



A Framework for Generating 2D Voronoi Meshes

Joachim Poudoux
Kitware SAS



M. Berndt, M. Kenamond, M. Shashkov
Los Alamos National Laboratory



MultiMat 2015

Motivation

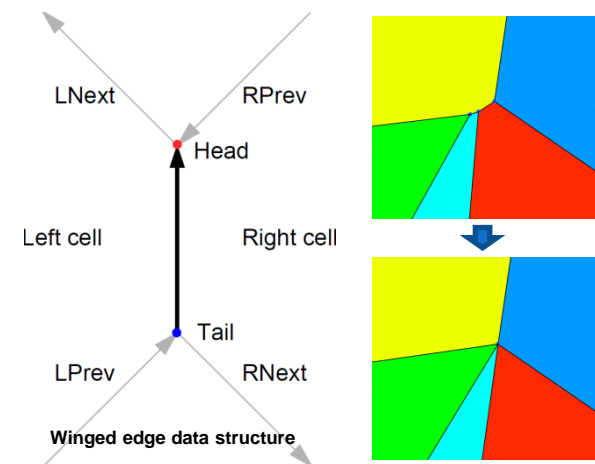
- Computing a 2D Voronoi tessellation is often considered as a straightforward task
- Not so easy if you want it
 - Robust
 - Fast
 - Reusable
- Need a trusted tool to create **2D Voronoi meshes**
 - ReALE context at LANL
 - Clean, robust, documented, easy to build, to use, etc.

What is ShaPo?

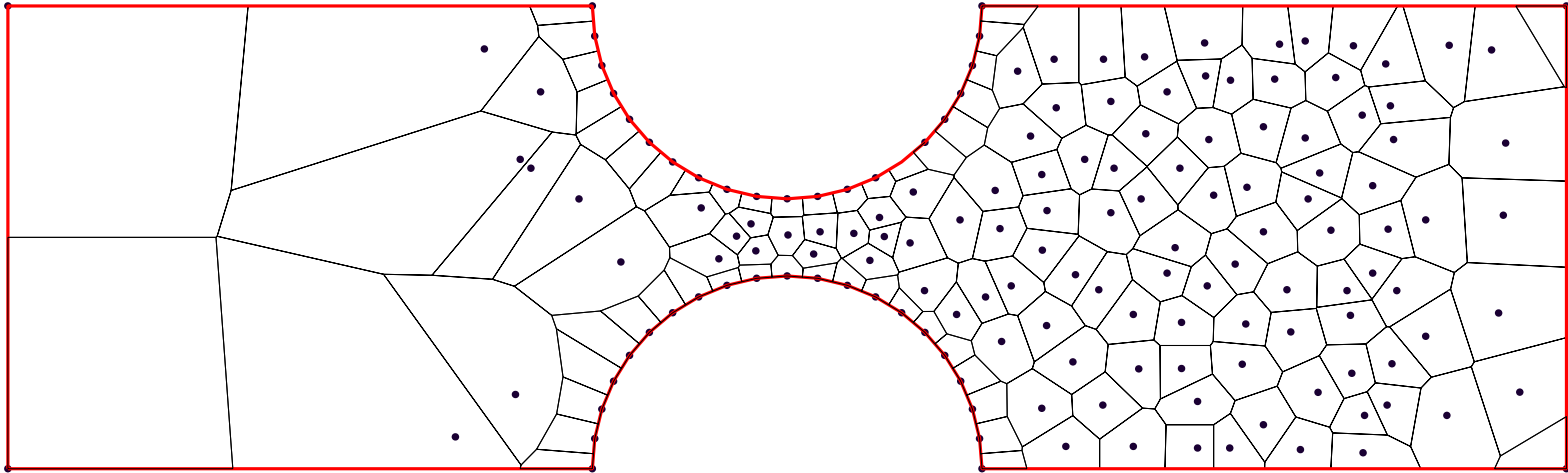
- **Cross-platform library** developed in C++ - based on VTK **to build 2D Voronoi meshes from a set of generators**
- Developed at Kitware (France) for LANL since 2013
- Binding layers to use the tool in C, Fortran and Python
- Works in **serial** or in **parallel** (MPI)
- Computations performed in **double precision**
- Use Shewchuck's **robust** predicates - orientation test
- **Self-diagnostic** & error management
- **Documentations:** doxygen & manuals
- Many **examples**
- **Testing** and dashboard reports – CMake/CDash

ShaPo main features

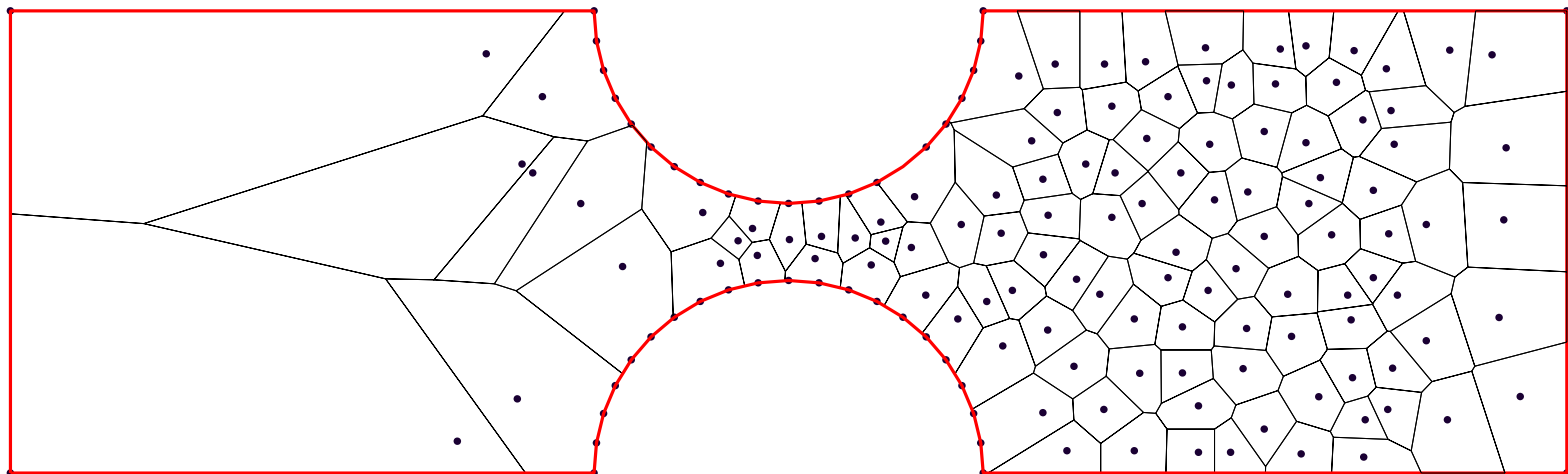
- 2D Voronoi tessellation from a **user provided set of generators within a closed domain**
- Domain boundary: a polygon that can be
 - Convex
 - Simply connected
 - Non-convex
 - Multiply-connected - with or without holes
- Provides **full connectivity** of the generated mesh along with other useful information
 - Points, edges & cells connectivity - using winged edges
 - Input boundary edge a mesh edge stands on, new points introduced on a boundary edge, etc.
- Can **merge** small edges
- Can perform mesh relaxation – **Centroidal Voronoi**
- Mesh **export** and **visualization** functions
- **Remeshing** layer to remesh part of existing mesh



ShaPo 2D tessellators



Constrained Voronoi Tessellation – boundary points are generators



Clipped Voronoi Tessellation – boundary points are NOT generators

Constrained Voronoi tessellation

- Boundary defined by some generators
- Can be computed from the dual graph obtained from the Constrained Delaunay Triangulation [Joe and Wang 1993]
- Algorithm similar to [Tournois et al. 2010] algorithm

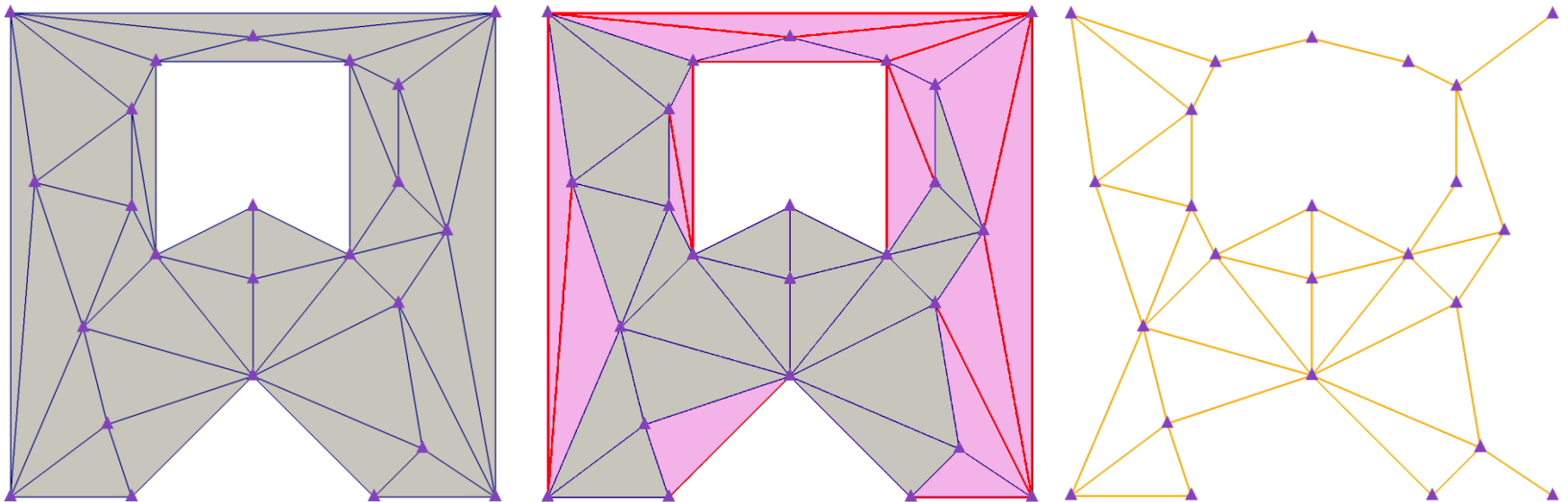
Constrained Voronoi tessellation

Algorithm overview

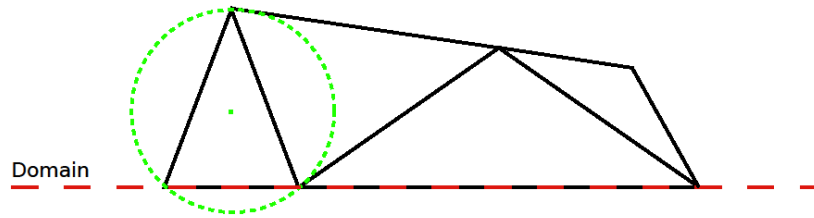
1. Compute the Constrained Delaunay Triangulation (CDT)
 - Shewchuck's Triangle or our implementation of the Divide and Conquer algorithm
2. Analyse the CDT to mark edges that are not part of the Constrained Voronoi dual
3. Build Voronoi cells according the dual graph

Constrained Voronoi dual

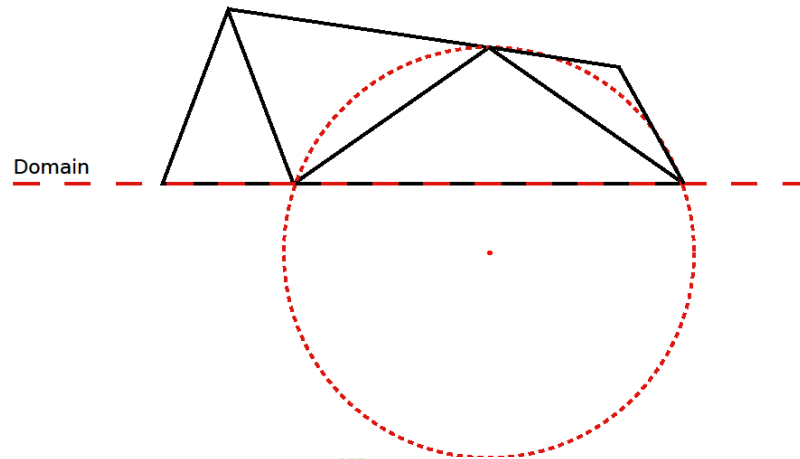
- Constrained Voronoi dual is a subgraph of the CDT
 - Some edges (& triangles) of the CDT must be removed
- Non dual edges link together two generators which cells would meet outside the boundaries



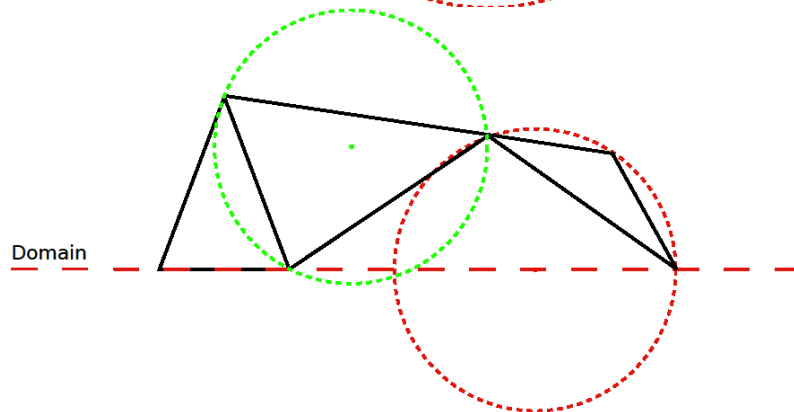
Constrained Voronoi dual construction



Boundary triangle circumcenter is on the left side* the boundary edge
=> the boundary edge **is** a dual edge.
Continue with next boundary triangle.



Boundary triangle circumcenter is on the right side the boundary edge
=> the boundary edge is **NOT** a dual edge.
Tag the edge and the triangle and continue by analyzing its two neighbor triangles

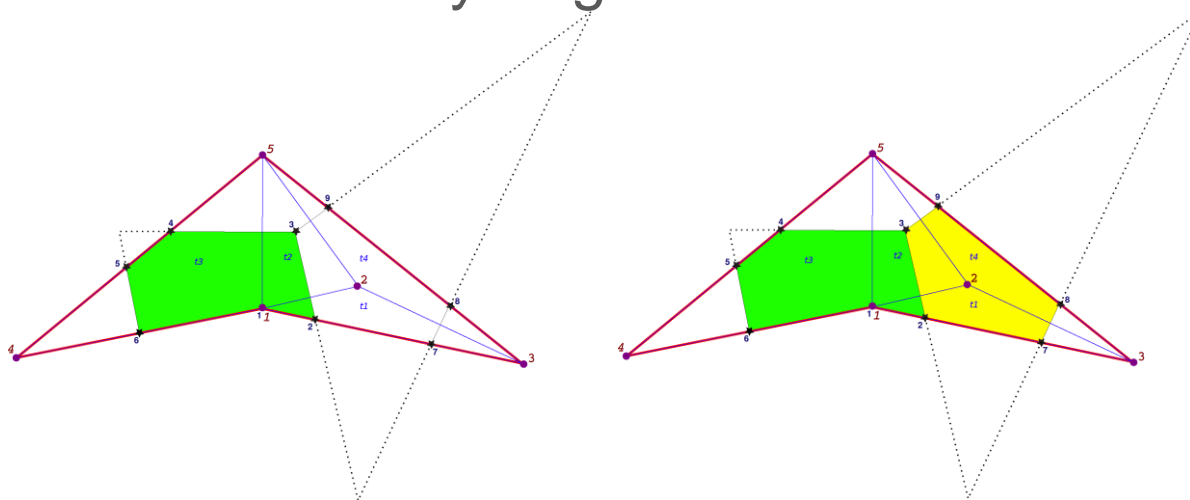


Left triangle circumcenter is on the left side the boundary edge => edge is a dual edge.
Right triangle circumcenter is on the right side the boundary edge => edge is NOT a dual edge.
Continue by analyzing the two neighbor triangles

...

Constrained Voronoi cells construction

- Generated using the obtained dual
- Boundary and internal generator cells are constructed differently by analyzing the triangle fan around the generator. Cell points are either
 - triangle circumcenters
 - or segment-segment intersections computed with identified boundary edges



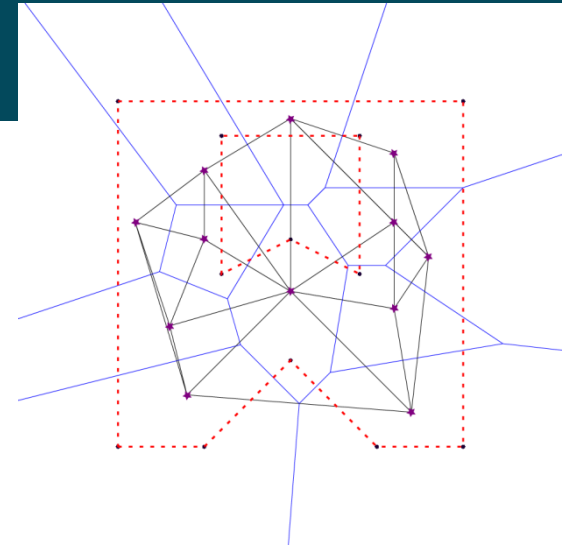
Clipped Voronoi tessellation

- Boundary points are NOT generators – but generators can be set on boundary points
- Based on LLNL Polytope algorithm [Starinshak et al. 2014]

Clipped Voronoi tessellation

Algorithm overview

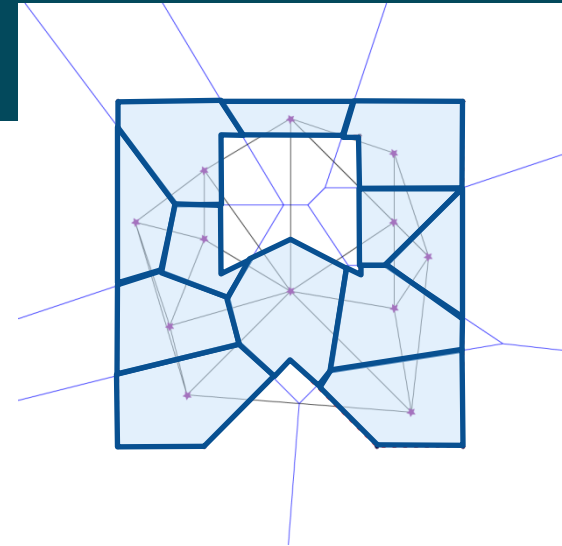
1. Compute the Unbounded Voronoi tessellation from its Delaunay's dual
2. Clip obtained cells against the boundaries



Clipped Voronoi tessellation

Algorithm overview

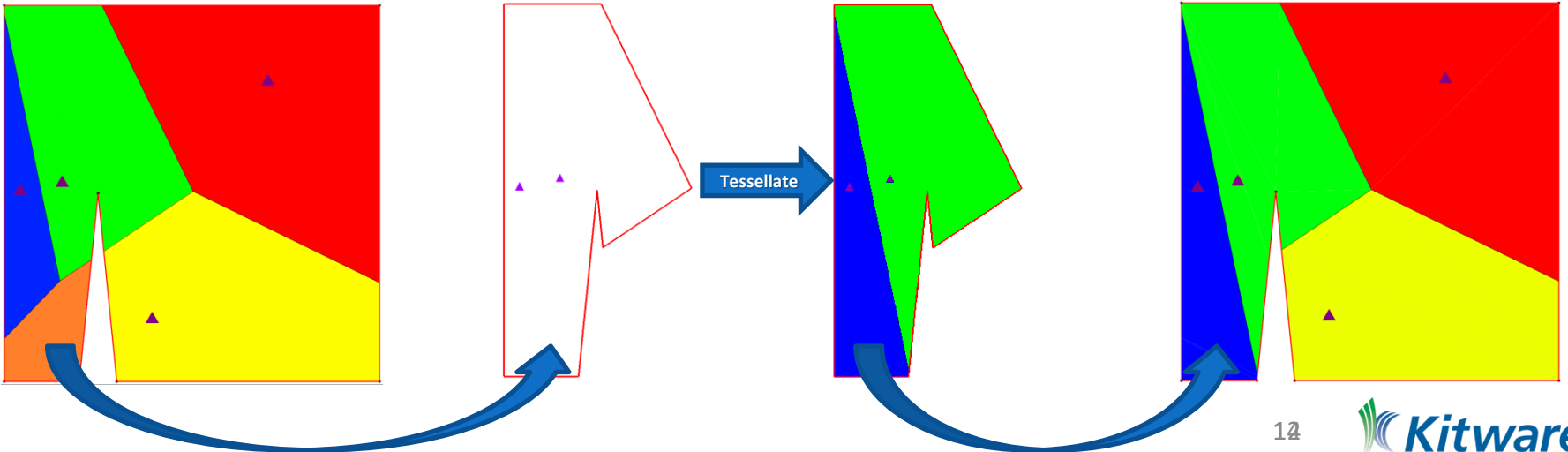
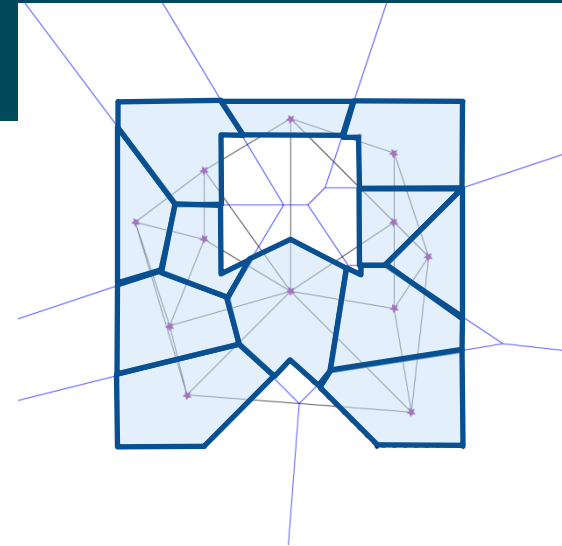
1. Compute the Unbounded Voronoi tessellation from its Delaunay's dual
2. Clip obtained cells against the boundaries



Clipped Voronoi tessellation

Algorithm overview

1. Compute the Unbounded Voronoi tessellation from its Delaunay's dual
2. Clip obtained cells against the boundaries
3. Process orphan cells
 - Clipping against concave boundaries can split a cell in multiple piece.
Orphan cells: Pieces that do not contain a generator
 - Local tessellation is performed to “adopt” orphans cells, then merged



Clipping

- Clipping polygons is an expensive operation
 - We perform convex polygon vs. multi-connected non-convex boundary polygon
 - ShaPo implements an extended version of [Greiner and Hormann 98] algorithm to support degeneracies
- Is it really necessary to clip ALL cells?
 - Most internal cells do not intersect boundaries, can't we avoid the useless intersection test?

Clipped Voronoi tessellation

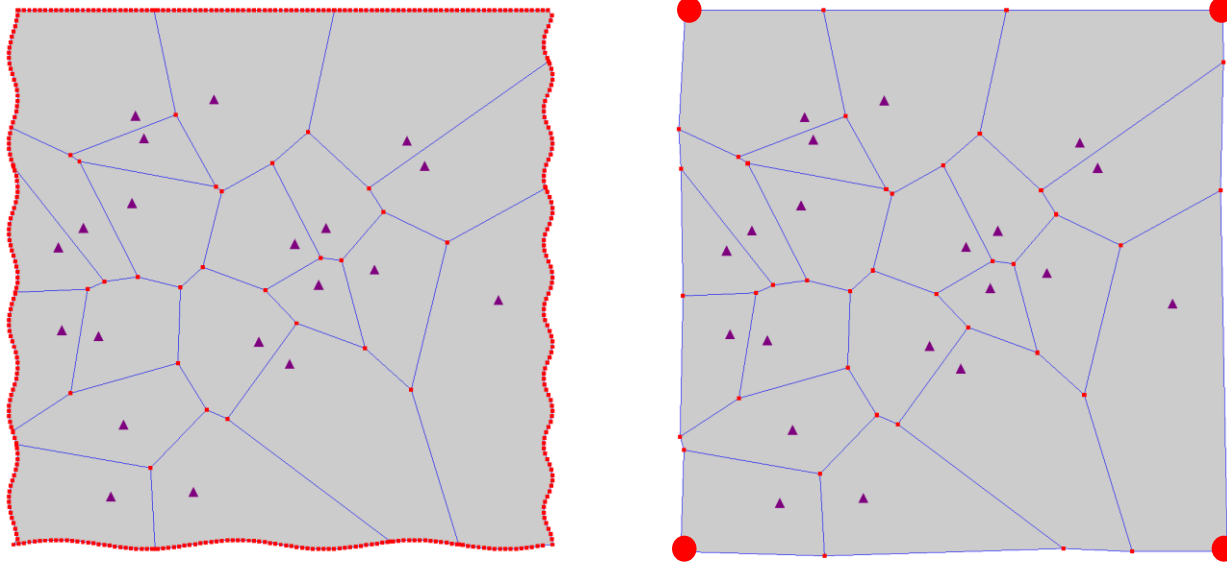
Improved algorithm overview

1. Compute the Unbounded Voronoi tessellation from its Delaunay's dual
2. Compute the Constrained Voronoi dual (CVD) from generators and boundary
3. Clip cells that are connected to a boundary point in the CDT or belongs to a non CVD triangle against the boundaries
4. Process orphan cells

Empirical speed-up: from 0.8 to 13

Clipping with high-res boundaries

- You don't like heptacontagon and hectogons?



Some boundary points are important corners (user's choice), other are not inserted in the output mesh

Not a post-processing step: performed on the fly as we master the clipping algorithm

Parallel tessellation

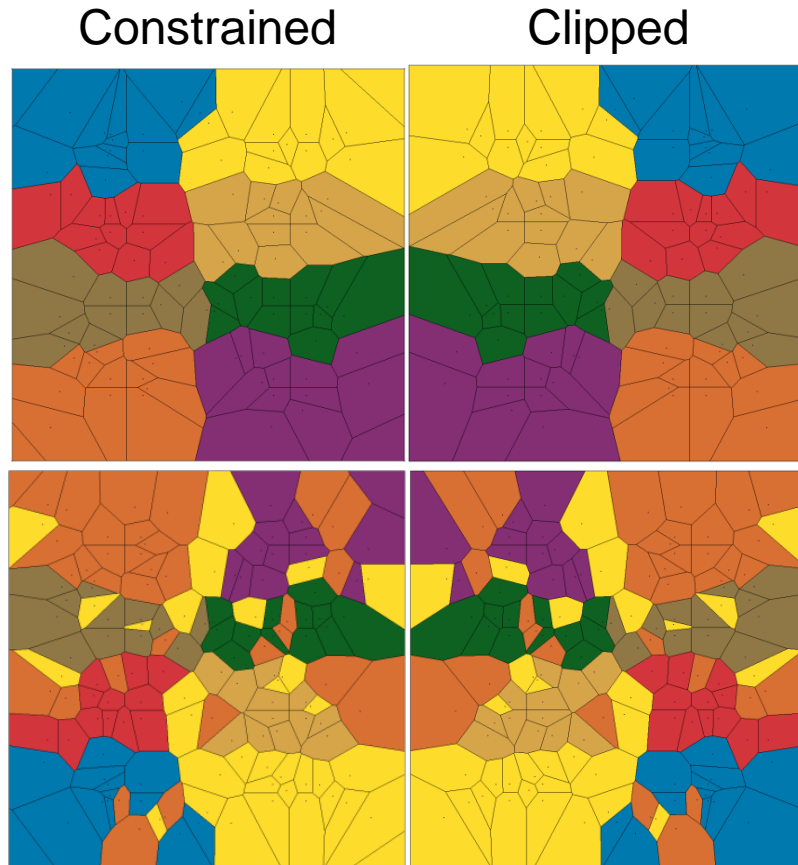
- Perform Voronoi tessellation (Constrained or Clipped) in a distributed memory context using MPI
- Each processor provides its generator set and potentially some parts of the boundary edges that define the domain
- Each processor obtains the Voronoi cells corresponding to the generators it owns
- Based on [Starinshak et al. 2014] algorithm
 - Robust approach but can lead to unsovable problems
 - Eg. A processor may have to compute the whole problem

Parallel tessellation

Algorithm overview

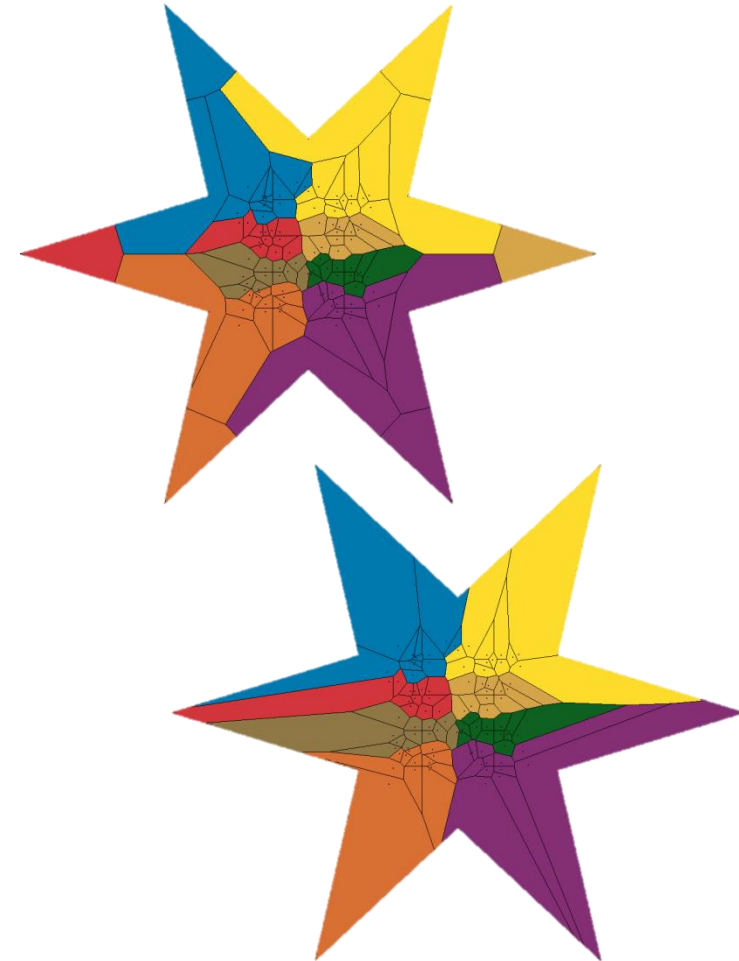
1. Compute the locally extended input
 1. Compute and gather convex hull of local inputs
 2. Compute the DT of the global convex hull
 3. All gather boundary edges/points
 4. Determine the neighborhoo processors
 - by checking DT edges that involve a local point
 - by intersection of subdomain convex hull
 5. Insert neighbor generators to the local problem
2. Compute Voronoi tessellation with serial algorithm
3. Compute communication information - points & edges

Constrained & clipped in parallel



Up: parallel tessellation on 8 processors

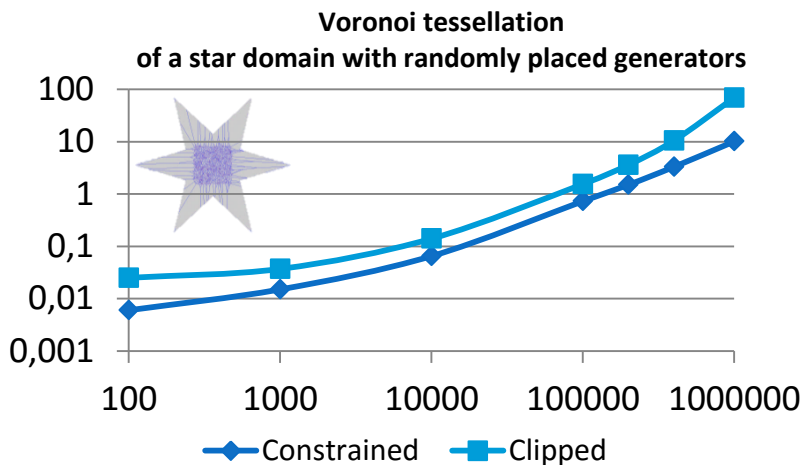
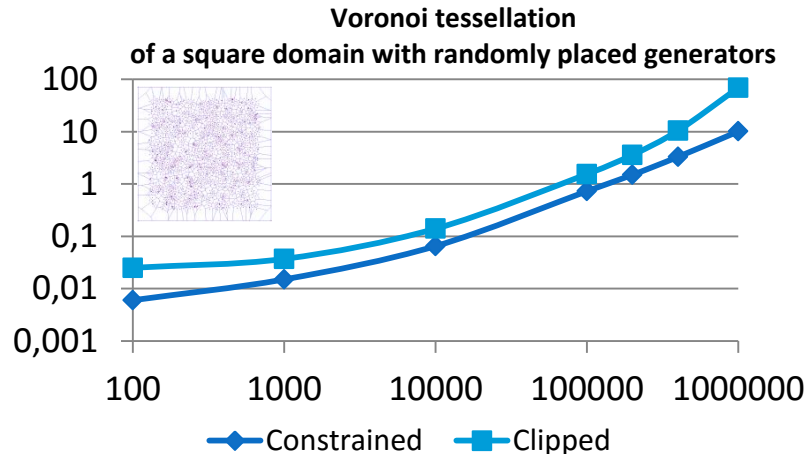
Down: 2 processors (yellow and orange) have generators spread in the whole domain area (worst case)



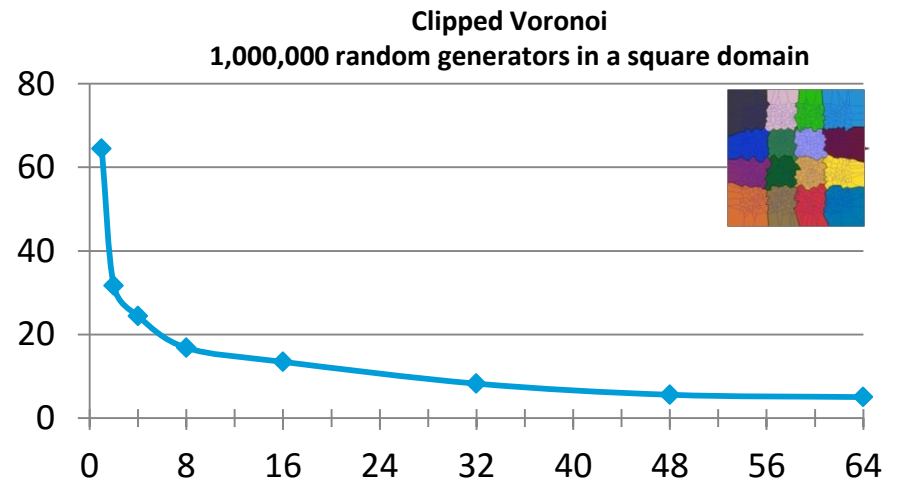
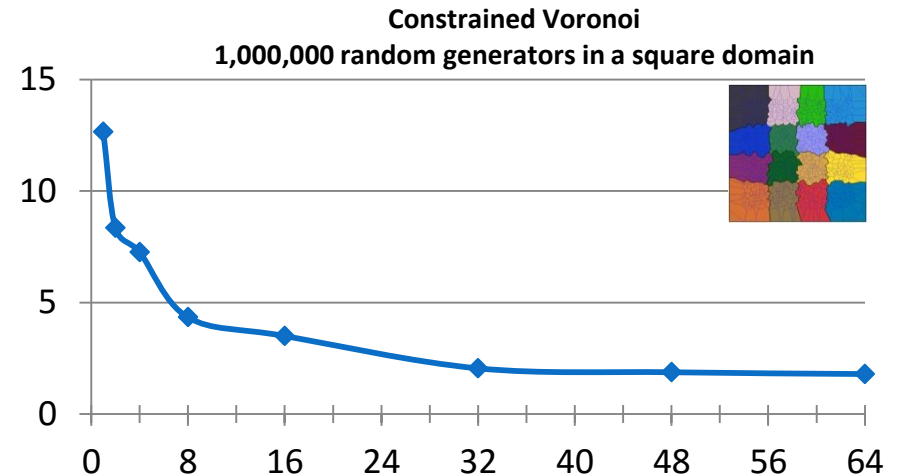
100 generators on 8 processors

Results*

Serial



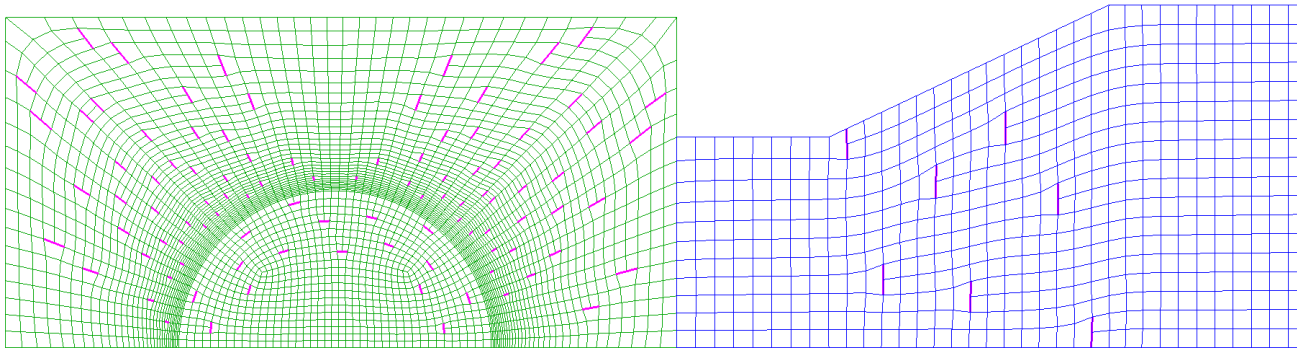
Parallel



* Performed on a big computer with 64 cores on 4 sockets (AMD Opteron 6386 SE 2.8 Ghz), 1/2 To of RAM

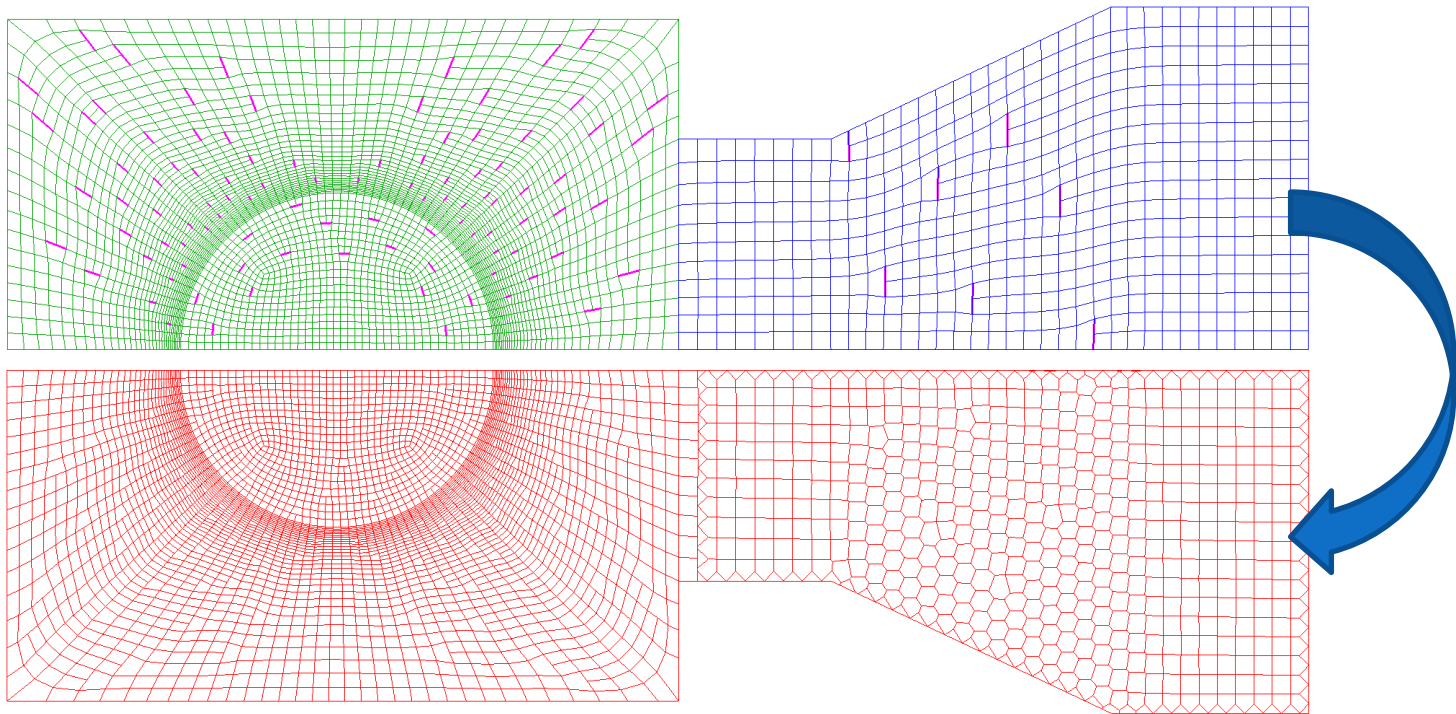
Remeshing

- Consists in replacing some regions of an existing unstructured mesh with a Voronoi mesh



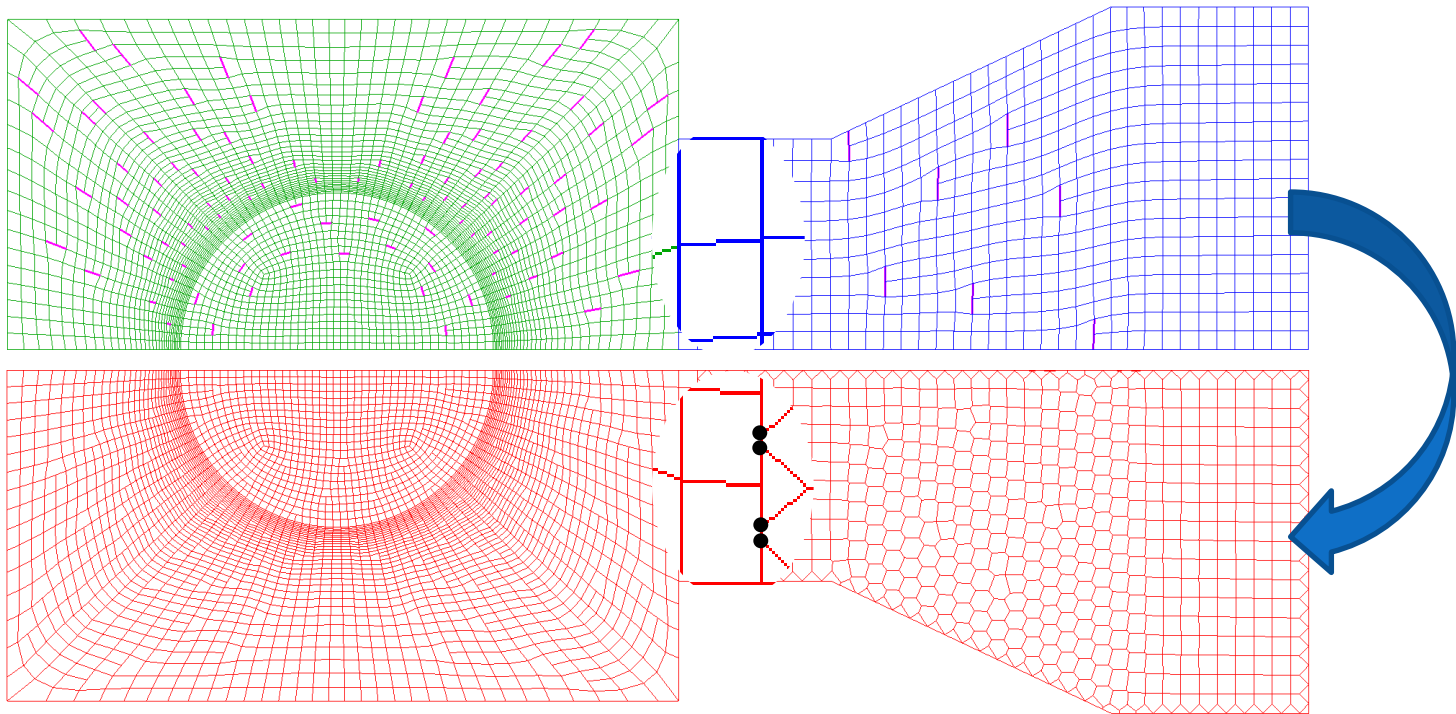
Remeshing

- Consists in replacing some regions of an existing unstructured mesh with a Voronoi mesh



Remeshing

- Consists in replacing some regions of an existing unstructured mesh with a Voronoi mesh
- Shape of boundary cells do not change but their connectivity does

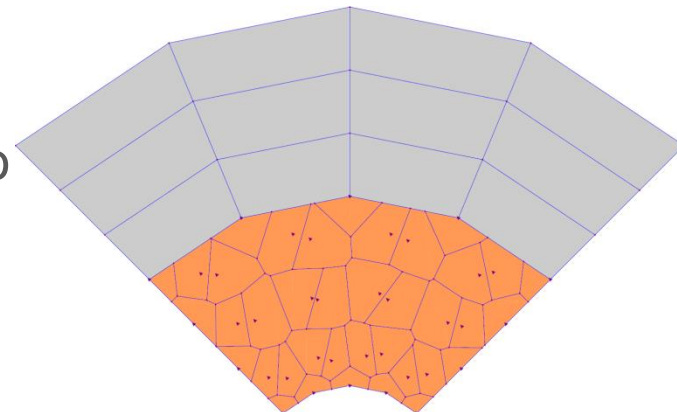
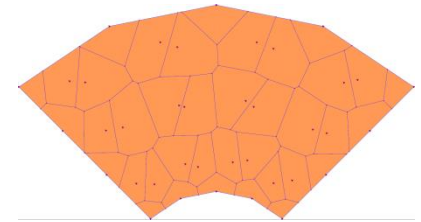
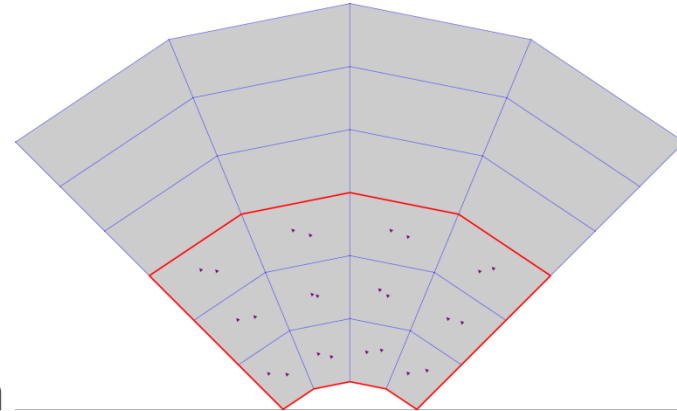


Remeshing features

- Input
 - Unstructured mesh geometry & topology
 - Edges that define the closed subdomain to remesh
 - Can be multi-connected (contains hole)
 - Convex or non-convex
 - Set of generators inside the remeshing subdomain
 - Type of Voronoi tessellation
- Output
 - Unstructured mesh with full connectivity
 - Maps to retrieve indices of old elements in the new mesh
 - Master points pair for every slave point added on a subdomain boundary edge
 - Connectivity information in parallel

Remeshing Algorithm Overview

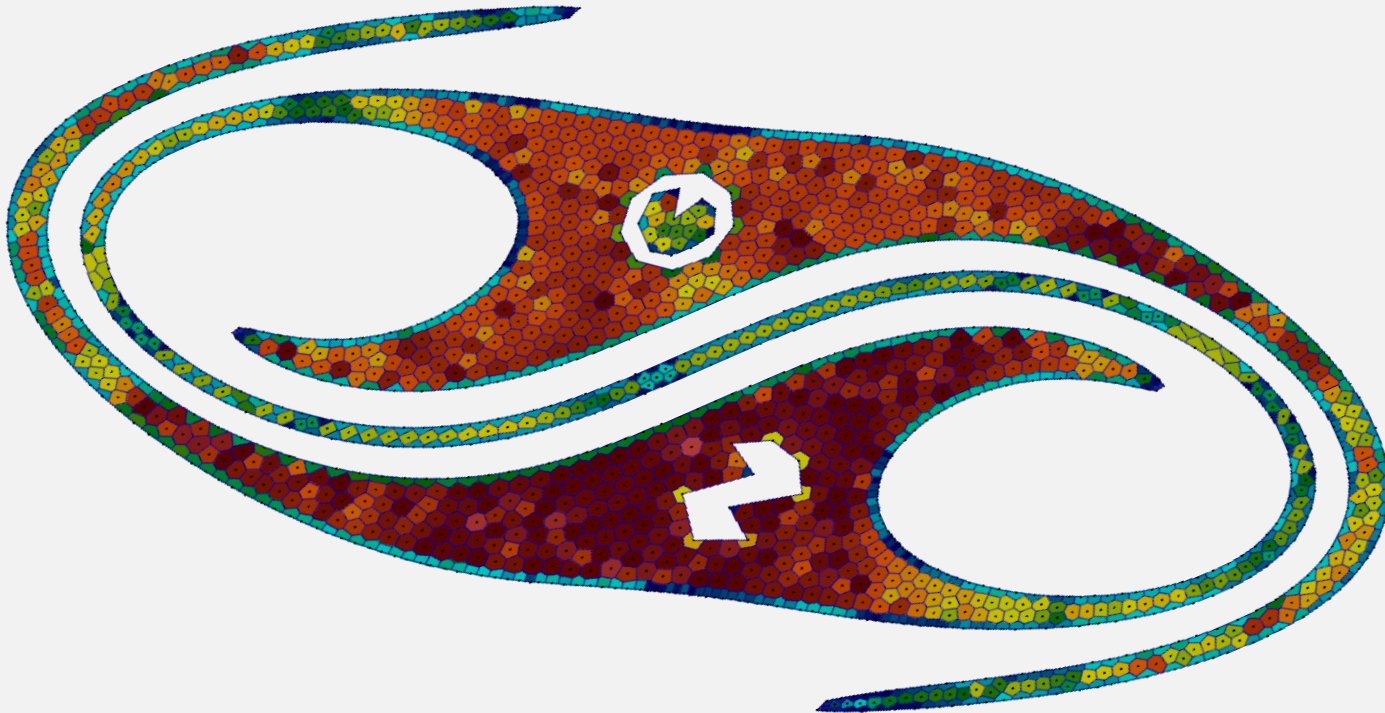
1. Extract the sub-domain boundary edges and flag all the input mesh cells that are outside the sub-domain to remesh
2. Solve the tessellation of the sub-domain with the provided generators
3. Browse initial boundary edges and fetch the new edge chain that replace each of them
4. Merge the initial flagged mesh cells and points that stand outside the sub-domain into the obtained tessellation and resolve input boundary edges by their new edge chain



Summary

- ShaPo provides a complete toolkit to generate 2D Voronoi meshes from generators in serial and in parallel
- Future work
 - Optimization of parallel tessellators
 - Redistribute generators over processors
 - **3D tessellation**

Danke schön



This work was performed under the auspices of the National Nuclear Security Administration of the US Department of Energy at Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396. We gratefully acknowledge the partial support of the US Department of Energy National Nuclear Security Administration Advanced Simulation and Computing (ASC) Program.

References

- B. Joe and C.A. Wang, « Duality of constrained voronoi diagrams and Delaunay triangulations », *Algorithmica*, 9(2), pp. 142-155, 1993.
- J. Tournois, P. Alliez and O.Devillers, « 2D Centroidal Voronoi Tessellations with Constrains », *Numerical Mathematics: Theory, Methods and Applications*, 3(2), pp. 212-222, 2010
- D.P. Starinshak, J.M. Owen and J.N. Johnson, « Polytope: A New Parallel Framework for Computing Voronoi Meshes on Complex Boundaries », *Proc. SIAM Conf. on Parallel Processing and Scientific Computing*, 2014
- G. Greiener and K. Hormann, « Efficient clipping of arbitrary polygons », *Association for Computing Machinery, Transactions on Graphics*, 17(2), pp. 71-83, 1998