# CUDA-SCN Usage Document

The TLM-SCN CUDA project consists of a 'gpu' class. This class contains the necessary methods to create an SCN Mesh structure, as well as boundaries, excitation structures, as well as sampling probes. In this example project, a WR-28 filter was implemented. This document walks through the process of coding the filter structure where boundaries, excitation, sampling among others are described. As of this writing, the CUDA driver used was version 2.3, although version 3.0 is currently being debugged.

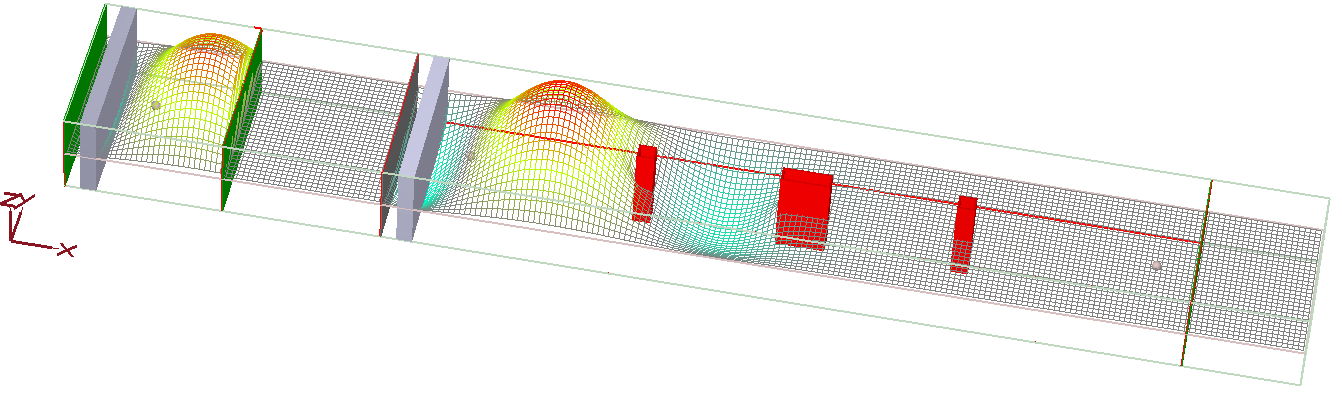


Figure : WR-28 Filter example included

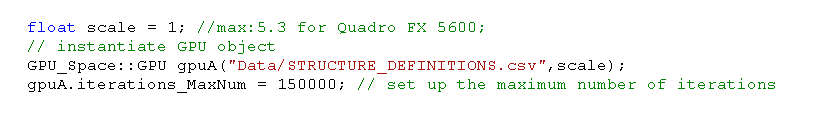


Figure : Instantiating the GPU object

**GPU Constructor:** Instantiating the GPU object is shown in figure 2, where the "STRUCTURE\_DEFINITIONS.csv" file contains the definition of the overall mesh dimensions, as well as all boundary definitions. Scale can be also passed as part of the constructor, where 1 (float type) should be used initially. The scale is used to test the same structure at various scales for performance.

**Iterations:** Iterations can be set within the main program, however at this time, iterations have been hard-coded.

## STRUCTURE DEFINITIONS

A TLM-SCN structure can be defined by editing the "STRUCTURE\_DEFINITIONS.csv" file, an example of which is shown in figure 3. This file can be found in the DATA folder. If it is desired to create a new structure, the it is recommended to use an EXCEL tool also found in the DATA folder called:

"STRUCTURE\_DEFINITIONS\_Tool.xlsx"

This file contains instructions on the first tab on how to create a mesh structure, where resolution and absolute dimensions of boundaries can be entered (in meters). The second tab of the excel sheet will generate node dimensions. This can be copied and pasted into the "STRUCTURE\_DEFINITIONS.csv"file.

The "STRUCTURE\_DEFINITIONS.csv" file's entries are described as:

* "Mesh Dimensions(X Y Z) " -defines the X,Y,Z dimensions of the 3D mesh (Note: the X mesh dimension must be a multiple of 64 until the CUDA-SCN code can be altered to compensate)
* "Resolution(m)" - define the distance between nodes
* "#Boundaries" - number of boundaries that are defined in the later part of this file
* "Ref Coef Index"- an incrementing index used along with "Ref Coef" to help define an array of reflection coefficients to be used by this structure.
* "Ref Coef" - a list of reflection coefficients that will be used by the boundaries
* "Filter" Column - Name of boundary (discarded upon CUDA code read)
* "Ref Coef Index" Column - uses list of "ref Coef" as a lookup to associate reflection coefficient with specified boundary
* "Xmin,Xmax...Zmin,Zmax" - Define dimension of boundary (must be a plane, not a 3D block)



Figure : Sample "STRUCTURE\_DEFINITIONS.csv" file

## Excitation:

# 

Figure : Excitation

At least two steps are required to include excitation.

**"excitation\_CreateStructure(..) :** First is to create an excitation structure object with "excitation\_CreateStructure(..)" (figure 4) where propagation direction, polarization, and the excitation plane dimensions are defined. The dimensions must be defined as a plane or an error will be returned. This method returns an excitaiton\_ID to be used later.

**" excitation\_generate\_SineGaussian\_array(..)":** The Second step is to generate the excitation itself. In this case a gaussian sine wave. The parameters of which are listed in figure 4. This will embed the gaussian sine wave excitation into an internal array within the excitation object created previously.

**"excitation\_Save(..)":** The excitation waveform can saved to a csv file.

## Sample Probe

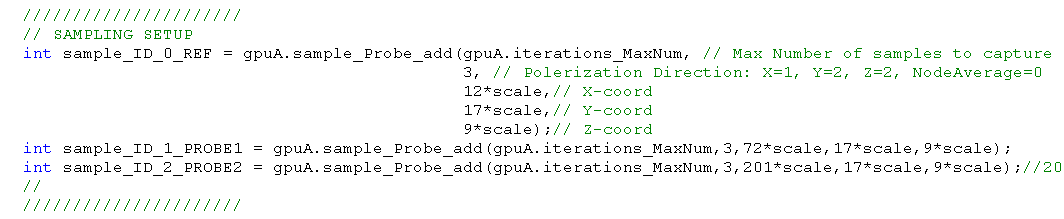


Figure : Sample Object Creation

Sample probe object are created next. Polarization direction can be set captured or the average of a nodes voltages can be sampled.

## Enable Timer

# 

Figure : Timer Enable

A timer can be enabled to measure the elapsed time of the iterations. On-screen status updates can also be enabled (figure 7).

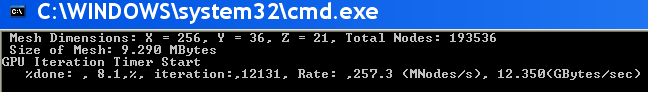


Figure : Command Window Status Updates every two seconds

## Excitation Execution:

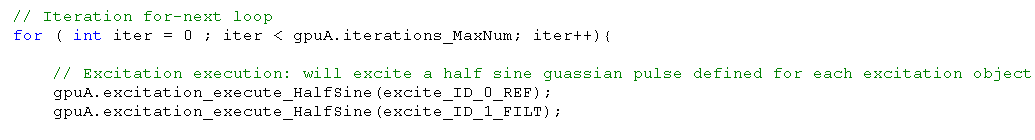


Figure : Excitation Execution and start of iterations

excitation\_execute\_HalfSine(ID\_of\_excitation\_obj): Executes the excitation kernel with the predefined excitation object for a half-sine excitation. The internally defined array of a gaussian sine wave will be used to impose a excitation magnitude as the iteration count marches in time. Note that iteration count does not need to be passed to this method.

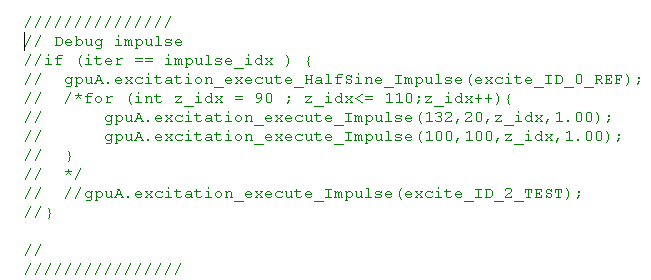


Figure : Single point Impulse and HalfSine Impulse

For completeness the example project includes commented code (figure 9) that show the use of two impulse excitation methods.

**excitation\_execute\_Impulse(x,y,z,magnitude):** This function imposes an impulse at node x,y,z of a specified magnitude. This is a standalone method which does not require an excitation object to be created. At this time, all 12 voltages of the specified node are imposed with the magnitude. Future code will include polarization as a parameter.

**excitation\_execute\_HalfSine\_Impulse(excite\_ID\_0\_REF):** Uses the excitation object that was created earlier, but ignores the gaussian sine array and imposes only a magnitude of '1' on the excitation plane. Future code to include a magnitude.

## Iteration Execution:



Figure : Iteration command

The iteration command (figure 10) requires no parameters, and will execute the appropriate scattering kernels and advance the timestep counter forward one timestep.

## Sample Execution:

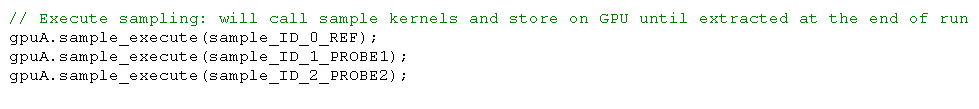


Figure : Sample execution. Captures samples and stores in sample arrays on GPU

The sample execution methods require the sample object ID passed as a parameter. Once executed it will execute a kernel on the GPU to take a sample and store it into an array located internal to the GPU. This GPU sample array can be extracted later for analysis.

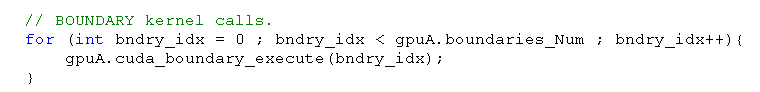


Figure : Boundary Kernel Calls

Boundary execution is shown in figure 12. Each boundary that was defined in a csv file are executed in the for-next loop where the boundary index is passed as a parameter. This sample also shows access to the number of boundaries defined (gpuA.boundaries\_Num).

## End of Iteration For-Next Loop and Timer Stop

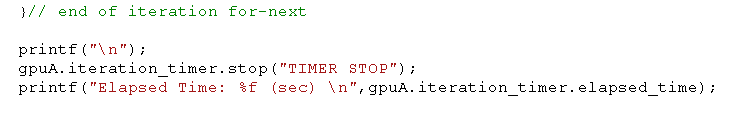


Figure : End of iteration loop and timer stop

Figure 13 shows the end of the iteration loop and the timer stop command. Elapsed can also be accessed and printed to the command window.

## Mesh Extraction

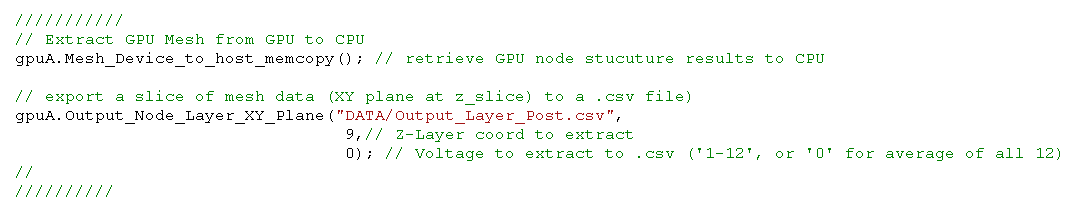


Figure : Extract GPU Mesh + export a layer of mesh nodes

Once the for-loop of iterations have completed the mesh can be copied from the GPU for analysis (figure 14). Also, a layer of the mesh can be saved to a csv:

Output\_Node\_Layer\_XY\_Plane("file.csv",z\_layer, VoltageID): This will save an XY layer of the mesh to a '.csv' file. The z-layer must be specified as well as which of the 12 SCN voltages. If zero is specified for a voltage then an average of the 12 voltages per node will be saved. (note: currently only the XY plane has been implemented)

## Sample Extraction:

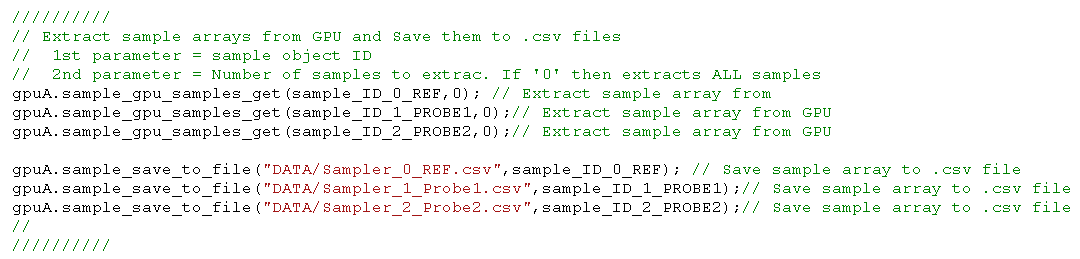


Figure : Sample extraction and save samples to .csv files

Samples are extracted from the GPU.

**float \*sample\_gpu\_samples\_get(SampleID, NumOfSamples):** retrieves the sample array from the GPU and stores it internally (in sample object). The number of samples must also be passed. If zero is passed then all samples are to be retrieved. The function returns a pointer to the array of samples.

**sample\_save\_to\_file("file.csv",sample\_ID):** saves the sample arrays just retrieved from the GPU to a .csv file.

# WR-28 Filter Analysis

When the project is compiled and run, it will first read the " STRUCTURE\_DEFINITIONS.csv" file and instantiate a 'GPU' object. Once the iterations are complete, the program will save the sample data from the Ref probe, probe 1, and probe 2 to three respective .csv files.

## CUDA\_MATLAB S11-S21 CALCULATIONS

When the CUDA-SCN code is complete, the Matlab program "FFT\_of\_Cuda\_savetoExcel.m" can be run. This will read in the three csv files generated by the CUDA-SCN application (Sampler\_0\_REF.csv, Sampler\_1\_Probe1.csv, Sampler\_2\_Probe2.csv). Be sure that this matlab .m file is located in the same directory as the csv files. The matlab file will calculate S11 and S21 parameters and then save the results by automatically pasting them into the file "MatlabFFT Results.xlsx". (this file must be closed before matlab is able to paste results into it)

## MatlabFFT Results.xlsx

Open the MatlabFFT Results.xlsx file. The results of the CUDA-SCN S11,S21 parameters are shown in the graphs. What is also shown are the results of Mefisto (static data) that is used to compare and validate the CUDA-SCN code structure.

