IMPLEMENTATION OF AUDIO FILTER AND EFFECTS IN UNITY

Rathinkumar H. Kothiya

Computer Audio (CS 827)

University of Regina, Regina, SK, Canada
rathinkothiya@gmail.com

Abhijit A. Sathe

Computer Audio(CS 827)

University of Regina, Regina, SK, Canada
abhijitsathe1995@gmail.com

Abstract— **Digital audio processing is done by using different techniques to create various effects on audio signals. Audio Effects in games plays a vital role. It talks about the missing filters and effects in unity, then it describes the implementation of Tremolo effect in Unity as it has not been implemented in Unity yet. This paper also talks about the implementation an application known as the "underwater effect" which can be implemented using the tremolo filter. It then describes a different approach to use the unity's echo filter by making it more dynamic. It also talks about the real-time processing of audio signals. Furthermore, it compares the results of this research to the audacity's filter results. Lastly, it concludes the paper by taking about the future scope for this project and what improvements can be done.**

Keywords—Time-domain, Digital filter, Digital Signal Processing, clipping, power, amplitude, frequency, GameObject, RayCast.

I. INTRODUCTION

Unity is a game engine which is used to create cross platform standalone games. Unity can create two-dimensional as well as three-dimensional games and supports around 27 platforms. It is a powerful tool with numerous physics concepts implemented inside the engine which can be directly used with less scripting. Unity uses Javascript and C# as its programming languages. There are also number of audio filters inside the unity which are already implemented. [1]Those filters are as follows:

- Low Pass Filter
- High Pass Filter
- Echo Filter
- Distortion Filter
- Reverb Filter
- Chorus Filter

Various other effects such as Flange, Normalize, Parametric Equalizer, Pitch Shifter, Compressor and SFX Reverb are also implemented in unity. Above filters and effect can be used in most of the application.

Even though there are many effects already implemented in this game engine there are still some important effects missing. The main aim of this project is to explore these areas which haven't been implemented yet. One of these effects is known as the "**Tremolo**" effect. It is an Italian word which means trembling. [2] Using Tremolo filter, a shuddered effect or modulated effect can be created by slight and rapid change in the amplitude. Tremolo is often referred as the underwater effect. The implementation of the tremolo effect in not currently present in unity. To create an underwater effect the tremolo filter is required. But tremolo by itself generates a trembling sound. We need to mix this effect with a low-pass filter then the resulting effect sounds a lot like "Underwater effect".

Another effect that we implemented in this project is the Dynamic Echo effect. Echo is already implemented in unity, but we discuss a different approach to use the same filter. Echo is a delayed sound that is heard by listener along with the change in the original sound followed by an original sound. When a signal is sent by an audio source in a room with very less sound absorbing material, it hits the hard surface where some signals are absorbed, and the rest of it is reflected with the change in its original form.

[3][4] Echo can be heard by human if the time period of reflection is more than 0.1 seconds. The speed of sound is 343 m/s. So, the minimum distance from the reflecting surface should be

17.15 meters. The echo effect can be seen in the hall and conference room where size of room is more and have less absorbing material.

II. IMPLEMENTATION

In this section, we will discuss some of the filters and effects that we have implemented in this project and then will also discuss how the results are displayed.

The implementation is done in C# scripting language, where the script is applied on the "GameObject" present in the scene. This object is considered as the "SoundObject" in this project which is generally the source of the sound. This sound object is attached with a AudioSource Component which is responsible for playing the audio clips.

A. TREMOLO EFFECT

In this section, we will first discuss the implementation of the effect known as "Tremolo" This effect hasn't been implemented in Unity yet.

It is a well known effect in computer audio which comes under the category of modular effects. This effect works with amplitude of an audio file, i.e., it changes the amplitude of a particular audio file to create a trembling sound. This effect is also used to generate the "Underwater effect" which is done by mixing it with a low-pass filter.

As this effect is also known as the underwater effect, We have implemented a scene which consists of a swimming pool asset. This SoundObject(smart phone in this case) will be attached to the first person controller(Player). As the game starts and the scene loads a soundtrack will play in a loop on this "SoundObject". As soon as the player enters into the swimming pool the Tremolo filter is applied with Unity's low-pass filter to the soundtrack currently playing on the sound object and the modified audio is played back to the user. This effect is disabled as soon as the player comes out of the water and is enabled again when player dives in the swimming pool.

The "**Tremolo.cs**" script is attached to the SoundObject. The script first reads the PCM file playing on that GameObject sample by sample and then applies the effect and then again uploads the filtered file on to the "AudioSource" component of the object.

We are currently playing a sine wave which is currently stored in a PCM file known as "mystery_tones.wav" input in this implementation.

The general equation that we used to apply the tremolo effect is as follows:

Tremolo = (1 + Depth * Sin(2 * PI * samples * (speed / sample_rate)))

Where,

- Depth = limit of loss of volume in the effect.
- Speed = The effect which lets us decide the tempo of the audio.

The variables depth and speed are public variables, so they are accessible and can be edited through the inspector panel.

The figure below shows the scene in unity. It shows the simulation environment which is used for testing this particular effect.

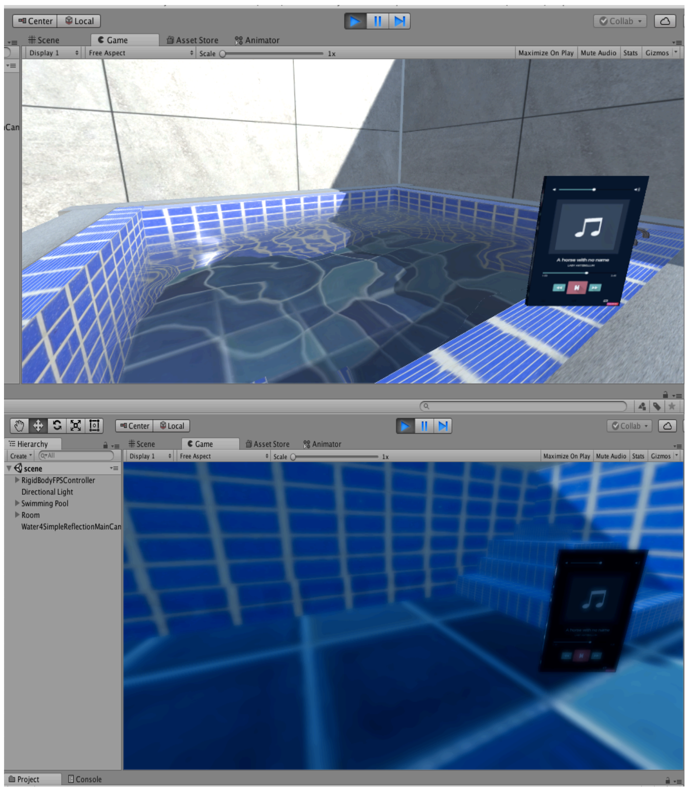


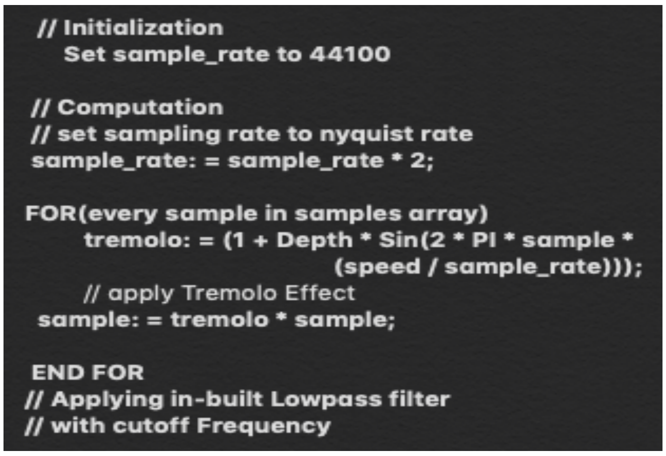
Figure 1: Unity Environment for testing Tremolo effect.

In the figure above, the SoundObject plays the original sound when it is outside the water. As soon as it goes underwater is applies the filter and plays the sound.

Pseudo-code:

Input: PCM Formatted file(.wav file in this case).

Output: Filtered PCM array of samples.



```
 // Initialization
   Set sample_rate to 44100

 // Computation
 // set sampling rate to nyquist rate
 sample_rate: = sample_rate * 2;

FOR(every sample in samples array)
      tremolo: = (1 + Depth * Sin(2 * PI * sample *
                              (speed / sample_rate)));
      // apply Tremolo Effect
  sample: = tremolo * sample;

 END FOR
 // Applying in-built Lowpass filter
 // with cutoff Frequency
```

As it can be seen from the above mentioned pseudo-code, each sample is read in an array called 'samples' and tremolo effect is then multiplied with each sample in the array.

The C# Script for the above-mentioned pseudo-code can be found in the link in the References [7].

B. ECHO EFFECT

In this section, we will discuss about creating a simulated environment in which we can play echo effects according to the distance travelled by a signal.

Unity already has a predefined echo filter which can be manipulated and can be used for the simulation of echo in a room. Another concept used in this implementation is the **Ray Casting** which helps to generate a ray from an object to the given direction and to trace that ray, the concept of **Line Rendering** is used which basically draws the path of the ray.

To get the path of propagation of a signal, we are using ray cast function which will help us to send the signal in the form of a ray. We are sending 3 rays at an angle 130°, 145° and 160° in the Y-axis. The reflection of signal (sound) from hard surface is can be described in the environment as the reflection of ray from the wall with the help of **Vector3.reflect** class of Unity. As some portion of the signal is absorbed by the surface, we are restricting the number of reflections in the room to 10.

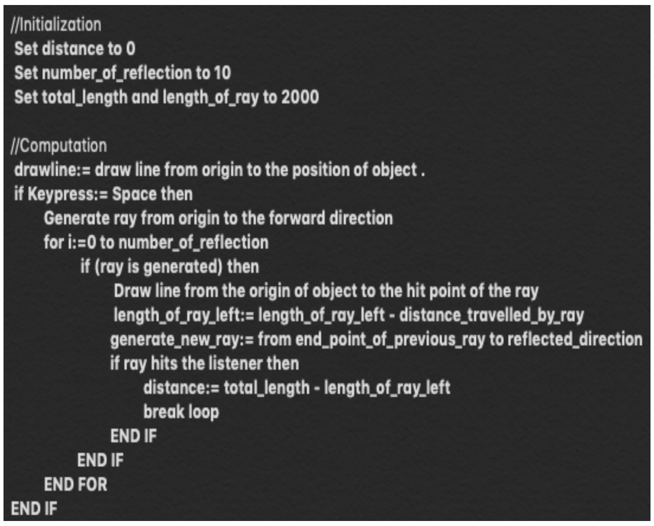
The ray is not visible to the user. To trace the path line rendering is used which draws the line of the ray. Line rendering requires starting and ending position unlike ray. So, starting position of a line is taken as an origin position of the GameObject from which ray is fired. To get the end point of the line, we use **RaycastHit** class in which we get the position at which the ray hits the collider. Similarly, this is again repeated by taking initial position as the previous hit point and end position as the ray hit point.

Similar concept can be applied to calculate the distance travelled by the ray. Maximum distance a sound can travel is set to 2500. Using hit point position and the origin we can get the distance of single ray and continuously subtract the distance of each reflected ray until it strikes the listener. We will get the total distance travelled by ray from the maximum distance which can be used to generate echoes. If it does not hit the listener, it will set distance to 0.

Pseudo-code:

Input: Starting point and direction of ray.

Output: Distance travelled by the ray if it hits



```
//Initialization
 Set distance to 0
 Set number_of_reflection to 10
 Set total_length and length_of_ray to 2000

//Computation
 drawline:= draw line from origin to the position of object .
 if Keypress:= Space then
     Generate ray from origin to the forward direction
     for i:=0 to number_of_reflection
          if (ray is generated) then
               Draw line from the origin of object to the hit point of the ray
               length_of_ray_left:= length_of_ray_left - distance_travelled_by_ray
              generate_new_ray:= from end_point_of_previous_ray to reflected_direction
              if ray hits the listener then
                   distance:= total_length - length_of_ray_left
                   break loop
              END IF
         END IF
     END FOR
END IF
```

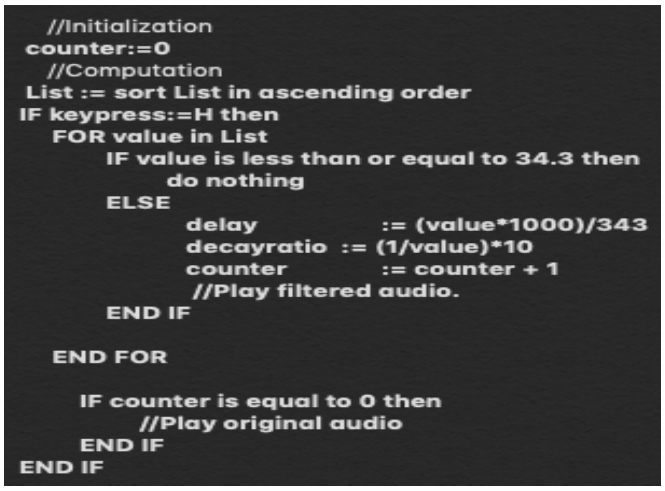
the listener otherwise will get the default value of distance i.e. 0.

Now, to generate an echo effect according to the distance travelled by the ray, we are storing the data of each ray in the list. This list is then sorted in the ascending order. So, we are continuously checking list from top to bottom, so that if the size of reflection is greater than the 34.3 meters only then it will play the echo according to the distance. The pseudo-code for the effect can be given as below

<u>Pseudo-code:</u>

Input: List of distance of the reflection of multiple ray(sound).

Output: Echo effects applied according to the distance travelled by ray(sound).



```
  //Initialization
 counter:=0
  //Computation
 List := sort List in ascending order
IF keypress:=H then
   FOR value in List
        IF value is less than or equal to 34.3 then
             do nothing
        ELSE
             delay            := (value*1000)/343
             decayratio  := (1/value)*10
             counter          := counter + 1
              //Play filtered audio.
        END IF

   END FOR

     IF counter is equal to 0 then
          //Play original audio
     END IF
END IF
```

As shown pseudo-codes, first code is used to calculate the distance travelled by the ray and second pseudo-code is used to play the echo effect according to the distance obtained from the first part.

C. REAL-TIME AUDIO PROCESSING

In this section, We are going to introduce the concept of processing the real-time audio. This can be done using the Unity's microphone function. This part of the implementation has not yet been completed and it is still in the development process. So this section provides a step-by-step procedure on how filters can be applied on real-time audio data.

Unity has an in-built Microphone function which reads the user's voice through microphone and has the ability to play it back to the user in real-time. The purpose of this implementation is to apply filters on the user input and then play back the edited audio back to the user. This processing will be mainly used in games where user wants to change their voice with a filtered voice that matches their character. The step-wise procedure is as follows:

- This can be done by first recording the user's voice from a microphone into an array.
- The array then can be converted into an PCM formatted file.
- The final step would be to apply a particular filter on PCM formatted file and the re-uploading the file on to the microphone's audio clip component.
- Once the file gets uploaded it can then be played back to the user with filtered audio.

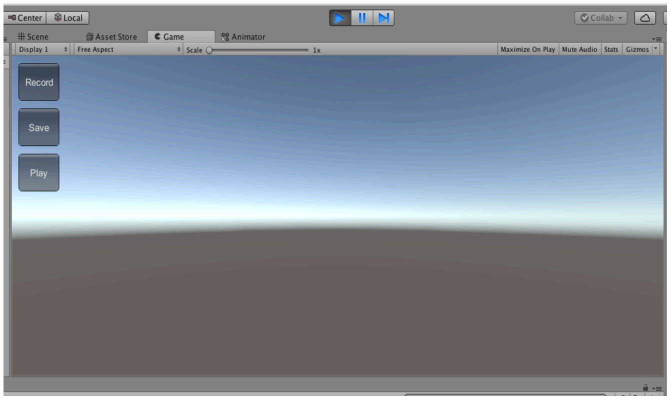


Figure 2: Unity environment for Recording Real-time audio

As it can be seen from figure 2, the user is provided with three buttons namely, "Record", "Save" and "Play". The User can first press Record button in order to start the microphone and also start recording the audio input.

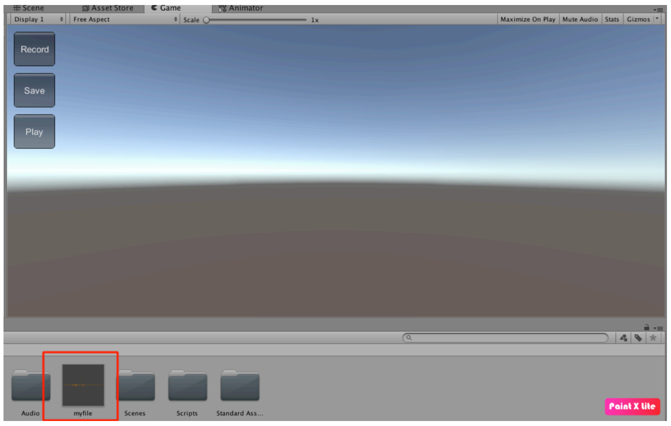


Figure 3: Saving the Recorded audio into a PCM formatted file.

As soon as the user hits the save button on the screen, the user's recorded audio is stored and converted in to a .WAV file which is a PCM formatted audio file. As it can be seen from Figure 3, the audio is saved inside the assets folder as .wav file called "**myfile.wav**". This is done using a **SavWav.cs** script. The link to the source code of this C# file can be found in the references[5] and [6].

The next step of this implementation would be to apply the filters on this saved PCM file and re-uploading the file. That part hasn't been implemented yet. This work can surely be completed in the future.

III. COMPARISON

A. COMPARISON OF TREMOLO EFFECT

In this section, We compare the results of the Effects implemented in unity with Audacity's Tremolo effect and we then compare the results.

The figure 4 shows the original sound file. We have used a sine wave called "Mysterytones.wav".

In figure 5, we can see the output generated by our Tremolo algorithm and Audacity's in-built algorithm.

As it can be seen from the above figure, both waveforms are almost identical. It can also be seen that the waveform generated by the audacity has a smaller amplitude as compared to our waveform. The audacity's Tremolo generates a waveform with amplitude of 1.8 whereas, our algorithm generates the wave with amplitude of 2.8.

B. COMPARISON OF ECHO EFFECT

As mentioned earlier, Unity already comes with an in-built echo filter but it is static in nature. Our implementation works on making the filter dynamic in nature. The unity's echo filter does not change its attributes according to the Audio Source's distance.

Our algorithm works on making this process dynamic, that means the echo filter attributes such as, delay, decay , etc., change

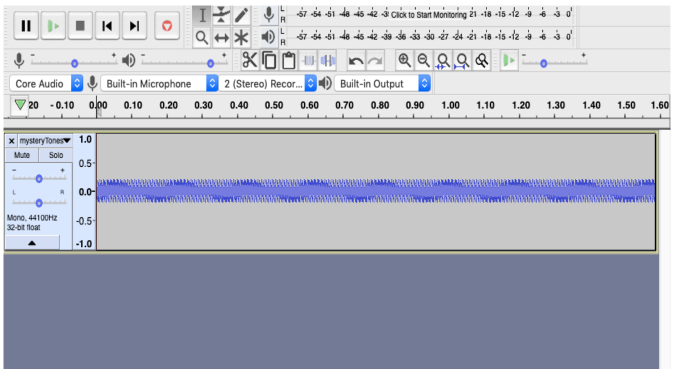


Figure 4: Original wave file

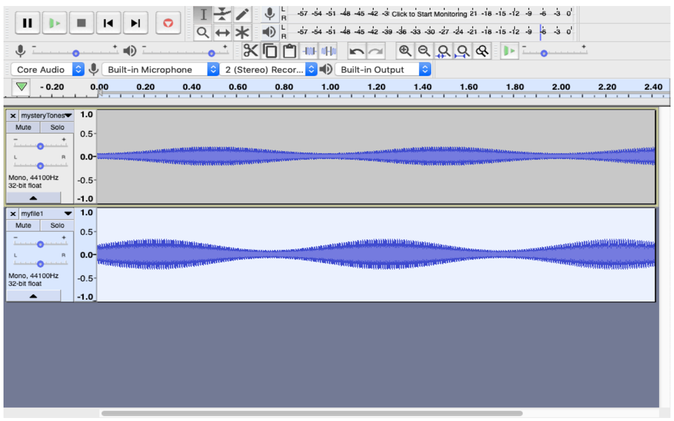
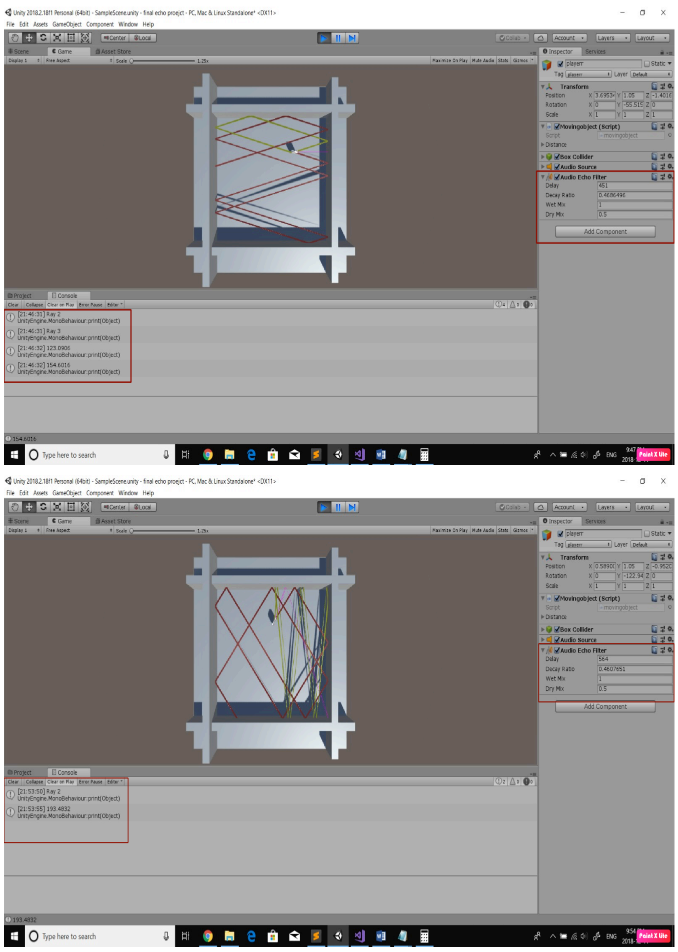


Figure 5: Comparing the results of Our and Audacity's Tremolo algorithms


Unity 2018.2.18f1 Personal (64bit) - SampleScene.unity - final echo proejct - PC, Mac & Linux Standalone* <DX11>

playerr, Tag playerr, Layer Default

Transform	X	Y	Z
Position	3.69534	1.05	-1.4016
Rotation	0	-55.515	0
Scale	1	1	1

Movingobject (Script): Script movingobject

Audio Echo Filter	
Delay	451
Decay Ratio	0.4686496
Wet Mix	1
Dry Mix	0.5

```
[21:46:31] Ray 2
UnityEngine.MonoBehaviour:print(Object)
[21:46:31] Ray 3
UnityEngine.MonoBehaviour:print(Object)
[21:46:32] 123.0906
UnityEngine.MonoBehaviour:print(Object)
[21:46:32] 154.6016
UnityEngine.MonoBehaviour:print(Object)
```

154.6016

Unity 2018.2.18f1 Personal (64bit) - SampleScene.unity - final echo proejct - PC, Mac & Linux Standalone* <DX11>

playerr, Tag playerr, Layer Default

Transform	X	Y	Z
Position	0.58900	1.05	-0.9520
Rotation	0	-122.94	0
Scale	1	1	1

Movingobject (Script): Script movingobject

Audio Echo Filter	
Delay	564
Decay Ratio	0.4607651
Wet Mix	1
Dry Mix	0.5

```
[21:53:50] Ray 2
UnityEngine.MonoBehaviour:print(Object)
[21:53:55] 193.4832
UnityEngine.MonoBehaviour:print(Object)
```

193.4832

Figure 5: Dynamic Echo effect

according to the user's distance. We are using Ray casting in order to calculate the distance and apply the filter.

As it can be seen from figure 5, the ray returns the total distance it has travelled and changes the delay and decay attributes of the echo filter dynamically. This doesn't happen in unity's in-built echo filter. By doing this, the echo heard by the player is of a higher quality as the values are set precisely according to the distance.

IV. CONCLUSION

In this research paper, we have discussed some of the implemented filters in unity. We have explained our implementations of tremolo and dynamic echo effects. We then have also compared their respective results and it can be seen from the comparison that our implemented Tremolo algorithm produces almost identical results as compared to audacity and Dynamic Echo approach provides user with better echo. We also discussed a new implementation approach that can be implemented to process the real-time audio data.

Although this project is not completely ready, it shows great potential and has a lot of scope in the future for improvements and can be applied in many different areas of game development.

V. FUTURE SCOPE

There are a several improvements that can be done on this implementations in the future.

The top thing on that list would be the completion of the real-time audio processing in which a filter is applied to the recorded audio and then the filtered file is played back to the user. Another feature that can be worked on is processing the file concurrently as the user is talking through the microphone and playing it back to the user at the same time.

Apart from real-time processing the Tremolo implementation can also be improved to match the exact output generated by the audacity's tremolo filter. Also we can improve the algorithm to calculate the decay function.

REFERENCES

[1] https://docs.unity3d.com/Manual/class-AudioEffect.html

[2] Dhananjay Sharma, Radheshyam Madge, Rishabh Mishra and Nissim Shewade, "A Sound Project Using Python", Nissim Shewade et al Int. Journal of Engineering Research and Applications ISSN : 2248-9622, Vol. 4, Issue 5(Version 1), May 2014, pp.08-11

[3] Sound Waves and Music, Reflection, Refraction and Diffraction , chapter 03 <https://www.physicsclassroom.com/class/sound/ Lesson-3/Reflection,-Refraction,-andDiffraction>

[4] Mcdonough, John & Wölfel, Matthias. (2008). Distant Speech Recognition: Bridging the Gaps. 2008 Hands-free Speech Communication and Microphone Arrays, Proceedings, HSCMA 2008. 108 - 114. 10.1109/HSCMA.2008.4538699.

[5] Calvin Rien (2012) Unity3D: script to save an AudioClip as a .wav file. [Online forum comment]. Message posted to - https://gist.github.com/2317063

[6] fil (Nov 26, 2012) Save audio to a file [Online forum comment]. Message posted to - https://answers.unity.com/questions/354401/save-audio-to-a-file.html#answer-form

[7] Rathinkumar H. Kothiya, github, November 2018 <https://github.com/rathinkothiya/Audio_project>