

SOLPS-ITER

Code Camp 2026

19-23 October 2026

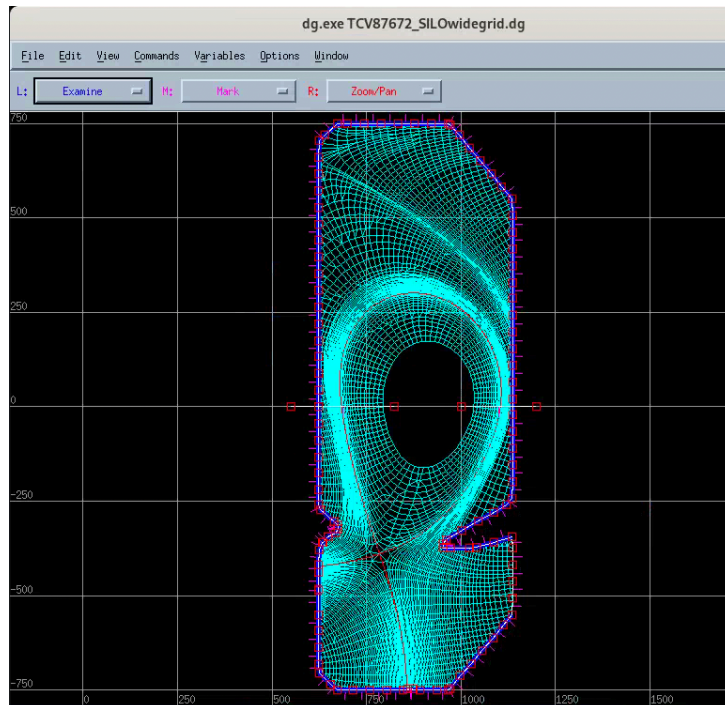
EPFL - Lausanne, Switzerland

**Workflow to generate a WG grid with GOAT and
all the input files for coupled runs:
my personal (non-elegant but working) fix**

Goal: having a working coupled SOLPS simulation, with impurities, sputtering without modifying/writing “by hand” input.dat and the other input files

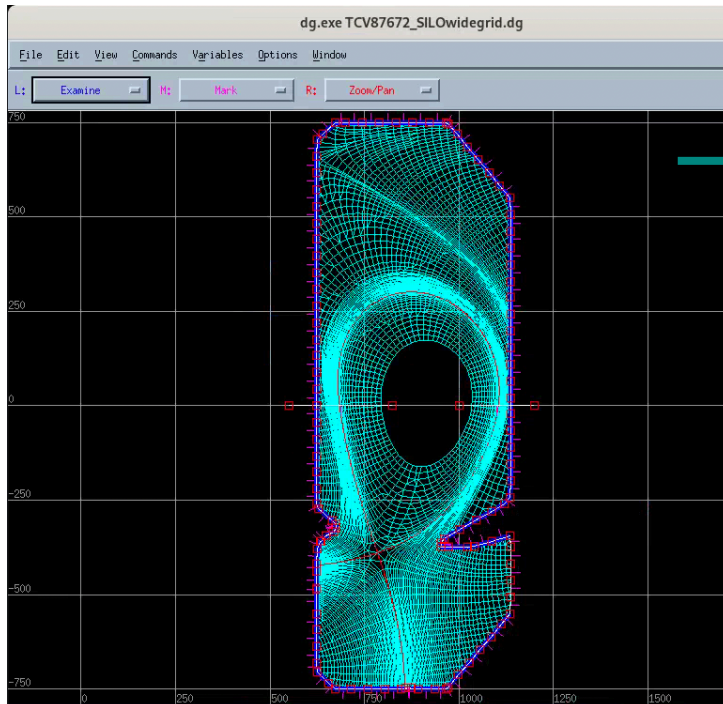
Goal: having a working coupled SOLPS simulation, with impurities, sputtering without modifying/writing “by hand” input.dat and the other input files

We still use DG to set the geometry



Goal: having a working coupled SOLPS simulation, with impurities, sputtering without modifying/writing “by hand” input.dat and the other input files

We still use DG to set the geometry

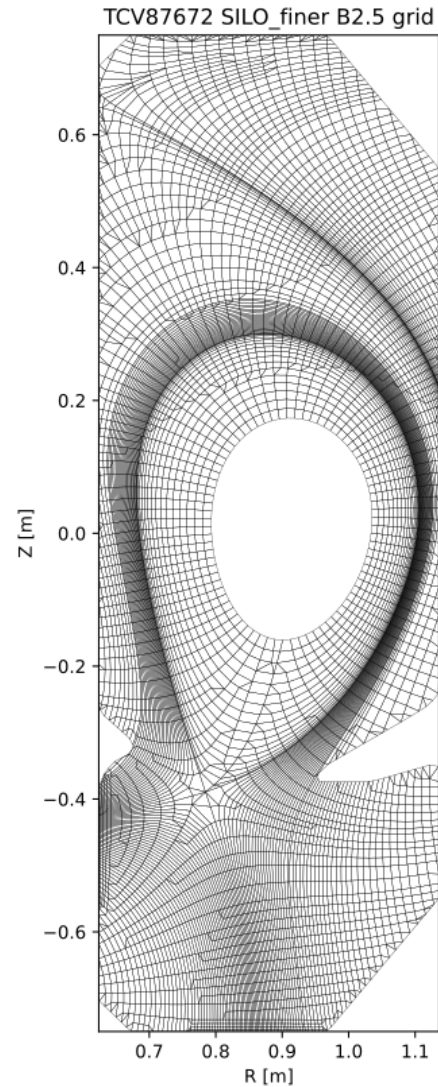


We want to be able to use UINP to write the input files for us

UINP

input.eir, input_avg.eir
input_fin.eir
b2a*.dat
b2.boundary.parameters
b2.neutrals.parameters
b2.user.parameters

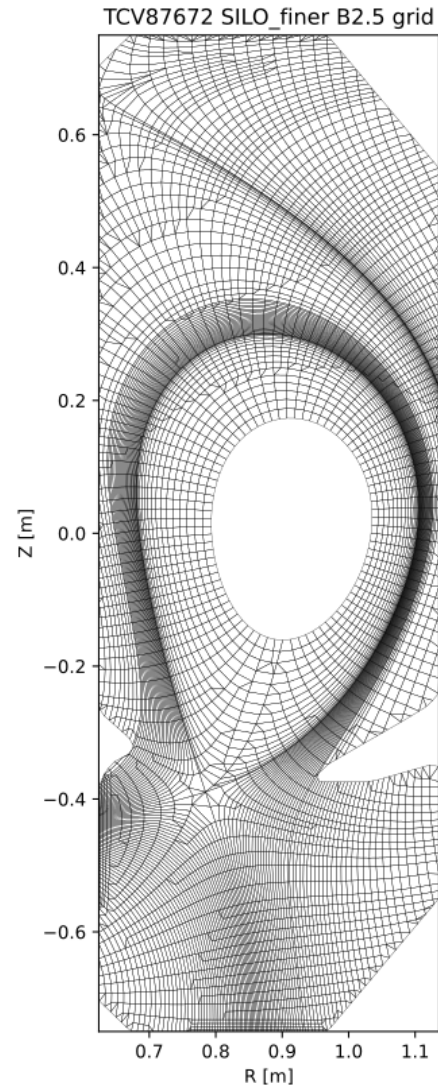
Assuming we successfully built a mesh with GOAT



> goat.exe

* This works using Atonello's branch of GOAT, where the volume and face regions are set according to the standard B2.5

Assuming we successfully built a mesh with GOAT

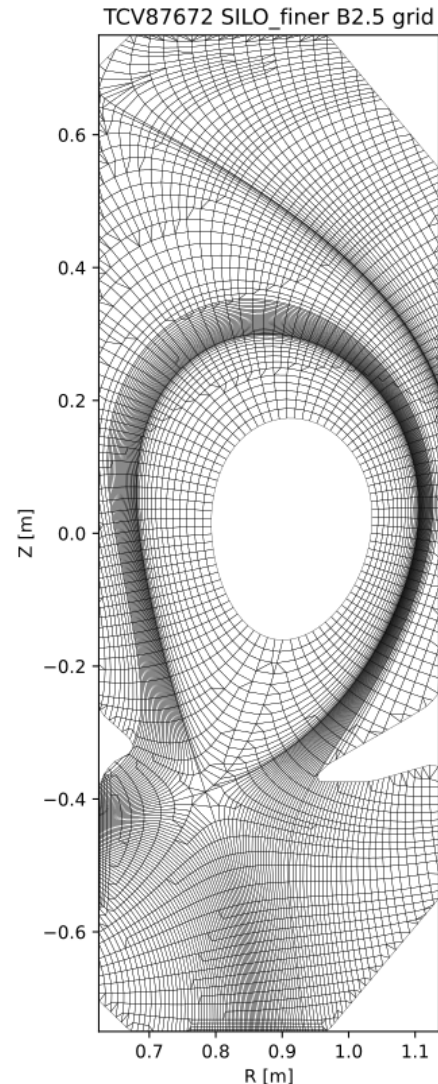


```
> goat.exe
```

```
{ GOAToptions.dat  
  rzpsi.dat (or EQDSK.equ)  
  structure.dat
```

* This works using Atonello's branch of GOAT, where the volume and face regions are set according to the standard B2.5

Assuming we successfully built a mesh with GOAT



```
> goat.exe
```

```
{
  GOAToptions.dat
  rzpsi.dat (or EQDSK.equ)
  structure.dat
}
```

* This works using Atonello's branch of GOAT, where the volume and face regions are set according to the standard B2.5

Since we are using DG to set up your geometry (Target Specification, Vessel Specification, General Surface Data...), we have all the info needed to build the structure file in the DG Output (dg.dgo).

The approach keeps the same fclbl as in DG:

```
> python3 dgo_to_structure.py --help
```

```
usage: dgo_to_structure.py [-h] [--in INFILE] [--out OUTFILE] [--tol TOL]
```

Convert a translator `.dgo` file (sections `p1`, `p2`, `fclbl`) into a `structure.dat` file for GOAT.

Usage:

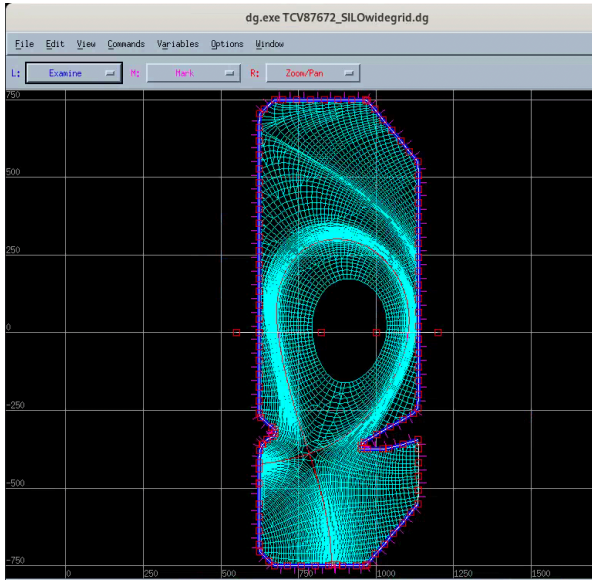
```
dgo_to_structure.py [--in dg.dgo] [--out structure.dat] [--tol 1e-6]
```

```
--in/-i  input .dgo file (default dg.dgo)
```

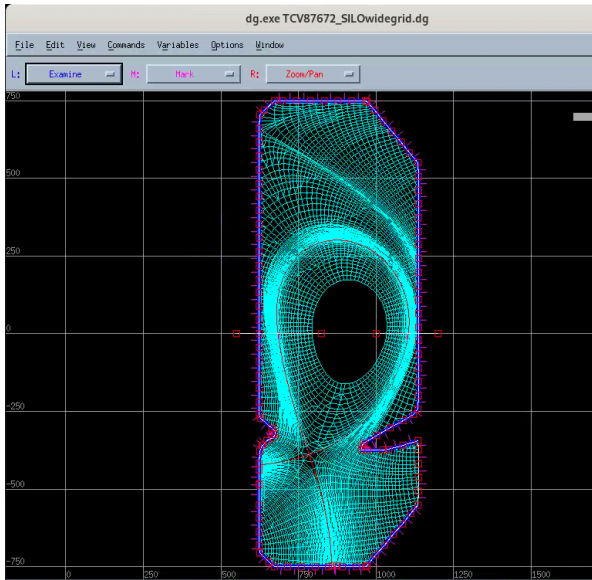
```
--out/-o output file (default structure.dat)
```

```
--tol    tolerance in metres for deciding that two segment end points coincide (default 1e-6)
```

Uinp workflow

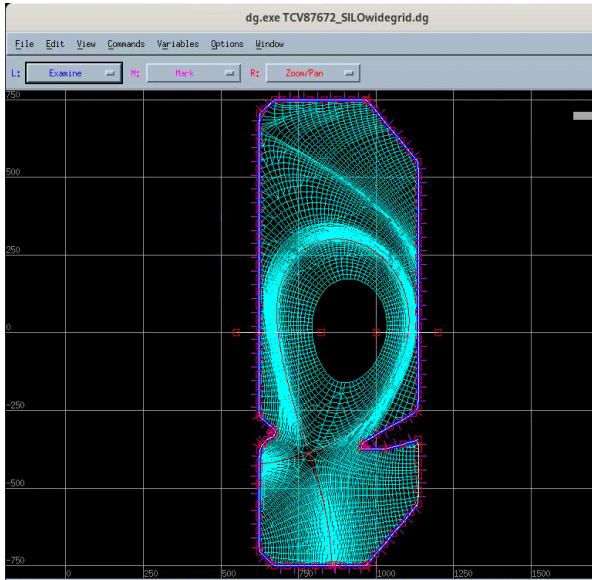


Uinp workflow



TCV87672_SIL0widegrid.dgo -> `uinput.dg`
TCV87672_SIL0widegrid.str
TCV87672_SIL0widegrid.trg -> `uinput.trg`
??? -> `uinput.crr` if CarreMode >= 2

Uinp workflow



TCV87672_SIL0widegrid.dgo -> `uinput.dg`
TCV87672_SIL0widegrid.str
TCV87672_SIL0widegrid.trg -> `uinput.trg`
??? -> `uinput.crr` if CarreMode >= 2

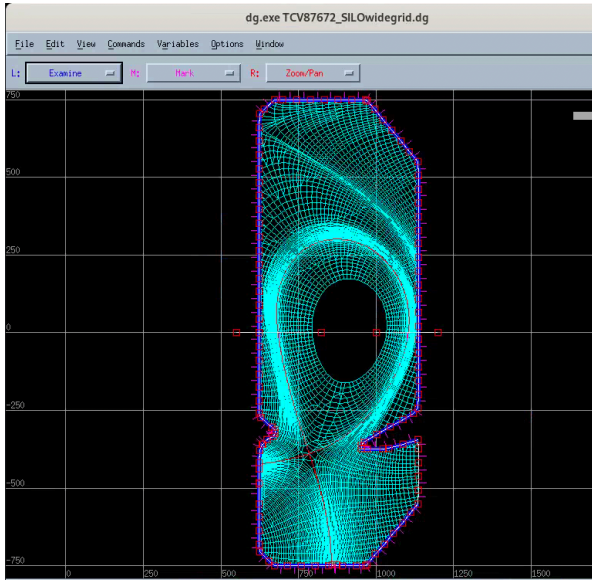
↓

UINP

↓

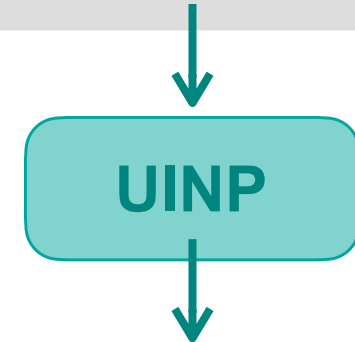
`input.eir, input_avg.eir`
`input_fin.eir`
`b2a*.dat`
`b2.boundary.parameters`
`b2.neutrals.parameters`
`b2.user.parameters`

Uinp workflow



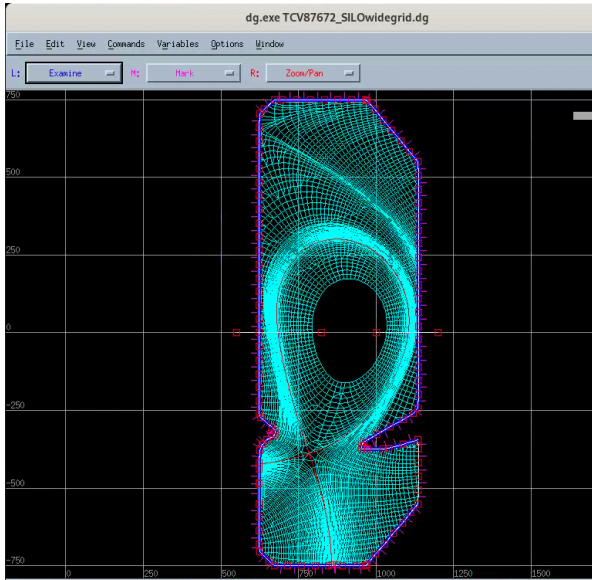
TCV87672_SIL0widegrid.dgo -> `uinput.dg`
TCV87672_SIL0widegrid.str
TCV87672_SIL0widegrid.trg -> `uinput.trg`
??? -> `uinput.crr` if CarreMode >= 2

**How to create a
uinput.crr without
running Carre2?**



`input.eir, input_avg.eir`
`input_fin.eir`
`b2a*.dat`
`b2.boundary.parameters`
`b2.neutrals.parameters`
`b2.user.parameters`

Uinp workflow



TCV87672_SIL0widegrid.dgo -> `uinput.dg`
 TCV87672_SIL0widegrid.str
 TCV87672_SIL0widegrid.trg -> `uinput.trg`
 `???`
 -> `uinput.crr` if CarreMode >= 2

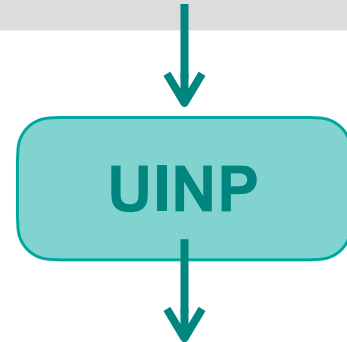
**How to create a
uinput.crr without
running Carre2?**

`traduit_to_uinput_crr.py`

Standalone script: Generate `uinput.crr` from `traduit.out.b2us`

Usage:

`traduit_to_uinput_crr.py <traduit_file> [output_file]`



`input.eir`, `input_avg.eir`
`input_fin.eir`
`b2a*.dat`
`b2.boundary.parameters`
`b2.neutrals.parameters`
`b2.user.parameters`

traduit_to_uinput_crr.py

The script extracts grid counts (n_{Ci} , n_{Fc} , n_{Cg}) from the traduit header and generates a pseudo carre.out file suitable for uinp input in coupled EIRENE cases.

The synthetic file does not contain the real geometry that Carre2 would produce: it only contains cell/face flags arranged so that uinp's uincrr reproduces the grid counts of the unstructured grid.

The number of regions is taken from the largest cvReg of the interior cells ($nnreg = \max(cvReg) - 1$).

Only a single-null (SN) topology, $nnreg = 3$, is supported; any other value is an error. The mesh parameters nptseg/npr and the per-region dimensions (nprad = 25, 3, 3) are fixed values in write_uinput_crr. Run without arguments or with -h/--help to print this text.

Now we can run uinp

```
> traduit_to_uinput_crr.py traduit.out.b2us uinput.crr
```

```
> uinp
```

Now we can run uinp

```
> traduit_to_uinput_crr.py traduit.out.b2us uinput.crr
```

```
> uinp
```

Almost all good... except b2ag.dat

```
*dimens      (nx, ny, nx1, ny1; free format)
      117      30      117      30
```

Now we can run uinp

```
> traduit_to_uinput_crr.py traduit.out.b2us uinput.crr
```

```
> uinp
```

Almost all good... except b2ag.dat

```
*dimens      (nx, ny, nx1, ny1; free format)
      117      30      117      30
```

```
> update_b2ag_dimens.py b2ag.dat
```

```
*format
      unstructured
*dimens      (nCi,nFc,nVx; free format)
      8621      17333      8712
```

Bugs (or similar) that I might have found

```
*format
    unstructured
*dimens      (nCi,nFc,nVx; free format)
    <nCi>      <nFc>      <nVx>
*param      (param(0:99); free format)
    -1.0, 99*0.0
'b2agfs_min_pitch'      '0.01'
'b2agdr_redef_gmtry'    '0'
'b2agfs_geometry'      'traduit.out.b2us'
'b2agmt_style'          '1'
'b2us_prep_mod_hc'      '1'
'b2us_prep_Qalfmax'     '0.004'
'b2us_prep_Qalfmin'     '0.002'
'b2us_prep_Pbs_reduce_fac' '1e-6'
'b2us_prep_set_beta_0'  '1'
```

All these switches are only used at b2us step, but because the grid is already unstructured, b2us does nothing on b2fgmtry following this workflow!

See Nikita's talk from Code Camp 2025

Bugs (or similar) that I might have found

The default values in b2fgmtry are used independently on the values set in b2ag.dat

```
*cf:      real          1      Qalfmin  
      1.00000000000000E-003  
*cf:      real          1      Qalfmax  
      1.00000000000000E-003
```

These will modify the boundary conditions for shallow flux tubes near the wall (see Nikita's talk from Code Camp 2025). The b2us_prep_Pbs_reduce_fac = small number is not implemented.

```
*cf:      real          1      Qalfmin  
      0.00000000000000E+000  
*cf:      real          1      Qalfmax  
      0.00000000000000E+000
```

It might be better to remove geometry/boundary condition modifications by default (?)

Bugs (or similar) that I might have found

When running with drifts, the perpendicular viscosity current is not ramp up:

Current	Routine	Gate when the drift factor is 0	Scaled by the factor?
Diamagnetic (dia_cur)	b2tdia.F:52	off	yes, facdrift
Inertial (b2tiner_inert)	b2tiner.F:58	off	yes, facdrift
Parallel viscosity (vis_par)	b2tvspa.F:86	off	yes, fac_vis
Heat-viscosity (vis_q)	b2tvsq.F:75	off	yes, fac_vis
Perpendicular viscosity (vis_per)	b2tvspr.F:69	off	no
Ion-neutral (b2tfhe_neutral)	b2tinnt.F:90 and the conductivity add-on in b2tfch_.F	off	no, only gated
Parallel (prl_cur), anomalous (prl_anomalous), vis_kt	b2tcpa, b2tanml, b2tvskt	always on	n/a

Starting a simulation with drifts and current with `b2tfhe_vis_per = 1.0` makes the simulation crash at the very first attempt of solving the potential equation.

Bugs (or similar) that I might have found

The sign of `parm_vla` now tells nothing about the actual direction of the pinch velocity:

In 3.0.9, `parm_vla < 0` meant an inward (towards the magnetic axis) pinch velocity, independent of the I_p and B_T directions.

In 3.2.1, the sign of `parm_vla` needs to be decided depending on the actual geometry file `b2fgmtry`

```
def vla_inward_sign(self, path: Path):
    """Which way does a POSITIVE anomalous radial velocity (vla, parm_vla) point on this mesh?

    B2.5 builds the radial face coefficient as fcS*fcQalf(:,1)*vla and counts a flux positive
    from fcCv(:,1) to fcCv(:,2), so the direction of +vla is set by the geometry, not by the
    input. This evaluates sign(fcQalf_y * dpsi(fcCv1 -> fcCv2)) over all radial faces (faces
    between two different flux tubes), with psi taken per cell as the mean of vxPsi over its
    vertices, and compares it with the direction towards the core (the closed flux tubes).

    Returns (sign, consistency): sign is +1 if positive vla points towards the core, -1 if it
    points away from it, so 'sign * speed' is the parm_vla that gives an inward pinch 'speed'.
    'consistency' is the fraction of radial faces that agree with that sign (1.0 on a clean mesh).
    """
```