libfftaudio API

Provides common interface for performing audio fft analysis using *fftw3* and *cuda* libraries. The version compiled is determined by the compilation flags 'USE_CUDA_API' and 'USE_FFTW_API'. Both versions can be compiled to the same or different library names; which version is used is determined by which one is linked to.

Class: FFTAudioBase

Abstract base class providing a common interface for all implemented api's.

Class: FFTAudio

Implementation of FFTAudio class, using either *FFTW3* or *Cuda* interfaces, depending on compilation flag.

Class FFTAudio()

Class providing implementation of common api.

- *`FuncInitWindowCB`* `window_type`
  - FFT window to use (a function declared in *fftaudio_windows.h*)
  - `Rectangle, Triangluar, Bartlett, Sine, Hann, Hamming, Welch, Blackman, Nuttall, BlackmanNuttall, BlackmanHarris, FlatTop`
- *`int`* `sample_rate`
  - Sample rate of window's audio, in hertz
- *`int`* `window_size`
  - Size of window, in samples
- *`int`* `padded_window_size`
  - Size the window data should be padded to, 0-valued samples are concatenated to the window data to reach this padded size. Values less than or equal to '*window_size*' result in no padding.
- *`int`* `batch_count`
  - Number of windows to concurrently perform FFT on

initialize()

This must be called before using any other functions of the class.

Returns '*fftaStatus*' type (see *ffta_status.h*), should always be checked, any failure renders the object useless (all calls to execute() will fail)

execute()

Executes a batch of FFT's.

- *`const short`* `*data`
  - Pointer to 16-bit signed sample data. Size should be '*batch size * window size*', with each window's data arranged consecutively.
- *`const short * const`* `*data_ptrs`
  - Pointer to array of pointers to 16-bit signed sample data. The array of pointers should equal '*batch count*', and each array pointed to should contain '*window size*' samples.

getBinValue()

Retrieves a bin result value after *execute()* is called. May optionally call a user callback to process raw value before returning it (see *setGetBinValueUserCallback()*)

- *`int`* `batch_index`
  - The batch index to get the bin result for
- *`int`* `bin`
  - Bin index to get result for, from 0 to N. 'N - 1' is equal to ***'padded frame size / 2'***, 'N' is the Nyquist frequency bin.

Returns *float* value with batch/bin result.

setGetBinValueUserCallback()

Sets optional user callback called when *getBinValue()* is called. This allows additional processing to be performed before the result value is returned.

The bin values are automatically processed by the following:

```
p1 = real^2 + complex^2
p2 = sqrt(p1);
p3 = p2 * 2.0
p4 = p3 * window_sum
```

The callback is called after the automatic post-processing is done.

`FuncGetBinCB cb_func`
: callback function, see below for description of *FuncGetBinCB* type

`void *user_ptr`
: optional user-specified pointer passed to callback function

FuncGetBinCB Type

Callback function type for post-processing bin results.

`int bin_index`
: index of bin (0 --> 'padded_frame_size' / 2)

`float &bin_value`
: input (raw) / output (modified by callback) bin result value

`void *user_ptr`
: user pointer associated with callback

Returns *void*

Other Functions:

```
int    getSampleRate()
int    getFrameSize()
int    getPaddedFrameSize()
int    getBatchCount()
int    getBinCount()
float  getBinFrequency(int bin)
```

FuncInitWindowCB Type

Callback function for initializing window data. Only needed if making custom window functions.

`int frame_size`
: Frame size (unpadded) for window

`float &window_sum`
: (input) arbitrary window parameter, depends on window implementation
: (output) the sum of the frame's sample multipliers, the value a sample is multiplied by when applying the window

`float *output`
: (input) Array of '*frame_size*' floats provided by caller
: (output) '*output*' is filled in with the frame sample multipliers for each sample index