High-fidelity CFD with the Nektar++ spectral/hp element framework

David Moxey
College of Engineering, Maths & Physical Sciences, University of Exeter

T106C turbine blade
Compressible Navier-Stokes
Contours of temperature

Outline

- Motivation
- What are high order methods and why are they useful?
- Nektar++: a spectral/*hp* element framework
- Challenges (and some solutions!)
- Applications

Lombard, Moxey et al., *AIAA J* (2016)

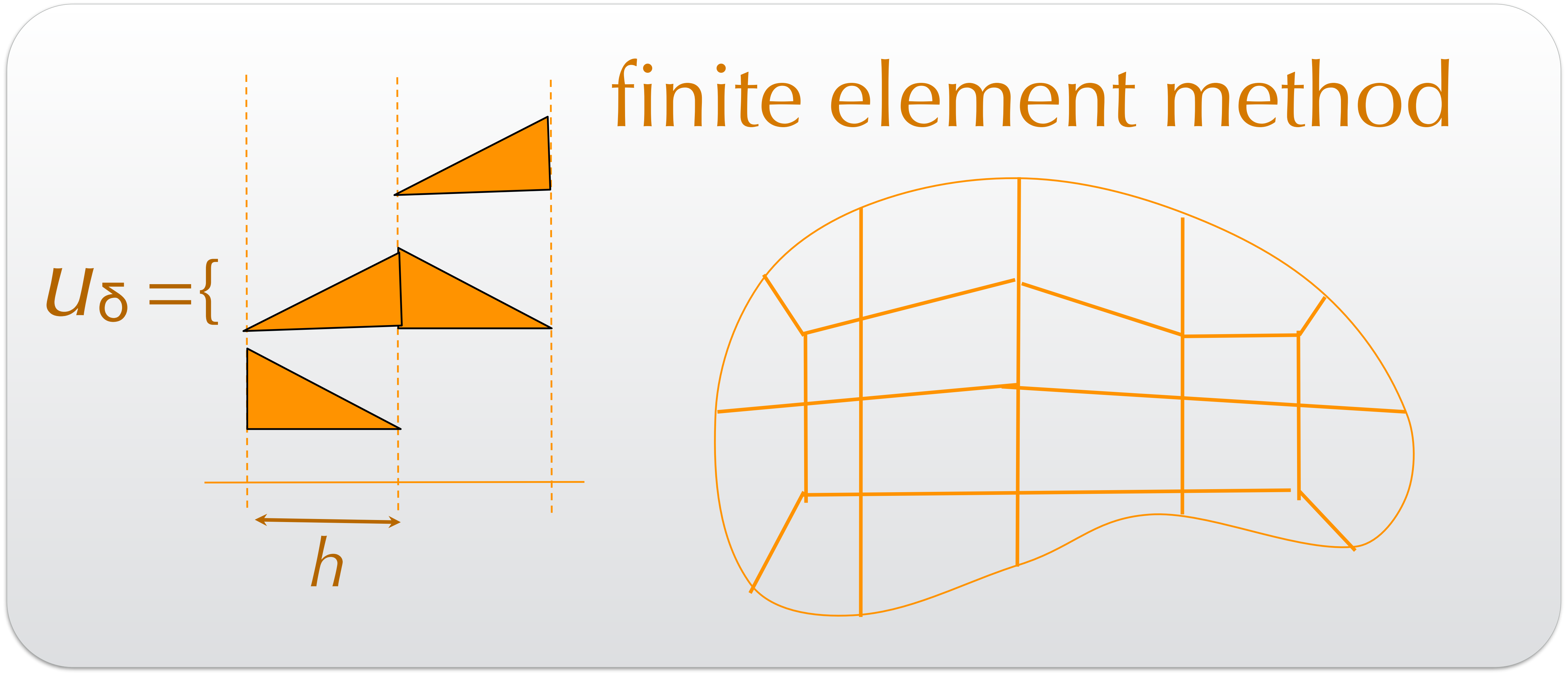
Increasing desire for **high-fidelity** simulation in high-end engineering applications.

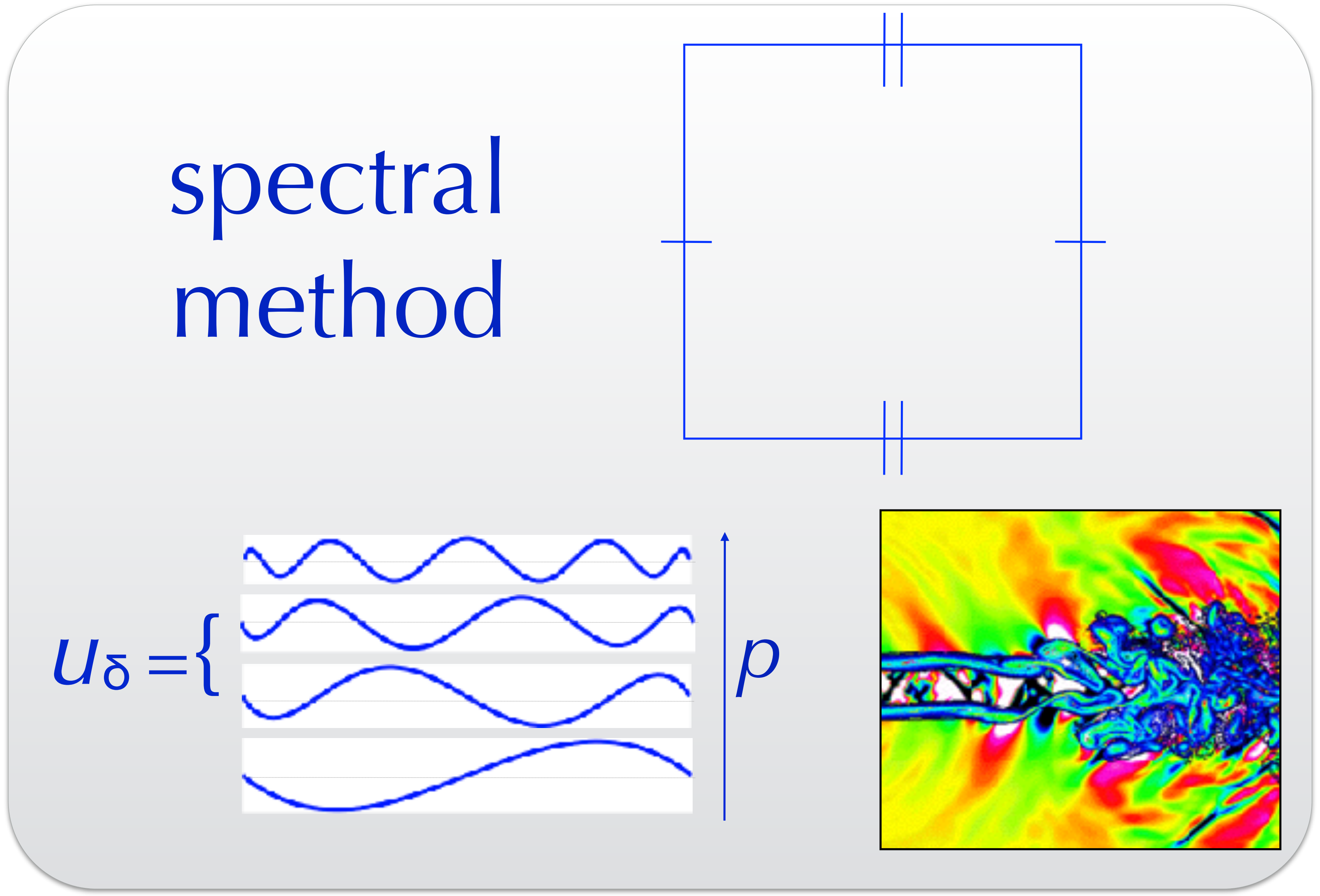
Want to accurately model difficult features:

- strongly separated flows
- feature tracking and prediction
- vortex interaction

Move towards methods and techniques for making LES affordable

What is a spectral/*hp* element?





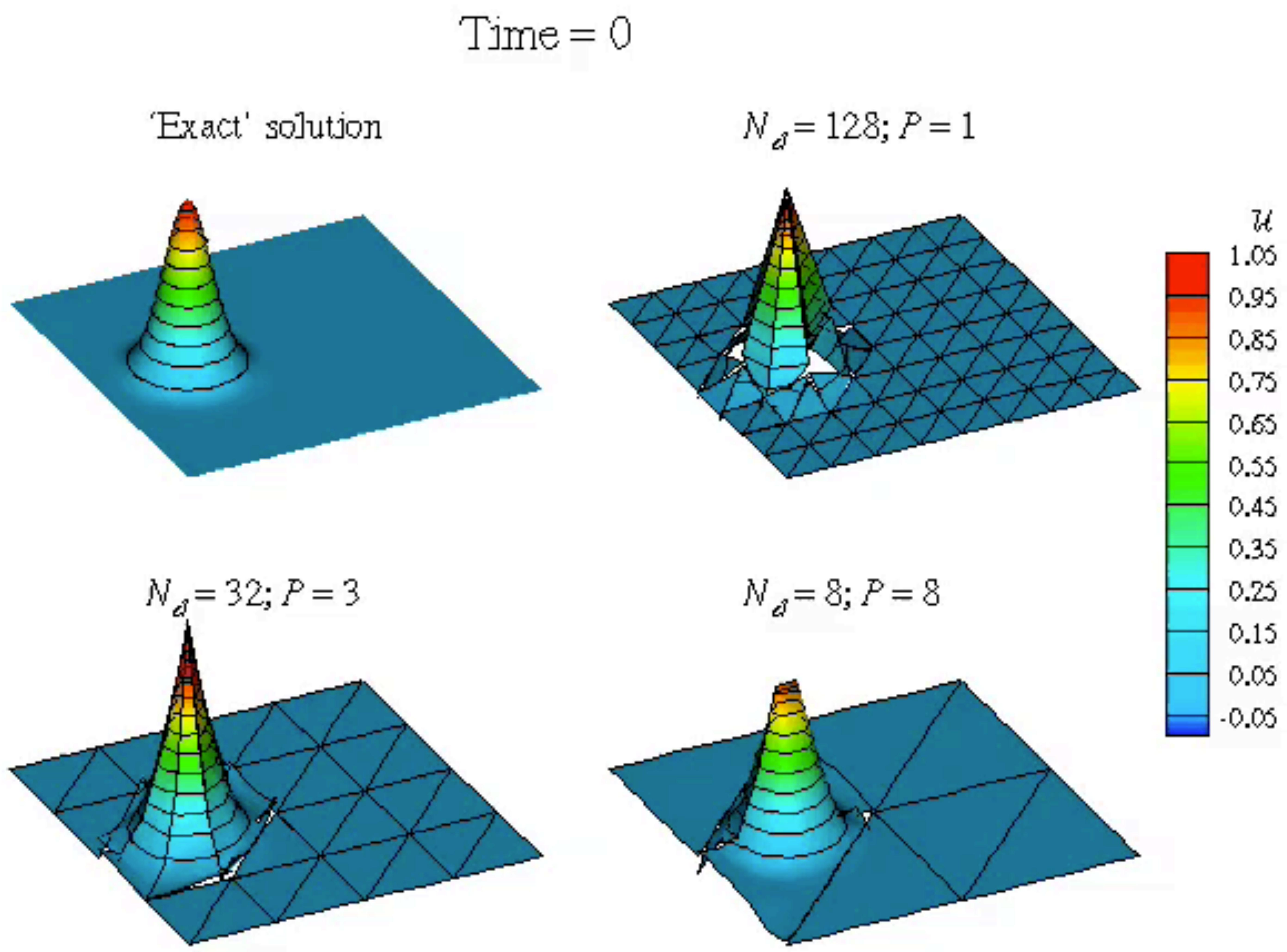
spatial flexibility (*h*)

+

accuracy (*p*)

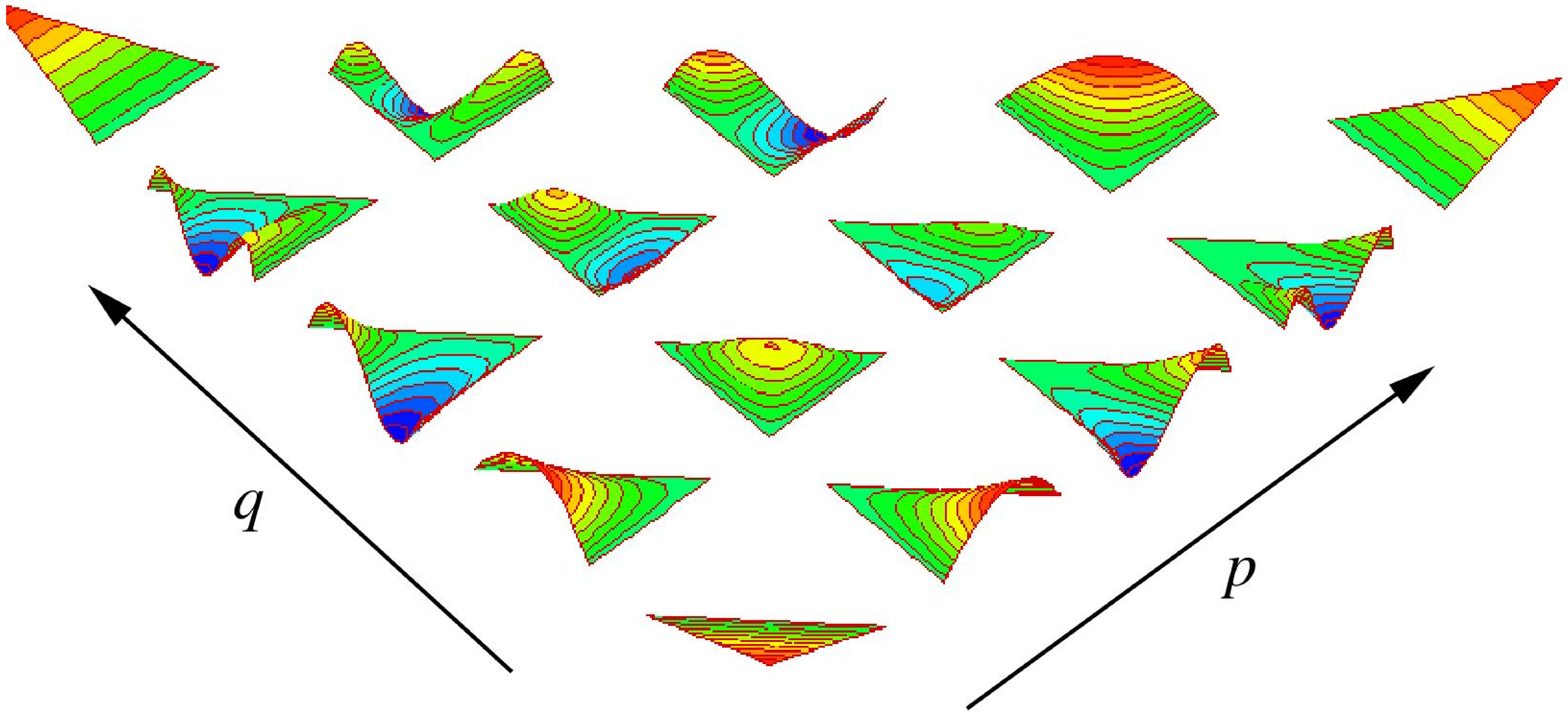
⟶ spectral/*hp* element

Why use a high-order method?

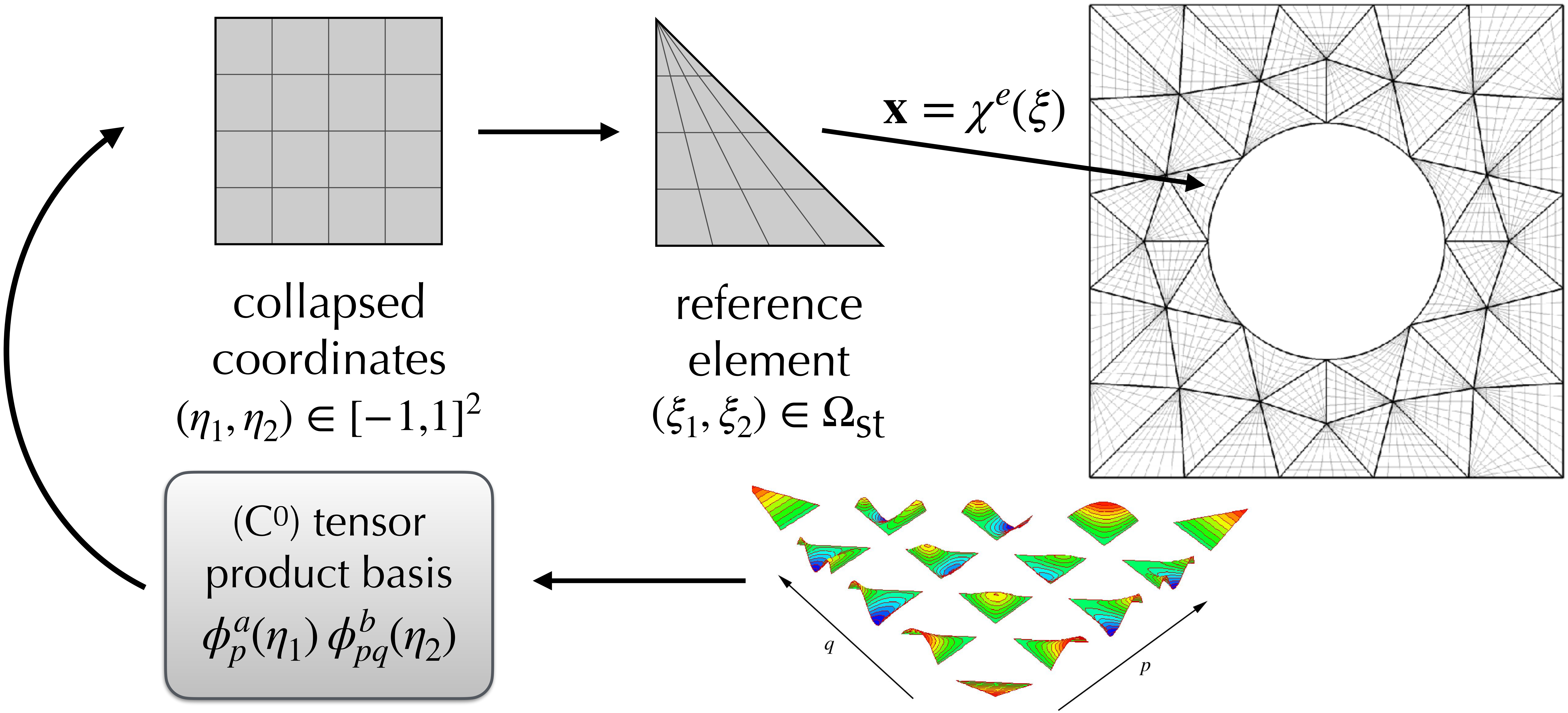


Higher-order expansions

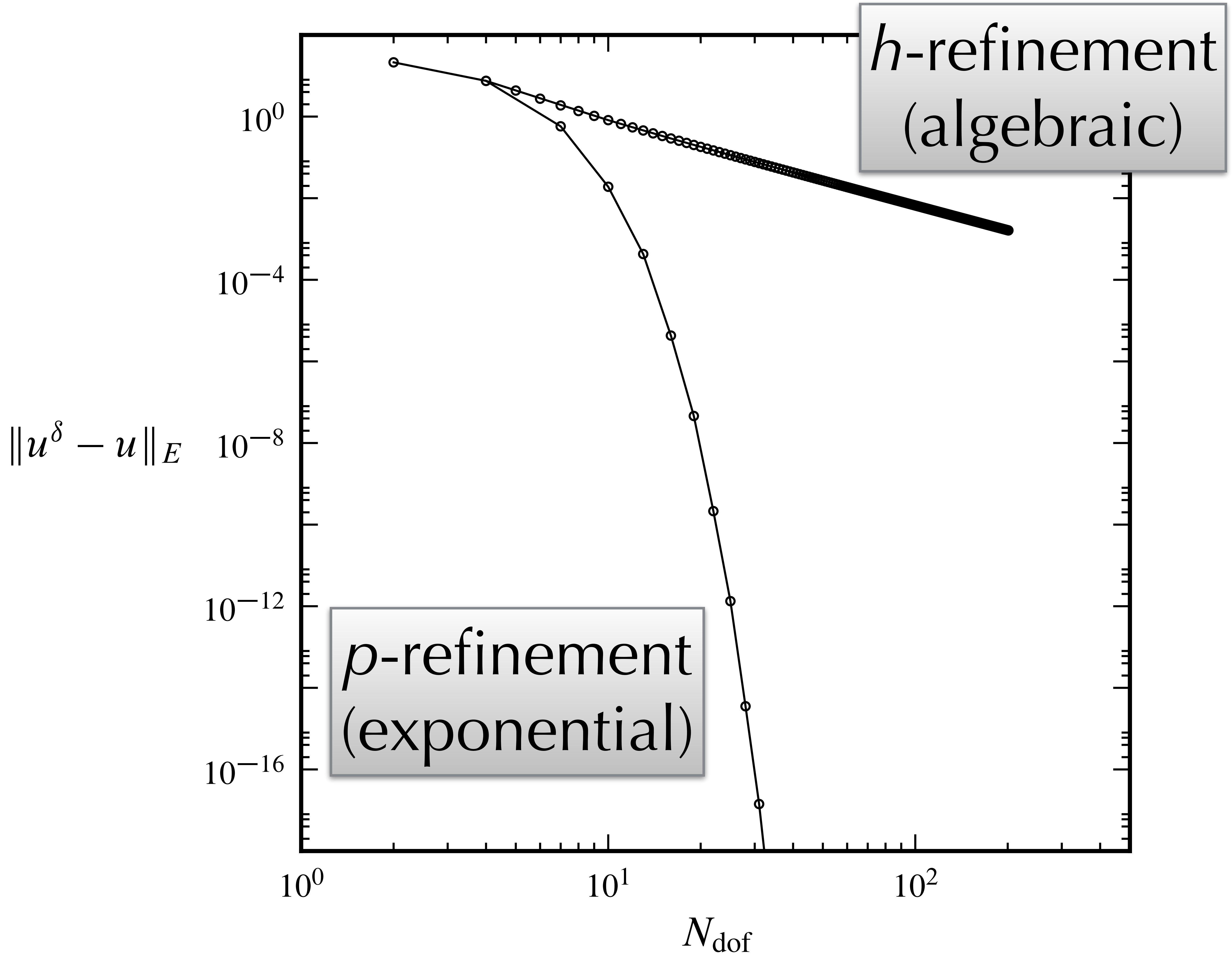
- Extend traditional FEM by adding higher order polynomials of degree *P* within each element.
- Traditional linear element has 3 degrees of freedom per element (each vertex).
- High-order has (*P*+1)(*P*+2)/2 at a given order *P*.
- Key defining feature of spectral/*hp*: **tensor product basis**.



Spectral/*hp* element formulation



Why use a high-order method?

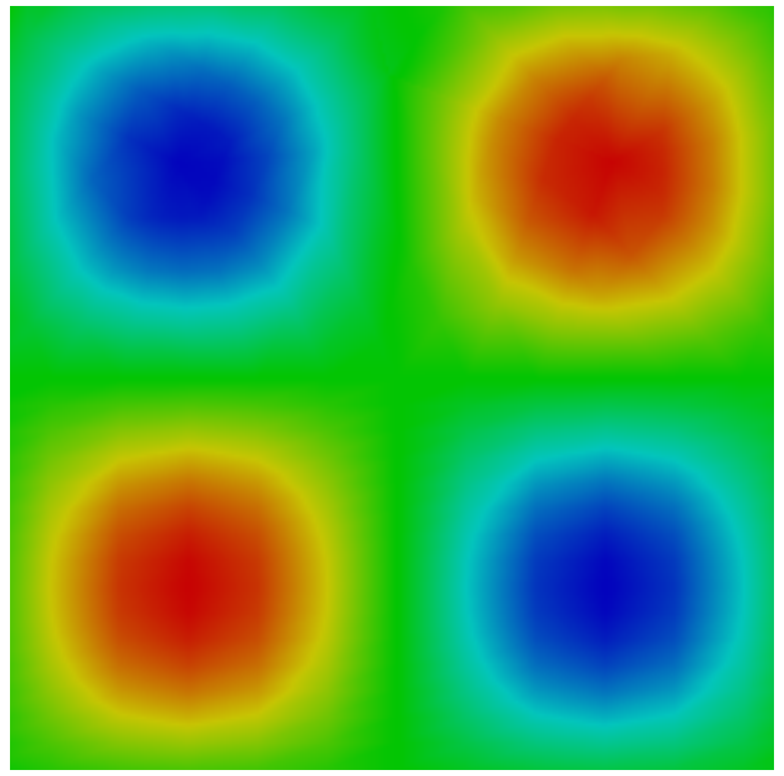


$$\nabla^2 u(x) - \lambda u(x) = -f(x)$$

Method of manufactured solutions on square domain

$$u(x) = \sin(\pi x)\sin(\pi y)$$

$$\Rightarrow f(x) = (\nabla^2 - \lambda)u(x)$$



So why doesn't everyone use high-order?

Stuff I'll discuss today:

- Pre-processing (mesh generation), particularly for complex geometries.
- Efficient linear algebra techniques & operator implementations.
- Implementation effort and difficulties.

Other stuff:

- Post-processing and visualisation, stability and robustness, preconditioning…

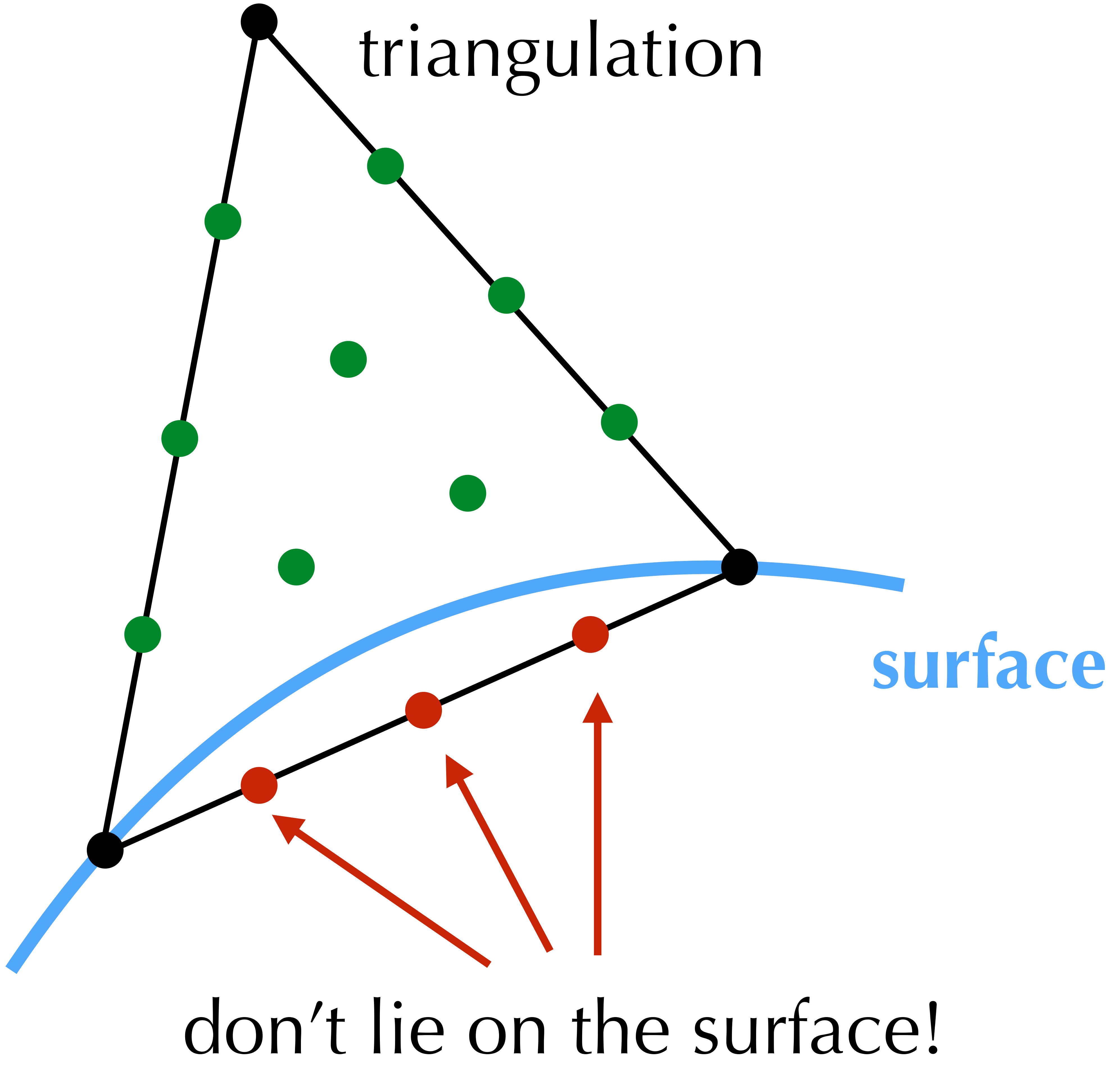
Challenge 1: high-order mesh generation

Complex geometries
look like this

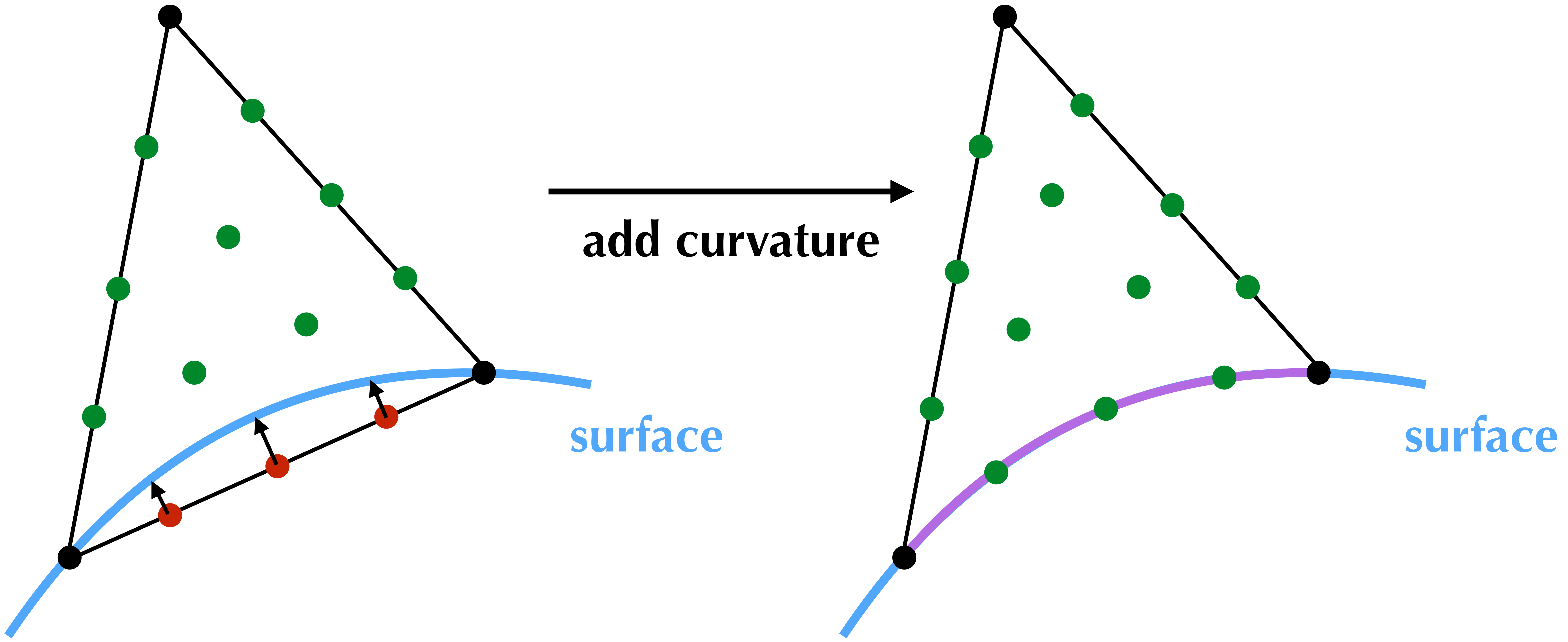
Not like this

High-order mesh generation

- Good quality meshes are **essential** to finite element and finite volume simulations.
- You can have a very fancy solver, but if you can't mesh your geometry then you **can't run your simulation**!
- At high orders we have an additional headache, as we must **curve the elements** to fit the geometry.

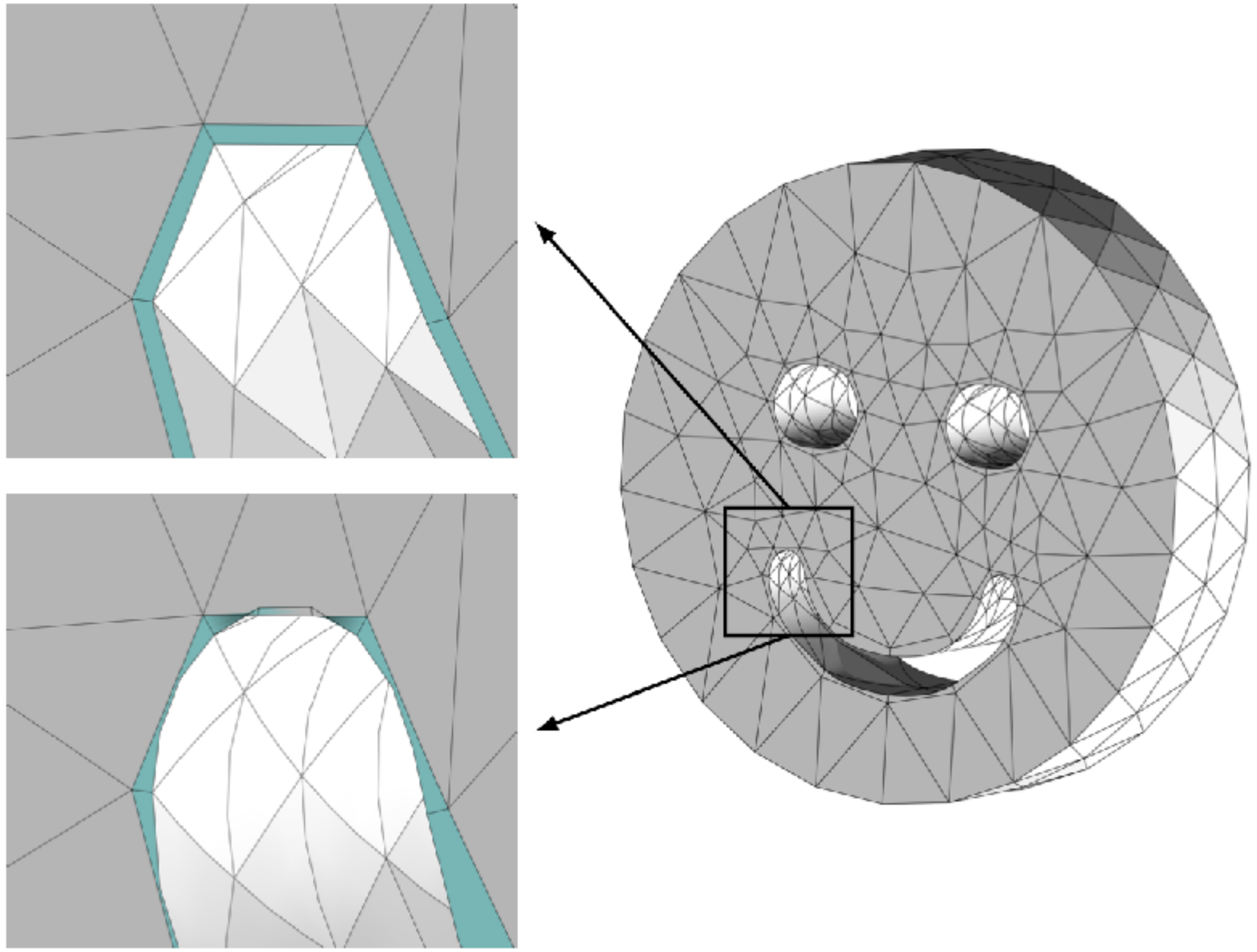


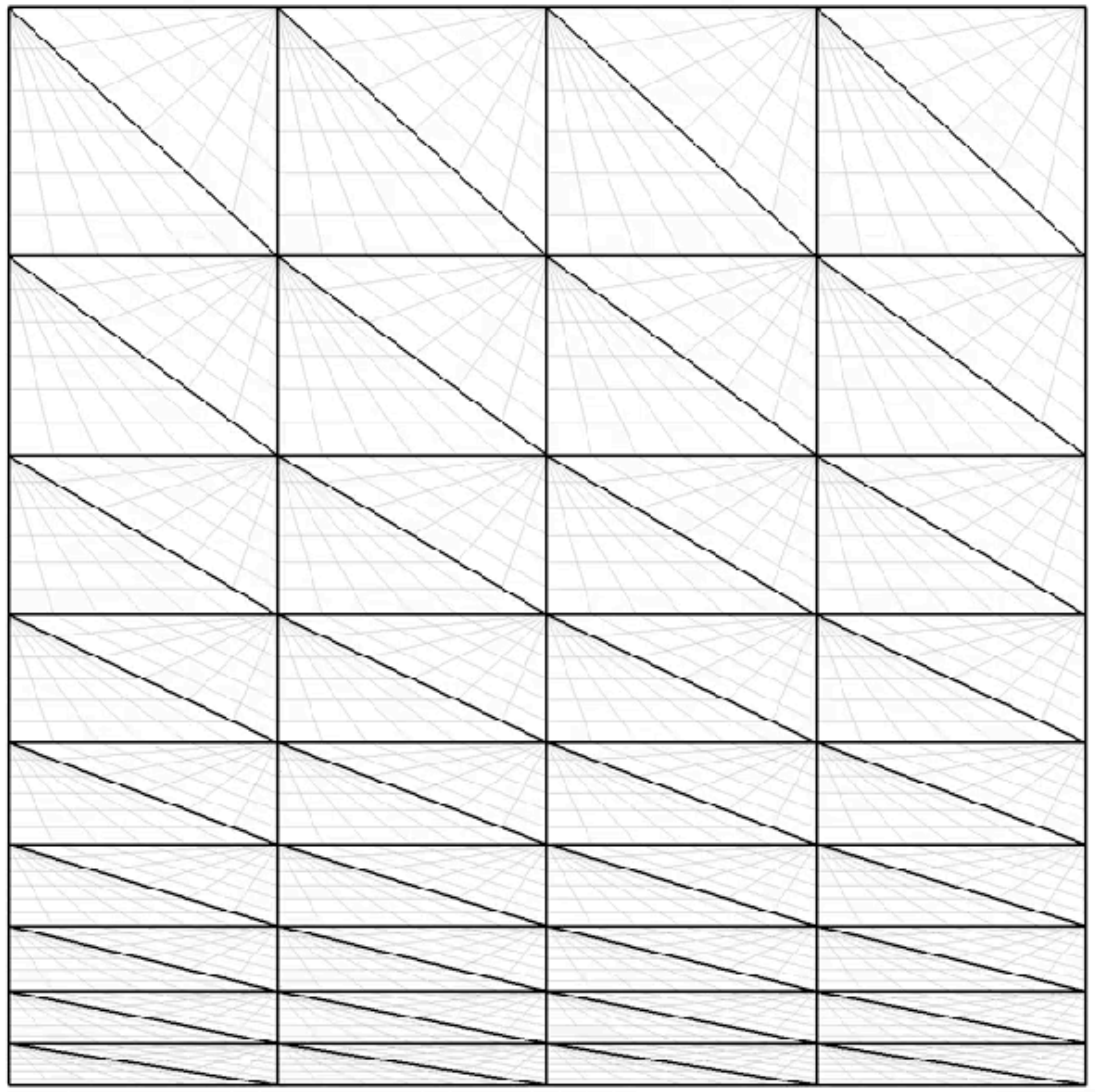
High-order mesh generation



High-order mesh generation

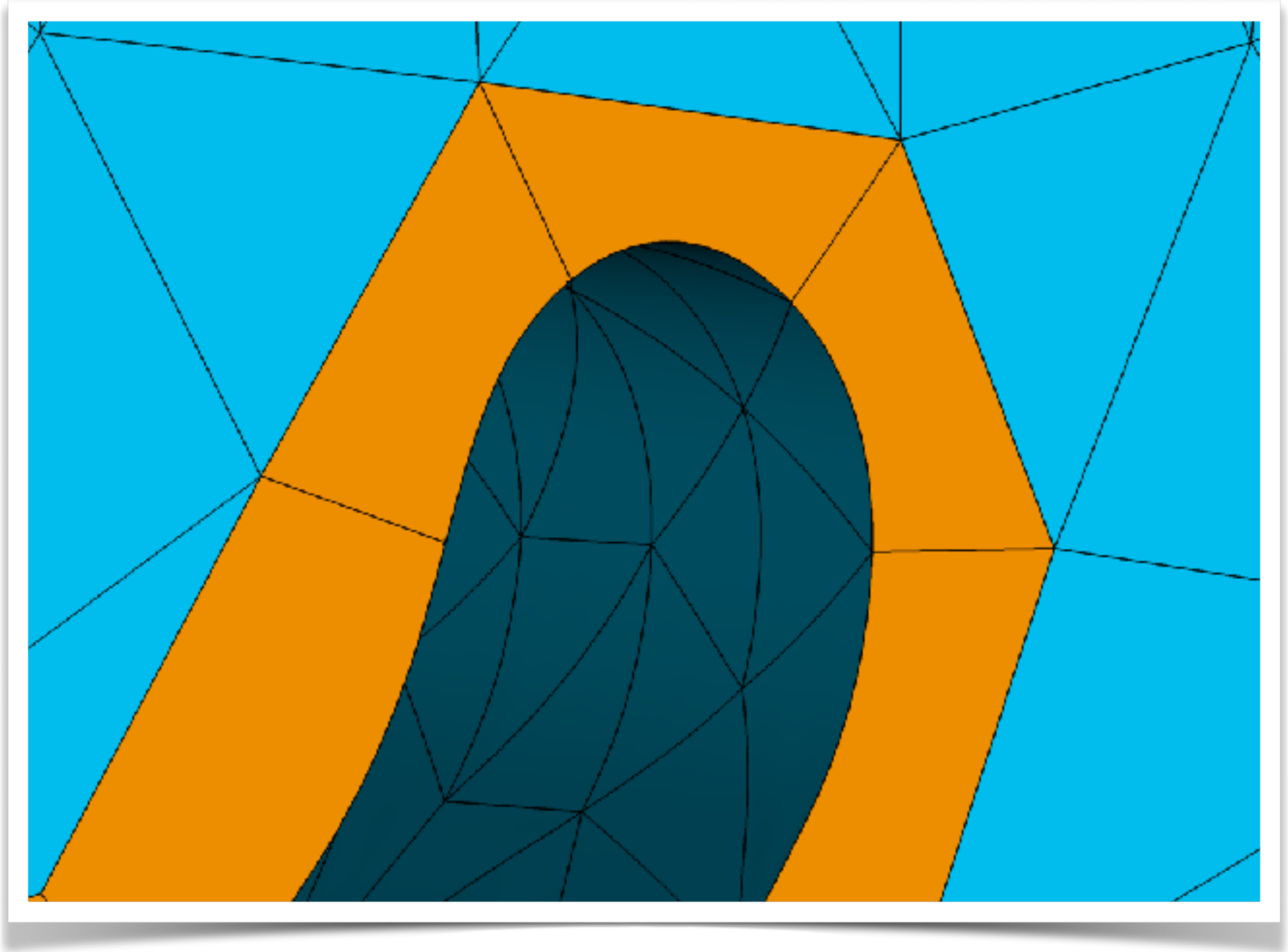
Curving coarse meshes leads to invalid elements
Most existing mesh generation packages cannot deal with this

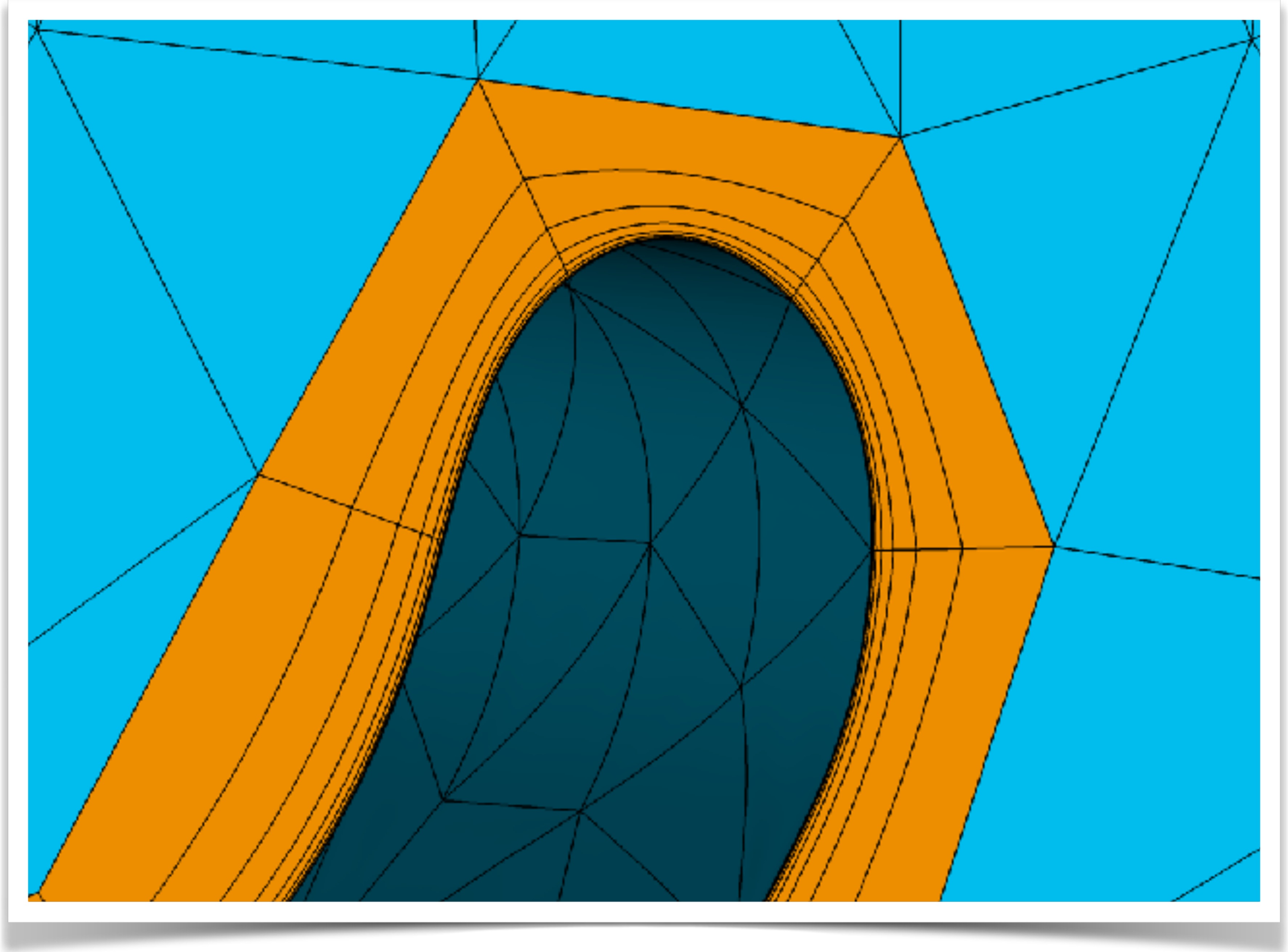




High-order technologies

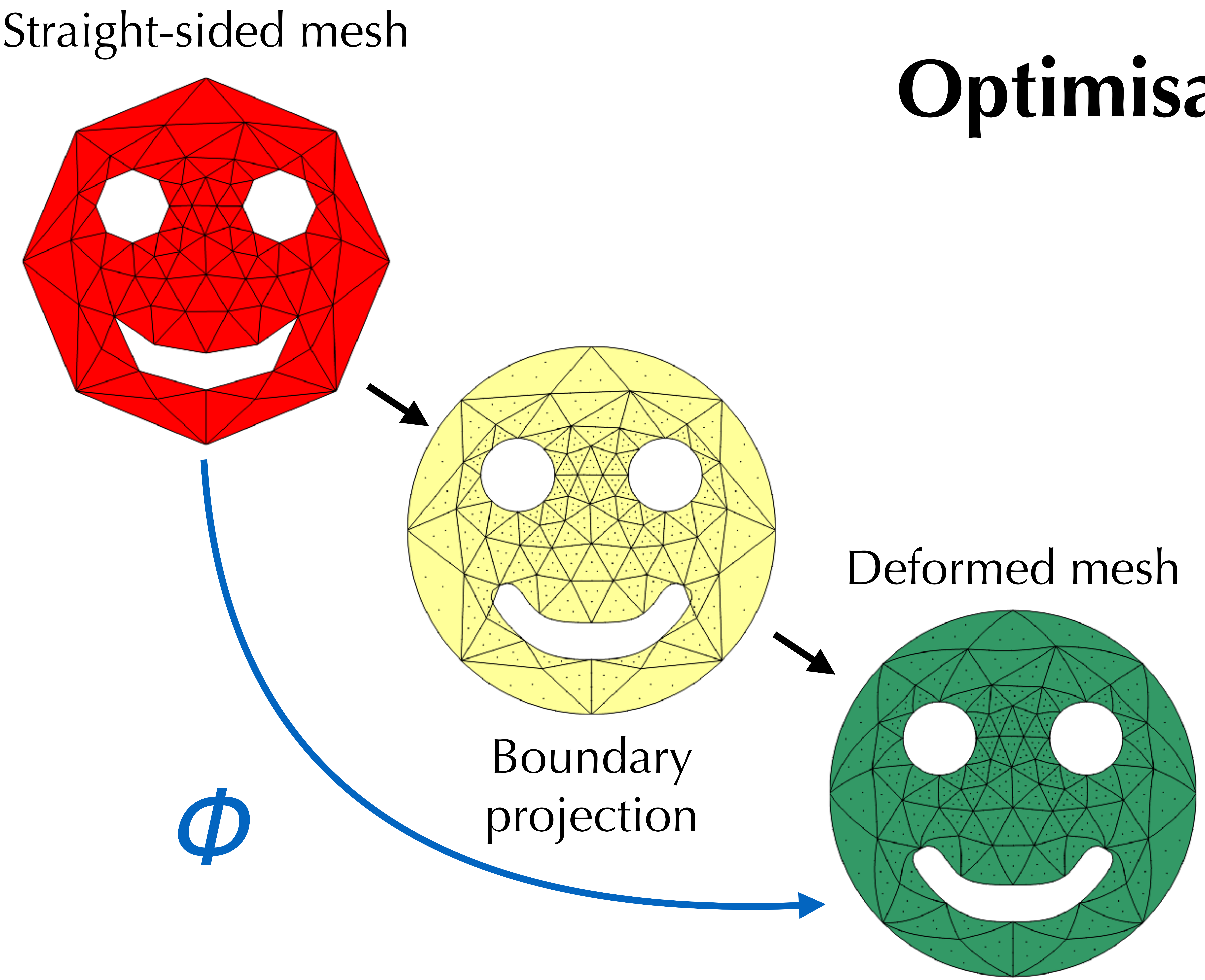
Isoparametric splitting of high-order boundary layers



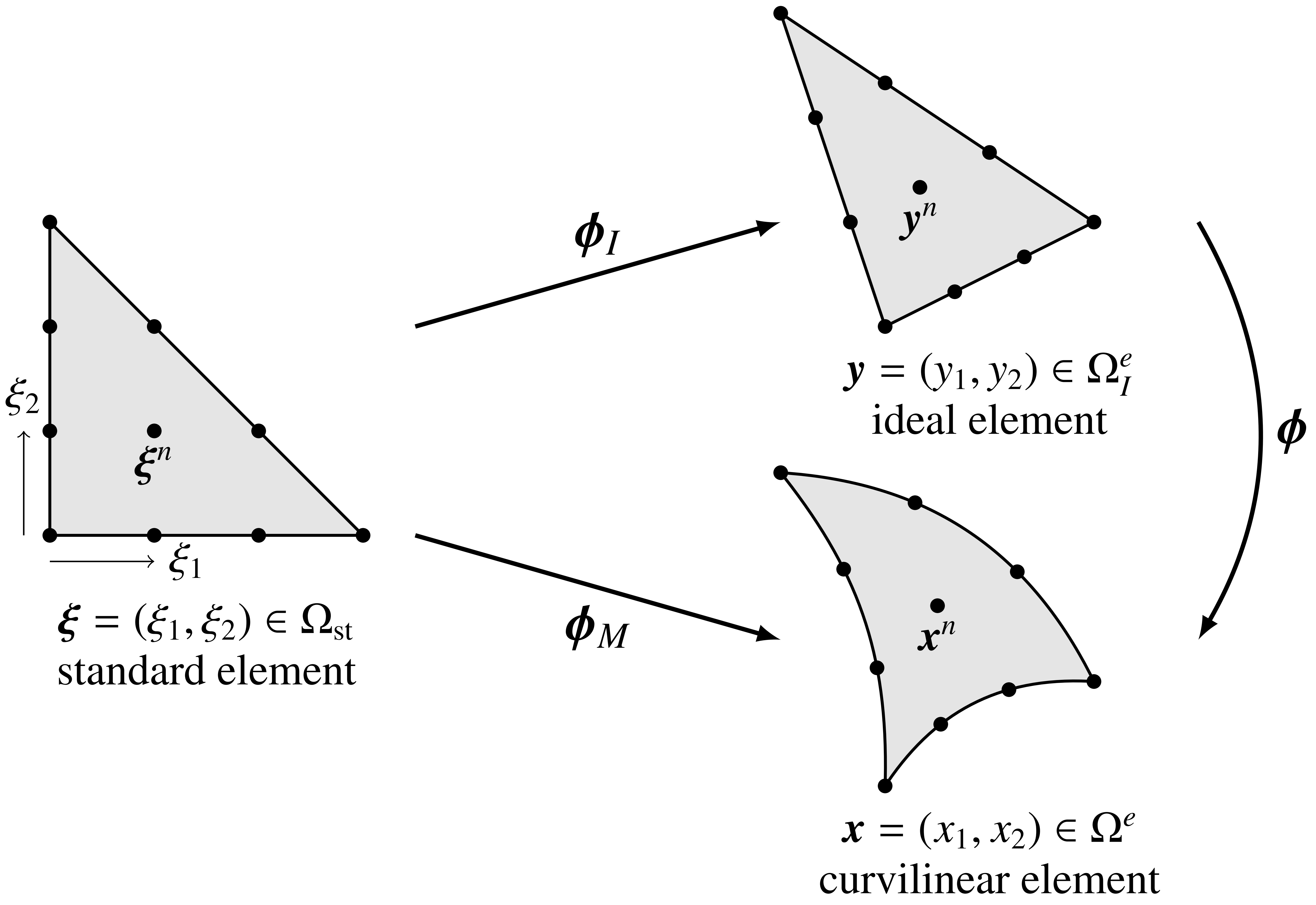


Moxey et al., Comp. Meth. Appl. Mech. Eng **283** pp. 636-650 (2015)

Optimisation



Variational approach



Recast PDE as energy minimisation and solve

$$\min_{\phi} \mathscr{E}(\phi) = \min_{\phi} \int_{\Omega_I} W(\nabla \phi)\, dy$$

Different *W* give PDE and optimisation methods in a single framework

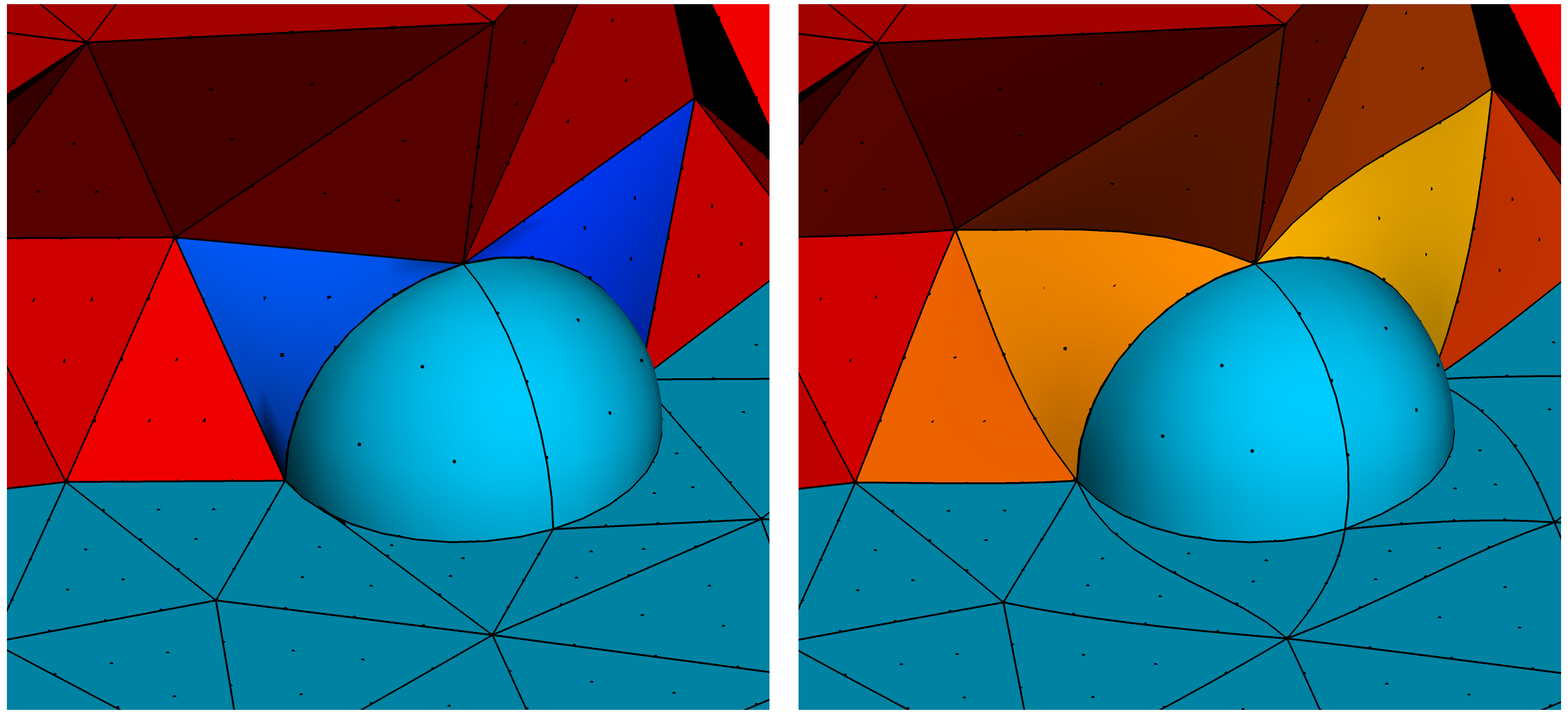
M. Turner, J. Peiró, D. Moxey, *Curvilinear mesh generation using a variational framework*, Computer Aided Design **103** 73-91 (2018)

Choice of functional

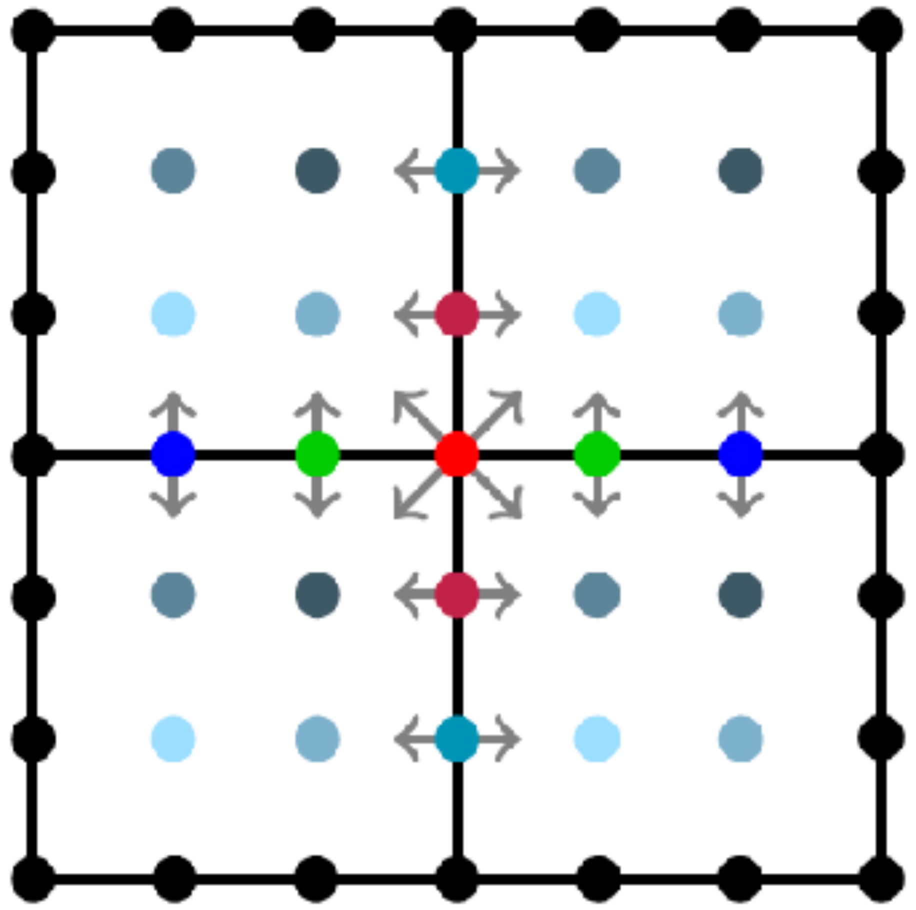
$$\mathsf{F} = \nabla\phi \qquad J = \det \mathsf{F}$$

- **Linear elasticity:** $$W = \frac{\kappa}{2}(\ln J)^2 + \mu\,\mathbf{E} : \mathbf{E}; \quad \mathbf{E} = \frac{1}{2}(\mathbf{F}^t\mathbf{F} - \mathbf{I})$$
- **Non-linear elasticity:** $$W = \frac{\mu}{2}(\mathbf{F} : \mathbf{F} - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2$$
- **Winslow:** $$W = J^{-1}\,(\mathbf{F} : \mathbf{F})$$
- **Distortion:** $$W = \frac{1}{d}|J|^{-d/2}(\mathbf{F} : \mathbf{F})$$

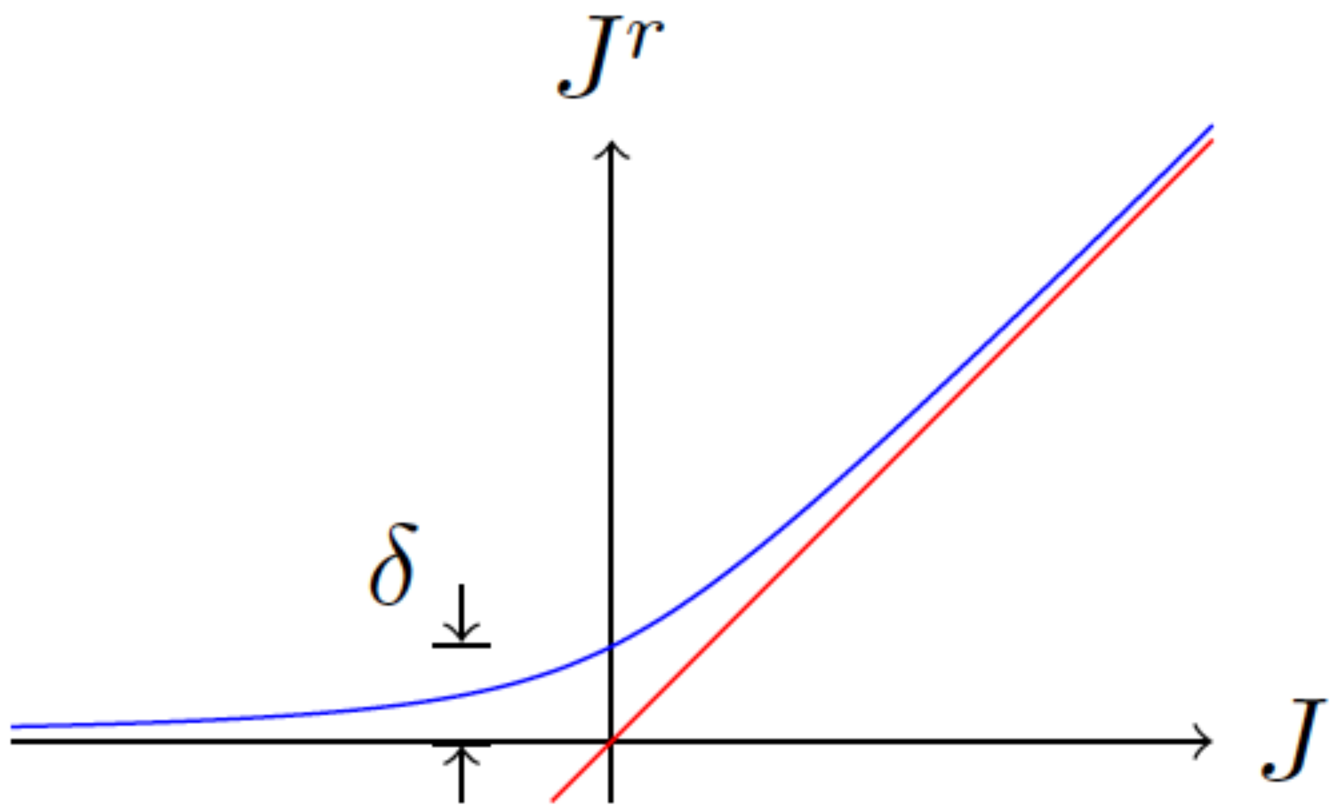
Benefits



CAD sliding

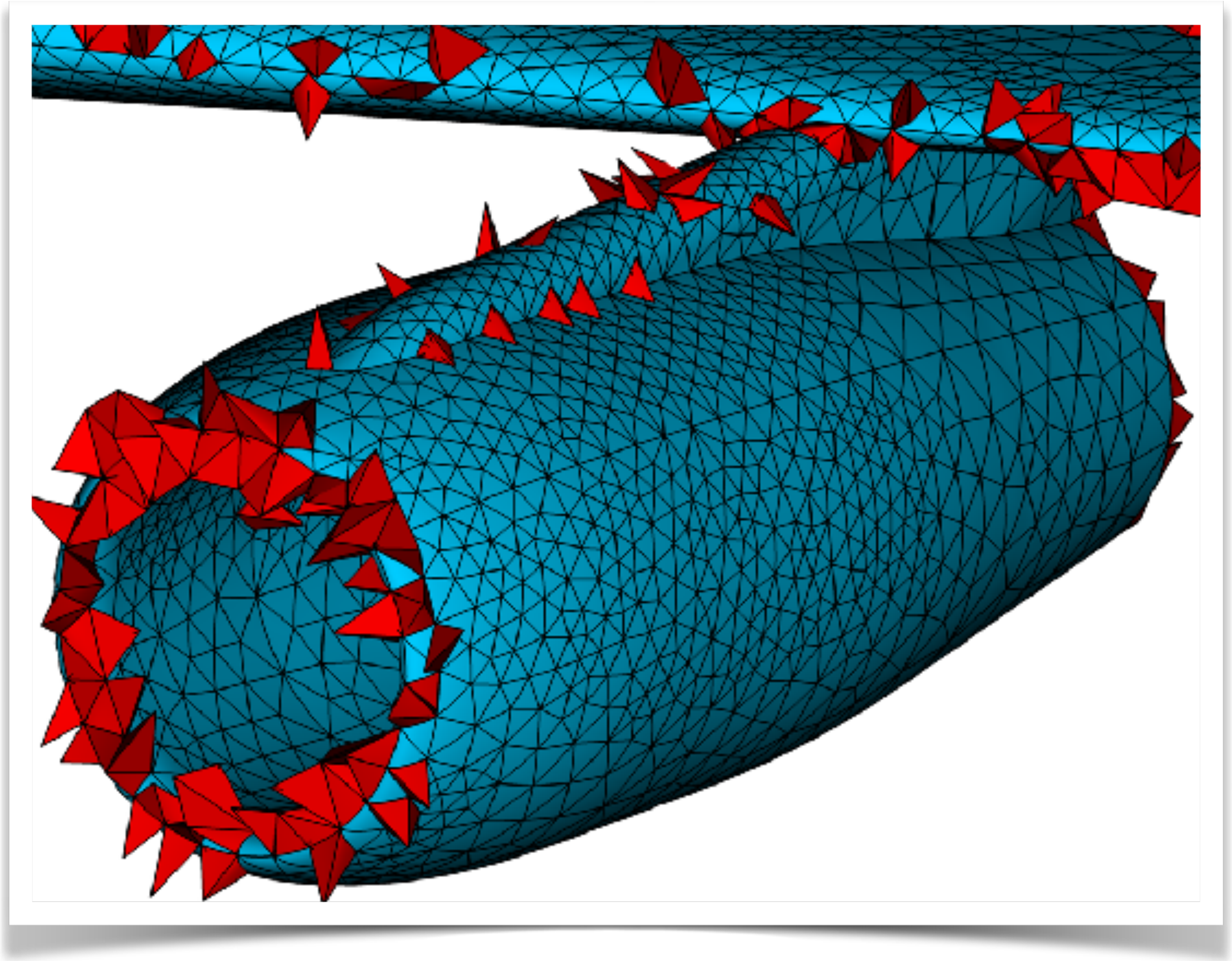


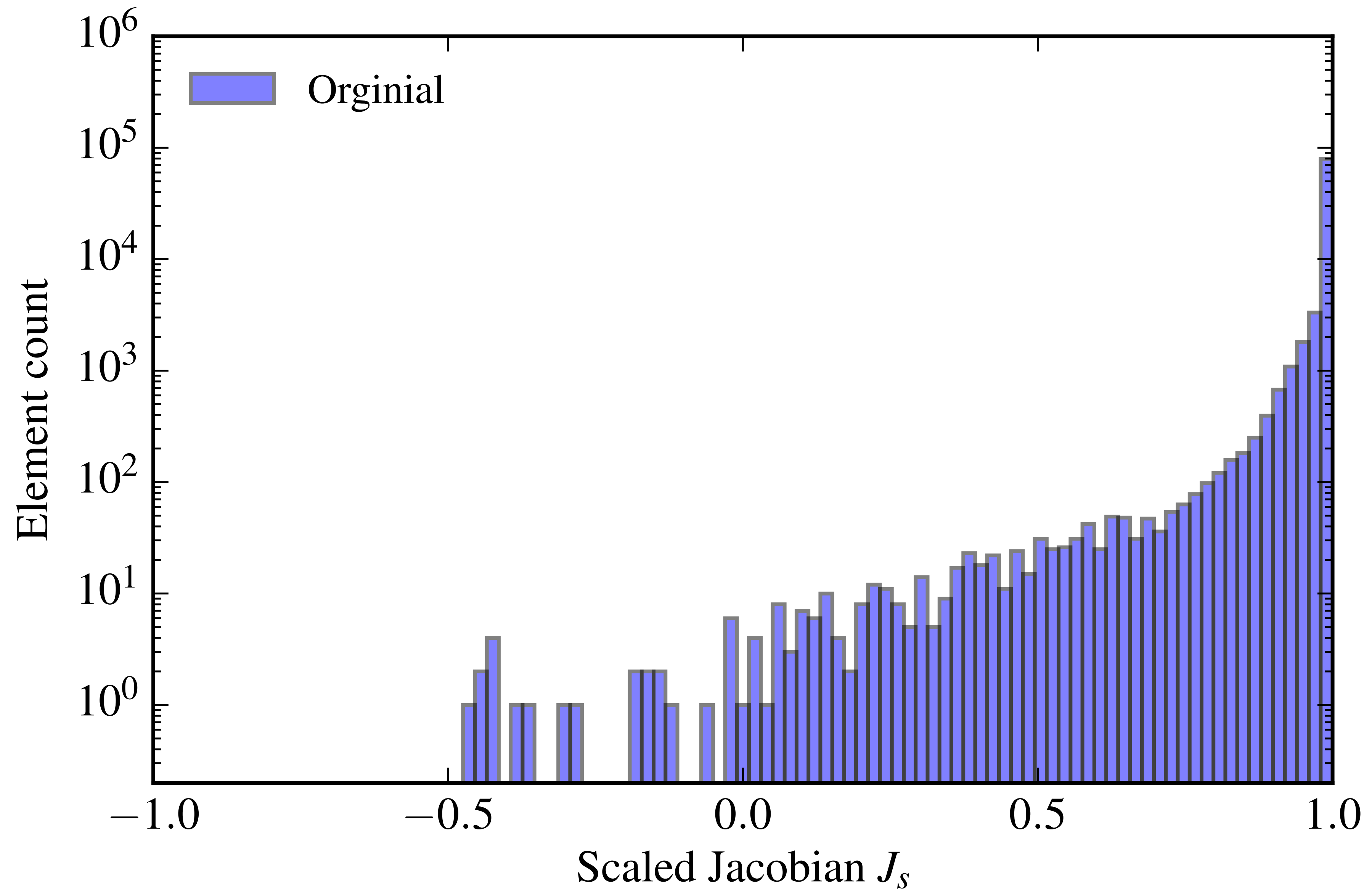
Multi-core parallelisation
relaxation optimisation approach

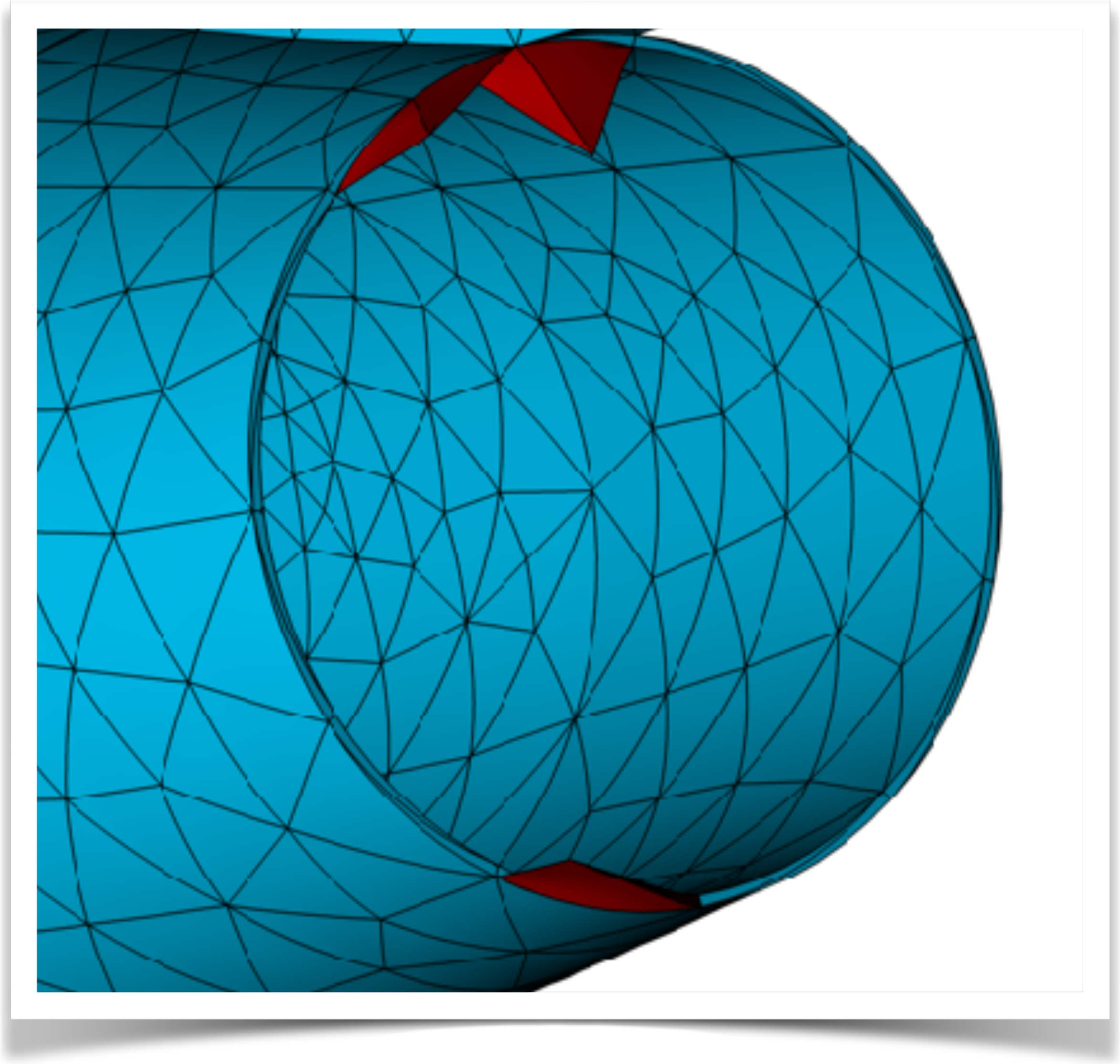


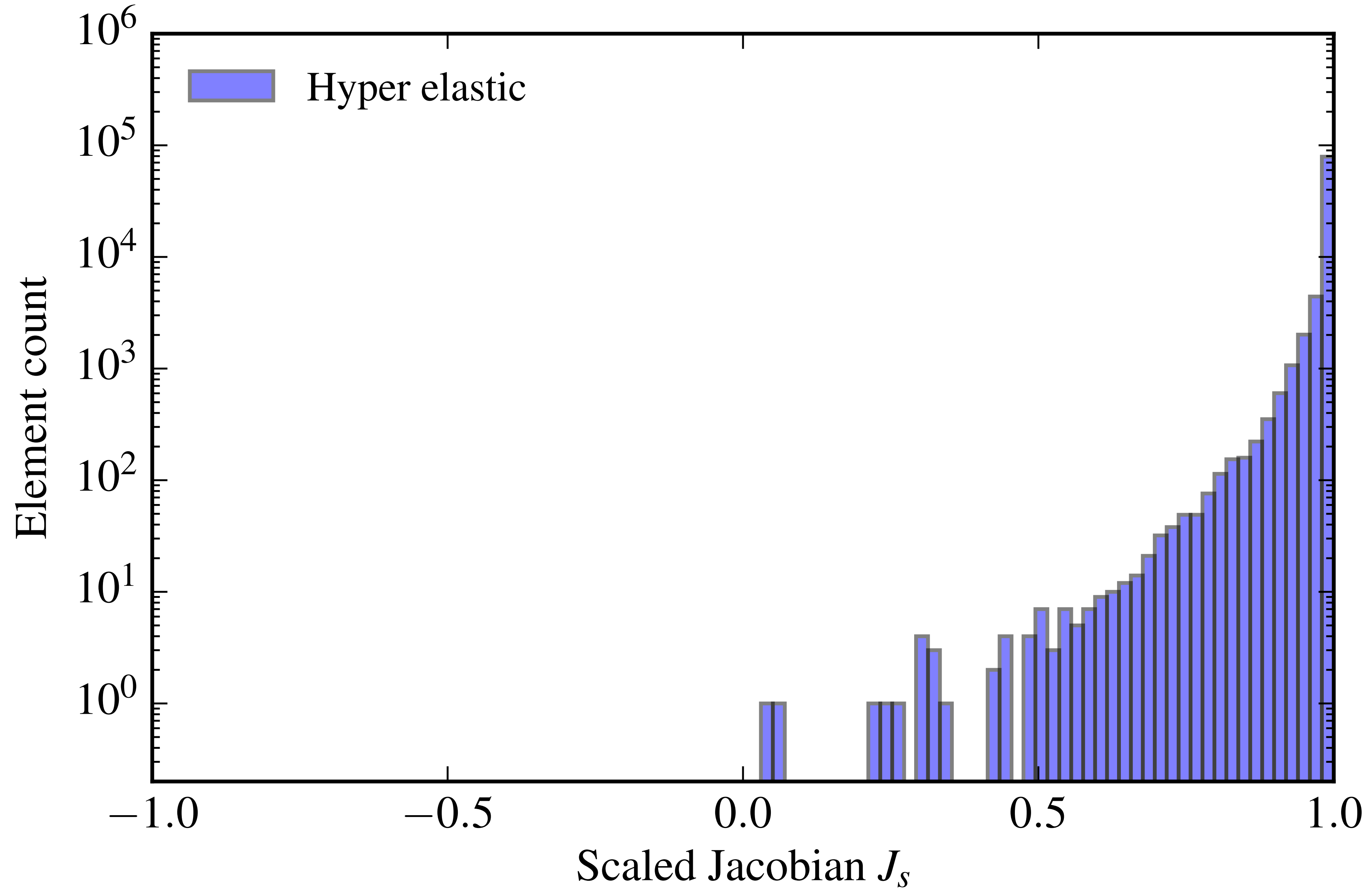
Untangles meshes
using Jacobian regularisation

Example: DLR F6 engine





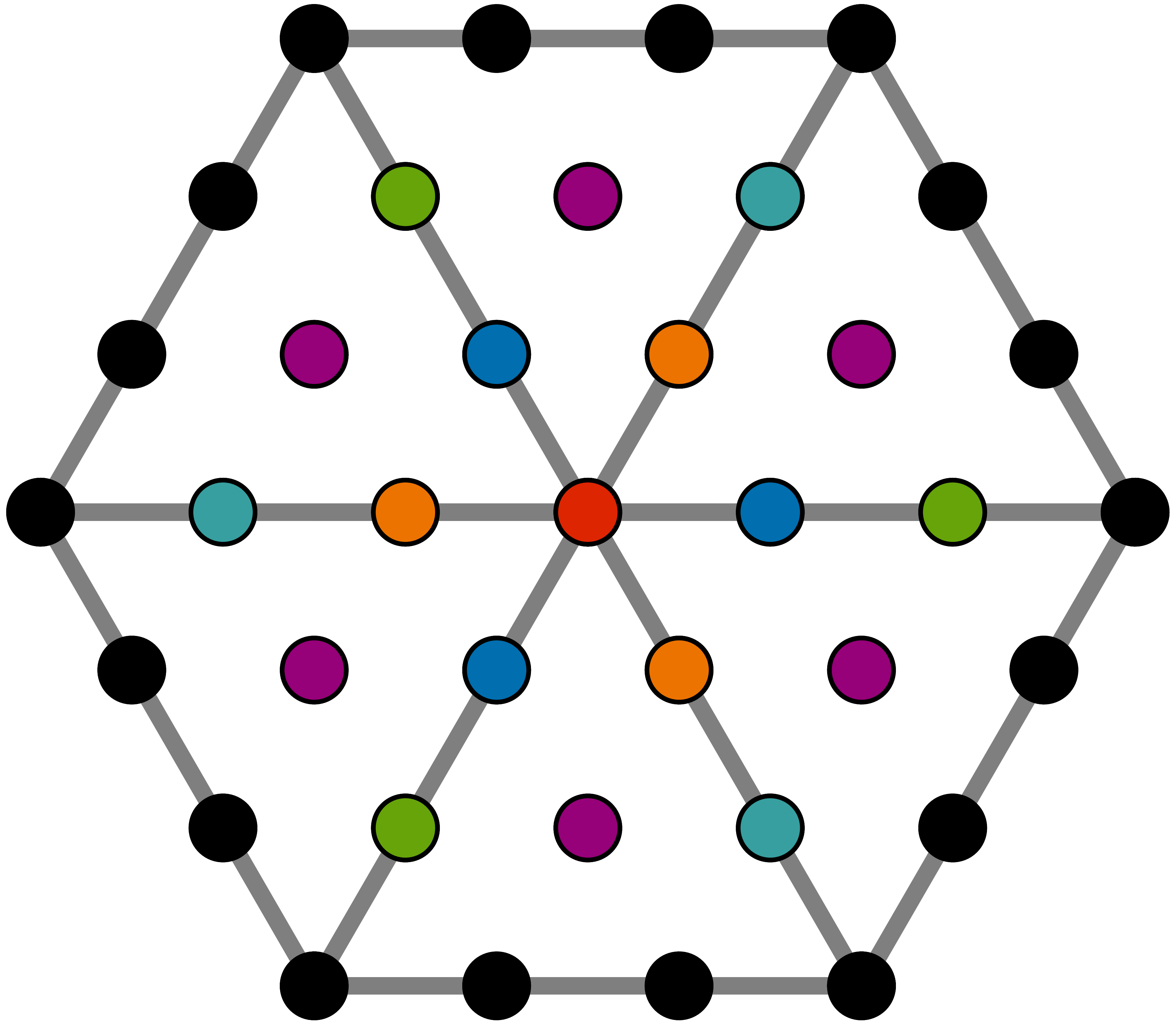




Speeding up optimisation

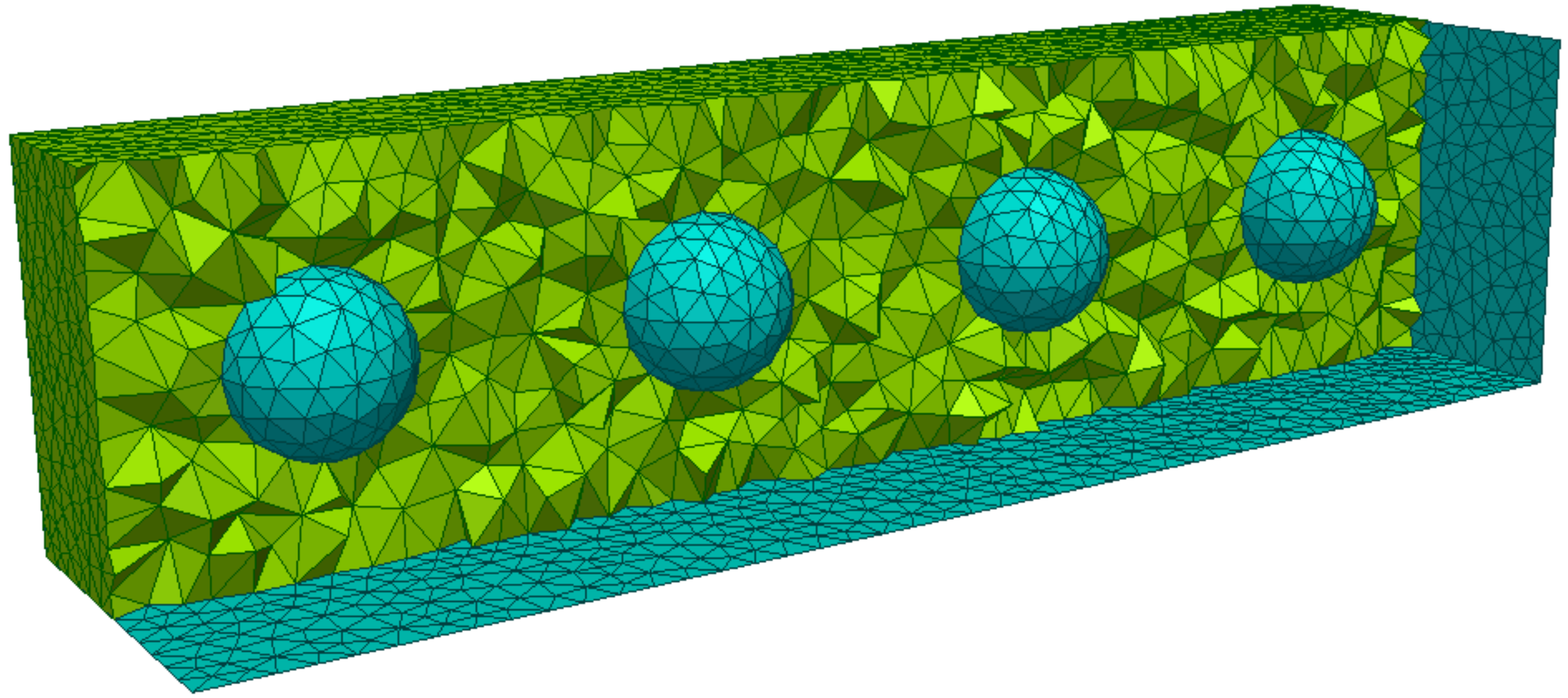
- Meshing usually accomplished on a single workstation, generally repeated as part of many design iterations.
- Optimisation process is resource intensive, but GPUs have lots of compute density.
- Can we leverage parallelism of the method effectively on a GPU?
- How do we do this in a code-friendly way?

Node colouring

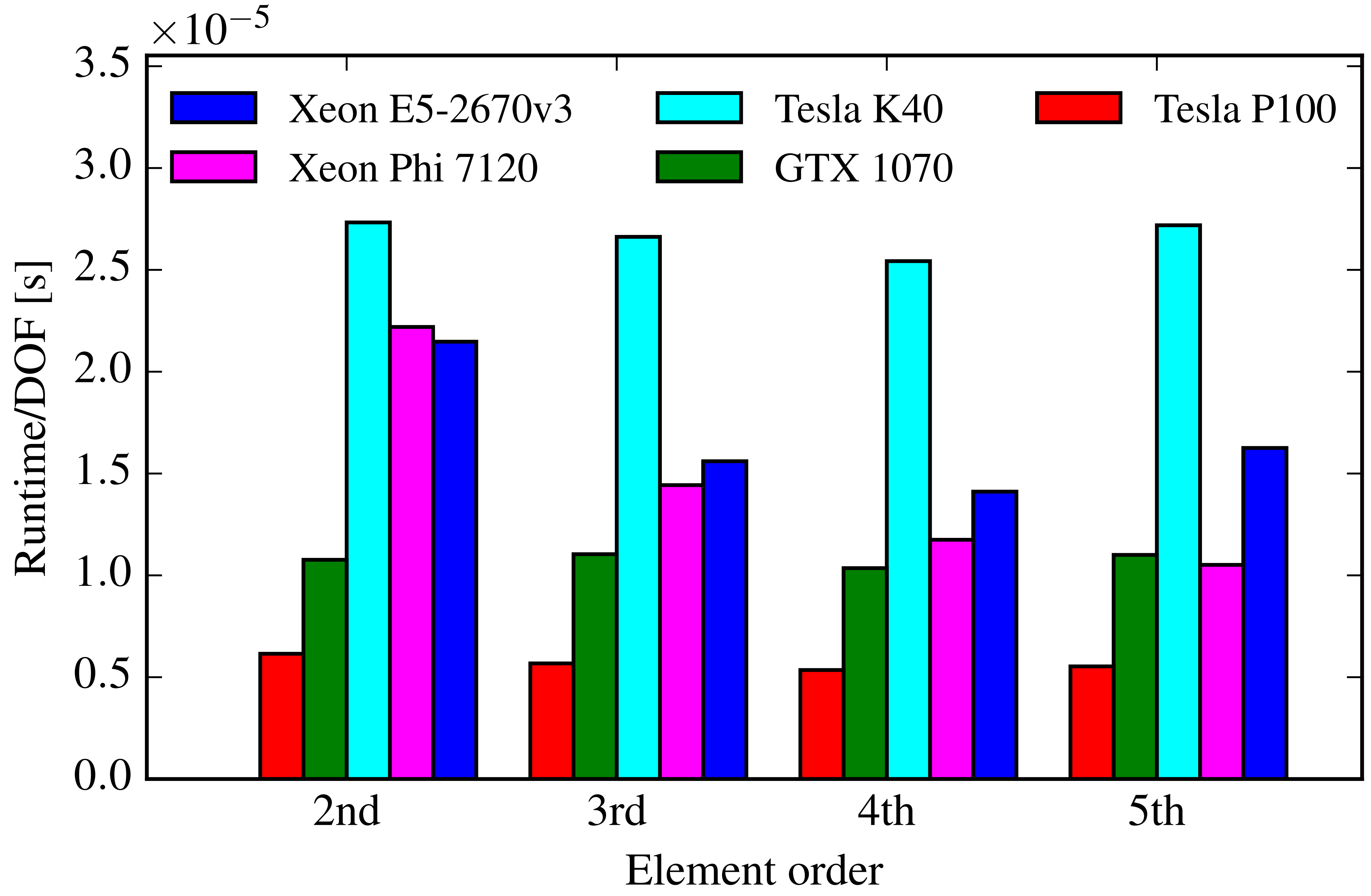


- For each node solve local minimisation problem
- Calculate functional + gradients analytically
- Uses multi-level threading to exploit GPU hierarchy: use Kokkos
- Iterate until global functional residual is small

Results



Four spheres in a box, 33k tetrahedra, ~400k nodes at $p = 5$

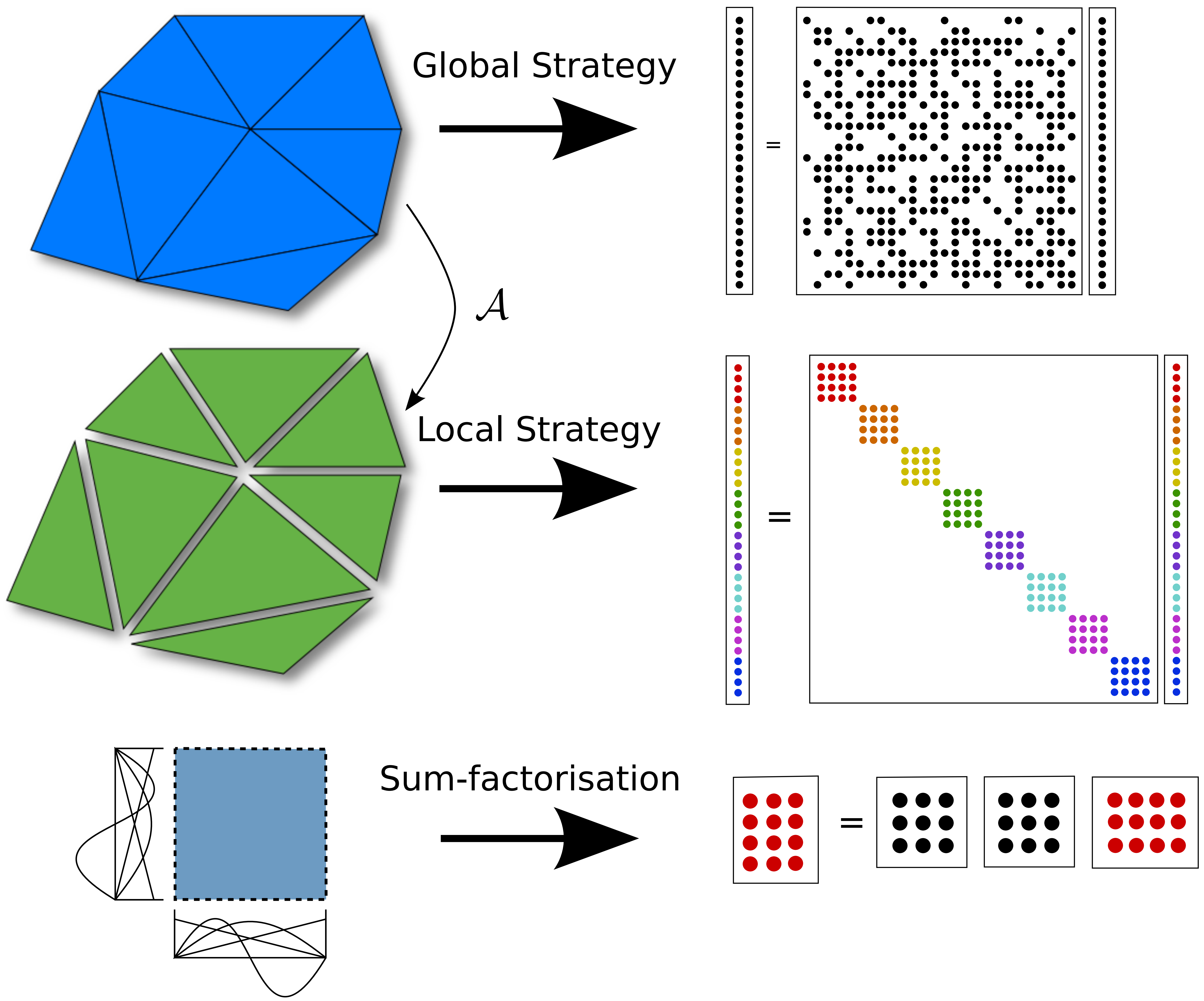


Reasonably consistent runtimes per DoF across polynomial orders

Challenge 2: efficient implementation

- Today's computational hardware: lots of FLOPS available, but really hard to use them.
- Algorithms will only use hardware effectively if they are **arithmetically intense:** i.e. high ratio of FLOPS per byte of memory transfer.
- One of the reasons that current CFD codes don't often make best use of hardware on offer.
- High-order has potential in this area through **matrix-free formulations.**

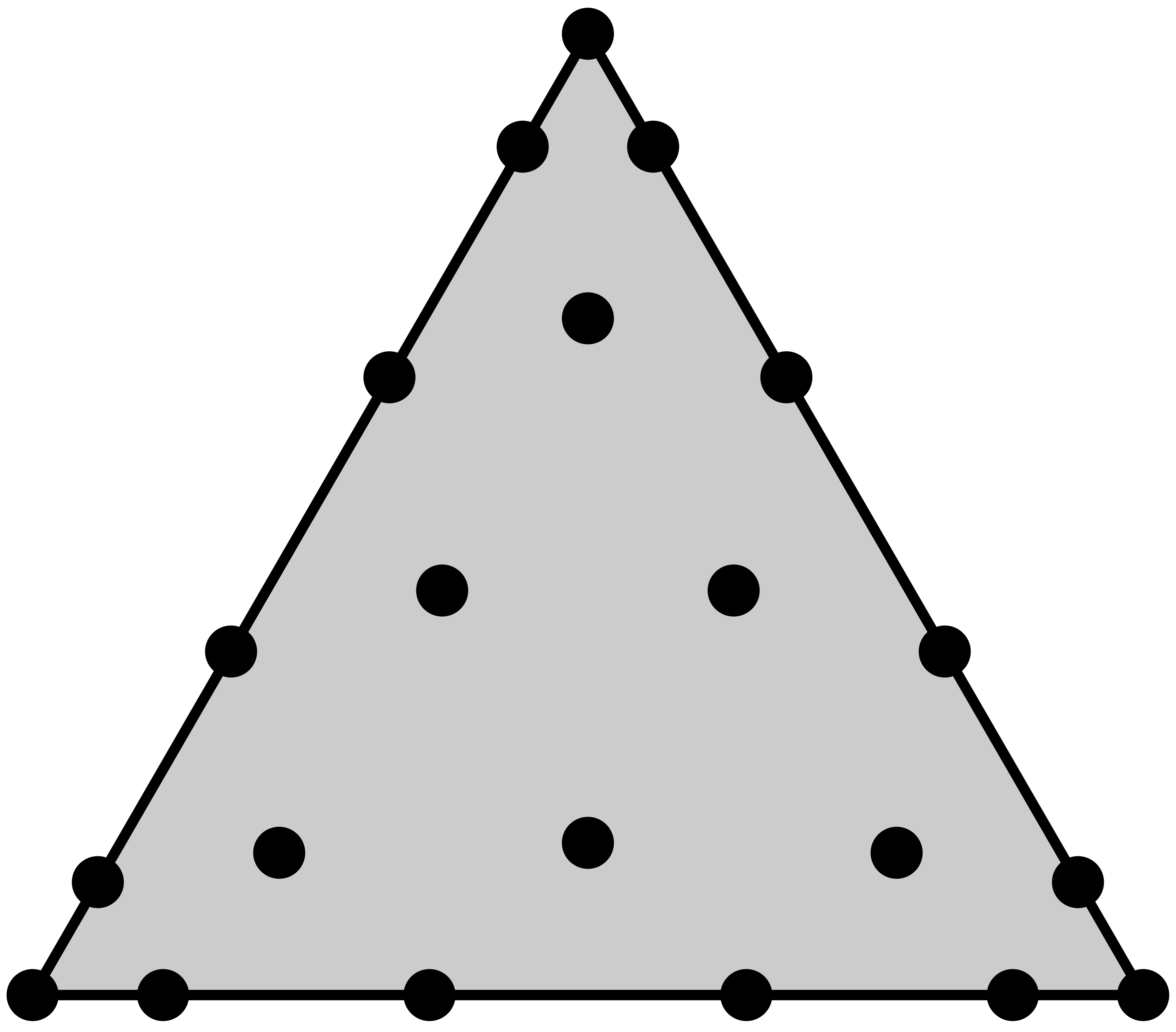
Implementation choices



Increasing polynomial order

More localised memory access

Unstructured elements



P5 triangle, Fekete points

- Typically unstructured elements make use of Lagrange basis functions (although not always).
- Combine this with a suitable set of quadrature (cubature) points: no tensor-products structure.
- However, spectral/*hp* does have a tensor product structure!

Sum-factorisation

Key to performance at high polynomial orders: complexity $O(P^{2d})$ to $O(P^{d+1})$!

$$\sum_{p=0}^{P}\sum_{q=0}^{Q}\hat{u}_{pq}\phi_p(\xi_{1i})\phi_q(\xi_{2j}) = \sum_{p=0}^{P}\phi_p(\xi_{1i})\left[\sum_{q=0}^{Q}\hat{u}_{pq}\phi_q(\xi_{2j})\right]$$

store this

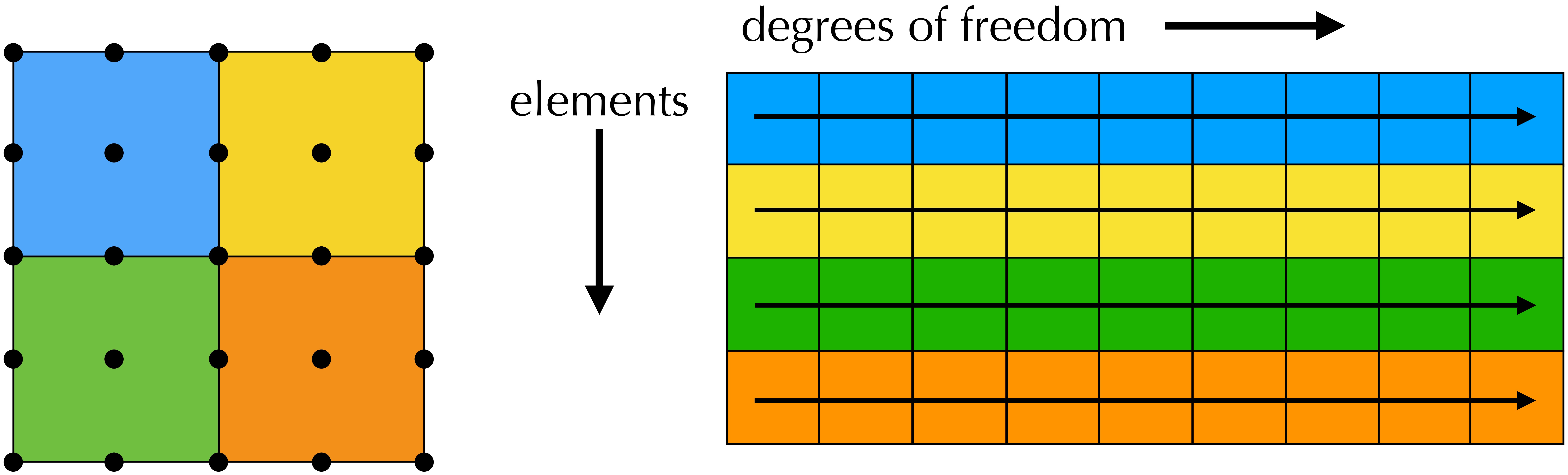
This works in essentially the same way for more complex indexing:

$$\sum_{p=0}^{P}\sum_{q=0}^{Q-p}\hat{u}_{pq}\phi_p^a(\xi_{1i})\phi_{pq}^b(\xi_{2j}) = \sum_{p=0}^{P}\phi_p^a(\xi_{1i})\left[\sum_{q=0}^{Q-p}\hat{u}_{pq}\phi_{pq}^b(\xi_{2j})\right]$$

store this

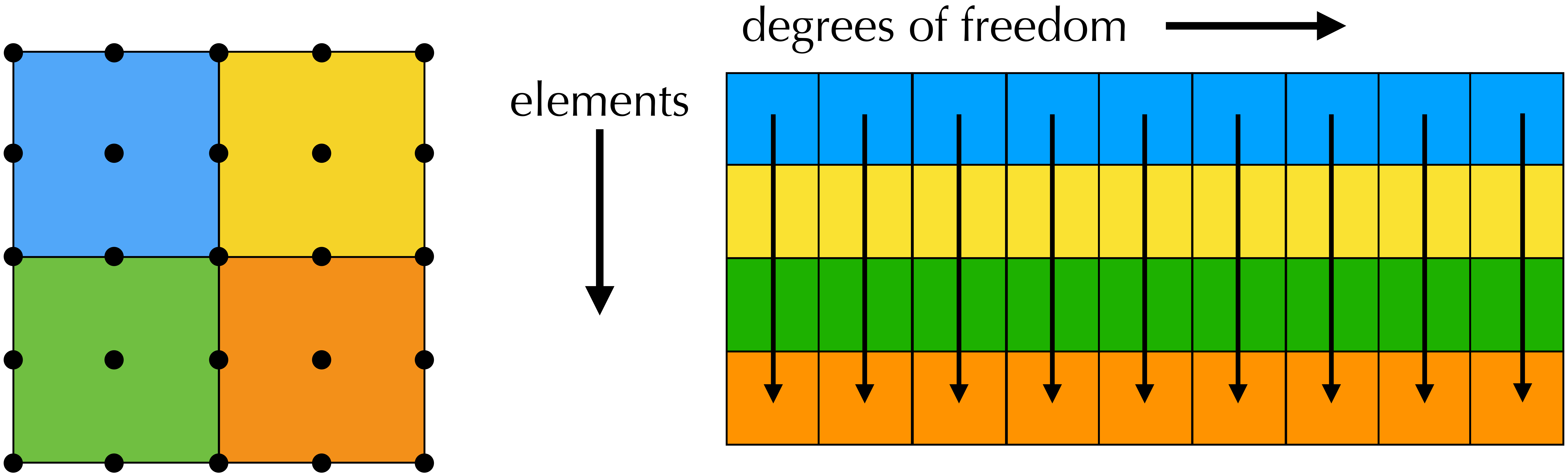
Data layout

Natural to consider data laid out element by element



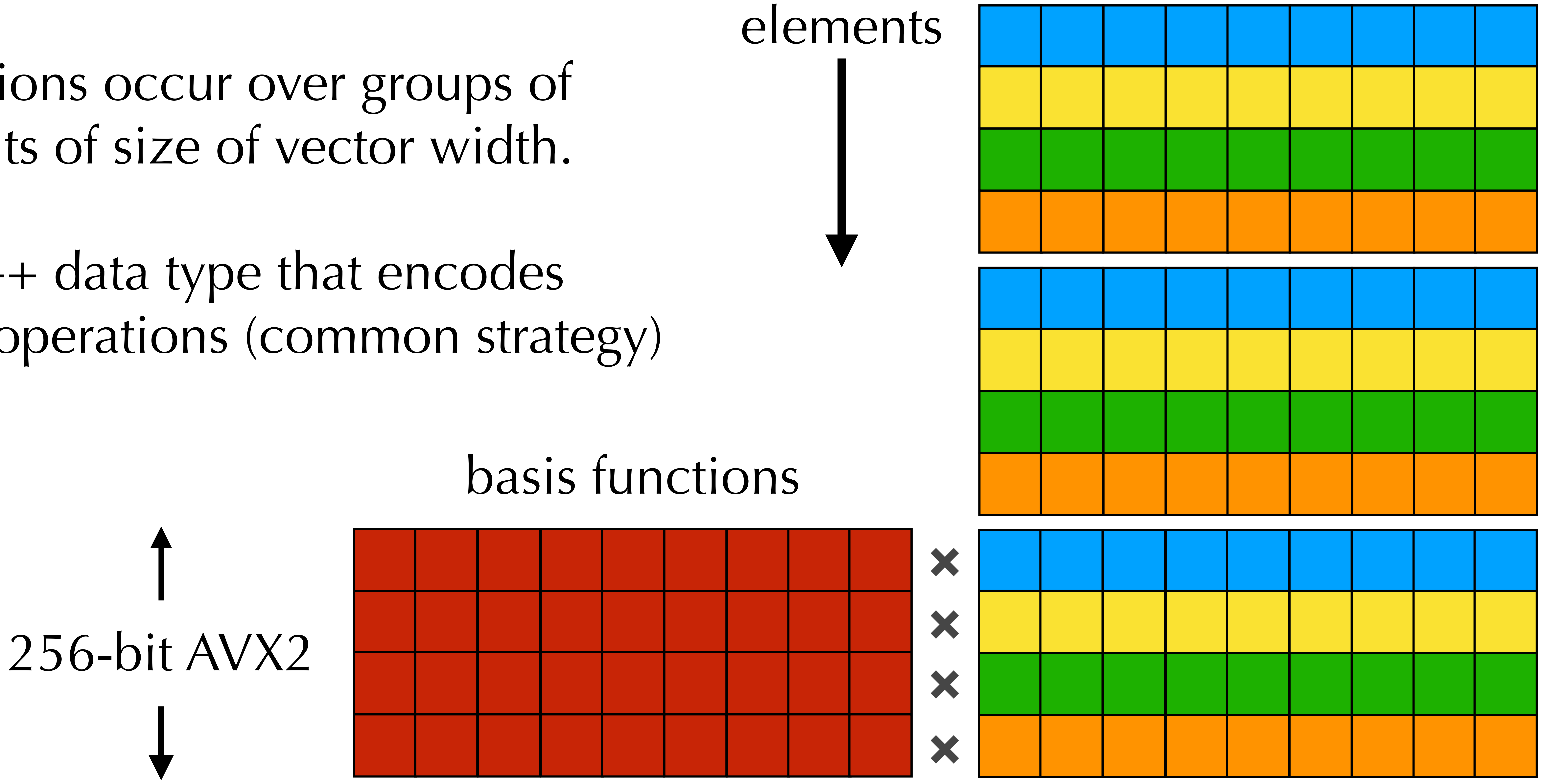
Data layout

Exploit vectorisation by grouping DoFs by vector width

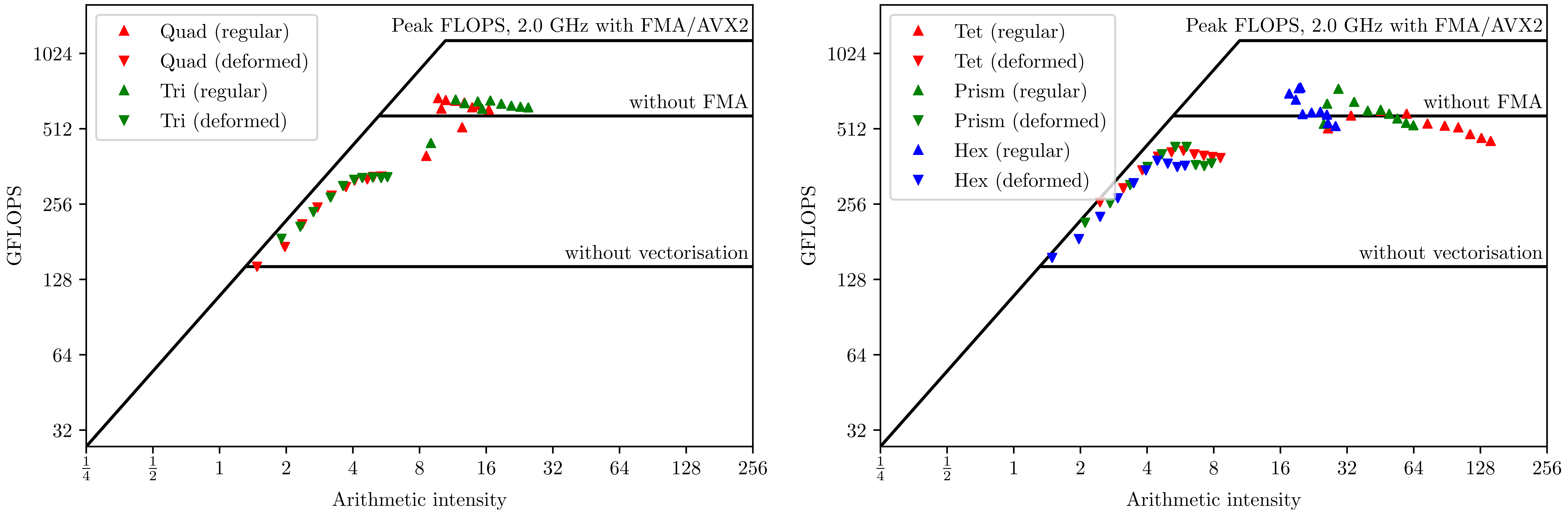


Data layout

- Operations occur over groups of elements of size of vector width.
- Use C++ data type that encodes vector operations (common strategy)



Roofline results



2D: Quads, triangles

3D: Hexahedra, prisms, tetrahedra

Use of ~50-70% peak FLOPS for regular elements

Challenge 3: implementation effort

- High-order methods have potential to bring some nice numerical and computational benefits to bear on complex problems.
- Offer high(er) fidelity at equivalent or lower costs, as they have good implementation characteristics.
- However, one of the main barriers to using high-order methods is that they are **difficult to implement**.

Nektar++

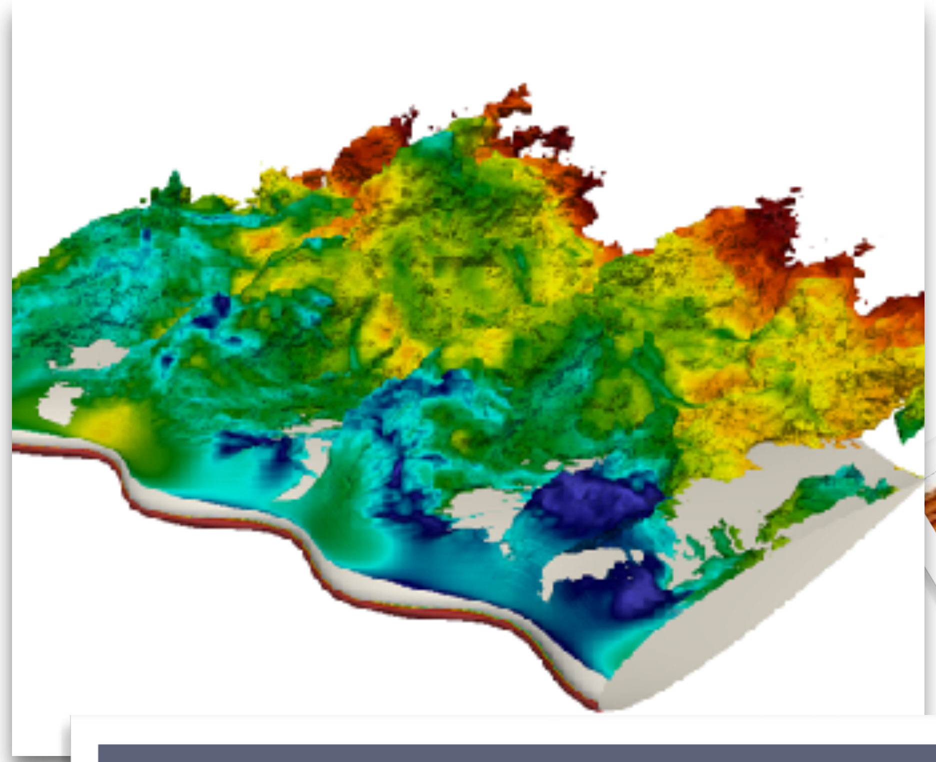
spectral/hp element framework

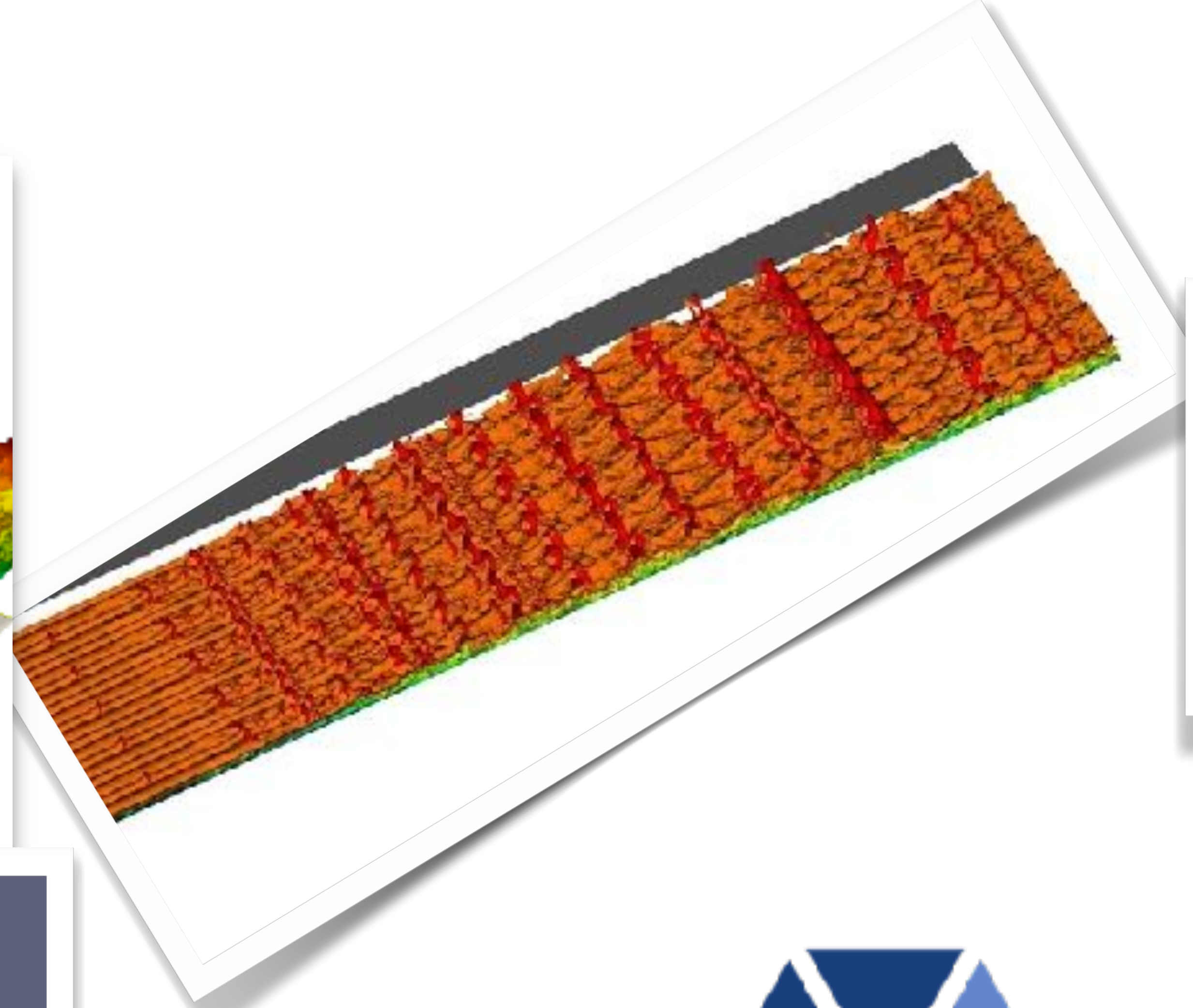
Nektar++

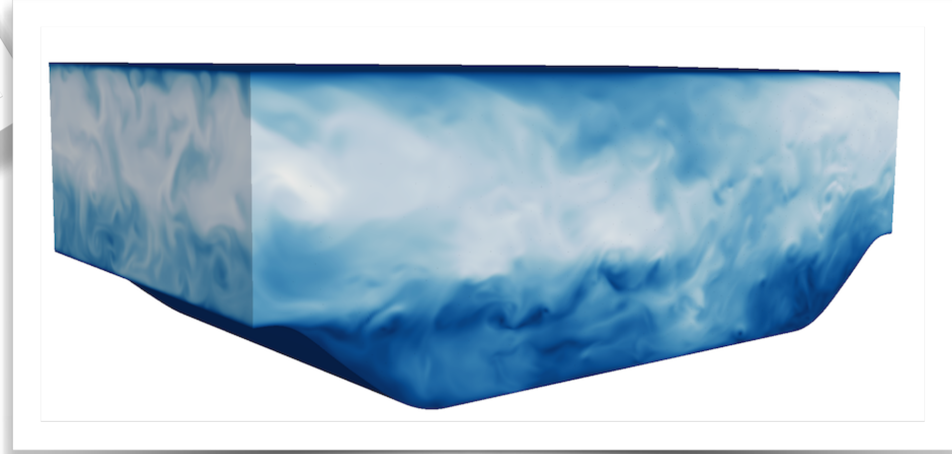
spectral/hp element framework

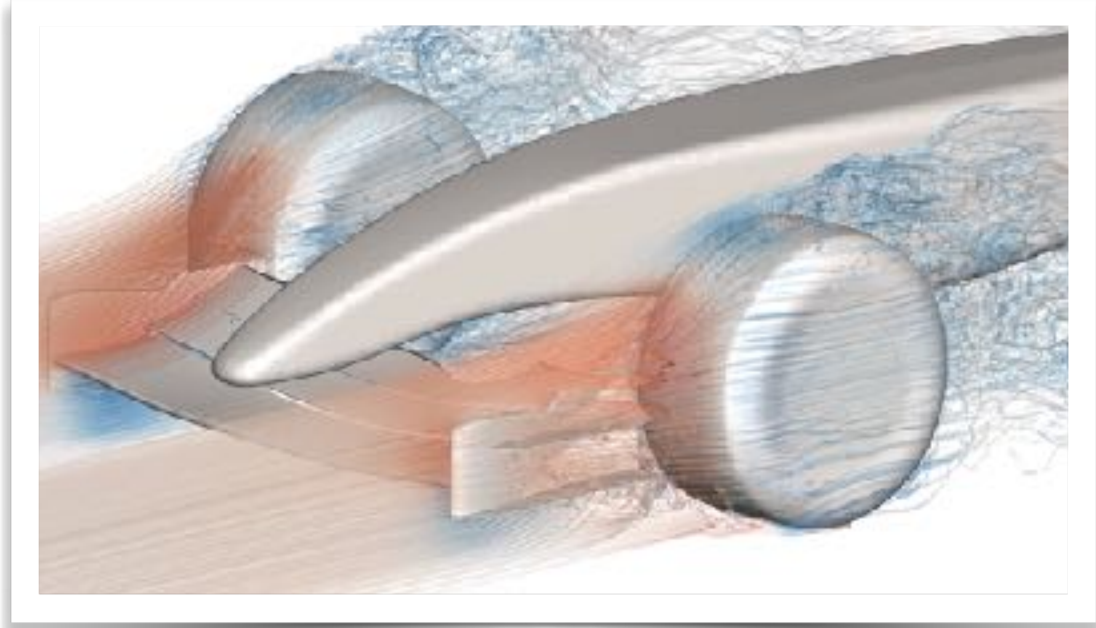
- Nektar++ is an **open source framework** for high-order methods.
- Although fluids is a key application area, we try to make it easier to use these methods in many areas, **not just fluids**.
- C++ API, with ambitions to bridge current and future hardware diversity (e.g. many-core processors, GPUs).
- Modern development practices with continuous integration, git, etc.

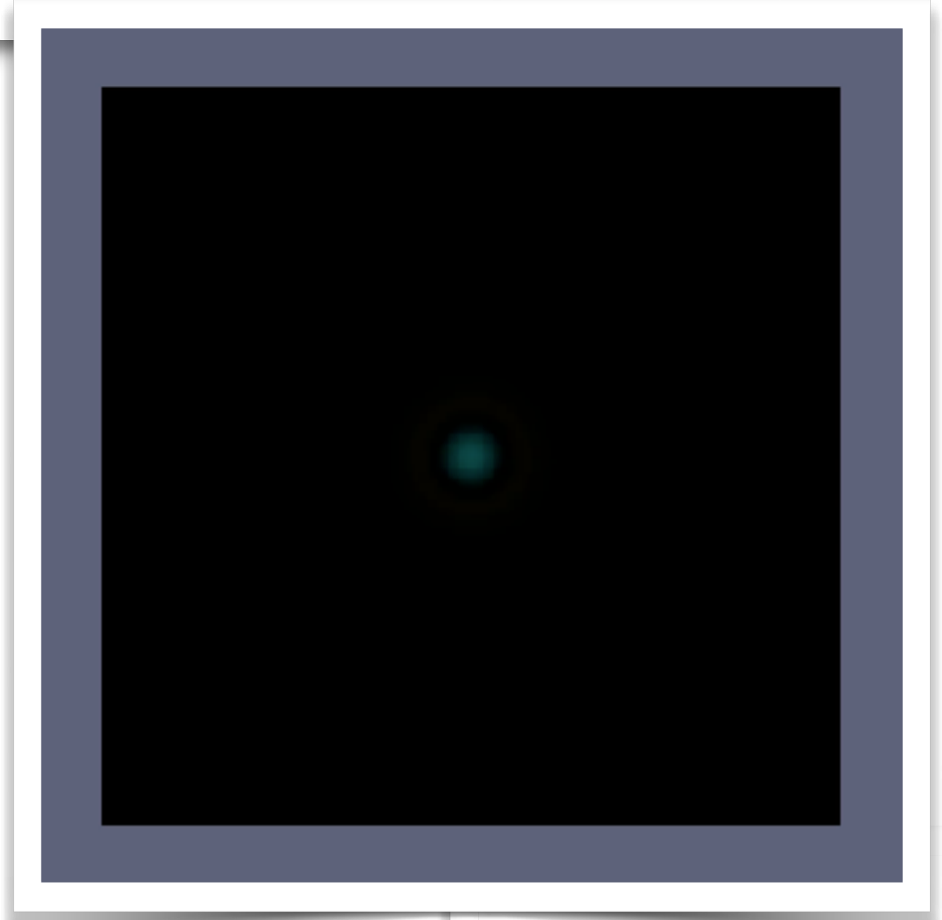
Some application areas



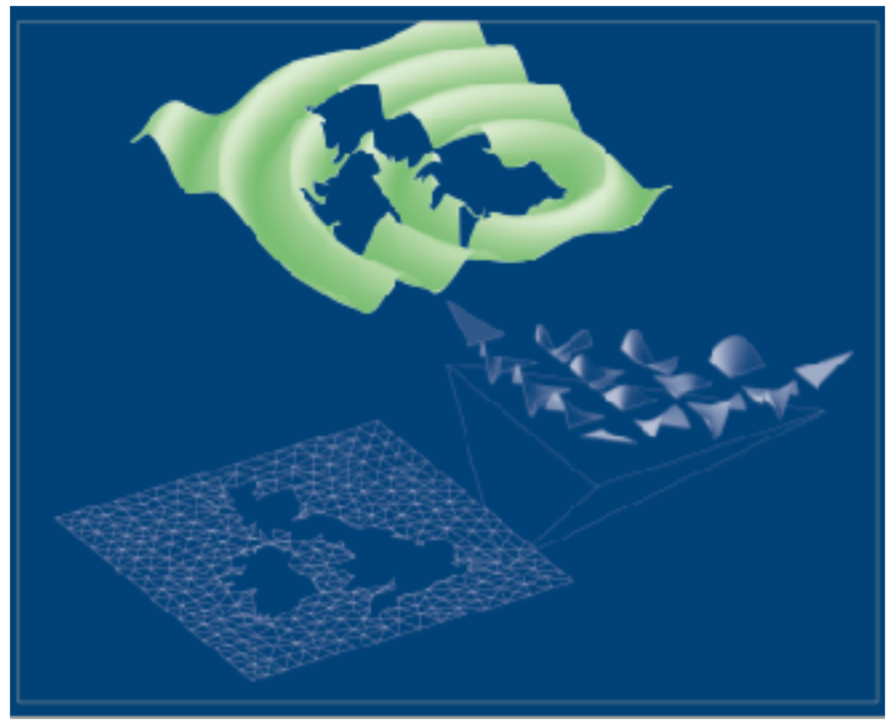


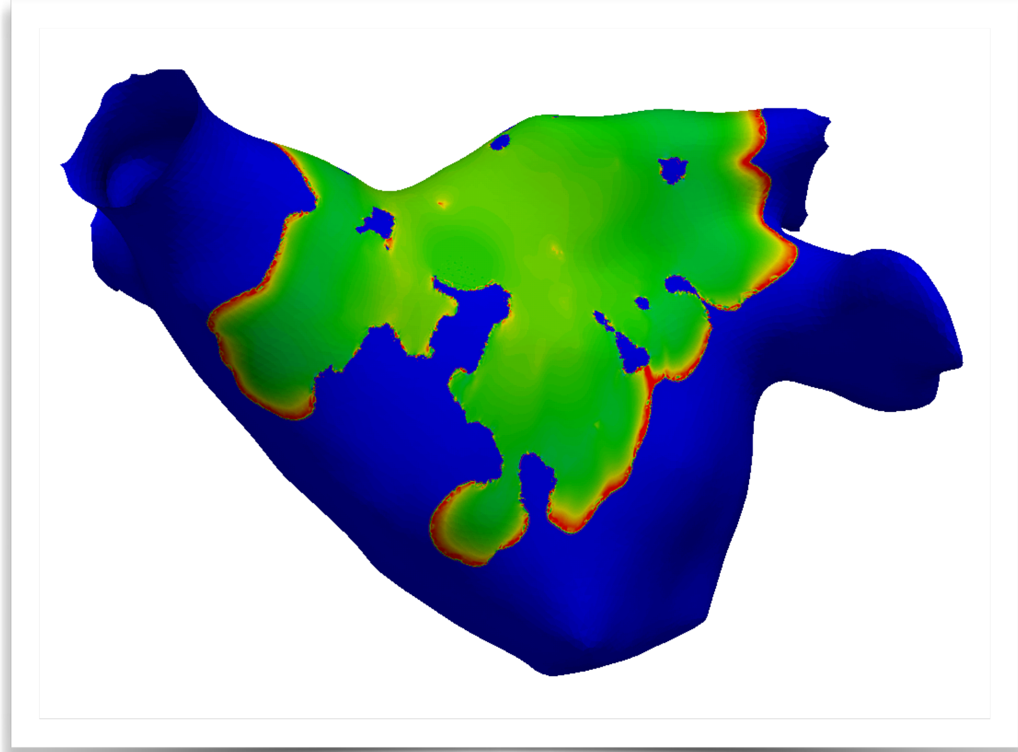


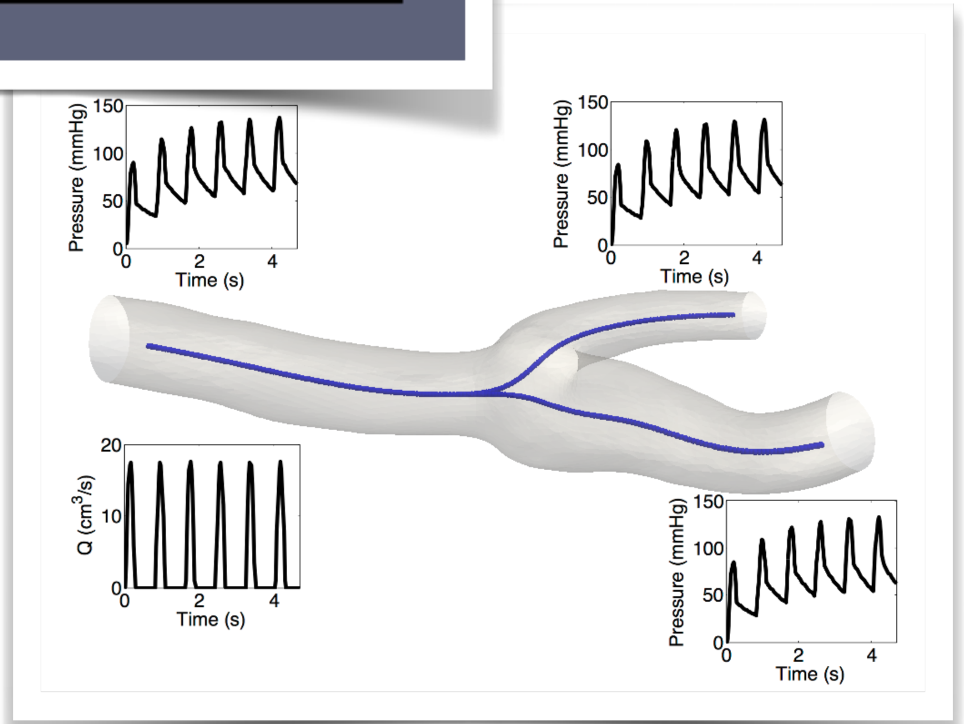


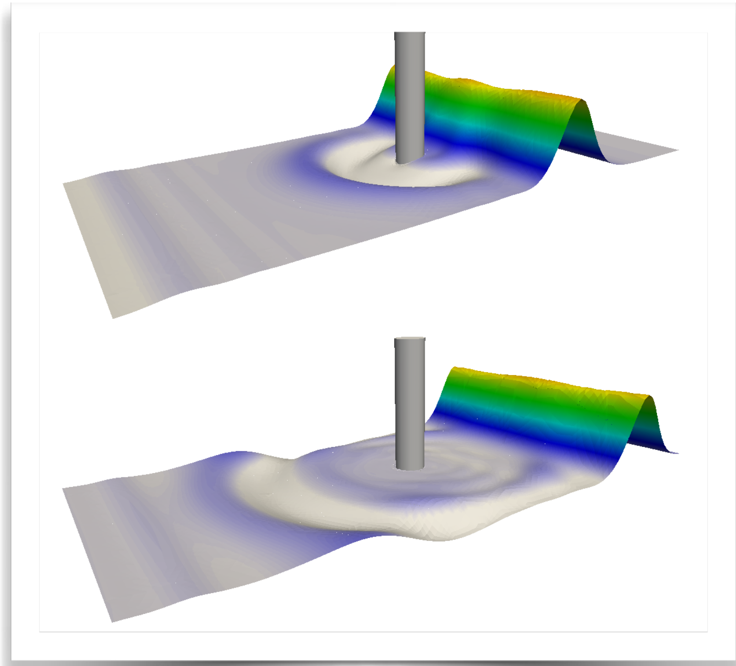


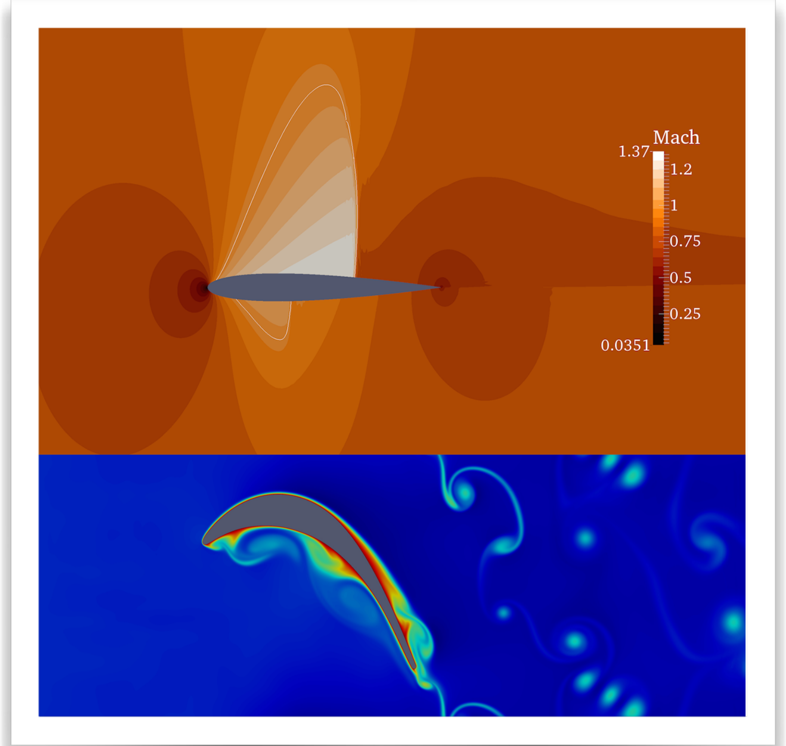
www.nektar.info

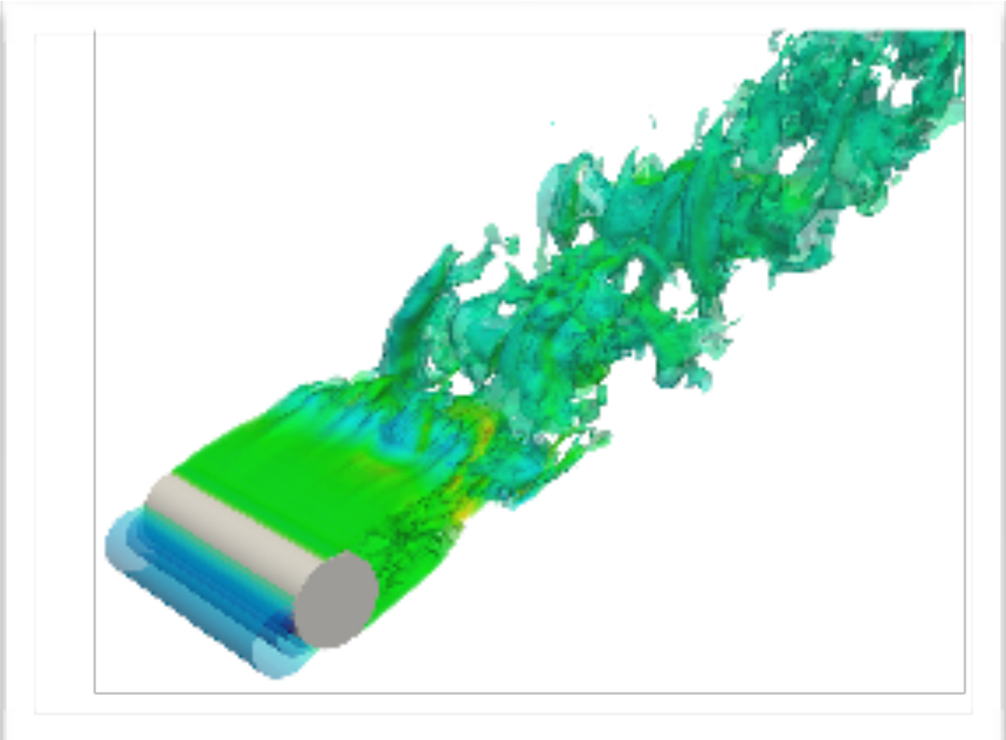




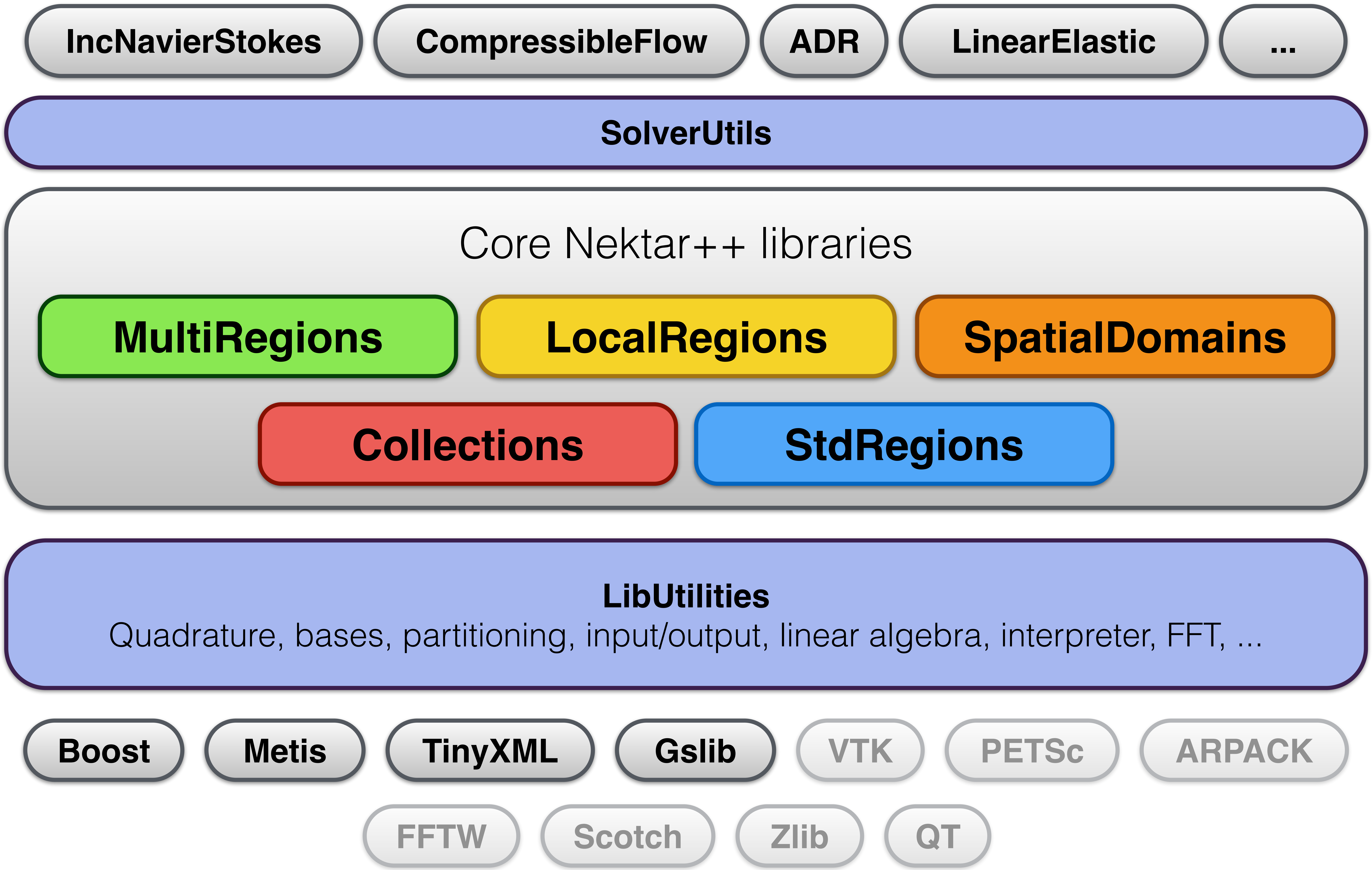




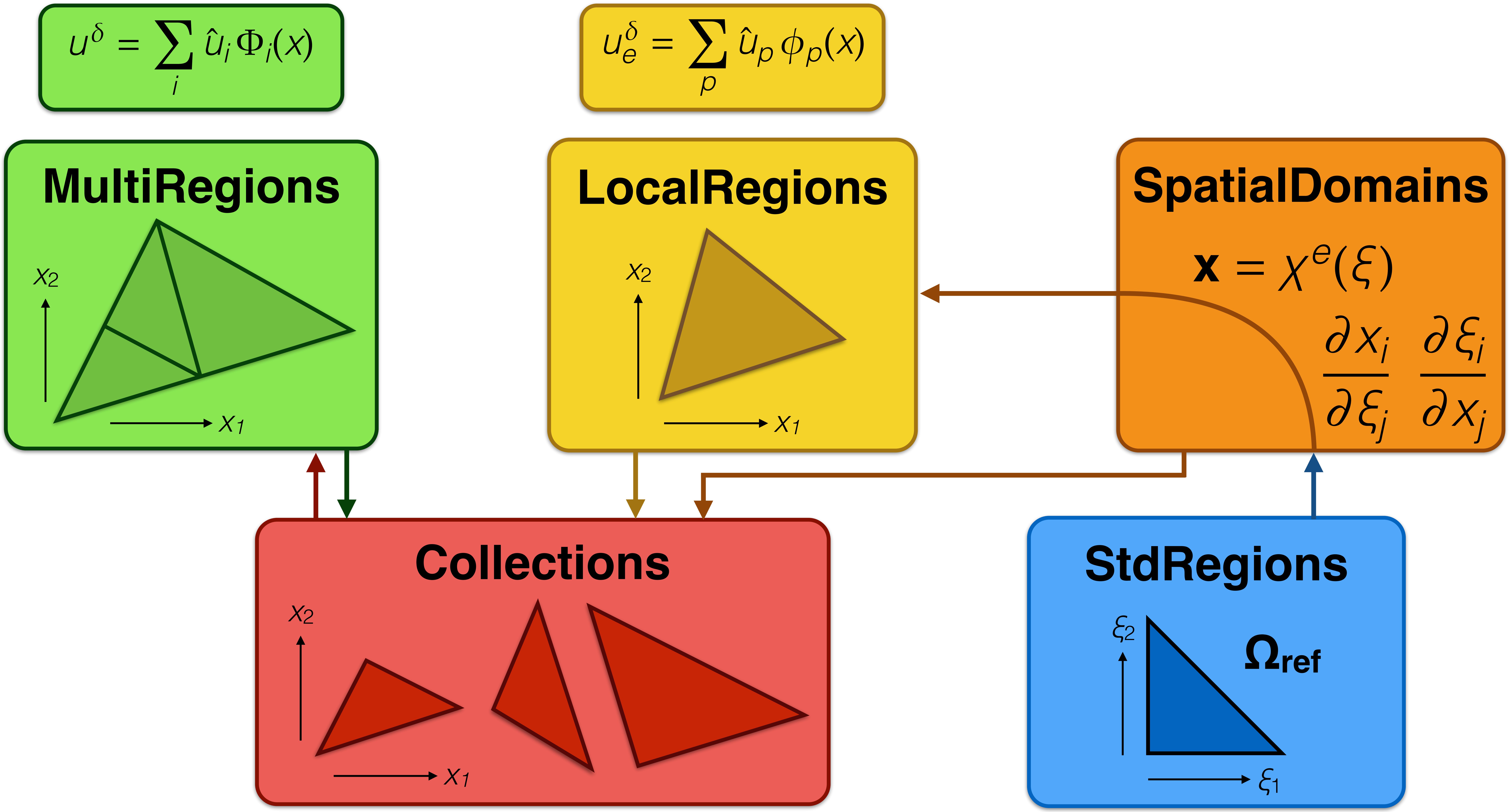




Framework design



Framework design



Coming in v5: Python interface

```cpp
#include <LibUtilities/BasicUtils/SessionReader.h>
#include <SpatialDomains/MeshGraph.h>

session = SessionReader::CreateInstance(argc, argv);
mesh    = SpatialDomains::Read(session);
cout << mesh->GetMeshDimension() << endl;
```

C++

```python
from NekPy.LibUtilities import SessionReader
from NekPy.SpatialDomains import MeshGraph

session = SessionReader.CreateInstance(sys.argv)
mesh    = MeshGraph.Read(session)
print(mesh.GetMeshDimension())
```

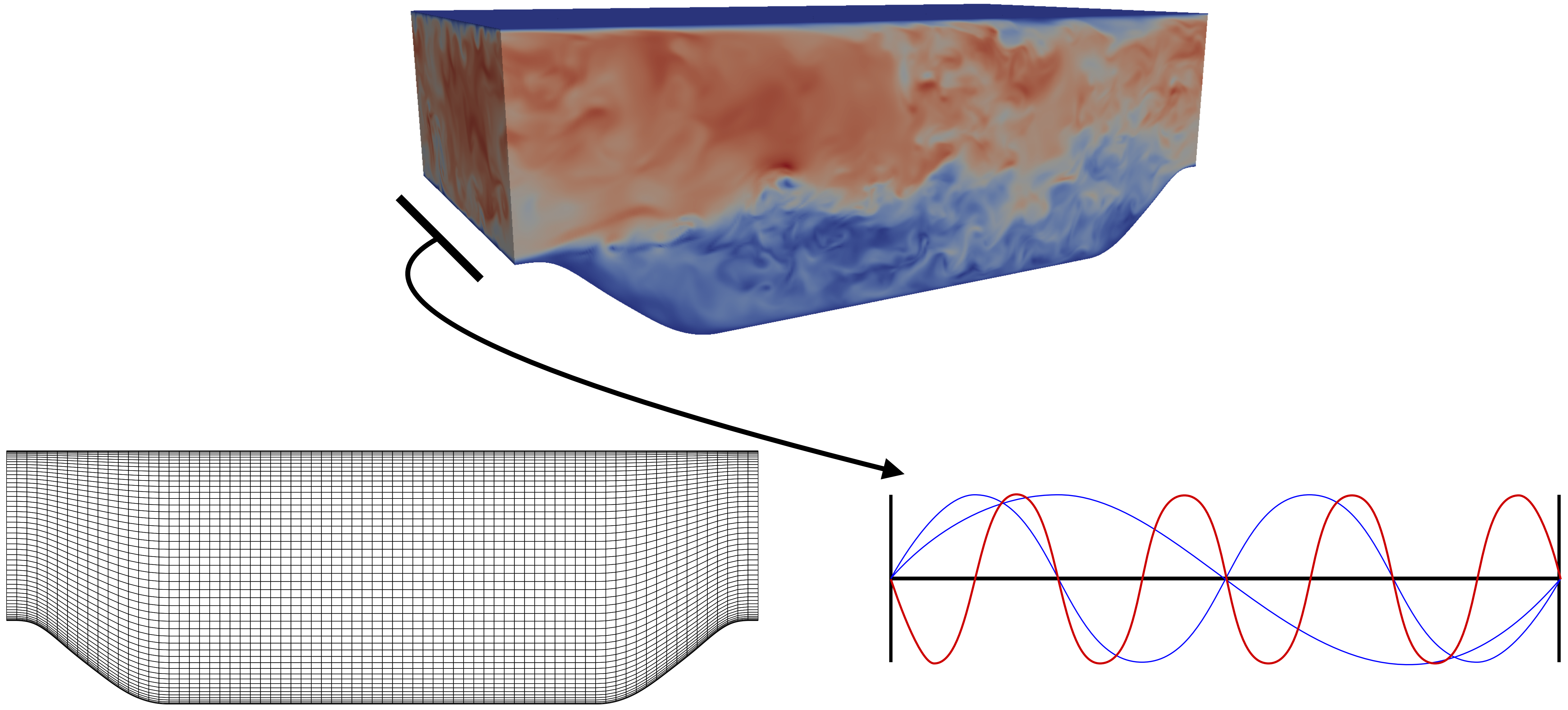
Python

- Python a great ‘glue' language for different software packages
- Also a good teaching aid
- Automated bindings really don't work for big codes (at least ours)
- Use boost::python, good support for inheritance, shared pointers

Quick demo: mesh visualisation

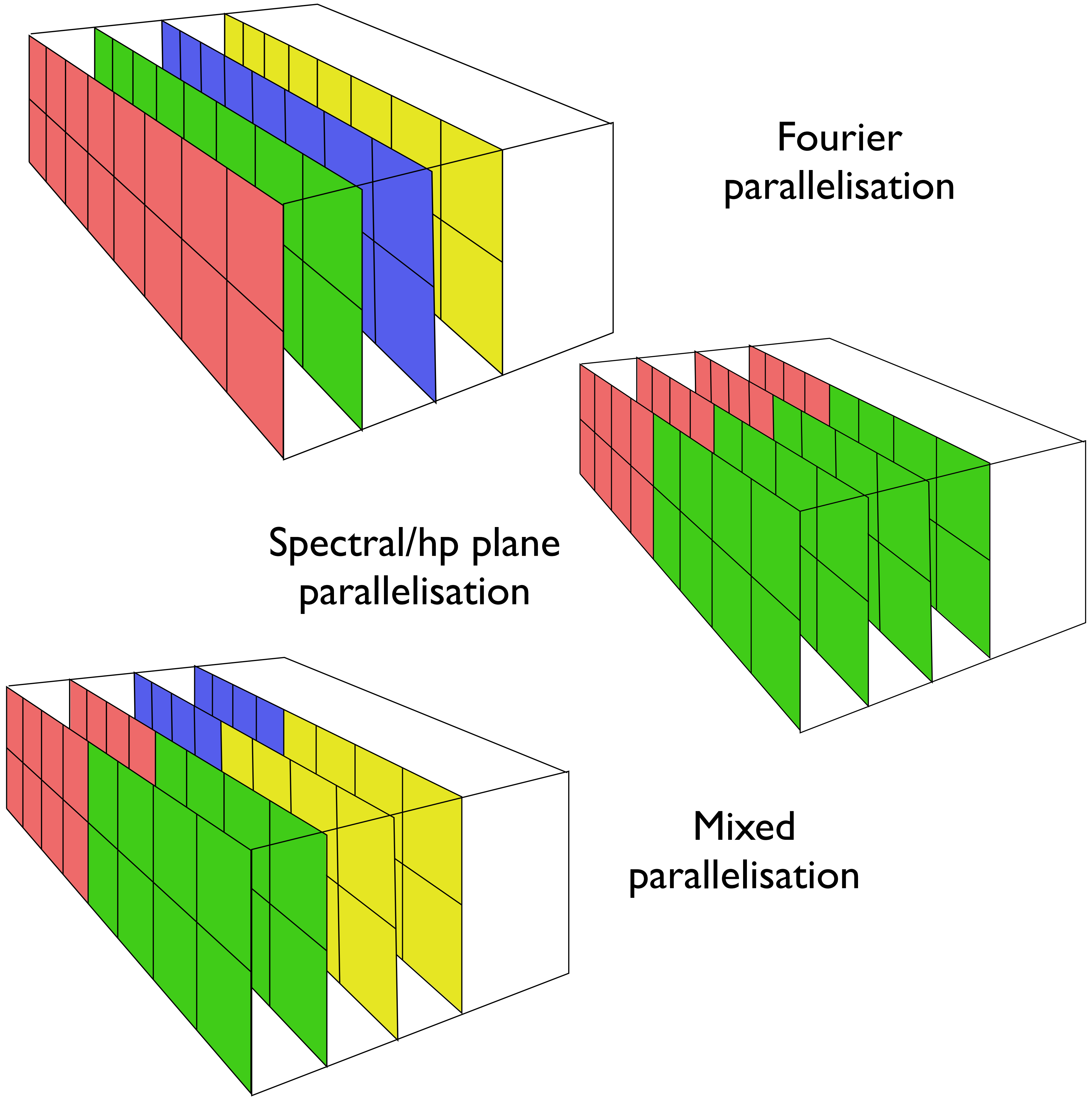
- Curved mesh visualisation is a challenging problem at present; stopgap is to create many samples/subdivisions and use existing linear methods.
- Want to evaluate isoparametric mapping at lots of points within reference element: trivially parallelisable so could use GPU for calculation, OpenGL interop to visualise
- Demo of Nektar++ wrappers in a modern OpenGL 4.2 (shaders) environment, using sympy to code generate basis functions & run on GPU in ~1k lines of code.

Other features

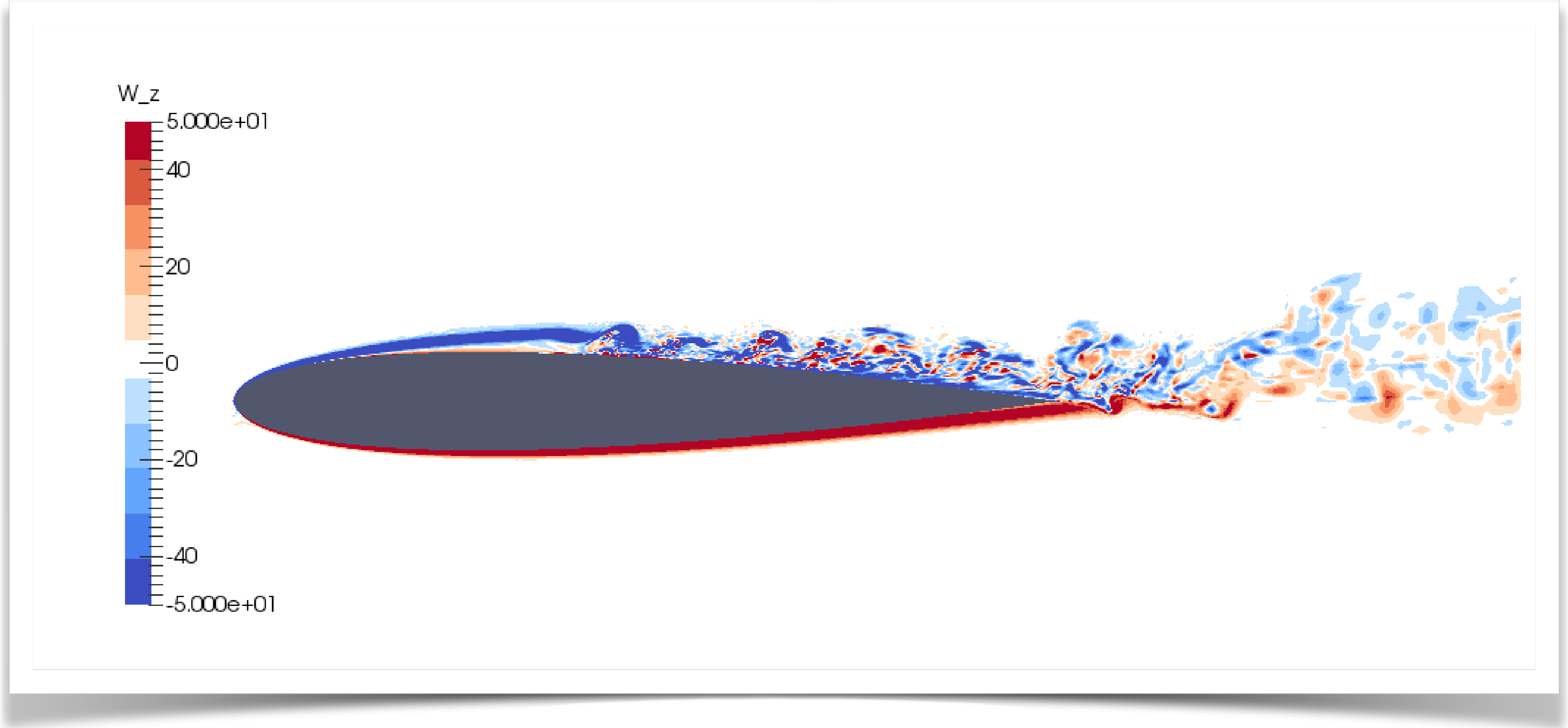


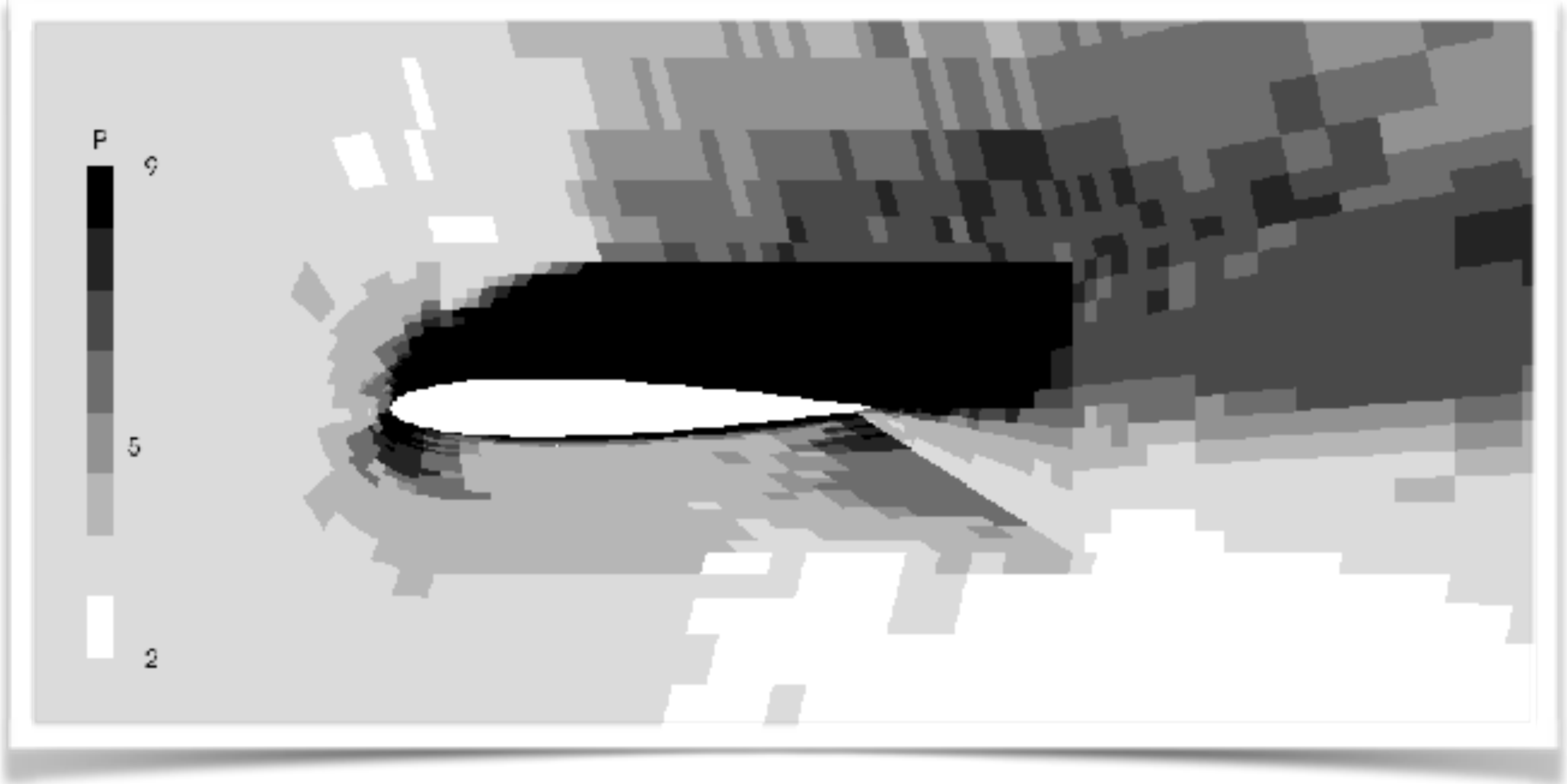
2D Spectral element mesh + 1D Fourier expansion

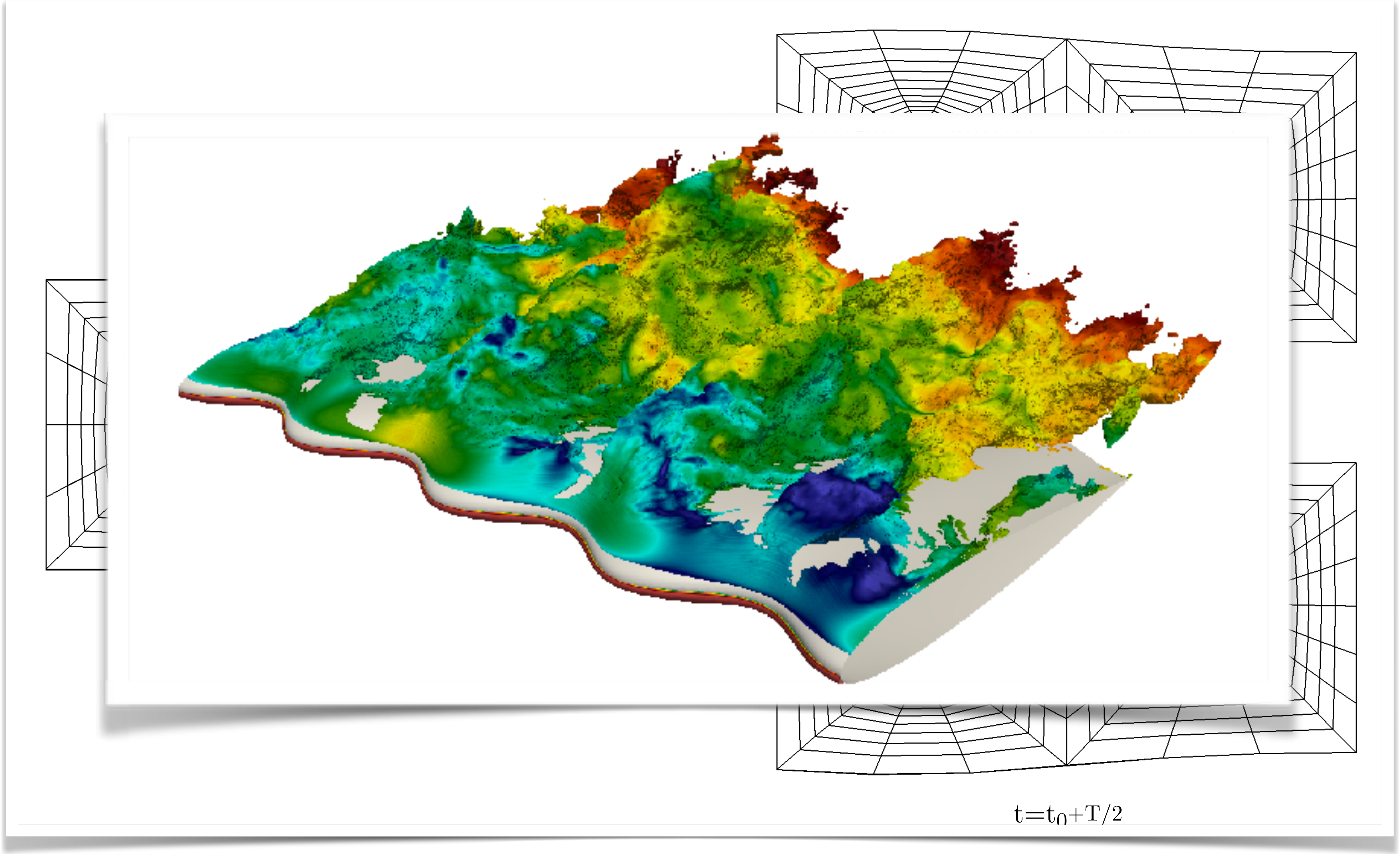
Hybrid discretisation



Other features







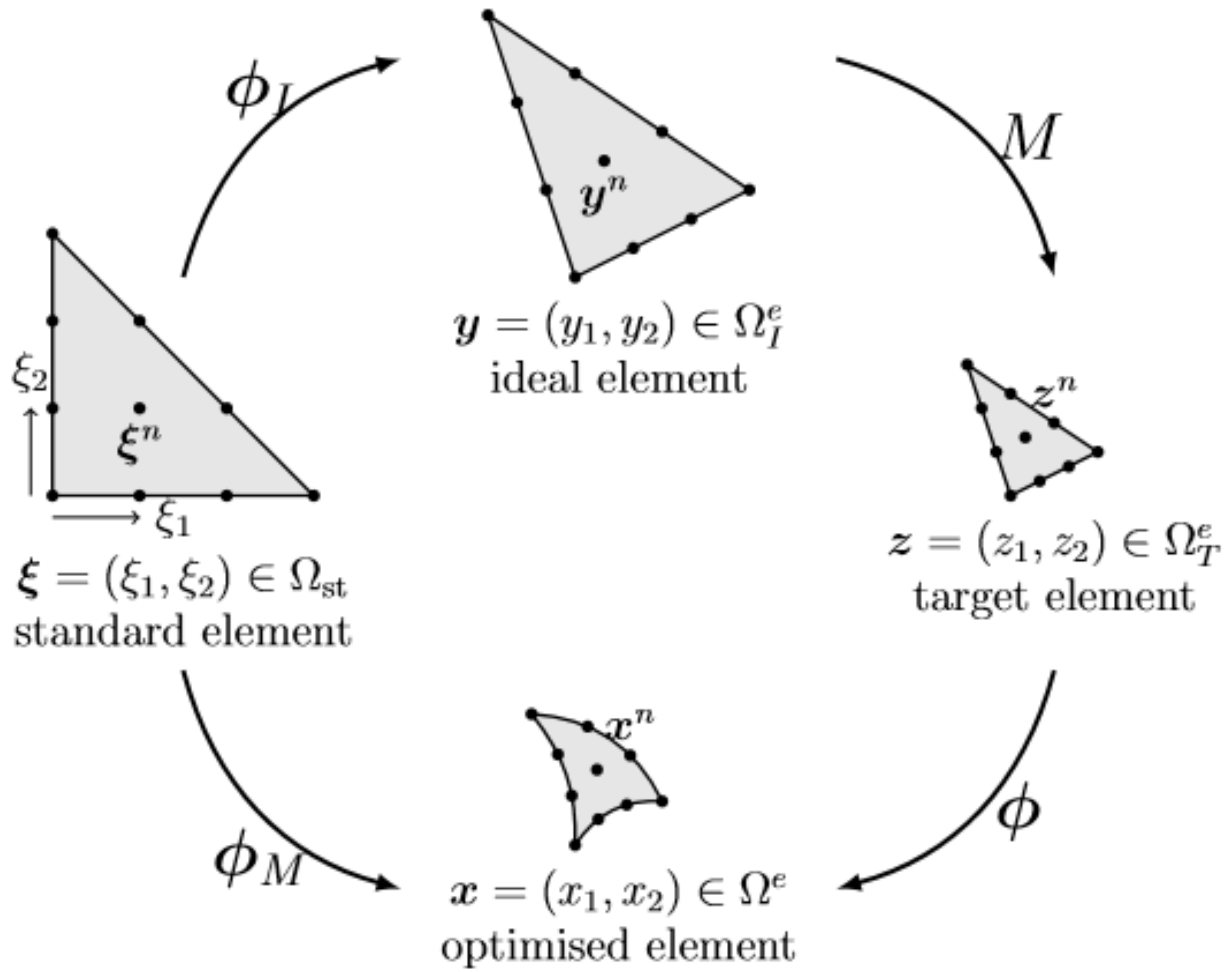
Spatially varying polynomial orders

D. Moxey et al, Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016, pp. 63–79

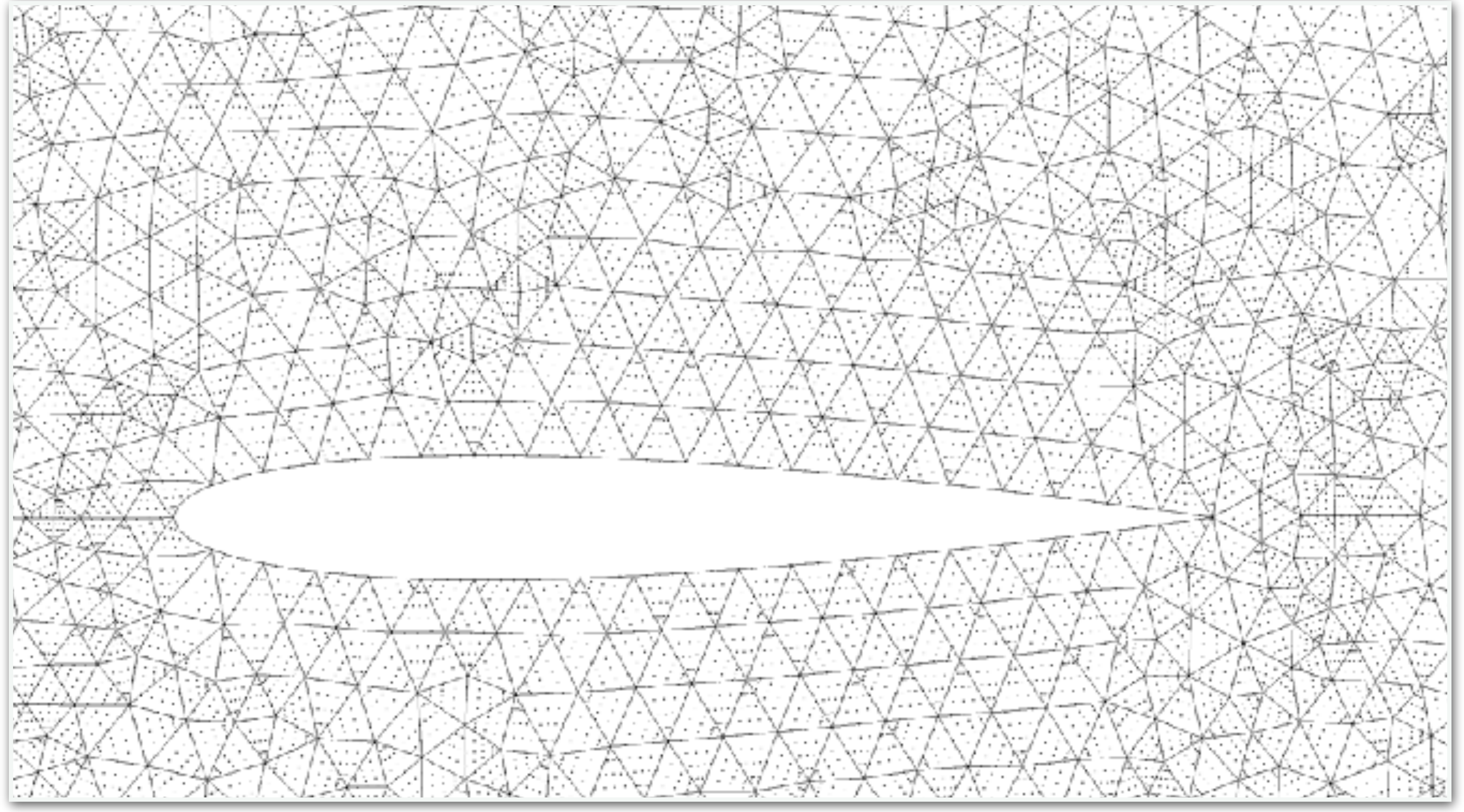
Coordinate mapping

D. Serson, J. Meneghini, and S. Sherwin, J. Comp. Phys. **316**, 243-254 (2016)

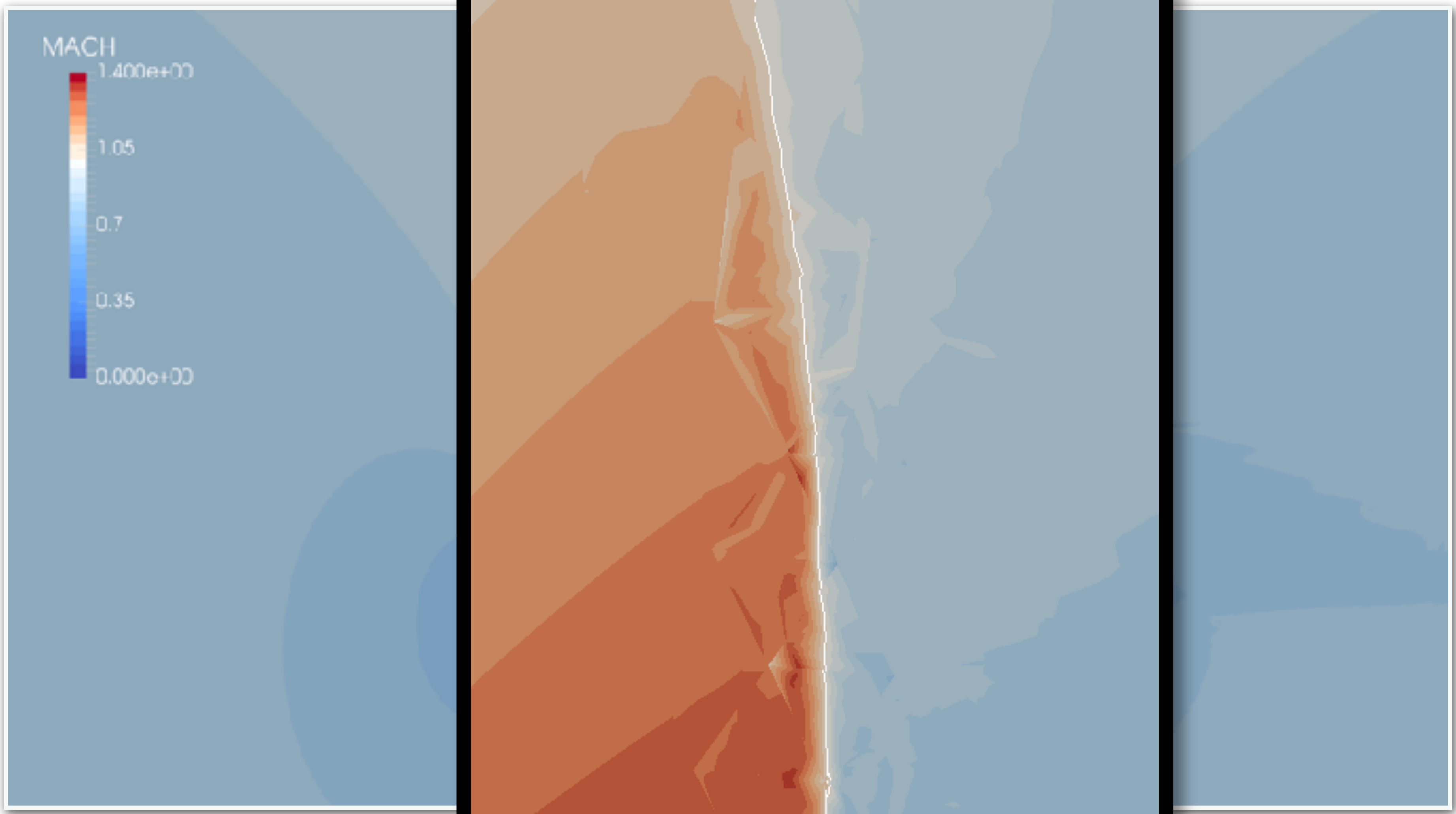
r-adaption: targeting element size

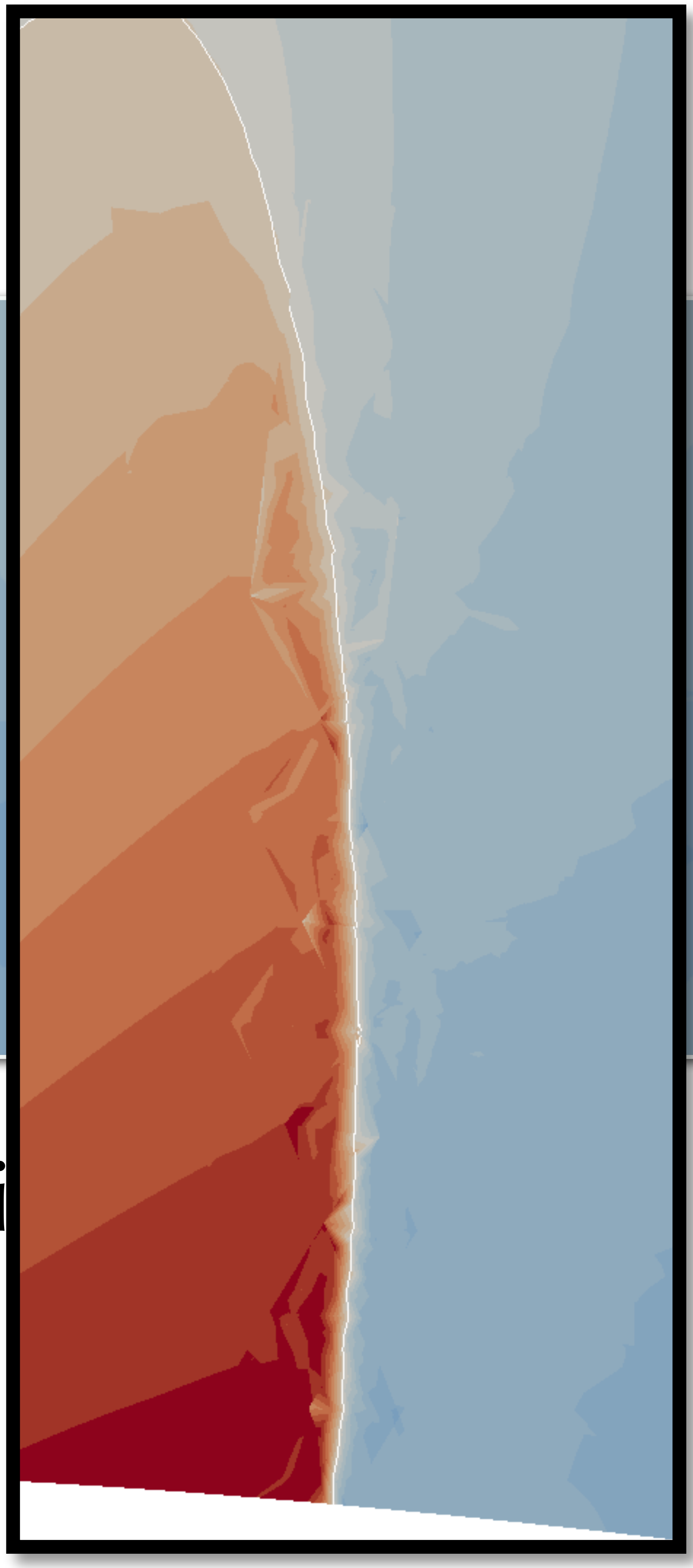


Example: NACA 0012 transonic



Starting mesh

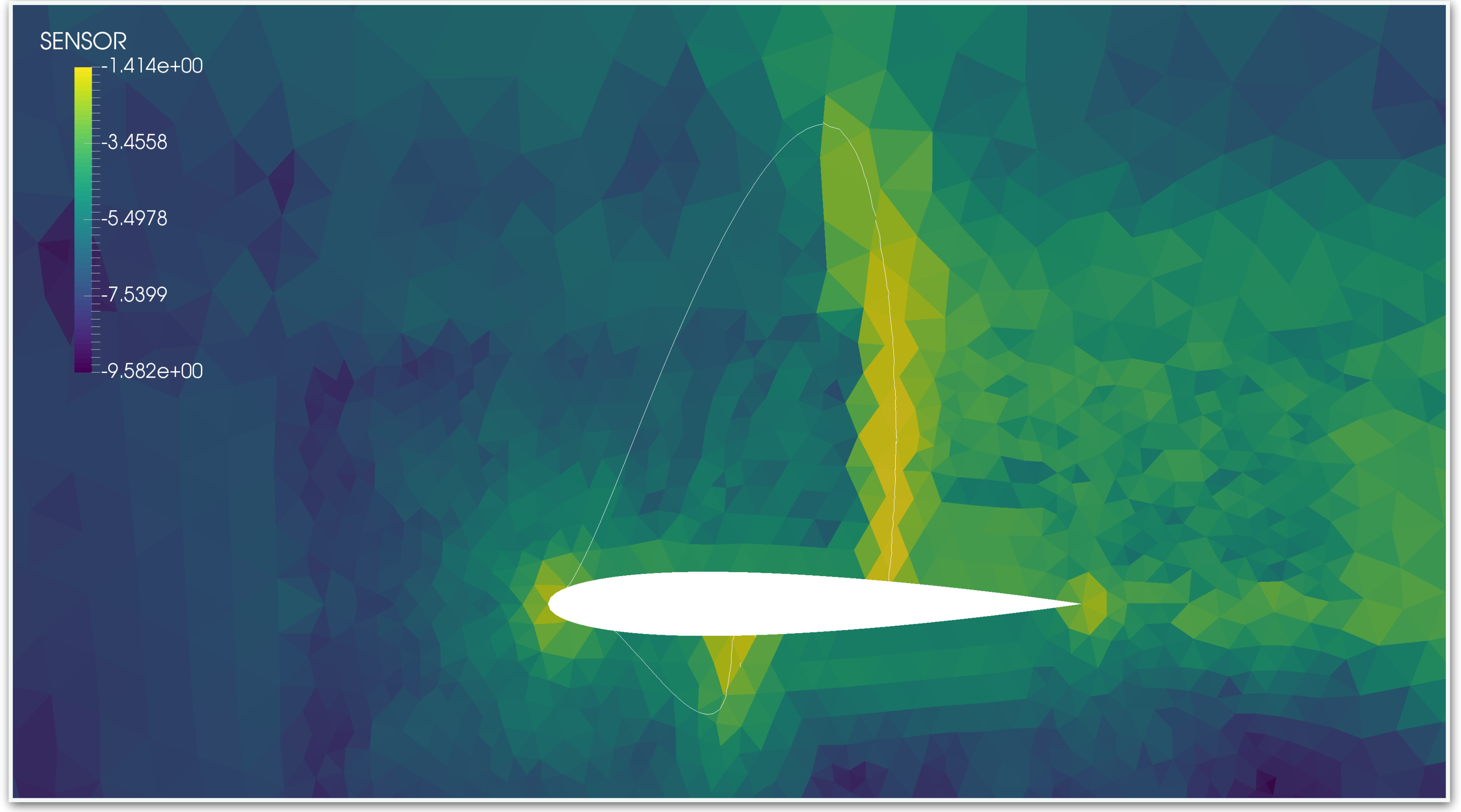




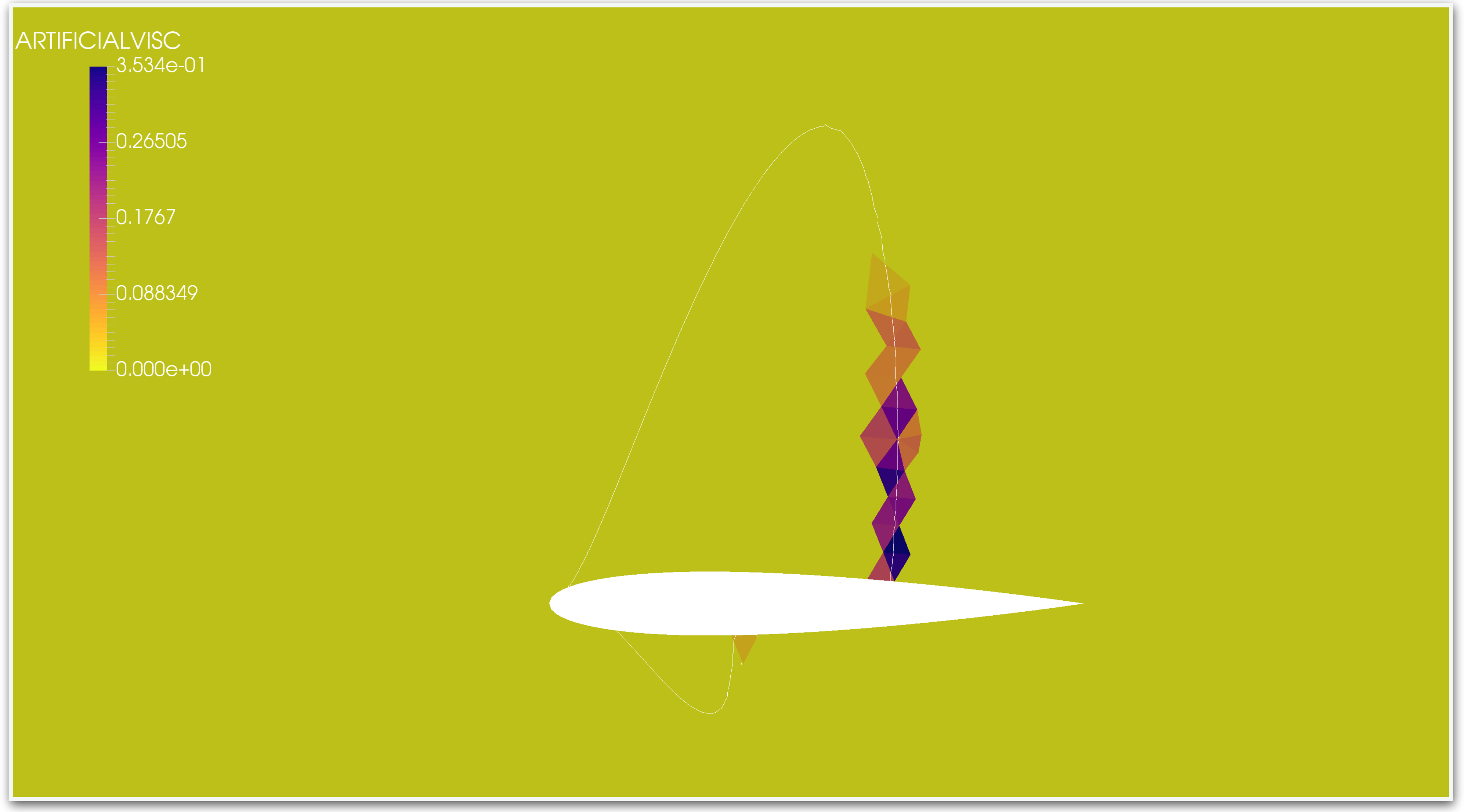
Ini

Ma = 0.8, 1.25° AoA

Example: NACA 0012 transonic

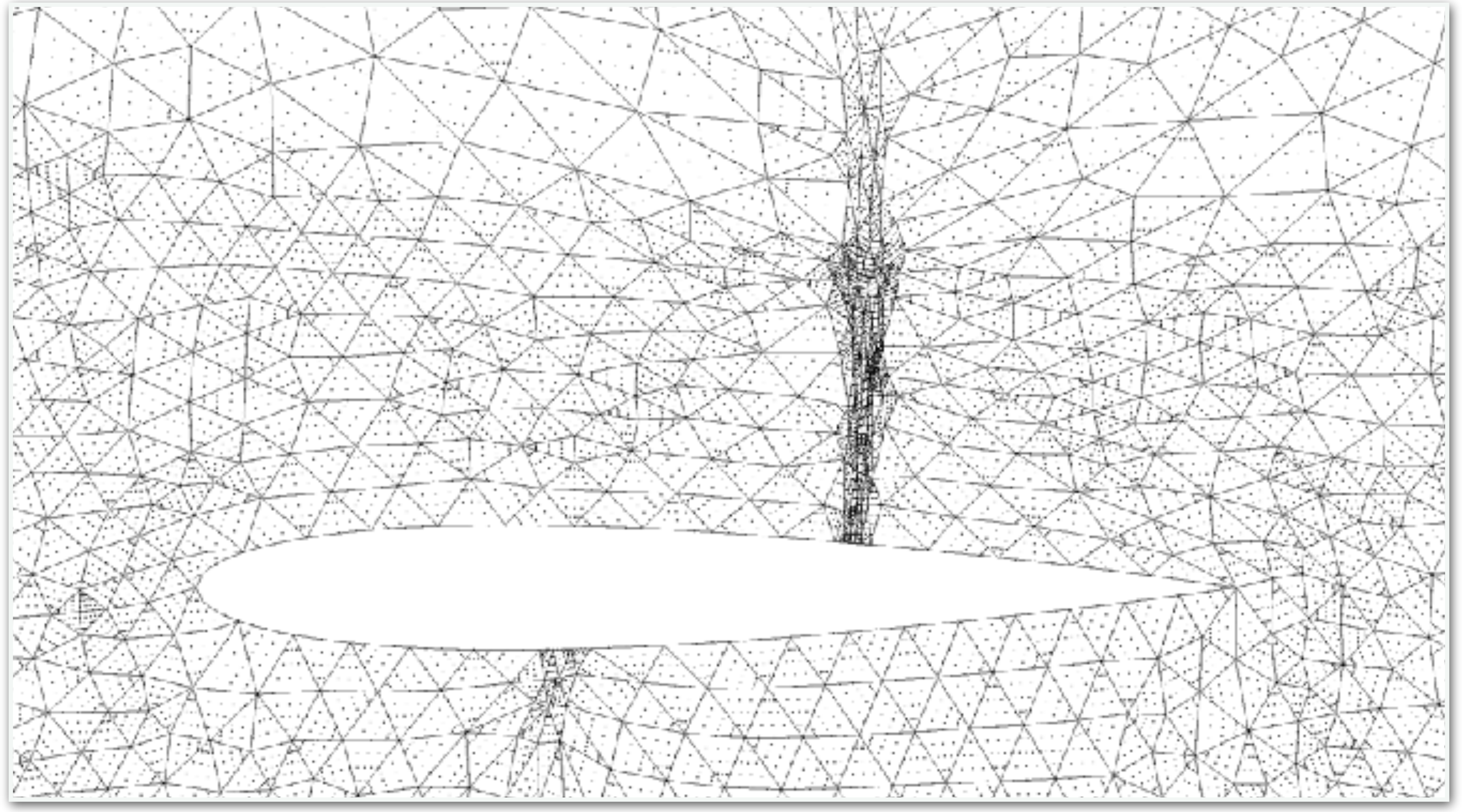


Discontinuity sensor

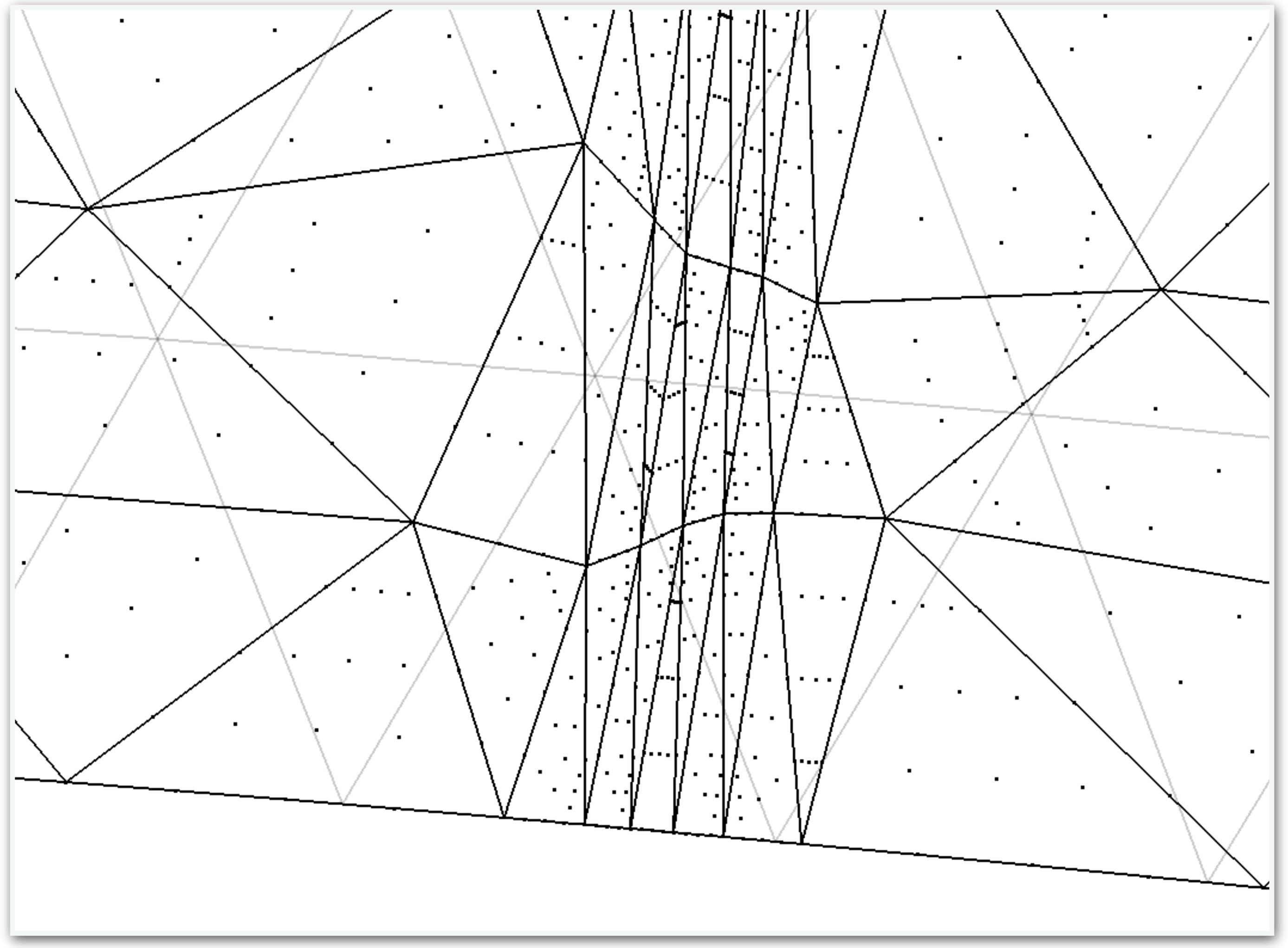


Artificial viscosity

Example: NACA 0012 transonic

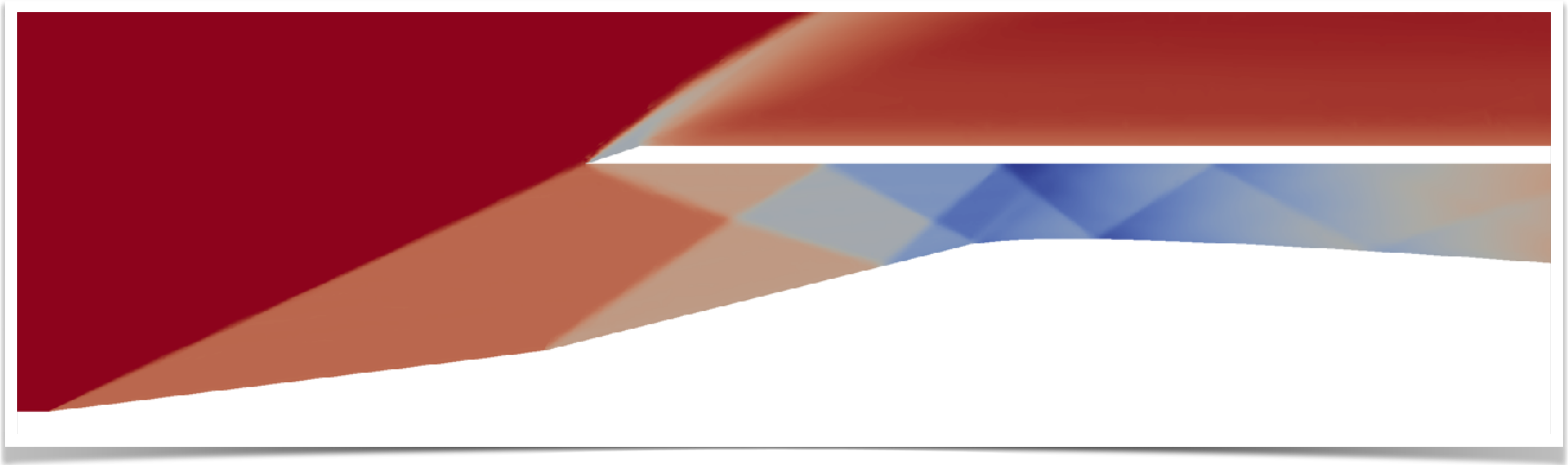


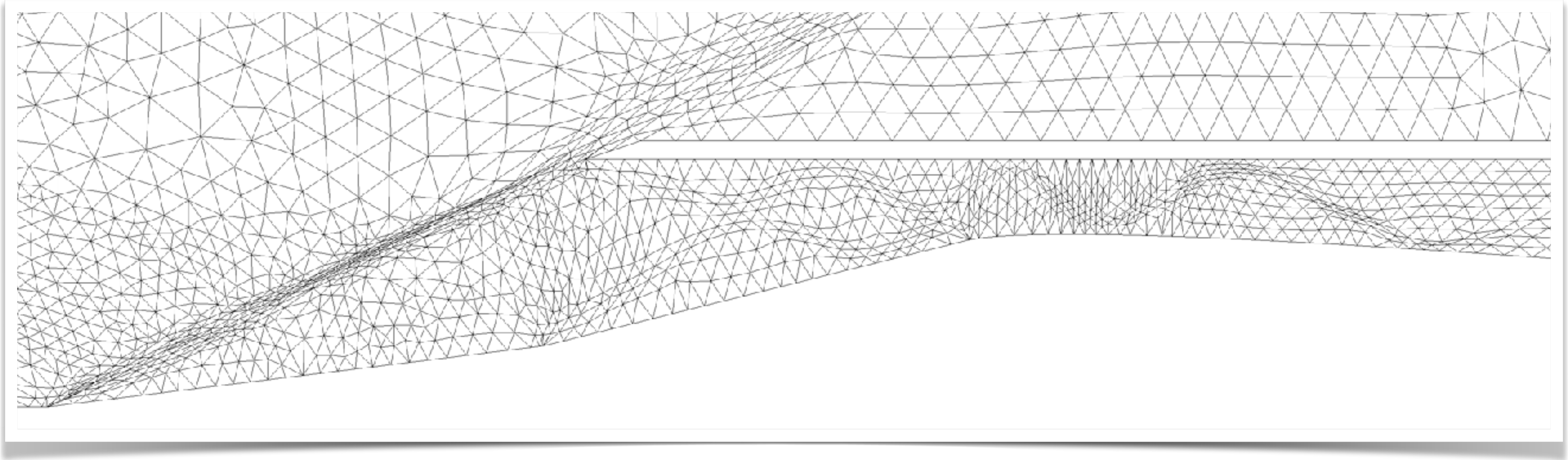
Calculate target size
& do r-adaptation



Use of CAD sliding

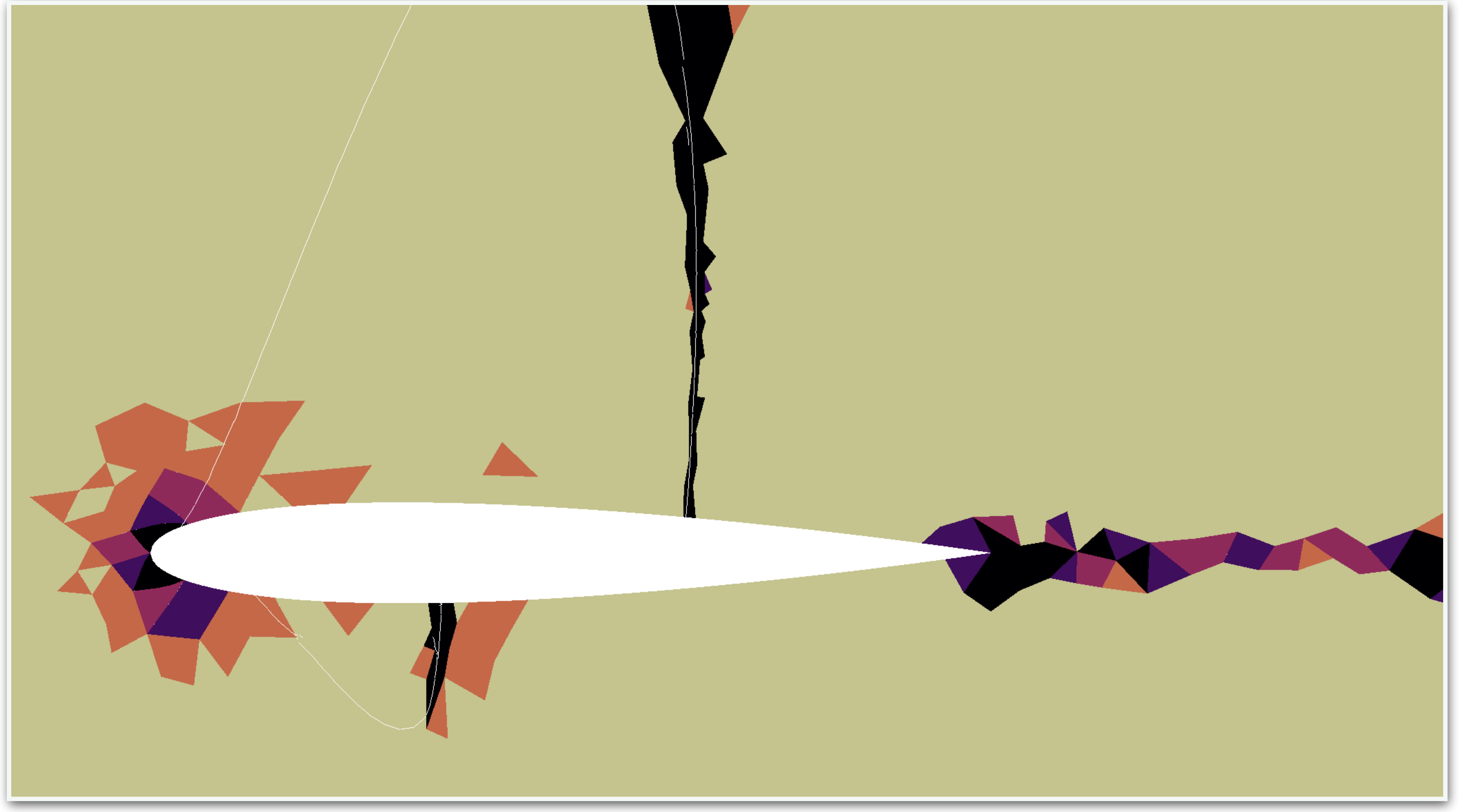
Supersonic example



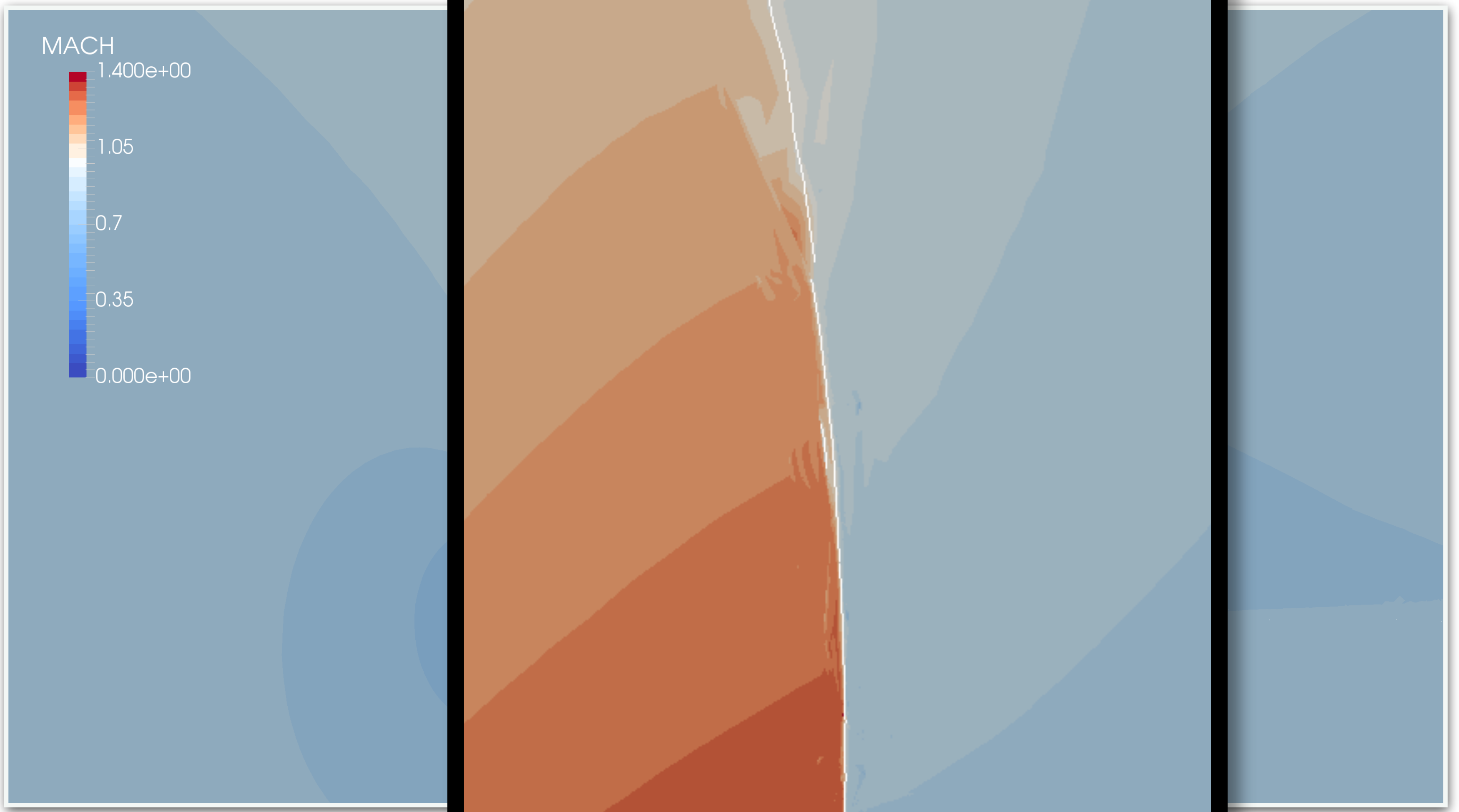


Supersonic intake
Ma = 1.0

Example: NACA 0012 transonic



Translate to variable p



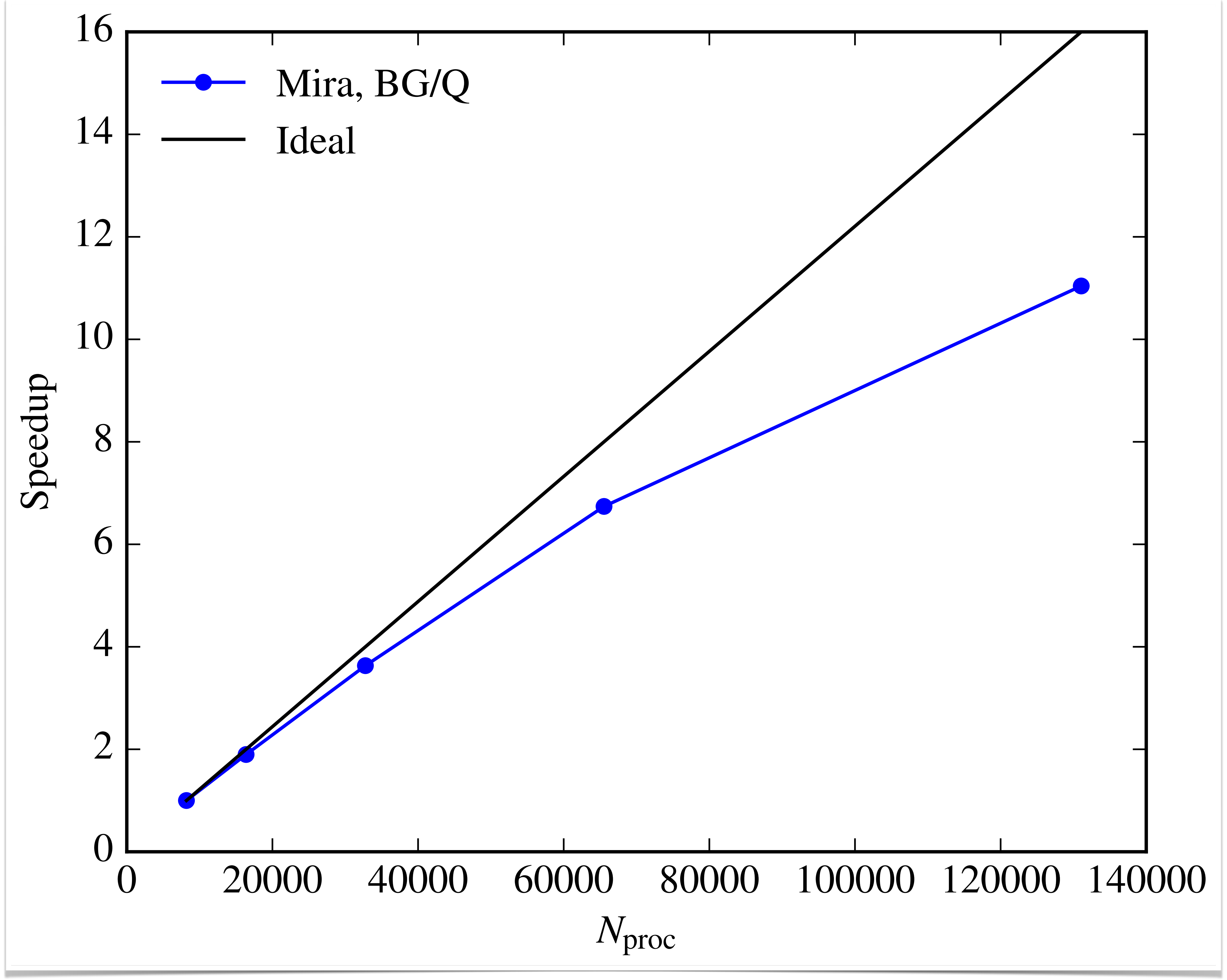
Improv ng

High-order fluid simulations

- Now that we have some kind of a route to a mesh, the next step is to work out how to do something useful with it!
- Particular focus on **incompressible flow** simulations and, in particular, **high-fidelity** simulations.
- Consider inherently **unsteady flows**: investigate use of **implicit LES**.
- Our message: still computationally expensive & requires HPC, but **should not be prohibitive** and **should scale** with high-order simulations.

Solving at scale

- Relying on HPC means we need efficient and scalable linear solvers.
- Mesh is decomposed across processors; local dense matrices formed for each element, communication with `gslib`.
- Core of the code scales well on Mira: test case of a ~5m element F1 geometry at fifth order.
- However still some work to do on scalable preconditioning!



High-order splitting scheme

Navier–Stokes:

$$\partial_t \mathbf{u} + \mathbf{N}(\mathbf{u}) = -\nabla p + \nu \nabla^2 \mathbf{u}$$
$$\nabla \cdot \mathbf{u} = 0$$

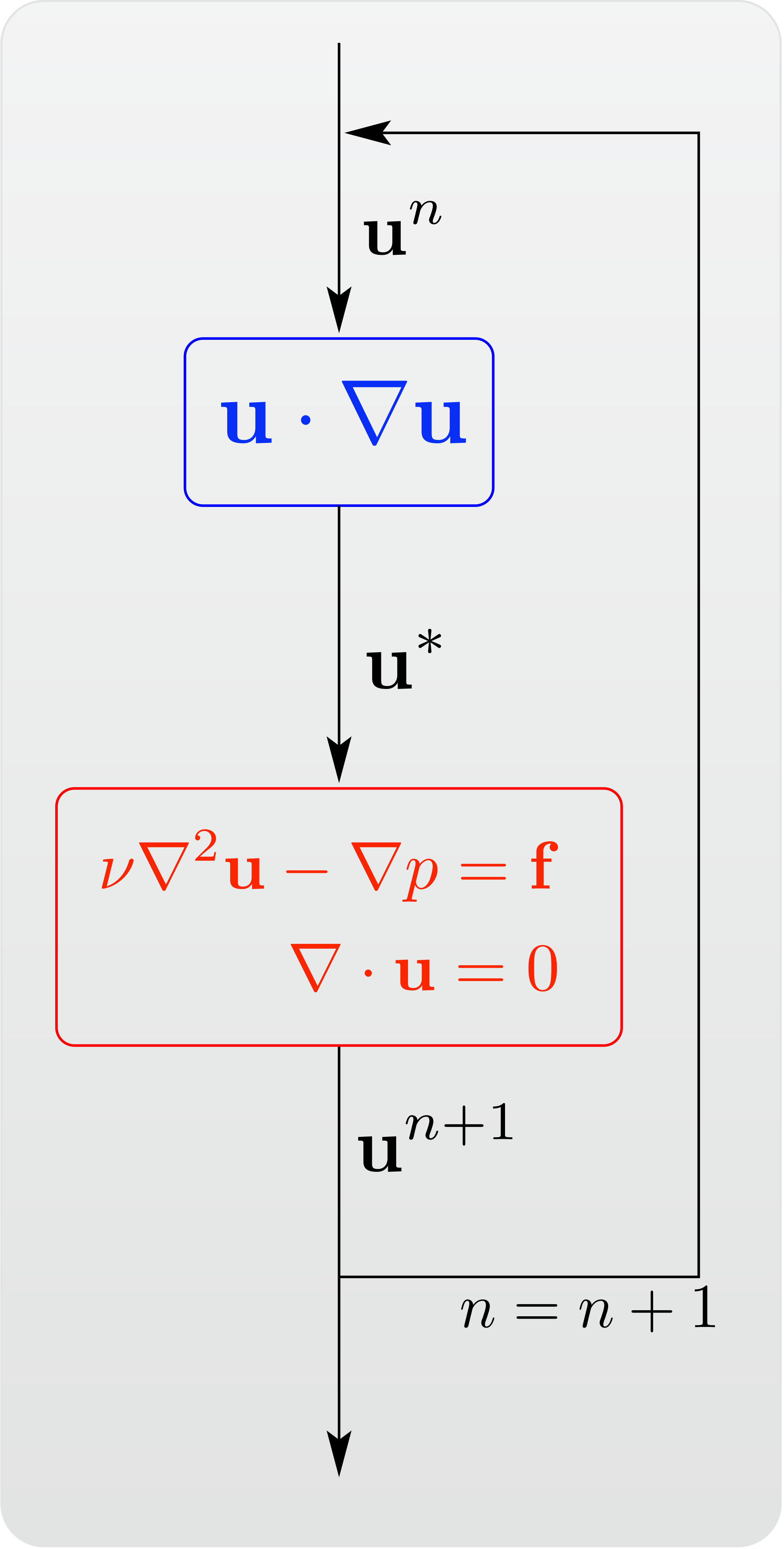
Velocity correction scheme *(aka stiffly stable)*:

Orszag, Israeli, Deville (90), Karnaidakis Israeli, Orszag (1991), Guermond & Shen (2003)

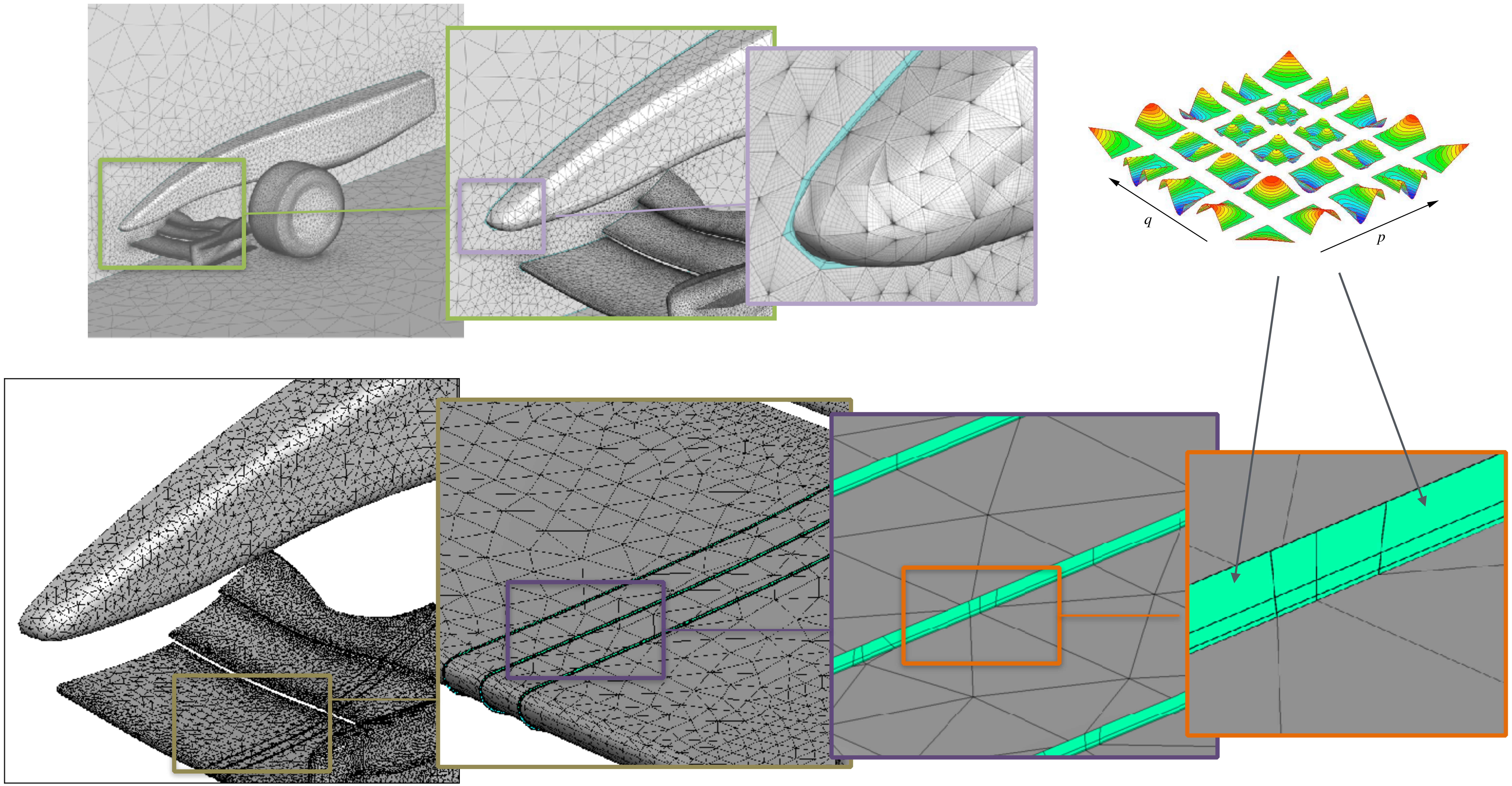
Advection: $u^* = -\sum_{q=1}^{J} \alpha_q \mathbf{u}^{n-q} - \Delta t \sum_{q=0}^{J-1} \beta_q \mathbf{N}(\mathbf{u}^{n-q})$

Pressure Poisson: $\nabla^2 p^{n+1} = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*$

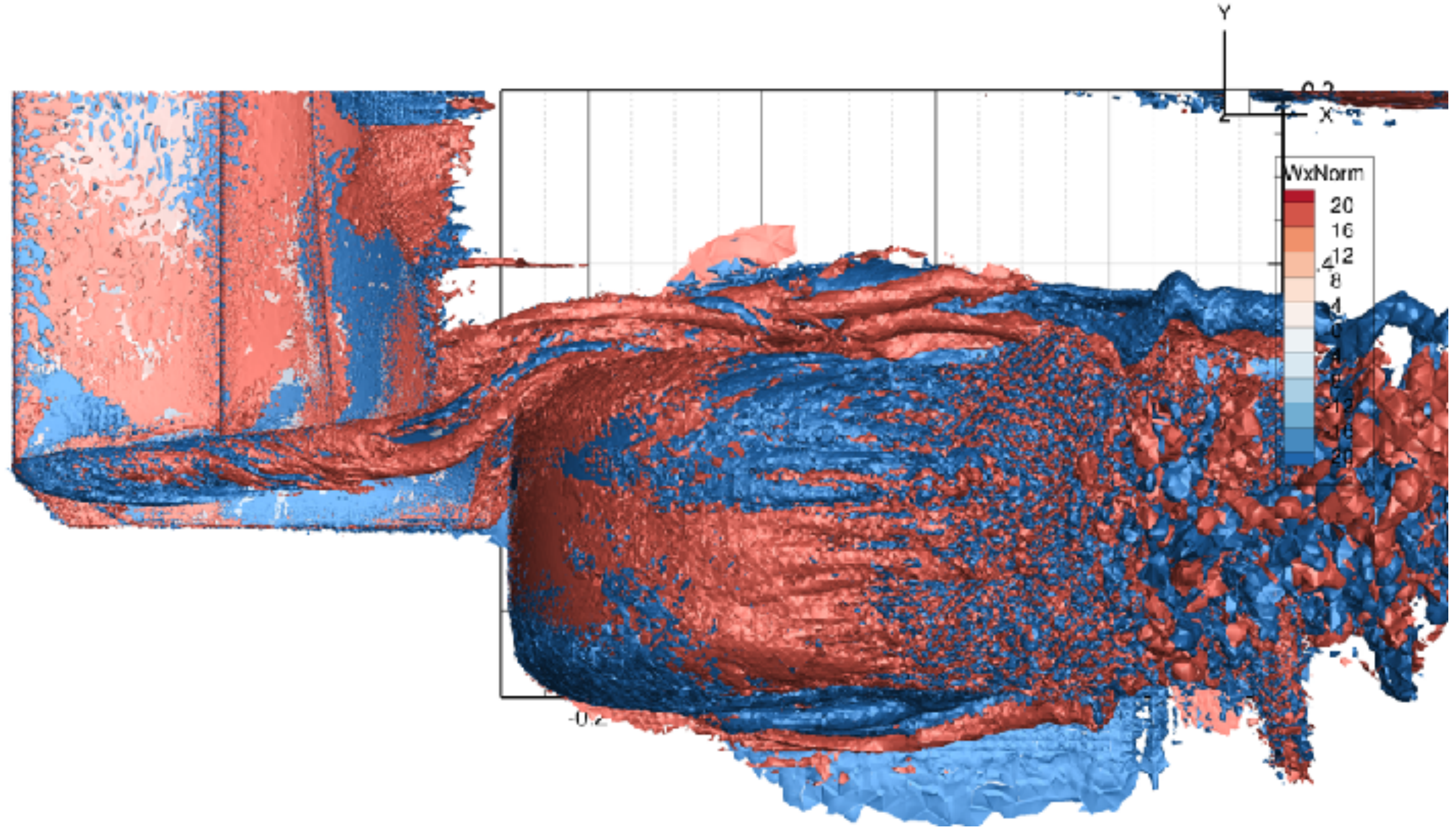
Helmholtz: $\nabla^2 \mathbf{u}^{n+1} - \frac{\alpha_0}{\nu \Delta t} \mathbf{u}^{n+1} = -\frac{\mathbf{u}^*}{\nu \Delta t} + \frac{1}{\nu} \nabla p^{n+1}$

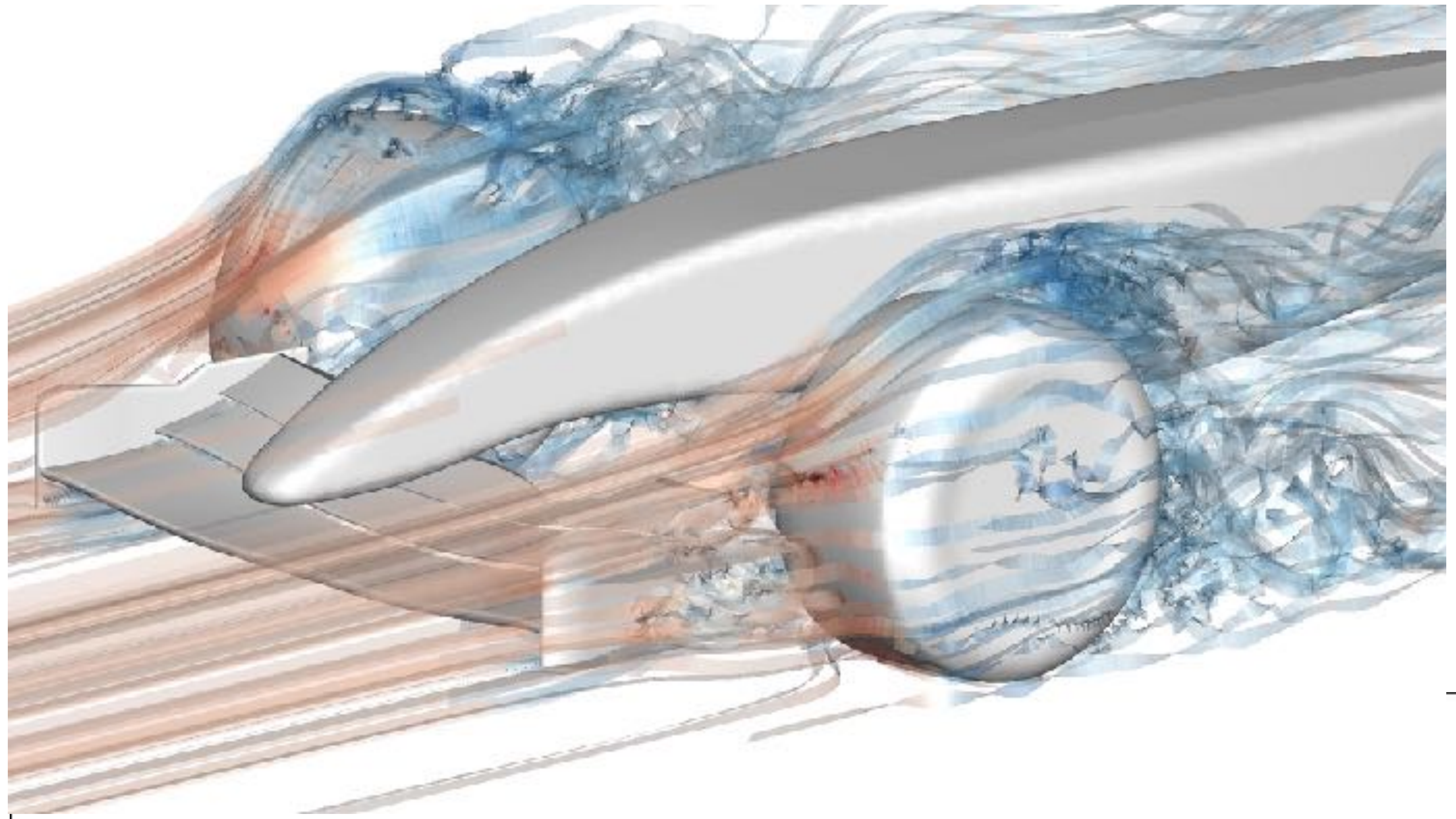


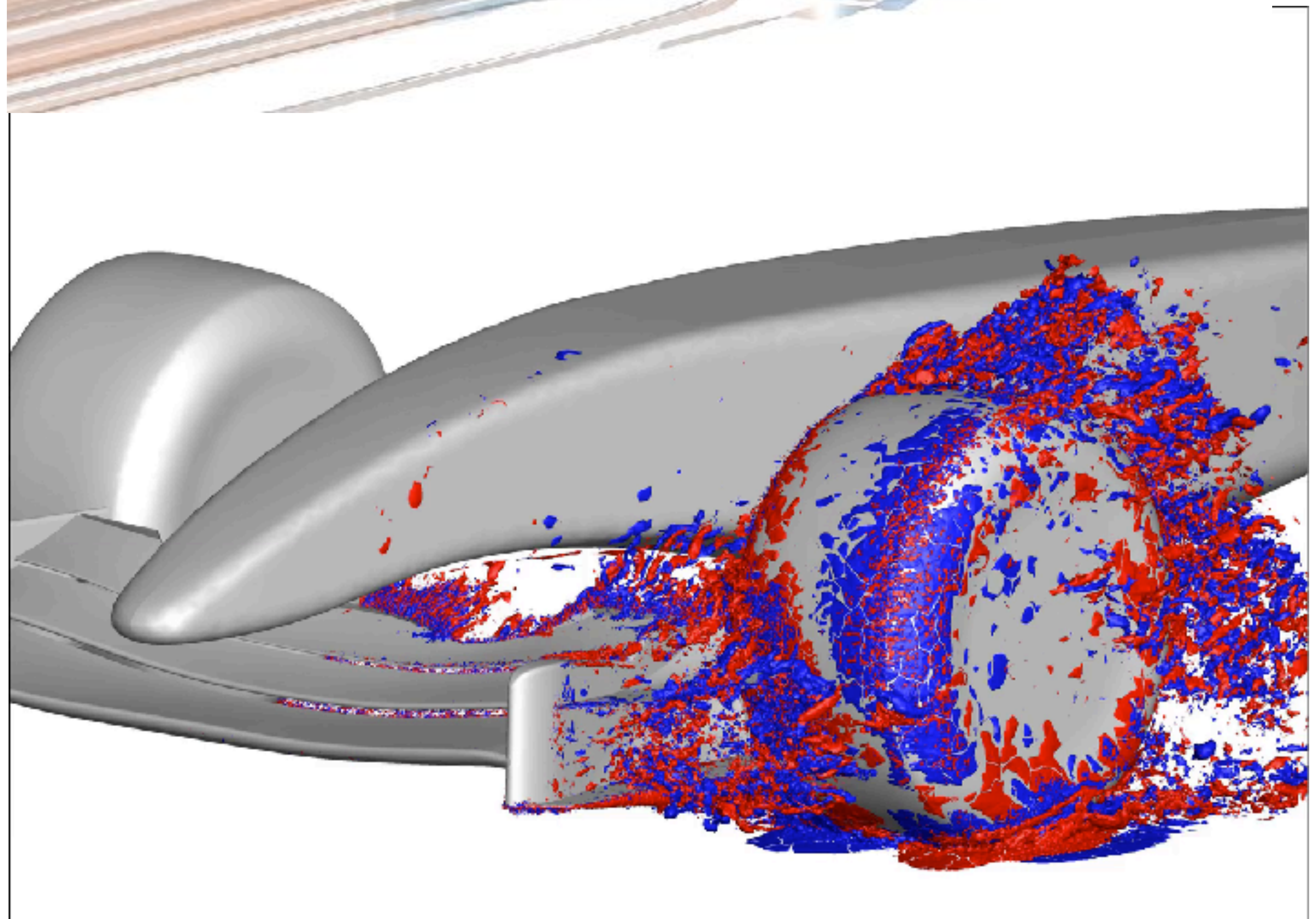
Meshing for F1 applications



More complex geometries



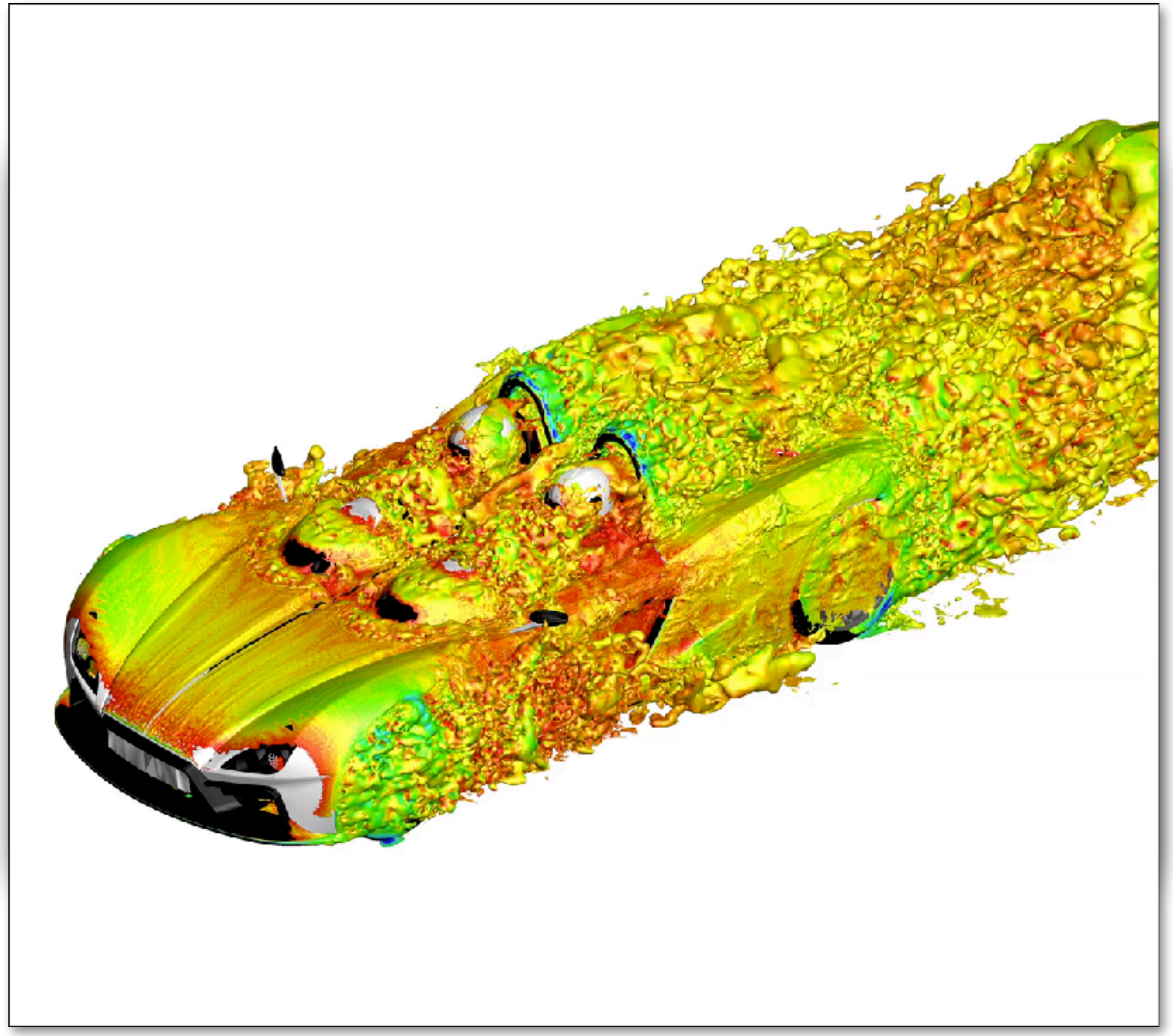


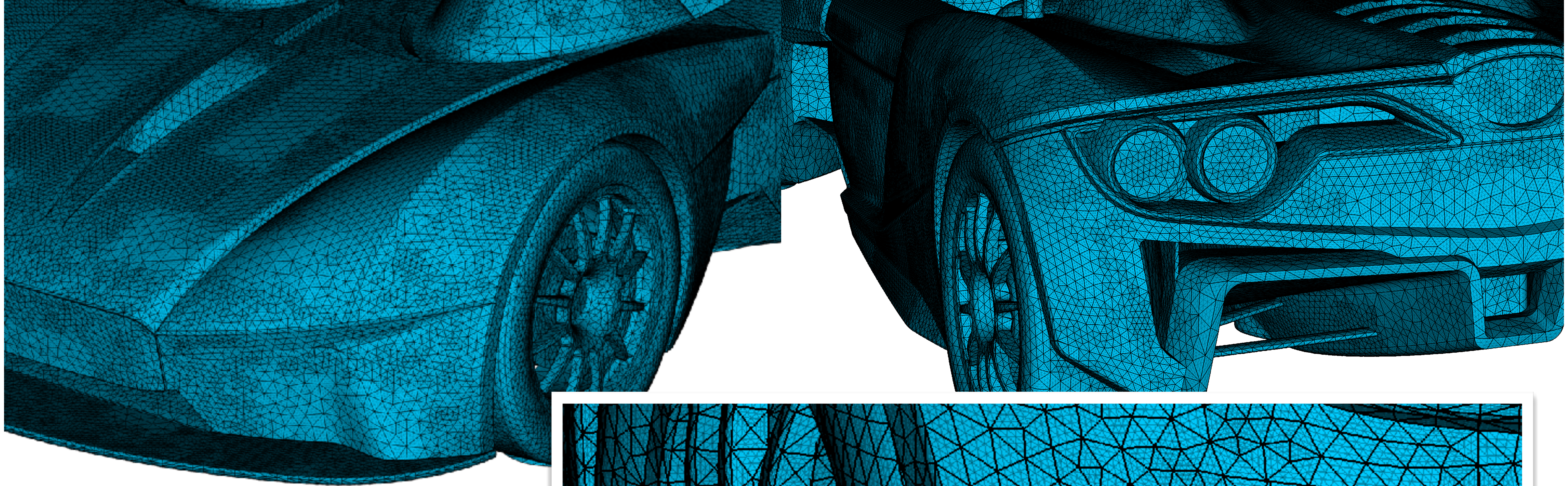


Supported by ARCHER leadership award (20m CPU hours)

Elemental road racing car

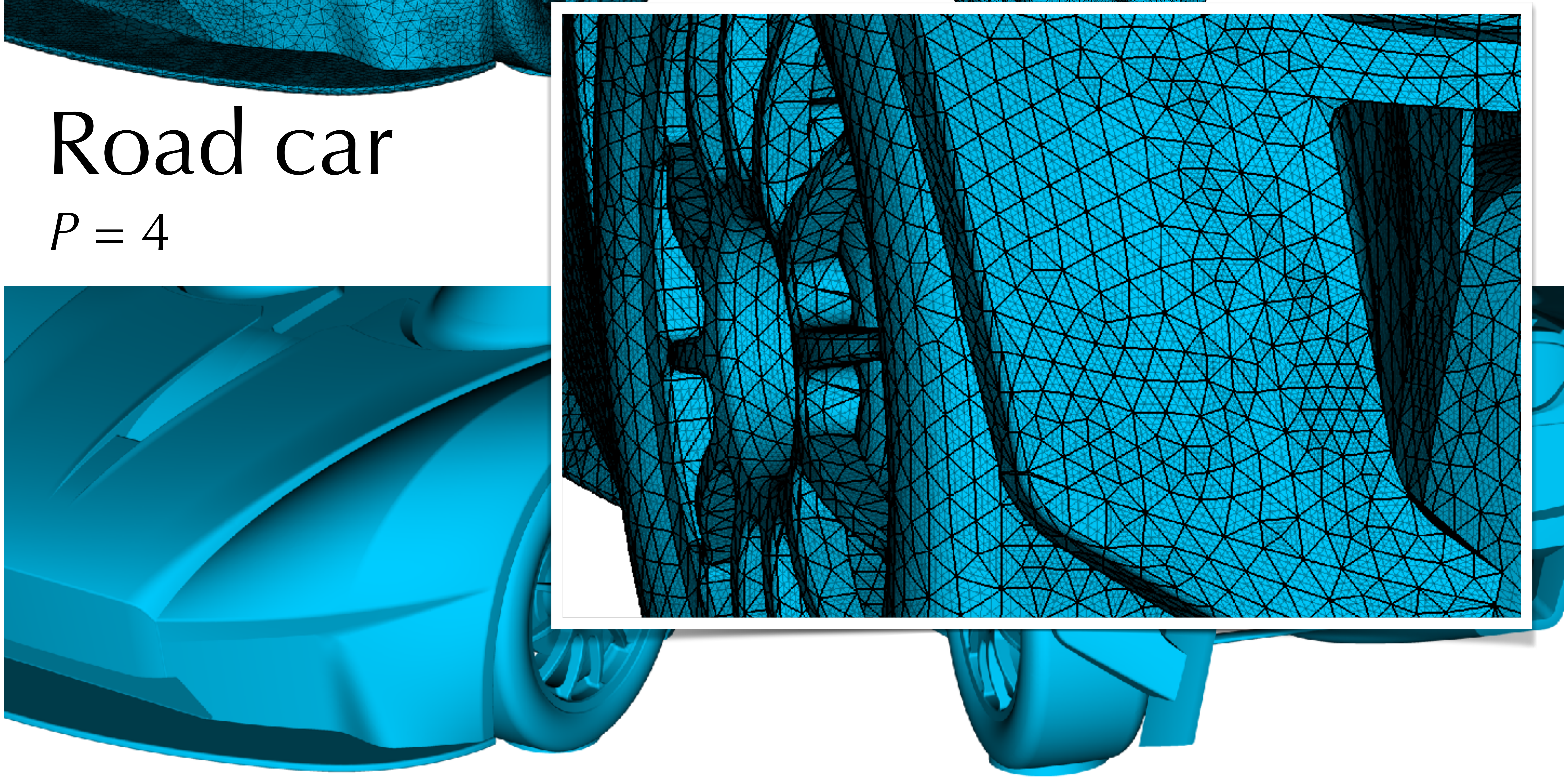
- Most challenging case undertaken with Nektar++ to date (that I know of!)
- Re ~ 1m, around 1bn dof.
- Simulated at $P = 5$ with a matching high-order mesh and SVV-LES.
- Aim to identify aerodynamic issues and refine design.



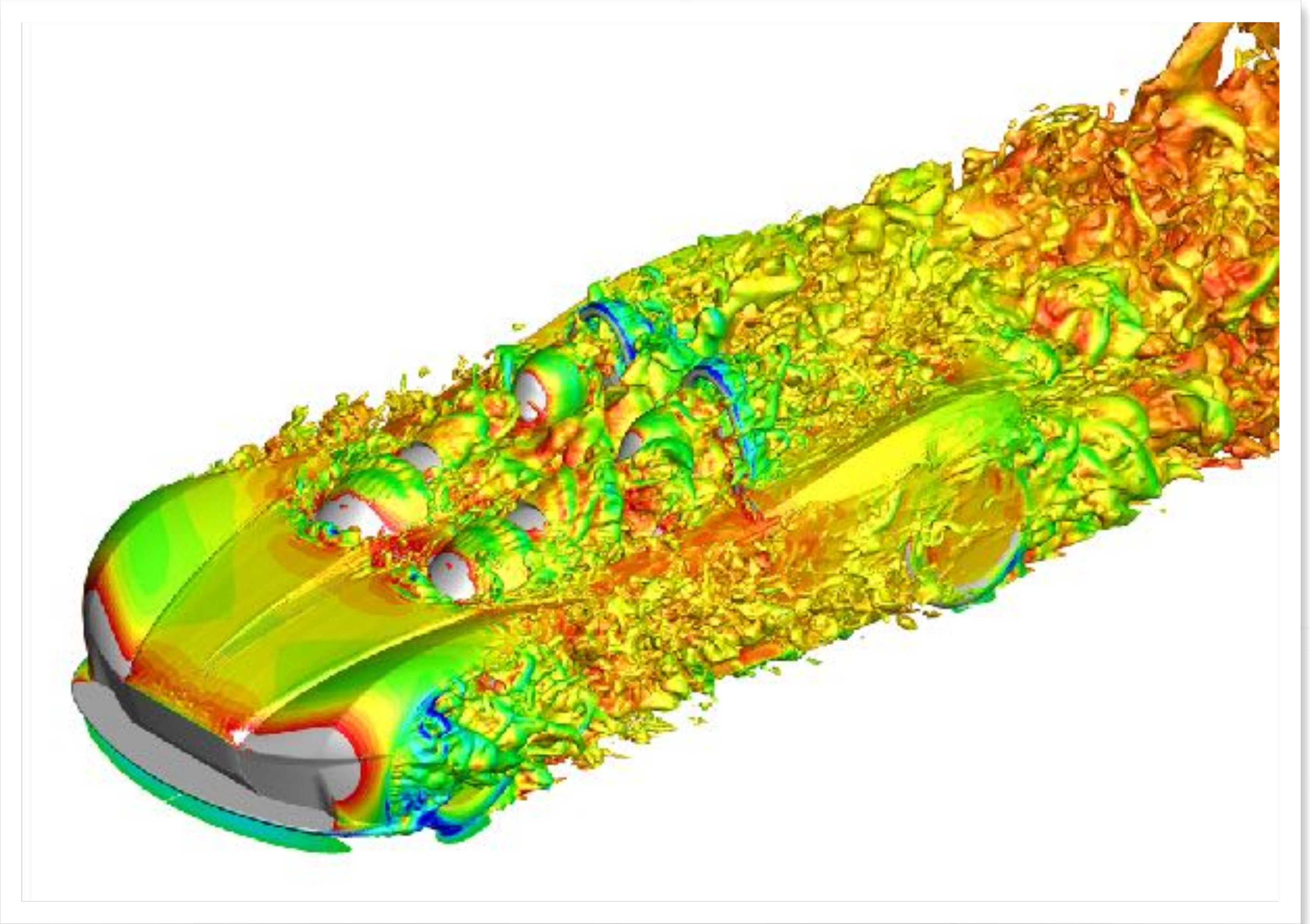


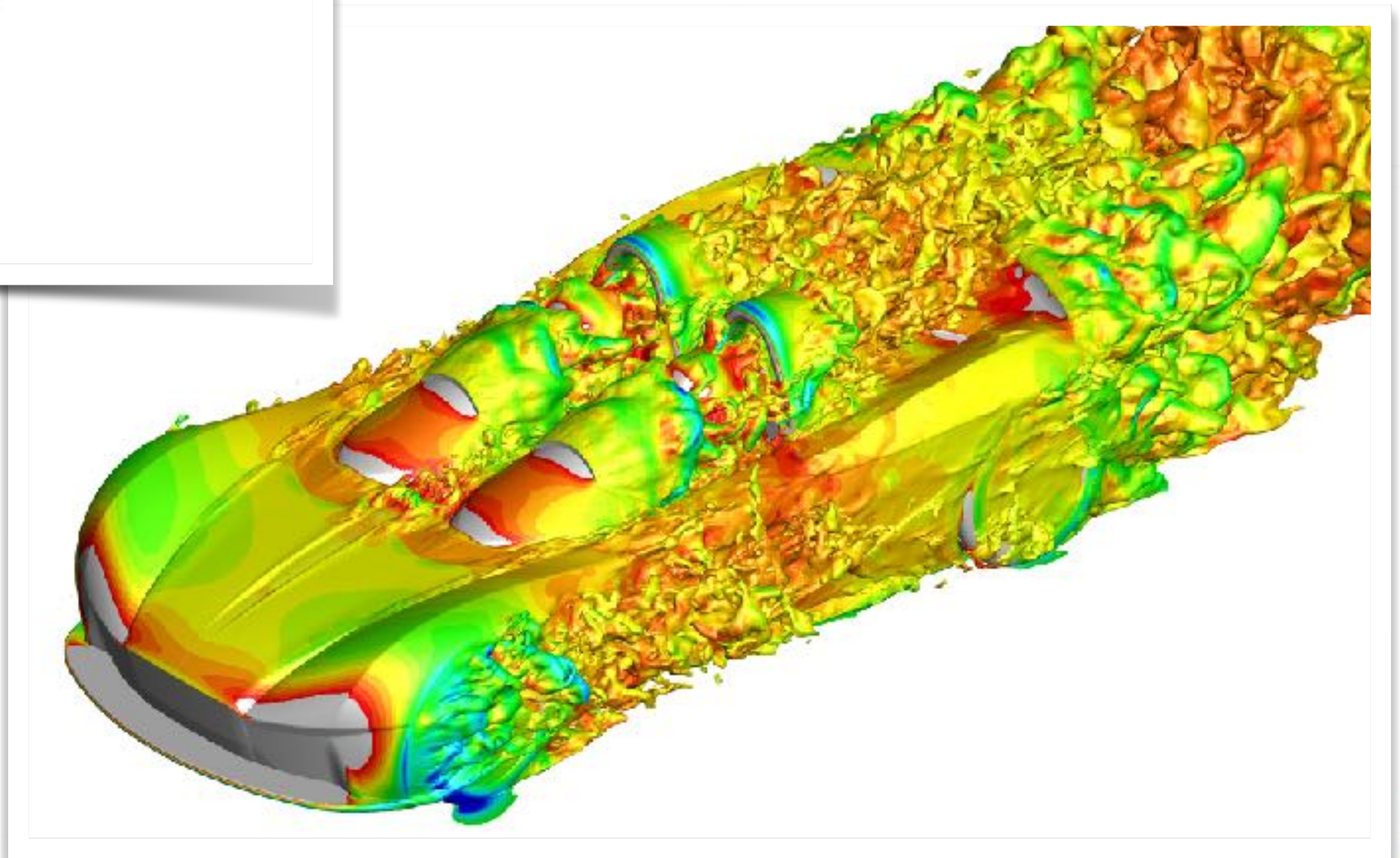
Road car

$P = 4$

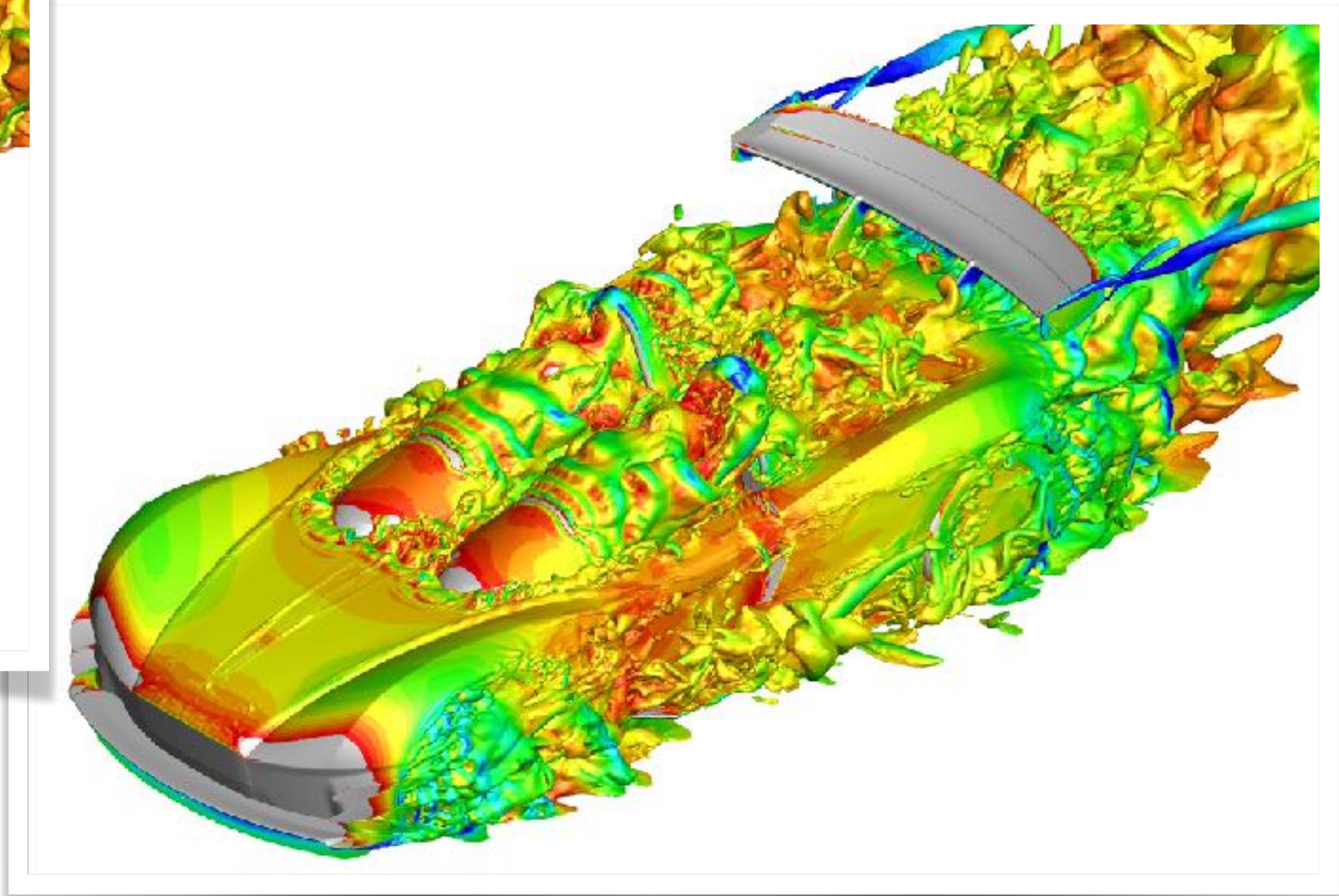


Elemental road race car





5th order
Re = 1m



Design 2: +33% Downforce

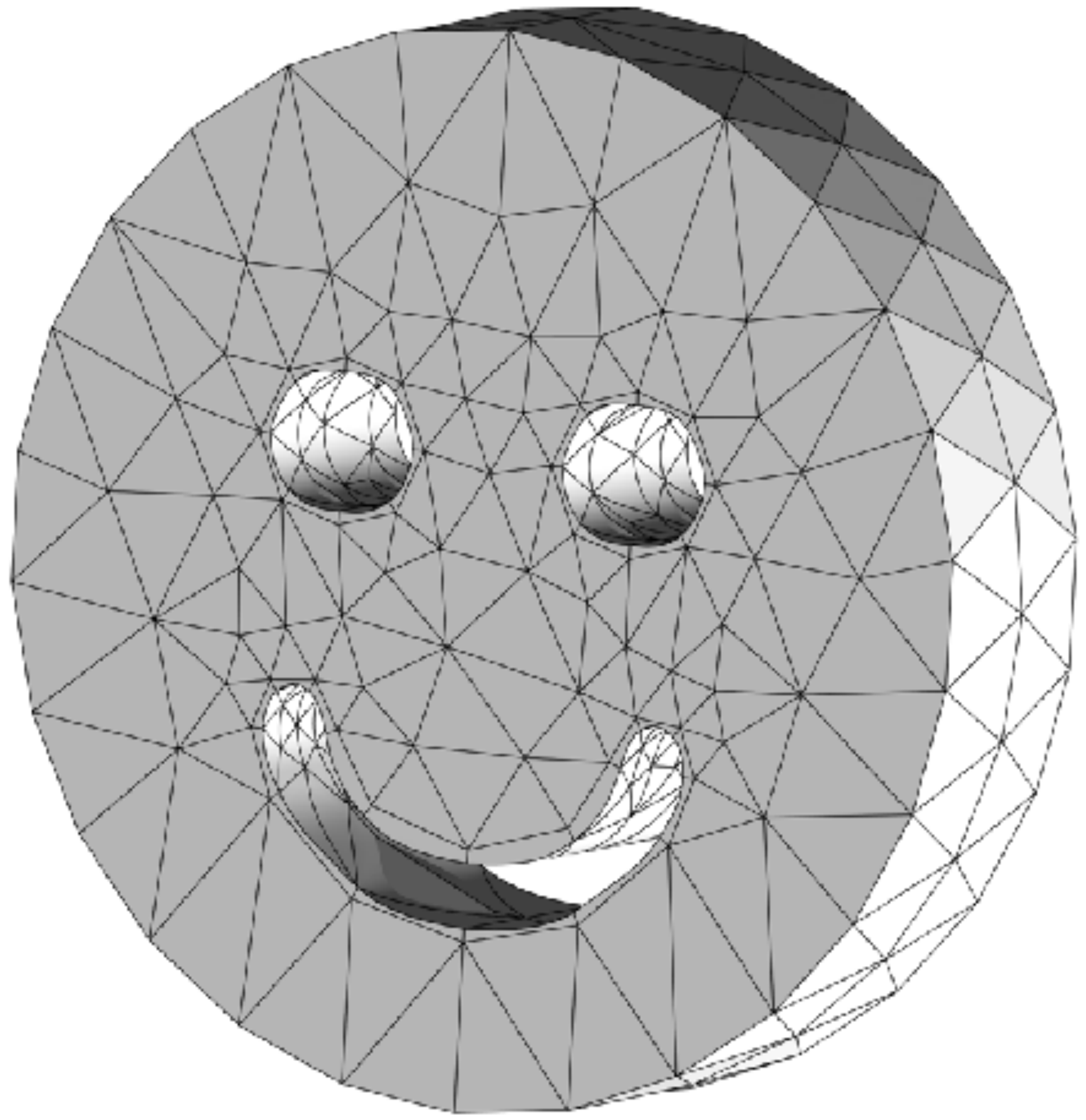
Design 3: +270% Downforce

Moxey, Turner, Jassim, Taylor, Peiro & Sherwin

Summary

- We can certainly spectral/*hp* element techniques to challenging industrial flow problems and succeed!
- Accurate, transient flow modelling is an **enabling technology** for high-end engineering/physics.
- But… there is still a way to go yet!
  - Meshing for 3D geometries is a specialist skill.
  - Robustness still requires more analysis.

Thanks for listening!



https://davidmoxey.uk/ @davidmoxey

d.moxey@exeter.ac.uk

www.nektar.info