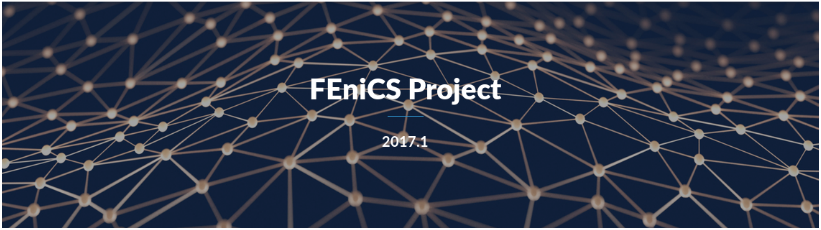
Shape Optimization with Multiple Meshes in the FEniCS-framework

Jørgen S. Dokken[1] , Simon W. Funke[1], August Johansson[1], Marie E. Rognes[1], Stephan Schmidt[2]

Simula Research Laboratory, Fornebu, Norway[1],
University of Würzburg, Würzburg, Germany[2]

September 28, 2017

FENICS'18 GOOGLE SOC'17 DOWNLOAD DOCUMENTATION COMMUNITY GOVERNANCE CITING DONATE ARCHIVES



The FEniCS computing platform

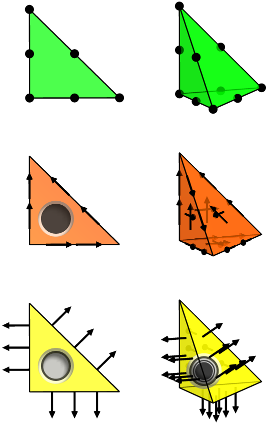
FEniCS is a popular open-source (LGPLv3) computing platform for solving partial differential equations (PDEs). FEniCS enables users to quickly translate scientific models into efficient finite element code. With the high-level Python and C++ interfaces to FEniCS, it is easy to get started, but FEniCS offers also powerful capabilities for more experienced programmers. FEniCS runs on a multitude of platforms ranging from laptops to high-performance clusters.

[fenicsproject.org]

- FEniCS is a software for solving PDEs via the finite-element method
- FEniCS is an international open source software and research project
- FEniCS is user-friendly: estimated $10^3 - 10^4$ users world-wide
- FEniCS is efficient: parallel performant up to (at least) 25 000 cores.

FEniCS provides automated generation of bases for a wide range of finite element spaces

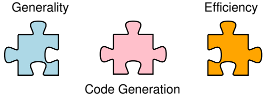
```
from dolfin import *

# Import meshes
mesh = Mesh("cable.xml")
subdomains = MeshFunction("size_t", mesh,
                          "cable_vf.xml")

# Define finite element spaces
V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
T = Function(V)

# Problem specific variables
f= Expression("cos(x[0])*exp(sin(x[1]))", degree=3)
lmb = Expression("...", degree=3)
T_ex = 20.
c = 0.01

# Define variational form
a = inner(lmb*grad(u), grad(v))*dx+u*v*ds-c*u*v*dx
l = f*v*dx+T_ex*v*ds

# Solve a(T,v) = l(v) with respect to T
solve(a == l, T)
```



FEniCS provides an expressive form language close to mathematical syntax

```
from dolfin import *

# Import meshes
mesh = Mesh("cable.xml")
subdomains = MeshFunction("size_t", mesh,
                          "cable_vf.xml")

# Define finite element spaces
V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
T = Function(V)

# Problem specific variables
f= Expression("cos(x[0])*exp(sin(x[1]))", degree=3)
lmb = Expression("...", degree=3)
T_ex = 20.
c = 0.01

# Define variational form
a = inner(lmb*grad(u), grad(v))*dx+u*v*ds-c*u*v*dx
l = f*v*dx+T_ex*v*ds

# Solve a(T,v) = l(v) with respect to T
solve(a == l, T)
```

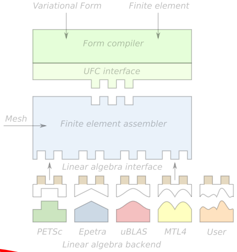
Language for variational forms



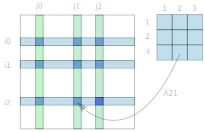
FEniCS provides automated form evaluation and assembly of the linear system

```
from dolfin import *

# Import meshes
mesh = Mesh("cable.xml")
subdomains = MeshFunction("size_t", mesh,
                          "cable_vf.xml")

# Define finite element spaces
V = FunctionSpace(mesh, "CG", 1)
u = TrialFunction(V)
v = TestFunction(V)
T = Function(V)

# Problem specific variables
f= Expression("cos(x[0])*exp(sin(x[1]))", degree=3)
lmb = Expression("...", degree=3)
T_ex = 20.
c = 0.01

# Define variational form
a = inner(lmb*grad(u), grad(v))*dx+u*v*ds-c*u*v*dx
l = f*v*dx+T_ex*v*ds

# Solve a(T,v) = l(v) with respect to T
solve(a == l, T)
```

High performance linear algebra



Automated assembly

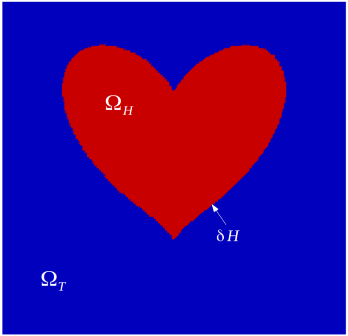


Mixed-dimensional methods[1]

[1] Cecile Daversin-Catty and Marie E. Rognes. "Automated abstractions for Mixed-Dimensional Finite Element Methods". In: *Preparation* ().

Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!

```
# W = V(Omega_H) x V(Omega_H U Omega_T)
V = FunctionSpace(mesh, "Lagrange",1) # Heart + Torso
H = FunctionSpace(submesh_heart, "Lagrange",1) # Heart
W = FunctionSpaceProduct(H,V)

# v, psi_v in V(Omega_H)
# u, psi_u in V(Omega_H U \Omega_T)
(v,u) = TrialFunction(W)
(psi_v,psi_u) = TestFunction(W)

# Integration on the heart domain Omega_H
dH = Measure("dx",domain=W.sub_space(0).mesh())
# Integration on the whole domain Omega_H U Omega_T
dV = Measure("dx",domain=W.sub_space(1).mesh())

# Variational formulation
A = v*psi_v*dH\
    + th*dt*Mi*inner(grad(v),grad(psi_v))*dH
C = (dt/th)*(Mi+Me)*inner(grad(u),grad(psi_u))*dV\
    + (dt/th)*Mt*inner(grad(u),grad(psi_u))*dV
B = dt*Mi*inner(grad(u),grad(psi_v))*dH
BT = dt*Mi*inner(grad(v),grad(psi_u))*dH
a = A + C + B + BT
L = c + d

sol = Function(W)
solve(a == L, sol)
```



[C. Daversin-Catty, cecile@simula.no]

Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!

```
# W = V(Omega_H) x V(Omega_H U Omega_T)
V = FunctionSpace(mesh, "Lagrange",1) # Heart + Torso
H = FunctionSpace(submesh_heart, "Lagrange",1) # Heart
W = FunctionSpaceProduct(H,V)

# v, psi_v in V(Omega_H)
# u, psi_u in V(Omega_H U \Omega_T)
(v,u) = TrialFunction(W)
(psi_v,psi_u) = TestFunction(W)

# Integration on the heart domain Omega_H
dH = Measure("dx",domain=W.sub_space(0).mesh())
# Integration on the whole domain Omega_H U Omega_T
dV = Measure("dx",domain=W.sub_space(1).mesh())

# Variational formulation
A = v*psi_v*dH\
    + th*dt*Mi*inner(grad(v),grad(psi_v))*dH
C = (dt/th)*(Mi+Me)*inner(grad(u),grad(psi_u))*dV\
    + (dt/th)*Mt*inner(grad(u),grad(psi_u))*dV
B = dt*Mi*inner(grad(u),grad(psi_v))*dH
BT = dt*Mi*inner(grad(v),grad(psi_u))*dH
a = A + C + B + BT
L = c + d

sol = Function(W)
solve(a == L, sol)
```

$$\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}$$

$\phi_H^i : V(\Omega_H)$ basis functions

$\phi_{HT}^i : V(\Omega_H \cup \Omega_T)$ basis functions

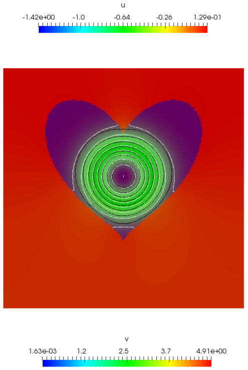
$$A_{ij} = \int_{\Omega_H} \phi_H^j \phi_H^i + \theta \Delta t \int_{\Omega_H} M_i \nabla \phi_H^j \cdot \nabla \phi_H^i$$

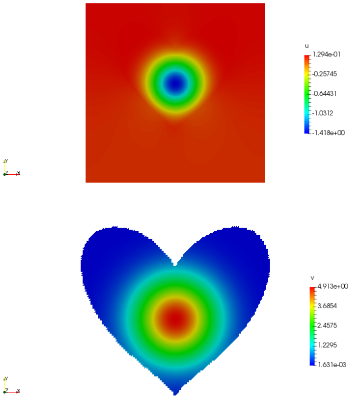
$$B_{ij} = \Delta t \int_{\Omega_H} M_i \nabla \phi_H^j \cdot \nabla \phi_{HT}^i$$

$$C_{ij} = \frac{\Delta t}{\theta} \int_{\Omega_H} (M_i + M_e) \nabla \phi_{HT}^j \cdot \nabla \phi_{HT}^i + \frac{\Delta t}{\theta} \int_{\Omega_T} M_T \nabla \phi_{HT}^j \cdot \nabla \phi_{HT}^i$$

[C. Daversin-Catty, cecile@simula.no]

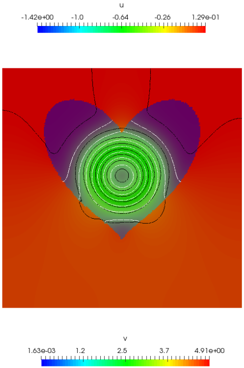
Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!

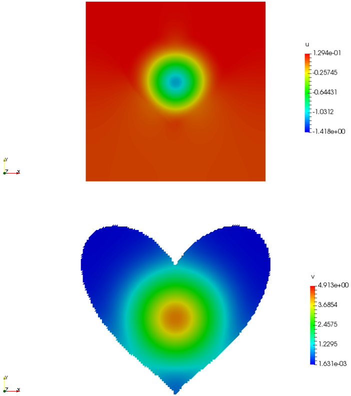




[C. Daversin-Catty, cecile@simula.no]

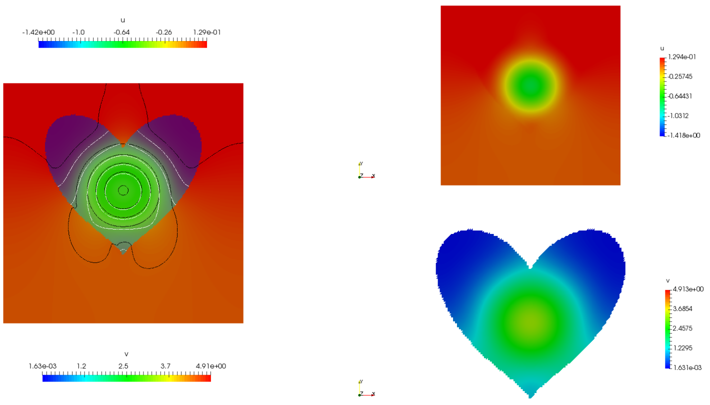
Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!





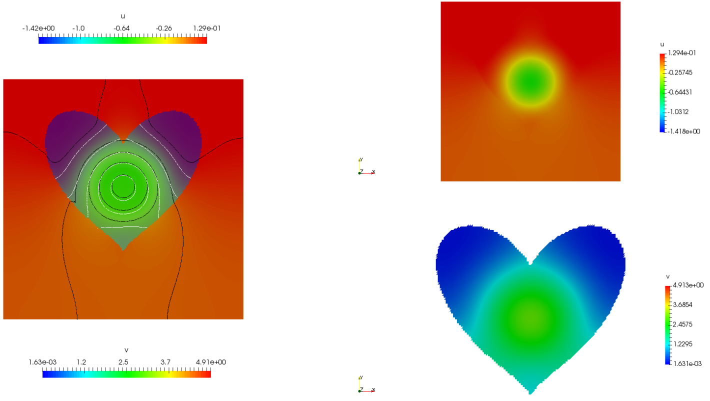
[C. Daversin-Catty, cecile@simula.no]

Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!



[C. Daversin-Catty, cecile@simula.no]

Multi-physics problems require efficent mixed-dimensional and mixed domain coupling: emerging features in FEniCS!

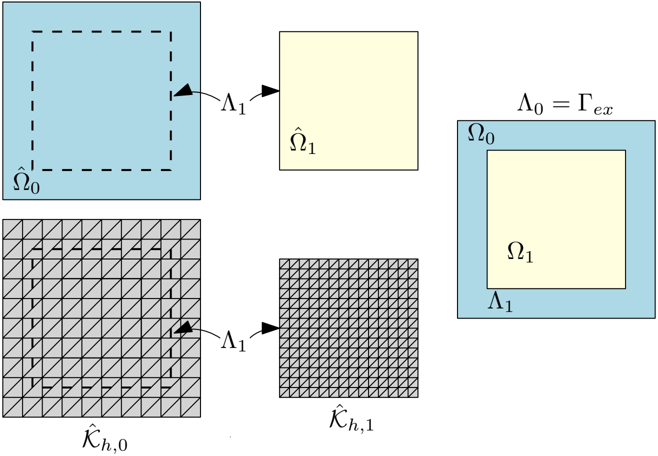


[C. Daversin-Catty, cecile@simula.no]

CUT Finite Element Methods: MultiMesh[2]

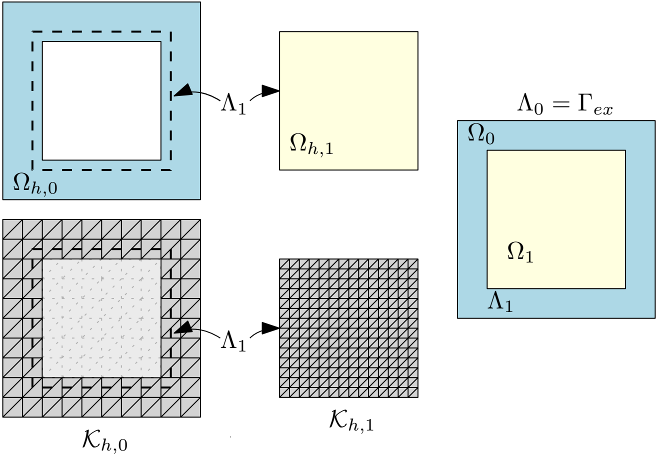
[2] August Johansson et al. "Finite Element Methods for Arbitrary Many Intersecting Meshes: Multimesh". In: *Preparation* ().

The computational domain is represented by an arbitrary number of overlapping meshes



August Johansson et al. "Finite Element Methods for Arbitrary Many Intersecting Meshes: Multimesh". In: *Preparation* ().

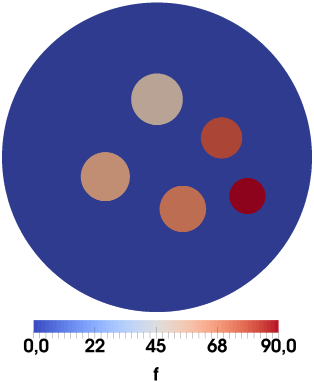
A finite element function space is introduced on each indvidual mesh, ignoring completely covered cells



August Johansson et al. "Finite Element Methods for Arbitrary Many Intersecting Meshes: Multimesh". In: *Preparation* ().

We illustrate the method by considering the stationary heat equation with a reaction coefficient

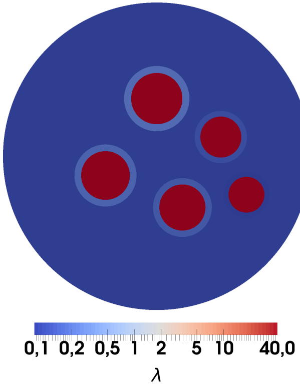
$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f \quad \text{in } \Omega, \\
\lambda_{ex} \frac{\partial T}{\partial n} + (T - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex}, \\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n} \right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}.
\end{aligned}
$$



Distribution of the source f in the computational domain.

We illustrate the method by considering the stationary heat equation with a reaction coefficient

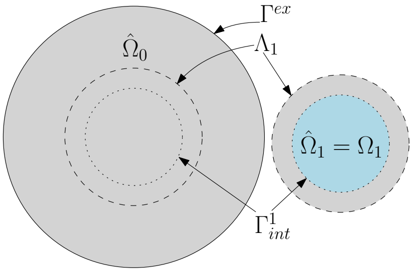
$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f \quad \text{in } \Omega, \\
\lambda_{ex} \frac{\partial T}{\partial n} + (T - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex}, \\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}.
\end{aligned}
$$



Heat diffusion coefficient λ in the computational domain.

Continuity of the solution is enforced over the artificial interface Λ_1

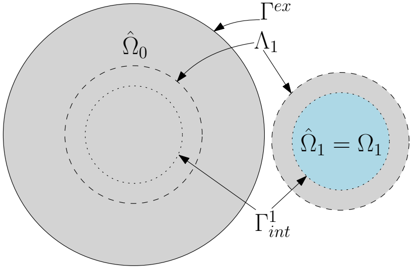
$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f \quad \text{in } \Omega, \\
\lambda_{ex} \frac{\partial T}{\partial n} + (T - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex}, \\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n} \right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}.
\end{aligned}
$$



Schematic of the composition of multiple overlapping meshes.

Continuity of the solution is enforced over the artificial interface Λ_1

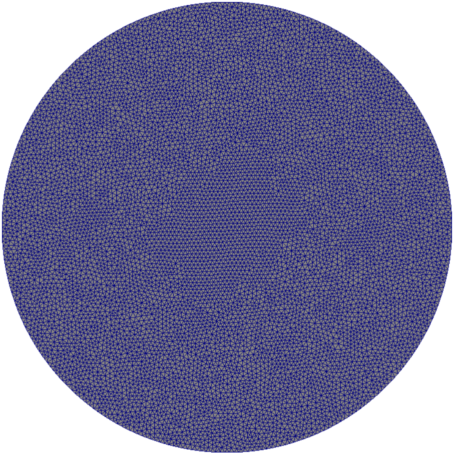
$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T_0) - cT_0 &= f \quad \text{in } \Omega_0, \\
-\nabla \cdot (\lambda \nabla T_1) - cT_1 &= f \quad \text{in } \Omega_1, \\
\lambda_{ex} \frac{\partial T_0}{\partial n} + (T_0 - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex}, \\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int} \\
[T] &= 0 \quad \text{on } \Lambda_1, \\
\left[\frac{\partial T}{\partial n}\right] &= 0 \quad \text{on } \Lambda_1.
\end{aligned}
$$



Schematic of the composition of multiple overlapping meshes.

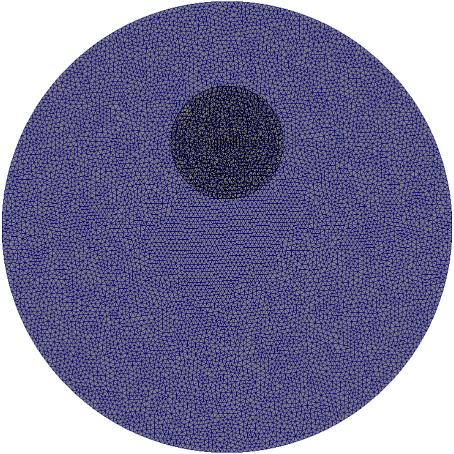
We create a MultiCable in FEniCS by initializing the MultiMesh object and add a background mesh

```
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
            -f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```



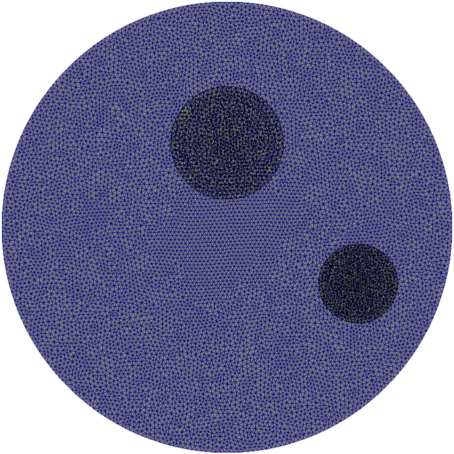
We add multiple internal cables on top of the background cable

```python
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
           -f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```



We add multiple internal cables on top of the background cable

```
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
          -f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```



Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

```
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
          -f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```

$$0 = F_s(T, v)$$

$$F_s(T, v) = \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \, \mathrm{d}x + \int_{\Gamma^{\text{ex}}} (T_0 - T^{\text{ex}}) v \, \mathrm{d}s = 0$$

Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

```python
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
        - f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```

$$0 = F_s(T, v) + F_N(T, v)$$

$$F_s(T, v) = \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \, \mathrm{d}x + \int_{\Gamma^{\text{ex}}} (T_0 - T^{\text{ex}}) v \, \mathrm{d}s = 0$$

$$F_N(T, v) = -(\langle \lambda \mathbf{n}_1 \cdot \nabla T \rangle, [v])_{\Lambda_1} - ([T_h], \langle \lambda \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([T], [v])_{\Lambda_1},$$

Nitsche terms	Jump	Average	
$F_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	

Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

```
from dolfin import *

multimesh = MultiMesh()
multimesh.add(Mesh("outer_cable.xml"))
for i in range(num_cables):
    cable = Mesh("inner_cable.xml")
    # Scale and move internal cables
    # ....
    multimesh.add(cable)
multimesh.build()

# Create function space for temperature
V = MultiMeshFunctionSpace(multimesh, "CG", 1)
T = MultiMeshFunction(V, name="Temperature")
u,v = TrialFunction(V), TestFunction(V)

# Problem Specific variables
f = Expression("sin(x[0]*x[1])", degree=3)
lmb = Expression("...", degree=3)
T_ex, c = 3.2,0.04

alpha, beta = 4.0,4.0
n = FacetNormal(multimesh)
h =2.0*Circumradius(multimesh)
h = (h('+') + h('-')) / 2
F = inner(lmb*grad(u), grad(v))*dX \
          -f*v*dX -c*v*u*dX+ (u-T_ex)*v*ds
F += - inner(avg(lmb*grad(u)), jump(v, n))*dI \
     - inner(avg(lmb*grad(v)), jump(u, n))*dI \
     + alpha/h*jump(u)*jump(v)*dI  \
     + beta*inner(jump(grad(u)), jump(grad(v)))*dO

# Assemble multimesh form
A = assemble_multimesh(lhs(F))
b = assemble_multimesh(rhs(F))
solve(A, T.vector(), b, 'lu')
```

$$0 = F_s(T, v) + F_N(T, v) + F_O(T, v)$$

$$F_s(T, v) = \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \,\mathrm{d}x + \int_{\Gamma^{\mathrm{ex}}} (T_0 - T^{\mathrm{ex}})v \,\mathrm{d}s = 0$$

$$F_N(T, v) = -(\langle \lambda \mathbf{n}_1 \cdot \nabla T \rangle, [v])_{\Lambda_1} - ([T_h], \langle \lambda \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([T], [v])_{\Lambda_1},$$

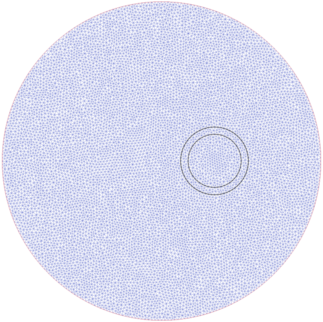
$$F_O(T, v) = ([\nabla T], [\nabla v])_{\Omega_{h,0} \cap \Omega_1}.$$

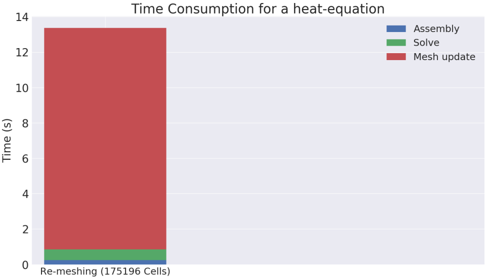
Nitsche terms	Jump	Average	Stability on overlap
$F_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	$F_O(u, v)$

Shape-optimization With Overlapping Domains[4]

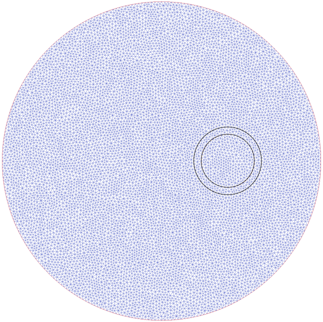
[4] Jørgen S. Dokken et al. "Shape Optimization on Multiple Meshes". In: *Preparation* ().

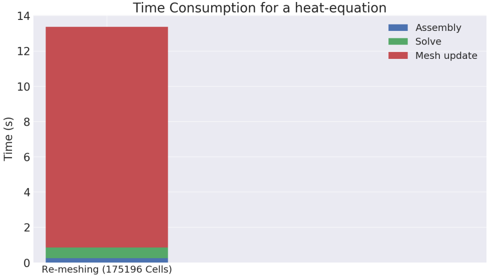
Re-meshing guarantees good mesh-quality, but it is a very costly operation



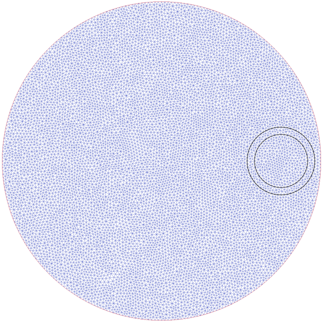


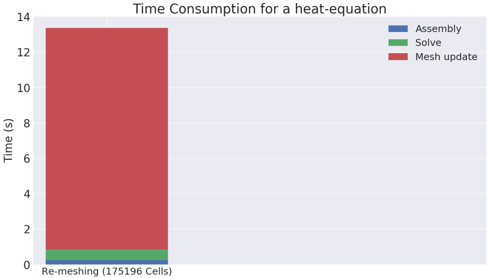
Re-meshing guarantees good mesh-quality, but it is a very costly operation



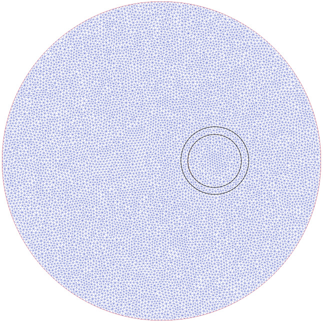


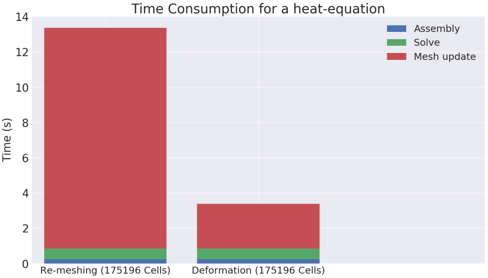
Re-meshing guarantees good mesh-quality, but it is a very costly operation



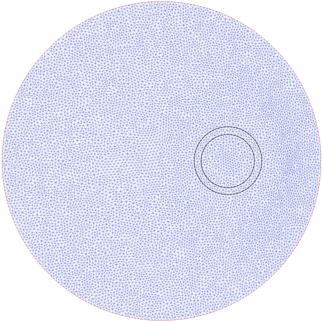


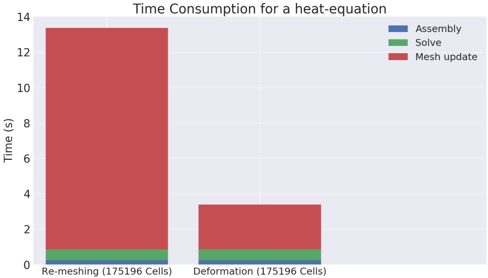
Deformation of the mesh is cheaper than re-meshing but degenerates for large changes



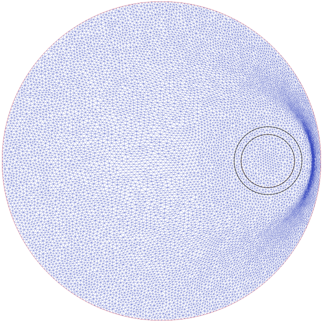


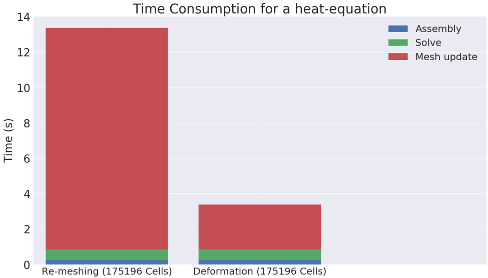
Deformation of the mesh is cheaper than re-meshing but degenerates for large changes



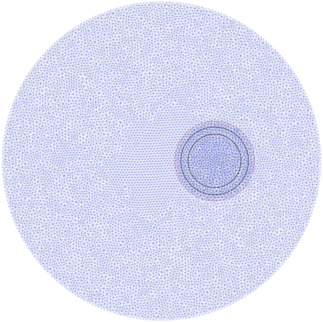


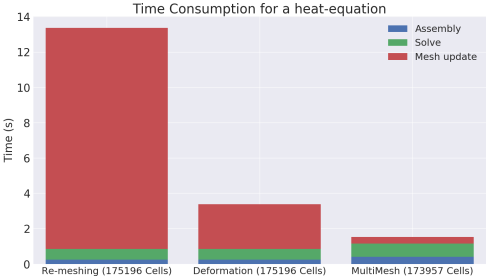
Deformation of the mesh is cheaper than re-meshing but degenerates for large changes



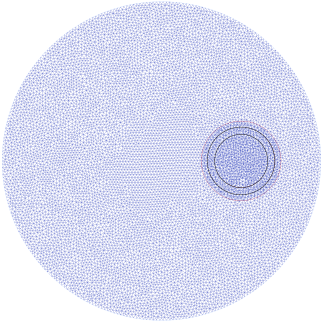


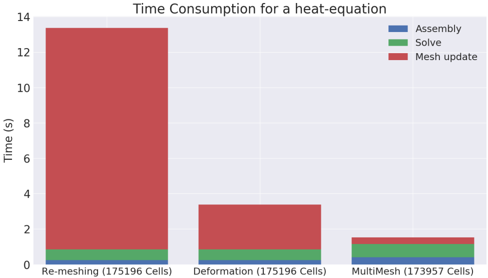
Multiple overlapping meshes is very efficient and preserves mesh quality



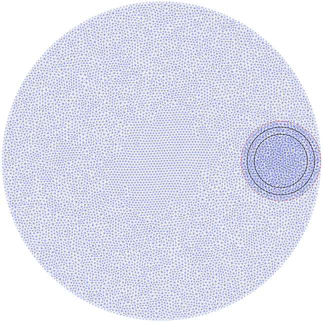


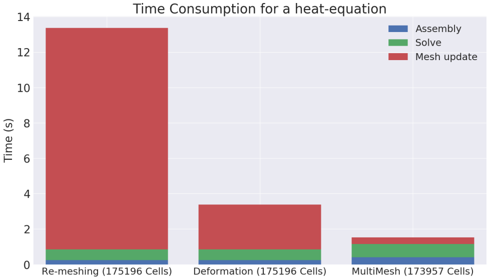
Multiple overlapping meshes is very efficient and preserves mesh quality





Multiple overlapping meshes is very efficient and preserves mesh quality





We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u,\Omega) \quad \text{s.t.} \quad E(u,\Omega) = 0.$$

We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u,\Omega) \quad \text{s.t.} \quad E(u,\Omega) = 0.$$

Algorithm: Shape-optimization on multiple domains

Init : **Domain composition** $\hat{\Omega}^0 = \cup_{i=0,\dots N}\hat{\Omega}_i^0$

Param: $k = 0$

while *not converged* **do**

end

Result: Optimized domain

We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u, \Omega) \quad \text{s.t.} \quad E(u, \Omega) = 0.$$

Algorithm: Shape-optimization on multiple domains

Init : Domain composition $\hat{\Omega}^0 = \cup_{i=0,\dots N} \hat{\Omega}_i^0$
Param: $k = 0$
while *not converged* **do**
    Solve state equations on $\hat{\Omega}^k$;
end
Result: Optimized domain

We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u,\Omega) \quad \text{s.t.} \quad E(u,\Omega) = 0.$$

Algorithm: Shape-optimization on multiple domains

Init : Domain composition $\hat{\Omega}^0 = \cup_{i=0,\dots N} \hat{\Omega}_i^0$
Param: $k = 0$
while *not converged* **do**
  Solve state equations on $\hat{\Omega}^k$;
  Compute the shape-derivatives d$J/$dΩ;
end
Result: Optimized domain

We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u,\Omega) \quad \text{s.t.} \quad E(u,\Omega) = 0.$$

Algorithm: Shape-optimization on multiple domains

Init : Domain composition $\hat{\Omega}^0 = \cup_{i=0,\dots N}\hat{\Omega}_i^0$
Param: $k = 0$
while *not converged* **do**
  Solve state equations on $\hat{\Omega}^k$;
  Compute the shape-derivatives $\mathrm{d}J/\mathrm{d}\Omega$;
  Update the subdomains,$\hat{\Omega}_i^{k+1}, i = 0, \dots N$**;**
end
Result: Optimized domain

We propose a new algorithm for solving PDE-constrained shape optimization problems

$$\min_{u,\Omega} J(u,\Omega) \quad \text{s.t.} \quad E(u,\Omega) = 0.$$

Algorithm: Shape-optimization on multiple domains

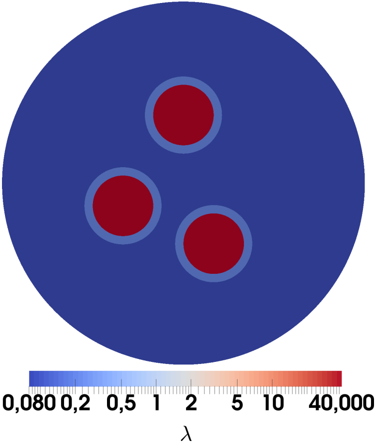
Init : Domain composition $\hat{\Omega}^0 = \cup_{i=0,\dots N}\hat{\Omega}_i^0$
Param: $k = 0$
while *not converged* **do**
  Solve state equations on $\hat{\Omega}^k$;
  Compute the shape-derivatives $\mathrm{d}J/\mathrm{d}\Omega$;
  Update the subdomains,$\hat{\Omega}_i^{k+1}, i = 0, \dots N$;
  Increment k and set $\hat{\Omega}^k = \cup_{i=0,\dots N}\hat{\Omega}_i^k$;
end
Result: Optimized domain

The solution of an optimization problem with three identical cables is an equilateral triangle

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3}|T|^3 \mathrm{d}x,$$

subject to

$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega,\\
\frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{\text{ex}}.\\
[T]_\pm &= 0 && \text{on } \Gamma_{int},\\
\left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma_{int}
\end{aligned}
$$



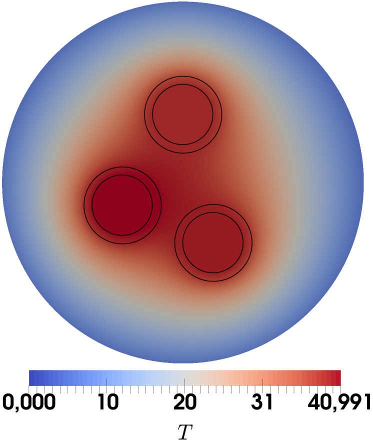
Three cables with the same thermal diffusivity.

The solution of an optimization problem with three identical cables is an equilateral triangle

$$\min_{\Omega, T} J(\Omega, T) = \int_{\Omega} \frac{1}{3}|T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned} -\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\ \frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex}. \\ [T]_{\pm} &= 0 && \text{on } \Gamma_{int}, \\ \left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 && \text{on } \Gamma_{int} \end{aligned}$$



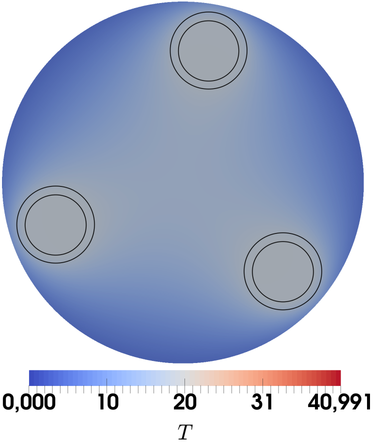
Initial cable positioning and corresponding temperature.

The solution of an optimization problem with three identical cables is an equilateral triangle

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3} |T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\
\frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{\mathrm{ex}}. \\
[T]_\pm &= 0 && \text{on } \Gamma_{int}, \\
\left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma_{int}
\end{aligned}$$



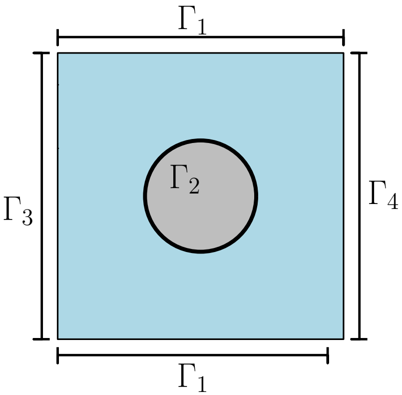
Optimal cable distribution and temperature.

A benchmark problem in shape-optimization is the optimal shape of an obstacle in Stokes-flow

$$\min_{(u,\Omega)} : J(\Omega) = \int_\Omega \sum_{i,j=1}^{2} \left(\frac{\partial u_i}{\partial x_j} \right)^2 \, \mathrm{d}A$$

subject to

$$\begin{aligned} -\Delta u + \nabla p &= 0 \quad \text{in } \Omega, \\ \nabla \cdot u &= 0, \\ u &= 0 \quad \text{on } \Gamma_2, \\ u &= u_0 \quad \text{on } \Gamma_1 \cup \Gamma_3, \\ p &= 0 \quad \text{on } \Gamma_4, \\ C &= C_0, \\ \mathrm{Vol} &= \mathrm{Vol}_0. \end{aligned}$$



Initial setup of the domain.

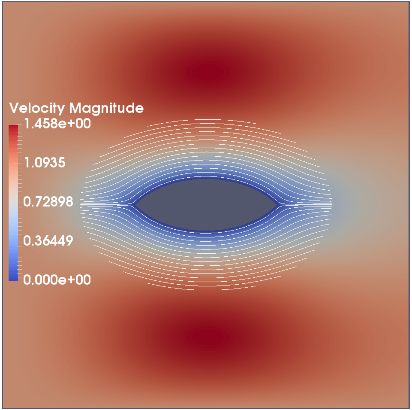
Olivier Pironneau. "On optimum design in fluid mechanics". In: *Journal of Fluid Mechanics* 64.1 (1974), pp. 97–110.

We achieve the analytical shape, a rugby-ball with a 90 degree front and back angle[5]

$$\min_{(u,\Omega)} : J(\Omega) = \int_\Omega \sum_{i,j=1}^{2} \left(\frac{\partial u_i}{\partial x_j}\right)^2 \, \mathrm{d}A$$

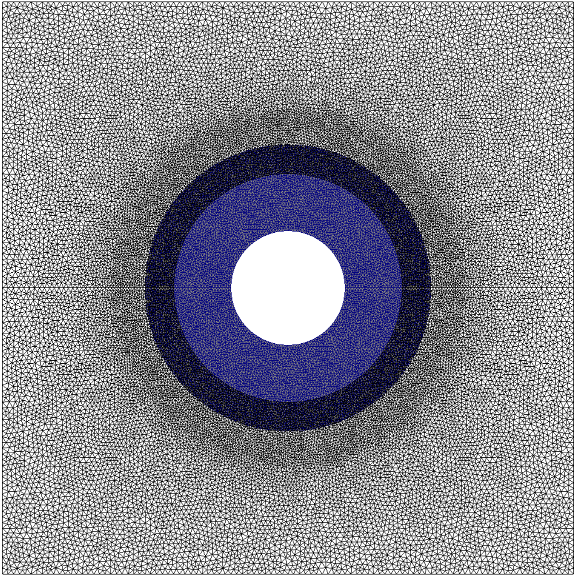
subject to

$$\begin{aligned} -\Delta u + \nabla p &= 0 \quad \text{in } \Omega, \\ \nabla \cdot u &= 0, \\ u &= 0 \quad \text{on } \Gamma_2, \\ u &= u_0 \quad \text{on } \Gamma_1 \cup \Gamma_3, \\ p &= 0 \quad \text{on } \Gamma_4, \\ C &= C_0, \\ \mathrm{Vol} &= \mathrm{Vol}_0. \end{aligned}$$

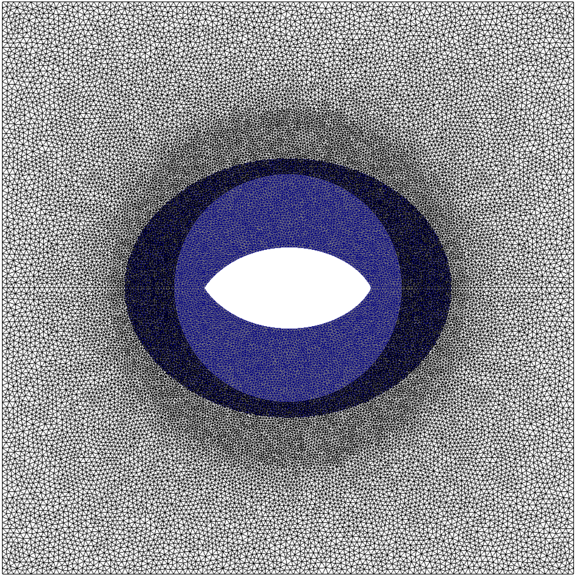


[5] Olivier Pironneau. "On optimum design in fluid mechanics". In: *Journal of Fluid Mechanics* 64.1 (1974), pp. 97–110.

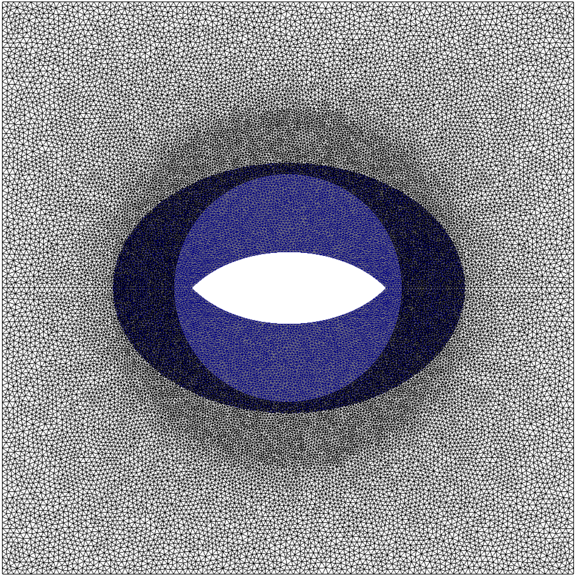
With multiple meshes, we can reduce the size of the mesh that has to be deformed



With multiple meshes, we can reduce the size of the mesh that has to be deformed

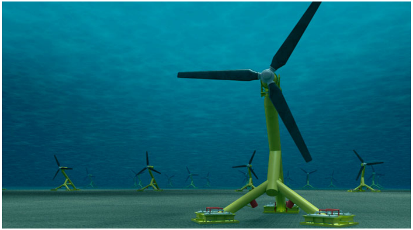


With multiple meshes, we can reduce the size of the mesh that has to be deformed



Further work

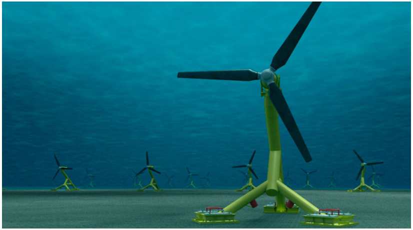
- Extend the multiple mesh formulation to to time dependent problems such as the NS-equation.
- Use shape-optimization to optimize power-output of a tidal turbine farm.



[islayenergytrust.org.uk/tidal-energy-project/]

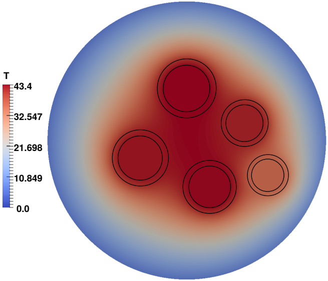
Further work

- Extend the multiple mesh formulation to to time dependent problems such as the NS-equation.
- Use shape-optimization to optimize power-output of a tidal turbine farm.



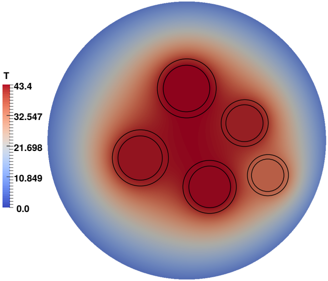
[islayenergytrust.org.uk/tidal-energy-project/]

Concluding, FEniCS is currently being extended to employ mixed-domain method and CUT-FEM, where the latter has been used for avoiding re-meshing in shape-optimization



This project is funded by the The Research Council of Norway

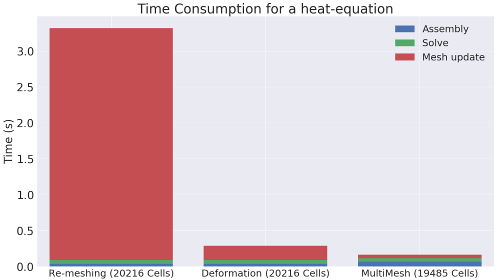
Concluding, FEniCS is currently being extended to employ mixed-domain method and CUT-FEM, where the latter has been used for avoiding re-meshing in shape-optimization

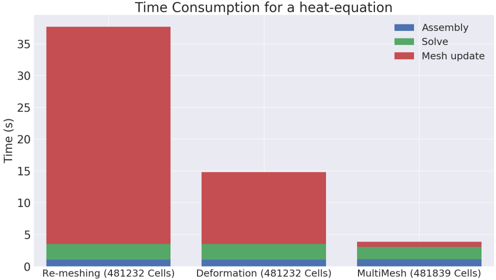


Questions?

This project is funded by the The Research Council of Norway

This trend is clear for both finer and coarser meshes.





The shape-derivative of a functional constrained by PDEs is found with the adjoint method and shape-calculus

$$\min_{\Omega} J(u, \Omega) \text{ s.t. } E(u, \Omega) = 0,$$
$$\hat{J}(\Omega) = J(u(\Omega), \Omega)$$

The shape-derivative of a functional constrained by PDEs is found with the adjoint method and shape-calculus

$$\min_{\Omega} J(u,\Omega) \text{ s.t. } E(u,\Omega)=0,$$
$$\hat{J}(\Omega)=J(u(\Omega),\Omega)$$

Lagrangian based adjoint equation

$$\mathcal{L}(u,\Omega)=J(u,\Omega)+(\lambda,E(u(\Omega),\Omega)).$$
$$\mathrm{d}\hat{J}(\Omega)[s]=\frac{\partial\mathcal{L}}{\partial\Omega}[s]=\frac{\partial J}{\partial\Omega}[s]+\left(\lambda,\frac{\partial E}{\partial\Omega}[s]\right),$$
$$\frac{\partial\mathcal{L}}{\partial u}=\frac{\partial J}{\partial u}[d]+\left(\lambda,\frac{\partial E}{\partial u}[d]\right)=0,\quad\forall d.$$

A linear state equation yields an adjoint equation similar to the state equation

$$\mathcal{L}(u, \Omega) = J(u, \Omega) + (\lambda, E(u(\Omega), \Omega)).$$
$$\frac{\partial J}{\partial u}[d] + \left(\lambda, \frac{\partial E}{\partial u}[d]\right) = 0, \quad \forall d.$$

A linear state equation yields an adjoint equation similar to the state equation

$$\mathcal{L}(u, \Omega) = J(u, \Omega) + (\lambda, E(u(\Omega), \Omega)).$$
$$\frac{\partial J}{\partial u}[d] + \left(\lambda, \frac{\partial E}{\partial u}[d]\right) = 0, \quad \forall d.$$

$$E(u) = Au + b,$$
$$\left(\lambda \frac{\partial E}{\partial u}[d]\right) = (\lambda, Ad).$$

The shape-derivative is transformed into surface integrals with the Hadamard theorem

$$\mathcal{L}(u, \Omega) = J(u, \Omega) + (\lambda, E(u(\Omega), \Omega)).$$

$$\mathrm{d}\hat{J}(\Omega)[s] = \frac{\partial \mathcal{L}}{\partial \Omega}[s] = \frac{\partial J}{\partial \Omega}[s] + \left(\lambda, \frac{\partial E}{\partial \Omega}[s]\right),$$

Theorem (Hadamard Theorem)

The shape-derivative is transformed into surface integrals with the Hadamard theorem

$$\mathcal{L}(u, \Omega) = J(u, \Omega) + (\lambda, E(u(\Omega), \Omega)).$$

$$\mathrm{d}\hat{J}(\Omega)[s] = \frac{\partial \mathcal{L}}{\partial \Omega}[s] = \frac{\partial J}{\partial \Omega}[s] + \left(\lambda, \frac{\partial E}{\partial \Omega}[s]\right),$$

Theorem (Hadamard Theorem)

Let $\hat{J}$ be shape differentiable. Then the relation

$$\mathrm{d}\hat{J}(\Omega)[V] = \int_\Gamma \langle V, n \rangle g \ \mathrm{d}S$$

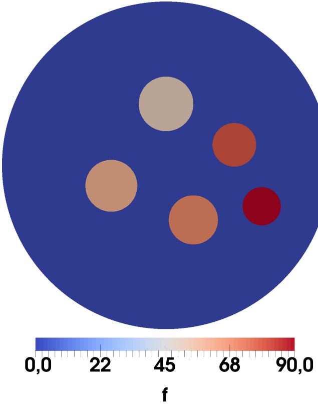
holds for all vector fields.

We consider minimization of the temperature in current-carrying MultiCables as a first example

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3}|T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega,\\
\frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex},\\
[T]_\pm &= 0 && \text{on } \Gamma^1_{int},\\
\left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma^1_{int}.
\end{aligned}$$



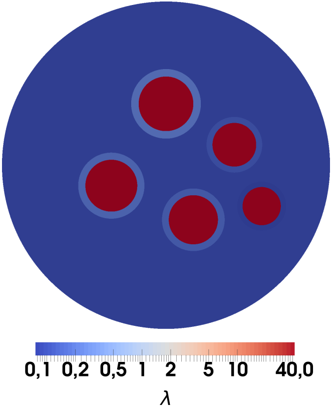
Three cables with the same thermal diffusivity.

We consider minimization of the temperature in current-carrying MultiCables as a first example

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3} |T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega,\\
\frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex},\\
[T]_\pm &= 0 && \text{on } \Gamma^1_{int},\\
\left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma^1_{int}.
\end{aligned}$$



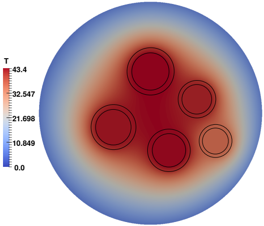
Initial cable positioning and corresponding temperature.

We consider minimization of the temperature in current-carrying MultiCables as a first example

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3}|T|^3 \mathrm{d}x,$$

subject to

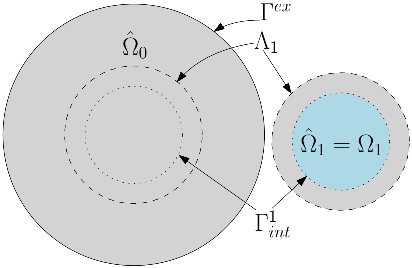
$$\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\
\frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex}, \\
[T]_\pm &= 0 && \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma^1_{int}.
\end{aligned}$$



Optimal cable distribution and temperature.

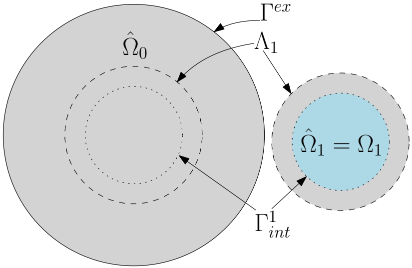
The multiple meshes strong formulation has additional terms for continuity over the artificial interface Λ_1

$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T) - cT &= f \quad \text{in } \Omega, \\
\lambda_{ex} \frac{\partial T}{\partial n} + (T - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex}, \\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}, \\
\left[\lambda \frac{\partial T}{\partial n} \right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}.
\end{aligned}
$$



The multiple meshes strong formulation has additional terms for continuity over the artificial interface Λ_1

$$
\begin{aligned}
-\nabla \cdot (\lambda \nabla T_0) - cT_0 &= f \quad \text{in } \Omega_0,\\
-\nabla \cdot (\lambda \nabla T_1) - cT_1 &= f \quad \text{in } \Omega_1,\\
\lambda_{ex}\frac{\partial T_0}{\partial n} + (T_0 - T_{ex}) &= 0 \quad \text{on } \Gamma^{ex},\\
[T]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int},\\
\left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 \quad \text{on } \Gamma^1_{int}\\
[u] &= 0 \quad \text{on } \Lambda_1,\\
\left[\frac{\partial u}{\partial n}\right] &= 0 \quad \text{on } \Lambda_1.
\end{aligned}
$$



Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

$$
\begin{aligned}
0 &= F_s(T, v) \\
F_s(T, v) &= \int_\Omega \lambda(\nabla T, \nabla v) - cTv - fv \,\mathrm{d}x \\
&\quad + \int_{\Gamma^{\mathrm{ex}}} (T - T^{\mathrm{ex}})v \,\mathrm{d}s = 0
\end{aligned}
$$

Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

$$
\begin{aligned}
0 &= F_s(T, v) \\
F_s(T, v) &= \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \,\mathrm{d}x \\
&\quad + \int_{\Gamma^{\mathrm{ex}}} (T_0 - T^{\mathrm{ex}}) v \,\mathrm{d}s = 0
\end{aligned}
$$

Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

$$0 = F_s(T, v) + F_N(T, v)$$

$$F_s(T, v) = \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \,\mathrm{d}x$$

$$+ \int_{\Gamma^{ex}} (T_0 - T^{ex}) v \,\mathrm{d}s = 0$$

$$F_N(T, v) = -(\langle \mathbf{n}_1 \cdot \nabla T \rangle, [v])_{\Lambda_1} - ([T_h], \langle \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([T], [v])_{\Lambda_1},$$

Nitsche terms	Jump	Average	
$F_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	

Nitsches method for weak enforcement of boundary conditions is used to obtain a stable finite element scheme

$$0 = F_s(T, v) + F_N(T, v) + F_O(T, v)$$

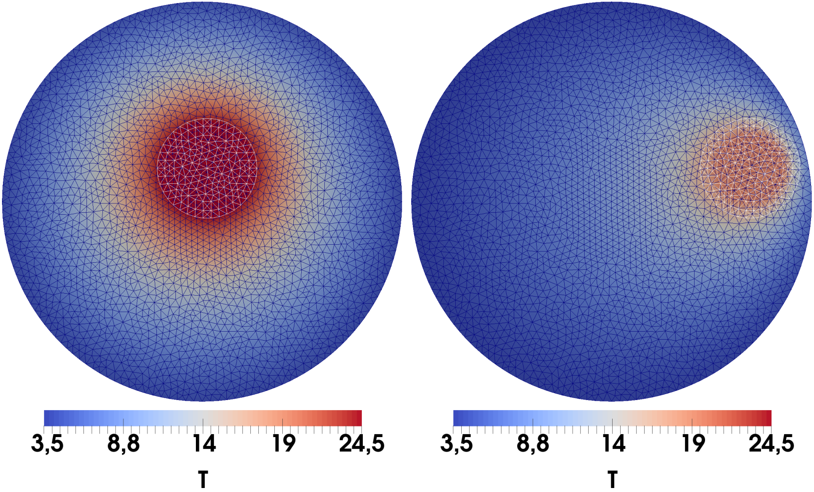
$$F_s(T, v) = \sum_{i=0}^{1} \int_{\Omega_i} \lambda(\nabla T, \nabla v) - cTv - fv \,\mathrm{d}x + \int_{\Gamma^{\mathrm{ex}}} (T_0 - T^{\mathrm{ex}}) v \,\mathrm{d}s = 0$$

$$F_N(T, v) = -(\langle \mathbf{n}_1 \cdot \nabla T \rangle, [v])_{\Lambda_1} - ([T_h], \langle \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([T], [v])_{\Lambda_1},$$

$$F_O(T, v) = ([\lambda \nabla T], [\nabla v])_{\Omega_{h,0} \cap \Omega_1}.$$

Nitsche terms	Jump	Average	Stability on overlap
$F_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	$F_O(u, v)$

The implementation of the shape-gradient is verified with a Taylor-test

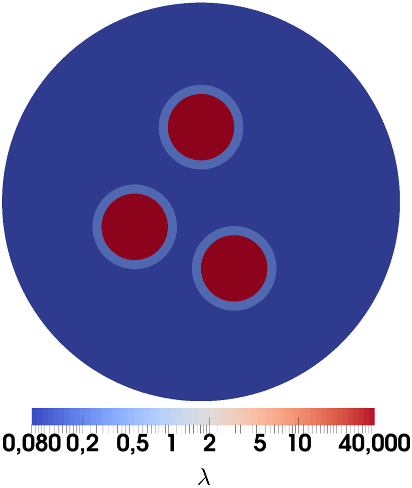


A first example is three internal cables with the same material properties

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3}|T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned} -\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\ \frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex}. \\ [T]_{\pm} &= 0 && \text{on } \Gamma^1_{int}, \\ \left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 && \text{on } \Gamma^1_{int} \end{aligned}$$



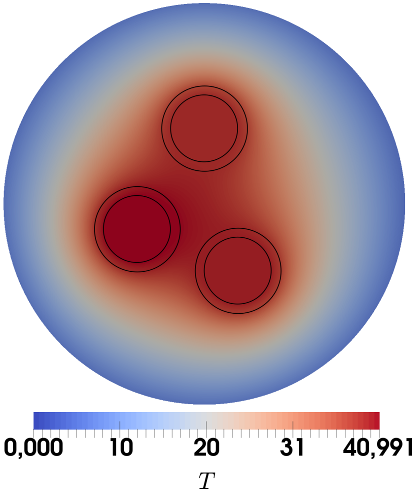
Three cables with the same thermal diffusivity.

A first example is three internal cables with the same material properties

$$\min_{\Omega, T} J(\Omega, T) = \int_{\Omega} \frac{1}{3} |T|^3 \mathrm{d}x,$$

subject to

$$\begin{aligned} -\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\ \frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex}. \\ [T]_{\pm} &= 0 && \text{on } \Gamma^{1}_{int}, \\ \left[\lambda \frac{\partial T}{\partial n}\right]_{\pm} &= 0 && \text{on } \Gamma^{1}_{int} \end{aligned}$$



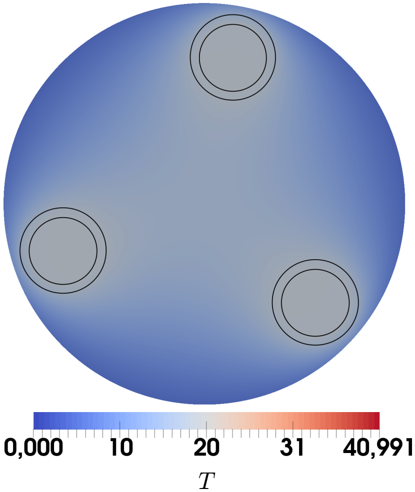
Initial cable positioning and corresponding temperature.

A first example is three internal cables with the same material properties

$$\min_{\Omega, T} J(\Omega, T) = \int_\Omega \frac{1}{3}|T|^3 \mathrm{d}x,$$

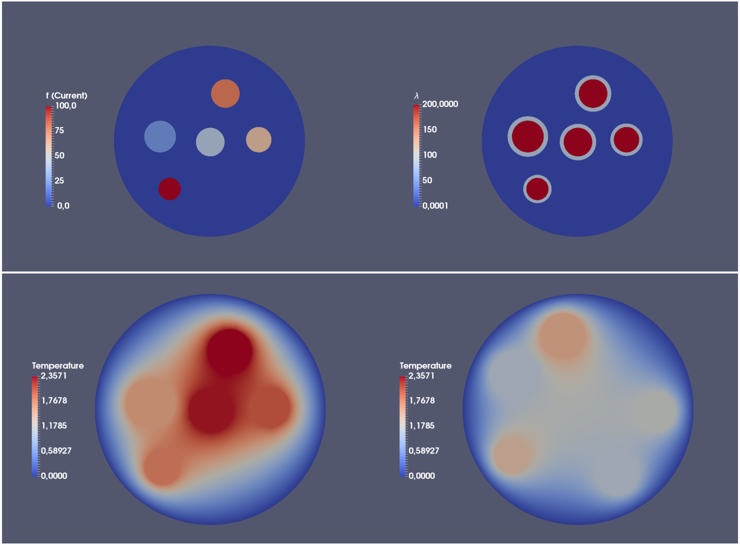
subject to

$$\begin{aligned} -\nabla \cdot (\lambda \nabla T) - cT &= f && \text{in } \Omega, \\ \frac{\partial T}{\partial n} + (T - T_{amb}) &= 0 && \text{on } \Gamma^{ex}. \\ [T]_\pm &= 0 && \text{on } \Gamma^1_{int}, \\ \left[\lambda \frac{\partial T}{\partial n}\right]_\pm &= 0 && \text{on } \Gamma^1_{int} \end{aligned}$$



Optimal cable distribution and temperature.

Optimization of a more complex example with 5 cables with different currents and sizes



The state-equation has been implemented in FEniCS and verified by the method of manufactured solutions

MultiMesh			SingleMesh		
Mesh size	L^2-error	Rate	Mesh size	L^2-error	Rate
0.059	2.41e-04	2.116	0.062	2.27e-04	2.008
0.030	5.92e-05	2.067	0.031	5.65e-05	2.003
0.015	1.46e-05	2.055	0.016	1.41e-05	2.002
0.008	3.68e-06	1.999	0.008	3.52e-06	2.001

Table: Convergence rates of a manufactured Poisson problem. Comparison between MultiMesh and the same mesh described as a single mesh approximated by piece-wise continuous linear elements.

The adjoint system for overlapping meshes has the same stabilization at the artificial interface as the state equation at the interface

$$\sum_{i=0}^{N} \Big((\lambda \nabla p, \nabla v)_{\Omega_i} - (cp, v)_{\Omega_i} \Big)$$

$$+ (\alpha'(T)(T - T^{\mathrm{ex}})p, v)_{\Gamma^{\mathrm{ex}}} + (\alpha(T)p, v)_{\Gamma^{\mathrm{ex}}} = -\sum_{i=0}^{N} (T|T|, v)_{\Omega_i}.$$

Standard terms	Nitsche terms	Overlap terms

The adjoint system for overlapping meshes has the same stabilization at the artificial interface as the state equation at the interface

$$\sum_{i=0}^{N}\Big((\lambda\nabla p,\nabla v)_{\Omega_i}-(cp,v)_{\Omega_i}\Big)$$

$$+(\alpha'(T)(T-T^{\mathrm{ex}})p,v)_{\Gamma^{\mathrm{ex}}}+(\alpha(T)p,v)_{\Gamma^{\mathrm{ex}}}$$

$$+\sum_{i=1}^{N}\Big(-(\langle\lambda\mathbf{n}_i\cdot\nabla p,\rangle[v])_{\Lambda_i}-([p],\langle\lambda\mathbf{n}_i\cdot\nabla v\rangle)_{\Lambda_i}+(\frac{\beta}{h}[p],[v])_{\Lambda_i}\Big)$$

$$=-\sum_{i=0}^{N}(T|T|,v)_{\Omega_i}.$$

Standard terms	Nitsche terms	Overlap terms

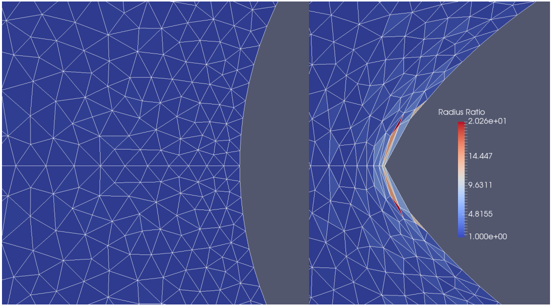
The adjoint system for overlapping meshes has the same stabilization at the artificial interface as the state equation at the interface

$$
\begin{aligned}
&\sum_{i=0}^{N} \Big((\lambda \nabla p, \nabla v)_{\Omega_i} - (cp, v)_{\Omega_i} \Big) \\
&+ (\alpha'(T)(T - T^{\mathrm{ex}})p, v)_{\Gamma^{\mathrm{ex}}} + (\alpha(T)p, v)_{\Gamma^{\mathrm{ex}}} \\
&+ \sum_{i=1}^{N} \Big(- (\langle \lambda \mathbf{n}_i \cdot \nabla p, \rangle [v])_{\Lambda_i} - ([p], \langle \lambda \mathbf{n}_i \cdot \nabla v \rangle)_{\Lambda_i} + (\frac{\beta}{h}[p], [v])_{\Lambda_i} \\
&+ ([\lambda \nabla p], [\nabla v])_{\Omega_{h,0} \cap \Omega_i} \Big) = - \sum_{i=0}^{N} (T|T|, v)_{\Omega_i}.
\end{aligned}
$$

Standard terms	Nitsche terms	Overlap terms

A Laplacian deformation scheme is not suited for large deformations

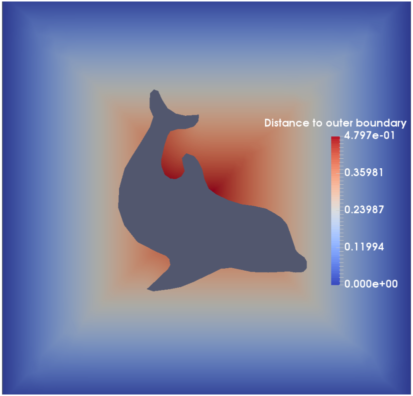
$$
\begin{aligned}
-\Delta w &= 0 \text{ in } \Omega, \\
w &= d \cdot n \text{ on } \Gamma, \\
w &= 0 \text{ on } \partial\Omega \backslash \Gamma.
\end{aligned}
$$



Γ	$d \cdot n$
Moving Boundary	Deformation

The Eikonal convection equation ensures better mesh-quality

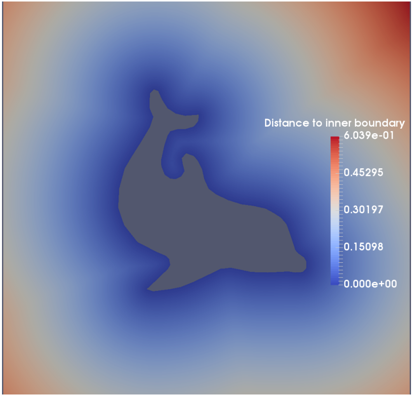
$$-h\Delta\epsilon_1 + ||\nabla\epsilon_1||_2^2 = 1 \text{ in } \Omega,$$
$$\epsilon_1 = 0 \text{ on } \partial\Omega\backslash\Gamma$$



Γ	ϵ_1	
Moving Boundary	Dist. to Γ	

The Eikonal convection equation ensures better mesh-quality

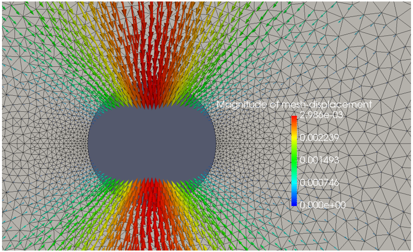
$$-h\Delta\epsilon_1 + ||\nabla\epsilon_1||_2^2 = 1 \text{ in } \Omega,$$
$$\epsilon_1 = 0 \text{ on } \partial\Omega\backslash\Gamma$$
$$-h\Delta\epsilon_2 + ||\nabla\epsilon_2||_2^2 = 1 \text{ in } \Omega,$$
$$\epsilon_2 = 0 \text{ on } \Gamma$$



Γ	ϵ_1	ϵ_2
Moving Boundary	Dist. to Γ	Dist. to $\partial\Omega\backslash\Gamma$.

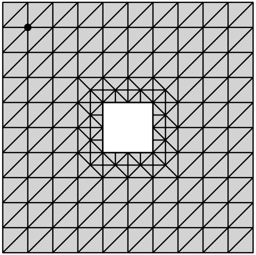
The Eikonal convection equation ensures better mesh-quality

$$
\begin{aligned}
-h\Delta\epsilon_1 + ||\nabla\epsilon_1||_2^2 &= 1 \text{ in } \Omega, \\
\epsilon_1 &= 0 \text{ on } \partial\Omega\backslash\Gamma \\
-h\Delta\epsilon_2 + ||\nabla\epsilon_2||_2^2 &= 1 \text{ in } \Omega, \\
\epsilon_2 &= 0 \text{ on } \Gamma \\
-\alpha\epsilon_2^2\Delta w + \mathrm{div}(\epsilon_1 w \otimes \nabla\epsilon_2) &= 0 \\
w &= d\cdot n \text{ on } \Gamma \\
w &= 0 \text{ on } \partial\Omega\backslash\Gamma.
\end{aligned}
$$



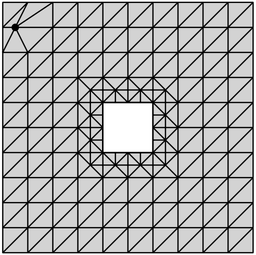
Γ	ϵ_1	ϵ_2
Moving Boundary	Dist. to Γ	Dist. to $\partial\Omega\backslash\Gamma$.

In the discrete case, the solution of a state equation u is dependent of volume nodes



$$J(u(\Omega), \Omega) = \int_{\Omega} u^2 \mathrm{d}\Omega$$

In the discrete case, the solution of a state equation u is dependent of volume nodes



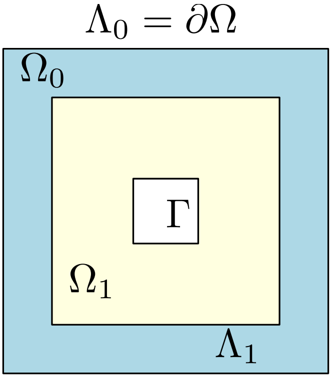
$$J(u(\Omega),\Omega) = \int_\Omega u^2 \mathrm{d}\Omega$$

$$\mathrm{d}J(u(\Omega),\Omega)[V] = \frac{\mathrm{d}}{\mathrm{d}\Omega}\left(\int_\Omega u^2 \mathrm{d}\Omega\right)[V],$$

$$V = \text{Displacement function}$$

The new strong formulation now has additional terms for continuity over the artificial interface Λ_1

$$
\begin{aligned}
-\Delta u_i &= f \text{ in } \Omega_i, \quad i = 0, 1, \\
u_1 + \frac{\partial u_1}{\partial n} &= 1 \text{ on } \Gamma, \\
u_0 + \frac{\partial u_0}{\partial n} &= 1 \text{ on } \partial\Omega \\
[u] &= 0 \text{ on } \Lambda_1, \\
\left[\frac{\partial u}{\partial n}\right] &= \text{ on } \Lambda_1,
\end{aligned}
$$



We need several extra terms to obtain a stable Finite Element scheme

$$0 = a_s(u, v) - l_s(v)$$

$$a_s(u, v) = (\nabla u, \nabla v)_\Omega + (u, v)_{\partial\Omega} + (u, v)_\Gamma,$$

$$l_s(v) = (f, v)_\Omega + (1, v)_{\partial\Omega} + (1, v)_\Gamma$$

We need several extra terms to obtain a stable Finite Element scheme

$$0 = a_s(u, v) - l_s(v)$$

$$a_s(u, v) = \sum_{i=0}^{1} [(\nabla u, \nabla v)_{\Omega_i}] + (u_0, v_0)_{\partial\Omega} + (u_1, v_1)_{\Gamma},$$

$$l_s(v) = \sum_{i=0}^{1} (f, v)_{\Omega_i} + (1, v_0)_{\partial\Omega} + (1, v_1)_{\Gamma}.$$

We need several extra terms to obtain a stable Finite Element scheme

$$0 = a_s(u, v) - l_s(v)$$

$$a_s(u, v) = \sum_{i=0}^{1} [(\nabla u, \nabla v)_{\Omega_i}] + (u_0, v_0)_{\partial\Omega} + (u_1, v_1)_{\Gamma},$$

$$l_s(v) = \sum_{i=0}^{1} (f, v)_{\Omega_i} + (1, v_0)_{\partial\Omega} + (1, v_1)_{\Gamma}.$$

$$a_N(u, v) = -(\langle \mathbf{n}_1 \cdot \nabla u \rangle, [v])_{\Lambda_1} - ([u_h], \langle \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([u], [v])_{\Lambda_1},$$

Nitsche terms	Jump	Average	
$a_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	

We need several extra terms to obtain a stable Finite Element scheme

$$
\begin{aligned}
0 &= a_s(u, v) - l_s(v) \\
a_s(u, v) &= \sum_{i=0}^{1} \left[(\nabla u, \nabla v)_{\Omega_i} \right] + (u_0, v_0)_{\partial\Omega} + (u_1, v_1)_{\Gamma}, \\
l_s(v) &= \sum_{i=0}^{1} (f, v)_{\Omega_i} + (1, v_0)_{\partial\Omega} + (1, v_1)_{\Gamma}.
\end{aligned}
$$

$$
a_N(u, v) = -(\langle \mathbf{n}_1 \cdot \nabla u \rangle, [v])_{\Lambda_1} - ([u_h], \langle \mathbf{n}_1 \cdot \nabla v \rangle)_{\Lambda_1} + \frac{\beta}{h}([u], [v])_{\Lambda_1},
$$

$$
a_O(u, v) = ([\nabla u], [\nabla v])_{\Omega_{h,0} \cap \Omega_1},
$$

Nitsche terms	Jump	Average	Stability on overlap
$a_N(u, v)$	$[w] = w_1 - w_0$	$\langle w \rangle = \frac{w_1 + w_0}{2}$	$a_O(u, v)$