Shoutcast VR: Using Audio Inputs For Better Immersion

Jose Saldaa
Stanford University
450 Serra Mall Stanford, CA 94305
jsaldana@stanford.edu

Abstract

The efficacy of Virtual Reality is the level of immersion it can provide users for numerous tasks from video games to education. In this project, we will add audio input commands to contribute to this immersion. We will test the Unity Speech Recognition API in a VR environment by saying commands and then evaluating the outcomes.

1. Introduction

Speech Recognition has seen a boon in recent years in terms of research and implementation from AI assistants to language translation. Speech recognition, however, has not had its day with VR. This is most likely because most researchers/developers are focused on making VR more accessible to a larger audience by lowering the computational requirements needed to run VR environments. This is an oversight that developers shouldn't ignore. VR has become more mainstream, but there isn't a killer application that has elevated VR above a novelty for consumers.

Speech recognition could add a whole new level of immersion to almost any VR application. In video games, developers can use audio cues to make commands impossible or difficult to implement with some of the controllers compatible with modern HMDs. Or imagine a game such as Skyrim where you can respond in real time to NPCs for additional interactions.

Additionally, VR and speech recognition could a person improve their public speaking, by analyzing their speech and their eye movements to give feedback. These are powerful applications that are have been a rarity so far with VR.

This begs the question, "Can speech recognition be used in VR today?" Fortunately, the answer is yes, but is it good enough for modern applications? This project aims to use the current speech recognition API in Unity to create a VR environment for the user to issue audio commands that will make different Unity prefabs shoot from their right-handed controllers. I will test whether the current API is viable enough to create more VR applications with speech input.

2. Related Work

2.1. How does speech recognition work

Speech recognition has been around since the mid 1950s when Bell Laboratories built "Audrey" which was a machine that could recognize whether certain users spoke any number from 0-9. Jump forward a few decades and speech recognition has taken over everyone's devices from Siri to menu control. But how does speech go from you to your computer?

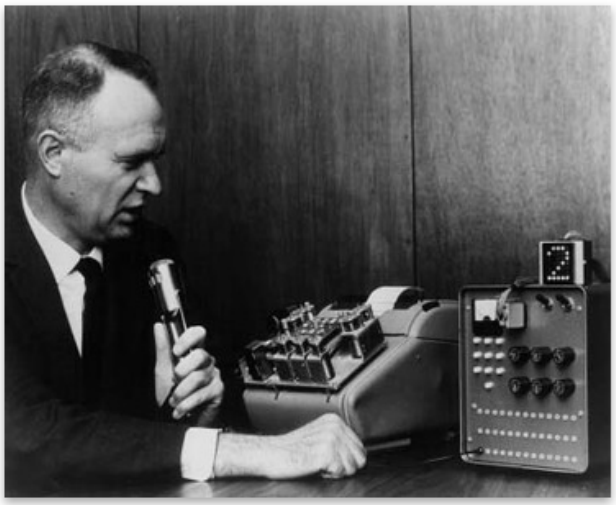

Figure 1. Audrey.

A person's speech/voice is vibrations which propagates as waves of pressure. These vibrations are converted into signals.[1] These signals are matched to phonemes, which are units of sound that differentiate one word from another in a language. Then, a statistical model analyzes the phoneme and the surrounding phonemes and compares it with a dictionary. If the confidence in the comparison between the phonemes and some subset of the dictionary is high enough then the machine can output the subset.

This is a very high-level explanation of speech recogni-

tion and the statistical model is the most complicated part of the whole equation as researchers and developers have labored for years for a highly accurate model.

2.2. Speech recognition in video games

Video games have approached speech recognition with mild curiosity. Games with speech inputs have mostly used it for minor gameplay opportunities such as navigating menus. However, some games such as Mass Effect 3 have used speech commands to operate heavy gameplay elements such as switching weapons or managing a squad.

VR game development has been similar to regular game development as speech recognition has only been used for menu commands for most applications. There are games such as SpaceForce: Orbital Dogfight which uses voice commands to operate a spaceship during fire fights, but there aren't many that use audio inputs with that complexity.

3. Game

The game uses an HTC Vive with the original Vive controllers.

Figure 2. HTC Vive.

3.1. Technology

I used the following hardware/software to help develop the project.

- 2018 MacBook Pro 15" Bootcamp w/Windows 10
- HTC Vive w/Vive controllers
- Onikuma Gaming Headset
- Unity 2018.3
- Virtual Studio Code

3.2. Game Design

The game uses Unity and SteamVR to implement virtual environments that can be interacted with a user with an HMD. SteamVR is a number of assets and materials made for Unity that lets developers easily add support for modern virtual reality devices such as the HTC Vive or the Oculus Rift.

The game uses these tools to create a rectangle environment with steel crates and wooden ramps where the goal of the user is shoot spells at teddy bears that are running around with voice commands.

The user can move by teleporting by clicking the touchpad (on either controller), aiming the arc that appears, and then releasing the touchpad to teleport to the location marked by the end of the arc. This was the easiest implementation for movement in order to focus on implemented the audio commands.

3.3. Speech input in Unity

In 2016, Unity released the KeywordRecognizer class which takes in user keywords and executes a function if one of the keywords is recognized. The KeywordRecognizer class also takes a second parameter which is a confidence (low, normal, high) defined by a Confidence enumeration.

A simple approach for adding the class to our scene was to create a script (UseSpell.cs in Assets/Scripts) which creates a new instance of the class, and executes the OnPhraseRecognized function when a keyword is recognized. This function instantiates Unity prefabs appropriate to the keyword recognized, and the script is attached to a GameObject in the scene.

3.4. Environment/Gameplay

The environmental is a big rectangle filled with crates and ramps. The environment has "teddy bears" which run and jump around and your goal is to try to shoot them using audio commands and aiming the right controller in the direction of the bears. The player can teleport by clicking the touchpad and releasing it when the desired location is marked.

The player can use four different commands: flame, fireball, meteor, and hover. Saying "flame" shoots a stream of fire from the end of the right controller. Fireball and meteor shoot a fireball and meteor, respectively. The hover command sends the user (10 in the world coordinate) up in the air where they can look down on the entire environment.

```
using System;
using System.Text;
using UnityEngine;
using UnityEngine.Windows.Speech;

public class KeywordScript : MonoBehaviour
{
    [SerializeField]
    private string[] m_Keywords;

    private KeywordRecognizer m_Recognizer;

    void Start()
    {
        m_Recognizer = new KeywordRecognizer(m_Keywords);
        m_Recognizer.OnPhraseRecognized += OnPhraseRecognized;
        m_Recognizer.Start();
    }

    private void OnPhraseRecognized(PhraseRecognizedEventArgs args)
    {
        StringBuilder builder = new StringBuilder();
        builder.AppendFormat("{0} ({1}){2}", args.text, args.confidence, Environment.NewLine);
        builder.AppendFormat("\tTimestamp: {0}{1}", args.phraseStartTime, Environment.NewLine);
        builder.AppendFormat("\tDuration: {0} seconds{1}", args.phraseDuration.TotalSeconds, Environment.NewLine);
        Debug.Log(builder.ToString());
    }
}
```

Figure 3. Example use of KeywordRecognizer.

Figure 4. A screenshot of the game.

The goal is to hit all the teddy bears in the game, but it is difficult as the bears run and do flips in the air to avoid

References

[1] E. . M. N. Morgan, Nelson / Fosler, “”speech recognition using on-line estimation of speaking rate”,” 1997. In EUROSPEECH-1997, 2079-2082. material .

oncoming fire from the user.

3.4.1 Challenges

The KeywordRecognizer class is not made for macOS, and therefore, I had to Bootcamp my MacBook. Additionally, the built-in microphone on the HTC Vive wasn't working as well as I wanted to – slowing down debugging/developing time – and I had to get an external microphone.

Additionally, the fireball and flame objects weren't firing in the direction of the right controller. Their position was the same as the controller in the scene, but the animation would fire both in the same direction. These objects have an initial direction that can't be changed (It was always firing at direction (0, 0, 1)). I fixed it by instead of trying to change the direction, I changed the rotation of the object to match the Quarternion of the controller when instantiating the objects.

The meteor object kept colliding with the player object if the initial position of the meteor was to close to the player.

4. Method

The main method to test Unity's speech recognition is to give Unity the audio commands. If the game executed the spell, then we know that Unity was able to recognize the command. I had three different commands with varying level of complexity: flame, fireball, and meteor. Additionally, I wanted to test how the confidence level given to the KeywordRecognizer would affect the outcome.

In order to test Unity's speech recognition, I wanted to make sure that the microphones wouldn't be an issue. The HTC Vive has a built-in microphone, but the quality of the microphone seems to be hit-or-miss for many consumers. Therefore, I tested not only with the HTC Vive's microphone, but also with an ONIKUMA Gaming Headset that I have used for gaming. The headset has an attached mic which comes just around two to three inches off the mouth. The manufacturers claim the headset has noise cancelling which would be ideal for this project. I didn't use the built-in mic on the computer as most people would not be near a computer when using audio commands (and in the case of the Oculus Quest, they wouldn't even need one).

As a result, I decided to test the speech recognition by uttering each command 100 times per confidence level (high, medium, low) and per microphone (built-in vs. gaming headset). I marked down the percentage of how each command was successful for each confidence level and microphone. I tried to use the same volume and cadence for each command, and I said the commands as if I was playing the actual game.

5. Results/Evaluation

First, I tested the audio commands for the built-in microphone on the HTC Vive.

Percentages for HTC Vive Built-in Mic			
Audio Command	Percentage High	Percentage Medium	Percentage Low
Flame	68	80	98
Fireball	78	91	99
Meteor	60	73	95

From a cursory glance, the results don't look to surprising as the quality of the built-in mic forced me to use the gaming headset when developing the game. The command ”meteor” on the high confidence level had the most trouble

being recognized by Unity.

Next, I did the same test but used the gaming headset.

Percentages for ONIKUMA Gaming Headset			
Audio Command	Percentage High	Percentage Medium	Percentage Low
Flame	92	99	100
Fireball	90	98	99
Meteor	90	99	100

The gaming headset is more accurate than the built-in microphone and is able to handle the command "meteor" on high confidence much better the HTC Vive could.

6. Conclusion/Future Work

Unity's speech recognition has its ups and downs. The built-in microphone had issues recognizing terms at the highest confidence level – especially the word "meteor." The gaming headset, on the other hand, was much more accurate in recognizing each command on high confidence. This could be for a few reasons. The mic on the gaming headset is much closer to the mouth, than the built-in mic. Additionally, the noise cancelling feature on the gaming headset might clear some of the noise that is going into the built-in mic. However, it's unclear what the HTC Vive's mic does to sound in terms of noise cancellation. Unfortunately, this means that speech recognition isn't suitable for the HTC Vive with the built-in mic, which would force consumers to buy an external microphone like I did.

However, once the confidence level was lowered, the built-in microphone was more accurate for each phrase, and the gaming headset was nearly perfect. There are two parts to this; First part, this is good if an application uses simple phrases and slight mistakes won't cause incidents. Second part, the system recognizes a huge amount of false positives which isn't great for more complex commands. For a video game like in this project, the lower confidence was ideal since I didn't care much about false positives, but this isn't the case for every application.

Unity's speech recognition seems to very good when tweaking the confidence level, but there are caveats that need to be considered.

- An external microphone is needed, at least for the HTC Vive, for better sound quality for the inputs. This could be cumbersome for consumers as it's an additional piece to wear and the microphone's chord might get in the way.
- I tested the speech recognition in an area of low environment noise. A more complex approach would to test against different levels of background noise and test the accuracy of the speech recognition.
- The commands are simple one-word commands. The KeywordRecognizer might not be robust enough for sentences or phrases of rising complexity. There are third party speech recognition systems that could be better if integrated into an Unity script.

Additionally, the tests in this project don't have the context that most VR applications have. For video games in a long session, users might grow weary from belting out commands and/or say the commands in a more slurred manner. Lowering the confidence level could help compensate, but false positives became an issue. I tried to use the same sound for each command on each try.

The project was also tested on my voice alone. People with different cadences/dialects/accents might have more trouble or more success with Unity's speech recognition system.

On the software side, I used a MacBook Pro, but I had to use Bootcamp to run the KeywordRecognizer clas as it was only available on Windows. It is unclear if the Bootcamp affected the performance of the speech recognition.

7. Acknowledgements

Thank you to the EE 267 staff for an incredible class. I learned so much about VR and have become interested in developing more applications for VR. This project was a ton of fun to develop and test.

References

[1] E. . M. N. Morgan, Nelson / Fosler, ""speech recognition using on-line estimation of speaking rate"," 1997. In EUROSPEECH-1997, 2079-2082. material .