INTEGRATING CHATGPT WITH A-FRAME FOR USER-DRIVEN 3D MODELING

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Masters of Science

By

Ivan Hernandez

May 2024

The Designated Project Committee Approves of the Project Titled

Integrating ChatGPT with A-Frame for User-Driven 3D Modeling

by

Ivan Hernandez

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2024

ABSTRACT

Integrating ChatGPT With A-Frame For User-Driven 3D Modeling

by Ivan Hernandez

ChatGPT is a large language model that is capable of creating conversational text and functional code that can be integrated into various technologies, including computer graphics software. Currently, 3D modeling applications can be relatively difficult for novices to learn and understand due to the overwhelming amount of graphical user interfaces. However, we can remedy this issue by leveraging ChatGPT's conversational language capabilities. Our project described in this report integrates ChatGPT with A-Frame, an online framework for developing virtual reality experiences, to create an immersive and user-friendly 3D modeling environment where users can create and modify 3D models through natural language prompts. We performed extensive testing across different temperature values that determine how repeatable ChatGPT's outputs are. The most reliable outcomes were achieved with temperatures of 0.5 and 1.0. We discovered that the quality of the outputs usually depended on the complexity of the given prompts. This meant that simpler commands produced more precise and relevant results in comparison to complex commands.

Index terms **– ChatGPT, A-Frame, Three.js, Virtual Reality, 3D modeling, Code Generation, Audio Commands**

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my project advisor, Chris Pollett, for his continuous support and guidance throughout the development of this project over the last two semesters. His knowledge and feedback have been invaluable in shaping the essence of this project and I am grateful for his mentorship. I would also like to extend my gratitude to my committee members, Genya Ishigaki and Kevin Smith, as well as the rest of the faculty members in the Department of Computer Science at San Jose State University for their efforts and insights that have helped me grow over my graduate journey. Finally, I also want to thank my family and peers for their constant motivation and encouragement throughout my academic experience.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Throughout the last couple of years, we have seen the emergence of a new large language model known as ChatGPT. ChatGPT is especially renowned for its natural conversational capabilities but has many other use cases. One impressive application of ChatGPT is code generation. This model can generate code for different programming languages [1]. From this, we can see its potential to be used not only as a chatbot or conversational model but also as a tool in software applications. Integrating it into applications like 3D modeling software could change the 3D modeling and scene building process by lowering the need for graphical user interfaces and by providing a conversational agent that can handle natural language commands. This leads to the main idea of our project, which is to merge ChatGPT's capabilities with A-Frame, an online tool that uses HTML and JavaScript to construct Virtual Reality (VR) scenes [2], to create a more interactive and easy-to-use 3D modeling environment. Our aim is to develop a system where ChatGPT is part of an A-Frame VR setting so that users can generate or alter 3D models through natural language descriptions and voice commands. This way the system can offer a real-time and user-driven experience for users to guide and modify the 3D modeling process.

For this project, we looked at the influence of the 'temperature' variable, which is an important element of ChatGPT. This variable can be set between the range of 0.0 and 2.0 and is responsible for the randomness or creativity of the responses generated by the GPT model. For example, if we have a low temperature value, like 0.2, the model will produce predictable and consistent results. This is crucial for a system that needs to maintain accuracy and consistency with 3D model generations. A higher temperature value will increase the randomness and enable the system to explore a wider range of modeling possibilities.

Furthermore, for this project, we wanted to create a system that accurately interprets the user's intents. Therefore, we conducted a lot of testing with various temperature values, ranging from 0.0 to 2.0, to determine which values created the best outcomes. This allowed us to properly gauge how the temperature variable affects the relevance and accuracy of ChatGPT's responses in a virtual reality environment.

This report will be divided into four main sections. In the upcoming section, Chapter 2, we discuss the background information regarding the different technologies used in this project, particularly A-Frame and ChatGPT. Chapter 3 presents the preliminary work that was achieved during the first semester of the project, which contains information about the setup of the virtual environment and initial ChatGPT integration with A-Frame. The following section, Chapter 4, discusses the implementation of the main deliverables achieved during the second semester of the project. Chapter 4 also includes the specific approach, implementation details, and results for each of the deliverables. Finally, Chapter 5 outlines the testing approach along with the metrics used to evaluate the system. This chapter visualizes the testing results with a series of graphs, which will be analyzed to determine how temperature affected the accuracy and relevance score of the 3D models. To conclude, we will provide a project summary and discuss future developments.

CHAPTER 2

BACKGROUND

In this chapter, we discuss the background information that is needed to understand the project. First, we cover the related work. Then, the next sections of this chapter discuss the two major technologies used in this project, A-Frame and ChatGPT. Each section defines the key attributes of the particular technology and explains how they contribute to the project's objectives.

2.1 Related Work

Currently, there is limited academic research that evaluates the results of combining A-Frame and ChatGPT for 3D modeling. Despite ChatGPT becoming a popular platform, its integration into applications like A-Frame has not been well documented in academic or industry research as of now. However, there have been some videos online demonstrating personal projects combining the use of A-Frame and ChatGPT. For instance, Takashi Yoshinaga, an AR/VR engineer, posted a video online demonstrating a basic collaboration between the ChatGPT API and A-Frame, which was able to generate simple 3D models through the use of a graphical user interface [3].

Moreover, considering that for this project we are generating content using artificial intelligence, there has been research related to evaluating the quality of generated content, but for 2D instead of 3D. There has been significant research that assesses the quality of 2D generated images using models like Midjourney, Luna, and DALLE-2. In [4], a comprehensive evaluation of multiple image generation models, including Midjourney and DALLE-2, is provided. The authors use their own dataset, called AVT-AI-Image-Dataset, to serve as a baseline to assess the quality, appeal, and realism of the 2D generated images [4]. They collected the ratings from a group of 25 participants and determined that both Midjourney and DALLE-2 provided the best results in terms of appeal and quality. Moreover, the authors of [5] performed a qualitative comparison between the 2D images generated by DALLE-2 and Luna. They point out that due to the subjective nature of art, measuring the quality of the images relied on the opinions of the testers as opposed to common metric calculations. After gathering the results from their testing surveys, they found that DALLE-2 outperformed Luna in both alignment and fidelity, meaning that the users preferred DALLE-2's outputs and deemed them more accurate than the 2D images generated with the Luna model [5].

2.2 A-Frame

One of the main technologies that we used for this project is called A-Frame. A-Frame is an online framework that is designed to create immersive VR experiences that can run on web browsers. It provides developers with a powerful yet accessible tool to create VR environments without having extensive knowledge of VR programming or plugins. A-Frame scenes can be developed by using its markup language that resembles HTML, thus making it simple to create 3D objects with HTML-like tags that are particular to A-Frame. For example, users can create primitive models, such as boxes and spheres, by using the built-in '<a-box>' and '<a-sphere>' tags, respectively. This approach makes it more accessible to developers who want to build simple VR scenes but might not have the general knowledge or experience required by other VR applications, such as Unity3D and Unreal Engine.

In terms of key attributes, A-Frame, similar to Unity3D, has an entity-component system (ECS). This means that every object in the A-Frame scene is an entity, and any entity can have components that define its appearance and functionality. Components are blocks of JavaScript code that the developer can program for advanced functionality or behavior. This allows users to not only extend functionality for their entities but also have reusability, where components can serve as modular pieces of code that can be applied to other entities. Moreover, A-Frame is built on top of the three.js JavaScript API, which handles a significant portion of the WebGL rendering and mathematics used in A-Frame. The three.js library provides the foundation for A-Frame's built-in 3D primitive objects and their geometric appearance. Therefore, if the developer wants to have custom geometry or advanced functionality in their scene, they would need to program it in a custom A-Frame component and then attach that component to an existing A-Frame entity.

In the context of this project, A-Frame serves as the environment or virtual space in which we can create, modify, and remove 3D objects. It also becomes the framework that allows us to add functionality to entities so that we can interact with them using the VR headset controllers. This technology allows for the creation of an immersive VR environment that we can design to have no graphical user interfaces for user input, thus reducing the barrier to entry and developing a system that is more accessible and intuitive for general users. Instead of user interfaces, we can implement voice commands that are properly processed and recognized by the system. Users can then simply describe their design intentions in natural language through voice commands, and the system will transform these commands into actionable A-Frame 3D modeling instructions that are then visualized and updated in the A-Frame scene. This demonstrates the ideal or practical application that the combination of A-Frame and artificial intelligence can provide to enhance the user experience during the 3D modeling process.

2.3 ChatGPT

The other major technology that we used for this project was ChatGPT. ChatGPT, developed by OpenAI, is a large language model that is capable of performing natural language processing tasks, ranging in complexity from summarization to long context-aware conversations. OpenAI released ChatGPT in late 2022, and it quickly gained popularity among the general public. Individuals have used it for many different purposes. Generally, it has been used to answer questions in human-like text and to maintain long conversations with users. However, its capabilities are not limited to just being a chatbot. It has been trained with massive amounts of online data and can generate content that people may not be aware of. For instance, it can generate music, which is represented in text as chords or bar notation. The quality of this type of content generation from a large language model will vary, as presented in [6], but it demonstrates the potential of these models as they continue to improve over time. Moreover, ChatGPT can also generate programming code. Given a programming task, it is capable of generating code that implements the specified functionality. However, it can also end up providing responses with poor quality, since the code might have errors or not follow the best guidelines. Nonetheless, ChatGPT can be used for various purposes and its capabilities will only continue to improve as OpenAI works on the next GPT version.

Regarding its structure, ChatGPT is based on the GPT series. Generative Pre-Trained Transformer, or GPT, is a language model developed by OpenAI. As indicated by its name, ChatGPT maintains a transformer architecture that consists of an encoder-decoder and a self-attention mechanism [1]. Over the last couple of years, the GPT model has gone through several iterations, and is currently, as of writing this report, in the model version called GPT-4 Turbo. This version has a 128k context window, allowing it to keep track of very long back-and-forth conversations. It also has a more recent cutoff in knowledge, with training data that extends up to the end of 2023. The current free version for ChatGPT is based on the GPT3.5 Turbo model, which only has a context window of about 16k tokens and training information that goes up to only the end of 2021.

In the context of this project, we use ChatGPT for the purpose of natural language processing. Specifically, we leverage ChatGPT's capabilities to understand context-aware user prompts so that it can generate A-Frame code that accurately interprets and reflects the intent of the user's commands. In order to get the best results, we use the latest version of ChatGPT, which is the GPT-4 Turbo model. The model's capability to understand everyday language makes it incredibly useful to create a user-friendly 3D modeling application, that would otherwise need specific commands or user interfaces to create 3D objects. This type of functionality streamlines the 3D modeling process and makes it more intuitive for general users who might not be familiar with traditional 3D user interfaces. Furthermore, combining this technology with A-Frame, which provides an immersive virtual environment, offers a different approach to 3D modeling and design that has not been explored extensively. Allowing individuals to use natural language commands to create and modify 3D models provides a user-driven modeling approach that shifts focus from traditional 3D modeling tools to a more user-friendly and self-controlled design process.

CHAPTER 3

PRELIMINARY WORK

In this chapter, we discuss the key parts of the project that were established in the first semester. This includes four main deliverables that focused on developing the working foundation for the system. These four deliverables helped achieve the following goals: set up the A-Frame virtual environment, establish a simple communication between A-Frame and the ChatGPT API, set up the proper integration of ChatGPT for generating A-Frame code, and recognize and process voice commands.

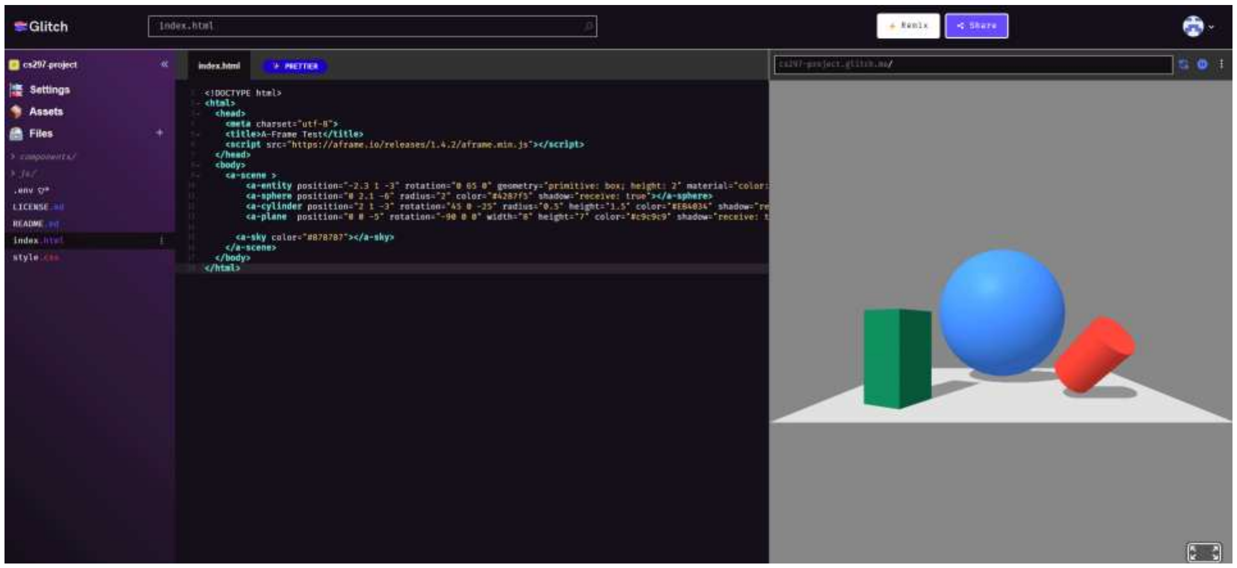
3.1 Glitch and A-Frame Setup

The first goal was to create a working VR environment using A-Frame and to test simple 3D primitive creation in the scene. After that, we performed a basic comparison between the workflows of A-Frame and Unity to see how A-Frame's 3D object positioning and manipulation stack up against other 3D applications.

Glitch (https://glitch.com) is an online platform that provides hosting and a code editor for A-Frame scenes. So, we used the Glitch editor to create and host a simple VR A-Frame scene that features three different geometric primitives. This basic A-Frame scene is demonstrated in Figure 1. For this initial deliverable, we also explored the workflow differences between A-Frame and Unity3D. The comparison entailed creating a basic virtual scene with various primitives in both applications. We used the following tools to set up these scenes: A-Frame with HTML/JavaScript, Glitch for hosting, Unity3D with C#, and the Oculus Quest 2 for testing.

First, we built the scene in A-Frame by creating '<a-entity>' tags for each object and adjusting their position within the scene. This workflow required consistent adjustments between the 3D view and HTML code to get everything to be positioned correctly. We added textures to have more realism in the scene and made the door interactive within the VR environment. After creating the virtual scene in A-Frame, we then created the same scene in Unity3D, following a similar process as before. First, we created and adjusted the locations of the objects in the editor. Then, we applied the textures to the objects and added basic door functionality.

The results are shown in Figure 2. From these images, we can notice that the scenes are very similar, with the main difference being in how each software handles its lighting. Our experience confirmed what others have found, as mentioned in [2], where adjusting objects in A-Frame ends up being inefficient and tedious due to the constant need to tweak their attributes in HTML. In contrast, Unity has a visual editor that provides useful widgets to manipulate objects, making it much easier to work with. For texturing, the process was straightforward in both Unity and A-Frame. Lastly, since both Unity and A-Frame have an entity-component system architecture, the functionality for the door worked similarly in both environments.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>A-Frame Test</title>
    <script src="https://aframe.io/releases/1.4.2/aframe.min.js"></script>
  </head>
  <body>
    <a-scene >
        <a-entity position="-2.3 1 -3" rotation="0 65 0" geometry="primitive: box; height: 2" material="color:
        <a-sphere position="0 2.1 -6" radius="2" color="#4287f5" shadow="receive: true"></a-sphere>
        <a-cylinder position="2 1 -3" rotation="45 0 -25" radius="0.5" height="1.5" color="#EB4034" shadow="re
        <a-plane  position="0 0 -5" rotation="-90 0 0" width="8" height="7" color="#c9c9c9" shadow="receive: t

      <a-sky color="#878787"></a-sky>
    </a-scene>
  </body>
</html>
```

Fig 1. Glitch online editor and simple A-Frame scene.

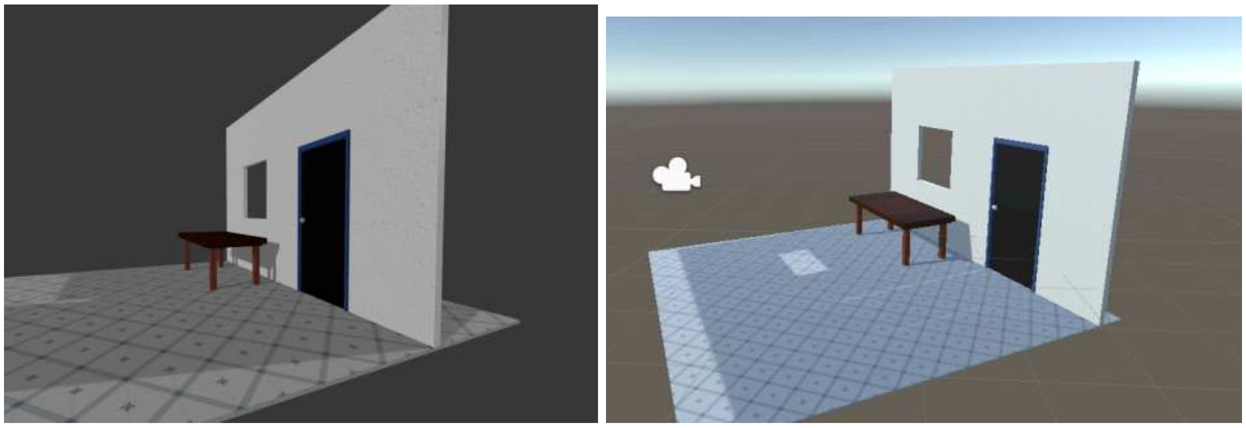


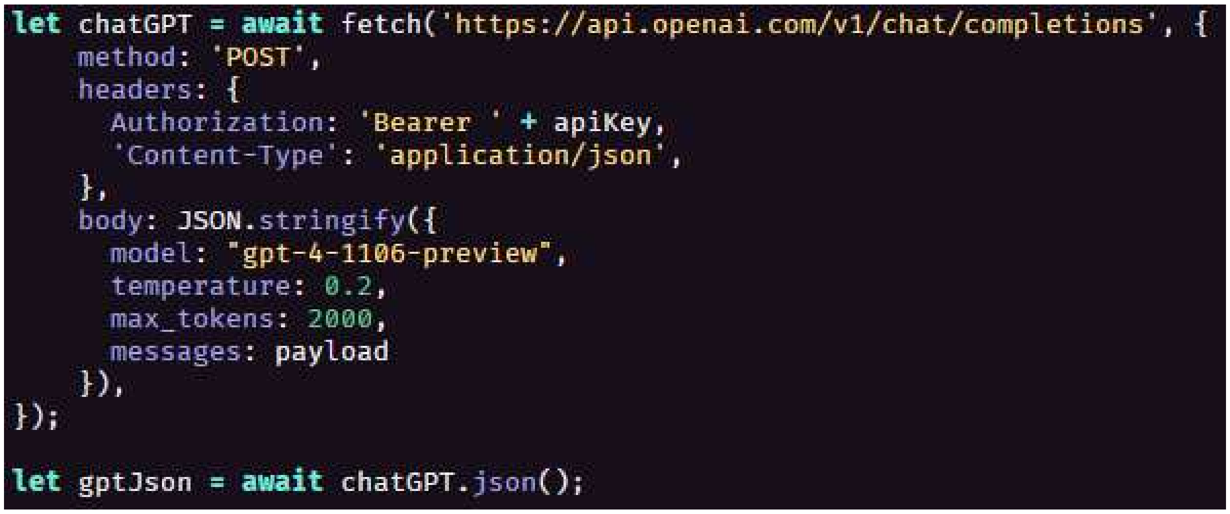
Fig 2. Basic virtual scene in both A-Frame(left) and Unity3D(right)

3.2 ChatGPT API Communication

For this section, we integrated a general ChatGPT API with the A-Frame environment. To enable simple interaction with the ChatGPT API, we set up a basic user interface that provided text fields to enter user prompts and receive ChatGPT's responses. Although we added voice commands in a later deliverable, this initial user interface setup was useful for our tests.

We explored the different models provided by OpenAI along with the costs for using these models. Once we had an OpenAI API key, we could use not only their GPT models but also their audio models for the voice commands. To communicate with their GPT model, we made sure that our simple user interface could handle the user requests that are sent to the GPT API.

With the OpenAI API key, we implemented a function in which we send a POST request to the endpoint ‘https://api.openai.com/v1/chat/completions’. Figure 3 demonstrates the code for these API calls. This figure also shows the specific GPT model that we use along with the rest of the model parameters, such as the “temperature” and ‘max_tokens’ parameters. As mentioned before, a low ‘temperature’ means that the model will be more consistent and predictable. The ‘max_tokens’ parameter indicates the maximum number of tokens or words that the model can provide in its response. We used the GPT-4 Turbo model, which offers a much higher context window and costs less than earlier models. Figure 3 also presents the ‘messages’ parameter, which captures the full conversation between the user and ChatGPT that we send in the POST request. To keep the full context of a given conversation, we ensure that the 'payload' variable records all messages, including both the prompts we send and ChatGPT's replies. Finally, we receive ChatGPT's response and extract the JSON file to display the response on our user interface.



```
let chatGPT = await fetch('https://api.openai.com/v1/chat/completions', {
    method: 'POST',
    headers: {
      Authorization: 'Bearer ' + apiKey,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      model: "gpt-4-1106-preview",
      temperature: 0.2,
      max_tokens: 2000,
      messages: payload
    }),
});

let gptJson = await chatGPT.json();
```

Fig 3. Post request to OpenAI's ChatGPT model

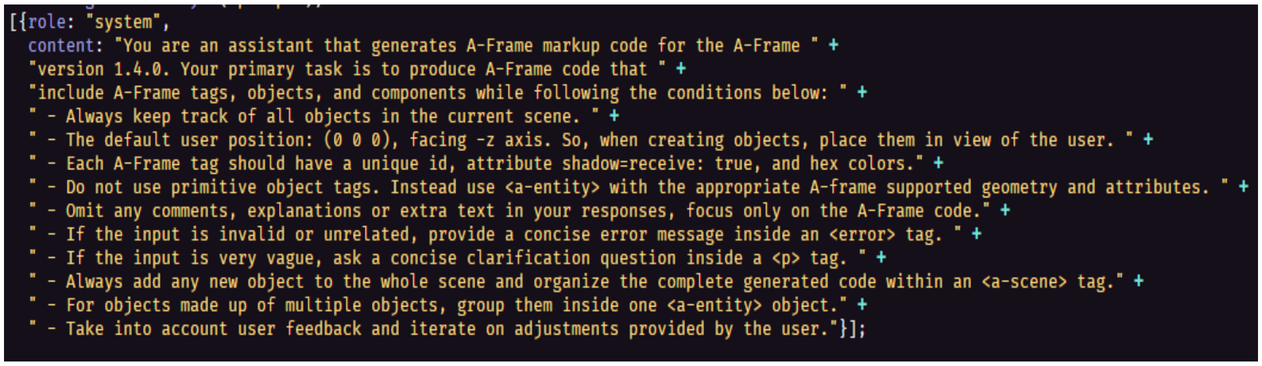
3.3 ChatGPT Modeling Interpreter

After successfully linking A-Frame and ChatGPT, we moved to translating user prompts into specific commands for A-Frame. This step included setting up a command structure for adding, altering, and deleting objects to ensure that the user inputs were properly interpreted and turned into modeling commands.

OpenAI allows users to customize how the system should behave by setting specific rules. In Figure 4, we can see the set of guidelines that we used for our system during the first semester. These instructions helped us develop a specialized artificial intelligence assistant tailored to generate A-Frame code based on a given request. We designed these rules to direct ChatGPT to focus on producing A-Frame code that is specifically inside the '<a-scene>' tags while ignoring any extra text that is typically generated for added context. Also, some rules give spatial information to ChatGPT, such as “The default user position: (0 0 0), facing -z axis. So, when creating objects, place them in view of the user,” to make sure that ChatGPT knows the orientation of the user within the A-Frame environment. We also set rules for ChatGPT to identify invalid and vague commands so that these commands are handled properly and displayed as a warning or clarification message in the user interface. Once we receive the properly formatted responses, we use a custom function to extract the code using regex strings and then embed it into the main ‘<a-scene>’ tag to update the A-Frame scene according to the user’s modifications.

Furthermore, with these guidelines, we were able to build an A-Frame assistant that can generate results for both simple and complex prompts. For instance, simple commands like “Create a red cube,” were handled well and accurately. More complex commands often required further adjustments from the user. Figure 5 includes results from complex commands. This figure shows how well the system understands concepts and objects, such as distinguishing a couch from its differently colored cushions. The commands we used to construct the UFO seen in Figure 5 were: “Create a cool UFO in the scene” and “Add buttons to the inside and outside of the UFO.” In Figure 6 we can see sequential snapshots of a UFO animation, for which we prompted the system to animate the UFO flying up into the sky away from the user. The results were quite impressive, showing that ChatGPT not only recognized that the UFO should be moving away from the user but also rotate as it was flying away, indicating an interesting understanding of how a UFO might behave.

While the system performed well with complex commands, we also tested how consistently ChatGPT could create A-Frame code from simple commands. In [7], the authors conducted tests to assess ChatGPT’s consistency across three different consistency types: negation, semantic, and symmetric. We used the same consistency types for our tests and discovered that all of the tests for negation consistency passed, similar to the findings in [7]. As the authors noted, ChatGPT is very accurate at determining negation and contradictions. For the other consistency types, the system generally performed well, though there was one test case for each type where ChatGPT was inconsistent. For semantic consistency, there was a test case with inputs that had the same meaning but were phrased differently, and ChatGPT incorrectly marked them as two different inputs. Similarly, for the symmetric consistency results, one of the tests was labeled inconsistent when its sequence of commands strongly influenced ChatGPT's outputs. Overall, ChatGPT performed relatively well at delivering consistent outcomes for simple modeling prompts.



```
[{role: "system",
  content: "You are an assistant that generates A-Frame markup code for the A-Frame " +
  "version 1.4.0. Your primary task is to produce A-Frame code that " +
  "include A-Frame tags, objects, and components while following the conditions below: " +
  " - Always keep track of all objects in the current scene. " +
  " - The default user position: (0 0 0), facing -z axis. So, when creating objects, place them in view of the user. " +
  " - Each A-Frame tag should have a unique id, attribute shadow=receive: true, and hex colors." +
  " - Do not use primitive object tags. Instead use <a-entity> with the appropriate A-frame supported geometry and attributes. " +
  " - Omit any comments, explanations or extra text in your responses, focus only on the A-Frame code." +
  " - If the input is invalid or unrelated, provide a concise error message inside an <error> tag. " +
  " - If the input is very vague, ask a concise clarification question inside a <p> tag. " +
  " - Always add any new object to the whole scene and organize the complete generated code within an <a-scene> tag." +
  " - For objects made up of multiple objects, group them inside one <a-entity> object." +
  " - Take into account user feedback and iterate on adjustments provided by the user."}];
```

Fig 4. Specific instructions given to ChatGPT to specialize its system

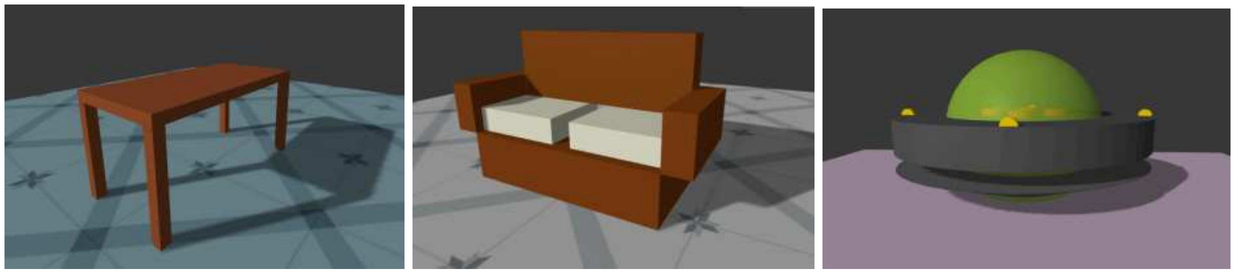


Fig 5. Results from complex input tests

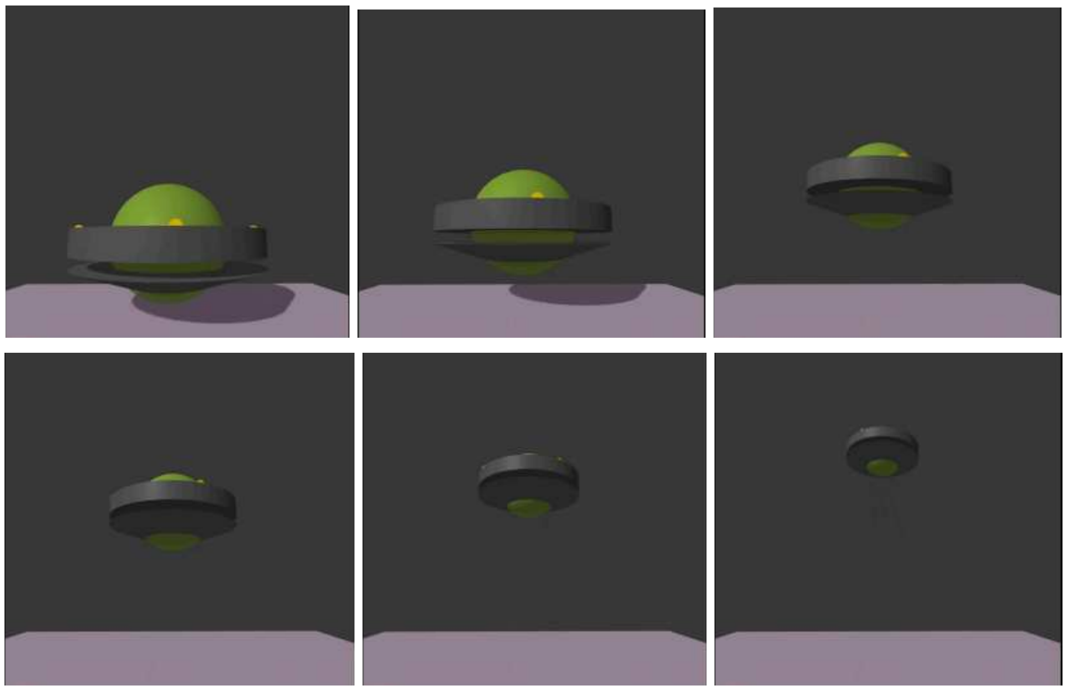


Fig 6. Sequential snapshots of UFO animation in A-Frame

3.4 Voice Commands

In this section, we implemented voice interactions into the A-Frame VR scene. For our project, voice commands are meant to replace user interfaces and make the experience more user-friendly. This task included the following features: voice recognition, speech-to-text transcription, integration with ChatGPT, and vocal feedback responses. These features allowed us to optimize user communication and enhance the experience within the VR environment.

For managing voice interactions, we utilized OpenAI's audio models, specifically the Whisper and Text-To-Speech models. The Whisper model is designed to transcribe audio into text. For our project, we used it to transcribe captured speech from the Oculus Quest 2 headset microphone. We recorded the audio and then processed it before sending it to the Whisper API for transcription. Then, we gave the transcribed text to ChatGPT and updated the A-Frame scene based on ChatGPT's responses. Additionally, we implemented vocal feedback after the user makes a command. For this feedback, we used the Text-To-Speech model, which is another audio model provided by OpenAI. After users issue a command, they hear preset audio responses, such as “Processing request,” confirming that their command is being processed.

Moreover, we configured the controls for capturing audio with the Oculus Quest 2 headset. This included setting the ‘A’ and ‘B’ Oculus Touch buttons to start and stop recording, respectively. We also added a small microphone icon to show when the system is recording the user’s audio. The audio was saved as a .wav file and sent to the Whisper model endpoint via a POST request. The model transcribed the audio to text, which we then passed to ChatGPT. Figure 7 illustrates this process, which is similar to the steps followed in [8], except that in our project we use ChatGPT for natural language processing within the A-Frame VR environment. Not shown in the diagram is the immediate use of OpenAI’s TTS model after receiving voice commands and ChatGPT’s responses.

Figure 8 displays video snapshots from a test run of this system. We used the Oculus Quest 2 headset to record the video and audio for this test. The specific commands used were: “Add a light blue floor to the scene” and “Add a light red box to the scene, and make it spin constantly.” These commands were quickly transcribed and sent to ChatGPT. From these images, we can see how the system operates within the virtual environment, allowing users to provide modeling commands without needing a keyboard or traditional user interface.

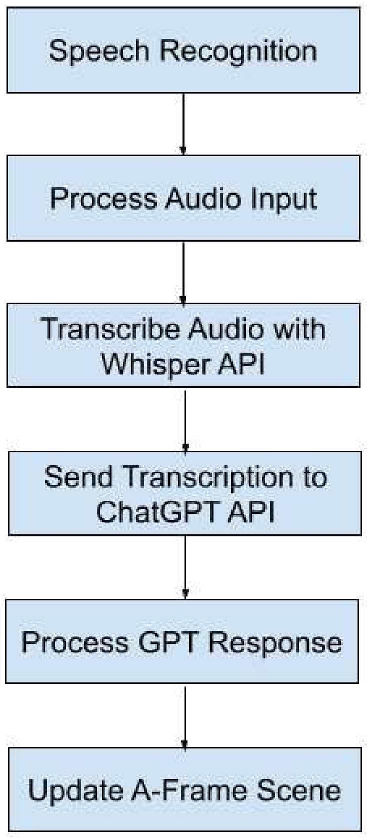


Fig 7. Process diagram of an example test run

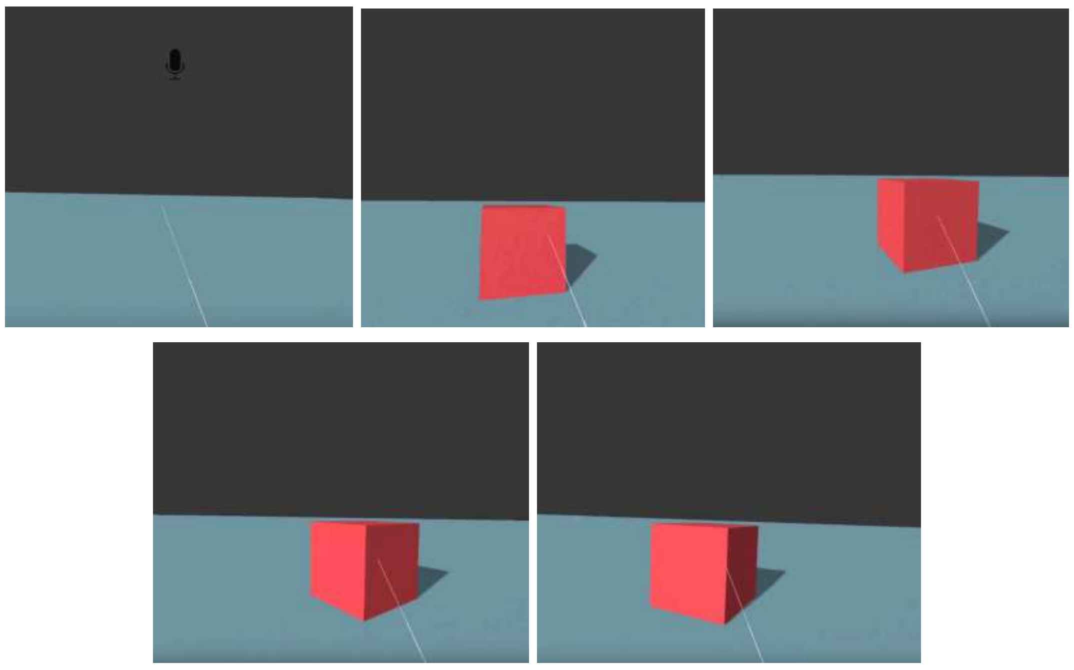


Fig 8. Sequential snapshots of virtual reality video test

CHAPTER 4

IMPLEMENTATION

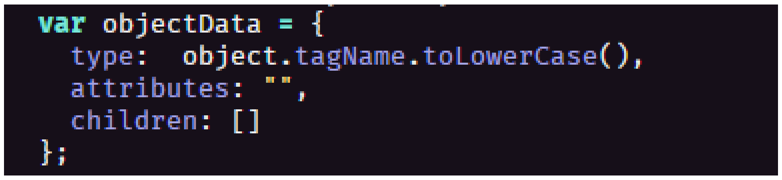
In this chapter, we discuss the implementation and functionality introduced in the second semester of the project. We go through three deliverables that focus on improving the accuracy and reliability of the system. The first deliverable concentrated on implementing the serialization and deserialization of A-Frame scenes so that users can save a 3D scene and return to it in a later session. The second deliverable extended the basic functionality of the project so that we could generate entities that used custom functionality as opposed to the default and built-in components. Finally, the third deliverable focused on extending the custom geometry in our system. This improvement allowed us to take advantage of complex three.js classes as well as make sure that our system remained updated with the latest three.js standards.

4.1 Serialization

The goal for the first deliverable was to capture or serialize a given A-Frame scene into a JSON file. This deliverable also includes a deserialization function that reconstructs or loads the serialized A-Frame scene back into the 3D modeling environment. Through voice commands, the user will be able to easily save their modeling work during a session and reload it in future sessions.

Serialization is a critical component in the 3D modeling process. It enables developers to save their current 3D modeling environment, capturing all of the details they have made during a session, and storing it somewhere in their device. Then, in a later session, through deserialization, they can load their saved scene and continue working on it. For these significant reasons, we decided to implement a serialization and deserialization mechanism into our project.

For serialization of an A-Frame scene, we convert each entity in the scene into a JSON object and then aggregate all objects and save them into a JSON file. Figure 9, demonstrates the structure of the JSON object for each entity. It records its tag, which is generally an '<a-entity>' tag, along with all of its attributes, such as position, rotation, and A-Frame components. Also, there could be nested entities or scene objects that are made up of multiple entities in our scene. Therefore, we iterate through all of the entity's children and serialize those entities as well in a recursive manner.



```
var objectData = {
  type:  object.tagName.toLowerCase(),
  attributes: "",
  children: []
};
```

Fig 9. JSON object format for each entity

For deserialization, we load an A-Frame scene from a JSON file. This JSON file, as mentioned above, is composed of multiple JSON objects. We read the file and extract all JSON objects for further processing. Each object is reconstructed into an '<a-entity>' and added to the current scene. This results in the successful deserialization of the JSON file and an updated A-Frame scene.

Moreover, we updated the system instructions to include support for handling serialization through voice commands. A user can simply say "Save this scene as basic scene" and the system will recognize the appropriate action and return a flag to A-Frame, where we handle serialization of the current scene. The flag is returned in the form of an HTML tag. ChatGPT will generate an '<s>' or '<l>' tag for saving or loading, respectively. Also, if provided by the user, the system will return the filename and storage location for the JSON file. It supports saving and loading from the computer's disk and from the local browser storage. After we receive the ChatGPT tag containing the necessary information, we parse the response and proceed accordingly to either save or load the JSON file. In general, we were able to make the saving functionality work with voice commands. However, in the virtual environment, for loading a scene, the user needs to press the ‘X’ button on the Oculus Touch Controller for this feature to work. This was due to a browser restriction that prevented the loading of a file without the use of a button or gesture interaction. Nevertheless, the system provides users with the capability to save and load their desired A-Frame scenes.

4.2 Custom A-Frame Components

For the second deliverable, we created a system that constructs custom A-Frame components based on user input. Components provide the appearance, behavior, and functionality of entities in A-Frame. Thus, generating custom A-Frame components that are interpreted from user commands would likely enhance the capabilities of our system. This deliverable also includes a parser that validates the generated A-Frame components to make sure that the components are dynamically executed and appropriately integrated with the A-Frame environment.

At this point in the development of our project, the options for functionality or behavior of entities were very limited to default and built-in A-Frame components. This includes the simple animation component that is in A-Frame, which provides very basic animation trajectories or modifications to the scene entities. However, we want to develop a system that can generate A-Frame components that provide more advanced and accurate behavior or functionality for the objects in the scene. This means that, on top of generating A-Frame HTML code, we need a system that generates functional JavaScript code for A-Frame components to correctly interpret the user’s intentions.

A-Frame components are blocks of JavaScript code that add functionality to A-Frame entities. They have a structure that includes member variables and functions that are triggered or executed to handle particular lifecycle events, such as the ‘Tick()’ function that is executed every single frame of the render loop. Since we want ChatGPT to generate these A-Frame components, we updated the system rules with a new instruction, which specifies the structure of an A-Frame component so that ChatGPT understands its particular format and standardizes the generated components. Furthermore, we provide several more instructions that focus on fine-tuning the system so that it understands that for advanced functionality, it needs to generate a custom A-Frame component as opposed to using default or built-in components. We also indicate that the formatted code has to be generated inside a '<js>' HTML custom tag so that we can parse the generated response and extract the JavaScript code by using regex strings. After extracting the code, we use the 'eval()' JavaScript function to dynamically execute the A-Frame component. Since an A-Frame component cannot be executed more than once, we keep track of the existing components inside a list so that we can check if a component with the same name already exists in our system.

With this added functionality, we improve the reliability and relevance of the generated ChatGPT results. By equipping ChatGPT with the ability to handle broader commands, some of which might be more advanced, we create a system that can process user commands with better accuracy. This in turn reduces the chances of the system failing at generating something and increases the likelihood of generating something that is more relevant to the user's commands. Figure 10 shows a test result from this deliverable, where we can see a sequence of images depicting several spheres orbiting around a main red sphere, with each orbiting sphere having different speeds and trajectory paths.

We also needed to update our serialization mechanism to save and load these components. Therefore, on top of transforming entities into JSON objects, we also transform all existing custom components into JSON objects and include them in the JSON file. When loading a JSON file, we extract all JSON objects, update the scene with the loaded entities, and dynamically execute every custom component with the 'eval()' function.

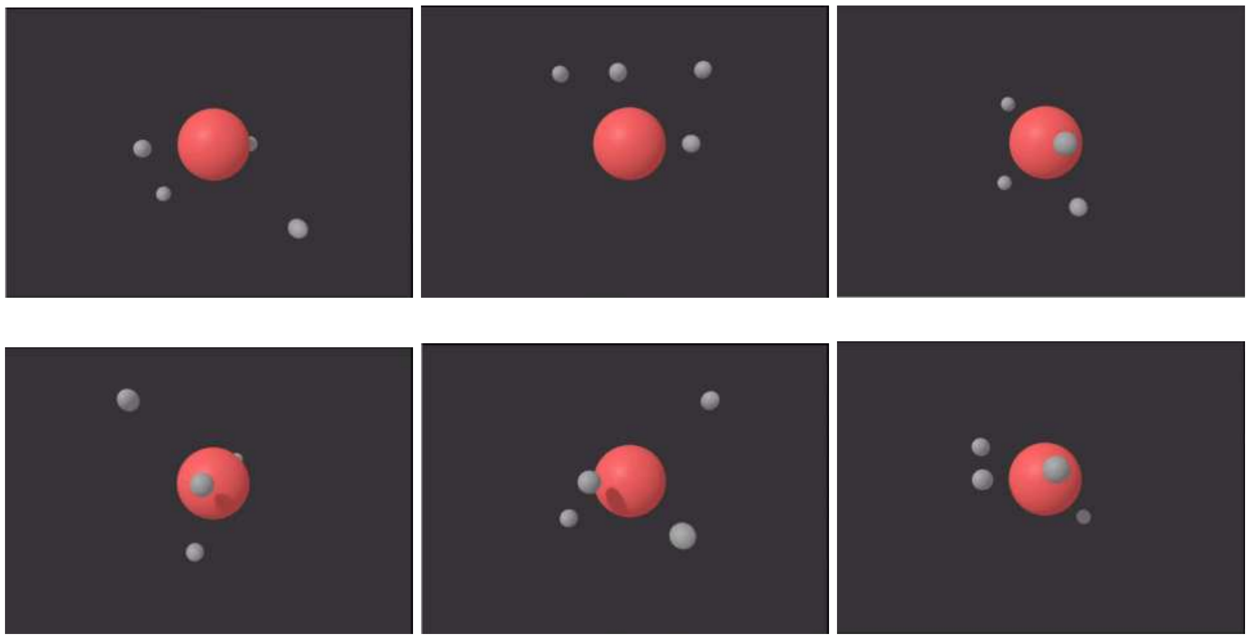


Fig 10. A series of images depicting the orbiting motion of several spheres

4.3 Extended Three.js Geometry

For the final deliverable, we aimed to build upon the current system and support more 3D modeling capabilities that are within the limits of three.js. Since A-Frame is built on top of three.js, we can leverage the three.js libraries to add more complexity to the system. This includes modeling features that modify the geometry and shape of primitives. Therefore, this deliverable added further modeling support to the system, so that if users wanted to create specific shapes, the system would be able to utilize the specific three.js classes to generate the desired output.

Three.js is a powerful JavaScript API that handles 3D graphics in a web browser. It also has vector math libraries that are used for rendering and other computer graphics tasks. A-Frame is built on top of three.js, which means that it has access to some of these advanced libraries and functions capable of producing advanced functionality and geometry modification. However, A-Frame's built-in components and primitive geometric shapes do not use these advanced and complex three.js libraries. Therefore, we wanted to enhance the modeling capabilities of our system by adding more complexity through the use of three.js libraries. However, introducing more complexity can lead to new problems, which means that we need to properly increase the complexity while also trying to maintain a consistent system that is still reliable and capable of providing accurate results based on user commands.

At this point in the project, we were only working with basic geometric shapes, such as geometric boxes, spheres, cylinders, etc. In order to support more advanced modeling features, we needed to add new guidelines into the system that would adjust its knowledge to understand how three.js should be used in custom components to generate new geometric shapes. For this purpose, we implemented three A-Frame JavaScript components into the system that would serve as examples or references for ChatGPT to understand how to use certain three.js classes. We focused on three particular classes: Shape, Extrude, and BufferGeometry. For each of these classes, we programmed an A-Frame component demonstrating the key aspects of the class and how they are typically used to generate interesting shapes using bezier curves, extruded geometry, and morphing attributes. On top of providing these components as references, we also provide more instructions that try to steer the system away from using deprecated three.js functions. We do this by indicating what functions and classes have been deprecated, as well as what changes have been made since the last three.js version that ChatGPT is aware of.

Figure 11 demonstrates some successful and relevant results generated from this deliverable. We can observe that we can now create beveled objects so that we no longer always have entities with sharp edges. Also, as shown by the couch, tree, and double helix examples, we can observe that ChatGPT can understand the concepts of some complex shapes relatively well. The star shape also demonstrates how the system uses curves for specific shapes and extrudes them into 3D to create some interesting entities that we could not create before.

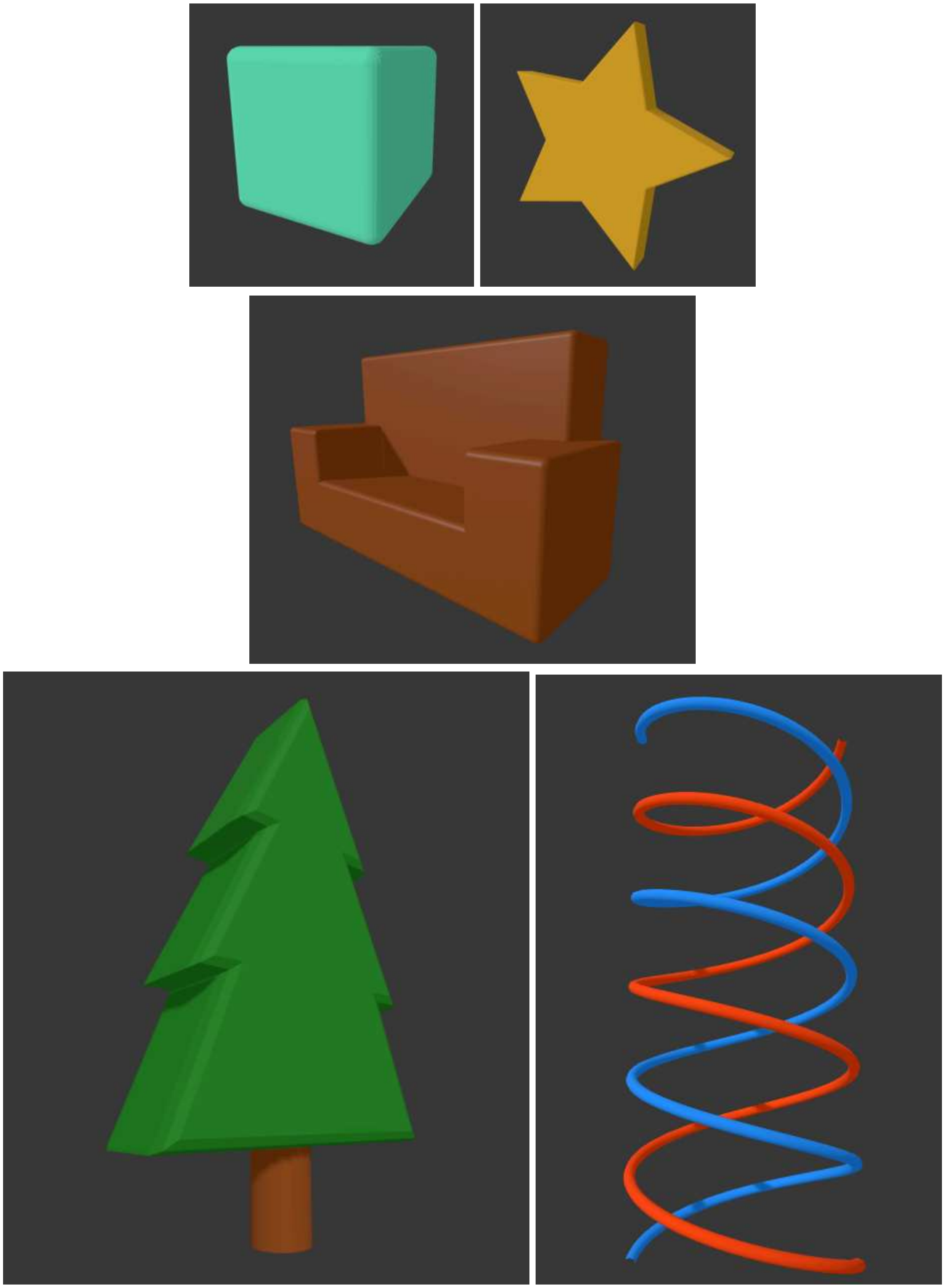


Fig 11. Complex geometric entities

CHAPTER 5

EXPERIMENTS

In this chapter, we describe the experiments that were applied to our system so that we could properly evaluate how temperature affects the generated outputs and determine which temperature values yield the best results. We also discuss our testing metrics, procedure, and testing results.

5.1 Testing Metrics and Variables

To evaluate the reliability and accuracy of the project, we used two metrics; The first is a quantitative metric that measures the success rate, which is quantified by how often the system reliably produces a result when given a request. A failure would be evident when the system does not provide an output due to some error in the generated code or an internal server error. A success would happen when the system successfully generated something and was displayed in the A-Frame scene. The second metric is a qualitative measurement for the relevance score of a result. This determines how well the output matches the specific user command or prompt. Considering the subjective nature of art, this qualitative metric requires careful interpretation, as the judgment of a result's 'relevance' can vary between users. So, to handle this metric, we give each test case result a score ranging from 'zero' to 'three'. A score of 'zero' means that the generated result is not relevant at all. A score of 'one' indicates that the result is partially relevant but misses key aspects. A score of 'two' means that the result is mostly relevant but could be improved. Finally, a score of 'three' would indicate that the result is fully relevant. With this approach, we can properly assess the results of the test cases and have a more appropriate measurement for the overall accuracy of the system. While our focus was on these two metrics, we also recorded the response time, as well as the standard deviation between relevance scores for performance and error analysis.

In terms of testing variables, we used the ‘temperature’ setting of ChatGPT as our independent variable. Temperature works as the creativity slider for ChatGPT. The lower it is, the more predictable and consistent the results will be. However, increasing this variable will yield more random and potentially interesting results. Therefore, to determine the best configurations for ChatGPT to provide the most accurate results based on the user’s prompts, we decided to test the system with multiple temperature values. For ChatGPT, the temperature variable is within the range of 0.0 to 2.0. So, with increments of 0.5, we tested the project with the following temperature values: 0.0, 0.5, 1.0, 1.5, and 2.0.

5.2 Testing Procedure

This section explains the particular testing plan or procedure we took for this project. We came up with 50 unique test cases. These are divided into the following eight testing categories: System, Simple Generation, Simple Functionality, Complex Generation, Complex Functionality, Scenario, Stress, and Error Handling. Not every category has the same number of tests. For instance, System Testing, which tests the basic functionality of the system, has five unique test cases, whereas Complex Generation has ten unique test cases. This is mostly due to the constraint of time, as originally each category had the same number of test cases but in order to meet deadlines we had to scale down to 50 unique test cases. Furthermore, to mitigate the general randomness of ChatGPT, we run each unique test case three times. This provides us with a more accurate measurement for both the overall relevance score and success rate for each unique test case. Overall, we ran all 50 unique test cases three times for each of the five temperature values, meaning that we went through a total of 750 test executions.

For the testing environment, ideally, we would have used the Oculus Quest 2 headset, but again due to the constraint of time, we mostly used a computer to run all of the test cases. However, I did personally run all 50 unique test cases on the Oculus Quest 2 for a single execution during the testing of temperature 0.0 and noticed that the results were very much equivalent to the results we got with the computer. The following shows the computer specifications: Windows 11, Intel Core i7 eight-core processor, 2.30 GHz, 16.0 GB RAM.

5.3 Results

This section covers the results after going over all 750 test executions. Figure 12 shows some interesting results from the Complex Generation test cases. We can observe that some results, particularly the table and couch, look different from the same generations that we have demonstrated in previous chapters. This is likely due to the change in temperature, as these were generated with the temperature set to 1.5. Figure 13 demonstrates the Scenario Testing results, where we went through five sequential test cases that would simulate a likely scenario in which a user would model the basic structure of a house. It has several key aspects of a basic house, including a floor, roof, chimney, door, and walls. However, when placing the door, it seems to have also removed one of the walls, thus reducing the overall accuracy of this generated output.

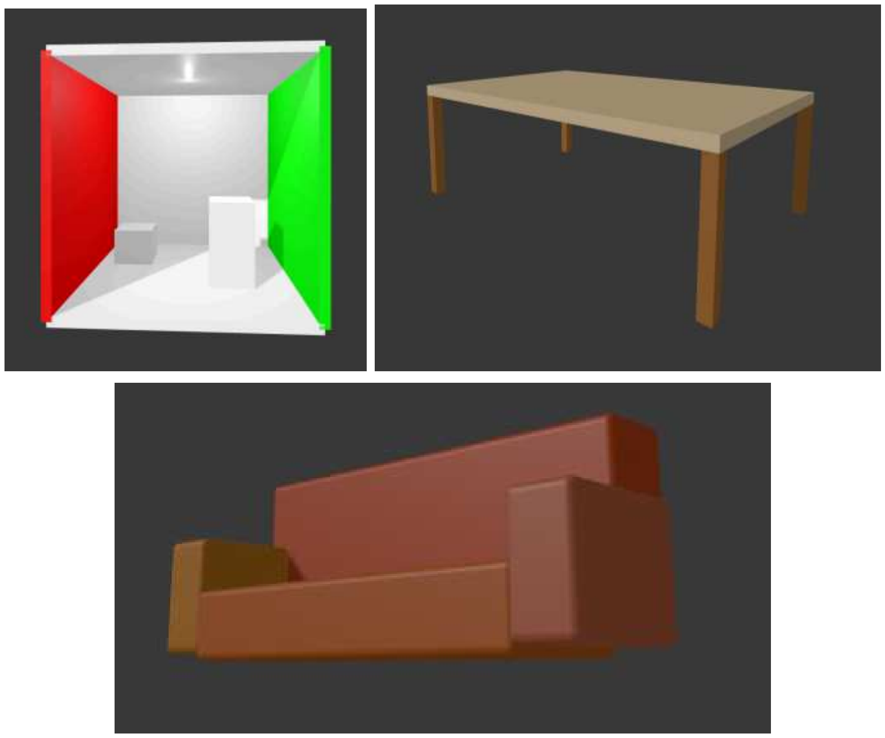


Fig 12. Complex generation testing examples

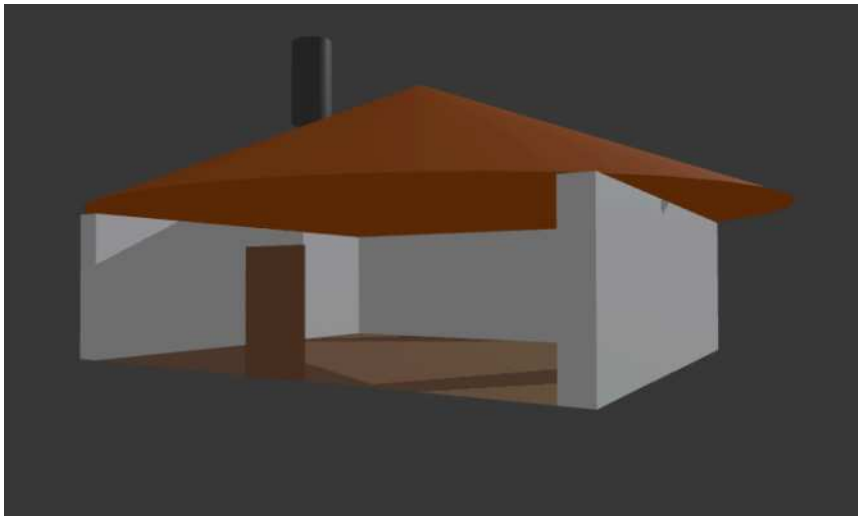


Fig 13. Scenario testing example

5.3.1 Success Rate

Figure 14 has all eight graphs, one for each of the testing categories, that visualize the relationship between success rate and temperature. We can observe that for most testing categories, as we get to the higher levels of temperature, the success rate drops significantly. For the most part, temperature values below 1.5 seem to provide some of the best levels of reliability in the system, with values around 0.5 and 1.0 maintaining some of the highest success rates. Therefore, in regards to reliability, we want to set our temperature to be around the 0.5 to 1.0 range. This would provide the best results for the system to succeed and generate something when given a prompt. The failures are generally due to ChatGPT generating code that contains errors, such as incorrect mathematical calculations, using external libraries, or using outdated programming functions. Also, some failures, specifically when testing with the temperature values 1.5 and 2.0, were 'Internal Server' errors, which would execute but not return anything from the GPT API. After further inspection, we noticed that the results for these particular failures had not produced programming code at all, but instead consisted of random words, characters, numbers, and symbols. This occurs with ChatGPT when its temperature is set to the highest values, making the randomness so high that it generates nonsensical text for its responses. An example of this type of text is provided in Figure 15.

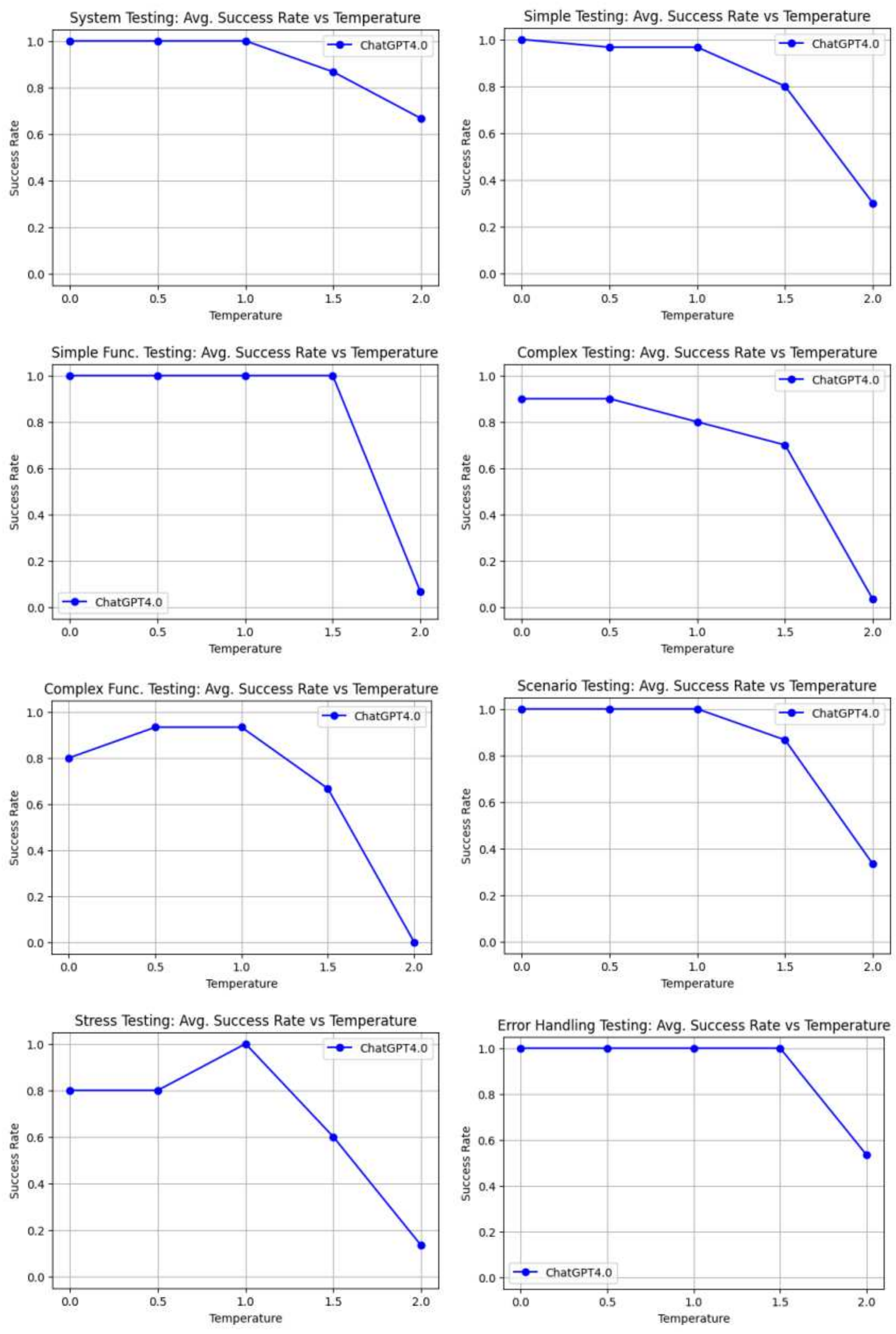
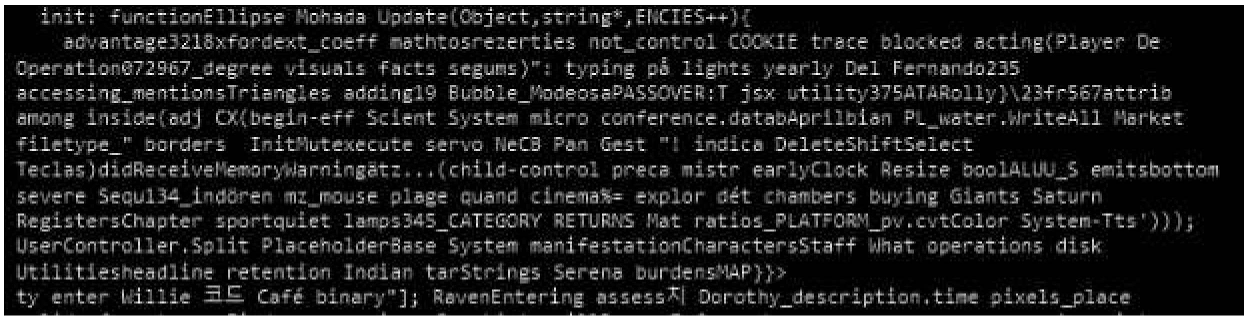


Fig 14. Success Rate vs Temperature graphs



```
  init: functionEllipse Mohada Update(Object,string*,ENCIES++){
    advantage3218xfordext_coeff mathtosrezerties not_control COOKIE trace blocked acting(Player De
Operation072967_degree visuals facts segums)": typing på lights yearly Del Fernando235
accessing_mentionsTriangles adding19 Bubble_ModeosaPASSOVER:T jsx utility375ATARolly}\23fr567attrib
among inside(adj CX(begin-eff Scient System micro conference.databAprilbian PL_water.WriteAll Market
filetype_" borders  InitMutexecute servo NeCB Pan Gest "! indica DeleteShiftSelect
Teclas)didReceiveMemoryWarningätz...(child-control preca mistr earlyClock Resize boolALUU_S emitsbottom
severe Sequ134_indören mz_mouse plage quand cinema%= explor dét chambers buying Giants Saturn
RegistersChapter sportquiet lamps345_CATEGORY RETURNS Mat ratios_PLATFORM_pv.cvtColor System-Tts')));
UserController.Split PlaceholderBase System manifestationCharactersStaff What operations disk
Utilitiesheadline retention Indian tarStrings Serena burdensMAP}}>
ty enter Willie 크드 Café binary"]; RavenEntering assess지 Dorothy_description.time pixels_place
```

Fig 15. Nonsensical results from high temperature values

5.3.2 Relevance Score

Figure 16 shows the results for the relationship between the relevance score and temperature values. The relevance score measurements serve as the qualitative metric for our system. We can observe from these graphs that, similar to the success rate, the relevance score generally goes down as the temperature increases. However, in this case, there is a gradual decrease in the average relevance score for each testing category as we progressively increase the temperature. Also, the graphs demonstrate how different testing categories greatly affect the relevance score. For instance, we can see how the System and Simple Generation testing categories started and maintained a higher average relevance score than categories like Complex Generation. This indicates that ChatGPT's accuracy is not only impacted by the temperature of the system but also by the level of complexity in the user's command. This concept is better illustrated in Figure 17, where we can observe the relationship between the average relevance score against the eight testing categories. In this graph, each category has five bars, each representing one of the five tested temperature values. From the information provided by both figures, we can conclude that for each testing category, the best relevance or accuracy score is provided when the temperature is below 1.5. Also, when looking at the average relevance scores across all testing categories in Figure 17, we can see that simpler commands, unlike the commands from the Complex Generation or Complex Functionality categories, will provide outputs with higher average relevance and accuracy scores.

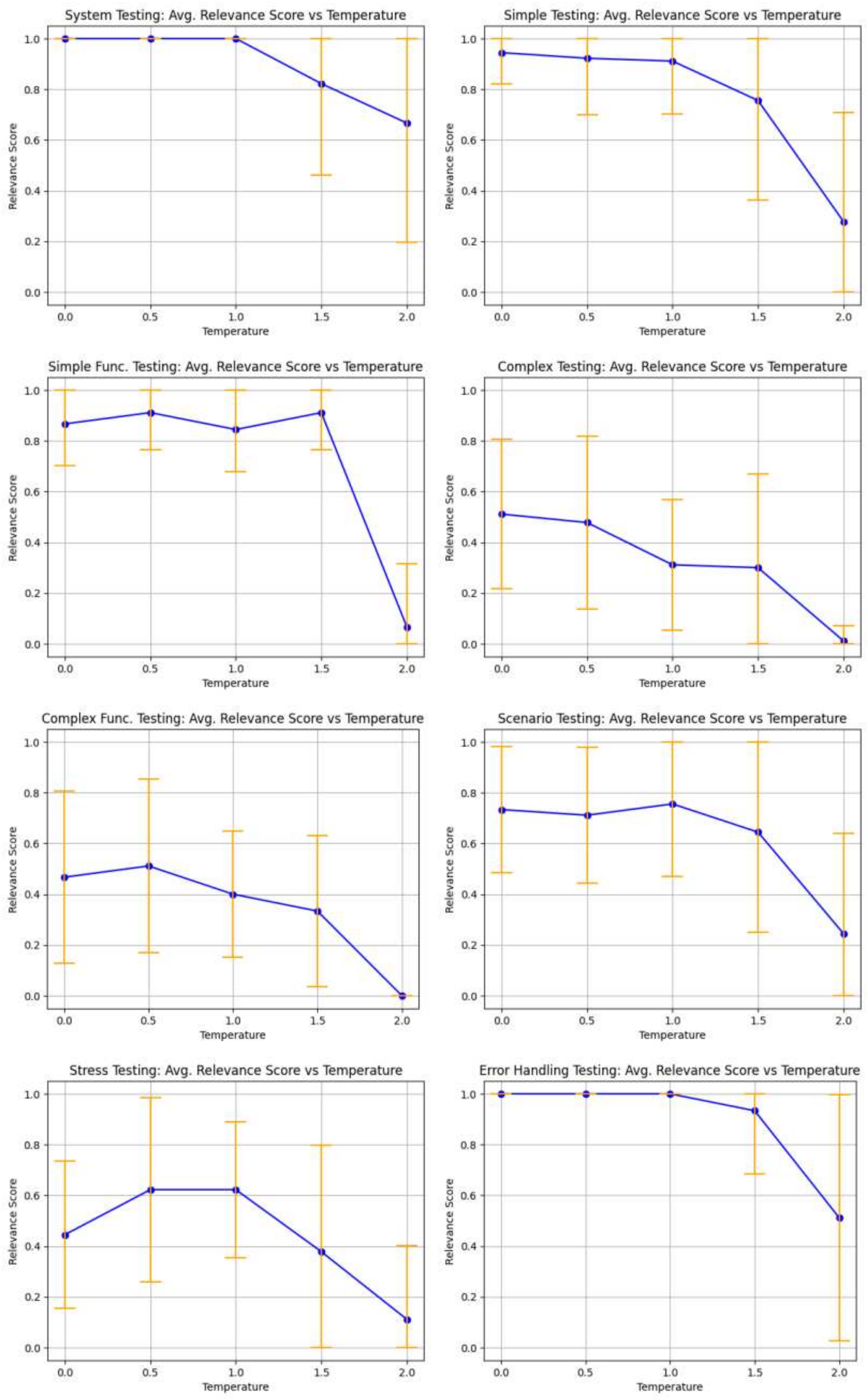


Fig 16. Average relevance scores along with their standard deviations

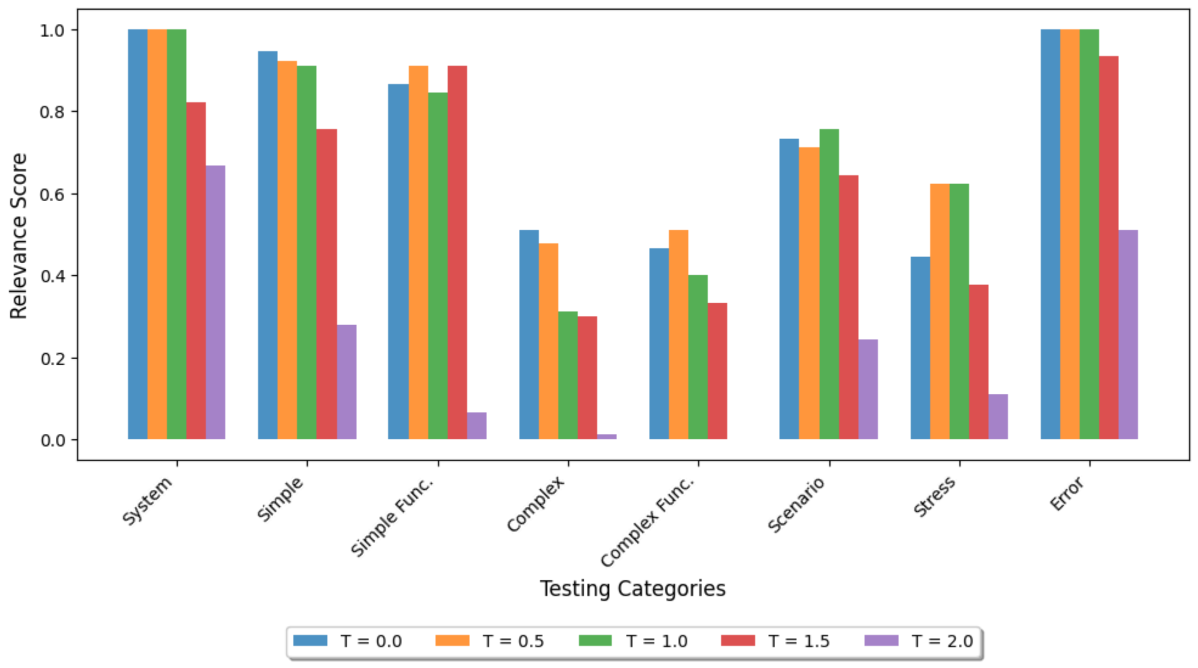


Fig 17. Relevance Score vs Temperature for each of the 8 testing categories

5.3.3 Response Time and Standard Deviation

Finally, we recorded some data for performance and error analysis. For error analysis, we used standard deviation to measure the variance of the relevance scores. To record this measurement, we standardized the relevance scores into a range of zero and one. Then using Python and the Numpy standard deviation function, we computed the results for every category, which are presented in both Figure 16 and Table 1. A lower standard deviation meant that the results were more consistent, whereas a higher standard deviation indicated that there was more variance between the relevance scores. From Table 1, we can observe that for the simpler testing categories, such as System, Simple Generation, Simple Functionality, and Error Handling, the standard deviation started low and gradually went up as we increased the temperature. However, the more complex categories, including Complex Generation and Complex Functionality, do not follow this pattern, and instead start off relatively high and drop very low by the last temperature. This happens because at high temperatures, the system consistently generates bad or irrelevant results, due to the high randomness. Also, the bottom row of Table 1 displays the average standard deviation across all categories for each temperature. This average indicates that the system has higher consistency at lower temperatures, which is expected since temperature is designed to control the randomness of ChatGPT. In regards to the performance of the system, we recorded the response time, which was measured by the average response time from the moment a prompt was sent to ChatGPT to the moment that a response was received. Figure 18 visualizes the response times against all eight testing categories. The results highlight a drawback of our system, which is that users typically have to wait for their prompt to be processed and executed before the A-Frame scene is updated. Specifically, as indicated by Figure 18, more complex prompts would result in longer waiting times, making the system not as practical or engaging as we would prefer. Also, in this case, temperature did not affect the response time as much until the temperature was above 1.5. Therefore, the complexity of the prompts was the main factor in the response time of ChatGPT, with complex prompts taking longer compared to simpler prompts.

ID	Category	S.D (T = 0.0)	S.D (T = 0.5)	S.D (T = 1.0)	S.D (T = 1.5)	S.D (T = 2.0)
1	System	0.00	0.00	0.00	0.36	0.47
2	Simple Gen.	0.12	0.22	0.21	0.34	0.43
3	Simple Func.	0.16	0.14	0.17	0.15	0.25
4	Complex Gen.	0.29	0.34	0.26	0.37	0.05
5	Complex Func.	0.33	0.34	0.25	0.30	0.00
6	Scenario	0.24	0.27	0.29	0.39	0.39
7	Stress	0.28	0.36	0.27	0.42	0.29
8	Error Handling	0.00	0.00	0.00	0.25	0.48
	Mean =	0.17	0.20	0.21	0.32	0.30

Table 1. Standard Deviations for each category across all temperatures

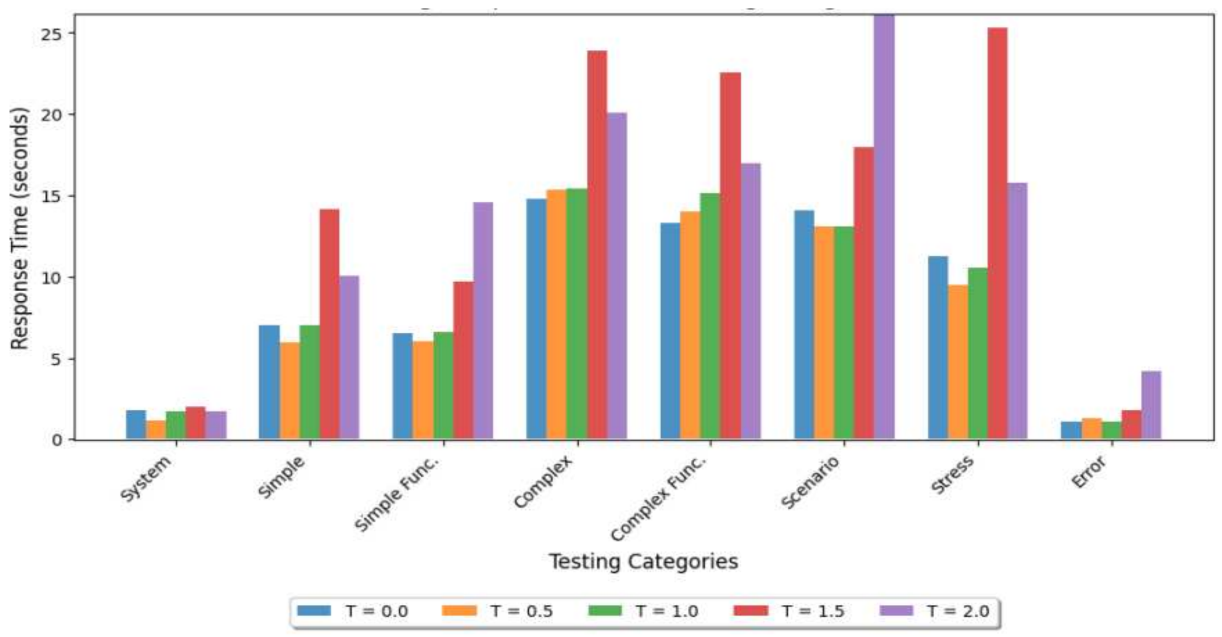


Fig 18. Response Time vs Testing Categories

CHAPTER 6

CONCLUSION

In conclusion, we were able to implement an application that leverages ChatGPT's natural language processing power to create a user-friendly 3D modeling environment. Users can simply provide a voice command that is transformed into an actionable A-Frame instruction to update the current 3D scene. The enhancements made within the second semester of working on the project were focused on improving both the reliability and accuracy of the system. As our results showed, the system provides the best success rates when using temperatures around 0.5 and 1.0. In regards to relevance, the system provides the most accurate results when we have temperatures below 1.5 and when we use simpler modeling prompts. While our system is also capable of handling advanced prompts, it tends to result in longer response times compared to simpler commands. Furthermore, in terms of future work, we could look to work with different or new large language models to enhance our system. As organizations, including OpenAI, continue to further improve their models, not only would they decrease our response times but could also potentially increase the accuracy of the results to the point where users will be able to create even more complex and interactive virtual worlds.

REFERENCES

[1] M. Abdullah, A. Madain and Y. Jararweh, "ChatGPT: Fundamentals, Applications and Social Impacts," *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, Milan, Italy, 2022, pp. 1-8, doi: 10.1109/SNAMS58071.2022.10062688.

[2] S. G. Santos and J. C. S. Cardoso, "Web-based Virtual Reality with A-Frame," *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, Coimbra, Portugal, 2019, pp. 1-2, doi: 10.23919/CISTI.2019.8760795.

[3] T. Yoshinaga. "Collaboration between ChatGPT API and A-Frame," *Youtube*. 2023. Available: https://www.youtube.com/watch?v=FFQSKJUCBm0

[4] S. Göring, R. R. Ramachandra Rao, R. Merten and A. Raake, "Appeal and quality assessment for AI-generated images," *2023 15th International Conference on Quality of Multimedia Experience (QoMEX)*, Ghent, Belgium, 2023, pp. 115-118, doi: 10.1109/QoMEX58391.2023.10178486.

[5] M. Cahyadi, M. Rafi, W. Shan, H. Lucky and J. V. Moniaga, "Accuracy and Fidelity Comparison of Luna and DALL-E 2 Diffusion-Based Image Generation Systems," 2023 *IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, BALI, Indonesia, 2023, pp. 108-112, doi: 10.1109/IAICT59002.2023.10205695.

[6] K. Suzuki, J. Cai, J. Li, T. Yamauchi, and K. Tei, "A Comparative Evaluation on Melody Generation of Large Language Models," *2023 IEEE International Conference on*

Consumer Electronics-Asia (ICCE-Asia), Busan, Korea, Republic of, 2023, pp. 1-4, doi: 10.1109/ICCE-Asia59966.2023.10326362.

[7] M. Jang and T. Lukasiewicz, “Consistency Analysis of ChatGPT,” arXiv:2303.06273 [cs], Mar. 2023, Available: https://arxiv.org/abs/2303.06273

[8] C. Li and B. Tang, “Research on Voice Interaction Technology in VR Environment,” *2019 International Conference on Electronic Engineering and Informatics (EEI)*, Nanjing, China, 2019, pp. 213-216, doi: 10.1109/EEI48997.2019.00053.