Cindy Bui
3D Audio
Final Project Paper

I. Abstract

Graphics processing units (GPUs) are computational accelerators. They were originally designed to handle real-time computer graphics processing but have been adapted to be used for high performance computing (HPC) in scientific research, machine learning, artificial intelligence, and several other fields. This adaptation of GPUs is known as general purpose computing on graphics processing units (GPGPU). The HPC community has furthered the literature on the usage of GPGPU for offline processing, but recent research has been trying to expand GPGPU into real-time processing, known as online processing. One facet of 3D audio that can benefit from the GPU is HRTFs and binaural audio. Traditionally, HRIRs needed to be very short, around 128 to 512 samples. Research has also been done on converting the HRIR FIR filters into IIR filters to reduce computational complexity. With GPUs, the reduction in computational complexity is no longer necessary. Instead, research can disregard the bottleneck of computational complexity and focus on ways to enhance user experience and perception. One of the major topics on HRTFs and perception is HRTF interpolation. This project and paper will be an application of GPGPU for HRTF interpolation, distance scaling, and a switching method.

II. Background and Motivation

Measuring a person's HRTFs is an incredibly tedious process. The speakers need to be setup very precisely, and the person needs to be absolutely still if the HRTFs are of a person and not a dummy head. The more positions there are, the better the resolution of the dataset. However, all datasets will have some sort of fixed resolution of HRTFs in terms of azimuth and elevation. This causes some artifacts in the time domain, especially with fast moving sound objects moving along a specific path. Typically, this is heard as an audible click or switch.

Research has already been done on HRTF interpolation methods, but these methods are limited to the computation power of the computers that they're done on.

As a solution to this, I wanted to implement an HRTF interpolation algorithm for GPUs that does not include any sort of truncation or reduction in quality.

III. Description - Theory

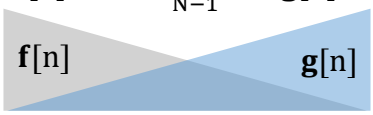
This project is an implementation of Jose Belloch's paper, "Headphone-Based Virtual Spatialization of Sound with a GPU Accelerator." His methodology purposefully selects techniques that do not decrease the length or the coefficients of the HRIRs. He performed a perceptual study and evaluation on switching and interpolation techniques.

A. Switching Algorithm

For switching techniques, he tried five different types of vectors and did a subjective test with a sample size of twenty people. He asked his participants to rate the different techniques in order of perceptual smoothness. The most preferred switching technique was a Fourier function, with a ramp function following closely after. This project uses the ramp function, which is a linear crossfade. Objectively, these two techniques have the lowest percentage of out of band energy. The equation below is for the crossfade function where $*$ denotes convolution, $\otimes$ denotes elementwise multiplication, and $\oplus$ denotes elementwise addition.

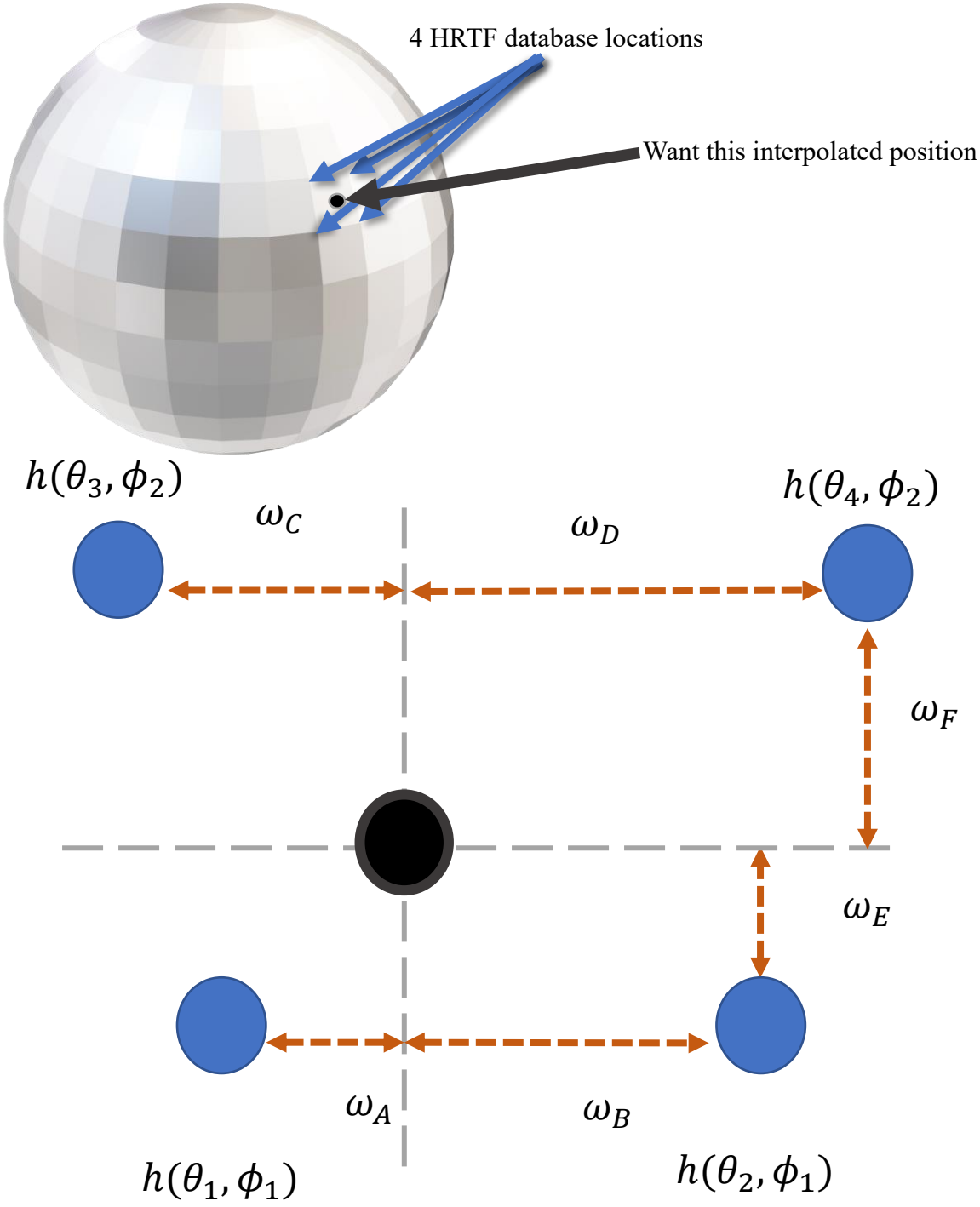
$$\mathbf{y_i} = \left((\mathbf{h}(\theta_{old}, \phi_{old}, r_{old}) * \mathbf{x}_i) \otimes \mathbf{f}\right) \oplus \left((\mathbf{h}(\theta_{new}, \phi_{new}, r_{new}) * \mathbf{x_i}) \otimes \mathbf{g}\right)$$

The ramp technique is where $\mathbf{f}[n] = 1 - \frac{n}{N-1}$ and $\mathbf{g}[n] = \frac{n}{N-1}$.



B. Interpolation Algorithm

In terms of interpolation techniques, he purposefully chose one that did not reduce the length or the HRIR coefficients in any shape or form. This technique was based off previous rational HRTF interpolation approaches.



For any given virtual HRTF position, up to four sampled HRTFs can be used to create it, as shown by the figure above. The end result will be the sum of the input convolved with all four sources, however the sources need to be scaled according to how close the source is to each of them.

$$\boldsymbol{\omega_A} = \frac{\theta_S - \theta_1}{\Delta\theta} \quad \boldsymbol{\omega_B} = \frac{\theta_2 - \theta_S}{\Delta\theta}$$

$$\boldsymbol{\omega_C} = \frac{\theta_S - \theta_3}{\Delta\theta} \quad \boldsymbol{\omega_D} = \frac{\theta_4 - \theta_S}{\Delta\theta}$$

$$\boldsymbol{\omega_E} = \frac{\phi_S - \phi_1}{\Delta\phi} \quad \boldsymbol{\omega_F} = \frac{\phi_2 - \phi_S}{\Delta\phi}$$

$$Y(\theta_s, \phi_s) = \omega_F \omega_B Y(\theta_1, \phi_1)$$

$$+\omega_F \omega_A Y(\theta_2, \phi_1)$$

$$+\omega_E \omega_D Y(\theta_3, \phi_2)$$

$$+\omega_E \omega_C Y(\theta_4, \phi_2)$$

C. Distance Virtualization Algorithm

For distance scaling, Belloch also gives an equation for a complex vector as a function of r.

$$\mathbf{R}(r) = \frac{e^{\alpha}}{1 + \frac{f_s}{c} \cdot (r - r_0)^2}$$

$$\text{where } \alpha = \frac{-j2\pi\frac{f_s}{c}(r - r_0)k}{N}$$

$$\text{and } k \in \mathbb{Z} \in [0, N)$$

When r is 0, the magnitude spectrum is an identity function. As r increases, the magnitude will scale down, but the phase response will begin to increase in negative slope, causing a constant group delay, which manifests as a time domain delay.

IV. Description – Coding and Technical

This project is named Jefferson in honor of a cartoon character created by a 3D media artist. It uses OpenGL to create a 3D interactive visualization of Jefferson and a sound source. The sound source can be moved with six degrees of freedom, and the audio changes accordingly. This project is written in C and C++.

A. Tools

OpenGL – short for Open Graphics Library. It is an API/library in several programming languages to draw 2D and 3D images. It's portable and it's implemented primarily in each computer's hardware.

ASSIMP – Acronym for Open Asset Import Library. It "is a portable Open Source library to import various well known 3D model formats in a uniform manner." This was used to import an FBX file into OpenGL.

Documentation: http://www.assimp.org

PortAudio – C/C++ library to play audio in real-time.

Libsndfile – C/C++ library to import wave files.

CUDA – acronym for Compute Unified Device Architecture. It's "a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs)". It is a proprietary but free API in several different programming languages to speak directly to NVIDIA hardware and utilize parallel processing.

General knowledge: https://developer.nvidia.com/cuda-zone

Download link: https://developer.nvidia.com/cuda-downloads

Documentation: https://docs.nvidia.com/cuda/

cuFFT – NVIDIA CUDA Fast Fourier Transform library.

General knowledge: https://developer.nvidia.com/cufft.

Documentation: https://docs.nvidia.com/cuda/cufft/index.html.

FFTW – Acronym for the Fastest Fourier Transform in the West. This is a C++ library to perform Fourier transforms on the CPU.

OpenMP – This is a library and SDK to do parallel processing on a CPU. The syntax looks like `#pragma omp parallel for` above a for loop.

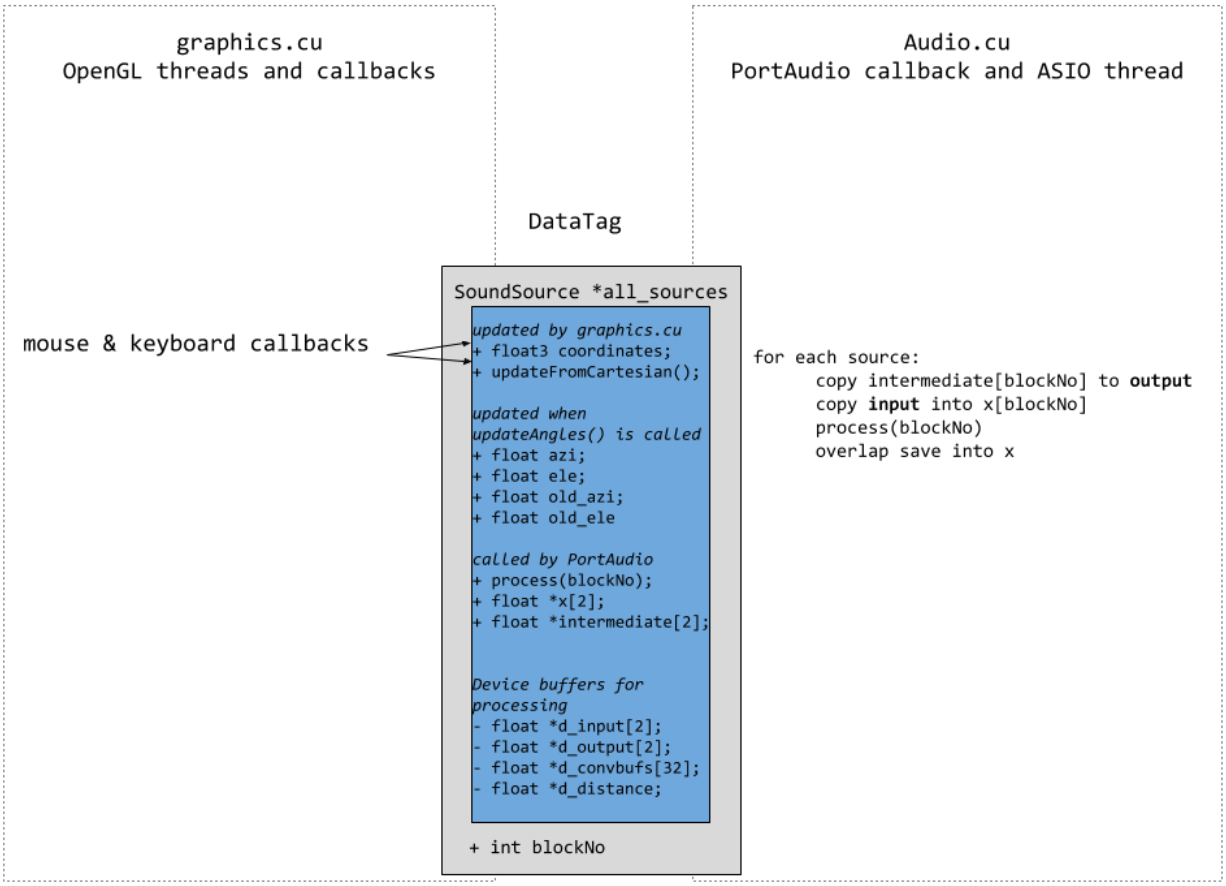
B. Program Flow

When the program first starts up, there is a lot of preprocessing that needs to occur before the visualization can begin. This primarily has to deal with loading external files and libraries into the program. The most notable external files are the input source, an optional reverberation file, and the fbx file for Jefferson. Another important aspect of the preprocessing stage is to process the first two buffers of audio for each sound source. This is because the audio is double buffered in the program. When the callback starts running, the output is filled with the previous callback's input after it has been processed. While not completely necessary, this allows extra room for the asynchronous GPU calls in case one of them takes longer than one callback function to compute.

- Preprocessing:
    - Reading audio files
        - Read input file
        - Read reverb file
            - (optional) convolve input with reverb
        - Read all KEMAR HRIR files
            - Transfer HRIRs to the GPU
            - FFT all HRIRs into HRTFs
    - Preprocess the first two buffers of audio for each sound source
    - Initialize PortAudio and start audio
    - Initialing OpenGL
        - Initialize callbacks

- Create VBO mesh of the floor
- Initialize the lighting and material properties
- Load the FBX model
- Start graphics loop

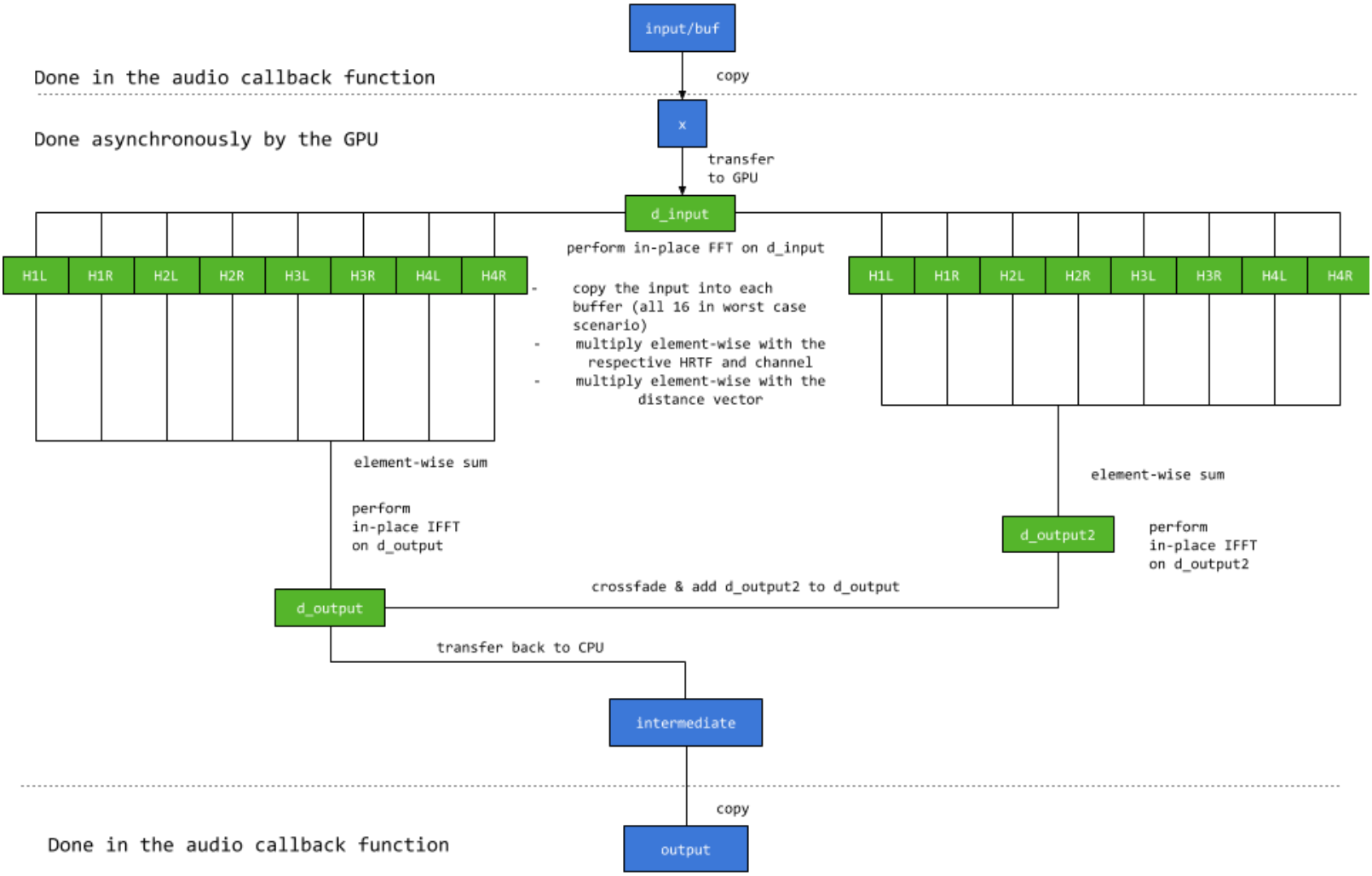
Real-time work flow



There is a single object that encapsulates all the data that gets passed back and forth between the OpenGL threads, PortAudio threads, and GPU threads. That object is of type DataTag, and it contains an array of sources. Currently, there is only one sound source. This infrastructure allows the number of sources to easily scale. Each source has several member variables and methods. The most important ones are `float3 coordinates`, `void updateFromCartesian()`, and `void process(int blockNo)`.

The graphics callbacks will update `coordinates` and call `updateFromCartesian()` to update the object's azimuth and elevation values. Asynchronously, the PortAudio callback

thread will copy the data from `intermediate`, play that out to the speakers, and then copy the newest buffer into `x`, call `void process(int blockNo)`, and perform overlap save on any of the two buffers whose GPU processing has finished. This diagram below details how the GPU performs the interpolation, switching, and distance scaling algorithms.



V. Conclusion and Future Work

After implementing these algorithms for a GPU, I found that the worst-case scenario where sixteen convolutions were necessary resulted in a processing time of 0.5 ms on a GPU. The current implementation only has one sound source, but Belloch's paper proved that it is

possible to follow this algorithm for 240 moving sound sources. This code is also the only open source public repository that uses GPUs for real-time audio.

As for future work, this project can be enhanced further for perceptual studies and objective benchmarking statistics. The code is currently in place to benchmark CPU algorithms and GPU algorithms with and without interpolating. Further processing and optimization can happen on both the GPU and CPU level. As of right now, each side of the tree in the last figure happens in the same GPU stream. The two trees run independently, but they run in series within each tree. This can be optimized so the eight elementwise multiplication operations happen in parallel with each other. For the CPU level, OpenMP can be used more extensively to launch separate CPU threads. These threads can even call their own GPU kernels.

An interesting future study will be on the percentage of GPU usage for audio compared to graphics. The graphical interface here was chosen because it is the easiest way to visualize 3D space and to have a user interact with objects in said space. However, this graphical interface is also being processed on the same GPU. Desktop PCs have the capability for more than one GPU, so it's possible to have one GPU for graphics and one GPU for audio processing. There was originally a vertex buffer array of a waveform of the input that passed through the source and towards Jefferson. As the sound source moved, the waveform would rotate and appear on the linear path between the source and listener. This VBO had to be removed since I noticed significant lag with the audio processing. A future iteration of this project can be on a desktop PC instead of a laptop. Alternatively, this can be turned into a VR experience where the user interaction happens using controllers.

In terms of perceptual studies, this code can be modified to allow different types of HRTF databases. Another interesting experiment would be to observe what the largest resolution

is for a smooth end user experience. The KEMAR HRTFs have an elevation resolution of 10 degrees and azimuth resolution of 5 to 7 degrees, depending on the elevation. A subjective experiment can be done where HRTF points are taken out, causing the azimuth resolution to be every 10 degrees or 15 degrees to see if the real-time interpolation can make up for the lack of resolution. That study can show the minimum required number of HRTFs required for a smooth perceptual experience. More research can also go into perceptually evaluating different interpolation algorithms that don't truncate or reduce the HRIR coefficients, because now there are devices that can operate on the computational complexity.

Invoking another area of GPGPU and real-time audio processing, this project also has the potential to be integrated with ray-traced room reverberation to allow for a complete 6DoF experience. Belloch's paper from 2013 set the gold standard for what a GPU can do with audio, and now it's time to build on it and push the standard even higher.

VI. References

Belloch, J. A., Ferrer, M., Gonzalez, A., Martinez-Zaldivar, F. J., & Vidal, A. M. (2013). Headphone-based virtual spatialization of sound with a GPU accelerator. Journal of the Audio Engineering Society, *61* (7/8), 546-561.

Keyrouz F., Diepold K., (Sept. 2006) "A Rational HRTF Interpolation Approach for Fast Synthesis of Moving Sound," Digital Signal Processing Workshop, 12th - Signal Processing Education Workshop, 222–226.

Keyrouz F., Diepold K., (2008). "A New HRTF Interpolation Approach for Fast Synthesis of Dynamic Environmental Interaction," *J. Audio Eng. Soc*, vol. 56, 28–35