Skills 03 - Gmsh Geometry for FEM

Programmatic CAD and boundary tagging for Exercise 4

1 Gmsh

For all grid-based methods such as the finite element method, we need to discretize the domain into a mesh. In this exercise, we will use Gmsh.

> **Why Gmsh?**
>
> There are many other mesh generators, also proprietary ones, but Gmsh is free, open-source and academically oriented (see, e.g., [1] or this talk), and it has a Python API that we can use to generate meshes directly from our Python code.

> **By the end of this skill**
>
> You should be able to create simple CAD geometry, assign physical tags for a PDE solver, export a `.msh` file, and check whether the mesh and boundary labels match the intended model.

The basic workflow is:

1. Create a geometrical model of the domain.
2. Define physical groups for boundaries, subdomains, or material regions.
3. Optionally define mesh properties such as element size, element type, or boundary layers.
4. Generate the mesh and write it to disk.

> **Kernel choice**
>
> Gmsh has two kernels: the built-in kernel and the OpenCASCADE (OCC) kernel. The built-in kernel is easier to use, but it is limited in terms of the geometrical features it can handle. The OCC kernel is more powerful and can handle more complex geometries, but it is also more complex to use.

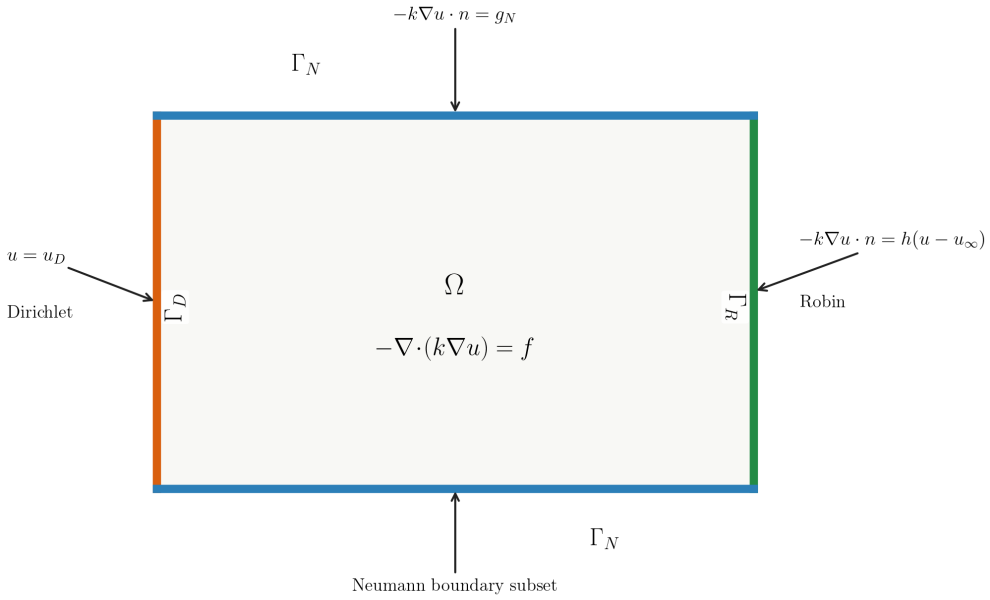


Figure 1: An example geometrical model

2 Basics

The first examples use the OCC kernel because it is the better default once geometries involve boolean operations or imported CAD objects.

> **Minimal OCC checklist**
>
> 1. Create primitives (`addRectangle`, `addDisk`, …).
> 2. Combine with boolean operations (`cut`, `fuse`, `fragment`).
> 3. Synchronize via `gmsh.model.occ.synchronize()`.
> 4. Create physical groups for cells and facets.
> 5. Generate the mesh and write `.msh`.

2.1 Primitive Construction in Gmsh

The next cell uses Gmsh itself to add a rectangle, a standalone line, and two overlapping disks that are fused into one surface. It prints the entity list before and after `gmsh.model.occ.synchronize()`, then meshes the objects so the notebook output shows what changed visually.

> **Synchronization is not optional**
>
> OCC construction calls create CAD objects inside the OCC kernel first. Gmsh only sees them as model entities after `gmsh.model.occ.synchronize()`.

```
import gmsh
if gmsh.isInitialized():
    gmsh.finalize()
```

```python
gmsh.initialize()
gmsh.option.setNumber("General.Terminal", 0)
gmsh.model.add("primitive_demo")

print("Visible model entities before adding primitives:", gmsh.model.getEntities())

# Surface primitive: a rectangle with lower-left corner (x, y, z) and size dx by dy.
rect = gmsh.model.occ.addRectangle(0.0, 0.0, 0.0, 2.0, 1.0)

# Curve primitive: a line needs two point tags first.
p1 = gmsh.model.occ.addPoint(2.55, 0.05, 0.0)
p2 = gmsh.model.occ.addPoint(4.15, 0.95, 0.0)
line = gmsh.model.occ.addLine(p1, p2)

# Boolean primitive operation: fuse two overlapping disks into one surface.
disk_a = gmsh.model.occ.addDisk(5.0, 0.5, 0.0, 0.45, 0.45)
disk_b = gmsh.model.occ.addDisk(5.55, 0.5, 0.0, 0.45, 0.45)
fused_disks, _ = gmsh.model.occ.fuse(
    [(2, disk_a)],
    [(2, disk_b)],
    removeObject=True,
    removeTool=True,
)
fused_disk_surfaces = [tag for dim, tag in fused_disks if dim == 2]

print("OCC calls returned these tags:")
print(f"  rectangle:       dim=2, tag={rect}")
print(f"  line:            dim=1, tag={line}, point tags=({p1}, {p2})")
print(f"  input disks:     dim=2, tags=({disk_a}, {disk_b})")
print(f"  fused disk area: dim=2, tags={fused_disk_surfaces}")
print("Visible model entities before synchronize:", gmsh.model.getEntities())

gmsh.model.occ.synchronize()
print("Visible model entities after synchronize:", gmsh.model.getEntities())

gmsh.option.setNumber("Mesh.MeshSizeMax", 0.12)
gmsh.model.mesh.generate(2)

gmsh.write("primitive_demo.msh")
gmsh.write("primitive_demo.geo_unrolled")
gmsh.write("primitive_demo.brep")

# gmsh.finalize()
```

```
Visible model entities before adding primitives: []
OCC calls returned these tags:
  rectangle:       dim=2, tag=1
  line:            dim=1, tag=5, point tags=(5, 6)
  input disks:     dim=2, tags=(2, 3)
  fused disk area: dim=2, tags=[2]
Visible model entities before synchronize: []
```

```
Visible model entities after synchronize: [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0,
```

```
import os

import matplotlib.pyplot as plt
import matplotlib.tri as mtri
import numpy as np

node_tags, node_coords, _ = gmsh.model.mesh.getNodes()
points = node_coords.reshape(-1, 3)[:, :2]
node_index = {tag: i for i, tag in enumerate(node_tags)}

triangles = []
for etype, _, enodes in zip(*gmsh.model.mesh.getElements(2)):
    if etype == 2:  # 3-node triangle
        triangles.extend(np.array([node_index[t] for t in enodes]).reshape(-1, 3))

segments = []
for etype, _, enodes in zip(*gmsh.model.mesh.getElements(1)):
    if etype == 1:  # 2-node line segment
        segments.extend(np.array([node_index[t] for t in enodes]).reshape(-1, 2))

fig, ax = plt.subplots(figsize=(9, 3.4), constrained_layout=True)
if triangles:
    triang = mtri.Triangulation(points[:, 0], points[:, 1], np.array(triangles))
    ax.triplot(triang, lw=0.45, color="0.35")
    ax.tripcolor(triang, facecolors=np.ones(len(triangles)), alpha=0.18, cmap="Blues")

for segment in segments:
    xy = points[segment]
    ax.plot(xy[:, 0], xy[:, 1], color="#9a4d00", lw=1.3)

ax.text(0.15, 0.82, f"rectangle surface tag {rect}", color="#1f5a85", fontsize=10)
ax.text(2.7, 0.8, f"line curve tag {line}", color="#7b3f00", fontsize=10)
ax.text(4.72, 0.5, f"fused disk surface tag(s) {fused_disk_surfaces}", color="#2f6b24", fontsi
ax.set_aspect("equal")
ax.set_title("Gmsh OCC primitives and a fused disk pair after mesh generation")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.grid(True, color="0.9", lw=0.6)
plt.show()
```

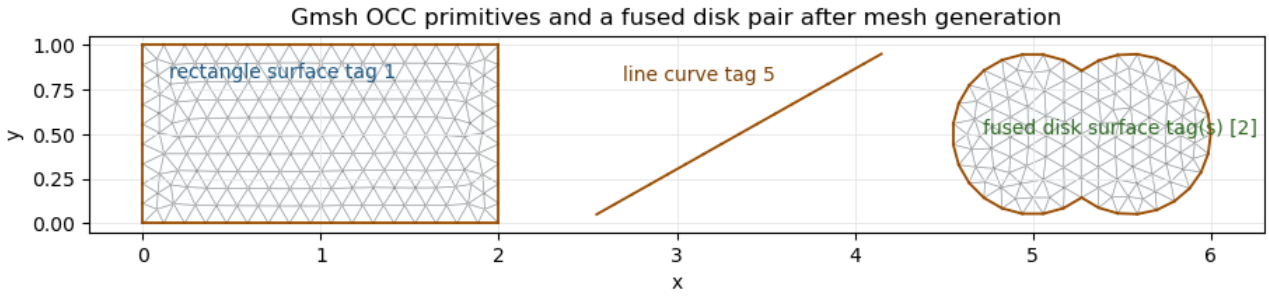


Figure 2: Gmsh OCC primitives and a fused disk pair after mesh generation

2.2 Physical Tags for Use with the `.msh` Format

For FEniCSx/DOLFINx, the mesh alone is not enough. Boundary conditions and subdomain data should travel with the mesh as physical tags.

> **!** Tags are the solver contract
>
> `gmsh.model.addPhysicalGroup()` is how we mark cells and facets for later use. The dimension selects the entity type (`0` for points, `1` for curves, `2` for surfaces), the entity tags select the objects, and the optional physical name makes the file readable.

The next cell tags the rectangle as the domain and all boundary curves as one boundary group. Larger projects usually split the boundary into named parts such as `inlet`, `outlet`, and `wall`.

```
import gmsh

gmsh.initialize()
gmsh.model.add("demo")

rect = gmsh.model.occ.addRectangle(0.0, 0.0, 0.0, 2.0, 1.0)
gmsh.model.occ.synchronize()

boundary_curves = [tag for dim, tag in gmsh.model.getBoundary([(2, rect)], oriented=False)]

gmsh.model.addPhysicalGroup(2, [rect], tag=100)
gmsh.model.setPhysicalName(2, 100, "domain")
gmsh.model.addPhysicalGroup(1, boundary_curves, tag=1)
gmsh.model.setPhysicalName(1, 1, "boundary")

gmsh.model.mesh.generate(2)
gmsh.write("demo_rect_occ.msh")
gmsh.write("demo_rect_occ.geo_unrolled")
gmsh.finalize()
```

```
# show that physical groups are part of the msh file
with open("demo_rect_occ.msh", "r") as f:
    for line in f:
        if line.strip() == "$PhysicalNames":
            print("Found PhysicalNames section in msh file:")
```

```
            print(line.strip())
            # Print the next few lines to show the content of the PhysicalNames section
            for _ in range(5):
                print(f.readline().strip())
            break
```

```
Found PhysicalNames section in msh file:
$PhysicalNames
2
1 1 "boundary"
2 100 "domain"
$EndPhysicalNames
$Entities
```

> **Reading the mesh in DOLFINx**
>
> `dolfinx.io.gmshio.read_from_msh()` reads the mesh and returns the DOLFINx mesh together with cell tags and facet tags. Those tag objects are what boundary conditions and marked integrals will use later.

```
# show that FEniCSx can read the mesh and physical tags
import importlib
import numpy as np
from mpi4py import MPI

gmshio = None
for module_name in ("dolfinx.io.gmshio", "dolfinx.io.gmsh"):
    try:
        gmshio = importlib.import_module(module_name)
        break
    except ImportError:
        pass

if gmshio is None:
    raise ImportError("Could not import a DOLFINx Gmsh I/O module")

mesh_data = gmshio.read_from_msh("demo_rect_occ.msh", MPI.COMM_WORLD, gdim=2)
if hasattr(mesh_data, "mesh"):
    msh, cell_tags, facet_tags = mesh_data.mesh, mesh_data.cell_tags, mesh_data.facet_tags
else:
    msh, cell_tags, facet_tags = mesh_data[:3]
print(f"Mesh dimension: {msh.topology.dim}")
print(f"Cell tag ids: {np.unique(cell_tags.values)}")
print(f"Facet tag ids: {np.unique(facet_tags.values)}")
```

```
Info    : Reading 'demo_rect_occ.msh'...
Info    : 9 entities
Info    : 207 nodes
Info    : 412 elements
```

```
Info    : Done reading 'demo_rect_occ.msh'
Mesh dimension: 2
Cell tag ids: [100]
Facet tag ids: [1]
```

2.3 Kernels: Built-in vs OCC

Gmsh can construct geometry through the built-in `geo` kernel or through the OpenCASCADE `occ` kernel. Both produce meshes, but their exported geometry text looks quite different.

> **OCC export is not a teaching format**
>
> For OCC models, `geo_unrolled` usually points to a CAD-style `.xao` file. This is useful for reproducibility, but it is not meant to be read like a hand-written `.geo` script.

```python
# show geo_unrolled and xao file
with open("demo_rect_occ.geo_unrolled", "r") as f:
    print("Contents of demo_rect_occ.geo_unrolled:")
    for line in f:
        print(line.strip())
with open("demo_rect_occ.geo_unrolled.xao") as f:
    print("\nContents of demo_rect_occ.geo_unrolled.xao:")
    for _ in range(30):
        print(f.readline().strip())
    print("... (xao file continues)")
```

```
Contents of demo_rect_occ.geo_unrolled:
Merge "demo_rect_occ.geo_unrolled.xao";

Contents of demo_rect_occ.geo_unrolled.xao:
<?xml version="1.0" encoding="UTF-8"?>
<XAO version="1.0" author="Gmsh">
<geometry name="demo">
<shape format="BREP"><![CDATA[
CASCADE Topology V1, (c) Matra-Datavision
Locations 0
Curve2ds 0
Curves 4
1 0 0 0 1 0 0
1 2 0 0 0 1 0
1 2 1 0 -1 0 0
1 0 1 0 0 -1 0
Polygon3D 0
PolygonOnTriangulations 0
Surfaces 1
1 1 0.5 0 0 0 1 1 0 -0 -0 1 0
Triangulations 0

TShapes 11
Ve
```

```
1e-07
0 0 0
0 0

0101101
*
Ve
1e-07
2 0 0
0 0
... (xao file continues)
```

2.4 Built-in Kernel

The built-in kernel is more explicit: points, lines, curve loops, and plane surfaces are written in a way that maps directly to the `.geo` language.

> **When to use it**
>
> Use `gmsh.model.geo` for small instructional examples and simple planar geometries. Switch to `gmsh.model.occ` when boolean operations or imported CAD geometry become important.

```python
import gmsh

if gmsh.isInitialized():
    gmsh.finalize()

gmsh.initialize()
gmsh.model.add("geo_rectangle_demo")

lc = 0.1  # mesh size at points

p1 = gmsh.model.geo.addPoint(0.0, 0.0, 0.0, lc)
p2 = gmsh.model.geo.addPoint(1.0, 0.0, 0.0, lc)
p3 = gmsh.model.geo.addPoint(1.0, 1.0, 0.0, lc)
p4 = gmsh.model.geo.addPoint(0.0, 1.0, 0.0, lc)

l1 = gmsh.model.geo.addLine(p1, p2)
l2 = gmsh.model.geo.addLine(p2, p3)
l3 = gmsh.model.geo.addLine(p3, p4)
l4 = gmsh.model.geo.addLine(p4, p1)

curve_loop = gmsh.model.geo.addCurveLoop([l1, l2, l3, l4])
surface = gmsh.model.geo.addPlaneSurface([curve_loop])

# Sync CAD kernel data to the Gmsh model before meshing/exporting.
gmsh.model.geo.synchronize()

gmsh.model.addPhysicalGroup(2, [surface], tag=100)
gmsh.model.setPhysicalName(2, 100, "unit_square")
gmsh.model.addPhysicalGroup(1, [l1, l2, l3, l4], tag=1)
```

```
gmsh.model.setPhysicalName(1, 1, "boundary")

gmsh.model.mesh.generate(2)
gmsh.write("demo_rect.msh")
gmsh.write("demo_rect.geo_unrolled")

gmsh.finalize()
```

```
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 0.000139792s, CPU 0.000216s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.00135437s, CPU 0.001951s)
Info    : 142 nodes 286 elements
Info    : Writing 'demo_rect.msh'...
Info    : Done writing 'demo_rect.msh'
Info    : Writing 'demo_rect.geo_unrolled'...
Info    : Done writing 'demo_rect.geo_unrolled'
```

2.5 `geo_unrolled` File

With the built-in kernel, the unrolled geometry is much easier to inspect. This makes it a useful bridge between Python API calls and hand-written `.geo` files.

> **Reading the output**
>
> Look for the same construction pattern as in the Python code: characteristic length, points, lines, a curve loop, a plane surface, and physical groups.

```
with open("demo_rect.geo_unrolled", "r") as f:
    geo_content = f.read()
print("\nContents of demo_rect.geo_unrolled:\n")
print(geo_content)
```

```
Contents of demo_rect.geo_unrolled:

cl__1 = 0.1;
Point(1) = {0, 0, 0, cl__1};
Point(2) = {1, 0, 0, cl__1};
Point(3) = {1, 1, 0, cl__1};
Point(4) = {0, 1, 0, cl__1};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
```

```
Line(4) = {4, 1};
Curve Loop(1) = {1, 2, 3, 4};
Plane Surface(1) = {1};
Physical Curve("boundary") = {1, 2, 3, 4};
Physical Surface("unit_square") = {1};
```

You can also mesh this `geo` or `geo_unrolled` file directly with the `gmsh` command-line tool:

```
gmsh -2 demo_rect.geo_unrolled -o output_filename.msh
```

2.6 Open the Gmsh FLTK Viewer (Interactive)

Use this for quick visual inspection of entities, physical groups, and mesh quality.

> Interactive cell
>
> The viewer opens a GUI window and blocks the notebook until the window is closed. Keep the cell disabled for automated rendering and enable it only when working locally.

```
import gmsh

if gmsh.isInitialized():
    gmsh.finalize()

gmsh.initialize()
gmsh.open("bridge_holes.msh")

# Optional display tweaks
gmsh.option.setNumber("Mesh.SurfaceEdges", 1)
gmsh.option.setNumber("Mesh.Lines", 1)

# Blocks until you close the FLTK window
# avoid this in automated workflows, but it's useful for interactive exploration and debugging
# gmsh.fltk.run()
# gmsh.finalize()
```

```
Info    : Reading 'bridge_holes.msh'...
Info    : 15 entities
Info    : 766 nodes
Info    : 1536 elements
Info    : Done reading 'bridge_holes.msh'
```

The interactive viewer should look similar to this:

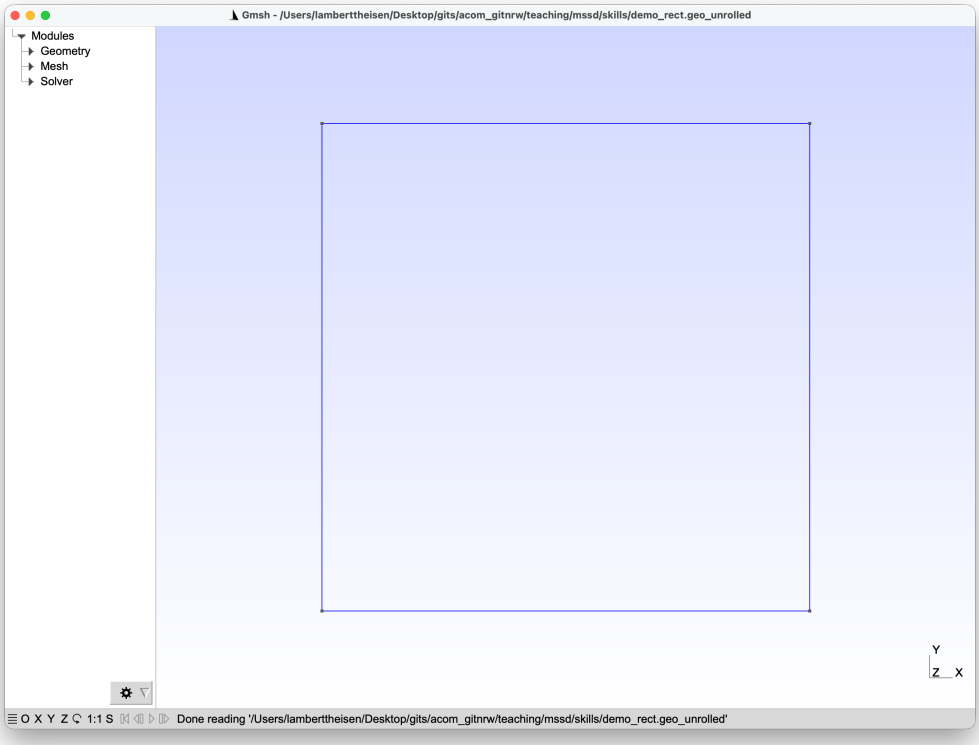


Figure 3: GMSH View

3 Advanced Meshing

3.1 Reading a `.geo` File with parameters

The Python API is convenient for parametrized scripts, but many Gmsh examples use the native `.geo` language. The ideas are the same: points define geometry, curves connect points, curve loops define closed boundaries, surfaces define the 2D domain, and physical groups carry the tags used later in FEniCSx.

Here we write a small `.geo` file from the notebook so that students can inspect the text and immediately see the mesh it creates.

> **Source of the geometry**
>
> The example `.geo` file comes from the fenicsR13 solver [2]. More mesh examples are available in this repository.

```
from pathlib import Path

geo_text = r'''
// Command line parameter: try p = 0, 1, 2 to refine the mesh.
If(!Exists(p))
  p = 0;
EndIf
```

```
// Settings
res = 100;
Mesh.MeshSizeMax = 1.0 * 2^(-p);
Mesh.MshFileVersion = 2.0;

// Parameters
R1 = 0.5;
R2 = 2.0;

l = 1.0;
L = 2*l;

Point(1) = {0,  0, 0, res};
Point(2) = {l,  0, 0, res};
Point(3) = {-l, 0, 0, res};

Point(1010) = {l,   R1, 0, res};
Point(1011) = {-l,  R1, 0, res};
Point(1012) = {-l, -R1, 0, res};
Point(1013) = {l,  -R1, 0, res};

Point(1020) = {l,   R2, 0, res};
Point(1021) = {-l,  R2, 0, res};
Point(1022) = {-l, -R2, 0, res};
Point(1023) = {l,  -R2, 0, res};

Line(1010) = {1010,1011};
Circle(1011) = {1011,3,1012};
Line(1012) = {1012,1013};
Circle(1013) = {1013,2,1010};

Line(1020) = {1020,1021};
Circle(1021) = {1021,3,1022};
Line(1022) = {1022,1023};
Circle(1023) = {1023,2,1020};

// Physical curve tags are what the PDE code will see.
Physical Curve("it",3010) = {1010};
Physical Curve("il",3011) = {1011};
Physical Curve("ib",3012) = {1012};
Physical Curve("ir",3013) = {1013};

Physical Curve("ot",3020) = {1020};
Physical Curve("ol",3021) = {1021};
Physical Curve("ob",3022) = {1022};
Physical Curve("or",3023) = {1023};

Curve Loop(4010) = {1010,1011,1012,1013};
Curve Loop(4020) = {1020,1021,1022,1023};
Plane Surface(4000) = {4020,4010};
Physical Surface("mesh",4000) = {4000};
```

```
'''

Path("capsule_annulus.geo").write_text(geo_text)
print(geo_text)
```

```
// Command line parameter: try p = 0, 1, 2 to refine the mesh.
If(!Exists(p))
  p = 0;
EndIf

// Settings
res = 100;
Mesh.MeshSizeMax = 1.0 * 2^(-p);
Mesh.MshFileVersion = 2.0;

// Parameters
R1 = 0.5;
R2 = 2.0;

l = 1.0;
L = 2*l;

Point(1) = {0,  0, 0, res};
Point(2) = {l,  0, 0, res};
Point(3) = {-l, 0, 0, res};

Point(1010) = {l,   R1, 0, res};
Point(1011) = {-l,  R1, 0, res};
Point(1012) = {-l, -R1, 0, res};
Point(1013) = {l,  -R1, 0, res};

Point(1020) = {l,   R2, 0, res};
Point(1021) = {-l,  R2, 0, res};
Point(1022) = {-l, -R2, 0, res};
Point(1023) = {l,  -R2, 0, res};

Line(1010) = {1010,1011};
Circle(1011) = {1011,3,1012};
Line(1012) = {1012,1013};
Circle(1013) = {1013,2,1010};

Line(1020) = {1020,1021};
Circle(1021) = {1021,3,1022};
Line(1022) = {1022,1023};
Circle(1023) = {1023,2,1020};

// Physical curve tags are what the PDE code will see.
Physical Curve("it",3010) = {1010};
Physical Curve("il",3011) = {1011};
Physical Curve("ib",3012) = {1012};
```

```
Physical Curve("ir",3013) = {1013};

Physical Curve("ot",3020) = {1020};
Physical Curve("ol",3021) = {1021};
Physical Curve("ob",3022) = {1022};
Physical Curve("or",3023) = {1023};

Curve Loop(4010) = {1010,1011,1012,1013};
Curve Loop(4020) = {1020,1021,1022,1023};
Plane Surface(4000) = {4020,4010};
Physical Surface("mesh",4000) = {4000};
```

Before meshing, open the `.geo` file as geometry and plot the CAD curves. No call to `gmsh.model.mesh.generate(2)` happens in this cell.

> **Geometry before mesh**
>
> This separates two checks that are often mixed up: first verify that the CAD curves and physical names are correct, then generate and inspect the mesh.

```python
def read_triangles_from_msh(filename):
    if gmsh.isInitialized():
        gmsh.finalize()

    gmsh.initialize()
    gmsh.model.add("view_mesh")
    gmsh.merge(filename)

    node_tags, node_coords, _ = gmsh.model.mesh.getNodes()
    xy = node_coords.reshape(-1, 3)[:, :2]
    node_to_local = {int(tag): i for i, tag in enumerate(node_tags)}

    triangles = []
    for _, entity_tag in gmsh.model.getEntities(2):
        element_types, _, element_nodes = gmsh.model.mesh.getElements(2, entity_tag)
        for e_type, e_nodes in zip(element_types, element_nodes):
            # type 2: linear triangles, type 9: quadratic triangles
            if e_type == 2:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 3)
            elif e_type == 9:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 6)[:, :3]
            else:
                continue
            triangles.append(np.vectorize(node_to_local.get)(conn))

    gmsh.finalize()

    if len(triangles) == 0:
        raise RuntimeError(f"No triangular elements found in {filename}")

    tri = np.vstack(triangles)
```

```
    return xy, tri


def plot_msh(ax, filename, title):
    xy, tri = read_triangles_from_msh(filename)
    triang = mtri.Triangulation(xy[:, 0], xy[:, 1], tri)
    ax.triplot(triang, lw=0.35, color="0.2")
    ax.set_aspect("equal")
    ax.set_title(title)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
```

```
def sample_gmsh_curve(tag, n=120):
    lower, upper = gmsh.model.getParametrizationBounds(1, tag)
    params = np.linspace(float(lower[0]), float(upper[0]), n)
    points = np.array([gmsh.model.getValue(1, tag, [u]) for u in params])
    return points[:, :2]


def plot_current_gmsh_geometry(ax, title, show_entity_tags=True):
    cmap = plt.get_cmap("tab10")
    physical_curve_to_color = {}
    physical_curve_to_label = {}

    for color_i, (dim, ptag) in enumerate(gmsh.model.getPhysicalGroups(1)):
        name = gmsh.model.getPhysicalName(dim, ptag) or f"physical {ptag}"
        color = cmap(color_i % 10)
        for curve_tag in gmsh.model.getEntitiesForPhysicalGroup(dim, ptag):
            physical_curve_to_color[curve_tag] = color
            physical_curve_to_label[curve_tag] = f"{ptag}: {name}"

    shown_labels = set()
    for _, curve_tag in gmsh.model.getEntities(1):
        xy = sample_gmsh_curve(curve_tag)
        color = physical_curve_to_color.get(curve_tag, "0.25")
        label = physical_curve_to_label.get(curve_tag)
        ax.plot(
            xy[:, 0],
            xy[:, 1],
            lw=2.0,
            color=color,
            label=label if label and label not in shown_labels else None,
        )
        if label:
            shown_labels.add(label)

        if show_entity_tags:
            mid = xy[len(xy) // 2]
            ax.text(mid[0], mid[1], str(curve_tag), fontsize=8, ha="center", va="center")

    point_xy = []
```

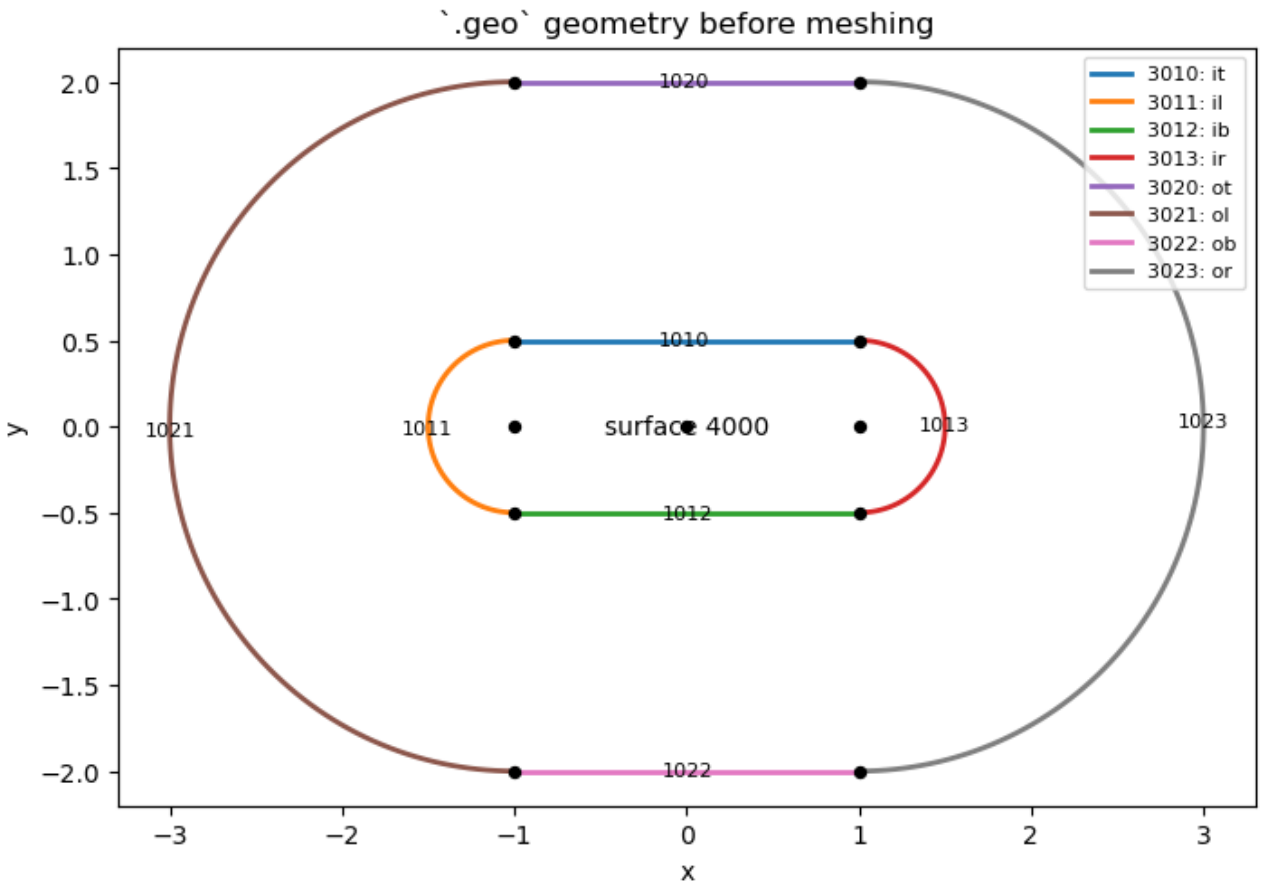
```
    for _, point_tag in gmsh.model.getEntities(0):
        point_xy.append(gmsh.model.getValue(0, point_tag, []))
    if point_xy:
        point_xy = np.array(point_xy)[:, :2]
        ax.scatter(point_xy[:, 0], point_xy[:, 1], s=18, color="black", zorder=3)

    for _, surface_tag in gmsh.model.getEntities(2):
        xmin, ymin, _, xmax, ymax, _ = gmsh.model.getBoundingBox(2, surface_tag)
        x, y = 0.5 * (xmin + xmax), 0.5 * (ymin + ymax)
        ax.text(x, y, f"surface {surface_tag}", fontsize=10, ha="center", va="center")

    ax.set_aspect("equal")
    ax.set_title(title)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    if shown_labels:
        ax.legend(loc="upper right", fontsize=8, frameon=True)
```

```
def plot_msh_with_physical_tags(ax, filename, title):
    if gmsh.isInitialized():
        gmsh.finalize()

    gmsh.initialize()
    gmsh.model.add("tag_view")
    gmsh.merge(filename)

    node_tags, node_coords, _ = gmsh.model.mesh.getNodes()
    xy = node_coords.reshape(-1, 3)[:, :2]
    node_to_local = {int(tag): i for i, tag in enumerate(node_tags)}

    triangles = []
    for _, entity_tag in gmsh.model.getEntities(2):
        element_types, _, element_nodes = gmsh.model.mesh.getElements(2, entity_tag)
        for e_type, e_nodes in zip(element_types, element_nodes):
            if e_type == 2:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 3)
            elif e_type == 9:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 6)[:, :3]
            else:
                continue
            triangles.append(np.vectorize(node_to_local.get)(conn))

    if triangles:
        tri = np.vstack(triangles)
        triang = mtri.Triangulation(xy[:, 0], xy[:, 1], tri)
        ax.triplot(triang, lw=0.25, color="0.75")

    cmap = plt.get_cmap("tab10")
    color_i = 0
    for dim, ptag in gmsh.model.getPhysicalGroups():
        if dim != 1:
```

```
            continue
        pname = gmsh.model.getPhysicalName(dim, ptag) or f"tag {ptag}"
        color = cmap(color_i % 10)
        color_i += 1

        shown_label = False
        for ent in gmsh.model.getEntitiesForPhysicalGroup(dim, ptag):
            etypes, _, enodes = gmsh.model.mesh.getElements(1, ent)
            for et, e in zip(etypes, enodes):
                nper = gmsh.model.mesh.getElementProperties(et)[3]
                conn = np.array(e, dtype=np.int64).reshape(-1, nper)
                for row in conn:
                    n0, n1 = int(row[0]), int(row[-1])
                    i0, i1 = node_to_local[n0], node_to_local[n1]
                    ax.plot(
                        [xy[i0, 0], xy[i1, 0]],
                        [xy[i0, 1], xy[i1, 1]],
                        color=color,
                        lw=2.0,
                        label=f"{ptag}: {pname}" if not shown_label else None,
                    )
                    shown_label = True

    gmsh.finalize()

    ax.set_aspect("equal")
    ax.set_title(title)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend(loc="upper right", fontsize=9, frameon=True)
```

```
if gmsh.isInitialized():
    gmsh.finalize()

gmsh.initialize()
gmsh.open("capsule_annulus.geo")

fig, ax = plt.subplots(figsize=(7, 7), constrained_layout=True)
plot_current_gmsh_geometry(ax, "`.geo` geometry before meshing")
plt.show()

gmsh.finalize()
```

```
Info    : Reading 'capsule_annulus.geo'...
Info    : Done reading 'capsule_annulus.geo'
```



The parameter `p` controls the largest allowed element size. In the command line this would be `gmsh capsule_annulus.geo -2 -setnumber p 1`. In the notebook we read the `.geo` file and then set the same mesh-size option explicitly, because this makes the visual comparison independent of command-line execution.

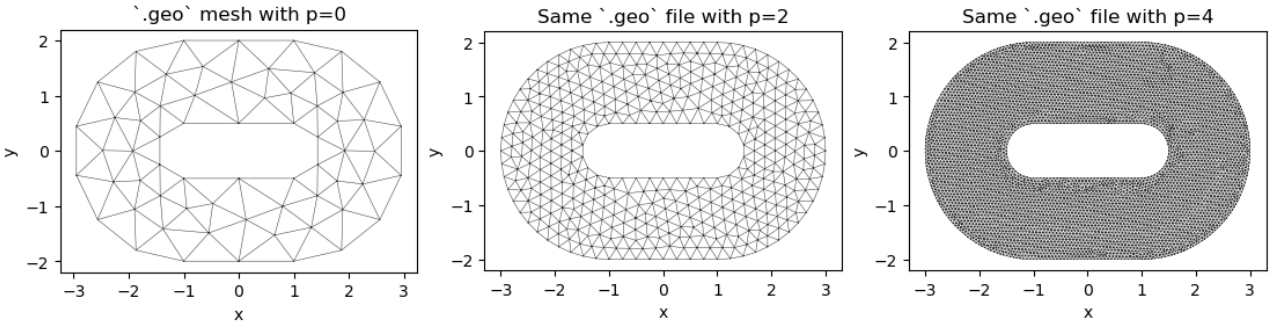
> **Refinement convention**
>
> Here larger `p` means smaller elements because the notebook uses `1.0 * 2 ** (-p)` as the maximum mesh size. The three plots below show the same geometry at increasing resolution.

```
def mesh_from_geo(filename, p=0, msh_filename=None):
    if gmsh.isInitialized():
        gmsh.finalize()

    if msh_filename is None:
        msh_filename = f"{Path(filename).stem}_p{p}.msh"

    gmsh.initialize()
    gmsh.open(filename)
    gmsh.option.setNumber("Mesh.MeshSizeMax", 1.0 * 2 ** (-p))
    gmsh.model.mesh.generate(2)
    gmsh.write(msh_filename)
    gmsh.finalize()
    return msh_filename
```

```
geo_mesh_p0 = mesh_from_geo("capsule_annulus.geo", p=0)
geo_mesh_p2 = mesh_from_geo("capsule_annulus.geo", p=2)
geo_mesh_p4 = mesh_from_geo("capsule_annulus.geo", p=4)

fig, axes = plt.subplots(1, 3, figsize=(11, 5), constrained_layout=True)
plot_msh(axes[0], geo_mesh_p0, "`.geo` mesh with p=0")
plot_msh(axes[1], geo_mesh_p2, "Same `.geo` file with p=2")
plot_msh(axes[2], geo_mesh_p4, "Same `.geo` file with p=4")
plt.show()
```

```
Info    : Reading 'capsule_annulus.geo'...
Info    : Done reading 'capsule_annulus.geo'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1010 (Line)
Info    : [ 20%] Meshing curve 1011 (Circle)
Info    : [ 30%] Meshing curve 1012 (Line)
Info    : [ 40%] Meshing curve 1013 (Circle)
Info    : [ 60%] Meshing curve 1020 (Line)
Info    : [ 70%] Meshing curve 1021 (Circle)
Info    : [ 80%] Meshing curve 1022 (Line)
Info    : [ 90%] Meshing curve 1023 (Circle)
Info    : Done meshing 1D (Wall 0.000333792s, CPU 0.000442s)
Info    : Meshing 2D...
Info    : Meshing surface 4000 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.000708208s, CPU 0.001054s)
Info    : 67 nodes 139 elements
Info    : Writing 'capsule_annulus_p0.msh'...
Info    : Done writing 'capsule_annulus_p0.msh'
Info    : Reading 'capsule_annulus.geo'...
Info    : Done reading 'capsule_annulus.geo'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1010 (Line)
Info    : [ 20%] Meshing curve 1011 (Circle)
Info    : [ 30%] Meshing curve 1012 (Line)
Info    : [ 40%] Meshing curve 1013 (Circle)
Info    : [ 60%] Meshing curve 1020 (Line)
Info    : [ 70%] Meshing curve 1021 (Circle)
Info    : [ 80%] Meshing curve 1022 (Line)
Info    : [ 90%] Meshing curve 1023 (Circle)
Info    : Done meshing 1D (Wall 0.000259791s, CPU 0.000353s)
Info    : Meshing 2D...
Info    : Meshing surface 4000 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.00462625s, CPU 0.007416s)
Info    : 445 nodes 895 elements
Info    : Writing 'capsule_annulus_p2.msh'...
Info    : Done writing 'capsule_annulus_p2.msh'
Info    : Reading 'capsule_annulus.geo'...
Info    : Done reading 'capsule_annulus.geo'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1010 (Line)
```

```
Info    : [ 20%] Meshing curve 1011 (Circle)
Info    : [ 30%] Meshing curve 1012 (Line)
Info    : [ 40%] Meshing curve 1013 (Circle)
Info    : [ 60%] Meshing curve 1020 (Line)
Info    : [ 70%] Meshing curve 1021 (Circle)
Info    : [ 80%] Meshing curve 1022 (Line)
Info    : [ 90%] Meshing curve 1023 (Circle)
Info    : Done meshing 1D (Wall 0.0003335s, CPU 0.000589s)
Info    : Meshing 2D...
Info    : Meshing surface 4000 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.0655314s, CPU 0.106388s)
Info    : 5647 nodes 11299 elements
Info    : Writing 'capsule_annulus_p4.msh'...
Info    : Done writing 'capsule_annulus_p4.msh'
Info    : Reading 'capsule_annulus_p0.msh'...
Info    : 64 nodes
Info    : 128 elements
Info    : Done reading 'capsule_annulus_p0.msh'
Info    : Reading 'capsule_annulus_p2.msh'...
Info    : 442 nodes
Info    : 884 elements
Info    : Done reading 'capsule_annulus_p2.msh'
Info    : Reading 'capsule_annulus_p4.msh'...
Info    : 5644 nodes
Info    : 11288 elements
Info    : Done reading 'capsule_annulus_p4.msh'
```



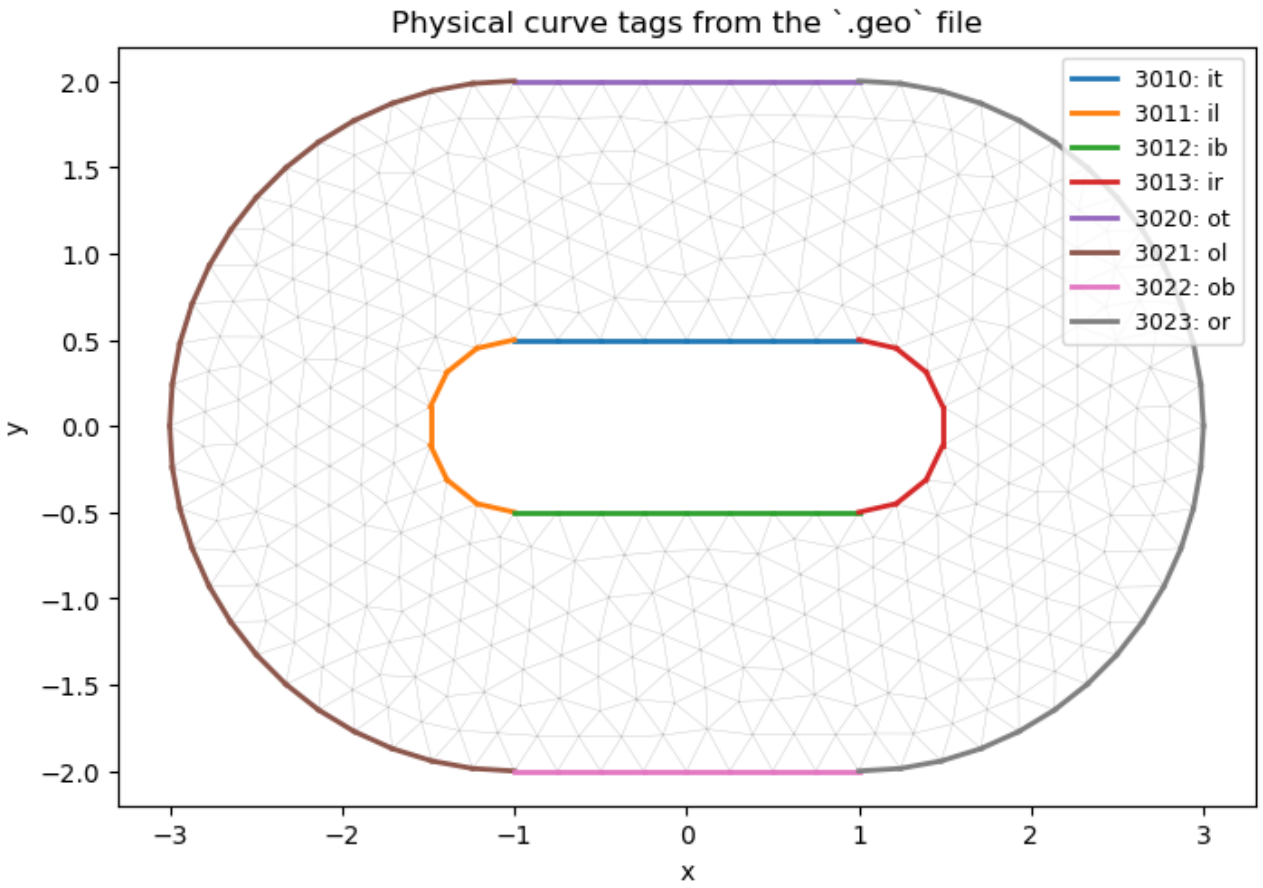
The plot below overlays the physical curve names from the `.geo` file. This is the important check before using the mesh in a PDE solver: the labels must match the intended boundary conditions.

> **Inspecting tags interactively**
>
> Alternatively, you can hover over the curves in the Gmsh FLTK viewer to see the tags and physical group names.

```
fig, ax = plt.subplots(figsize=(7, 7), constrained_layout=True)
plot_msh_with_physical_tags(ax, geo_mesh_p2, "Physical curve tags from the `.geo` file")
plt.show()
```

```
Info    : Reading 'capsule_annulus_p2.msh'...
```

```
Info    : 442 nodes
Info    : 884 elements
Info    : Done reading 'capsule_annulus_p2.msh'
```



3.2 Mesh-Size Fields in the Python API

Global mesh sizes are often too blunt. A mesh-size field lets us ask Gmsh for smaller elements near selected entities and larger elements farther away. The next function refines near the boundary of a square using a `Distance` field followed by a `Threshold` field.

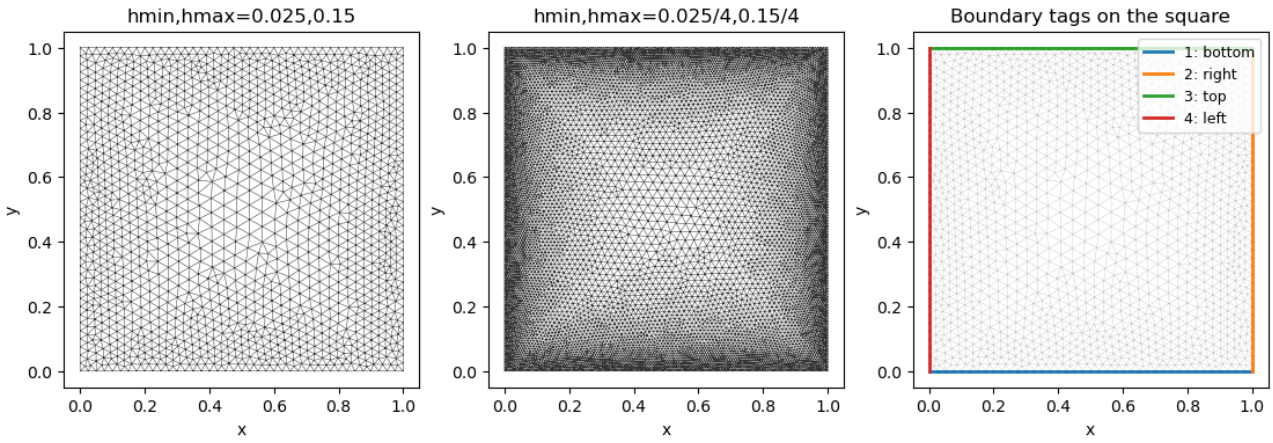
> **Field recipe**
>
> Important details to notice:
>
> - `CurvesList` selects the boundary curves that generate the distance field.
> - `DistMin` and `DistMax` describe the transition zone from fine to coarse mesh.
> - `MeshSizeExtendFromBoundary`, `MeshSizeFromPoints`, and `MeshSizeFromCurvature` are turned off so that the background field is the main size rule.

```
def create_square_with_boundary_refinement(filename="square_boundary_refined.msh", hmin=0.025,
    if gmsh.isInitialized():
        gmsh.finalize()

    gmsh.initialize()
```

```
    gmsh.model.add("square_boundary_refined")

    rect = gmsh.model.occ.addRectangle(0, 0, 0, 1, 1)
    gmsh.model.occ.synchronize()

    boundary_lines = gmsh.model.getBoundary([(2, rect)], oriented=False)
    boundary_line_tags = [tag for dim, tag in boundary_lines if dim == 1]

    gmsh.model.mesh.field.add("Distance", 1)
    gmsh.model.mesh.field.setNumbers(1, "CurvesList", boundary_line_tags)
    gmsh.model.mesh.field.setNumber(1, "Sampling", max(2.0 / hmin, 100))

    gmsh.model.mesh.field.add("Threshold", 2)
    gmsh.model.mesh.field.setNumber(2, "InField", 1)
    gmsh.model.mesh.field.setNumber(2, "SizeMin", hmin)
    gmsh.model.mesh.field.setNumber(2, "SizeMax", hmax)
    gmsh.model.mesh.field.setNumber(2, "DistMin", 3 * hmin)
    gmsh.model.mesh.field.setNumber(2, "DistMax", 16 * hmax)
    gmsh.model.mesh.field.setAsBackgroundMesh(2)

    gmsh.option.setNumber("Mesh.MeshSizeExtendFromBoundary", 0)
    gmsh.option.setNumber("Mesh.MeshSizeFromPoints", 0)
    gmsh.option.setNumber("Mesh.MeshSizeFromCurvature", 0)

    gmsh.model.addPhysicalGroup(2, [rect], tag=1)
    gmsh.model.setPhysicalName(2, 1, "Omega")

    labels = {"bottom": 1, "right": 2, "top": 3, "left": 4}
    for dim, tag in boundary_lines:
        x, y, _ = gmsh.model.occ.getCenterOfMass(dim, tag)
        if abs(y - 0.0) < 1e-12:
            name = "bottom"
        elif abs(x - 1.0) < 1e-12:
            name = "right"
        elif abs(y - 1.0) < 1e-12:
            name = "top"
        else:
            name = "left"
        gmsh.model.addPhysicalGroup(1, [tag], tag=labels[name])
        gmsh.model.setPhysicalName(1, labels[name], name)

    gmsh.model.mesh.generate(2)
    gmsh.write(filename)
    gmsh.finalize()
    return filename


square_field_mesh = create_square_with_boundary_refinement(filename="square_field_mesh.msh", h
square_field_mesh2 = create_square_with_boundary_refinement(filename="square_field_mesh2.msh",

fig, axes = plt.subplots(1, 3, figsize=(11, 5), constrained_layout=True)
```

```
plot_msh(axes[0], square_field_mesh, "hmin,hmax=0.025,0.15")
plot_msh(axes[1], square_field_mesh2, "hmin,hmax=0.025/4,0.15/4")
plot_msh_with_physical_tags(axes[2], square_field_mesh, "Boundary tags on the square")
plt.show()
```

```
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 0.0001985s, CPU 0.000335s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.02453s, CPU 0.036586s)
Info    : 1483 nodes 2968 elements
Info    : Writing 'square_field_mesh.msh'...
Info    : Done writing 'square_field_mesh.msh'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 0.000312417s, CPU 0.000447s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.184211s, CPU 0.298308s)
Info    : 10843 nodes 21688 elements
Info    : Writing 'square_field_mesh2.msh'...
Info    : Done writing 'square_field_mesh2.msh'
Info    : Reading 'square_field_mesh.msh'...
Info    : 9 entities
Info    : 1483 nodes
Info    : 2964 elements
Info    : Done reading 'square_field_mesh.msh'
Info    : Reading 'square_field_mesh2.msh'...
Info    : 9 entities
Info    : 10843 nodes
Info    : 21684 elements
Info    : Done reading 'square_field_mesh2.msh'
Info    : Reading 'square_field_mesh.msh'...
Info    : 9 entities
Info    : 1483 nodes
Info    : 2964 elements
Info    : Done reading 'square_field_mesh.msh'
```



3.3 Quadrilateral and Structured Meshes

Gmsh is not limited to triangular meshes. For many rectangular or logically rectangular geometries, quadrilateral elements are also possible. The usual Gmsh word for converting triangles into quads is **recombination**.

> Two separate ideas
>
> `setRecombine()` asks Gmsh for quadrilateral elements. `setTransfiniteCurve()` and `setTransfiniteSurface()` additionally impose a structured, grid-like node layout.

The next example compares the same square as an unstructured triangular mesh, a recombined quadrilateral mesh, a structured transfinite quadrilateral mesh, and a structured transfinite triangular mesh.

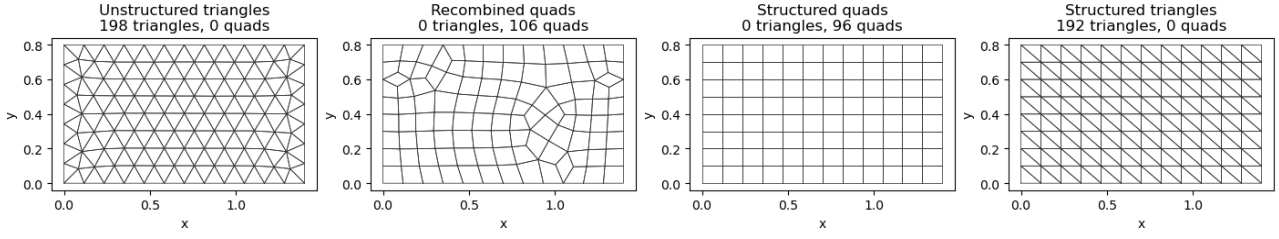
```
def create_square_mesh(filename, recombine=False, structured=False, nx=13, ny=9, lc=0.12):
    if gmsh.isInitialized():
        gmsh.finalize()

    gmsh.initialize()
    gmsh.model.add(Path(filename).stem)

    p1 = gmsh.model.geo.addPoint(0.0, 0.0, 0.0, lc)
    p2 = gmsh.model.geo.addPoint(1.4, 0.0, 0.0, lc)
    p3 = gmsh.model.geo.addPoint(1.4, 0.8, 0.0, lc)
    p4 = gmsh.model.geo.addPoint(0.0, 0.8, 0.0, lc)

    l1 = gmsh.model.geo.addLine(p1, p2)
    l2 = gmsh.model.geo.addLine(p2, p3)
    l3 = gmsh.model.geo.addLine(p3, p4)
    l4 = gmsh.model.geo.addLine(p4, p1)
    loop = gmsh.model.geo.addCurveLoop([l1, l2, l3, l4])
    surface = gmsh.model.geo.addPlaneSurface([loop])

    if structured:
        gmsh.model.geo.mesh.setTransfiniteCurve(l1, nx)
        gmsh.model.geo.mesh.setTransfiniteCurve(l3, nx)
```

```
        gmsh.model.geo.mesh.setTransfiniteCurve(l2, ny)
        gmsh.model.geo.mesh.setTransfiniteCurve(l4, ny)
        gmsh.model.geo.mesh.setTransfiniteSurface(surface)

    if recombine:
        gmsh.model.geo.mesh.setRecombine(2, surface)

    gmsh.model.geo.synchronize()
    gmsh.model.addPhysicalGroup(2, [surface], tag=1)
    gmsh.model.setPhysicalName(2, 1, "Omega")
    gmsh.model.addPhysicalGroup(1, [l1, l2, l3, l4], tag=2)
    gmsh.model.setPhysicalName(1, 2, "boundary")

    gmsh.option.setNumber("Mesh.RecombinationAlgorithm", 1)
    gmsh.model.mesh.generate(2)
    gmsh.write(filename)
    gmsh.finalize()
    return filename


def read_triangles_and_quads_from_msh(filename):
    if gmsh.isInitialized():
        gmsh.finalize()

    gmsh.initialize()
    gmsh.model.add("view_quads")
    gmsh.merge(filename)

    node_tags, node_coords, _ = gmsh.model.mesh.getNodes()
    xy = node_coords.reshape(-1, 3)[:, :2]
    node_to_local = {int(tag): i for i, tag in enumerate(node_tags)}

    cells = []
    counts = {"triangles": 0, "quads": 0}
    for _, entity_tag in gmsh.model.getEntities(2):
        element_types, _, element_nodes = gmsh.model.mesh.getElements(2, entity_tag)
        for e_type, e_nodes in zip(element_types, element_nodes):
            if e_type == 2:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 3)
                counts["triangles"] += len(conn)
            elif e_type == 3:
                conn = np.array(e_nodes, dtype=np.int64).reshape(-1, 4)
                counts["quads"] += len(conn)
            else:
                continue
            cells.extend(np.vectorize(node_to_local.get)(conn))

    gmsh.finalize()
    return xy, cells, counts
```

```
def plot_cells(ax, filename, title):
    xy, cells, counts = read_triangles_and_quads_from_msh(filename)
    for cell in cells:
        polygon = xy[np.r_[cell, cell[0]]]
        ax.plot(polygon[:, 0], polygon[:, 1], lw=0.55, color="0.2")
    ax.set_aspect("equal")
    ax.set_title(f"{title}\n{counts['triangles']} triangles, {counts['quads']} quads")
    ax.set_xlabel("x")
    ax.set_ylabel("y")


tri_square = create_square_mesh("square_triangles.msh")
quad_square = create_square_mesh("square_recombined_quads.msh", recombine=True)
structured_square = create_square_mesh("square_structured_quads.msh", structured=True, recombi
structured_triangles = create_square_mesh("square_structured_triangles.msh", structured=True)

fig, axes = plt.subplots(1, 4, figsize=(14, 4.2), constrained_layout=True)
plot_cells(axes[0], tri_square, "Unstructured triangles")
plot_cells(axes[1], quad_square, "Recombined quads")
plot_cells(axes[2], structured_square, "Structured quads")
plot_cells(axes[3], structured_triangles, "Structured triangles")
plt.show()
```

```
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 0.000110375s, CPU 0.000209s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Frontal-Delaunay)
Info    : Done meshing 2D (Wall 0.00108654s, CPU 0.001635s)
Info    : 119 nodes 240 elements
Info    : Writing 'square_triangles.msh'...
Info    : Done writing 'square_triangles.msh'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 0.000107666s, CPU 0.000161s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Plane, Frontal-Delaunay)
Info    : Blossom: 301 internal 40 closed
Info    : Blossom recombination completed (Wall 0.000842375s, CPU 0.00121s): 106 quads, 0 tria
Info    : Done meshing 2D (Wall 0.00201304s, CPU 0.002895s)
Info    : 127 nodes 150 elements
Info    : Writing 'square_recombined_quads.msh'...
Info    : Done writing 'square_recombined_quads.msh'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
```

```
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 8.4542e-05s, CPU 0.00011s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Transfinite)
Info    : Done meshing 2D (Wall 2.3541e-05s, CPU 3.1e-05s)
Info    : 117 nodes 140 elements
Info    : Writing 'square_structured_quads.msh'...
Info    : Done writing 'square_structured_quads.msh'
Info    : Meshing 1D...
Info    : [  0%] Meshing curve 1 (Line)
Info    : [ 30%] Meshing curve 2 (Line)
Info    : [ 60%] Meshing curve 3 (Line)
Info    : [ 80%] Meshing curve 4 (Line)
Info    : Done meshing 1D (Wall 8.075e-05s, CPU 0.000119s)
Info    : Meshing 2D...
Info    : Meshing surface 1 (Transfinite)
Info    : Done meshing 2D (Wall 2.0167e-05s, CPU 3.1e-05s)
Info    : 117 nodes 236 elements
Info    : Writing 'square_structured_triangles.msh'...
Info    : Done writing 'square_structured_triangles.msh'
Info    : Reading 'square_triangles.msh'...
Info    : 9 entities
Info    : 119 nodes
Info    : 236 elements
Info    : Done reading 'square_triangles.msh'
Info    : Reading 'square_recombined_quads.msh'...
Info    : 9 entities
Info    : 127 nodes
Info    : 146 elements
Info    : Done reading 'square_recombined_quads.msh'
Info    : Reading 'square_structured_quads.msh'...
Info    : 9 entities
Info    : 117 nodes
Info    : 136 elements
Info    : Done reading 'square_structured_quads.msh'
Info    : Reading 'square_structured_triangles.msh'...
Info    : 9 entities
Info    : 117 nodes
Info    : 232 elements
Info    : Done reading 'square_structured_triangles.msh'
```



3.4 3D Example

The same `.geo` language can describe 3D construction. This example sweeps two cross-sections along a spline wire to create a curved pipe-like geometry.

> **From 2D to 3D**
>
> The important new idea is `Extrude ... Using Wire`: instead of filling a plane surface, Gmsh moves a surface along a path and creates a volume-like object.

```
from pathlib import Path

geo_text = r'''
// from https://gitlab.onelab.info/gmsh/gmsh/-/blob/master/examples/boolean/pipe.geo
SetFactory("OpenCASCADE");

Mesh.MeshSizeMin = 0.1;
Mesh.MeshSizeMax = 0.1;
Geometry.NumSubEdges = 100; // nicer display of curve

nturns = DefineNumber[ 1, Min 0.1, Max 1, Step 0.01, Name "Parameters/Turn" ];
npts = 20;
r = 1;
rd = 0.1;
h = 1 * nturns;

For i In {0:npts-1}
  theta = i * 2*Pi*nturns/npts;
  Point(i + 1) = {r * Cos(theta), r * Sin(theta), i * h/npts};
EndFor

Spline(1) = {1:npts};
Wire(1) = {1};

Disk(1) = {1,0,0, rd};

Rectangle(2) = {1+2*rd,-rd,0, 2*rd,2*rd,rd/5};
Rotate {{1, 0, 0}, {0, 0, 0}, Pi/2} { Surface{1,2}; }

Extrude { Surface{1,2}; } Using Wire {1}
Delete{ Surface{1,2}; }
'''

Path("pipe.geo").write_text(geo_text)
print(geo_text)
```

```
// from https://gitlab.onelab.info/gmsh/gmsh/-/blob/master/examples/boolean/pipe.geo
SetFactory("OpenCASCADE");

Mesh.MeshSizeMin = 0.1;
```

```
Mesh.MeshSizeMax = 0.1;
Geometry.NumSubEdges = 100; // nicer display of curve

nturns = DefineNumber[ 1, Min 0.1, Max 1, Step 0.01, Name "Parameters/Turn" ];
npts = 20;
r = 1;
rd = 0.1;
h = 1 * nturns;

For i In {0:npts-1}
  theta = i * 2*Pi*nturns/npts;
  Point(i + 1) = {r * Cos(theta), r * Sin(theta), i * h/npts};
EndFor

Spline(1) = {1:npts};
Wire(1) = {1};

Disk(1) = {1,0,0, rd};

Rectangle(2) = {1+2*rd,-rd,0, 2*rd,2*rd,rd/5};
Rotate {{1, 0, 0}, {0, 0, 0}, Pi/2} { Surface{1,2}; }

Extrude { Surface{1,2}; } Using Wire {1}
Delete{ Surface{1,2}; }
```

The rendered geometry looks like this:

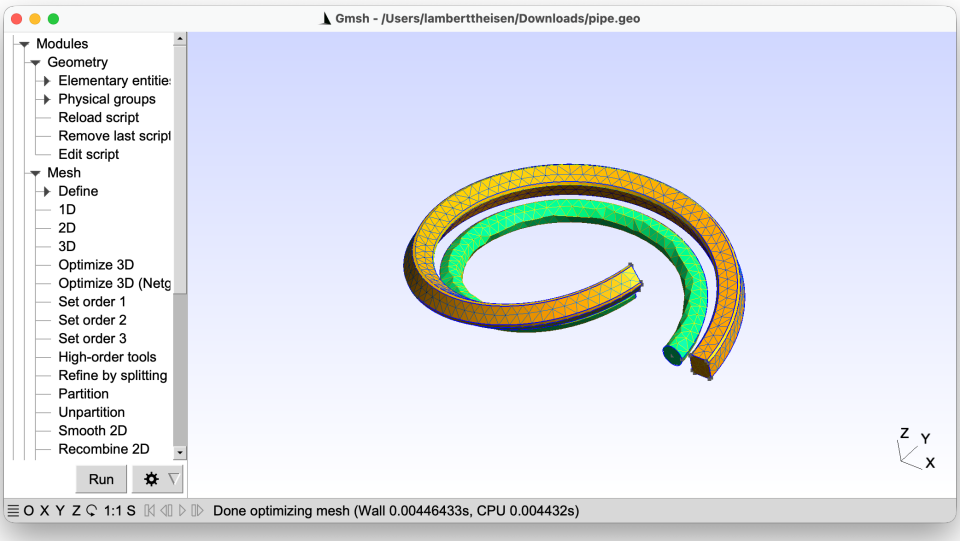


Figure 4: 3D Mesh in Gmsh

4 Further Reading

Where to go next

Use the official documentation for API details, then compare with domain-specific tutorials when importing tagged meshes into FEniCSx/DOLFINx.

- Gmsh Documentation
- Geo-file example from Gmsh Git
- Geo-files from F. Cuvelier
- Talk by C. Geuzaine and J.-F. Remacle
- Course by C. Geuzaine and J.-F. Remacle
- nice tutorial by J. Dokken

[1] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *Numerical Meth Engineering*, vol. 79, no. 11, pp. 1309–1331, Sep. 2009, doi: 10.1002/nme.2579.

[2] L. Theisen and M. Torrilhon, "fenicsR13: A Tensorial Mixed Finite Element Solver for the Linear R13 Equations Using the FEniCS Computing Platform," *ACM Trans. Math. Softw.*, vol. 47, no. 2, pp. 17:1–17:29, Apr. 2021, doi: 10.1145/3442378.