PolyMesher: A General-Purpose Mesh Generator for Polygonal Elements Written in Matlab

Anderson Pereira, Cameron Talischi, Ivan F. M. Menezes and Glaucio H. Paulino

11th U.S. National Congress on Computational Mechanics

25 - 28 july 2011 - Minnesota, USA

Tecgraf - Computer Graphics Technology Group

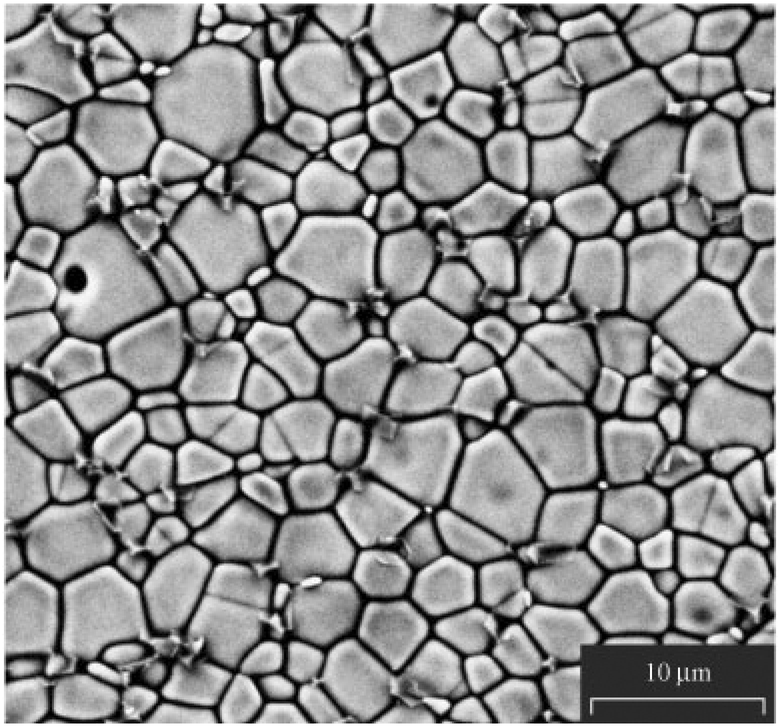
Department of Civil and Environmental Engineering
University of Illinois at Urbana-Champaign

Motivation

- Voronoi diagrams offer a simple way to discretize geometries. They have been widely used to describe the material structure in:

Motivation

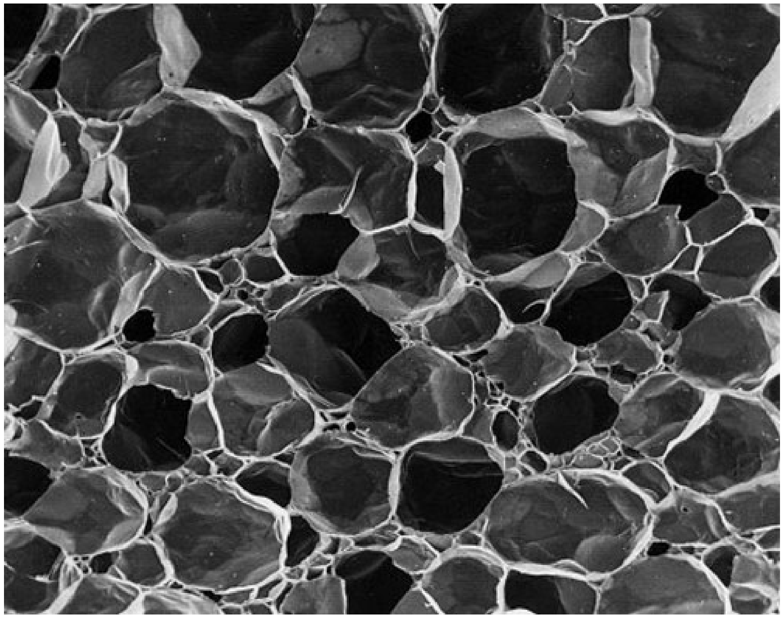
- Voronoi diagrams offer a simple way to discretize geometries. They have been widely used to describe the material structure in:
  - polycrystalline microstructures,



PR Bueno and JA Varela. Electronic ceramics based on polycrystalline SnO2, TiO2 and (Sn xTi1-x)O2 solid solution. *Mat. Res.* [online]. 2006, vol.9, n.3, pp. 293-300

Motivation

- Voronoi diagrams offer a simple way to discretize geometries. They have been widely used to describe the material structure in:
  - polycrystalline microstructures,
  - cellular foams,



Cellular polyethylene

http://www.koepp.de

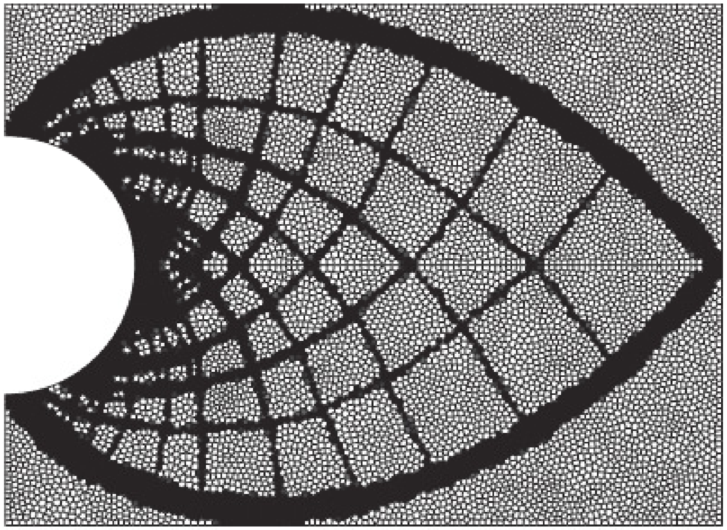
Motivation

- Voronoi diagrams offer a simple way to discretize geometries. They have been widely used to describe the material structure in:
  - polycrystalline microstructures,
  - cellular foams,
  - and other materials that exhibit cell-like features.
- For such applications, numerical modeling and simulation of Voronoi meshes is a natural choice.
- Finite element analyses can also be based on the Delaunay/Voronoi dual tessellations for both defining the computational mesh and approximating the field quantity within each element.

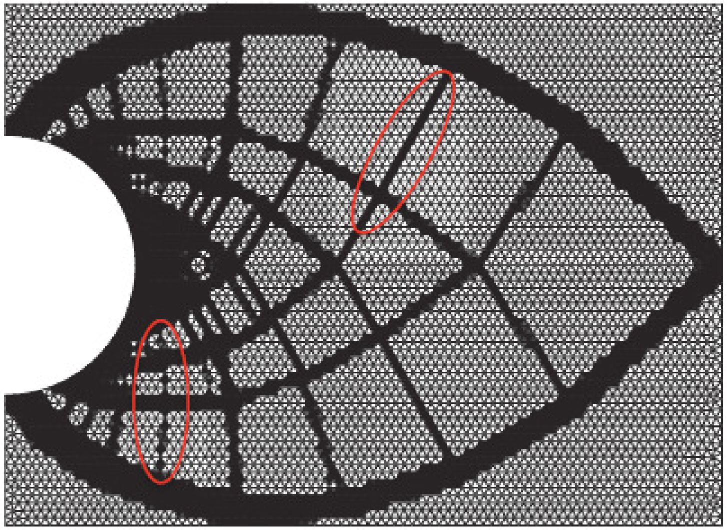
Motivation

- Recently, polygonal meshes were used in topology optimization yielding good results.

Polygonal Elements



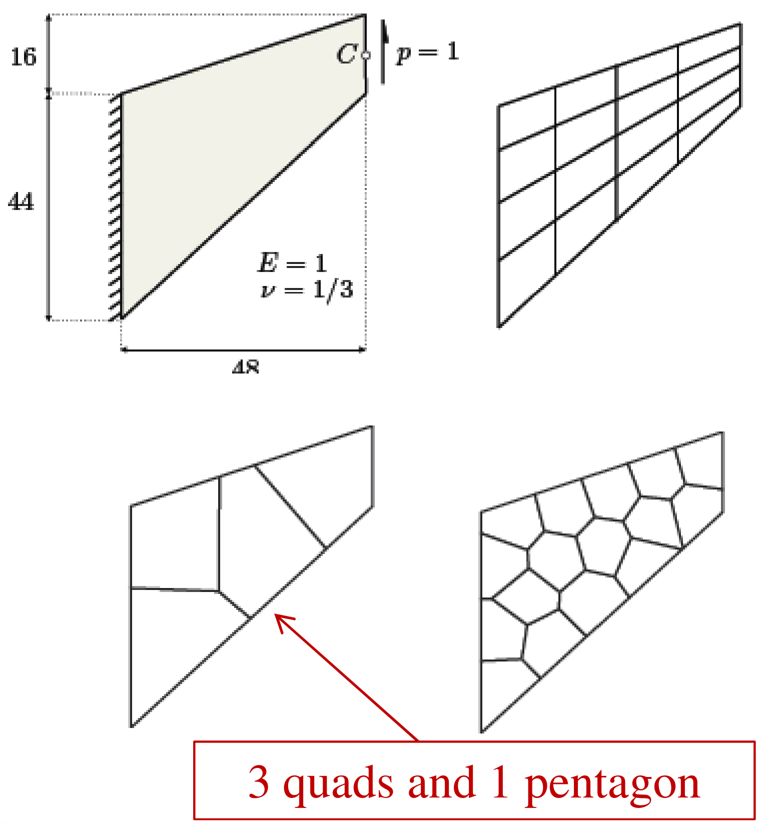
T6 Elements

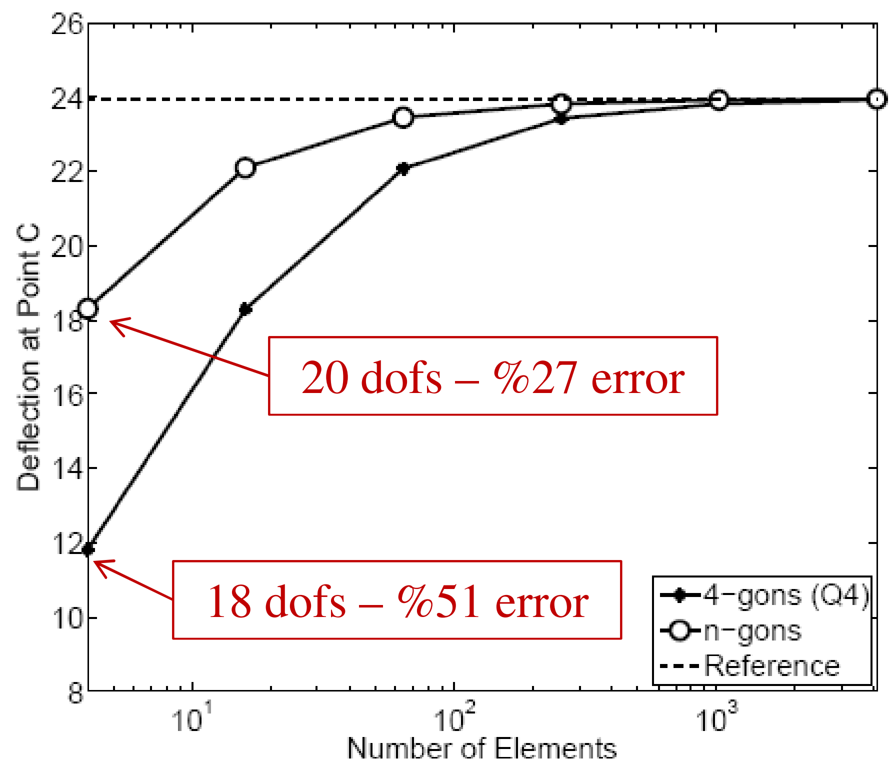


C. Talischi, G. Paulino, A. Pereira and IFM Menezes. Polygonal finite elements for topology optimization: A unifying paradigm. **IJNME**, 82(6):671-698, 2010

Motivation

- Recently, polygonal meshes were used in topology optimization yielding good results.

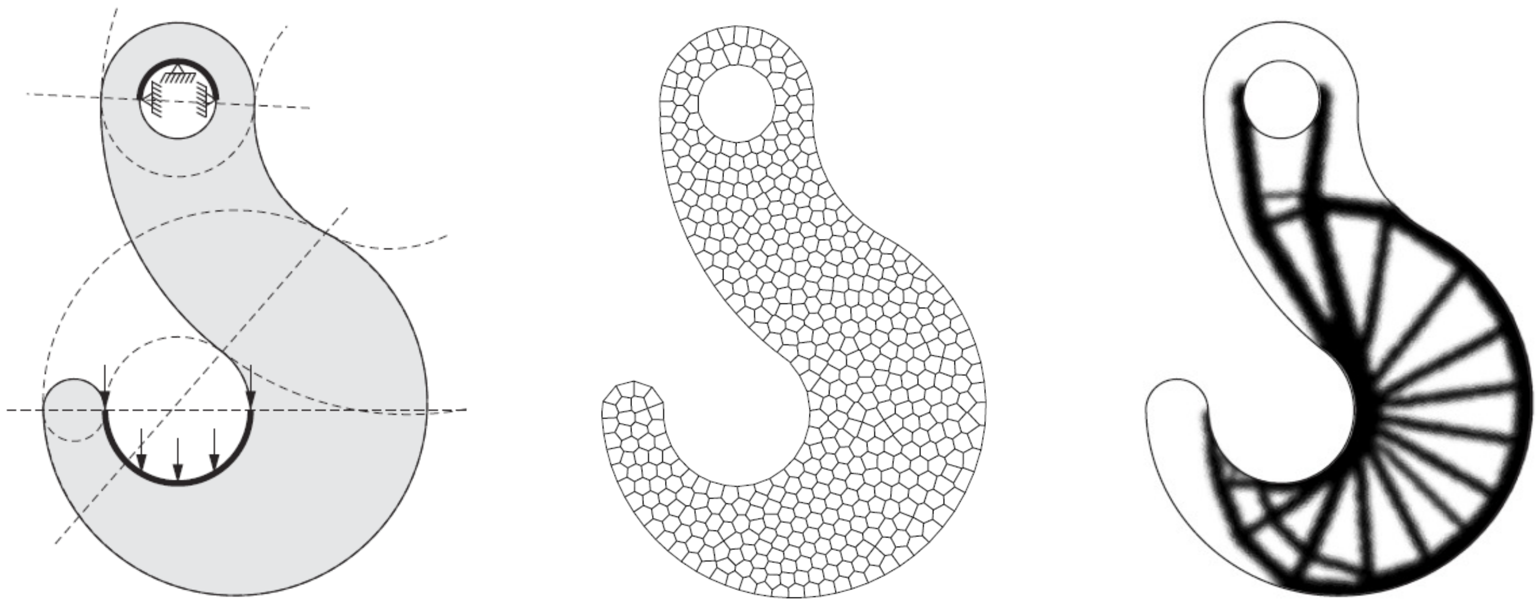




C. Talischi, G. Paulino, A. Pereira and IFM Menezes. Polygonal finite elements for topology optimization: A unifying paradigm. **IJNME**, 82(6):671-698, 2010

Motivation

- Our main goal here is to provide the users a self-contained Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. In this presentation we will explain PolyMesher, responsible for the polygonal discretization.



PolyMesher PolyTop

C Talischi, GH Paulino, A Pereira, IFM Menezes, "PolyTop: A Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes", **JSMO**, 2011

Outline

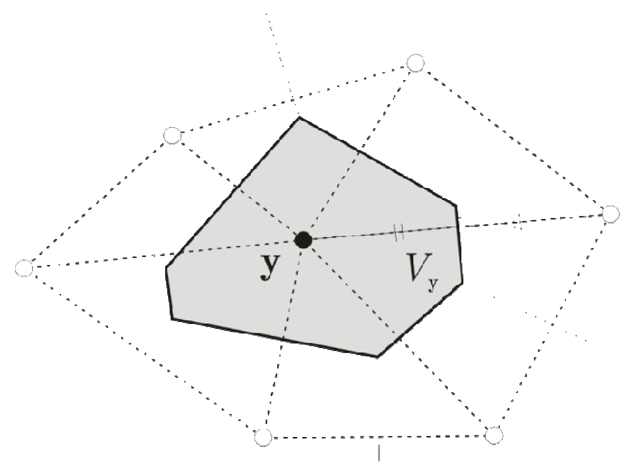
- Voronoi diagrams, CVTs and Lloyd’s algorithm
- Meshing algorithm
- Implicit representation
- Matlab implementation
- Examples
- Concluding remarks
- Ongoing work

Voronoi diagrams, CVTs and Lloyd's algorithm

The concept of Voronoi diagrams plays a central role in our meshing algorithm.

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.

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



Delaunay triangulation

Voronoi diagram

Voronoi diagrams, CVTs and Lloyd's algorithm

Centroidal Voronoi tesselations (CVTs) enjoy a higher level of regularity which are suitable for use in finite element analysis.

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

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

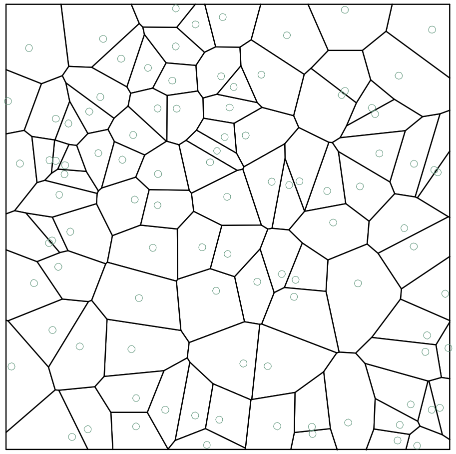
For computing CVTs we used the ***Lloyd's algorithm***, which iteratively replaces the given generating seeds by the centroids of the corresponding Voronoi regions. Lloyd's algorithm can be thought of as a fixed point iteration for the mapping:

$$\mathbf{L} = (\mathbf{L}_{\mathbf{y}})^T_{\mathbf{y} \in \mathbf{P}} : \mathbb{R}^{n\times 2} \to \mathbb{R}^{n\times 2}$$

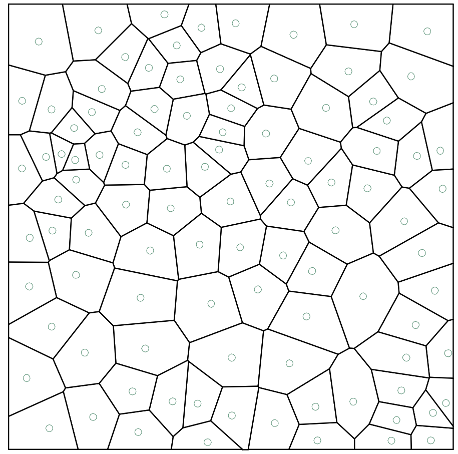
$$\mathbf{L}_{\mathbf{y}}(\mathbf{P}) = \frac{\int_{V_{\mathbf{y}}(\mathbf{P}) \cap \Delta} \mathbf{x}\mu(\mathbf{x})d\mathbf{x}}{\int_{V_{\mathbf{y}}(\mathbf{P}) \cap \Delta} \mu(\mathbf{x})d\mathbf{x}}$$

PUC RIO

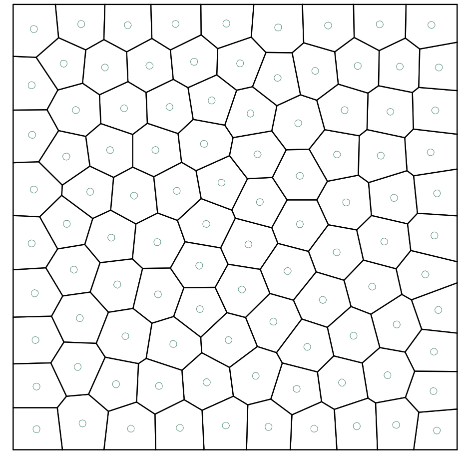
Voronoi diagrams, CVTs and Lloyd's algorithm



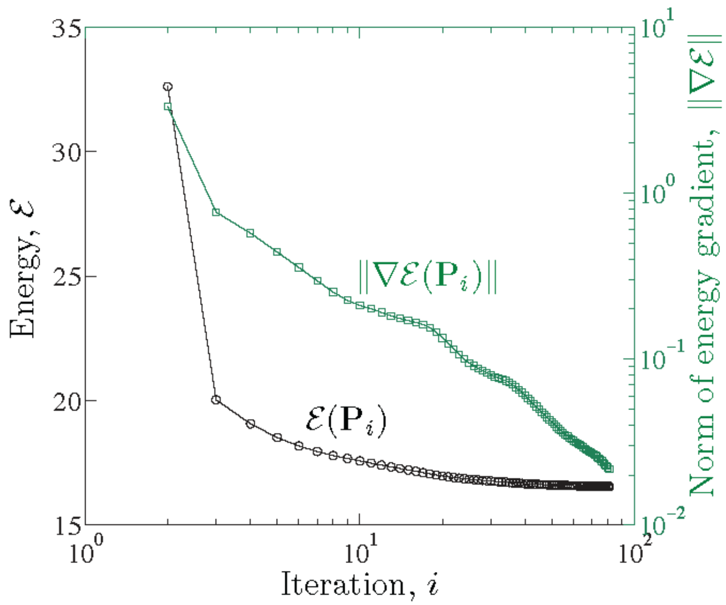
Initial Random points



First iteration



After 80 iterations

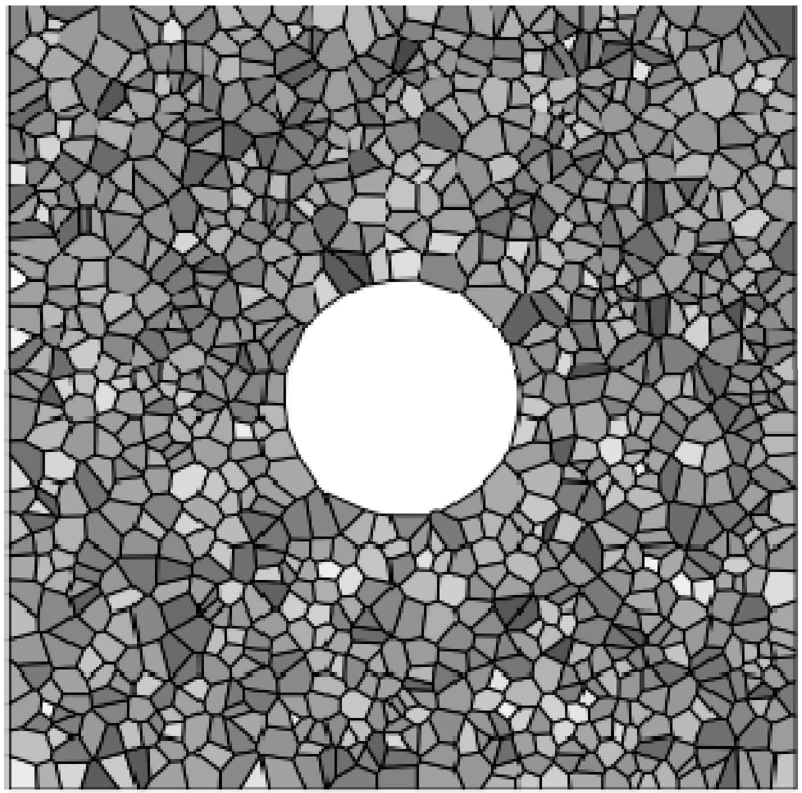


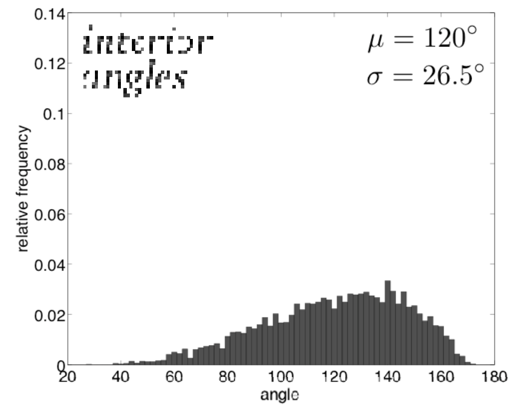
$$\mathcal{E}(\mathbf{P};\Delta) = \sum_{\mathbf{y}\in\mathbf{P}} \int_{V_{\mathbf{y}}(\mathbf{P})\cap\Delta} \mu(\mathbf{x}) \left\|\mathbf{x}-\mathbf{y}\right\|^2 d\mathbf{x}$$

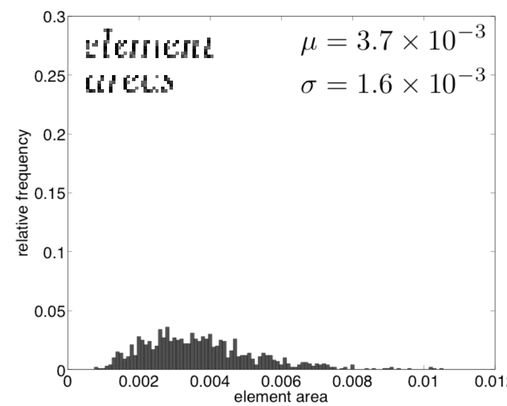
$$\nabla_{\mathbf{y}}\mathcal{E} = 2m_{\mathbf{y}}\left(\mathbf{y}-\mathbf{y}_c\right)$$

$$m_{\mathbf{y}} = \int_{V_{\mathbf{y}}\cap\Delta} \mu(\mathbf{x})d\mathbf{x}$$

Voronoi diagrams, CVTs and Lloyd's algorithm

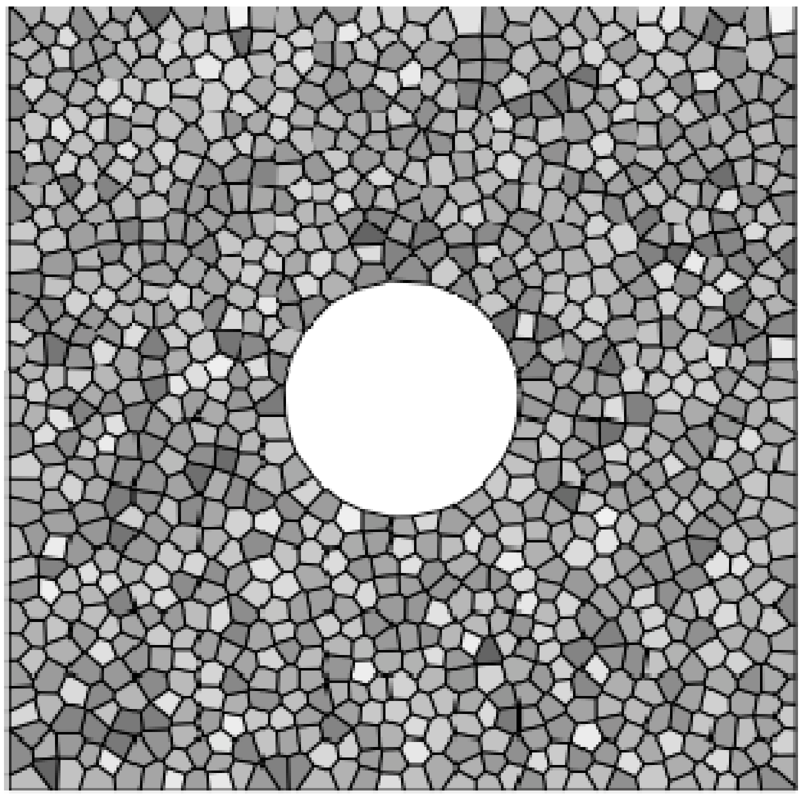


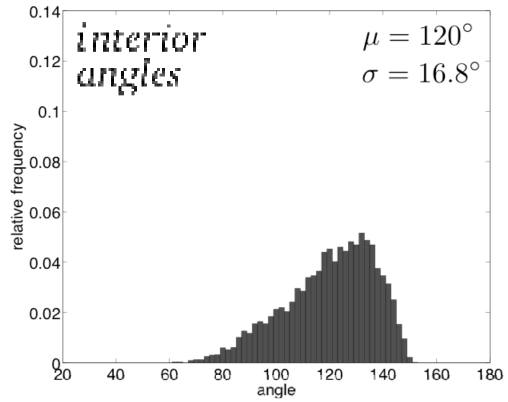


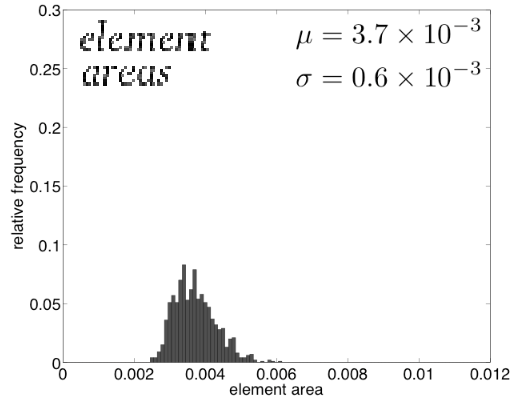


PUC RIO

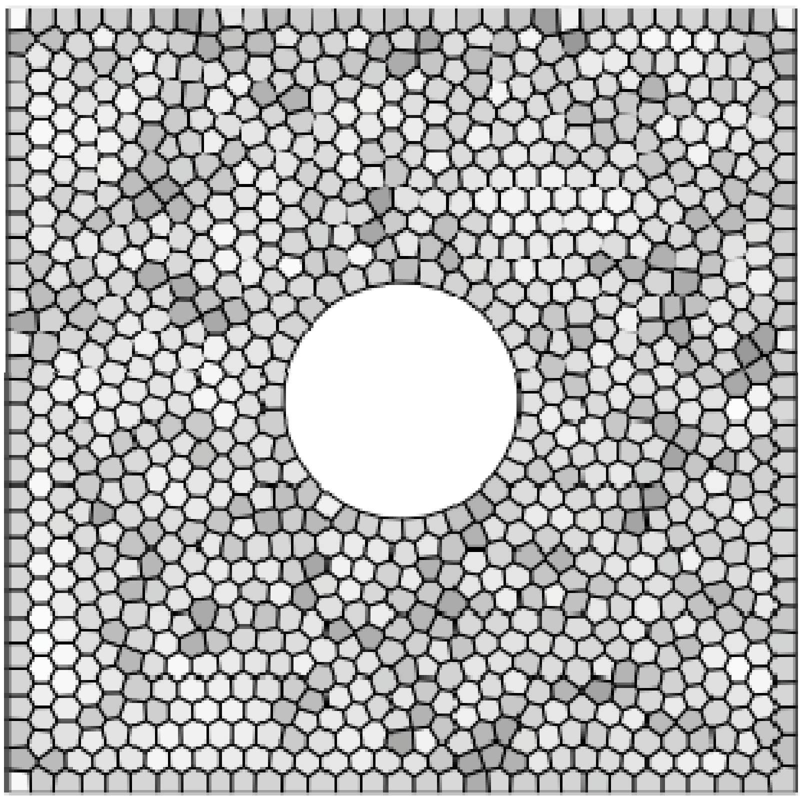
Voronoi diagrams, CVTs and Lloyd's algorithm

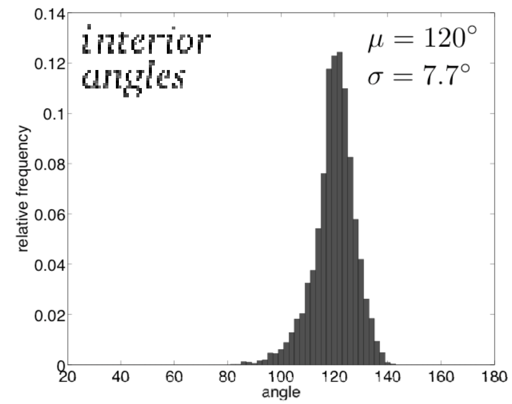


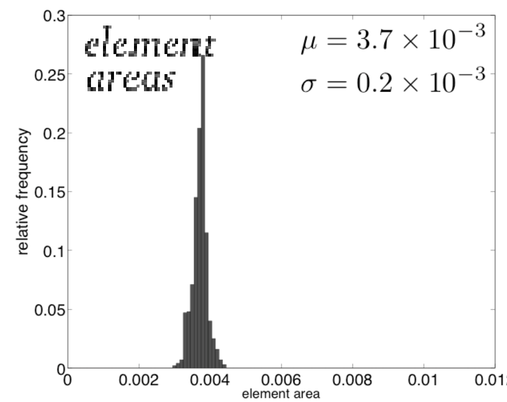




Voronoi diagrams, CVTs and Lloyd's algorithm

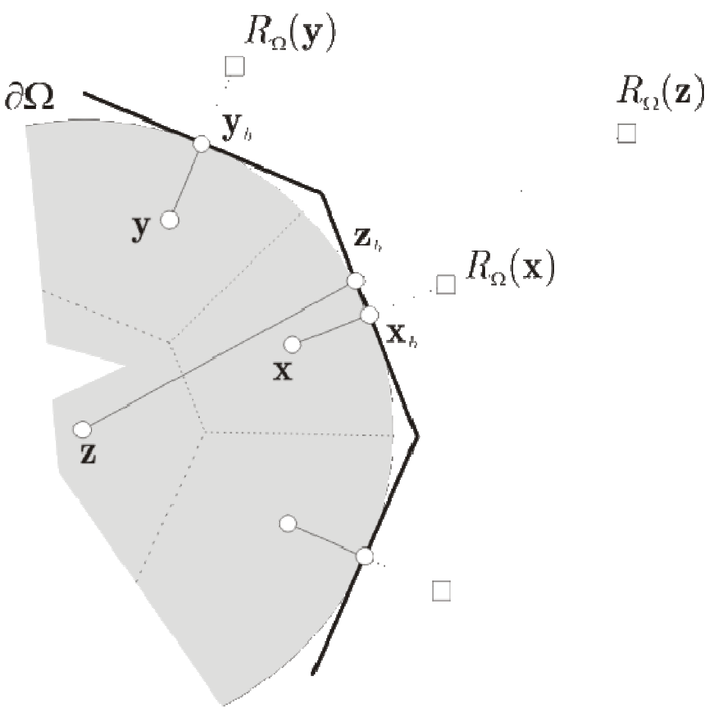






Meshing approach: basic ideas

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 $R_\Omega(\mathrm{P})$.

- 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.

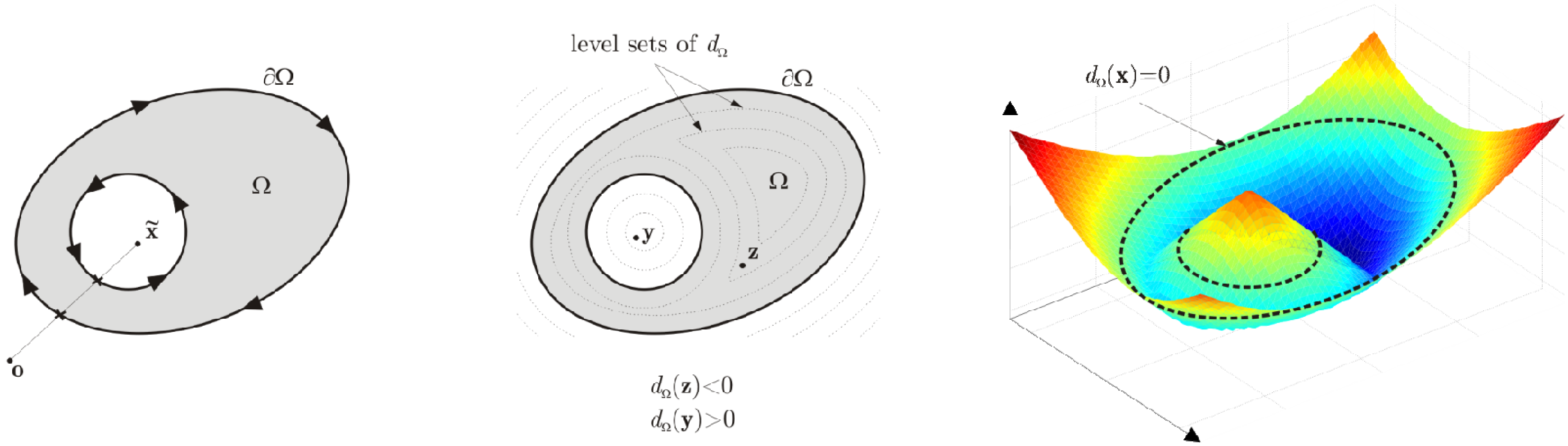
$$R_\Omega(\mathbf{x}) = \mathbf{x} - 2d_\Omega(\mathbf{x})\nabla d_\Omega(\mathbf{x})$$

$$R_\Omega(\mathbf{P}) := \{R_\Omega(\mathbf{y}) : \mathbf{y} \in \mathbf{P}\}$$

PUC RIO

Implicit representation

One of the main ingredients of our mesh generator is the **implicit representation** of the domain:



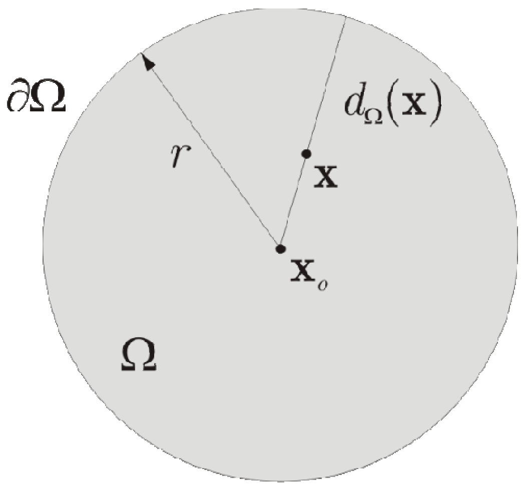
The signed distance function contains all the essential information about the meshing domain needed in our mesh algorithm.

$$d_\Omega(\mathbf{x}) = s_\Omega(\mathbf{x}) \min_{\mathbf{y}\in\partial\Omega} \|\mathbf{x}-\mathbf{y}\|$$

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

Implicit representation: construction of signed distance functions

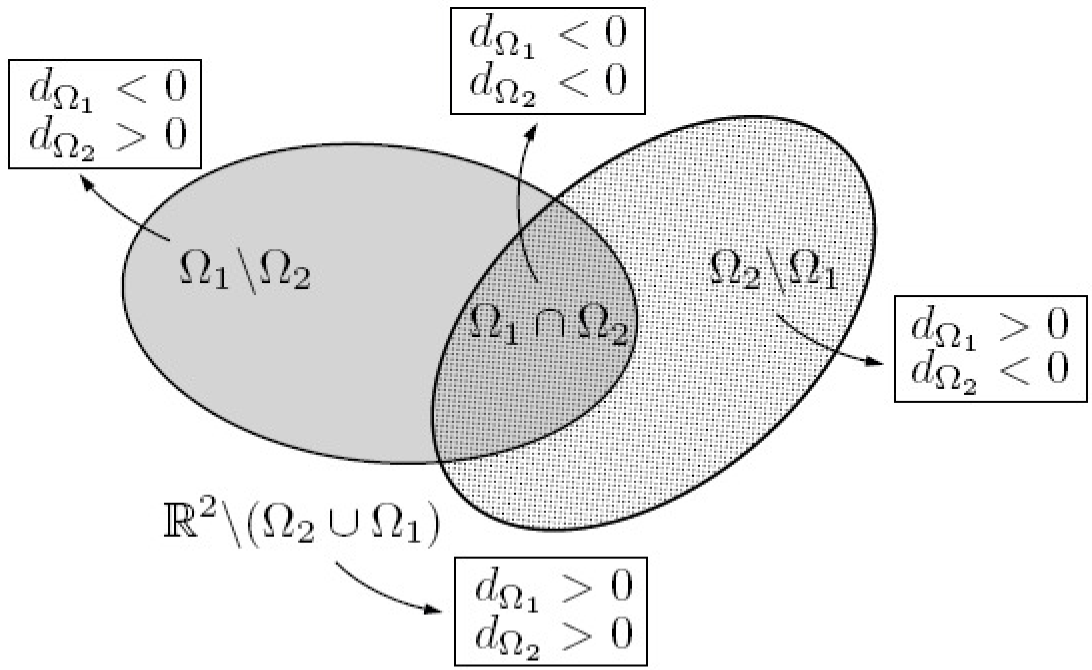
For many simple geometries, the signed distance function can be readily identified, for example:



$$d_\Omega(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_o\| - r$$

Implicit representation: construction of signed distance functions

Moreover, set operations such as union, intersection, and complementation can be used to piece together and combine different geometries:



$$
\begin{aligned}
d_{\Omega_1 \cup \Omega_2}(\mathbf{x}) &= \min\left(d_{\Omega_1}(\mathbf{x}), d_{\Omega_2}(\mathbf{x})\right) \\
d_{\Omega_1 \cap \Omega_2}(\mathbf{x}) &= \max\left(d_{\Omega_1}(\mathbf{x}), d_{\Omega_2}(\mathbf{x})\right)
\end{aligned}
$$

PUC RIO

Matlab implementation

Based on the previous considerations, an algorithm was proposed and implemented in Matlab.

The code has fewer than 135 lines and it is composed by the following main functions:

- `PolyMesher` Meshing kernel
  - `PolyMshr_RndPtSet` generate initial random points
  - `PolyMshr_Rflct` reflect the seeds
  - `PolyMshr_CntrdPly` compute areas and centroids

Matlab implementation: meshing kernel

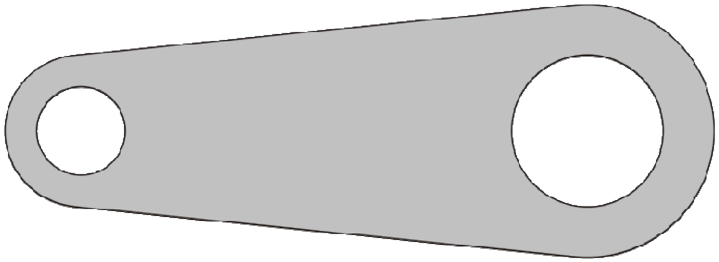
```
function [Node,Element,Supp,Load,P] = PolyMesher(Domain,NElem,MaxIter,P)
if ~exist('P','var'), P=PolyMshr_RndPtSet(NElem,Domain); end
NElem = size(P,1);
Tol=5e-3; It=0; Err=1; c=1.5;
BdBox = Domain('BdBox');
Area = (BdBox(2)-BdBox(1))*(BdBox(4)-BdBox(3));
Pc = P; figure;
while(It<=MaxIter && Err>Tol)
  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
BC=Domain('BC',Node); Supp=BC{1}; Load=BC{2};           %Recover BC arrays
PolyMshr_PlotMsh(Node,Element,NElem,Supp,Load);         %Plot mesh and BCs
```

Matlab implementation: Domain function

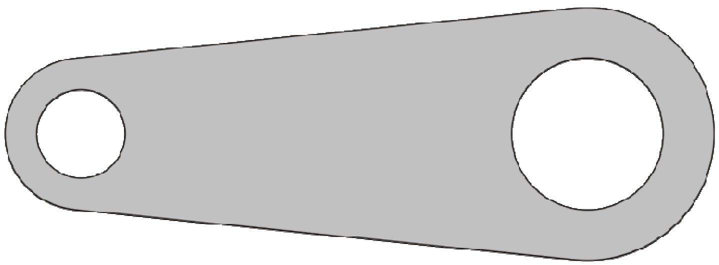
All domain-related information are included in `Domain` defined outside the meshing kernel.

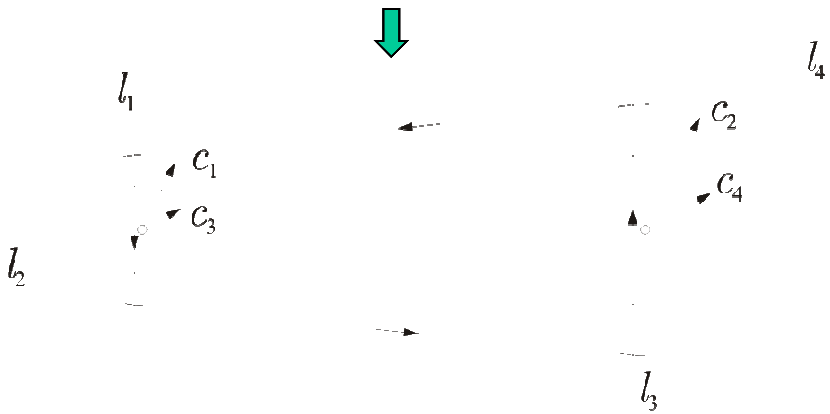
Use	Input	Output
bounding box	`Domain('BdBox')`	coordinates of bounding box `[xmin,xmax,ymin,ymax]`
distance values	`Domain('Dist',P)`	$(m+1) \times n$ matrix of distance values for point set $\mathbf{P}$ consisting of n points
boundary conditions	`Domain('BC',Node)`	cell consisting of `Load` and `Supp` arrays

Example: Wrench

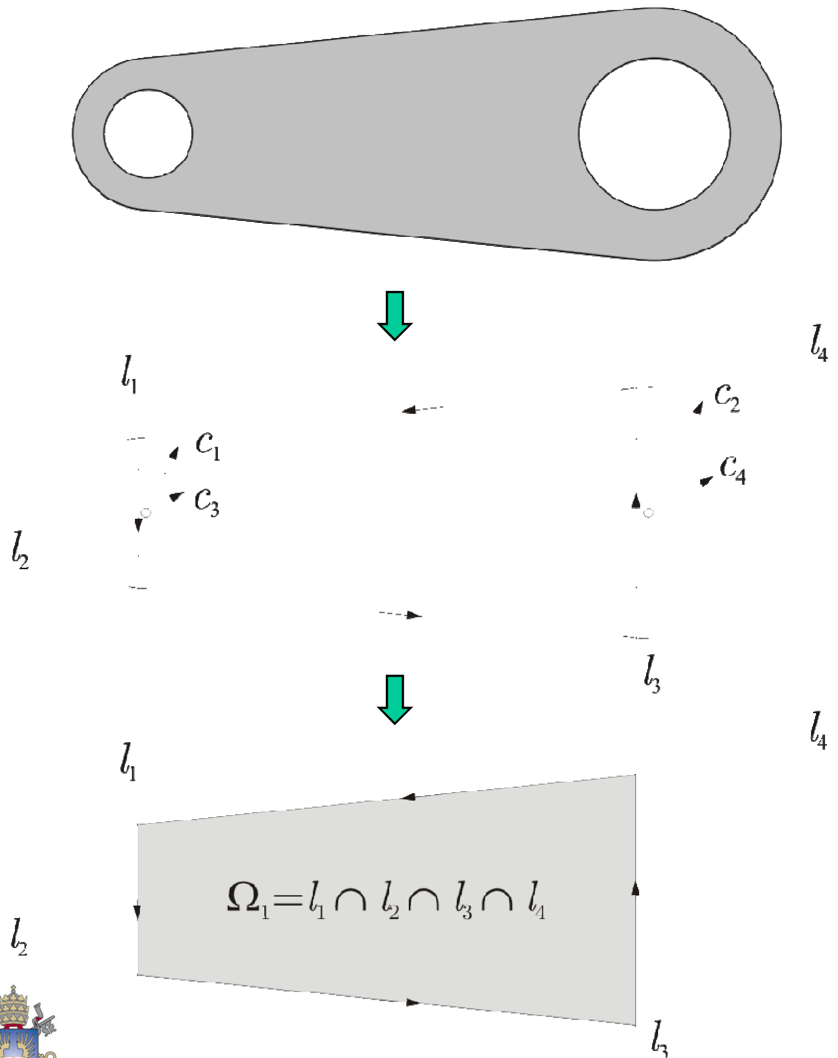


Example: Wrench

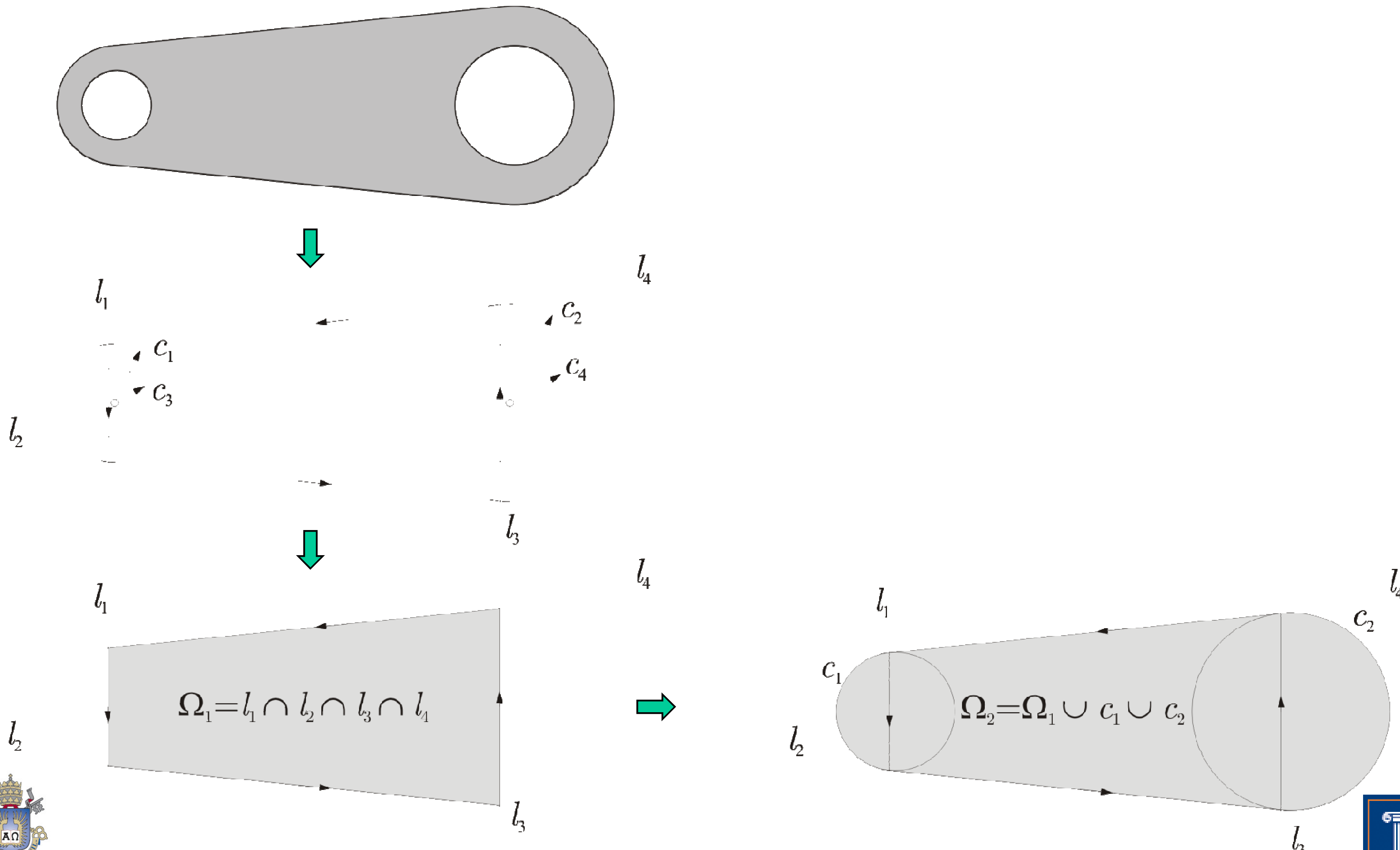




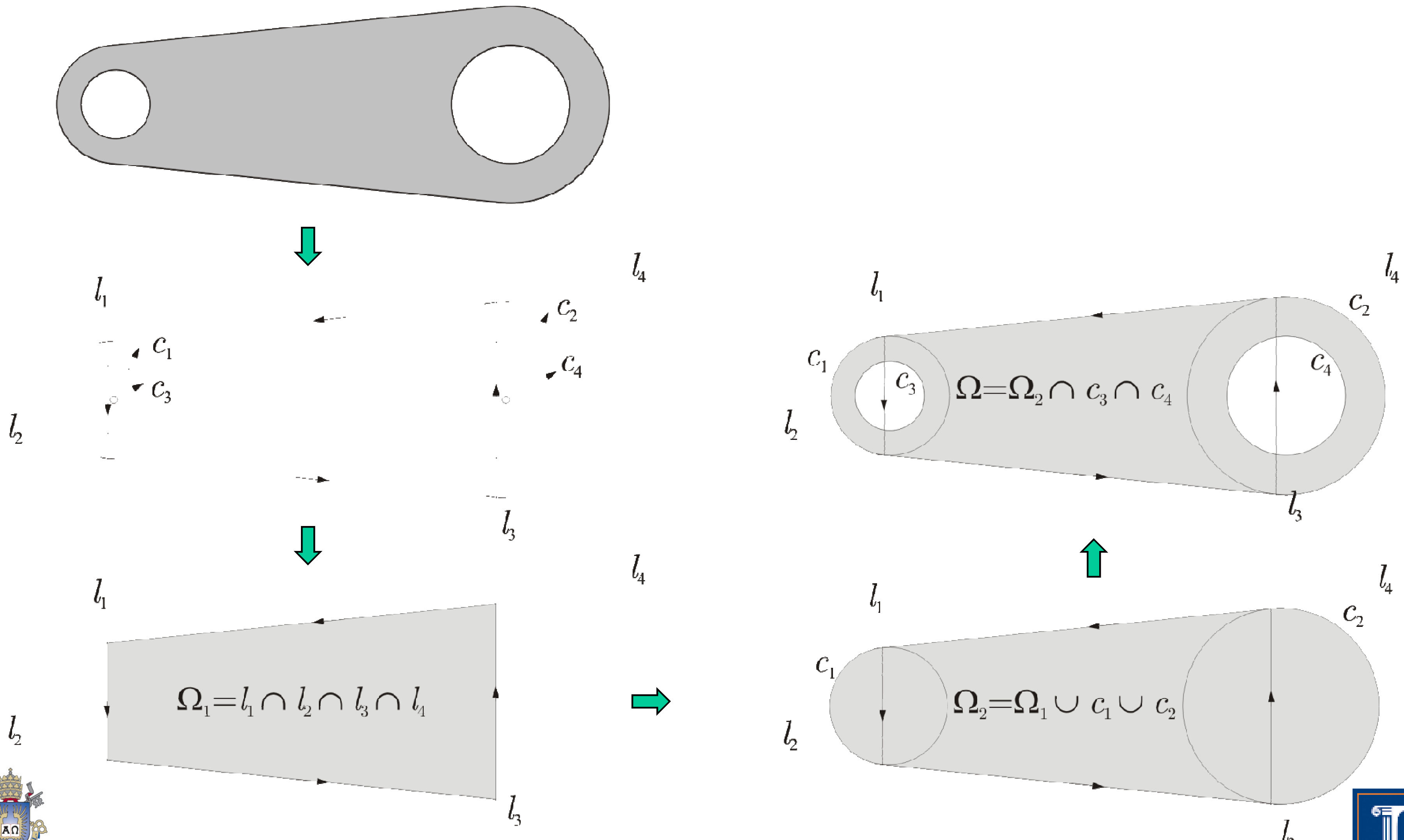
Example: Wrench



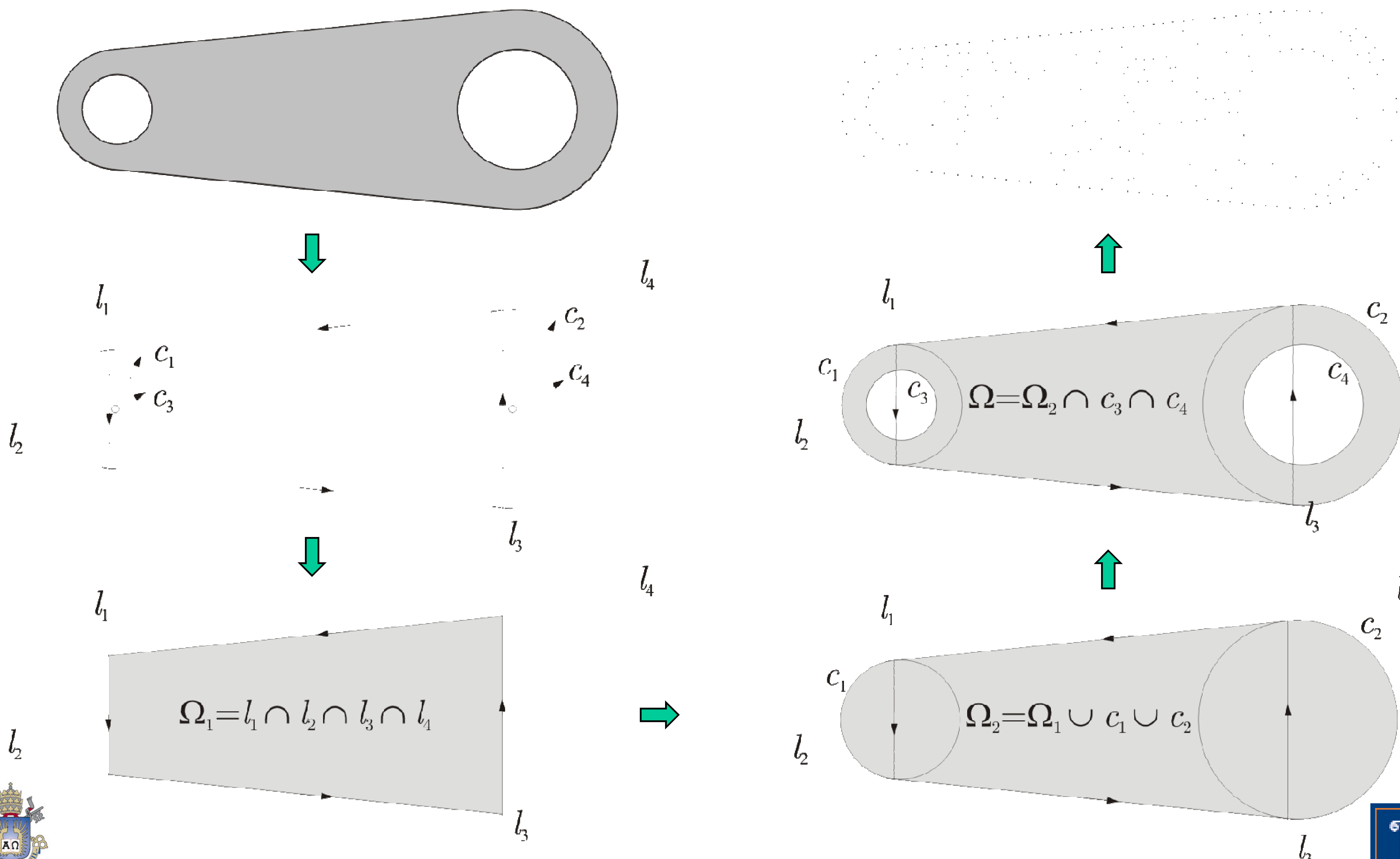
Example: Wrench



Example: Wrench



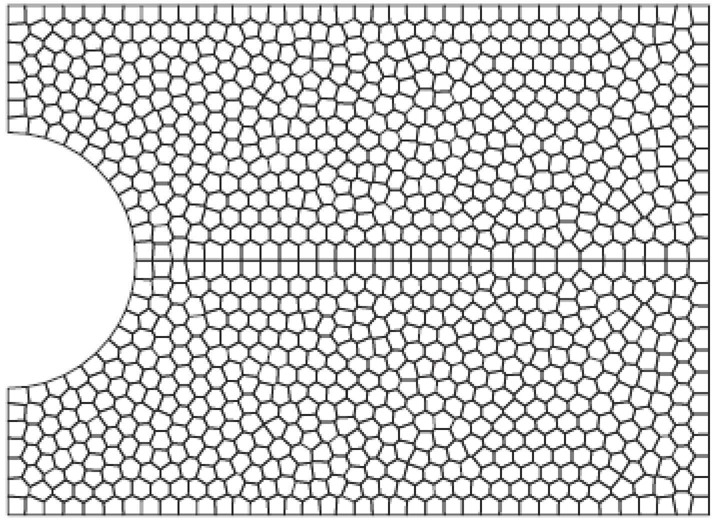
Example: Wrench



Examples

- Matlab demo

Example: symmetric Michell cantilever

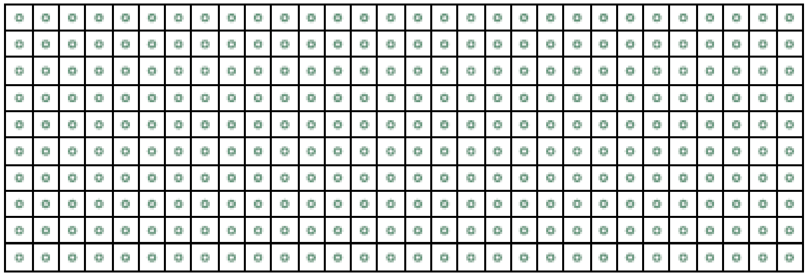


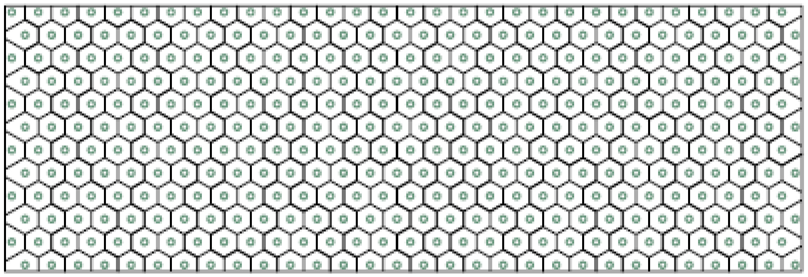
```
function [Node,Element,Supp,Load,P] = PolyMesher(Domain,NElem,MaxIter,P)
...
while(It<=MaxIter && Err>Tol)
  Alpha = c*sqrt(Area/NElem);
  P = Pc;                                        %Lloyd's update
  R_P = PolyMshr_Rflct(P,NElem,Domain,Alpha);    %Generate the reflections
  [Node,Element] = voronoin([P;R_P]);            %Construct Voronoi diagram
  [Pc,A] = PolyMshr_CntrdPly(Element,Node,NElem);
  ...
end
```

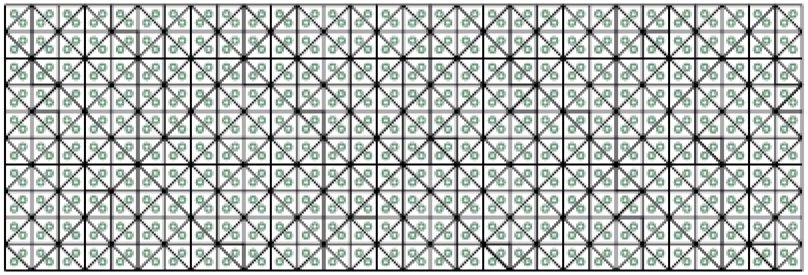
```
  P = [Pc(1:NElem/2,:);[Pc(1:NElem/2,1),-Pc(1:NElem/2,2)]];
```

Example: uniform discretizations

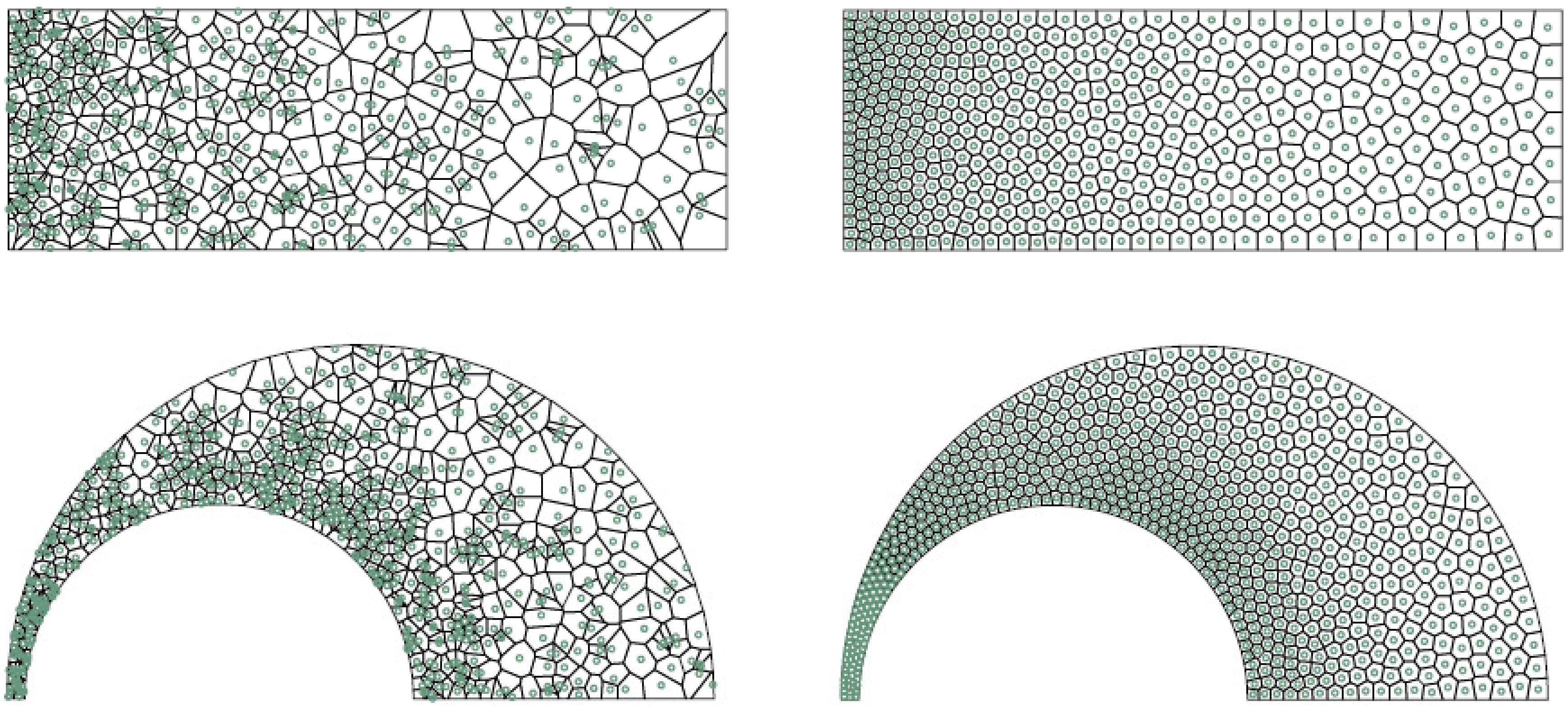
```
>> nelx = 30; nely = 10;
>> dx = 3/nelx; dy = 1/nely;
>> [X,Y] = meshgrid(dx/2:dx:3,dy/2:dy:1);
>> P = [X(:) Y(:)];
>> [Node,Element,Supp,Load] = PolyMesher(@MbbDomain,0,0,P);
```







Example: non-uniform meshes



Concluding remarks

- Voronoi models arise in nature in various situations. In particular, polygonal meshes have been prominent in modeling structural problems;

- A simple and robust code based on the concept of Voronoi diagrams was presented. Using a simple and effective approach allows to discretize two-dimensional geometries with convex polygons;

- Its range of applications is broad, including optimization (shape, topology, etc), and other applications.

Acknowledgment

Tecgraf - Computer Graphics Technology Group at the Pontifical Catholic University of Rio de Janeiro - PUC-Rio - Brazil

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