Triangular meshing for seismology

Keith Roberts, Rafael Gioria,
other Workstream 4 members
STMI Project

Universidade de São Paulo

Research Centre for Gas Innovation

Outline

Universidade de São Paulo

1. Purpose
2. Software architecture
3. Mesh sizing function
4. Mesh generation algorithm
5. Applications

Purpose

Universidade de São Paulo

- This work aims to create end-to-end workflows to build quality two- and three-dimensional (2D and 3D) unstructured triangular and tetrahedral meshes for seismic domains suitable for numerical wave propagators.

- Workstream 4 has been developing:
  - ***SeismicMesh***: Software for triangular mesh generation for seismology.
    - Automatic (i.e., no manual geometry creation).
      - No point clicking or drawing lines!
    - Support for distributed memory parallelism in both 2D and 3D.

https://github.com/krober10nd/SeismicMesh

-Open-source
-CI (89% code coverage)
-PEP compliance.
-Cmake build system
-Self-documentation (in progress)

Software Architecture

Universidade de São Paulo

- Python and C++ bound together using Pybind11.
    - Computational Graphic Algorithms Library (CGAL) and Boost are used for all low level geometrical operations.
    - MPI4py, Numpy, Scipy, MeshIO.
- Pre- and post processing utilities (e.g., input file creation, mesh size function class, boundary conditions, etc.).

Software Architecture

Universidade de São Paulo

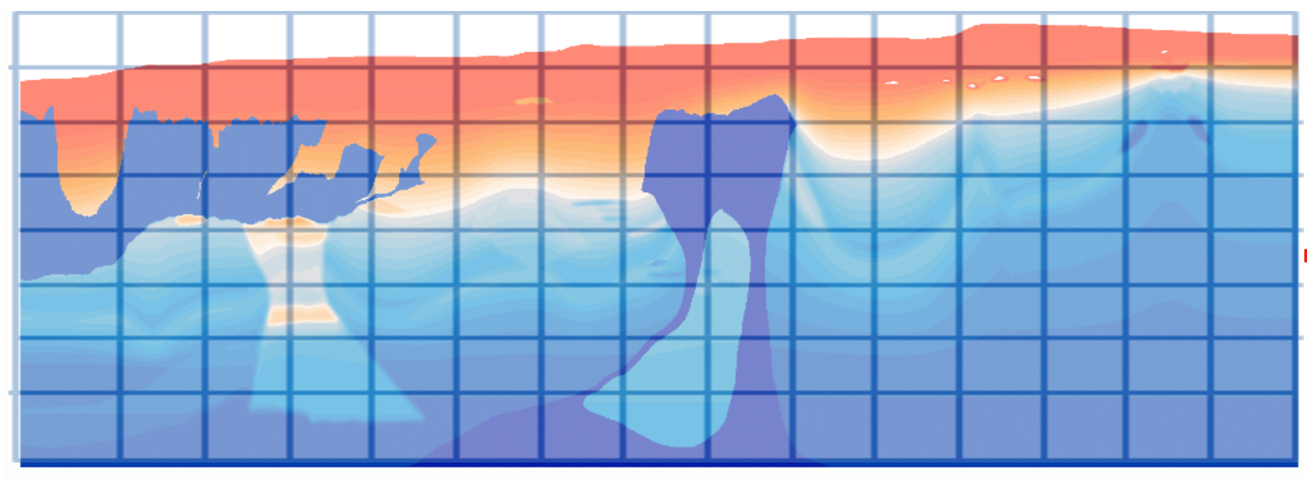
```
import SeismicMesh


def example_2D():
    # Name of SEG-Y file containg velocity model.
    fname = "velocity_models/vel_z6.25m_x12.5m_exact.segy"  input
    bbox = (-12e3, 0, 0, 67e3)  input

    # Construct mesh sizing object from velocity model
    ef = SeismicMesh.MeshSizeFunction(
        bbox=bbox,
        model=fname,
        domain_ext=1e3, input
        dt=0.001,  input
        grade=0.15,  input
        freq=5,  input
        wl=5,  input
        hmax=1e3,  input
        hmin=50.0,  input
    )

    # Build mesh size function
    ef = ef.build()

    ef.WriteVelocityModel("BP2004")

    # Visualize mesh size function
    ef.plot()

    # Construct mesh generator
    mshgen = SeismicMesh.MeshGenerator(
        ef, method="cgal"
    )  # if you have cgal installed, you can use method="cgal"

    # Build the mesh (note the seed makes the result deterministic)
output    points, facets = mshgen.build(max_iter=50, nscreen=1, seed=0)
```

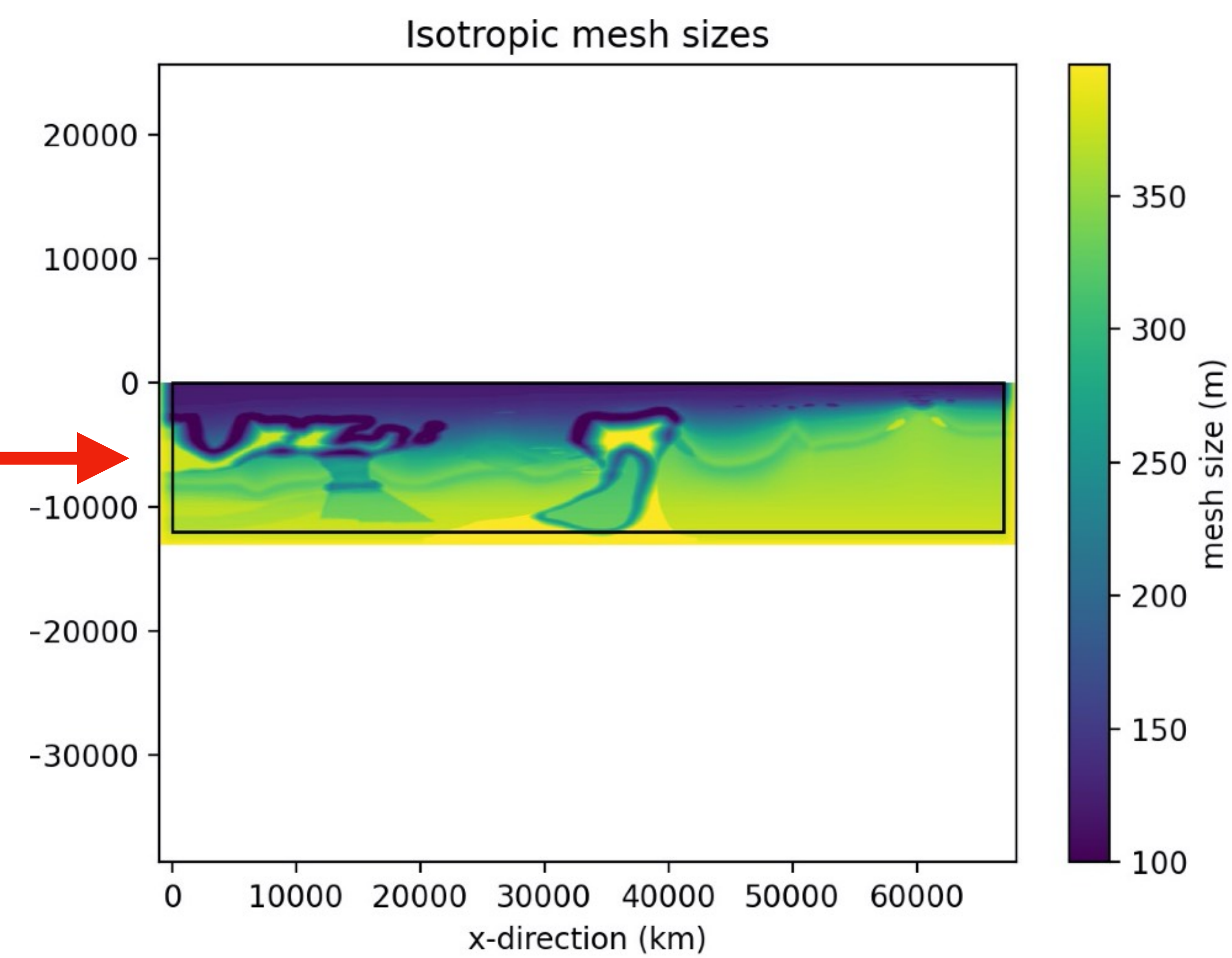
A python package MeshIO is used for file i/o

Mesh sizing functions

Universidade de São Paulo

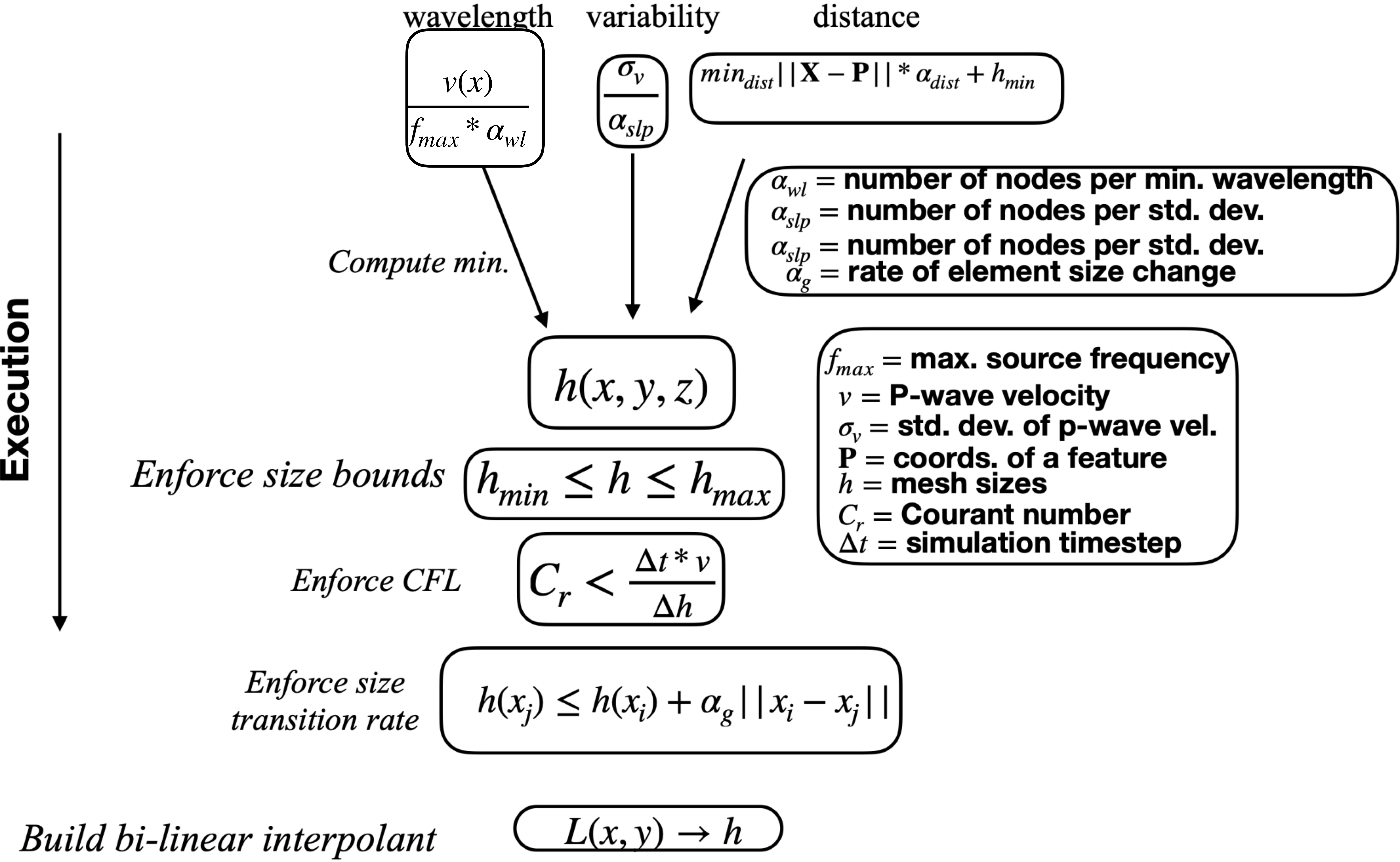
- User parameterizes the distribution of mesh resolution:





$$wavelength - to - gridscale(x) = \frac{v_p(x)}{f_{max} * \alpha_{wl}}$$

Building the sizing func.

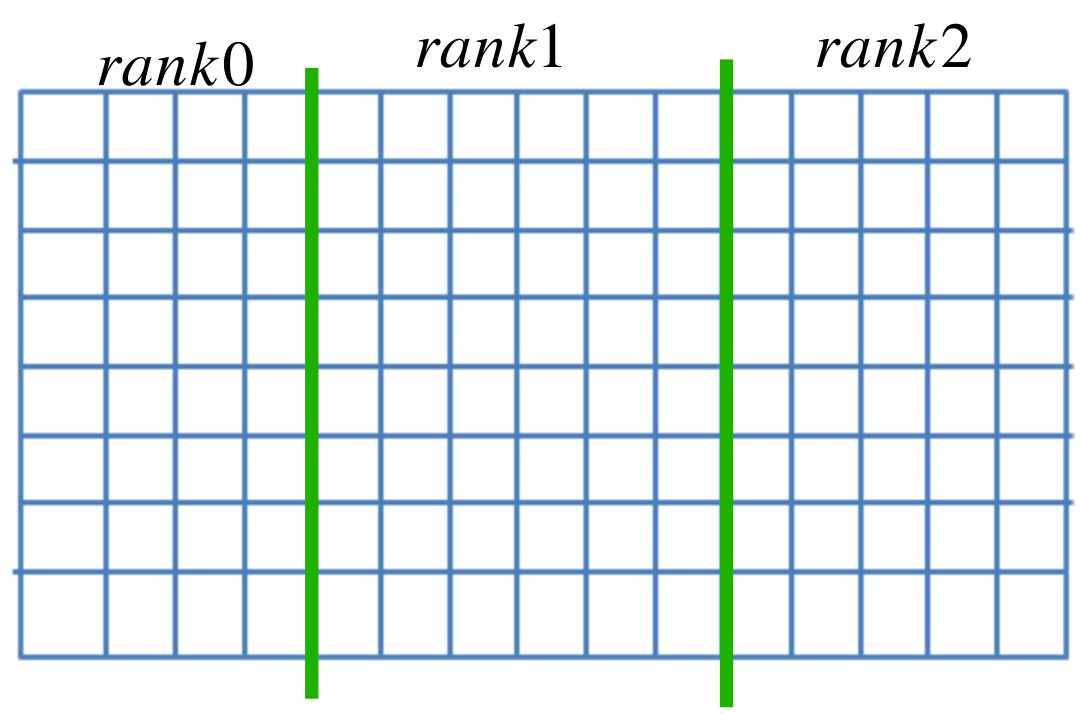


Mesh sizing functions

Universidade de São Paulo

- Sizing functions are defined on Cartesian grids
  - Faster query than unstructured.
  - No need to store connectivity of grid.
  - Easy to parallelize.

- Stored as a Scipy.RegularGriddedInterpolant.



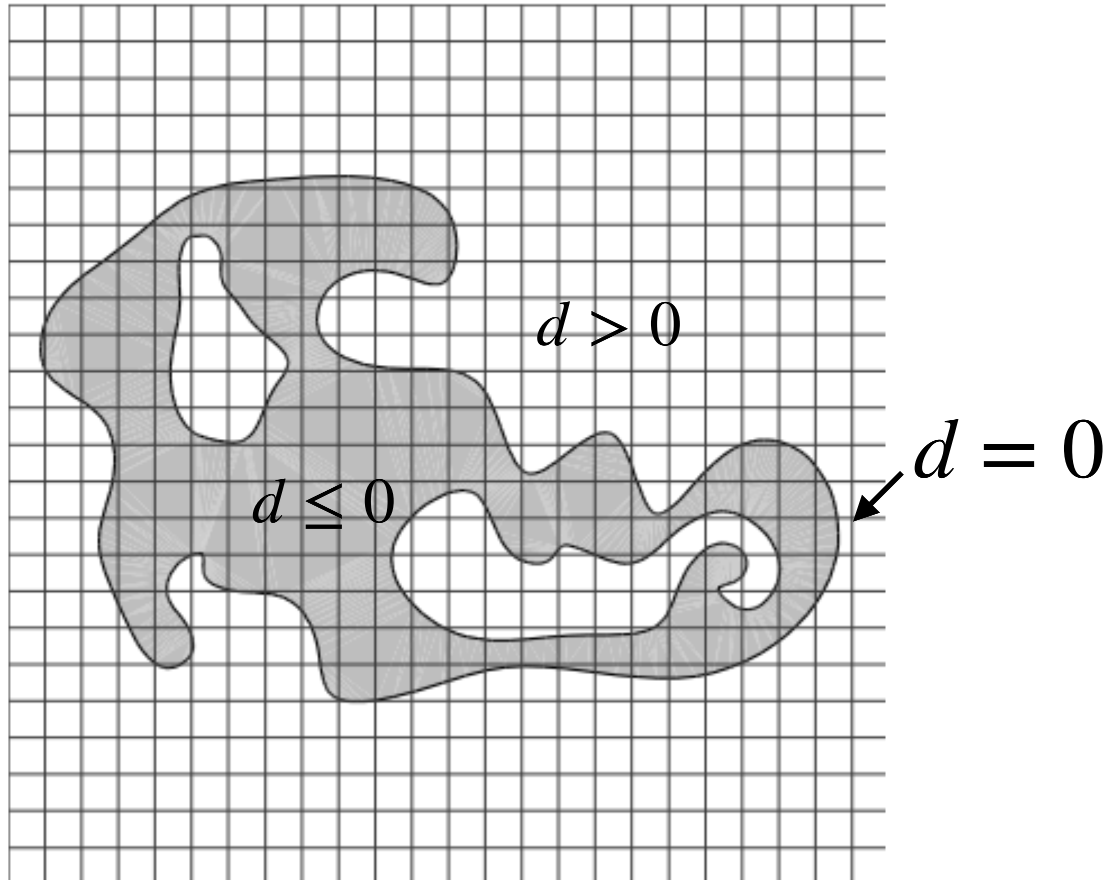
Signed distance functions

- Signed distance function/Implicit domain definition:

$$\Omega := \left\{ \boldsymbol{x} \in \mathbb{R}^2 : d(\boldsymbol{x})_\Omega \leq 0 \right.$$

$$\partial\Omega := \left\{ \boldsymbol{x} \in \mathbb{R}^2 : d(\boldsymbol{x})_\Omega = 0 \right.$$

- Similar to the mesh sizing functions, signed distance functions can also be defined on structured grids and stored as gridded interpolants



Signed distance functions (SDFs)

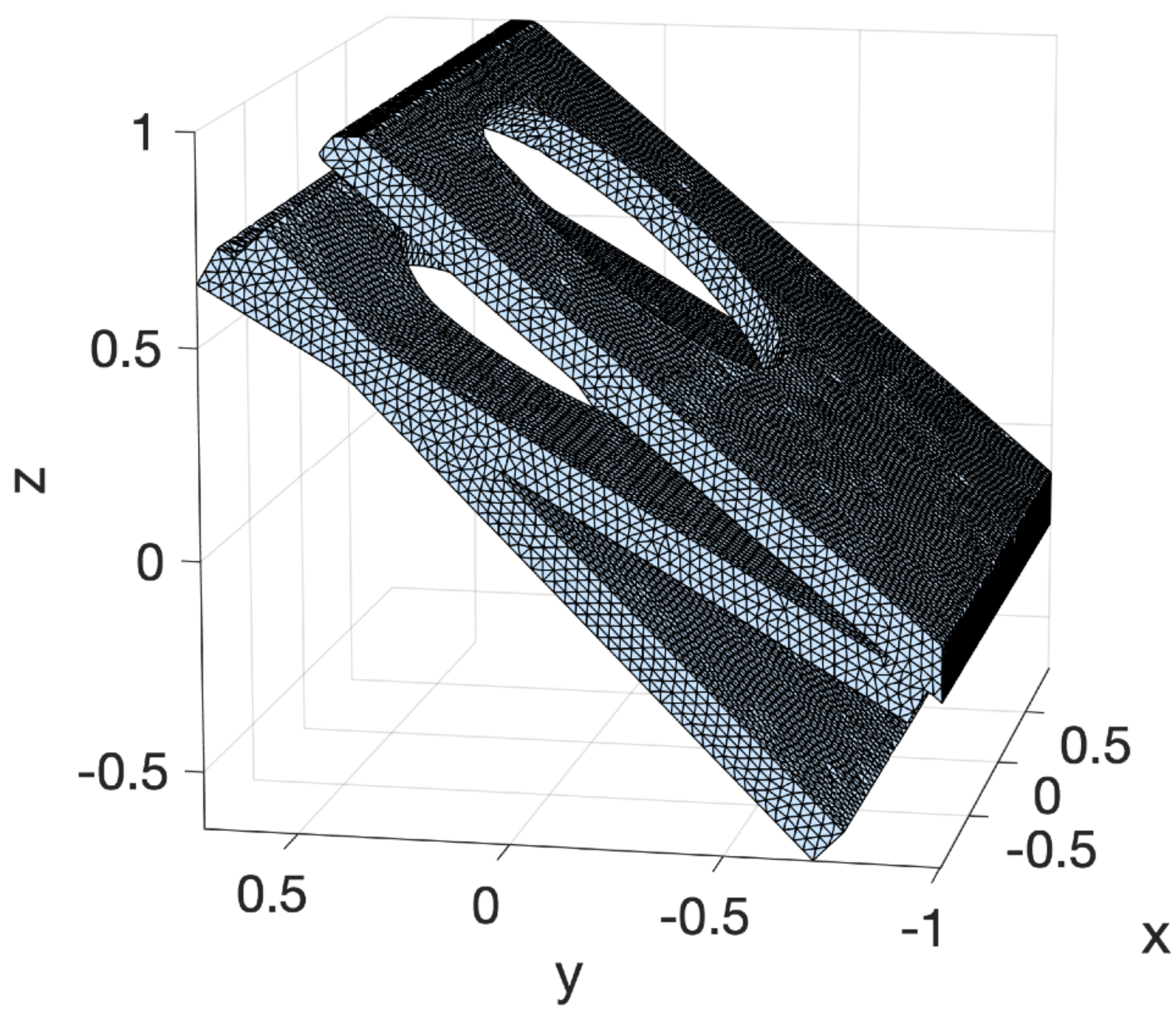
- For some geometries, analytical signed distance function exists.
  - Simple primitives such as cubes, conics, and spheres can be used.

Minimum distance to a rectangle:

$$d = -\min(\min(\min(-y_1 + Y, y_2 - Y), -x_1 + X), x_2 - X)$$

Set operations with SDF:

```
function d=opSmoothUnion(d1, d2, k )
   h = max( k-abs(d1-d2), 0.0 );
   d = min( d1, d2 ) - h.*h.*0.25./k;
end

function d=opSmoothIntersect(d1, d2, k )
   h = max(k-abs(d1-d2),0.0);
   d = max(d1, d2) + h*h*0.25/k;
end

function d=opSmoothSubtraction( d1, d2, k)
   h = max(k-abs(-d1-d2),0.0);
   d = max(-d1, d2) + h*h*0.25/k;
```



Mesh generation

Universidade de São Paulo

- Modifications to DistMesh [2] algorithm.
  - Uses signed distance functions to define the domain.

Algorithm 1: The *DistMesh* algorithm modified from [3].

Result: A high-quality Delaunay triangulation adapted to a user-defined sizing map and conforming to a domain defined by the user-defined signed distance function.

1. Form initial point distribution according to sizing map (done in parallel if enabled).;

if $iter < max\_iter$ **then**

  2. Compute Delaunay triangulation of points.;
  3. Remove triangles with centroids outside of domain.; → **query SDF**
  4. Move points based on forcing function; → **query sizing function**
  5. Project any points outside the domain back inside; → **query SDF**

  if *parallel* **then**

    6. Remove halo vertices added in 2.

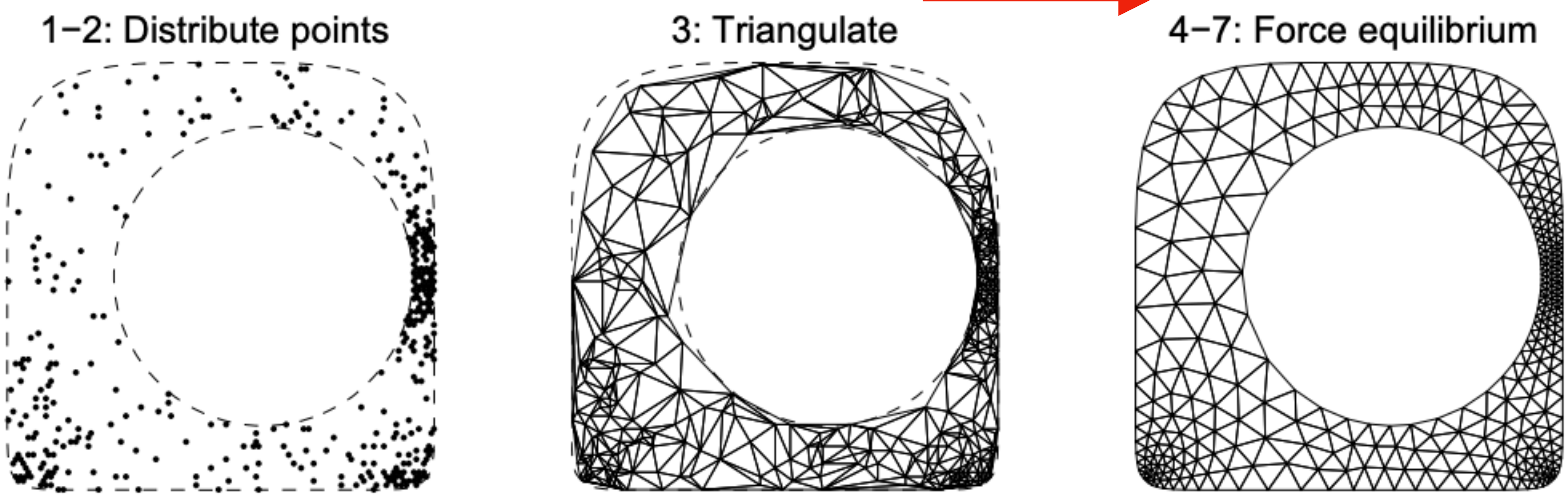
  end

end

Mesh generation

Universidade de São Paulo

50-100 meshing iterations

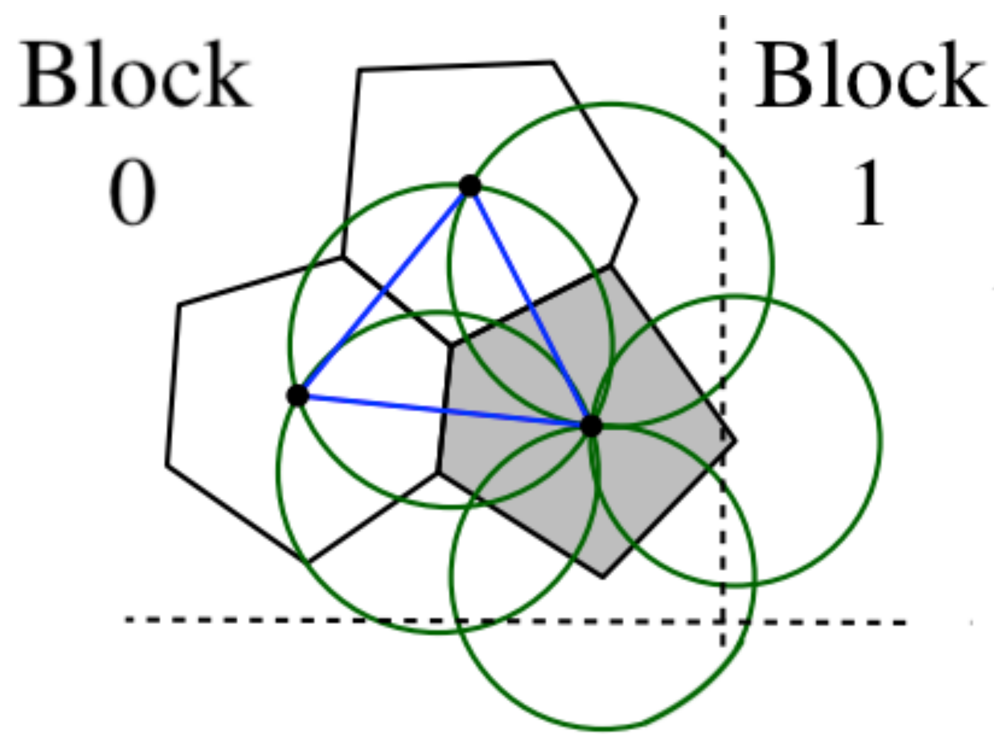


Parallelism

1. *DistMesh* requires a re-triangulation each meshing iteration.
2. Requires ~50-100 iterations to converge to a “high”-quality triangulation.

If we can parallelize Delaunay re-triangulation all other components of DistMesh are trivially parallel.

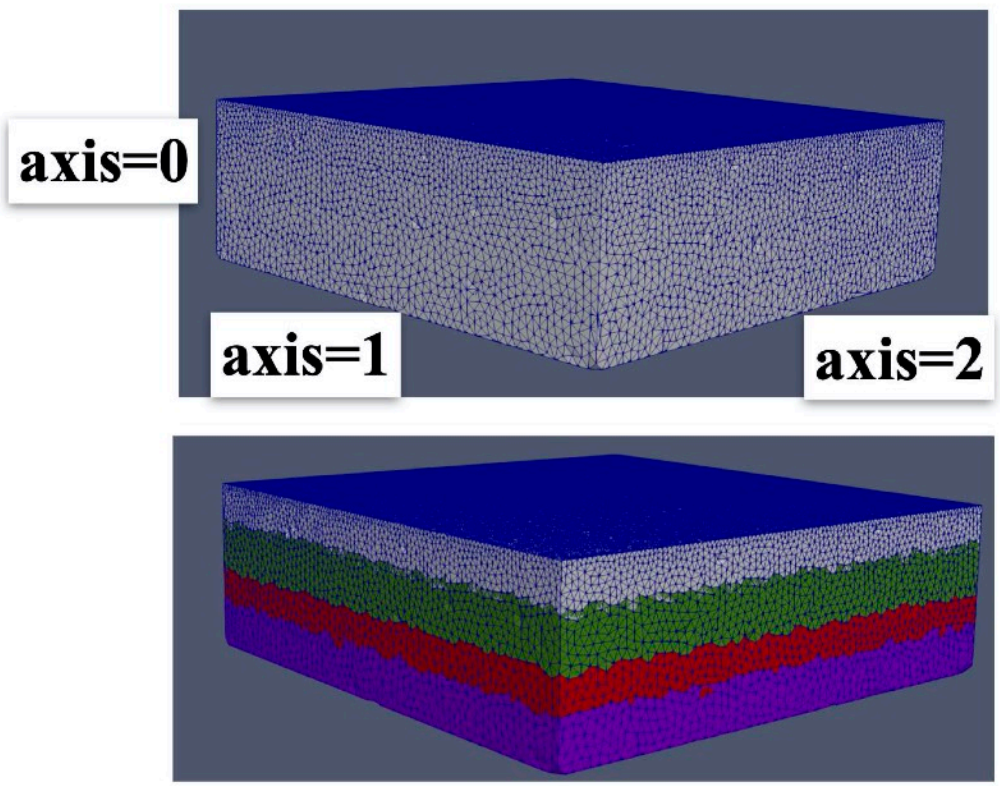
Modified the methods proposed [4]:

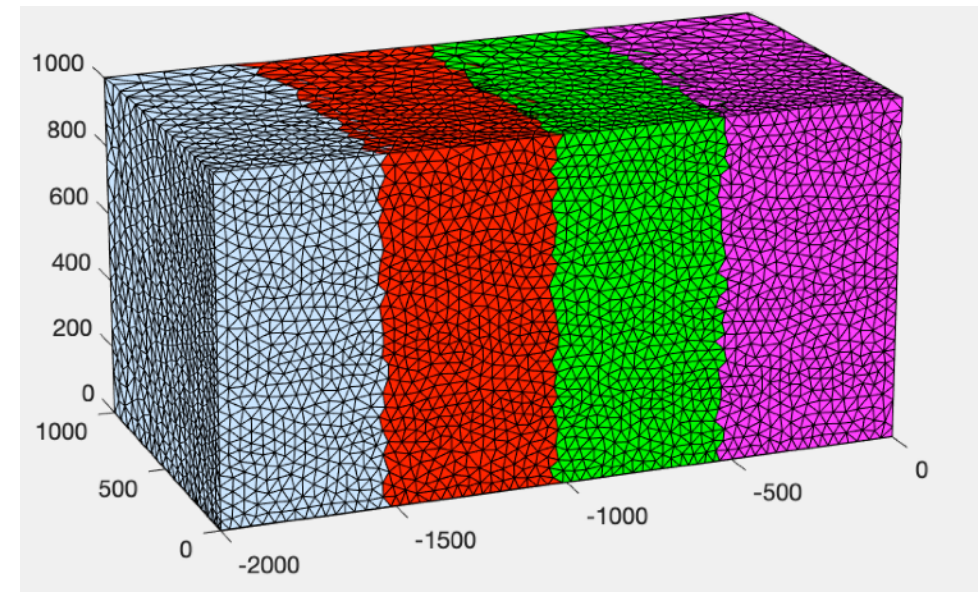


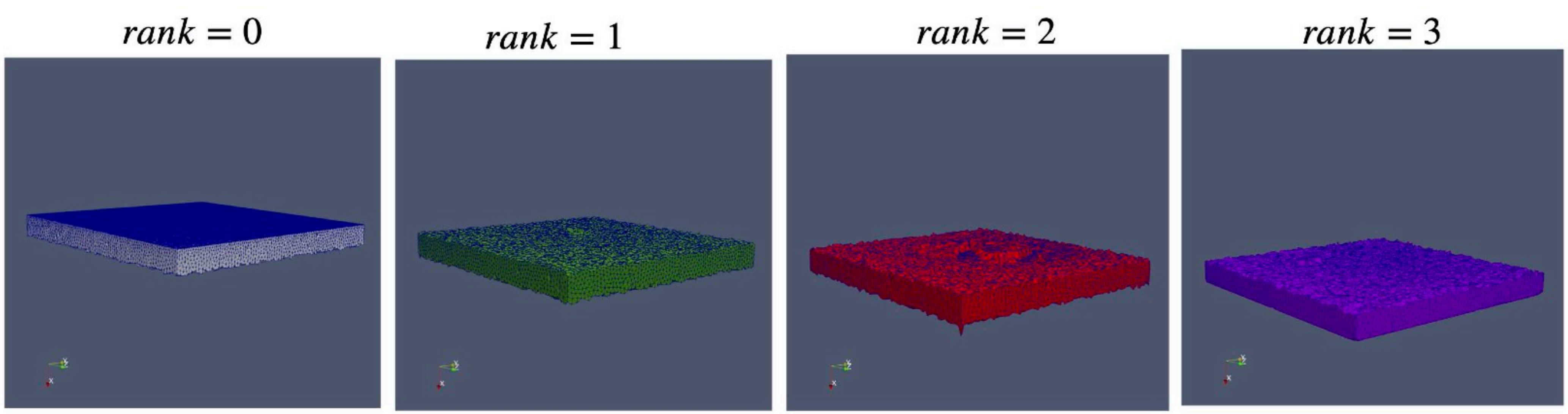
Requires one communication step per meshing iteration to re-triangulate point set in parallel. Simplicity comes from, in part, the domain decomposition topology and Delaunay property.

Domain decomposition

- Load balancing has not yet been considered.

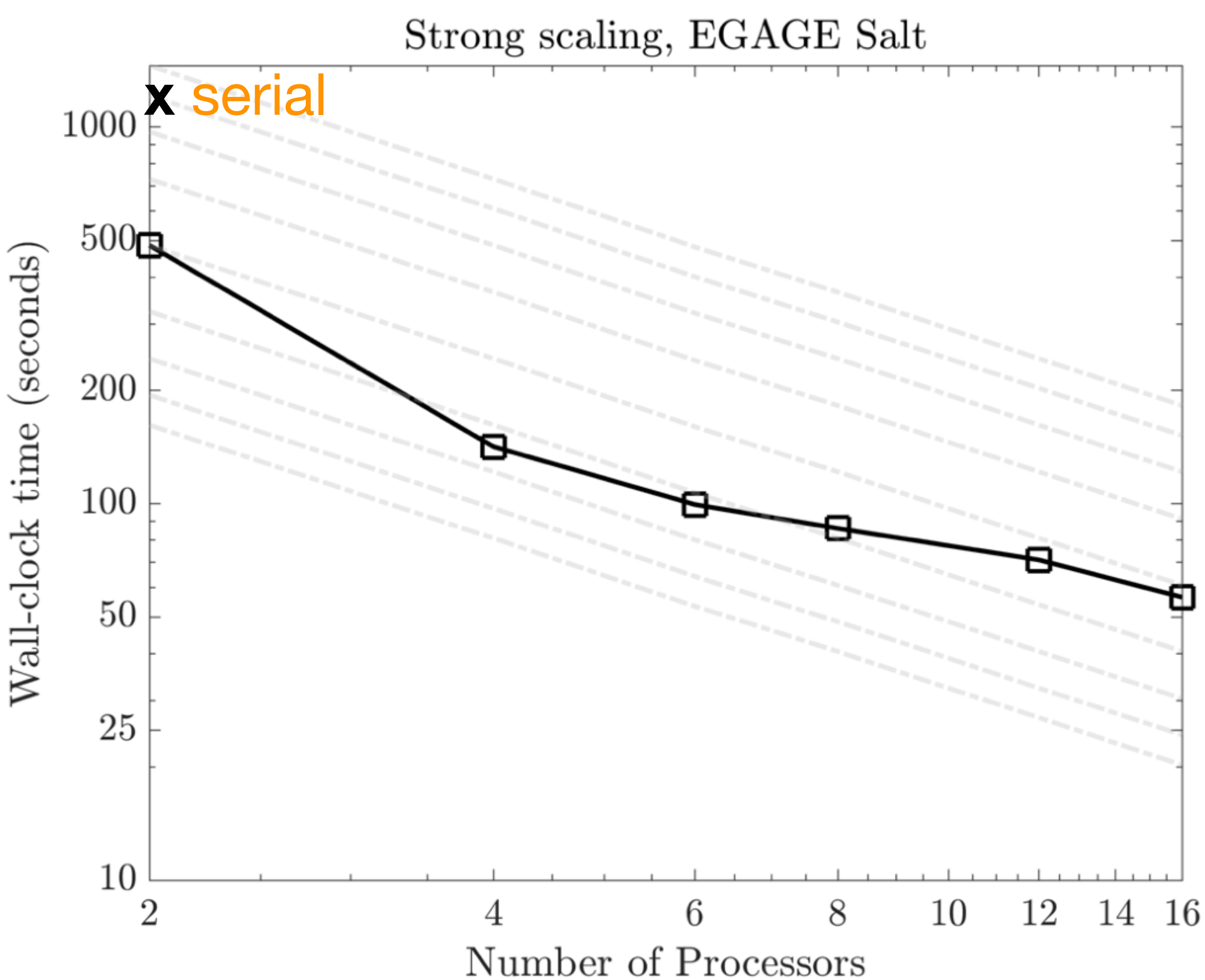


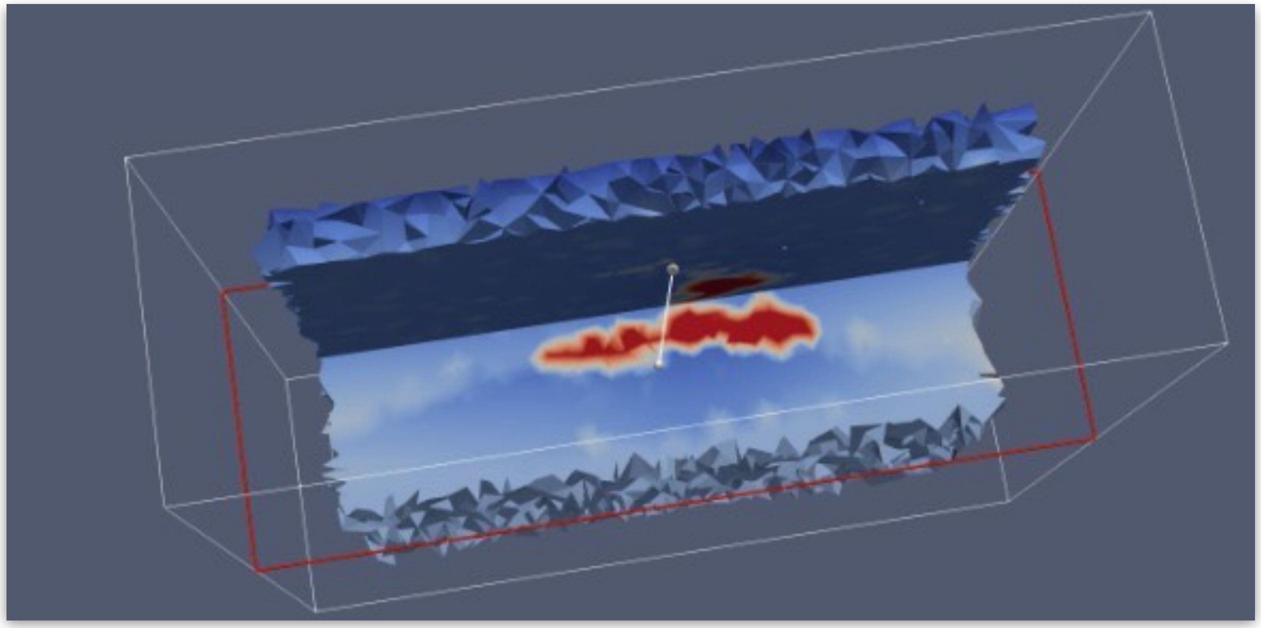




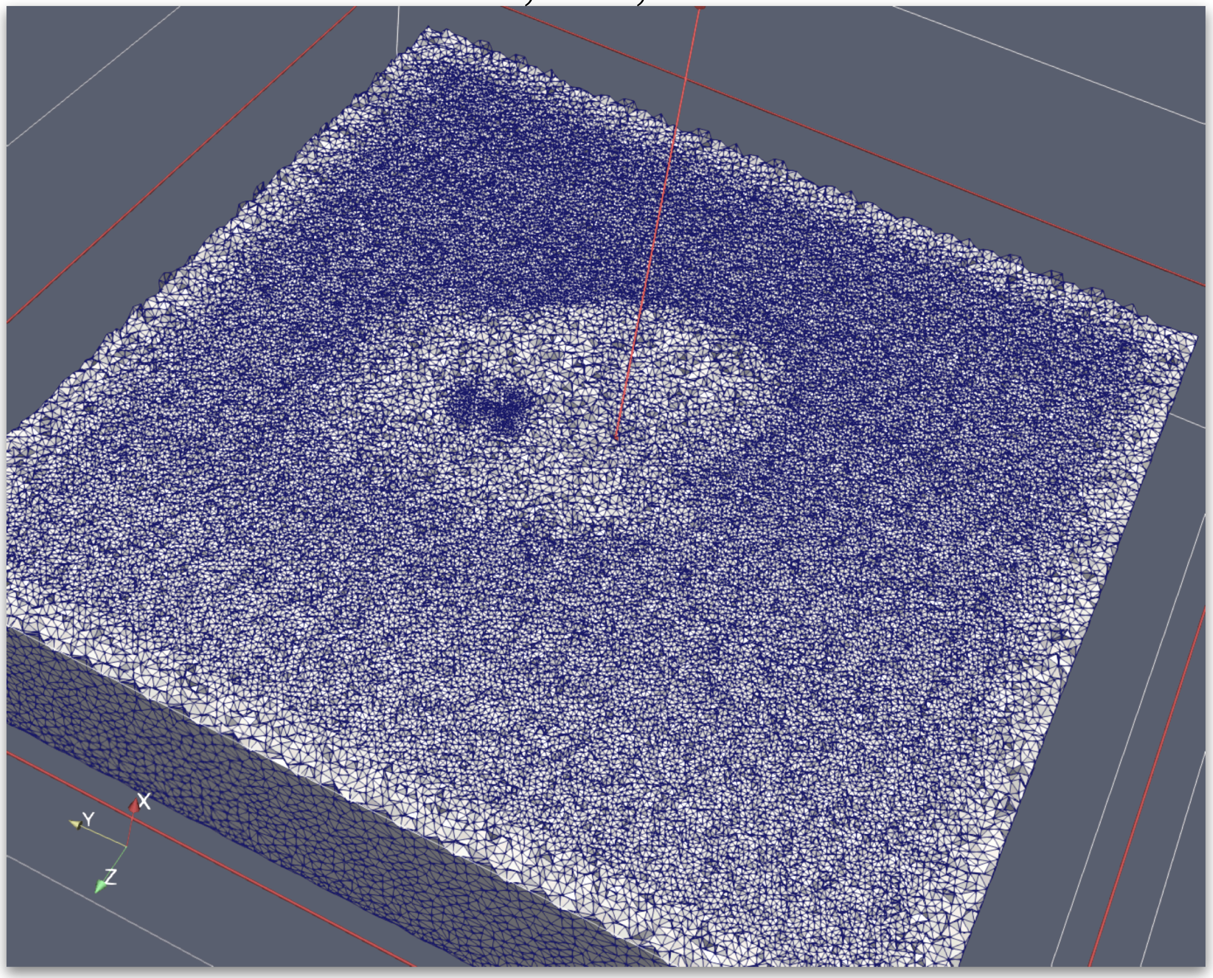
Performance

- 3D parallel mesh generation
  - Load balancing has not yet been considered.
  - 50 meshing iterations





$N \approx 37{,}000{,}000$ *cells*



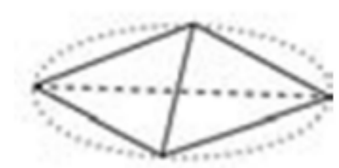
Robustness

- All Delaunay-based methods suffer from degenerate flat elements called *slivers*
  - Implemented a method to remove the *slivers* in parallel following [3].
  - Sizing distribution is preserved since only *slivers* are incrementally moved.

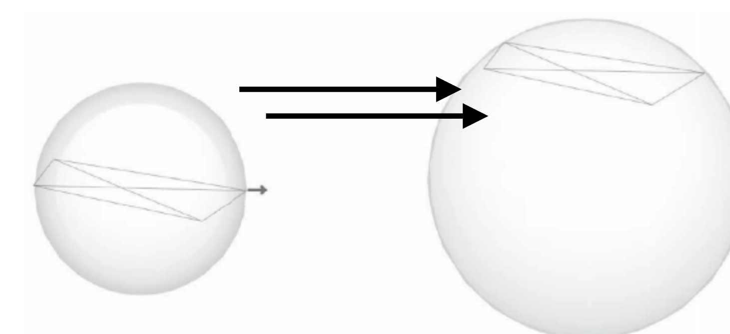
Algorithm 4: Steps to remove *slivers* from a 3D tetrahedral mesh.

Result: Sliver removal.
Given p vertices, a SDF defining the domain, and a threshold d degrees for the min. dihedral angle.;
$k \leftarrow 0$;
while *slivers exist* **do**
  1. Compute Delaunay triangulation of p^k. ;
  2. Calculate dihedral angles of tetrahedral. ;
  3. slivers $\leftarrow$ tetrahedral with dihedral angles less than d degrees.;
  for *each sliver tetrahedral i with vertices j for* $j = 0, 1, 2, 3$ **do**
    2. Calculate perturbation vector p_v of p_{i0}^k so that circumradius of the tetrahedral i increases the quickest.;
    $p_{ij}^{k+1} = p_{ij}^k + \alpha * p_v$;
  end
  5. Project any points outside the domain back inside;
  if *parallel* **then**
    6. Remove halo vertices added in 1.
  end
  k += 1;
end

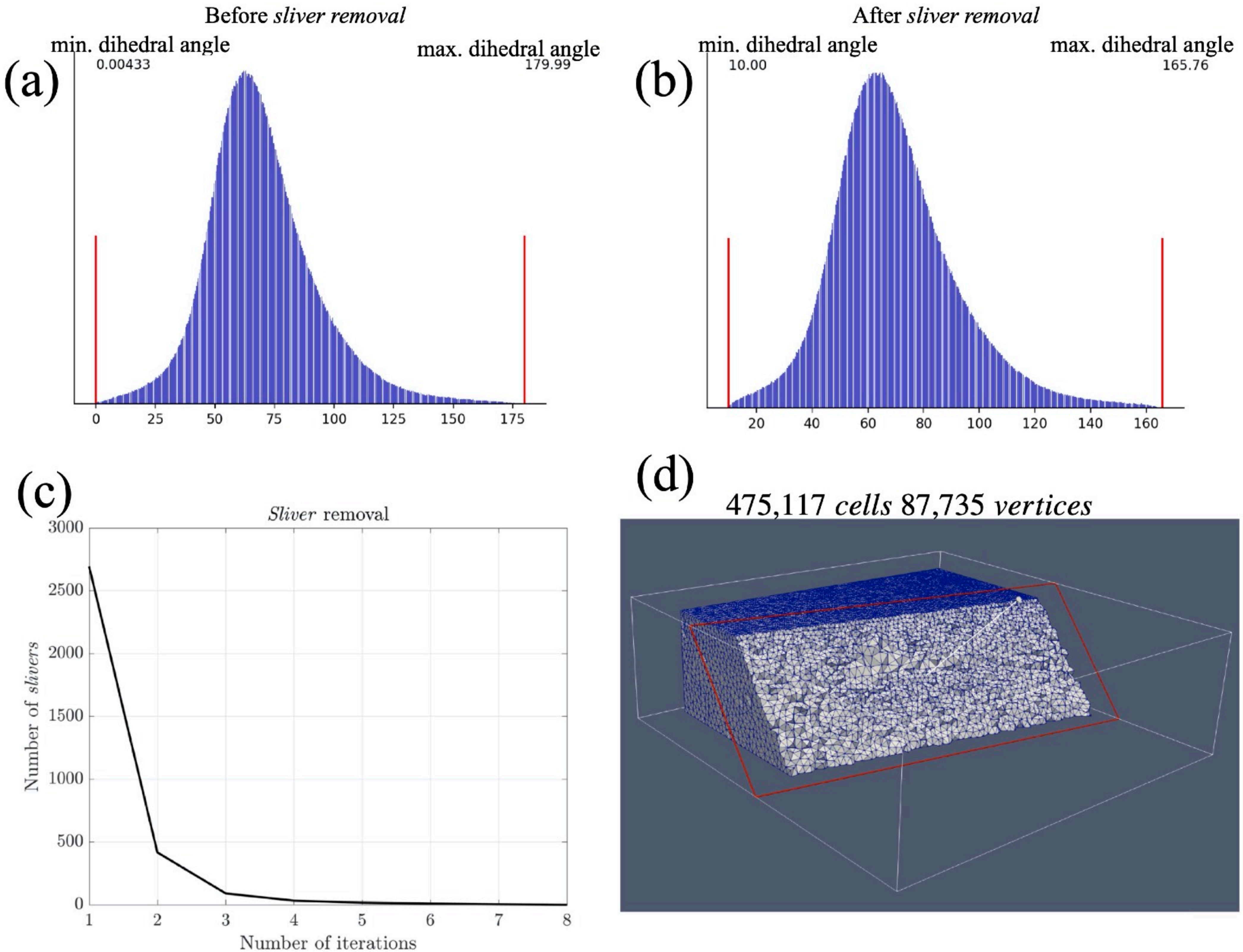
sliver element



vertex perturbation



Sliver removal



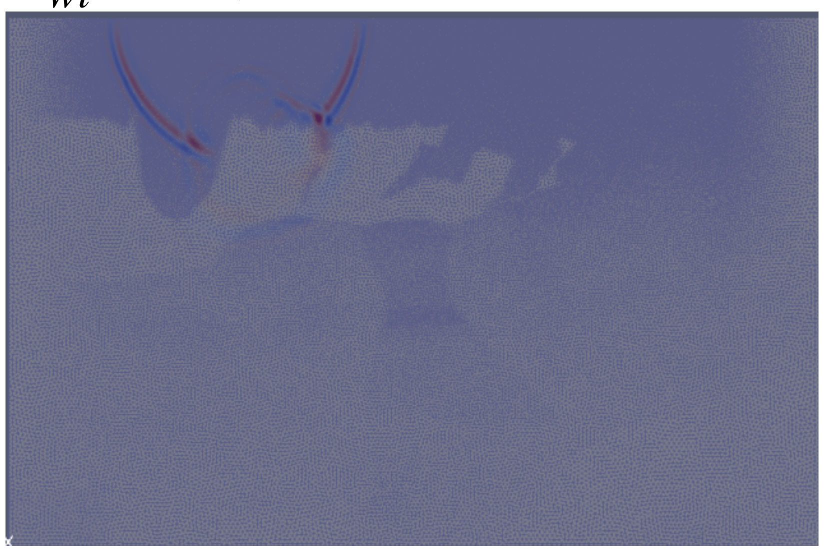
Ongoing applications

Universidade de São Paulo

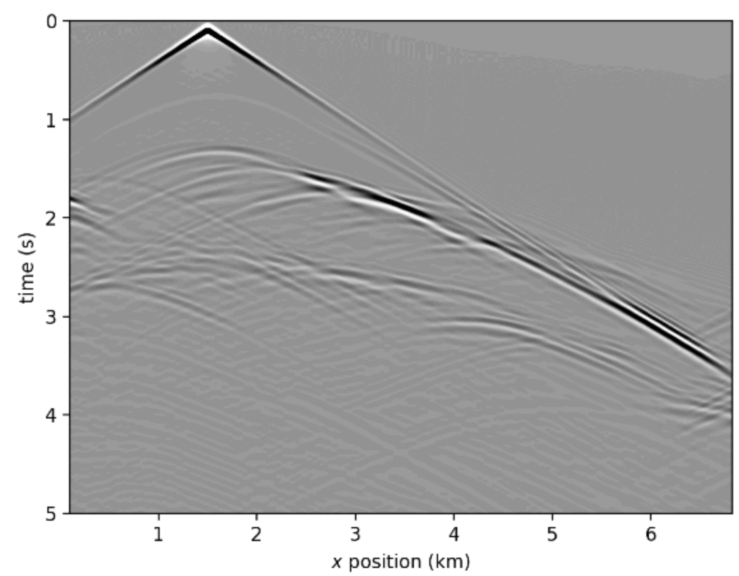
- Meshes are used in numerical wave propagators (acoustic and elastic).

$$acoustic, dt = 0.001s, T = 5s, f_{max} = 10Hz, 4procs\ , Intel, IPDG, explicit\ Newmark, w/PML$$

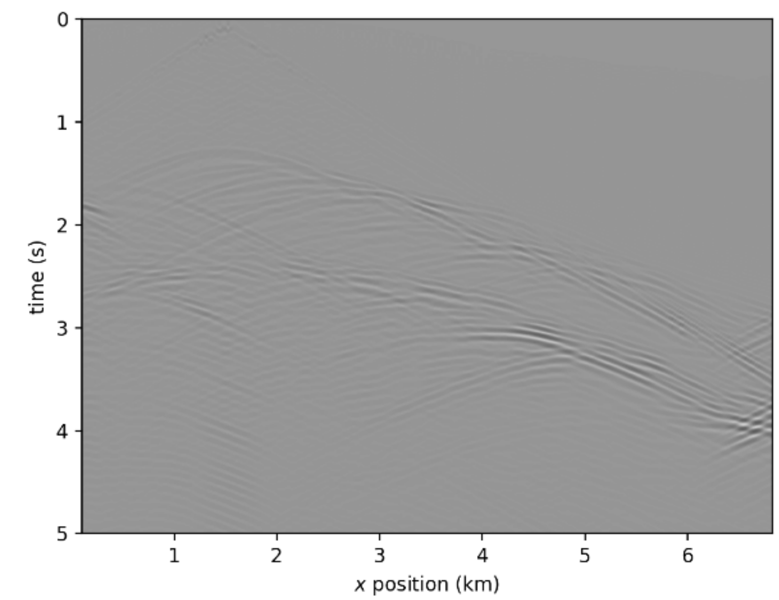
$N = 61{,}987\ vertices$
$\alpha_{wl} = 10,\ P = 1$



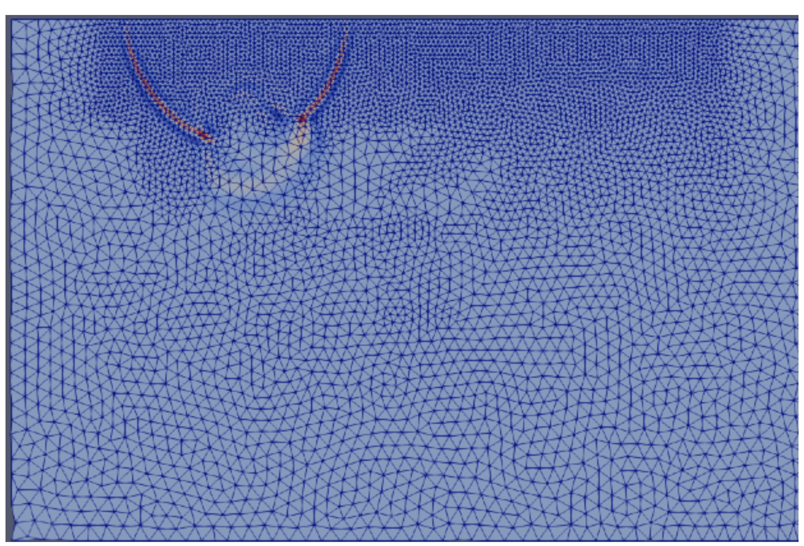
$time = 213.94s$



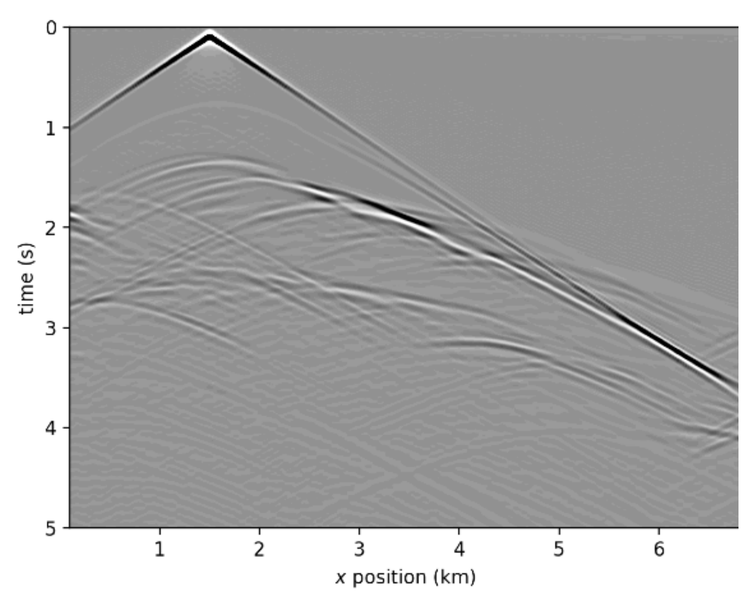
difference from uniform mesh



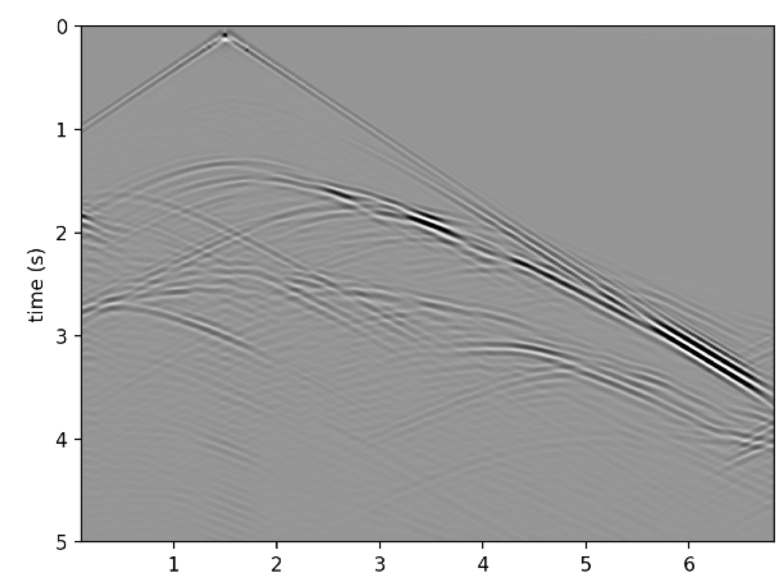
$N = 5{,}818\ vertices$
$\alpha_{wl} = 3,\ P = 3$



$time = 279.62s$



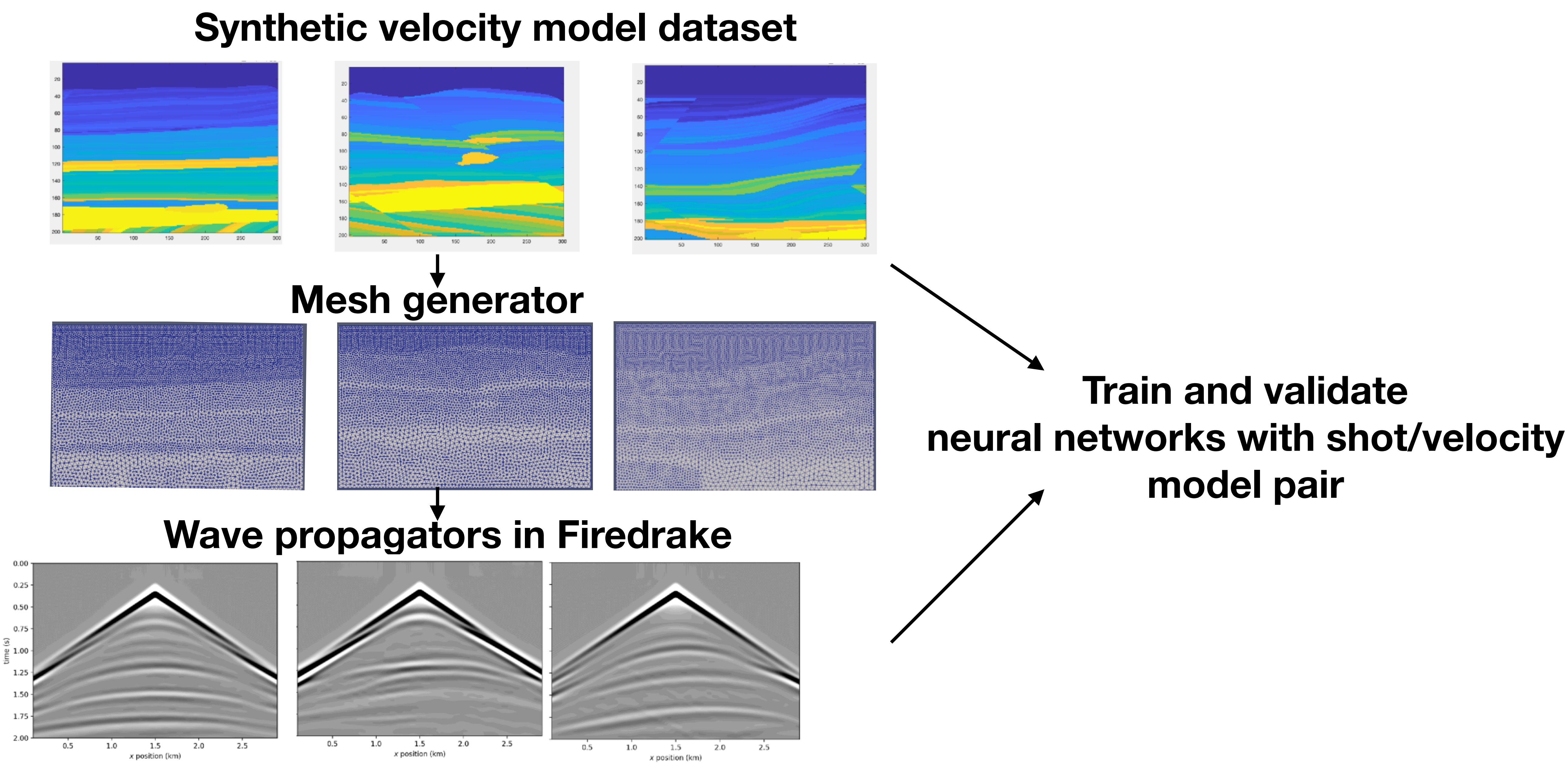
difference from uniform mesh



Ongoing applications

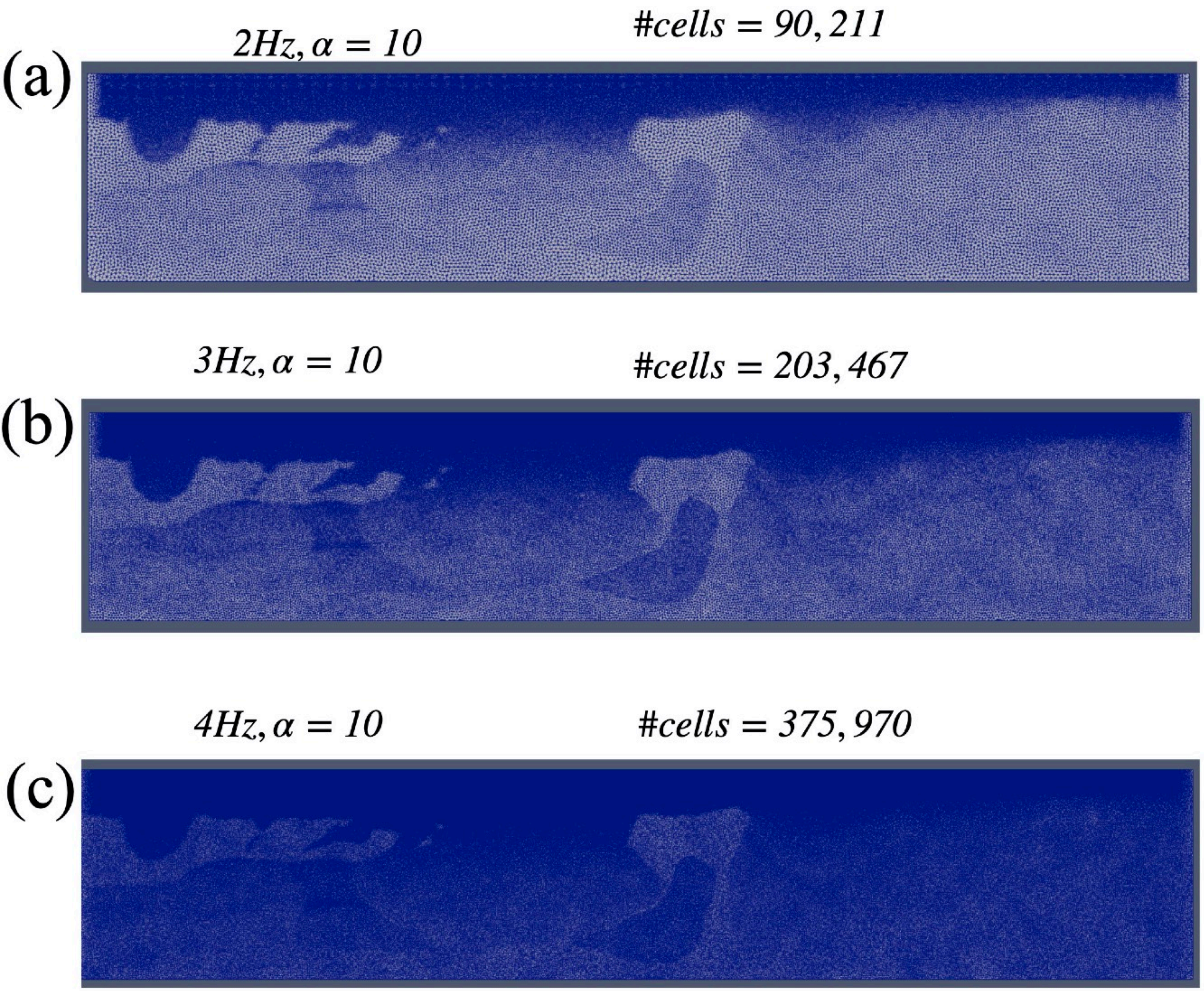
- Used in the training and validation of neural networks.

- Several thousand meshes are generated from synthetic velocity models.
  - The pair of synthetic velocity models, meshes, and seismograms are used to validate predictions of velocity models made by neural networks.
- Meshing workflows ensure consistent results.



Ongoing applications

- Performing “grid sequencing” or “continuation” in the time-domain for FWI.
- During FWI after each frequency band is complete, mesh is re-generated to a new source frequency given the previous model updates.



Algorithm 5: Multiscale time-domain full waveform inversion with mesh adaptation

```
Result: Optimized velocity model c(X) over a range of source frequencies freq.
c^0 ← initial velocity model;
k ← 0;
for freq ← freq_min to freq_max do
    Assign source frequency freq;
    Perform mesh adaptation for peak freq and c^{k+1};
    while ∇J > 0 & J > 0 or k ≤ (iter_max − 1) do
        for iter ← 0 to (iter_max − 1) do
            for shot ← 0 to (nshots − 1) do
                Compute forward simulations for shot;
                Calculate J_n at receivers;
                Compute local gradient ∇J_n via discrete adjoint;
            Sum J_n onto master from all shots;
            Sum ∇J_n onto master from all shots;
            if rank is master then
                Given ∇J and J using L-BFGS produce Δc^k;
                c^{k+1} += Δc^k;
            Broadcast c^{k+1} from master.
```

Thanks for listening!

Research Centre for Gas Innovation

USP Universidade de São Paulo

References

[1] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. Mcrae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Softw.*, 43(3):24:1–24:27, 2016. URL: http://arxiv.org/abs/1501.01809, arXiv:1501.01809, doi:10.1145/2998441.

[2] Per Olof Persson and Gilbert Strang. A Simple Mesh Generator in MATLAB. SIAM Review, 46:2004, 2004

[3] Tournois, Jane, Rahul Srinivasan, and Pierre Alliez. "Perturbing slivers in 3D Delaunay meshes." *Proceedings of the 18th international meshing roundtable*. Springer, Berlin, Heidelberg, 2009. 157-173.

[4] Peterka, Tom, Dmitriy Morozov, and Carolyn Phillips. "High-performance computation of distributed-memory parallel 3D Voronoi and Delaunay tessellation." *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014.

[5] Persson, P. Mesh size functions for implicit geometries and PDE-based gradient limiting. *Engineering with Computers* **22,** 95–109 (2006). https://doi.org/10.1007/s00366-006-0014-1

Software Architecture

Universidade de São Paulo