Curvilinear GMSH Reader

Aleksejs Fomins

1 GMSH conventions

1.1 Structure of .msh files

```
$MeshFormat
ver f_type data_size                # This line is mostly irreleva
$EndMeshFormat
$Nodes
n_vertices
1 x y z
2 x y z
.......
n_vertices x y z
$EndNodes
$Elements
n_elem
1 elem_type n_tags (process_tags) v_1 v_2 ... v_N
2 elem_type n_tags (process_tags) v_1 v_2 ... v_N
.......
n_elem elem_type n_tags (process_tags) v_1 v_2 ... v_N
$EndElements
```

where

- *ver* - version of the GMSH file
- *f_type* - type of file (irrelevant)
- *data_size* - size of file (irrelevant)
- *n_vertices* - number of vertices of the mesh
- *i x y z* - index of the vertex and its coordinates

- *n_elem* - number of elements of the mesh
- *elem_type* - Integer which determines element type and interpolation order
- *n_tags* - Total number of tags. If > 2, then have *process_tags*
- *process_tags* - Tags which determine the process the vertex belongs to. Only if GMSH is told to partition the mesh
- *v_1 v_2 ... v_N* - Indices of interpolatory vertices associated with this element (includes corners)

1.2 Convention for numbering interpolatory vertices

Each curvilinear element posesses a set of interpolatory vertices. For order 1 (linear elements) this is just the set of corners of the element, but for higher orders there are additional points located in on the inside of the elements, their faces and edges. To number these vertices, GMSH uses recursive convientionn.

1. First number all corners, then all edges, then all faces, then vertices inside element, then internal vertices of the element
2. Inside edge, vertices are numbered sequentially
3. For a triangle, the order of edges is $(0, 1)$, $(1, 2)$, $(2, 0)$. (in 2D)
4. For a tetrahedron, the order of edges is $(0, 1)$, $(1, 2)$, $(2, 0)$, $(3, 0)$, $(3, 2)$, $(3, 1)$.
5. For a tetrahedron, the order of faces is $(0, 2, 1)$, $(0, 1, 3)$, $(0, 3, 2)$, $(3, 1, 2)$, including orientation
6. If there are vertices associated with element itself(for example, on the triangle or inside the tetrahedron), a smaller element is created inside this triangle preserving its orientation, and is then numbered recursively.

Unfortunately, this convention does not match the grid convention used in DUNE, namely

```
for (z = 0 to 1, y = 0 to 1-z, x = 0 to 1-z-y) { vertex(x,y,z);
```

There is no good expression which maps from GMSH to DUNE convention, so it was implemented it by hand for simplex geometries up to order 5.

- Triangle Order 1: {0, 1, 2}

- Triangle Order 2: {0, 3, 1, 5, 4, 2}
- Triangle Order 3: {0, 3, 4, 1, 8, 9, 5, 7, 6, 2}
- Triangle Order 4: {0, 3, 4, 5, 1, 11, 12, 13, 6, 10, 14, 7, 9, 8, 2}
- Triangle Order 5: {0, 3, 4, 5, 6, 1, 14, 15, 18, 16, 7, 13, 20, 19, 8, 12, 17, 9, 11, 10, 2}
- Tetrahedron Order 1: {0, 3, 1, 2}
- Tetrahedron Order 2: {0, 7, 3, 4, 9, 1, 6, 8, 5, 2}
- Tetrahedron Order 3: {0, 11, 10, 3, 4, 17, 14, 5, 15, 1, 9, 18, 12, 16, 19, 6, 8, 13, 7, 2}
- Tetrahedron Order 4: {0, 15, 14, 13, 3, 4, 25, 27, 19, 5, 26, 20, 6, 21, 1, 12, 28, 29, 16, 22, 34, 31, 24, 32, 7, 11, 30, 17, 23, 33, 8, 10, 18, 9, 2}
- Tetrahedron Order 5: {0, 19, 18, 17, 16, 3, 4, 34, 39, 36, 24, 5, 37, 38, 25, 6, 35, 26, 7, 27, 1, 15, 40, 43, 41, 20, 28, 52, 55, 46, 33, 53, 49, 30, 47, 8, 14, 45, 44, 21, 31, 54, 51, 32, 50, 9, 13, 42, 22, 29, 48, 10, 12, 23, 11, 2}

2 Parallel Implementation

The idea of parallel implementation is that all data - vertex coordinates, internal elements and boundary elements - are split between processes, not to exceed the single core memory. Thus the strategy for reading data on a process i is as follows:

1. Compute the total number N_{elem} of non-boundary (internal) elements.

Loop over all elements in the file, and count the number of elements with dimension equal to the world dimension

2. Read and store internal elements belonging to this process. If elements are split in consequent equal chunks among processes, then process $rank$ should read the elements with indices $interv(rank) = \left\lfloor [rank, rank+1] \cdot N_{elem}/p_{tot} \right\rfloor + 1$.

- Loop over all elements in the file
- Store all internal elements for which $index \in interv(rank)$
- Add global indices of vertices belonging to selected elements to a set

3. Read and store boundary elements belonging to this process - those which match the subentities of some elements.

   - Only read the boundaries for which all **corners** have already been included to the element vertex set.

4. Read and store vertices belonging to this process. Namely, all the vertices that are necessary to construct the elements and boundaries belonging to this process.

   - Local index of a vertex is a number $[0, n_p)$ where n_p is the total number of vertices on the process.
   - Local index of a vertex is given by the order they are inserted to the grid factory. It is in the ascending order of the global index, just that certain vertices of global index are not on this process. To keep track of this we fill the global-to-local map for vertices.

5. Add boundary elements to factory. It is necessary to connect the boundary elements to the internal elements they share a face with, because during load balance, the boundaries need to be communicated together with the corresponding elements.GMSH does not provide information on this interconnection.

   - Important! This must be done before adding internal elements to factory, as we also need to add the interconnection array.
   - Add global element index as well

Currently using brute-force, because it is not much slower than improved for

Trivial Algorithm: (Complexity $O(12N_{elem}N_\beta/p_{tot}^2)$)
Loop over all stored boundary elements β_i, and over all stored internal elements E_j. If $\beta_i \in E_j$ for some j then store β_i

Improved Algorithm: (Complexity $O(12N_{elem}\log_2(N_\beta/p_{tot})/p_{tot})$

1. Construct map from boundary vertex index set to boundary id
2. Add all boundaries to the map
3. Loop over each face of all internal elements
   (a) If $map[face]$ is non-null, link the element and boundary

6. Add internal elements to factory

- For debugging purposes write each element to a .vtk file using CurvilinearVTKWriter.
- Add element vertices and global element index to factory
- If creating grid with boundaries, also pass *internal_to_boundary_element_linker*. This array stores the indices of boundaries which are connected to this element (if any).

3 Curvilinear VTK Writer

3.1 Implementation

1. For each tetrahedron, define interpolation class, which gives its global curved coordinates based on local parametric coordinates and interpolation points.

   - the interpolation points are passed only once when initialising the class

2. parametrically Loop over all surfaces of the tetrahedron

   - Shrink the surfaces by a little bit for better visibility

   - Calculate a fixed number of equally spaced points on each face using square mesh, add all points to a map, which notes which points have already been added. This can be done either by using the interpolation class to get the interpolated points, or simply re-using the interpolation points not to invoke the interpolation process at all thus testing it.

   - Split each square into 2 triangles and add them to the triangle list. If the square is at the edge, it produces only 1 triangle.

3. Write a .VTK file which has all triangles

3.2 Optimal Sampling Rate

Here we would like to discuss how many sampling points are required to represent the bounded curve using linear interpolation. This is necessary, for example, for visualisation. For a line segment, interpolating the curve bounded by 2 points, the analytic error is bounded by

$$\epsilon = |f(x) - g(x)| \leq \frac{1}{8} \max_{z \in [x_a, x_b]} \{g''(z)\}(x_a - x_b)^2 \tag{1}$$

where $f(x)$ is the line segment, $g(x)$ is the original function. x_a and x_b are arguments of the endpoints of the interval.

Given a large interval L which we split into N equal segments such that $x_b - x_a = L/N$, the maximal error over all interpolated intervals is bounded by

$$\epsilon_{\max} \leq \frac{L^2}{8N^2} \max_{z \in L} \{g''(z)\} \tag{2}$$

This can be rewritten to obtain the necessary interval number per edge

$$N = \left\lceil \alpha \sqrt{\max_{z \in L} \{g''(z)\}} \right\rceil \tag{3}$$

where α depends on precision and length of the interval. We observe that there is a problem with this formula, namely that if $g''(z) = 0$ we get 0 intervals, and if $g''(z)$ is small, we get 1 interval, and we would like 2, because there is at least some curvature, and it should be emphasized that the case is different from just straight line. Therefore we empirically modify the formula

$$N = \left\lceil \alpha \sqrt{\max_{z \in L} \{g''(z)\}} + 1 \right\rceil \tag{4}$$

Finally, since this equation is not exactly accurate for surfaces, and it is rather hard to make sense of optimal relative error, it is much easier to choose α empirically. We will generally operate with unit length edges ($L = 1$), and would like to, for example, have 5 points per edge with standard quadratic curvature $g(x) = x^2$. We obtain that $\alpha \approx 2.8$.

As a rule of thumb, we can calculate the expected number of intervals for a standard function $g(x) = \sum_i x^i$ using this formula. The first 5 orders will produce edge intervals $\{1, 5, 9, 17, 35\}$, which will correspond to the number of vertices per edge $\{2, 6, 10, 18, 36\}$, and thus the number of surface vertices ($i(i+1)/2$) will be given by $\{3, 21, 55, 171, 666\}$.

.1 Calculate the average number of elements that are common to a point

Triangles:

- Average angle of a triangle $\gamma = \pi/3$.
- Average triangles per point $2\pi/\gamma = 6$

Tetrahedra:

- Average solid angle per tetrahedron $\gamma = 0.55$
- Average tetraherda per point $4\pi/\gamma \approx 23$

.2 Algorithm: Boundary-Internal element match search. Complexity Calculation

Let the total number of elements be N_e, with largest expected size $N_e \approx 10^7$.
Want to approximate the number of boundary elements

- Sphere: $V = 4\pi r^3/3$, $A = 4\pi r^2 \approx 7.79\ V^{2/3}$
- Cube: $V = r^3$, $A = 6r^2 = 6\ V^{2/3}$
- Sheet: $V = \epsilon r^3$, $A = 2r^2 + 4\epsilon r^2 \approx 2(1/\epsilon)^{2/3} V^{2/3}$

So, $N_\beta = \alpha N_e^{2/3}$, where $\alpha \approx [10^1 \text{ to } 10^2]$ depending on how flat the computational domain is.

For a simple algorithm, we would loop over all all boundaries and all elements, then over all points of each and compare. The complexity of that is

$$3 * 4 * (N_e/p_{tot}) * N_\beta = 12\alpha N_e^{5/3}/p_{tot}$$

For the complicated algorithm

- For each element adding its name to the vertex map: $O(4(N_e/p_{tot}) \log(N_e/p_{tot}))$
- For each corner of each boundary access this map to get element set: $O(3N_\beta \log(N_e/p_{tot}))$
- Intersecting 3 sets of indices, of length M_T: $O(2 * N_\beta 4M_T)$

where M_T for tetrahedra is approx 23 as shown in the above section. This results for a total complexity

$$N_e(\log(N_e^{2/3}/p_{tot})(\frac{4N_e^{1/3}}{p_{tot}} + 3\alpha) + 184 * \alpha) \approx N_e^{2/3}\alpha(3\log(N_e/p_{tot}) + 184)$$

Conclusion

- For small mesh with 10^4 elements and 10^1 processes, the complexities are $\sim 4.3 \cdot 10^7$ and $\sim 7.6 \cdot 10^5$
- For large mesh with 10^7 elements and 10^3 processes, the complexities are $\sim 4.3 \cdot 10^{10}$ and $\sim 8.0 \cdot 10^7$

Therefore, the advanced method is essential