Pre- and post-processing in Nektar++

D. Moxey, C. Cantwell, R. M. Kirby, S. Sherwin
Department of Aeronautics, Imperial College London

Nektar++ workshop
7th July 2015
Outline

• Motivation
• Pre-processing with MeshConvert
• Post-processing with FieldConvert
Motivation

• Nektar++ is a (highly-parallel) framework

• For it to be useful we need flexible utilities

• Legacy utilities: monolithic, one-per-need, lots of duplication

• **Pre-processing:** how do we read different formats and also make curvilinear meshes?

• **Post-processing:** how do visualise output, particularly for very large-scale simulations?
Example: Bioflows

Different time dependent shear metrics
Workflow: mesh generation

Linear mesh from Star-CCM+ → Convert to high-order using spherigons → Output Nektar++ XML
Workflow: simulation processing

- Extract top surface
- Solve for inflow condition using ADRSolver
- Use as boundary condition in IncNavierStokes
- Run solver to obtain base flow solution
- Visualise flow field interior streamlines
Workflow: advection-diffusion

1. Extract prism layer
2. Refine prism layer but preserve curved elements
3. Run adv-diff. using base field as advection
4. Calculate scalar gradient
5. Visualise on surface field
Preprocessing

Many preprocessing requirements:

- Lots of different input formats
- Boundary layer refinement
- Simplex element generation
- Surface smoothing
- Surface extraction

Different applications have different requirements: need flexible approach
Solution: flexible pipeline

MeshConvert: Utilises Nektar++ libraries with pipeline concept: makes preprocessing easier.
Factory patterns

Kept modular through use of factory pattern: given a key and registered classes, return an object
Factories and Nektar++

• Factories are pretty useful and being used all over Nektar++

• Straightforward to define and use with the NekFactory class inside LibUtilities

• In MeshConvert:
  • One factory for input/output/processing modules
  • Another for element type
How do I use it?

- **MeshConvert**, like everything else Nektar++, is driven through its command line interface

  ```
  MeshConvert \
  -m module1:opt1=a:opt2=b \
  -m module2:opt3=c:opt4   \
  input.xml output.xml
  ```

- Each module specifies its own options

- Input/output modules use file extensions

- Processing modules specified using `-m` and run in the order specified
Some examples

Extract a surface:

MeshConvert -m extract:surf=1-4 \ in.xml out.xml

Refine a boundary layer:

MeshConvert -m bl:surf=1:layers=5:r=4 \ in.xml out.xml

Apply a scalar function to a surface:

MeshConvert -m scalar:surf=1:scalar=x^2+y^2 \ in.xml out.xml
High-order mesh generation

B-Rep
High-order mesh generation

Curving mesh often leads to invalid elements
Isoparametric mapping

Shape function is a mapping from reference element (parametric coordinates) to mesh element (physical coordinates)

An isoparametric approach to high-order curvilinear boundary-layer meshing
Boundary layer mesh generation

Subdivide the reference element in order to obtain a boundary layer mesh

An isoparametric approach to high-order curvilinear boundary-layer meshing
Flexibility

- Use of geometric progression allows sequence of meshes to be generated

\[ r = 1 \quad r = 1\frac{1}{2} \quad r = 2 \]
More complex transforms

To demonstrate this point, we first examine the problem of figure 3, which depicts an example where a quadrilateral is split along a diagonal edge in order to obtain two triangles. We may again utilise an affine mapping $f(x) = x$ in order to map $\Omega_{\text{tri}}$ onto a subdomain $f(\Omega_{\text{quad}})$. From our previous argument we see that each component of $z = c \circ f$ has degree 2 in general if the original quadrilateral is of order $P$. Since $z^2 \subset P(\Omega_{\text{tri}})$ we must select a sufficiently large polynomial order for the triangular space so that all terms of the expansion are represented in the resulting expansion. To guarantee this for a general quadrilateral-to-triangle split, given a quadrilateral of order $P$ we must generate triangles of order $2P$. Then the space $P(\Omega_{\text{quad}}) \rightarrow P(\Omega_{\text{tri}})$ and thus $z$ captures all curvature of the original mapping. For a visual illustration of this, we may represent the polynomial spaces of the triangular and quadrilateral elements in the form of a Pascal’s triangle as shown in figure 4.

Figure 5 illustrates the problem of using triangular elements which are not sufficiently enriched. On the left, a second-order ($P_2$) quadrilateral is split into two second-order triangles. Splitting the quadrilateral into two $P_2$ triangles leads to the generation of degenerate elements. In this case, the symmetry of the deformed element coupled with the quadratic order of the triangles means that the diagonal edge which bisects the quadrilateral is forced to remain straight and thus causes a self-intersection. We note that in this example, the interior quadrilateral mode $x_1^2 x_2^2$ is not captured.

Quads to triangles  Prisms to tetrahedra

On the generation of curvilinear meshes through subdivision of isoparametric elements
Inside MeshConvert

• Boundary layer splitting is the in \textit{bl} module
  • Does prism and hex refinement

• Prism to tet splitting is in the \textit{tetsplit} module
  • In theory this can be extended to other element to tet splitting
FieldConvert

• Like MeshConvert, but for post-processing

• Same command line usage, but now you use multiple input files (since you generally have .xml and .fld files)

• Supports parallel execution, uses Nektar++ parallel format (directories with one file per process)

• Also has a wider range of command line options

• Wide range of processing modules

• Tecplot and VTK output formats
FieldConvert modules

- **extract**: Extract a boundary region
- **c0projection**: Apply a $C^0$ projection to the field
- **deform**: Deform the mesh according to input
- **equispaced**: Create equispaced output files
- **grad**: Calculate gradient fields
- **isocontour**: Create linear isocontours of field
- **multishear**: Compute shear components
- **qcritereion**: Calculate $Q$-criterion for vortex detection
- **scalgard**: Calculate scalar gradient field
- **vorticity**: Compute vorticity field of fluid
- **wss**: Calculate wall shear stress
- **interpfield**: Interpolate one field onto another
Some examples

Convert to VTK:  
FieldConvert in.xml in.fld out.vtu

Optionally specify a range + output order:
FieldConvert -r -2,3,1,2 -n 10 in.xml in.fld out.vtu

Interpolate data from one mesh to another:
FieldConvert -m interpfield:fromxml=f.xml:fromfld=f.fld \ out.xml out.fld

Generate vorticity:
FieldConvert -m vorticity in.xml in.fld out.fld
Conclusions

• We now have a range of flexible pre- and post-processing strategies

• **Coming soon:**
  
  • Incorporate mesh generation from CAD (M. Turner)
  
  • Better parallel file formats for very large scale jobs based on HDF (R. Nash)
  
  • Condensed mesh geometry formats to reduce memory footprint and solver pre-processing time
Thanks for listening!

@davidmoxey

d.moxey@imperial.ac.uk

nektar-users@imperial.ac.uk