A GENERAL TOPOLOGY OPTIMIZATION FRAMEWORK FOR POLYGONAL FINITE ELEMENT MESHES IN ARBITRARY DOMAINS

Ivan Menezes (PUC-Rio)

Anderson Pereira (PUC-Rio) - Cameron Talischi (UIUC) - Glaucio Paulino (UIUC)

WCSMO-10

MOTIVATION

MOTIVATION

- Traditionally, the topology optimization problem has been solved on uniform grids consisting of Lagrangian-type finite elements (e.g. **triangles and linear quads**)

MOTIVATION

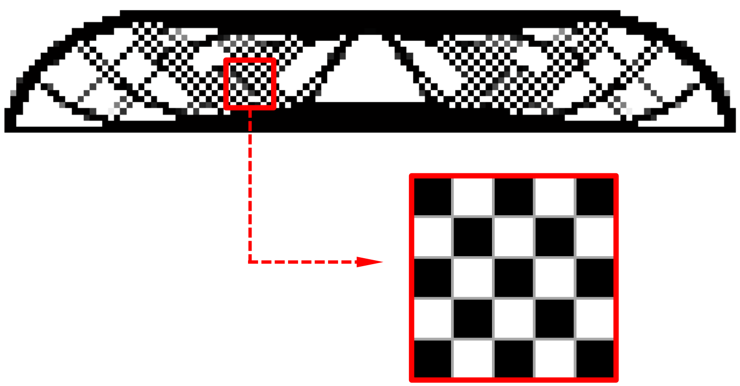
- Traditionally, the topology optimization problem has been solved on uniform grids consisting of Lagrangian-type finite elements (e.g. **triangles and linear quads**)

↳ **Numerical Instabilities**

MOTIVATION

- Traditionally, the topology optimization problem has been solved on uniform grids consisting of Lagrangian-type finite elements (e.g. **triangles and linear quads**)

Numerical Instabilities

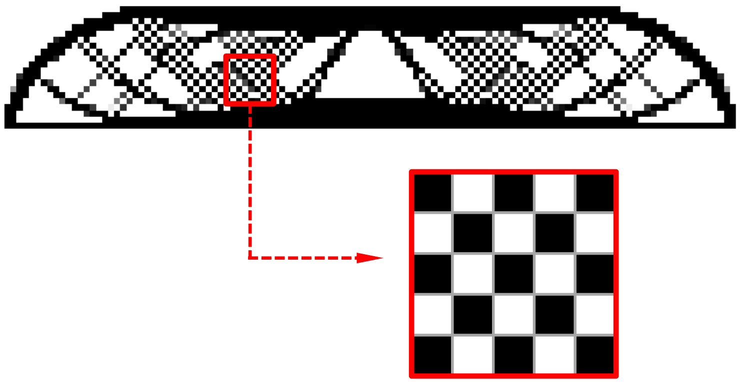


Checkerboard Patterns

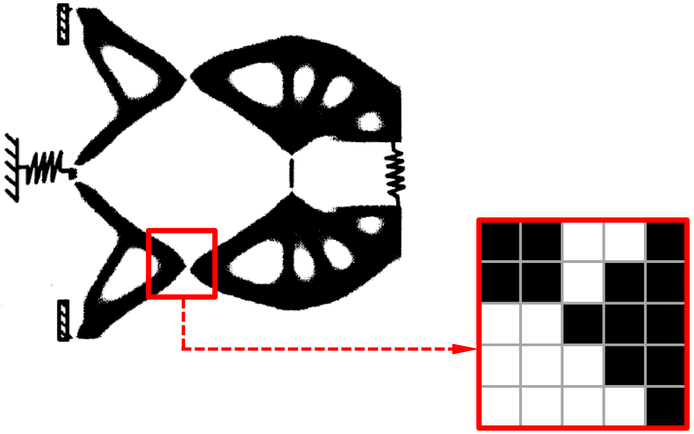
MOTIVATION

- Traditionally, the topology optimization problem has been solved on uniform grids consisting of Lagrangian-type finite elements (e.g. **triangles and linear quads**)

Numerical Instabilities



Checkerboard Patterns



One-node Connections

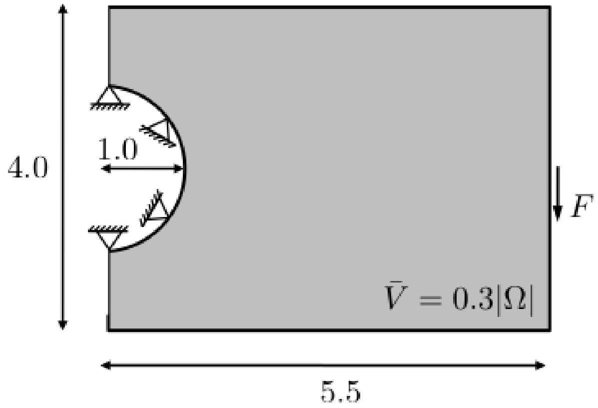
MOTIVATION

- Moreover, the use of simplex elements can cause **bias** in the orientation of members, leading to **mesh-dependent sub-optimal designs**

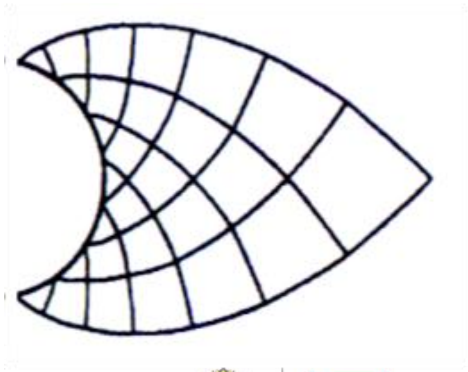
MOTIVATION

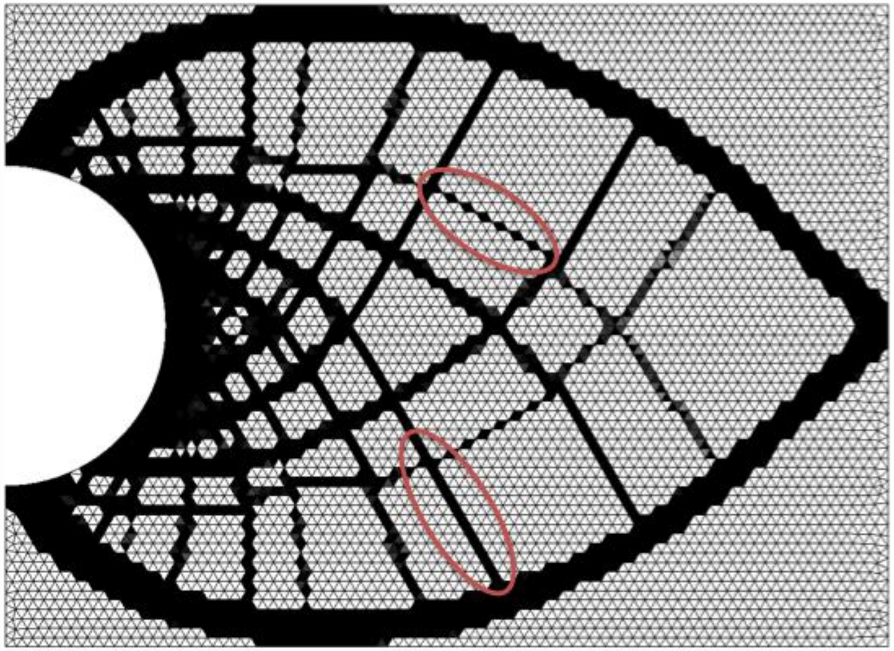
- Moreover, the use of simplex elements can cause **bias** in the orientation of members, leading to **mesh-dependent sub-optimal designs**

Example: Michell problem



Expected Solution





Solution obtained with **8584** T6 Elements

Main Contribution of this Research

MOTIVATION

Main Contribution of this Research

Investigate the behavior of **Polygonal Elements** in the context of Topology Optimization Theory

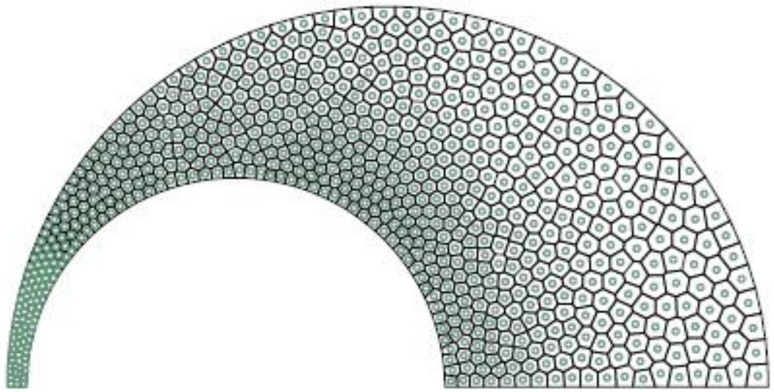
POLYGONAL FINITE ELEMENTS

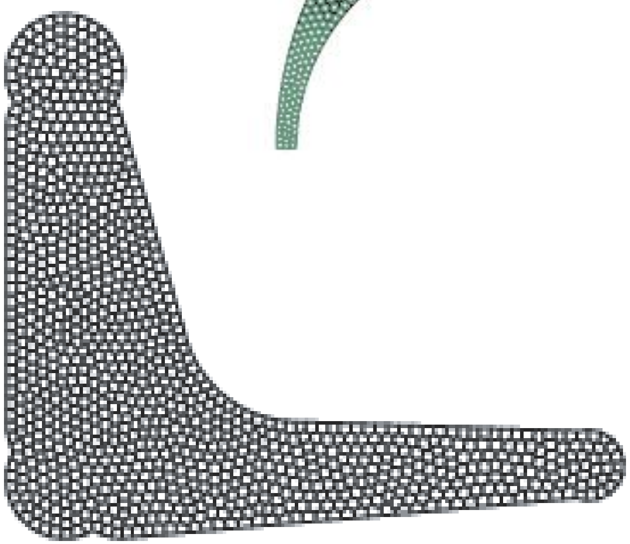
POLYGONAL FINITE ELEMENTS

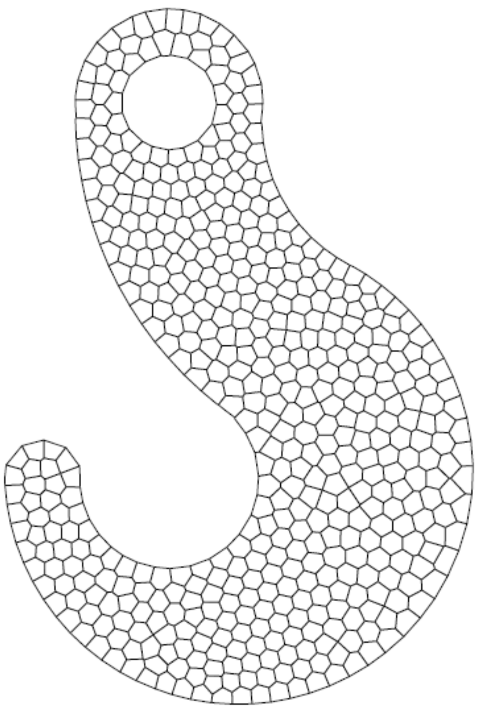
- ✓ **Provide great flexibility in discretizing complex domains**
- ✓ Naturally exclude checkerboard layouts and one-node connections
- ✓ Not biased by the standard FEM simplex geometry (triangles and quads)
- ✓ **Present "better" finite element solution of elasticity problems**
- ✓ **Stable elements w.r.t. Babuska-Brezzi condition**

POLYGONAL FINITE ELEMENTS

Provide great flexibility in discretizing complex domains







POLYGONAL FINITE ELEMENTS

- ✓ Provide great flexibility in discretizing complex domains
- ✓ **Naturally exclude checkerboard layouts and one-node connections**
- ✓ Not biased by the standard FEM simplex geometry (triangles and quads)
- ✓ **Present "better" finite element solution of elasticity problems**
- ✓ **Stable elements w.r.t. Babuska-Brezzi condition**

POLYGONAL FINITE ELEMENTS

Naturally exclude checkerboard layouts and one-node connections

Q4 Elements

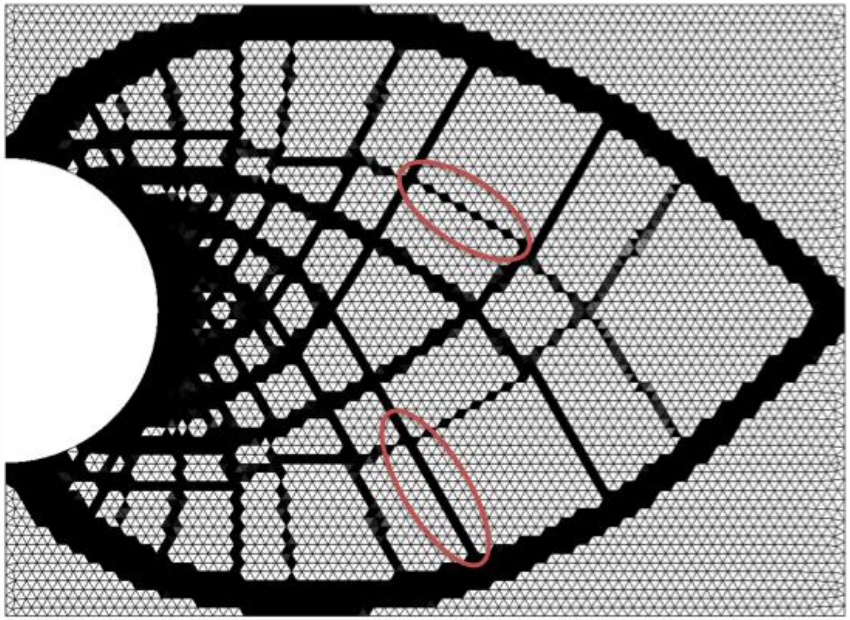
Polygonal Elements

POLYGONAL FINITE ELEMENTS

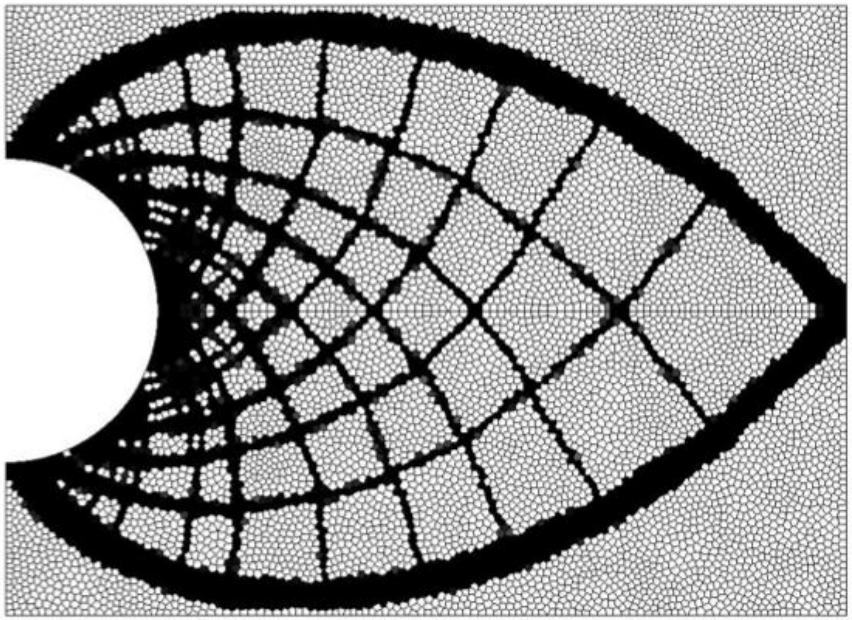
- ✓ Provide great flexibility in discretizing complex domains
- ✓ Naturally exclude checkerboard layouts and one-node connections
- ✓ **Not biased by the standard FEM simplex geometry (triangles and quads)**
- ✓ **Present "better" finite element solution of elasticity problems**
- ✓ **Stable elements w.r.t. Babuska-Brezzi condition**

POLYGONAL FINITE ELEMENTS

Not biased by the standard FEM simplex geometry
(based on triangles and quads)



T6 Elements



Polygonal Elements

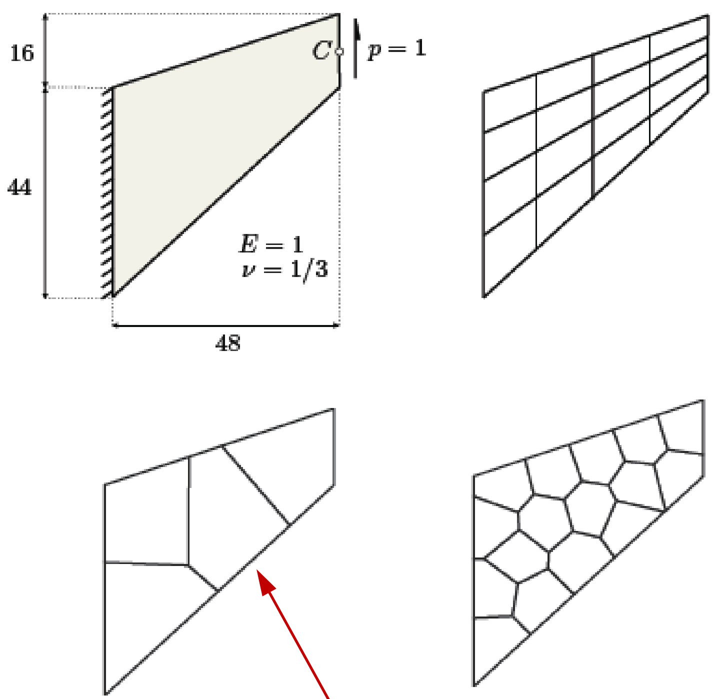
Talischi, C. , Paulino, G.H., Pereira, A. and Menezes, I.F.M., "*Polygonal Finite Elements for Topology Optimization: A Unifying Paradigm*", **IJNME**, 82(6):671-698, **2010.**

POLYGONAL FINITE ELEMENTS

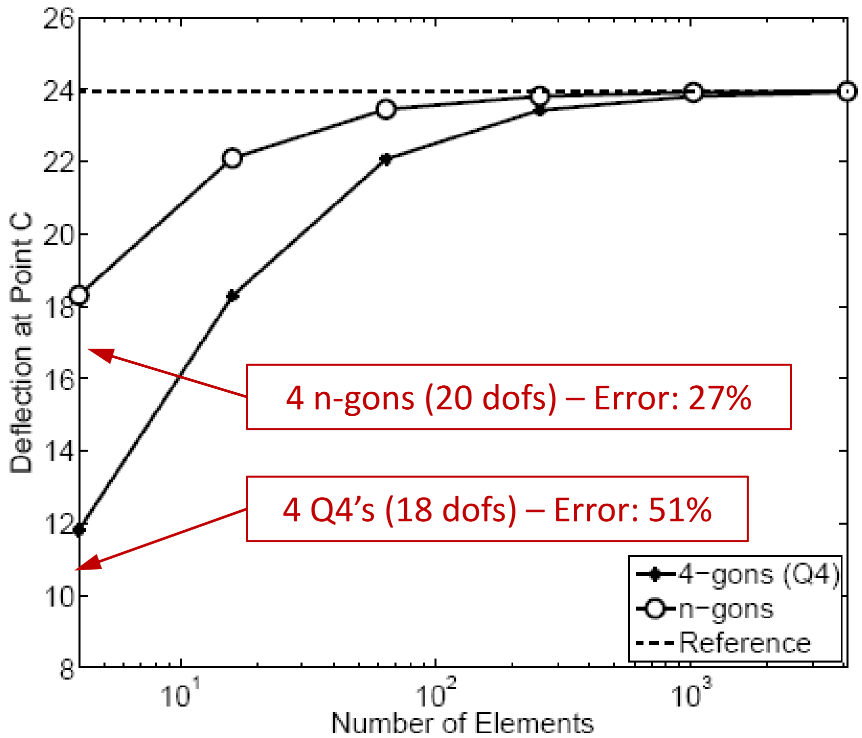
- ✓ Provide great flexibility in discretizing complex domains
- ✓ Naturally exclude checkerboard layouts and one-node connections
- ✓ Not biased by the standard FEM simplex geometry (triangles and quads)
- ✓ **Present “better” finite element solution of elasticity problems**
- ✓ **Stable elements w.r.t. Babuska-Brezzi condition**

POLYGONAL FINITE ELEMENTS

Present “better” finite element solution of elasticity problems



3 quads and 1 pentagon



Cook, Malkus and Plesha, “*Concepts and Applications of FE Analysis”* (4th Edn), Wiley, New York, **2002**.

POLYGONAL FINITE ELEMENTS

- ✓ Provide great flexibility in discretizing complex domains
- ✓ Naturally exclude checkerboard layouts and one-node connections
- ✓ Not biased by the standard FEM simplex geometry (triangles and quads)
- ✓ Present “better” finite element solution of elasticity problems
- ✓ **Stable elements w.r.t. Babuska-Brezzi condition**

Stable elements w.r.t. Babuska-Brezzi condition

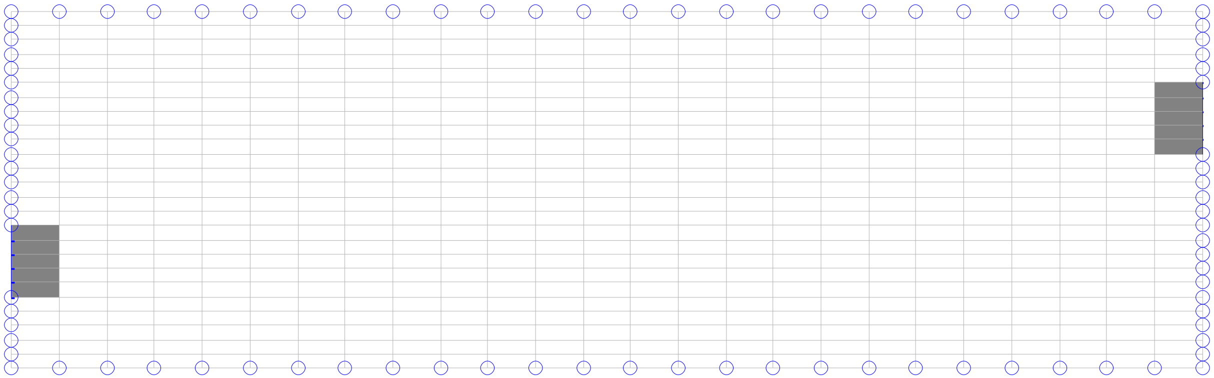
In-Out Flow Problem*: Geometry Description

Interpolation Scheme: **Linear** variation for the Velocity Field and **Constant** Pressure

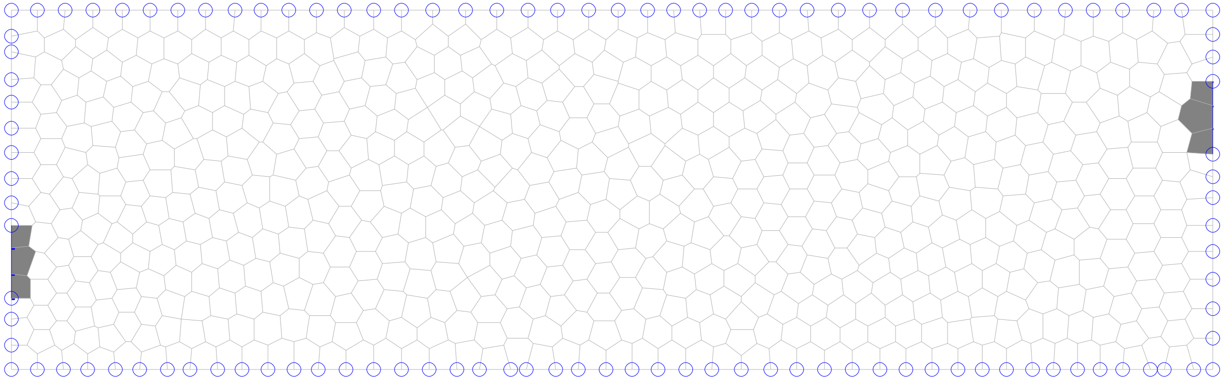
** Mixed Variational Formulation (pressure-velocity) of the Stokes Flow Problem*

Stable elements w.r.t. Babuska-Brezzi condition

Finite Element Meshes



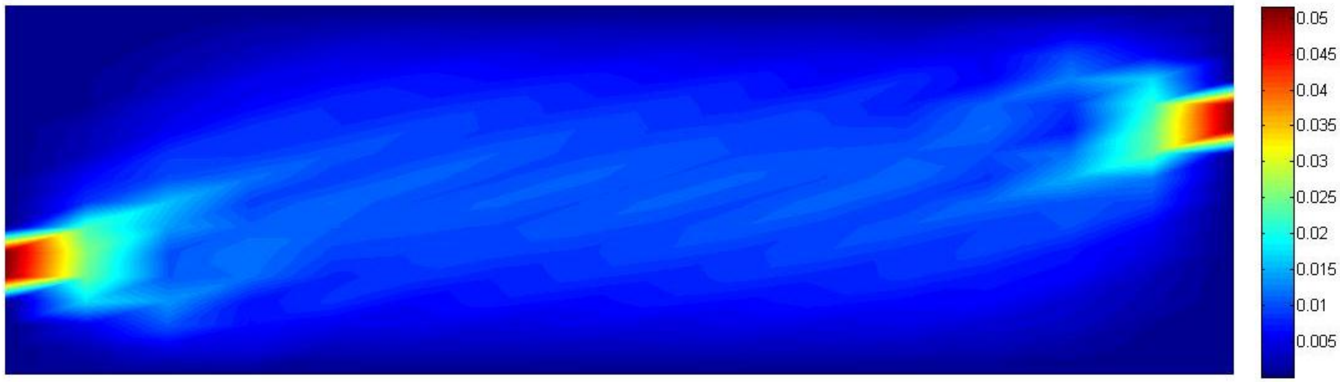
Q4 Elements



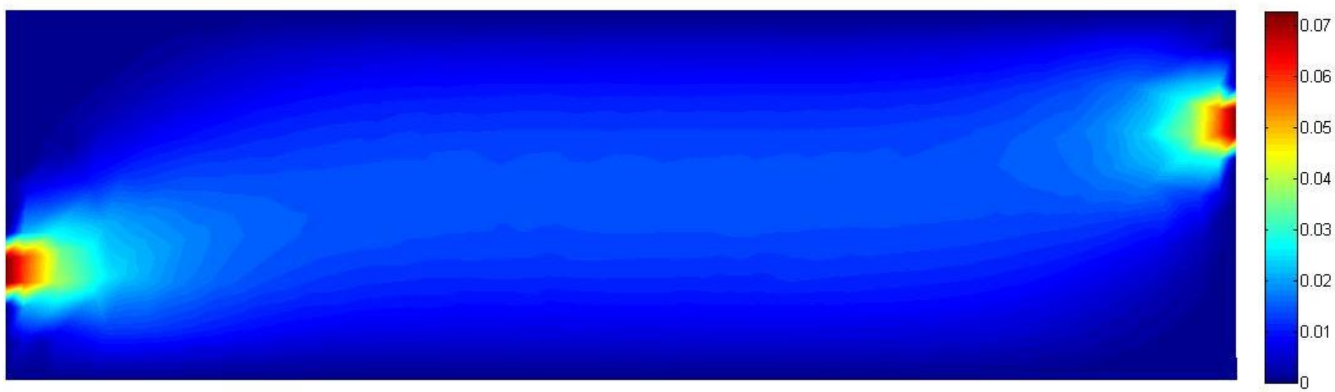
Polygonal Elements

Stable elements w.r.t. Babuska-Brezzi condition

Velocity Fields



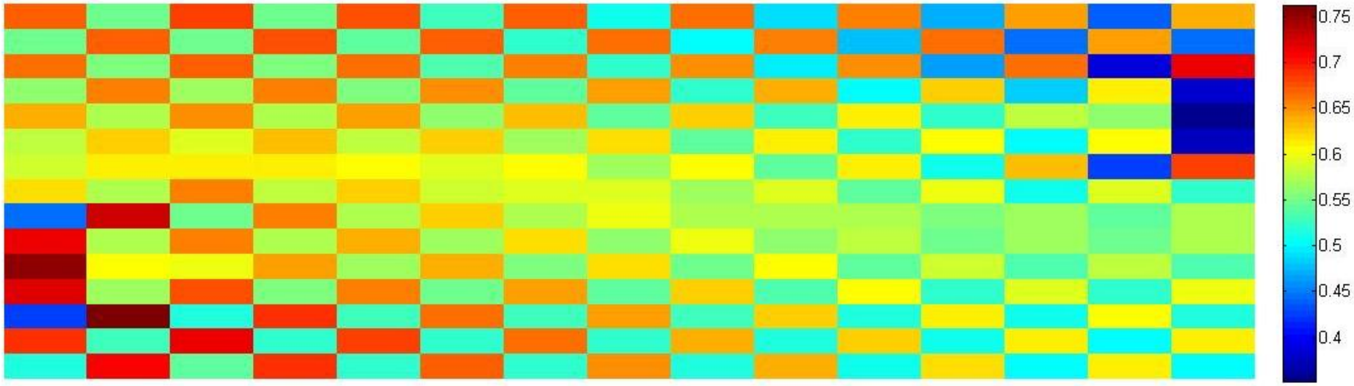
Q4 Elements



Polygonal Elements

Stable elements w.r.t. Babuska-Brezzi condition

Pressure Fields



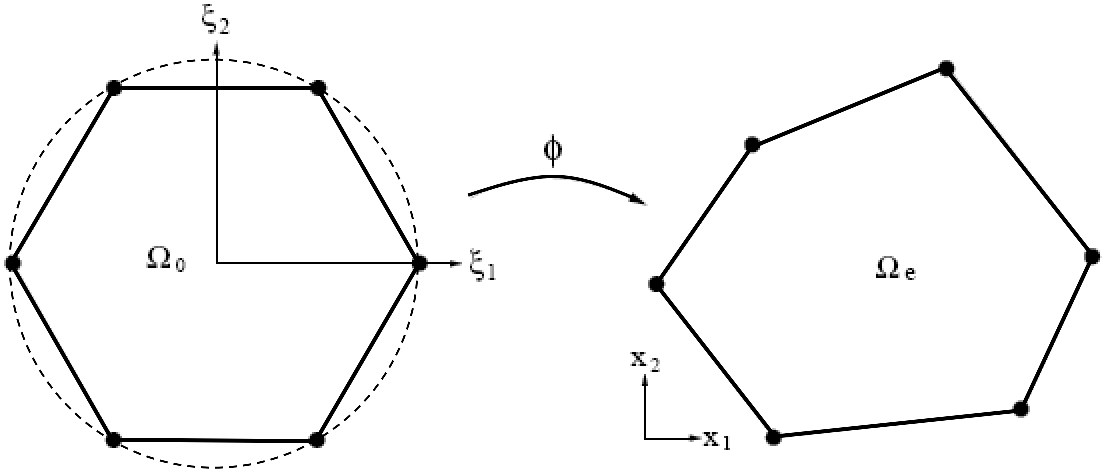
Q4 Elements

Polygonal Elements

ELEMENT FORMULATION

ELEMENT FORMULATION

Constructed using **Laplace Shape Functions**

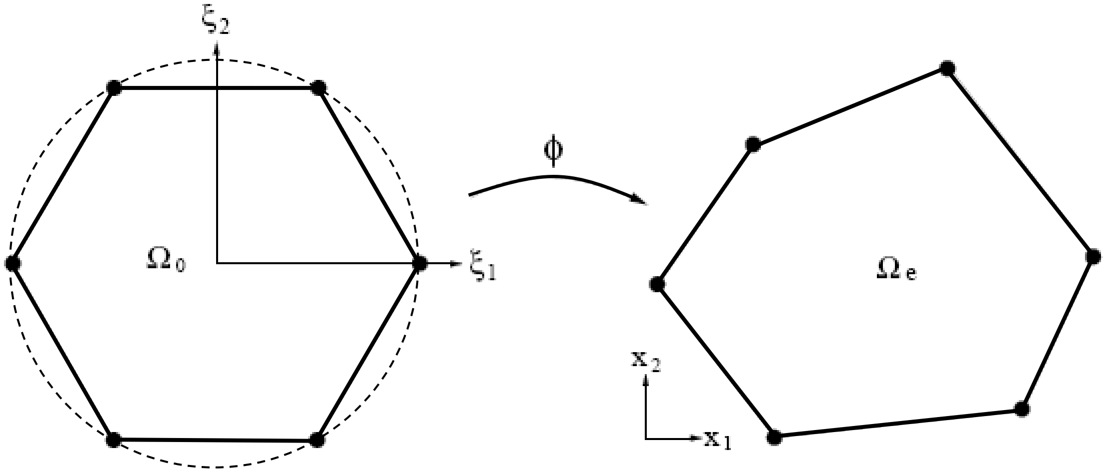


Isoparametric Mapping

Sukumar, N. and Malsch, E.A., "*Recent Advances in the Construction of Polygonal Finite Element Interpolants*", Archives of Computational Methods in Engineering, **13**(1), 129-163, **2006.**

ELEMENT FORMULATION

Constructed using **Laplace Shape Functions**



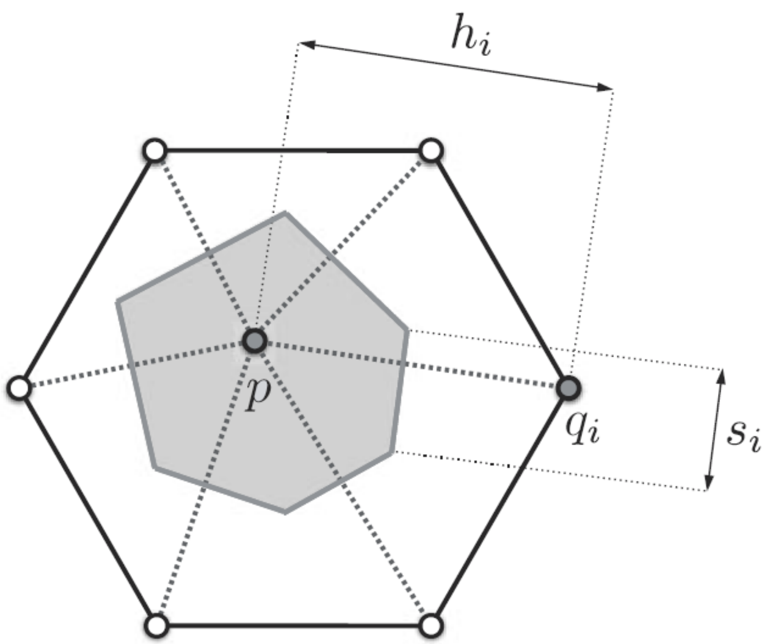
Isoparametric Mapping

“can be viewed as an extension of the common triangles and linear quads to all convex n-gons”

Sukumar, N. and Malsch, E.A., “*Recent Advances in the Construction of Polygonal Finite Element Interpolants*”, Archives of Computational Methods in Engineering, **13**(1), 129-163, **2006.**

ELEMENT FORMULATION

Laplace Shape Functions



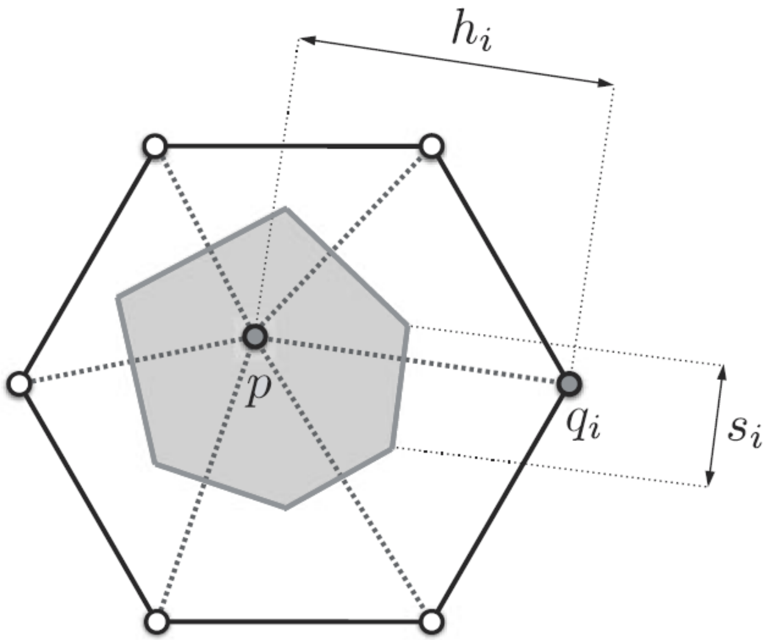
$$\phi_i(\boldsymbol{x}) = \frac{w_i(\boldsymbol{x})}{\sum_j w_j(\boldsymbol{x})}$$

$$w_i(\boldsymbol{x}) = \frac{s_i(\boldsymbol{x})}{h_i(\boldsymbol{x})}$$

Properties:

$$\begin{cases} 0 \leqslant \phi_i(\boldsymbol{x}) \leqslant 1 \\ \sum_i \phi_i(\boldsymbol{x}) = 1 \end{cases}$$

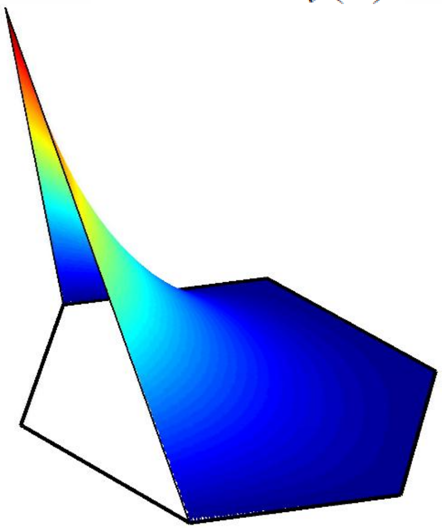
Laplace Shape Functions



$$\phi_i(\boldsymbol{x}) = \frac{w_i(\boldsymbol{x})}{\sum_j w_j(\boldsymbol{x})}$$

$$w_i(\boldsymbol{x}) = \frac{s_i(\boldsymbol{x})}{h_i(\boldsymbol{x})}$$

Properties:
$$\begin{cases} 0 \leqslant \phi_i(\boldsymbol{x}) \leqslant 1 \\ \sum_i \phi_i(\boldsymbol{x}) = 1 \end{cases}$$



Typical Laplace Shape Function

PROPOSED FRAMEWORK

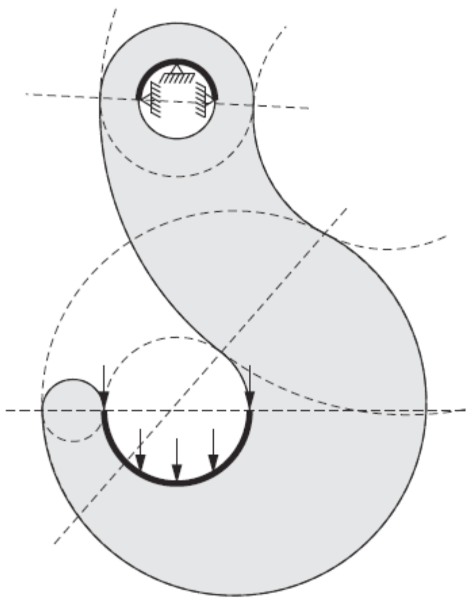
PROPOSED FRAMEWORK

PolyTop† – General Topology Optimization Framework using Unstructured Polygonal Finite Element Meshes

† Talischi, C., Paulino, G.H., Pereira, A., and Menezes, I.F.M., "*PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes*", **JSMO**, 45:329–357, **2012.** doi:10.1007/s00158-011-0696-x

PROPOSED FRAMEWORK

PolyTop† – General Topology Optimization Framework using Unstructured Polygonal Finite Element Meshes

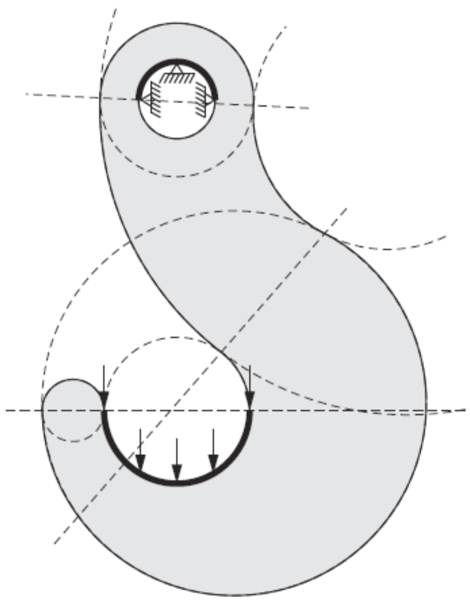


Geometry & BC's

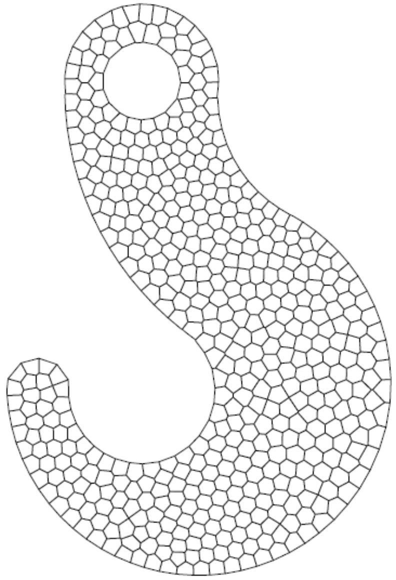
† Talischi, C., Paulino, G.H., Pereira, A., and Menezes, I.F.M., "*PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes*", **JSMO**, 45:329–357, **2012.** doi:10.1007/s00158-011-0696-x

PROPOSED FRAMEWORK

PolyTop† – General Topology Optimization Framework using Unstructured Polygonal Finite Element Meshes



Geometry & BC's

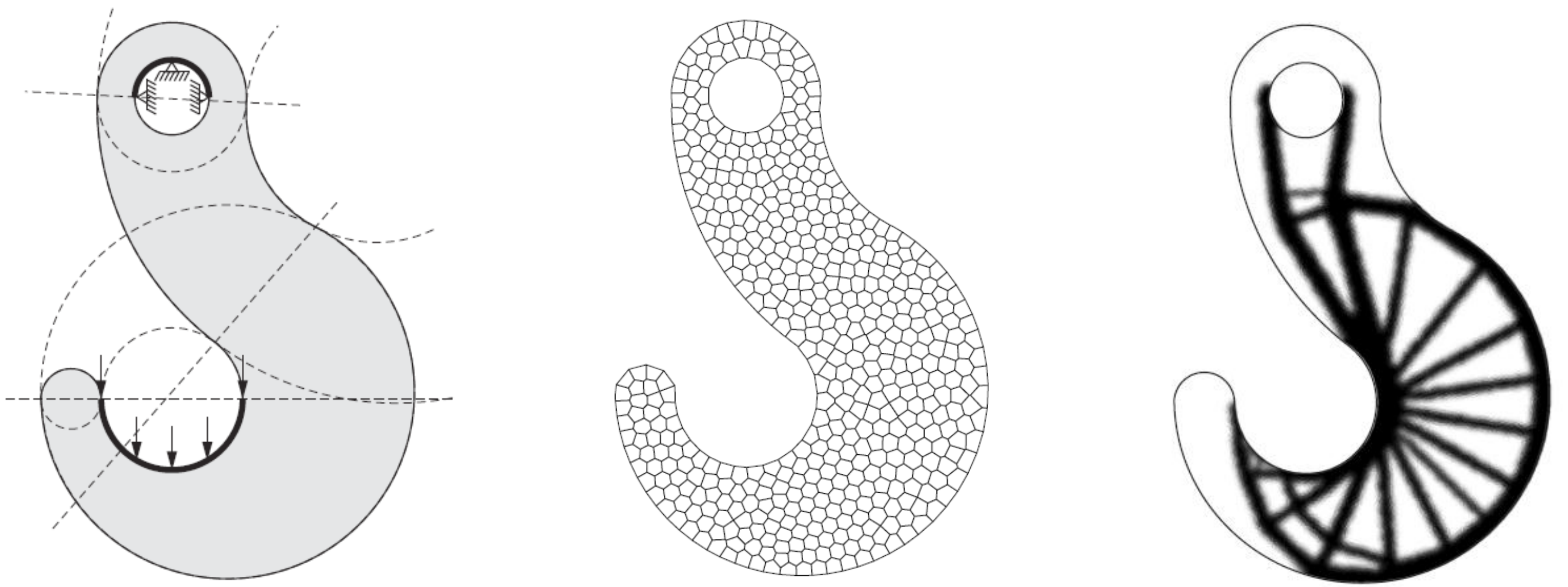


Polygonal Mesh

† Talischi, C., Paulino, G.H., Pereira, A., and Menezes, I.F.M., "*PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes*", **JSMO**, 45:329–357, **2012.** doi:10.1007/s00158-011-0696-x

PROPOSED FRAMEWORK

PolyTop† – General Topology Optimization Framework using Unstructured Polygonal Finite Element Meshes



Geometry & BC's   Polygonal Mesh   PolyTop

† Talischi, C., Paulino, G.H., Pereira, A., and Menezes, I.F.M., "*PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes*", **JSMO**, 45:329–357, **2012.** doi:10.1007/s00158-011-0696-x

PolyTop FEATURES

PolyTop FEATURES

- Developed in MATLAB
- Possesses **189** lines, of which 116 lines pertain to the Finite Element Analysis (including 81 lines for the element stiffness calculations for polygonal elements)

PolyTop FEATURES

- Developed in MATLAB

- Possesses **189** lines, of which 116 lines pertain to the Finite Element Analysis (including 81 lines for the element stiffness calculations for polygonal elements)

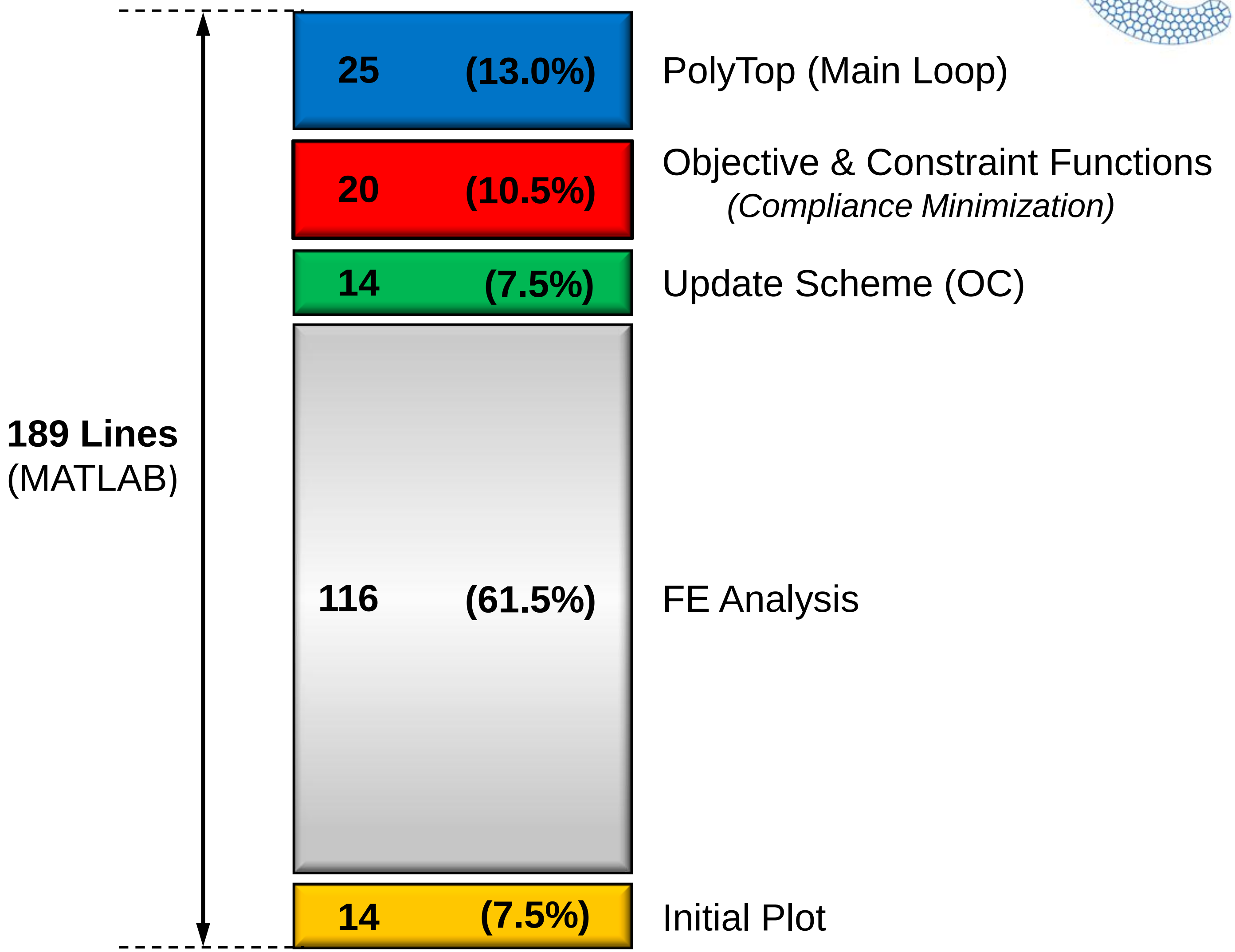
- Modular Framework

  - analysis routine and the optimization algorithm are separated from the specific choice of topology optimization formulation

  - the finite element and sensitivity analysis routines contain no information related to the formulation and thus can be extended, developed and modified independently

CODE STRUCTURE

CODE STRUCTURE



CODE STRUCTURE

Input Data Structures

The input to the main kernel PolyTop consists of two MATLAB structure arrays containing the analysis and optimization fields:

fem	opt
fem.NNode	opt.zMin
fem.NElem	opt.zMax
fem.Node	opt.zIni
fem.Element	opt.MatIntFnc
fem.Supp	opt.P
fem.Load	opt.MaxIter
fem.ShapeFnc	opt.Tol
...	...

CODE MODULARITY AND FLEXIBILITY

CODE MODULARITY AND FLEXIBILITY

- **Material interpolation functions (e.g. SIMP, RAMP)**
- Different optimizers (e.g. OC, MMA, SLP)
- Analysis routine (e.g. FEM or other method)
- Objective Function (e.g. Compliance, Compliant Mechanism)

CODE MODULARITY AND FLEXIBILITY

Material Interpolation Functions

```
function [E,dEdy,V,dVdy] = MatIntFnc(y,type,param)
eps = 1e-4;  %Ersatz stiffness
switch(type)
  case('SIMP')
    penal = param;
    E = eps+(1-eps)*y.^penal;
    V = y;
    dEdy = (1-eps)*penal*y.^(penal-1);
    dVdy = ones(size(y,1),1);
  case('RAMP')
    q = param;
    E = eps+(1-eps)*y./(1+q*(1-y));
    V = y;
    dEdy = ((1-eps)*(q+1))./(q-q*y+1).^2;
    dVdv = ones(size(y));
end
```

CODE MODULARITY AND FLEXIBILITY

- Material **interpolation functions** (e.g. SIMP, RAMP)
- **Different optimizers (e.g. OC, MMA, SLP)**
- Analysis routine (e.g. FEM or other method)
- Objective Function (e.g. Compliance, Compliant Mechanism)

Select a Different Optimizer

```
%-------------------------------------------- OPTIMALITY CRITERIA UPDATE
function [zNew,Change] = UpdateScheme(dfdz,g,dgdz,z0,opt)
zMin=opt.zMin; zMax=opt.zMax;
move=opt.OCMove*(zMax-zMin); eta=opt.OCEta;
l1=0; l2=1e6;
while l2-l1 > 1e-4
  lmid = 0.5*(l1+l2);
  B = -(dfdz./dgdz)/lmid;
  zCnd = zMin+(z0-zMin).*B.^eta;
  zNew = max(max(min(min(zCnd,z0+move),zMax),z0-move),zMin);
  if (g+dgdz'*(zNew-z0)>0),  l1=lmid;
  else                        l2=lmid;  end
end
Change = max(abs(zNew-z0))/(zMax-zMin);
```

CODE MODULARITY AND FLEXIBILITY

- Material **interpolation functions** (e.g. SIMP, RAMP)
- Different **optimizers** (e.g. OC, MMA, SLP)
- **Analysis routine (e.g. FEM or other method)**
- Objective Function (e.g. Compliance, Compliant Mechanism)

CODE MODULARITY AND FLEXIBILITY

- Material **interpolation functions** (e.g. SIMP, RAMP)
- Different **optimizers** (e.g. OC, MMA, SLP)
- Analysis routine (e.g. FEM or other method)
- **Objective Function (e.g. Compliance, Compliant Mechanism)**

Change the Objective Function

Compliance:
(self-adjoint problem)

$$f = \mathbf{F}^T\mathbf{U} \quad ; \quad \frac{\partial f}{\partial \mathbf{E}} = -\mathbf{U}^T \frac{\partial \mathbf{K}}{\partial \mathbf{E}} \mathbf{U} = -\sum \mathbf{U}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$$

Compliant:
(non self-adjoint problem)

$$f = \mathbf{L}^T\mathbf{U} \quad ; \quad \left.\begin{array}{r} \dfrac{\partial f}{\partial \mathbf{E}} = -\boldsymbol{\lambda}^T \dfrac{\partial \mathbf{K}}{\partial \mathbf{E}} \mathbf{U} \\ \\ \mathbf{K}\boldsymbol{\lambda} = \mathbf{L} \end{array}\right\} -\sum \boldsymbol{\lambda}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$$

CODE MODULARITY AND FLEXIBILITY

Change the Objective Function

Compliance:
(self-adjoint problem)

$$f = \mathbf{F}^T\mathbf{U} \quad ; \quad \frac{\partial f}{\partial \mathbf{E}} = -\mathbf{U}^T \frac{\partial \mathbf{K}}{\partial \mathbf{E}} \mathbf{U} = -\sum \mathbf{U}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$$

Compliant:
(non self-adjoint problem)

$$\left.\begin{array}{l} f = \mathbf{L}^T\mathbf{U} \quad ; \quad \dfrac{\partial f}{\partial \mathbf{E}} = -\boldsymbol{\lambda}^T \dfrac{\partial \mathbf{K}}{\partial \mathbf{E}} \mathbf{U} \\ \\ \mathbf{K}\boldsymbol{\lambda} = \mathbf{L} \end{array}\right\} -\sum \boldsymbol{\lambda}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$$

PolyTop can be easily extended to handle compliant mechanism design by changing **10** existing lines and adding **7** new ones!

CODE MODULARITY AND FLEXIBILITY

Change the Objective Function

```
< f = dot(fem.F,U);
< temp = cumsum(-U(fem.i).*fem.k.*U(fem.j));
< dfdE = [temp(1);temp(2:end)-temp(1:end-1)];
< while l2-l1 > 1e-4
<   B = -(dfdz./dgdz)/lmid;
<   fem.F = zeros(2*fem.NNode,1);  %external load vector
<   fem.F(2*fem.Load(1:NLoad,1)-1) = fem.Load(1:NLoad,2);
<   fem.F(2*fem.Load(1:NLoad,1))   = fem.Load(1:NLoad,3);







< K = sparse(fem.i,fem.j,E(fem.e).*fem.k);
< U = zeros(2*fem.NNode,1);
```

Compliance

```
> f = fem.DOut(3)*U(fem.DofDOut,1);
> temp = cumsum(fem.DOut(3)*U(fem.i,1).*fem.k.*U(fem.j,2));
> dfdE = -[temp(1);temp(2:end)-temp(1:end-1)];
> while (l2-l1)/(l2+l1) > 1e-4 && l2>1e-40
>   B = max(1e-10,-(dfdz./dgdz)/lmid);
>   fem.F = zeros(2*fem.NNode,2);
>   fem.F(2*fem.Load(1:NLoad,1)-1,1) = fem.Load(1:NLoad,2);
>   fem.F(2*fem.Load(1:NLoad,1),1)   = fem.Load(1:NLoad,3);
>   fem.DofDOut = 2*fem.DOut(1)-2+fem.DOut(2);
>   fem.F(fem.DofDOut,2) = -1;
>   NSpring = size(fem.Spring,1);
>   s = zeros(2*fem.NNode,2);  %spring vector
>   s(2*fem.Spring(1:NSpring,1)-1) = fem.Spring(1:NSpring,2);
>   s(2*fem.Spring(1:NSpring,1))   = fem.Spring(1:NSpring,3);
>   fem.s = spdiags(s(:),0,2*fem.NNode,2*fem.NNode);
> K = sparse(fem.i,fem.j,E(fem.e).*fem.k) + fem.s;
> U = zeros(2*fem.NNode,2);
```

Compliant Mechanism

CODE MODULARITY AND FLEXIBILITY

Change the Objective Function

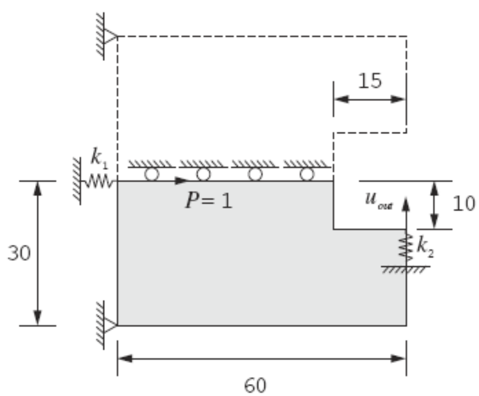
```
< f = dot(fem.F,U);
< temp = cumsum(-U(fem.i).*fem.k.*U(fem.j));
< dfdE = [temp(1);temp(2:end)-temp(1:end-1)];
< while l2-l1 > 1e-4
<   B = -(dfdz./dgdz)/lmid;
<   fem.F = zeros(2*fem.NNode,1);  %external load vector
<   fem.F(2*fem.Load(1:NLoad,1)-1) = fem.Load(1:NLoad,2);
<   fem.F(2*fem.Load(1:NLoad,1))   = fem.Load(1:NLoad,3);

< K = sparse(fem.i,fem.j,E(fem.e).*fem.k);
< U = zeros(2*fem.NNode,1);
```

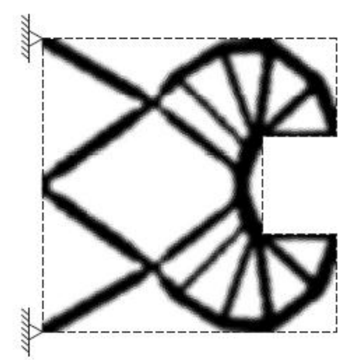
Compliance

```
> f = fem.DOut(3)*U(fem.DofDOut,1);
> temp = cumsum(fem.DOut(3)*U(fem.i,1).*fem.k.*U(fem.j,2));
> dfdE = -[temp(1);temp(2:end)-temp(1:end-1)];
> while (l2-l1)/(l2+l1) > 1e-4 && l2>1e-40
>   B = max(1e-10,-(dfdz./dgdz)/lmid);
>   fem.F = zeros(2*fem.NNode,2);
>   fem.F(2*fem.Load(1:NLoad,1)-1,1) = fem.Load(1:NLoad,2);
>   fem.F(2*fem.Load(1:NLoad,1),1)   = fem.Load(1:NLoad,3);
>   fem.DofDOut = 2*fem.DOut(1)-2+fem.DOut(2);
>   fem.F(fem.DofDOut,2) = -1;
>   NSpring = size(fem.Spring,1);
>   s = zeros(2*fem.NNode,2);  %spring vector
>   s(2*fem.Spring(1:NSpring,1)-1) = fem.Spring(1:NSpring,2);
>   s(2*fem.Spring(1:NSpring,1))   = fem.Spring(1:NSpring,3);
>   fem.s = spdiags(s(:),0,2*fem.NNode,2*fem.NNode);
> K = sparse(fem.i,fem.j,E(fem.e).*fem.k) + fem.s;
> U = zeros(2*fem.NNode,2);
```

Compliant Mechanism

Example
(Compliant Mechanism):

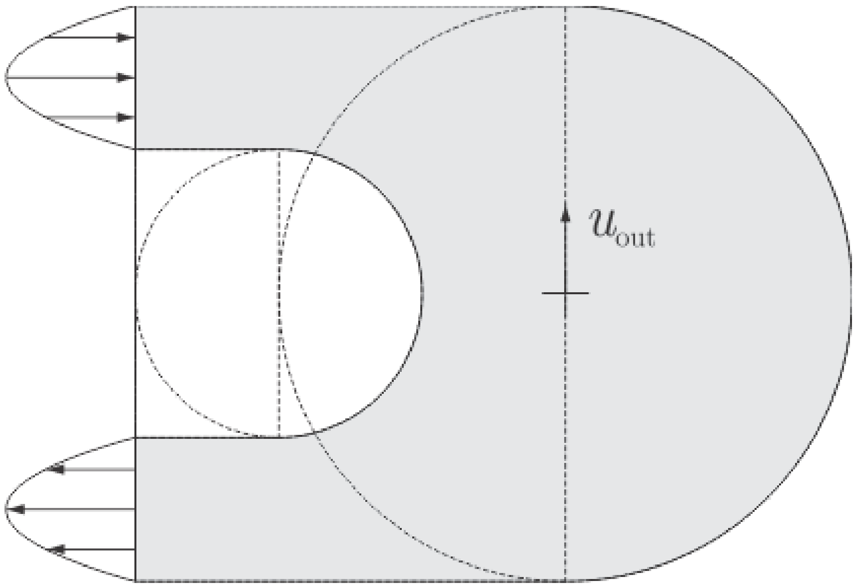




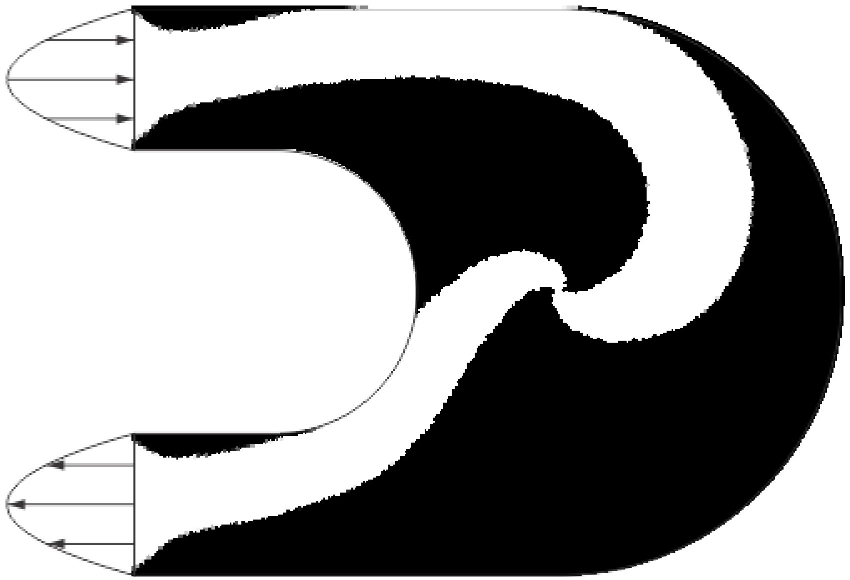
CODE MODULARITY AND FLEXIBILITY

Example: Fluid Mechanics (Stokes Flow)

Maximize the y-velocity at a given location:



Problem Description

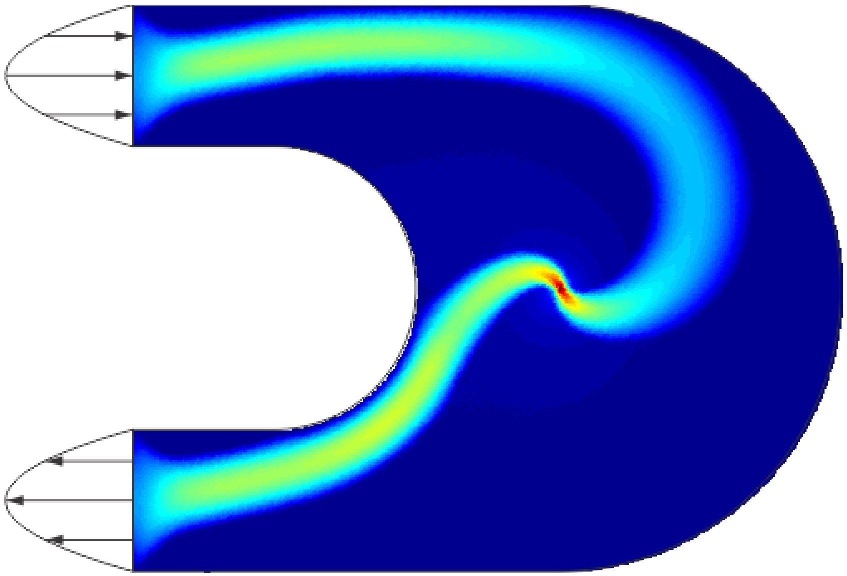


Optimal Solution

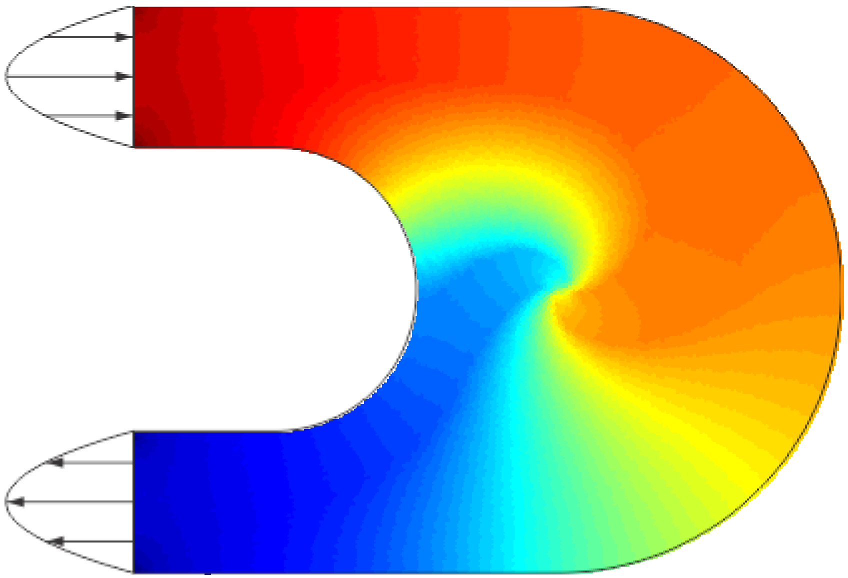
CODE MODULARITY AND FLEXIBILITY

Example: Fluid Mechanics (Stokes Flow)

Maximize the y-velocity at a given location:



Velocity Field



Pressure Field

CODE EFFICIENCY

CODE EFFICIENCY

Comparison with the 88-line code*

Mesh Size	90x30	150x50	300x100	600x200
PolyTop	11.9	**31.5**	**135.5**	**764.1**
88-line	**10.9**	33.0	252.2	3092.9

(time in sec for 200 optimization iterations)

* Andreassen E., Clausen A., Schevenels M., Lazarov B., Sigmund O., "*Efficient topology optimization in MATLAB using 88 lines of code*", **JSMO**, 43(1):1–16, **2011.** doi:10.1007/s00158-010-0594-7

CODE EFFICIENCY

Comparison with the 88-line code*

Mesh Size	90x30	150x50	300x100	600x200
PolyTop	11.9	31.5	135.5	764.1
88-line†	**9.7**	**24.3**	**119.7**	**708.8**

(time in sec for 200 optimization iterations)

† Design Volume
(OC Update Function)

$$V(\mathbf{z}) = \sum_{\ell=1}^{N} (\mathbf{Pz})_\ell = \mathbf{1}^T(\mathbf{Pz}) = \left(\mathbf{1}^T\mathbf{P}\right)\mathbf{z} = \left(\mathbf{P}^T\mathbf{1}\right)^T\mathbf{z}$$

* Andreassen E., Clausen A., Schevenels M., Lazarov B., Sigmund O., "*Efficient topology optimization in MATLAB using 88 lines of code*", **JSMO**, 43(1):1–16, **2011.** doi:10.1007/s00158-010-0594-7

MESH GENERATION – BASIC IDEAS

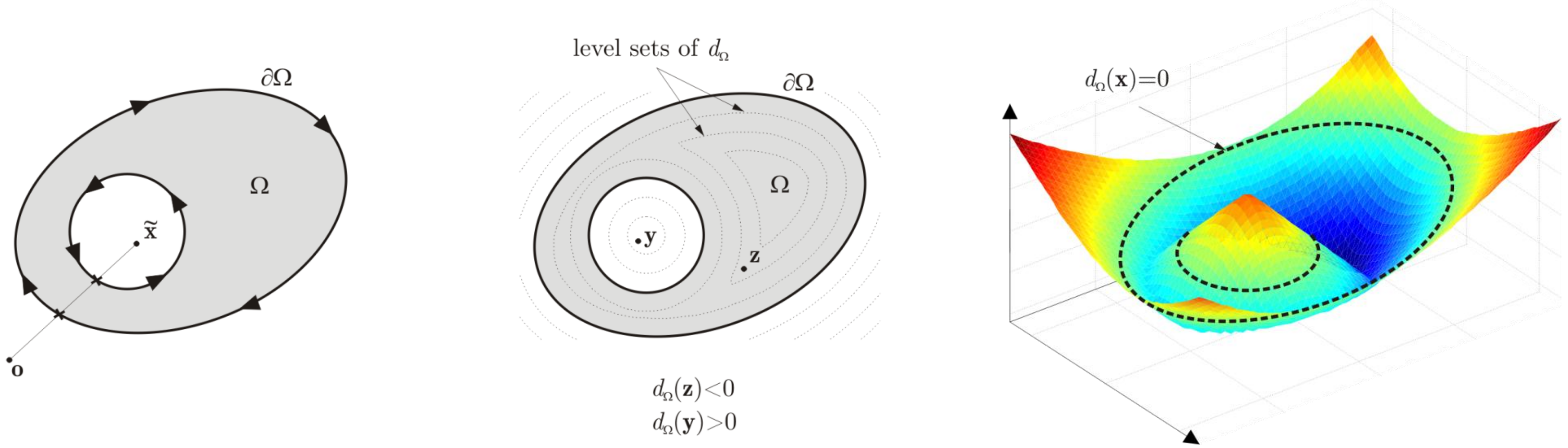
MESH GENERATION – BASIC IDEAS

- **Based on the Concept of Voronoi Tesselation**
- **Implicit Representation (signed distance functions)**

MESH GENERATION – BASIC IDEAS

Implicit Representation

- The implicit representation of the domain is one of the main ingredients of our meshing algorithm:



- The signed distance function contains all the essential information about the meshing domain needed in our mesh algorithm:

$$d_\Omega(\mathbf{x}) = s_\Omega(\mathbf{x}) \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\| \qquad s_\Omega(\mathbf{x}) := \begin{cases} -1, & \mathbf{x}\in\Omega \\ +1, & \mathbf{x}\in\mathbb{R}^2\backslash\Omega \end{cases}$$

PROPOSED FRAMEWORK

PROPOSED FRAMEWORK

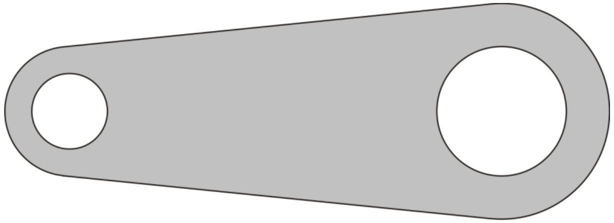
PolyMesher† – General-purpose Mesh Generator for Polygonal Elements

- Developed in MATLAB
- Possesses **133** lines, of which 69 lines pertain to the mesh post processing

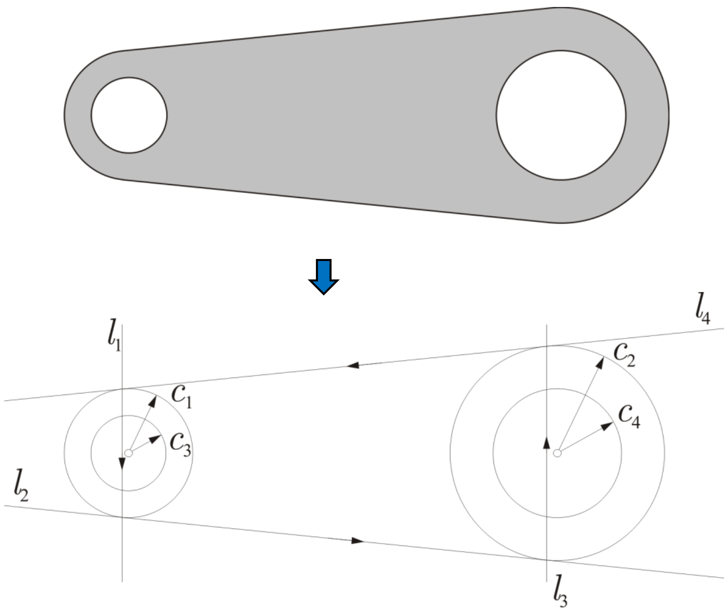
† Talischi, C., Paulino, G.H., Pereira, A., and Menezes, I.F.M., "*PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab*", **JSMO**, 45:309–328, **2012.** doi:10.1007/s00158-011-0706-z

PolyMesher EXAMPLE

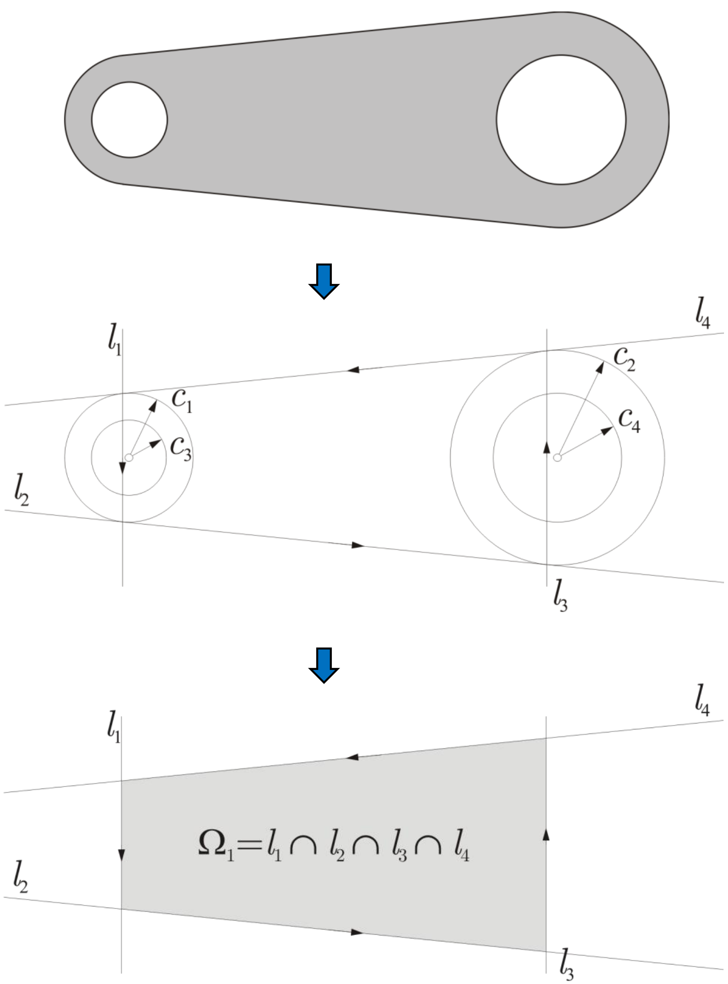
PolyMesher EXAMPLE: WRENCH



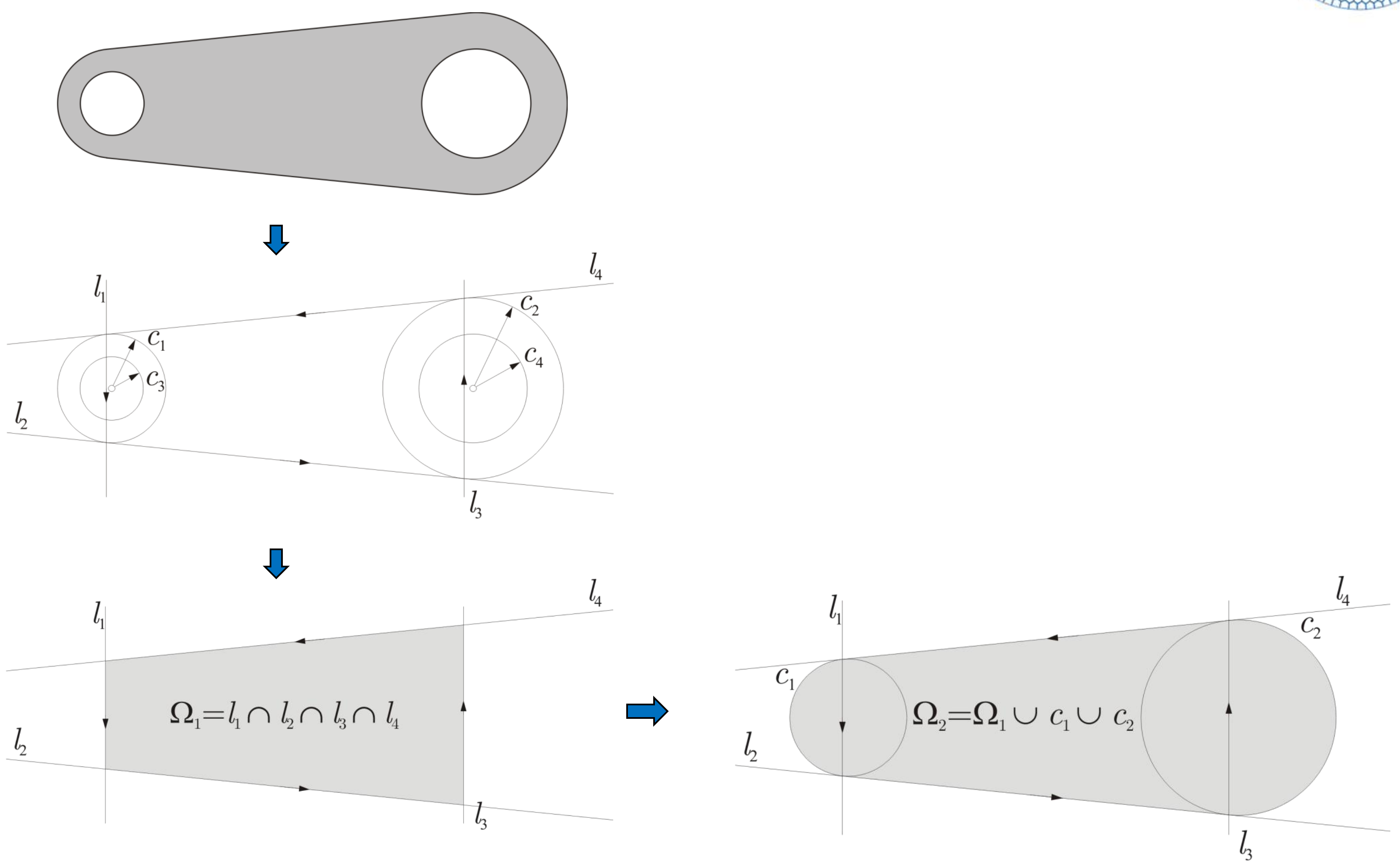
PolyMesher EXAMPLE: WRENCH



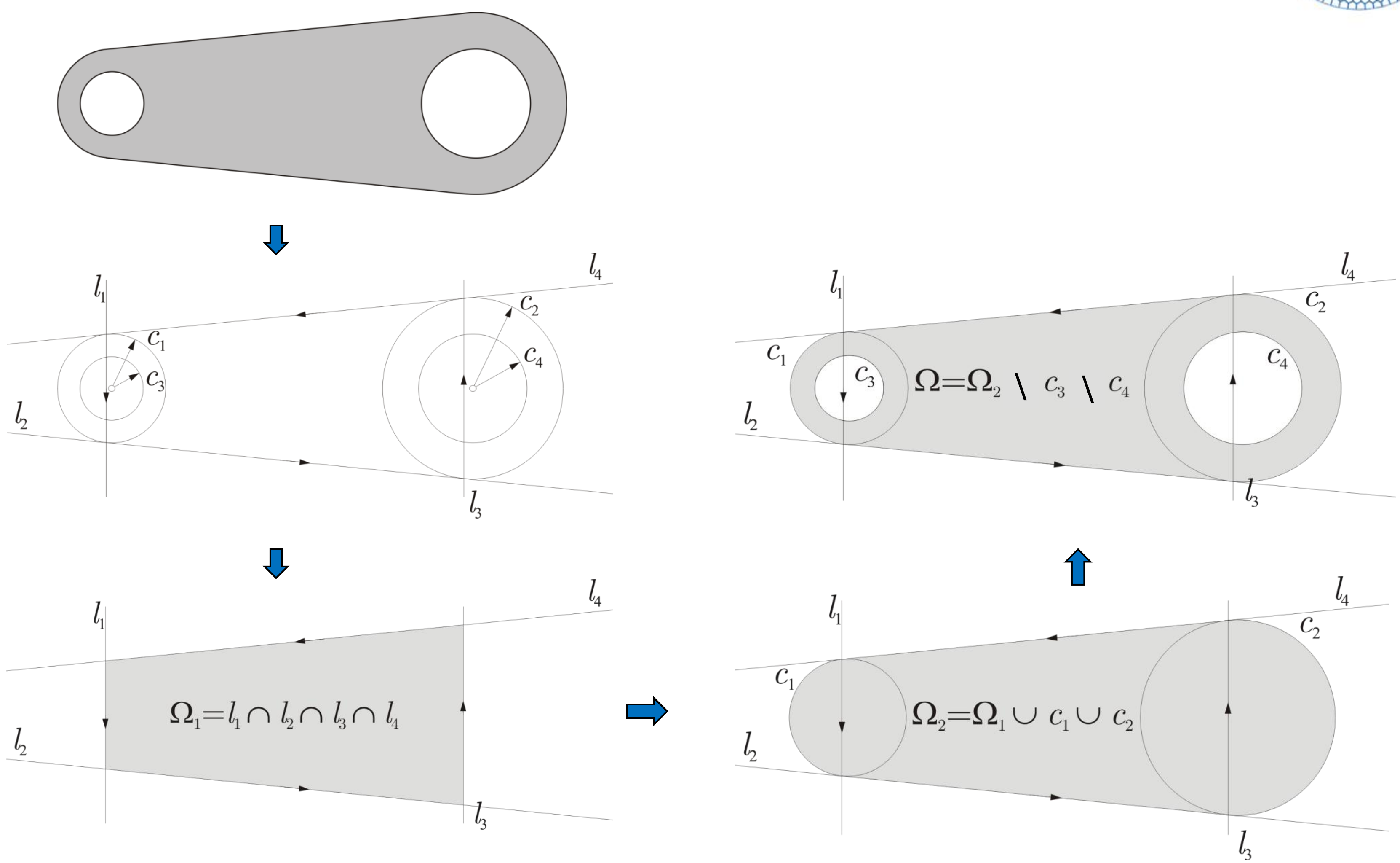
PolyMesher EXAMPLE: WRENCH



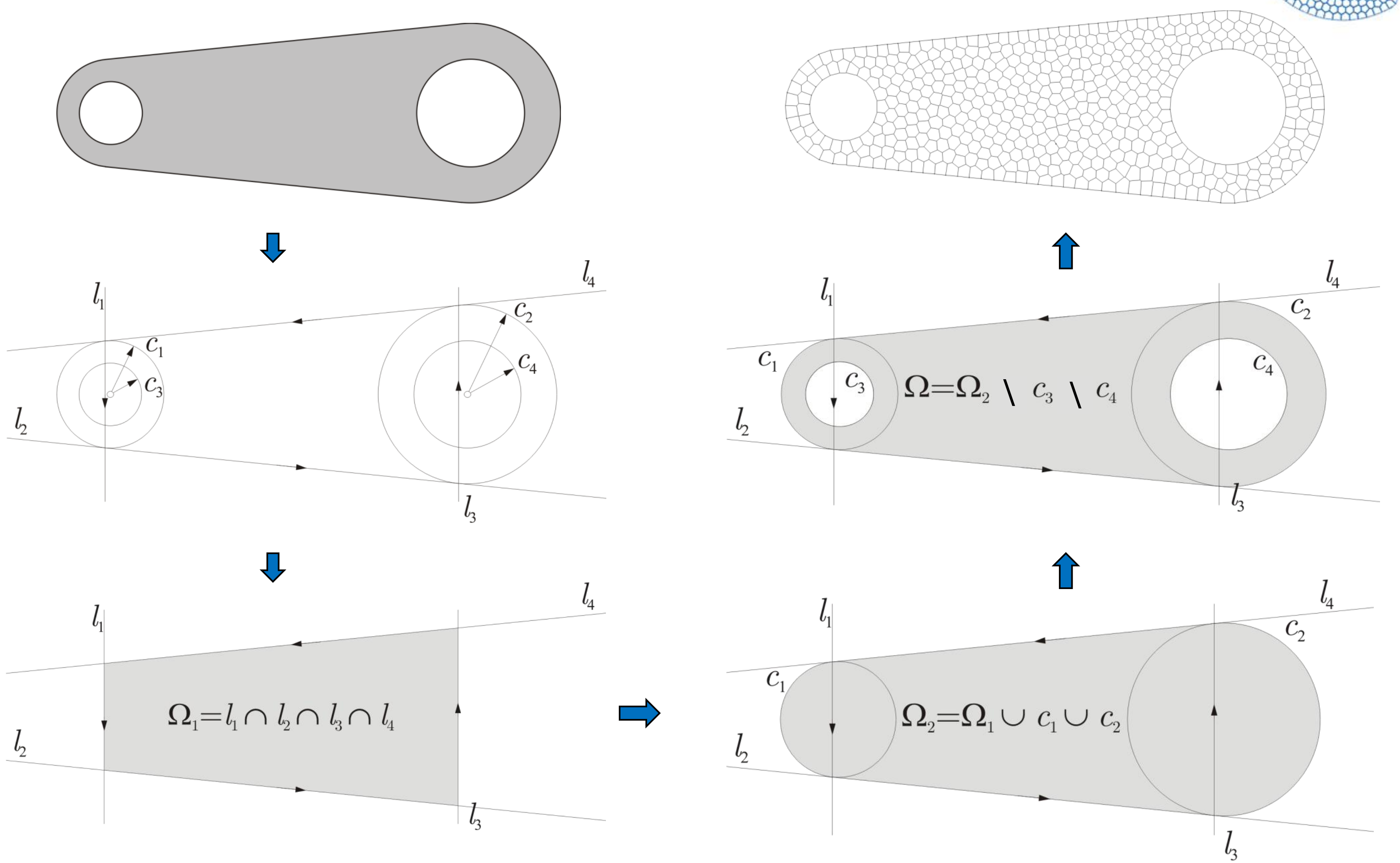
PolyMesher EXAMPLE: WRENCH



PolyMesher EXAMPLE: WRENCH



PolyMesher EXAMPLE: WRENCH



MATLAB DEMO

CONCLUDING REMARKS

CONCLUDING REMARKS

- The development of PolyMesher was motivated by the desire to present a complete, self-contained, efficient and useful code in MATLAB, including domain description and discretization algorithms;
- Due to the modularity and flexibility of the PolyTop code, analysis routine and optimization algorithm can be separated from the specific choice of topology optimization formulation;
- We would like to point out that PolyTop offers room for further exploration of finite elements and topology optimization formulations both for research and for practical engineering applications;
- We hope that the community can make use of PolyMesher & PolyTop in ways that we cannot anticipate.

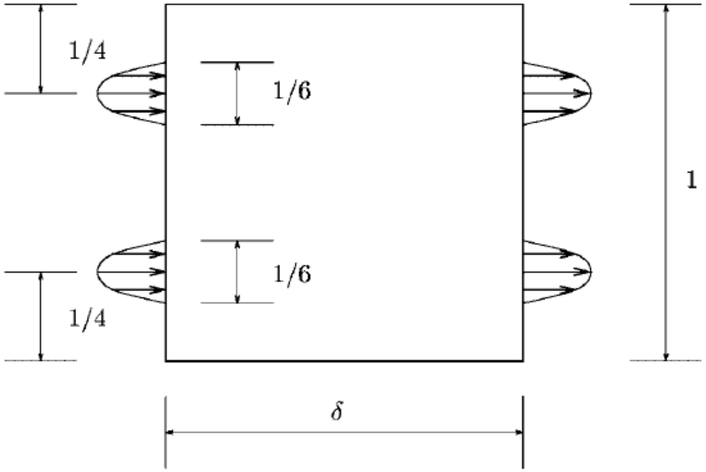
QUESTIONS ?

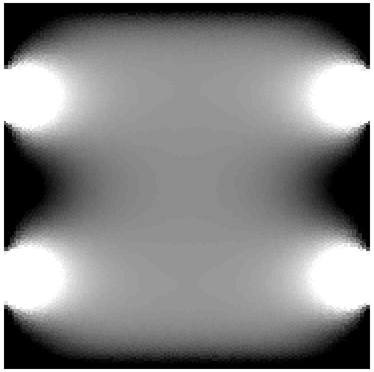
SUPPLEMENTARY MATERIAL

ONGOING RESEARCH

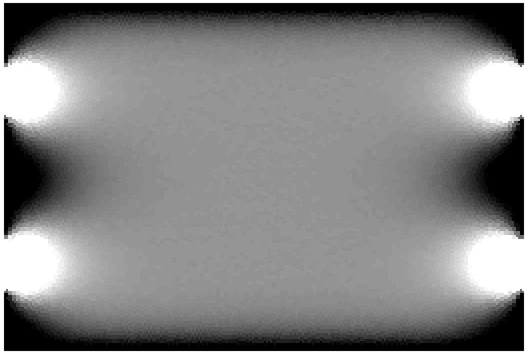
- Topology optimization of fluids in Stokes flow

Double pipe problem





Optimal double pipes for *d*=1



Optimal double pipes for *d*=1.5

CODE EFFICIENCY

Comparison with the 88-line code*

Mesh Size	90x30	150x50	300x100	600x200
PolyTop	11.9	31.5	135.5	764.1
88-line	10.9	33.0	252.2	3092.9
88-line†	**9.7**	**24.3**	**119.7**	**708.8**

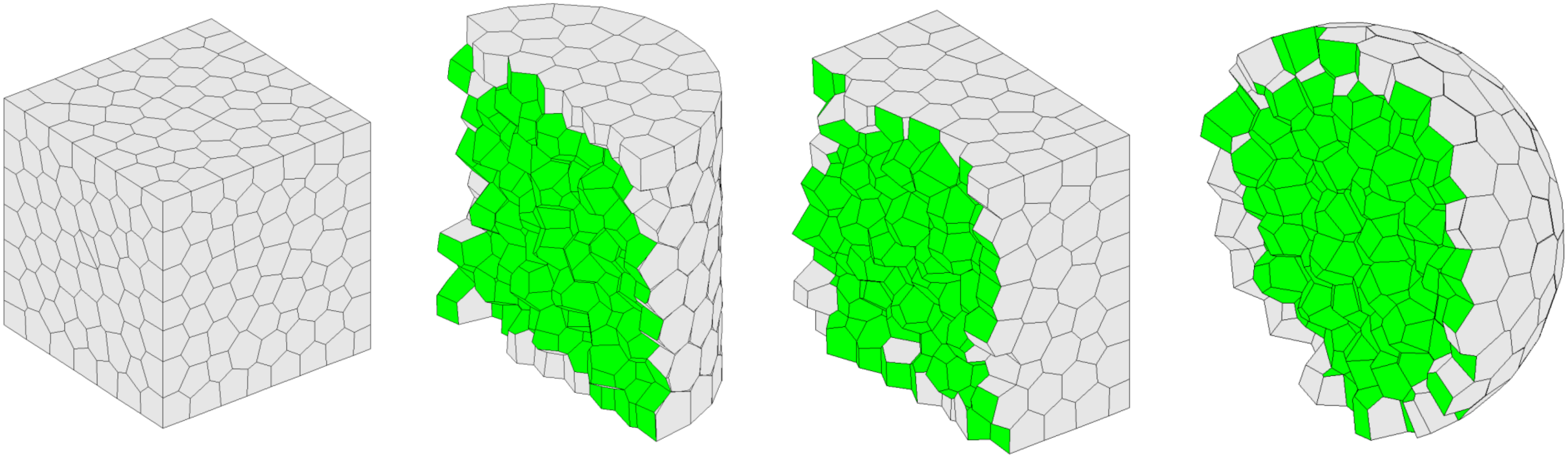
(times in seconds for 200 optimization iterations)

† Linearized Volume Constrained: $V(\mathbf{z}) = \sum_{\ell=1}^{N} (\mathbf{Pz})_\ell = \mathbf{1}^T (\mathbf{Pz}) = \left(\mathbf{1}^T\mathbf{P}\right)\mathbf{z} = \left(\mathbf{P}^T\mathbf{1}\right)^T \mathbf{z}$

* Andreassen E., Clausen A., Schevenels M., Lazarov B., Sigmund O., "*Efficient topology optimization in MATLAB using 88 lines of code*", **JSMO**, 43(1):1–16, **2011.** doi:10.1007/s00158-010-0594-7

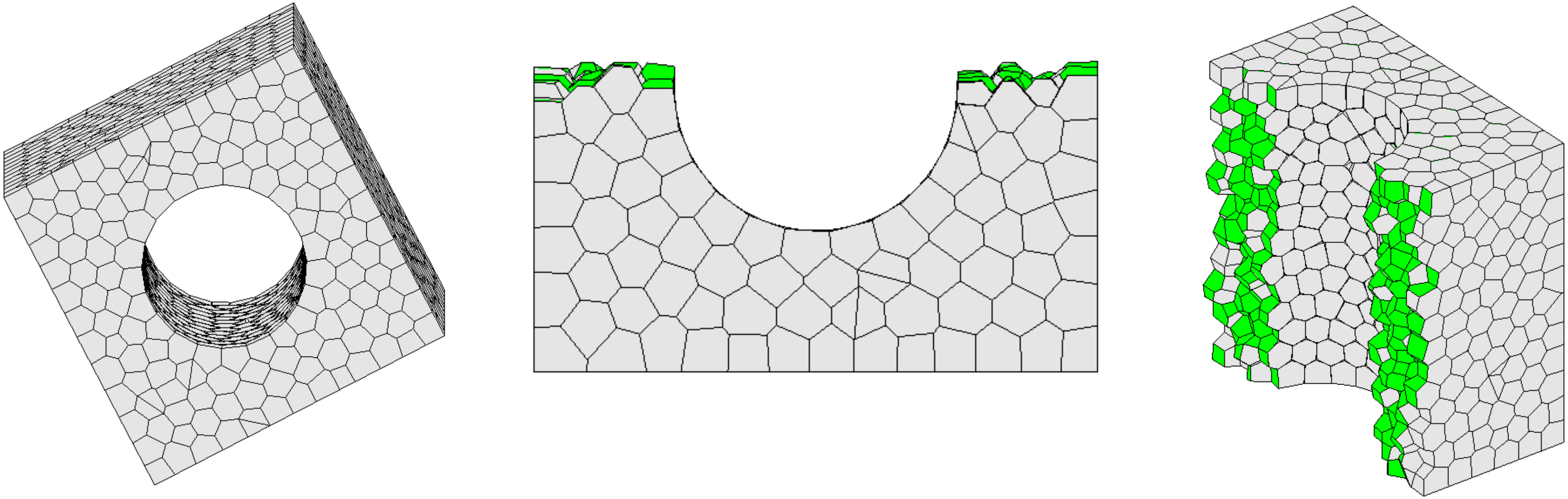
3D MESHES

500 Elements – 200 Lloyd's Iterations



3D MESHES

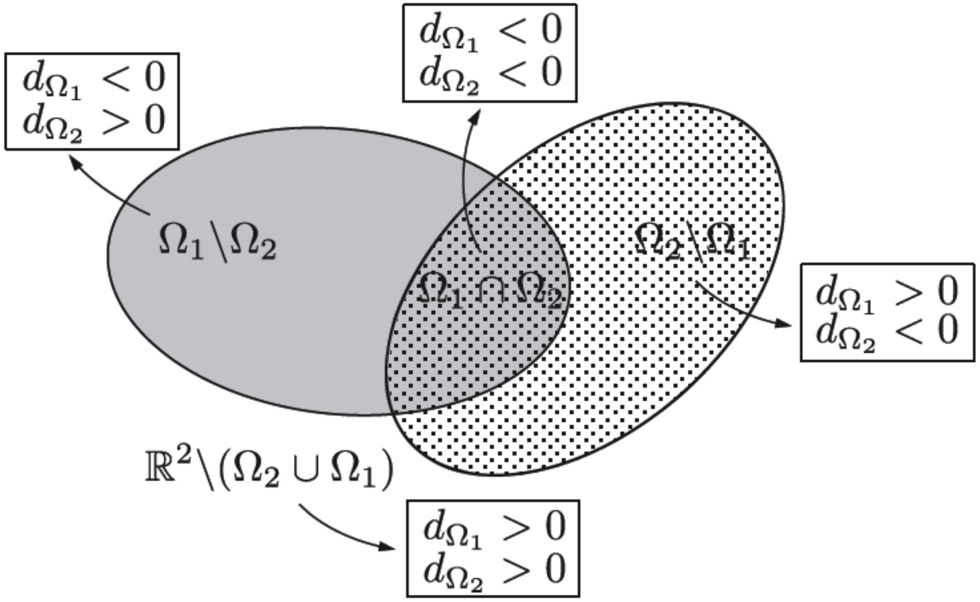
2000 Elements – 200 Lloyd's Iterations



MESH GENERATION – BASIC IDEAS

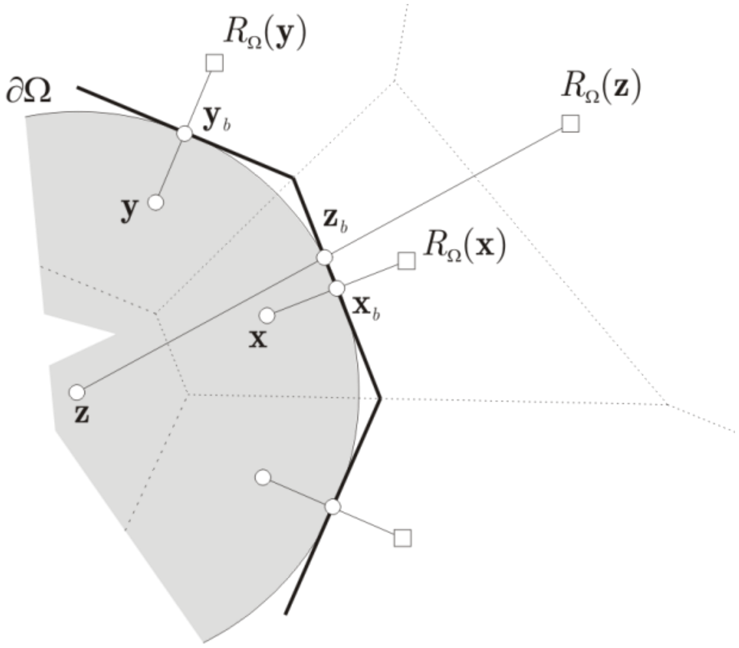
Implicit Representation

- Moreover, set operations such as **union**, **intersection**, and **difference** can be used to combine different geometries:



MESH GENERATION – ALGORITHM

A polygonal discretization can be obtained from the Voronoi diagram of a given set of seeds and their reflections:



- We first reflect each point in **P** about the *closest* boundary point of Ω and denote the resulting set of points by $R_\Omega(\mathbf{P})$.
- We then construct the Voronoi diagram of the plane by including the original point set as well as its reflection.
- Finally we incorporate Lloyd's iterations to obtain a point set **P** that produces a CVT.

$$R_\Omega(\mathbf{x}) = \mathbf{x} - 2d_\Omega(\mathbf{x})\nabla d_\Omega(\mathbf{x}) \qquad R_\Omega(\mathbf{P}) := \{R_\Omega(\mathbf{y}) : \mathbf{y} \in \mathbf{P}\}$$

CODE STRUCTURE

PolyTop (Main Loop)

```
function [z,V,fem] = PolyTop(fem,opt)
Iter=0; Tol=opt.Tol*(opt.zMax-opt.zMin); Change=2*Tol; z=opt.zIni; P=opt.P;
[E,dEdy,V,dVdy] = opt.MatIntFnc(P*z);
[FigHandle,FigData] = InitialPlot(fem,V);
while (Iter<opt.MaxIter) && (Change>Tol)
  Iter = Iter + 1;
  %Compute cost functionals and analysis sensitivities
  [f,dfdE,dfdV,fem] = ObjectiveFnc(fem,E,V);
  [g,dgdE,dgdV,fem] = ConstraintFnc(fem,E,V,opt.VolFrac);
  %Compute design sensitivities
  dfdz = P'*(dEdy.*dfdE + dVdy.*dfdV);
  dgdz = P'*(dEdy.*dgdE + dVdy.*dgdV);
  %Update design variable and analysis parameters
  [z,Change] = UpdateScheme(dfdz,g,dgdz,z,opt);
  [E,dEdy,V,dVdy] = opt.MatIntFnc(P*z);
  %Output results
  fprintf('It: %i \t Objective: %1.3f\tChange: %1.3f\n',Iter,f,Change);
  set(FigHandle,'FaceColor','flat','CData',1-V(FigData)); drawnow
end
```

CODE STRUCTURE

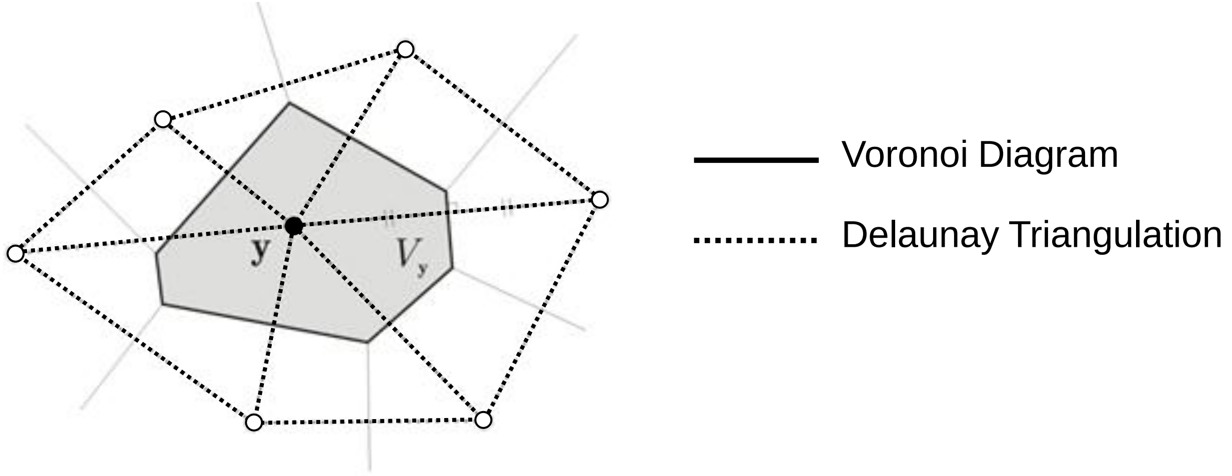
PolyMesher (Main Loop)

```
function [Node,Element,Supp,Load,P] = PolyMesher (Domain, NElem,
MaxIter,P)
if ~exist('P','var'), P=PolyMshr_RndPtSet (NElem,Domain); end
NElem = size(P,1);
Tol=5e-3; It=0; Err=1; c=1.5;
BdBox = Domain('BdBox');
Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
Pc = P; figure;
while(It<=MaxIter && Err>Tol)
  Alpha = c*sqrt(Area/NElem);
  P = Pc; %Lloyd's update
  R_P = PolyMshr_Rflct(P,NElem,Domain,Alpha); %Generate the reflections
  [Node,Element] = voronoin([P;R_P]); %Construct Voronoi diagram
  [Pc,A] = PolyMshr_CntrdPly(Element,Node,NElem);
  Area = sum(abs(A));
  Err = sqrt(sum((A.^2).*sum((Pc-P).*(Pc-P),2)))*NElem/Area^1.5;
  fprintf('It: %3d   Error: %1.3e\n',It,Err); It=It+1;
  if NElem<=2000, PolyMshr_PlotMsh(Node,Element,NElem); end;
end
[Node,Element] = PolyMshr_ExtrNds(NElem,Node,Element); %Extract node list
[Node,Element] = PolyMshr_CllpsEdgs(Node,Element,0.1); %Remove small
edges
[Node,Element] = PolyMshr_RsqsNds(Node,Element); %Reoder Nodes
```

The concept of Voronoi Tesselation (or diagram)

Given a set of “*n*” distinct points of seeds **P**, a Voronoi cell $V_{\mathbf{y}}$ consists of points in the plane closer to y than any other point in **P**, i.e.:

$$V_{\mathbf{y}} = \left\{ \mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x} - \mathbf{y}\| < \|\mathbf{x} - \mathbf{z}\|, \forall \mathbf{z} \in \mathbf{P} \setminus \{\mathbf{y}\} \right\}$$



MESH GENERATION – BASIC IDEAS

- The concept of Voronoi Tesselation plays a central role in our meshing algorithm

- **Centroidal Voronoi Tesselations (CVTs) enjoy a higher level of regularity which are suitable for use in Finite Element Analysis**

- Implicit representation

Centroidal Voronoi Tesselations (**CVT**s)

A Voronoi tesselation $\mathcal{T}(\mathrm{P};\Delta)$ is centroidal if, for every y $\in$ P: i.e.:

$$\mathbf{y} = \mathbf{y}_c \qquad \text{where} \qquad \mathbf{y}_c := \frac{\int_{V_{\mathbf{y}} \cap \Delta} \mathbf{x}\mu(\mathbf{x})d\mathbf{x}}{\int_{V_{\mathbf{y}} \cap \Delta} \mu(\mathbf{x})d\mathbf{x}}$$

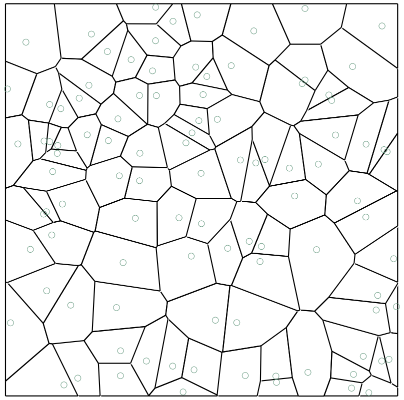
MESH GENERATION – BASIC IDEAS

Centroidal Voronoi Tesselations (**CVT**s)

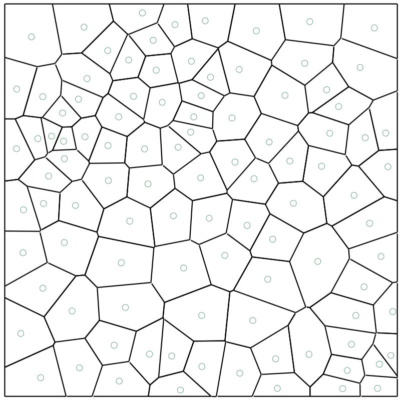
A Voronoi tesselation $\mathcal{T}(\mathbf{P};\Delta)$ is centroidal if, for every y $\in$ P: i.e.:

$$\mathbf{y} = \mathbf{y}_c \qquad \text{where} \qquad \mathbf{y}_c := \frac{\int_{V_{\mathbf{y}}\cap\Delta} \mathbf{x}\mu(\mathbf{x})d\mathbf{x}}{\int_{V_{\mathbf{y}}\cap\Delta} \mu(\mathbf{x})d\mathbf{x}}$$

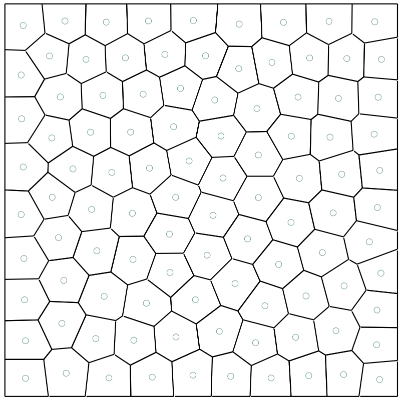
Lloyd's Iterations:



Initial random points



First iteration



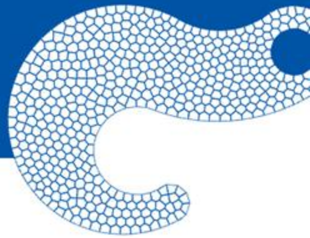
After 80 iterations

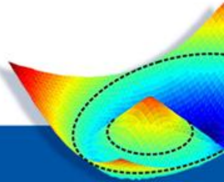
MESH GENERATION – BASIC IDEAS

- The concept of Voronoi Tesselation plays a central role in our meshing algorithm

- Centroidal Voronoi Tesselations (CVTs) enjoy a higher level of regularity which are suitable for use in Finite Element Analysis

- **Implicit representation**





ACKNOWLEDGEMENT

Tecgraf – Computer Graphics Technology Group at PUC-Rio

Department of Mechanical Engineering / PUC-Rio

Department of Energy Computational Science Graduate Fellowship Program of the Office of Science and National Nuclear Security Administration in the Department of Energy.