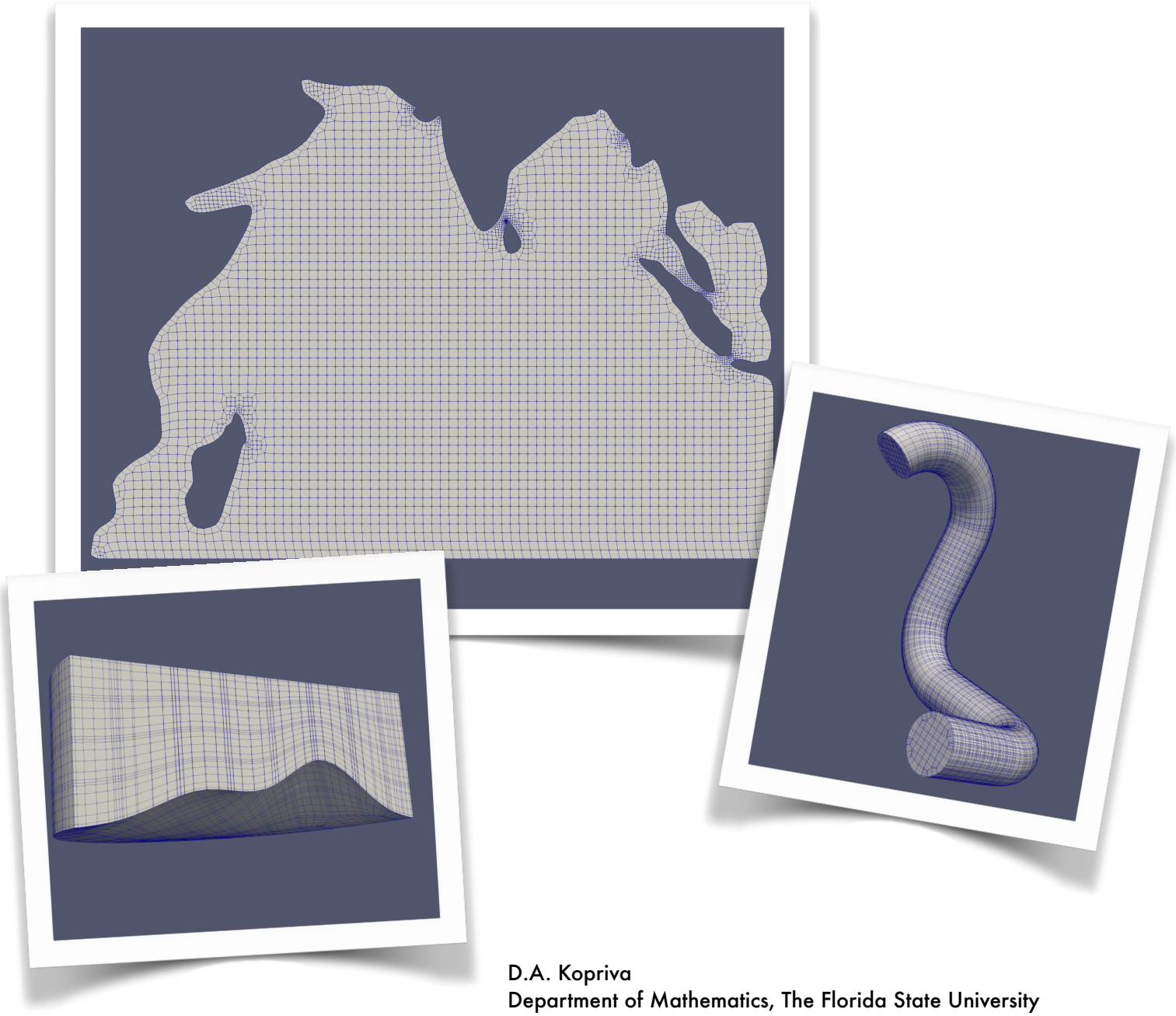
HOHQMesh
(High Order Hex-Quad Mesher)

User Manual



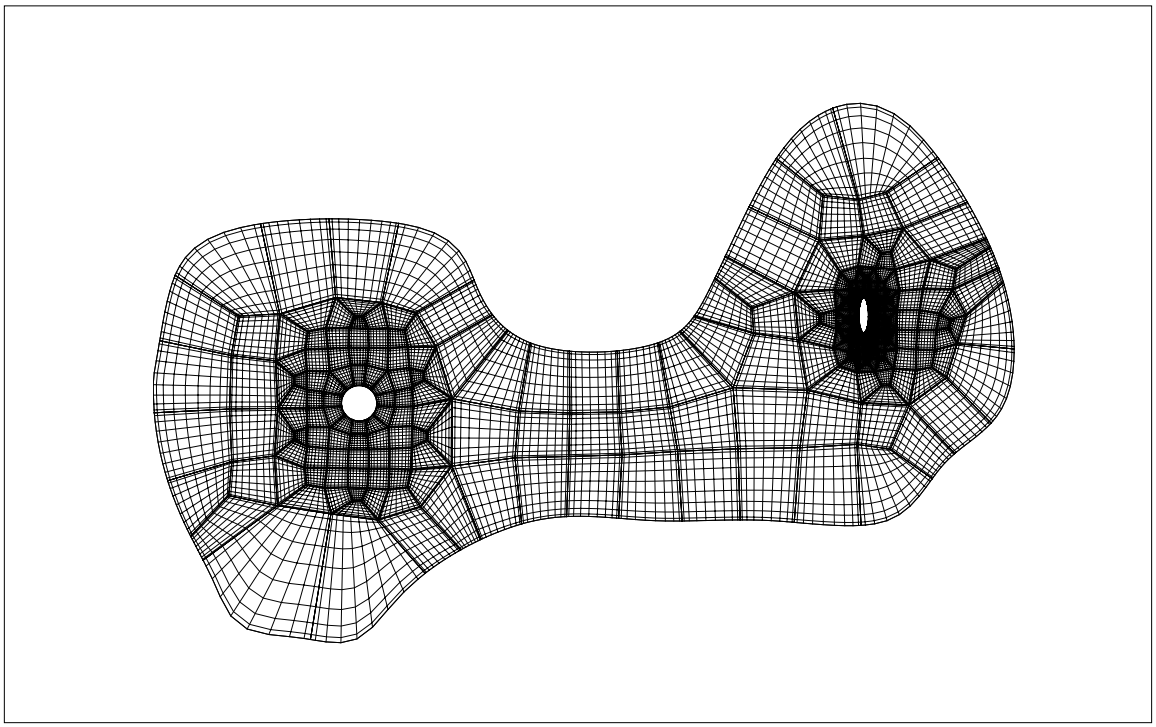
D.A. Kopriva
Department of Mathematics, The Florida State University
• May 4, 2021 Happy Star Wars Day!

Table of Contents

Chapter 1

HOHQMesh

OVERVIEW



A spectral element mesh with an outer boundary and two inner boundaries showing the internal degrees of freedom

Multidomain spectral methods, of which spectral element methods (SEMs) are a subclass, were introduced by Patera (for elliptic and parabolic equations) and by Kopriva (for hyperbolic systems) to increase the efficiency of spectral methods and to apply them to complex geometries. Although somewhat controversial at the time -- questions were raised whether it was wise to not use the highest order polynomial possible for a given number of degrees of freedom -- the methods have become so commonly used within the community that the updated book by Canuto et al. is subtitled "Fundamentals in Single Domains."

The features of spectral element methods are now well-established. Like low order finite element methods, they can be applied to general geometries, but have exponential convergence in the polynomial order. Discontinuous Galerkin (DGSEM) versions applied to hyperbolic problems have exponentially low dissipation and dispersion errors, making them well suited for wave propagation problems. They are also especially suitable when material discontinuities are present. Approximations exist for high order quad/hex and tri/tet elements. Numerous examples of the flexibility and the power of spectral element methods can be found in Canuto et al.'s third volume subtitled "Evolution to Complex Geometries". Textbooks on the subject now exist, such as those by Deville, Fischer and Mund, Sherwin and Karniadakis , Hesthaven and Warburton, and Kopriva.

What some are now calling "classical" spectral element methods use tensor product bases on quadrilateral or hexahedral meshes. These bases lead to very efficient implementations and have high order quadratures that can be used to approximate the integrals found in weak forms of the equations. The methods are being used in a wide variety of fields including fluid dynamics, electromagnetics, geophysics, and fluid-structure interaction problems, just to name a few.

Unfortunately, meshes for quad/hex elements are considered to be difficult to generate even for low order finite element approximations. This has lead to the development of triangular/tetrahedral spectral element bases. These methods can adapt the meshes generated by virtually all mesh generation packages today in two and three space dimensions. What one gives up in trade is the efficiency of the derivative evaluations, the Gauss quadratures, and meshes well-suited for boundary layer computations.

Spectral Element Grid Generation

The advantages not withstanding, a major frustration in - and impediment to - the application of spectral element methods has been the lack of appropriate general purpose mesh generation software. A survey of the literature, practitioners, and user manuals for available spectral element software packages such as SemTex, SEM2DPack, or Nekton, highlights these difficulties. Blackburn's SemTex page http://users.monash.edu.au/~bburn/semtex.html notes that "Mesh generation can be a significant hurdle to new users" and includes "a number of example meshes ... (most of which were generated by hand)." SEM2DPack's manual says it "can only generate a structured mesh for a single quadrilateral domain, possibly with curved sub-horizontal boundaries and curved sub-horizontal layer interfaces." Sherwin and Peiro's comment: "The ability to

construct suitable computational meshes is currently a significant limiting factor in the development of compact high-order algorithms in very complex geometries" still holds today. Canuto et al. do not even broach the subject.

Simply put, and avoiding the common colloquialism, the state of the art in spectral element grid generation has been dismal.

One finds that spectral element meshes are either generated "by hand", by special purpose mesh generators, or by low order finite element packages. Examples of hand generated meshes can be found in the textbooks listed above, for instance. SEM2DPack interfaces with the low order finite element mesh generator EMC2. The Nekton and SemTex packages interface with the finite element package GMSH. But the situation is particularly difficult for ``classical'' quad and hex element codes since even low order finite element mesh generators for these elements are hard to find. A consequence is that one even finds meshes in the literature that are simple quad/hex decompositions of low order triangular/tetrahedral meshes.

The meshes that practitioners generate differ greatly from those generated by finite element mesh generators. The reason is not just a matter of the tedium associated with the process. Spectral element approximations encourage the use of larger elements with curved boundaries approximated at high order. Meshes generated by hand or with simple templates tend to have fewer and larger elements.

Meshes generated by finite element packages designed for low order elements generate huge numbers of small elements and do not exploit the efficiency of high order spectral element approximations. The use of standard generators can lead one to use a high order method, yet approximate curved boundaries as segments of straight lines. Commercial mesh generators that generate "higher order elements", e.g. PATRAN, GMSH or Gambit, do exist, but high order usually means third order, tops. ICEM-HEXA will guarantee quad/hex spectral element type meshes only for block structured meshes. The costs of commercial packages, however, are so far above the budgets provided by the typical NSF grant or mathematics department and so aren't an option even if they could generate spectral element meshes.

HOHQMesh

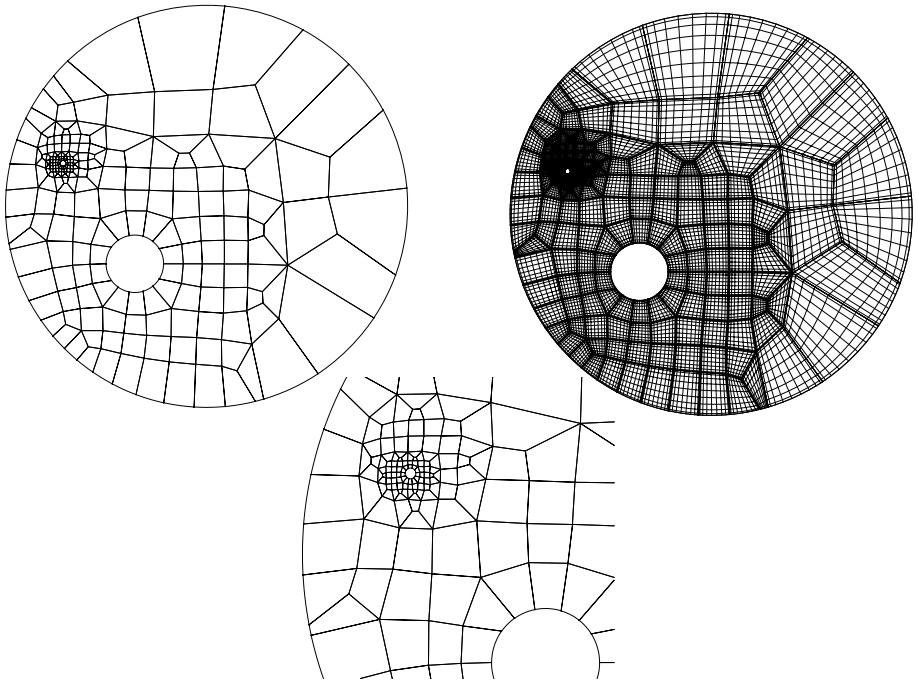
For these reasons we have developed the *High Order Hex-Quad Mesh* (HOHQMesh) package to automatically generate all-quadrilateral meshes with high order boundary information to be used

in spectral element computations. It also can take such two dimensional meshes and extrude them in the normal direction to general all hex meshes for simple extrusion type geometries. Sinc

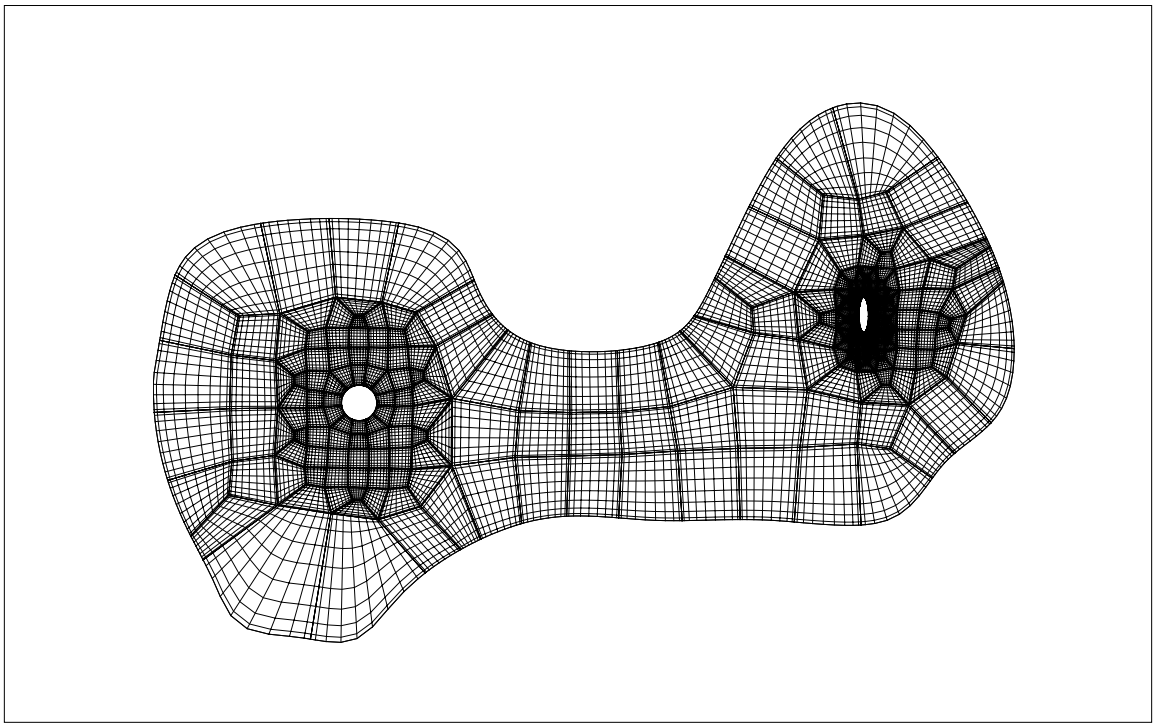
Example Meshes

Here we show meshes that have been generated by HOHQMesh. Control files for generating these meshes can be found in the ControlFiles directory. Some of the meshes show internal spectral element degrees of freedom and the fully accurate boundary representations. Others show only the quad or hex shape of the elements as given in the plot file generated by the program.

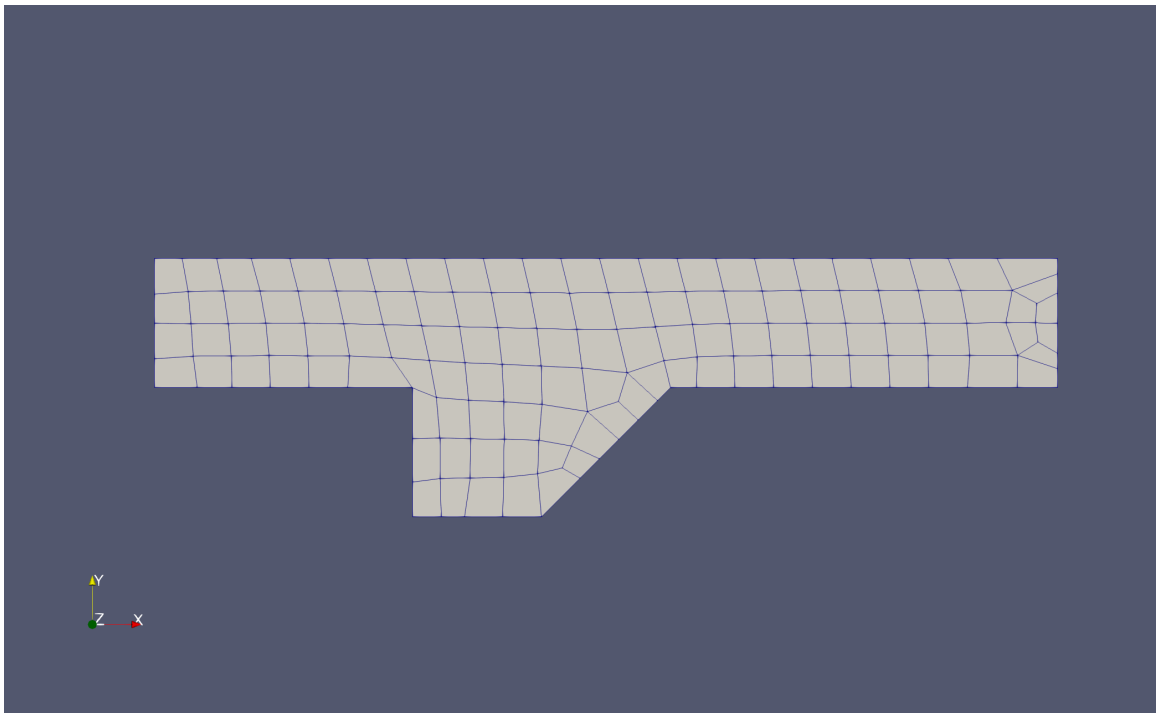
The first example is a full spectral element mesh for two circles within an outer circle. HOHQMesh is fully automatic and sizes the elements according to the geometry.



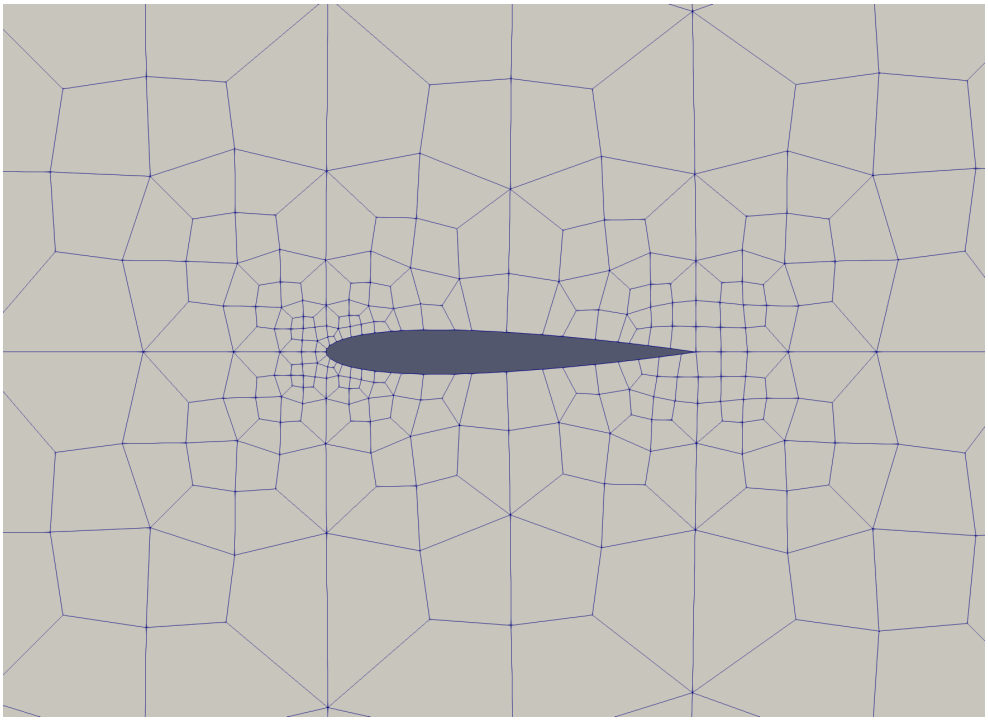
The second example example shows that general curves can be used to define the boundaries. This time, a set of points and a spline are use to define the outer boundary.

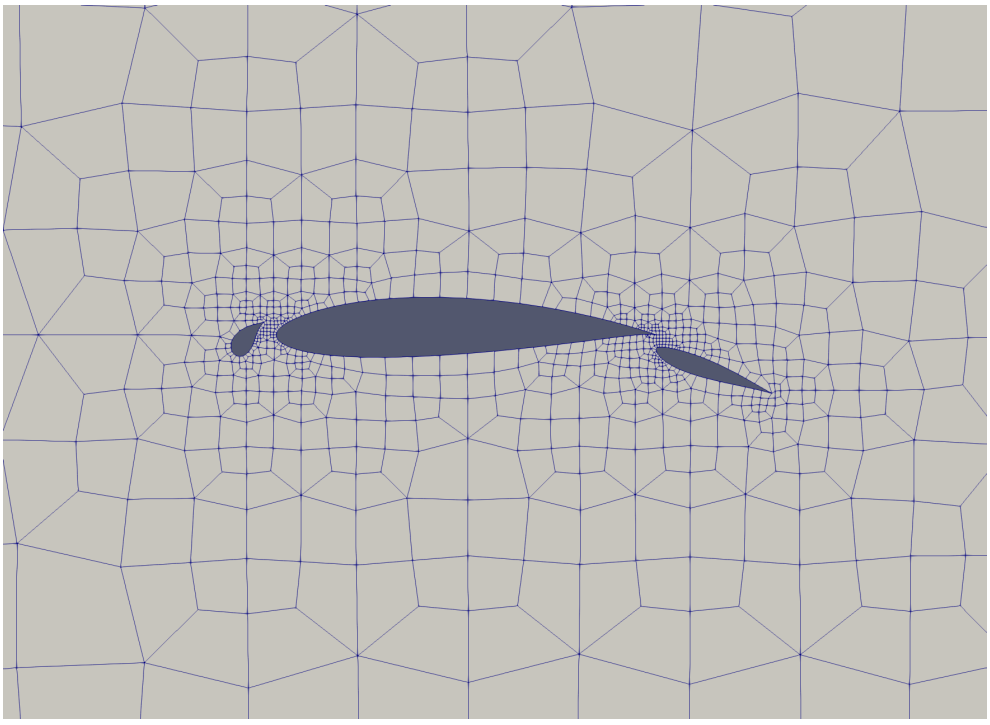


HOHQMesh has templates to automatically mesh around sharp corners.

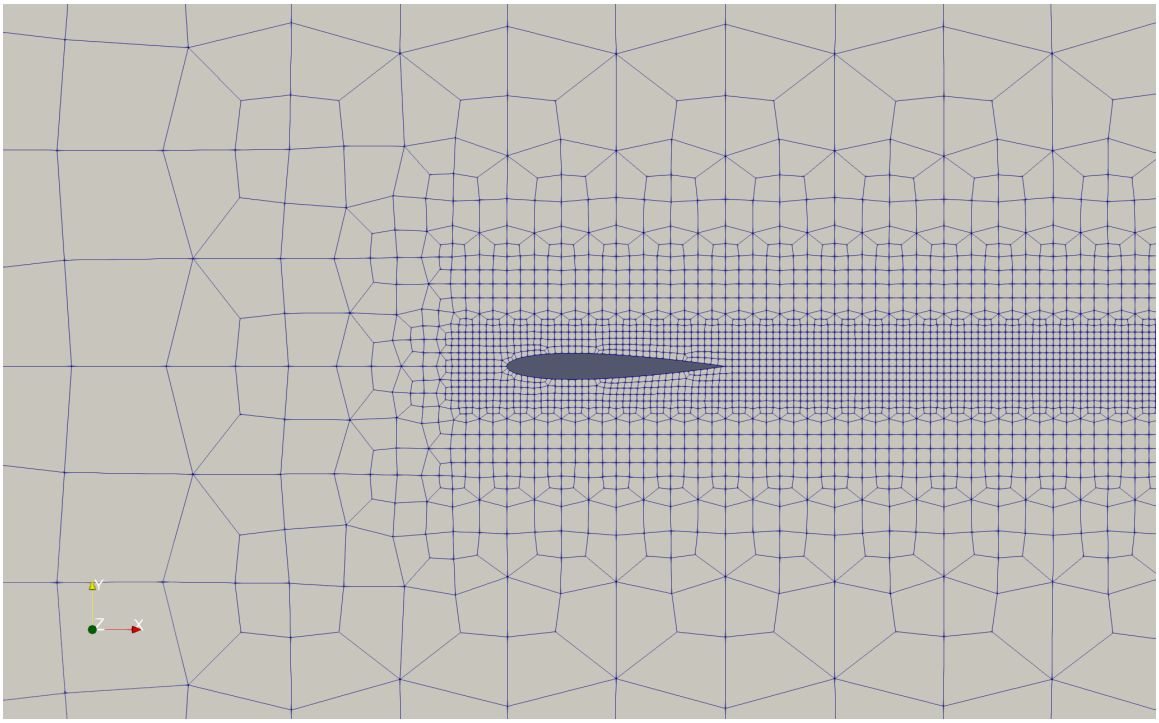


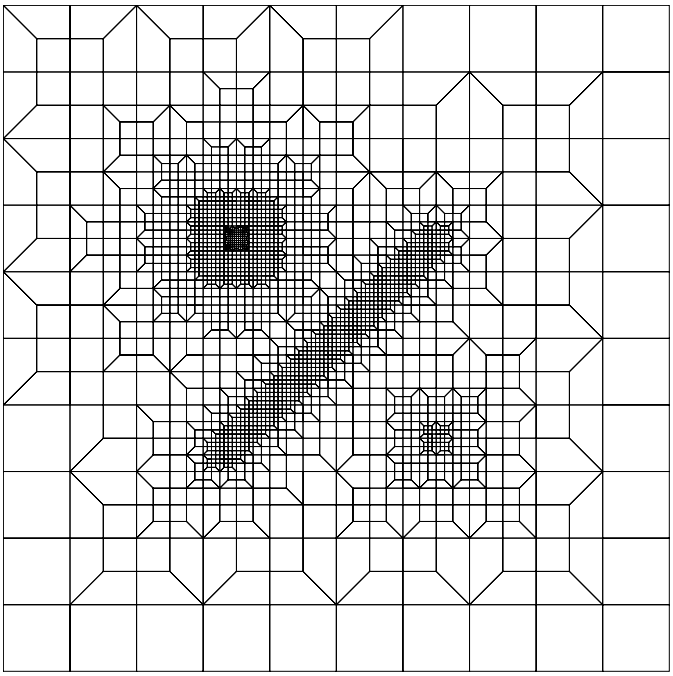
This makes it possible to mesh airfoil type geometries



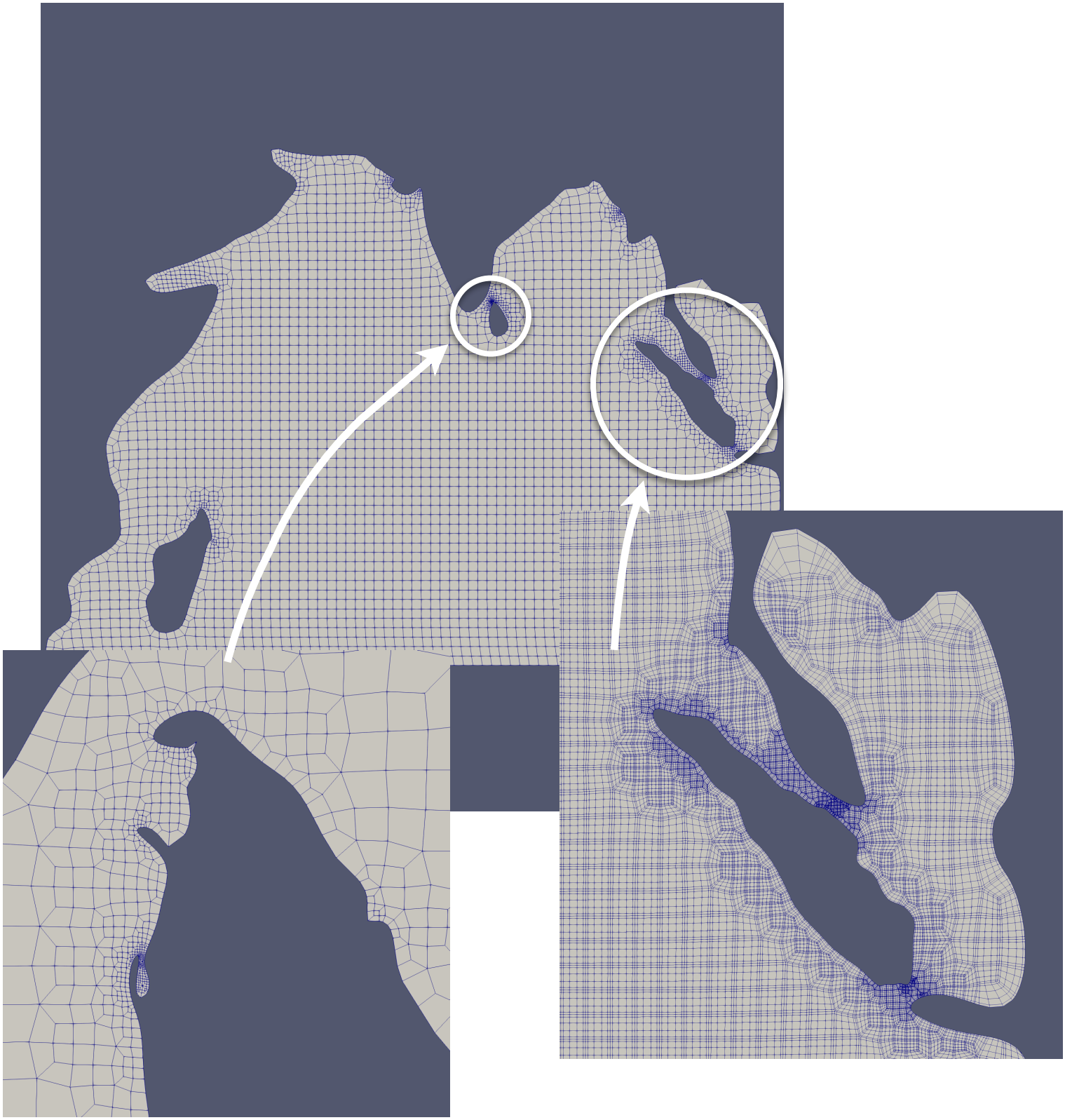


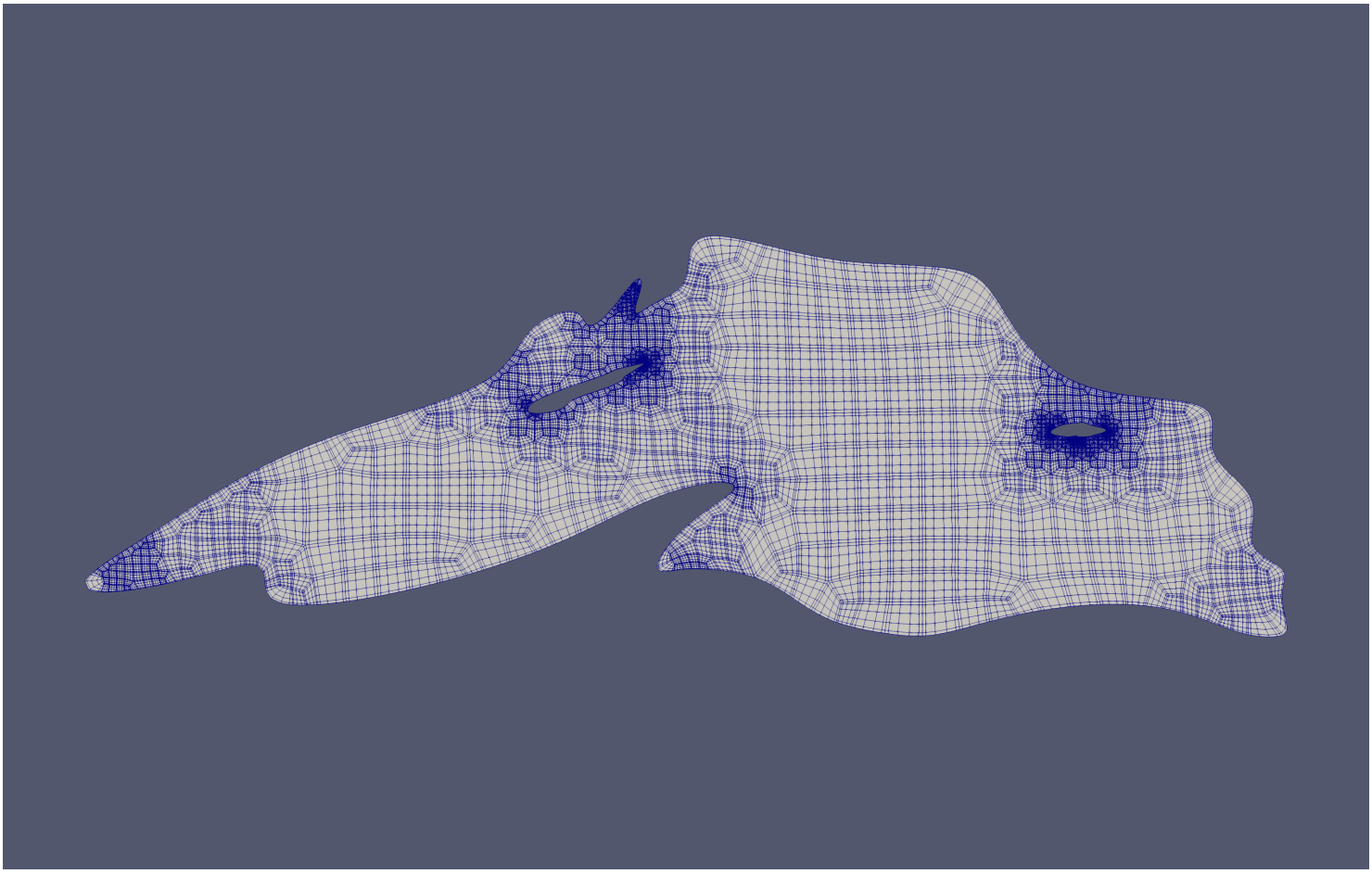
Local refinement can be added, for example to follow a wake.





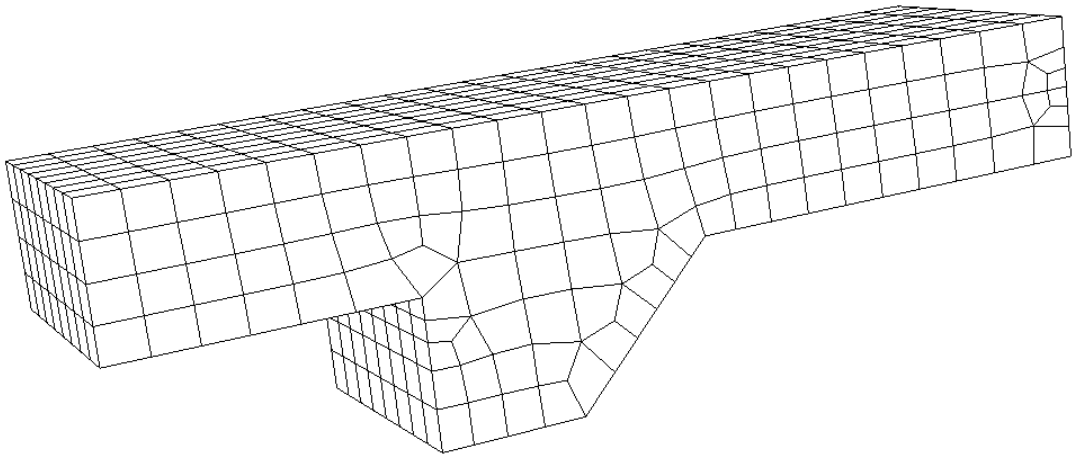
Truly complex geometries can be meshed, as shown in the following coastline models. Refinement around features is automatic.

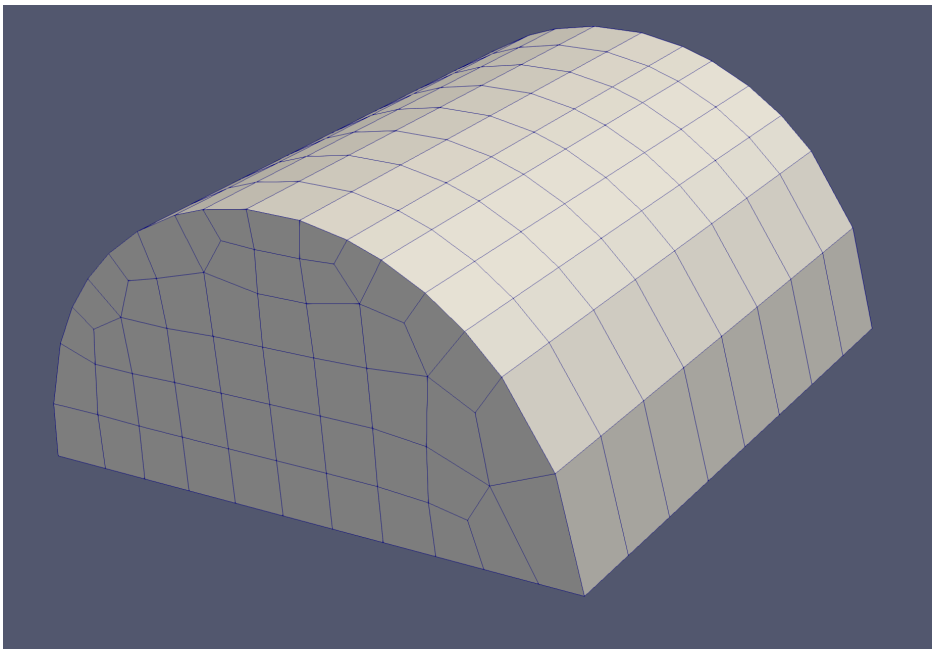




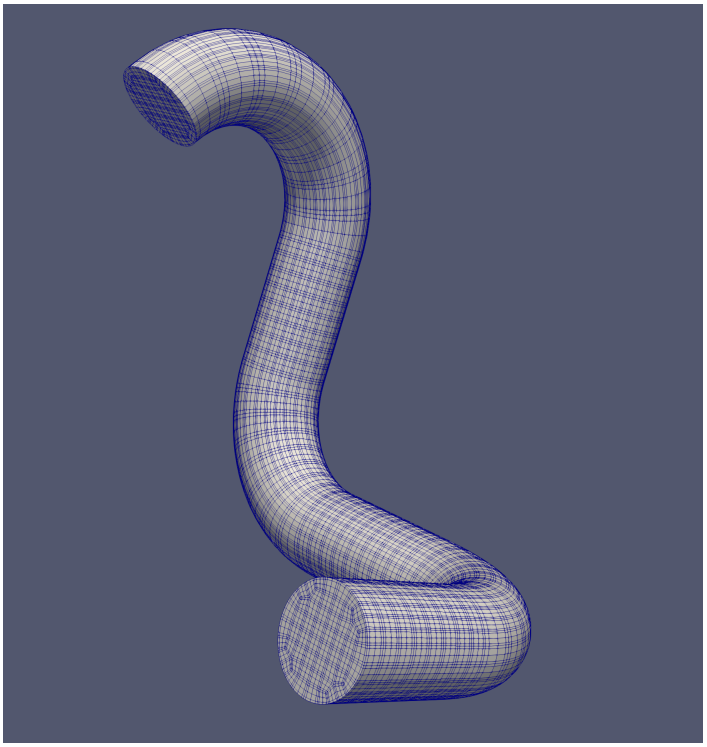
Lake Superior with spectral element nodes.

3D hex meshes can be generated by a simple extrusion of a 2D mesh

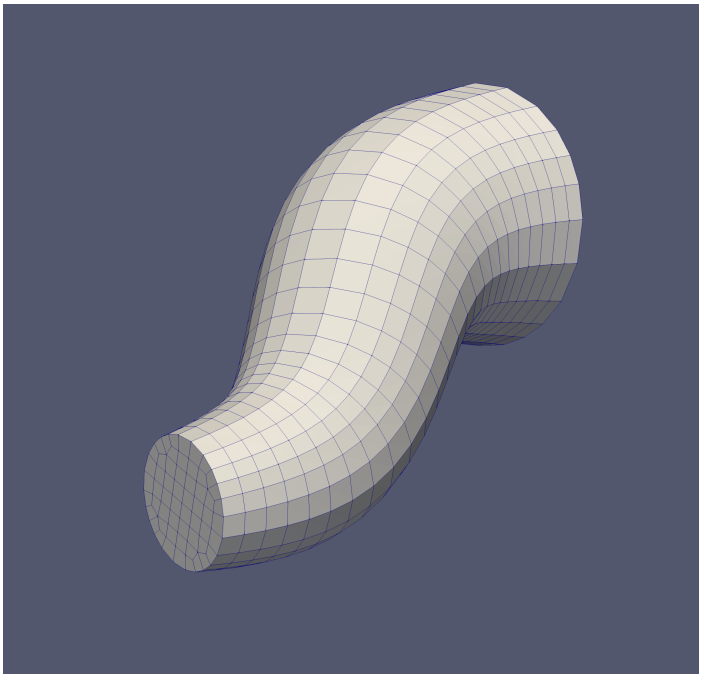




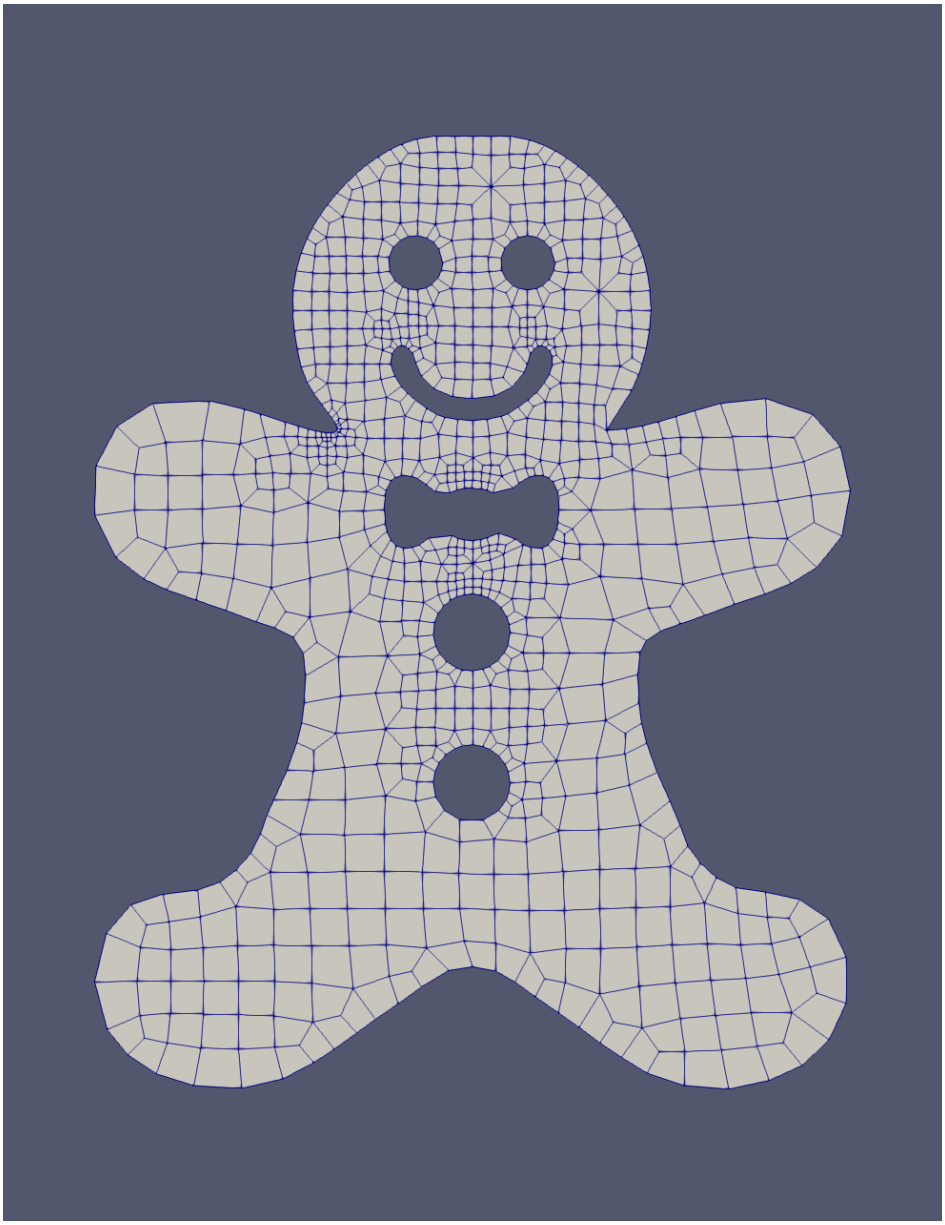
Hex meshes can also be created by sweeping a quad mesh along a designated curve.



Swept meshes can also be scaled along the curve.



Geometries can be quite general!



Chapter 2

HOHQMesh

THE MODEL

At the present time, HOHQMesh is designed to generate quadrilateral meshes in general two dimensional geometries like those shown below, and extrusions thereof to get three dimensional hex meshes.

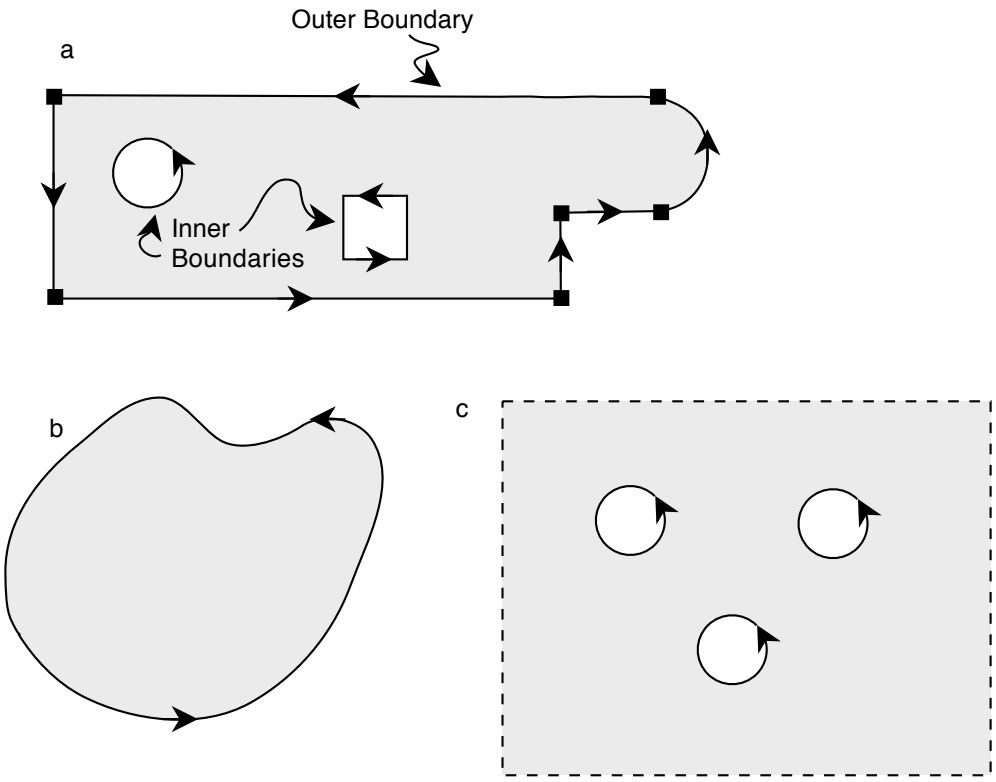


Fig. 2. Two dimensional geometries meshable by HOHQMesh

The two dimensional domain to be meshed can be bounded by at most one exterior boundary curve, as in (a) and (b), above, and any number of interior boundary curves that create holes. For purely external problems, a rectangular outer boundary can be automatically generated, as shown in (c).

Boundary Curves

Boundaries are constructed as closed chains of parametrized curves, with the parameter in the interval [0,1], oriented counter-clockwise. The chains can have one or more segments as seen in Fig. 2. In Fig. 2a the outer boundary is constructed from six curves, whereas in Fig. 2b it is bounded by a single one. The inner boundaries in Fig. 2a are a single circle and a square constructed by a chain of four lines.

A curve is defined by a block

`\begin{`*`CURVE_TYPE`*`}`

`\end{`*`CURVE_TYPE`*`}`

It is given a name so that boundary conditions can be applied segment-by-segment to a chain. Currently there are four types of curves that can be defined. The first is given by equation components. The second by a cubic spline interpolant of a set of nodal points. The third type is defined as a straight line between two points. Finally, a circular arc can define a curve. The architecture is designed to easily add curve definitions in the future by creating subclasses of the SMCurveClass.

The Parametric Equation Curve Definition.

Curves can be defined by strings that define the equations for the x- and y- components of the curve using the `ParametricEquationCurve` type. An example block for this kind of curve is

```
\begin{PARAMETRIC_EQUATION_CURVE}
     name = circle
     xEqn = x(t) = 14.0*cos(2*pi*t)
     yEqn = y(t) = 14.0*sin(2*pi*t)
     zEqn = z(t) = 0.0
\end{PARAMETRIC_EQUATION_CURVE}
```

The first line defines the name, followed by the x- , y- and z- equation definitions. right now, only meshes in the x-y plane can be generated, so the `z=0` equation must be set this way. This block defines a closed circular curve of radius 14 named “circle”. The indenting is optional, as is the ordering within the block. The keywords are “name”, “xEqn”, etc. and must be spelled correctly

or an error will be posted when the model is read in. The zEqn keyword is optional and can be left out.

The equations can be any legal representations of an equation as is standard in most computer languages. The first part, before the equals sign defines the parameter variable, in this case, `t`. On the right hand side is the formula that defines the curve. Exponentiation is defined as in BASIC, like `t^2`. For convenience, the constant `pi` is defined. Like BASIC, literals are defined as double precision values. There are no integer quantities. Standard functions like `sin, cos, tan, atan, log, log10, exp`, etc. are also available for use.

The Spline Curve Definition

The second type of curve is the `SplineCurve` type, which fits a cubic spline to a set of knots at given parameter values. The parameterization does not have to be uniform. An example of a spline-defined curve is

```
\begin{SPLINE_CURVE}
   name = SplineBoundaryCurve
   nKnots = 9
   \begin{SPLINE_DATA}
      0.000000000000000 -3.50000000000000  3.50000000000000 0.0
      3.846153846153846E-002 -3.20000000000000  5.00000000000 0.0
      7.692307692307693E-002 -2.00000000000000  6.00000000000 0.0
      0.769230769230769  0.000000000000000 -1.00000000000000 0.0
      0.807692307692308 -1.00000000000000 -1.00000000000000 0.0
      0.846153846153846 -2.00000000000000 -0.800000000000000 0.0
      0.884615384615385 -2.50000000000000  0.000000000000000 0.0
      0.923076923076923 -3.00000000000000  1.00000000000000 0.0
      1.00000000000000 -3.50000000000000  3.50000000000000 0.0
   \end{SPLINE-DATA}
 \end{SPLINE_CURVE}
```

As before, the first line after the `\begin` is the name of the curve. It is followed by the number of nodes in the spline. The data columns that follow are the nodes given by `tj,xj,yj,zj`. This particular spline is closed, so the location of the last node is the same as the first. Again, the `zj` values must currently be zero to ensure curves in the x-y plane.

End Points Line Definition

The next type of curve is the END_POINTS_LINE type that takes two end points and puts a straight line between them. An example is

```
\begin{END_POINTS_LINE}
      name   = B1
      xStart = [0.0,1.0,0.0]
      xEnd   = [2.0,1.0,0.0]
\end{END_POINTS_LINE}
```

Circular Arc Curve

The final type of curve defines a circular arc. The angles can be defined either in terms of degrees or radians. If the (optional) units keyword is not included, the default is radians.

```
\begin{CIRCULAR_ARC}
   name        = circle
   units       = degrees
   center      = [0.0, 0,0,0.0]
   radius      = 4.0
   start angle = 0.0
   end angle   = 180.0
\end{CIRCULAR_ARC}
```

Boundary Chains

To allow complex boundary curves and to allow different portions of a boundary to have different boundary conditions applied, curves can be chained together into a closed curve. A chain is defined by curves specified (in order) within a

```
\begin{CHAIN}

\end{CHAIN}
```

block. Any number of curves can be chained together. The chain itself is also given a name. An example of a chain that defines the boundary of a unit square is

```
\begin{CHAIN}
     name = UnitSquare
     \begin{PARAMETRIC_EQUATION_CURVE}
          name = bottom
          xEqn = f(t) = t
          yEqn = f(t) = 0
          zEqn = f(t) = 0
```

```
    \end{PARAMETRIC_EQUATION_CURVE}

    \begin{PARAMETRIC_EQUATION_CURVE}
        name = right
        xEqn = f(t) = 1
        yEqn = f(t) = t
        zEqn = f(t) = 0
    \end{PARAMETRIC_EQUATION_CURVE}

    \begin{PARAMETRIC_EQUATION_CURVE}
        name = top
        xEqn = f(t) = 1-t
        yEqn = f(t) = 1
        zEqn = f(t) = 0
    \end{PARAMETRIC_EQUATION_CURVE}

    \begin{PARAMETRIC_EQUATION_CURVE}
        name = bottom
        xEqn = f(t) = 0
        yEqn = f(t) = 1-t
        zEqn = f(t) = 0
    \end{PARAMETRIC_EQUATION_CURVE}
\end{CHAIN}
```

Again, the indentation is for readability only, as is the line spacing between the blocks. (Blank lines and lines starting with "%" are ignored.) Also remember that the chain is defined counter-clockwise, and the curves within the chain must be ordered and oriented properly. Chains cannot be chained together.

The Model

The model (there is only one) defines the region that is to be meshed. It is marked by `\begin{MODEL}`

`\end{MODEL}`

The model contains at most one outer boundary chain and any number of inner boundary chains. The outer boundary chain (if there is one) is defined by

`\begin{OUTER_BOUNDARY}`

`\end{OUTER_BOUNDARY}`

Within the OUTER_BOUNDARY block is a list of boundary curves that form a chain. There is no need to explicitly chain (by way of \begin{CHAIN}...\end{CHAIN}) the curves for the outer boundary, as that is implied.

Inner boundaries (if any) are defined within

\begin{INNER_BOUNDARIES}

\end{INNER_BOUNDARIES}

Within this block one defines as many CHAINs as there are inner boundaries. Each inner boundary must be defined within a chain. (*Even if there is only one curve in the chain.*) The order of the CHAINs is not important.

As an example, the following defines a model that has a single circular outer boundary and three inner circular boundaries. As usual, indentation is for the reader's eyes only. Note that between the blocks comments can be inserted starting with "%".

```
\begin{MODEL}
       \begin{OUTER_BOUNDARY}
          \begin{PARAMETRIC_EQUATION_CURVE}
               name = outer
               xEqn = x(t) = 14.0*cos(2*pi*t)
               yEqn = y(t) = 14.0*sin(2*pi*t)
               zEqn = z(t) = 0.0
          \end{PARAMETRIC_EQUATION_CURVE}
       \end{OUTER_BOUNDARY}
%
%      Inner boundaries, if any, are any number of chains
%      of curves.
%
       \begin{INNER_BOUNDARIES}
          \begin{CHAIN}
            name = Boundary 1
             \begin{PARAMETRIC_EQUATION_CURVE}
                     name = Circle1
                     xEqn = f(t) = -10.25 + 0.2*cos(2*pi*t)
                     yEqn = f(t) = 3.0 + 0.2*sin(2*pi*t)
                     zEqn = z(t) = 0.0
               \end{PARAMETRIC_EQUATION_CURVE}
          \end{CHAIN}

          \begin{CHAIN}
```

```
            name = Boundary 2
            \begin{PARAMETRIC_EQUATION_CURVE}
                    name = Circle2
                    xEqn = f(t) = -5.1 + 1.0*cos(2*pi*t)
                    yEqn = f(t) = 1.0*sin(2*pi*t) - 4.1
                    zEqn = z(t) = 0.0
             \end{PARAMETRIC_EQUATION_CURVE}
          \end{CHAIN}

          \begin{CHAIN}
            name = Boundary 3
            \begin{PARAMETRIC_EQUATION_CURVE}
                    name = Circle3
                    xEqn = f(t) = -12.0 + 0.5*cos(2*pi*t)
                    yEqn = f(t) = 0.5*sin(2*pi*t) - 0.5
                    zEqn = z(t) = 0.0
             \end{PARAMETRIC_EQUATION_CURVE}
          \end{CHAIN}
       \end{INNER_BOUNDARIES}
\end{MODEL}
```

Chapter 3

HOHQMesh

THE CONTROL INPUT

The meshing of the model is controlled by the `CONTROL_INPUT` block of the input control file, which gives all of the commands needed to mesh the model. Actually, there are not a lot of commands at the moment, so that's not too bad. The control block is

```
\begin{CONTROL_INPUT}

\end{CONTROL_INPUT}
```

Inside the control input block the `RUN_PARAMETERS`, `BACKGROUND_GRID`, `SMOOTHER`, and any number of `REFINEMENT_CENTER`s and `REFINEMENT_LINE`s are defined.

The Run Parameters

The `RUN_PARAMETERS` block defines the file information and the polynomial order at which the boundary curves will be defined in the mesh file. Three files are output by the mesher. The first is the actual mesh file. The second is a tecplot format file that can be used to visualize the mesh. The free programs VisIt or Paraview can be used to plot tecplot files. The final file (optional) is to report mesh statistics like the distribution of largest angle, Jacobian, etc. See the "Verdict Library Reference Manual" by Stimpson et al. if you are interested to learn about the different shape quality measures. Include a unix style path to choose the directory for the results.

The `RUN_PARAMETERS` block is:

```
\begin{RUN_PARAMETERS}
     mesh file name   = MeshFileName.mesh
     plot file name   = PlotName.tec
     stats file name  = StatsName.txt
     mesh file format = ISM
     polynomial order = 6
     plot file format = skeleton OR sem
 \end{RUN_PARAMETERS}
```

The names can be anything, since they are simply text files. However the ".tec" extension on the plot file will help VisIt/Paraview know how to read it. If you don't want a file created, simply choose the name to be `none`.

In the current version of HOHQMesh, there are two mesh file formats, `"ISM"` which stands for "Implementing Spectral Methods" . This is the file format described in the book. The other is `"ISM-v2"`, which provides the edge information needed by the approximations so that the edge generation algorithms in the appendix of the book are not needed. See the section in this manual on ISM-v2 for a description of the additional information provided. In the future, other file formats may be implemented, too. Finally, high order boundary information is conveyed by outputting an interpolant of the specified order. That information can be viewed using the "sem" plot file format.

The Background Grid

The meshing algorithm starts with a uniform background grid. If an outer boundary is specified in the model, HOHQMesh will create this background grid using the extents of the outer boundary and the background grid size specified in the `BACKGROUND_GRID` block. If there is no outer boundary, then the background grid must be specified in the control input. The `BACKGROUND_GRID` block specifies the coordinates of the lower left corner of the grid, the grid size in each coordinate direction, and the number of grid cells in each direction:

```
\begin{BACKGROUND_GRID}
       x0 = [-10.0, -10.0, 0.0]
       dx = [2.0, 2.0, 0.0]
       N  = [10,10,0]
\end{BACKGROUND_GRID}
```

The example above creates a uniform mesh with lower left corner at `(-10,10)` and upper right corner at `(10,10)`.

Alternatively, if there is an outer boundary curve, you want to specify the background grid size and let HOHQMesh compute the rest of the parameters:

```
\begin{BACKGROUND_GRID}
     background grid size = [2.0,2.0,0.0]
\end{BACKGROUND_GRID}
```

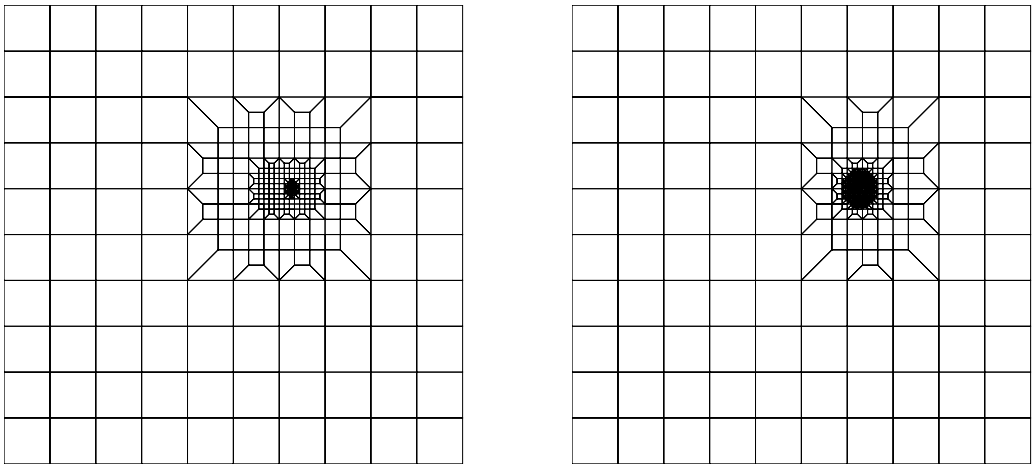


Fig. 3. Smooth (left) and sharp (right) refinement around a point (no smoothing).

The Smoother

It is generally necessary to smooth the mesh after it is generated. This is done by the `Smoother`.

The `SPRING_SMOOTHER` uses a spring-dashpot model and time relaxation to smooth the mesh. There are two spring topologies "LinearSpring" and "LinearAndCrossbarSpring". The first only has springs between the nodes along the edges. The latter also puts springs along the diagonals of an element. The latter is preferred. The springs have a spring constant associated with them and a dashpot with a damping coefficient. The nodes have mass. The linear ODE system that describes the motion of the nodes is integrated with a forward Euler (Explicit!) approximation for which a time step and number of time steps are given. The `SPRING_SMOOTHER` block, if one is used (Recommended!) is

```
\begin{SPRING_SMOOTHER}
    smoothing             = ON (Optional)
    smoothing type        = LinearAndCrossbarSpring
    spring constant       = 1.0 (Optional)
    mass                  = 1.0 (Optional)
    rest length           = 0.0 (Optional)
    damping coefficient   = 5.0 (Optional)
    number of iterations = 20
    time step             = 0.1 (Optional)
 \end{SPRING_SMOOTHER}
```

Just leave out any of the optional parameters if you want the default values to be used.

Refinement Centers

It is possible to ask HOHQMesh to locally refine the mesh at particular locations. This is done with a `REFINEMENT_CENTER` placed as desired. Two types of centers are available. One is "`smooth`", which refines near a specified point and gradually de-refines towards the neighboring mesh size. The other is "`sharp`", which keeps the refined size in the neighborhood of the center. (See Fig. 3.) The desired mesh size and the size of the center are also parameters. An example of a refinement center is

```
\begin{REFINEMENT_CENTER}
   type = smooth
   x0   = [1.0,1.0,0.0]
   h    = 0.20
   w    = 0.5
\end{REFINEMENT_CENTER}
```

This will place a center at (1,1,0) with mesh size of 0.2 over a circular region of radius 0.5 . Any number of `RefinementCenters` can be included in the `ControlInput` block. The order is not important.

Refinement Lines

The mesh can also be refined along a line using a `REFINEMENT_LINE`. Like the centers, there are two types, "smooth" and "sharp". To refine along a line, include a block of the form

```
\begin{REFINEMENT_LINE}
   type = smooth
   x0   = [-3.5,-3.5,0.0]
   x1   = [3.0,3.0,0.0]
   h    = 0.20
   w    = 0.5
\end{REFINEMENT_LINE}
```

Here, `x0` and `x1` are the starting and ending points of the line, `h` is the desired mesh size and `w` tells how far out from the line the refinement extends. An example of center and line refinements can be seen in Fig. 4.

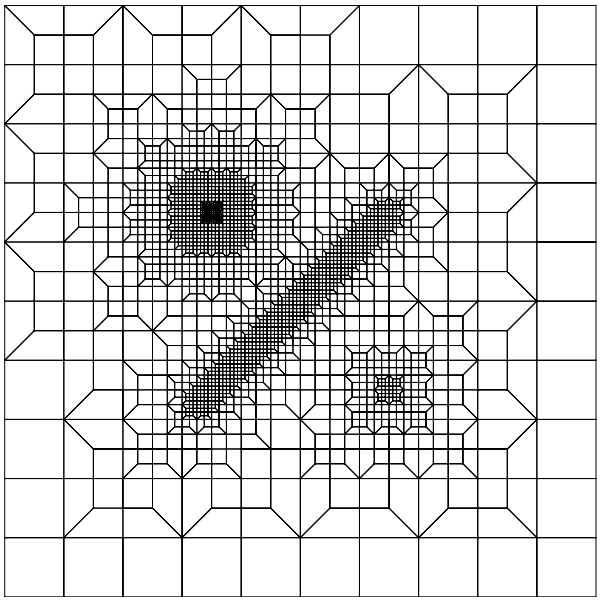


Fig. 4. A mesh with both center and line refinement.

Refinement regions are defined within a `REFINEMENT_REGIONS` block, e.g.

```
\begin{REFINEMENT_REGIONS}

     \begin{REFINEMENT_LINE}
        type = nonsmooth
        x0   = [-3.0,-3.0,0.0]
        x1   = [3.0,3.0,0.0]
        h    = 0.3
        w    = 0.3
      \end{REFINEMENT_LINE}

      \begin{REFINEMENT_CENTER}
        type = smooth
        x0   = [3.0,-3.0,0.0]
        h    = 0.1
        w    = 0.3
      \end{REFINEMENT_CENTER}

 \end{REFINEMENT_REGIONS}
```

Chapter 4

HOHQMesh

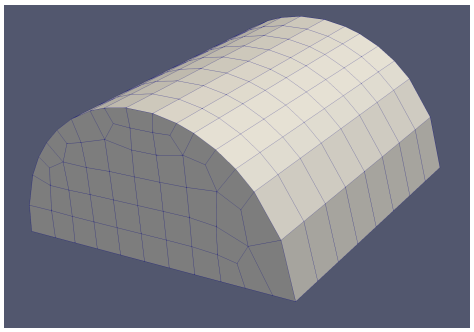
THREE DIMENSIONAL MESHES

HOHQMesh can also generate 3D hexahedral meshes by extruding or sweeping a two dimensional mesh. To tell the mesher that you want a hex mesh, you add an algorithm block to the `CONTROL_INPUT` block for how the 3D extrusion will be done. Currently there are three: Simple extrusion, simple rotation and sweeping.

Simple Extrusion

The first hex-meshing algorithm is the `SIMPLE_EXTRUSION` algorithm.

```
\begin{SIMPLE_EXTRUSION}
  direction          = 3
  height             = 8.0
  subdivisions       = 8
  start surface name = bottom
  end surface name   = top
\end{SIMPLE_EXTRUSION}
```



Simple Extrusion of a semi-circular mesh

The direction (`x = 1, y = 2, z = 3`) says which direction the extrusion is done. Note that even though the initial 2D mesh is in the x-y plane, the results are rotated to give a mesh extruded in the requested direction. The height tells how far to extrude. A name is given to the bottom and top faces created by the extrusion so that boundary conditions can be attached. Otherwise, the names of the faces are given by the 2D curve names.

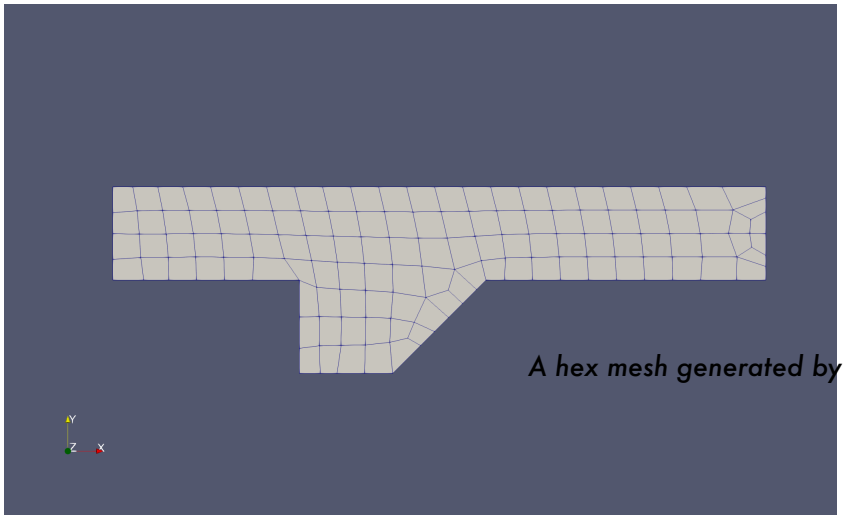
Rotation of a two-dimensional mesh to generate a hex mesh

Simple Rotation

The second algorithm is the `SIMPLE_ROTATION,` which rotates the two dimensional mesh about an axis

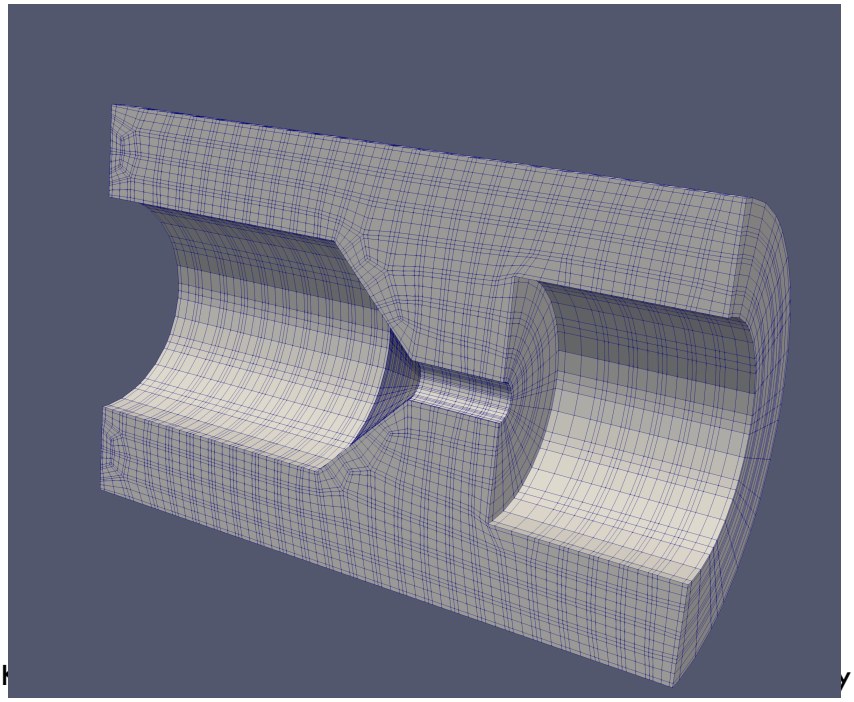
```
\begin{SIMPLE_ROTATION}
  direction             = 1
  rotation angle factor = 1.0
  subdivisions          = 8
  start surface name    = bottom
  end surface name      = top
\end{SIMPLE_EXTRUSION}
```

An example is shown below of an original two dimensional mesh

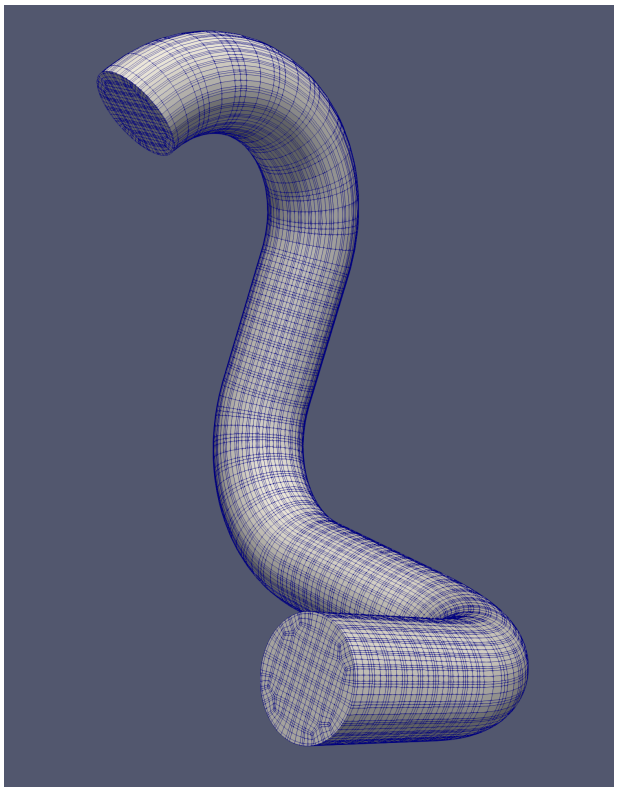


A hex mesh generated by sweeping a circular quad mesh

and its rotation about the x axis (`direction = 1`) is



Sweeping



A cylinder mesh with scaling applied along the sweep direction

The most general algorithm for generating hex meshes in HOHQMESH is to sweep a two-dimensional mesh along a prescribed curve, `SWEEP_ALONG_CURVE`. To sweep along a curve, one does two things: add a `SWEEP_ALONG_CURVE` block to the `CONTOL_INPUT` block and add the curve along which the sweeping is to be done to the `MODEL` block.

There are currently two sweeping algorithms available. The default is a simple rotation algorithm that has no method to counteract twisting of the mesh as it follows the curve. (Think of a roller-coaster that can turn upside down as it follows a curved track.) The default algorithm is exact so will sweep the curve to high order, but will only produce an untwisted mesh if the curve is planar.

The second is a parallel transport algorithm due to Hanson and Ma that keeps arbitrary vector in a particular orientation with respect to its initial direction. The parallel transport approach minimizes the twisting of the hex mesh, but is only second order accurate. [A fourth order algorithm may be implemented in the future.]

To implement sweeping, include a `SWEEP_ALONG_CURVE` block in the `CONTOL_INPUT` block:

```
\begin{SWEEP_ALONG_CURVE}
  algorithm                = Hanson (optional)
  subdivisions per segment = 8
  start surface name       = bottom
  end surface name         = top
\end{SWEEP_ALONG_CURVE}
```

The algorithm keyword is optional. If not present, the sweeping will not include the parallel transport correction. Since the sweep curve can be a chain with slope or curvature singularities, the number of subdivisions per segment is defined. This ensures that a singularity occurs along element boundaries so that accuracy is not lost.

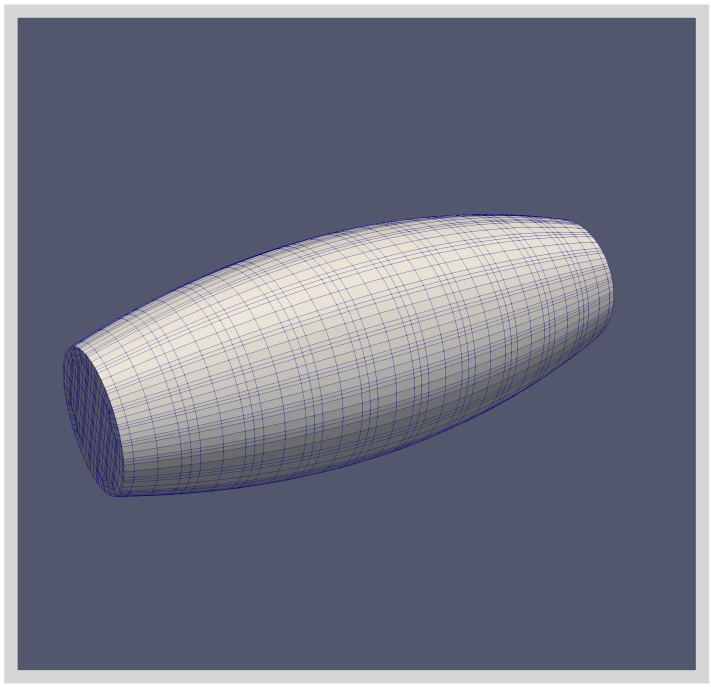
The curve itself is defined in the `MODEL` block.

```
\begin{SWEEP_CURVE}

\end{SWEEP_CURVE}
```

The `SWEEP_CURVE` block implicitly defines a `CHAIN`, like the `OUTER_BOUNDARY` block, and so only needs a list of curves to define the sweep.

Scaling



The mesh can also be scaled in the direction normal to the sweep curve when sweeping is used.

To scale the mesh, add a

```
\begin{SWEEP_SCALE_FACTOR}

\end{SWEEP_SCALE_FACTOR}
```

block to the `MODEL`. Like the `SWEEP_CURVE` and `OUTER_BOUNDARY` blocks, the `SWEEP_SCALE_FACTOR` block implicitly defines a `CHAIN`. You do not need to have the number chain segments match the number in the `SWEEP_ALONG_CURVE` block, but it is probably best to not introduce slope or curvature singularities except at element interfaces.

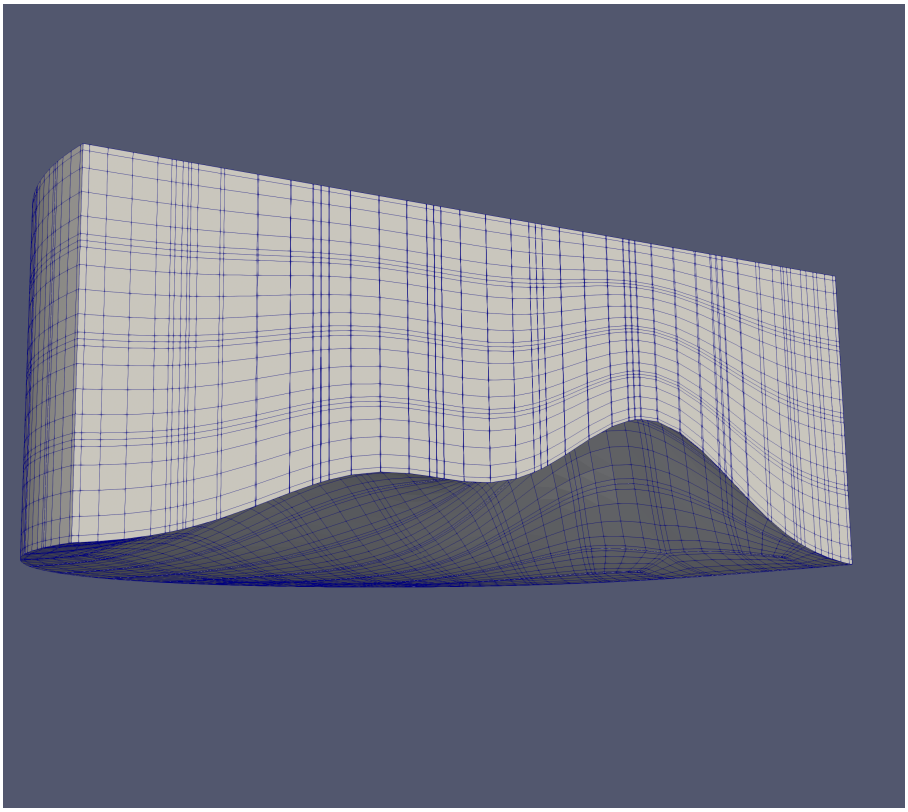
The equation for the scaling is scalar `PARAMETRIC_EQUATION` (as opposed to a `PARAMETRIC_EQUATION_CURVE`). It is defined, for example like this:

```
\begin{PARAMETRIC_EQUATION}
```

```
   eqn = r(t) = 1.0 + 2.5*t*(1-t)
\end{PARAMETRIC_EQUATION}
```

Bottom Topography

When using the `SIMPLE_EXTRUSION` algorithm, bottom topography can be added as shown below:



At this time, the bottom topography is defined only as an equation in a TOPOGRAPHY block, e.g.

```
\begin{TOPOGRAPHY}
   eqn = h(x,y) = x^2 + y^2
\end{TOPOGRAPHY}
```

The height function takes two arguments, which are the *physical* x-y coordinates, unlike the parametric coordinates that define boundary curves.

Chapter 5
HOHQMesh

THE CONTROL FILE

The `MODEL` and the `CONTROL_INPUT` blocks described above are put into a single file called the control file, which is finished with a `\end{FILE}` command. An example of a control file that meshes a model with a circular outer boundary and two inner circular boundaries (See the front cover) is shown below:

```
\begin{CONTROL_INPUT}

   \begin{RUN_PARAMETERS}
      mesh file name    = Circles3Mesh.mesh
      plot file name    = Circles3Plot.tec
      statistics file name    = Circles3Stats.txt
      mesh file format = ISM
      polynomial order = 6
   \end{RUN_PARAMETERS}

   \begin{BACKGROUND_GRID}
     background grid size = [4.0,4.0]
   \end{BACKGROUND_GRID}

   \begin{SPRING_SMOOTHER}
     smoothing type       = LinearAndCrossbarSpring
     number of iterations = 20
  \end{SPRING_SMOOTHER}

\end{CONTROL_INPUT}
```

```
\begin{MODEL}

      \begin{OUTER_BOUNDARY}
         \begin{PARAMETRIC_EQUATION_CURVE}
             name = outer
             xEqn = x(t) = 14.0*cos(2*pi*t)
             yEqn = y(t) = 14.0*sin(2*pi*t)
             zEqn = z(t) = 0.0
         \end{PARAMETRIC_EQUATION_CURVE}
      \end{OUTER_BOUNDARY}

      \begin{INNER_BOUNDARIES}

         \begin{CHAIN}
             name = Boundary 1
            \begin{PARAMETRIC_EQUATION_CURVE}
                  name = Circle1
                  xEqn = f(t) = -10.25 + 0.2*cos(2*pi*t)
                  yEqn = f(t) = 3.0 + 0.2*sin(2*pi*t)
                  zEqn = z(t) = 0.0
             \end{PARAMETRIC_EQUATION_CURVE}
         \end{CHAIN}

         \begin{CHAIN}
             name = Boundary 2
            \begin{PARAMETRIC_EQUATION_CURVE}
                  name = Circle2
                  xEqn = f(t) = -5.1 + 1.0*cos(2*pi*t)
                  yEqn = f(t) = 1.0*sin(2*pi*t) - 4.1
                  zEqn = z(t) = 0.0
            \end{PARAMETRIC_EQUATION_CURVE}
         \end{CHAIN}

         \begin{CHAIN}
             name = Boundary 3
            \begin{PARAMETRIC_EQUATION_CURVE}
                  name = Circle3
                  xEqn = f(t) = -12.0 + 0.5*cos(2*pi*t)
                  yEqn = f(t) = 0.5*sin(2*pi*t) - 0.5
                  zEqn = z(t) = 0.0
            \end{PARAMETRIC_EQUATION_CURVE}
         \end{CHAIN}
      \end{INNER_BOUNDARIES}
\end{MODEL}

\end{FILE}
```

Chapter 6
HOHQMesh

COMPILING AND RUNNING HOHQMESH

Compiling the Mesher

HOHQMesh is currently being distributed on bitbucket in a (private) project of that name. In that repository is a makefile, plus source, documentation (which contains this document) and a directory of examples.

HOHQMesh is written in fortran90. It is known to compile and run with the Intel iFort 13.1 and the gFortran compilers. Let me know if there are issues with other compilers. Or more recent versions of those.

The mesher has one dependency, my FTObjectLibrary, which supplies the container classes and exception classes. It is also available on bitbucket under a project of that name.

The makefile is for gmake. Use it to compile the mesher. Before doing so, modify the top three lines of the makefile to give the paths to your fortran compiler, the HOHQMesh Source, and the FTObjectLibrary Source.

To compile, type on a command line

```
>make -f HOHQMesh.mak
```

That will create the executable named HOHQMesh.

Running the Mesher

To mesh a model defined in a control file, type

```
>./HOHQMesh < controlFileName
```

For example, to mesh the `GingerbreadMan` model in the Examples directory, type

```
>./HOHQMesh < Examples/2D/GingerbreadMan/GingerbreadMan.-
control
```

Alternatively, you can run it with the -f flag:

```
>./HOHQMesh -f Examples/2D/GingerbreadMan/GingerbreadMan.-
control
```

The mesh and plot files will be created relative to the directory of the executable. For the moment, until things get really robust, diagnostic information is printed as the program executes.

Three compiler flags are also defined:

-version	Gives the version number of the code
-help	Does nothing at the moment. Sorry. RTM.
-verbose	Determines whether messages are printed or not.

Use these as usual, e.g.

```
>./HOHQMesh -verbose -f Examples/2D/GingerbreadMan/Ginger-
breadMan.control
```

Chapter 7

HOHQMesh

EXAMPLES

Several examples are gathered in the Examples directory. The table below shows those found in the Examples/2D/ directory. The table lists the control file names and a short description of the domain. Most importantly, it lists which model and gridding properties are featured in each example.

Two Dimensional Meshing Examples

Name	Description	Outer Boundary	Inner boundaries	Chain	Parametric Equation	Spline	Line	Refinement Center	Refinement Line
Box with refinement	Square domain with refinement regions								
Cavity Ramp	A cavity domain with a sloping ramp on the exit side								
Circles3	Two circles enclosed by a large circle		3						
EllipseAndFourCircles			4						
GingerbreadMan	Geometry with lots of holes		6						
Half Circle									
Indian Ocean	Complex domain with islands and inlets		3						
KT3Element	Three element Karman-Treffitz airfoil		3						

Name	Description	Outer Boundary	Inner boundaries	Chain	Parametric Equation	Spline	Line	Refinement Center	Refinement Line
NACA0012	Standard Airfoil geometry		1						
Pill	Oblong domain with interior circles		3						
SplineGeometry	Free form domain defined as a spline								

Appendix-A

HOHQMesh

ADDITIONS FOR THE ISM-V2 MESH FORMAT

The ISM-v2 adds edge information to the mesh file. The first line of the mesh file will state that fact, that is, if the first line is ISM-V2 then it will have the edge information.

Line 1:

```
ISM-V2
```

The second line now also includes the number of edges in the mesh as follows:

```
#nodes, #edges, #elements, polynomial order of boundary edges
```

The edges are read immediately after the nodes. For each edge the following are listed:

```
start node ID, end node ID, element ID on left, element ID on right, side of left element, side of right element
```

These are the quantities that are computed in Alg. 148. If the edge is a boundary edge, then the second side element will be ID = 0 and the side of that element will be 0. If the sides have indices that increase in opposite directions, then the last column in the data will be negative.

Appendix-B

HOHQMesh

SUMMARY: HOW TO SPECIFY BOUNDARY CURVES

Defining a parametric equation:

```
\begin{PARAMETRIC_EQUATION_CURVE}
     name = <name>
     xEqn = x(t) = <x-equation>
     yEqn = y(t) = <y-equation>
     zEqn = z(t) = 0.0
\end{PARAMETRIC_EQUATION_CURVE}
```

Defining a Spline:

```
\begin{SPLINE_CURVE}
   name = <name>
   nKnots = # of nodes
   \begin{SPLINE_DATA}
      t x y z
      .
      .
      .
   \end{SPLINE_DATA}
\end{SPLINE_CURVE}
```

Defining a Straight Line

```
\begin{END_POINTS_LINE}
     name   = <name>
     xStart = [x,y,0]
     xEnd   = [x,y,0]
\end{END_POINTS_LINE}
```

Defining a Circular Arc

```
\begin{CIRCULAR_ARC}
     name        = <name>
     units       =degrees/radians(Optional.Default:radians)
     center      = [x,y,0]
     radius      = r
     start angle = Tstart
     end angle   = Tend
\end{CIRCULAR_ARC}
```

Chaining curves

```
\begin{CHAIN}
     name = <Chain Name>
     First curve definition
     Second curve definition
     ...
     Last curve definition
\end{CHAIN}
```

Appendix-C

HOHQMesh

SUMMARY: HOW TO SPECIFY THE MODEL

No inner boundaries:

```
\begin{MODEL}
      \begin{OUTER_BOUNDARY}
            First curve definition
            Second curve definition
            ...
            Last curve definition
      \end{OUTER_BOUNDARY}
\end{MODEL}
```

No outer boundaries:

```
\begin{MODEL}
      \begin{INNER_BOUNDARIES}
            First chain definition
            Second chain definition
            ...
            Last chain definition
      \end{INNER_BOUNDARIES}
\end{MODEL}
```

Both inner and outer boundaries:

```
\begin{MODEL}
      \begin{OUTER_BOUNDARY}
            First curve definition
            Second curve definition
            ...
            Last curve definition
      \end{OUTER_BOUNDARY}
      \begin{INNER_BOUNDARIES}
            First chain definition
            Second chain definition
            ...
            Last chain definition
      \end{INNER_BOUNDARIES}
\end{MODEL}
```