Chapter 1

Mesh generation based on Tetgen

In this document we demonstrate how to generate unstructured tetrahedral meshes for oomph-lib, based on the output from Hang Si's open-source mesh generator Tetgen . The mesh generation is performed in a two-stage process. First we use Tetgen to generate the mesh "offline". Then we process the output files generated by
Tetgen to generate an oomph-lib mesh.

1.1 Quick guide for the use of Tetgen

The Tetgen home page contains a comprehensive User's Guide for the code and its many options, therefore we only present a brief overview of the code's most basic usage.
Tetgen creates the mesh based on the information about the mesh boundaries provided in an input file, filename.poly, say. By default, three output files, filename.1.node, filename.1.ele, and filename.1.face are created. They contain the information about the nodal positions, the element connectivity lists and the boundary faces respectively.

1.1.1 The input file format

The input file for Tetgen usually has the extension *.poly and has the following format:

Node list
First line: [number of nodes] [dimension (must be 3)] [number of attributes] [number of boundary markers (0 or 1)]
Remaining lines: [node index] [x] [y] [z] [[attributes]] [[boundary marker]]
Facet list
One line: [number of facets] [boundary markers (0 or 1)]
Following lines are a list of the facets: [facet index]
where each facet has the following format:
One line: [number of polygons] [[number of holes]] [[boundary marker]]
Following lines: [number of corners] [corner 1] [corner 2] ... [last corner]
Following lines: [hole index] [x] [y] [z]
Hole list
One line: [number of holes]
Following lines: [hole index] [x] [y] [z]
Region attributes list
One line: [number of region]
Following lines: [region index] [x] [y] [z] [region number] [region attribute]

Comments:

- The data between [ ] must be provided. The attributes and boundary markers must only be specified if the corresponding number of boundary markers or the number of attributes is nonzero.

- Boundary markers can be used to identify which nodes are located on which domain boundaries. If domain boundaries are to be identified the number of boundary markers should be set to 1, otherwise it must be set to zero. If the number of boundary markers is set to 1, boundary markers must be specified for every node and facet. A boundary marker 0 should be used for nodes that are not located on domain boundaries; a boundary marker `b+1` should be used to indicate that a node is located on the mesh boundary `b` in the final `oomph-lib` mesh.
- `oomph-lib` does not use the "attributes" so their number should be set to zero.
- The boundary markers have to be specified in the node list AND in the facet list.
- If a node is located on multiple boundaries, the boundary marker of the node is chosen arbitrarily.
- Holes are identified by the coordinates of a single point in their interior. The holes in the facet list are holes in the facet, they are different from the holes in the hole list, which are holes in the volume.

See the Tetgen home page for further information.

1.1.2 How to run Tetgen

To create the mesh from a given input file the command is

```
./tetgen filename.poly
```

With these commands, Tetgen will generate as few tetrahedra as possible. Finer meshes may be generated by imposing additional constraints via command line arguments. For instance

- A maximum tetrahedron volume can be specified with `-a`*n* where *n* is the maximal volume wanted. (There is no space between `-a` and the number specifying the volume!)
- ...

Again, we refer to the
Tetgen home page for a comprehensive listing of all available options.

1.1.3 How to visualise a mesh generated by Tetgen

To visualise the mesh, the program `tetview` (distributed with Tetgen) can be used.

1.2 An example: A cube with a cube hole

To illustrate the procedure, we demonstrate how to generate a mesh for the cube domain with a hole shown in the figure below. Note that the node numbers correspond to those in the Tetgen input file. Boundary 1 is shown in blue; boundary 2 in magenta. In the corresponding `oomph-lib` mesh, the boundaries are numbered from zero.

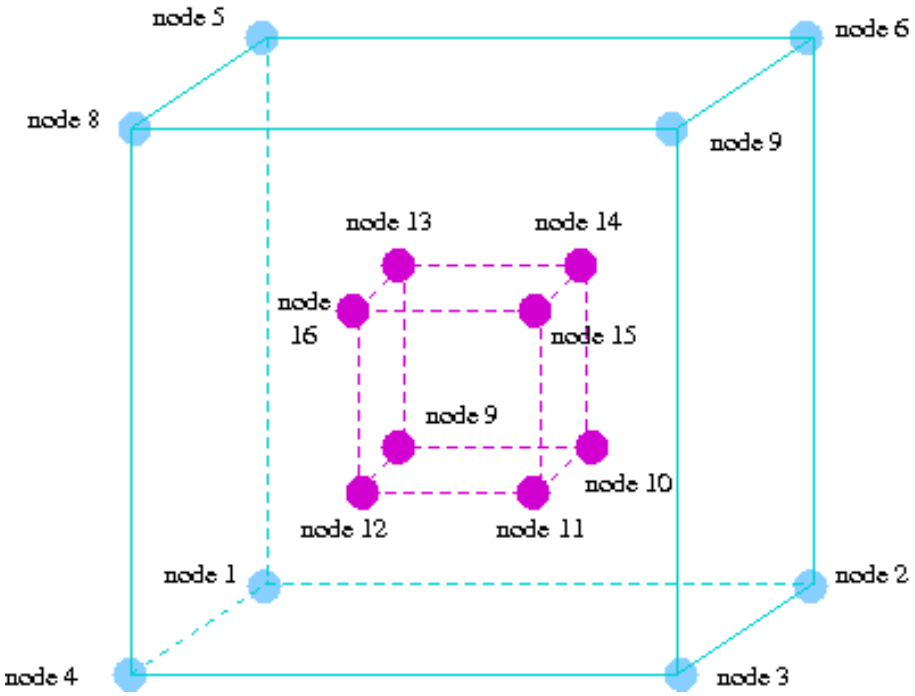

Figure 1.1 the cube with a hole domain

Here is a listing of the corresponding input file, `cube_hole.poly`:

```
# This file represents a cube with a cube hole in the middle.

# Part 1 - node list
16 3 0 1
1     0 0 0 1
2     3 0 0 1
3     3 3 0 1
4     0 3 0 1
5     0 0 3 1
6     3 0 3 1
7     3 3 3 1
8     0 3 3 1
9     1 1 1 2
10    2 1 1 2
11    2 2 1 2
12    1 2 1 2
13    1 1 2 2
14    2 1 2 2
15    2 2 2 2
16    1 2 2 2

# Part 2 - facet list
12 1
# bottom 1
1 0 1
4 1 2 3 4
# top 1
1 0 1
4 5 6 7 8
# left 1
1 0 1
4 1 4 8 5
# right 1
1 0 1
4 2 3 7 6
# front 1
1 0 1
4 3 4 8 7
# back 1
1 0 1
4 1 2 6 5
# bottom 2
1 0 2
4 9 10 11 12
# top 2
1 0 2
4 13 14 15 16
# left 2
1 0 2
4 9 12 16 13
```

```
# right 2
1 0 2
4 10 11 15 14
# front 2
1 0 2
4 11 12 16 15
# back 2
1 0 2
4 9 10 14 13

# Part 3 - hole list
1
1 1.5 1.5 1.5

# Part 4 - region list
0
```

The output files that `Tetgen` creates with the command

```
./tetgen -a0.2 cube_hole.poly
```

are `cube_hole.1.node`, `cube_hole.1.ele` and `cube_hole.1.face`.
Here is a sketch of the resulting discretisation, as displayed by `tetview`:

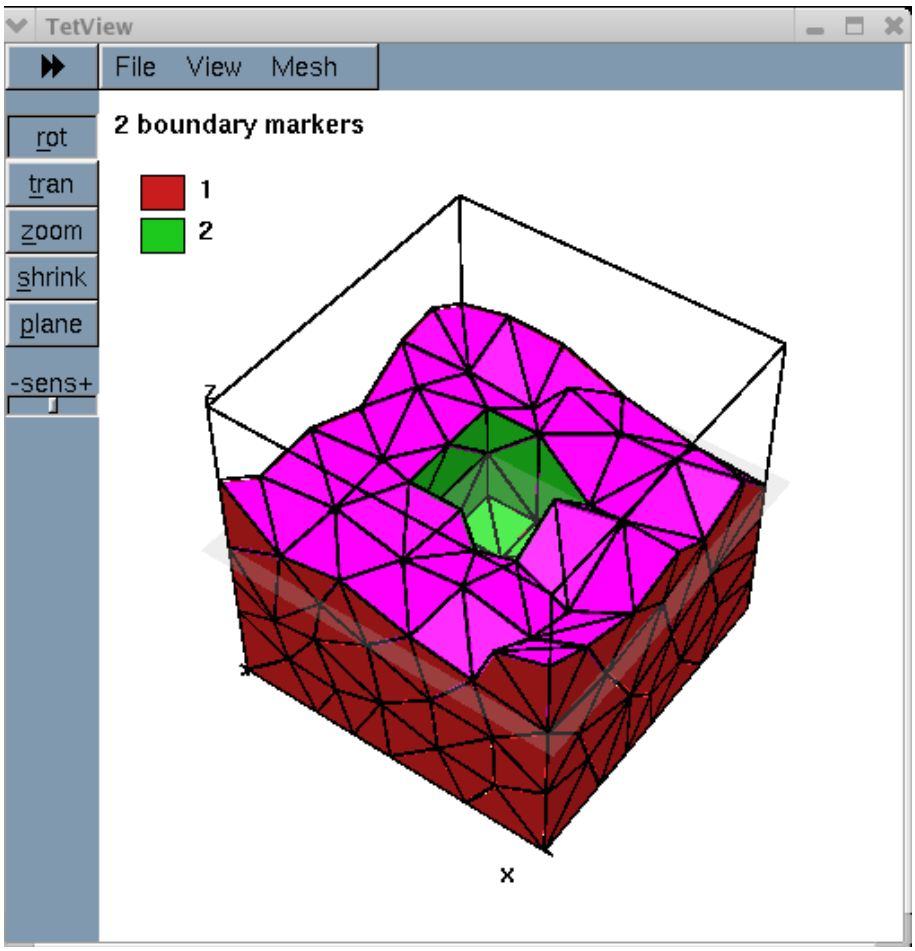

Figure 1.2 Cut plane of the tetrahedral mesh.

1.3 Creating an oomph-lib mesh based on output files generated by Tetgen

`oomph-lib` provides a mesh, `TetgenMesh`, that uses the output from `Tetgen` to generate an unstructured `oomph-lib` Mesh containing elements from the `TElement<3,NNODE_1D>` family of the tetrahedral elements. The relevant interface is:

```
//=========start of TetgenMesh class======================================
///  Unstructured tet mesh based on output from Tetgen:
/// http://wias-berlin.de/software/tetgen/
//========================================================================
template<class ELEMENT>
```

```
class TetgenMesh : public virtual TetMeshBase
{
public:
  /// Empty constructor
  TetgenMesh()
  {
    // Mesh can only be built with 3D Telements.
    MeshChecker::assert_geometric_element<TElementGeometricBase, ELEMENT>(3);
  }

  /// Constructor with the input files
  TetgenMesh(const std::string& node_file_name,
             const std::string& element_file_name,
             const std::string& face_file_name,
             TimeStepper* time_stepper_pt = &Mesh::Default_TimeStepper,
```

1.3.1 Example 1: A Poisson problem

The driver code mesh_from_tetgen_poisson.cc demonstrates the use of this mesh for the solution of a 3D Poisson problem in the "cube domain with a hole", described in the previous section.
The code expects the names of *.node, *.ele and *.face files generated by Tetgen as command line arguments and stores them in the namespace CommandLineArgs

```
int main(int argc, char* argv[])
{
// Store command line arguments
 CommandLineArgs::setup(argc,argv);

 // Check number of command line arguments: Need exactly three.
 if (argc!=4)
  {
   std::string error_message =
    "Wrong number of command line arguments.\n";
   error_message +=
    "Must specify the following file names  \n";
   error_message +=
    "filename.node then filename.ele then filename.face\n";

   throw OomphLibError(error_message,
                       OOMPH_CURRENT_FUNCTION,
                       OOMPH_EXCEPTION_LOCATION);
  }
```

The names of these files are then passed to the mesh constructor. Since the rest of the driver code is identical to that in the corresponding example with a structured mesh, we do not provide a detailed listing but simply show the plot of the computed results, together with a plot of the exact solution, for linear (four-noded) elements

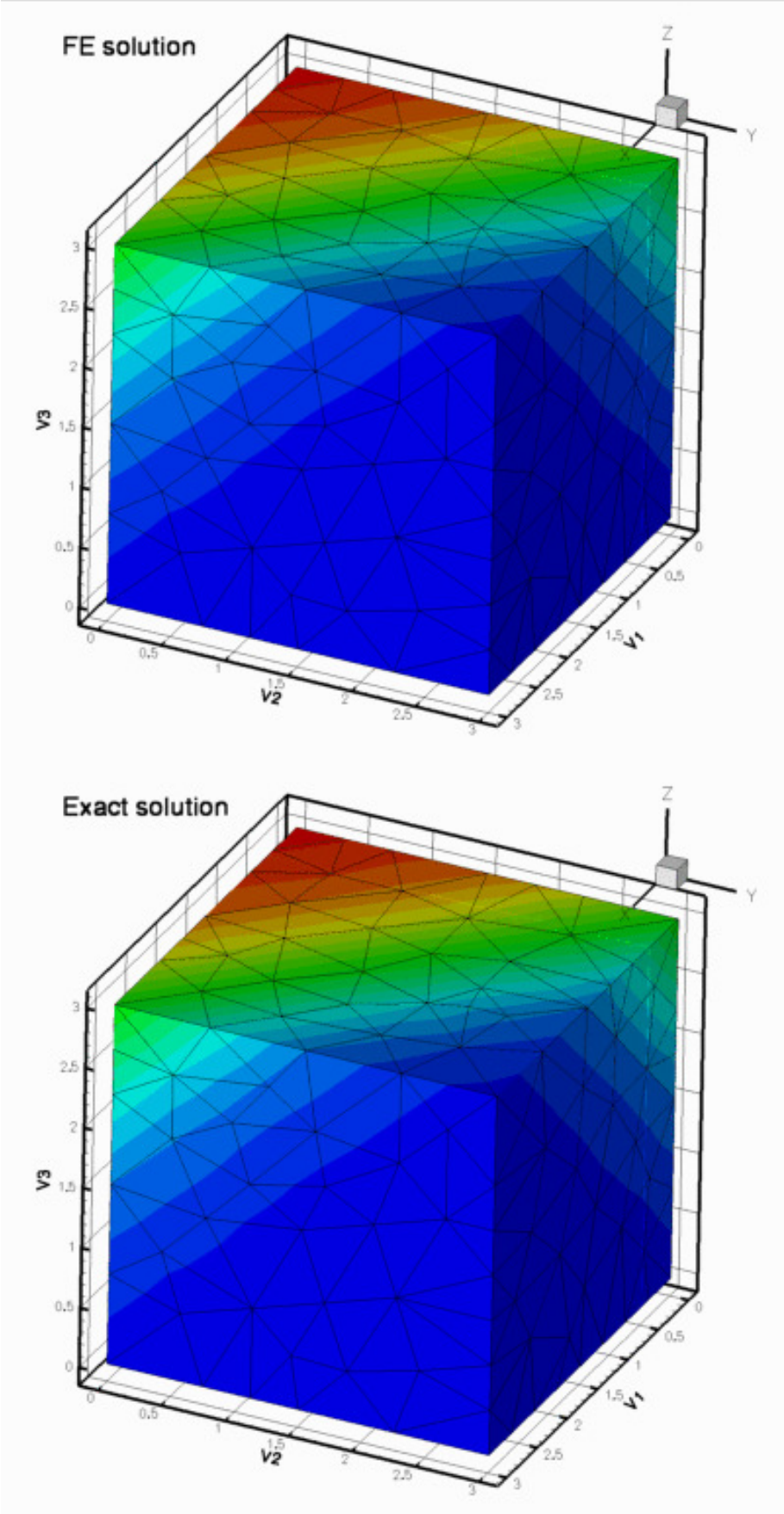

Figure 1.3 FE and exact solution with linear elements.

...and quadratic (ten-noded) elements:

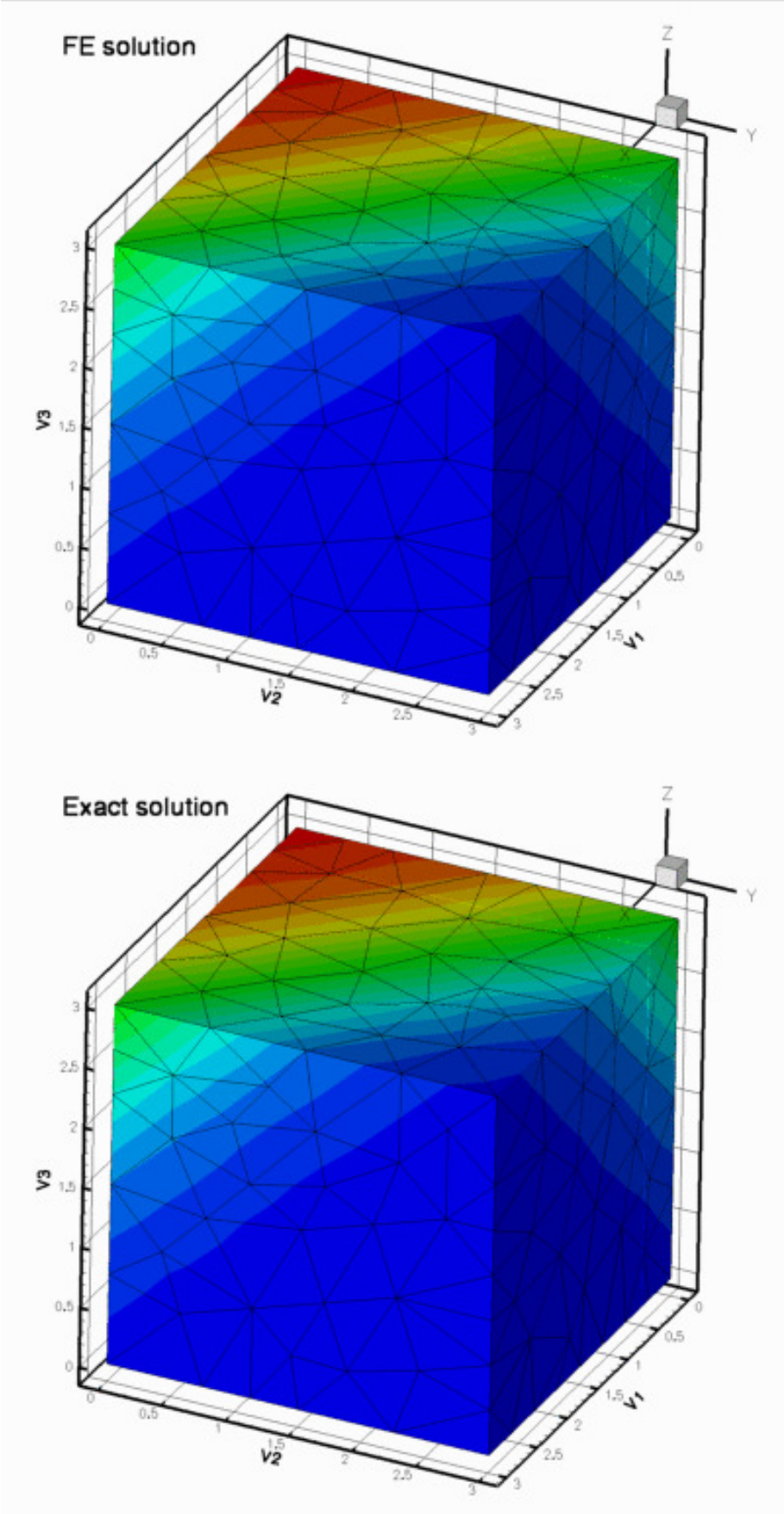

Figure 1.4 FE and exact solution with quadratic elements.

1.3.2 Example 2: A Navier-Stokes problem

The driver code `mesh_from_tetgen_navier_stokes.cc` demonstrates the use of a `TetgenMesh` for the solution of a 3D Navier-Stokes problem in the same geometry as in the previous example. We apply no-slip boundary conditions on the central hole and impose a unit vertical velocity. On the outer boundaries we pin the y and z velocity components and set them to zero, while leaving the axial velocity unspecified. The resulting flow field, shown in the figure below, corresponds to the flow that is generated when the rigid block in the centre of the domain rises vertically inside a rigid duct with "slippery walls". The duct is open at the ends $x = const$, where parallel, axially-traction free outflow is imposed. The various slices show pressure contours and the in-plane velocity vectors.

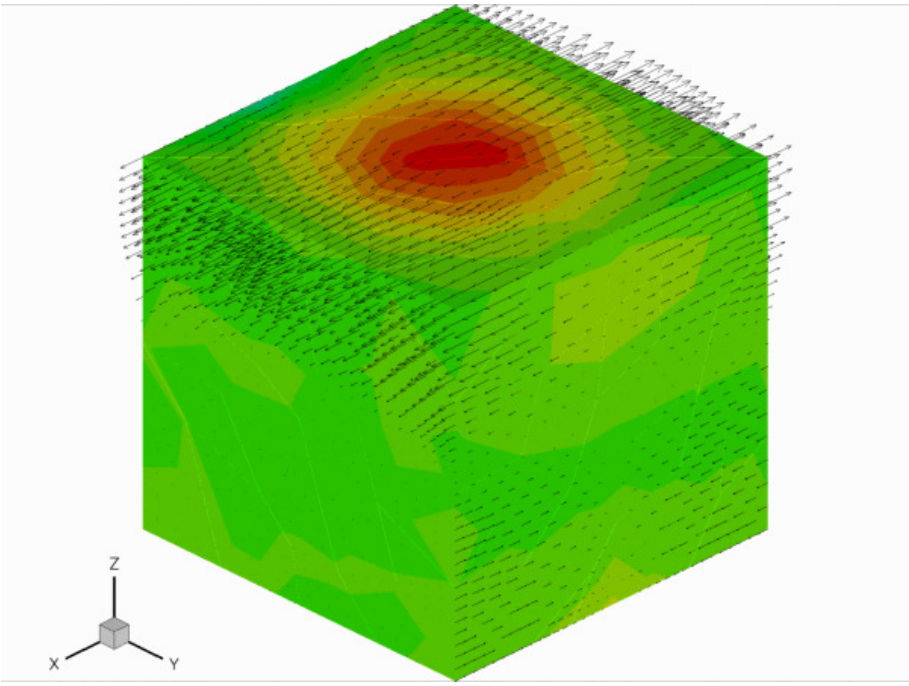

Figure 1.5 Flow in a duct with slippery walls, driven by the vertical motion of the central block.

1.4 Comments and Exercises

1.4.1 Checking the boundary numbers

We re-iterate that `Tetgen` does not allow nodes to be located on multiple boundaries. It is therefore important to check the boundary numbers allocated by `Tetgen` , e.g. by using the function `Mesh::output_boundaries(...)`. Boundary nodes should always be placed on the boundary with the most restrictive boundary conditions. If this is not possible, some post-processing of the mesh may be required.

1.4.2 Higher-order tets

Currently, `TetgenMesh` can only be used to generate four and ten-node tets (i.e. tets with tri-linear and tri-quadratic shape functions). It should be easy to generalise the "scaffold-mesh"-based mesh generation procedure to higher-order elements but this has not been done yet. Any volunteers?

1.4.3 Exercises

1. Download and install `Tetgen,` , and create your own meshes.
2. Experiment with the options that allow the specification of maximum element volumes etc.

1.5 Source files for this tutorial

- The source files for this tutorial are located in the directory:

demo_drivers/meshing/mesh_from_tetgen/

- The driver code is:

demo_drivers/meshing/mesh_from_tetgen/mesh_from_tetgen_poisson.cc

1.6 PDF file

A pdf version of this document is available.