

# Meshing and Postprocessing for Acoustic Topology Optimization

Felix Huber

Institute of Mechanics and Mechatronics - TU Wien

09.04.2026

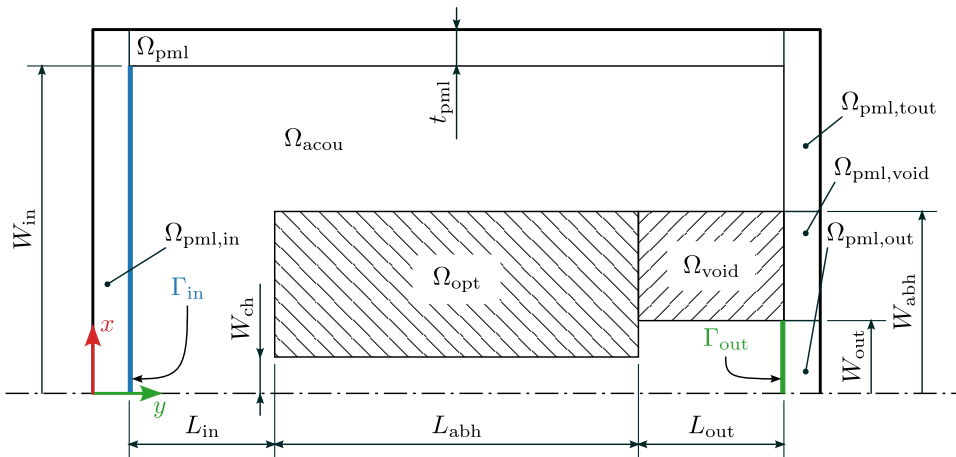
For simple regular meshes for topology optimization we have `create_mesh.py`.  
Adaption of this workflow to acoustics:

- ⊕ Pure Python meshing, no dependencies
- ⊕ Quick and parameterizable
- ⊕ Meshes in Ansys text format (used for post processing)
- ⊖ Only for specific problems and simple geometries
- ⊖ Limited parameters

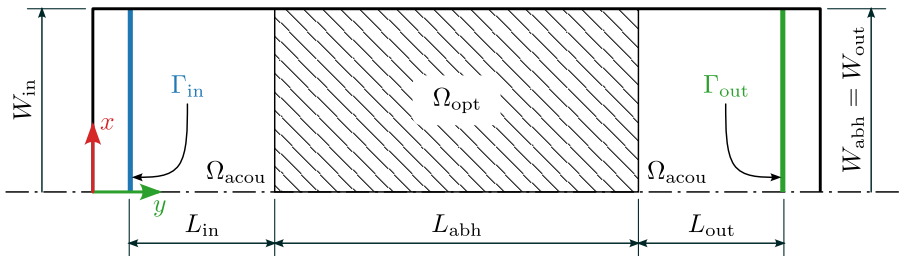
# make\_mesh.py

```
$ make_mesh.py
```

```
usage: make_mesh.py [-h] [-lin LIN] [-labh LABH] [-lout LOUT] \  
[-win WIN] [-wch WCH] [-wabh WABH] [-wout WOUT] [-tpml TPML] [-o O] nx
```

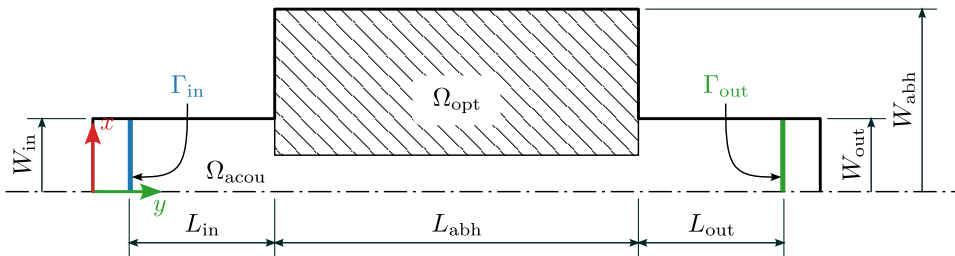


```
$ make_mesh.py -lin 0.02 -labh 0.05 -lout 0.02 \  
-win 0.025 -wabh 0.025 -wout 0.025 50
```



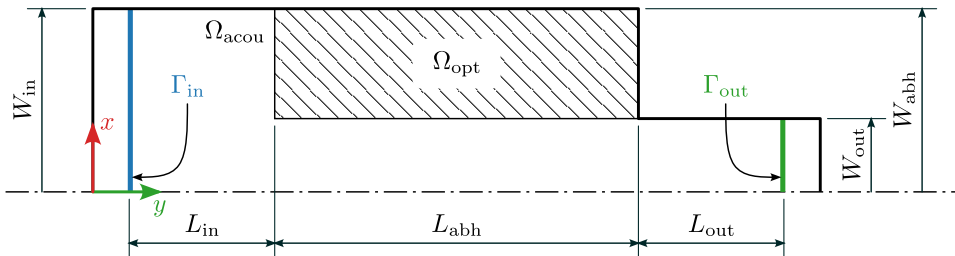
# Resonator

```
$ make_mesh.py -lin 0.02 -labh 0.05 -lout 0.02 \  
-win 0.01 -wch 0.005 -wabh 0.025 -wout 0.01 50
```

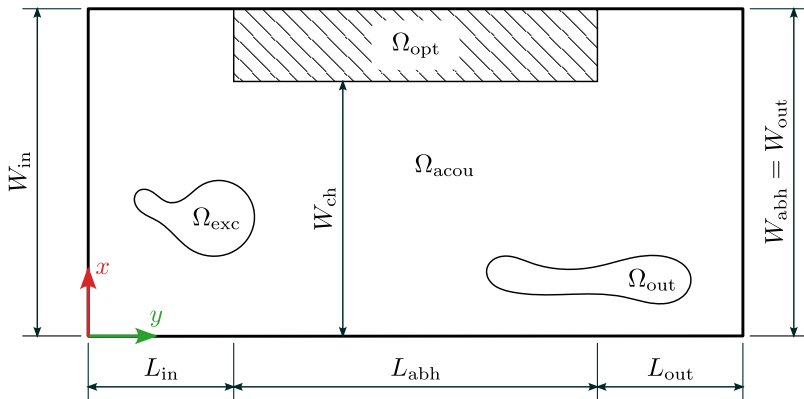


# Waveguide Focusing

```
$ make_mesh.py -lin 0.02 -labh 0.05 -lout 0.02 \  
-win 0.025 -wch 0.01 -wabh 0.025 -wout 0.01 50
```

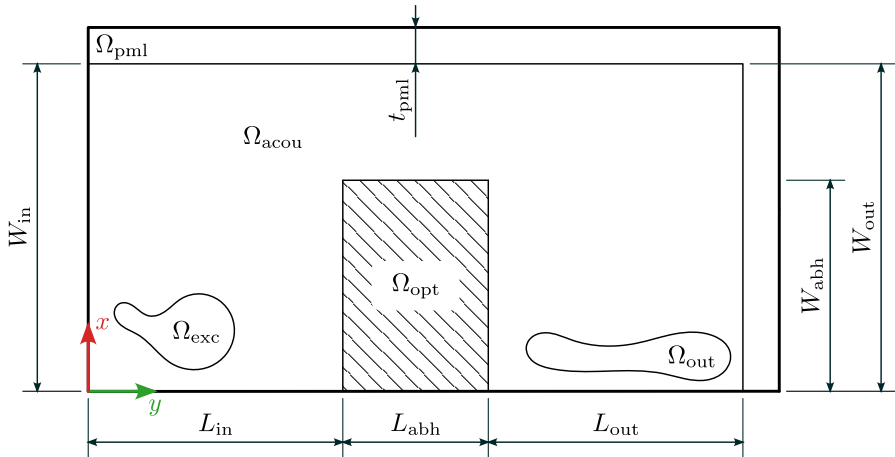


```
$ make_mesh.py -lin 2 -labh 5 -lout 2 \  
-win 4.5 -wch 4 -wabh 4.5 -wout 4.5 90
```



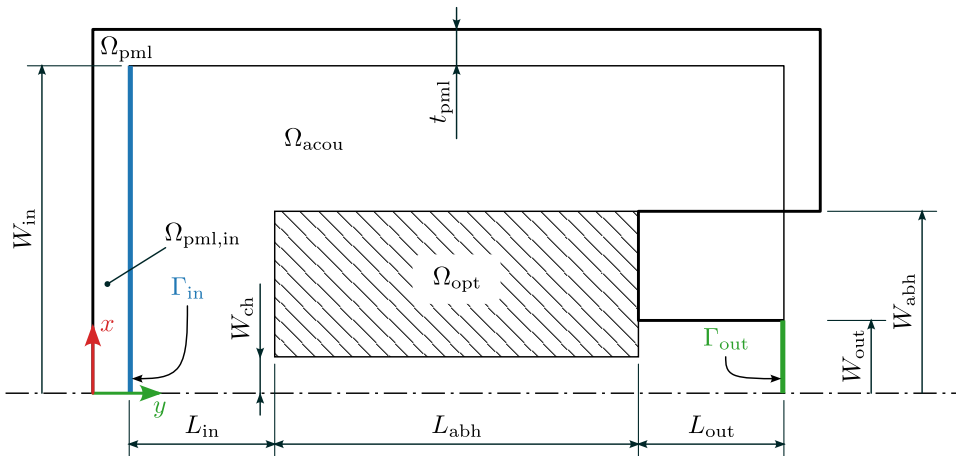
# Noise barrier

```
$ make_mesh.py -lin 3.5 -labh 2 -lout 3.5 \
-win 4.5 -wabh 3 -wout 4.5 -tpml 0.1 90
```



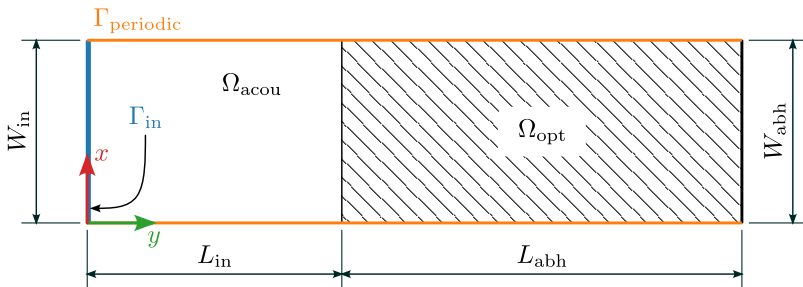
# Freefield Focusing

```
$ make_mesh.py -lin 0.02 -labh 0.05 -lout 0.02 \
-win 0.045 -wch 0.005 -wabh 0.025 -wout 0.005 -tpml 0.001 90
```



# Cone

```
$ make_mesh.py -lin 0.2 -labh 0.7 -win 0.25 -wabh 0.25 50
```



For parallelizing optimization runs and post processing, I propose `pool.py`.

- ⊕ Input in JSON format
- ⊕ Parallel and serial execution
- ⊕ Pure Python implementation, OS independent
- ⊕ Collect logs
- ⊖ No bash operators
- ⊖ SIGTERM does not work, use SIGINT

## pool.py

```
$ pool.py
usage: pool.py [-h] [-e] [-m MAXPROCESSES] \
[-n [NAME ...]] [-s SKIP] [--clean] input
```

Example input.json:

```
{
  "twoecho": ["echo 'Hello World'", "sleep 5", "echo 'Nap successful'"],
  "erroron2nd": ["echo Hi", "false", "echo `Don't reach!`"]
}
```

# Example Output

```
$ pool.py -m 2 input.json
1/2:   starting:       twoecho[0]
2/2:   starting:       erroron2nd[0]
1:     advancing:     -> twoecho[1]
2:     advancing:     -> erroron2nd[1]
2:     exiterror:     erroron2nd[1] (EXIT = 1)
1:     advancing:     -> twoecho[2]
1:     finished:      twoecho[2]
```

```
$ pool.py -m 1 input.json
1/2:   starting:       twoecho[0]
1:     advancing:     -> twoecho[1]
1:     advancing:     -> twoecho[2]
1:     finished:      twoecho[2]
2/2:   starting:       erroron2nd[0]
2:     advancing:     -> erroron2nd[1]
2:     exiterror:     erroron2nd[1] (EXIT = 1)
```

We create a log file for each process

```
$ cat twoecho.log  
Hello World  
Nap successful
```

```
$ cat erroron2nd.log  
Hi
```

With the `-e` option, we would get

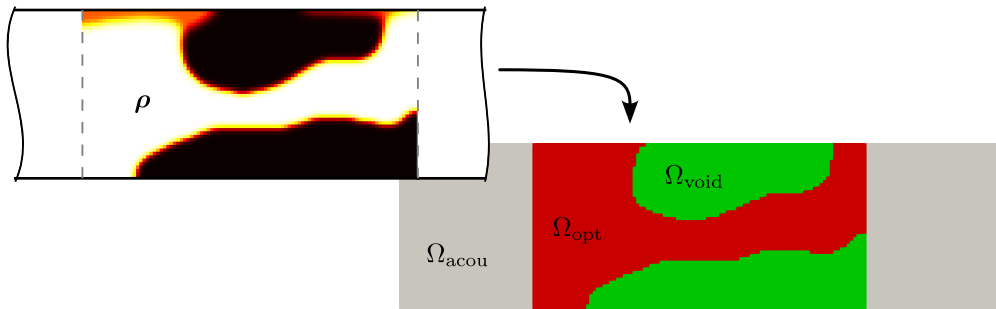
```
$ cat erroron2nd.log  
Hi  
Dont reach!
```

# Threshold

To generate a thresholded mesh from the `.density.xml` output from openCFS, we propose `threshold.py`.

- ⊕ Simple design
- ⊖ Only works for ANSYS mesh format

```
$ threshold.py  
usage: threshold.py [-h] [-th TH] density mesh newmesh
```

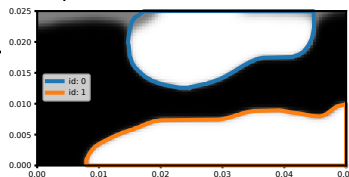
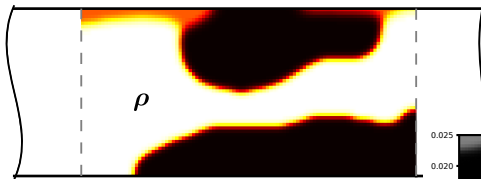


To revolve axisymmetric optimization results into 3D shapes, we propose `revolve.py`.

- ⊕ Versatile open-source workflow
- ⊕ Export in different formats
- ⊕ Easy modification of resulted structures
- ⊖ `revolve.py` not generalizable yet

```
$ revolve.py
usage: revolve.py [-h] -nx NX -labh LABH -win WIN \
[-a ANGLE] [--save SAVE] image
```

# Revolve Workflow



```

1 $fn = 200;
2 $vpd = 0.19;
3 $vpt = [0, 0.0225, -0.0015];
4 $vpr = [60, 0, 60];
5
6 rotate(a = [-90, 45, 0]) {
7   union() {
8     %rotate extrude(angle = 225.0) {
11    rotate extrude(angle = 225.0) {
14    rotate extrude(angle = 225.0) {
17    }
18  }
19 }

```

