NekMesh: High-order mesh generation (from STEP)

Tutorials

May 15, 2024

Department of Aeronautics, Imperial College London, UK
Scientific Computing and Imaging Institute, University of Utah, USA

2

Chapter 1

Introduction

This tutorial covers the basic functionalities of the curvilinear mesh generator that is provided with NekMesh, the mesh utility program provided with Nektar++. This tutorial is designed to show the main features of the two-dimensional mesh generator.

To achieve the goals of this tutorial you will need an installation of the latest `master` version of Nektar++, available at nektar.info. This is because the features you will be using are relatively new and have yet to be included in the latest release. Standard installation instructions are applicable as described in the user guide, with one addition that CMake should be configured to include `NEKTAR_USE_MESHGEN=ON`. This will activate the installation of the relevant meshing routines.

Tip

When using the mesh generation routines Nektar++ will require an OpenCascade installation to be present on your machine. Nektar++ is capable of downloading and installing OpenCascade for you, but this is highly not recommended as the installation time is very large. OpenCascade Community Edition (OCE) can be installed through package managers:

- **Homebrew:** `brew install homebrew/science/oce`
- **MacPorts:** `port install oce`
- **Ubuntu/Debian:** `sudo apt-get install liboce-foundation-dev liboce-modeling-dev`
- Other Linux installations have a version of OCE in their package manager, usually called `liboce`.

Nektar++ will be able to pick up on any of these OCE installations. Installation of NekMesh on Windows with mesh generation is not possible due to the OpenCascade requirements at this time.

This tutorial additionally requires:

- Nektar++ IncNavierStokesSolver with pre- and post-processing tools,
- A visualisation tool such as Paraview or VisIt

1.1 Goals

After the completion of this tutorial, you will be familiar with:

- the generation of a 2D mesh from a CAD file;
- the visualisation of the mesh in Paraview or VisIt
- running a simple simulation on this mesh with IncNavierStokesSolver; and
- the post-processing of the data and the visualisation of the results in Paraview or VisIt.

Task 1.1

Prepare for the tutorial. Make sure that you have:

- Installed and tested *Nektar++* v5.6.0 compiled from source. We will refer to the directory where you installed *Nektar++* as `$NEK` for the remainder of the tutorial.
- Make a directory of your choosing, for example `tutorial`, and download the tutorial files from `http://doc.nektar.info/tutorials/5.6.0/mesh-generation/2d-step/mesh-generation-2d-step.tar.gz` into this directory.
- Unpack the tutorial files by using

  ```
  tar -xzvf mesh-generation-2d-step.tar.gz
  ```

  to produce a directory `2d-step` with subdirectories called `tutorial` and `complete`.

The tutorial folder contains:

- a .stp file of a 2D cylinder geometry;
- an .mcf file;
- a session file which will run the mesh.

1.2 Background

Curvilinear mesh generation capability was recently added to NekMesh. These routines aim to take a geometric definition, which is usually in the form of a CAD STEP file, and produce a valid curvilinear mesh. This is not always an easy task, but the system is fast and robust in 2D, so in this tutorial we shall focus our efforts there.

The design of the mesh generation system focuses on automation. As such, once provided with a CAD description of the domain only 4 numerical parameters are required to produce a mesh. The system uses a curvature based refinement and smoothness algorithms to obtain mesh sizings automatically. In this tutorial you will be introduced to using manual refinement fields to achieve more control over the mesh sizing.

NekMesh works on input and output files and both need to be specified as command line arguments. In the case of mesh generation the input is a .mcf, mesh configuration file, which is formatted similarly to the Nektar++ XML. This file contains the instructions and parameters the mesh generation system needs.

A barebones example of an .mcf file is:

```
<NEKTAR>
    <MESHING>
        <INFORMATION>
            <I PROPERTY="CADFile" VALUE="cyl.stp" />
            <I PROPERTY="MeshType" VALUE="2D" />
        </INFORMATION>

        <PARAMETERS>
            <P PARAM="MinDelta" VALUE="0.5" />
            <P PARAM="MaxDelta" VALUE="3.0" />
            <P PARAM="EPS" VALUE="0.1" />
            <P PARAM="Order" VALUE="4" />
        </PARAMETERS>
    </MESHING>
</NEKTAR>
```

Six key pieces of information are required to run the mesh generator:

- `CADFile` specifies the geometry file which will be meshed;
- `MeshType` tells the mesher what type of elements to produce: 3D, 3DBndLayer, 2D, 2DBndLayer;
- `MinDelta`, `MaxDelta` and `EPS` determine the minimum and maximum element sizes; and
- `Order` defines the order of the high-order mesh.

The system automatically determines element sizing, δ as

$$\delta = 2R\sqrt{\varepsilon(2-\varepsilon)} \tag{1.1}$$

where R is the radius of curvature of the CAD entity and `MinDelta` and `MaxDelta` place limits on the value of δ. It should be noted that, in the case of constant radius geometries, such as cylinders, changing the value of `MaxDelta` and `MinDelta` may not alter the mesh. That is because in this setting, `EPS` is controlling the sizing.

Currently the preferred CAD file format for use with NekMesh is the `step203` format, which is a standard CAD format. This is because it contains topological information that makes high-order meshing much easier compared to other open CAD formats such as `IGES`.

Chapter 2

2D cylinder mesh

The mesh we make is for a simple 2D cylinder in a rectangular domain. We do this from a .stp file which has been provided.

Task 2.1

Create a coarse triangular mesh according to the .mcf specified above. This is provided in the tutorial folder. To make this mesh, run the command

```
$NEK/NekMesh cyl.mcf cyl.xml
```

More details about the generation process can be seen by enabling the *verbose* command line option `-v`:

```
$NEK/NekMesh -v cyl.mcf cyl.xml
```

This will produce a Nektar++ mesh, which we now need to visualise.

Task 2.2

Visualise the mesh using FieldConvert. For Tecplot output for visualisation in VisIt, run the command

```
$NEK/FieldConvert cyl.xml cyl.dat
```

or, for VTK output, for visualisation in ParaView or VisIt, run the command

```
$NEK/FieldConvert cyl.xml cyl.vtu
```

As these meshes are converted to a linear mesh, since this is what the visualisation software supports, the curved elements can look quite faceted. To counteract this, we can add an optional `-nX` command line option to FieldConvert, which increases the number of subdivisions used to view the elements through interpolation.

Task 2.3

Run the command

```
$NEK/FieldConvert -n 8 cyl.xml cyl.vtu
```

and visualise the output, which should now look far smoother around the cylinder's edge and give a better visualisation of the high-order elements.

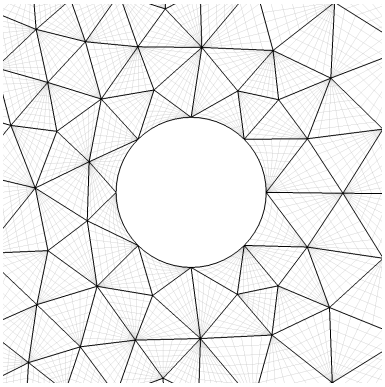

Figure 2.1 Initial triangular mesh generated by NekMesh and visualised in VisIt, using 8 subdivisions.

The initial mesh can be seen in figure 2.1. Since we want to perform a simulation of flow over the cylinder, we now need to do some additional refinement, in order to produce a suitable mesh for our solvers. The first step is to change the configuration to generate a boundary layer mesh. To do this, we first change the mesh type in the configuration file to read

```
<I PROPERTY="MeshType" VALUE="2DBndLayer" />
```

In addition to this, we need to provide information to specify the boundary to be refined and the level of refinement required. This is done by adding four additional parameters to the configuration file:

```
<P PARAM="BndLayerSurfaces" VALUE="5,6,7,8" />
<P PARAM="BndLayerThickness" VALUE="0.25" />
<P PARAM="BndLayerLayers" VALUE="2" />
<P PARAM="BndLayerProgression" VALUE="1.5" />
```

This controls: the CAD surfaces (in this 2D case, the CAD curves), onto which the boundary layer will be added; the total thickness of the boundary layer; the number of layers in the boundary layer mesh; and the geometric progression used to define the reducing height of the boundary layer elements as they approach the surface.

Task 2.4

Begin to build on this mesh by adding a quadrilateral boundary layer mesh. Edit the `cyl.mcf` file, and change the mesh type to generate a boundary layer mesh using the parameters above.

We can now try to run some simulations on this mesh. A session file `session_cyl.xml` for the cylinder mesh is provided, which will execute a low Reynolds number incompressible simulation yielding classical vortex shedding. It is a relatively quick simulation and can run a large number of convective lengths in a short time. But, using the mesh configuration described above, the simulation will be quite under-resolved. You will notice that the vortices will shed at a significant angle to the freestream.

Task 2.5

Execute the solver with

```
$NEK/IncNavierStokesSolver cyl.xml session_cyl.xml
```

Process the final flow field visualisation with

```
$NEK/FieldConvert cyl.xml cyl.fld cyl.vtu
```

and visualise the results.

Flow around a cylinder is a canonical example of unsteady flow physics. It is possible to visualise the unsteady flow by getting the solver to write checkpoint files every certain number of timesteps. The output frequency is controlled by the parameter

```
1 <P> IO_CheckSteps = 50 </P>
```

Task 2.6

Rerun the solver with the checkpoint I/O turned on, by enabling the parameter in the parameters tag in the session file. Re-run the solver with

```
$NEK/IncNavierStokesSolver cyl.xml session_cyl.xml
```

Finally, process these checkpoint files for visualisation by using the command

```
for i in {0..200}; do $NEK/FieldConvert cyl.xml cyl_${i}.chk
cyl_${i}.vtu; done
```

Note that this is a `bash` command – if you use an alternative shell, such as `tcsh`, you may need to alter this syntax. Load the files into your visualiser as a group, and use the frame functions to view the flow as an unsteady animation.

The mesh which has been generated is very coarse and the simulation is lower order ($P = 2$), and as such the simulation is under-resolved. You should now be in a position to create a new mesh with your specifications and alter the simulation accordingly. You

should note that increasing the resolution, you may affect the CFL condition and need to reduce the timestep.

A useful feature for this task is the ability to manually add refinement controls to the mesh generation, which can be used to refine the wake of the cylinder so that vortices can be captured accurately. This is done by adding a refinement tag to the .mcf file, inside the `MESHING` XML tag, which for this example takes the form:

```
<REFINEMENT>
    <LINE>
        <X1> -1.0 </X1>
        <Y1> 0.0 </Y1>
        <Z1> 0.0 </Z1>
        <X2> 15.0 </X2>
        <Y2> 0.0 </Y2>
        <Z2> 0.0 </Z2>
        <R> 1.5 </R>
        <D> 0.4 </D>
    </LINE>
</REFINEMENT>
```

In this configuration, we define a line using two points, with coordinates (`X1`,`Y1`,`Z1`) and (`X2`,`Y2`,`Z2`). `R` s the radius of influence of the refinement around this line, and `D` defines the size of the elements within the refinement region.

If under your conditions the mesh has invalid elements you can also activate the variational mesh generation tool, which will attempt to perform interior deformation by adding the following tag inside the `MESHING` tag:

```
    <BOOLPARAMETERS>
        <P VALUE="VariationalOptimiser" />
    </BOOLPARAMETERS>
```

Task 2.7

Change the mesh and solver parameters to produce different meshes and flow results, so that vortices can be captured accurately. Suggested parameters can be seen in the completed solutions.