# 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

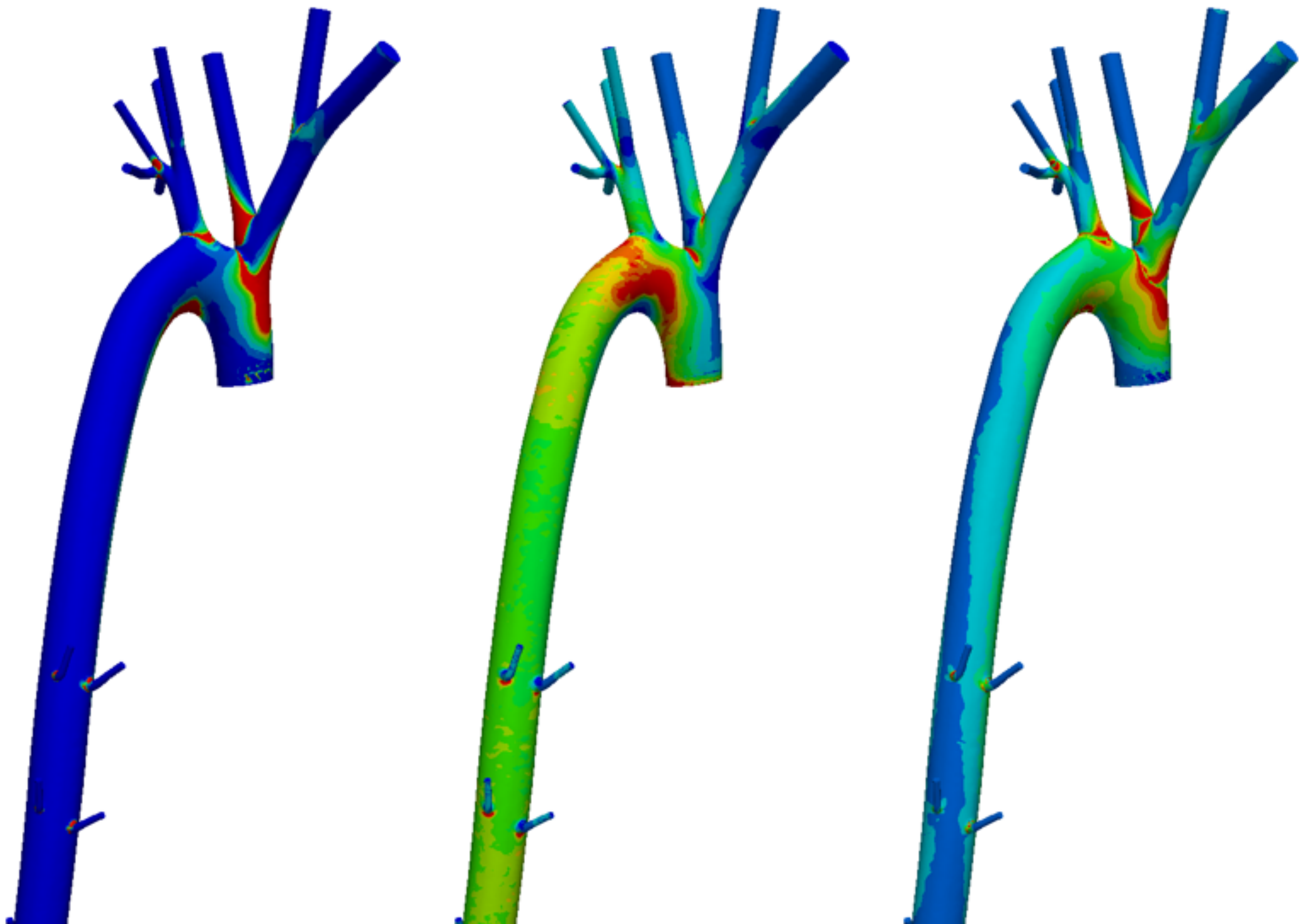
- Motivation
- Pre-processing with `MeshConvert`
- Post-processing with `FieldConvert`

# 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?

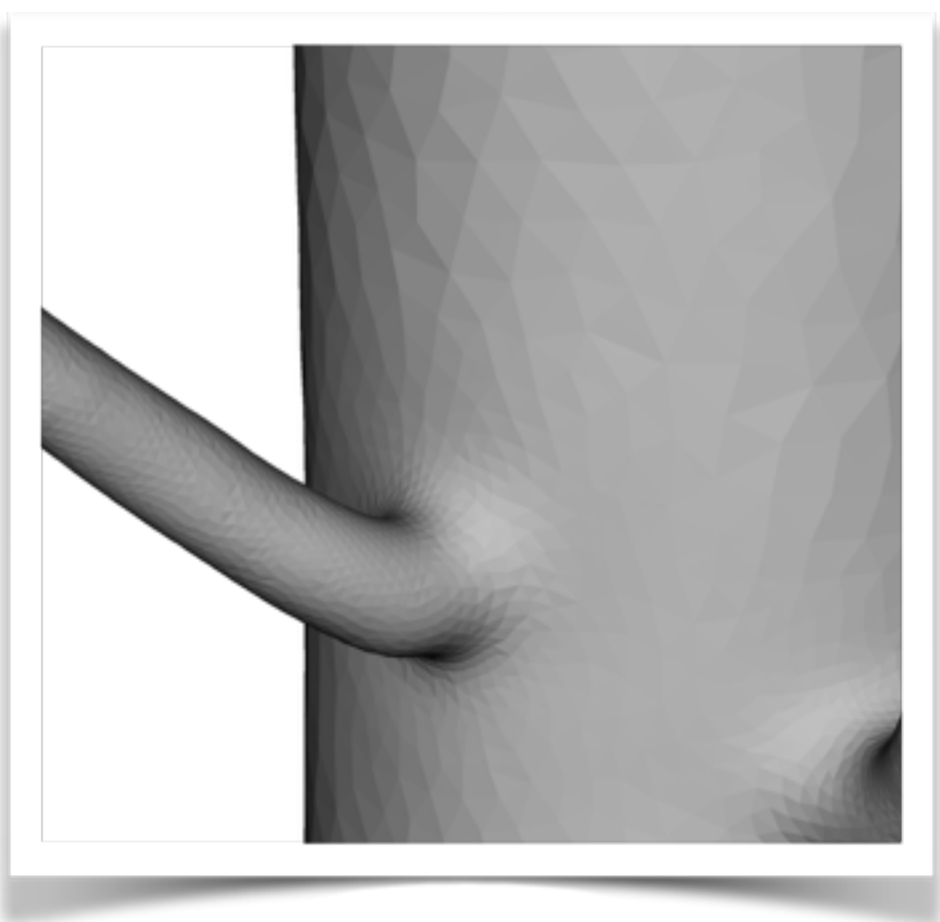
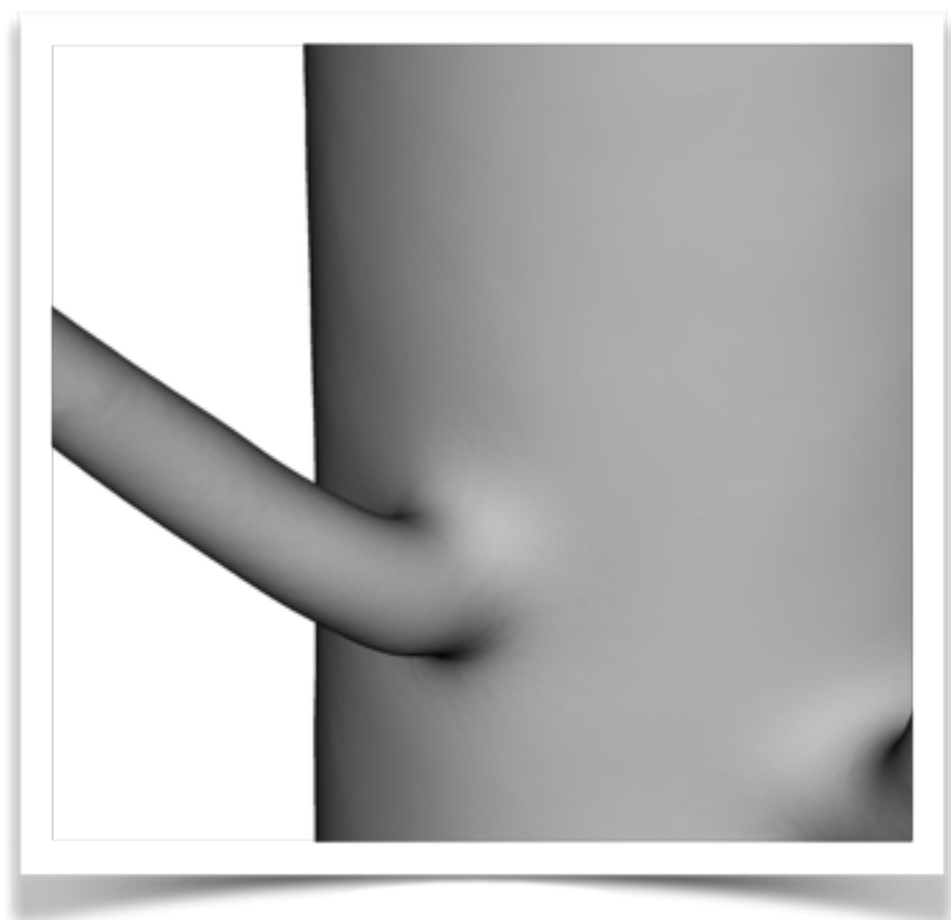
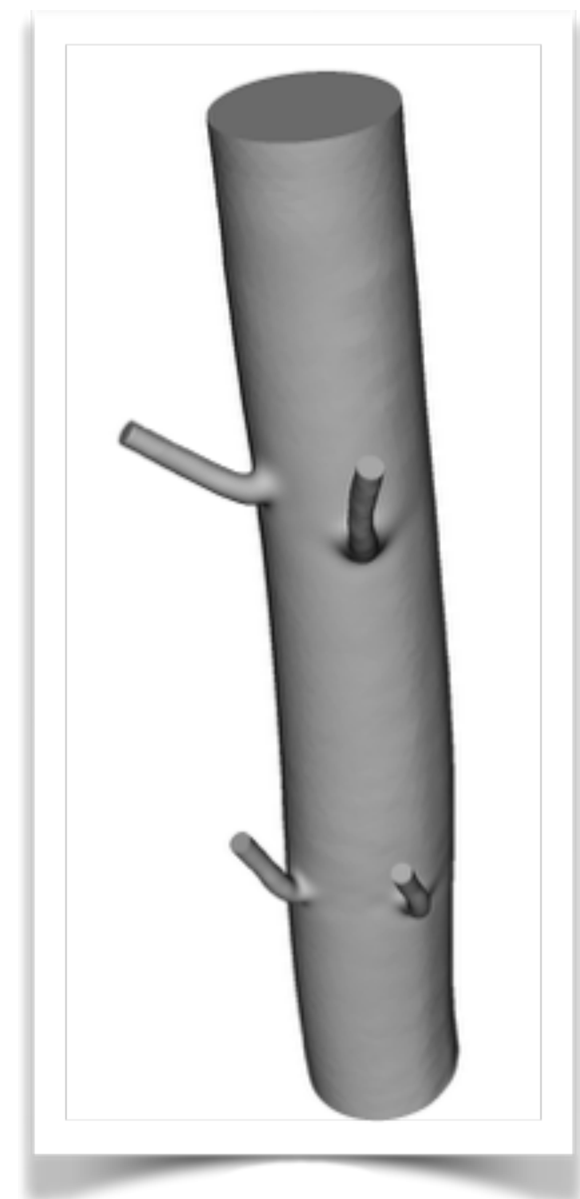
# Example: Bioflows

Different time dependent shear metrics

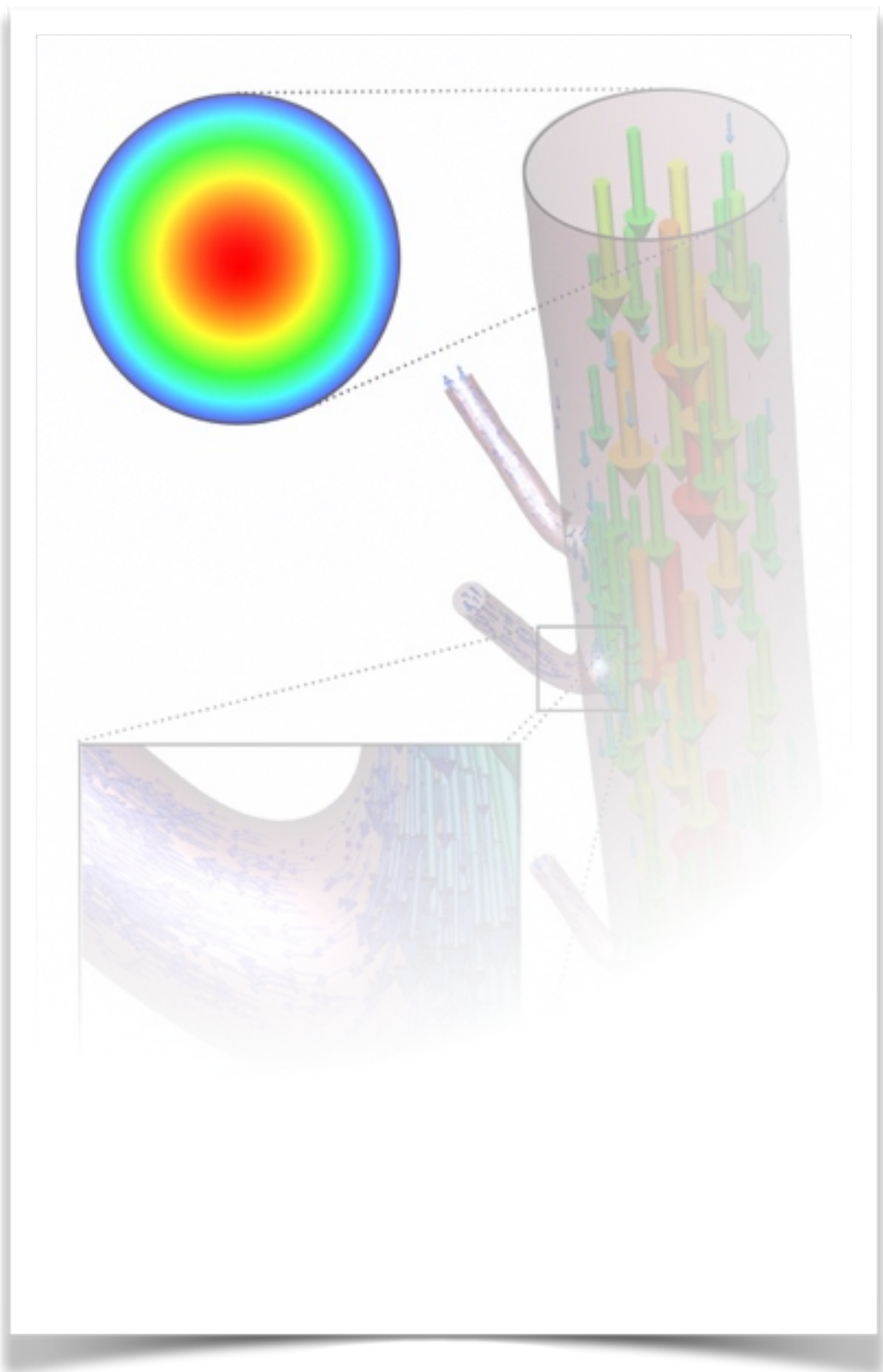


# Workflow: mesh generation

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

↓

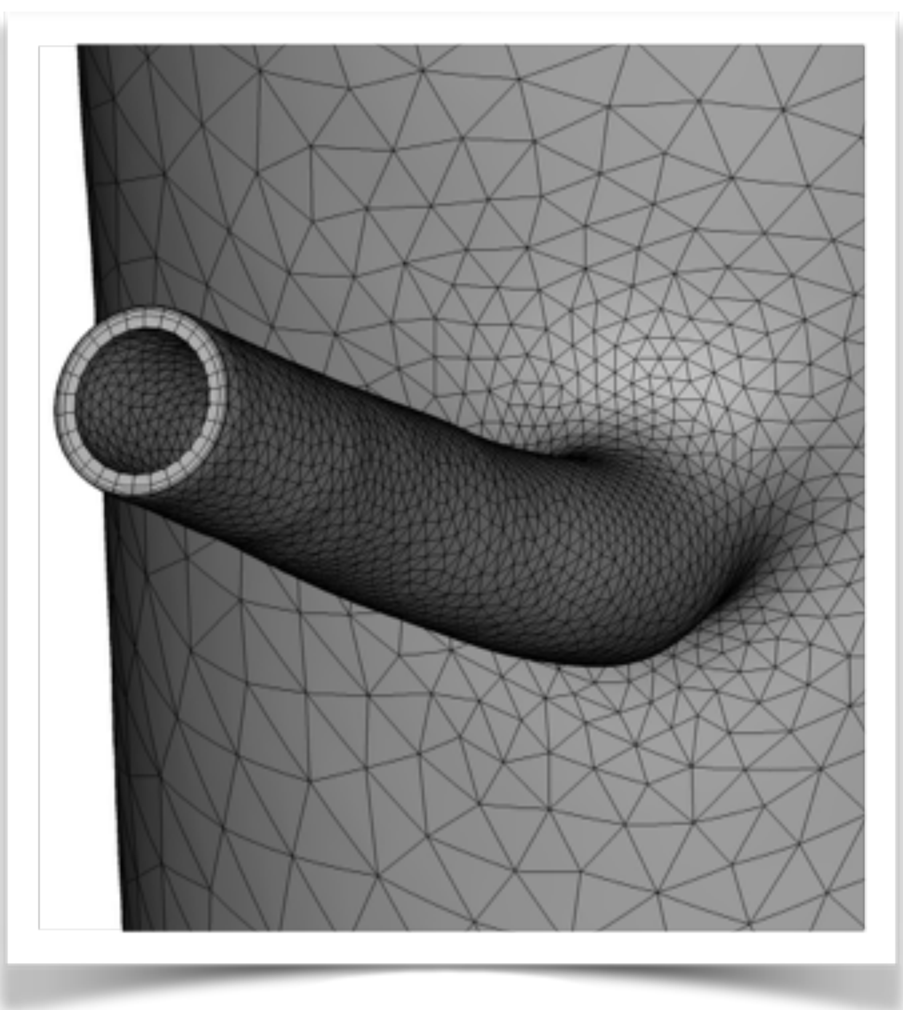
Run solver to obtain base flow solution

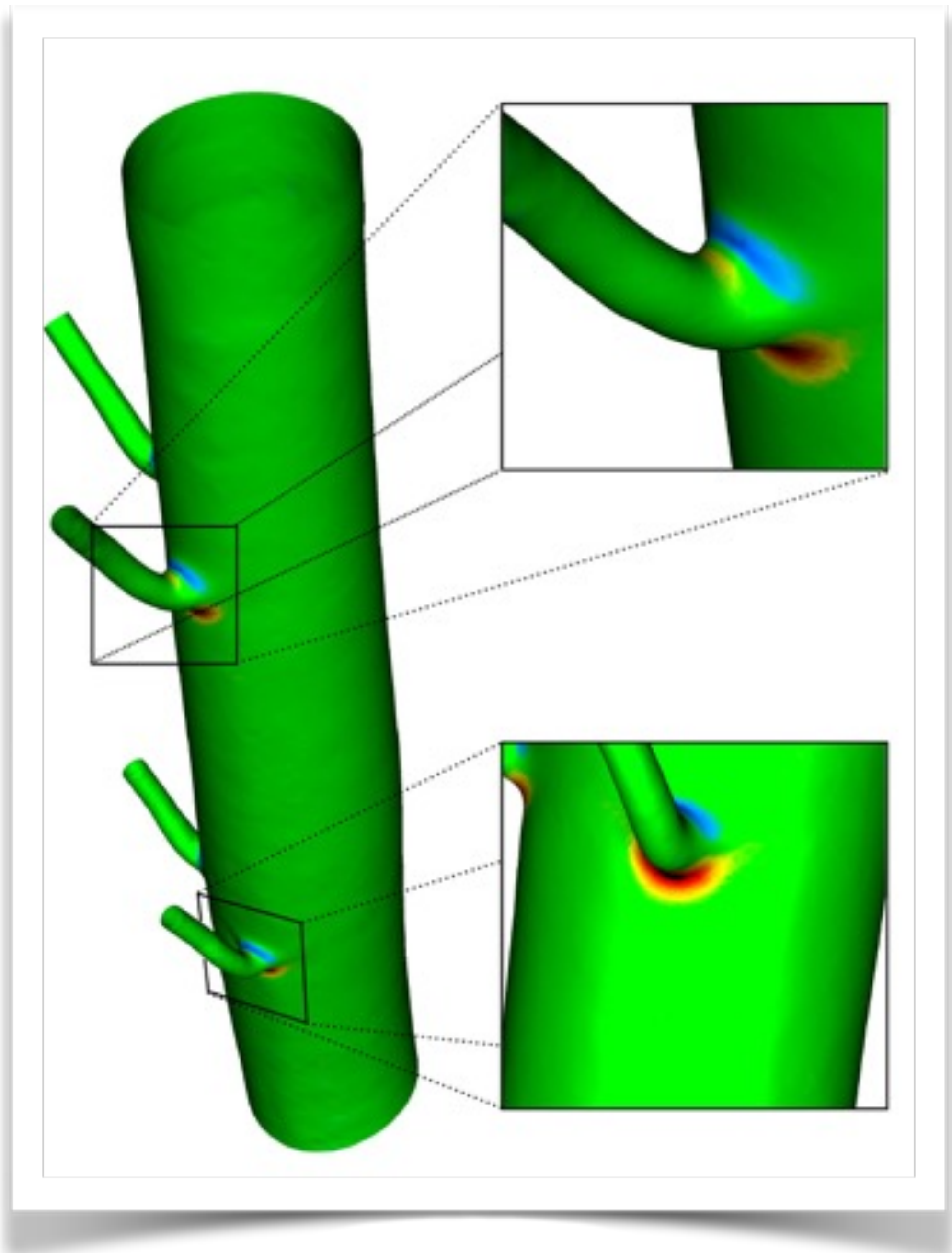
↓

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





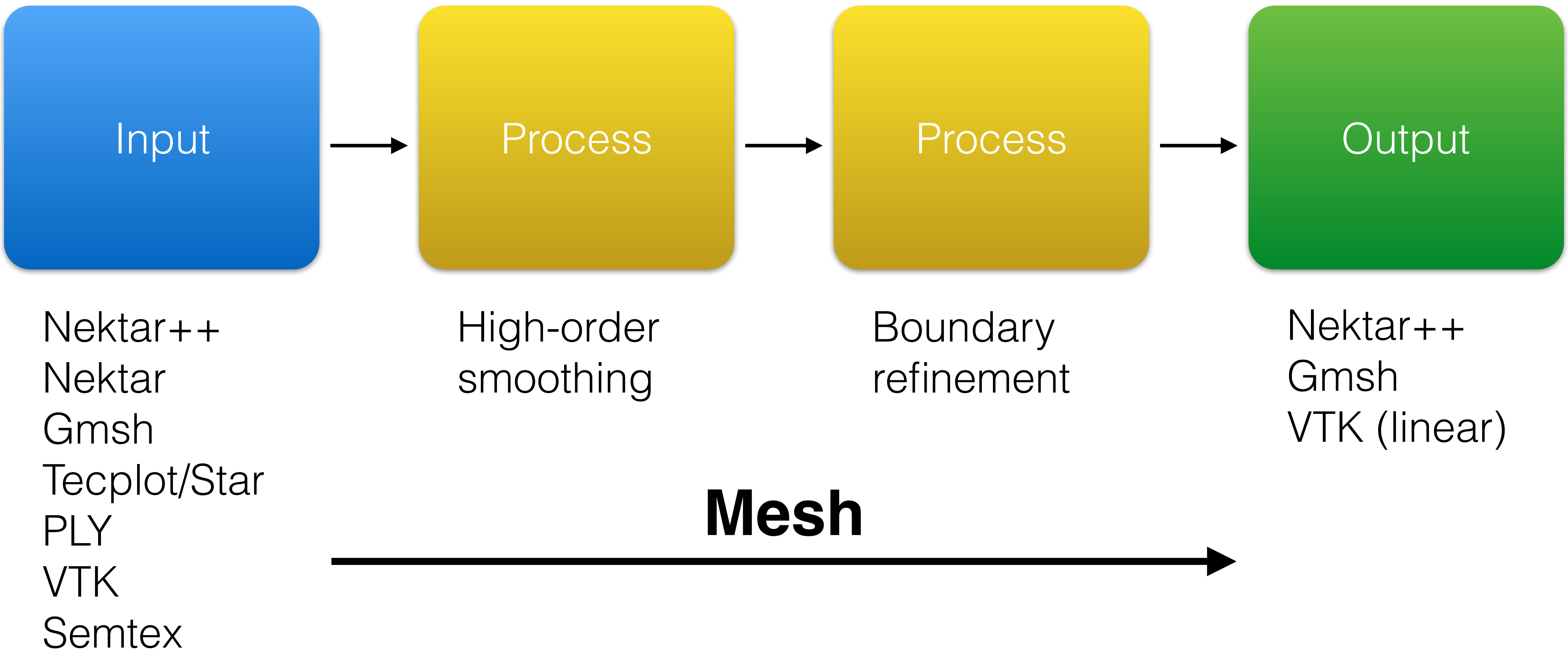
# Preprocessing

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

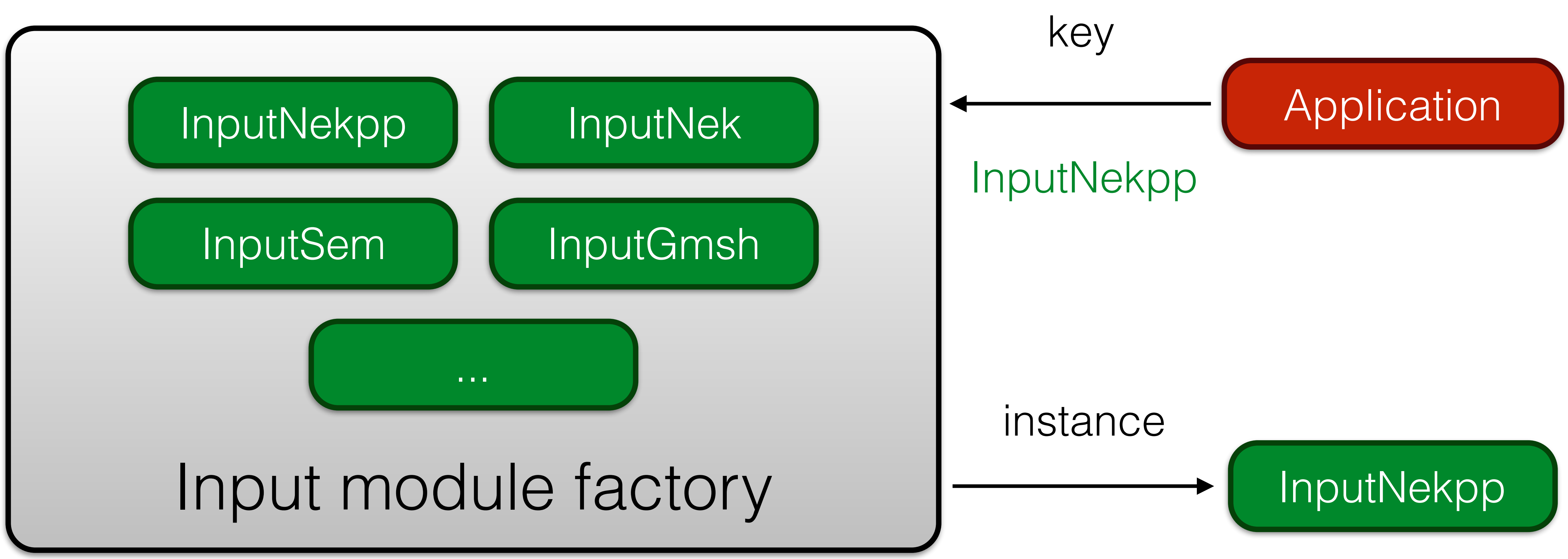
# Solution: flexible pipeline




`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
```

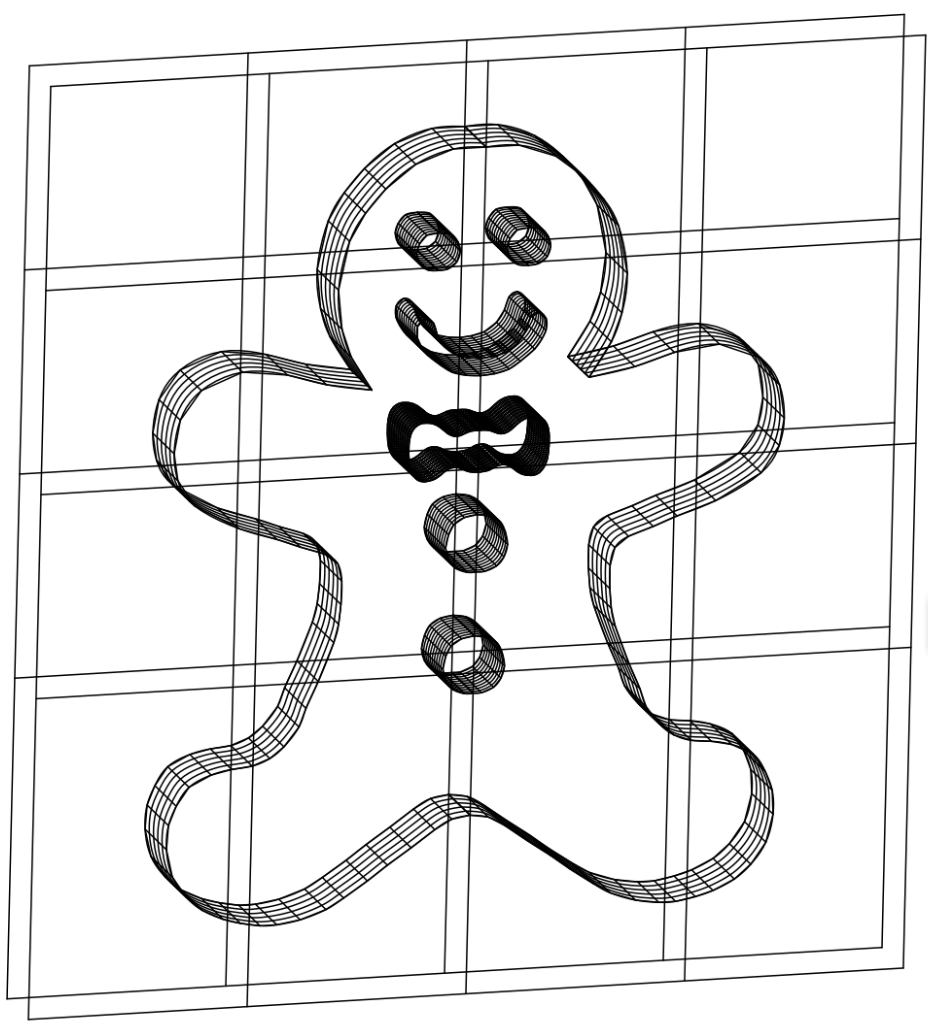
**Refine a boundary layer:**

```
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
```

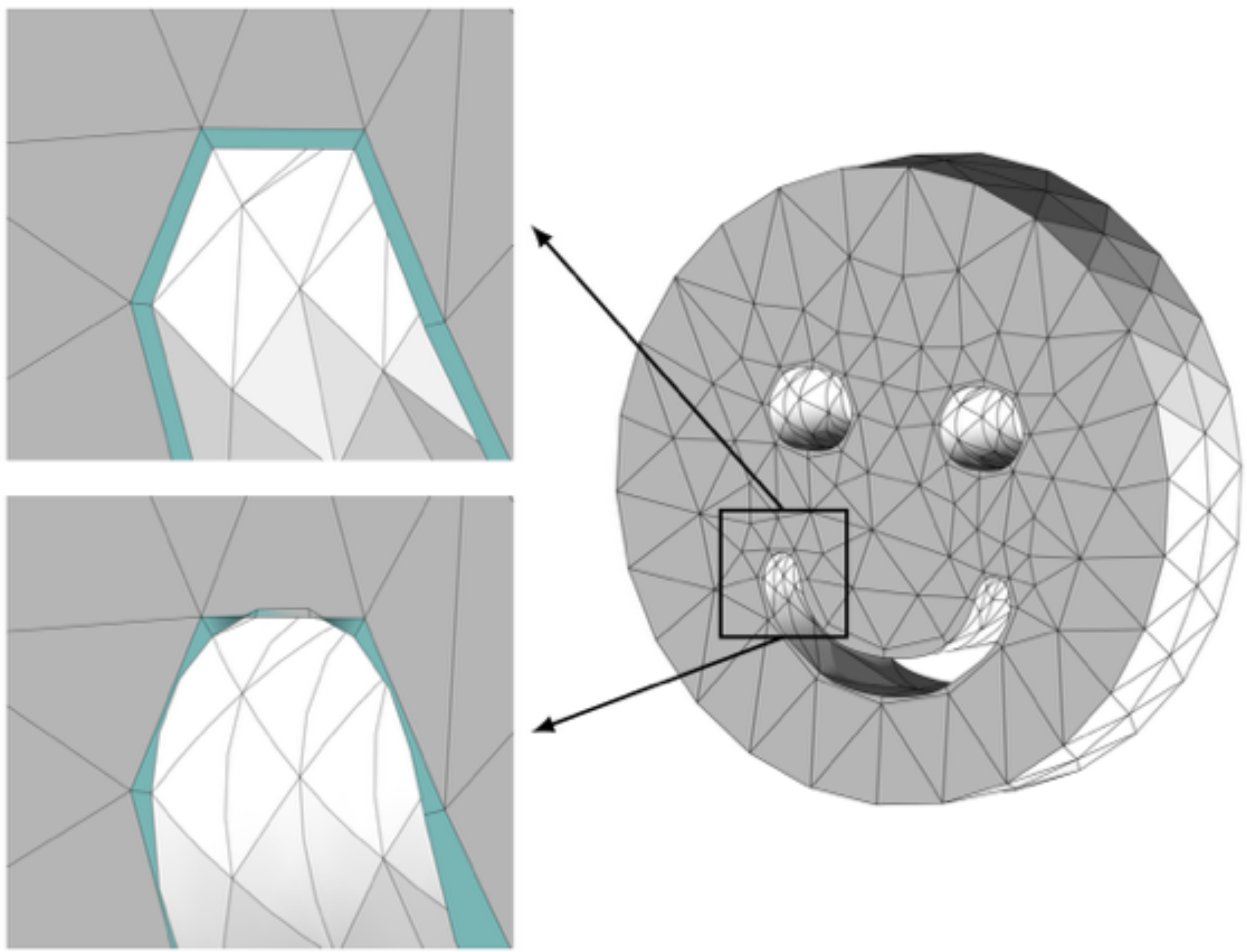
# High-order mesh generation



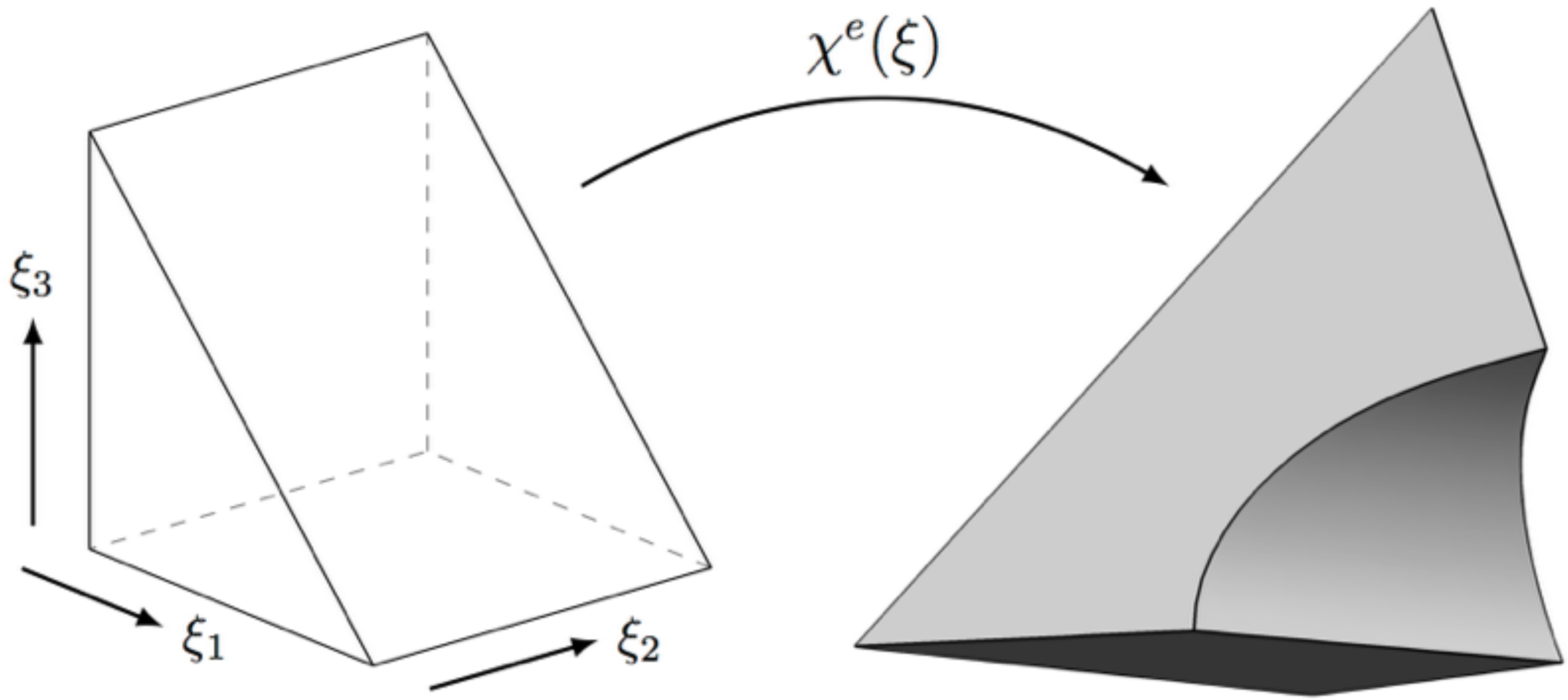
B-Rep

# High-order mesh generation

Curving mesh often leads to invalid elements



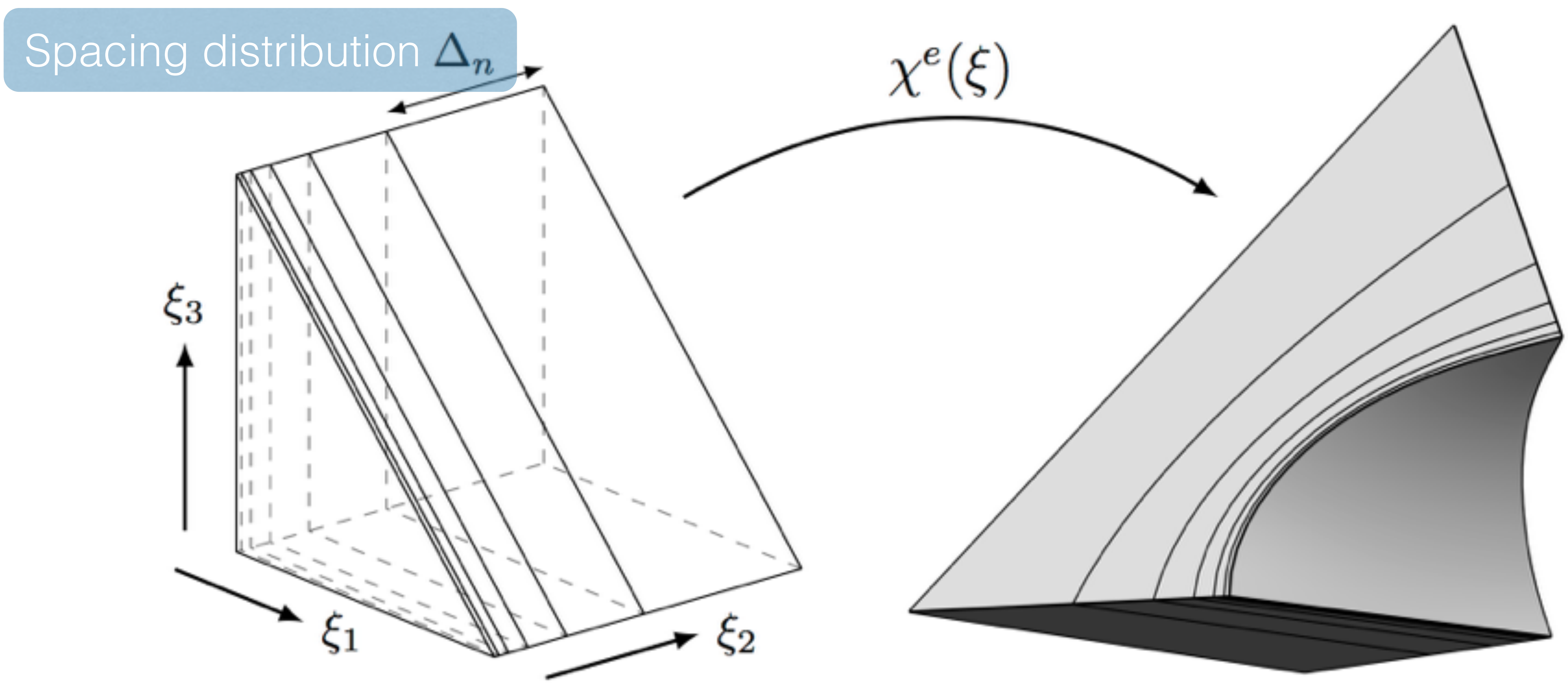
# Isoparametric mapping




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*
D. Moxey, M. Hazan, S. J. Sherwin, J. Peiró, Comp. Meth. Appl. Mech. Eng. **283**, 636-650, 2015

# Boundary layer mesh generation

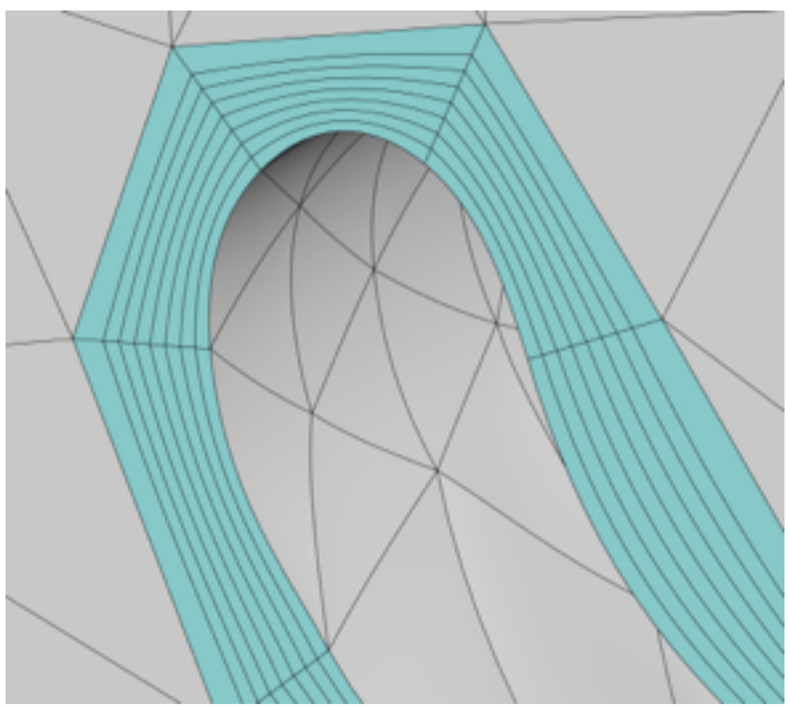



Subdivide the reference element in order to obtain a boundary layer mesh

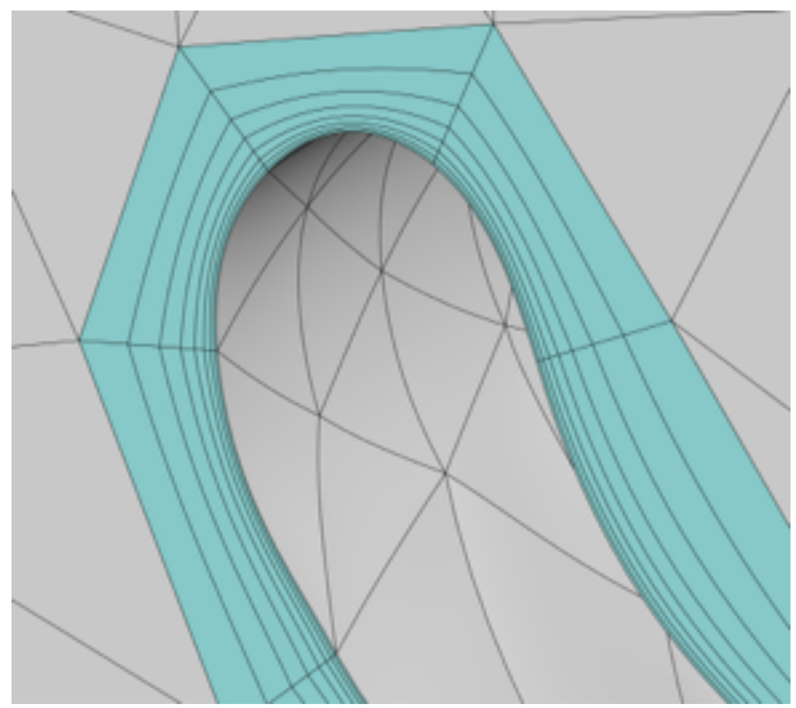
*An isoparametric approach to high-order curvilinear boundary-layer meshing*
D. Moxey, M. Hazan, S. J. Sherwin, J. Peiró, Comp. Meth. Appl. Mech. Eng. **283**, 636-650, 2015

# Flexibility

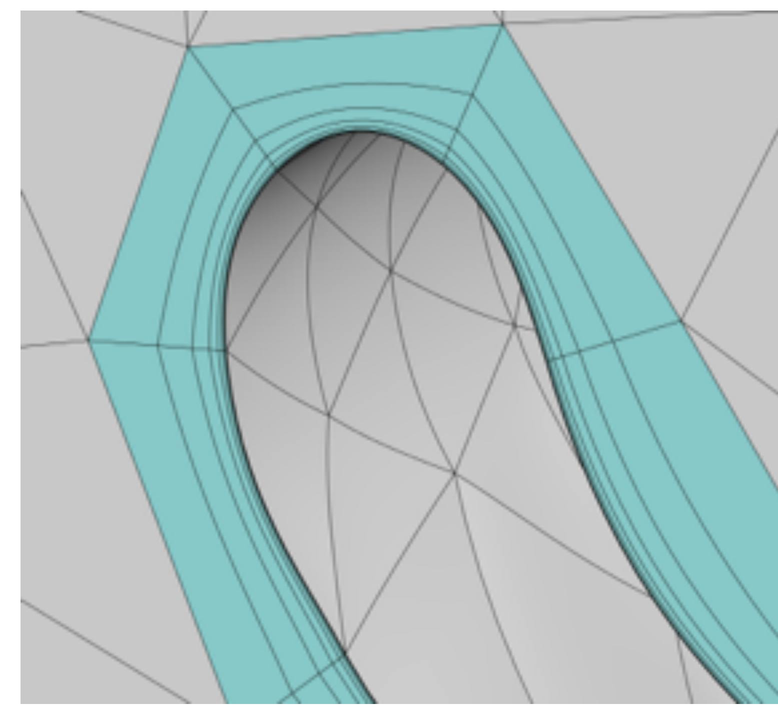
- Use of geometric progression allows sequence of meshes to be generated



$r = 1$

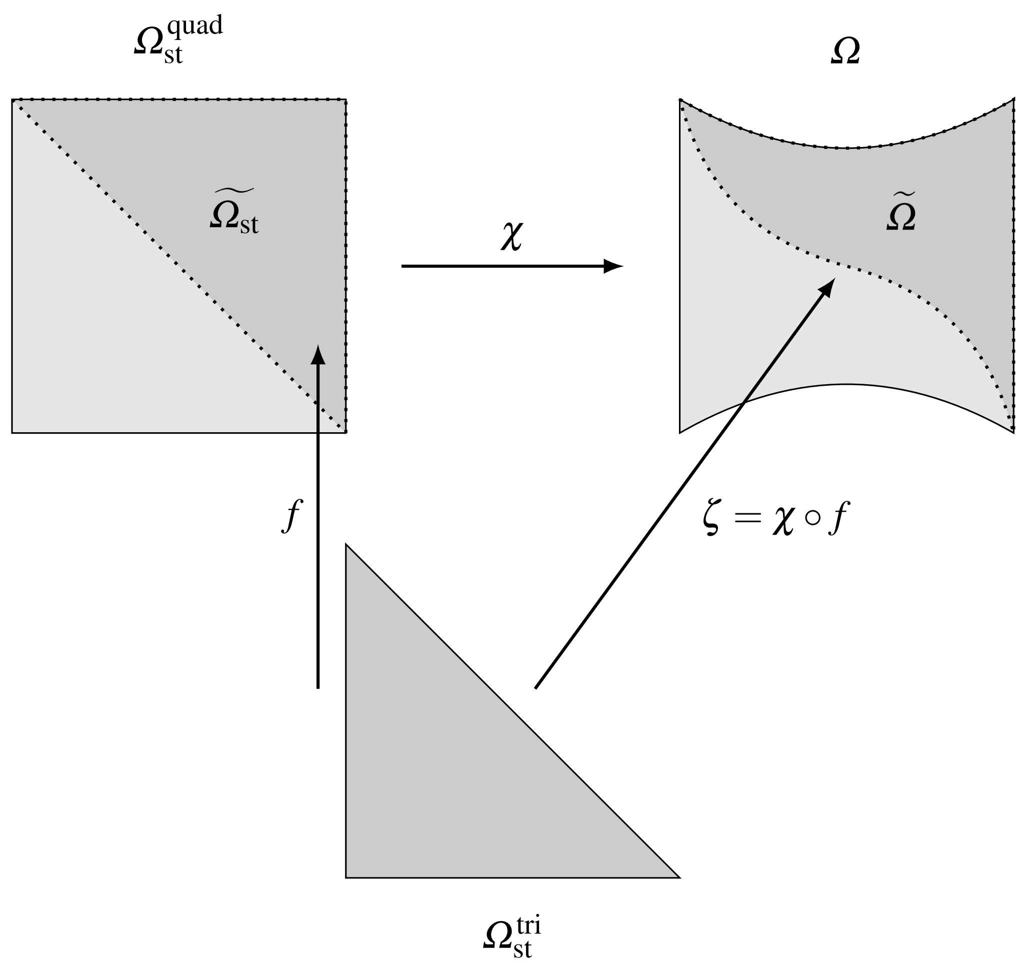


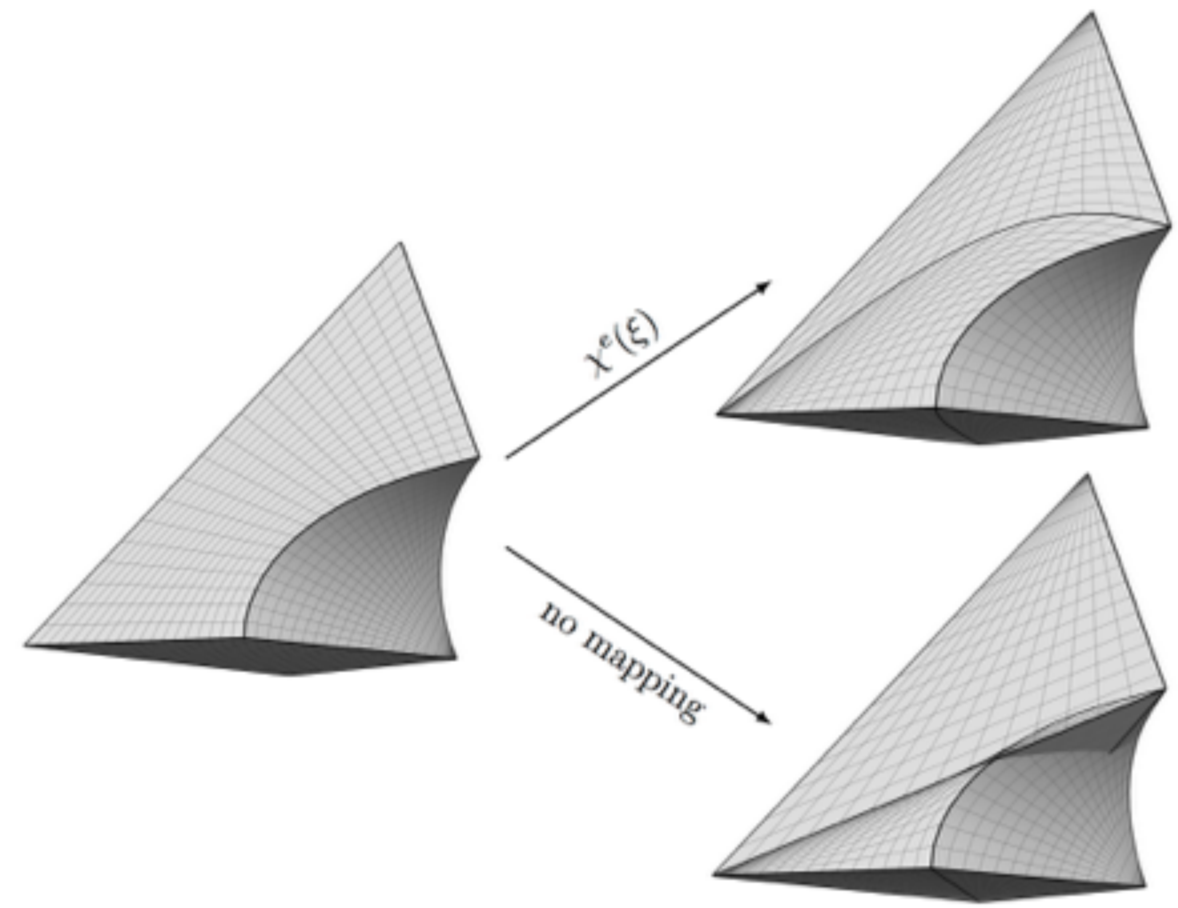
$r = 1\frac{1}{2}$



$r = 2$

# More complex transforms




Quads to triangles

Prisms to tetrahedra

*On the generation of curvilinear meshes through subdivision of isoparametric elements*
D. Moxey, M. D. Green, S. J. Sherwin, J. Peiró, New Challenges in Grid Generation and Adaptivity for Scientific Computing pp. 203-215

# Inside `MeshConvert`

- Boundary layer splitting is the in `bl` module
  - Does prism and hex refinement
- Prism to tet splitting is in the `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`<br>Extract a boundary region | `c0projection`<br>Apply a $C^0$ projection to the field | `deform`<br>Deform the mesh according to input |
| `equispaced`<br>Create equispaced output files | `grad`<br>Calculate gradient fields | `isocontour`<br>Create linear isocontours of field |
| `multishear`<br>Compute shear components | `qcriterion`<br>Calculate $Q$-criterion for vortex detection | `scalgrad`<br>Calculate scalar gradient field |
| `vorticity`<br>Compute vorticity field of fluid | `wss`<br>Calculate wall shear stress | `interpfield`<br>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