

# Goat GG tutorial

Author: Sander Van den Kerkhof  
([sander.vandenkerkhof@kuleuven.be](mailto:sander.vandenkerkhof@kuleuven.be))

# Intention of the tutorial

- Workflow/best practices for setting up a new case from scratch without too much prior knowledge on the intrinsic working of the toolbox.
- Not for installing the code (assumed done before)
- Not for running example cases (that is easy when installation is done by running goat.exe in the example folder)

# Learning goals

- Obtain an understanding of the grid generation mechanics to interpret grid generator output
- Set up a new case based on rzpsi.dat, structure.dat (if desired .equ and .dgo)
- Apply the basic topological mesh and grid adaptations
- Deal with commonly occurring issues (small cells, local refinement)
- Basic steps to run SOLPS

# Not treated

- Advanced troubleshooting of known edge cases
- Advanced topological adaptations (nearly connected X-points, in-vessel structures, arbitrary tube removal)

# Summary

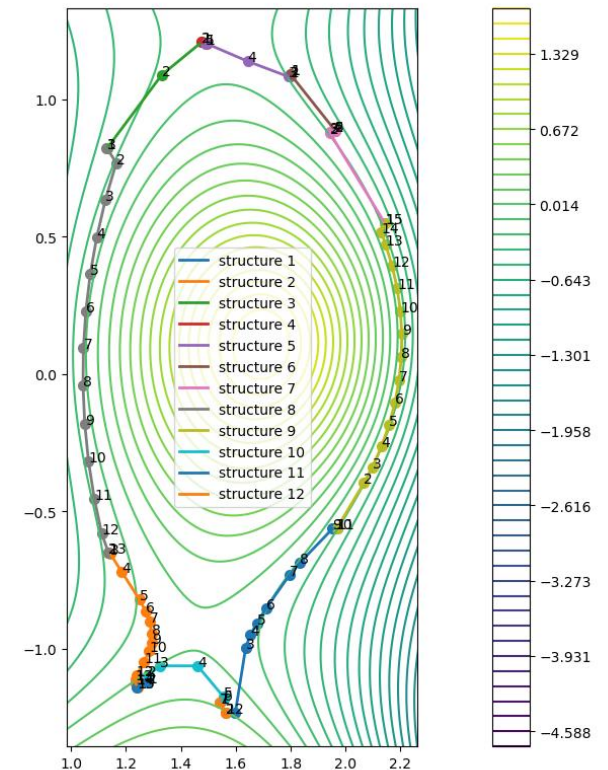
- Typical workflow and main options – only for reference
- Hands-on session(s)

# Step 1: input files

- Goat options (only GG ones relevant for this tutorial): GOAToptions.dat  
→ relevant options will be dealt with in the coming steps. **For a new case:** copy a GOAToptions.dat file from an example case that resembles your new case in terms of geometry and magnetic field – note: all inputs are printed out to the log file when running goat
- Magnetic field data: rzpsi.dat (or equ.dat, but not thoroughly tested) and **Rbtor value to be specified in GOAToptions**
- Vessel data: structure.dat (or .dgo file, but not thoroughly tested)

# Step 1: input files

- **Important:** vessel structure should form closed, non-intersecting polygons (may be nested)
  - **Important:** RBtor value to be specified *unless* starting from.equ file
- ```
# Data  
'goat.mf.RBtor'           '6.66000014543533'
```
- Visualization: 'pgginput' reads structure.dat and rzpsi.dat and shows vessel structures and some magnetic field contours



## Step 2: vessel treatment

- Often, need to **refine vessel** (tangency point determination is based on vessel vertices: too coarse → horrible errors in topomesh construction)
- Sometimes, need to **exclude vessel structures** (e.g. triangles from CARRE2 mesh setup...)
- Representation may be smoothed (but then inaccurate w.r.t. original polygon)

# Step 2: vessel treatment

Vessel refinement options

```
# Refinement options for vessel  
'goat.vessel.refinevessel'      '1'  
'goat.vessel.maxvesseldist'    '1e-2'
```

Exclude structures

```
# Structures to be excluded when constructing vessel (if empty, all are included)  
'goat.vessel.exclude'          '[3, 5]'
```

Vessel representation

- stick with closedpolygon\_exact,  
rest is then not relevant

```
# Vessel field description (similar to magnetic field interpolant,  
# but for vessel - only used for vessel distance penalty, and if shape  
# method is the same, also used for boundary constraints)  
'goat.vessel.shapemeth'        'closedpolygon_exact'  
'goat.vessel.resx'             '100'  
'goat.vessel.resy'             '100'  
'goat.vessel.offsetfracx'      '0.1'  
'goat.vessel.offsetfracy'      '0.1'  
'goat.vessel.interpC'          '3'  
'goat.vessel.interpM'          '6'
```

# Step 2: vessel treatment

- Recommendations:
  - Refine vessel to 1 cm, go finer if very strange errors in topomesh construction pop up before a basic topological mesh was constructed
  - Take 'closedpolygon\_exact' representation

# Step 3: topological mesh settings

- Base topological mesh should be constructed without further user interference and is saved in './output/topomesh\_base.dat'
- Final topomesh is saved in './output/topomesh.dat'
- Existing topomesh can also be loaded in (then also load the tracers used to construct the topomesh)

```
# Read existing file?
'gg.tm.readexistingTM'          '0'
'gg.tm.readexistingtracers'     '0'
'gg.tm.writedebugoutput'       '0'
'gg.tm.TMfilepath'             './output/topomesh.dat'
```

# Step 3: topological mesh settings

- Available adaptations:
  - Inserting core contours (often combined with core region removal)
  - Removing wide grid regions (any region without a separatrix segment as aligned boundary)
  - Removing user-specified regions (advanced, if needed see user guide)
  - Region merging
  - Aligned vessel part insertion
- We go briefly over the options, see user guide for more details

# Step 3: topological mesh settings

Main switch to apply adaptations

~~# Core boundary and region treatment~~

'gg.tm.doadaptations' '1'

Core region (and PF) manipulation

'gg.tm.addcoreboundaries' '1'

'gg.tm.addPFboundaries' '0'

'gg.tm.cbnd.frac' '0.1' #percentage of psi-values

'gg.tm.PFbnd.frac' '0.3'

'gg.tm.removecoreregions' '1'

Wide grid region removal

'gg.tm.removewidegridregions' '0'

Redacted: deprecated option

Specific vessel region removal

'gg.tm.removevesselregions' '0'

'gg.tm.rvrretain' '0'

'gg.tm.rvrvesselIDs' '[1,7,8]'

'gg.tm.rvrdocascade' '0'

'gg.tm.rvrcascadedir' 'up'

'gg.tm.rvrfullycovered' '0'

Aligned vessel part insertion near tangency points

'gg.tm.alignvesselparts' '1'

'gg.tm.avpminangle' '10.0'

Redacted: deprecated option

# Step 3: topological mesh settings

Region merging options

```
477 'gg.tm.mergetubes'      '1'  
478 'gg.tm.mergetubes.meth' 'extensive'  
479 'gg.tm.dpsimintubes'   '0.0'  
480 'gg.tm.lradmintubes'  '0.5e-2'  
481
```

# Step 3: topological mesh settings

- Recommendations:
  - **(for SOLPS) Remove core regions** by adding core boundaries ('gg.tm.addcoreboundaries' '1') and removing core regions ('gg.tm.removecoreregions' '1')
  - Remove wide grid regions if desired
  - **Merge tubes** (tubes=regions) **based only on radial width** and set that to desired minimal resolution (use the 'extensive' setting)
  - **Align vessel parts at tangency points**, angle values between **5°-10°** are often sufficient
- If things go wrong, they'll likely go wrong in the merging or aligned vessel part adaptations

# Step 4: grid generator settings

- Mainly radial and poloidal resolution has to be specified at this point
- Roughly two ways to manipulate this:
  - Vertex density (#  $m^{-1}$  radial/poloidal length) distribution
  - Local refinement (if possible) based on maximal and minimal length distributions
- The distribution functions are used in a first step to obtain an initial distribution on the topomesh faces. Then, the refiners are applied

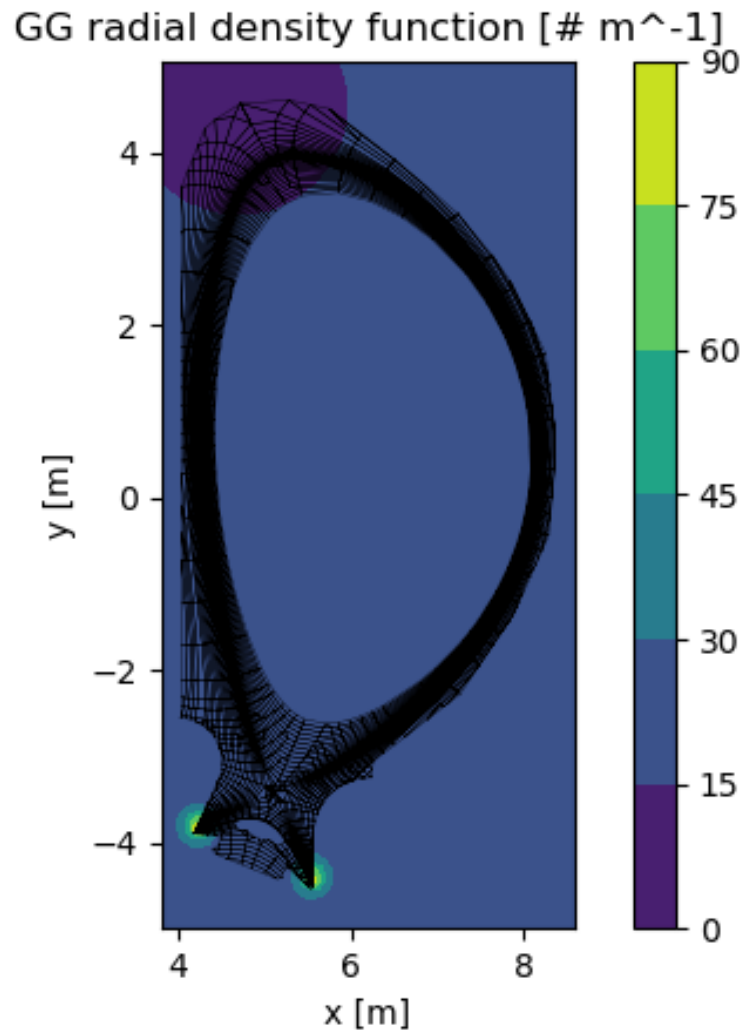
# Step 4: grid generator settings

- Though influencing the poloidal resolution is rather straightforward, the radial one is not because we can't refine locally in the radial direction
- For each topological region, a face is chosen on which points are distributed to trace flux surfaces → this leads inevitably to mismatch far away from that surface due to flux expansion effects
- It is not always clear which topological face is used → radial refinement can be tricky (still working on a solution for that...)

# Step 4: grid generator settings

- **Most refinement options are set with distribution functions**, so some explanation on those
- Currently formulated as a sum of decaying exponentials that are characterized by (x, y) coordinates, a decay length, and a value at the point. Also the value at infinity should be set
- **The value is interpreted as number of lines per meter poloidal/radial length**. So e.g. 1000 corresponds to millimeter-wide cells
- Depending on the case, some points of interest can automatically be added (e.g. X-points, strike points, entire targets, ...)

# Step 4: grid generator settings



```
557 # Options for density based distribution, radial
558 'gg.vd.rd.distribution.dosp'      '1'
559 'gg.vd.rd.distribution.decaylengthsp' '0.05'
560 'gg.vd.rd.distribution.densityatp'  '125'
561 'gg.vd.rd.distribution.densityatinf' '20'
562 'gg.vd.rd.distribution.points.x'    '[4.7]'
563 'gg.vd.rd.distribution.points.y'    '[4.55]'
564 'gg.vd.rd.distribution.points.d'    '[1.0]'
565 'gg.vd.rd.distribution.points.val'  '[2]'
```

# Step 4: grid generator settings

- Poloidal distribution settings (not so important actually)

Type of distribution function ('uniform' or 'densitybased')

Desired face length (only for uniform distribution)

Distribution function options for density-based method (density at vessel and infinity can be specified, plf (polygon levelset function) can be based on targets ('target') or entire vessel ('vessel'))

```
# Distribution of vertices on flux surfaces (poloidal)
'gg.vd.pd.type' 'uniform'

# Options for uniform distribution
'gg.vd.pd.distribution.facelength' '1e-2';

'gg.vd.rd.distribution.fieldwidth' '1.25e-1';
'gg.vd.rd.distribution.facelength' '1e-1';

# Options for density based distribution
'gg.vd.pd.distribution.plftype' 'target';
'gg.vd.pd.distribution.decaylengthplf' '0.1';
'gg.vd.pd.distribution.densityatvessel' '5';
'gg.vd.pd.distribution.densityatinf' '5';
```

# Step 4: grid generator settings

- Poloidal refiner settings (more important)

Refine near X-points/vessel structures?

Minimal and maximal length at infinity

Structures to consider in structure.dat where you want to do refinement + their distribution options for minimal and maximal length distributions

Same as for vessel structure, but now for X-points

```
# Refinement options
'gg.ref.readexistingrefdata'      '0'
'gg.ref.meth'                     'lengthbased'
#'gg.ref.meth'                    'no'
'gg.ref.LB.doxp'                  '1'
'gg.ref.LB.dovessel'              '1'
'gg.ref.LB.Lmininf'               '0.1'
'gg.ref.LB.Lmaxinf'               '0.8'
'gg.ref.LB.structureIDs'          '[11, 12]'
'gg.ref.LB.Lminstructure'         '[0.02, 0.02]'
'gg.ref.LB.Lmaxstructure'         '[0.08, 0.08]'
'gg.ref.LB.decaylengthstructure'  '[0.2, 0.2]'
'gg.ref.LB.Lminxp'                '0.0001'
'gg.ref.LB.Lmaxxp'                '0.05'
'gg.ref.LB.decaylengthxp'         '0.4'
```

# Step 4: grid generator settings

- Radial distribution settings

Set distribution near strike points  
(mainly for SOL refinement)

Set distribution near user specified points

Set distribution near x-points (mainly for core refinement)

```
# Options for density based distribution, radial
'gg.vd.rd.distribution.dosp'      '1'
'gg.vd.rd.distribution.decaylengthsp'  '0.1'
'gg.vd.rd.distribution.densityatsp'    '125'
'gg.vd.rd.distribution.densityatinf'   '20'
'gg.vd.rd.distribution.points.x'      '[4.7]'
'gg.vd.rd.distribution.points.y'      '[4.55]'
'gg.vd.rd.distribution.points.d'      '[1.0]'
'gg.vd.rd.distribution.points.val'    '[2]'

'gg.vd.rd.distribution.doxp'      '1'
'gg.vd.rd.distribution.decaylengthxp'  '0.1'
'gg.vd.rd.distribution.densityatxp'    '10'
```

# Step 4: grid generator settings

- Radial refiner options (less important)

```
# Refinement options
'gg.radref.meth'           'lengthbased'
#'gg.radref.meth'         'no'
'gg.radref.LB.lengthtype' 'radial'
'gg.radref.LB.dosp'       '0'

# Options for (radial) length type
'gg.radref.LB.Lmininf'    '0.0001'
'gg.radref.LB.Lmaxinf'    '1000.0'
'gg.radref.LB.Lminsp'     '0.04'
'gg.radref.LB.Lmaxsp'     '0.10'
'gg.radref.LB.decaylengthsp' '0.5'
```

# Step 4: grid generator settings

- Boundary layers are applied through the refiner by setting additional boundary layer options:
  - Poloidal direction: near targets, entire vessel, or user defined structures
  - Radial direction: near strike points only

```
# Boundary layer options (only for length based refinement for now)
'gg.ref.BL.dotarget'      '0'
#'gg.ref.BL.nctarget'     '7'
#'gg.ref.BL.dltarget'     '[0.002, 0.003, 0.004, 0.006, 0.008, 0.012, 0.016]'
'gg.ref.BL.nctarget'     '7'
'gg.ref.BL.dltarget'     '[0.0005, 0.00075, 0.001, 0.0015, 0.002, 0.003, 0.004]'

'gg.ref.BL.dovessel'     '0'
'gg.ref.BL.ncvessel'     '4'
'gg.ref.BL.dlvessel'     '[0.0005, 0.00075, 0.001, 0.0015]'
#'gg.ref.BL.dlvessel'     '[0.0005, 0.00075, 0.001, 0.0015, 0.002, 0.003, 0.004, 0.006, 0.008]'
'gg.ref.BL.dllengthbased' '1'

'gg.ref.BL.dostructure'  '1'
'gg.ref.BL.structureID'  '[1, 3, 5, 7, 10]'
'gg.ref.BL.ncstructure'  '4'
'gg.ref.BL.dlstructure'  '[0.0005, 0.00075, 0.001, 0.0015]
```

Options for poloidal boundary layer:

- nc: number of cells
- dl: lengths in [m]
- dotarget/dovessel/dostructure: switches to apply BL there
- structureID: structures to consider

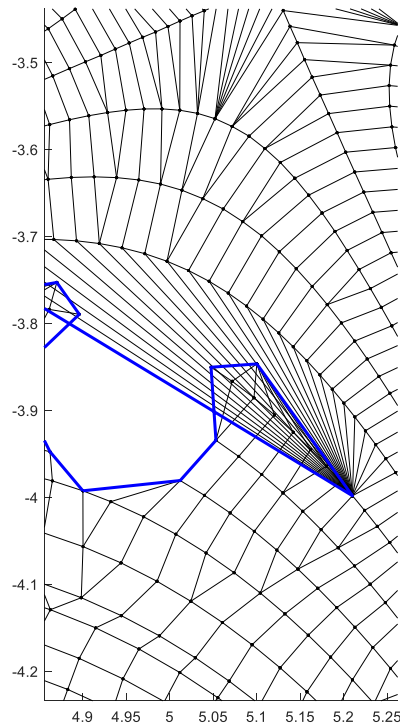
# Step 4: grid generator settings

- Radial boundary layer options: only near strike points (radial width should be given, options similar to poloidal boundary layer)

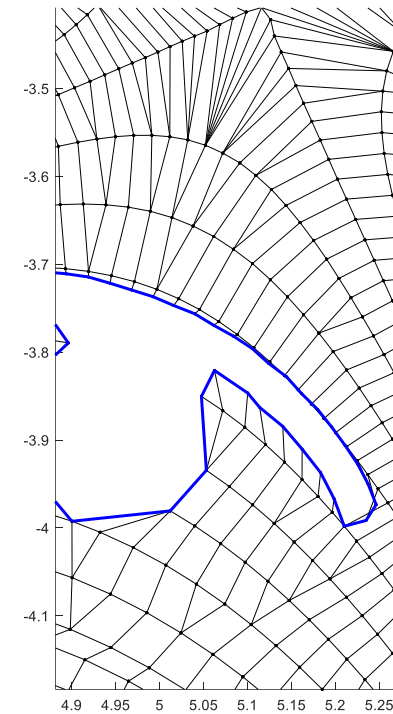
```
# 'Boundary' layer options for radial direction
'gg.radref.BL.dosp'          '0'
'gg.radref.BL.ncsp'          '9'
'gg.radref.BL.dlsp'          '[0.0010, 0.0015, 0.002, 0.003, 0.0045, 0.0067, 0.015, 0.015, 0.025]'
```

# Step 4: grid generator settings

- Finally, field lines may be extended with vessel parts to deal with otherwise large boundary faces.



No extension → large stacked triangles



Extension → Cut-cell like behavior, vessel geometry maintained

# Step 4: grid generator settings

- Currently only applied based on total vessel part length, which should be specified as a threshold

Apply extension? (just put both to one, options will be merged in the future)

Maximal vessel length above which extension is applied (in [m])

```
# adaptations
'gg.adap.extendtptubes'      '1'
'gg.adap.extendvesseltubes'  '1'
'gg.adap.evt.maxvessellength' '0.05'
```

# Step 4: grid generator settings

- Recommendations:
  - Manipulate **poloidal resolution through refiner** and **radial resolution through distribution function** → yields best results it seems
  - Poloidal resolution: set initial distribution to be very coarse, then use  $L_{\max}/L_{\min}$  for refinement
  - Radial resolution: set refiner min and max length very far apart so it is not triggered, set initial distribution to be the desired one
  - **Apply tube extension** at relatively large lengths (e.g. 0.1 meter)
  - **Apply boundary layers** at targets/desired structures or near strike points
  - **Use ‘pggoutput’ to visualize topological mesh, distribution functions, ...**

# Step 4: grid generator settings

- After a first step, you may have to troubleshoot. Some common issues:
  - Overlapping cells are typically caused if vessel tube extension length was set too large → decrease extension length threshold
  - Poloidal boundary layers are not imposed if vessel tubes are extended → increase radial resolution
  - Lack of resolution/too large poloidal resolution is often caused by user input
  - The radial resolution often requires tuning due to flux expansion effects at X-points (also those that are not in the mesh...)
  - If overall resolution is too coarse, anything may happen (rule of thumb: cell length scales should be below magnetic field/vessel length scales)
  - Combination of boundary layers and density based distributions may lead to small features if coarsening is not applied (e.g. radial direction)

# Step 5: post-processing

- Two things can be done:
  - Checking for overlapping cells (expensive: we check intersections of each face pair, so scales as  $O(N^2)$  with  $N$  number of grid faces). When using pggoutput, intersections are visualized if computed and present (also written out in output)
  - Adjusting face labels etc for simulation with SOLPS – recommended (note: face *regions* won't necessarily be updated!)

```
402     'gg.labels.structurebased'           '1'  
403     'gg.dogriddiagnostics'               '1'
```

# Step 5: post-processing

- For SOLPS compatibility:
  - Face labels are best based on vessel structures for boundary conditions
  - Negative labels are assigned to last flux surfaces (for core: -21, -25, -29, ... are reserved)
  - Cell regions are random in general, apart for core (there: cvReg = 1, 5, 9, ... )
  - Face regions are random in general
- There is a *very rudimentary* implementation that attempts to identify the topology and remap also fcReg and cvReg to the correct values, but please don't trust this!

# Step 5: post-processing

- Grid output is written in case folder, name can be changed:

```
# Output file locations  
'goat.writefilepath'           'traduit.out.b2us'
```

- Note that by default post-processing scripts assume 'traduit.out.b2us' to be the grid file
- Also an .ogr file is written if you want to visualize in DivGeo...

# Step 6: visualization of output

- Analogously to 'pgginput', you can type 'pggoutput' in the terminal to visualize:
  - Grid, assuming name is 'traduit.out.b2us'
  - Topomesh, assuming in './output/' and named 'topomesh.dat'
  - Structure, assuming name is 'structure.dat'
- Still need to support DivGeo input, but using that route you get the 'vessel\_dgo.dat' and 'vessel\_tria\_dgo.dat' files that can be plotted as a structure.dat file

# Hands-on session 1

setting up a new case based on rzpsi.dat and structure.dat files and dealing with commonly encountered issues on the grid side

Git commit SHA: 24f3e7dda6fd17ee614837dd3d1d85b7bbf450b8

# Scope

- Go through the entire workflow for a single case setup, including pre- and post-processing
- Based on the ASDEx example case, but can be any case by just replacing the rzpsi.dat and structure.dat files
- Focus on most common issues encountered in grid generation, not on getting it to work in SOLPS (if interested in that, just skip all the grid improvement steps and work with the first grid you get)

# Step 0: initialization

- Assuming the code has been installed correctly, just open a terminal in the goat top directory and do the setup steps:
  - 'tcsh' (if not in tcsh shell)
  - 'source setup.csh'
  - 'make goat' or 'make goat\_debug'
- Create the 'Runs' directory parallel to the 'Examples' directory in the repo (will be ignored by git)

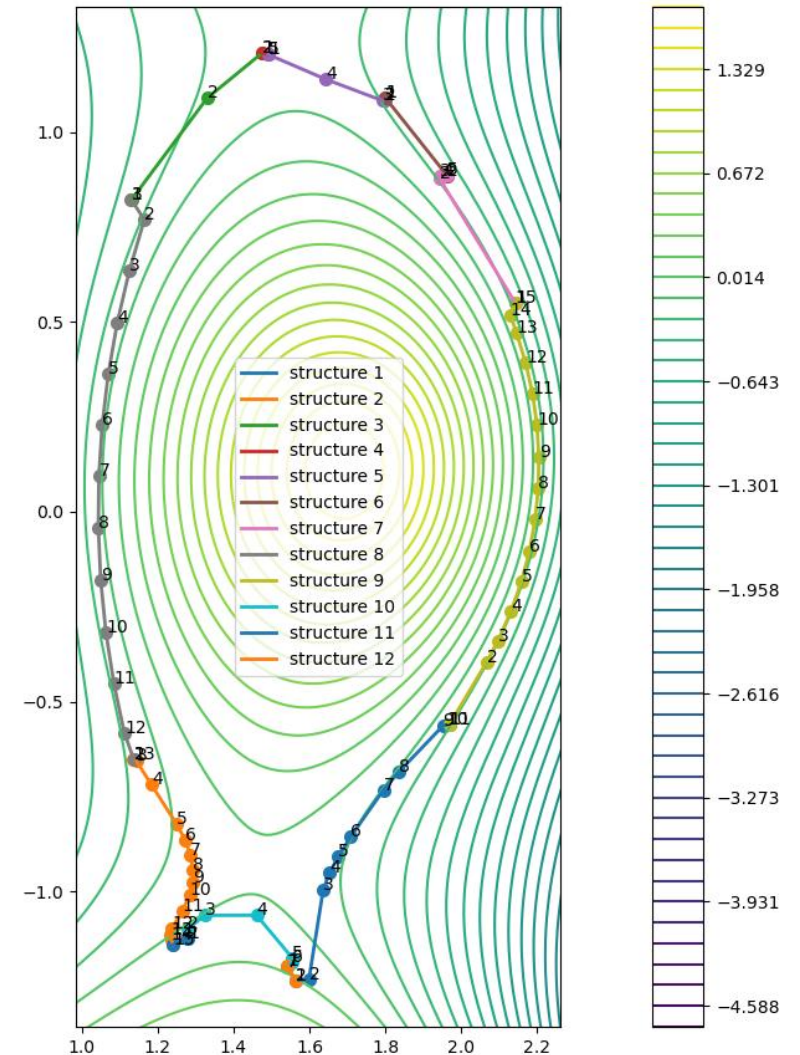
# Step 1: input files

- Create a new folder in Runs, e.g. 'MyASDEX'
- Copy the GOAToptions.dat, rzpsi.dat and structure.dat from Examples/ASDEX into this folder
  - Note: GOAToptions.dat can also be set up from scratch, but this is error prone, labour intensive and inefficient
- Change the Rbtor value if needed (may be obtained by btor.dat)
- Navigate in the terminal to that folder (for the remainder of the session, we assume that you are in that location!)

```
105
106 # Data
107 'goat.data.Rbtor' '-4.496825999658907'
108
```

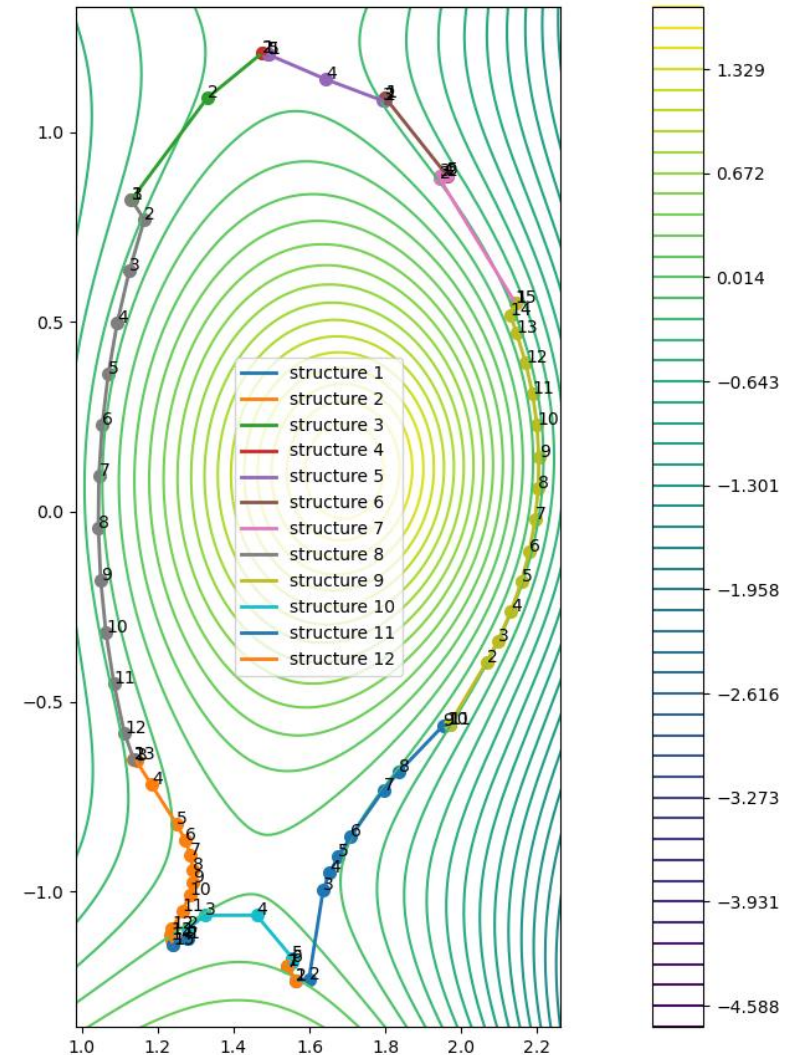
# Step 1: input files

- To have an idea what we are up against, we can plot some Qols of the input files for the grid generator by executing 'pgginput' in the terminal (see figure to the right)
- Most important here are the structures: the structure numbers are typically used for refinement etc. This is also a first check to see if the structure forms a closed polygon
- Note: pgginput expects rzpsi.dat and structure.dat to be present, otherwise nothing is shown



# Step 1: input files

- Important observation 1: the vessel polygon is quite coarse, so we need to refine it to capture tangency points
- Important observation 2: structures 1 and 2 are target structures where we want refinement
- Important observation 3: no CARRE-like triangle structures etc present, which is good
- Important observation 4: the structures form closed polygons that don't self-intersect



# Step 1: input files

- If you want, you can also change the names of input/output files
- Here we assume that standard names are used (rzpsi.dat, structure.dat, traduit.out.b2us)

# Step 2: vessel treatment

- Based on previous observations, we do the following:
  - We don't mark any vessel structures that need to be ignored
  - We do vessel refinement, centimeter-sized vessel segments as starting point (make smaller if needed)

```
# Environment (including vessel etc) options
# Vessel geometry & reading method
'goat.vessel.readmeth'      'read_structure'

# Refinement options for vessel
'goat.vessel.refinevessel'  '1'
'goat.vessel.maxvesseldist' '1e-2'

# Target plate definition
'goat.vessel.TP'           '[1, 2]'
'goat.vessel.TPind'        '[1, 2]'

# Structures to be excluded when constructing vessel (if empty, all are included)
'goat.vessel.exclude'      '[]'
```

We read from a structure.dat file

We do vessel refinement and set the maximal vessel edge distance to 1cm

We define the target plates (not really used anymore, legacy setting to be removed)

All vessel structures to be included

# Step 3: generating an initial topomesh (and grid)

- Instead of going through all options at once, let's try to generate an initial grid without doing any topological mesh adaptations
- This is a good first step, because this will give an indication where things may go wrong.

# Step 3: generating an initial topomesh (and grid)

- To do so, set the following settings in the GOAToptions.dat file:
  - Start from scratch
  - No topological mesh adaptations
  - Keep all other options as-is (grid may be bad, but we don't care yet at this stage)

```
#-----#
#                TOPOLOGICAL MESH OPTIONS                #
#-----#
# Read existing file?
'gg.tm.readexistingTM'      '0'
'gg.tm.readexistingtracers' '0'
'gg.tm.writedebugoutput'   '0'
'gg.tm.TMfilepath'         './output/topomesh.dat'
'gg.tm.TMfieldtracerfilepath' './output/TMfieldtracer.dat'
'gg.tm.TMvesseltracerfilepath' './output/TMvesseltracer.dat'

# Topomesh adaptations
'gg.tm.doadaptations'      '0'
'gg.tm.addcoreboundaries'  '1'
'gg.tm.addPFboundaries'    '0'
'gg.tm.cbnd.frac'          '0.1'
'gg.tm.PFbnd.frac'         '0.1'
'gg.tm.removeconnections'  '1'
```

Start from scratch

Don't do any adaptations

# Step 3: generating an initial topomesh (and grid)

- Execute 'goat.exe' or 'goat\_debug.exe' in the terminal
- A lot of output will be printed:
  - All key-value pairs from the input file and whether they were read in (check this if you get weird results, may be that an option is not read in because of a typo)
  - Several warning messages etc
- All of this may be ignored if the output files are generated and you see something like this at the end:

```
!=====!  
Running SOLPS topology identification. There is  
    NO GUARANTEE WHATSOEVER  
that this routine will yield proper results for SOLPS.  
Please examine the resulting fcReg, cvReg and ftReg  
carefully afterwards  
!=====!  
Identified a general topology  
ErrorHandler: no recorded errors  
ErrorHandler: not tracking errors
```

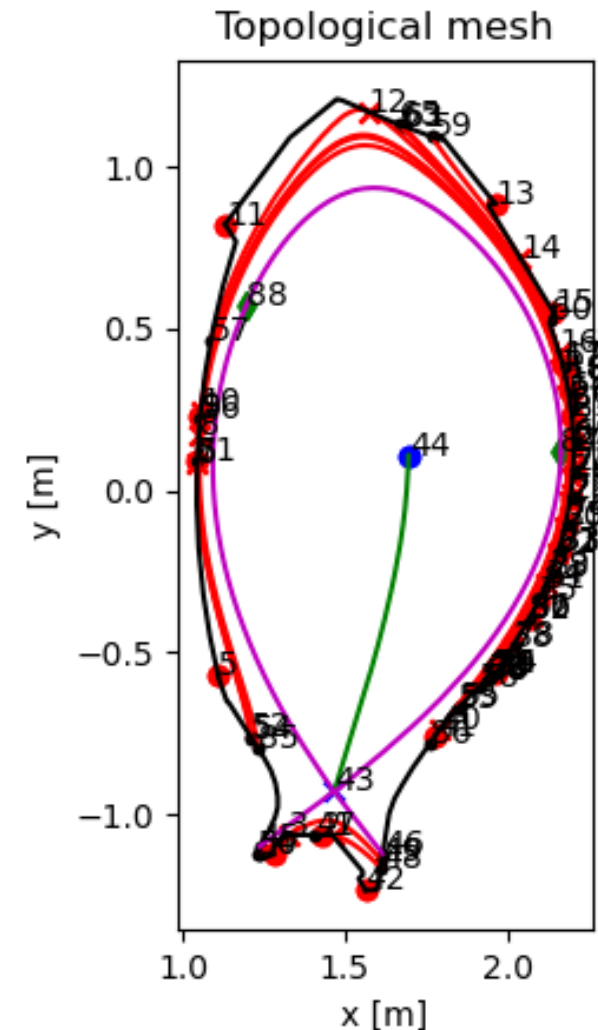
## Step 3: generating an initial topomesh (and grid)

- Seems like everything went quite ok. Let's have a look at the output by running 'pggoutput' (this will print quite a lot of figures)
- Note: if you changed names you can also do 'pggoutput [grid\_name] [topomesh\_name] [structure\_name]'
- It is assumed that the topomesh is stored in './output'

# Topomesh

- This is the base topomesh that will not change unless input is changed
- Important observation 1: the core region is still present, as expected
- Important observation 2: there are many tangency points due to the nearly aligned vessel
- Important observation 3: there is no closed flux surface ('core boundary') in the core plasma

Figure 1



# Grid

- Looks OK *visually*
- But looking at cell size/transitions, there's a lot of crap at the LFS due to the topomesh
- Figure 9 shows the smallest cell in the grid, which has a surface area of  $<10^{-7} \text{ m}^2$  ... not great

Figure 8

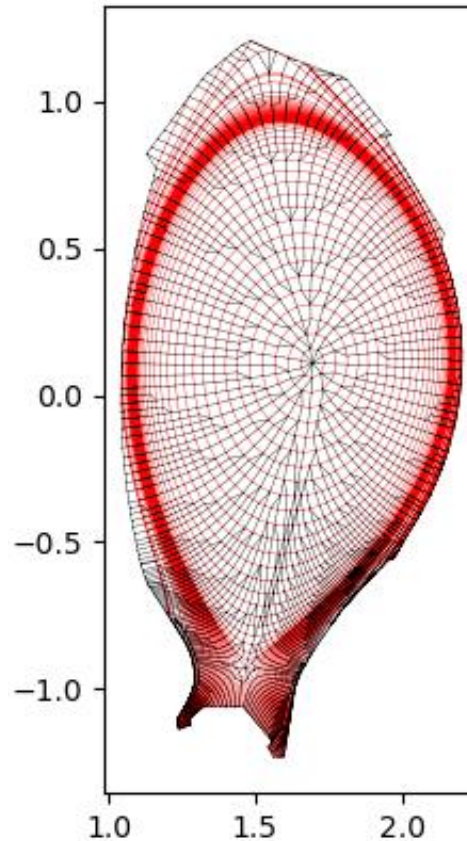
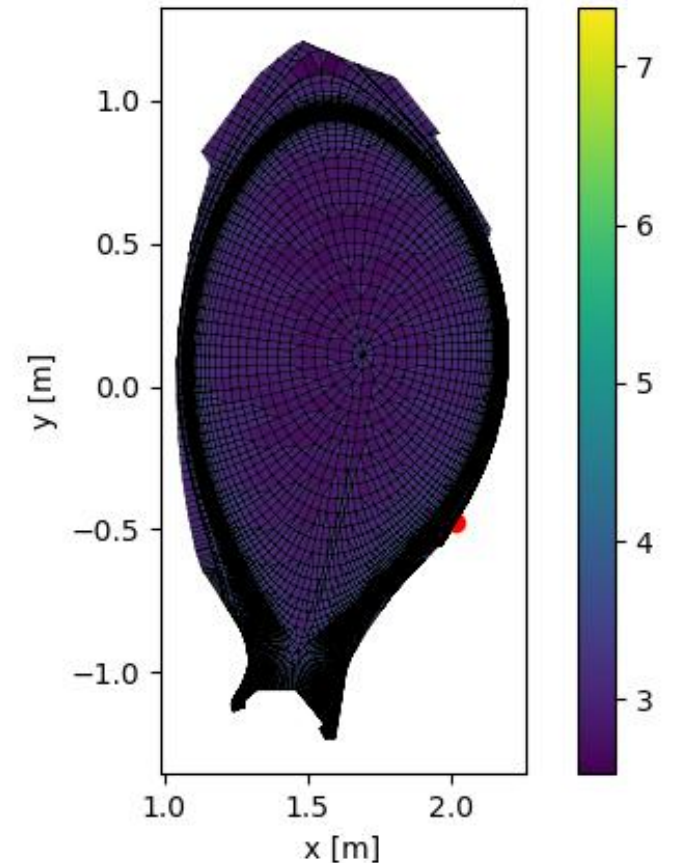


Figure 9

Log10 of grid cell surface area (inverse) [ $\text{m}^{-2}$ ]



# How do we improve this?

- **It is key to realize that the grid generator generates grids based on the topomesh flux tubes**
- This means that if those tubes are narrow, the grid cells will be narrow there. This is often not desired.
- Hence, we need to modify the topological mesh using adaptations
- If things go wrong, it is very likely in this step

# Applying topological mesh adaptations

- Let's say we do want to have a wide grid, but don't want a core, nor we want to resolve all the small features near the LFS. Therefore:
  - We remove the core region in the topomesh
  - We merge topomesh tubes until tubes are at least, say, 1 mm wide in the radial direction (this is only approximate!)
  - To save time, we load in the base topomesh and start adaptations from there.

```
# Read existing file?  
'gg.tm.readexistingTM'      '1'  
'gg.tm.readexistingtracers' '1'  
'gg.tm.writedebugoutput'   '0'  
'gg.tm.TMfilepath'         './output/topomesh_base.dat'  
'gg.tm.TMfieldtracerfilepath' './output/TMfieldtracer.dat'  
'gg.tm.TMvesseltracerfilepath' './output/TMvesseltracer.dat'
```

Sander Van den Kerkhof, KU Leuven, department of  
mechanical engineering

Read the previously generated base topomesh, which is located in `./output/topomesh_base.dat`. Always read in the generated tracers too for discrete consistency (to be automated in the future)

# Applying topological mesh adaptations

```
# Topomesh adaptations
'gg.tm.doadaptations'      '1'
'gg.tm.addcoreboundaries'  '1'
'gg.tm.addPFboundaries'    '0'
'gg.tm.cbnd.frac'          '0.1'
'gg.tm.PFbnd.frac'         '0.1'
'gg.tm.removecoreregions'  '1'
'gg.tm.removewidegridregions' '0'
```

Enable adaptations (main switch)

Add core boundaries

Remove core plasma

```
'gg.tm.mergetubes'         '1'
'gg.tm.mergetubes.meth'    'extensive'
'gg.tm.dpsimintubes'       '0.0'
'gg.tm.lradmintubes'       '1.0e-3'
'gg.tm.mergetubes.allowseparatrix' '0'

'gg.tm.simplifyseparatrix'  '1'
'gg.tm.simplifyseparatrix.lpolsepmin' '1e-3'

'gg.tm.removevesselregions' '0'
'gg.tm.rvrretain'           '0'
'gg.tm.rvrvesselIDs'        '[2, 3]'
'gg.tm.rvrdocascade'        '1'
'gg.tm.rvrcascadedir'       'up'
'gg.tm.rvrfullycovered'     '1'

'gg.tm.alignvesselparts'    '0'
'gg.tm.avpminangle'         '5.0'
```

Merge tubes (ignore the method for now)

Minimal tube radial width (don't work with dpsim, this is not recommended)

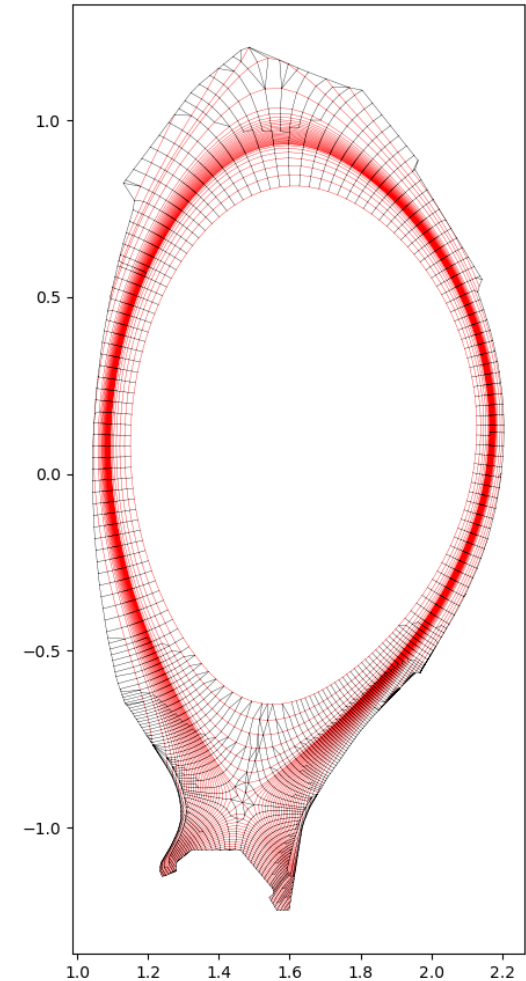
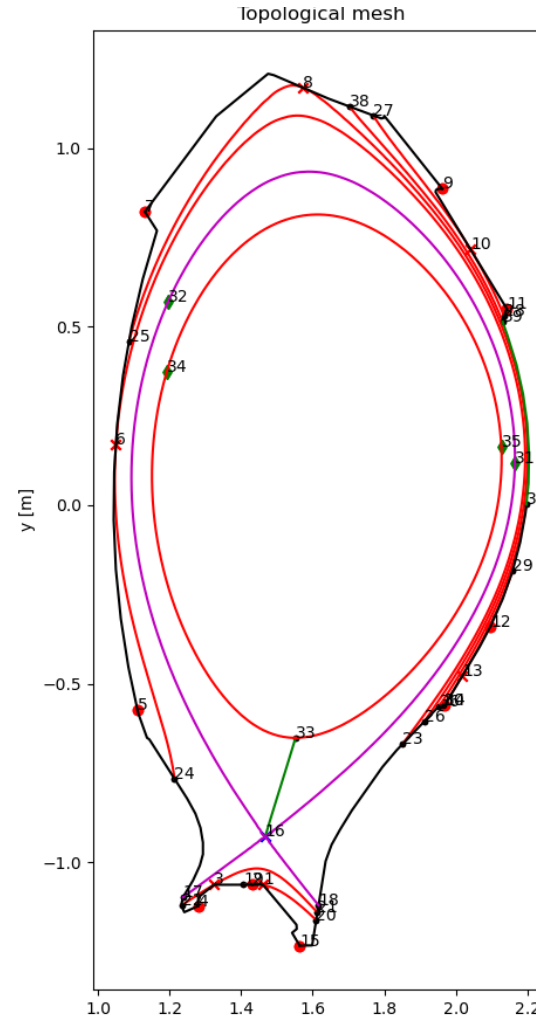
Don't allow separatrix merging (advanced topic)

Don't remove any other vessel regions

Don't align at tangency points

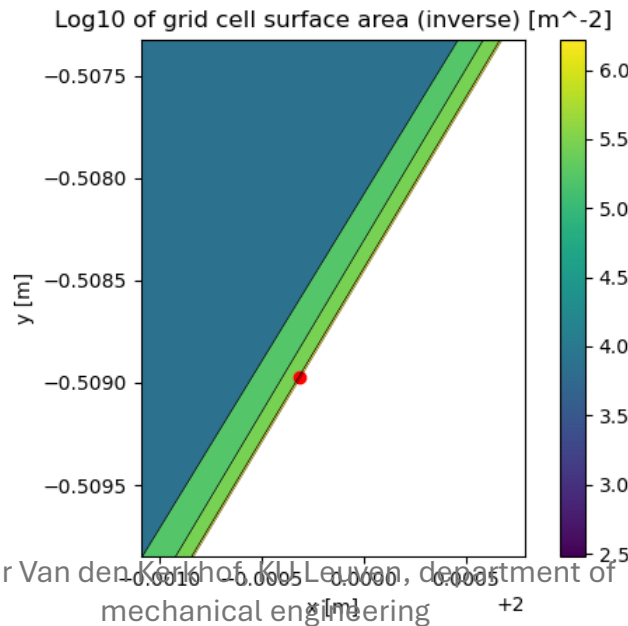
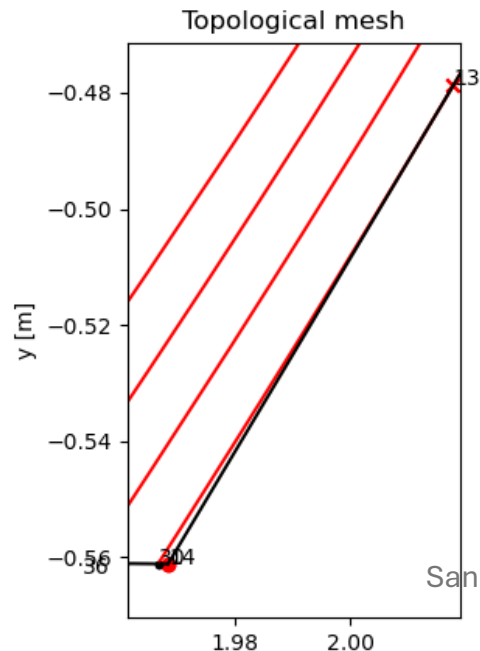
# Applying topomesh adaptations

- Let's see what we got by executing 'pggoutput' again
- Observation 1: core removed, as requested
- Observation 2: less junk at LFS leading to cleaner grid

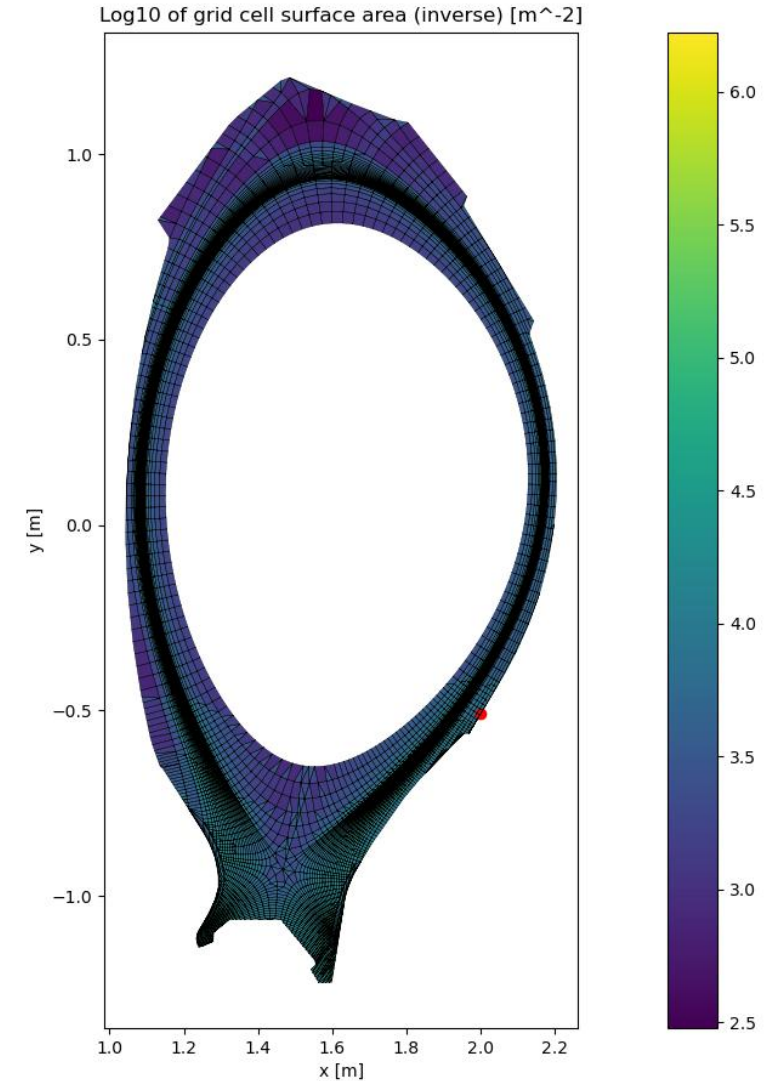


# Applying topomesh adaptations

- Let's have a look at the cell volumes
- 'Not great, not terrible'
- But why? See topomesh...



Sander Van den Kerkhof, KU Leuven, department of mechanical engineering



# Why do we see this?

- Observation 1: we ‘see’ that small triangles are formed there
- Observation 2: this happens in a ‘tangency point tube’
- GG mechanic 1: tubes with type 1 tangency points (contour outside of vessel) consist of a flux surface and a tangency point. Hence, without special care, only triangles are formed in that tube
- GG mechanic 2: by allowing ‘extension’ of the flux surfaces/tangency point, also parts of the vessel are considered, leading to *possibly* better conditioned cut-cell-like cells

# How do we improve this?

- Several (combinable) options:
  - Increase the minimal tube radial width in the adaptations (by itself will only lead to larger triangles)
  - Allow extension of tube flux surfaces – this is what we need here
  - Reduce local grid resolution (not really desired/necessary)
- How does this (currently) work?
  - IF the vessel part between both flux surfaces is large enough (see options) the vessel part is added as flux surface (but still considered to be non-aligned of course)
  - This will lead to vertices on that flux surface

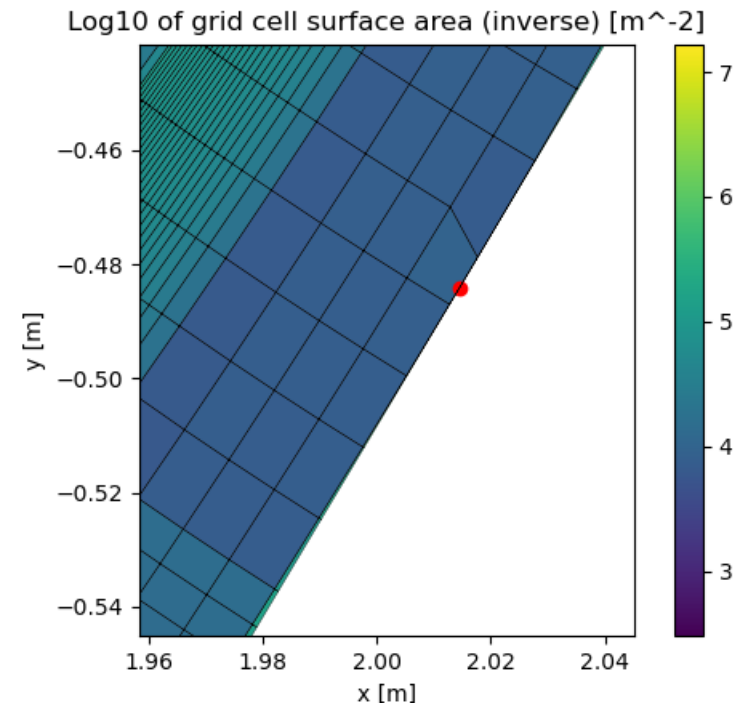
# Dealing with bad triangles near vessel parts

- We can set the following options:
  - Extendtptubes: extend only tubes with tangency points
  - Extendvesseltubes: extend (if needed) also all other tubes
  - Maxvessellength [m]: extend if the length of that part exceeds the set length
- Note: this length parameter is a bit annoying to work with, hope to improve this in the future based on local desired resolution etc
- This was already on! Then why didn't the grid generator do this?

```
# adaptations
'gg.adap.extendtptubes'      '1'|
'gg.adap.extendvesseltubes'  '1'
'gg.adap.evt.maxvessellength' '0.1'
```

# Dealing with bad triangles near vessel parts

- Let's put maxvessellength to 0,05.
- We get what we expected, but still small cells!
- Why? Inherent because of the almost aligned vessel! Cells will always be extremely small in the radial direction there due to the tangency point

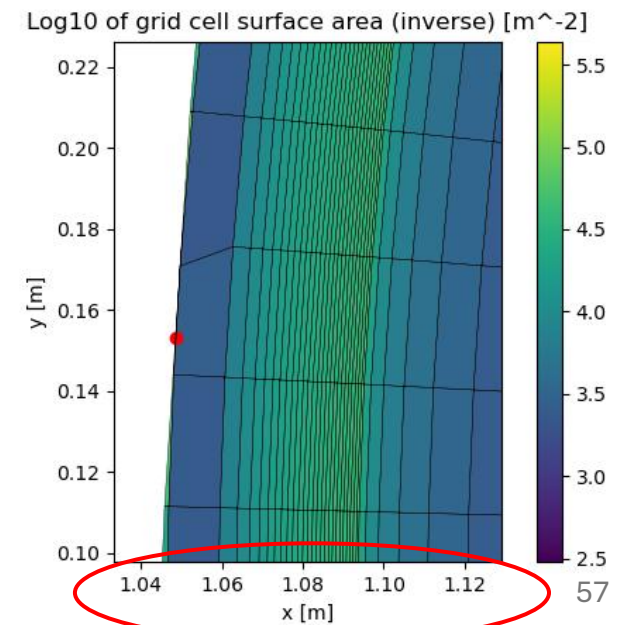
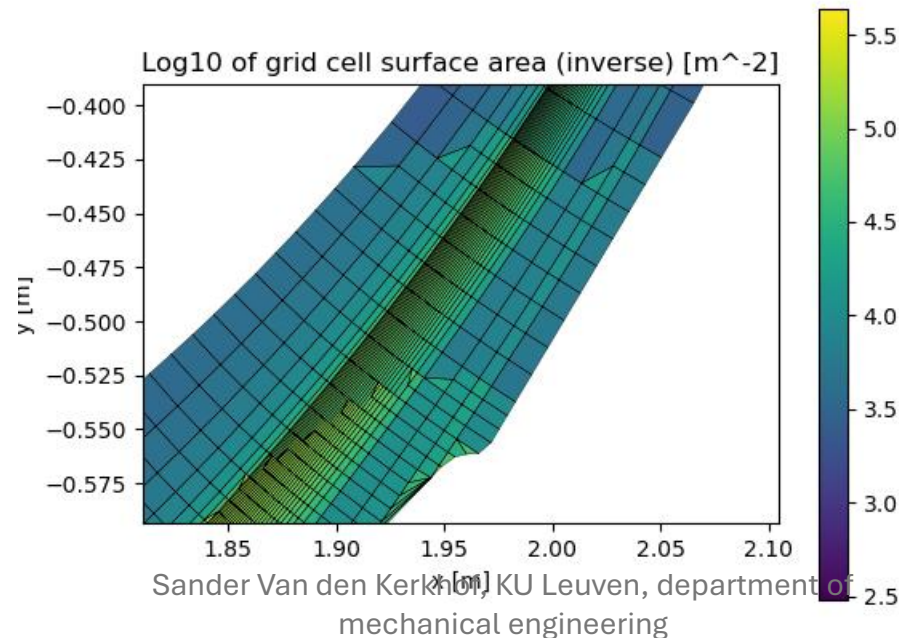


# How can we resolve this?

- Best option here is to increase the tube size such that the small ‘cavity’ is removed altogether
- Let’s increase ‘lradmintubes’ to 2mm and see...

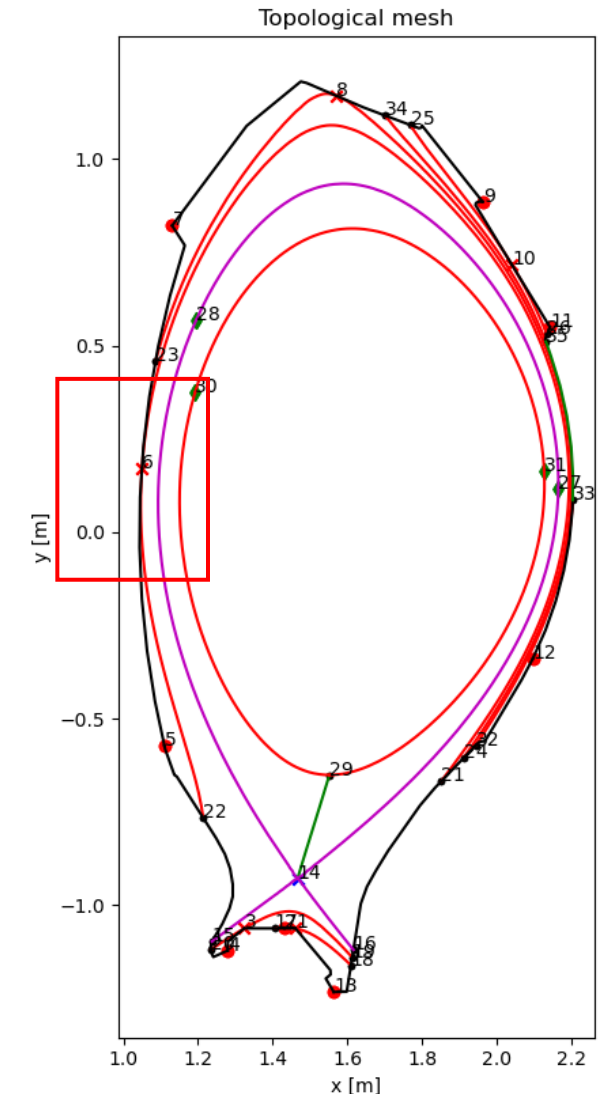
```
'gg.tm.mergetubes'      '1'  
'gg.tm.mergetubes.meth' 'extensive'  
'gg.tm.dpsimintubes'    '0.0'  
'gg.tm.lradmintubes'    '2.0e-3'  
'gg.tm.mergetubes.allowseparatrix' '0'
```

We now get one big ‘cut-cell’ like tube, so seems OK. However, smallest cell is now shifted to the HFS



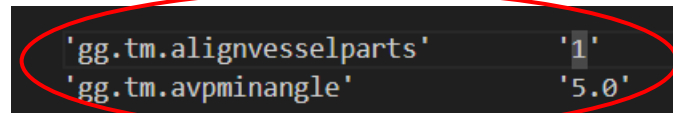
# Why do we see this?

- Observation 1: it happens near a tangency point, in combination with a nearly aligned vessel wall (see topomesh)
- GG mechanic 1: nearly aligned vessel wall parts have been extended, which is why we see the cut-cell like cells
- GG mechanic 2: because of near alignment (and precise alignment in the tangency point), cell widths/angles may be arbitrarily small



# How can we resolve this?

- Two options:
  - Merge away those tubes by increasing tube radial width in merge adaptation settings (may lead to excessive loss of radial resolution here)
  - Insert an aligned vessel parts near tangency points to ensure (if possible) a minimal angle between vessel and flux surface – we'll show this approach here
- Options of interest:

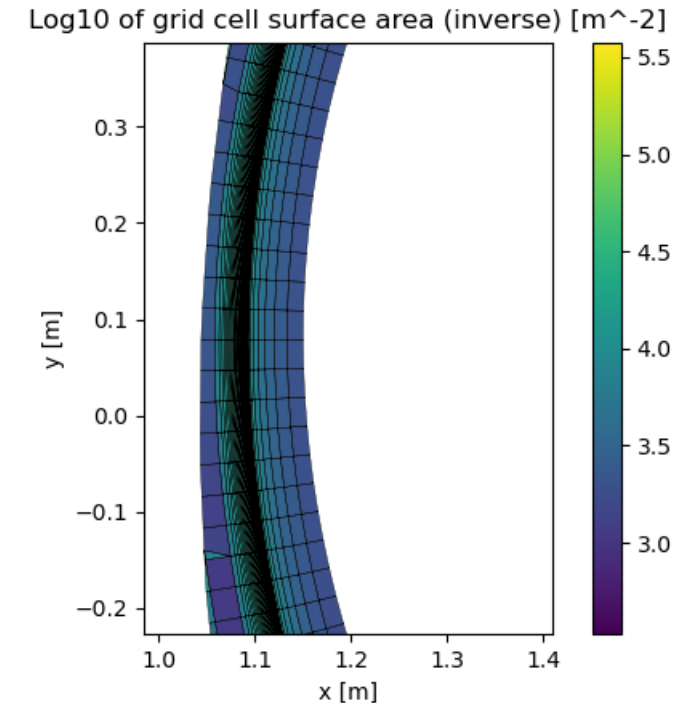
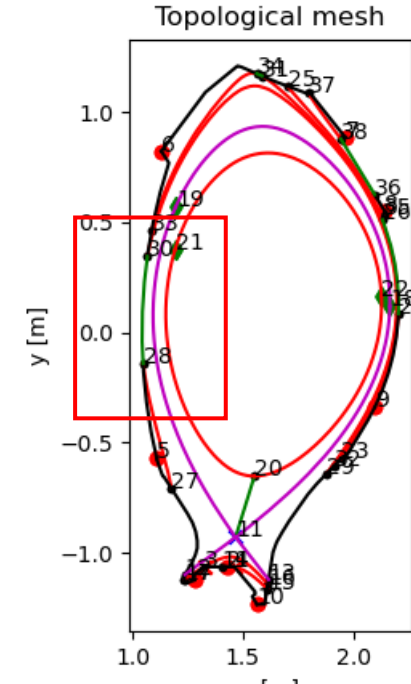


```
'gg.tm.alignvesselparts'      '1'  
'gg.tm.avpminangle'         '5.0'
```

Note: angle is in degrees

# Dealing with small cells near tangency points

- Observation 1: tangency point has been removed there
- Observation 2: cut-cell like approach is taken



# Take-aways from this part

- **Key takeaway 1:** bad grid features may be difficult to spot since they are typically small
- **Key takeaway 2:** the badness often comes from the topological mesh/vessel geometry:
  - **Nearly-aligned walls** are a pain, especially if they lead to small ‘cavities’ (e.g. LFS of ASDEX case seen before)
  - **Simplify the topological mesh** is almost always the best action to take

# Next up: local refinement

- Boundary layers (poloidal, radial)
- Local grid refinement (poloidal, somewhat radial)
- Note: for the remainder of this section, we don't adjust the topomesh (because only grid-based adaptations now):

```
# Read existing file?
'gg.tm.readexistingTM'          '1'
'gg.tm.readexistingtracers'    '1'
'gg.tm.writedebugoutput'       '0'
'gg.tm.TMfilepath'             './output/topomesh.dat'
'gg.tm.TMfieldtracerfilepath'  './output/TMfieldtracer.dat'
'gg.tm.TMvesseltracerfilepath' './output/TMvesseltracer.dat'
```

```
# Topomesh adaptations
'gg.tm.doadaptations'          '0'
'gg.tm.addcoreboundaries'      '1'
'gg.tm.addPFboundaries'        '0'
'gg.tm.cbnd.frac'              '0.1'
'gg.tm.PFbnd.frac'             '0.1'
'gg.tm.removecoreregions'      '1'
'gg.tm.removewidegridregions'  '0'
```

# Applying poloidal boundary layers

- In poloidal direction, boundary layers can be specified on:
  - Targets: in the topological sense – any vessel segment adjacent to a strike point (not recommended for most use cases)
  - Entire vessel (very much not recommended for most use cases)
  - Structure based → in this tutorial
- Relevant options (may not be in options file yet):

```
# Boundary layer options (only for length based refinement for now)
'gg.ref.BL.dostructure'      '1'
'gg.ref.BL.structureID'      '[]'
'gg.ref.BL.ncstructure'      '4'
'gg.ref.BL.dlstructure'      '[0.002, 0.004, 0.008, 0.016]'
```

Main switch

Structure IDs to consider

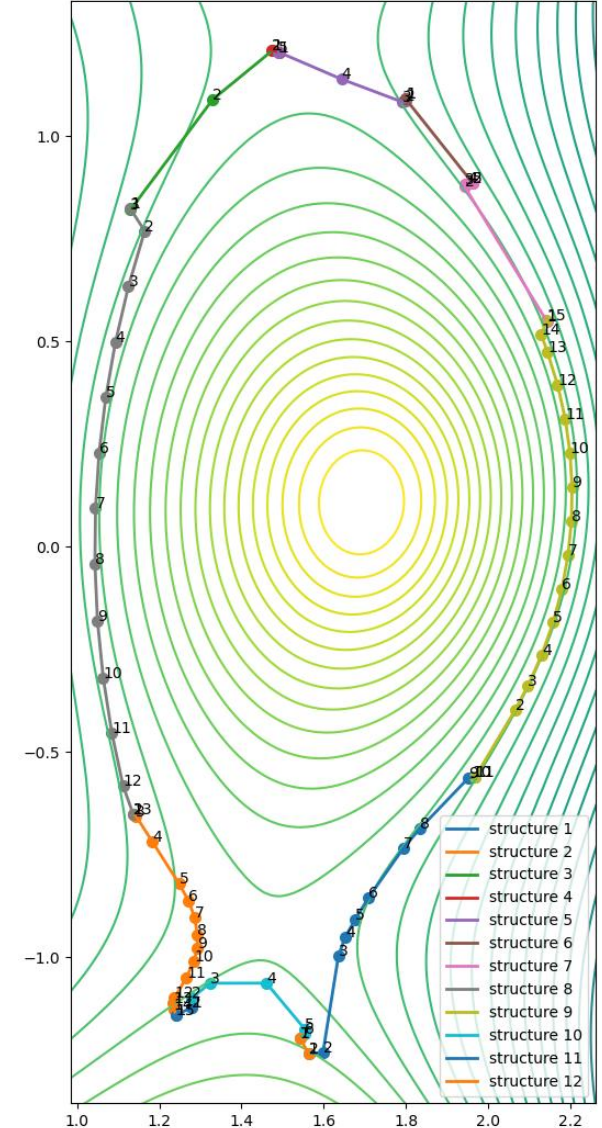
Number of cells of the boundary layer

Boundary layer cell widths (starting from vessel, in [m])

# Applying poloidal boundary layers

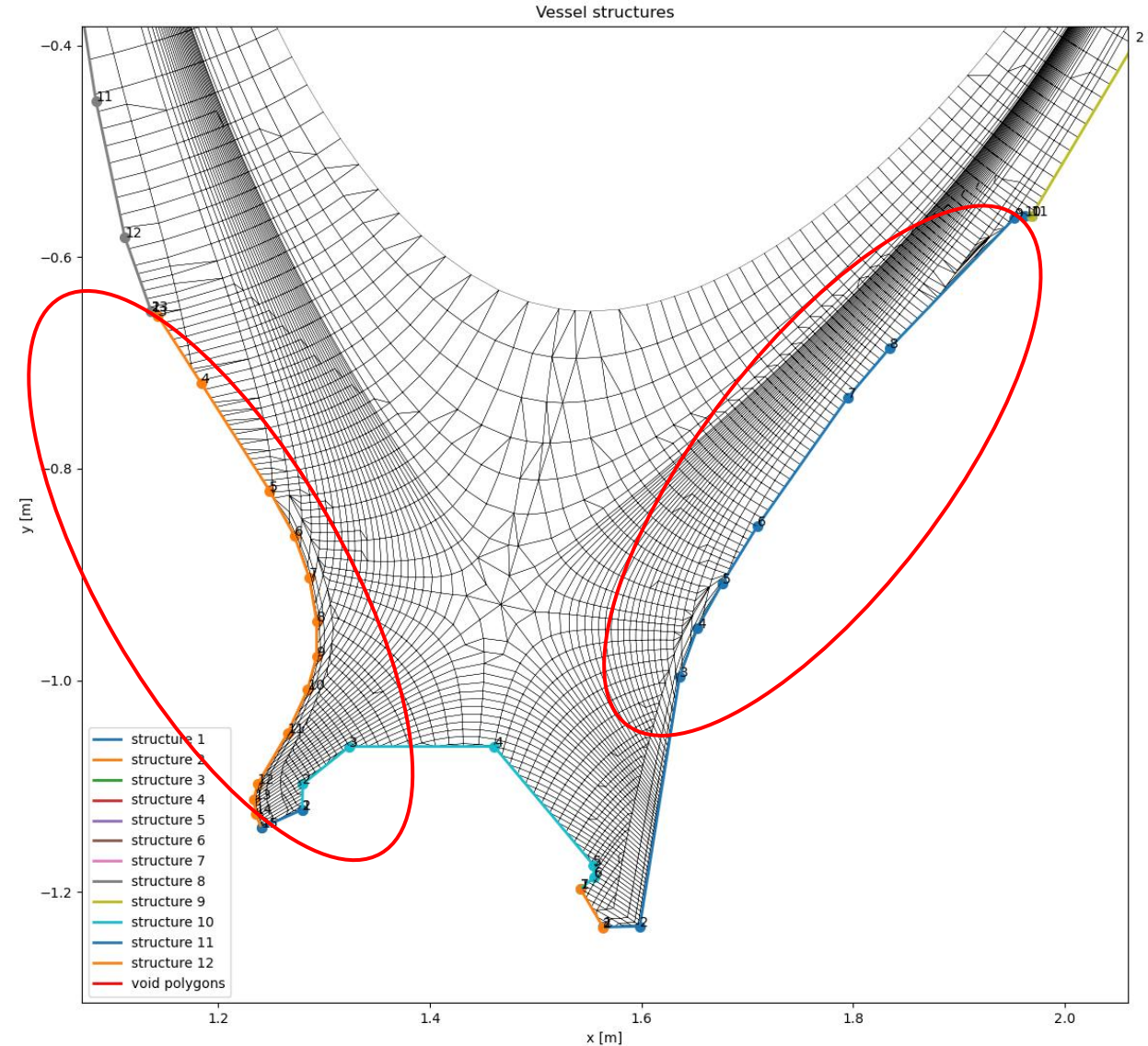
- Use 'pgginput' to see which structures you have
- Here: probably refine near structures 1 and 2.

```
# Boundary layer options (only for length based refinement for now)
'gg.ref.BL.dostructure'      '1'
'gg.ref.BL.structureID'     '[1, 2]'
'gg.ref.BL.ncstructure'     '4'
'gg.ref.BL.dlstructure'     '[0.002, 0.004, 0.008, 0.016]'
```



# Applying poloidal boundary layers

- We get a BL *almost* everywhere along the target – why?
- Because these tubes are extended (recall the vessel part extension!) and this overrides the boundary layer option.
- Why? Because this would very likely result in bad cells (see e.g. structure 1...)



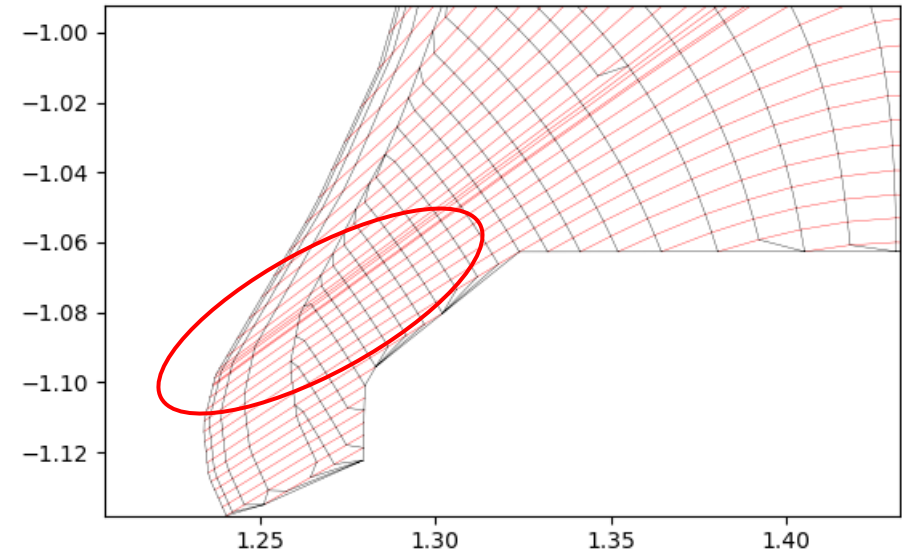
# Applying radial boundary layers

- Can only be specified near strike points (always both sides)
- Not really recommended since often bad interaction with local radial refinement (see later), but sometimes useful
- Relevant options:

```
# 'Boundary' layer options for radial direction
'gg.radref.BL.dosp'          '1'
'gg.radref.BL.ncsp'          '2'
'gg.radref.BL.dlsp'          '[0.001, 0.002]'
'gg.radref.BL.dlengthbased'  '1'
```

# Applying radial boundary layers

- Observation 1: boundary layer is imposed at one of the two targets
- Observation 2: there is, however, a 'random' smaller tube next to it → because boundary layer 'overwrites' original distribution and does not affect the rest → hence not really recommended

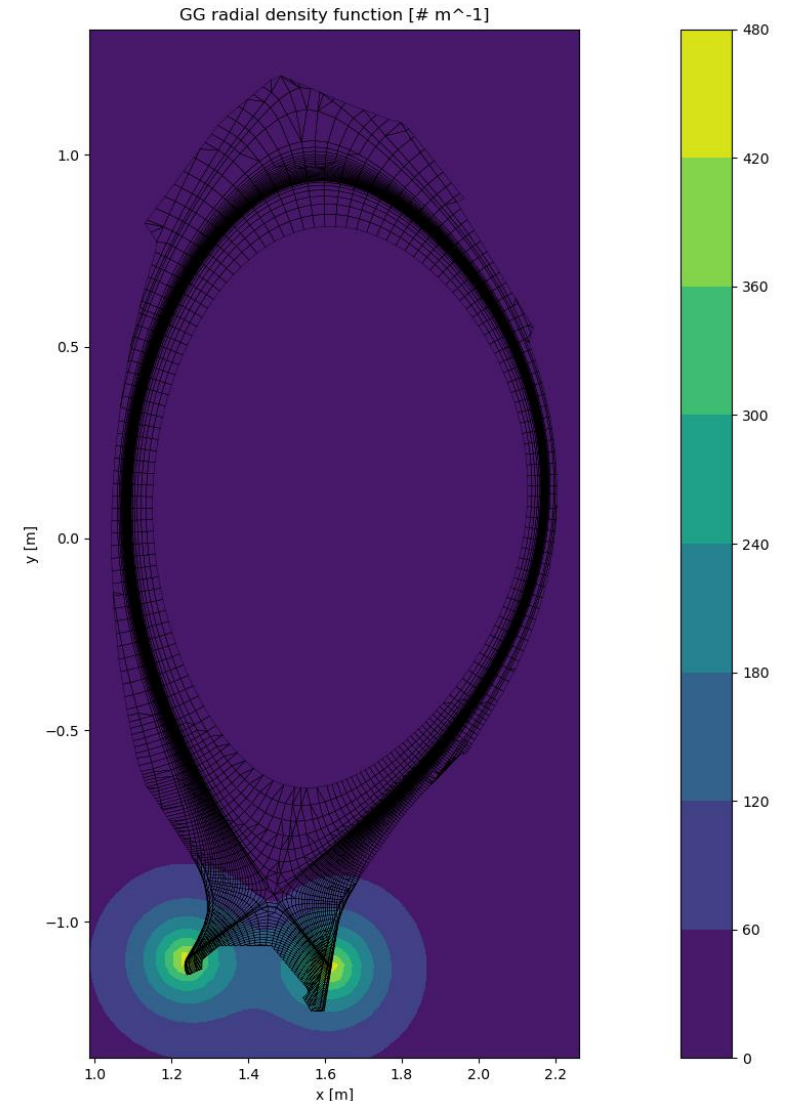


# Takeaways from this part

- Key takeaway 1: poloidal boundary layers work in a rather straightforward way, keep in mind that too long vessel segments are often dealt with in a cut-cell approach
- Key takeaway 2: radial boundary layers are possible but may give you surprising results in its current state

# Local radial refinement

- Let's disable the radial boundary layer and look at the other option: specify a distribution function (in [#surfaces/meter radial length])
- Will yield smooth distribution, somewhat more difficult to impose



# Local radial refinement

```
# Distribution of flux surfaces (radial)
'gg.vd.rd.type'                'densitybased'
'gg.vd.rd.distribution.dosp'    '1|'
'gg.vd.rd.distribution.decaylengthsp' '0.1'
'gg.vd.rd.distribution.densityatsp' '500'
'gg.vd.rd.distribution.densityatinf' '20'
#'gg.vd.rd.distribution.points.x'    '[4.7]'
#'gg.vd.rd.distribution.points.y'    '[4.55]'
#'gg.vd.rd.distribution.points.d'    '[1.0]'
#'gg.vd.rd.distribution.points.val'  '[2]'
```

Always do 'density based' (other is 'uniform')

Switch to refine near strike points

Exponential settings: decay length (at SP), density (at SP), density at infinity

Exponential settings for own defined points  
(coordinates, decay length, density value at points) –  
commented out here

# Local radial refinement: caveats

- **Beware of the flux expansion effect:**
  - Flux expansion near X-point(s) may lead to excessive resolution in core → specify smaller density at X-points
  - X-points just outside of vessel may also lead to similar issues, but then in other random tubes. This point has to be added 'manually' because the X-point is not present in the topomesh. See e.g. ITER case...
- It is not always clear which radial face is taken for the initial distribution sadly enough → specify distribution at both sides of a tube
- All of this would be solved if true local radial refinement (i.e. flux tube splitting) would be allowed by the solver...

# Local poloidal refinement

- In principle also possible based on distribution function but typically bad results:
  - Incompatible with orthogonal way of gridding (bad transitions near topomesh tube boundaries)
  - Often extremely non-orthogonal cells
- Recommended approach: take initial, coarse distribution on faces, construct orthogonal faces, and refine/coarsen where length is too big/too small. This is again defined by distribution functions...

# Local poloidal refinement: initial distribution options

```
# Distribution of vertices on flux surfaces (poloidal)
'gg.vd.pd.type'                'uniform'

# Options for uniform distribution
'gg.vd.pd.distribution.facelength'  '16e-1';
```

Uniform distribution with (ridiculously) large initial face length (here 1,6 m)

# Local poloidal refinement: refiner options

```
# Refinement options
'gg.ref.meth'                'lengthbased'
#'gg.ref.meth'                'no'
'gg.ref.LB.doxp'              '1'
'gg.ref.LB.dovessel'          '1'
'gg.ref.LB.Lmininf'           '0.01'
'gg.ref.LB.Lmaxinf'           '0.05'
'gg.ref.LB.structureIDs'      '[1, 2]'
'gg.ref.LB.Lminstructure'      '[0.002, 0.002]'
'gg.ref.LB.Lmaxstructure'      '[0.01, 0.01]'
'gg.ref.LB.decaylengthstructure' '[0.2, 0.2]'
'gg.ref.LB.Lminxp'            '0.01'
'gg.ref.LB.Lmaxxp'            '0.05'
'gg.ref.LB.decaylengthxp'      '0.2'
```

Switches for refinement near X-point(s) and vessel structures (specify using structureIDs)

Lmin: minimal length, if smaller → face is merged

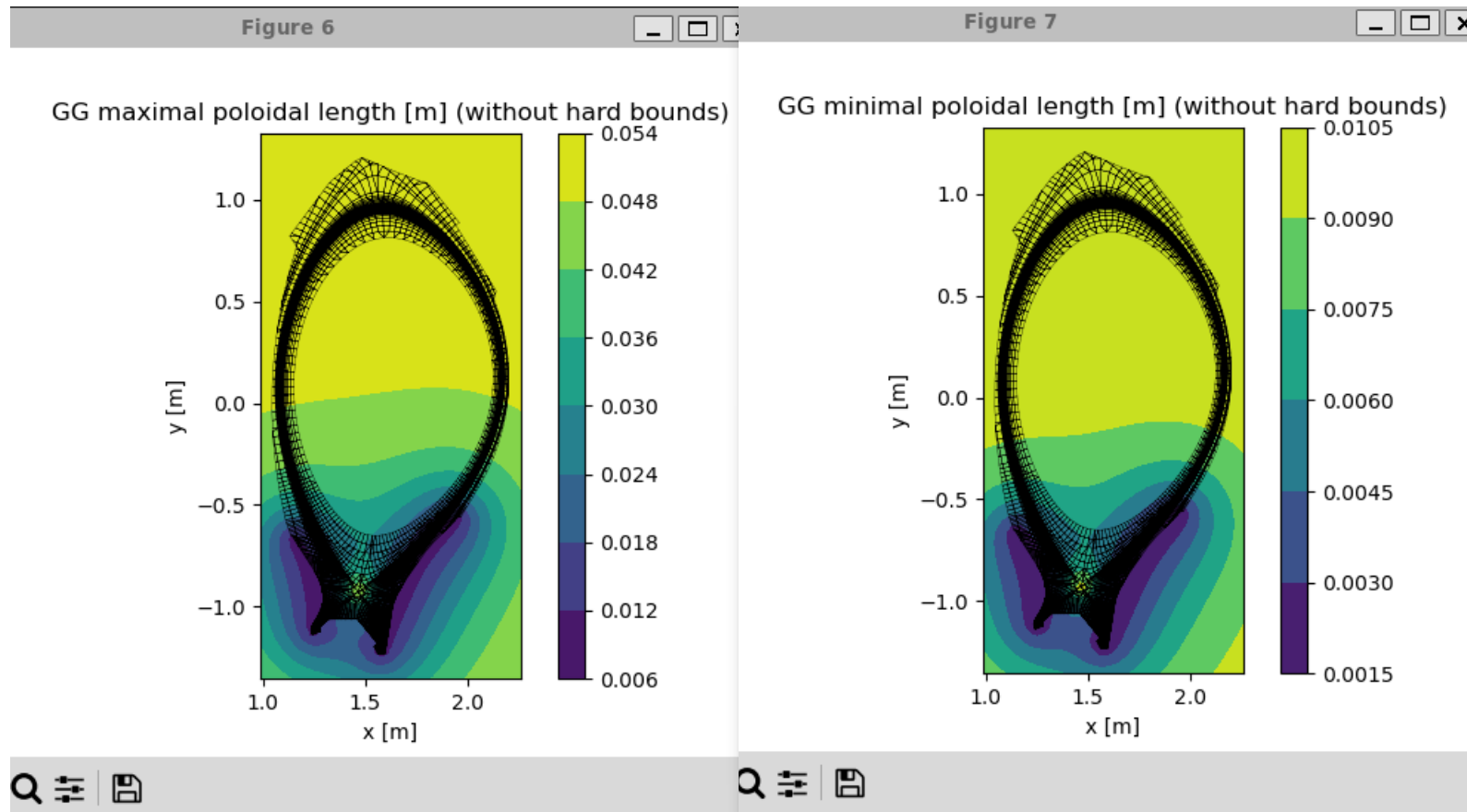
Lmax: maximal length, if larger → face is split

Both are distribution functions with exponentials (analogous to radial refinement)

To refine/coarsen: decrease/increase Lmin, Lmax

# Local poloidal refinement: refiner options

- After running GG, visualize distributions with 'pggoutput'



# Hands-on session 2

Running a SOLPS-ITER case (B2,5 and Eirene) with the newer traduit format (version number to be determined) using the new experimental workflow

# Main ideas

- Go through similar route as CARRE in DivGeo but *without* checks on variables (many are not needed anymore)
- Run goat to get the plasma grid and void polygon for triangulation
- Run Uinp similar to previous workflow
- Run b2mn

# Preliminaries

- Functionality available through experimental branches on github:
  - SOLPS: feature/svdk/wg-workflow
    - DivGeo: feature/svdk/wg-workflow
    - Eirene: feature/svdk/goat-integration
    - B25: goat-integration-clean-rebased
  - Goat: feature/goat-uinp-integration
- No guarantees, but may at least speed up case construction
- Here: only the base needed to get something technically working, case not tested for actual convergence!

# Preliminaries

- Currently, several new file and file format updates are needed (automatically handled by the code):
  - Traduit.out.b2us contains additional topological data from goat
  - B2fgmtry copies over this data
  - Goat creates an additional fort\_goat.78 file with additional void polygon data for Uinp
  - For Eirene, an additional fort.36 file is created to deal with precision effects
- Example case sent around (ask otherwise)

# Test case: VNS-like geometry

- In the example case folder you should find the initial geometry & equilibrium to start from, including a GOAToptions.dat in the 'baserun' directory
- Copy this folder to \$SOLPSTOP/runs/VNS/testcase\_codecamp, such that there is a \$SOLPSTOP/runs/VNS/ testcase\_codecamp/baserun directory (not strictly necessary but recommended)
- Open a fresh shell and do source setup.csh in the SOLPS-ITER top directory as usual. Navigate to the 'baserun' folder – we assume to work there this session, unless stated otherwise

# DivGeo: overview

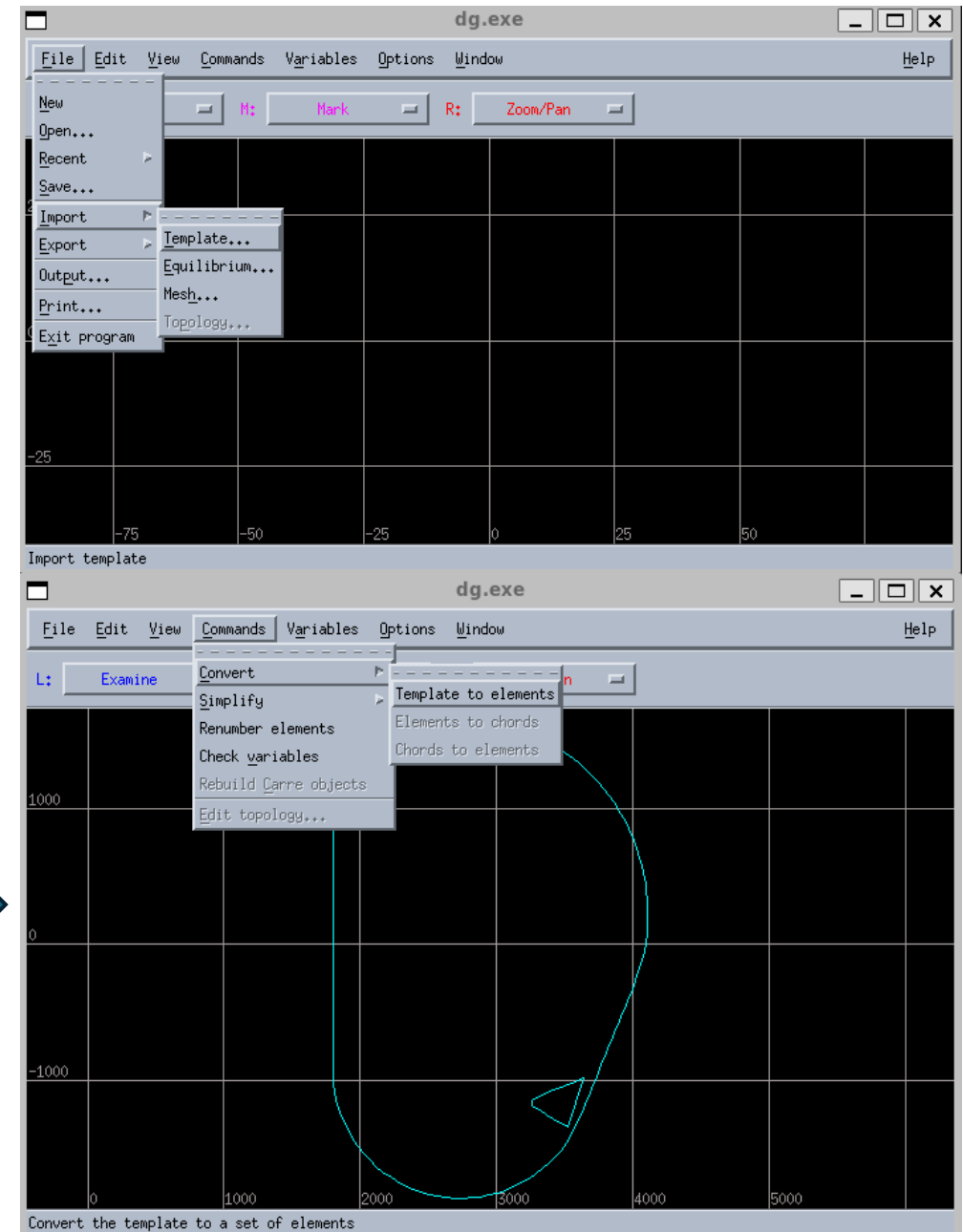
- Most work to be done is actually in DivGeo:
  - Importing the geometry (here: through template files)
  - Setting wall properties, plasma species, Eirene settings, ... → largely skipped: independent of grid generator route
  - Setting face labels (currently in carre2 setup...)
  - Marking vessel elements
  - Setting the grid generator
  - Setting the midplane
  - Setting void elements for Eirene (and elements not for Eirene)
  - Outputting files

# DivGeo: useful shortcuts & tricks

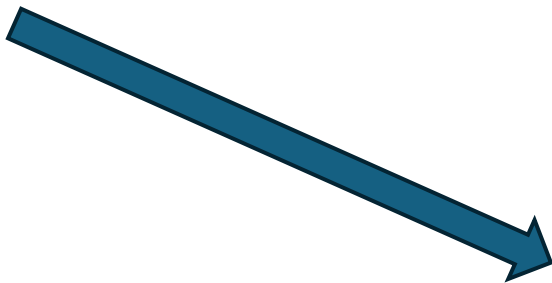
- Ctrl + U: unmark all elements
- Ctrl + mark: mark entire polygon at once
- Mark (hold) and move mouse: select elements along the polygon
- For variables: right-mouse click in a field to show values or mark elements with a certain value
- Ctrl + P: reset view

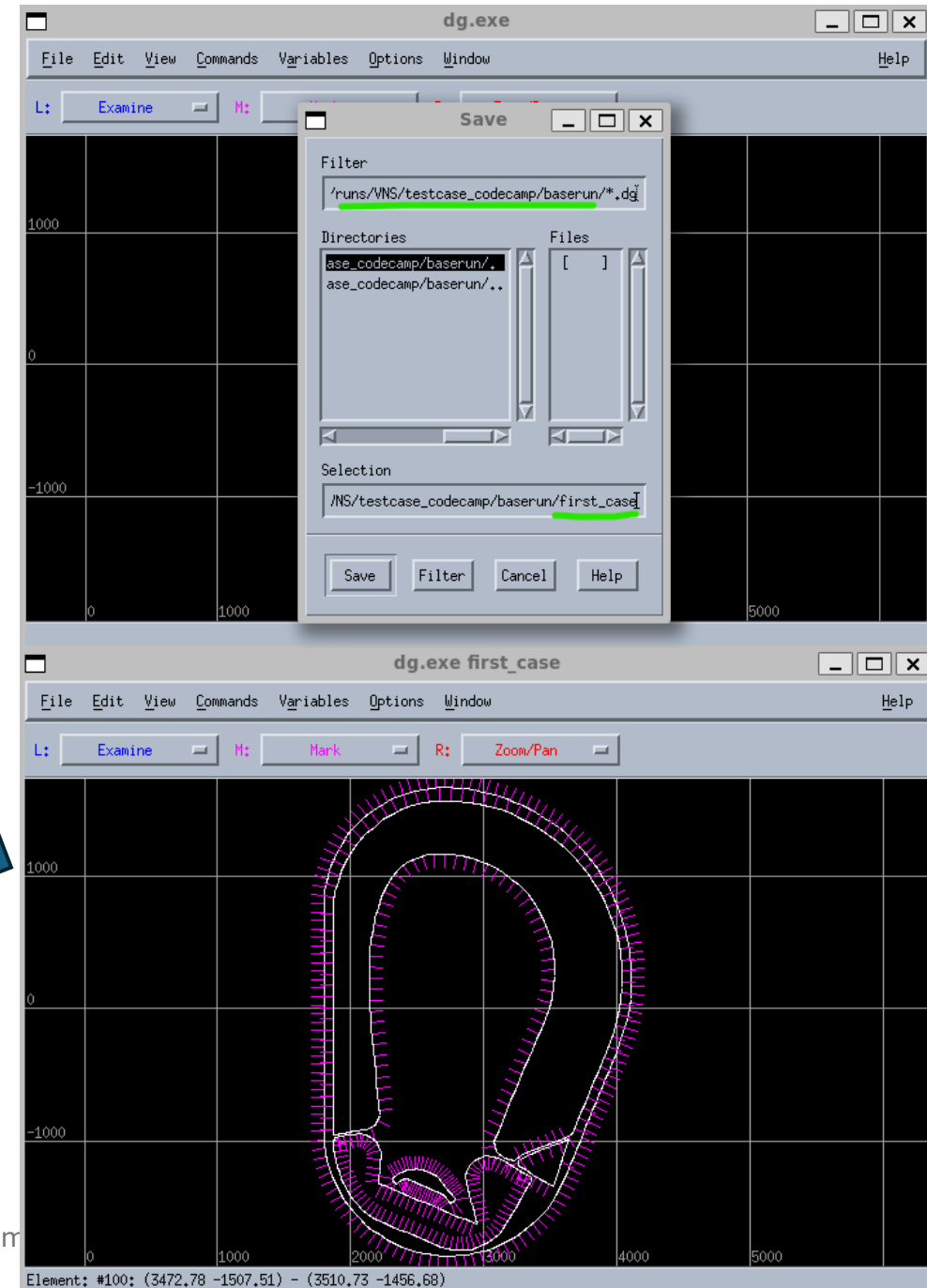
# DivGeo: geometry setup

- First, set the environment variable DEVICE by executing 'setenv DEVICE VNS' in the terminal
- Run dg (type 'dg &', enter)
- Import the 'VV\_inner\_shell.ogr template'
- Convert template to elements
- Save to 'first\_case'



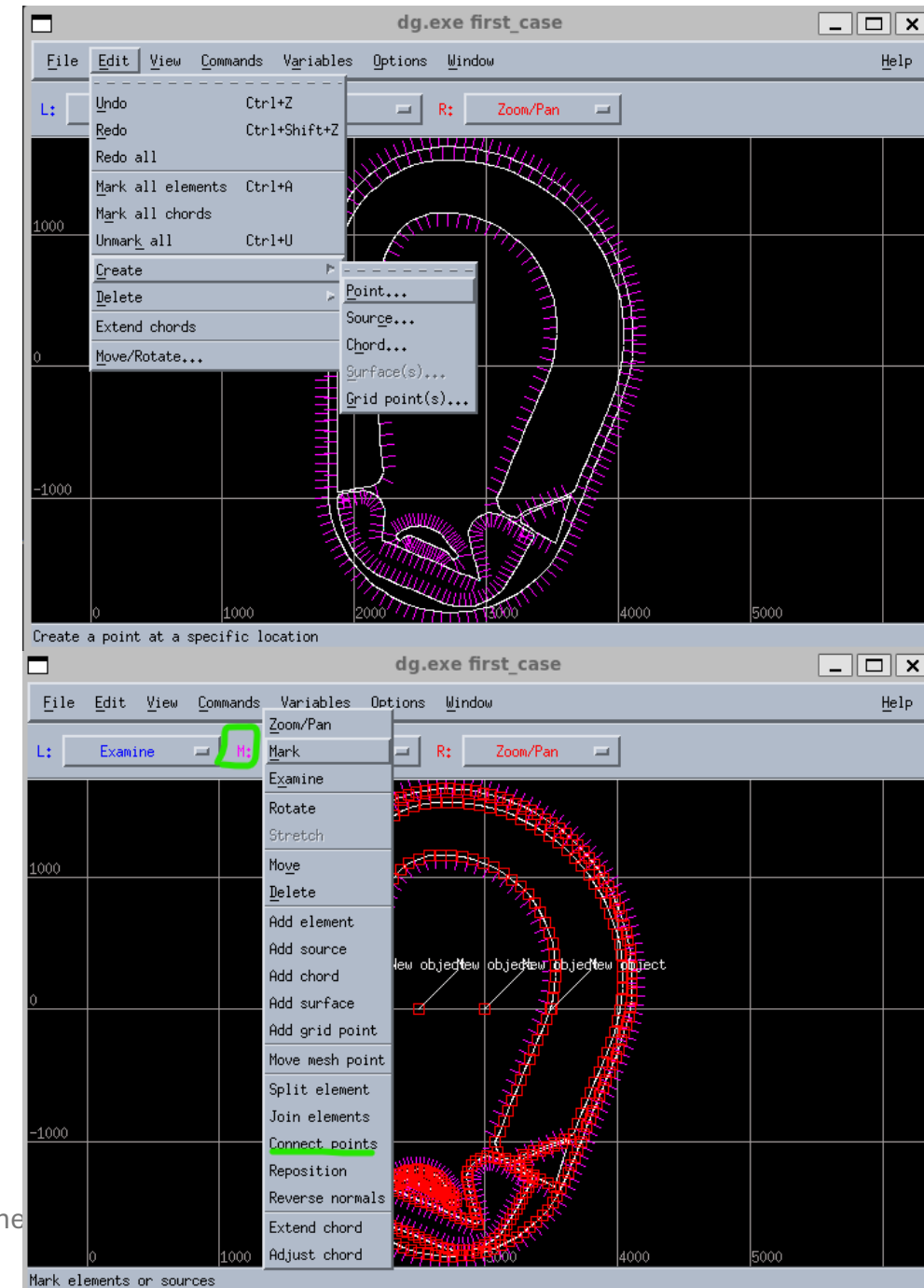
# DivGeo: geometry setup

- Save to 'first\_case' 
- Load and convert the other templates in the same way, end result should look like: 



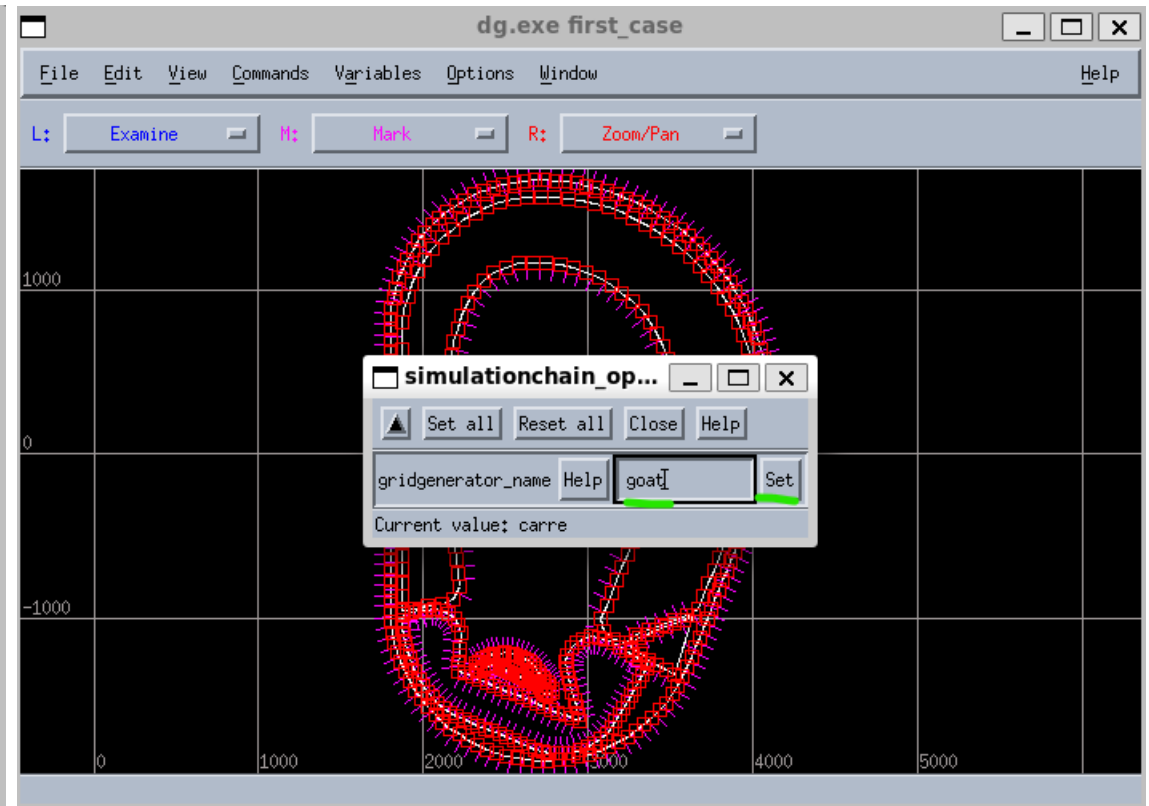
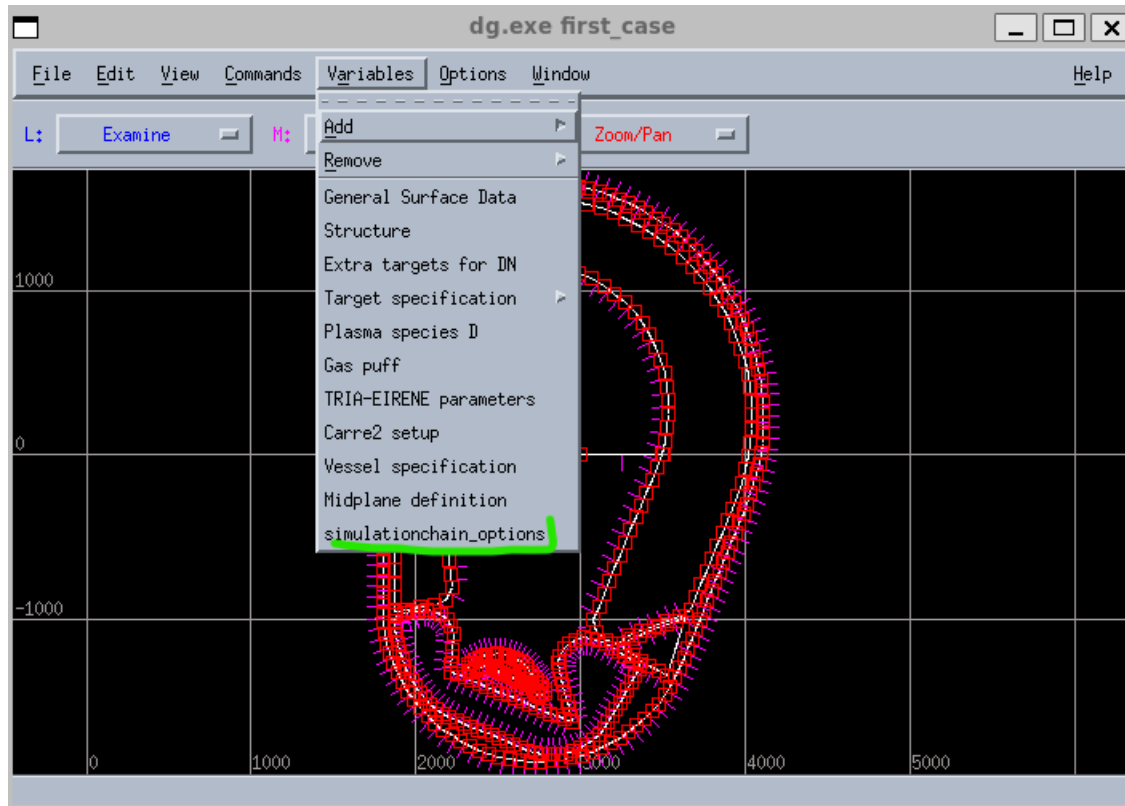
# DivGeo: geometry setup

- We also need to define the OMP/IMP. For this we generate points and connect them to obtain elements
- Here, e.g.  $x = [2000, 2500, 3000, 3500], y = 0$
- Create elements by connecting points (first two for IMP, others for OMP)



# DivGeo: setting the grid generator

- Simply to follow correct route in triang (don't forget to press 'set')



# DivGeo: setting OMP/IMP

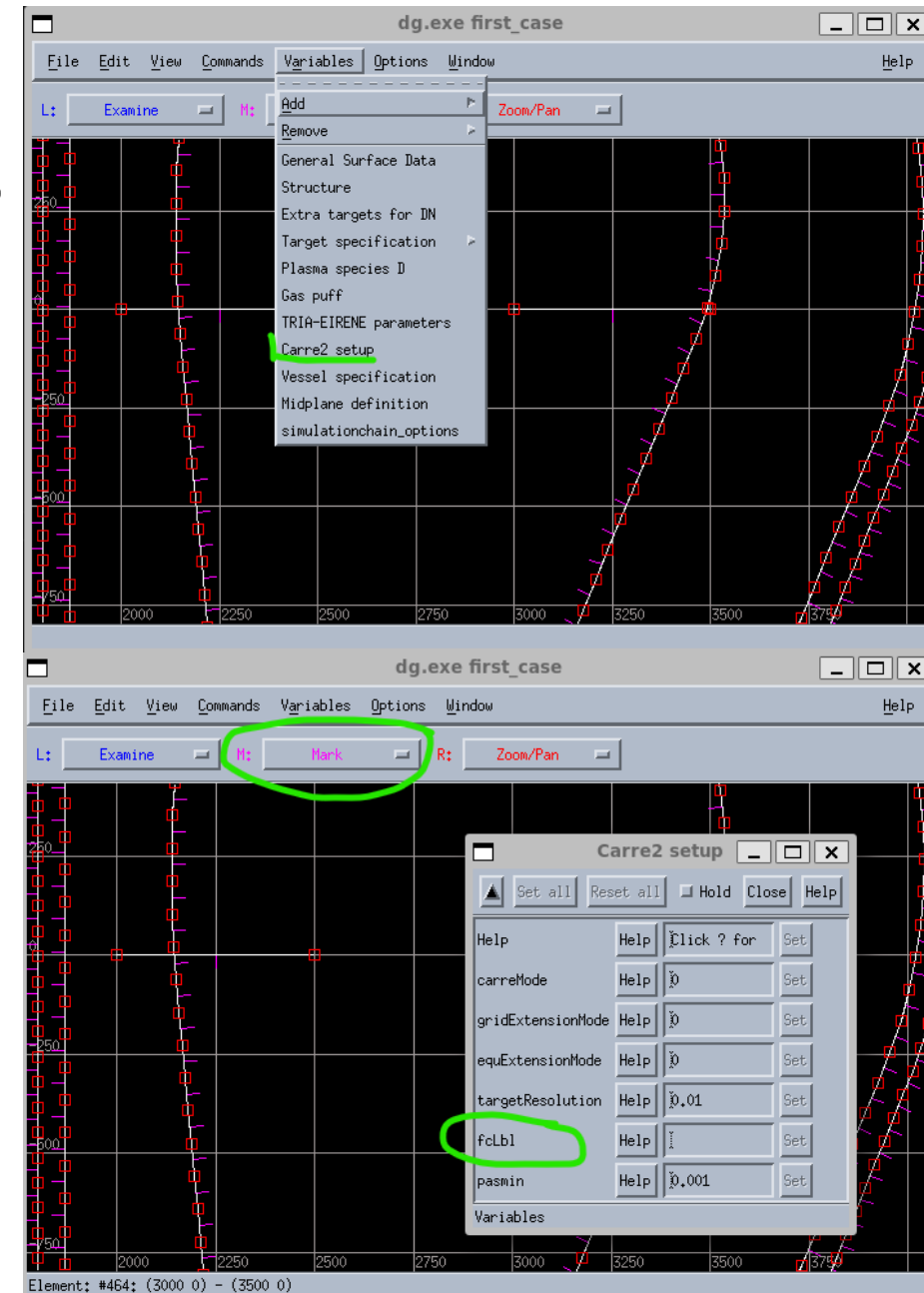
- Select 'variables > midplane definition'
- Mark element for OMP and 'set'
- Mark element for IMP and 'set'
- Check by using the 'mark' button in 'midplane definition'

# DivGeo: setting the face labels

- Choice is important:
  - Boundary conditions are imposed based on labels
  - Grid refinement (cfr. Goat options)
  - Simulated extent of vessel domain (i.e. void boundaries)
- Convention on labels (same as CARRE2 workflow):
  - Positive value: actual vessel part
  - Negative value: void interface

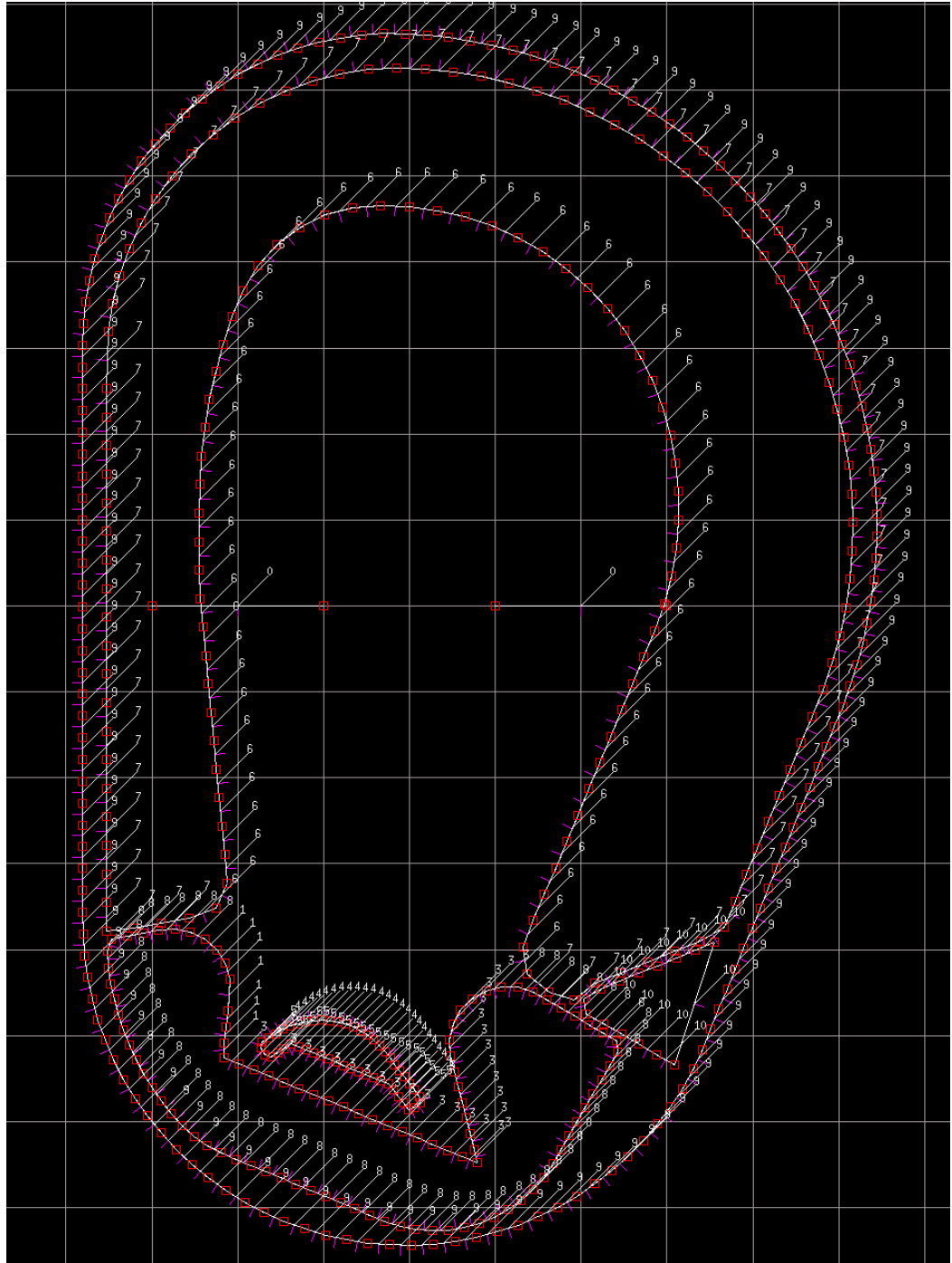
# DivGeo: setting the face labels

- Note: currently still done under Carre2\_setup variable, even for goat...
- Choice of labels free, but here we took:
  - IT/OT: 1 and 2 (for distinction & BCs)
  - Cassette under dome: 3
  - Top of dome: 4
  - Bottom of dome: 5
  - Inner part of blanket: 6
  - Outer part of blanket: 7
  - Outer cassette: 8
  - VV: 9
  - Triangle: 10



# DivGeo: setting the face labels

- Make sure to always unmark all elements before marking a new set of elements (edit > unmark all)
- First mark, then edit fcLbl, then set value
- If you right-click the value box of fcLbl, you can choose to display the values, or mark elements that have a certain value – handy to check what happened



# DivGeo: setting face labels

- Note that IMP/OMP don't get labels
- If your labels don't exactly correspond, don't worry too much. Final results may differ though.

# Tutorial cross-road

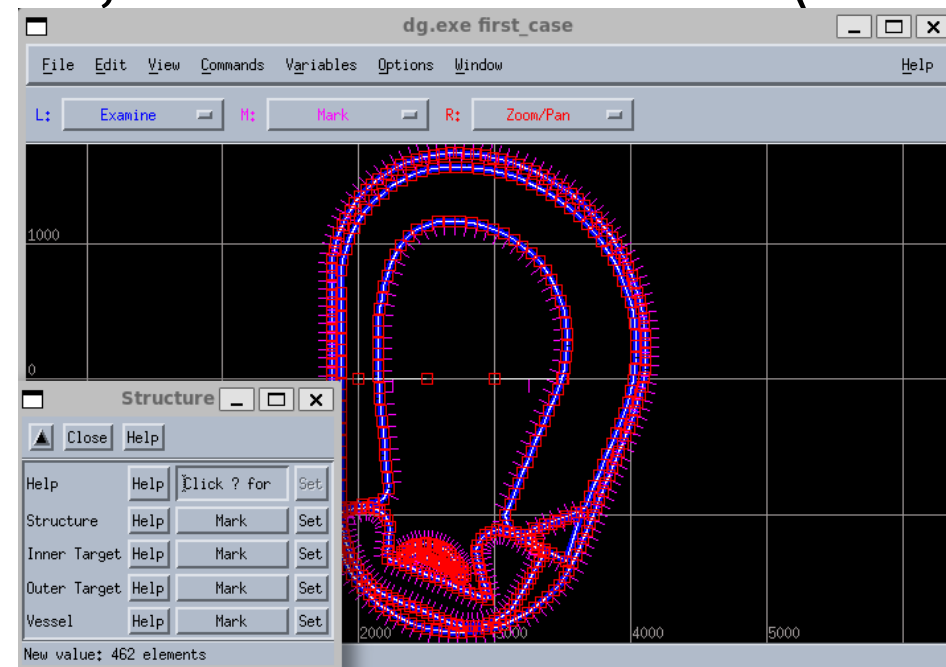
- **From here, the tutorial splits**, depending on choices made in DivGeo:
  - **Route 1:** we consider the **entire domain**, including subdivertor structures, for the plasma edge grid and deal through **goat for removing unwanted regions** → continue on next slide
  - **Route 2:** We introduce ‘**void boundaries**’ to **limit the plasma grid *a priori*** and **retain the void regions** for Eirene → continue on
- **Save this initial DivGeo setup** under ‘first\_case\_general’, so you can restart from that one to do the different routes!

# Route 1: full domain

- Start from the DivGeo 'first\_case\_general'
- The remaining DivGeo steps are:
  - Save this setup under a different name, e.g. 'first\_case\_route1'
  - Vessel elements (simply all)
  - Elements not for Eirene (IMP/OMP)
- Then, grid generation:
  - Create an initial grid
  - Trim the topomesh
  - Refine where desired
- Finally, input file generation (triang) and running the case

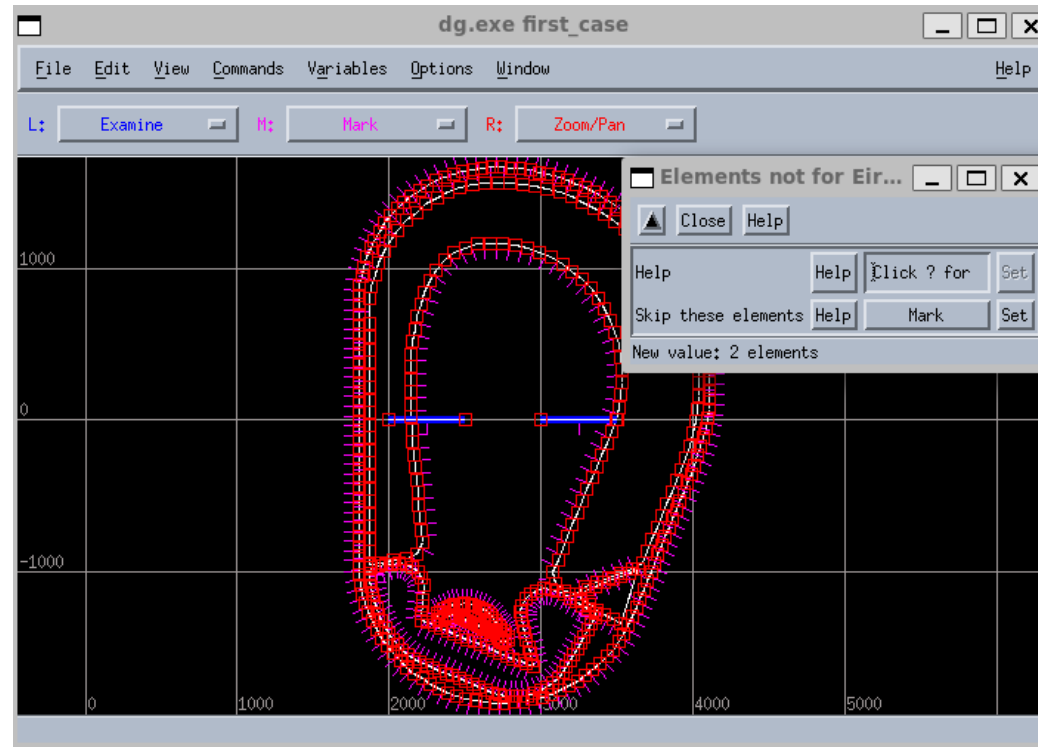
# DivGeo: vessel elements

- Mark all elements with 'edit > mark all elements', unmark OMP/IMP
- Then, under 'variables > structure', set these as 'vessel' (other fields don't matter)



# DivGeo: elements not for Eirene

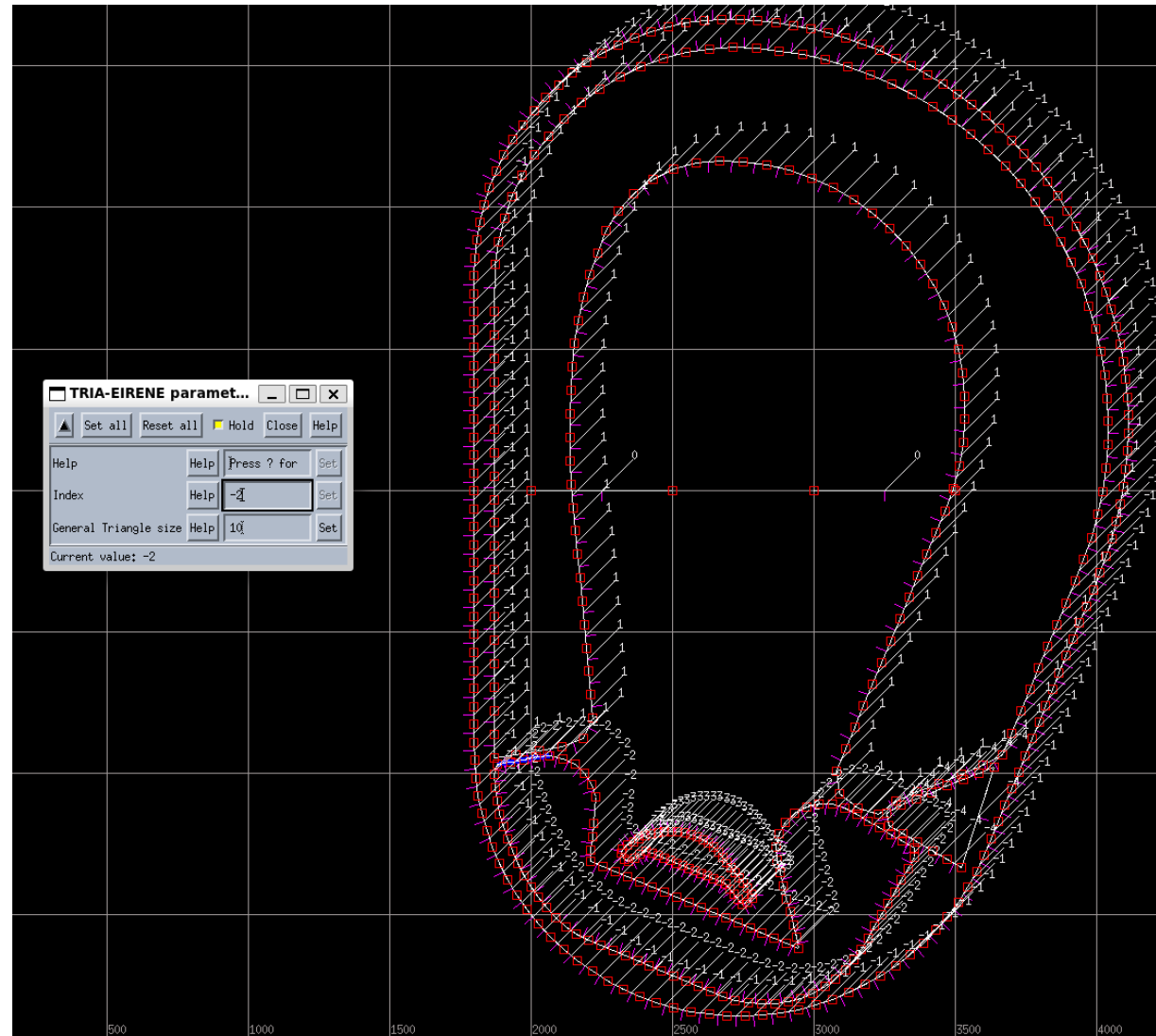
- Mark IMP/OMP elements, under 'variables > add > elements not for eirene', set these elements to skip



# DivGeo: Eirene index

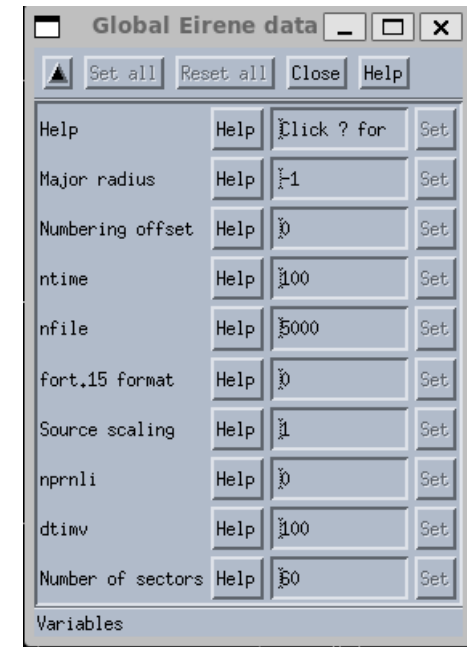
- Needs to be specified for later matching of nondefault standard surfaces (if zero, will not be matched)
- Just give each polygon an index: 'variables > TRIA-EIRENE parameters'. Then, select elements and set the 'index' variable for these
- Also useful to adjust triangle size: here, 1E+01 should suffice
- (See next slide for picture)

# DivGeo: Eirene index



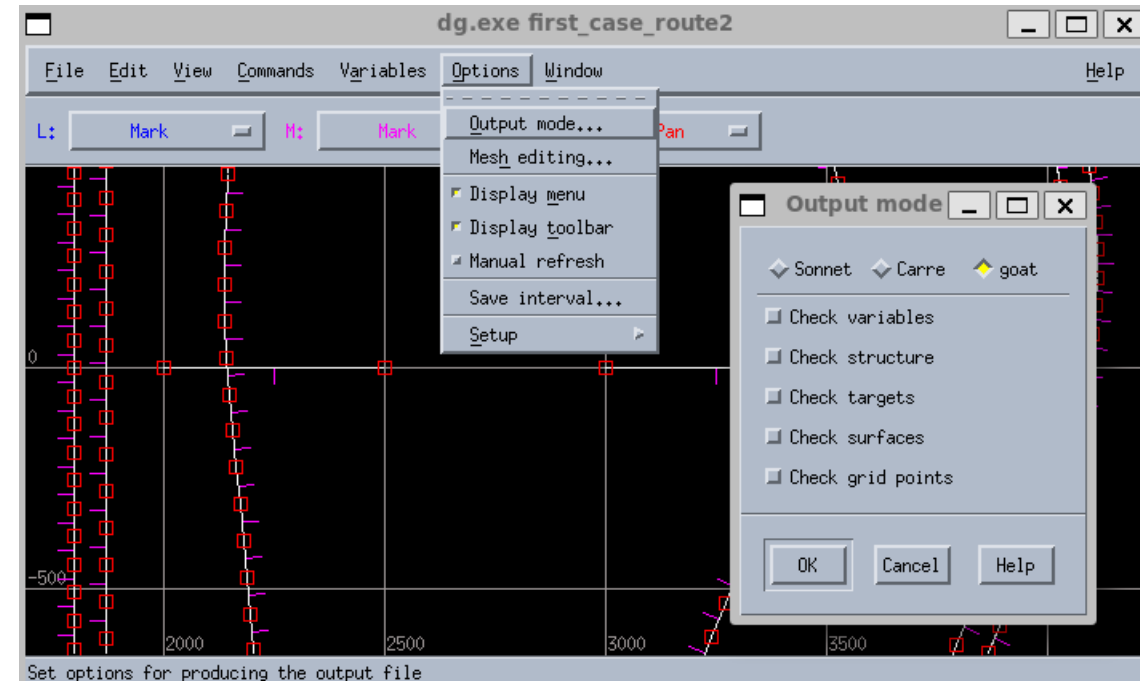
# DivGeo: additional data

- Best to also add Eirene data (e.g. major radius, should be negative apparently) – otherwise issues later on
- Other data may also be required to get useful results!  
E.g. input power etc, not treated here



# DivGeo: data output

- To get the necessary .dgo file for goat, go to 'output mode' and select 'goat' (this doesn't really matter, carre is also possible). Uncheck all checks – many of those are not necessary
- Then, do 'file > output' and you should see a message saying it wrote a set of files (only .dgo important)



# Goat: grid generation setup

- First, open a fresh tcsh terminal and navigate to your goat installation – so this is different from the shell you’ve been working in!
- In the goat top directory, do ‘source setup.csh’ as usual
- Navigate in the terminal to the directory containing the case in SOLPS-ITER (then fort.78, fort\_goat.78 and traduit.out.b2us are immediately in the correct folder – alternatively, copy these files from the directory you use in goat to the one of SOLPS)

# Goat: GOAOptions.dat

- Ensure correct MF filepath and point structure to .dgo file
- Set read method to 'equ' for MF, and 'read\_dg' for structure
- Rbtor value will automatically be computed using data in .equ

```
7  #-----#
8  #                               MAIN OPTIONS                               #
9  #-----#
10 # General options
11 'goat.debug'                '0'
12 'goat.meth'                 'GG'
13
14 # Input file locations
15 '#goat.gridfilepath'        'traduit.out.b2us_smoothed2'
16 '#goat.gridfilepath'        'traduit.out.b2us_basegeom'
17 'goat.gridfilepath'         'traduit.out.b2us_gg'
18 'goat.magneticfieldfilepath' 'Equil_VNS_07_2026_39d8s_opt_COCOS11.equ'
19 'goat.structurefilepath'     'first_case_route1.dgo'
20 'goat.GDinputfilepath'       'GOAOptions.dat'
21
22 # Output file locations
23 'goat.writefilepath'         'traduit.out.b2us'
24
```

```
88 #-----#
89 #                               MAGNETIC FIELD OPTIONS                               #
90 #-----#
91 # Read method
92 'goat.mf.readmeth'          'equ'
93
```

```
102 #-----#
103 #                               VESSEL STRUCTURE OPTIONS                               #
104 #-----#
105 # Environment (including vessel etc) options
106 # Vessel geometry & reading method
107 'goat.vessel.readmeth'      'read_dg'
```

# Goat: GOAToptions.dat

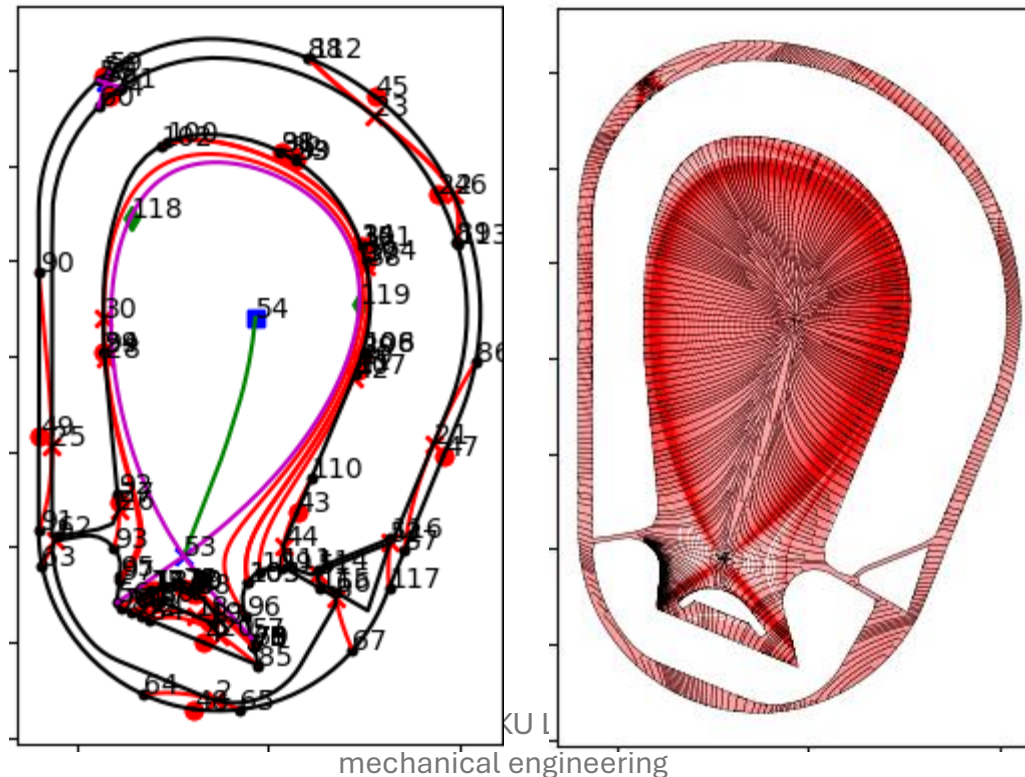
- Ensure 'gg.tm.readexistingTM' and 'gg.tm.readexistingtracers' are set to 0
- Let's leave off the adaptations for now and see what gives: run 'goat.exe' (or 'goat\_debug.exe' if you only compiled debug version)
- This may be slow – try to run with some OpenMP threads

```
444 #-----
445 #                               TOPOLOGICAL MESH OPTIONS
446 #-----
447
448 # Read existing file?
449 'gg.tm.readexistingTM'          '0'
450 'gg.tm.readexistingtracers'    '0'
451 'gg.tm.writedebugoutput'      '0'
452 'gg.tm.TMfilepath'            './output/topomesh.dat'
453 'gg.tm.TMfieldtracerfilepath' './output/TMfieldtracer.dat'
454 'gg.tm.TMvesseltracerfilepath' './output/TMvesseltracer.dat'
455
```

```
474 # Core boundary and region treatment
475 'gg.tm.doadaptations'          '0'
476 'gg.tm.addcoreboundaries'      '1'
477 'gg.tm.addPFboundaries'        '0'
478 'gg.tm.cbnd.frac'              '0.1'
479 'gg.tm.PFbnd.frac'             '0.1'
480 'gg.tm.removecoreregions'      '1'
481 'gg.tm.removewidegridregions'  '0'
482
```

# First try

- To plot output, execute 'pggoutput' in the terminal (if you also want the vessel structure to show: 'pggoutput traduit.out.b2us topomesh.dat vessel\_dgo.dat')



# First try

- Not bad, not great:
  - Likely no plasma in many 'remote' regions
  - No need to mesh the core
  - Rather small cells due to small flux tubes
- Adaptations:
  - Usual: core removal, tube merging
  - Unusual: region removal based on structures

# Basic adaptations

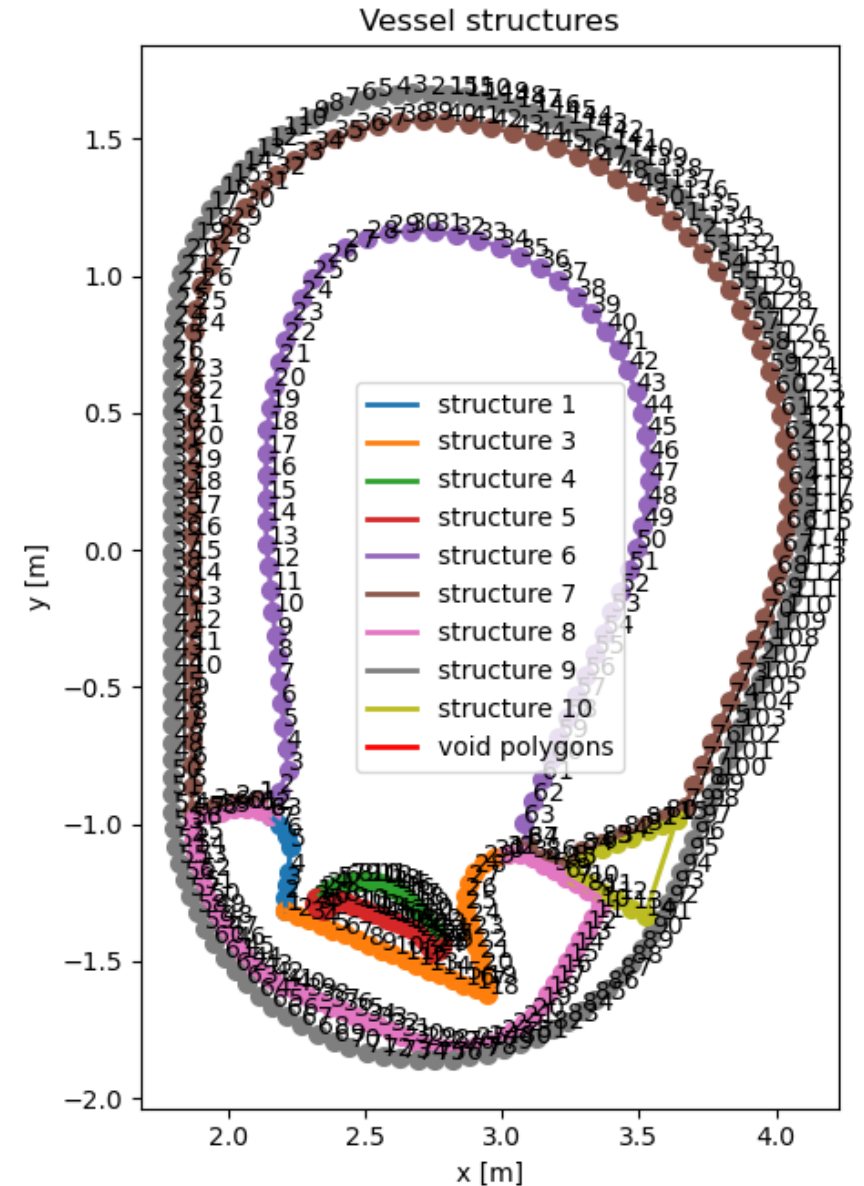
- Don't restart from scratch but from topomesh\_base.dat (base topological mesh)
- Remove core
- Merge tubes

```
448 # Read existing file?
449 'gg.tm.readexistingTM'      '1'
450 'gg.tm.readexistingtracers' '1'
451 'gg.tm.writedebugoutput'   '0'
452 'gg.tm.TMfilepath'        './output/topomesh_base.dat'
453 'gg.tm.TMfieldtracerfilepath' './output/TMfieldtracer.dat'
454 'gg.tm.TMvesseltracerfilepath' './output/TMvesseltracer.dat'
455
```

```
474 # Core boundary and region treatment
475 'gg.tm.doadaptations'      '1'
476 'gg.tm.addcoreboundaries'  '1'
477 'gg.tm.addPFboundaries'    '0'
478 'gg.tm.cbnd.frac'         '0.1'
479 'gg.tm.PFbnd.frac'        '0.1'
480 'gg.tm.removecoreregions'  '1'
481 'gg.tm.removewidegridregions' '0'
482
483 'gg.tm.mergetubes'         '1'
484 'gg.tm.dpsimintubes'       '0.0'
485 'gg.tm.lradmintubes'       '1.0e-3'
```

# Structure-based region remo

- Basically need to remove any regions that are adjacent to structures 7-10



# Structure-based region removal

Do removal? (yes)

Keep these tubes instead of removing them, and remove others? (no)

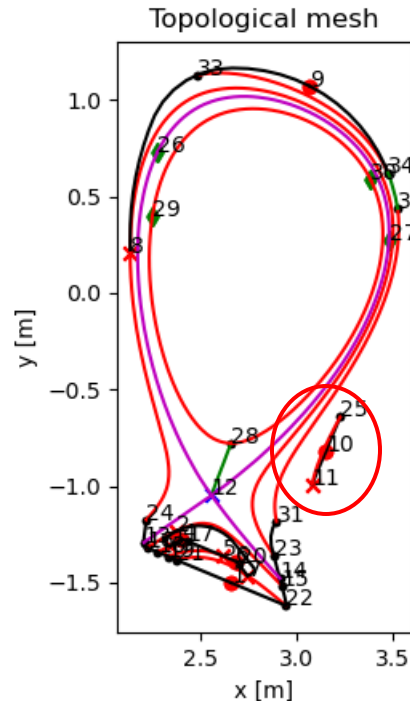
Which structure labels? (7, 8, 9, 10)

Should all downstream/upstream (in terms of  $\Psi$  values) regions be removed? (no)

Should the entire region radial boundaries be covered by specified vessel structures to be removed? (no)

```
491 'gg.tm.removevesselregions' '1'
492 'gg.tm.rvrretain' '0'
493 'gg.tm.rvrvesselIDs' '[7:10]'
494 'gg.tm.rvrdocascade' '0'
495 'gg.tm.rvrcascadedir' 'up'
496 'gg.tm.rvrfullycovered' '0'
497
```

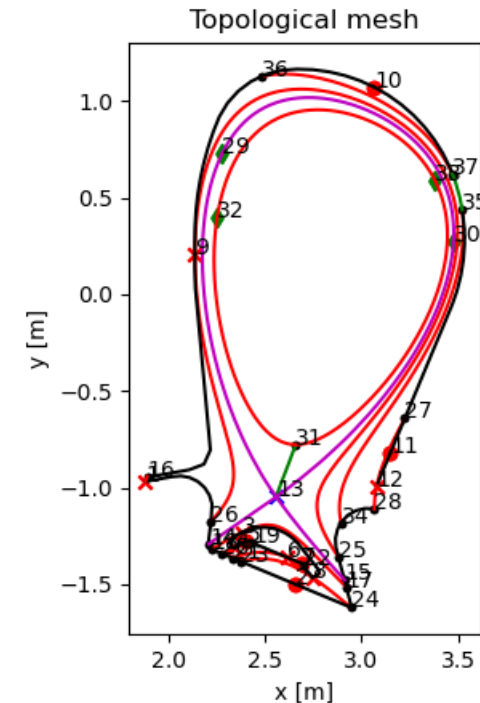
# Results after rerunning goat.exe



Oops, a hanging region! Structure 8 seems to extend too much...

Try again, but say that regions need to be covered fully by structures to retain more regions

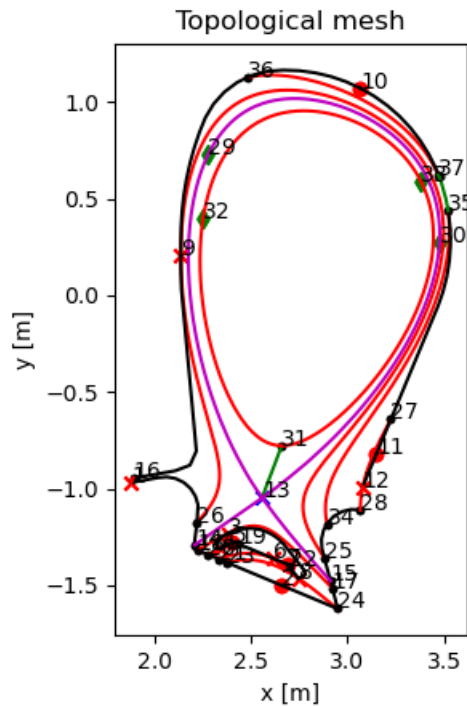
```
491 'gg.tm.removevesselregions' '1'  
492 'gg.tm.rvrretain' '0'  
493 'gg.tm.rvrvesselIDs' '[7:10]'  
494 'gg.tm.rvrdocascade' '0'  
495 'gg.tm.rvrcascadedir' 'up'  
496 'gg.tm.rvrfullycovered' '1'  
497
```



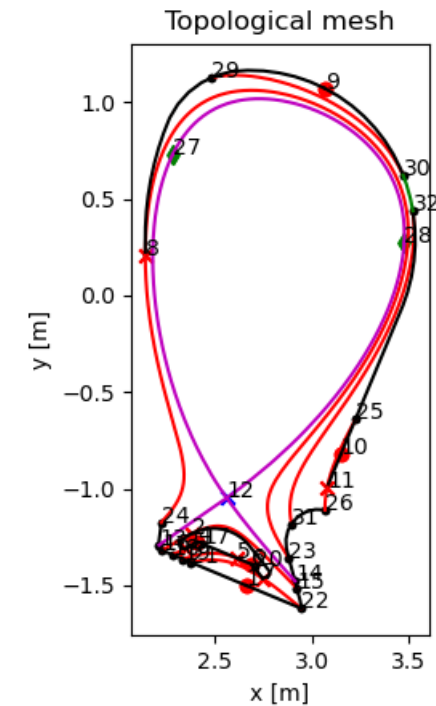
Better, but not great with the duct at the HFS

# Another try

- What if we choose to *retain* regions adjacent to structures 1-6?



```
491 'gg.tm.removevesselregions' '1'  
492 'gg.tm.rvrretain' '1'  
493 'gg.tm.rvrvesselIDs' '[1:6]'  
494 'gg.tm.rvrdocascade' '0'  
495 'gg.tm.rvrcascadedir' 'up'  
496 'gg.tm.rvrfullycovered' '1'  
497  
498 'gg.tm.alignvesselparts' '0'  
499 'gg.tm.avpminangle' '5.0'  
500
```

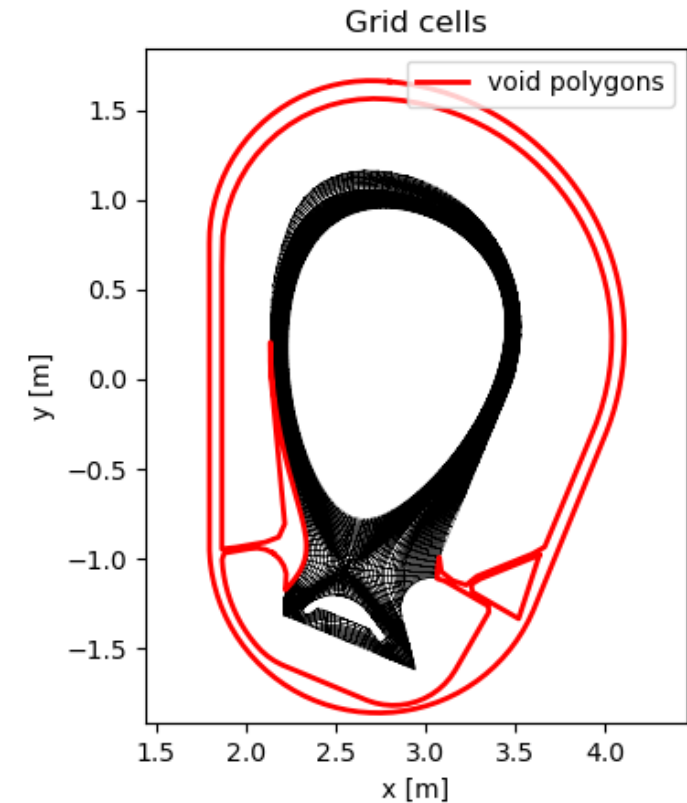
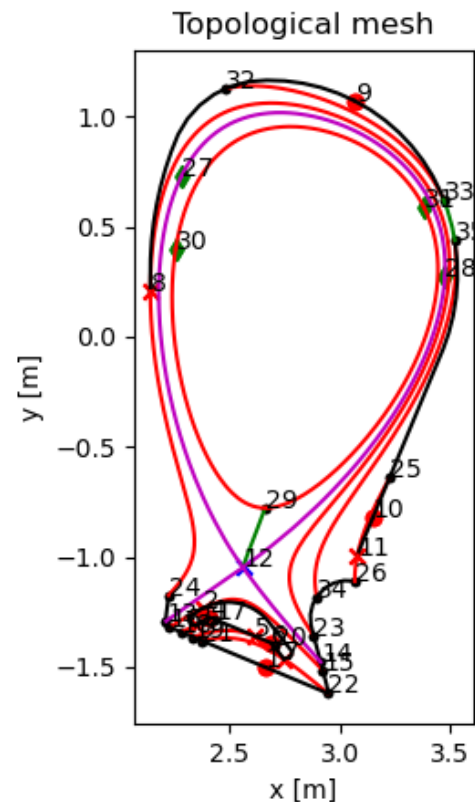


Looks good, but: no core!  
Is not adjacent to structures...

# Another another try

```
'gg.tm.removevesselregions' '1'  
'gg.tm.rvrretain' '1'  
'gg.tm.rvrvesselIDs' '[1:6]'  
'gg.tm.rvrdocascade' '1'  
'gg.tm.rvrcascadedir' 'down'  
'gg.tm.rvrfullycovered' '1'
```

This means that structures that are connected downstream (or upstream if `rvrcascadedir` is 'up') according to the Psi gradient are also considered for deletion/retention – should make sure the core is kept



Good enough for now... Void polygons are also given in fort.78

# Triang/Uinp

- Go back to the terminal where the SOLPS environment is loaded, and navigate back to the case folder, if necessary
- Run 'lns first\_case\_route1' (or however you've called your DivGeo setup – should create necessary links for triang)

# Triang/Uinp

- Run triang, should yield:

```
DEVICE VNS
Goat grid generator detected in DG model
Looking for traduit.out.b2us in current folder...
Grid name set to 'traduit.out.b2us'
Looking for fort_goat.78 in current folder...
Void polygon file name set to 'fort_goat.78'
Help, Uinp, B2ag, triaiNp, Tria, b2us (G), Plot, View, Store, Convert, List, Remove, reMap, Inquire, Quit ? (U)
```

- When running Uinp (press enter), you might get this error:

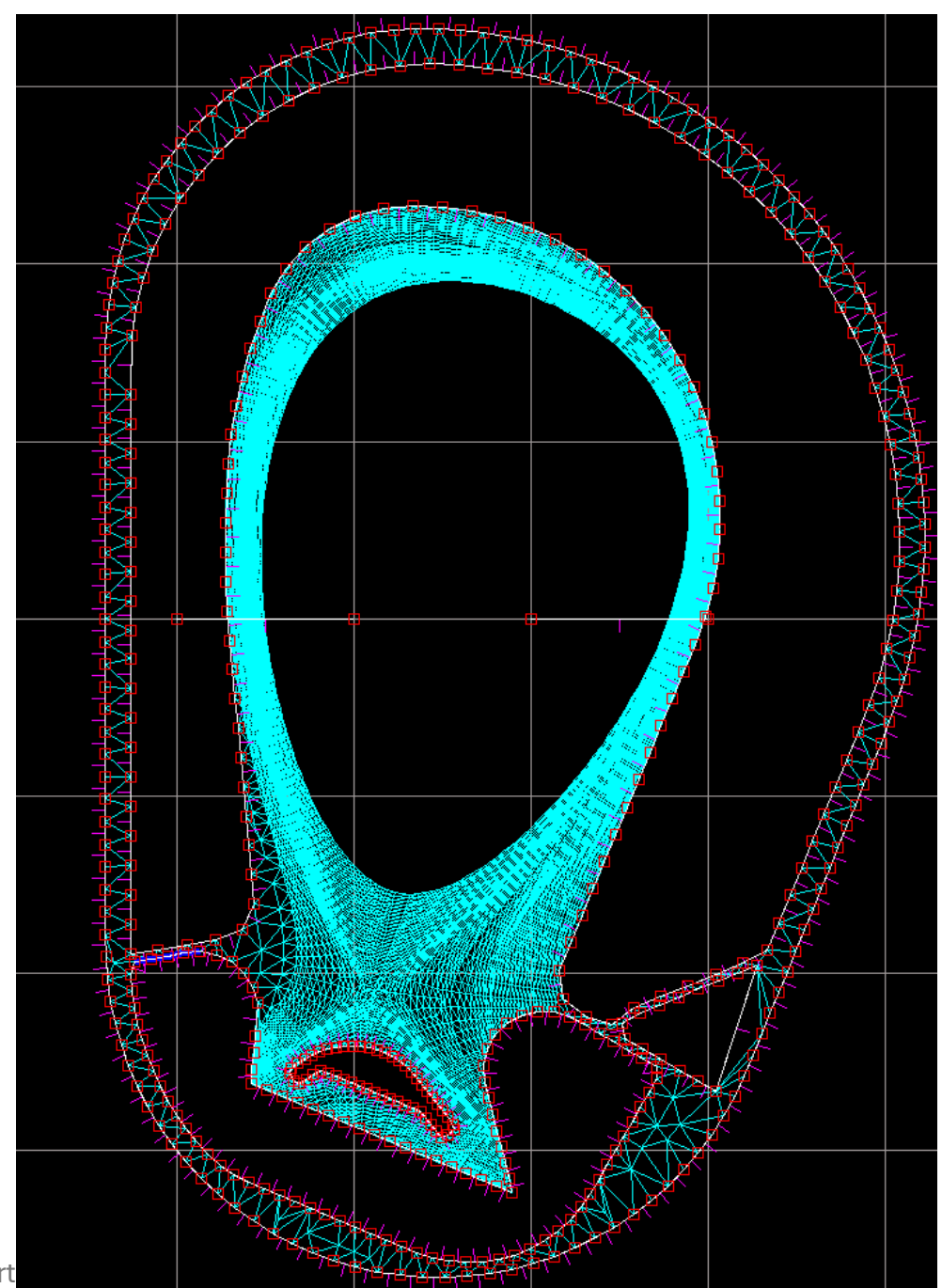
```
uinam: species      1      1
      ==> D
The slot specification is missing for gas puffing stratum      1
*** XERRAB: program will stop. ***
==> Check the "Gas puff" variables in the DG model
Call chain follows.
  uinam
  uininp
  uinput
STOP xerrab--error
```

# Triang/Uinp

- This is because by default a gas puff is added. Let's just remove this one for now in DivGeo: 'variables > remove > gas puff'
- Save and do 'output' again to create new .dgo file
- Rerun triang (first remove any b2\* or input\* files already generated)
- Note: you may want to review some of the input files for b2, e.g. b2ag etc, to see if you need to modify any switches. Here, we assume this is not the case...

# Triang/Uinp

- After a successful run, you should be able to import the template (see figure)
- If something went wrong, make sure to really start Uinp from scratch again! Otherwise, some files may not be overwritten (best to `rm b2*`; `rm input*` before restarting)
- Note: `b2mn.dat` file you may want to store (template is really bare)



# Running b2mn

- Remember to do 'setup\_baserun\_eirene\_links' in your run folder if not done already!
- The b2mn.dat file created by Uinp is rather a b2us.dat file... However, it should be sufficient to technically run the code (results will not be satisfactory though)

# Route 2: adding void interfaces

- If you did route 1, you noticed that region removal through goat is not straightforward (may be improved though).
- Adding void interfaces gives much more direct control, at the cost of not having non-aligned void interfaces (may not be an issue)
- **Important:** void interfaces must have negative face label and belong to vessel elements and Eirene void polygon

# Route 2: overview

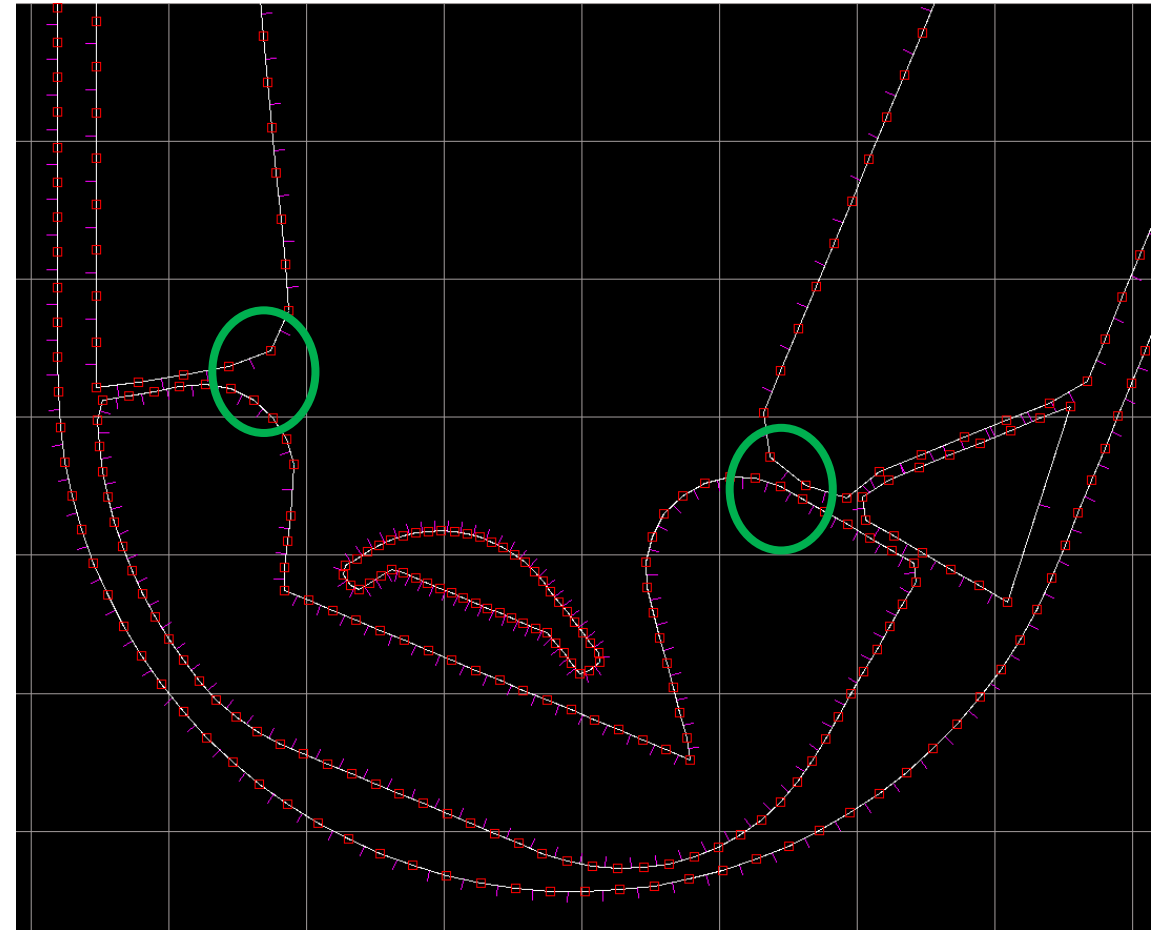
- DivGeo: starting from 'first\_case\_general'
  - Adding void interfaces
  - Specifying vessel elements (face labels can remain the same)
  - Specifying Eirene void polygons
  - Goat
  - Triang/Uinp

# DivGeo: preliminaries

- First, create a new directory, e.g. 'run\_route2' and copy-paste files from 'baserun' there
- Open the 'first\_case\_general' setup and save this setup as 'first\_case\_route2' (just as a safety)
- Open this case in DivGeo

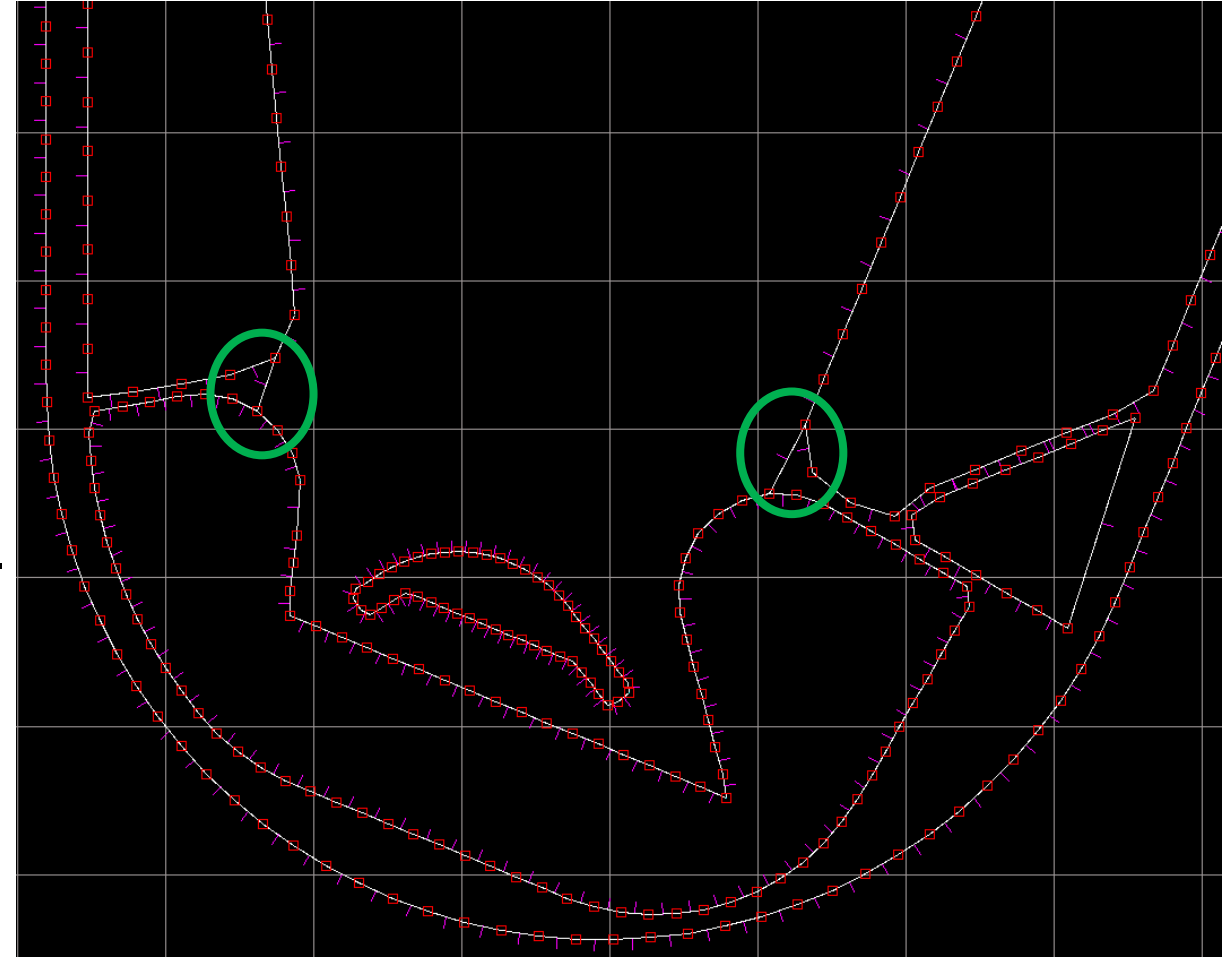
# DivGeo: void interfaces

- Looking at the geometry, we can simply connect points near the two green circles to limit the plasma
- Do so by adding elements by using 'connect points'

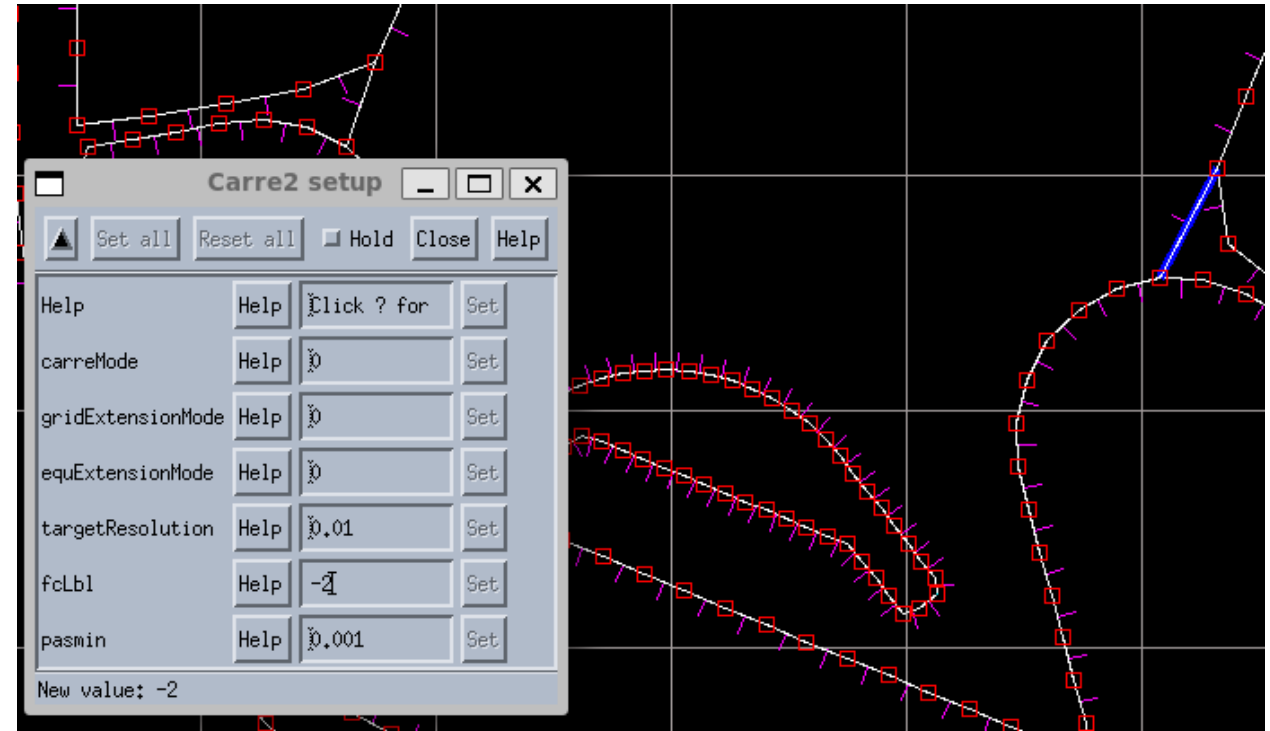
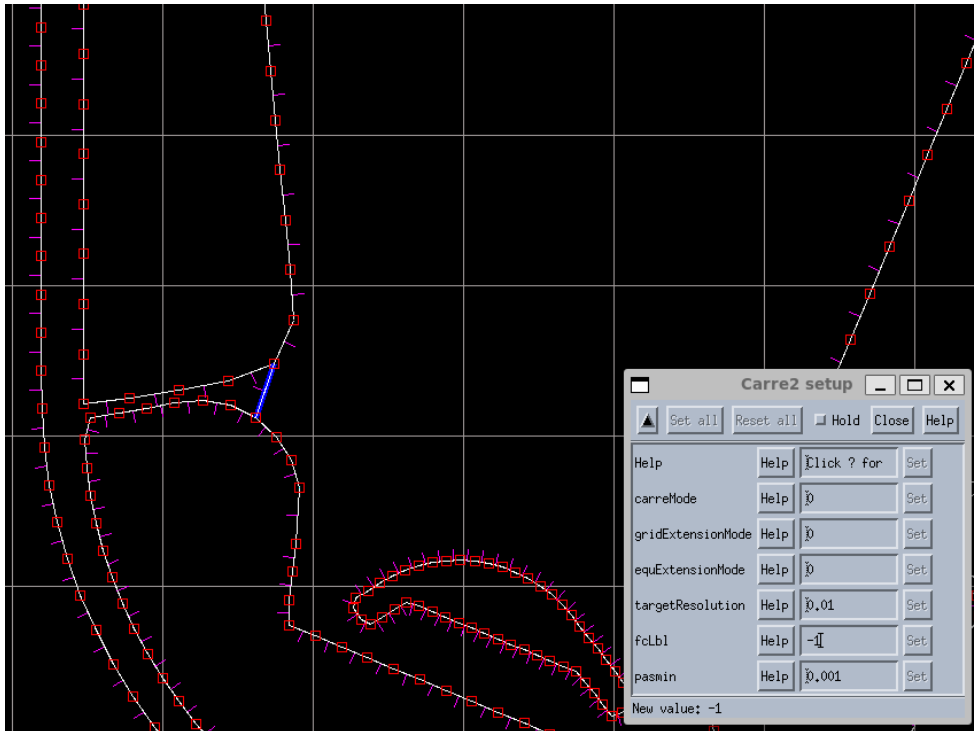


# DivGeo: void interfaces

- Result: see figure.
- Now, we still need to add labels in the carre2 setup layer (next slide)
- Note: these labels must be different and negative! Also: avoid labels reserved for core (-21, -25, -29, ...)

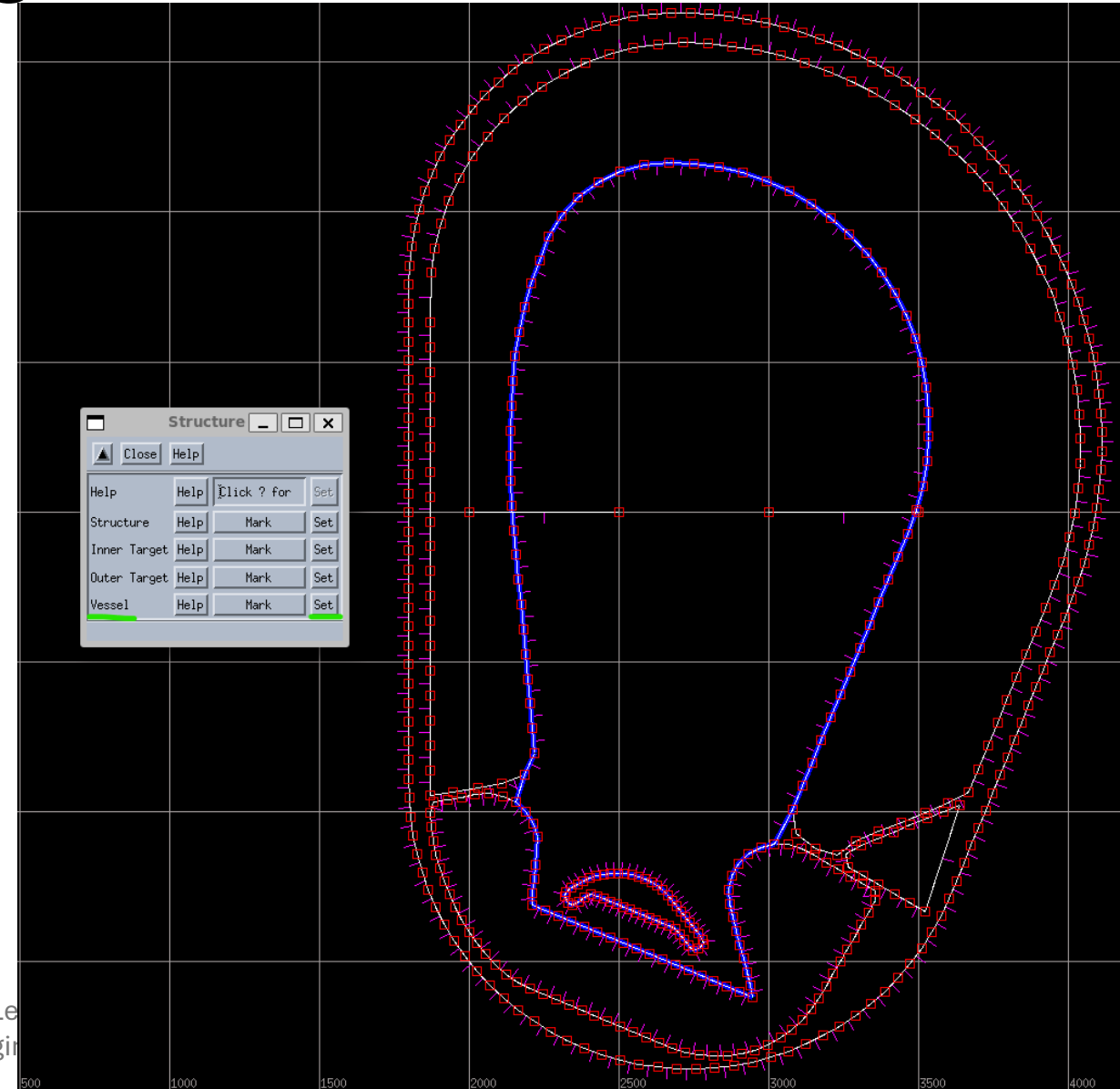


# DivGeo: void interfaces



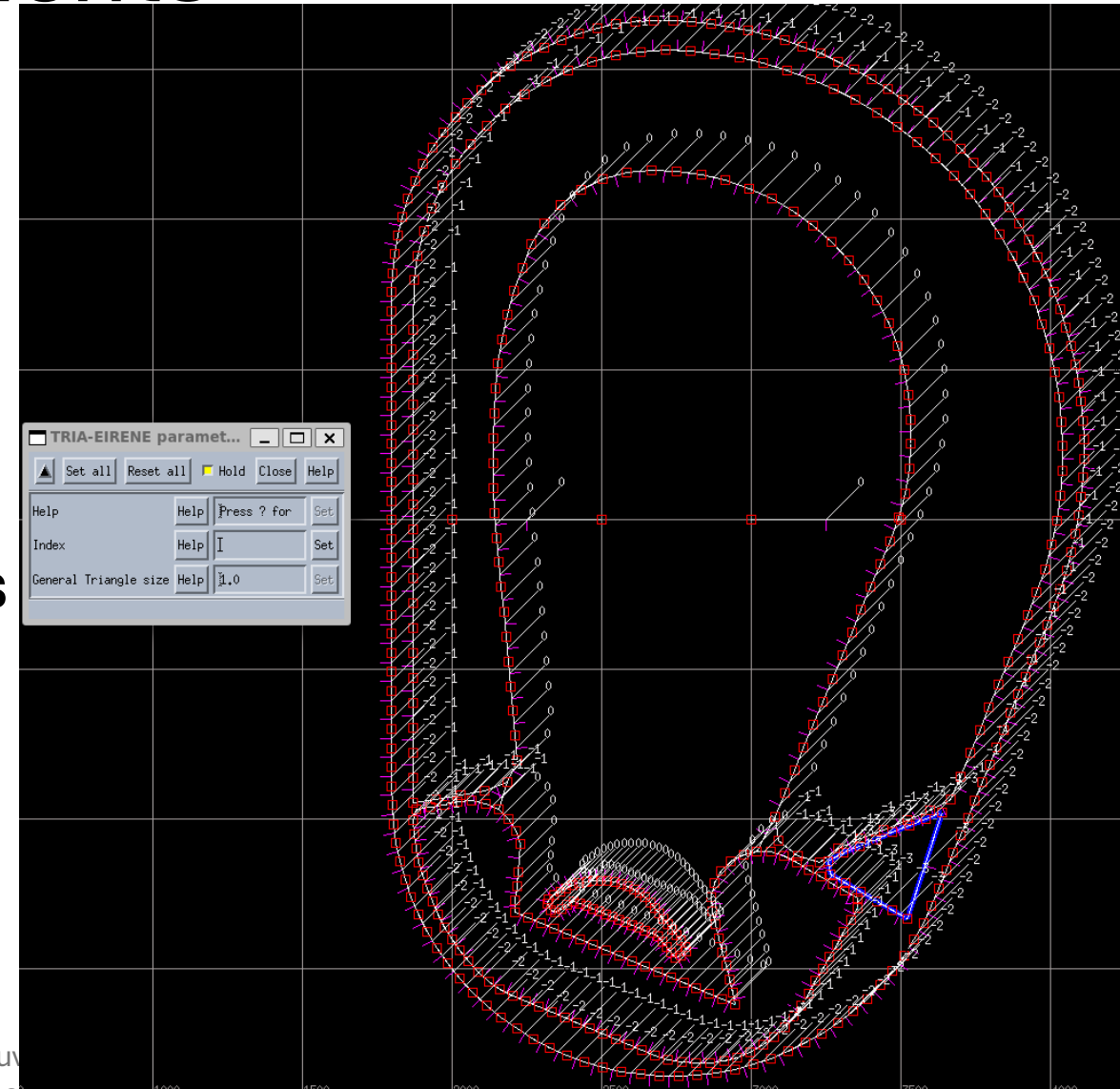
# DivGeo: vessel elements

- Now, we need to specify which elements belong to the vessel:
  - This includes the void interfaces!
  - This shouldn't include any other additional Eirene void polygon
- See figure for all elements to mark under 'variables > Structure > Vessel'



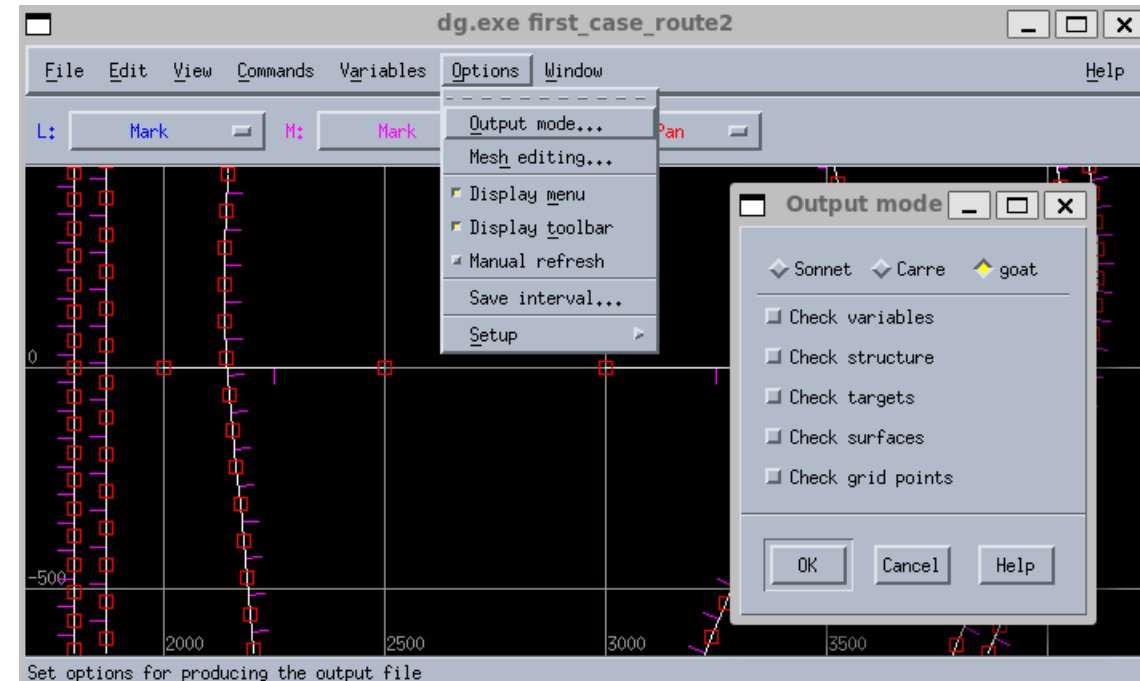
# DivGeo: Eirene void elements

- Since we know that goat will cover the entire vessel region, we can specify only the actual void polygons
- For an example, see figure. Values of index don't really matter here, as long as they are non-zero



# DivGeo: data output

- To get the necessary .dgo file for goat, go to 'output mode' and select 'goat' (this doesn't really matter, carre is also possible). Uncheck all checks – many of those are not necessary
- Then, do 'file > output' and you should see a message saying it wrote a set of files (only .dgo important)



# Running goat

- Ensure correct input files & read method
- Start goat from scratch
- Adaptations:
  - Core removal
  - Region/tube merging
- After this, run goat ('goat.exe' or 'goat\_debug.exe')

```
'goat.magneticfieldfilepath'      'Equil_VNS_07_2026_39d8s_opt_COCOS11.equ'  
'goat.structurefilepath'         'first_case_route2.dgo'
```

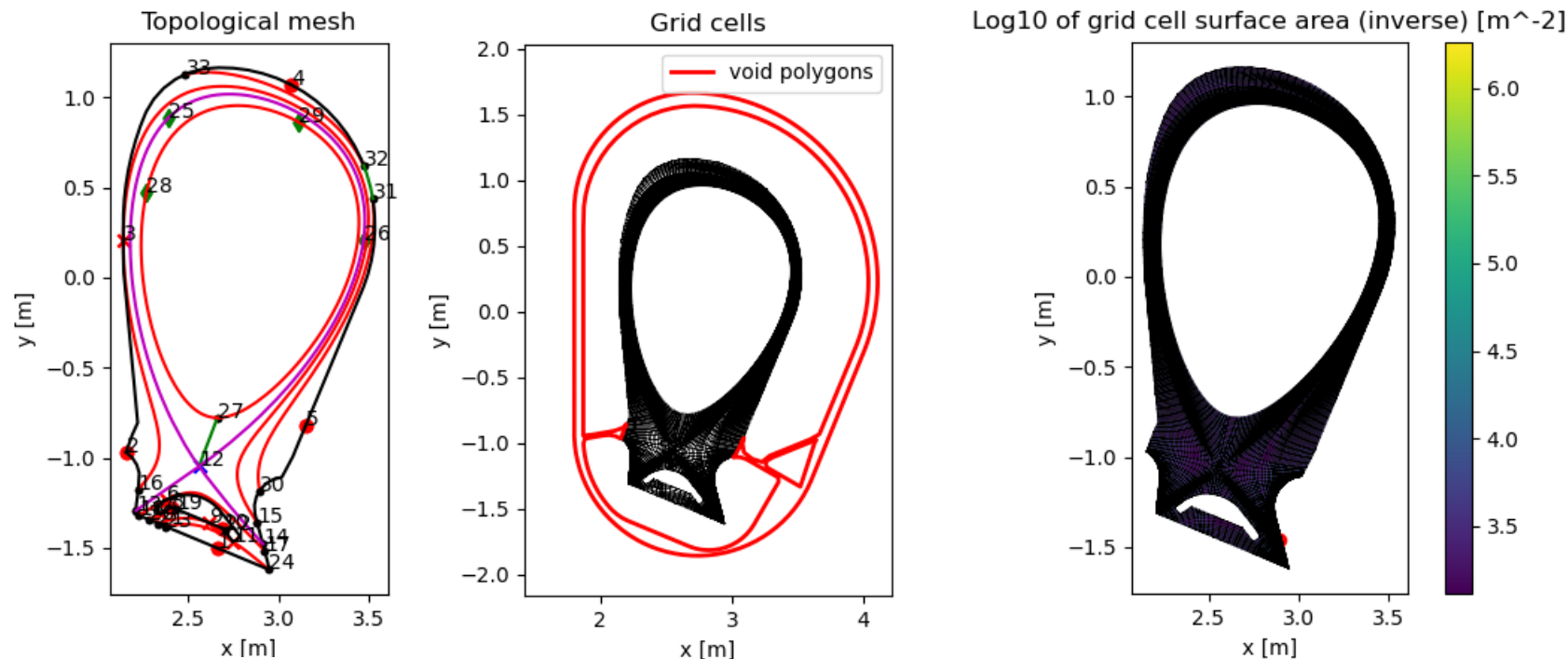
```
91  # Read method  
92  'goat.mf.readmeth'              'equ'
```

```
105 # Environment (including vessel etc) options  
106 # Vessel geometry & reading method  
107 'goat.vessel.readmeth'          'read_dg'  
108
```

```
448 # Read existing file?  
449 'gg.tm.readexistingTM'           '0'  
450 'gg.tm.readexistingtracers'     '0'  
474 # Core boundary and region treatment  
475 'gg.tm.doadaptations'            '1'  
476 'gg.tm.addcoreboundaries'       '1'  
477 'gg.tm.addPFboundaries'         '0'  
478 'gg.tm.cbnd.frac'               '0.1'  
479 'gg.tm.PFbnd.frac'              '0.1'  
480 'gg.tm.removecoreregions'       '1'  
481 'gg.tm.removewidegridregions'   '0'  
482  
483 'gg.tm.mergetubes'              '1'  
484 'gg.tm.dpsimintubes'            '0.0'  
485 'gg.tm.lradmintubes'            '1.0e-3'  
486 'gg.tm.mergetubes.oldstyle'     '0'  
487 'gg.tm.mergetubes.allowseparatrix' '1'  
488 'gg.tm.simplifyseparatrix'       '0'  
489 'gg.tm.simplifyseparatrix.lpolsepmin' '1e-3'  
490  
491 'gg.tm.removevesselregions'     '0'
```

# Goat output

- Run 'pggoutput' to see what goat gave – should be good enough
- Note that voids are retained in the void polygon, but ignored by goat



# Triang/Uinp

- Run 'lns first\_case\_route2' to link correct files for triang
- Run triang, should yield:

```
DEVICE VNS
Goat grid generator detected in DG model
Looking for traduit.out.b2us in current folder...
Grid name set to 'traduit.out.b2us'
Looking for fort_goat.78 in current folder...
Void polygon file name set to 'fort_goat.78'
Help, Uinp, B2ag, triaiNp, Tria, b2us (G), Plot, View, Store, Convert, List, Remove, reMap, Inquire, Quit ? (U)
```

- When running Uinp (press enter), you might get this error:

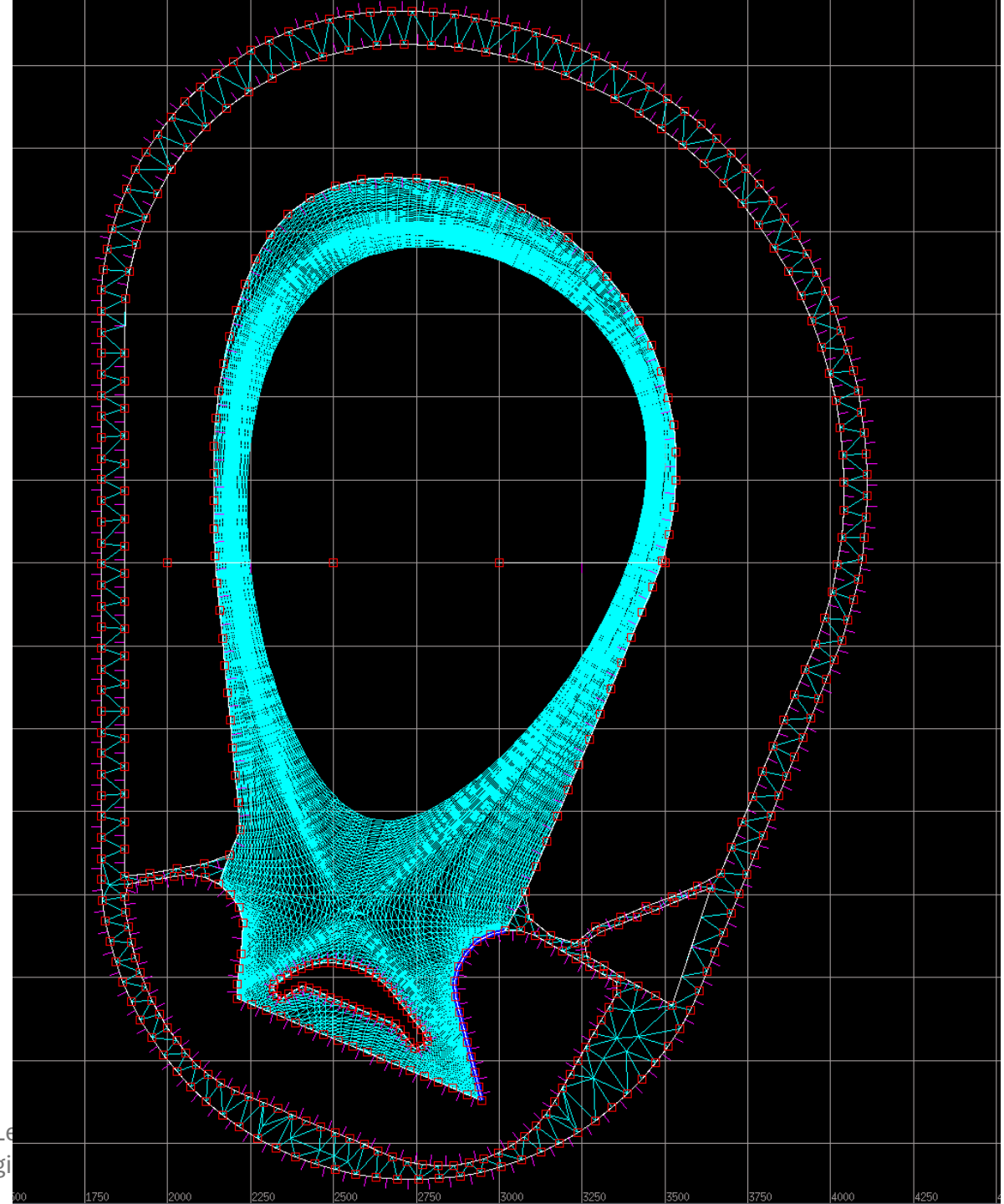
```
uinnam: species      1      1
                        ==> D
The slot specification is missing for gas puffing stratum      1
*** XERRAB: program will stop. ***
==> Check the "Gas puff" variables in the DG model
Call chain follows.
  uinnam
  uininp
  uinput
STOP xerrab--error
```

# Triang/Uinp

- This is because by default a gas puff is added. Let's just remove this one for now in DivGeo: 'variables > remove > gas puff'
- Save and do 'output' again to create new .dgo file
- Rerun triang (first remove any b2\* or input\* files already generated)
- Note: you may want to review some of the input files for b2, e.g. b2ag etc, to see if you need to modify any switches. Here, we assume this is not the case...

# Triang/Uinp

- After successfully pressing enter, you should have converted and stored the template files to read into DivGeo



# Running b2mn

- Remember to do 'setup\_baserun\_eirene\_links' in your run folder if not done already!
- The b2mn.dat file created by Uinp is rather a b2us.dat file... However, it should be sufficient to technically run the code (results will not be satisfactory though)