FEM Solution Manipulation

Introduction

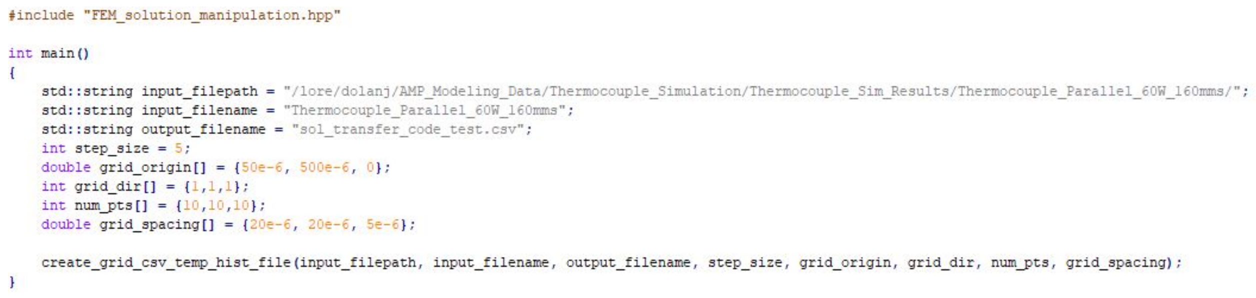
The FEM Solution Manipulation code is designed to be a multi-purpose program for various tasks that require reading in FE results data, performing computations and filtering, and writing the output in usable formats. Currently, the main directory for this code is `/lore/dolanj/3D_Manufacturing/FEM_results_manipulation`. The `build_with_base.sh` script is a template, where the user can go in an add their top-level C++ file into the build with the base code. The top level C++ file should contain a main function that will perform the desired task. More information on the available functions and associated input values are discussed in the following sections.

The basic steps required to use this code:

1. Create a new C++ file for your specific use case and include the main `FEM_solution_manipulation.hpp` header file
2. Write a main function that defines the relevant input arguments and calls the desired function
3. Modify the `build_with_base.sh` script to compile with the new C++ file and create an executable
4. Run the executable

Function: create_grid_csv_temp_hist_file

The `create_grid_csv_temp_hist_file` function imports solutions from the Albany FEM code, interpolates results from the unstructured FE grid onto a user-specified regular grid, and writes the regular grid results to a proprietary csv file for use as an input to the RPI phase field model. An example of a top level C++ file is shown below. In order to link and compile with the code base, the user must have `#include “FEM_solution_manipulation.hpp”` in their cpp file. The input arguments for the `create_grid_csv_temp_hist_file` function are discussed below.



```cpp
#include "FEM_solution_manipulation.hpp"

int main()
{
    std::string input_filepath = "/lore/dolanj/AMP_Modeling_Data/Thermocouple_Simulation/Thermocouple_Sim_Results/Thermocouple_Parallel_60W_160mms/";
    std::string input_filename = "Thermocouple_Parallel_60W_160mms";
    std::string output_filename = "sol_transfer_code_test.csv";
    int step_size = 5;
    double grid_origin[] = {50e-6, 500e-6, 0};
    int grid_dir[] = {1,1,1};
    int num_pts[] = {10,10,10};
    double grid_spacing[] = {20e-6, 20e-6, 5e-6};

    create_grid_csv_temp_hist_file(input_filepath, input_filename, output_filename, step_size, grid_origin, grid_dir, num_pts, grid_spacing);
}
```

Input Arguments

- input _filepath
    - The relative or absolute path to the folder containing the results subfolders for each timestep from an Albany FE simulation
- input_filename
    - The base name of the results subfolders and the simulation pvd file
- output_filename
    - The complete name of the file to which the interpolated regular grid results will be written to. Include the .csv extension
- step_size
    - The frequency with which timesteps from the FE solution will be interpolated. For example, a step size of 5 will read and interpolate the results from every 5th timestep from the Albany simulation.
- grid_origin
    - An array of three numbers defining the x, y, and z coordinates respectively of the origin corner of the regular grid
- grid_dir
    - An array of three numbers defining the direction that the regular grid extends, either in the positive or negative x, y, and z directions respectively. These numbers shall be defined as 1 for the positive direction or -1 for the negative direction
- num_pts
    - An array of three numbers defining the number of regular grid points to be created in the x, y, and z directions respectively. The total number of points generated will be the product of these three values
- grid_spacing
    - An array of three numbers defining the spacing between each grid point in the x, y, and z directions respectively. The total length of the regular grid in each direction is the product of the number of points and the grid spacing in the given direction.

Function Call

Call the function with the input arguments ordered as shown in the figure above.