Chair for Computational Analysis of Technical Systems

RWTH AACHEN UNIVERSITY

Parallel Computing for Computational Mechanics

Summer semester 2022

Homework 1

Mesh Visualization and partitioning

The MIXD format

In this homework, we want you to get familiar with storing and organizing unstructured meshes for FEM solvers. The format we consider in this homework is used for the in-house simulations at our institute and is called MIXD. Meshes in this format contain the following files (amongst others, which are not necessary in the scope of this class):

- **minf** file contains general mesh information about the number of elements (`ne`), number of nodes (`nn`), number of nodes per element (`nen`) and number of spatial dimensions (nsd) as well as the location of the other mesh files.
- **mxyz** file contains the spatial coordinates of all nodes in a binary format. Each coordinate entry (1 per spatial dimension) is (`sizeof(double)`).
- **mien** stores the connectivity. This means, there are `nen * ne` entries in this file, providing the information which element is made up of which nodes. Each entry is stored (`sizeof(int)`).

minf

Not really part of the format, but a traditional name for a text file that lists the number of nodes and elements, each preceded by a keyword that is understood by XNS, Visual3, etc. An example:

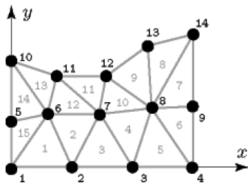
```
ne  6000
nn 14322
```

(a) Description and an example of the `minf` file

mxyz

This file contains the nodal coordinates. There are nsd entries for each node containing the coordinates of that node. In the example shown, this file contains the following entries:

node	x coordinate	y coordinate
1	0.00	0.00
2	0.80	0.00
3	1.60	0.00
4	2.40	0.00
5	0.00	0.60
6	0.50	0.70
7	1.15	0.70
8	1.85	0.75
9	2.40	0.80
10	0.00	1.40
11	0.60	1.20
12	1.25	1.20
13	1.80	1.60
14	2.40	1.75

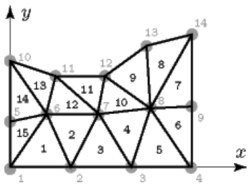


(b) Description and and example of the `mxyz` file

mien

This file contains the finite element connectivity array. There are nen entries for each element containing the global node number, from 1 to nn, of the element nodes. In the example shown, this file contains the following entries:

element	node	node	node
1	1	2	6
2	2	7	6
3	2	3	7
4	3	8	7
5	3	4	8
6	4	9	8
7	8	9	14
8	8	14	13
9	8	13	12
10	7	8	12
11	7	12	11
12	6	7	11
13	6	11	10
14	5	6	10
15	1	6	5



(c) Description and an example of the `mien` file

Skeleton Code Structure

The subject of this homework is to read a mesh in the MIXD format and post-process it for visualization in Paraview. A skeleton code is provided in `hw1.tar.gz` (found on RWTH Moodle). The code is written in C++. It contains the following files:

- `CMakeLists.txt`: necessary for the CMake build process.
- `constants.h`: header file with necessary constants.

- `postProcessor.cpp`: routines for the writing of vtk-files.
- `postProcessor.h`: header file for `postProcessor.cpp`
- `read-mesh.cpp`: main file containing the program structure.
- `settings.cpp`: contains routines for reading the settings file.
- `settings.h`: header file for settings.cpp
- `tri.cpp`: contains everything related to reading the MIXD format files.
- `tri.h`: header file for `tri.cpp` with the classes `triNode`, `triElement` and `triMesh`.

Compiling the converter

To build the executable, we use CMake. To compile the code, use the following set of commands from a terminal:

```
mkdir build
cd build
cmake ..
make
```

Running the converter

In the `test` folder, we provide a set of files containing the settings and the MIXD files for the converter for the 3 different meshes (coarse, fine, finest).To run the code (if you compiled it succesfully as written above), do the following (assuming you start in the folder `skeleton`):

```
cd test
../build/read-mesh settings.disk-{coarse,fine,finest}.in
```

Implementation tasks

Your task is to provide missing code lines that read in the relevant information from the MIXD files and stores them, so that the postprocessing unit of the code can convert the mesh to a vtk file that can be visualized in paraview. You can visualize the element grid from the vtk files by choosing *Surfaces with Edges* from the *Representation* dropdown menu within Parview. Most of your work will take place in the file called `tri.cpp` (tri for triangles).
Complete the following tasks by inserting code in the marked positions in `tri.cpp`:

- **Task 1** Make the program open the `minf` file, read and store `nn` and `ne`. Allocate the data structures for *node* and *elem* based on those sizes.
- **Task 2** Make the program open the `mxyz` file, go over all the nodes, read and store `x` and `y` coordinates for each node into the `xyz` array and the `x` and `y` attributes of each node of the `triNode class`.
- **Task 3** Open the mien file, go over all the elements and nodes per element and store the global connectivity for each element with the setConn attribute. Here you have to use the function `swapBytes` to change the endianness of the read in data.

Implementation skeleton and helper functions

Data from a file can be read in C++ as follows by using the routines *open*, *seekg*, *read* and *close*:

```
#include <constant.h> // provided

main(int argc, char *argv)
{
    dummy = settings->getMxyzFile();
    file.open(dummy.c_str(), ios::in|ios::binary|ios::ate);

    readStream = new char [nsd*sizeof(double)];
    file.seekg (0, ios::beg);

    ...
    file.read(readStream, nsd*sizeof(double));
    swapBytes(readStream, nsd, sizeof(double));
    ...
    file.close();
}
```

Use the helper functions stated below to put the mesh information in the right place in the code structure. The helper functions can be found in the header files (`*.h`). A description of the necessary functions is given hereafter:

- `getMinfFile()`: Return the path to the `minf` file.
- `getMxyzFile()`: Return the path to the `mxyz` file.
- `getMienFile()`: Return the path to the `mien` file.
- `node[i].setX(x = value)`: Set the x-coordinate value to "value" for node `i`.
- `node[i].setY(y = value)`: Set the y-coordinate value to "value" for node `i`.

- `elem[i].setConn(j, connValue)`: Set the connectivity for element `i`. Local node number `j` with global node index "`connValue`".

After the mesh is read, the code will export it into a "`*.vtk`" file. This file can be opened with Paraview. Additionally, we ask one question (**Q1**) and have a bonus task (**BT**) which anticipates future course content: Question:

- **Q1**: In a typical implementation, each mesh edge is drawn multiple times. Can you think of an algorithm that would avoid that? (You do not need to implement the algorithm, just sketch a method).
- **BT**: For this task, we are already looking into the topic of partitioning the mesh into smaller chunks to distribute amongst multiple processors (PEs). In general, there are different strategies to parallelize FEM code. One of them is to distribute the data among the available processors and let each of them work on separate chunks of data and communicate the overlap. This approach is called distributed memory parallelism. For the bonus task, we want you to distribute the nodes of the mesh amongst a number of processors. The number of processors is read in from the `settings.coarse,fine,finest.in` as `npes`. For this bonus task, there is an additional data structure of the `triNode` class of size `nn`, that assigns a processor ID (`peID`) to each node to emulate the partitioning of the mesh. Your task is to distribute the nodes as evenly as possible amongst all the available processors (=go over all nodes and assign a values from 0 to `npes` to the `peID` array). Can you think of different ways to distribute the nodes? Try to find at least two methods based on the nodes attributes like node number, node position etc. Can you think of a way to distribute the nodes to minimize the communication among the processors? (Think about the special geometry of the mesh here).

You do not need to turn in anything for this homework. We will discuss the results and solution in the upcoming exercises.

Contact:
Maximilian Schuster · schuster@cats.rwth-aachen.de
Michel Make · make@cats.rwth-aachen.de