High-order methods and their implementation in Nektar++

David Moxey, Department of Engineering, King's College London
Chris Cantwell, Department of Aeronautics, Imperial College London

UKAEA Nektar++ Training Workshop, 1st August 2022

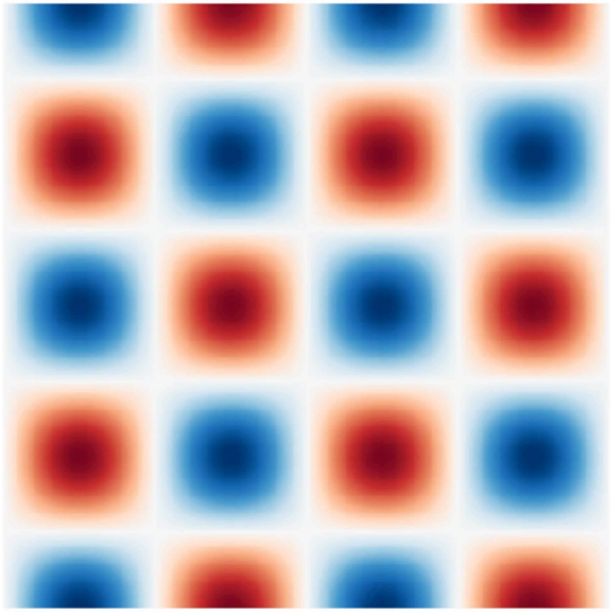
Outline & agenda

- This morning (10am-12pm): this talk
    - Overview of high-order finite element methods.
    - Outline of their high-level implementation within Nektar++.
- Meant to be a dynamic presentation; please interrupt if you have questions!
- This afternoon:
    - 1:30pm - 3:30pm: NEPTUNE-specific talks (particles, review and overview of our proxyapps to date)
    - 3:30pm - 4:00pm: more hands-on demo from Bin Liu & Mashy Green: need to install docker if you don't have it
- Dinner this evening? Let Wayne/us know by lunchtime today...

High-order methods

How do we model a physical system?

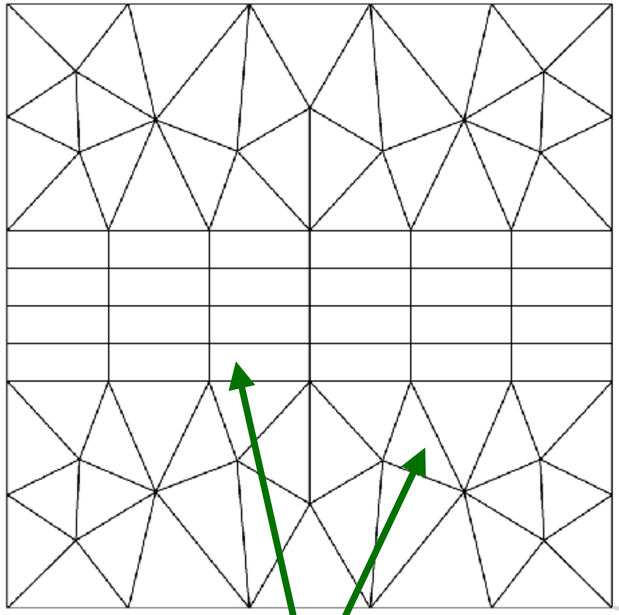
form underlying equations



advection velocity **v**

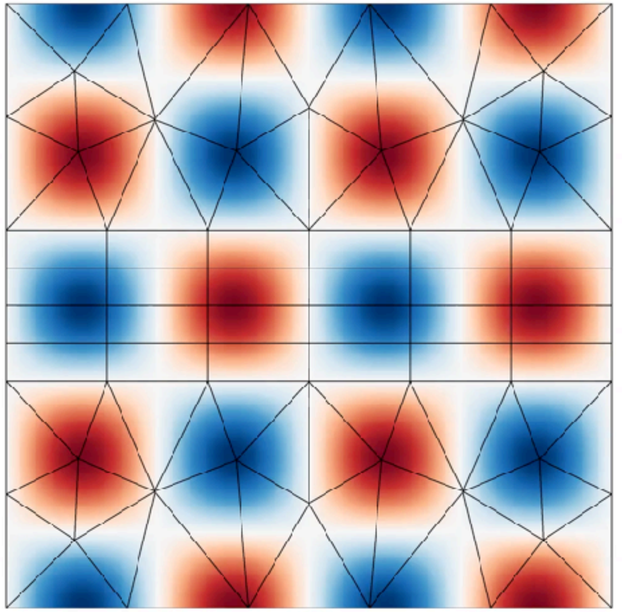
$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u = 0$$

discretise a typically complex domain



elements on which the equations can be solved

solve resulting system



$$\mathbf{Ax} = \mathbf{b}$$

(+ timestepping)

What is a spectral/*hp* element?

Consider an approximation to a functions in terms of modes ϕ_n.

$$u^{\delta}(x) = \sum_{n=1}^{N} \hat{u}_n \phi_n(x)$$

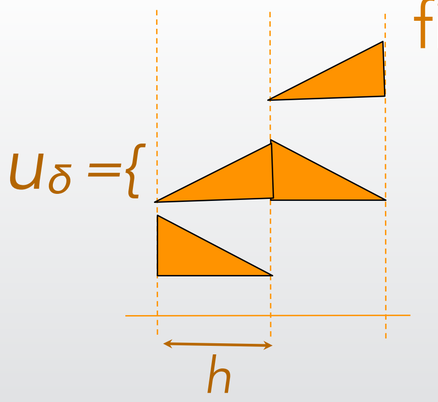
spatial flexibility (*h*)

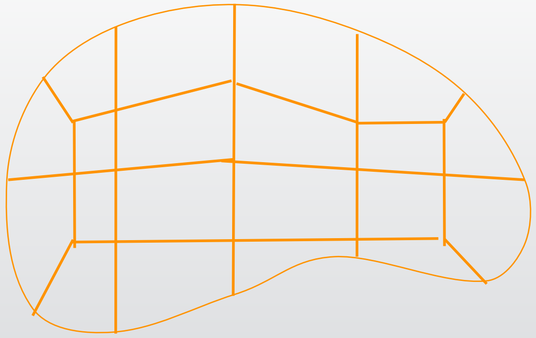
+

accuracy (*p*)

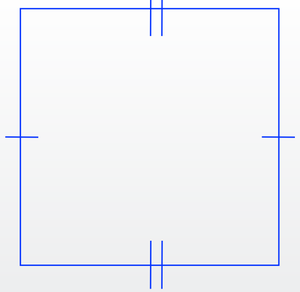
→ **spectral/hp element**

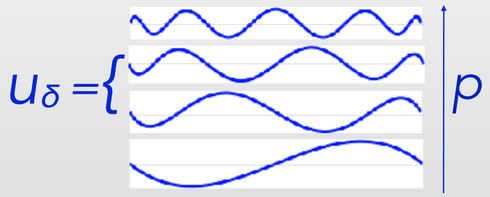
finite element method

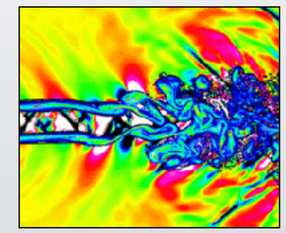




spectral method

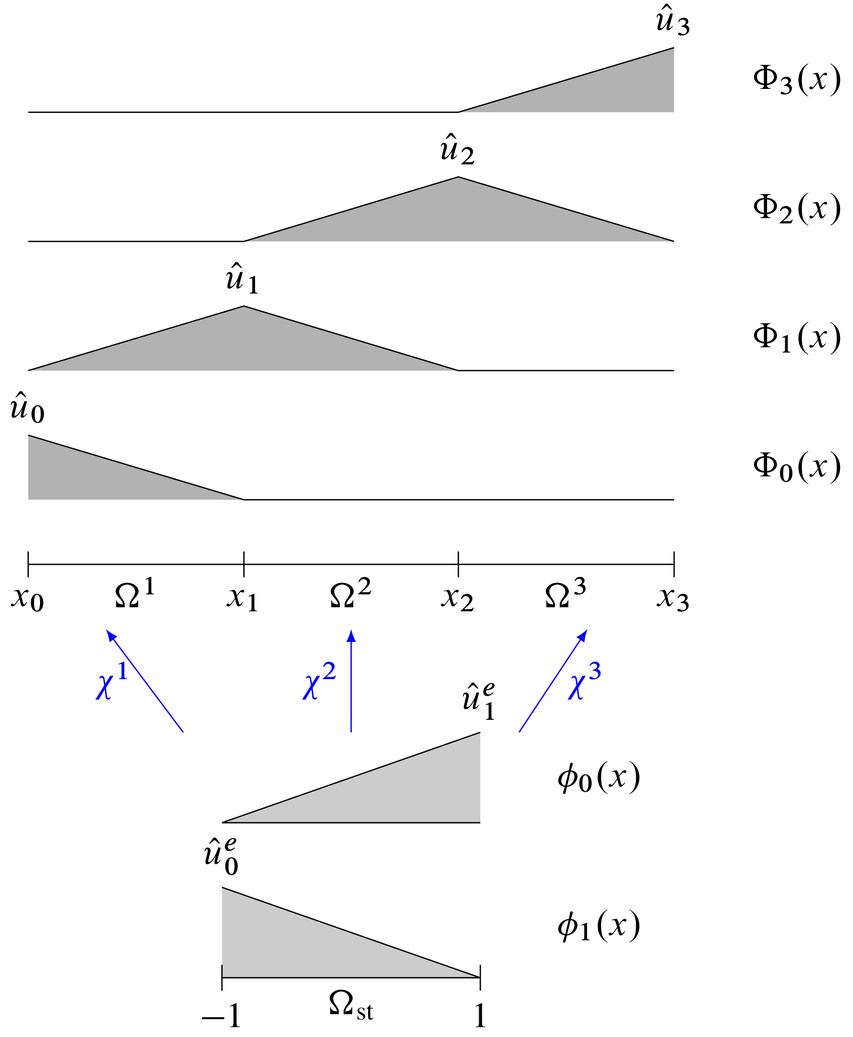




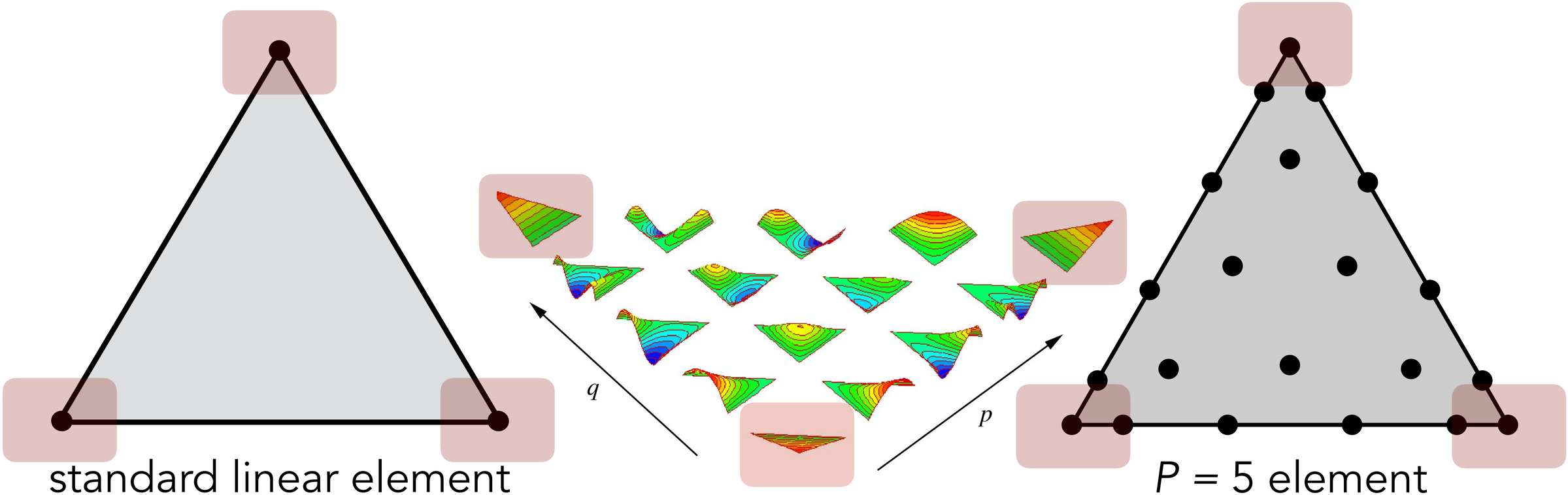


Traditional finite elements

- In traditional linear FEM, the modes that represent functions are given by ‘hat’ functions.
- Linear interpolation then occurs between the vertices of each element.
- However to avoid need for modes in every element, we can define them instead inside of a standard element.
- Each element's modes are given by mapping χ into world-space element.
- For a **high-order element**, we increase the order of the polynomial space that's being used.



Anatomy of a spectral/*hp* element



standard linear element

3 degrees of freedom

one per vertex (or cell)

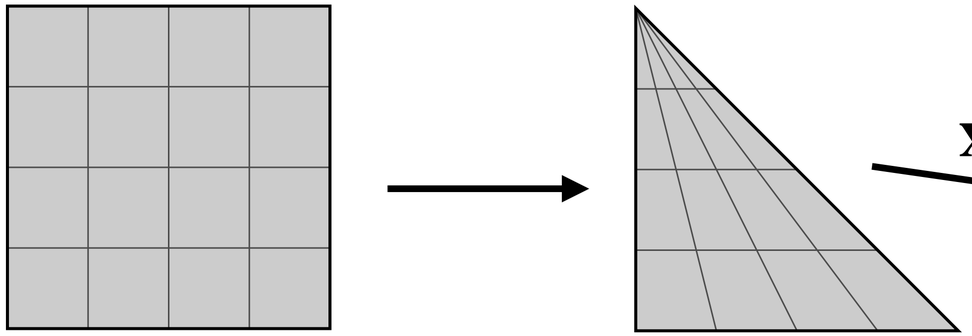
hierarchical basis

$P = 5$ element

21 degrees of freedom

vertex, edge and interior ‘modes’

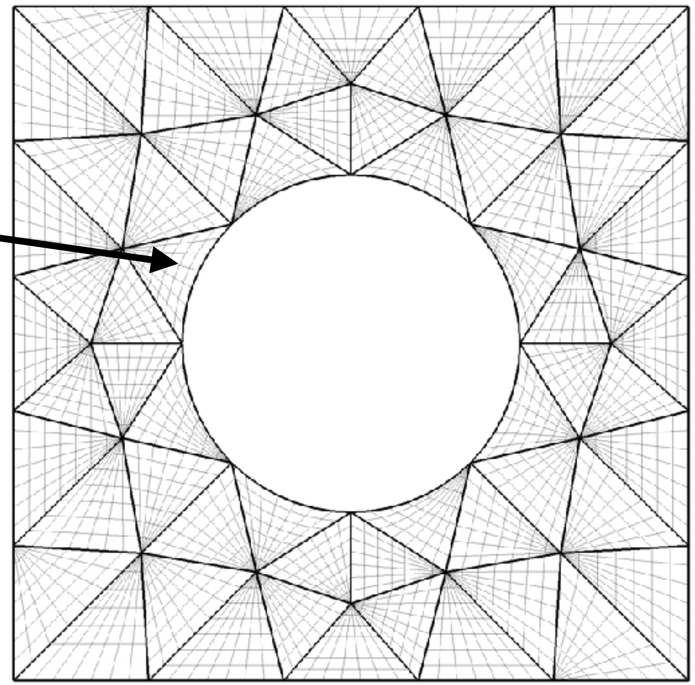
Spectral/*hp* element methods



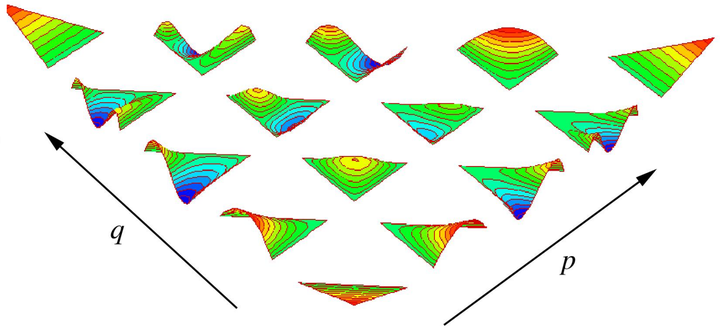
collapsed coordinates
$(\eta_1, \eta_2) \in [-1,1]^2$

reference element
$(\xi_1, \xi_2) \in \Omega_{\text{st}}$

$\mathbf{x} = \chi^e(\xi)$

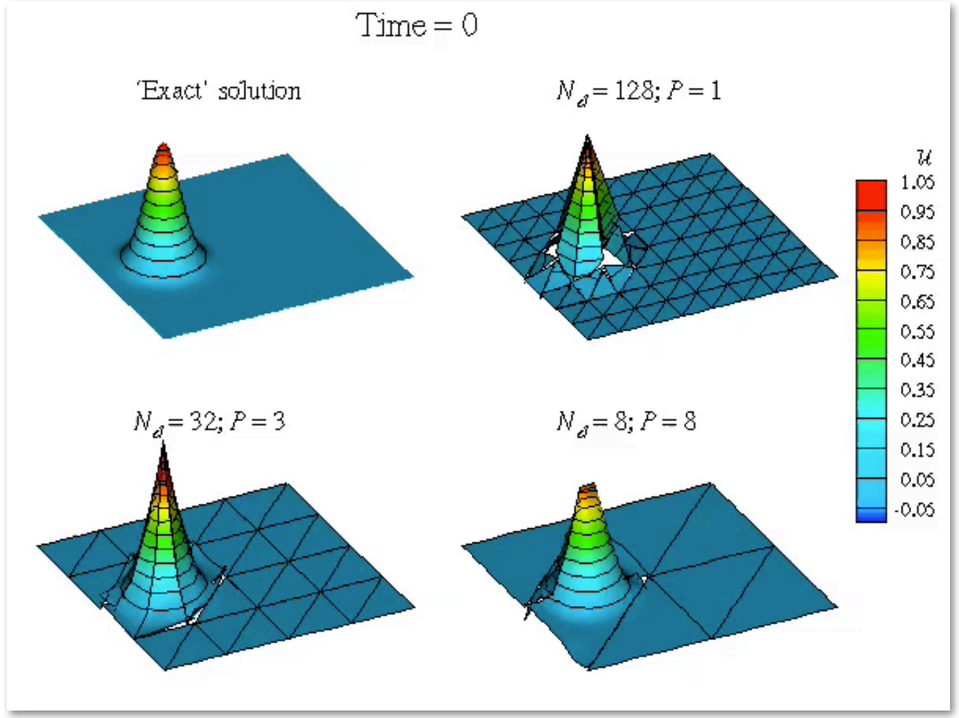


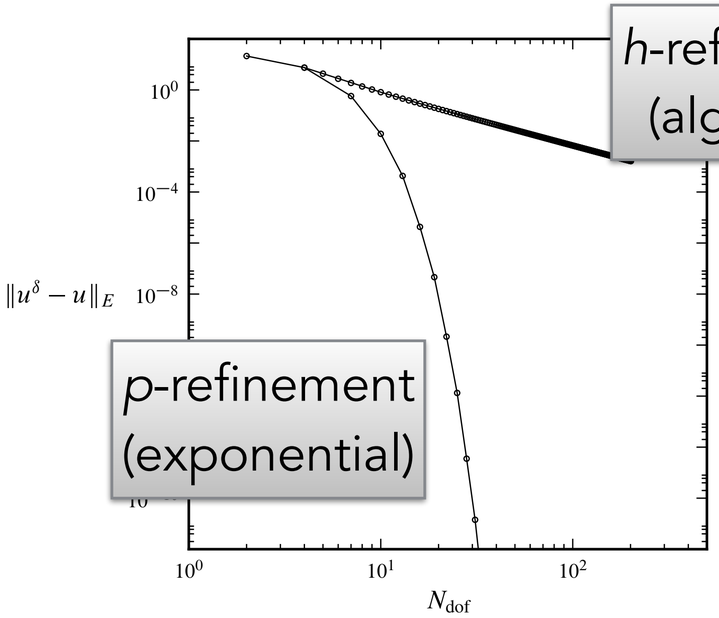
(C⁰) tensor product basis
$\phi_p^a(\eta_1)\,\phi_{pq}^b(\eta_2)$

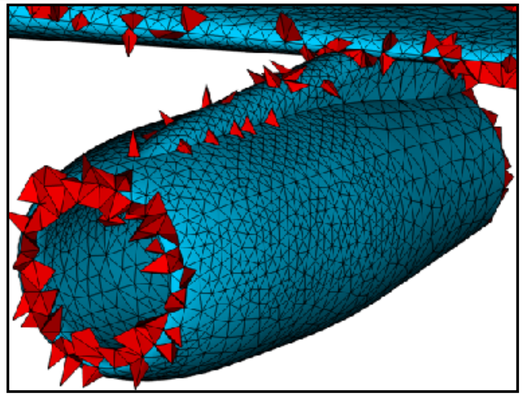


Why use a high-order method?

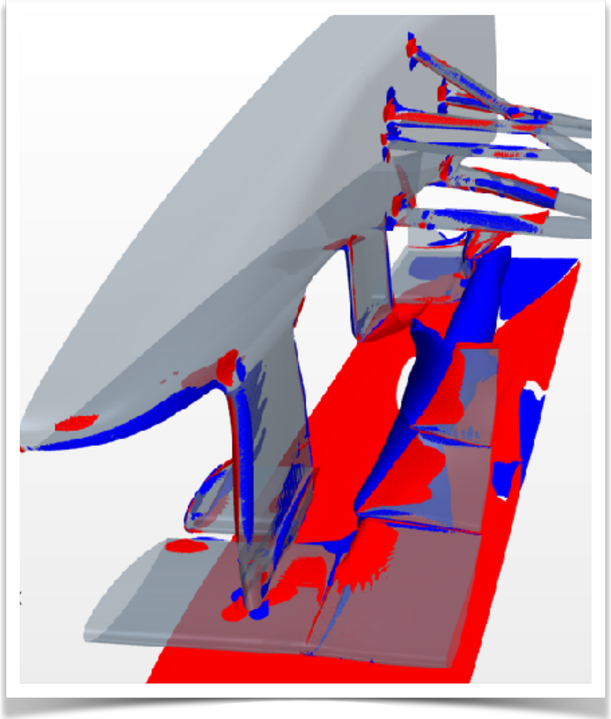
- ✓ error decays *exponentially* (smooth solutions);
- ✓ favorable diffusion & dispersion characteristics;
- ✓ model complex domains
- ✓ computational advantage: reduced memory bandwidth, better use of hardware.



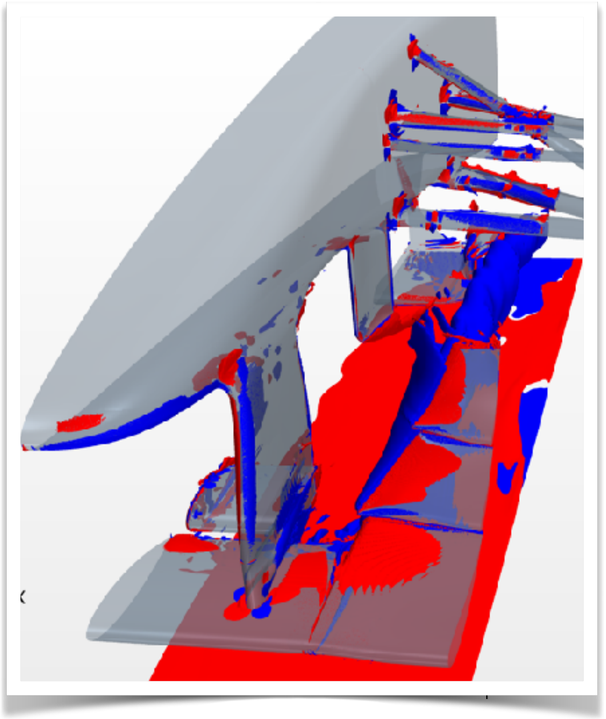




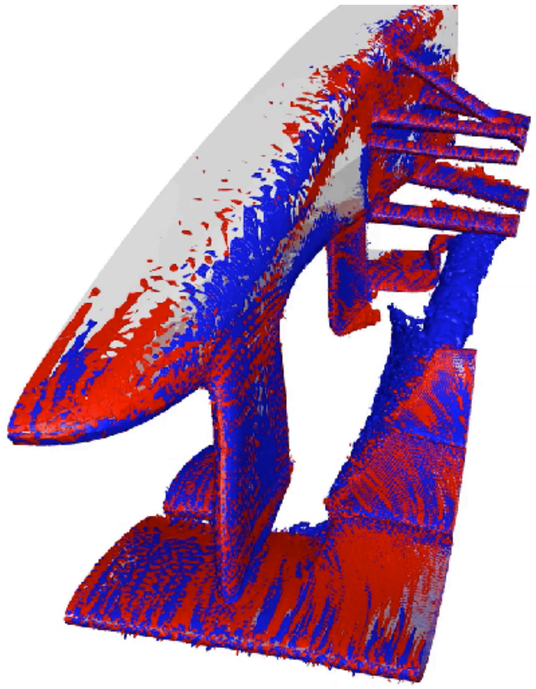
Next generation modelling



Reynolds Averaged Navier-Stokes

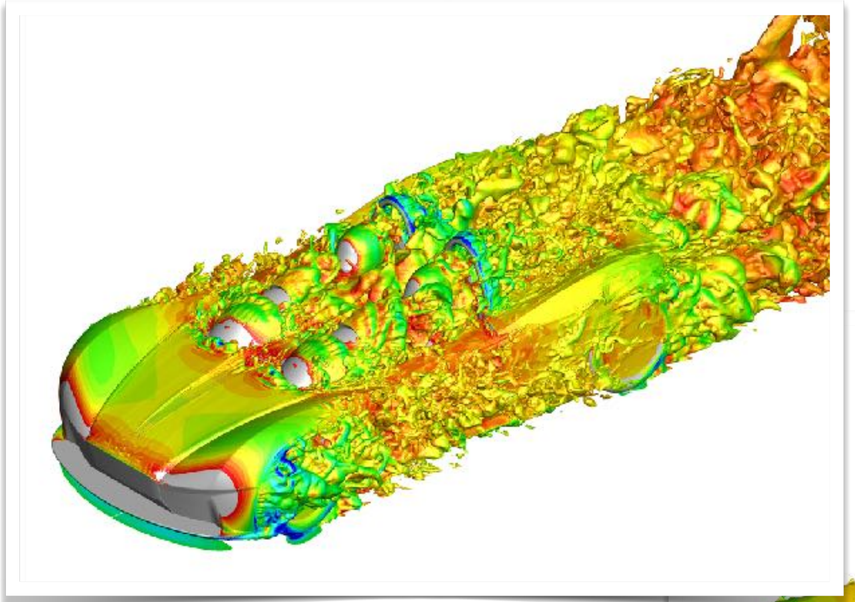


Detached Eddy Simulation

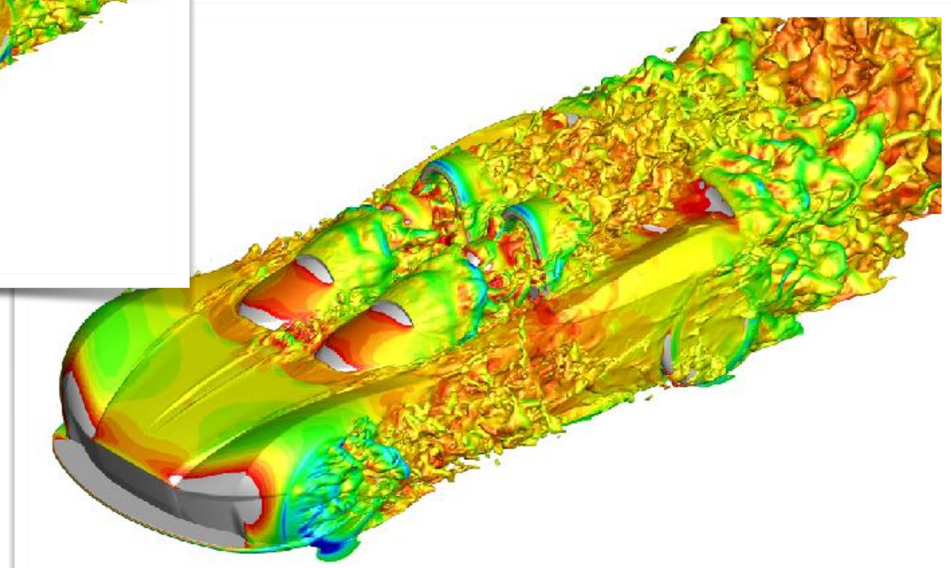


under-resolved Direct Numerical Simulation

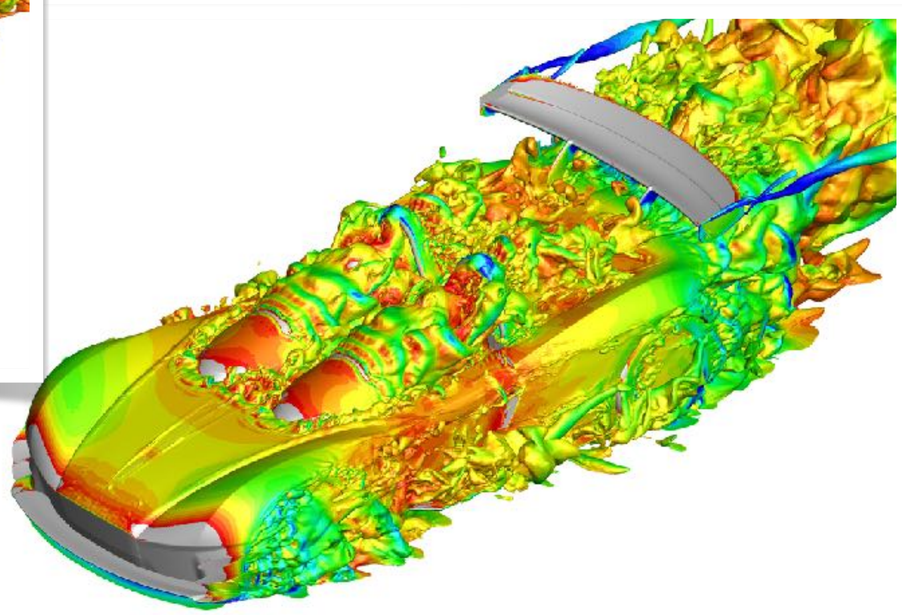
Elemental road race car



5th order
Re = 1m



Design 2: +33% Downforce



Design 3: +270% Downforce

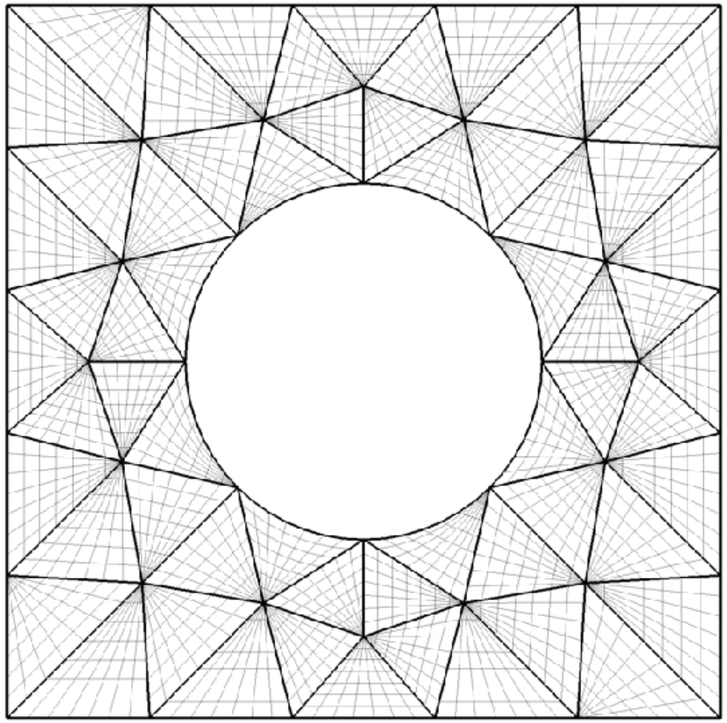
Moxey, Turner, Jassim, Taylor, Peiro & Sherwin

Continuous formulations

The lingo of high-order elements

- Different high-order variants depending on which polynomial basis you're using, and how you bolt the elements together.
  - **spectral element:** typically uses Lagrange interpolants $\ell_p(x)$
  - **hp-FEM:** automatic *hp* adaptivity, typically with quad/hex meshes and hanging node constraints; seems to be used more in structural mechanics applications.
  - **discontinuous Galerkin (DG):** Galerkin approach where continuity between elements is removed; recovers finite volume methods at $P = 0$.
  - **flux reconstruction:** similar to DG, but uses correction functions to enforce continuity of fluxes between elements.
  - ...and many more!

Classical C^0 finite elements



domain Ω

boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$

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

Boundary conditions

$u(x) = g_D(x)$ Dirichlet, $\partial\Omega_D$

$\frac{\partial u}{\partial n} = g_N(x)$ Neumann, $\partial\Omega_N$

Classical C^0 finite elements

$$\mathscr{U} = \{u \mid u|_{\Omega^e} \in \mathscr{P}_N(\Omega^e), u|_{\partial\Omega_D} = g_D\}$$
$$\mathscr{V} = \{v \mid v|_{\Omega^e} \in \mathscr{P}_N(\Omega^e), u|_{\partial\Omega_D} = 0\}$$

Choose test and trial spaces (C^0)

$$\int_\Omega v\nabla^2 u\, dx - \int_\Omega \lambda u\, dx = -\int_\Omega fv\, dx$$

Multiply by test function and integrate

$$\int_{\partial\Omega_N} v\frac{\partial u}{\partial n}\, ds - \int_\Omega \nabla u \nabla v\, dx$$

Apply the divergence theorem

$$u = u^D + u^H$$

Lift Dirichlet solution

$$\int_\Omega (\nabla u \nabla v + \lambda u) dx = \int_\Omega fv\, dx + \int_{\partial\Omega_N} vg_N ds$$

Arrive at the weak form

Into matrix form

- In this form we can substitute expansions from the test and trial spaces to arrive at a matrix problem.

$$u^{\delta}(x) = \sum_{n=1}^{N} \hat{u}_n \phi_n(x) \xrightarrow{\text{substitute}} (\mathbf{L} + \lambda \mathbf{M})\hat{\mathbf{u}} = \hat{\mathbf{f}}$$

$$\mathbf{M}_{ij} = \int \phi_i(x)\phi_j(x)\,dx \qquad \mathbf{L}_{ij} = \int \nabla\phi_i(x)\,\nabla\phi_j(x)\,dx \qquad \hat{\mathbf{f}}_i = \int f(x)\,\nabla\phi_i(x)\,dx$$

mass matrix | Laplacian matrix | forcing inner product

- To compute these matrices we split them up element by element, and then assemble across elements using a sparse assembly operator $\mathscr{A}$.

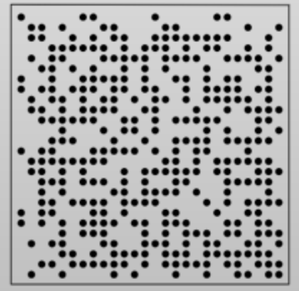
- Also require sensible choice of quadrature.

Solving the weak form

Solving the resulting matrices can be done in a number of ways, depending on the size of the mesh.

Smaller problems

- Use a *direct* approach
- Form large SPD matrix
- Solve using e.g. LAPACK (Cholesky factorisation)
- Typically done in serial



Larger problems

- Use an *iterative* approach
- No global matrix formed: only **local** matrices
- Symmetric problems solved using e.g. PCG
- Done in parallel

$\mathscr{A}$ + comms.

Some numerical considerations

Direct approach

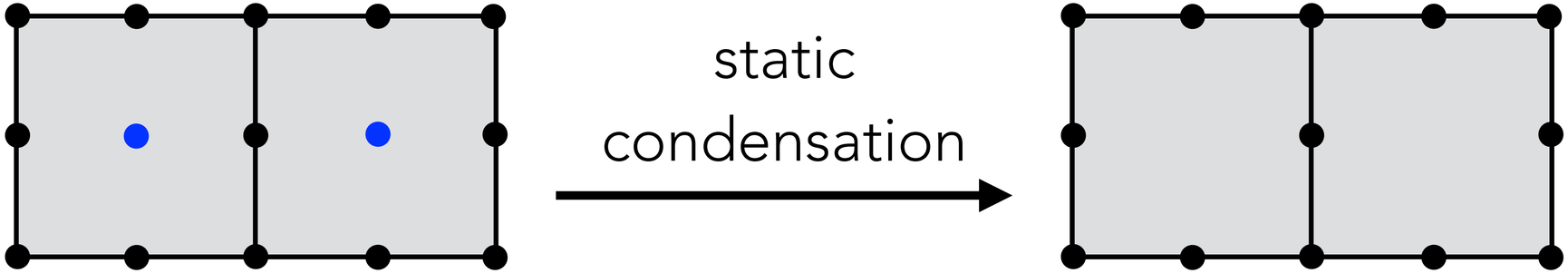
- Bandwidth of the system - use a reordering, e.g. reverse Cuthill-McKee
- Whether the problem is too big to fit in memory/be solved in a reasonable time
- Nearly all 3D problems are too large to solve this way

Iterative approach

- Generally slower than using a direct solver
- Performance heavily governed by iteration count
- Needs effective preconditioning, particularly if $\lambda = 0$
- Effective communication between elements needed in parallel

Evaluation costs

- In iterative solvers, most cost comes from evaluation of finite element operator.
- In general, local matrix is rank $O(P^{2d})$
- Two strategies to reduce this: **static condensation** and **matrix-free** evaluation.



$$H \rightarrow \begin{bmatrix} H_{bb} & H_{ib} \\ H_{bi} & H_{ii} \end{bmatrix}$$

Rank of $H_{bb} = \mathcal{O}(P^{2(d-1)})$

Discontinuous formulations

Hyperbolic equations

- Consider the hyperbolic conservation law

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{u}) = \mathbf{0} \qquad \mathbf{F}(\mathbf{u}) = \left[\mathbf{F}_1(\mathbf{u}), \mathbf{F}_2(\mathbf{u})\right] \text{ in 2D}$$

- This encapsulates a wide range of equations: e.g. compressible Euler or Navier-Stokes equations.

- Can apply C^0 methods to this, but perhaps not the most natural method and some concerns around e.g. local conservation properties between elements.

- Discontinuous Galerkin methods very popular in this setting: have several advantages including favourable stability properties.

Discontinuous Galerkin method

Consider Galerkin approach on a single element, but remove the continuity of functions between elements. This leads to the formulation

$$\int_{\Omega^e} \mathbf{v}\frac{\partial \mathbf{u}}{\partial t}\,dx + \int_{\Omega^e} \mathbf{v}\nabla\cdot\mathbf{F}(\mathbf{u})\,d\mathbf{x} = 0$$

integration by parts

$$\int_{\Omega^e} \mathbf{v}\frac{\partial \mathbf{u}}{\partial t}dx - \int_{\Omega^e} \nabla\mathbf{v}\nabla\cdot\mathbf{F}(\mathbf{u})\,dx + \oint_{\partial\Omega^e} \mathbf{v}\cdot\hat{\mathbf{F}}(\mathbf{u}^+,\mathbf{u}^-,\mathbf{n})ds = 0$$

this numerical flux appears because *u* is discontinuous across element boundaries

Discontinuous Galerkin method

- The discretisation gives the semi-discrete ODE system on each element:

$$\mathbf{M}^e \frac{\mathrm{d}\hat{u}^e}{\mathrm{d}t} = \sum_{d=1}^{D} (\mathbf{D}_d^e \mathbf{B}^e)^\top \mathbf{W}^e F_d(u) - \mathbf{b}^e$$

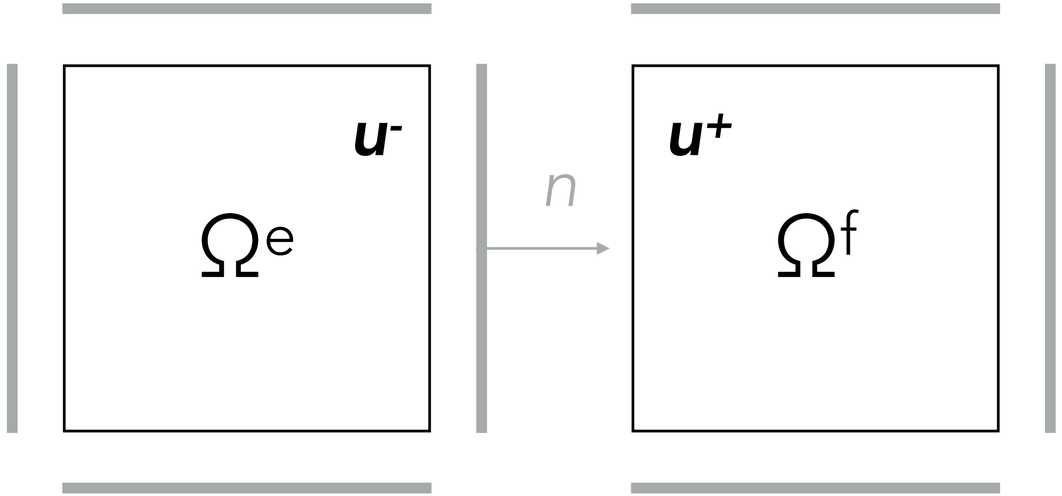
inner product w.r.t. derivative of basis

boundary flux contributions

- Note that elements aren't decoupled: boundary flux couples them together.
- Can now apply time-stepping schemes: e.g. forward Euler, Runge-Kutta, Adams-Bashforth, ...

Discontinuous Galerkin method

- Need some consistent definition of which element is 'forward' and 'backward'
- Can then employ the usual machinery from finite volume methods:
  - e.g. for simple transport problems we can upwind based on forward and backward spaces
  - More complex Riemann solvers for nonlinear problems such as compressible N-S equations
- Other stabilisation techniques such as slope limiters also apply in this setting.



trace or skeleton space

advection velocity *v*

$$F(u^+, u^-, n) = \begin{cases} u^-, & v \cdot n \geq 0, \\ u^+, & v \cdot n < 0 \end{cases}$$

Elliptic operators in DG

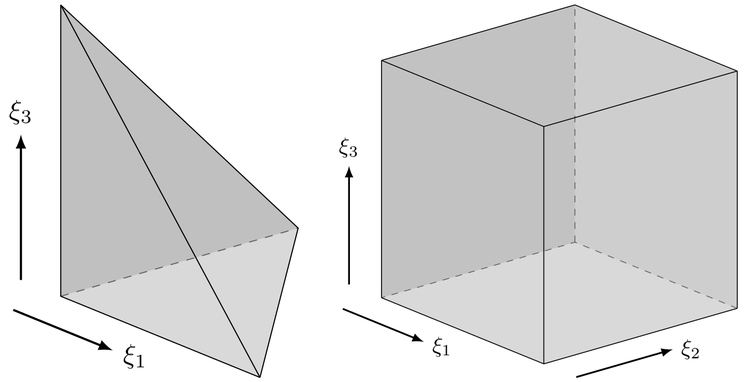
- Elliptic operators require careful consideration to ensure stability.
- For example in the Poisson equation $-\nabla^2 u = f$, we introduce an auxiliary variable $\mathbf{q} = \nabla u$, so that we instead solve the system

$$\begin{cases} -\nabla \cdot \mathbf{q} = f \\ \mathbf{q} = \nabla u \end{cases}$$

- Applying the DG formulation to the above equations leads to a weak form similar to that shown previously.
- A careful choice of fluxes for $\mathbf{q}$ in particular (e.g. local discontinuous Galerkin, BR2) ensures convergence and stability.

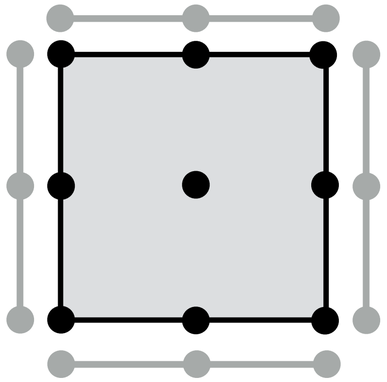
Hybridisable discontinuous Galerkin (HDG)

- An alternative is to consider static condensation in the formulation of DG.
- This leads to **HDG** where boundary matrices are formed for the trace space.
- Partially supported in Nektar++ but only for Helmholtz operator.



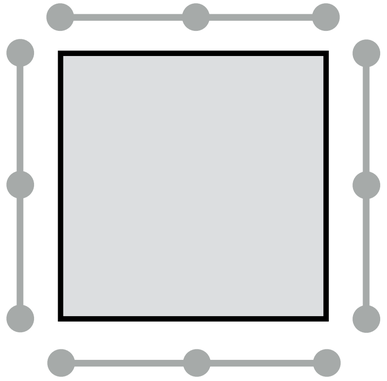
$\boldsymbol{q} = \nabla u$

$u = \mathrm{div}(\boldsymbol{q})$ on Ω



$u = \lambda$ on $\partial\Omega$

(static condensation)



A_λ	local boundary system
A_i	interior solve
$A_{\lambda \to u}$	project λ onto u

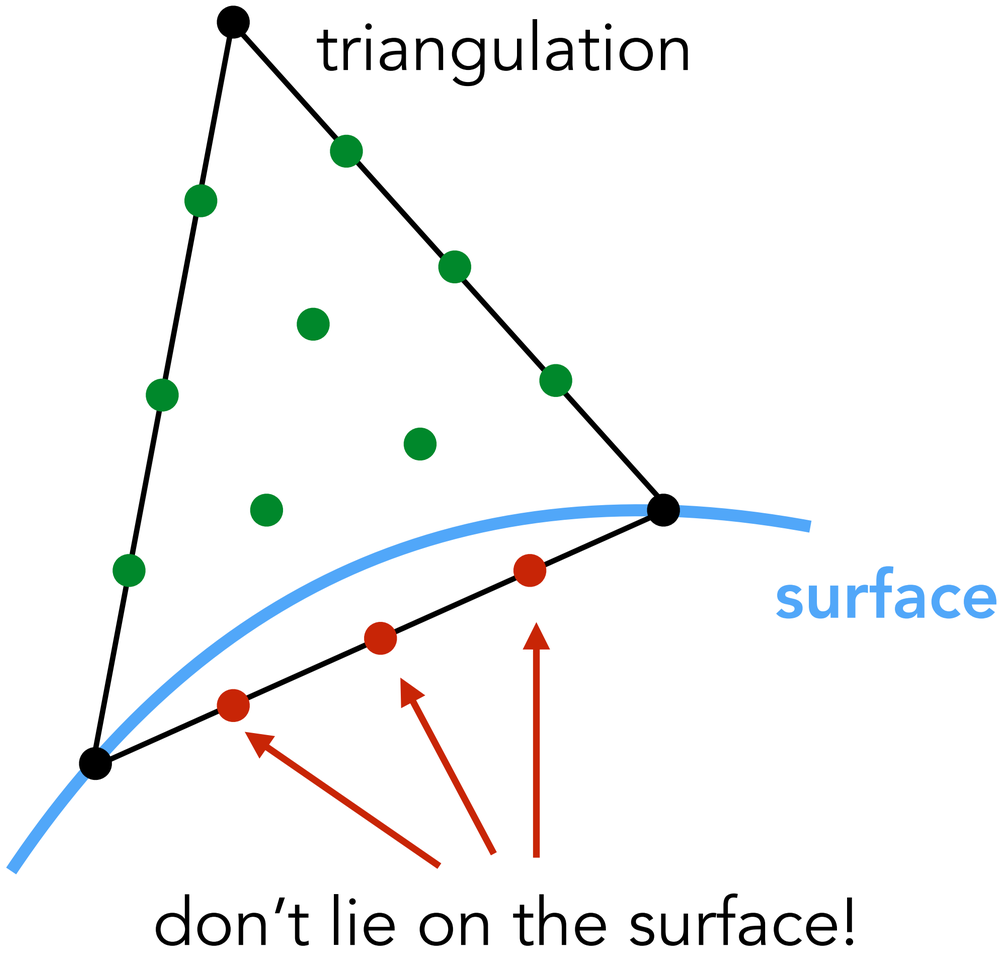
Challenges

The challenge of implementation

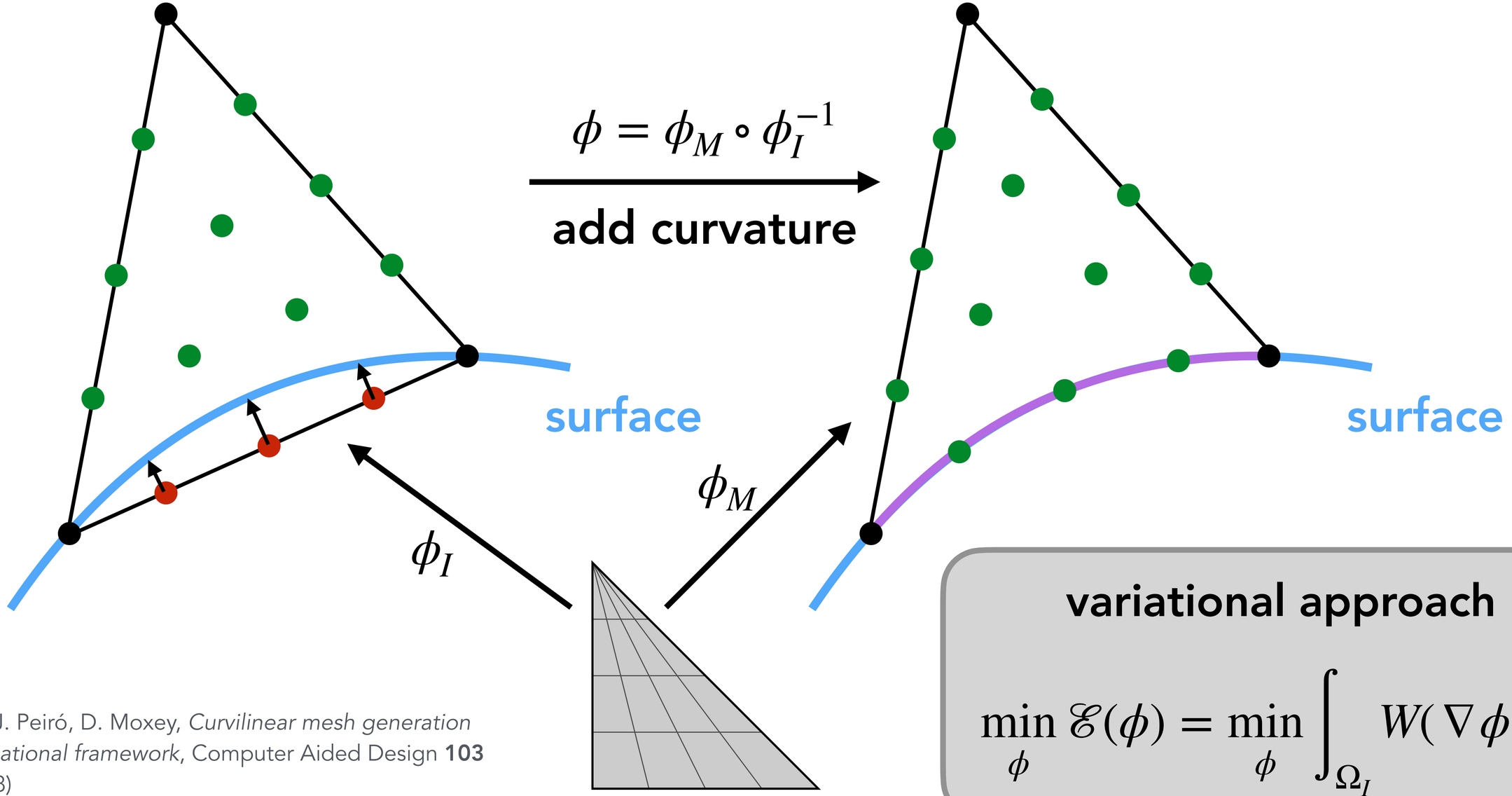
- High-order methods have potential to bring some nice numerical and computational benefits to bear on complex problems.
- Offer high(er) fidelity at equivalent or lower costs, as they have good implementation characteristics.
- However, one of the main barriers to using high-order methods is that they are **difficult to implement**.
- There are a number of other issues too including **stabilisation** and **mesh generation**.
- We'll touch on these briefly here.

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.
- This can easily cause self intersection, particularly when dealing with high-aspect ratio elements (e.g. in a fluid boundary layer).



High-order mesh generation

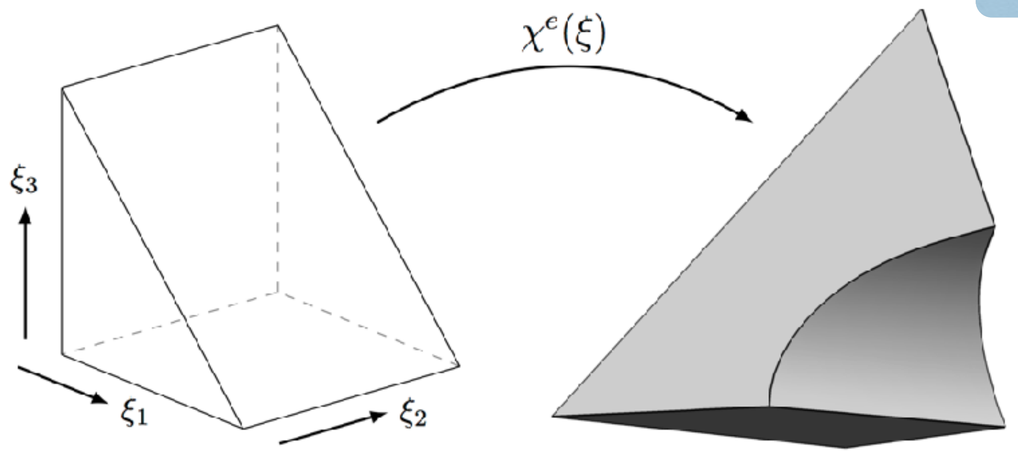


M. Turner, J. Peiró, D. Moxey, *Curvilinear mesh generation using a variational framework*, Computer Aided Design **103** 73-91 (2018)

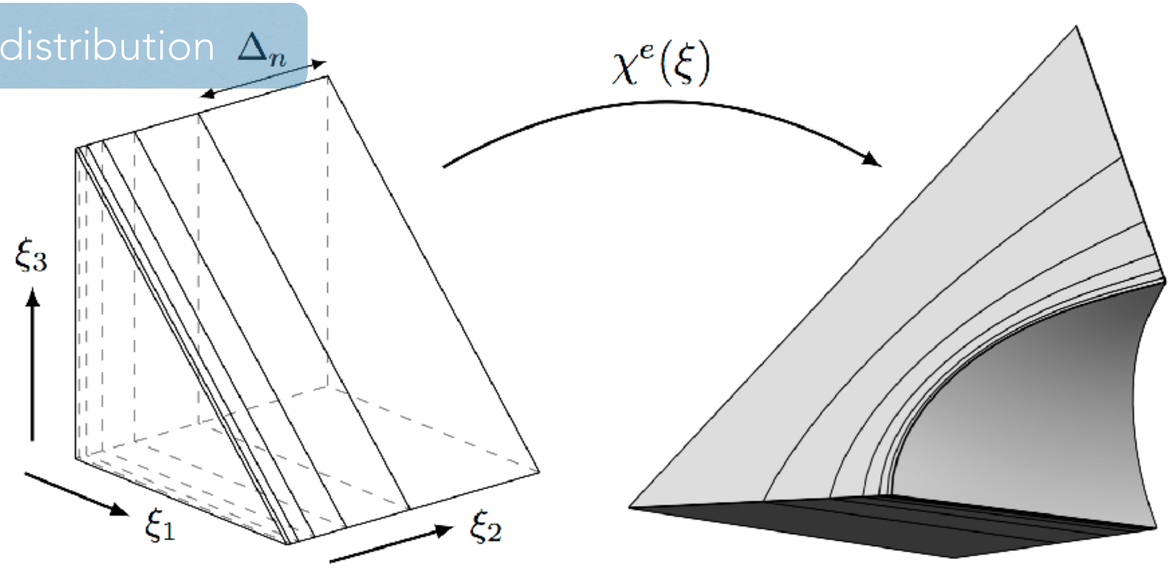
variational approach

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

Boundary layer meshing



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



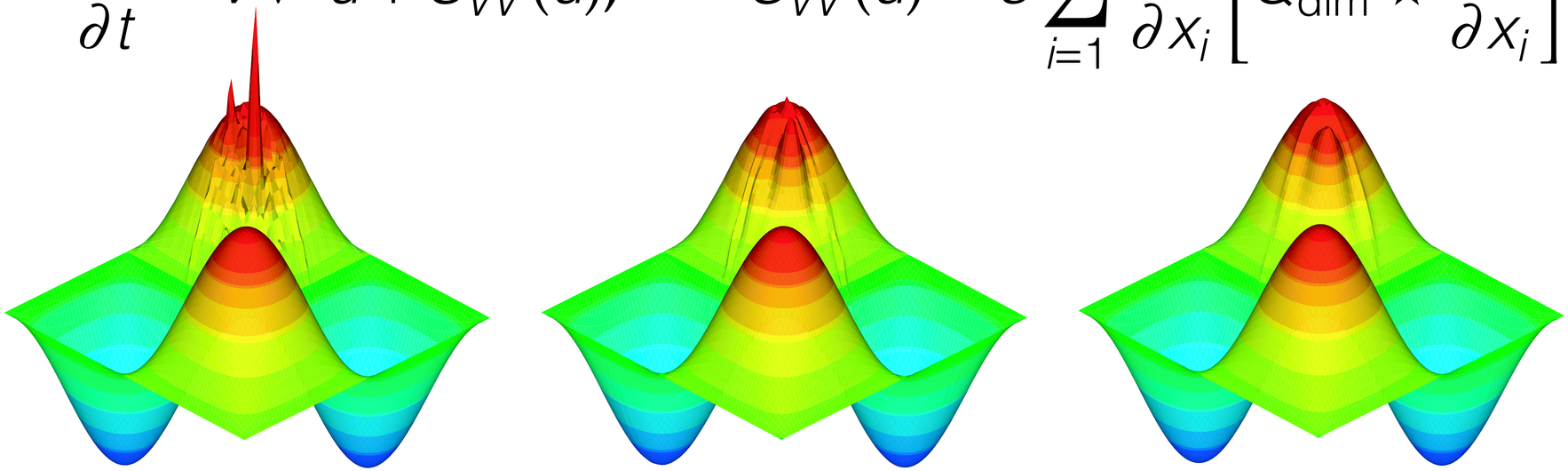
Subdivide the reference element in order to obtain a boundary layer mesh

An isoparametric approach to high-order curvilinear boundary-layer meshing
D. Moxey, M. Hazan, S. J. Sherwin, J. Peiró, under review in Comp. Meth. Appl. Mech. Eng.

Stabilisation

- Significant effort into stability analysis for under-resolved problems, much based around spectral vanishing viscosity (SVV):

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u + S_{VV}(u), \qquad S_{VV}(u) = \varepsilon \sum_{i=1}^{\dim} \frac{\partial}{\partial x_i} \left[Q_{\dim} \star \frac{\partial u}{\partial x_i} \right]$$



No SVV

$P_{cut} = 7,$

$\epsilon_{SVV} = 0.1$

$P_{cut} = 3,$

$\epsilon_{SVV} = 0.1$

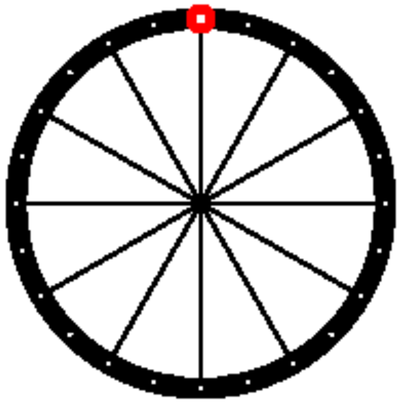
R. Kirby, S. Sherwin, *Comp. Meth. Appl. Mech. Eng.*, 2006

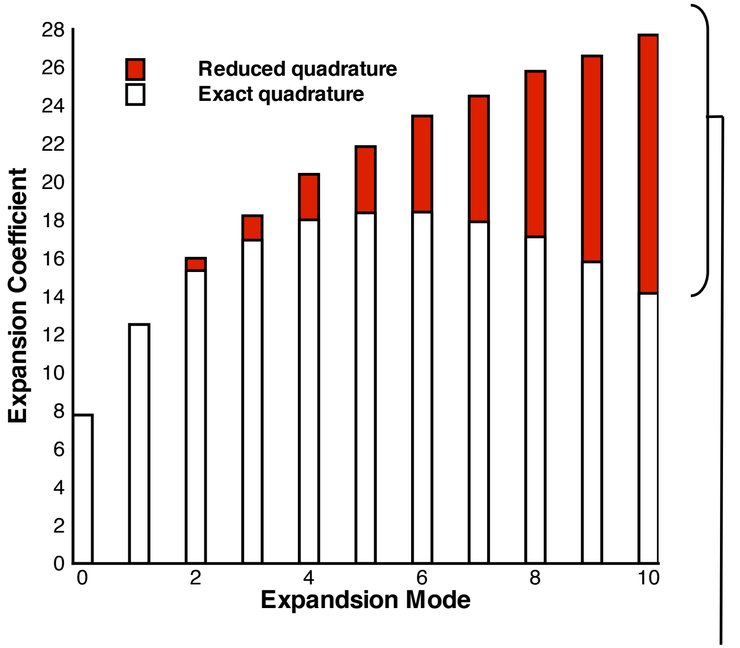
Stabilisation

- Large body of work more recently on emulating the stabilisation properties of DG for classical CG elliptic operators via eigenspectrum analysis.
- Naturally the DG method acts to dampen higher-order oscillations (and is one of the reasons that it is so popular for e.g. LES).
- See several references below!

- Mengaldo G, De Grazia D, Moura RC, Sherwin SJ, 2018, Spatial eigensolution analysis of energy-stable flux reconstruction schemes and influence of the numerical flux on accuracy and robustness, Journal of Computational Physics, 358, 1-20
- Mengaldo G, Moura RC, Giralda B, Peiró J, Sherwin SJ 2017, Spatial eigensolution analysis of discontinuous Galerkin schemes with practical insights for under-resolved computations and implicit LES, Computers and Fluids, 0045-7930
- Moura RC, Sherwin SJ, Peiro J, 2016, Eigensolution analysis of spectral/hp continuous Galerkin approximations to advection-diffusion problems: Insights into spectral vanishing viscosity, Journal of Computational Physics, 307, 401-422
- Moura RC, Sherwin SJ, Peiro J, 2015, Linear dispersion-diffusion analysis and its application to under-resolved turbulence simulations using discontinuous Galerkin spectral/hp methods, Journal of Computational Physics, 298, 695-710,

Other considerations: aliasing

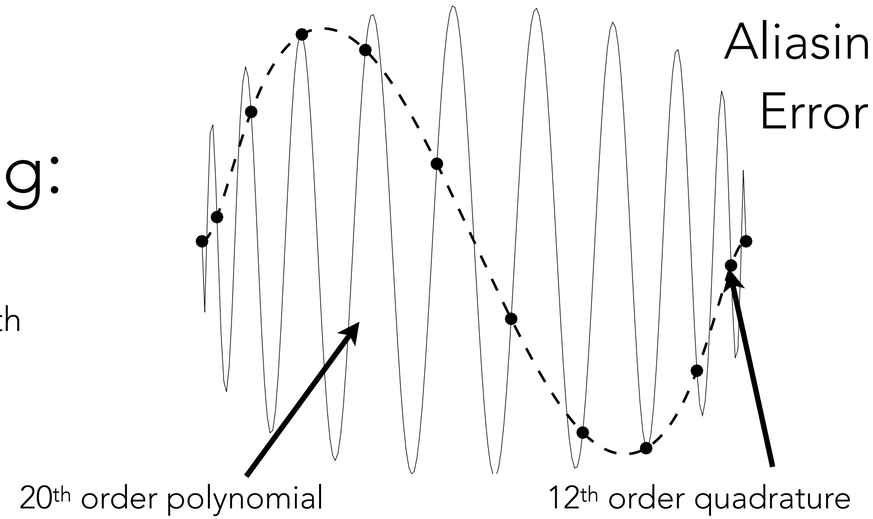




Example: $u(\xi) = \sum_{i=0}^{10} \phi_i(\xi)$

Galerkin projection of u^2 using:

- $Q = 17$ – exact Quadrature
- $Q = 12$ – sufficient for integrating 20^{th} degree polynomials



Example from Kirby & Karniadakis, J. Comp. Phys (2003)

Nektar++

spectral/hp element framework

High-fidelity numerical methods
Highly parallel, designed for unsteady problems

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.

Development team

Mike Kirby

Spencer Sherwin

Chris Cantwell

David Moxey

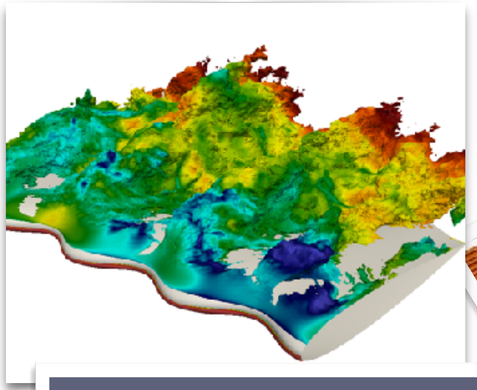
THE UNIVERSITY OF UTAH

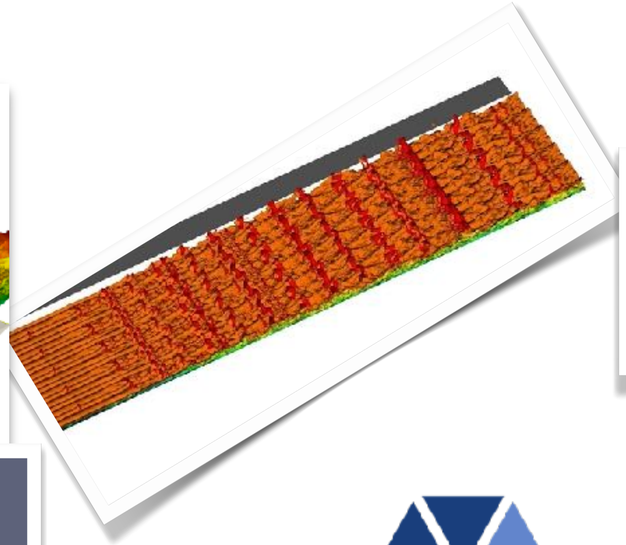
Imperial College London

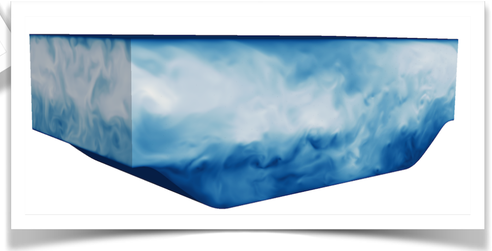
KING'S College LONDON

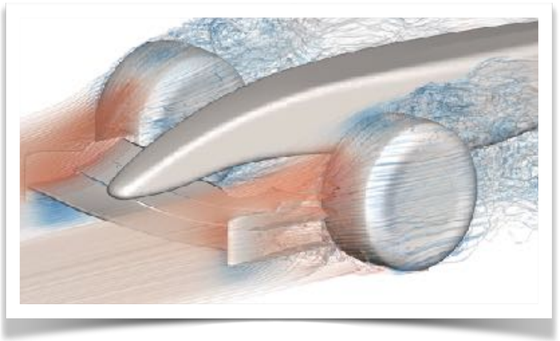
- **Project coordinators:** Joaquim Peiró, Gianmarco Mengaldo
- **Senior developers:** Giacomo Castiglioni, Ankang Gao, Mohsen Lahooti, Zhenguo Yan

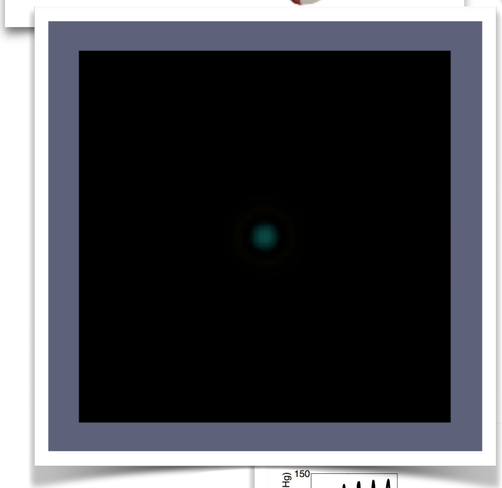
Some application areas



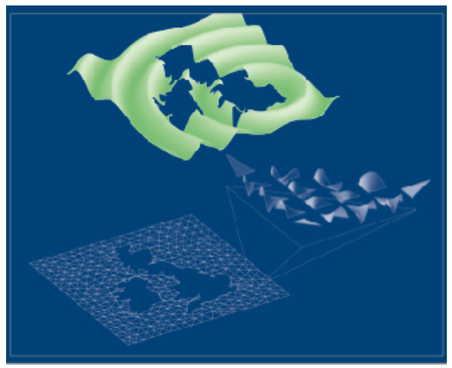


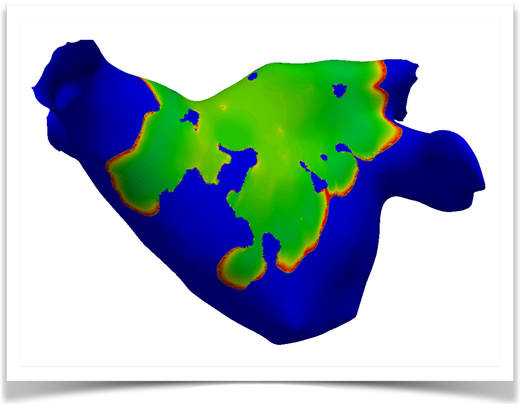


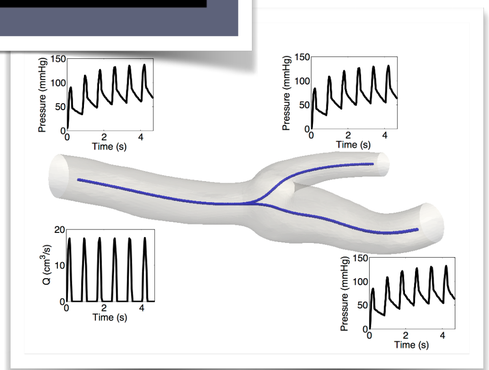


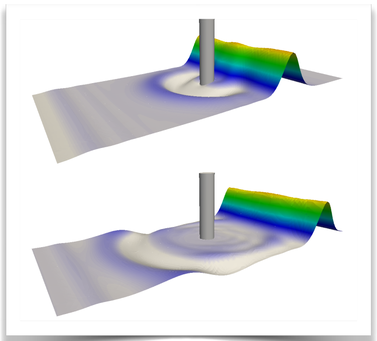


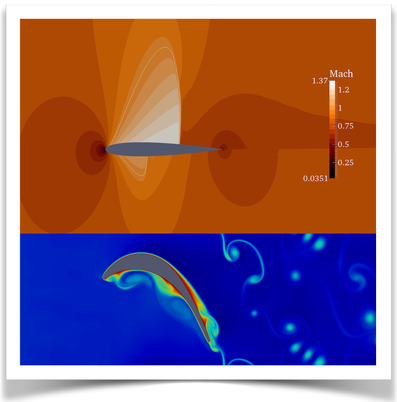
www.nektar.info

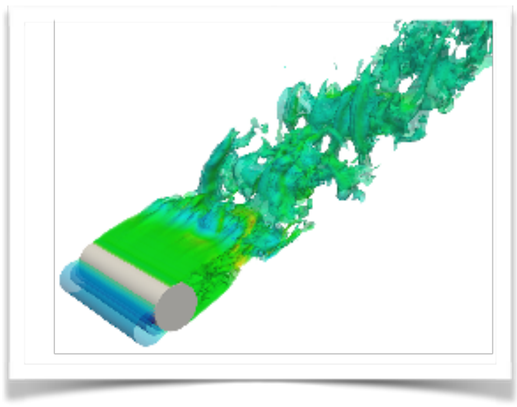








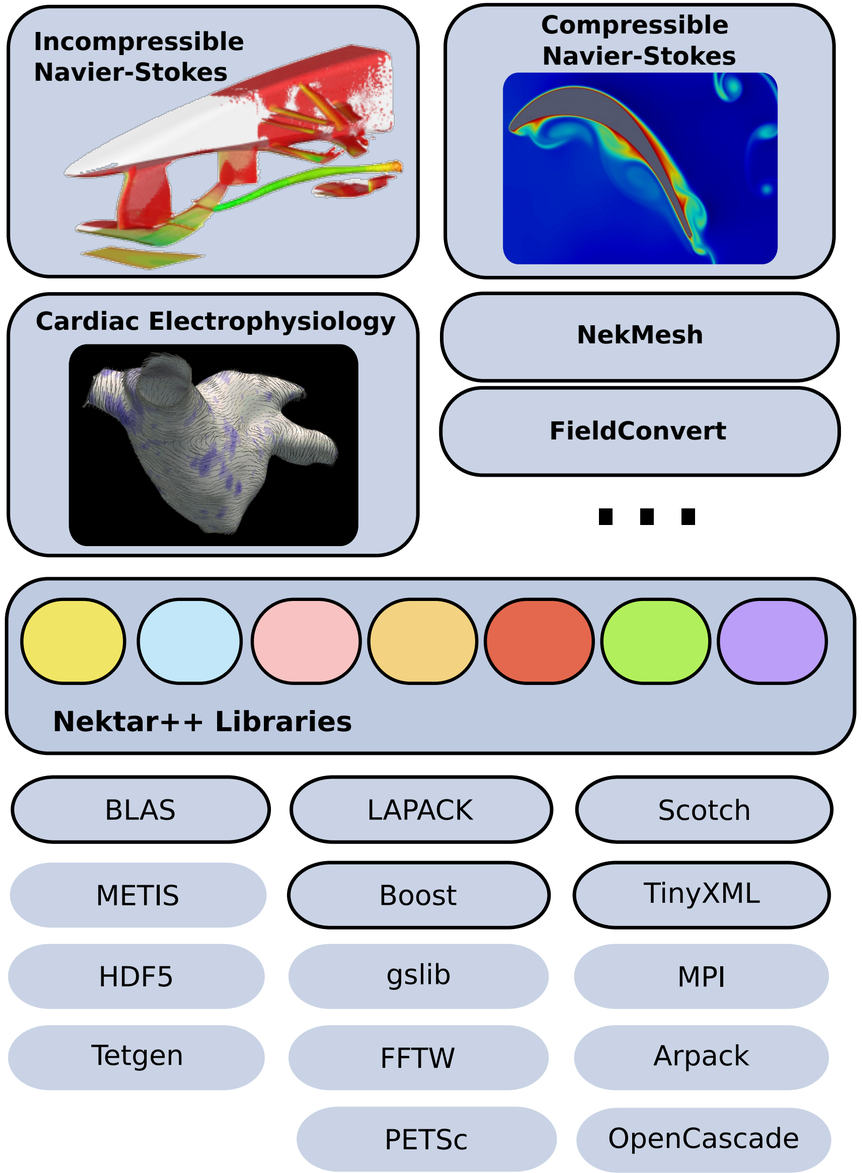




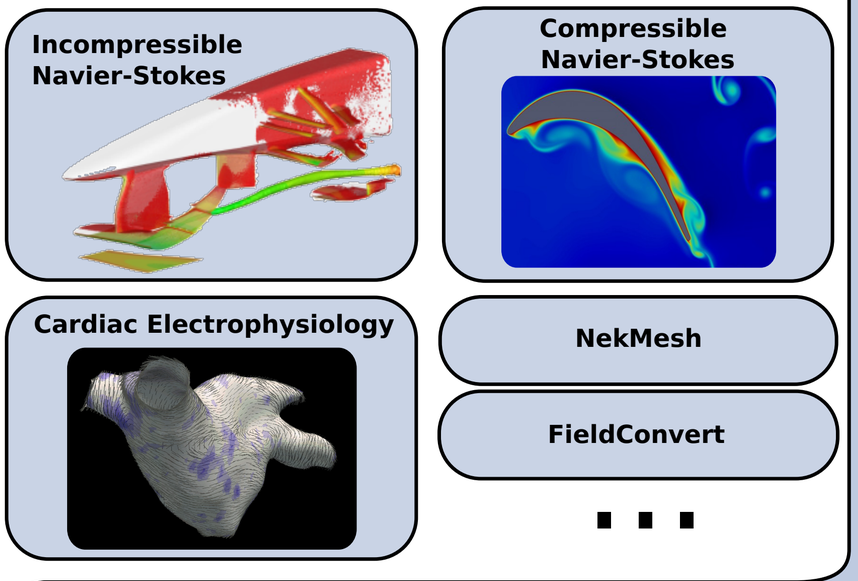
Design Approach

- **Challenge:** Large software project, many developers.
    - Hide algorithmic complexity.
    - Tie implementation to mathematics wherever possible.
    - Compartmentalise features / algorithms / data structures.
    - Choice of abstraction - *“there is more than one way to skin a cat”.*
- Coding paradigms: template pattern, factory pattern
    - Switch algorithms transparently
    - Aids design of suitable interfaces between components
    - Decoupling of code - faster build times, more robust

Library structure



Library structure



Operator Construction

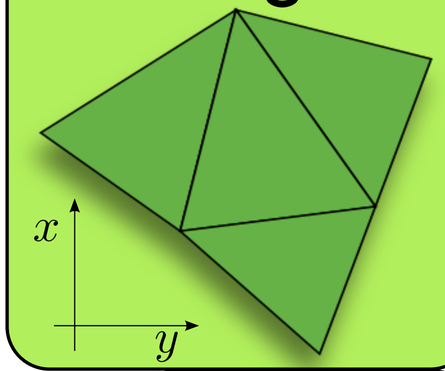
$$u^\delta = \sum_i \hat{u}_i \Phi_i(x)$$

$$u_e^\delta = \sum_p \hat{u}_p \phi_p(x)$$

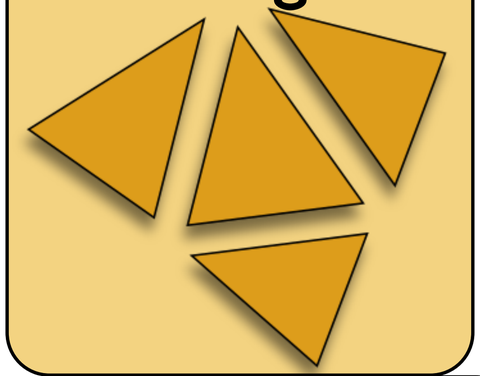
Helmholtz problem: $\nabla^2 u + \lambda u = f$

Weak form + IBP: $-(\nabla u, \nabla v) + \lambda(u, v) + (\nabla u, v)|_{d\Omega} = (f, v)$

MultiRegions



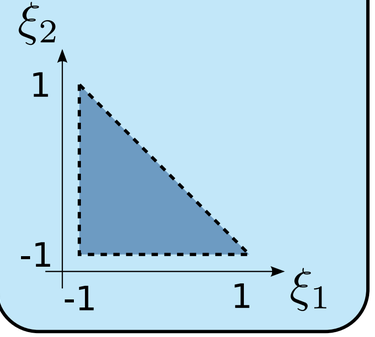
LocalRegions



SpatialDomains

$\chi^e(\xi)$

StdRegions



$$\mathbf{f}[i] = \int_\Omega \Phi_i(x) f(x) dx = \sum_e^{N^{el}} \sum_p \int_{\Omega^e} \phi_p(x) f(x) dx$$

$$= \sum_e \sum_p \int_{\Omega^e} \phi_p(\chi^e(\xi)) f(\chi^e(\xi)) J^e d\xi$$

Operator Construction

$$u^\delta = \sum_i \hat{u}_i \Phi_i(x)$$

$$u_e^\delta = \sum_p \hat{u}_p \phi_p(x)$$

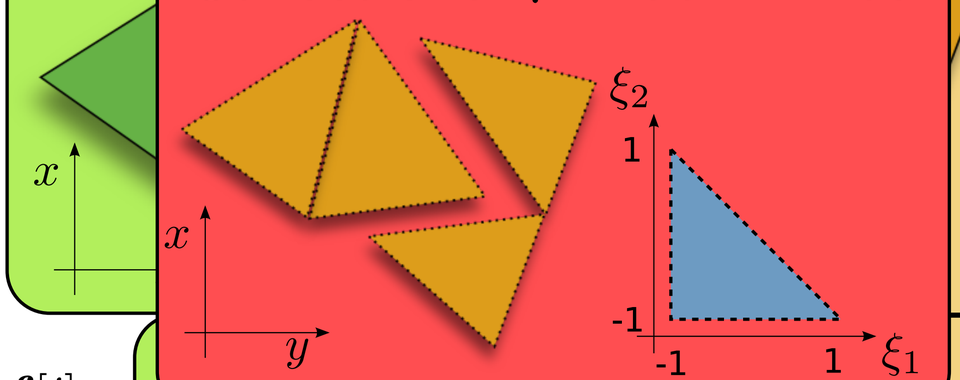
Helmholtz problem: $\nabla^2 u + \lambda u = f$

Weak form + IBP: $-(\nabla u, \nabla v) + \lambda(u, v) + (\nabla u, v)|_{d\Omega} = (f, v)$

MultiRegions

LocalRegions

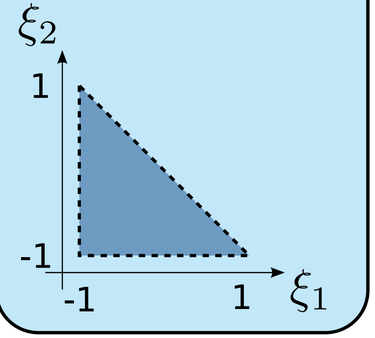
Collections + Matrix Free



SpatialDomains

$\chi^e(\xi)$

StdRegions



$$\mathbf{f}[i] = \int_\Omega \ldots$$

$$\sum_e \sum_p \int_{\Omega^e} \ldots (x)dx$$

$$= \sum_e \sum_p \int_{\Omega^e} \phi_p(\chi^e(\xi)) f(\chi^e(\xi)) J^e d\xi$$

Diffusion solver

- Quick example: solve the diffusion equation with backward Euler:

$$\frac{\partial u}{\partial t} = \varepsilon \nabla^2 u \longrightarrow \frac{u^{n+1} - u^n}{\Delta t} = \varepsilon \nabla^2 u^{n+1} \longrightarrow \nabla^2 u^{n+1} - \frac{u^{n+1}}{\varepsilon \Delta t} = -\frac{u^n}{\varepsilon \Delta t}$$

- Involves the solution of a **Helmholtz equation**.

- We'll use basic routines to highlight how you might solve this using fundamentals of the library.

- In Nektar++, see `solvers/DiffusionSolver` for full source code.

Diffusion solver

Create session

```
session = LibUtilities::SessionReader::CreateInstance(argc, argv);
```

Set up a mesh

```
string var = session->GetVariable(0);
auto mesh = SpatialDomains::MeshGraph::Read(session);
auto field = MemoryManager<MultiRegions::ContField2D>
    ::AllocateSharedPtr(session, mesh, var);
```

Initial conditions

```
int nq = field->GetNpoints();
Array<OneD, double> x0(nq), x1(nq), x2(nq);
field->GetCoords(x0, x1, x2);
icond->Evaluate(x0, x1, x2, 0.0, field->UpdatePhys());
```

Parameters

```
double epsilon = session->GetParameter("epsilon");
double delta_t = session->GetParameter("delta_t");
```

Diffusion solver

Time integrate

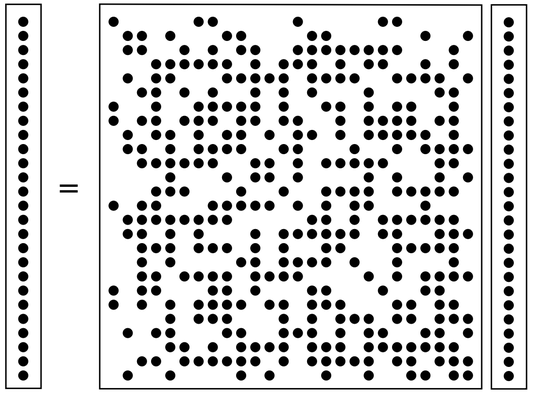
```
for (int n = 0; n < nSteps; ++n)
{
    Vmath::Smul(nq, -1.0/delta_t/epsilon,
                    field->GetPhys(),    1,
                    field->UpdatePhys(), 1);

    field->HelmSolve(field->GetPhys(),
                    field->UpdateCoeffs(),
                    NullFlagList,
                    factors);

    field->BwdTrans (field->GetCoeffs(),
                    field->UpdatePhys());
}
```

Output

```
fldIO->Write(outFile, FieldDef, FieldData);
```

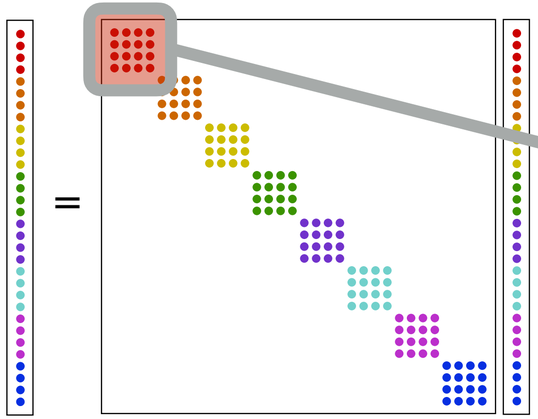
Implementation choices

Finite element operation evaluations (e.g. mass matrix) form bulk of simulation cost; however can be evaluated in several ways.



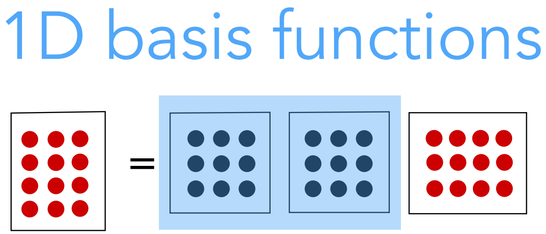
Global matrix

assemble a sparse matrix



Local evaluation

create elemental dense matrices + assembly map



Matrix free

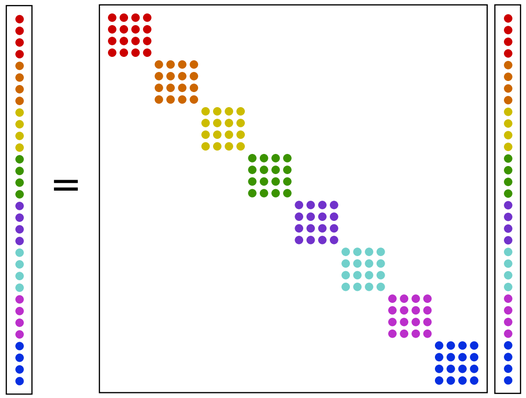
no local matrices at all

sum factorisation speedup

increasing arithmetic intensity

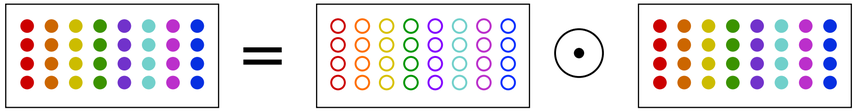
Collections

Local Matrix

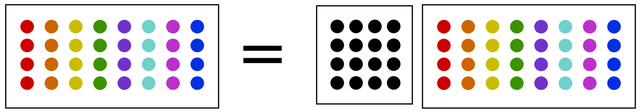


StdMat

1. Apply Jacobian (L1)

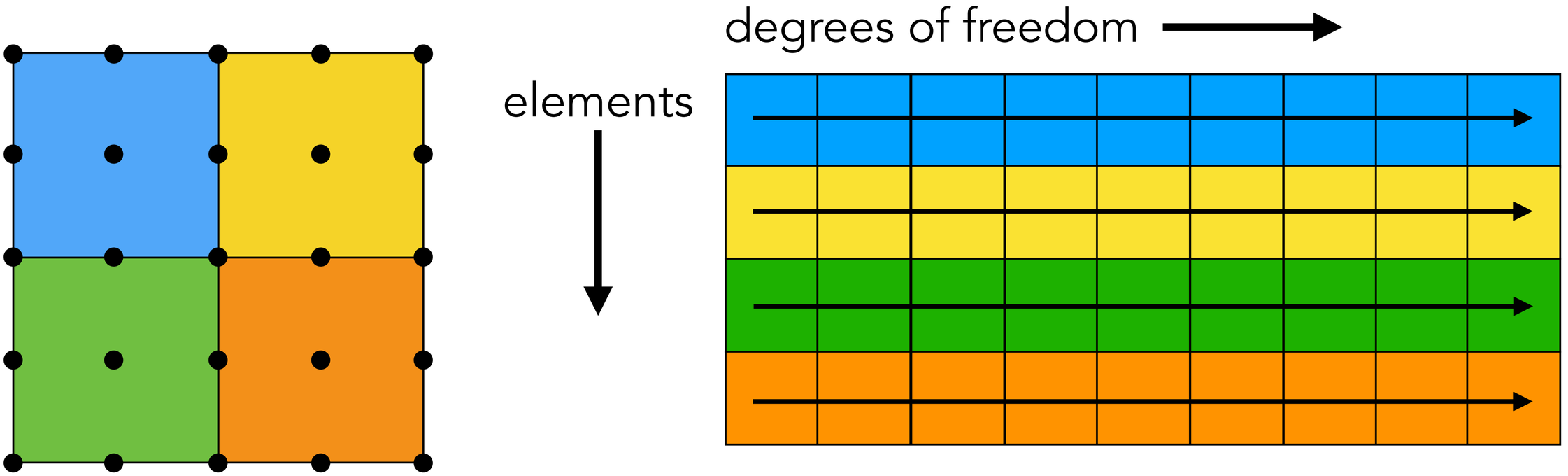


2. L3 Multiply by ref. matrix



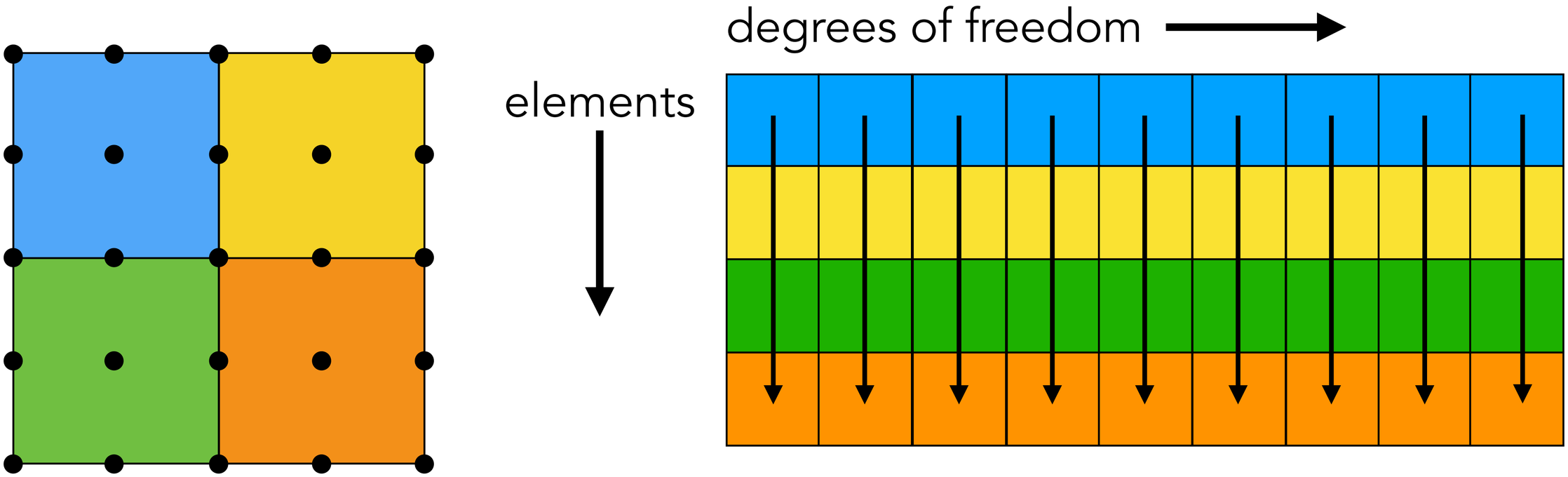
Data layout

Natural to consider data laid out element by element



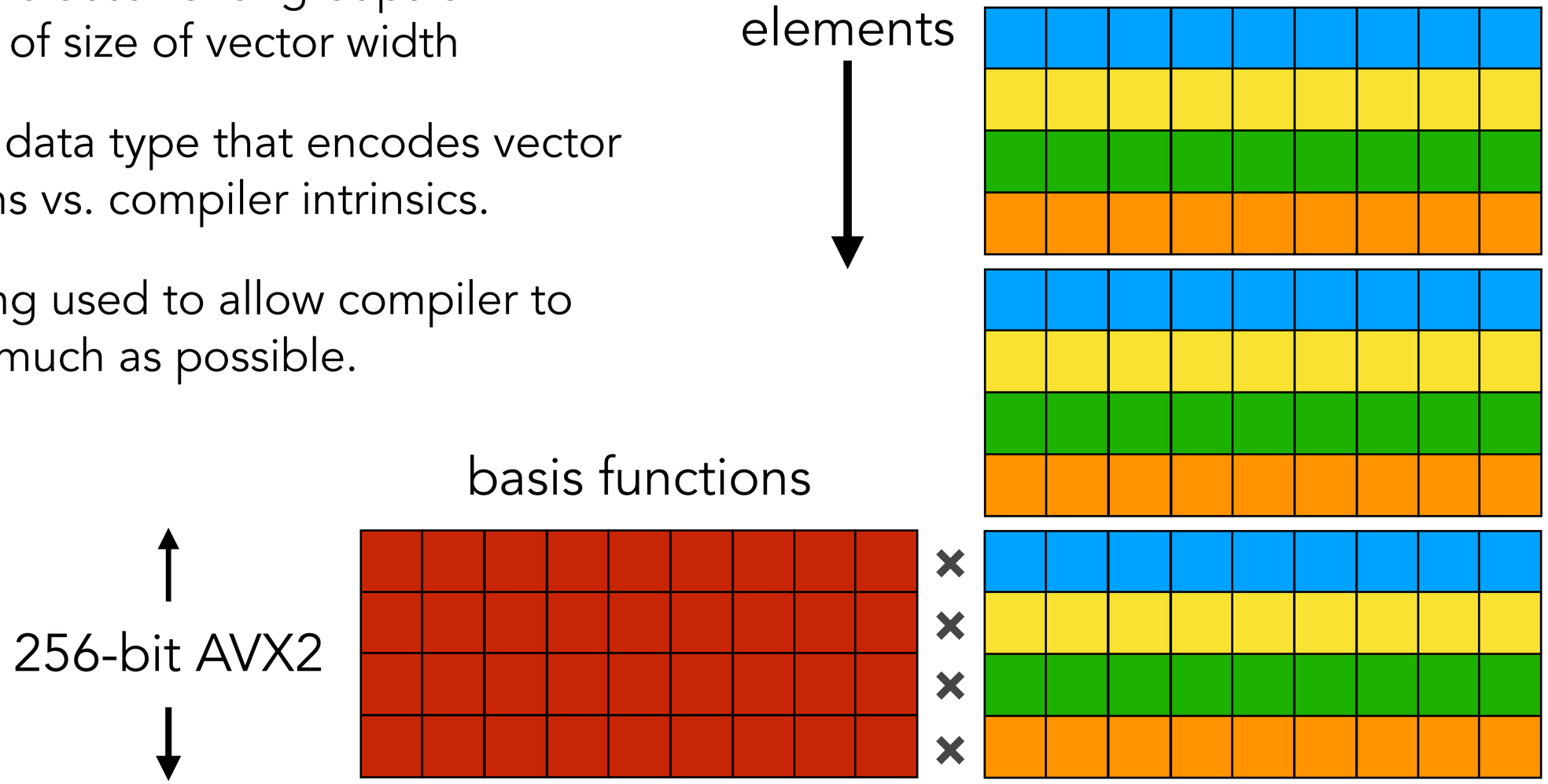
Data layout

Exploit vectorisation by grouping DoFs by vector width



Use of vectorisation

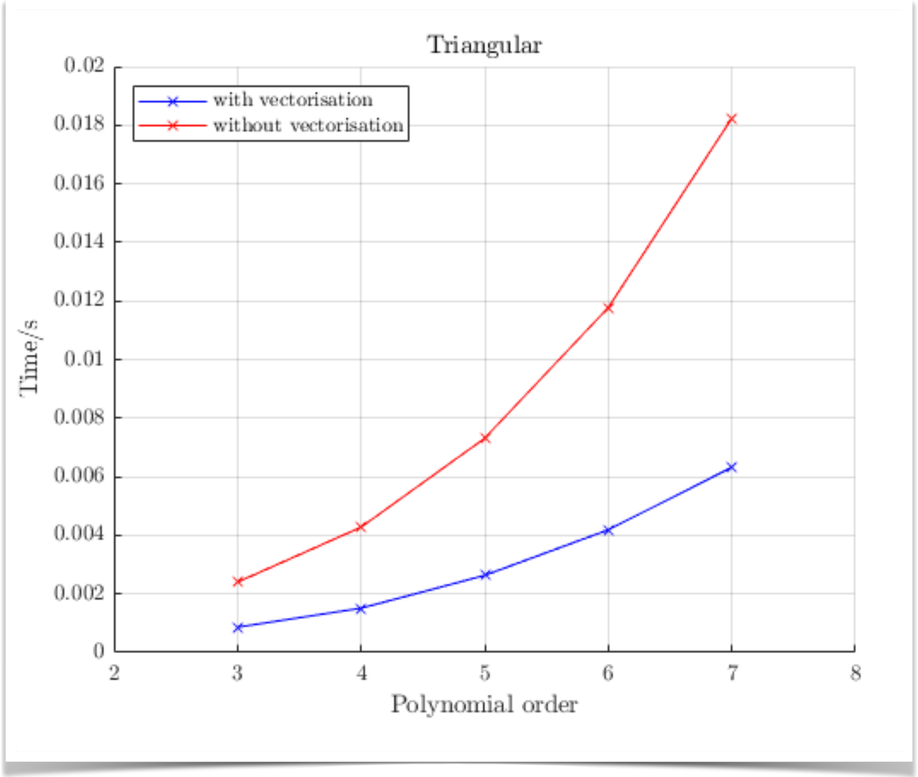
- Operations occur over groups of elements of size of vector width
- Use C++ data type that encodes vector operations vs. compiler intrinsics.
- Templating used to allow compiler to unroll as much as possible.

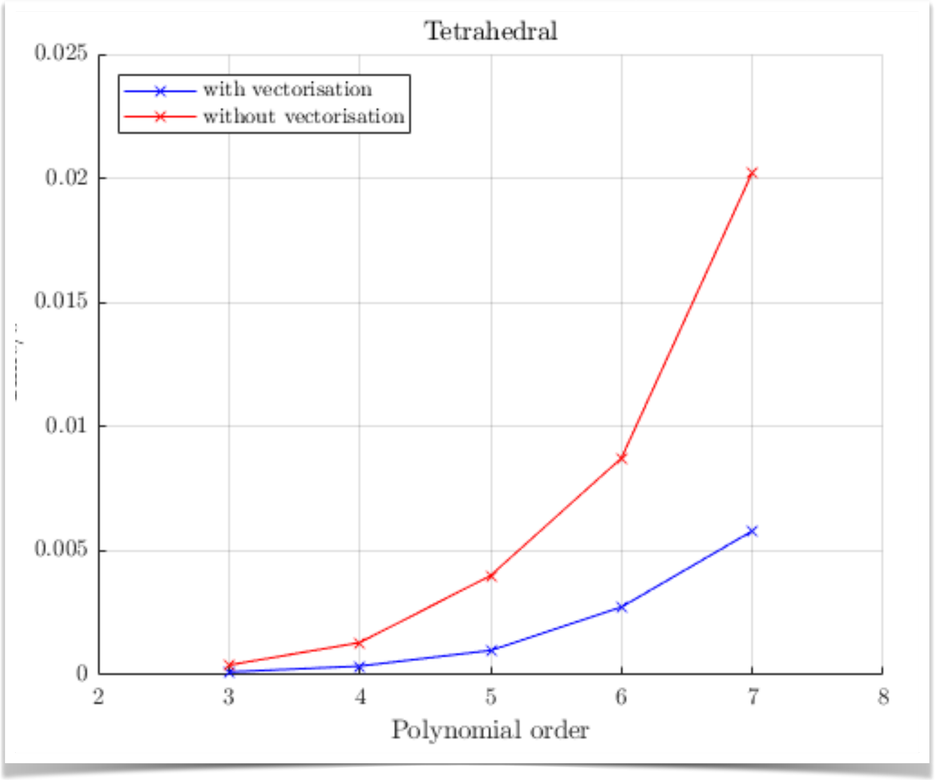


Software implementation

$$\mathscr{L}(u) = \nabla[D \cdot \nabla u]$$

d x *d* spatially-constant diffusion tensor

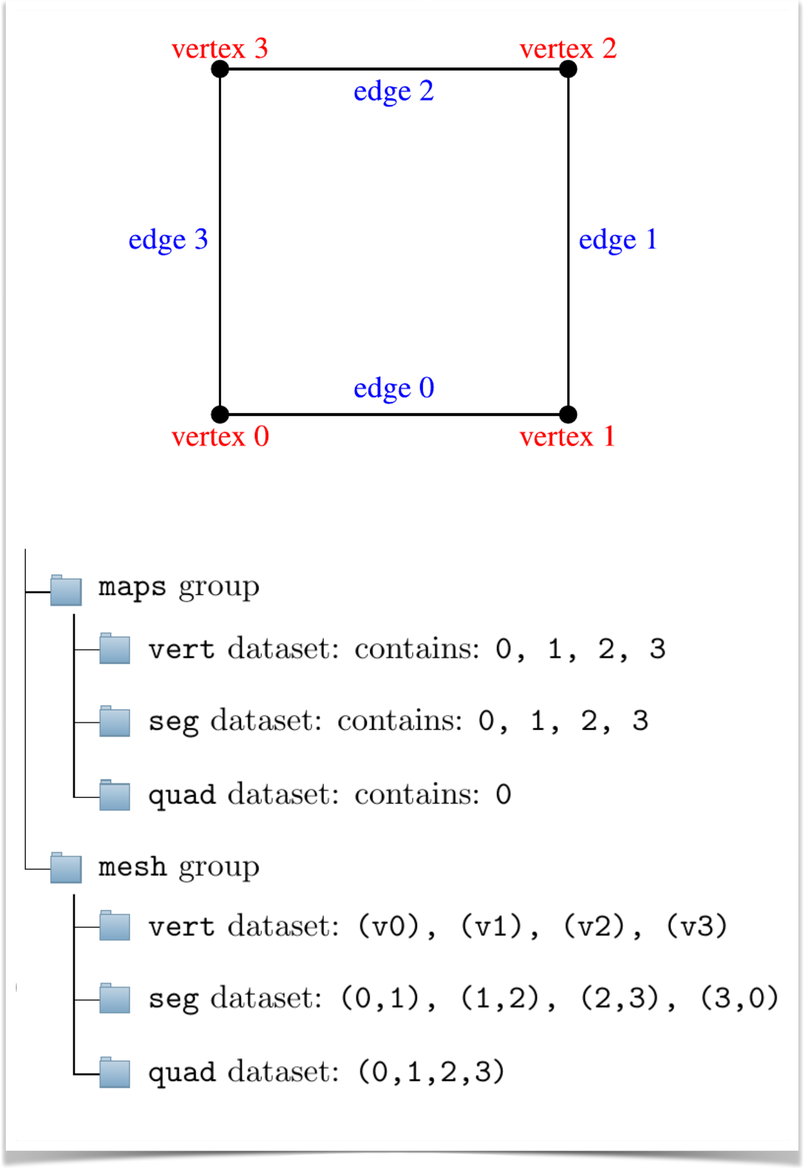




Good speedups observed over non-vectorised versions of this operator for both 2D & 3D elements

Parallel I/O

- **Hierarchical Data Format**
  - Scalable and high-performance
  - Used for both input and output
  - Read/write in parallel
- **Usage**
  - Distribute readingof elements
  - Parallel partition: PTScotch
  - Top-down reading of mesh
  - Bottom-up object construction
  - Read ancillary data (composites).

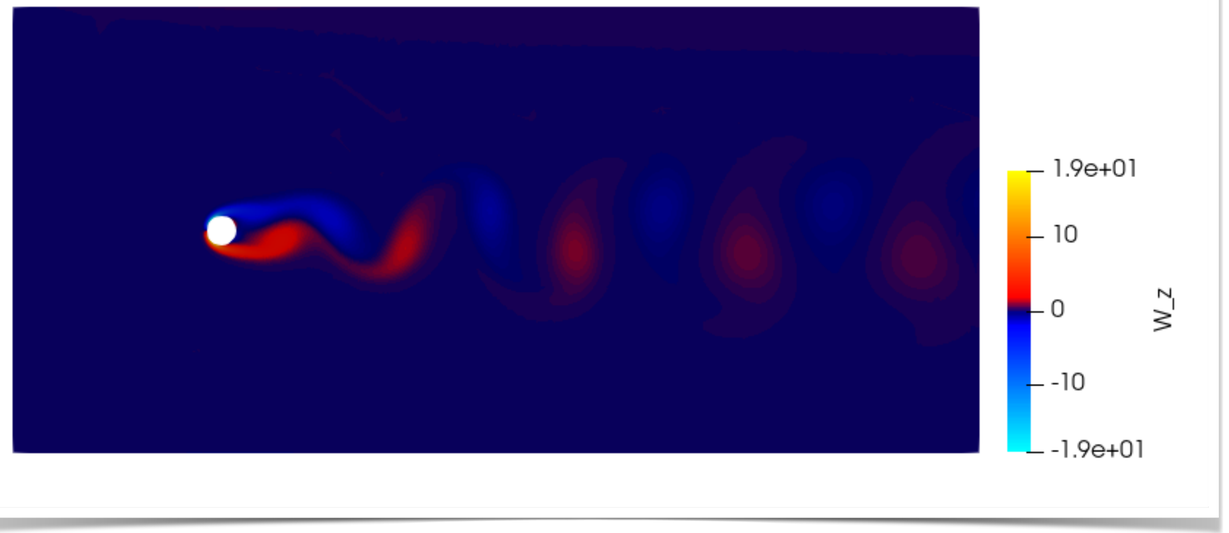


Mesh structure & its layout within a HDF5 mesh file.

In-situ processing

- Post-processing poses storage/ memory challenges.
- In-situ filter for using existing post-processing algorithms.
- Enables efficient calculation of derived quantities during run.
- Modules specified in XML session file.

```
<FILTER TYPE="FieldConvert">
  <PARAM NAME="OutputFile"> vorticity.vtu </PARAM>
  <PARAM NAME="OutputFrequency"> 100 </PARAM>
  <PARAM NAME="Modules">
    vorticity
    removefield:fieldname=u,v,p
  </PARAM>
</FILTER>
```



Computation of vorticity performed every 100 timesteps using the FieldConvert filter.

Python interface

- Bindings for key libraries.
- Simplify usage for new users and for teaching.
- Access from other software packages.
- Leverages Boost.Python.
- Interoperability with NumPy: explicit memory sharing between Nekta r++ arrays.

```
import NekPy.LibUtilities as LibUtil
import NekPy.StdRegions as StdReg
import numpy as np

# Set P = 8 modes and Q = P + 1 quadrature points.
nModes = 8
nPts   = nModes + 1

# Create GLL-distributed quadrature points.
pType  = LibUtil.PointsType.GaussLobattoLegendre
pKey   = LibUtil.PointsKey(nPts, pType)

# Create modified C^0 basis on these points.
bType  = LibUtil.BasisType.Modified_A
bKey   = LibUtil.BasisKey(bType, nModes, pKey)

# Create quadrilateral expansion using this basis
# in each coordinate direction (tensor product).
quad   = StdReg.StdQuadExp(bKey, bKey)

# L^2 projection of f(x,y) = cos(x)*cos(y) onto the
# quadrilateral element. Note x,y are numpy ndarrays
# and evaluation of cos() is performed using numpy.
x, y   = quad.GetCoords()
fx     = np.cos(x) * np.cos(y)
proj   = quad.FwdTrans(fx)

# Integrate function over the element.
print("Integral = {:.4f}".format(quad.Integral(fx)))
```

Simple example of integration within a quadrilateral region.

Summary

Summary

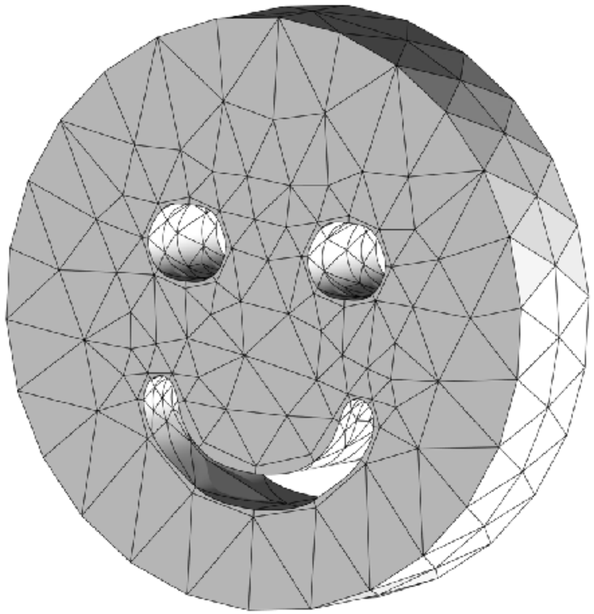
- Hopefully a useful overview of high-order methods and some of our implementation within Nektar++.
- We see high-order methods as an enabling technology for many of these challenging physics problems.
- Some areas for us to work on (some discussion on this later):
  - Mesh generation continues to be a focus and typically quite a challenge for new users.
  - Performance portability across architectures and multi-architecture support.

Nektar++ workshop

- We're holding our 6th annual(ish) workshop for Nektar++.
- Good opportunity to see (or contribute!) some talks around recent developments in Nektar++ and application areas around it.
- Also good opportunity to connect with the various developers around Nektar++.
- Will be held **13-16th September 2022** at Imperial College London.
- Registration free!

https://www.nektar.info/community/workshops/nektar-workshop-2022/

Thanks for listening! Questions?



david.moxey@kcl.ac.uk
c.cantwell@imperial.ac.uk

@davidmoxey

www.nektar.info
gitlab.nektar.info

Mailing list: nektar-users@imperial.ac.uk