3D audio source simulation on iOS devices

Orizio Riccardo,
Rizzini Mattia,
Zucchelli Maurizio

University of Brescia

xx july 2014

Overview

The project goal is to realize a 3D audio simulator for the iPad, in which an audio source can be moved using a graphical interface [or it follows the user's head orientation automatically].
To simulate a particular source position we use the KEMAR HRTF database and an interpolation that uses Delaunay triangulation.

Overview

The app is written in multiple languages:

- **Ruby** A *Ruby* script is used to translate the database in a *C++* header
- **Matlab** A *Matlab* script is used to perform the Delaunay triangulation offline
- **Pure Data** The audio I/O is managed using a *Pure Data* patch
- **C++** The functional core of the patch is a PD external written entirely by us
- **Swift** The new Apple's language for iOS systems is used to develop the graphical inferface of the app, which communicates with the PD patch

Outline

KEMAR Database Preprocessing

The KEMAR database is a list of HRTF recorded using a manikin with two microphones in place of the ears.
The HRTF are measured in the two ears separately, and each couple is associated with the source's position.

KEMAR Database Preprocessing

For the purposes of this project, being it written in C++, we decided to extract the useful information from the database and copy it inside a C++ header, in order to have them available as data inside global variables at runtime without any furter elaboration.
A Ruby script does the trick, by defining two multi-dimensional vectors, one for the HRTFs and one for the points' coordinates (distance, azimuth and elevation).

KEMAR Database Preprocessing- Delaunay Triangulation

In order to reach our goal, we need to estimate the three points closest to the source's actual position.
To do this, we decided to simplify the points' space by calculation a *Delaunay Triangulation*.
This is a way to split the space in a set of triangles such that the minimum angle of all the angles of the triangles in the triangulation is maximized.

KEMAR Database Preprocessing- Delaunay Triangulation

..Or, in other words, such that no triangle has a point inside of it.

KEMAR Database Preprocessing- Delaunay Triangulation

This subdivision of the space can be performed in many ways. We decided to use a Matlab script, since Matlab is great in this things. The script is called by the Ruby one, and the output is put in a third array in the C++ header, each position of the array containing the indexes of the points of a triangle.

Outline

Pure Data patch

//TODO insert patch screen here

Pure Data patch

The patch is very simple: it inputs the signal along with the source's actual azimuth and elevation as inlets to the block orz_hrtf∼, which produces as outlets the left and right channel of the filtered signal.
The whole filtering is performed by that block, implemented as C++ external.

Outline

HRTF processing and signal filtering

A Pure Data external is structure in a very precise manner and, actually, must be written in C (the passage to C++ is simply done by writing the needed methods in a wrapper).
A *setup* method is called when the block is loaded in the patch, and it must initialize a "class" with the inlets, outlets and the methods to call when a signal is given in PD. In particular, for the "signal" externals a method to process the *DSP* signal must be given.
A *new* method instantiates every element of the class after its creation by setup.
A method to handle the DSP signal, in our case called *perform*.

HRTF processing and signal filtering

The perform method receives the signal and azimuth, elevation and range parameters on the inlets, then proceeds in three separate steps:

- Find the triangle formed by the points that are closest to the source
- Determine the coefficients for the HRTF interpolation and calculate the HRTF in the source's position
- Filter the signal separately with the left and right HRTFs

HRTF processing and signal filtering- Triangle search

To search the triangle formed by the points closest to the source, we use the nth_element C function.
This function analyzes all the triangles produced by Matlab (and easily accessible as data inside a multi-dimensional vector) and finds the 2% having the vertices closest to the source, where the closeness is determined by a simple distance function.
The function puts the triangles, ordered, in a given structure.

HRTF processing and signal filtering- HRTF computation

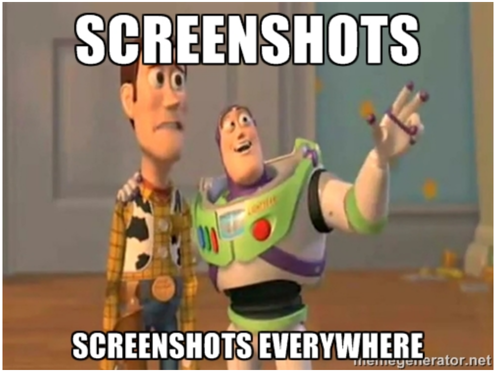
Given the set of 2% best triangles, we search for the first having coefficients for the linear combination of the HRTFs that are greater than zero.
These coefficients g of the combination are computed as //CMON equation Y DONT U WORK Where H is a matrix having the points' coordinates as column, and s is a vector containing the source's coordinates.
Having the coefficients, we normalize them to their own summation and, then calculate the source's HRTF as simple linear combination of the HRTFs of the points in the triangle.

HRTF processing and signal filtering- Filtering

The filtering is done by performing a convolution of the signal with the two computed HRTFs.
Past filters are kept as external's attributes to perform an overlap and add.
//FIXME extend?

Outline

The iPad interface



The iPad interface

The iPad interface