Title: DSP Coursework Report; Supervisor: Tillman Weyde
Name: James Box; E-mail address: James.Box@city.ac.uk; Student Number: 160071889

DSP Coursework Report

Introduction

This report details the steps I took to complete tasks 1, 2a and 2c. I chose to address 2a and the first task in 2c despite working on my own, as I was very interested in learning how to perform these tasks.

I used the code from Lab05 of the Computer Graphics module. This had already suitable objects implemented in the scene, e.g. a spaceship flying in a circle and a cube class. For task 2a, I downloaded a looping engine noise (https://freesound.org/people/bolkmar/sounds/420360/) and mainly edited Game.cpp, Audio.cpp, with minor changes in Camera.cpp and Cube.cpp and with appropriate changes in the relevant header files. Additionally, I used a song by my band for the music.

Task 1 - Creating a DSP that changes with input

Task 1 was to create a controllable FIR filter using the Filter Designer from MatLab, but implementing it using FMOD in C++.

I created a band-pass filter based on the example provided in the class resources. I performed this task iteratively, starting with the four-coefficient averaging filter that was provided as an example. Due to the sample being used having a high sampling rate, the four-coefficient filter did not do anything noticeable until I increased the number of coefficients to twenty. This resulted in a more noticeable effect. As a next step, I replaced the averaging filter values with those of the twenty-one coefficient filter created in MatLab. Once this was working, I replaced the hard-coded approach with a loop. Then I created a second filter in MatLab based on the second example from class, which is distinct from the first due to different settings. Finally, I used the sin and cos functions to interpolate between the two filters.

The code where this task was performed was in Audio.cpp. The functions and variable used were:

float coefficients[] //the first filter coefficients
float coefficients2[] //the second filter coefficients
Static float *buffer
FMOD_RESULT F_CALLBACK DSPCallback()

The callback function uses a circular buffer to store the values of the sound samples passed in via the inchannels parameter in the callback function.

I needed to declare the filter multiplier as global because the callback function was not a member function of the Audio class.

For the dynamic filter task, I created a new dsp callback using the first filter listed above. I attached this to a sound (https://freesound.org/people/qubodup/sounds/146770/#comments) that I play through PlayEventNoise. This is toggled on by the player pressing the 1 key. It loops until the player presses 1 again. In order to affect the filter dynamically, the 3 and 4 keys are used to decrease and increase respectively the amount of filter applied.

The properties I used for this functionality are listed below:
float coefficients[] //the same filter as the first above
bool switchOffLoop //used to stop the sound looping when the player presses 1.

I set the sound to come from the sphere located behind the player.
For this,it may be useful to turn down the music using the minus key (it can be turned up using the plus key) as well as actually approach the sphere to hear the sound properly.

Task 2a - Creating a 3D moveable sound source

Specifically this task is to create an object in the scene that can move with the keyboard or mouse. Taking a different approach, I created a moving object which the player can approach using the keyboard and mouse which is the source of an engine noise. I needed to calculate the velocity of both the object and the camera to create this effect, so I stored the previous location of each and used the current location minus the last location scaled with delta time (dt). This velocity was then passed in for both the sound emitting object and the camera, along with their positions, to the update function in the Audio class. Next, I converted these vector quantities into the FMOD equivalent. The appropriate environmental settings for 3D were set in the Audio Initialise function. This created the required result, with a spaceship flying in a circle, with a realistic doppler effect as the player and it pass each other.

Task 2c - Creating a sound-occluding object in the scene

Despite working individually, I decided to work on this task as I wanted to apply my learning. I adapted the cube class in the project, so that I could alter its shape (so it is no longer a cube, strictly speaking). I created this as a wall in the scene, with the player situated behind it relative to the spaceship. As the spaceship moves, however, it flies over the wall and above the player's starting position. The example code supplied for this coursework contained a function called CreateWall, which takes as parameters the centre of the object as a vector called position, as well as the width and height desired. It creates geometry using FMOD's addPolygon method, which registers this information in FMOD's view of the scene. The result can be heard when the

spaceship flies over the wall, as the player is suddenly able to hear it. If the player flies up over the wall, the occlusion effect is lost, as expected.

Summary

I enjoyed working on this coursework, and wished that it was possible to spend additional time in class learning more about FMOD. I feel that what I have accomplished here and the feedback I will receive will help to improve the quality of my projects in the future.

References

Ship engine noise (creative commons license) downloaded from here: https://freesound.org/people/bolkmar/sounds/420360/

Engine noise used by dynamic filter (attribution license) downloaded from here: (https://freesound.org/people/qubodup/sounds/146770/#comments

Song by Bloodwork from World Without End - End Time Records 2013 (composed by James Box, Jeff Mortimer, Michael Bryzak and Jon Rushforth)

Space ship (creative commons license) downloaded from http://www.psionic3d.co.uk