Triangular mesh generation for seismology

Keith Roberts, Rafael Gioria

June 2020

Abstract

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. An open-source Python software package called SeismicMesh has been developed to this end. A focus is placed on parallel unstructured mesh generation and adaptation capabilities for numerical simulations using the finite element method. The capabilities of the software are exposed to the user via a simple application program interfaces (API) written in Python. The software can be scaled up on distributed memory clusters. The technology has been used to generate meshes in 2D and 3D acoustic and elastic wave propagators written in Firedrake's Uniformed Form Language, which are used in Full Waveform Inversion algorithms.

1 Software architecture

The software is implemented in a mixed language environment (Python and C++). The Python language is used for the API while computationally expensive operations are performed in C++. The two languages are linked together with *Pybind11*. The Computational Geometry Algorithms Library [1] is used to perform geometric operations that use floating point arithmetic to avoid numerical precision issues. Besides this, we use several very common Python packages: *Numpy*, *Scipy*, *MeshIO*, *SegyIO*, and *MPI4py*. In particular, *MeshIO* enables to the program to output meshes in dozens of commonly used different formats.

A typical workflow consists of two stages: the development of a mesh size map and a mesh generation step with the developed sizing map. Both steps are script-able and can be written in less than 10 lines of code. The program creates this sizing map with a set of options that are defined by the user. Three examples are provided on the public Github using commonly encountered seismic velocity models: Marmousi II, BP 2004, and the EAGE Salt to demonstrate how to configure your own scripts and how to use the software.

2 Inputs

The only required input to generate a mesh is a binary file (e.g., SEG-y) containing the P-wave velocities of the domain. The software makes the assumption the domain can be approximated by a rectangle/cube. Thus, the user specifies the domain geometry as a tuple of coordinates in meters: the bottom front corner and the top back corner. The program automatically generates the rectangle/cube domain geometry used during meshing. Interior structures are meshed using signed distance functions.

3 Triangle sizing map

Given a coordinate in $\mathbb{R}^n$ where $n = 2, 3$, the sizing map returns the desired mesh size h. The mesh sizing capability provides a method to draft new meshes in a consistent and repeatable manner. The sizing map is built on a Cartesian grid, which simplifies implementation details especially in regard to distributed memory parallelism. Furthermore, seismic velocity models are available on structured grids and thus the same grid can be used to build the sizing map on.

The notion of an adequate mesh size is determined by a combination of the physics of acoustic/elastic wave propagation, the desired numerical accuracy of the solution (e.g., polynomial order, timestepping method, etc.), and the computational cost of the model. In the following sub-sections, each mesh adaptation strategy is described briefly.

3.0.1 Wavelength-to-gridscale

The highest frequency of the source wavelet f_{max} and the smallest value of the velocity model v_{min} define the shortest scale length of the problem since the shortest spatial wavelength λ_{min} is equal to the ratio v_{min}/f_{max}. For marine domains, v_{min} is approximately $1,484$ m/s, which is the speed of sound in seawater, thus the finest mesh resolution is near the water layer. The user is able to specify the number of vertices per wavelength α_{wl} and the peak source frequency f_{max}. This sizing heuristic also be used to take into account varying polynomial orders for finite elements.

3.0.2 Courant-Friedrichs Lewey (CFL) condition

To ensure numerical stability for simulation, a conservative CFL condition is enforced. For the 2D linear acoustic wave equation, the CFL condition is described by Eq. 1.

$$C_r = \frac{(\Delta t * v_p)}{h} \tag{1}$$

where h is the diameter of the circumball that inscribes the element. We rearrange 1 to find the minimum mesh size possible for a given v_p and Δt, based on some user-defined value of $Cr \leq 1$. If there are any violations of the CFL, they are edited to satisfy that the maximum Cr is less than some conservative threshold. We normally apply $Cr = 0.5$, which provides a solid buffer but this can but this can be controlled by the user.

3.0.3 Resolving seismic P-wave velocity gradients

Finer mesh resolution is deployed inversely proportional to the local standard deviation of P-wave velocity. The local standard deviation is calculated in a sliding window around each point on the velocity model. The user chooses the mapping relationship between the local standard deviation of the velocity model and the values of the corresponding mesh size nearby.

3.0.4 Mesh size gradation

Sharp seismic velocity contrasts can lead to variations in mesh size that can become substantially large. The construction of a mesh with relatively quick spatial variations in mesh sizes leads to low geometric quality meshes that can compromise accuracy. Thus, a mesh size smoothness limit is enforced such that the local increase in mesh size is bounded above by a threshold. We adopt the method to smooth the mesh size function originally proposed by [2] that works in both 2D and 3D. An example of applying the gradient limiter in 3D is depicted in Fig. 2. The user chooses the mesh size gradient threshold.

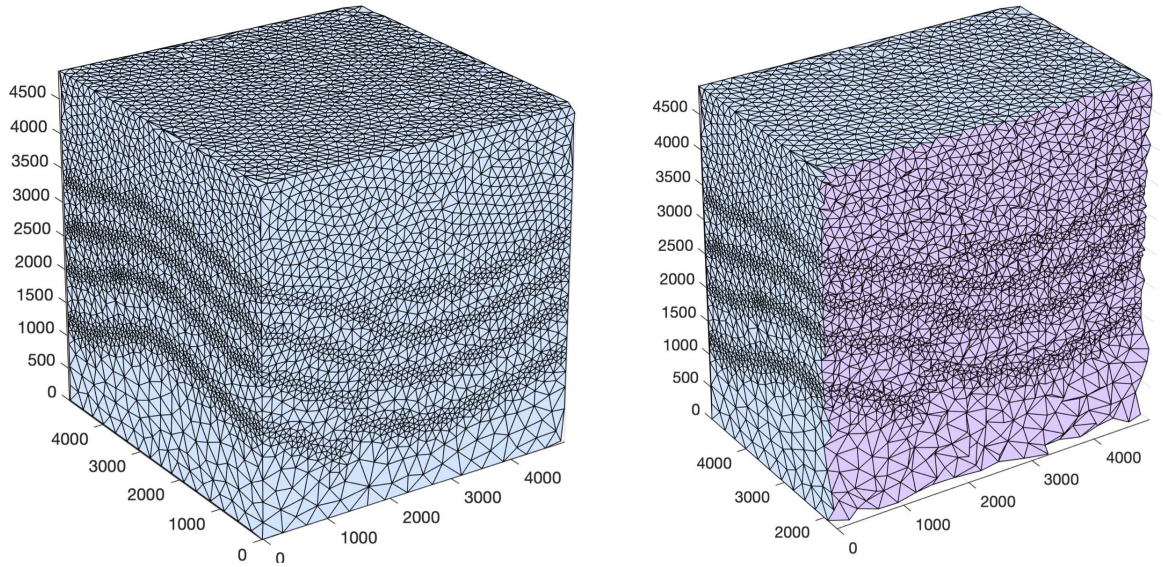

Figure 1: Resolving seismic P-wave velocity gradients. Note that the meshes depicted here have a minimum element size of 10-m and a $g < 0.15$, which acts to smooth mesh sizes nearby sharp material transitions.

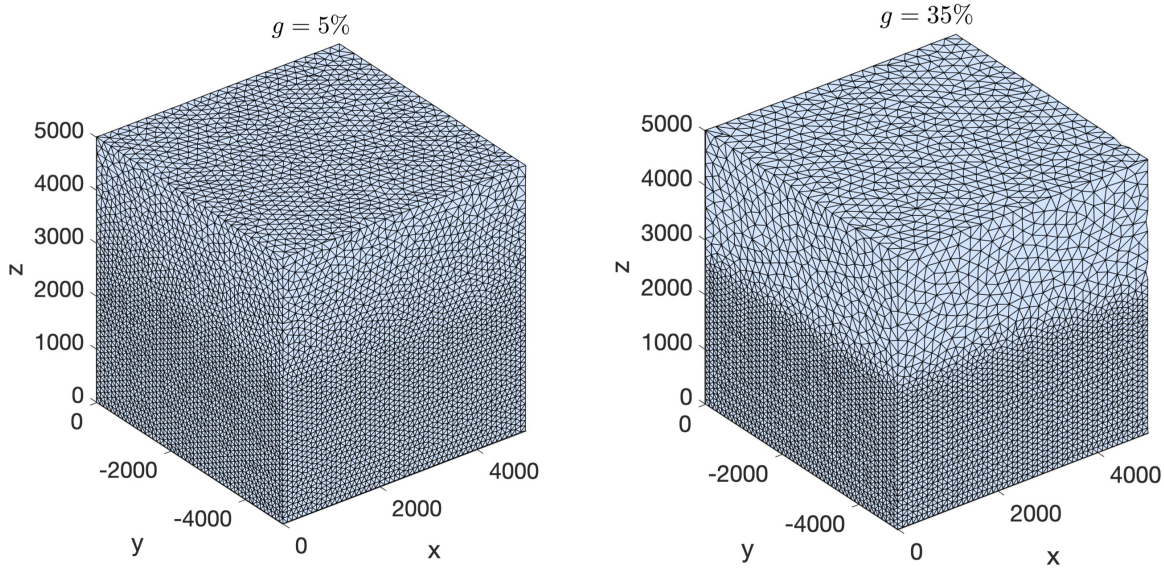

Figure 2: Examples of meshes graded differently by bounding above the gradation rate in the sizing function and then constructing a mesh with this sizing function.

3.0.5 Meshing for absorbing boundary conditions

The user has the option to specify a domain extension and a method to size elements in this region of the domain. Currently, there are three different styles of padding/domain extension supported, which follow along with the jargon laid forth in the *numpy.pad* function. For example, inside the domain extension layer, mesh sizes can either increase or decrease linearly, be reflected about the domain boundary (sans the extension), or be set to a constant mesh size.

3.0.6 Input file generation

The program writes several input files relevant for the numerical wave propagators that we have written in the Firedrake UFL language. For instance, the seimismic velocity model required to build the mesh is written to a HDF5 binary file, which is later projected onto the mesh in our FEM codes during execution. The sizing options that were used to build the mesh (i.e., a Python dictionadry) are also saved as a Python pickle. This pickle can be used to re-build meshes so to ensure all possible options are the same as the previous meshing instance.

4 Mesh generator

Multiple options are supported for high-quality mesh generation. Currently, there is support for a 3D Octree-splitting approach (*yamg-geo*) based on Dr. Gorman's work and a 2D/3D Delaunay-based mesh generation approach (*DistMesh*). In recent months, the majority of technical developments have occurred using the Delaunay mesh generation approach that uses signed distance functions to define the domain and features within the domain.

4.1 *DistMesh*

Briefly, the *DistMesh* algorithm is a iterative spring-based mesh generation method that moves an initial distribution of points according to a forcing function (Algorithm 1) [3]. The movement of points is calculated by a function that compares measured edge lengths to the desired edge lengths as specified in the sizing map. The approach often generates meshes with remarkably high element quality surpassing *tetgen* and *gmsh*. Two downsides of the *DistMesh* approach however are: 1) each meshing iteration requires a re-triangulation and 2) existing implementations are serial. However, I have successfully parallelized this *DistMesh* algorithm and detail these developments below.

In the *DistMesh* algorithm, the problem's domain is defined with an implicit function such as a signed distance function (SDF). A SDF defines the extent of the problem's domain as the 0-level set of the function and this technique is commonly employed in computer vision. Considering the domain in marine hydrocarbon exploration can be well represented by a cube or rectangle, the SDF can be efficiently defined in 2D (and 3D as well but not depicted) through Eq. 2.

$$d = -\min(\min(\min(-y_1 + Y, y_2 - Y), -x_1 + X), x_2 - X) \tag{2}$$

where x_1, x_2, y_1, and y_2 are the x- and y- coordinates of the corners of the rectangle and X and Y are the coordinates of the query point in $\mathbb{R}^2$. Note that many primitive shapes such as circles, spheres, cones, conics, and others can be used in combination through set operations (e.g., union, difference, intersection, etc.) to create more irregular structures.

Work with WS1 demonstrates the possibility to represent salt-geometries using SDFs although this is not yet shown here. It may be possible to utilize the SDF information from WS1's FWI algorithm to automatically construct meshes of salt bodies. Further, as their FWI process updates

the structure of the salt body, new meshes could be automatically generated.

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.;
    4. Move points based on forcing function;
    5. Project any points outside the domain back inside;
    if *parallel* **then**
        6. Remove halo vertices added in 2.
    end
end

4.2 Parallel mesh generation

High quality mesh generation for realistic geophysical domains can become a computational expensive task and mesh adaptation capabilities need to occur relatively quickly in order to avoid slowing the overall application down. Further, the serial computation of the Delaunay triangulation in the *DistMesh* algorithm (Algorithm 1) becomes the main computational bottleneck. These aspects motivate the development of parallel mesh generation algorithms.

4.2.1 Domain decomposition

For parallel execution, the domain is decomposed into axis-aligned blocks (Fig. 3). Each block is owned by one rank (in a distributed memory computer) and thus has at most two neighboring blocks. The block ownership is such that rank N borders the blocks owned by rank $N-1$ and rank $N+1$, if applicable. Note that blocks are decomposed such that there exists a small region of overlap between neighboring blocks. Through this approach, communication operations are simplified but at the cost of sub-optimal load balancing since this approach does not consider spatial point density.

4.2.2 Description of algorithm

Following the method and proof proposed by [4], the Delaunay property of the triangulation leads to an elegant and efficient way to parallelize Delaunay triangulation (Algorithm 2). All other steps of the Algorithm 1 are embarrassingly parallel because the Delaunay triangulation's connectivity is identical across the block's boundary. Thus the forcing function that is used to incrementally move points produces the same movement of points across block boundaries each meshing iteration.

Algorithm 2: Steps to compute a Delaunay triangulation in parallel.

Result: A Delaunay triangulation.

1. Compute initial local triangulation.;
2. Exchange initial neighbor points.;
3. Compute augmented local tessellation with points from 2.;
4: Remove faces/facets that have no vertices inside the local block.;

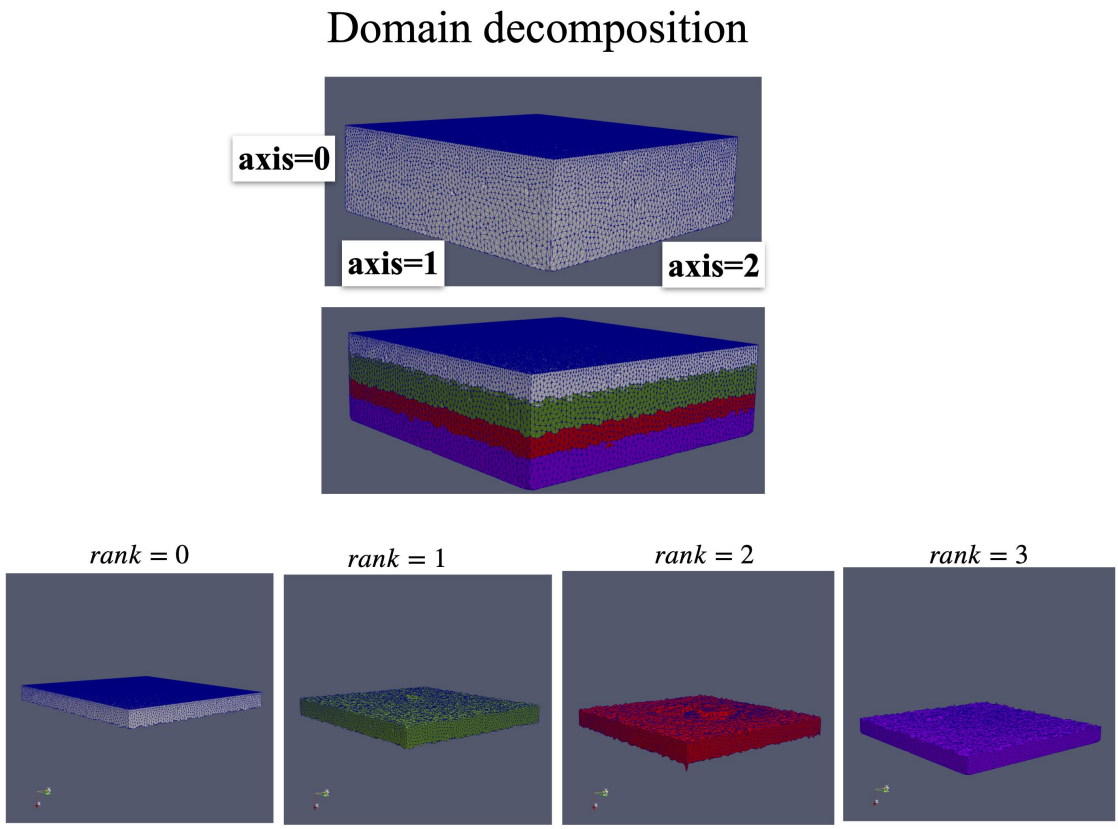

Figure 3: The domain decomposition strategy. The rank onwership matches the color of the subdomain and, in this figure, the axis= 0 is split. Axis= 1 or axis= 2 could also have been split to form the decomposition but these are not shown.

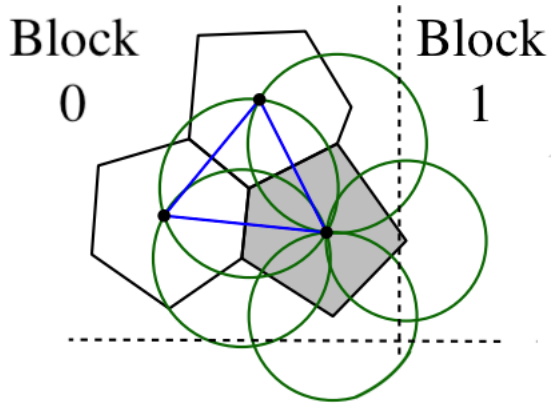

Figure 4: An illustration of the neighborhood communication idea (Algorithm 3). Block boundaries are indicated with the dashed-black lines. The point in the center of the gray Voronoi cell will be communicated to the neighboring block (block 1) because the Voronoi vertex has a circumsphere that intersects with a neighboring block. This figure was adapted from [4].

Algorithm 3: Steps to communicate points to form a valid Delaunay triangulation in parallel.

Result: Neighbor point exchange.
1: Enqueue points that are near to block boundaries to neighbor blocks within circumsphere radii.;
2: Exchange enqueued neighbor points.;
3: Compute new local tessellation;

The steps are laid out in Algorithm 2. After decomposing an initial set of points into blocks (c.f., Section 4.2.1), each local block is triangulated independently. If a vertex is connected to a triangle that has a circumsphere intersecting with a neighboring block, a copy of this point is sent to the block it intersects with. These vertices are termed *halo vertices* on the processor that receives them. Due to the decomposition topology, this implies the need to check if a set of circumspheres intersects with, at most, two axis-aligned boxes.

The algorithm only requires to determine which points need to be sent to either of the two neighboring blocks. Thus, only a single communication step that exchanges a small amount of point coordinates is required. Once the local blocks are augmented with the points communicated from neighboring blocks, a new local triangulation is computed using an incremental Delaunay algorithm such as the Bowyer-Watson algorithm. In this updated local triangulation with the halo vertices, the connectivity becomes identical across block boundaries. The mathematical proof for the validity of this algorithm is detailed in [4]. One critical point to note is that after the local block is augmented and re-triangulated, any triangle with all of its vertices outside of the local block is removed from the triangulation.

4.2.3 Performance

The wall-clock time to build a 3D mesh of the EGAGE Salt domain is highlighted in Fig. 5 using the algorithms detailed above. Overall, good performance was obtained when scaling the problem up as a reduction in execution time was measured as more processors were used. In the current implementation, the aspect of load balancing is not considered, which is one explanation for the deviance from the optimal linear scaling curves. For example, the interior of the EGAGE Salt domain features a large salt-body with relatively higher P-wave seismic velocities. This leads to a large variation in mesh sizes in the horizontal direction of the domain. Consequently, some domains have a largely unequal amount of vertices that varies as a function of number of processors used, and this affects parallel performance negatively.

4.3 Degenerate *sliver* removal

3D mesh generation is significantly more challenging than 2D due to a) the increased computational cost related to the additional spatial dimension and, for Delaunay approaches b) the presence of degenerate elements called *slivers* whose existence would otherwise ruin a high geometric quality mesh. If any sliver exists in a 3D mesh, the FEM solution will become numerically unusable. Fortunately, this problem does not occur in 2D. To tackle this problem, I implemented a method similar to that of [5] aimed at removing *slivers* while preserving the triangulation and sizing distribution.

The technique used in this program fits within the *DistMesh* framework. Like the mesh generation approach, the algorithm operates iteratively. However, in this approach, it perturbs only vertices associated with *slivers* so that the circumsphere's radius of the sliver tetrahedral increases rapidly (e.g., gradient ascent of the circumsphere radius) (Algorithm 4). Further, it operates on an existing mesh that already has a high-geometry mesh quality. The perturbation of a vertex of the *sliver* leads to a local combinational change in the nearby mesh connectivity to maintain Delaunayhood, thus destroying the *sliver* in lieu of elements with larger dihedral angles.

Note here, we define *sliver* elements by their dihedral angle (i.e., angle between two surfaces) of which a tetrahedral has 6. Generally, if a 3D mesh has a minimum dihedral angle less than 1 degree, it will not be numerically stable to simulate with.

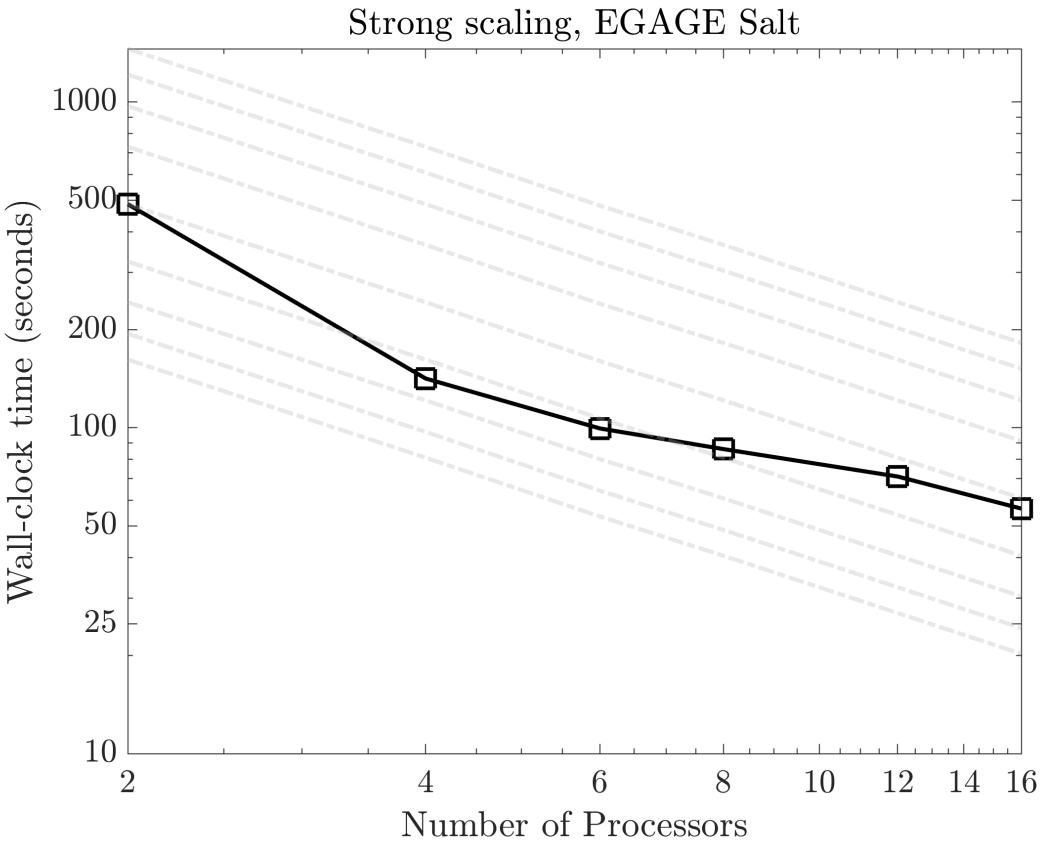

Figure 5: Strong scaling curve illustrating the wall-clock time in seconds spent generating a 3D mesh of the EGAGE Salt model adapted for a source wavelet with a peak frequency of 2 Hz using 5 elements per wavelength with several processor configurations. The domain was decomposed along the 1st axis (c.f., Fig. 3). The faint grey lines serve as a reference for optimal linear scaling curves. Experiments were ran on the Mintrop cluster using the mpich implementation of message-passing and the gcc 8.3.0 compiler.

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^k_{i0} so that circumradius of the tetrahedral i increases the quickest.;
    $p^{k+1}_{ij} = p^k_{ij} + \alpha * p_v$;
  end
  5. Project any points outside the domain back inside;
  if *parallel* **then**
    6. Remove halo vertices added in 1.
  end
  k += 1;
end

The *sliver* removal technique combined with the enforcement of the domain boundaries through SDFs and the distributed memory parallelism enable efficient and effective removal of all *slivers* in only a handful of iterations (Fig. 6). In practice, the software can achieve tetrahedral meshes that have a minimum dihedral angle approximately around 10 degrees, while requiring little additional computational cost. Note that this algorithm exhibits faster convergence rates when more meshing iterations are performed using the Algorithm 1. In fact, the best performance was observed when around 50 meshing iterations were used prior to *sliver* removal.

5 Mesh generation inside FWI

Time domain FWI is often started with a relatively low source frequency (approx. 2 to 3 Hz) to avoid cycle-skipping. The lower source frequency permits the use of a coarser mesh resolution with larger elements and overall fewer degrees-of-freedom. Once the lower frequency background components of the velocity model have been imaged, higher frequency sources can then be used that consequently require more finely resolved meshes. While it is possible to generate a hierarchy of meshes adapted to different source frequencies before starting FWI, these meshes would not be adapted to the model updates that arise during FWI.

Thus, we are incorporating the mesh generation step inside the FWI loop at the start of each frequency band (Algorithm 5). In this way, we avoid the complexity of mesh adaptation algorithms that would have to occur while the process is timestepping while benefiting from the unstructured nature of the mesh connectivity to accelerate the FWI process. For instance, in Fig. 7 there is $4x$ difference in number of cells between a mesh adapted to a 2 Hz wavelet and a 4 Hz wavelet. In 3D, the growth of the computational problem as the source frequency is increased becomes significantly more severe than $4x$ increase for a doubling of source frequency.

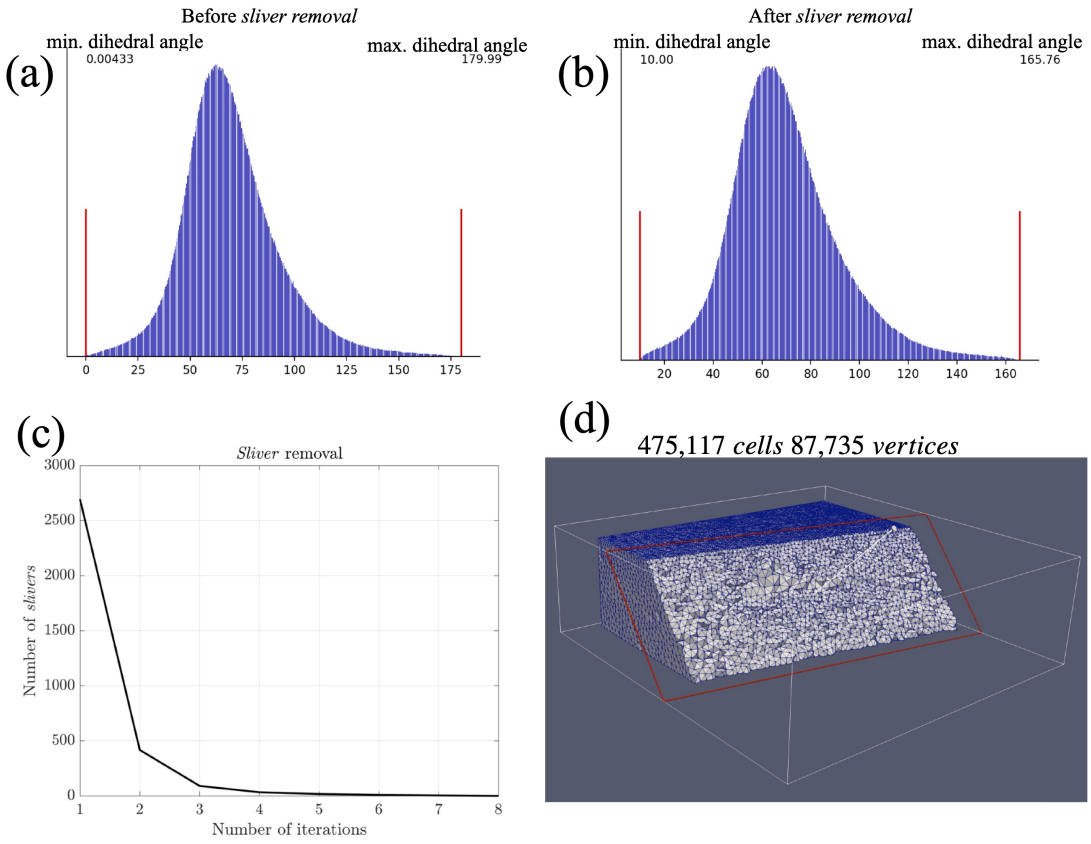

Figure 6: Distribution of dihedral angles in a 3D mesh (475, 117 cells and 87, 735 vertices) of the EGAGE Salt domain before (a) and (b) after the application of *sliver removal*. Note 40 iterations of the *DistMesh* algorithm were performed prior to the application of the *sliver removal*. Panel (c) shows the number of slivers (i.e., tetrahedrals with min. dihedral angles ≤ 10 degrees) against the number iterations of Algorithm 4. Panel (d) shows a slice through the final mesh around the salt-body.

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

Result: Optimized velocity model $c(X)$ over a range of source frequencies $freq$.
$c^0 \leftarrow$ *initial velocity model*;
$k \leftarrow 0$;
for $freq \leftarrow freq_{min}$ **to** $freq_{max}$ **do**
  Assign source frequency $freq$;
  Perform mesh adaptation for peak $freq$ and c^{k+1};
  while $\nabla J > 0$ *&* $J > 0$ *or* $k \leq (iter_{max} - 1)$ **do**
    for $iter \leftarrow 0$ **to** $(iter_{max} - 1)$ **do**
      for $shot \leftarrow 0$ **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.

6 Discussion

Significant progress has been made in terms of automatically developing meshes for seismic velocity imaging using the finite element method from P-wave velocity models. An open source package for 2D and 3D end-to-end mesh generation has been created called *SeismicMesh* https://github.com/krober10nd/SeismicMesh. The approach to mesh generation and some algorithms were briefly detailed in this work. In this approach, the user creates a simple Python script to script the development of the mesh without any manual intervention or geometry creation.

Three aspects of ongoing work with this mesh generation technology that we hope to report on soon are:

1. Meshing salt bodies using signed distance functions obtained via WS1's FWI algorithm.
2. Multiscale time-domain Full Waveform Inversion with mesh generation adapting to each source frequency band (5).
   - Performance speed-up?
   - Accuracy improvement (enhanced convergence rates?)?.
3. Incorporating sea-floor bathymetry data into the mesh generation procedure through signed distance functions.
   - Comparing seismograms.

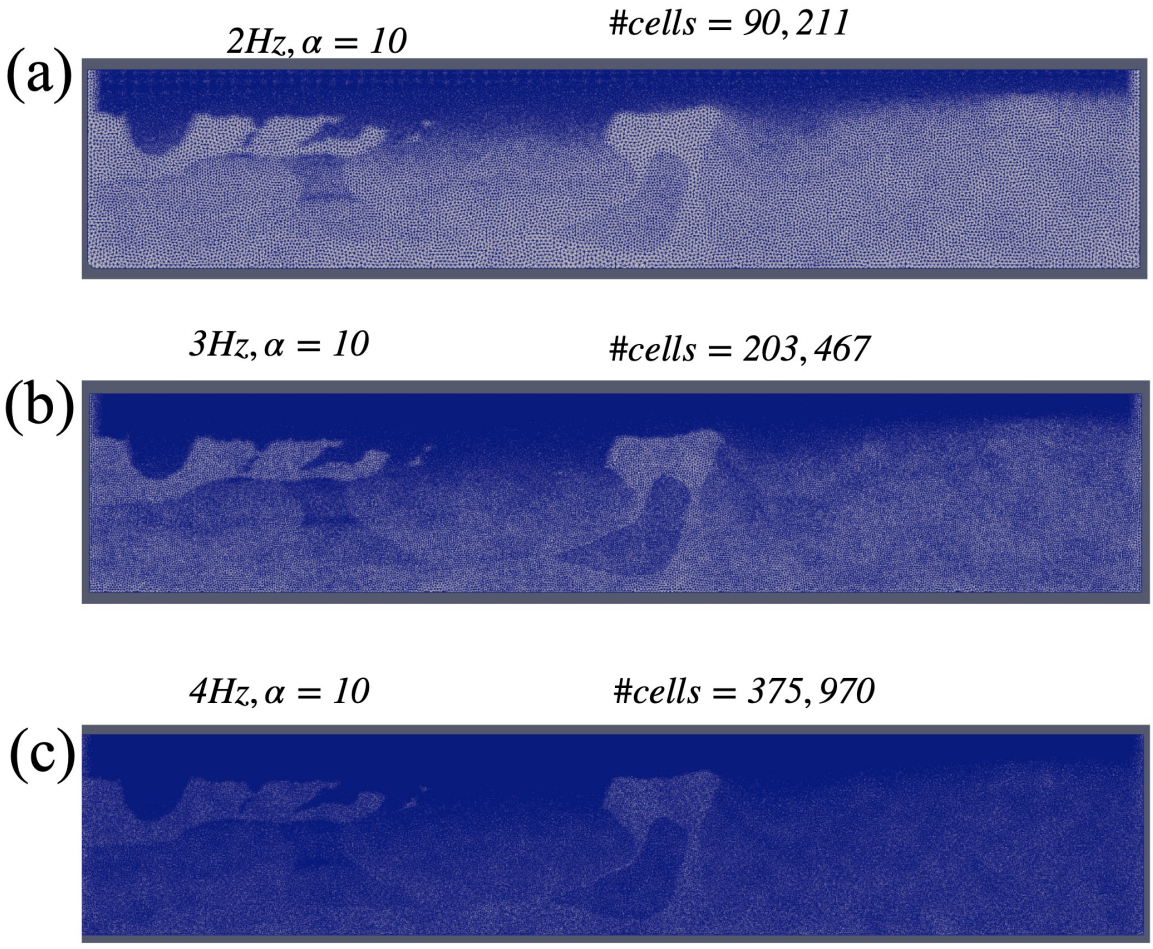

Figure 7: A sequence of meshes adapted to the BP 2004 benchmark P-wave seismic velocity model assuming different peak source frequencies are used. (a) Mesh connectivity adapted for a $2Hz$ source, (b) for a $3Hz$ source, and (c) for a $4Hz$ source. All meshes adapt to the wavelength-to-grid scale (c.f., 3) with $\alpha = 10$ vertices per wavelength. The number of cells is indicated in each title.

References

[1] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020.

[2] Per Olof Persson. Mesh size functions for implicit geometries and PDE-based gradient limiting. *Engineering with Computers*, 22(2):95–109, 2006.

[3] Per-olof Persson and Gilbert Strang. A Simple Mesh Generator in MATLAB. *SIAM Rev.*, 46:2004, 2004.

[4] T. Peterka, D. Morozov, and C. Phillips. High-performance computation of distributed-memory parallel 3d voronoi and delaunay tessellation. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 997–1007, 2014.

[5] Jane Tournois, Rahul Srinivasan, and Pierre Alliez. Perturbing slivers in 3d delaunay meshes. In Brett W. Clark, editor, *Proceedings of the 18th International Meshing Roundtable*, pages 157–173, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.