: The description does not mention any dust or particles in the air, so we can set the dust\_density to 0.
- ozone: The description does not provide any specific information about the ozone density, so we can set it to 1.
- cloud\_density: The clouds are described as "moderate" and "spread out across the sky in a gentle, billowing fashion". Let's set the cloud\_density to 0.02.

We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=-5, sun_rotation=0, air_density=0, dust_density=0, ozone=1, cloud_density=0.02)
'''
```

Figure 5. Communication Between System and Modeling Agent.

**Document:**

**Explanation:** The function takes a built natural scene as input and adds base trees to it.

**Inputs:**

scene: The built natural scene.

density(float): The density of the trees.

distance\_min(float): Minimum distance between trees.

leaf\_type(string): The type of leaf on the tree. Select one from the list ['leaf', 'leaf\_broadleaf', 'leaf\_ginko', 'leaf\_maple', 'flower', 'None'].

'Leaf' allows further custom settings like changing the leaf shape and color. 'Leaf\_broadleaf', 'leaf\_ginko', and 'leaf\_maple' build the leaves with predefined shapes: broad, ginko, and maple respectively. 'Flower' will create flowers on the tree instead of leaves. 'None' will not generate leaves on the tree.

fruit\_type(string): The type of fruit on the tree. Select one from the list ['apple', 'blackberry', 'coconut\_green', 'durian', 'starfruit', 'strawberry', 'custom\_fruit', 'None']. Each of the fruit types will create the corresponding fruit on the tree. 'Custom\_fruit' can create customizable fruit on the tree with further adjustments. If the desired fruit is not on the list, 'custom\_fruit' should be chosen. 'None' will not generate any fruit on the tree.

**Code:**

```
'''python
import TreeFactory
def add_trees(scene, density, distance_min, leaf_type, fruit_type, leaf_density):
    trees = TreeFactory.create(density, leaf_type, fruit_type, leaf_density) # create trees
    scene.placement(trees, distance_min) # place trees in the scene
    return
'''
```

**Information:**

"Possible trees in the scene, tree types, leaf density, possible seasons, tree size, appearance of the tree branch, branch type, and leaf color."

**Example:**

**Question:** Given the text description of the scene, analyze the function parameters and call the function to add more trees that fit the text description.

**Solution:** From the text description, three different types of trees are identified in the scene: birch, dogwood, and Japanese maple. From "a cluster of," we infer the tree density is relatively high; let's set it to 0.2. Birch tree leaves are typically oval or triangular, so let's set the leaf type to 'leaf' to allow further customization. There are no fruits mentioned for the birch tree in the description. Given the scene likely depicts autumn, the leaf\_density should not be too high; let's set it to 0.02. For the dogwood tree, as the leaves are described as oval or elliptical with smooth edges, the leaf type is set to 'leaf'. It is mentioned that "clusters of small, vibrant berries cling to its branches," so we can choose the fruit\_type as 'blackberry'. The leaf\_density is set to 0.02. For the Japanese maple tree, the leaf type is set to 'leaf\_maple'. No fruits are mentioned for the maple tree. As the scene likely takes place in autumn, the leaf density for the maple tree should be high; let's set it to 0.1. The function can be called three times to create three different trees:

```
'''python
add_trees(scene, density=0.2, distance_min=1, leaf_type='leaf', fruit_type='None', leaf_density=0.02) # create birch trees
add_trees(scene, density=0.1, distance_min=1, leaf_type='leaf', fruit_type='blackberry', leaf_density=0.02) # create dogwood trees
add_trees(scene, density=0.1, distance_min=1, leaf_type='leaf_maple', fruit_type='None', leaf_density=0.1) # create Japanese maple trees
'''
```

Figure 6. Prompt Example of Adding Trees.



**Document:****Input:**

sun\_intensity: Multiplier for sun disc lighting. (Choose from 'low', 'median', 'high')

sun\_elevation: Rotation of the sun from the horizon (in degrees). (0: sunset, sun rising, 90: daytime, -10: night)

sun\_rotation: Rotation of the sun around the zenith (in degrees).

air\_density: Density of air molecules. (0: no air, 1: clear day atmosphere, 2: highly polluted day)

dust\_density: Density of dust and water droplets. (0: no dust, 1: clear day atmosphere, 5: city-like atmosphere, 10: hazy day)

ozone: Density of ozone molecules; useful to make the sky appear bluer. (0: no ozone, 1: clear day atmosphere, 2: city-like atmosphere). A higher value yields a bluer sky.

cloud\_density: Density of the clouds, ranging from 0 to 0.04. (0.01: very thick cloud, 0.04: very heavy cloud)

**Output:**

texture\_color\_output: The output texture color of the sky.

**Code:**

```
'''python
import nishita_sky_modelling
def sky_texture_node(sun_intensity, sun_elevation, sun_rotation, air_density, dust_density, ozone, cloud_density):
    texture = nishita_sky_modelling.model_sky(sun_intensity, sun_elevation, sun_rotation, air_density, dust_density, ozone, cloud_density)
    return texture
'''
```

**Information:**

The time of day depicted in the scene (morning/noon/evening), the possible season, the condition of the air, the blueness of the sky, and the cloud density.

**Example:****Question:**

Given the text description of the scene: "The river, reflecting the clear blue of the sky, glistened like a silver ribbon as it wound its way through the lush valley, its tranquil waters whispering secrets to the ancient trees," analyze the function parameters and call the function to generate the sky.

**Solution:**

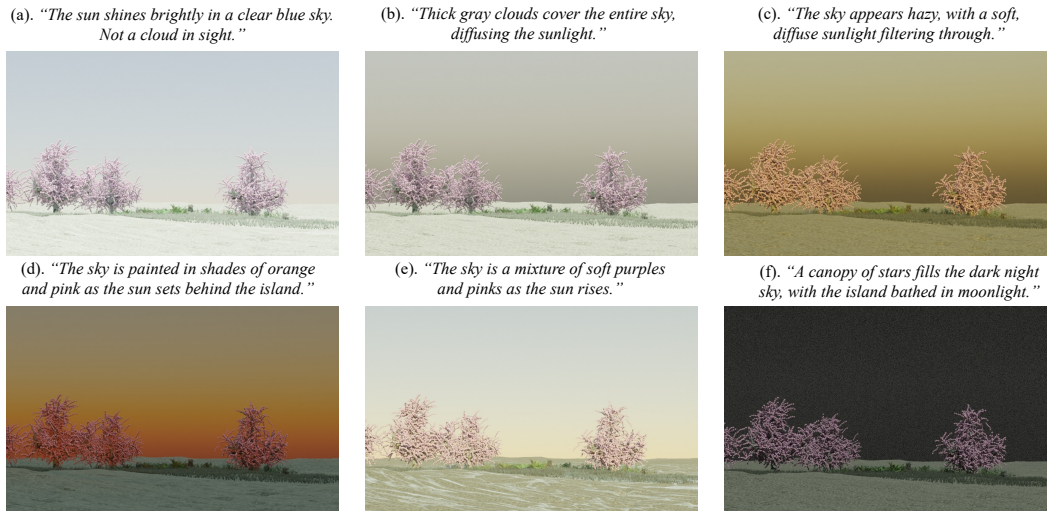
From the description of a "clear blue sky," we infer that sun\_intensity cannot be low, let's set it as "median". As the sky is clear and blue, the sun\_elevation must not be very low, indicating it's not morning or evening; we can set it to 50 degrees.

Sun rotation does not impact the appearance of the sky significantly; thus, we will set it to 0 degrees. To align with the description of "clear blue," we should choose lower values for air\_density and dust\_density. Hence, we will set air\_density to 1 and dust\_density to 0. To enhance the blueness of the sky, we can increase the ozone value to 2.

As the description suggests a clear, blue sky, the cloud density must be low. Therefore, we will set it to 0.0005. The sky can be modeled by invoking the function as follows:

```
'''python
sky_texture_node(sun_intensity='median', sun_elevation=50, sun_rotation=0, air_density=1, dust_density=0, ozone=2, cloud_density=0.0005)
'''
```

Figure 7. Prompt Example of Sky Modeling Function.



(a). **Solution:** From the description of "clear blue sky" and "not a cloud in sight", we can infer that the sun\_intensity should be high, the sun\_elevation should be around 90 (daytime), the sun\_rotation can be set to 0. The air\_density and dust\_density should be low, so we can set them to 1 and 0 respectively. Since there are no clouds, the cloud\_density should be 0. To make the sky appear bluer, we can increase the ozone value, so let's set it to 2. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='high', sun_elevation=90, sun_rotation=0, air_density=1, dust_density=0, ozone=2, cloud_density=0)
'''
```

(b). **Solution:** From the description of "Thick gray clouds cover the entire sky", the sun\_intensity should be low to account for the diffused sunlight. Let's set it as "low". The sun\_elevation should be high to indicate that it is daytime. Let's set it to 90. The sun\_rotation does not affect the sky appearance, so we can set it to 0. The air\_density and dust\_density should be high to represent the presence of thick clouds. Let's set air\_density to 2 and dust\_density to 5. Since the sky is covered with thick gray clouds, the ozone value can be low. Let's set it to 0. As mentioned, the entire sky is covered with thick gray clouds, so the cloud\_density should be high. Let's set it to 0.04. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=90, sun_rotation=0, air_density=2, dust_density=5, ozone=0, cloud_density=0.04)
'''
```

(c). **Solution:** From the description of "hazy sky" and "soft, diffuse sunlight", we can infer that the sun\_intensity should be low. Let's set it as "low". The sun\_elevation should be low as well, indicating a time close to sunrise or sunset. Let's set it to -10. The sun\_rotation does not affect the sky appearance, so we can set it to 0. To fit the description of a hazy sky, the air\_density and dust\_density should be higher. Let's set air\_density to 2 and dust\_density to 5. The ozone value can be set to 1 to maintain a natural sky color. Since the sky is hazy, the cloud\_density should be higher as well. Let's set it to 0.02 to represent a moderately cloudy sky. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=-10, sun_rotation=0, air_density=2, dust_density=5, ozone=1, cloud_density=0.02)
'''
```

(d). **Solution:** From the description of "shades of orange and pink", the sun\_intensity can be set as "low" to create a softer and warmer lighting. As the sun is setting, the sun\_elevation can be set to 0. The sun\_rotation does not affect the sky appearance, so it can be set to 0. To fit the description of "shades of orange and pink", the air\_density and dust\_density should be low, so we can set air\_density to 1 and dust\_density to 0. To create a warmer and more vibrant sky, we can set the ozone value to 1. As the sun is setting, there might be some clouds in the sky, so we can set the cloud\_density to 0.01. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=0, sun_rotation=0, air_density=1, dust_density=0, ozone=1, cloud_density=0.01)
'''
```

(e). **Solution:** From the description of "soft purples and pinks", the sun\_intensity can be set to "low" to create a softer lighting effect. Since the sun is rising, the sun\_elevation can be set to a low value, such as 10. The sun\_rotation does not affect the sky appearance, so it can be set to 0. To create a purple and pink sky, the air\_density and dust\_density can be set to low values, such as 1 and 0. The ozone value can be set to 0 to avoid making the sky blue. Since the sky is described as a mixture of colors, the cloud\_density can be set to a low value, such as 0.001. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=10, sun_rotation=0, air_density=1, dust_density=0, ozone=0, cloud_density=0.001)
'''
```

(f). **Solution:** From the description of "dark night sky" and "canopy of stars", we can assume that the sun is not visible in the sky. Therefore, the sun\_intensity can be set to "low". Since it is a night scene, the sun\_elevation can be set to -10. The sun\_rotation does not affect the sky appearance, so it can be set to 0. As it is a dark night sky, the air\_density and dust\_density can be set to 0. The ozone value can also be set to 0 as there is no mention of blueness in the description. Since there are no clouds mentioned, the cloud\_density can be set to 0. We can model the sky by calling the following function:

```
'''python
sky_texture_node(sun_intensity='low', sun_elevation=-10, sun_rotation=0, air_density=0, dust_density=0, ozone=0, cloud_density=0)
'''
```

Figure 8. **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.



## References

- [1] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. *arXiv preprint arXiv:2209.14988*, 2022. [1](#)
- [2] 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](#)