Report 2048465-TN-02 D1.1: r-adapted meshes for the tokamak edge region

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

28th March 2022

Contents

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 second report that comprises Deliverable 1.1 of our workplan, we extend the two-dimensional high-order mesh generation process for producing meshes for the tokamak edge region, with a particular focus on anisotropic element clustering leveraging the r-adaptation method.

2 Introduction

The main objective of Task 1.1 is to modify *NekMesh* with capabilities to generate high-order meshes with adequate resolution for modelling the X-point, plasma separatrix and, if required, the flux surfaces in the tokamak edge region. It is paramount that the refined mesh elements conform exactly to the geometry of these internal features. We propose the use of internal CAD curves which will guide the mesh generation process in *NekMesh* on how refine and adapt the mesh appropriately, using modified mesh refinement techniques and leveraging the r-adaptation scheme of [1] within the variational framework outlined in [2].

This report discusses the extensions added to *NekMesh* to achieve the goals stated above beyond those already reported on in our previous deliverable [3]. We assume the reader is familiar with the processes described there and, in order to reduce the length of this report, we do not repeat the same information here and will refer to specific sections of the previous report instead.

The report is structured as follows: first, we discuss the technical modifications implemented in *NekMesh* for both the mesh refinement and r-adaptation in Section 3, followed by detailed user documentation of how to use the new features in Section 4. Two examples are then provided in Section 5: first of a simple circle in a square, where the CAD preparation and pipeline process are discussed in detail, followed by a prototype high-order mesh of the tokamak edge region with the plasma separatrix and X-points used for refinement and clustering of mesh elements. Finally, conclusions and future work are discussed in Section 6.

3 New features in *NekMesh*

To achieve the requirements specified above, two additional features have been added to *NekMesh*. The first extends the ability to specify mesh spacing along CAD curves which increases the mesh density in a desired regions, as explained in Section 3.1. The second additional feature uses the r-adaptation scheme, initially developed for shock capturing in compressible fluid simulations. This process shifts mesh nodes towards regions of interest, whilst preserving the connectivity and the number of degrees of freedom in the mesh. This scheme is modified to shift the mesh towards user-selected CAD curves, resulting in anisotropic elements that conform exactly the curve. These changes are documented in Section 3.2.

3.1 CAD curve defined refinement

The current release version of *NekMesh* uses an automatic refinement technique based on curvature where the mesh spacing is obtained with the help of an underlying octree, as explained in Section 3.2 of [3] and in more detail in reference [4]. In addition to the automatic mesh spacing, an option to define *refinement lines* in the mesh is provided where a user can specify two points, the mesh spacing, D, and a radius of influence around the line, R, to provide additional control of the mesh refinement to the user. To achieve the objectives of task 1.1, it is paramount to provide additional refinement control where D and R can be specified over more complex geometries than just straight lines. As such, we extended this functionality to work on any CAD edge or curve.

The approach we have taken is similar to the line refinement technique and informs the surface meshing routines of the minimum mesh spacing between that obtained by the octree and the manually specified one in the defined region. When the surface mesh is generated using the adapted version of the code *Triangle*, it queries the octree where the triangle is to be formed for its spacing, δ, which is obtained using equation (1) in report D1.3 [3]. We note that δ is defined

by the curvature of the CAD curve to set the mesh spacing not only for triangles formed on the CAD curve but for all triangles within the same octant. The user defined refinement adds an additional constraint on the mesh spacing, where if any of the triangle's vertices are a distance of R or less from a line or a CAD curve, the triangle's spacing is the minimum between the specified value and the one assigned from the octree, i.e. $\delta = min(D, \delta)$.

Querying the octree for the mesh spacing in the `MeshGen` module

```
NekDouble Octree::Query(Array<OneD, NekDouble> loc)
{
  // starting at master octant 0 move through succsesive m_octants which
  // contain the point loc until a leaf is found.

  // first search through sourcepoints
  NekDouble tmp = numeric_limits<double>::max();

  // search line sourcepoints
  for (int i = 0; i < m_lsources.size(); i++)
  {
    if (m_lsources[i].withinRange(loc))
    {
      tmp = min(m_lsources[i].delta, tmp);
    }
  }

  // search curve sourcepoints
  for (int i = 0; i < m_csources.size(); i++)
  {
    if (m_csources[i].withinRange(loc))
    {
      tmp = min(m_csources[i].delta, tmp);
    }
  }

  OctantSharedPtr n = m_masteroct;
  bool found = false;

  while (!found)
  {
    // ... A lot more code to find the leaf octant ...
    n = n->GetChild(quad);
    if (n->IsLeaf())
    {
      found = true;
    }
  }
  // obtain the minimum delta between the leaf octant and the refinement
  // lines / curves
  return min(n->GetDelta(), tmp);
}
```

The query uses a new struct for the CAD curves used as refinement sources.

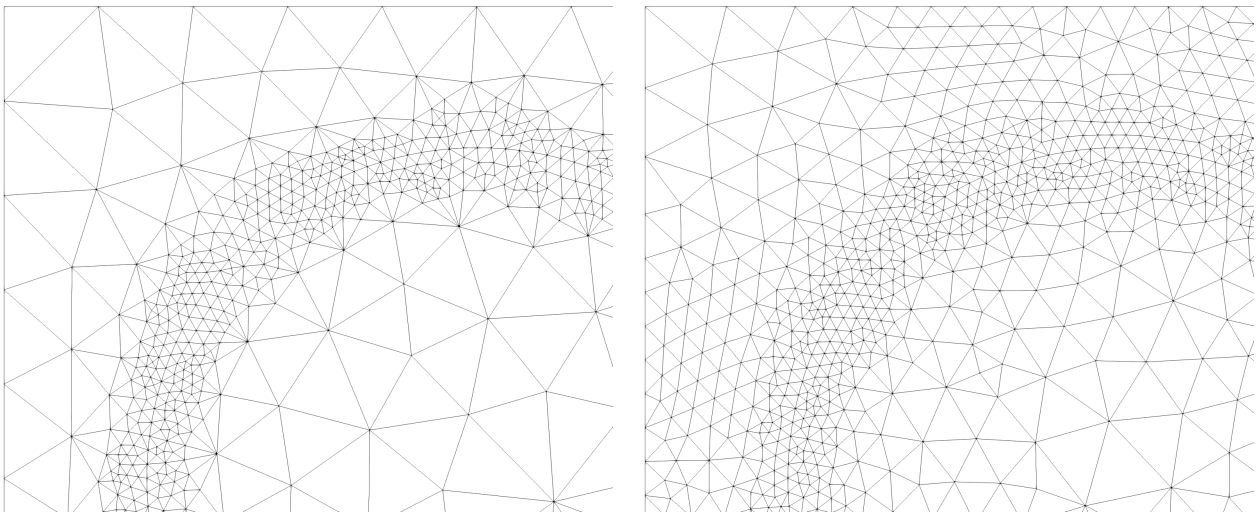
`curvesource` struct used by the `Octree` in the `MeshGen` module

```
struct curvesource
{
  CADCurveSharedPtr curve;
  NekDouble R, delta;

  curvesource(CADCurveSharedPtr c, NekDouble r, NekDouble d)
      : curve(c), R(r), delta(d)
  {
  }

  // Tests if a point is within a specified range, R, from the curve
  bool withinRange(Array<OneD, NekDouble> p)
  {
    if (curve->GetMinDistance(p) <= R)
    {
      return true;
    }
    else
    {
      return false;
    }
  }
};
```

We note that this modification required the ability to obtain distances from an arbitrary point in the domain to a CAD curve, implemented in the CAD engine interface of *NekMesh* as a function `CADCurveOCE::GetMinDistance`. Although *NekMesh* is designed to support multiple backends (specifically OpenCASCADE and the ITI interface CFI), at present we have only implemented this capability in the OpenCASCADE backend and hence is not currently supported in other CAD systems.

Obtaining the minimum distance of a point from the CAD curve.

```
// Nektar++ distances are in m, OpenCASCADE are in mm
NekDouble CADCurveOCE::GetMinDistance(Array<OneD, NekDouble> xyz)
{
    gp_Pnt loc(xyz[0] * 1000.0, xyz[1] * 1000.0, xyz[2] * 1000.0);
    GeomAPI_ProjectPointOnCurve proj(loc, m_c, m_b[0], m_b[1]);
    if(proj.NbPoints())
    {
      return proj.LowerDistance() / 1000.0;
    }
    else
    {
      return std::min(loc.Distance(m_c->Value(m_b[0])),
      loc.Distance(m_c->Value(m_b[1]))) / 1000.0;
```



(a) (b)

Figure 1: Portion of a mesh using curve refinement (a) without octree manipulation; (b) with octree manipulation.

```
    }
}
```

A limitation of the above approach is that it works *on-top* of the underlying octree structure (by specifying the minimum spacing in a specific region) which often results in an undesirably abrupt decrease in mesh spacing, and consequently some level of anisotrapy, as can be seen in Figure 1(a). To rectify this issue and obtain better meshes, we have implemented additional logic to manipulate the underlying octree structure. This is done by encouraging the octants where a refinement line or curve is found to continue subdividing until their spacing is equal to or smaller than the user defined spacing D of the line or CAD curve. This results in a more gradual decrease of mesh spacing at the expense of additional degrees of freedom as seen in Figure 1(b). We note that the octree manipulation exerts the influence of the refinement beyond the strictly defined region in a manner that is hard to predict due to the nature of the octree's subdivision process. The source code for this change is heavily fragmented and left out of this report. It required significant changes to the source point structs in `Octree/SourcePoints.hpp` and most notably the algorithms in `Octree::CompileSourcePointList()` and `Octant::Octant`.

3.2 CAD curve based r-adaptation

r-adaptation is a novel scheme available in *NekMesh*, whereby the position of mesh vertices are adjusted to obtain additional resolution where it is needed in the domain without changing the connectivity of the mesh, consequently fixing the number of degrees of freedom. The application of this process can be seen in [1], where this was combined with p-adaptation to better resolve discontinuities in the flow region by *squeezing* elements toward the shock region, resulting in anisotropic elements surrounding the shock structure. The method utilises the variational mesh optimisation technique [2] briefly described in Section 3.3.3 of report D1.3 [3], while periodically scaling the Jacobian of the element mappings based on the discontinuity found in the solution field during the optimisation iterations.

For our purposes of generating highly anisotropic mesh elements conforming to the shape of the plasma core separatrix and flux surfaces, we have adapted the r-adaptation method to be able

to scale and shift elements towards user specified CAD curves. In this way the user can control *a priori* where additional resolution will be required. The user needs to specify the scaling factor and, optionally, a radius of influence around the curve where elements are scaled. This process is aided by the ability of the optimisation procedure to *slide* mesh nodes (excluding the end vertices which remain fixed) along CAD curves which ensures the mesh remains conformal to the curve during the adaptation process, thus exactly capturing the desired feature within the domain.

To allow the mesh nodes more freedom of movement, it is best to apply the r-adaptation on a linear mesh and elevate it to a high order mesh after the adaptation procedure. The r-adaptation procedure when applied to the CAD curves using a linear mesh however is very sensitive to the scaling factor and radius of influence, which are problem specific parameters that require manually tuning. If the scaling factor is too small, this can result in such highly stretched elements that the high-order optimisation scheme cannot be guaranteed to resolve under lower resolutions. Thus, depending on the desired level of anisotropy, a finer resolution might be required using the CAD curve refinement process described above in Section 3.1. Some discussion on safe values found through trial and error can be found in Section 5. However, a detailed study of the parameters is left for future work, for the main reason that in upcoming Task 1.2, we will consider quad-based mesh generation approaches to this problem. One significant part of this is to utilise isoparametric boundary layer mesh generation in these regions, based on the work in [5]. This may give better control over the anisotropy required in this region.

From the implementation perspective, several changes to the elements class of the optimisation module, `ElUtil`, were required. The current function where the mapping is updated was overhauled to become CAD aware and to allow for either the shock-based or the CAD-based adaptation to work seamlessly. The CAD-based r-adaptation was further split for performance reasons into cases where elements have a vertex on a CAD curve and those where a radius of influence extending from the curve is defined. In the latter, a brute force approach is utilised to check the distance of each vertex of each element from the CAD curve.

`UpdateMapping` function using CAD curves inside the `ProcessVarOpti` module's elements utility `ElUtil`

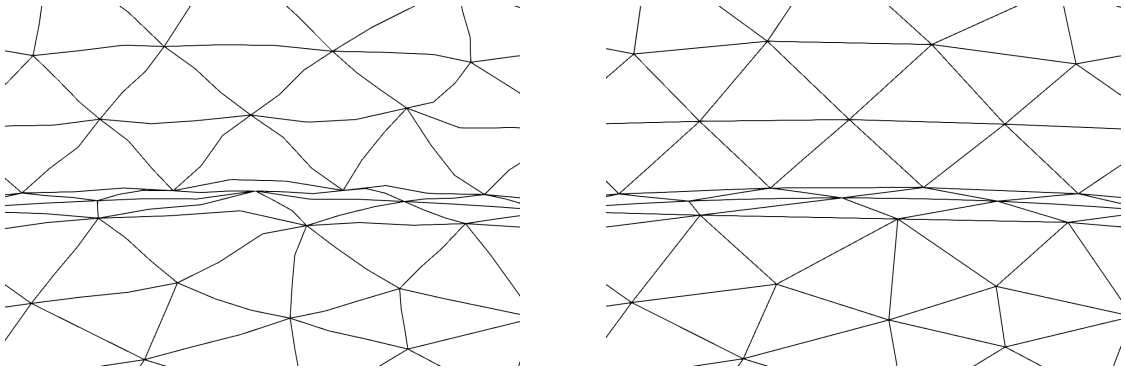
```
if (m_adapt_scale)
{
    // Using elements with a node with a radius from the curve
    if(m_adapt_radius)
    {
        [&]{
            for (auto &vert :m_el->GetVertexList())
            {
                Array<OneD, NekDouble> x(3);
                x[0]   = vert->m_x;
                x[1]   = vert->m_y;
                x[2]   = vert->m_z;
                for (auto &curve : m_adaptcurves)
                {
                    if (curve->GetMinDistance(x) < m_adapt_radius)
                    {
                        scaling = m_adapt_scale;
```

```
                        return;  // return lambda function to break loop
                    }
                }
            }
        }();
    }
    // Using only elements with a node on the curve
    else
    {
        [&]{
            for (auto &vert :m_el->GetVertexList())
            {
                for (auto &curve : vert->GetCADCurves())
                {
                    std::vector<CADCurveSharedPtr>::iterator it =
                        std::find(m_adaptcurves.begin(), m_adaptcurves.end(),
↪   curve);
                    if (it != m_adaptcurves.end())
                    {
                        scaling = m_adapt_scale;
                        return;  // return lambda function to break loop
                    }
                }
            }
        }();
    }
}

if (scaling)
{
    for (int i = 0; i < m_maps.size(); ++i)
    {
        for (int j = 0; j < 9; ++j)
        {
            maps[i][j]    = m_maps[i][j] / scaling;
            mapsStd[i][j] = m_mapsStd[i][j] / scaling;
        }

        if (m_dim == 2)
        {
            maps[i][9]    = m_maps[i][9] * scaling * scaling;
            mapsStd[i][9] = m_mapsStd[i][9] * scaling * scaling;
        }
        else if (m_dim == 3)
        {
            maps[i][9]    = m_maps[i][9] * scaling * scaling * scaling;
            mapsStd[i][9] = m_mapsStd[i][9] * scaling * scaling * scaling;
        }
        else
```

```
        {
            ASSERTL0(false, "not coded");
        }
    }
}
else
{
    maps = m_maps;
    mapsStd = m_mapsStd;
}
```

This approach is suitable for the prototype phase, but further improvements such as utilising the octree structure to reduce the search space can be made. However, within the 2D framework considered in this report, and taking into account the adaptation process that takes place on a linear mesh, we found the performance to be acceptable.

Additionally, a bug related to the identification of nodes that are attached to the CAD curves was found, whereby the end vertices of CAD curves were not identified and were allowed to be move as regular optimisation nodes (i.e. not attached to a CAD curve). Since the mesh is generated in a bottom-up fashion, this can lead to some elements no longer conforming to the curve and/or general degradation of the mesh quality, as demonstrated by Figure 2(a). To address this issue, the end vertices of CAD curve are now removed from the optimisation and are considered to be permanently fixed in space. This avoids any degradation of the geometry during the optimisation process, as seen in 2(b).



(a) (b)

Figure 2: Section of mesh showing the intersection between two edges after applying r-refinement to a linear mesh then upgrading it to higher order: (a) bug - CAD curve vertices unidentified and allowed to move; (b) bug fix - CAD curve vertices fixed in space.

4 User documentation

In the following sections, a description of how to use the additional *NekMesh* features is provided. First, the CAD requirements for generating meshes with internal curves is given followed by a description of the additional fields used in the *NekMesh* configuration *.mcf* file used for the curve refinement feature. Finally the additional command line options used for the curve-based r-adaptation feature are outlined.

4.1 CAD requirement for *NekMesh*

NekMesh mainly supports the OpenCASCADE CAD system [6] and the use of *STEP* files. A discussion on the CAD representation can be found in Section 3.1 of report D1.3 [3]. Here, we discuss the requirements for generating a valid STEP file for *NekMesh* with internal edges (or curves). CAD curves read into *NekMesh* must:

1. not be periodic, i.e. has distinct two distinct end vertices, or in other words that for the parametrisation of the curve, $r(t)$, there does not exist a constant T such that $r(t+T) = r(t)$;
2. form an edge loop, i.e. both its end vertices must also be an end vertex of another curve, or in other words it is part of a set of edges that together form a closed loop.

The latter restriction also implies that the internal curves have to split the domain into multiple unique faces that share some of the same edges with the neighbouring faces. This requires the decomposition of the computational domain in a particular manner that may not seem intuitive from the CAD software perspective. Two examples, one of a simple circle in a square and one of a geometry more representative of the cross section of tokamak reactor with the plasma separatrix are provided in Sections 5.1 and 5.2 respectively.

4.2 Configuration file (*.mcf*) for refinement definitions

The *.mcf* file is the configuration file for the import module in *NekMesh*. It uses a *xml* file formal. In principle, for most aeronautics applications, meshing is done mostly automatically and only a few basic options have to be explicitly provided. As an example, a typical basic 2D *.mcf* file would contain the following:

```
<NEKTAR>
    <MESHING>
        <INFORMATION>
            <I PROPERTY="CADFile" VALUE="case.stp"/>
            <I PROPERTY="MeshType" VALUE="2D"/>
        </INFORMATION>

        <PARAMETERS>
            <P PARAM="MinDelta" VALUE="0.01"/>
            <P PARAM="MaxDelta" VALUE="0.01"/>
            <P PARAM="EPS"      VALUE="0.1"/>
            <P PARAM="Order"    VALUE="4"/>
        </PARAMETERS>
    </MESHING>
</NEKTAR>
```

where the parameters for `MinDelta`, `MaxDelta` and `EPS` are explained in Section 3.2 of report D1.3 [3] and `Order` is the order of the mesh, in this example 4th order. For more advanced meshes containing boundary layers, additional options can be defined, which are described in the *NekMesh* user guide [7].

Refinement lines use a separate XML tag with the following format, where the values follow the same notation use in Section 3.1. The following example shows how a refinement line between

points (x_1, y_1, z_1) and (x_2, y_2, z_2) is defined with radius $R = 0.005$, wherein the mesh spacing is forced to a value $\delta = 0.001$.

```
<REFINEMENT>
    <LINE>
        <X1> -0.01 </X1>
        <Y1> -0.01 </Y1>
        <Z1> 0.0 </Z1>
        <X2> 0.01 </X2>
        <Y2> 0.01 </Y2>
        <Z2> 0.0 </Z2>
        <R>  0.005 </R>
        <D>  0.001 </D>
    </LINE>
</REFINEMENT>
```

For consistency reasons, we have followed the same format for CAD curve refinement, except we have replaced the start and end points of the line with the edge ID as defined in the *STEP* file.

```
<REFINEMENT>
    <CURVE>
        <ID> 1 </ID>
        <R> 0.005 </R>
        <D> 0.001 </D>
    </CURVE>
</REFINEMENT>
```

Any number of lines and curves can be specified within the same refinement tag in any order, one at a time. An example using several curves and lines can be found in the provided plasma core separatrix test case attached to this report.

4.3 Command line options for r-adaptation

In contrast to the curve refinement which is read by the input module, r-adaptation is part of the variational optimisation module, `varopti`. Its parameters are set using the command line options for `varopti`, which is important as it allows for multiple instances of the module to be piped to each other using different parameters.

The option for r-adaptation using curves is activated when a curve is specified using the option `radaptcurves` that can take a combination of curve ids in list and range format as follows:

```
NekMesh -m varopti:<optimisation_method>:radaptcurves=<int id_item1>,<int
 ↪  id_item2>,<int id_range1_start>-<int id_range1_end>:nq=<order_of_mesh>
```

where `optimisation_method` is the functional used for the optimisation process as described in [2] and `nq` is the number of integration points per element edge, which is equivalent to the order of the mesh plus 1, i.e. `nq=2` for a first order linear mesh and `nq=5` for a 4th order mesh.

Additionally, a scaling factor must be set for the adaptation to take effect, using the option

```
-m varopti:<previous_options>:radaptscale=<double scaling_factor>:...
```

Setting the radius is optional and can be set with

```
-m varopti:<previous_options>:radaptrad=<double scaling_radius_value>:...
```

Finally, an additional parameter for selecting the number of optimisation iterations between scaling updated can be set with

```
-m varopti:<previous_options>:subiter=<int number_of_iterations>
```

This option is required as applying the scaling too often can prevent the optimisation process to reach the desired tolerance.

Crucially, *NekMesh* can run a pipeline modules in a single command, so multiple passes through *varopti* can be easily run. For instance

```
NekMesh
  -m varopti:linearelsatic:radaptcurves=1-4:radaptscale=0.4:subiter=5:nq=2
  -m varopti:linearelsatic:radaptcurves=1-4:radaptscale=0.45:radaptrad=0.0
↪  05:subiter=15:nq=2
  -m varopti:hyperelastic:nq=5
  <case_name>.mcf <mesh_name>.xml
```

would be a typical way to use the r-adaptation process, where: the first use of `varopti` uses the linear elastic method for optimisation, curves 1-4 are selected for the adaptation, the scaling factor is 0.4 applied every 5th iteration and the optimisation is done on a linear mesh; the second use of `varopti` is identical to the first only with a larger scaling factor applied every 15th iteration to all elements with a vertex within 5 mm from the selected curves; the last call to `varopti` uses the hyper-elastic model on a 4th order mesh without applying any further adaptation and optimising the high order mesh nodes.

5 Test cases

The following section contains two test cases. The first is a simple circle embedded in a square used to illustrate in detail the pipeline process to generate a high-order anisotropic mesh refined and adapted towards the CAD curves of the circle. The second case resembles a tokamak cross section with the plasma separatrix. Due to the unavailability of the real geometry, this is our best attempt at recreating one by tracing an image in FreeCAD.

5.1 Circle in square

This test case consists of a square of width 100 mm with an embedded circle of radius 70 mm in its centre. Although simple, it serves as a demonstration of the entire pipeline, exploring the various modifications and their individual influences on the mesh generation process when considering internal curves for clustering elements in regions of interest.

Two files, a *STEP* file containing the CAD information, *Cases/2d_circle_square.stp*, and a configuration file, *Cases/2d_circle_square.mcf*, are attached to this report in order to recreate any steps of this example and made available as part of the *NekMesh* test suite.

5.1.1 CAD preparation

Following Section 4.1, the domain is split into two non-overlapping faces with eight distinct edges (see Figure 3). Edges 1–4 represent the curves of the circle and are present in both the square and the circle faces, as illustrated in Figures 4.

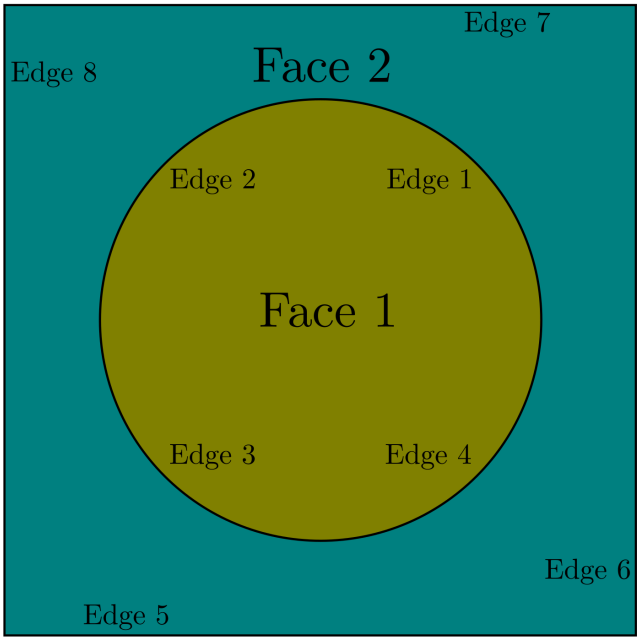


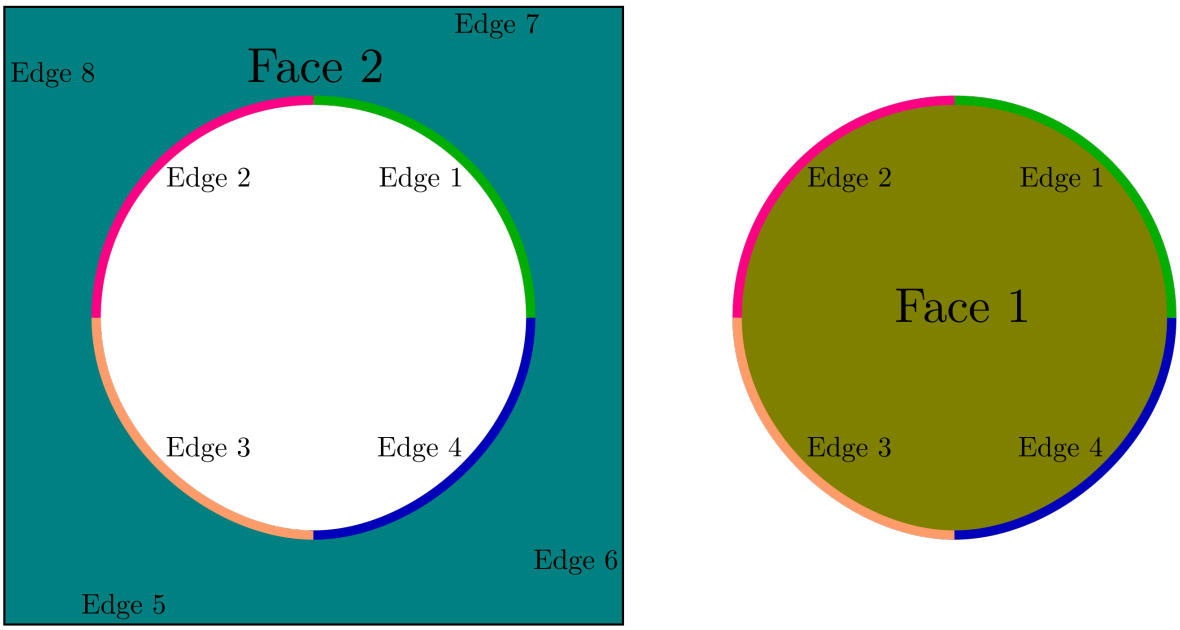
Figure 3: Illustration of the faces and edges of the circle in square test case.

5.1.2 Curve refinement

Meshing the above case with the unmodified *NekMesh* using a large initial mesh spacing of 10 mm, as seen in the *2d_circle_square.mcf* file below, we execute the following commands:

```
NekMesh -v -m varopti:hyperelastic:nq=5 2d_circle_square.mcf
 ↪  2d_circle_square.xml
```

```
<NEKTAR>
    <MESHING>

        <INFORMATION>
            <I PROPERTY="CADFile" VALUE="2d_circle-square.stp"/>
            <I PROPERTY="MeshType" VALUE="2D"/>
        </INFORMATION>

        <PARAMETERS>
            <P PARAM="MinDelta" VALUE="0.0015"/>
            <P PARAM="MaxDelta" VALUE="0.01"/>
            <P PARAM="EPS"      VALUE="0.1"/>
```



(a) (b)

Figure 4: Illustration of the separate faces and associated edges of the circle in square test case showing (a) the square region without the embedded circle; (b) the circle region without the surrounding square.

```
            <P PARAM="Order"     VALUE="4"/>
        </PARAMETERS>

    </MESHING>
</NEKTAR>
```

This results in a sparse 4th order mesh consisting of 294 elements and 4,470 degrees of freedom, as shown in Figure 5(a). We can see that no automatic refinement took place due to the curvature being adequately captured with the specified `MaxDelta` of 10 mm in mesh spacing, resulting in 10 discrete segments per edge of the square.

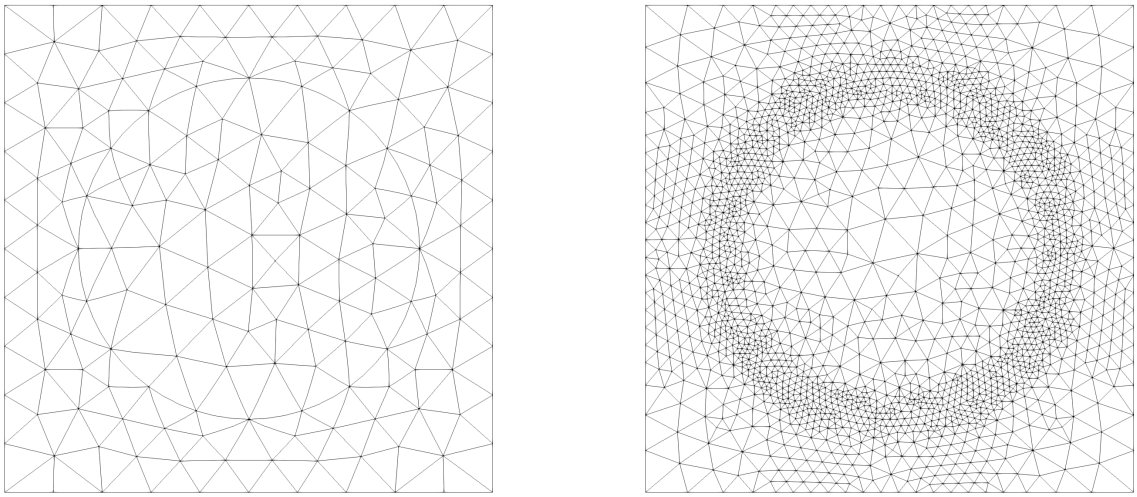
To better resolve the circle, we select curves 1–4 as refinement curves and set the mesh spacing to $D = 0.15$mm with a radius of influence $R = 0.5$ mm, by adding the following `<REFINEMENT>` tag to the *2d_circle_square.mcf*.

```
<REFINEMENT>
    <CURVE>
        <ID> 1 </ID>
        <R> 0.005 </R>
        <D> 0.0015 </D>
    </CURVE>
    <CURVE>
        <ID> 2 </ID>
        <R> 0.005 </R>
        <D> 0.0015 </D>
    </CURVE>
```

```
    <CURVE>
        <ID> 3 </ID>
        <R> 0.005 </R>
        <D> 0.0015 </D>
    </CURVE>
    <CURVE>
        <ID> 4 </ID>
        <R> 0.005 </R>
        <D> 0.0015 </D>
    </CURVE>
</REFINEMENT>
```

Note that in the provided configuration file the `<REFINEMENT>` tag is already present.

Using the modified *NekMesh* with the CAD curve refinement and executing the same command as the previous case with the unmodified code, we can see in Figure 5(b) that the resolution around the circle drastically increases. Additionally, the octree smoothing is exerting its influence throughout most of the domain, which further increases the resolution, resulting with a mesh containing 4,184 elements and 66,022 degrees of freedom.



(a) (b)

Figure 5: Mesh of the circle in square test case (a) the unmodified version of *NekMesh* with no refinement; (b) the modified version of *NekMesh* using refinement around the CAD curves of the circle.

5.1.3 r-adaptation

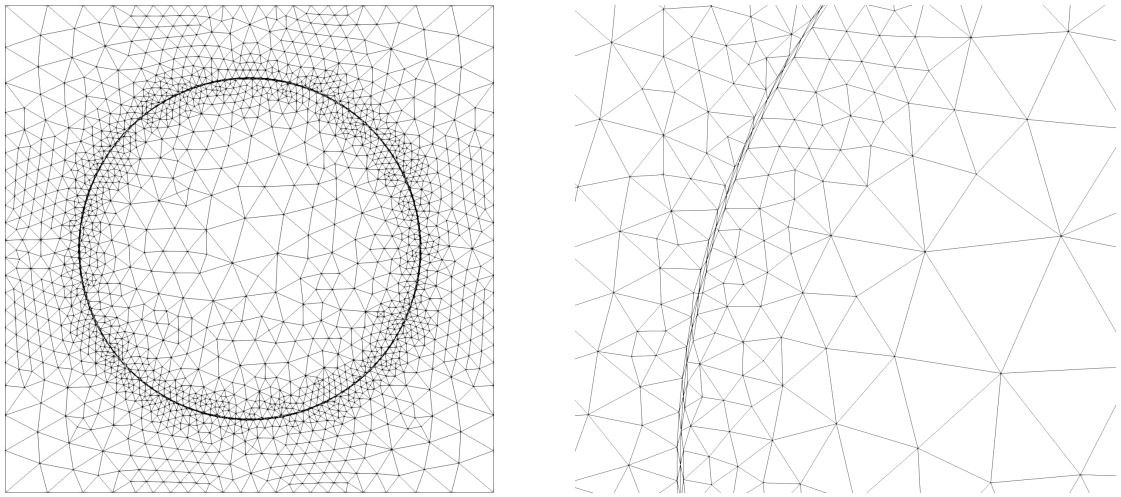
Despite the increase in resolution, the mesh produced by employing refinement is still isotropic and there is no clustering within the refined zone around the circle's curves. To achieve that, we employed the r-adaptation method, which shall be explored next. Keeping in mind Section 4.3, we will adopt a pipeline process using multiple executions of the variational optimisation module. The different parameters will be demonstrated in the following steps.

First, we attempt to generate a mesh using r-adaptation only on elements adjacent to the circle's curve, i.e. without specifying a radius. This is done with following command:

```
NekMesh -v
  -m varopti:linearelastic:radaptcurves=1-4:radaptscale=0.275:subiter=5:re⌋
↪  stol=1e-5:nq=2:numthreads=10
  -m varopti:hyperelastic:nq=5:numthreads=10
  2d_circle_square.mcf 2d_circle_square.xml
```

We note the use of additional options, namely `restol` and `numthreads` which were not previously explained. The former is the minimum residual required to halt the iterations in the variational optimisation process, and the latter is number of execution threads on a multi-core CPU. As explained in Section 3.2, the r-adaptation is executed on a linear mesh by using `nq=2`, followed by a second pass of the variational optimisation module without r-adaptation and using `nq=5` to promote the mesh to 4th order whilst ensuring the process of increasing the polynomial order does not degrade the mesh quality.

The resulting mesh contains the same number of elements and degrees of freedom as the refined mesh, but as seen in Figure 6, anisotropic mesh elements are now formed on the circle, while the elements connected to them are still isotropic, resulting in a sharp transition. The scaling factor of 0.275 was obtained by trial error by decreasing the value until the resulted elements were too skewed for the selected resolution.



(a) (b)

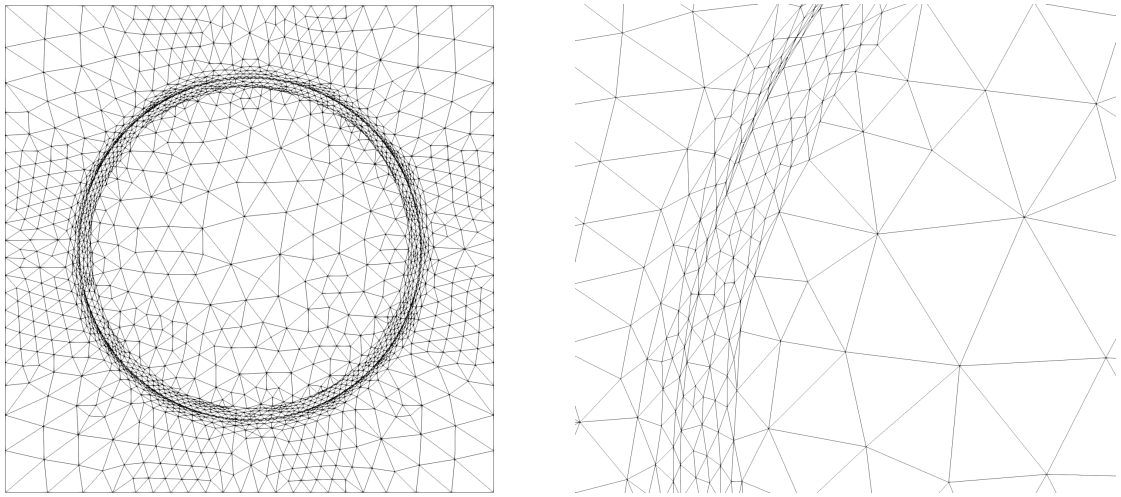
Figure 6: Circle in square mesh using curve refinement and r-adaptation applied only to elements adjacent to the circle's curve: (a) full view; (b) zoomed in view around a curve.

To address the sharp transition and to cluster more elements around the circle, a radius of influence of 0.2mm wide is used with the following commands:

```
NekMesh -v
  -m varopti:linearelastic:radaptcurves=1-4:radaptscale=0.275:radaptrad=0.⌋
↪  002:subiter=20:restol=1e-5:nq=2:numthreads=10
  -m varopti:hyperelastic:nq=5:numthreads=10
  2d_circle_square.mcf 2d_circle_square.xml
```

Once again, it was found that a scaling factor value smaller than 0.275 resulted in invalid

elements when upgrading the mesh to higher-order. As observed in Figure 7, the resulting mesh displays a wider band of anisotropic elements around the circle's curves, however the elements adjacent to the curves are less stretched than before. It is also noted that the value for `subiter` is relaxed from applying the scaling every 5 iterations to every 20 iterations. This is due to the scaling of more elements, which decreases the value of the functional being optimised within the variational setting (see reference [2]), and the desired residual set for the optimisation not being reached between the Jacobian mapping scaling updates.



(a) (b)

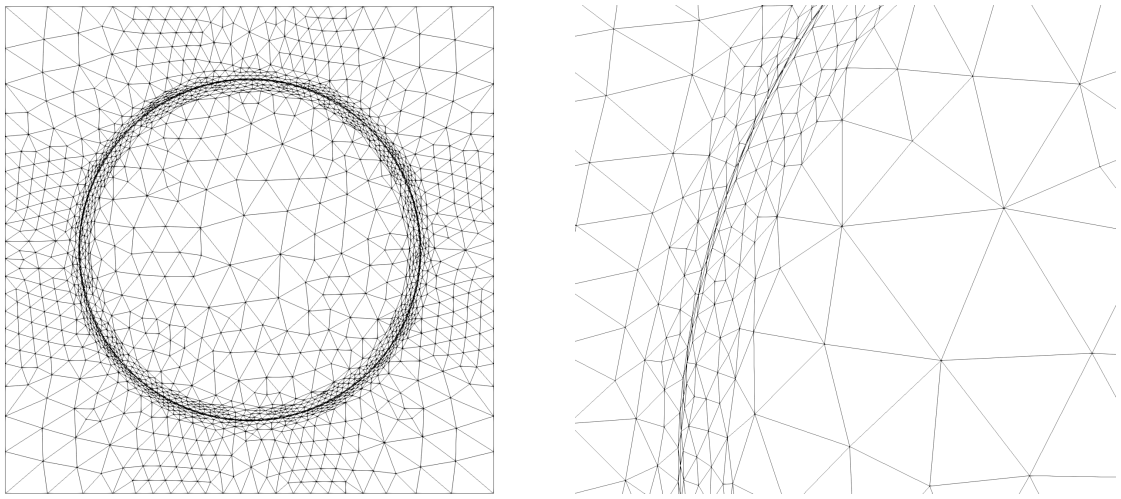
Figure 7: Circle in square mesh using curve refinement and r-adaptation applied to elements within a fixed radius from the circle's curve: (a) full view; (b) zoomed in view around a curve.

Ideally, we wish to combine the highly stretched anisotropic elements around the curve from the case without using a radius with the wider band of anisotropic elements obtained from applying the radius. To do so, we can chain both execution of the variational optimiser using r-adaptation, first without a radius and then with using the following commands:

```
NekMesh -v
  -m varopti:linearelastic:radaptcurves=1-4:radaptscale=0.4:subiter=5:rest
↪  ol=1e-5:nq=2:numthreads=10
  -m varopti:linearelastic:radaptcurves=1-4:radaptscale=0.4:radaptrad=0.00
↪  2:subiter=15:restol=1e-5:nq=2:numthreads=10
  -m varopti:hyperelastic:nq=5:numthreads=10
  2d_circle_square.mcf 2d_circle_square.xml
```

Finally, we obtain the desired mesh! As seen in Figure 8, the resulting mesh has both highly stretched elements adjacent to the curve as well as a band of anisotropic elements surrounding the curves with a smoother transition towards the isotropic elements in the rest of the domain. In this final trial, the scaling factor was increased to 0.4 to account for the fact that the scaling is applied twice to the elements adjacent to the curves.

To summarise, the pipeline process used to generate the desired meshes, starting from defining refinement curves in the *.mcf* file to the multiple executions of the variational optimisation module have been demonstrated, with each step in the pipeline explained. The values used in the different points in the pipeline are problem dependent, although from our experience



(a) (b)

Figure 8: Circle in square mesh using curve refinement and r-adaptation applied first only to elements adjacent to the circle's curves, then to elements within a fixed radius from the circle's curve: (a) full view; (b) zoomed in view around a curve.

the selected scaling factor values are similar for most cases and range between 0.3 to 0.65 when employing r-adaptation once without and once with a radius. Further reduction of the refinement mesh spacing can allow for the smaller limit of scaling factor range, although further studies into the relation between the various parameters is still required and left for future work as explained in Section 3.2.

5.2 Plasma core separatrix

The second test case is a schematic cross section of the tokamak first wall, where the CAD is a very rough and simplified approximation of the geometry obtained by tracing the illustration in Figure 9 in FreeCAD. We used mm as the units when tracing which gave an enclosing domain with a height of 222 mm and width of 106 mm. We are aware that this does not represent reality, however scaling the units is trivial for the mesh generation process.

Only the plasma separatrix of the edge-region was modelled internally. The flux surfaces were left out at this stage as: a) it is unclear if these would form part of the requirements; b) due to the CAD creating process, creating separate faces for each gap between the surfaces could be extremely time consuming and was judged to not be worth the effort without the availability of an accurate geometry. The resulting *STEP* file consists of three distinct and non-overlapping faces and thirty edges, shown in Figure 10. Edges 1, 2, 4 and 7–22 represent the walls, edge 3 the pump and edges 5, 6 and 23–30 the internal plasma separatrix. The latter group is mainly of interest for refinement and r-adaptation, however some outer edges also require refinement to enable a smooth mesh transition between the internal regions and the walls.

The two files for this test case are attached to this report in *Cases/cross_section.stp* for the CAD file and *Cases/cross_section.mcf* for the configuration file. It is highly recommended that the reader refers to the configuration file when following this section as the full content of the refinement definitions is too lengthy to include in the report.

The basic mesh configuration (excluding refinement) is similar to the previous test case. The minimum and maximum mesh spacing are both set to 10mm, noting that all refinement will

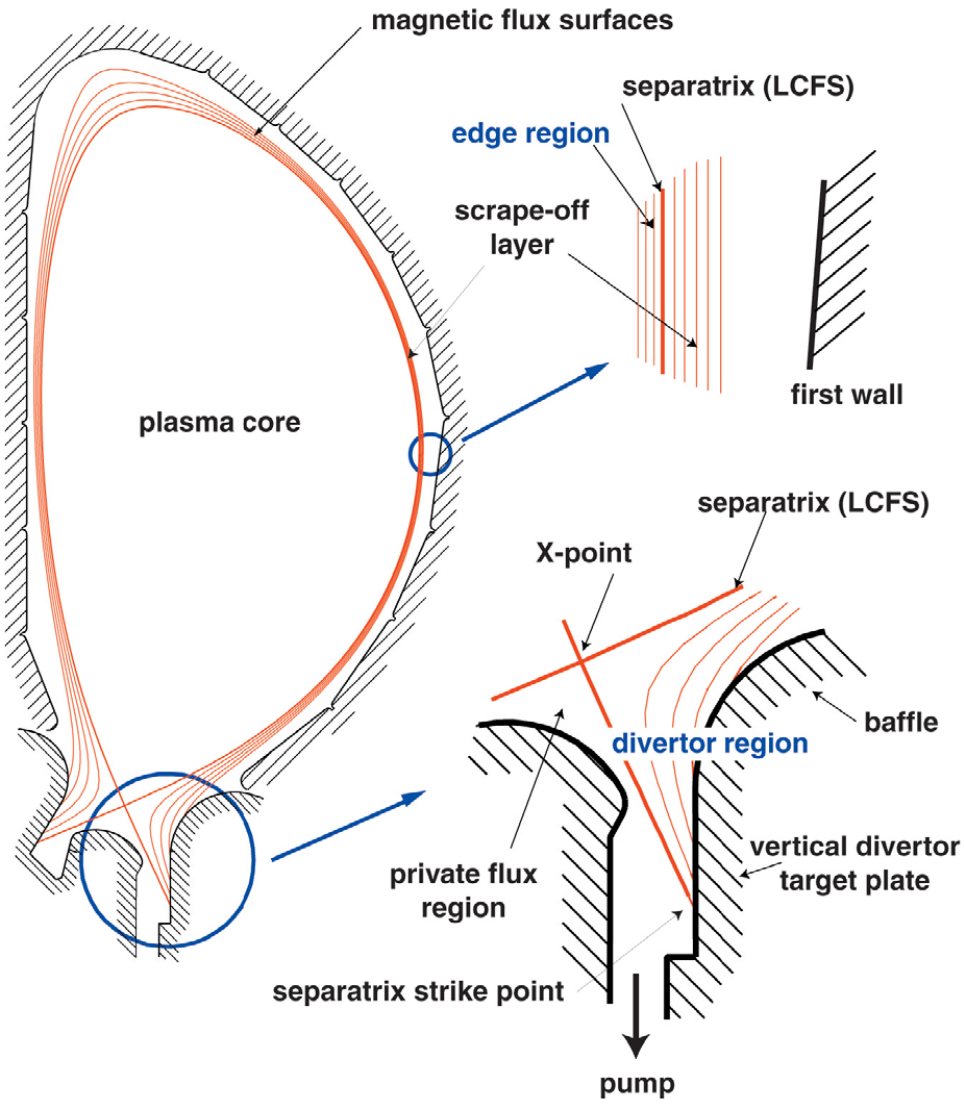


Figure 9: Illustrative example of a tokamak cross section. Taken from reference [8].

be user defined, and the $\epsilon = 1$. The order is set to 1 as we know in advance that refinement is to take place before the last pass of the variational optimisation upgrades the mesh to higher-order. Refinement is then applied to all internal edges, i.e. edges 5, 6 and 23–30 with a mesh spacing of 0.75 mm and a radius of influence of 3.5 mm. These parameters were carefully chosen so that the refinement zones around the internal curves do not contain parts of the outer edges while obtaining a smooth transition in the mesh. However, in the bottom region surrounding the X-point where the wall edges are in close proximity to the internal ones, further tuning was required. The wall edges were split into two groups with slightly different refinement parameters as follows: edges 1–4, 7–10, 12 and 13 form the first group of edges in close proximity the X-point, with a mesh spacing set to 1.5mm and different radius of influence values to obtain a smooth mesh; while edges 11 and 14 which are further away, as subscribed a mesh spacing of 0.25 mm, again with different radius values. Lastly, to further increase the resolution just around the X-point, two additional refinement lines were defined using a resolution of 0.2 mm as follows:

```
<LINE>
  <X1> -0.092 </X1>
```

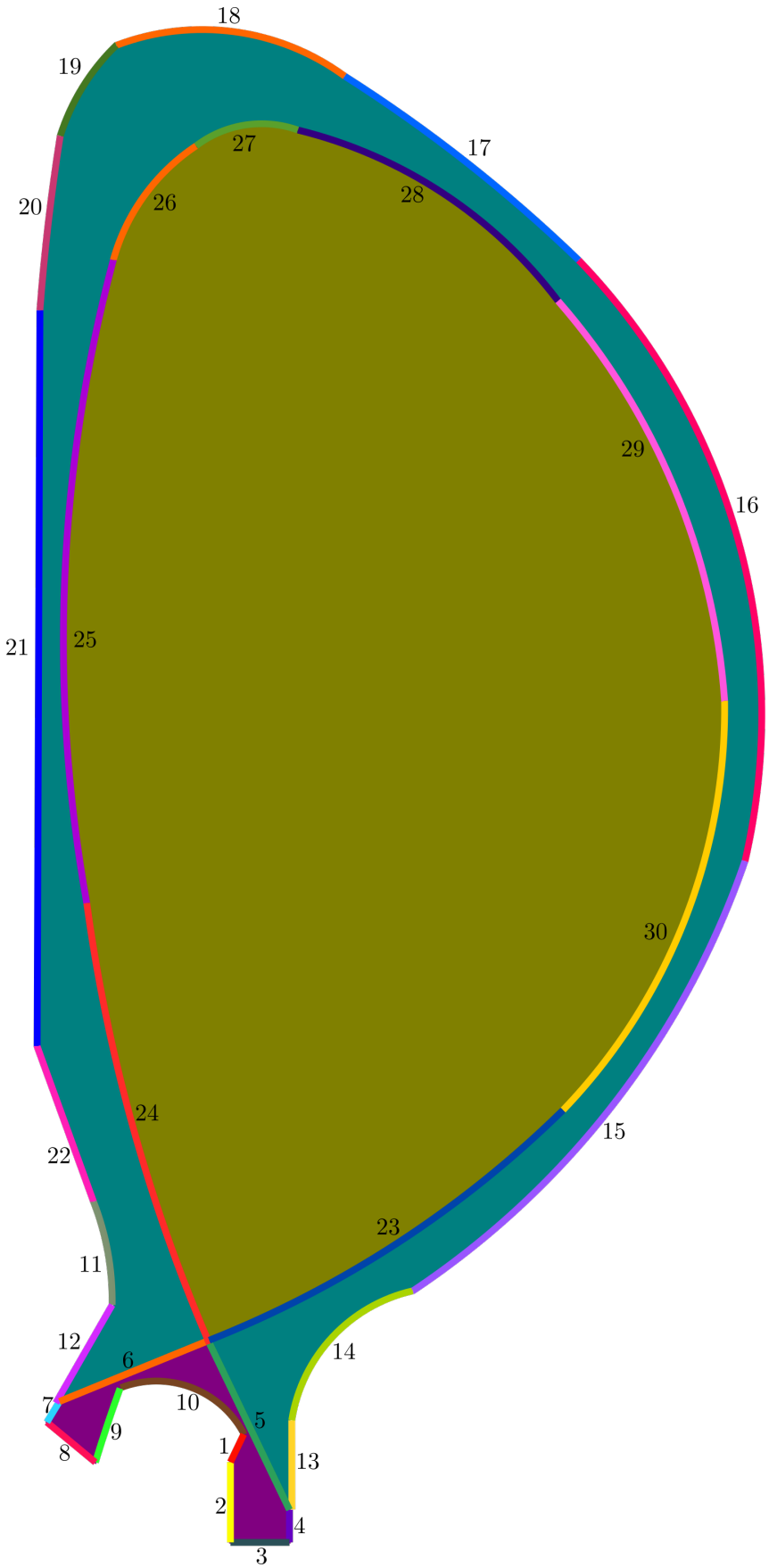


Figure 10: Illustration of the faces and edges of the tokamak cross section test case.

```
  <Y1> -0.062 </Y1>
  <Z1> 0.0 </Z1>
  <X2> -0.090 </X2>
  <Y2> -0.066 </Y2>
  <Z2> 0.0 </Z2>
  <R> 0.00020 </R>
  <D> 0.00020 </D>
</LINE>
<LINE>
  <X1> -0.0925 </X1>
  <Y1> -0.0645 </Y1>
  <Z1> 0.0 </Z1>
  <X2> -0.088 </X2>
  <Y2> -0.063 </Y2>
  <Z2> 0.0 </Z2>
  <R> 0.00020 </R>
  <D> 0.00020 </D>
</LINE>
```

For r-adaptation, only the internal curves are considered. In the first pass of the variational optimisation module, the scaling factor is set to 0.4, only targeting elements which are adjacent to the curves. In the second pass the scaling factor is slightly increased to 0.45 and a radius of influence of 1mm is set. Lastly, in the third and final pass no r-adaptation is employed and the mesh gets upgraded to 5th polynomial order. The full execution command is given below:

```
NekMesh -v
  -m varopti:linearelastic:radaptcurves=5,6,23-30:radaptscale=0.4:subiter=
↪  5:restol=1e-5:nq=2:numthreads=10
  -m varopti:linearelastic:radaptcurves=5,6,23-30:radaptscale=0.45:radaptr
↪  ad=0.0010:subiter=15:restol=1e-5:nq=2:numthreads=10
  -m varopti:hyperelastic:nq=6:numthreads=10
  cross_section.mcf cross_section.xml
```

The resulting mesh is shown in Figures 11–13. All the selected values for both the refinement curves and lines, as well as the r-adaptation, were manually tuned through a lot of trial and error and were the best combination we have tried.

6 Conclusions and future work

The goal of Task 1.1 is the generation of meshes capable of node clustering around and conforming to internal features, with the aim of generating a mesh suitable for the tokamak cross-section edge case. In this report, we have described in detail the additional features added to *NekMesh* to achieve said goal.

First, we extended and improved the user-defined refinement in *NekMesh* by introducing a feature that allows CAD curves to be used as sources for refinement and improving the octree subdivision to generate smoother meshes when user defined refinement is used. Next, r-adaptation, a method originally designed to cluster nodes and form anisotropic elements

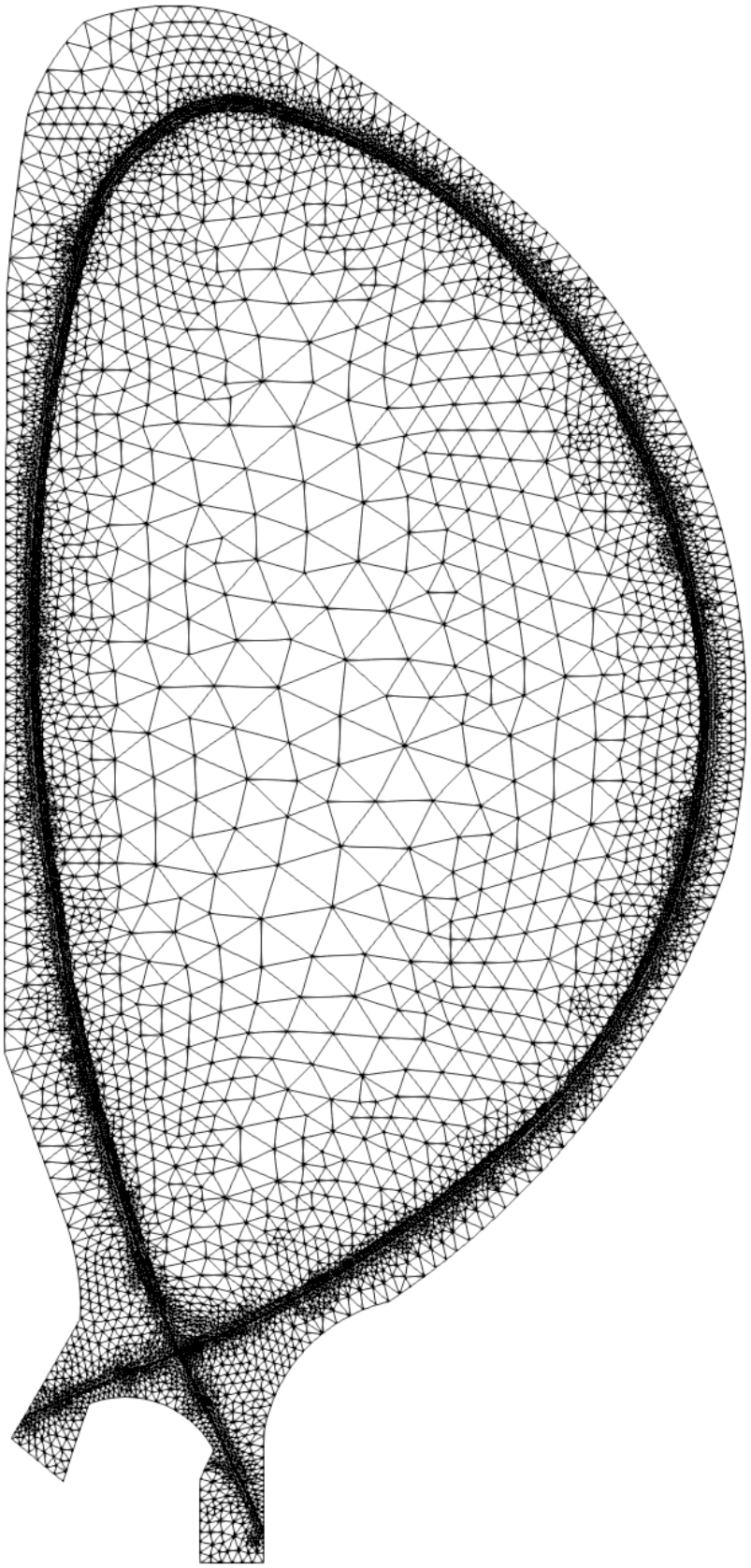
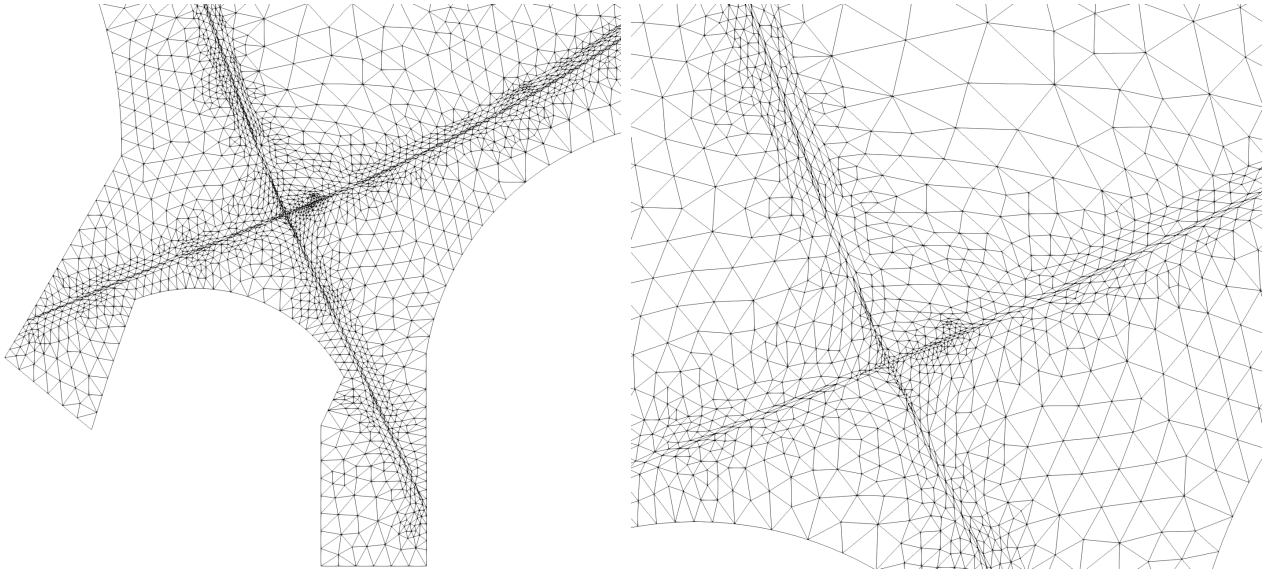


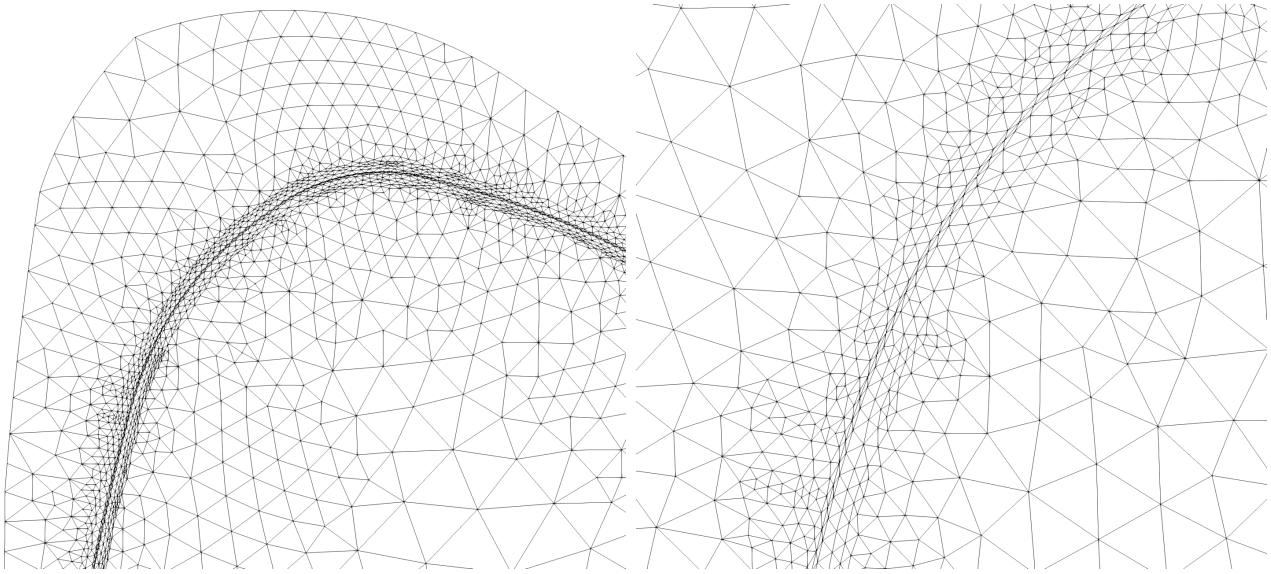
Figure 11: 5^{th} polynomial order mesh for the tokamak cross section test case with elements clustering around the plasma separatrix.

around fluid discontinuities in order to better resolve shock regions, was modified to work on user-defined CAD curves.



(a) (b)

Figure 12: 5^{th} polynomial order mesh for the tokamak cross section test case with elements clustering around the plasma separatrix. Zoomed in views of the lower region where the X-point and surrounding wall edges can be observed.



(a) (b)

Figure 13: 5^{th} polynomial order mesh for the tokamak cross section test case with elements clustering around the edge-case of the plasma separatrix. Zoomed in views of the top region.

User documentation for using the new features is then provided for each of the individual features as well the CAD preparation requirements. A pipeline process is then described from CAD preparation to mesh generation explaining each step in the process in detail through the use a simple test case of a circle embedded in a square. A brief discussion of the parameters and their choices is also presented, highlighting some of the limitations of the new features.

Finally, a simplified geometry mimicking a tokamak cross-section where the plasma separatrix is modelled using an internal curve is presented. The resulting mesh satisfies the goals set out for Task 1.1 and paves the way for Task 1.2. However, the manual nature of selecting refinement and scaling parameters by trial and error is highlighted as a significant drawback. Further understanding of the refinement and scaling parameters and their combination is clearly needed. This is left for future work with the knowledge of the inevitable changes following the completion of Task 1.2.

The next obvious next step is to mesh an accurate geometry of the tokamak cross section once it becomes available to us. Further, a decision of whether or not the flux surfaces are part of the requirements should be made to enable proper validation of the meshing capabilities. The variational optimisation code also requires a refactoring for better readability, maintainability and, crucially, performance. For instance, the current implementation uses *boost pthreads* for parallelisation, which for 2D cases is sufficient although could become a bottleneck when moving to 3D.

Additional work falls under Task 1.2 where the generation of thin boundary layer meshes will be explored as a means of refining curves with anisotropic quadrilateral elements surrounding the curves. This will require significant modifications to the current boundary layer generation which breaks down on meshes containing internal curves and sharp corners. Lastly, full quadrilateral mesh generation will be attempted to improve the overall mesh quality.

References

[1] Julian Marcon, Giacomo Castiglioni, David Moxey, Spencer J Sherwin, and Joaquim Peiró. rp-adaptation for compressible flows. *International Journal for Numerical Methods in Engineering*, 121(23):5405–5425, 2020.

[2] M. Turner, J. Peiró, and D. Moxey. Curvilinear mesh generation using a variational framework. *Computer-Aided Design*, 103:73–91, 2018.

[3] 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.

[4] M. Turner, D. Moxey, and J. Peiró. Automatic mesh sizing specification of complex three dimensional domains using an octree structure. In *24th International Meshing Roundtable*, 2015.

[5] 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.

[6] Open Cascade - software development company. https://www.opencascade.com/.

[7] Nektar++: Spectral/hp Element Framework – User Guide. https://doc.nektar.info/userguide/latest/.

[8] G Federici, C.H Skinner, J.N Brooks, J.P Coad, C Grisolia, A.A Haasz, A Hassanein, V Philipps, C.S Pitcher, J Roth, W.R Wampler, and D.G Whyte. Plasma-material interactions in current tokamaks and their implications for next step fusion reactors. *Nuclear Fusion*, 41(12):1967–2137, dec 2001.