Report 2053622-TN-03 D1.2: Augmentation of the *NekMesh* generator to provide quad-based meshes for 2D configurations

Mashy Green & David Moxey, King's College London
Chris Cantwell, Bin Liu & Spencer Sherwin, Imperial College London

28th March 2022

Contents

1 Executive summary **1**

2 Introduction **1**

3 Isoparametric boundary layer generation in *NekMesh* **2**
3.1 Linear boundary layer generation 3
3.1.1 Limitations and work in progress 3
3.2 High-order boundary layer splitting 5
3.2.1 Issues and work in progress 7
3.3 Examples of existing functionality 7

4 Curved 2D quadrilateral meshing **8**

5 Conclusions, work in progress, and future work **9**

1 Executive summary

This report focuses on the initial phase of work undertaken by King's College London and Imperial College London to investigate challenges for generation of high-order meshes for the NEPTUNE project. In this third report that comprises Deliverable 1.2 of our workplan, we extend the two-dimensional high-order mesh generation process for producing meshes for the tokamak edge region to contain quadrilateral elements. By utilising the isoparametric boundary layer mesh generation developed in *NekMesh*, we introduce anisotropic quad elements that greatly reduce the number of elements in the mesh and can be exploited by the matrix-free and hardware accelerated *Nektar++* solvers being developed for Deliverable D2.2 and D2.3 for significant increases in performance.

2 Introduction

The main objective of Task 1.2 is to generate 2D quadrilateral meshes, which would in later work lead to hexahedral 3D meshes. The benefits of these meshes are two-fold: first, us-

ing anisotropic quad-based elements in the refinement regions can greatly reduce the number of elements required (since e.g. two triangles must be used to construct a quadrilateral, or six tetrahedra to construct a hexahedron); second, using quadrilateral elements greatly improves the performance of the matrix-free operators being developed for tasks 2.2 and 2.3, since the tensor-product construction of quadrilateral and hexahedral elements admits a concurrent tensor-product structures in their basis functions, leading to simpler loop structures that are more readily optimised at compile time. Although full-quadrilateral or full-hexahedral meshes of an optimal quality remain an open question within the mesh generation community, meshes with dominant quadrilateral or hexahedral components can significantly reduce the required memory bandwidth (and thus performance) compared to applying them on triangular or tetrahedral elements.

The inclusion of the isoparametric boundary layer splitting also influences the procedure of refinement and adaption provided in Deliverable D1.1 [1], alleviating some of the difficulties with the manual nature of the parameter selection. The role of refinement and r-adaption is envisioned as extending the boundary layer regions rather than providing the sole mechanism for generating anisotropic meshes. The combination of these approaches will allow a user to refine the mesh in both tangential (refinement) and normal (boundary layer splitting) directions to the curve to better align with the anisotropy of the plasma heat transfer.

A large motivation for this work, as stated in the work proposal, is to increase the performance of the solver by providing quadrilateral elements which are highly beneficial to the matrix-free solvers optimised for different architectures (X86, ARM and Nvidia GPUs). This advantage will be easy to utilise also in a mixed-element unstructured mesh where the densest parts of the mesh (due to refinement along the plasma separatrix and flux surfaces) will contain quadrilateral elements, even without the full mesh being quad-based.

As well as the isoparametric boundary layer meshing, a procedure for generating full quad-based meshes, which already exists within the *NekMesh* framework, will be briefly discussed. The approach is inspired by the computer graphics community and the idea of cross fields. Whilst not production ready yet, this method will be explored as a potential building block for fully quadrilateral 2D meshes and mixed hexahedral and tetrahedral meshes in 3D.

The report is structured as follows: first, we discuss the isoparametric boundary layer mesh generation within the bottom-up process of *NekMesh* in section 3 and show the progress that has been made thus far. This is followed by a brief description of the fully quad 2D mesh generation method and how it could be utilised within the context of NEPTUNE in Section 4. Finally, a summary of the progress made, its conclusions and our plans for the future work is given in Section 5.

3 Isoparametric boundary layer generation in *NekMesh*

This section discusses the process of the isoparametric boundary layer mesh generation within the bottom-up process of *NekMesh* as described in Deliverable D1.3 [2]. First, the process to include simple quadrilateral boundary layer elements in the linear surface meshing stage is presented in section 3.1, followed by the adaption of the linear mesh to high-order using a novel isoparametric splitting approach in section 3.2. Finally, some examples of the existing functionality are provided in section 3.3.

3.1 Linear boundary layer generation

The *NekMesh* linear mesh generation system is equipped with a basic boundary layer mesh generation method. The goal of this procedure is to create a single layer of quadrilateral elements (in 2D) or prismatic elements (in 3D) on the boundary of the domain of height δ_b through the extrusion of a line or triangle in the surface-normal direction. The rest of the domain is then meshed as usual with unstructured triangles (2D) or tetrahedra (3D). The reason for generating only a single layer is because a subsequent high-order method for producing curved anisotropic elements is used. This is explained in more detail in section 3.2. Further, in the following we restrict the discussion to 2D, however the same concepts are implemented already in 3D using a triangular surface which generates the prismatic elements in the boundary layer. Generating hexahedral elements in the boundary layer would require quad-based surface meshing which will be touched upon in section 4.

The procedure for generating the quadrilateral elements begins with the definition of vectors (or normals) at each vertex of the edges on the meshed curves (i.e. the 1D mesh of the curves). New mesh vertices are then defined at a distance δ_b along this vector, providing a collection of four vertices that define the quadrilateral element. There are a number of important criteria to verify when considering the normal directions in complex domains. Firstly the normals to all the edges that connect to a vertex must be visible to each other in concave areas or areas with very sharp angles. Secondly, the size δ_b should be limited in convex regions to avoid quadrilaterals self-intersecting.

The procedure for calculating the mesh vertex normals closely follows the work of Aubrey et al. [3]. Here their first algorithm is used where the set of triangles, T, connected to a given vertex are utilised for the calculation. If the algorithm fails to converge, it is most likely that the normals of the edges are not visible to each other and the local curve will need re-meshing. Further to the calculation of the normal, the 1D mesh of the curve must be smoothed, which is achieved by a simple local averaging is used between connected curve mesh vertices. One last consideration must be made as it is quite likely that in convex regions quadrilaterals could be inverted. To counter this, the procedure post-processes all the quadrilateral elements, checking for validity. If the element is invalid the height of the prism is reduced until a valid point is reached. The heights of the quadrilateral layer are then smoothed to ensure a smooth variation through the mesh.

The significant limitation of this system is that no consideration is given to the possibility of two quadrilateral layers in close proximity, as might be found, for instance, in the gaps between flux surfaces. The height of the quadrilateral elements in these regions should be reduced to account for the proximity. This is a challenging task and must be considered when meshing geometries where this may be an issue using this system.

3.1.1 Limitations and work in progress

From the implementation perspective, the described approach in 2D was designed with a single surface region in mind and fails when including embedded curves within the domain that split the the surface into multiple distinct surfaces, e.g. as is required for the geometries of the tokamak edge region. This posseses several implementation challenges.

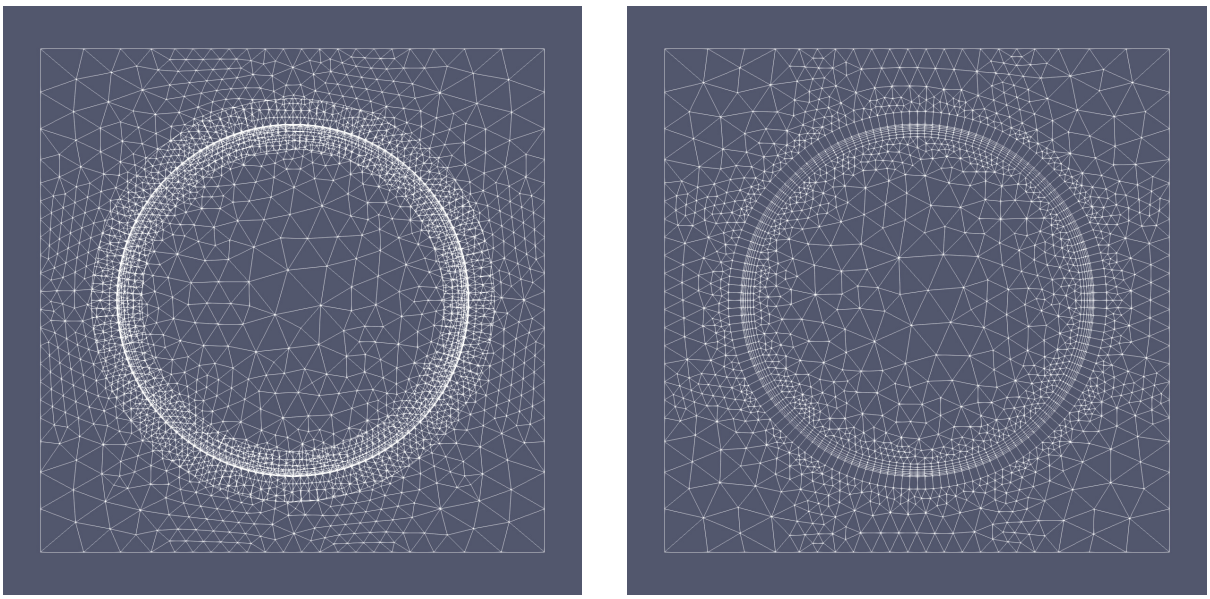
Firstly, the current version of *NekMesh* throws a segmentation fault due to memory access when dealing with boundary layer curves that are not of all surfaces – i.e. there is an assumption within the code that a boundary layer curve is connected to all surfaces within the geometry. This is a reasonable assumption in many of the 'traditional' NekMesh cases, where the domain is a single surface with features such as circles and aerofoils removed from it; however for

this use case, additional logic was required. The issue has been fixed by adding a check to determine whether the curve in question is on the current face, as shown in the algorithm below. This fix is temporary and will be abstracted out to the `EdgeLoop` struct in the `CADSurf` class `NekMesh/CADSystem/CADSurf.h`.

Beyond the segmentation fault, the underlying data structures are still not suitable for dealing with multiple surfaces, resulting in overlapping meshes as seen in figure 1(a). This issue is restricted only to the implementation and does not require conceptual changes to the methodology itself, although it has provided sufficient difficulties that it is where most of our development efforts have taken place recently. The main problem is related to the map `std::map<int, CurveMeshSharedPtr> m_curvemeshes` found in `NekMesh/2DGenerator/2DGenerator.cpp`, which contains the 1D meshed curve, that is manipulated during the generation of the boundary layer on the surface. As there is more than a single surface, the code fails when attempting to generate the boundary layer on the second surface connected to the same 1D meshed curve. The issue has mostly been resolved by changing `m_curvemeshes` to be a vector of maps and creating a new map for each surfaces. This in turn required substantial structural changes to the code which now requires additional loop over surfaces and additional logic to avoid duplication of data. Nevertheless, the approach is now working as a proof of concept on a case-by-case basis, as seen in figure 1(b). A robust implementation is currently being developed but, at the time of writing, remains a work in progress.

Fixing segmentation fault in `NekMesh/2DGenerator/2DGenerator.cpp` by checking if boundary layer curve is on the surface where the boundary layer is being generated.

```
void Generator2D::Process()
{
  ...
  for (int i = 1; i <= m_mesh->m_cad->GetNumSurf(); i++)
  {
    MakeBL(i);
  }
  ...
}

void Generator2D::MakeBL(int faceid)
{
    for (auto &it : m_blCurves)
    {
        std::vector<int> edgeIds;
        for (auto &edges_in_edgeloop :
↪   m_mesh->m_cad->GetSurf(faceid)->GetEdges())
        {
            for (auto &edge : edges_in_edgeloop->edges)
            {
                edgeIds.push_back(edge->GetId());
            }
        }
        if (std::find(edgeIds.begin(), edgeIds.end(), it) == edgeIds.end())
↪   continue;
```



(a) circle-in-square before fixed linear boundary layer generation.

(b) circle-in-square after fixed linear boundary layer generation.

Figure 1: circle-in-square test mesh with isoparametric boundary layer generation. Showing results before and after the fixes to the coarse linear boundary layer generation. In both cases the coarse quadrilateral boundary layer element is split only in one direction instead of both, as discussed in 3.2.

```
        CADOrientation::Orientation edgeo =
            m_mesh->m_cad->GetCurve(it)->GetOrienationWRT(faceid);
        vector<EdgeSharedPtr> es =
↪   m_curvemeshes[faceid-1][it]->GetMeshEdges();

        ...
    }
}
```

3.2 High-order boundary layer splitting

Computational fluid dynamics applications typically require meshes with elements with aspect ratio as high as 100:1 to adequately resolve anisotropic flow features such as boundary layers. While resolution is essential in the wall-normal direction, a coarser discretisation is desirable in the tangential direction to keep computational costs low. This leads to highly anisotropic elements in regions with strong boundary curvature. Similarly, the anisotropic nature of plasma heat transport in the tokamak edge region would benefit significantly from permitting anisotropic elements to align with the thermal gradients.

Highly stretched elements near curved boundaries are a challenge for high-order mesh generators. Indeed, the curving of boundary edges and faces of highly stretched elements leads to self-intersection and, consequently, invalidity. One possible way to make these elements valid again is to optimise them using the variational framework as done in report [1]. This can be challenging, impractical and computationally expensive due to the strong curvature and the large number of layers. Another approach consists of starting with a coarse boundary layer mesh with mostly isotropic elements and then refining it. This coarse layer can itself be opti-

mised using the variational optimiser, at a much lower cost, before splitting. We describe this idea of isoparametric boundary layer splitting in this section.

This approach was first proposed in [4]. The idea is to generate the coarse linear boundary layer mesh that can be easily curved to obtain valid high-order elements. Using the isoparametric mapping from reference space to physical space, we are able to introduce subdivisions in these coarse high-order boundary layer elements along the wall-normal direction. The subdivided elements are guaranteed to be valid themselves if the mapping satisfies certain criteria, as described in [4, 5]. This allows us to generate arbitrarily high aspect-ratio elements in the boundary layer region.

For valid high-order elements, there exists a bijective mapping $\chi : \Omega_{\text{st}} \to \Omega$, defined between a reference element Ω_{st} and a given high-order element Ω. The reference element Ω_{st} itself can be split into a stack of sub-elements. The mapping χ can then be applied to these to project the sub-elements to physical space. Figure 2 demonstrates the process for a simple two-dimensional quadrilateral element.

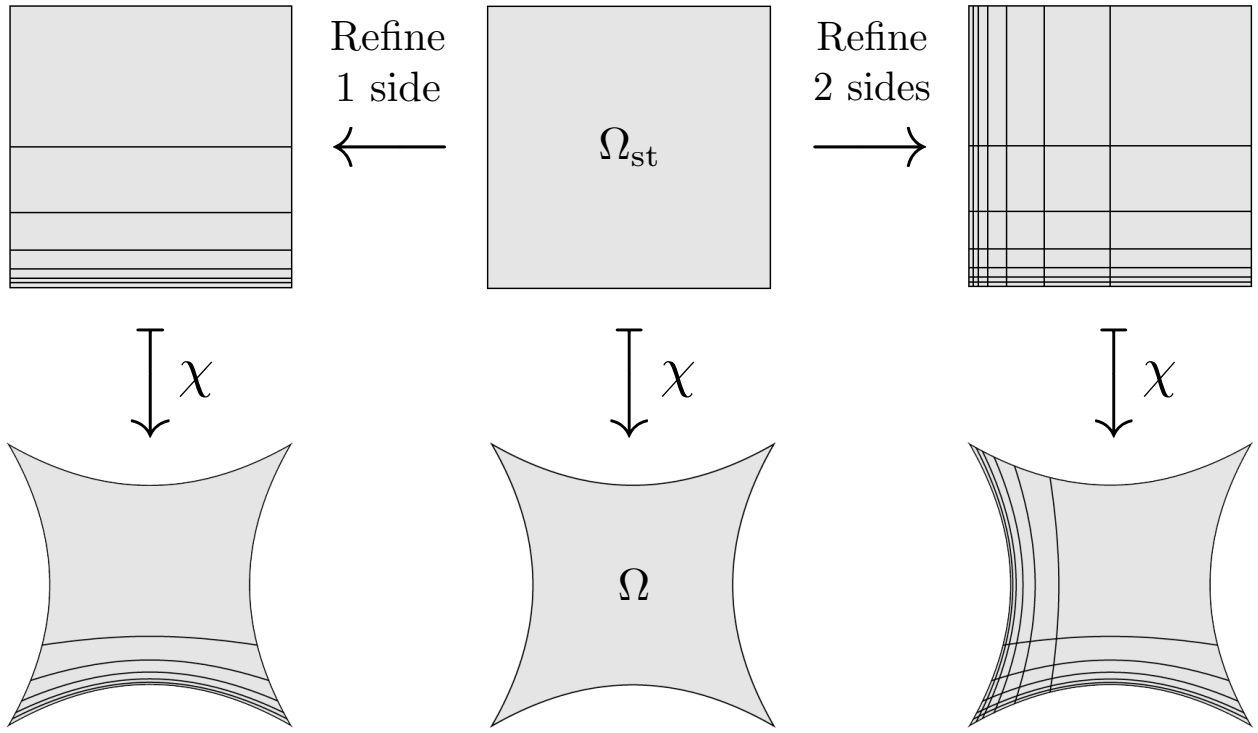


Figure 2: Schematic of the isoparametric splitting of a simple two-dimensional quadrilateral in one and two directions. Reproduced from [6].

The quality and validity of the split mesh is especially important here as highly stretched, curved elements are generated. We note that there exists an orientation-preserving affine mapping $f : \Omega_{\text{st}} \to \widetilde{\Omega}_{\text{st}}$ that represents the splitting of the reference element Ω_{st} into a reference sub-element $\widetilde{\Omega}_{\text{st}}$. The projection of the reference sub-element $\widetilde{\Omega}_{\text{st}}$ onto physical space then becomes the composition $f \circ \chi : \Omega_{\text{st}} \to \widetilde{\Omega}$, where $\widetilde{\Omega}$ is the curved sub-element. If both mappings, f and χ, are valid, their combination is valid too. In other words, if f is defined in such a way that the determinant of its Jacobian $J_f(\xi) > 0$ for all $\xi \in \Omega_{\text{st}}$, then the sub-elements are validly curved.

This approach constitutes the core of the `ProcessBL` module. Several refinements have been explored since first being described for simple prism splitting in [4]. For instance, in reference

[5] the approach was extended to the splitting of curved boundary layer prisms into tetrahedra, implemented in `ProcessTetSplit`, for tetrahedra-only solvers.

3.2.1 Issues and work in progress

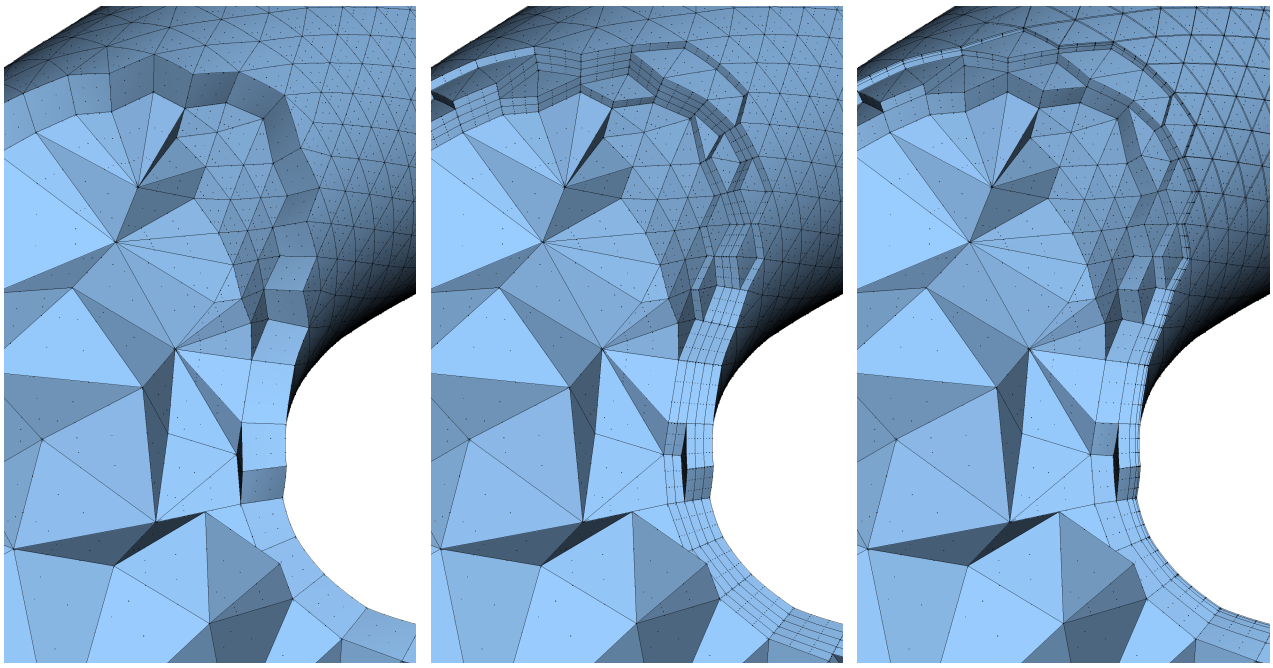
As with the linear meshing, `ProcessBL` was originally designed with only open boundaries in mind where a boundary layer is generated only on a single surface. The identification of elements that require splitting is done by a map of a pair of integers `std::map<int, int> m_boundaryLinks` in the element class `NekMesh/MeshElements/Element.h`, which creates a pairing between the 1D element on the CAD curve where a boundary layer is generated to a 2D element on the surface mesh. This means that the edges on the 1D curve that were used to generate the linear boundary layer can be linked to only a single 2D element ID (whichever one is set latest in the loop over 2D elements) and hence the splitting occurs only on one of the two faces instead of both, as briefly demonstrated by the algorithm below. This is the next major issue to be addressed. Once resolved, we should obtain a fully working prototype capable of generating improved meshes of the tokamak edge region such as the mesh demonstrated in Deliverable D1.1 [1], with the advantage of quadrilateral elements in the refinement region and improved control over the meshing parameters for the user.

Exerts from `Module::ProcessEdges` where the 1D elements on the edges are linked to 2D elements on the faces.

```
void Module::ProcessEdges(bool ReprocessEdges)
{

  ...

  // Create links for 1D elements
  for (int i = 0; i < m_mesh->m_element[1].size(); ++i)
  {
    // Update 2D element boundary map.
    pair<weak_ptr<Element>, int> eMap = (*it)->m_elLink.at(0);
    eMap.first.lock()->SetBoundaryLink(eMap.second, i);

    ...

  }

  ...

}
```

3.3 Examples of existing functionality

In *NekMesh*, the isoparametric splitting is typically carried out according to user-defined criteria. These criteria currently are: the number of subdivisions, or layers, along the wall-normal direction; and a growth, or progression, rate for the height of the elements along this same direction — this rate is defined by a factor r, corresponding to the ratio of heights of any two adjacent layers. Figure 3 shows an example of the isoparametric splitting module. The coarse boundary layer mesh of figure 3(a) is split into 5 layers with a progression rate $r = 1$ in



(a) Unsplit. (b) Split, $r = 1$. (c) Split, $r = 2$.

Figure 3: Isoparametric splitting of a coarse boundary layer mesh.

figure 3(b) and $r = 2$ in figure 3(c).

4 Curved 2D quadrilateral meshing

In addition to the isoparametric boundary layer meshing, *NekMesh* contains an experimental branch, *WIP: Cross field solver*, that contains full 2D quadrilateral mesh generation functionality using an adaptive spectral element solver [7]. The approach is inspired by quadrilateral mesh generation methods in the computer graphics community which rely on the idea of a *cross field*. In the *NekMesh* setting the process is roughly as follows: first a spectral element Laplace solver uses a triangular mesh (which is itself generated with *NekMesh* as outlined in D1.3 [2]) to compute a guiding field; the guiding field is then analysed to construct a separatrix graph based on the streamlines; finally, the CAD domain is split into quadrilateral blocks using splines of the streamlines which are trivial to mesh.

From a user perspective, the process requires four steps:

1. Generation in *NekMesh* of a triangular mesh as base for the cross/guiding field solution. At the same time, we export the list of geometry vertices, needed later to trace streamlines.
2. Solution of the cross / guiding field equations. This is done using a modified Laplace solve (in the ADR solver), which overwrites the boundary condition based on the geometrical orientation of the boundaries.
3. Processing of the field in *FieldConvert*, comprising 6 steps: import of geometry vertices; location of interior singularities; calculation of the valence of all vertices and singularities; tracing of streamlines; optional manipulation/merging of streamlines; export of streamlines.
4. Generation of the quadrilateral mesh in *NekMesh*, comprising 5 steps: re-import of the initial geometry; import of streamlines and splining; split of the geometry using the

streamlines; trivial meshing of the multi-surface geometry using 1 quad each; optional splitting of the quad mesh using the isoparametric approach

Our aim is to merge this branch with the work done so far on refinement, r-adaption and isoparametric boundary layer meshing, validate it on the tokamak edge case 2D geometry and better integrate it into the workflow.

While the approach is currently limited to 2D and is not trivially applicable to 3D, it could be used to generate quad-based surface meshes instead of triangular ones. This in turn will enable us to use the quad surfaces as the basis for the isoparametric boundary layer to generate hexahedrals instead of prisms. As explained previously, this will provide hexahedrals in the most dense regions of the mesh which in turn will benefit the matrix-free operators and significantly improve performance, despite providing an unstructured mixed element meshes rather than fully hexahedral.

5 Conclusions, work in progress, and future work

The inclusion of quad-based meshes within *NekMesh* is a challenging prospect. Utilising the isoparametric boundary layer meshing capabilities, we have shown the prospect for a process for generating quad-based elements in the most refined regions of the mesh. This approach offers a two-fold advantage: first is the ability to better control the refinement parameters; second is the advantages the quad-based elements will provide for the matrix-free operators developed for tasks 2.2 and 2.3.

However, *NekMesh* has been designed primarily with aeronautical applications in mind, and as such makes many assumptions about the type of geometries it can deal with. This has led to significant changes being required to data structures and logic to adapt the algorithms to deal with embedded curves within the domain when trying to use the isoparametric boundary layer splitting. Most of the issues have been identified and preliminary work to resolve most of them has been done on a case-by-case basis with concepts in place for a robust implementation to follow.

Additionally, a framework to provide fully quadrilateral 2D meshes has been presented. This method also shows promise for generating quadrilateral surfaces for unstructured mixed hexahedral / tetrahedral meshes in 3D by utilising the isoparametric boundary layer splitting method.

As such, our next steps are ordered as follows:

1. Resolve issue with 2D element splitting taking place on only one surface.
2. Improve robustness and performance of the 2D boundary layer generation implementation when applied to embedded curves and geometries with multiple faces.
3. Provide verified code to UKAEA with full documentation and examples, showing the simplified process of generating anisotropic meshes for the tokamak edge region.
4. Update tutorial for anisotropic heat transfer in deliverable 3.1 to include the boundary layer mesh generation.
5. Integrate the 2D fully quadrilateral mesh generation process into the tool chain.
6. Investigate the use of the 2D fully quadrilateral methodology to generate surface meshes for 3D geometries.

7. Combine the CAD based refinement, r-adaptation and isoparametric boundary layer splitting approaches with the quadrilateral surfaces for the generation of unstructured mixed element 3D meshes.

References

[1] M. Green, D. Moxey, C. Cantwell, B. Liu, and S. Sherwin. NEPTUNE Report for D1.1: r-adapted meshes for the tokamak edge region. Technical Report D1.1, King's College London & Imperial College London, February 2022.

[2] D. Moxey, S. Sherwin, and C. Cantwell. NEPTUNE Report for D1.3: Surface Mesh Generation. Technical Report D1.3, University of Exeter & Imperial College London, May 2021.

[3] Romain Aubry and Rainald Löhner. On the 'most normal' normal. *Communications in Numerical Methods in Engineering*, 24(12):1641–1652, 2008.

[4] D. Moxey, M. D. Green, S. J. Sherwin, and J. Peiró. An isoparametric approach to high-order curvilinear boundary-layer meshing. *Comp. Meth. Appl. Mech. Eng.*, 283:636–650, 2015.

[5] D. Moxey, M. D. Green, S. J. Sherwin, and J. Peiró. On the generation of curvilinear meshes through subdivision of isoparametric elements. In *New Challenges in Grid Generation and Adaptivity for Scientific Computing*, pages 203–215. Springer, 2015.

[6] J. Marcon, M. Turner, J. Peiró, D. Moxey, C. R. Pollard, H. Bucklow, and M. Gammon. High-order curvilinear hybrid mesh generation for CFD simulations. In *2018 AIAA Aerospace Sciences Meeting*, 2018.

[7] Julian Marcon, David A. Kopriva, Spencer J. Sherwin, and Joaquim Peiró. A high resolution pde approach to quadrilateral mesh generation. *Journal of Computational Physics*, 399:108918, 2019.