Progress during NEPTUNE & potential interactions with particle codes

David Moxey
Department of Engineering, King's College London

UKAEA Nektar++ Training Event, 1st August 2022

Overview

- As part of NEPTUNE, we have been investigating the application of high-order methods within prototype fusion problems.
- The work undertaken thus far focuses on three areas:

High-order mesh generation

- How do we generate meshes suitable for fusion applications?
- Consider surface mesh accuracy, anisotropy and geometric complexity.

Performance on modern hardware

- High-order methods show great potential for exascale platforms.
- Examining how performant kernels can be designed for **unstructured** elements.

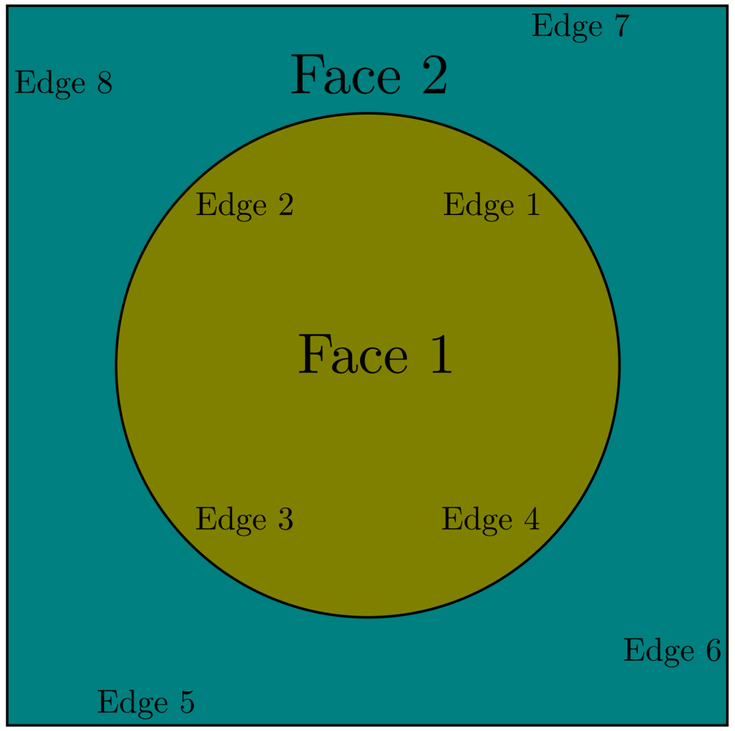
Community engagement with NEPTUNE

- NEPTUNE is composed of several partners, each developing proxyapps
- Developing coupling approaches, modern CI and coordination

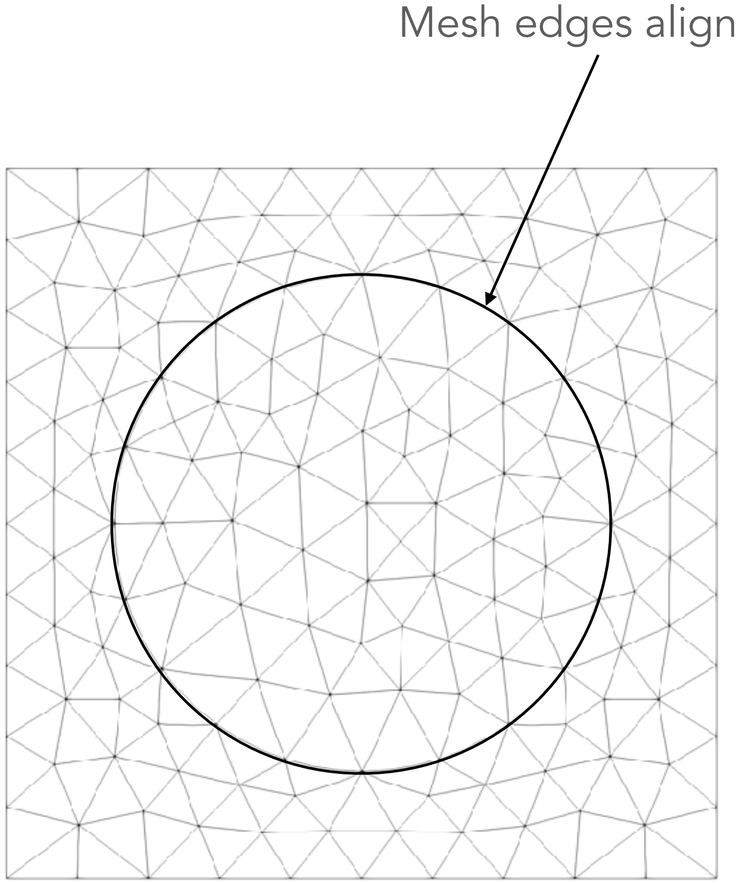
CAD-exact feature representation

- For fusion applications, certain internal features such as the plasma separatrix should be modelled as accurately as possible.
- Also beneficial to generate anisotropic elements within these regions to align with the corresponding physical models.
- Within the mesh generation process, we therefore consider the use of internal CAD curves which allow for exact representation of these features.
- Furthermore, the optimisation process can be made CAD-aware, so that during optimisation, nodes 'slide' along CAD curves & surfaces.
- Implementation of this within the high-order mesh generation software *NekMesh*, which is part of the Nektar++ spectral/*hp* element framework.

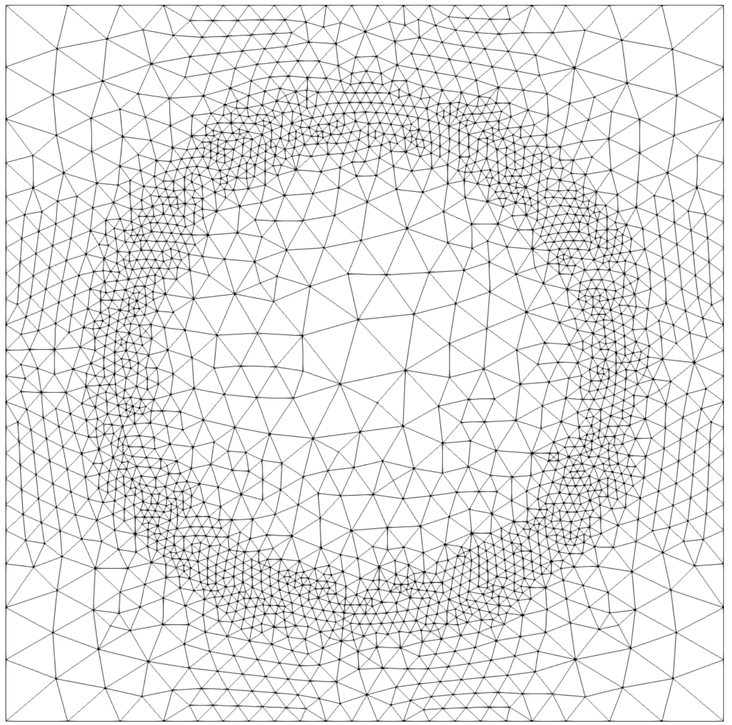
Example: circle in square



CAD boundary representation

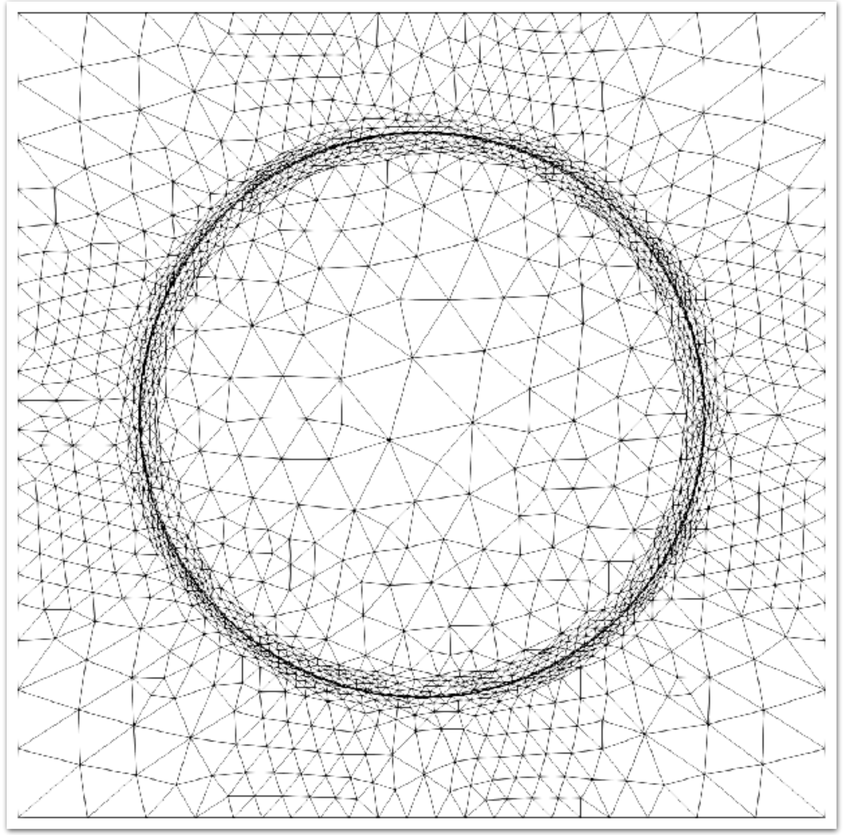


Coarse high-order grid

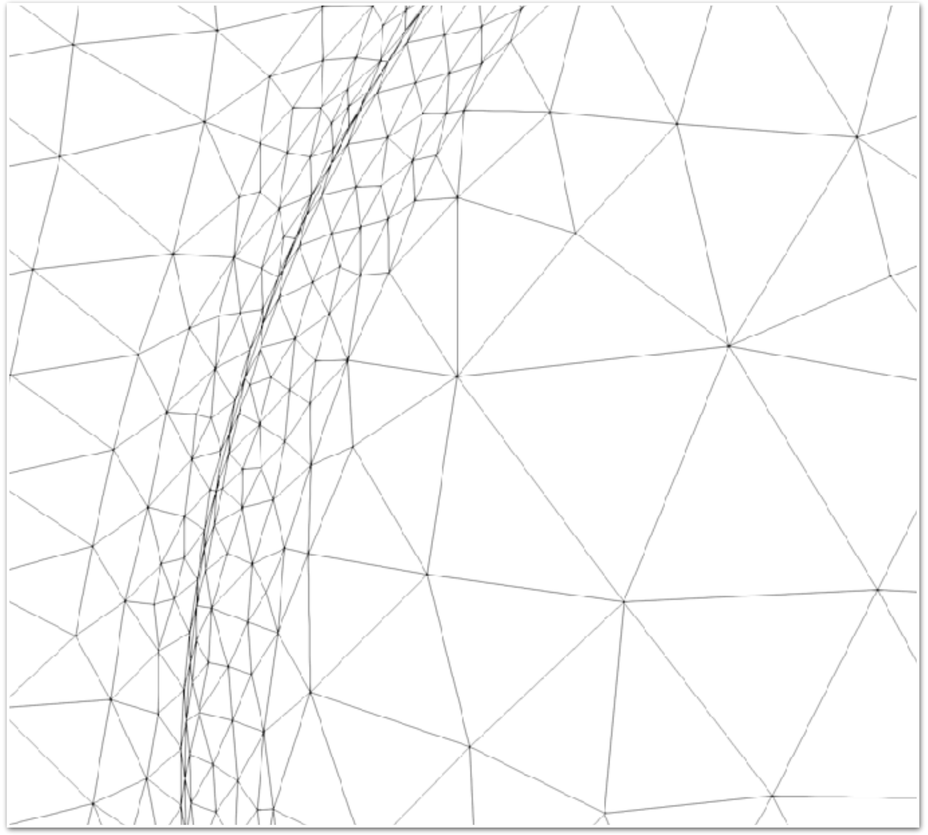


Isotropically-refined high-order grid

Example: circle in square



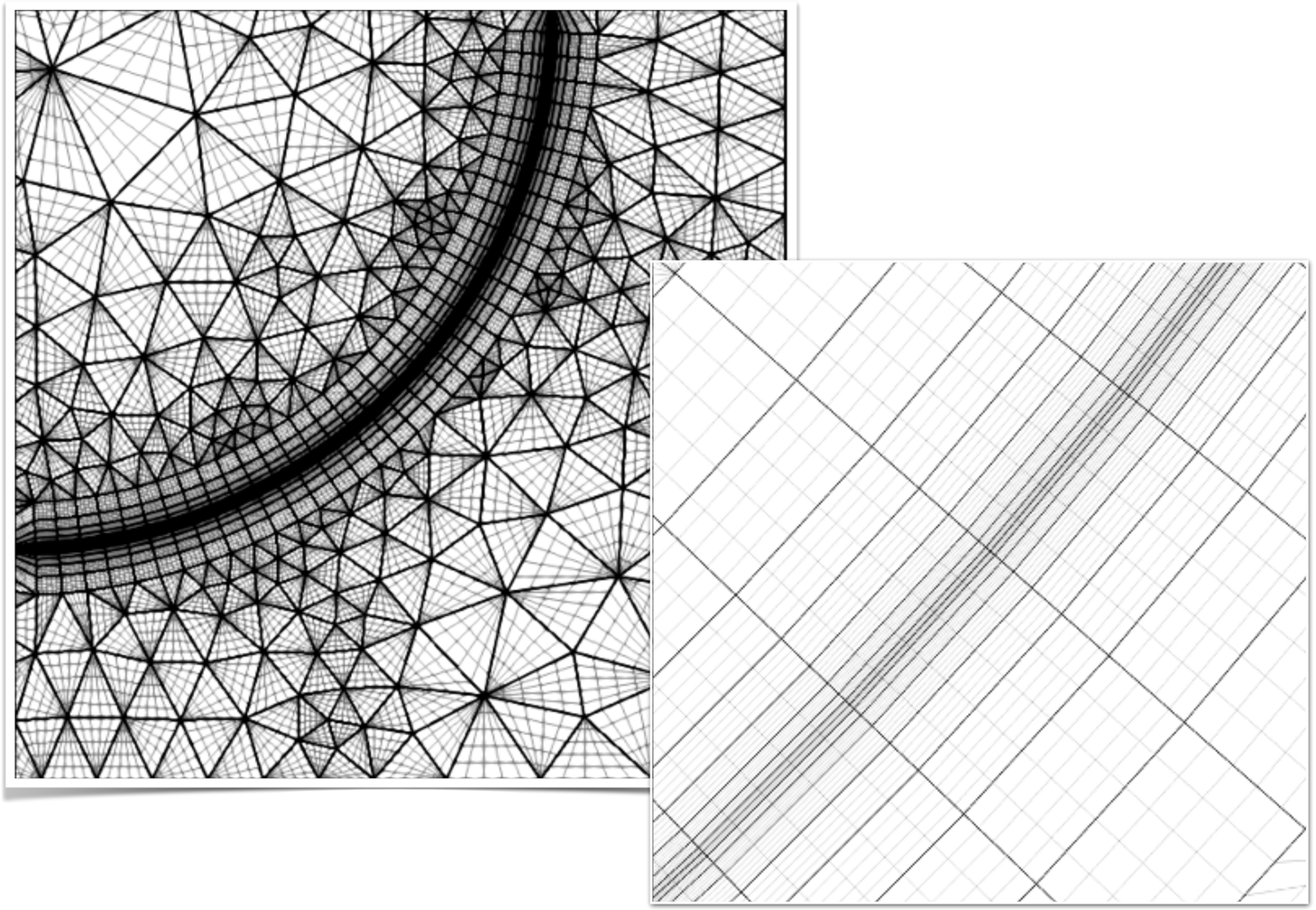
Applying anisotropic refinement



Close-up view of boundary of circle

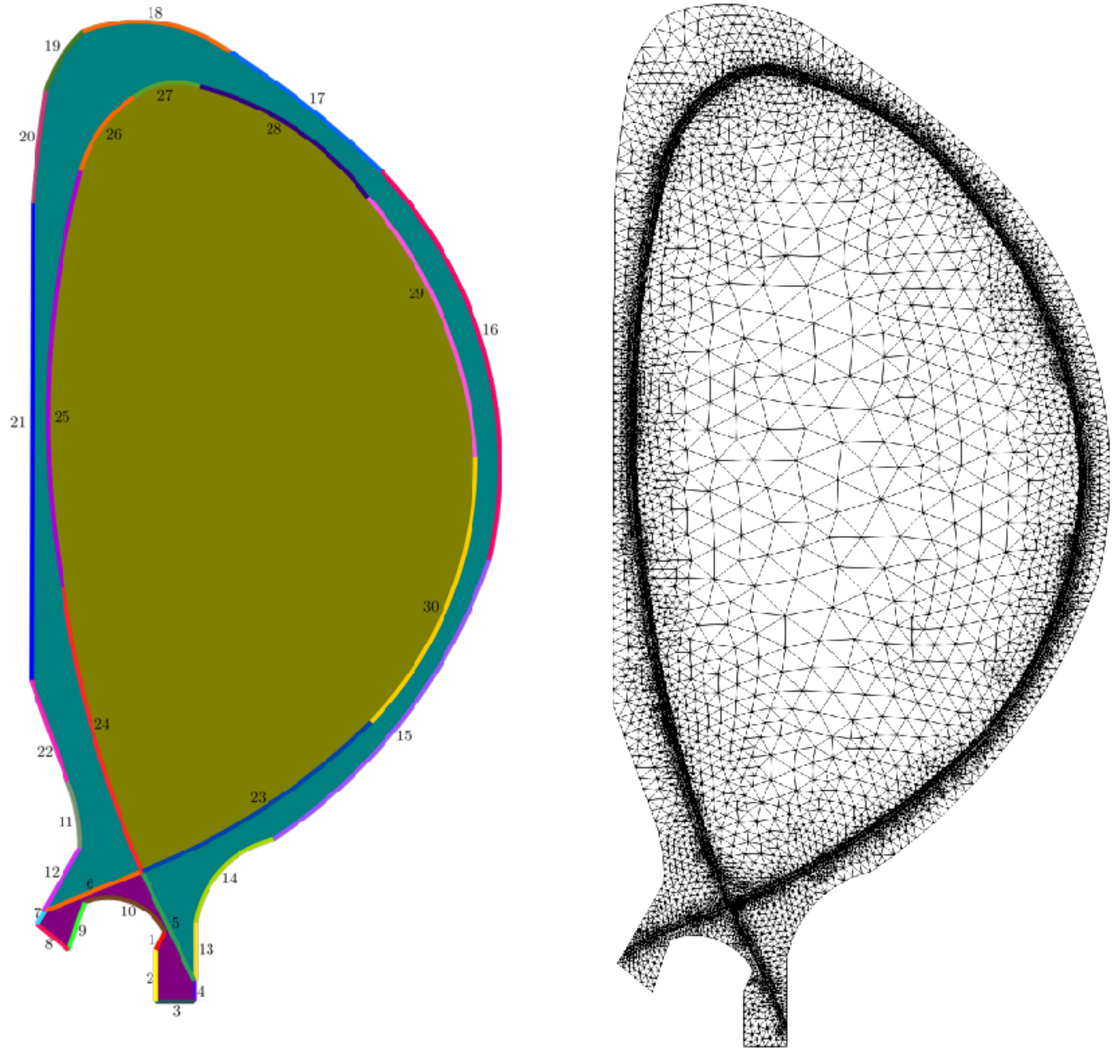
Interior layer refinements

- Have also begin process of generating refinement in a more structured manner.
- This uses boundary layer refinement strategies we typically adopt for external flows but on the internal CAD curve.
- Isoparametric approach allows arbitrary refinement as desired.



Example: sample plasma core separatrix

- Same process applied to a schematic tokamak cross-section.
- Fully valid mesh (no inverted elements) at order $P = 5$.
- Potential future directions:
  - extension to 3D;
  - improving performance via different optimisation strategies;
  - distributed-type parallelism, GPU offloading.



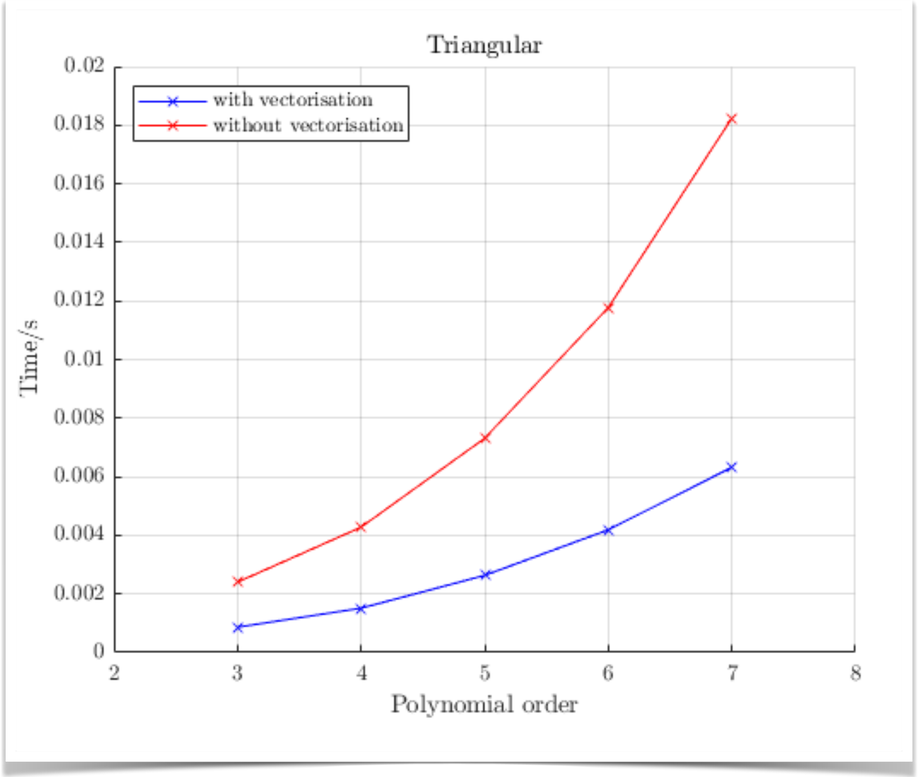
Performance & matrix-free operator evaluation

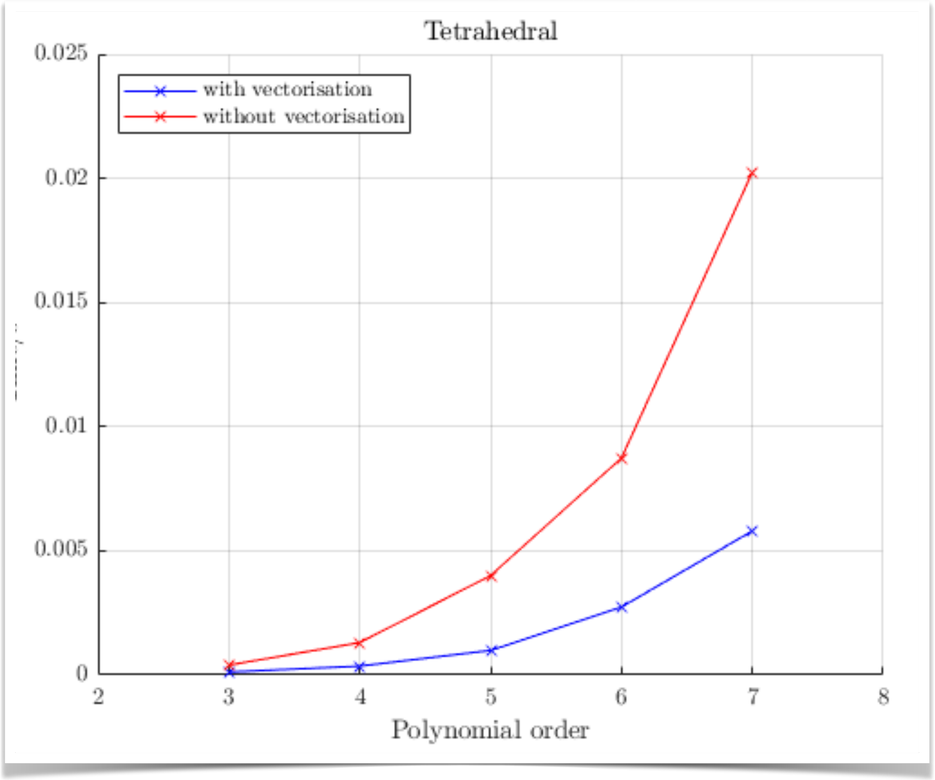
- A key ability of high-order methods is their **arithmetic intensity:** the ratio of floating-point operations (FLOPS) per byte of data transferred over memory.
- As P increases, so too does the amount of FLOPS needed for evaluation.
    - This sounds bad, but we get a corresponding increase in accuracy!
- Significantly reduce the impact of memory bandwidth limits found on modern hardware architectures & exascale platforms.
- Solution of the anisotropic heat transport involves evaluation of operators $\mathscr{L}(u) = \nabla[D \cdot \nabla u]$, where D is a spatially-constant $d \times d$ tensor.
- Second track of work focuses on performant-kernels for this operator, particularly focused on **unstructured elements** due to complexity of geometry found in realistic problems.

Software implementation

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

d x *d* spatially-constant diffusion tensor





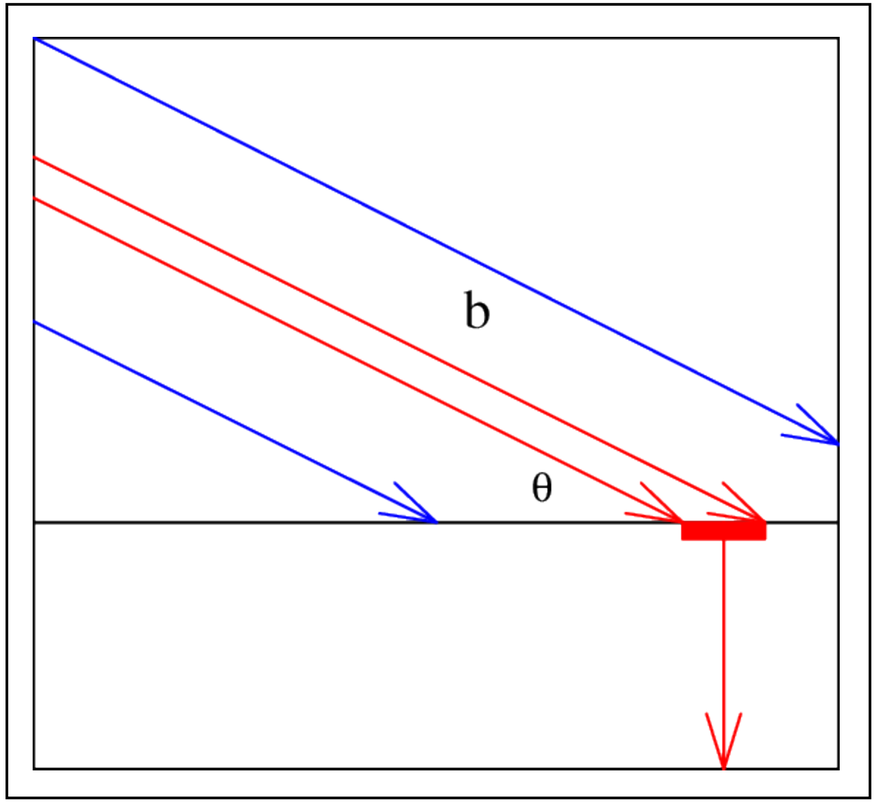
Good speedups observed over non-vectorised versions of this operator for both 2D & 3D elements

Test cases within NEPTUNE

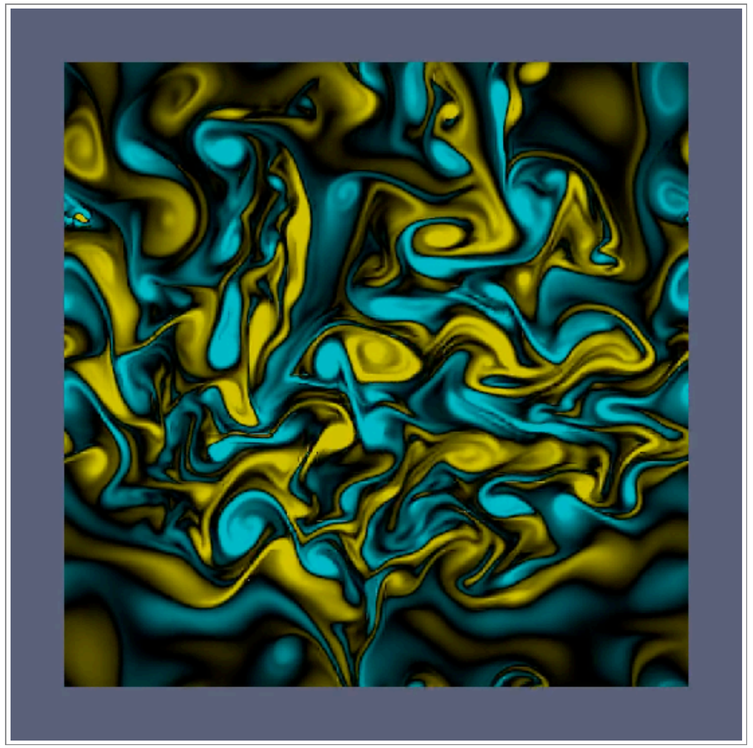
- Focuses around setup of a solver for a highly anisotropic heat transport case based on high-order elements.

$$\frac{3}{2}N\frac{\partial T}{\partial t} = \nabla \cdot \left(\kappa_{\parallel} \mathbf{b}[\mathbf{b} \cdot \nabla T] + \kappa_{\perp}(\nabla T - [\mathbf{b} \cdot \nabla T]) \right)$$

- Two aspects being considered:
  - **Physics:** for this model problem, how do high-order elements perform for high levels of anisotropy?
  - **Software:** how do we implement efficient realisations suitable for exascale platforms?

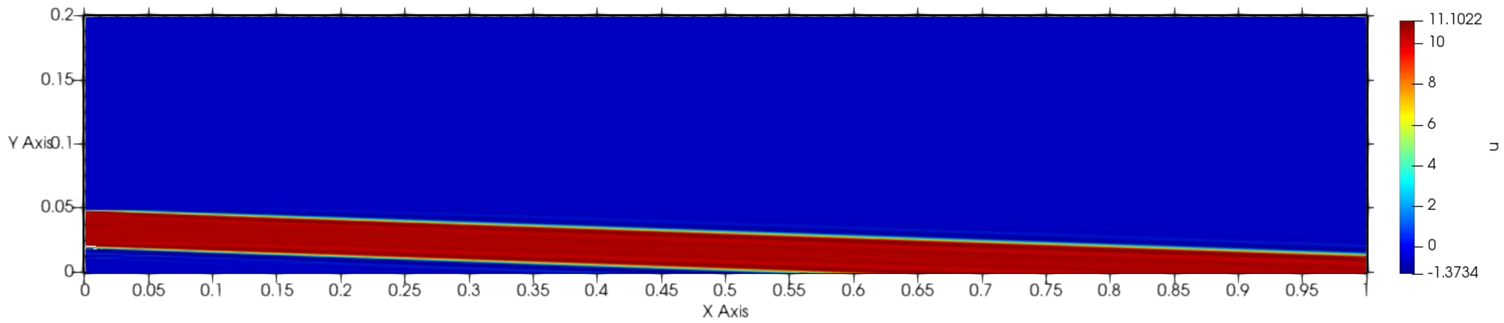


Development of proxyapps



Hasegawa-Wakatani drift-wave system

https://github.com/ExCALIBUR-NEPTUNE/nektar-driftwave



Anisotropic heat transport

https://github.com/ExCALIBUR-NEPTUNE/nektar-diffusion

SOL-solver (single species 1D solver)

https://github.com/ExCALIBUR-NEPTUNE/nektar-sol-1d

Particle interactions

System 2-6 progress

- (From my perspective!) a somewhat scary set of equations for a 2D fluid model.
- Thanks to Ed/James for a fully laid out version of this!
- To date progress mostly around scoping: what do we have, and what do we not have.
- One thing that will need to be addressed is interaction with particle codes.
- Will briefly outline how this could work with Nektar++.

$$\partial_t n_e + \nabla \cdot (n_e \mathbf{u}_e) = S_{n_e} - \frac{n_e}{\tau_{n_e}}$$

$$\partial_t \nabla \cdot \mathbf{E}^+ + \nabla \cdot \left(\nabla \cdot \left(\mathbf{u}_i \otimes \mathbf{E}^+\right)\right) = \nabla \cdot \left(n_i \left(\mathbf{u}_{\nabla Bi} + \mathbf{u}_{cx}\right) - \frac{1}{Z_i} n_e \mathbf{u}_{\nabla Be}\right) + \frac{1}{Z_i}\frac{n_e}{\tau_{n_e}} - \frac{n_i}{\tau_{n_i}} + \nabla \cdot \left(\nu \nabla_\perp \left(\nabla \cdot \mathbf{E}^+\right)\right)$$

$$\partial_t \mathcal{E}_e + \nabla \cdot (\mathcal{E}_e \mathbf{u}_e + p_e \mathbf{u}_e) = S_{\mathcal{E}_e} - \frac{\mathcal{E}_e}{\tau_{Ee}} + Q_{ie} + \nabla \cdot (\chi_{\perp e} n_e \nabla_\perp T_e)$$

$$\partial_t \mathcal{E}_i + \nabla \cdot (\mathcal{E}_i \mathbf{u}_i + p_i \mathbf{u}_i) = S_{\mathcal{E}_i} - \frac{\mathcal{E}_i}{\tau_{Ei}} - Q_{ie} + \nabla \cdot (\chi_{\perp i} n_i \nabla_\perp T_i)$$

$$\partial_t n_n = S_{n_n} + \nabla \cdot (D_n \nabla_\perp p_n)$$

Particles in Nektar++ to date

- Some attempts have been made in the past to introduce simple to moderately complex Lagrangian particle tracing within Nektar++.
- These were particularly motivated by physical fluid dynamics applications (e.g. erosive wear.)
- There are essentially three interactions required:
  - for a given particle location in world-space, which element does this correspond to in the grid (and in parallel execution), and what are the reference coordinates within that element;
  - given that location, evaluate the solution field(s) there to e.g. extract velocity;
  - given results of the particle code, project e.g. density into the finite element space.

Finding positions

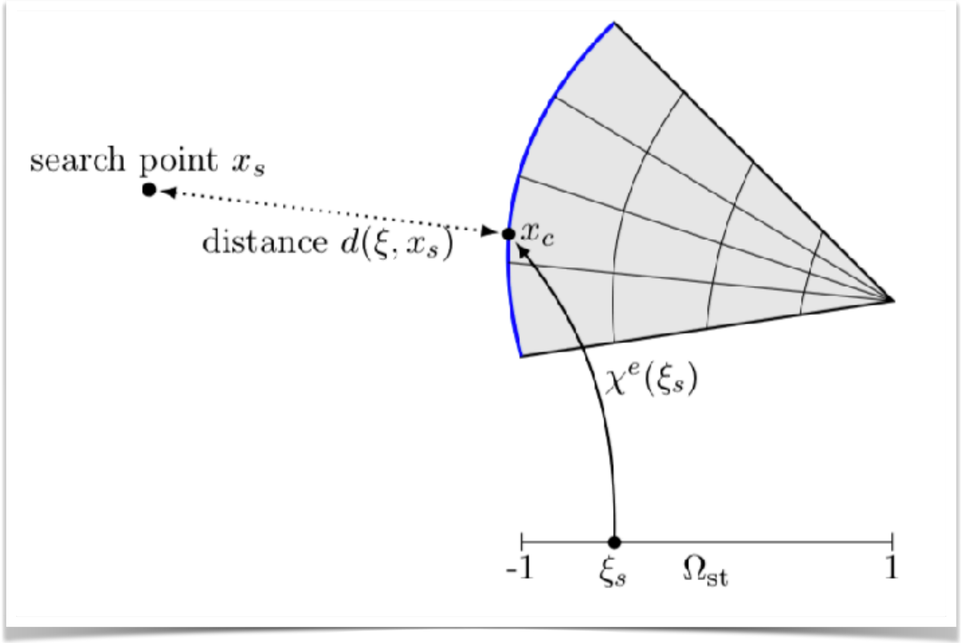
- This can be done at two levels within the library hierarchy:
    - Within `SpatialDomains`, each `Geometry` object has a `ContainsPoint` method, which accepts as parameter a position (x, y, z) in world-space.
    - Within `MultiRegions`, an `ExpList` object has a `GetExpIndex` method which performs essentially the same action, but has a couple of additional speed-ups and is also designed to handle e.g. manifolds.
-

Finding positions

- In particular we use the following process to speed up location searches within the mesh:
    - At mesh construction time, for each element of the mesh, construct the element's bounding box.
    - We then use the bounding boxes to construct an *r*-tree.
    - When we look up a point, we first use the *r*-tree to locate the subset of elements that might contain this point.
    - Then, we loop over those subset and call the `ContainsPoint` method.
- However the last point can still be expensive if we are querying curvilinear elements.

Finding reference locations

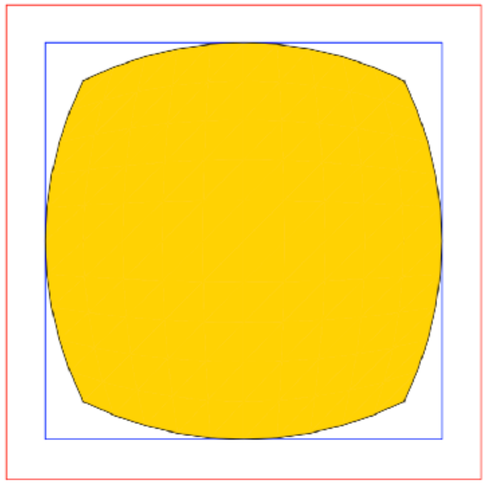
- Since we ultimately want to evaluate at points within the element, we need to know the location within the *reference element*.
- To do this requires inverting the mapping χ that is used to define the element.
- For straight sided elements, χ is affine and this is straightforward; however for curvilinear elements (or non-parallelepiped quads/hexes) this is non-linear.
- We solve via a Newton-Raphson search.



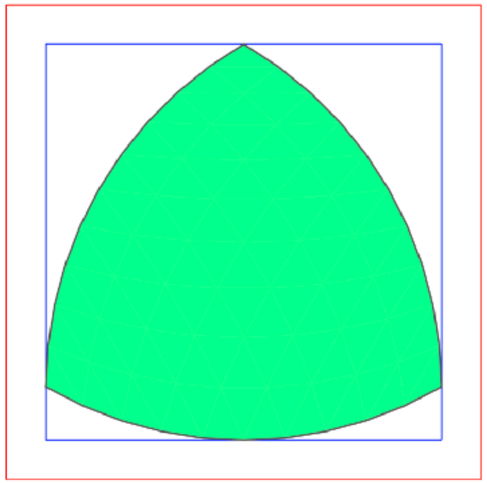
Finding minimum distance between a point and an element's edge

Bounding boxes for curvilinear elements

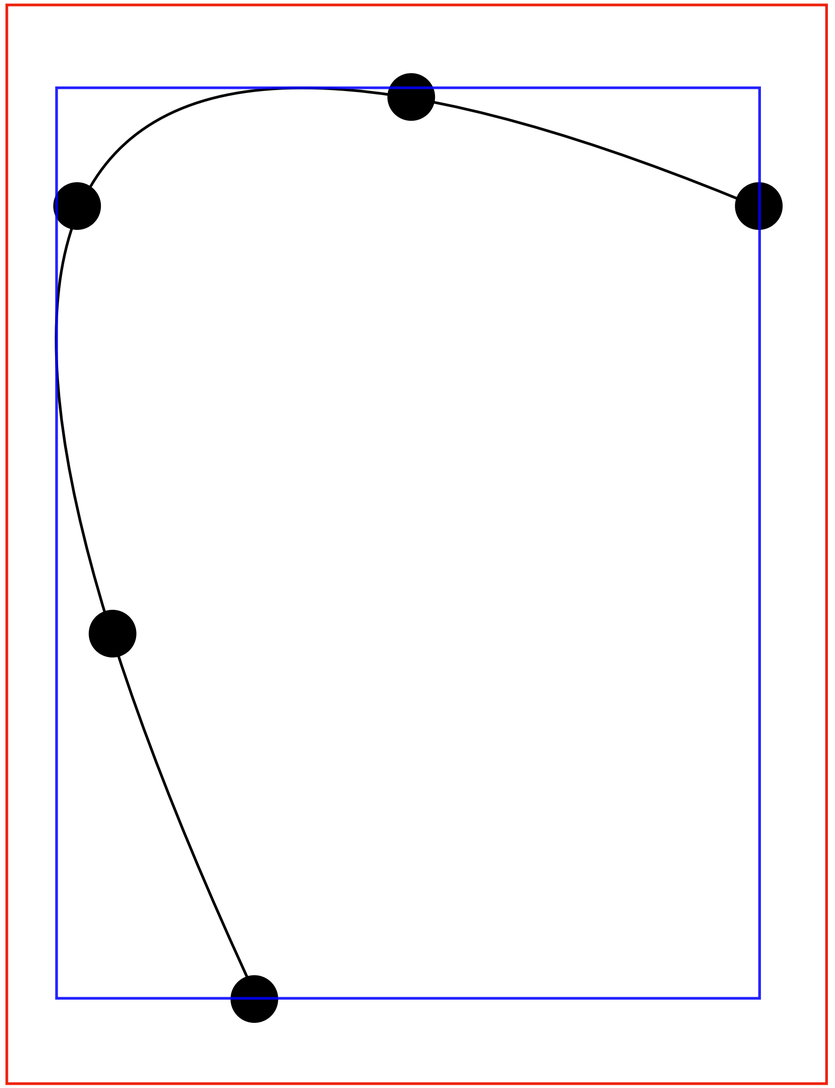
- Additional headache is bounding boxes are also difficult/expensive to compute.
- Cheap guess is to use quadrature points and add a small margin (e.g. 10%). Experimental branch where we compute exact boxes at additional costs.



(a) Convex quadtrilateral



(b) Convex triangle



Barycentric formulation for Lagrangian interpolation

- Given an element, a location within the reference element, and a field represented at the quadrature points, we can evaluate the field at that point.
- This is a straightforward polynomial interpolation, which can be performed by calling the `StdRegions` function `PhysEvaluate` call.
- Until relatively recently, this was quite expensive at $\mathcal{O}(P^2)$ for a 1D segment. However we can leverage tricks from spectral methods to reduce this via barycentric interpolation.
- Storing weights w_j allows us to perform polynomial interpolation at any location in $\mathcal{O}(P)$ time (and derivatives).

$$\tilde{u}(x) = \frac{\sum_{j=0}^{N} \frac{w_j}{x - x_j} a_j}{\sum_{j=0}^{N} \frac{w_j}{x - x_j}}$$

$$w_j = \frac{1}{\prod_{i \neq j} (x_j - x_i)}$$

Results - optimisation profiling

- Example in the FindDistance function shown before for 138 points along a curved interface.
- Benchmarking existing Lagragian interpolation method with old/new bounding boxes vs. new barycentric formulation.
- Significant speedup for this problem (nonconformal DG grid).

	FindDistance [%CPU]	Total computation time [s]	L2 Error
Lagrangian (old bbox)	91.12%	43.2631s	4.73493E-10
Lagrangian (new bbox)	85.15%	30.7281s	4.73493E-10
Barycentric (new bbox)	24.47%	4.86649s	4.73493E-10

Other considerations

- At present, these routines are not parallel aware so all local domains on each rank handle things independently.
- Need to investigate mesh halos (from Will's code) and use of appropriate logic to handle fringe cases (e.g. landing exactly on vertex of element shared across parallel partitions).
- Interactions with boundaries can also be difficult to handle where the boundary is curved: similar issues with solving nonlinear problems and can be easy to miscompute e.g. deflection angles when looking at particle-boundary collisions.