Stereo Source Separation Plugin

8903 Course Project Write-up

Annie & Shin

-About the project

The project comes as an audio plugin that works only for stereo input, utilizing the spatial information hidden in the stereo signal to accomplish source separation. The user manually specifies the azimuth and opening angle to pass or reject the sound source from that spatial location.

-About the algorithm

The source separation plugin is based on an algorithm called Azimuth Discrimination and Resynthesis (ADRess). It is a time-frequency processing utilizing the difference between left and right channels spectrogram resulting from the panning to filter out the specific source. So the real-time implementation requires a phase-vocoder block-by-block structure, requiring implantation of input and output buffer in addition to implementing the algorithm itself.

The algorithm tunes down the magnitude of one of the channels by ratios in equal intervals to find time-frequency points from left and right channels that cancel each other, regarding the cancelation point in the same tuning-down ratio from the same spatial source, and the tuning-down ratio is called azimuth in this context. After turning the cancellation points into spectral peaks, the peaks from the same azimuth can be resynthesized to construct the signal from one spatial source. To reduce artifacts, the algorithm allows resynthesizing peaks from a range of azimuths, so users can trade between separation effect and artifacts for their own preference. Thus there are two controlling parameters for the algorithm: the azimuth and the opening angle for sound collection.

- Upgrades from the original algorithm

During the algorithm research and implementation, we found some minor bugs and restrictions in the original paper, so we fixed bugs and extend the algorithm to get better sound.

In the original paper, when resynthesizing the source from the center, what is really resynthesized is the center source AND all the components that exist exclusively in the selected channel. We fixed this bug by expanding the azimuth range.

To get a better sound from the center, we apply two source separation processes instead of one, and put them respectively in the left and right channel output. Now the user can have stereo output when resynthesizing the sound from the center.

The paper does not talk about rejecting sources. The intuitive way to implement that is to do the source separation the other way round, just summing up all the magnitude peaks beyond the specified range. But one problem occurs that the source from the other channel cannot be included. We found out that the reason of the other channel being completely blocked is the calculation in converting cancelation point to magnitude peaks. We modified the calculation so that the signal from the other channel can be let in.

In addition to the algorithmic modification above, we added smoothing along the azimuth to reduce artifacts. We also added a frequency mask to pass or reject specific frequency components in the signal.

-Plugin usage design

The plugin should take in only stereo signal from the host, conduct the source separation and output the signal from a range of azimuths. The plugin should have a graphic user interface on which the azimuth and opening angle can be displayed and set.

-Plugin implementation

--Build a plugin in framework

One of the easiest ways to build up an audio plugin framework is Juce, which wraps up the APIs of different plugin formats, providing developer with a generic interface for plugin development over different platforms. Some small functions like post-script that automatically puts plugins into the plugin folder also saves the developer from tedious work of moving files.

-Class structure

The class structure of the project is straightforward. PluginEditor and PluginProcessor classes made by jucer are respectively responsive for the interface and audio processing. All algorithm-related parts are concealed into the ADRess class, which takes in blocks of audio samples, do the source separation and save to output back to the original buffer.

--ADRess class API

ADRess class is the core algorithmic part of the project. The interface of this class only includes get and set parameters and process data.

The class is initialized with two arguments: the block size and a beta value that is the azimuth resolution. Once initialized, these two values cannot be altered.

There are three parameters in this class. The first parameter is the plugin status, whether it bypasses the signal, extract the specified source or suppress the source. The second parameter is direction, the azimuth from which the user would like the sound. The third parameter is width, the opening angle for sound collection.

The data process interface is process function. To address the restriction that this algorithm only works for stereo signal, the two input arguments are left and right signal buffer separately. After the processing, the data will be saved back to the input buffer.

--PluginProcessor

The PluginProcessor defines the data behavior of the plugin. It initializes buffer, sets and gets the parameters and processes the input signal.

processBlock is the function that takes in blocks of data from the host and processes them. As the core algorithm is concealed in ADRess class that takes in data of FFT block size, the role of processBlock in PluginEditor is sending ADRess class blocks of data with a specified hop size.

An input and an output buffer are implemented in place within the processBlock function. They are sample-by-sample circular buffer, that checks at each sample whether there has been enough input samples to send ADRess an FFT block to process. After processed by ADRess, the data will be overlap-and-add in the output buffer. The proper setting of the initial value of outputBufferWritePosition\_ ensures the minimum delay of the output.

The block size and hop size is currently immutable in the project. The preset block size and hop size is necessary for the quality of source separation, which is tested in algorithm prototyping with Matlab. The computational power of most ordinary computers is able to endure release build with the current block size and hop size.

--PluginEditor

这部分仿照上面来写可以不用太多

-Graphic user interface

简介操作，贴图

-Third party library

Kiss\_fft, a third-party library is used for FFT in this project. It is in place, working only after including the source files. With an estimated speed around 30% slower than FFTW, Kiss\_fft is sufficient to carry on the real-time processing work. Another good point of this library is its variable type. It is using std::complex for ordinary FFT, and there is a choice of float-std::complex for real number FFT. The float number for real number input is easy to read and write, and the std::complex enables the usage of complex calculation such as plus, multiplication, std::abs, std::arg and std::polar, which makes computation really easy.

-Results

Although there are objective evaluations for source separation algorithms, they are abstract to some degree and not necessarily useful in this context of plugin development. We mainly look into the subjective sound quality and stability of the plugin for the evaluation of the project.

The plugin has a 4096 block size and 3/4 overlapping rate, high enough for a reasonable sound quality after phase-vocoder processing. The signal is added Hann window and scale down by the factor of 2 when bypassing, or is added a Hanning resynthesized and added window again when conducting source separation, ensuring no distortion with bypassing few clips with resynthesized signal. There are typical phase-vocoder artifacts in the output, like the flickering grains of sound from isolate peaks in the spectrogram, or the bathroom-like artifacts. But by increasing the opening angle, the artifacts can be significantly reduced, bringing a much more natural sound.

The plugin has good stability and responds to user operations fairly quickly. There are few crashes in the final version of the plugin. When dragging the arrow and slides on the GUI, the components and parameters are reacting responsively. Thanks to the sensitive parameter change and the nature of ADRess itself, there are few clicks when changing the parameters.