Getting Started with Binaural Spatializers **Ethan Geller**
MUS-470
Prof. Klassen Lab Date: 9/11/18

Summary

The goal for this lab is to build the beginnings of a binaural spatialization plugin and test it in a runtime environment. By the end of this lab, we will be able to experiment with key localization cues, with a focus on ITD.

Setting up your UPlugin

In order to test this engine, download the Unreal Engine 4.20.2 using the Epic Games launcher (epicgames.com). Next, copy the zipped example plugin into your engine's Plugins/Runtime directory. Alternatively, if you want to only use your plugin with a specific project, you can copy the plugin into that project's plugins directory.

There are several components of this plugin that will not be necessary to interact with during this lab. For the most part, we will be concerned with a singular set of files: `MySpatializer.h` and `MySpatializer.cpp`, Located in Source/Spatialization/Public and Source/Spatialization/Private, respectively.

For now, check out the struct `FSourceSpatializer`. this struct will hold everything you'll need to use, and ProcessSource will be your buffer callback for each individual source- the buffer contained in InputData contains a mono buffer of source audio as well as positional data, and OutputData contains a buffer meant to contain a stereo interleaved output for the left and right channels.

Exercise i. Write a basic panner in `FSourceSpatializer::ProcessSource`. This panner can either use linear or equal power panning.

Interaural Time Delay

Recall that ITD is defined as the difference in arrival times between the left and right ears. Robust ITD modeling can be done by either directly sampling an HRIR set or using the scattering function of a rigid sphere to model the head. For the purposes of this lab, we will simply use the direct path between the source and each of the ears to calculate the distance.

In order to calculate the ITD, we will need to know the locations of the left and right ears, as well as the speed of sound.

Exercise ii. Given a source's *azimuth* relative to the listener, as well as the *speed of sound* and the *diameter of the listener's head* (ear to ear), construct an algorithm that will determine the ITD.

Implementing the ITD

In order to properly emulate time delay, we will need to use delay lines. You may want to use the class `FDelay` for this, located in `DSP/Delay.h` in the AudioMixer module.

Exercise iii. Use `SpeedOfSoundCVar` and `HeadWidthCVar` for the speed of sound and the head width, and implement your ITD algorithm from exercise 2.

Observations

iii. You can adjust the head width parameter by pressing the button during playback and typing in `au.itd.SetHeadWidth`. Experiment with this value: What happens when you set it to too large a number? Too small a number?

iv. Place both of the example audio files in the level at two different locations in which they are both audible, but have a different ITD. The two example audio files are the left and right channels of a stereo piece of music and thus are closely correlated. What happens to the audio?

Where to go from here

v. Experiment with different ways to evaluate ITD beyond simply using the direct path.

vi. We have not dealt at all with Interaural Level Difference here. Experiment with either of the following:

- Applying attenuation to each ear based on it's distance from the source and the *inverse square law.*
- Applying a low pass filter that scales it's cutoff frequency with distance to emulate *head shadowing.*

vii. Pinnae filtering is a key cue for elevation and involves a complex set of comb filters caused by the physical dimensions and properties of one's outer ear. The end result of it can be loosely approximated with a notch filter around 10 kHz, with the notch gaining intensity as the elevation of a source increases. Implement this notch filter using an easily configurable set of parameters and see if you can successfully localize a source's elevation.