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

Project Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Ivan Hernandez

Fall 2023

ABSTRACT

ChatGPT has recently gained a significant amount of popularity as a large language model that provides users with context-aware conversations and interactions. Along with this recent rise in popularity, new opportunities for human-computer interactions have become possible. On the other hand, some software, such as traditional 3D modeling applications with overwhelming user interfaces, have lagged behind when it comes to improving their interactivity and ease of use. Therefore, we can leverage ChatGPT's natural language processing capabilities to enhance the 3D modeling experience. By integrating ChatGPT into A-Frame, an online framework for creating virtual reality experiences, we can develop an immersive and interactive 3D modeling environment where users can communicate their design intent through natural language commands in virtual reality. For the first semester, we worked on four deliverables. The first and second deliverables established the foundation of the project, which included the integration of ChatGPT into A-Frame. In the third deliverable, we fine-tuned the model to behave like an A-Frame assistant. Lastly, the fourth deliverable introduced voice interactions into the system. By the end of the first semester, we have a project that combines the advanced language processing power of ChatGPT and A-Frame's immersive technology to create a more engaging experience for 3D modeling with voice commands.

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

TABLE OF CONTENTS

I. Introduction

Over the past year, we have witnessed the rise of a cutting-edge large language model called ChatGPT. While many individuals have used ChatGPT for contextually aware conversations, its capabilities extend further beyond. An impressive application of ChatGPT is code generation, where it is capable of creating programming code snippets in a variety of programming languages [1]. People have also utilized ChatGPT as a chatbot or assistant capable of engaging in back-and-forth conversations. This type of human-computer interaction is very user-friendly and could be particularly beneficial for complex computer applications that may not be intuitive for all users, such as current 3D modeling software applications. Integrating natural language processing into the field of 3D modeling would not only improve the user experience but also streamline the content creation process. We can combine ChatGPT's capabilities with A-Frame, an online framework that uses HTML and Javascript for building Virtual Reality(VR) experiences[2], to create an immersive and interactive 3D modeling environment. Therefore, the goal of this project is to develop a system that integrates ChatGPT into an A-Frame-based VR environment, thus enabling users to create, modify, and interact with 3D models based on user-provided natural language descriptions. By leveraging ChatGPT's natural language understanding capabilities, this system can offer a more engaging experience for users to guide and modify the 3D modeling process in real-time.

This report will cover the four deliverables that we completed within the first semester. Each section will provide the following: a description of the deliverable, the approach, implementation, and results. The first deliverable will focus on A-Frame, discussing its features, advantages, disadvantages, and how it can be connected to Glitch. In the second deliverable, we will explore ChatGPT and its API integration with A-Frame. The third deliverable involves the

full integration of ChatGPT into A-Frame, enabling users to provide modeling commands that will be reflected in the 3D scene. The final deliverable introduces audio commands to add a new modality input and improve the user experience within the VR environment. To conclude, we will provide a project summary and discuss future developments.

II. Deliverable I: Basic A-Frame Virtual Reality Environment

The goal of the first deliverable was to create a functional VR environment using A-Frame and incorporate some simple 3D primitives into the scene. Afterwards, we compared and contrasted the workflow between A-Frame and Unity3D to see the advantages and disadvantages of A-Frame when it comes to 3D object positioning and manipulation.

The A-Frame documentation has a lot of useful information regarding A-Frame's entity-component system (ECS) along with simple example scenes that can be easily viewed online. Many of these A-Frame scenes are hosted through Glitch, an online editor capable of hosting A-Frame applications. For the first deliverable, we used the Glitch editor to write and host a simple VR A-Frame scene with three primitives, each with different object properties. Figure 1 shows the simple A-Frame scene along with the code in Glitch. We also wanted to compare the workflows of A-Frame and Unity3D. To accomplish this, we needed to create a basic virtual scene composed of a wall, table and functional door in both applications. The tools used for this task were as follows: A-Frame(HTML/Javascript), Glitch(server host), Unity3D(C#), and the Oculus Quest 2.

We first implemented the 3D scene in A-Frame. This entailed creating <a-entity> tags for each object and positioning them appropriately. The workflow required a significant amount of going back and forth between the 3D scene and HTML code to make sure that the objects were positioned correctly. Textures were added to the objects to make the scene look more realistic. Lastly, the door was given functionality so that it was interactable within the VR environment. After creating the virtual scene in A-Frame, we then created the same scene in Unity3D. A similar approach was followed for Unity3D. First, we built and positioned the objects in the 3D

editor. Then, we added the textures to the objects before incorporating some basic functionality to the door.

The results can be viewed in Figure 2. We can observe that the scenes are very similar, with the main difference being the lighting system of each software. For the workflow, we encountered similar results as those reported in [2], where transforming primitives in A-Frame becomes tedious and time consuming because of the back-and-forth attribute modifications. On the other hand, Unity provides a visual editor that includes widgets to translate, rotate, and scale objects, which makes it substantially easier to transform and align objects. In regards to texturing, the process was relatively similar and easy to implement in both Unity and A-Frame. Finally, for functionality, both Unity and A-Frame have an entity-component system architecture. Therefore, when adding the door functionality, the implementation process was very similar in both cases.

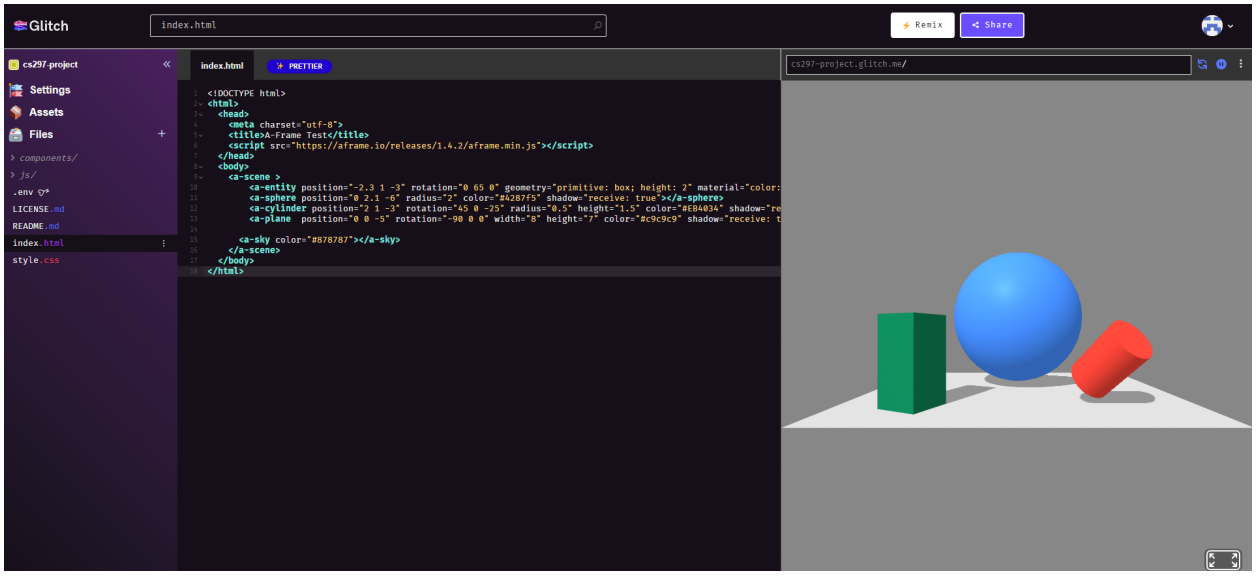

Fig 1. Glitch online editor and simple A-Frame scene.

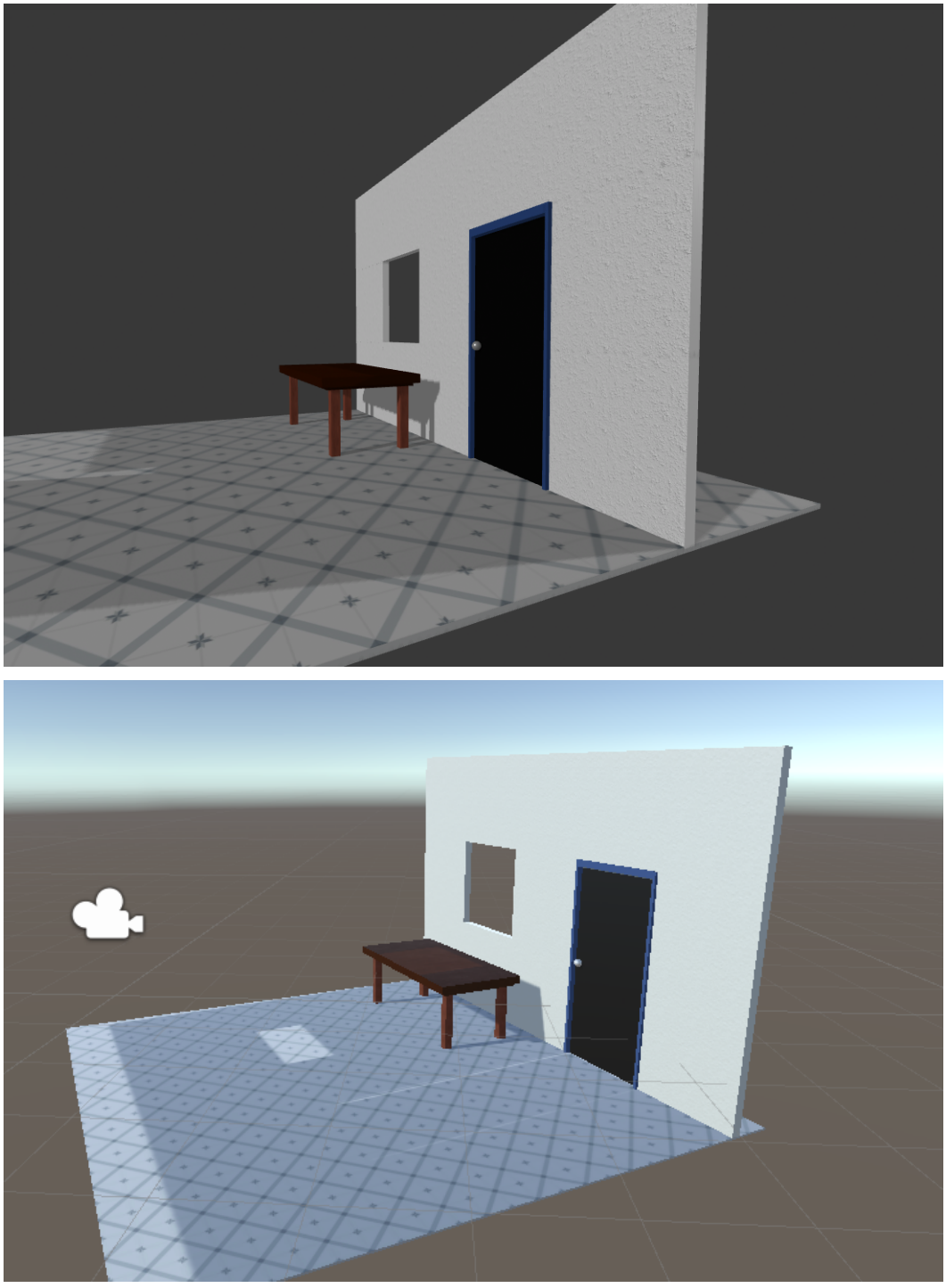

Fig 2. Basic virtual scene in both A-Frame(above) and Unity3D(below)

III. Deliverable II: ChatGPT Integration Into A-Frame

The goal of the second deliverable was to merge ChatGPT with the A-Frame environment. In order to interact with the ChatGPT API, a temporary user interface was made available, allowing users to input prompts and receive ChatGPT's responses. While later on we implemented voice commands, this initial user interface was very useful for testing and debugging during the initial parts of the project. Ultimately, we will not need the user interface, as the voice commands will help ensure that the ChatGPT integration works seamlessly with the virtual reality environment.

We first began our approach by learning more about ChatGPT and OpenAI's API documentation. This gave us a better understanding about the multiple models provided by OpenAI's API and the respective prices for using these models. Acquiring an OpenAI API key granted us access not only to their GPT models but also to the audio models that we later used for the fourth deliverable. Then, we designed a simple user interface to handle the prompt inputs. This allowed us to make requests with the API and communicate with ChatGPT. After we send a request to ChatGPT, we will get a response in the form of a json file, which we will then process before displaying it on the user interface. Finally, to make sure that the API keeps track of the conversation between the user and ChatGPT, we added context to the requests that we send to ChatGPT.

For the implementation, using the API key, we send a POST request to the 'https://api.openai.com/v1/chat/completions' endpoint, which allows us to use the GPT model in our project. Figure 3 shows the code for making the API calls. We also specify the particular model we want to use and set specific values for the "temperature" and "max_tokens" parameters. A lower "temperature" makes the model less random and more consistent. The

“max_tokens” determines the maximum number of tokens or words to generate in the response. For the model, we initially started using the GPT-4 model, but soon after moved to their newest and more powerful model GPT-4 Turbo, which introduces a much higher context window and is cheaper than the previous model. Also shown in Figure 3 is the “messages” parameter. This represents the conversation between the user and ChatGPT and is the information that we send in the POST request. In order to keep the context of the full conversation, we make sure that the “payload” variable records every message, including the user’s prompts and ChatGPT’s responses. This allows the program to keep track of information from previous messages, which is shown in Figure 4.

Once we send the request and get back the ChatGPT response, we extract the json file and display the response on the simple user interface. Figure 4 demonstrates the user interface working inside an A-Frame scene. This indicates that we were able to accomplish the task of integrating ChatGPT into A-Frame. As a result, users can easily interact with the virtual environment using natural language prompts, making the overall experience more intuitive and engaging.

```
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

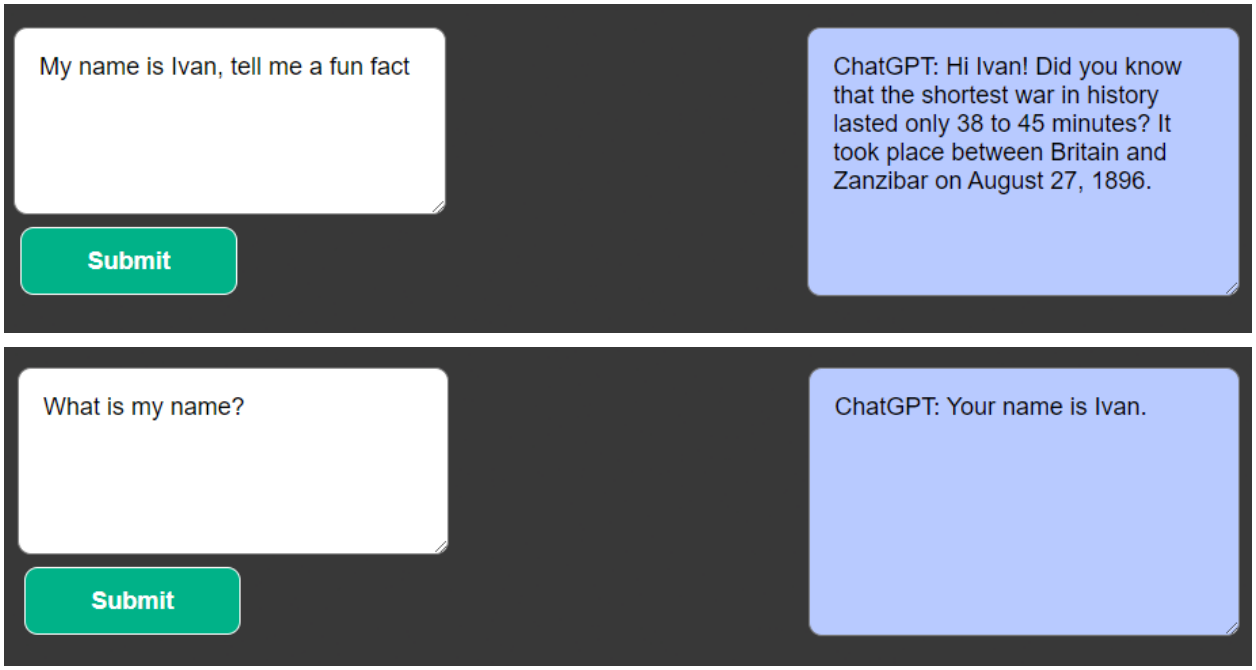

Fig 4. Simple UI and example of ChatGPT working inside A-Frame

IV. Deliverable III: ChatGPT Modeling Interpreter

The goal of the third deliverable was to ensure that the user inputs are converted into actionable commands for the A-Frame environment. This process involved defining a command structure for creating, modifying, and removing objects as well as establishing a foundation for user interactions to be properly interpreted and transformed into modeling commands. Therefore, the system needs to be robust, flexible, and capable of handling a wide range of user inputs.

Our first task was to define the specific instructions to fine-tune the system's behavior so that we get standardized ChatGPT outputs. These designed instructions will make ChatGPT an A-Frame assistant capable of creating A-Frame primitives, modifying their object properties, and removing them from the scene. Then, we will be able to extract the A-Frame code from the ChatGPT response and process the code so that it is fully integrated into the A-Frame scene. To test the system, we set up a series of test sets containing a variety of different commands, including simple and complex inputs. We also include error handling instructions for the system to recognize when there are invalid or ambiguous commands.

OpenAI provides API users the ability to customize the system's behavior by indicating particular rules. Figure 5 shows the set of instructions that we used for our system. Through these instructions, we create a specialized artificial intelligence assistant that generates A-Frame code based on the user's prompt. Many of these instructions are designed so that ChatGPT focuses only on providing A-Frame code that is formatted inside the <a-scene> tags and disregard extra text that ChatGPT generally provides for context. Also, some instructions provide useful 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 ensure that it understands the setup and context of the A-Frame environment. There are also other instructions for ChatGPT to handle

invalid and vague commands, so that such commands are immediately detected and properly returned to be displayed as warning or clarification messages in the user interface. Once we get the formatted responses back, we extract the code using regex strings and insert the A-Frame code inside the main <a-scene> tag. This updates the A-Frame scene and reflects the changes made by the user's commands

Moreover, with these instructions, we were able to create an assistant that is capable of handling simple and complex commands. Simple commands, such as "Create a red cube," were handled very well. On the other hand, the complex commands typically required follow-up adjustments from the user. Figure 6 shows some examples from the complex commands test set. We can observe that the system is able to understand objects very well, which is shown by the couch example where it is able to distinguish between the couch and the cushions with different colors. The two commands used to create the UFO in Figure 6 were as follows: "Create a cool UFO in the scene" and "Add buttons to the inside and outside of the UFO." For the UFO animation shown as snapshots in Figure 7, we used the following command: "Animate the whole UFO taking off to the sky at an angle away from me." The end result really demonstrates the power of ChatGPT, where it was not only able to recognize that the UFO should be moving away from the user but also rotate as it was flying away, thus understanding that a UFO should behave in a particular manner.

While the system yielded impressive results for the complex commands, we also tested how consistent ChatGPT was when creating A-Frame code given simple commands. In [3], the authors tested ChatGPT's consistency with three different consistency types: negation, semantic, and symmetric. We used the same consistency types, and presented the results in Figures 8, 9, and 10 respectively. In Figure 8, we can see that all of the tests for negation consistency passed,

which is similar to what the authors in [3] found. Generally, ChatGPT is very good at determining negation and contradictions. For the other types of consistencies, the system performs well but each consistency has one test case where ChatGPT does not provide the desired outcome. For semantic consistency, shown in Figure 9, the test 6 inputs have the same meaning but are phrased differently, yet ChatGPT did not treat them as the same, so it was inconsistent. Similarly, for the symmetric consistency results in Figure 10, the first test was inconsistent as the order of the commands affected the output from ChatGPT. However, overall ChatGPT does relatively well at providing consistent results for simple modeling commands.

```
[{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 5. Specific instructions given to ChatGPT to specialize its system

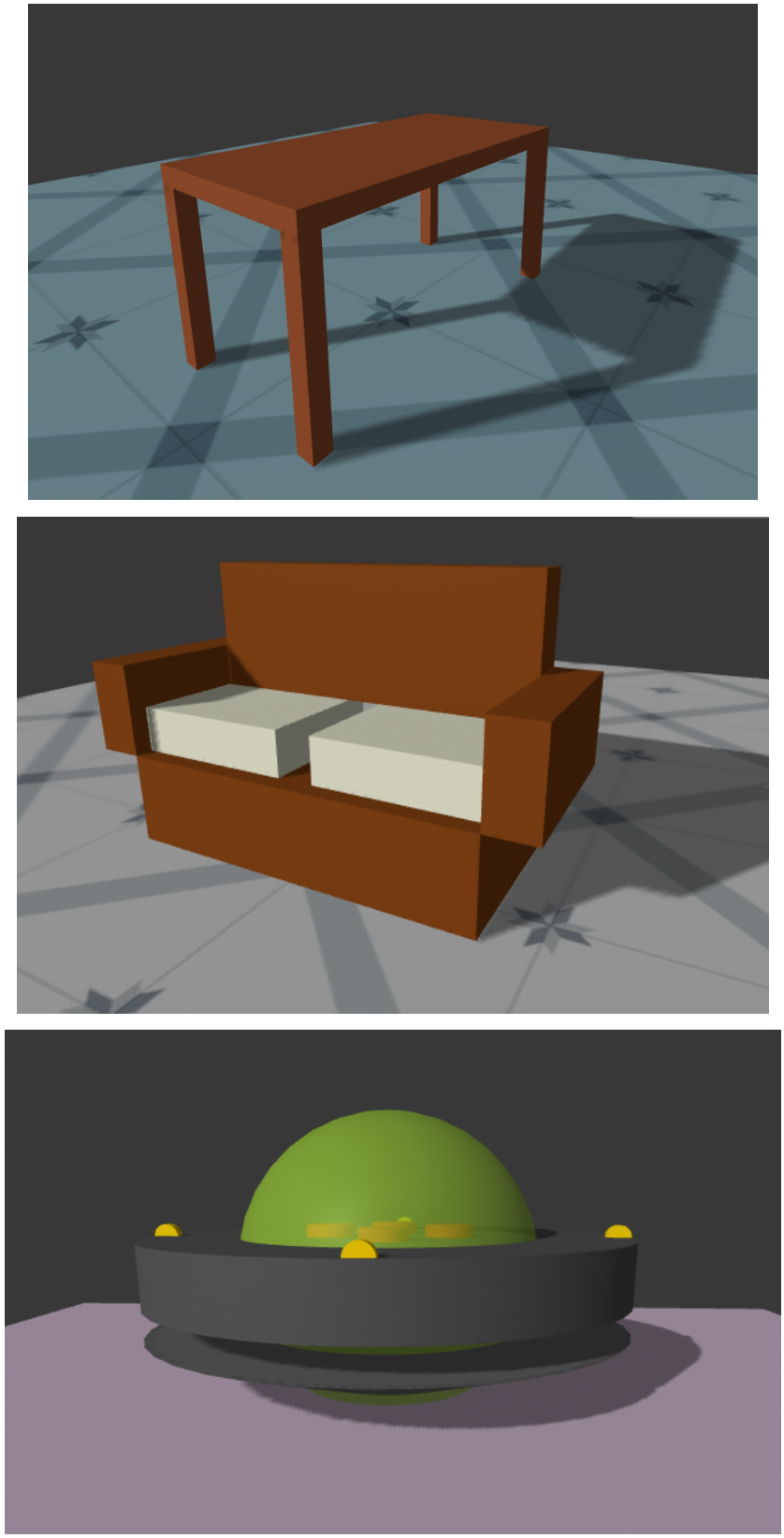

Fig 6. Results from complex input tests

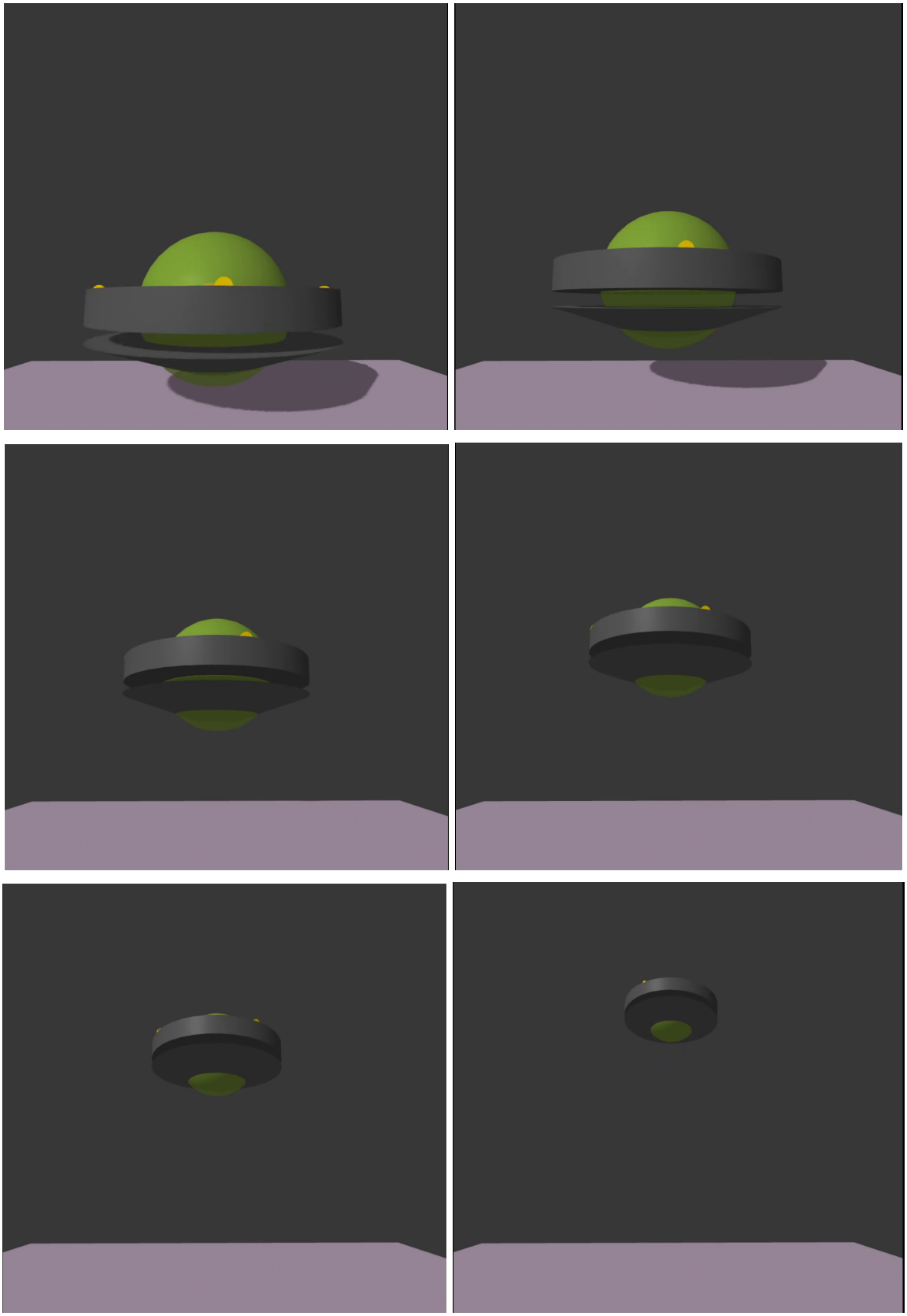

Fig 7. Sequential snapshots of UFO animation in A-Frame

Test	Input 1	Input 2	Equivalent
1	“Create a red cube”	“Do not create a red cube”	**No**
2	“Add a green sphere”	“Avoid adding a green sphere”	**No**
3	“Move the cube to the left”	“Don’t move the cube to the left”	**No**
4	“Move the sphere up”	“Prevent the sphere form moving up”	**No**
5	**“Rotate the cube 45 degrees”**	**“Do not rotate the cube 45 degrees”**	**No**
6	“Scale the sphere to half the size”	“Do not scale the sphere to half its size”	**No**
7	“Change the cube’s color to blue”	“Skip changing the cube’s color to be blue”	**No**
8	“Delete the sphere”	“Don’t delete the sphere”	**No**
9	“Delete the cube”	“Avoid deleting the cube”	**No**
10	“Create a yellow cone”	“Do not create a yellow cone”	**No**

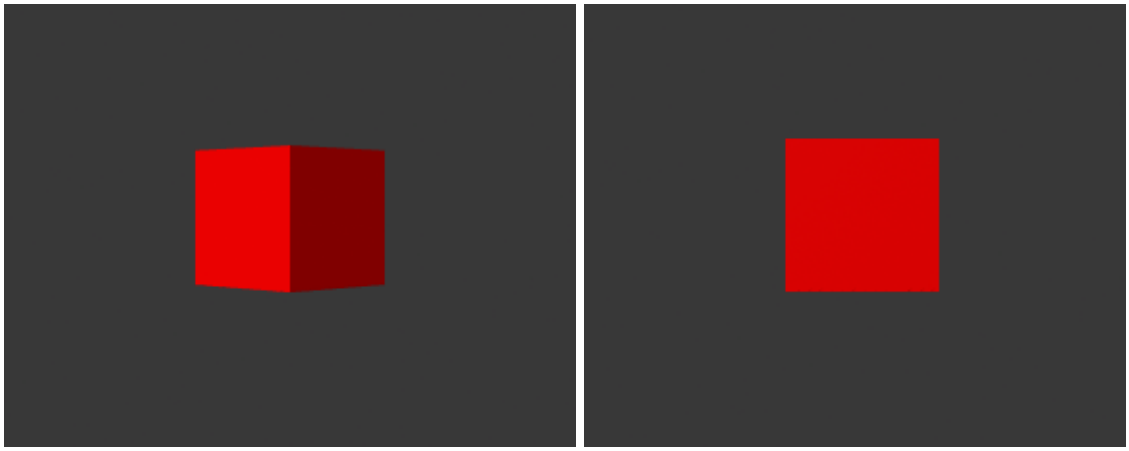

Fig 8. Negation consistency test set with an example of test 5.

Test	Input 1	Input 2	Equivalent
1	“Create a red cube”	“Generate a red cube”	**Yes**
2	“Add a green sphere”	“Include a green sphere”	**Yes**
3	“Move the cube to the left”	“Shift the cube to the left”	**Yes**
4	“Move the sphere up”	“Elevate the sphere”	**Yes**
5	“Rotate the cube 45 degrees”	“Turn the cube 45 degrees”	**Yes**
6	**“Scale the sphere to half the size”**	**“Make the sphere half its size”**	**No**
7	“Change the cube’s color to blue”	“Modify the cube to be blue”	**Yes**
8	“Delete the sphere”	“Eliminate the sphere”	**Yes**
9	“Delete the cube”	“Clear the box”	**Yes**
10	“Create a yellow cone”	“Craft a cone that is yellow”	**Yes**

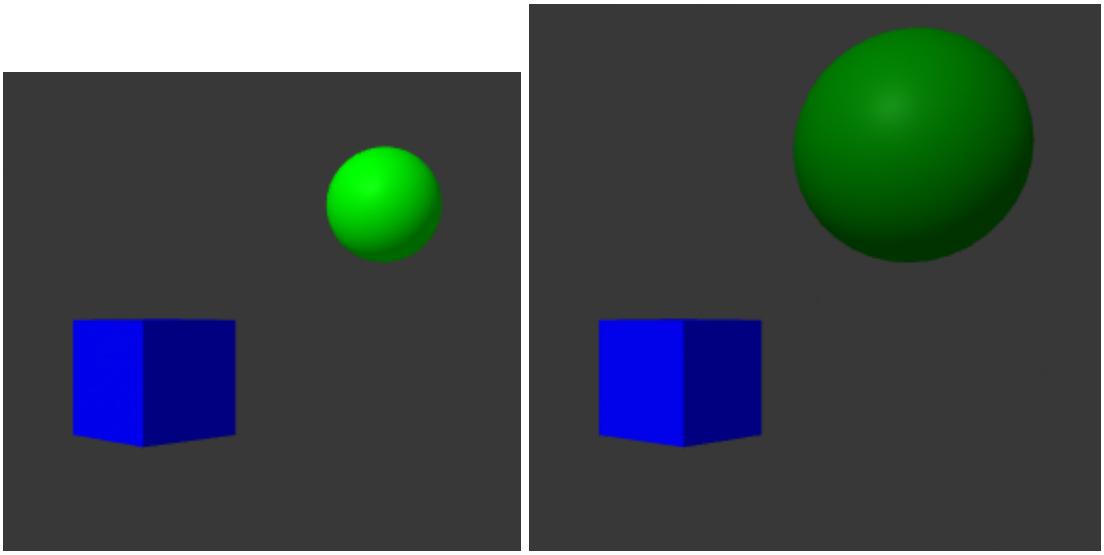

Fig 9. Semantic consistency test set with an example of test 6.

Test	Input 1	Input 2	Equivalent
1	“Create a red cube” “Add a green sphere”	“Add a green sphere” “Create a red cube”	**No**
2	“Move the cube to the left” “Move the sphere up”	“Move the sphere up” “Move the cube to the left”	**Yes**
3	**“Rotate the cube 45 degrees”** **“Scale the sphere to half the size”**	**“Scale the sphere to half the size”** **“Rotate the cube 45 degrees”**	**Yes**
4	“Change the cube’s color to blue” “Delete the sphere”	“Delete the sphere” “Change the cube’s color to blue”	**Yes**
5	“Delete the cube” “Create a yellow cone”	“Create a yellow cone” “Delete the cube”	**Yes**

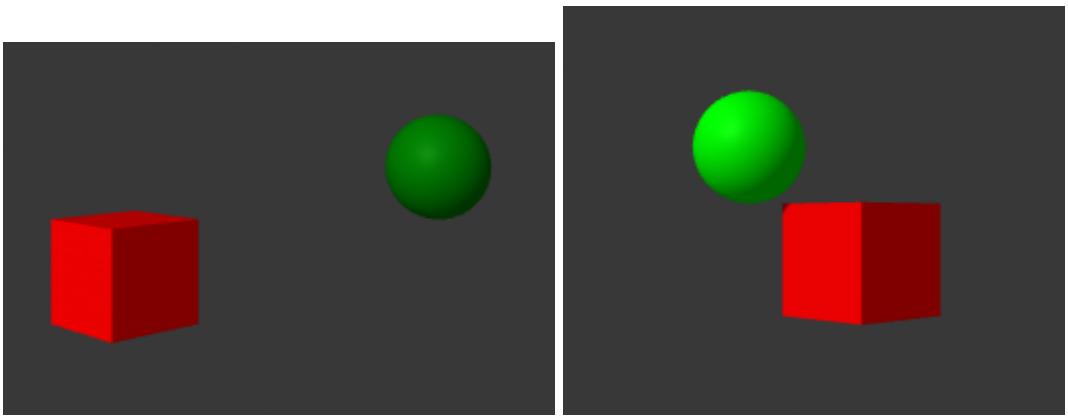

Fig 10. Symmetric consistency test set with an example of test 3.

V. Deliverable IV: Voice Command Integration

The goal of deliverable four was to incorporate voice interactions into the virtual reality environment. By introducing voice commands, we aim to create a more immersive and user-friendly experience. This integration included voice recognition, speech-to-text transcription, integration with ChatGPT, and vocal feedback responses. The overall focus was to optimize user communication and to create a seamless experience within the virtual reality environment.

We first started by going into the OpenAI documentation, where we found information about OpenAI's audio models, such as the Whisper and Text-To-Speech models. The Whisper model allows users to transcribe audio into text. For our project, it allowed us to transcribe captured speech from the Oculus Quest 2 headset. We recorded audio using the headset's microphone and then processed the audio before sending it to the Whisper API to transcribe it. Afterwards, we sent the transcription to ChatGPT and got back the response, which we handled like we did in the previous deliverables. Finally, we updated the A-Frame scene accordingly. On top of this voice interaction, we also wanted to add vocal feedback after the user makes a command. To accomplish this, we used the Text-To-Speech model provided by OpenAI. Whenever the user provides a command, we have preset strings, such as "Processing request," that are turned into audio and played out loud for the user to receive vocal feedback and confirmation.

For the implementation, we first created the controls to capture audio from the Oculus Quest 2 headset. We mapped the "A" and "B" buttons to start and stop recording, respectively. For clarity purposes, we included a small microphone icon as an indicator that the system is currently recording the user's audio. We then created a .wav audio file, which we sent in a POST

request to the Whisper model endpoint. The Whisper model returned the transcribed text which we then sent to ChatGPT. Figure 11 shows this process of going through a run of the system. These steps are very similar to the steps followed in [4], except that in this project we are using ChatGPT for natural language processing and A-Frame as our virtual reality environment. Also, something not indicated in the diagram is the use of the TTS model from OpenAI, which is used right after the user provides the voice command and after ChatGPT returns a response.

Figure 12 shows a series of snapshots of a video demonstrating the results of this deliverable. The video was taken using the Oculus Quest 2 headset, and we used its microphone to record the audio. We used the following commands: “Add a light blue floor to the scene” and “Add a light red box to the scene, and make it spin constantly.” These commands are immediately transcribed before they are sent to ChatGPT. In these images, we can observe how the project works within the virtual environment and how the users can provide modeling commands without the use of a keyboard or user interface.

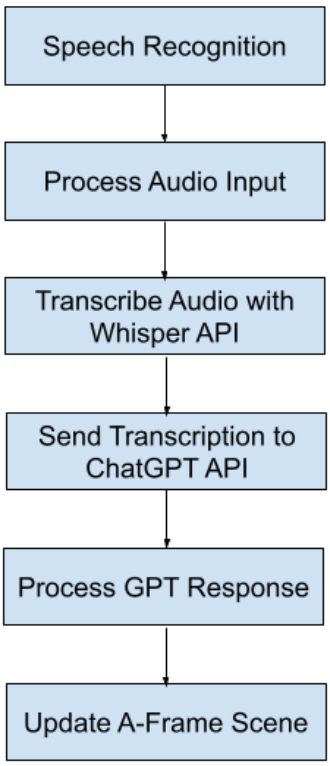

Fig 11. Process diagram of an example test run

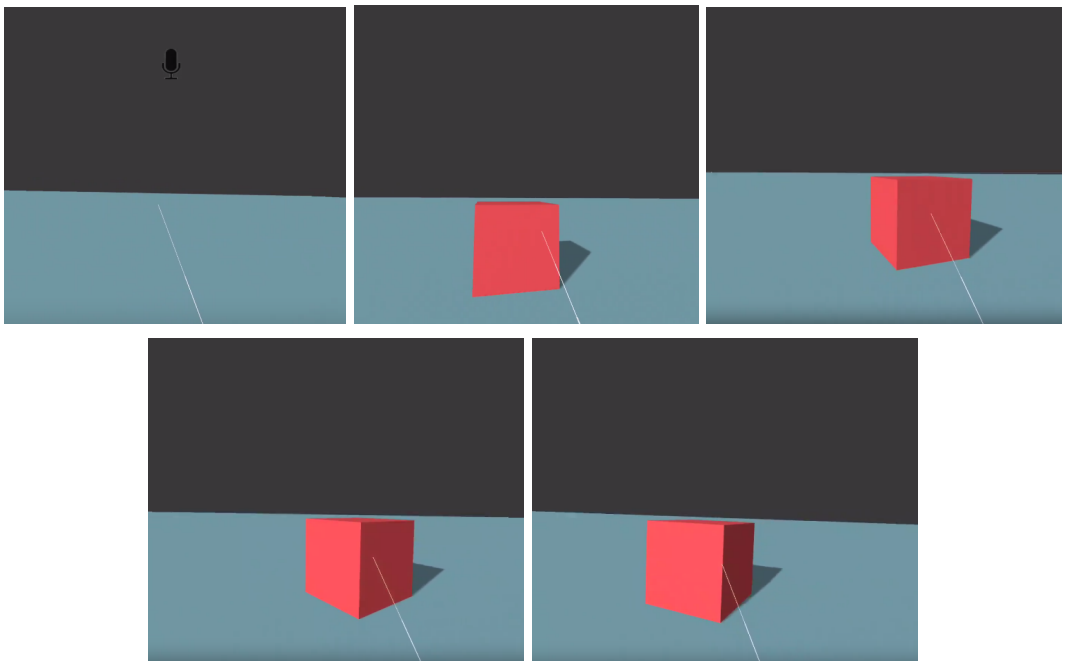

Fig 12. Sequential snapshots of virtual reality video test

VI. Conclusion

Through these four deliverables we were able to create an application that leverages ChatGPT's power to provide a more immersive and user-friendly 3D modeling experience. The first two deliverables were more focused on learning and understanding the relevant technologies of A-Frame and ChatGPT. The third deliverable provided the specific instructions for the system to behave as an assistant that interprets user's commands as A-Frame code. We performed a significant amount of testing for both simple and complex inputs as well as organized test sets to determine ChatGPT's consistency when generating A-Frame code. The last deliverable focused on adding speech recognition and audio support to our project. This not only removed the need of a user interface inside the virtual environment but also made the experience more engaging as the user provides 3D modeling voice commands in real-time. Moreover, as we move on to the next semester, we are looking forward to adding more features and expanding the project. While ideas are still being brainstormed, we do expect to add more instructions into ChatGPT to generate Javascript code so that we can create customized components that we can attach to A-Frame objects. We are also thinking about adding serialization into the project, so that if we have a scene or set of objects we want to reuse, we can save and then load up those objects in another session. With these and more features we will be able to further improve the functionality of the project and support more advanced 3D modeling capabilities.

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

[4] 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.