# Graal library’s build and programming guide.

**Abstarct.**

The document describes the current state of AMD Graal liabrray – GPU-base audio- accelerating library.

It supports a low-latency block convolution of an extremely long reverberation filter with a continuous audio stream with arbitrary number of sub-channels.

4 different flavors of algorithm are developed (see “Large convolution with a strict time constrain” white paper).

An independent FIR filter pipeline is available for a convolution with relatively small filters – less than 2K samples- with any number of sub-channels.

Full SW stack includes static library, flexible stand-alone application and VST2/3 plugging wrapper.

**SW package.**

The Graal SW package consists of 3 parts – a statically-linked library, a stand-alone application for testing, verification and performance measurement, a VST plugging wrapper.

Build process.

The Graal library together with the stand-alone application are built with the Graal.sln, VC++ 10 project.

Upon successful built the library, the interface definition file – Graal.h- and OpenCl kernel files are copied into a proper directories of the VST development tree.

Please, update the second argument in the Post-build event window in

Graal-> Property->Configuration Properties->Build Events->Post-Build Events

to a correct location of the ..\audio\.. tree on your local machine.

Name of the debug version is Graal32d.lib, release version is Graal32r.lib.

 An example of Graal VST pluging wrapper is located in

..\audio\vstsdk360\_22\_11\_2013\_build\_100\VST3 SDK\public.sdk\samples\vst\amdreverb

It is built with the amdRvrb\_vst2\_vc10.sln, VC++ 10 project, located in

…\audio\vstsdk360\_22\_11\_2013\_build\_100\VST3 SDK\public.sdk\samples\vst\amdreverb\win

The Graal library has to appear in

…\audio\vstsdk360\_22\_11\_2013\_build\_100\VST3 SDK\public.sdk\samples\vst\amdreverb\win

together with 6 .cl files.

Graal.h file has to appear in

…\audio\vstsdk360\_22\_11\_2013\_build\_100\VST3 SDK\public.sdk\samples\vst\amdreverb\source

To build the Graal VST plugin and to move it automatically into a proper application directory, please, update

graalVST2->Property->Configuration Properties->General->output Directory

to a location of your application’s VST plugins.

If the location of the application VST plugins are inside a system tree – Program File, Program Files(86) – please start VC10 as “administrator”

The name of the successfully built VST plugin,  both debug and release versions, is graalVST2.dll

**Graal library interface.**

|

int graalInitialize(graalHandle \* handle, const char \* ocl\_kernels\_path, int init\_flags);

returns a handle to the library context, that is going to be used in all following calls.

Possible initialization flag combinations:

\_\_INIT\_FLAG\_FHT\_\_ - uniform convolution.|

\_\_INIT\_FLAG\_FHT\_\_ | \_\_INIT\_FLAG\_HETEROGEN\_\_ - uniform heterogeneous convolution.

\_\_INIT\_FLAG\_FHT\_\_ | \_\_INIT\_FLAG\_2STREAMS\_\_ - non-uniform convolution.

\_\_INIT\_FLAG\_FHT\_\_ | \_\_INIT\_FLAG\_2STREAMS\_\_ | \_\_INIT\_FLAG\_HETEROGEN\_\_ = non-uniform heterogeneous convolution.

\_\_INIT\_FLAG\_FIR\_\_ - FIR pipeline.

For the use of initialization flags please see the Main.c file of the stand-alone application and the again.cpp file of the VST plugin sample.

Pipeline termination

int graalTerminate(graalHandle handle);

Set convolution block size.

int graalReverbSetBlockSize(graalHandle handle, int block\_size);

Set number input and output channels

int graalReverbSetNChannels(graalHandle handle, int n\_input\_channels, int n\_input\_subchannels, int n\_output\_channels, int n\_output\_subchannels);

Currently only single combination - 1, 2, 1, 2 - is accepted.

Load a reverberation kernel from an audio file.

int graalReverbSetupReverbKernelFromFile(graalHandle handle, const char \* file\_loc);

The routine reads, unpacks the kernel and uploads it into the GPU memory.

Check whether the reverberation kernel has been loaded and is ready to processing.

int graalReverbIsActive(graalHandle handle);

The routine returns 1 only when kernel data has been loaded and all kernel initialization steps have been successfully completed.

Process the new block of data for each sub-channel.

int graalReverbProcessing(graalHandle handle, \_\_FLOAT\_\_ \*\* input, \_\_FLOAT\_\_ \*\* output, int numSamples, int flags);

Possible flags:

\_\_PROCESSING\_FLAG\_VERIFY\_TRANSFORM1\_\_ - verify the FIR convolution or the FHT transform.

\_\_PROCESSING\_FLAG\_VERIFY\_TRANSFORM2\_\_ - verify the large FHT transform in the non-uniform convolution mode.

The routine is the library’s main calling point.

It calls different pipeline implementations depending on the library instantiation flags.

For examples of using the interface see Main.c or again.cpp files.

**Stand-alone application.**

The application’s source code is in a single file - Main.c.

The application is controlled by a set of arguments to make it into a front-end for debugging, performance measurement and verification.

A brief description of application parameters.

The application can run all possible pipeline implementations with different block sizes.

An arbitrary kernel and input files can be defined.

Both input data and kernel could be generated out off random data and can be of an arbitrary size.

The number of input data blocks sent from the application is arbitrary.

A verification of the pipeline can be done on a block and a sample level.

The pipeline GPU-based forward/inverse transforms could be verified.

All arguments and their description available by running the application with –h[elp] argument:

>GraalDriver –h[elp]

**VST2/3 pluggin wrapper.**

The plugging is developed based on open source VST3 SDK, vstsdk360\_22\_11\_2013\_build\_100.

It’s a modification of the again SDK sample with few additions and modifications

In particular, a windows file selector block has been made available. It requires a special define (VSTGUI\_NEW\_CFILESELECTOR), see winfileselector.cpp.

Again.cpp uses following Graal library interfaces:

graalInitialize

graalSetNChannels

graalTerminate

graalIsActive

graalProcessing

graalSetBlockSize

It sends the graalHandle handle - m\_amdRvrb - to the controller (see lines 161- 165).

The controller is implemented in the againcontroller.cpp file.

It accepts graalHandle handle, see AReverbController::notify,

obtains the kernel file name from the editor,

loads the kernel data from a .wav file in AReverbController::amdOCLLoadIR.

Selecting a kernel file name.

The editor is implemented in the againeditor.cpp file.

“Open!” button is mapped to “Send” message tag.

On processing the “Send” tag the AGainEditorView::runFileSelector is called (see line 365).

AGainEditorView::runFileSelector eventually calls VistaFileSelector::runInternal (see winfileselector.cpp).

VistaFileSelector::runInternal selects the file name and notifies the editor through AGainEditorView::notify.

The editor notifies the controller by calling AReverbController::amdOCLLoadIR.

Possible interface and design improvements.

1. to map a number of inputs/channels to a number of filters/filter channels;

2. to keep multiple reverb filters/FIRs;

3. to be able to replace filters easily;

4. to merge filters programmatically (linear combination or other forms);

5. to add gain or other (non-) linear transformations of the (input?/) output;

6. to make library easily expandable.