OceanMesh2D: User guide

Precise distance-based two-dimensional automated mesh generation toolbox intended for coastal ocean/shallow water flow models

Authors: Keith J. Roberts and William J. Pringle

University of Notre Dame, United States
Computational Hydraulics Lab

4 July 2019

Contents

1 Introduction

1.1 Foreword

OceanMesh2D is a set of MATLAB scripts to assemble and post-process two-dimensional (2D) triangular meshes used in finite element numerical simulations. It is designed with coastal ocean models in mind, although it can mesh any 2D region bounded by a polygon. It can be used to build meshes of varying size (up to 10-20 million vertices or so) based on user-defined parameters to edgelength functions that control how the resolution is distributed in space. The meshes created with the software are nearly reproducible since they are parameterizable and can be assembled quickly on a personal computer on the order of minutes to hours.

The mesh generation is accomplished by using documented Koko [2015] and original improvements to the seminal force equilibrium algorithm, *DistMesh* Persson and Strang [2004]. In this software, the mesh generator is a standalone class that only requires the specification of a polygonal region along with a target resolution (edgelength), much like *DistMesh*. However, in contrast to *DistMesh* the mesh generator class has been tuned to converge more quickly for complex polygonal regions and highly heterogeneous edgelength functions typically encountered in geophysical shallow water flow problems.

In automated mesh generation, the edgelength or mesh size function determines how resolution is distributed in space. While the topic of mesh generation is rich and well-studied, research on the impact of how mesh resolution effects the simulation of shallow-water physics has received much less focus. In this document, we describe some of the edgelength functions that were implemented to facilitate capturing shallow water flow both efficiently and accurately.

2 Things everyone should know

OceanMesh2D uses an object orientated programming (OOP) style to make the mesh generation process simpler through abstraction and overloading. These properties reduce the complexity of function calls and the number of lines the user needs to type. They also create consistent workflows that are reproducible.

There are four classes: `geodata`, `msh`, `meshgen`, and `edgefx` (Fig. 1). All the classes accept name/value pair arguments and are created by typing the class name (e.g., `geodata()`), which generates an object of the type class. All the methods of each class can be inspected by typing

```
methods('classname')/help methodname
```

although help documentation writing is an ongoing process. At the end of this user guide we elaborate more on some of these methods. There are no toolbox dependencies to build meshes, however, some select functions do require some toolboxes and these are mentioned when the method is described in the appendix. Here we list the basic requirements for this software:

- MATLAB (tested on versions post 2015a up to 2018b) n 64-bit Windows, Linux and Mac OS plus the following (free) toolboxes:

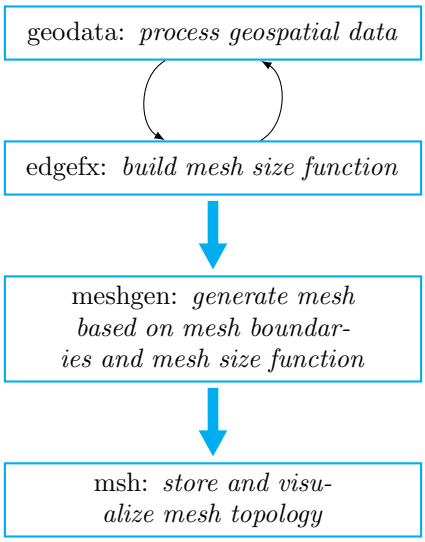


Figure 1: Standard workflow used to build meshes with OceanMesh2D.

  - free: m_map toolbox v1.4 (https://www.eoas.ubc.ca/~rich/map.html)

- optional: digital elevation model (DEM).
- optional: a shapefile representing the boundary of the area you want to mesh.

The m_map package must be placed within the working directory, we suggest the utilities directory.

2.1 Installation

The recommended method to install and use the software is through the version control software git.

If you feel like learning the hard way why git exists, the software can be installed by decompressing the zip archive file obtained from Github [1]. After unzipping the files, navigate to the 'OceanMesh2D/' or root directory. Download m_map and place it in the root directory. Create a directory called 'OceanMesh2D/datasets' and place your geospatial data (i.e., ESRI Shapfiles, NaN-delimited vectors, digital elevation models) within it. Note that we have already provided some freely available data from NOAA Bathymetric data viewer at [2] to run the examples included. Be sure to add the 'OceanMesh2D/utilities' and'OceanMesh2D/datasets' directories and **sub-directories** to your MATLAB path, e.g., 'addpath(genpath('datasets'/))', etc. To manipulate and manage the geospatial data necessary to run OceanMesh2D, it's recommended (but not necessary) to install the open-source package GDAL in order to easily exchange between file formats and extract ESRI Shapefiles and the like.

[1] https://github.com/CHLNDDEV/OceanMesh2D
[2] https://drive.google.com/open?id=1LeQJFKaVCM2K59pKO9jDcBO2yjTmJPmL

2.2 Mesh Properties

A mesh in our context is an unstructured triangulation composed of nt triangles and np vertices. More specifically, a mesh is a set of triangles $\boldsymbol{t}$ from tessellating np vertices $\boldsymbol{p}$ that lie in $\mathbb{R}^2$. In *OceanMesh2D* we create the well-known Delaunay triangulation of the point set $\boldsymbol{p}$ and then refine this triangulation iteratively until we reach a desired quality or exhaust a number of iterations. Each element or triangle of the mesh has three vertices making it C_1 continuous or a linear triangular element. A *valid* mesh for use with the Continuous Galkerin Finite Element framework must have the following properties:

1. $\boldsymbol{p}$ are arranged in an order (i.e., counter-clockwise) in each element.
2. There are no overlapping $\boldsymbol{t}$ (i.e., no elements intersect in space).
3. No disjoint or hanging $\boldsymbol{p_i}$ (i.e., vertices are always members of $\boldsymbol{t}$) and $\boldsymbol{p}$ are always shared between neighboring $\boldsymbol{t}$. The same holds for $\boldsymbol{t_i}$.
4. The boundary of the mesh must have only two traversal paths (i.e., one can travel from any starting point on it and move either clockwise or counter clockwise around the boundary and eventually reach the starting point).

However, a *valid* mesh is not necessarily a *high quality* one. We define a *high quality* mesh as a triangulation with the following properties:

1. All $\boldsymbol{t}$ are *nearly* equilateral
2. The edges of $\boldsymbol{t}$ do not vary more than g percent between their connected edges.

We use the following formula to quantify how equilateral the mesh is Bank [1998]:

$$q_E = 4\sqrt{3}A_E \left(\sum_{i=1}^{3} (\lambda_E^2)_i \right)^{-1} \tag{1}$$

where A_E is the area of the element and $(\lambda_E)_i$ is the length of the i^{th} edge of the element. $q_E = 1$ corresponds to an equilateral triangular element and $q_E = 0$ indicates a completely degenerated element. The consideration of what constitutes a *high quality* mesh rests on the statistical distribution of element quality, q_E. Generally a mesh with $\overline{q_E} - 3\sigma_{q_E} > 0.75$ (where the over-line and σ denote the mean and standard deviation respectively) is considered *high quality* and can be simulated without any changes to the mesh topology. Thus, when $\overline{q_E} - 3\sigma_{q_E} > 0.75$ is achieved, the mesh generation terminates and this generally occurs between 30-100 iterations in most ocean domains.
A restriction on the smoothness or the grade α_g of the triangulation is applied (Section 4.1). Sharp gradients in triangular resolution may produce simulation errors because often have poor geometry (e.g., high valency, large dihedral angles, etc.). Smooth transitions reduce these problems in the mesh.

An empty mesh for use within *OceanMesh2D* can be constructed by typing: `m = msh()` or, by populating it with various ADCIRC compliant files,
`m = msh('fname','type')` where type is either 13, 14, 15, or 24, which denote different ADCIRC input files (i.e., fort.13, fort.14, fort.15, fort.24, etc.) or some combination of them.

@msh class: example mesh

```
m =

                  msh with properties:
title: 'OceanMesh2D '← AGRID/run id
p: [4426948×2 double] ← vertices
t: [8182049×3 double] ← triangles
b: [4426948×1 double]← bathymetry
bd: [1×1 struct] ← no flux boundaries
op: [1×1 struct] ← elev. boundaries
bx: [] ← bathymetric slope in x-direction
by: [] ← bathymetric slope in y-direction
f11: [] ← fort.11 (baroclinic forcing) file
f13: []← fort.13 file
f15: [] ← fort.15 file
f19: [] ← fort.19 (non-periodic BCs.)
f24: [] ← Self-attraction and load (SAL) file
```

Figure 2: Example of a *msh* class and it's internal structure. The attributes that begin with `f` denote ADCIRC compliant input files required for numerical simulation. Note that this does not depict all attributes of the `msh` class as development is active.

2.3 Data storage/msh class

A msh object is a data storage class that contains triangulation-related attributes and support for solver-specific input files (Fig. 2). The format of the msh class uses MATLAB's dot-structure syntax that enables option-hierarchy and simplifies the interaction with the underlying data. Upon termination of the mesh generator, a msh class object containing the triangulation is returned and can be saved efficiently to disk as a .mat file. While the underlying purpose of the msh class is to store the mesh data, the object-orientated framework allows specific methods to be associated with it. This enables the msh class to act as an intermediary between the numerical solver and the user that can assist the user in generating solver-specific related files and perform popular data-driven operations on the mesh. For instance, there are a set of common techniques that must be applied to a mesh to ensure it's suitable for numerical simulation like renumbering, visually inspecting it, interpolating bathymetric data on it. Rather than have each user independently write their own methods to accomplish these tasks, we believe it to be more advantageous to place these static or dynamic methods inside the msh class.

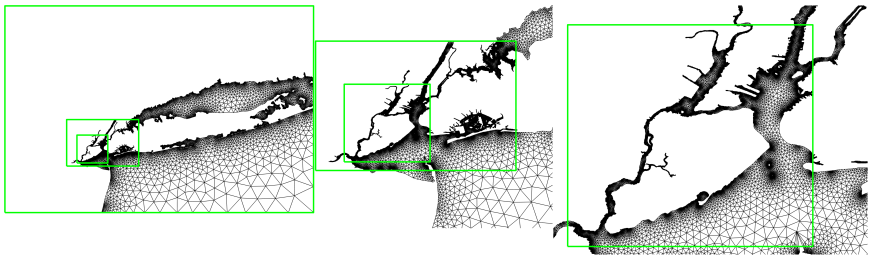


Figure 3: An example of nesting bounding boxes(green boxes) within each other. The outermost box uses a minimum resolution of 1-km while the innermost box uses a minimum resolution of 30-m. Notice how the transitions between boxes are smooth.

2.4 Meshing domain Ω

The region to be meshed is identified using a box or polygon referred to as a bounding box (bbox) along with a constant grid spacing denoted by the variable $h0$ in planar meters. The variable $h0$ represents the minimum mesh spacing or edgelength that is desired in your mesh for that particular region denoted by the bbox.

```
bbox = [min(x) max(x); min(y) max(y)]
```

OR

bbox = [[x$_1 x_2 ... x_N, NaN]', [y_1 y_2 ... y_N, NaN]'$]

where x and y are in horizontal WGS84 coordinates of the top left and bottom right co-ordinates of the box. Note bbox can also be a single, non-self-intersecting polygon composed of N points and thus not a closed box. This can be useful to mesh complicated coastal ocean environments. Bounding boxes can be partially or fully nested any number of times with varying options and geospatial used in each box (Figure 3). This is elaborated more in Section 5. The meshes from this software are built to be highly isotropic in a spherical coordinate system like the Earth and thus may appear distorted in a projected planar coordinate system.

2.5 Digital Elevation Model

A digital elevation model (DEM) is a structured grid with a fixed horizontal resolution that defines z data at each grid point denoted by x and y. Here the coordinate z represents the height in meters *above* a vertical datum that could be based off a tidal surface or an orthometric surface (e.g., mean sea level/MSL or NAVD88). A topobathy DEM is one that covers both overland ($\geq$ MSL) and seamlessly transitions underwater ($\leq$MSL) and a bathymetric DEM is one that has coverage for points $\leq$ MSL.

For some edgefunctions (explained below), a bathymetric DEM is required (e.g., wavelength, slope, channels, CFL). In these cases, we require the DEM to be in the WGS84 horizontal datum, have a vertical unit of meters, and be in the NetCDF file format. A NetCDF format for our software can be achieved by typing:

```
gdal_translate -of NetCDF in_filename out_filename.nc
```

The DEM can be passed to the same `geodata` class used to process the shapefiles above, such as:

```
data = geodata('DEM',filename_dem.nc)
```

And to visualize it:

```
plot(data,'dem')
```

It is strongly encouraged to inspect your DEM before building any meshes and thoroughly understand its shortcomings as these deficiencies can significantly affect the mesh generation process. If your primary DEM contains no-data values, then a secondary back-up DEM can be supplied using the optional name-value argument "backupdem" followed by the file name. This back-up DEM will replace out the fill-values on the primary DEM ensuring that all the edgefunctions can be utilized. However, it may be more intelligent to modify the initial dataset (to fill the NaNs once-and-for-all).

2.6 Mesh boundary

Shorelines, islands, and other coastal geomorphology in *OceanMesh2D* must be represented using polygons or polylines. These represent geospatial features as piecewise-linear segments (otherwise referred to as Planar Straight Line Graphs or PSLG). A polygon is a 2-tuple of x,y coordinates with features separated by a NaN in which the first coordinate is the same as the last. A polyline is the same as a polygon except it is not closed (i.e., first point doesn't equal the last). For use within the ocean modeling framework, the boundary polygon provided to *OceanMesh2D* can either be a ESRI shapefile or a NaN-delimited vector (e.g., PSLG).

One simple way to obtain a ESRI Shapefile of the meshing boundary for use within *OceanMesh2D* is to perform the following operation on your DEM dataset:

```
gdal_contour -fl 0.0 DEM name_of_shapefile.shp
```

Note that this method (using a contour algorithm) often produces polylines not polygons so the user most likely will have to close the shoreline boundary manually so that it does not self-intersect or intersect with the bounding box (Fig. 4).

We've found more success (than from using gdal) in obtaining a good meshing boundary by using GRASS's r.contour module with the 'cut' parameter set to 50-200 (depending on DEM resolution and quality) to prevent spurs and breaks in the shoreline boundary that can often occur with `gdal_contour`. This encourages contours to be traced that have a sufficiently long length and shorter ones to be discarded. Once the shapefile is obtained, it can be prepared for later use within other *OceanMesh2D* functions by constructing a `geodata` class. For instance, the user passes the filename of the shapefile, the minimum resolution/edgelength (in planar meters) to the `geodata` class constructor, and bbox:

```
data = geodata('shp',filename.shp,'bbox',bbox,'h0',h0)
```

OR

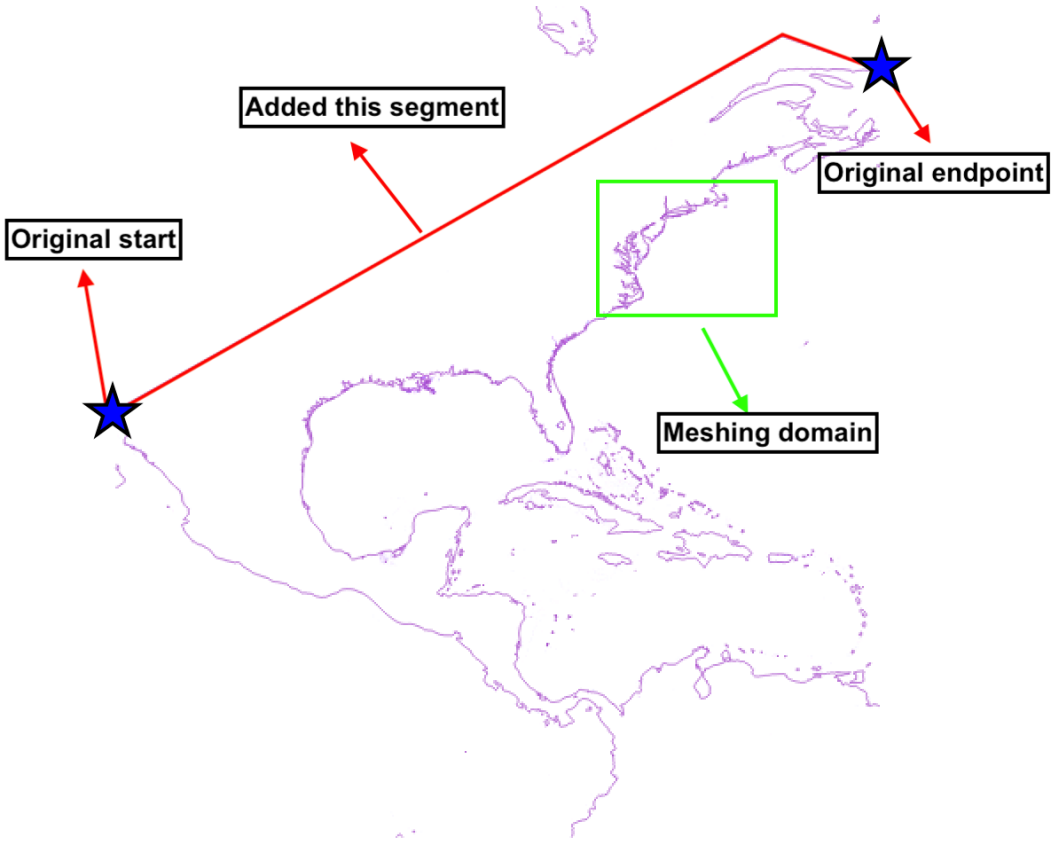


Figure 4: An example of how the user may have to manually close the shoreline contour (purple line) so that it does not intersect with the bounding box (green box) and forms a polygon (first point equals last point). Note the rest of South America is not shown but is a component of the depicted polyline.

```
data = geodata('pslg',NaN-delimited vector,'bbox',bbox,'h0',h0)
```

The shapefile can then be visualized by typing:

```
plot(data,'shp')
```

Note how MATLAB's OOP style has allowed us to overload the `plot` command for both shapefiles and DEMs; this style is used throughout the software.

It's noted that most automatic mesh generators for coastal ocean models require the shoreline boundary to be *simplified* to an extent that matches the desired minimum mesh resolution. This is the case since a very high level of refinement nearshore can lead to a number of numerical stability issues. Further, incongruence between the mesh size function and the boundary resolution can lead to poor quality triangles. However, in our paradigm, we build the mesh oblivious to the disparity between the mesh size function and the resolution of the boundary. Instead, we rely on a series of mesh cleaning steps (Section 4.3) after it is generated to *simplify* the shoreline boundary representation.

3 Edgelength functions

Here we discuss how the mesh resolution is distributed in 2D space. All edgelength functions and methods are encapsulated within the `edgefx` class. After constructing the `edgefx` class by using a variety of name-value arguments, the minimum of all edgelength functions is computed to form the final edgelength function. This is under the assumption that the user is most concerned with preserving the minimum mesh size constraints, which is typically the case.
In this user guide, an edgelength function is denoted by the variable f_h followed with a sub-subscript denoting the type of edgelength function. The edgelength function is calculated in the projected space set forth by the user using the m_map mapping toolbox.

3.1 Mesh size bounds

The minimum mesh size ($h0$) and maximum mesh size bounds (max_el) are supplied to the edgefx class in planar meters like so:

```
fh = edgefx(varargin'h0',50,'max_el',1e3)
```

If the user does not pass a value for $h0$ to the edgefx class it is inherited from the geodata class that is supplied as another variable argument in (varargin). Note that one can additionally constrain the maximum element size nearshore (within 0.1 °) of the shoreline to a value in planar meters by passing.

```
fh = edgefx(varargin,'max_el_ns',250)
```

If a DEM is supplied to the geodata class, then the user can bound the maximum element size over any number of depth/height ranges in meters like so:

```
'max_el', [250 inf,0; 50 0 -10]
```

where, in this case, the maximum element size would be bounded to 250 m overland and to 50-m between 0 and -10 m below the geoid.

3.2 Mesh constraints

Points that the user does not wish to move around in the mesh can be passed through the name-value argument pair *pfix* as a MATLAB 2-column matrix (npfix x 1) that contain the x and y co-ordinates in WGS84. Similarly, connections between *pfix* can also be preserved in the mesh by passing an additional array *egfix*, which is a MATLAB matrix with each row containing two integer values that correspond to the row index into the pfix array. The use of *egfix* assumes *pfix* is also passed. Note that the mesh size constraints are used **as is**, as such, if their point spacing is largely different from that of the desired mesh size function, then poorly formed triangulations may result in the vicinity of these constraints. One easy solution to this is to evaluate your *pfix* constraints with the mesh size function interpolant created inside the *edgefx class* and then use this probability to reject points. This is explained mathematically in the following equation:

$$pfix = pfix(rand(npfix, 1)) < \frac{f_h(pfix)^2}{h0^2} \tag{2}$$

where f_h is the final mesh size function interpolant that is evaluated at the $pfix$ vertices, $h0$ is the minimum mesh size in WGS84 degrees, and $rand$ is a npfix x 1 matrix of random integer values. This implies that if the mesh size function has a lower degree of mesh refinement than in pfix, then pfix may be decimated to help achieve a better final triangulation with these constraints included.

3.3 Distance mesh size function

A distance edge function is used to distribute mesh resolution proportional to its distance from a boundary such as a coastline. For example, in 2D coastal ocean modeling, typically higher resolution is needed nearshore since the spatial scale of nearshore features tends to become smaller and more geometrically complex than in offshore. Additionally, properties and man-made structures reside nearshore that are impacted from storm surge and are thus of interest to capture correctly in the mesh. Currently, two types of distance functions are available, as described below.

1. **Linear distance function (dis)**:
   The simplest distance function available is a linear function of distance from the nearest boundary point:
   $$f_{h_{dis}} = h0 + \alpha_d d \tag{3}$$
   where f_{hdis} is the resolution in space, $h0$ is the minimum resolution, α_d is the percent change of resolution per decimal degree, and d is the distance transform or the distance to the **nearest** boundary point in $\mathbb{R}^2$. To enable this option, enter the following command:

   ```
   fh = edgefx('geodata',geodata_obj,'dis',α_d)
   ```

   Where it is assumed you have already created a `geodata` class with a shapefile and then pass it to the `edgefx` class constructor. **This function is mutually exclusive with** $f_{h_{fs}}$

2. **Feature Size (fs)**:
   The feature size function places resolution according to how wide a 2D feature is. The user

controls the number of elements $\boldsymbol{R}$ that are used to resolve a feature of arbitrary width $\boldsymbol{W}$. **This function is mutually exclusive with** $f_{h_{dis}}$.

$$f_{h_{fs}} = \frac{W}{R} \tag{4}$$

The width of the feature is estimated by calculating the gradient of the distance transform and finding points where this gradient is ≤ 0.90 and $d \leq h0$ (see Koko [2015] for more details). These points are referred to as the approximate medial axis and the distance from the medial axis points to the nearest boundary points are d_{MA}. Then equation (4) becomes

$$f_{h_{fs}} = \frac{2(d_{MA} + d)}{R} \tag{5}$$

where d is the distance transform to the boundary as was described in the linear distance function. To enable this option, enter the follow command with a `geodata` object.

```
fh = edgefx('geodata',geodata_obj,'fs',R)
```

Note that the user does not have to use the mesh domain's boundary to determine the feature size, but this is by default the case. For instance, if one wanted to mesh the floodplain, they might want to mesh up until 5-m above LMSL but still want mesh resolution to conform to the geometric features defined in the 0-m LMSL shoreline. In this case, we require an additional argument to the edgefx class constructor

```
'lmsl',geodata
```

that contains a value argument of a `geodata` instance with this 0-m LMSL shapefile. In these cases, the geodata passed with this name/value pair will be used instead of the mesh domain's boundary to form the feature size edge function.

In V2.0 and on, a new option has been included that will adapt the parameter R in Equation 5 as a function of the estimated shoreline width W. This option can be actiavated by placing a minus sign in front of R to maximally use R elements across the shoreline's width and then automatically scale R to minimally one in thin shoreline areas.

3.4 Wavelength mesh size function

Numerically speaking, the pertinent wave modeled in the finite element framework should be resolved by a minimum number of elements. In 2D coastal ocean modeling, we often seek to resolve model the dominant tidal species (e.g., M2, K1, etc.) accurately to avoid aliasing the tidal wave in space Westerink et al. [1992].

For instance, in order to represent the most energetic semi-diurnal component of the tide in a mesh (e.g., M2), we can estimate its wavelength using shallow water wave theory to ensure a certain number of elements per M2 tidal wavelength:

$$f_{h_{wl}} = \frac{\lambda_{M2}}{\alpha_{wl}} \tag{6}$$

$$f_{h_{wl}} = \frac{T_{M2}}{\alpha_{wl}}\sqrt{gH} \tag{7}$$

where λ_{M2} and T_{M2} are the wavelength and period (12.42 hours) of the M2 tidal wave, H the total water column depth, and $\boldsymbol{\alpha_w}$ the user specified number of elements to resolve the wavelength. We assume that H is approximately equal to the bathymetric depth h, which is a good assumption in the deep ocean. If the M2 wavelength is sufficiently captured, the diurnal species will also be sufficiently resolved since their wavelengths are approximately twice as large as the M2. To enable this option:

```
fh = edgefx('geodata',geodata_obj,'wl',αwl)
```

Note if the user wants to apply different α_{wl} at different depths they can pass a maximum and minimum depth like so:

```
'wl', [100,-10 -100; 200 -150 -300]
```

Where an α_{wl} of 100 has been applied between -10 m and -100 m below the geoid and an α_{wl} of 200 between -150 and -300 m below the geoid. Note the semi-colon in the above name/value pair which implies that each parameter range is defined as the row of a matrix.

3.5 Slope mesh size function

Greater resolution may required in order to resolve bathymetric features such as the shelf break and slope, submarine ridges and troughs that are indicated by significant topographic gradients. The features may be important for coastal ocean models to capture dissipative (in particular due to internal tides in the deep ocean) and reflective effects (due to the shelf break) on tides, surge and trapped shelf waves. The scaling of the slope parameter, commonly called the topographic length scale, is usually represented by the following:

$$f_{h_{slp}} = \frac{2\pi}{\alpha_{slp}} \frac{h}{|\nabla h^*|} \tag{8}$$

where $2\pi/\alpha_{slp}$ is the number of elements that resolve the topographic slope, h is the bathymetric depth, and ∇h^* is the gradient of the filtered bathymetry evaluated on a structured grid of resolution $h0$. The 2π factor is a convention introduced by Lyard et al. [2006] so that α_{slp} can be set to a value equal or similar to α_{wl}, e.g. around 20-30.

Typically the gradient of the original bathymetric field can be rather noisy introducing a large number of nodes despite the fact that small features likely have marginal effects on shallow water flow. Thus a low-pass or bandpass filter (`filt2` obtained at [3], with dependencies on the image processing toolbox removed) is applied to the bathymetry before taking the gradients. The decision of the bandpass filter lengths $[fl_{low}, fl_{high}]$ can be defined by the user. By default, a low-pass filter length is **automatically** applied to the bathymetric based on an estimate of the *local* Rossby radius of deformation:

$$L_R = \frac{\sqrt{gb}}{f} \tag{9}$$

where f is the Coriolis force, g is gravity, and b is the bathymetric depth below the vertical datum in meters. By *local* we mean that we segment the meshing domain into partitions by binning the range of the Rossby radius within the domain and filter each region with a low-pass filter cutoff determined by that mesh domain partition's mean L_R. Partitioning the meshing domain is critical

[3] https://www.mathworks.com/matlabcentral/fileexchange/61003-filt2-2d-geospatial-data-filter

for this approach to filtering since meshing domains can span large regions of latitude with highly varying f. Further, we want to filter bathymetric features that are not relevant to the underlying physical processes and this changes as a function of location. From figure, the application of the Rossby radius slope filter focuses mesh resolution along long and deep features and avoids the placement of resolution in the deep ocean and on smaller scale features.

The slope edgelength function is enabled by typing:

`hh = edgefx('geodata',geodata_obj,'slp',`α_{slp}`,'fl',[`fl_{low}, fl_{high}`])`

Note: `'fl'` may be omitted in which case the automatic low-pass filter described above is applied; $[fl_{low}, fl_{high}]$ is a (n x 2) array for n different bandpass filter lengths; set $fl_{high} = 0$ to apply only a low-pass filter; a high-pass filter is not recommended. Also note that the same depth values as were shown in the wavelength mesh size function can be applied to the slope argument in the edgefx constructor in the same way. By default, the slope mesh size function is turned *off* 50-m below the vertical datum. Often this region (below 50-m) in the meshing domain would require an exceedingly fine mesh size function (i.e., smaller than the Nyquist frequency) to correctly low-pass filter the bathymetry.

3.6 Polyline mesh size function

There are often both dredged navigation channels, submerged river basins, and other coastal geomorphology in the nearshore and shelf region that are not captured in the aforementioned mesh size functions. Along and in marine navigation channels, the bottom friction is locally reduced due to deeper depth relative to neighboring regions. This can enhance the velocity along the centerline of the channel and relatively coarse resolution can alias these processes. Nearshore and immediately overland, channel entrances are locations where rivers and streams drain into the larger basins. The mouths of these streams and rivers are typically the first locations where flooding will occur during storm surge events so resolving them locally with higher resolution can be advantageous to correctly capture inundation patterns.

A stream or channel network can be found by thresholding upslope area and this can be done with many popular software packages (GRASS GIS, ArcGIS, TopoToolbox, Terraflow, etc.). It is often useful to visualize the stream network in a GIS program and then prune spurious channels using the available GIS modules. Once the stream network has been calculated, the channel edge function traverses each stream creating a circular stencil around each point (Fig. 5). The circular region is determined by assuming the channel has a v-shaped cross-section with the bathymetric depth at that particular channel's point and an angle of reslope of 60^o. In the stencil around each channel point, mesh resolution is assigned based on the following equation:

$$f_{h_{ch}} = \frac{b}{\alpha_{ch}} \tag{10}$$

where b is the bathymetry in meters below the vertical datum. The user needs to specify α_{ch}, which is how the mesh resolution near the channel scales with depth (as shown earlier) and a MATLAB cell-array of channel points like so:

`hh = edgefx('geodata',geodata_obj,'ch',`α_{ch}`,'channels',cell-array of points)`

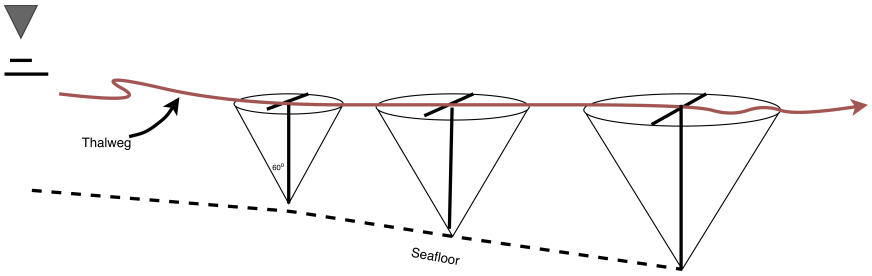


Figure 5: A schematic illustrating various aspects of the channel mesh size function. The thalweg is depicted as the maroon line. Along some points along the thalweg, cones are drawn to depict the regions inside the mesh size function where the mesh size function follows (10).

The user can optionally bound the minimum resolution in and along the channel by passing the constructor `'min_el_ch',150` (default min_el_ch is 100-m) to bound the resolution below by 100 m in the vicinity of the channels only, for instance.

Ridges, crestlines, and other overland features that impede the movement of floods can be identified in the same way as channels–through upslope area. To accomplish this for overland features however, we need to multiply the DEM by -1 or *flip it* so that overland ridges become overland channels. We then proceed to use a GIS software package to identify the channels on the flipped DEM via upslope area thresholding and can apply the same mesh size function.

4 Mesh stability and validity

4.1 Grading

The final stage of the development of an edgelength function f_h involves ensuring a size smoothness limit, g such that for any two points $\boldsymbol{x_i}$,$\boldsymbol{x_j}$, the local increase in size is bounded such as:

$$f_h(\boldsymbol{x_j}) \leq f_h(\boldsymbol{x_i}) + \alpha_g ||\boldsymbol{x_i} - \boldsymbol{x_j}|| \tag{11}$$

A smoothness criteria is necessary to produce a mesh that can simulate physical processes with a practical time step as sharp gradients in mesh resolution typically lead to highly skewed angles that result in poor numerical performance. We adopt the method to smooth f_h originally proposed by Persson [2006] and later adapted by Engwirda [2014]. We have further adapted this algorithm for support on structured grids. Details of the algorithm are not relevant for a user guide and curious readers should see Persson [2006], Engwirda [2014].

A smoother edgelength function is congruent with a higher overall element quality but with more triangles in the mesh. Generally, setting $0.2 \leq \alpha_g \leq 0.3$ produces good results. The user can control the grade of the mesh by passing an option `g` to the `edgefx` class.

```
hh = edgefx(varargin,'g',α_g)
```

Note that depth bounds can be given to `g` to vary the grade depending on depth (in a similar manner to the other mesh size functions).

4.2 CFL limiting

Many numerical models for shallow water flow are limited by the Courant-Friedrichs-Lewy (CFL) condition associated with explicit temporal integration schemes that are often employed. The Courant number, Cr is used to measure the stability in terms of the CFL condition Wirasaet et al. [2015]:

$$Cr = \frac{(|\boldsymbol{u}| + \sqrt{gH})\Delta t}{f_E} \tag{12}$$

where $|\boldsymbol{u}| \equiv \sqrt{u^2 + v^2}$ is the velocity magnitude, g is the acceleration due to gravity, H is the total water depth, Δt is the time step, and f_E is the element size, calculated as the diameter of the largest circle inscribed in the triangular element. The CFL condition requires that $Cr < 1$. Stricter conditions may also be relevant for some numerical models and due to nonlinearities in the governing equations Brufau et al. [2004].

It is beneficial to build the mesh with the CFL condition in mind so that a target time step may be used without fear of numerical instability. Unfortunately, $|\boldsymbol{u}|$ and H are not known *a priori*. However, the still water depth, h is known from the DEM used to build the mesh. h is a suitable proxy for H throughout most of the ocean, and $|\boldsymbol{u}|$ may be estimated from linear long theory:

$$|\boldsymbol{u}| = \eta\sqrt{\frac{g}{h}} \tag{13}$$

where η is the free surface elevation. η is on the order of $\sim$ 1 m in most of the ocean but may reach close to $\sim$ 10 m in coastal regions with very large tidal ranges such as the Bay of Fundy, King Sound, and Hudson Bay. Setting $\eta = 2$ m is probably suitable for most regions, hence this is the default value used in *OceanMesh2D*. Finally, we set $f_h = f_E$ and rearrange (12) to find the minimum edgelength, f_h possible for a given h and Δt, based on some value of $Cr \leq 1$ (we apply $Cr = 0.5$, which provides a solid buffer to allow for the effects of the nonlinearities).

The above approximations should work well in most of the ocean including nearshore when building meshes for shallow water phenomenon such as tides and surge. However, they will break down overland and when $\eta \sim h$, such that the linear long wave theory will not be relevant (so h cannot be used as a proxy for H and (13) is no longer useful). We handle this issue in *OceanMesh2D* by simply setting $H = \eta$ and $|\boldsymbol{u}| = \sqrt{gH}$ for all $h < \eta$ (η is 2 m by default). This assumes that the flow velocity is critical ($Fr = 1$, where Fr is the Froude number).

Finally, the software can automatically select a suitable value of Δt for CFL limiting based on the nearshore conditions. In other words, Δt that satisfies (12) at the coast is used to limit f_h everywhere else. This is determined by:

$$\Delta t = \min\left[\frac{f_{h_d} Cr}{|\boldsymbol{u}| + \sqrt{gH}}\right] \tag{14}$$

where f_{h_d} is either $f_{h_{fs}}$ or $f_{h_{dis}}$, depending on which is invoked. f_{h_d} is the major control of resolution at the coast so using (14) to determine Δt will preserve the resolution here.

To enable the automatic CFL limiting option the user simply passes a name/value pair of `'dt',0` to the `edgefx` class

```
fh = edgefx(varargin,'dt',0)
```

If the user wishes to specify a dt rather than use automatically detect it, then change the value of dt in the call above.

4.3 Ensuring Mesh Validity

Mesh validity is checked after it has been created using the `msh.build` routine. The properties of a valid mesh were described in Section 2.2. Properties numbered 1 to 3, referring to ensuring no disjoint or hanging nodes and the correct ordering of nodes, is handled with the `fixmesh` function that was provided with the *DistMesh* program. Property number 4, referring to ensuring that there are only two traversal paths along the mesh boundary, is handled through the `Make_mesh_boundaries_traversable` function described below.

The `Make_mesh_boundaries_traversable` routine alternates between checking and deleting exterior and interior portions of the graph (elements) exhaustively until convergence is obtained, defined as: having no nodes connected to more than two boundary edges (all the boundary nodes are traversable along a single mesh boundary pathway). The `fixmesh` function is called by `Make_mesh_boundaries_traversable` routinely to ensure no disjoint or hanging nodes after elements are deleted at each step. The function begins by calling the `delete_exterior_elements` sub-function to delete the exterior portion of the graph. After this the boundary edge requirements are checked. If convergence is not obtained the `delete_interior_elements` sub-function is called. This process is repeated until convergence.

The `delete_exterior_elements` sub-function finds small disjoint portions of the graph and re-

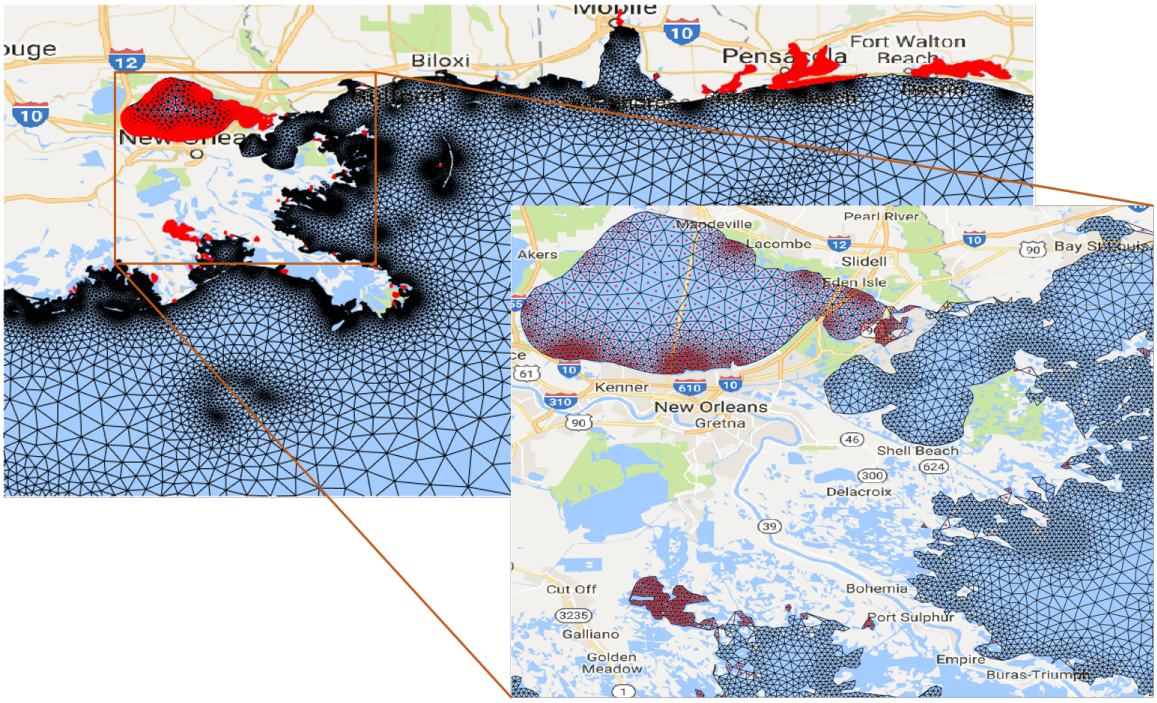


Figure 6: Red dots indicate the centroids of the elements that are deleted on the first pass of the `delete_exterior_elements` sub-function contained within the `Make_mesh_boundaries_traversable` routine. As shown on the blown-up graphic, Lake Pontchartrain in New Orleans is disconnected from the main grid and represents a very small area-fraction of the total mesh (the mesh spans the entire U.S. East and Gulf Coasts), so it is deleted.

moves them using a "spider-search" algorithm, or more precisely, a breadth-first search (BFS). The BFS starts at a random element of the graph, finds the neighboring elements and flags each of these elements that they have been checked. The same action will then be conducted on the each of the neighboring elements just flagged. This process will continue until there are no previously checked (flagged) elements remaining that are neighboring any of the elements in this disjoint portion of the graph. The individual disjoint portions are removed if their composition represents less than a specified area-fraction, μ_{co} of the initial total mesh area, which is set to be 0.25 (25%) by default. The μ_{co} area-fraction cutoff value can be used to control whether certain bays or seas that are separated from the major ocean portion of the graph are deleted or not. Alternatively setting $\mu_{co} \geq 1$ corresponds to an absolute area cutoff in km^2. This may be necessary if one is interested in keeping certain lakes and polders of a minimum size in the mesh (which may be later provided connectivity to by merging with another mesh). Use the following to define the desired value of μ_{co} in the `meshgen` class:

```
msh = meshgen(varargin,'dj_cutoff',μco)
```

An example of the disjoint portions to be removed on the first pass of `delete_exterior_elements` is shown in Figure 6. It is acknowledged that in this case the connectivity through to the lake from the sea is missing and ideally the addition of elements may be desired. Alas, it is far more cumbersome to add elements than delete them. Besides, this is an issue related to the minimum resolution ($h0$) being insufficient to resolve the channel, so the user can decrease $h0$ and re-mesh if required.

The `delete_interior_elements` sub-function deletes elements that are within the interior of the mesh that are found to be connected to more than two boundary edges. A classic example of this is illustrated in Figure 7 along the Pensacola Beach barrier island. Because the barrier island is thin compared to the elemental resolution, numerous nodes are connected to elements on both northern and southern sides of the barrier island, i.e. they are connected to four boundary edges. This `delete_interior_elements` sub-function deletes connecting elements to ensure that the offending nodes have only two connected boundary edges. In Figure 7 the arrow points to one of the elements that we wish to delete to make the connecting node traversable. The reason that this element is the one to be deleted is that all the nodes on the element are connected to a boundary edge, i.e. the element has two boundary edges. Note how the two other elements connected to the offending node have only one boundary edge. Unfortunately, the choice is not always as clear cut as this. In some instances there may be more than one connecting element that has two boundary edges. In this case the lowest quality *qualifying* connected element is deleted. In other instances there may be no connecting elements that have two boundary edges in which there are many examples contained within Figure 7. Here, the lowest quality connected element is deleted. It is worth pointing out that when there are no connecting element with two boundary edges the node will often remain untraversable after deleting one of the elements, but `Make_mesh_boundaries_traversable` iteratively calls `delete_interior_elements` until this condition is met.

5 Meshgen class

Mesh generation is accomplished with a standalone class called `meshgen` that only requires a mesh boundary and a mesh size function. The `meshgen` class contains the method `build` that calls the algorithm to generate the mesh using a modified force equilibrium approach.

First, the user passes their options and data to create an instance of the `meshgen` class.

```
mesh = meshgen('ef',{fh1},'geodata',{gdat})
```

where `fh` is the edgefx class object and where `gdat` is the geodata class object. Note how the user passes these instances in a MATLAB cell-array denoted by {}; this implies that the user may pass multiple instances of these class objects. If this is the case, then the class objects should be listed in a user-defined hierarchical order (coarse to fine mesh resolution). Most importantly, it is necessary and checked that the bounding box of the coarsest mesh size function fully encompass **all** of the finer nested mesh size functions. It's noted that, if the user only has one instance of the `edgefx` and `geodata` classes, then they need not pass the data as a cell-array to the `meshgen` class constructor. Following this, the user calls the mesh generation algorithm by typing:

```
mshopt = mshopt.build
```

where `mshopt` is an instance of the `meshgen` class with the users boundary data and mesh size options, which then displays the mesh (if `plot_on`= 1) as it is incrementally modified at `nscreen` intervals. After convergence (see 2.2 or reaching the maximum number of iterations `itmax` (100 by default), it produces a `meshgen` class object in which the user can access the triangulation's points and triangle table (meshgen.grd) and all the various options the mesh was built with. There are a

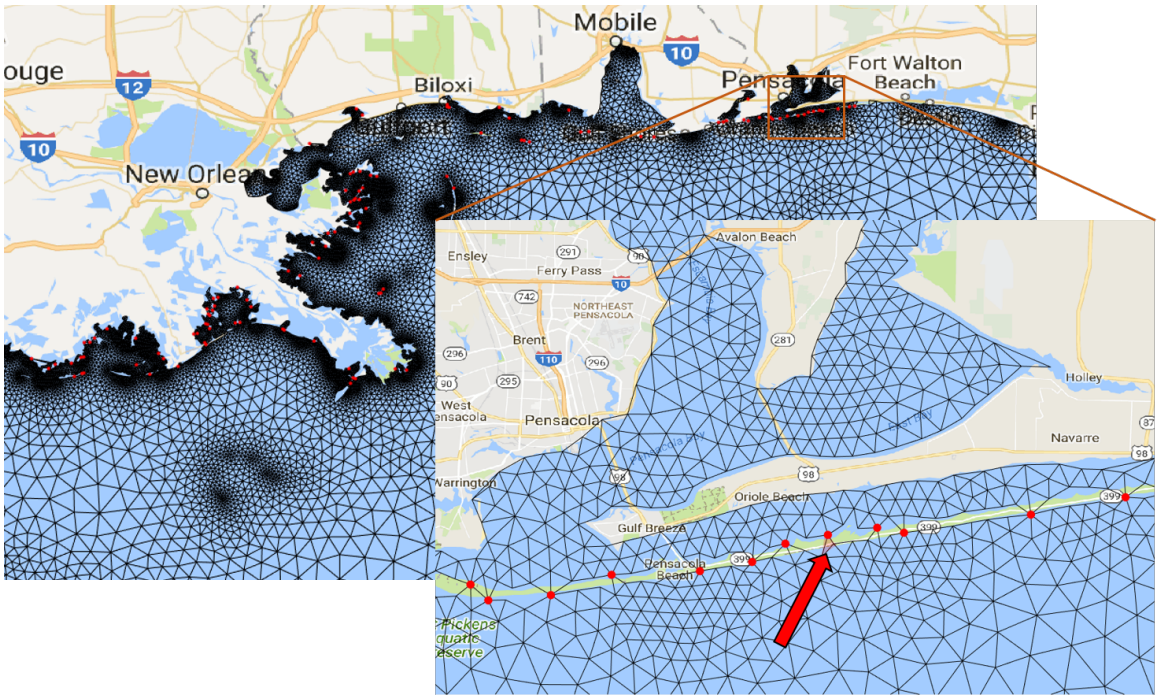


Figure 7: Red dots indicate the elements that are deleted on the first pass of the `delete_interior_elements` sub-function contained within the `Make_mesh_boundaries_traversable` routine. As shown on the blown-up graphic, a number of vertices along the thin Pensacola Beach barrier island are connected to more than two boundary edges. In order to deal with this, elements connected to the node must be deleted; for example the element directed at by the arrow.

number of methods that can be applied to `msh` class objects which will be elaborated in section

Here we present a series of examples concerning how to use the software and some its strengths. The mesh region could be modified for anywhere in the world by changing the extents of `bbox` **only if the ESRI shapefile has coverage in that region**. As mentioned earlier, it's recommended that the user create a 'datasets 'directory in the rootdir (i.e., Oceanmesh2D) of where they installed this software and place their ESRI shapefile(s) and DEM(s) they want to use there.

A couple of things to note when running the code:

1. If `plot_on == 1` the mesh starts off looking really bad when you run `mshopt.build`! In just a few iterations you will see the mesh start to improve. By default `plot_on` is not enabled.
    - Periodically, the algorithm will delete, add, and split long edges which will result in strange looking triangulation at times.
2. You may also notice some oversized and/or badly connected elements around channels, far inland, etc., during `msh.build`. These will be cleaned up after convergence of the mesh generator algorithm to ensure that a *valid* mesh is returned (refer Section 2.2).
3. Take care when defining `bbox` and the `min_el` to ensure you do not eat up all the memory on the computer. Similarly, take care when plotting the DEM and edgelength functions. It is very easy to slow the computer down and fill up your graphics card when plotting these.

5.1 Around the world: *Example_1_NZ.m*

This example illustrates the ability to mesh anywhere in the world if one has a shapefile in the region. Note that this example does not use a DEM.
Data sources:

- GSHHS: https://www.ngdc.noaa.gov/mgg/shorelines/

```
%% STEP 1: set mesh extents and set parameters for mesh. South Island of New Zealand
bbox      = [166  176   % lon_min lon_max
              -48 -40];  % lat_min lat_max
min_el    = 250;         % minimum resolution in meters.
max_el    = 20e3;        % maximum resolution in meters.
max_el_ns = 1e3;         % maximum resolution nearshore in meters.
grade     = 0.20;        % mesh grade in decimal percent.
R         = 3;           % number of elements to resolve features.

%% STEP 2: specify geographical datasets and process the geographical data to be ...
    used later with other OceanMesh classes...
coastline = 'GSHHS_f_L1';
gdat = geodata('shp',coastline,'bbox',bbox,'h0',min_el);
% NOTE: You can plot the shapefile with bounding box by using the overloaded plot ...
    command:
% plot(gdat,'shp');

%% STEP 3: create an edge function class
fh = edgefx('geodata',gdat,'fs',R,'max_el',max_el,'max_el_ns',max_el_ns,'g',grade);

%% STEP 4: Pass your edgefx class object along with some meshing options and build ...
    the mesh...
mshopts = meshgen('ef',fh,'bou',gdat,'plot_on',1);
% now build the mesh with your options and the edge function.
mshopts = mshopts.build;

%% STEP 5: Plot it and write a triangulation fort.14 compliant file to disk.
plot(mshopts.grd,'tri');
write(mshopts.grd,'South_Island_NZ');
```

Figure 8: A simple example of how to use *OceanMesh2D* with a feature size edge function along with some user defined bounds to control resolution in the meshing domain. Estimated time to completion 4 minutes.

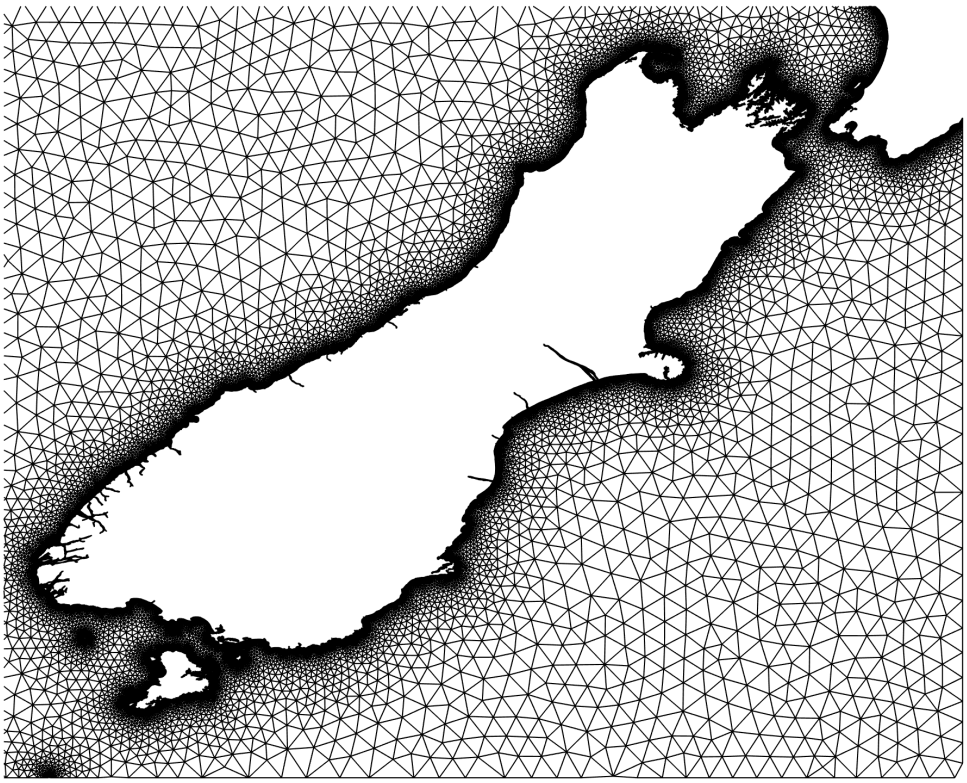


Figure 9: Result: Convergence after 30 iterations in 4 wall-clock minutes on PC, 51,812 vertices, 90,869 elements, $\overline{q_E} = 0.958$

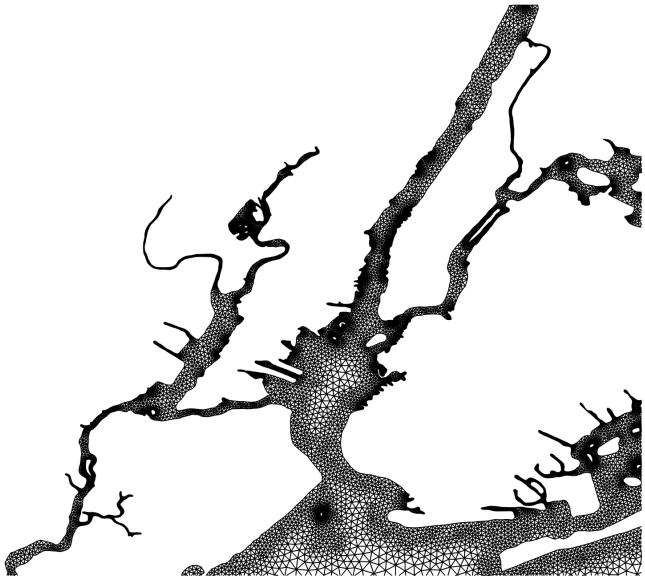
5.2 High fidelity: *Example_2_NY.m*

Using higher resolution digital elevation data to create the coastline boundary will dramatically improve the results (i.e., a more accurate meshing boundary). See the example below that uses the shapefile "PostSandyNCEI.shp" and the "PostSandyNCEI.nc" DEM, which was created by merging four Post-Sandy NCEI DEMs using the software GRASS Development Team [2017]. This example illustrates the ability to capture small regions with high fidelity.
Data sources:

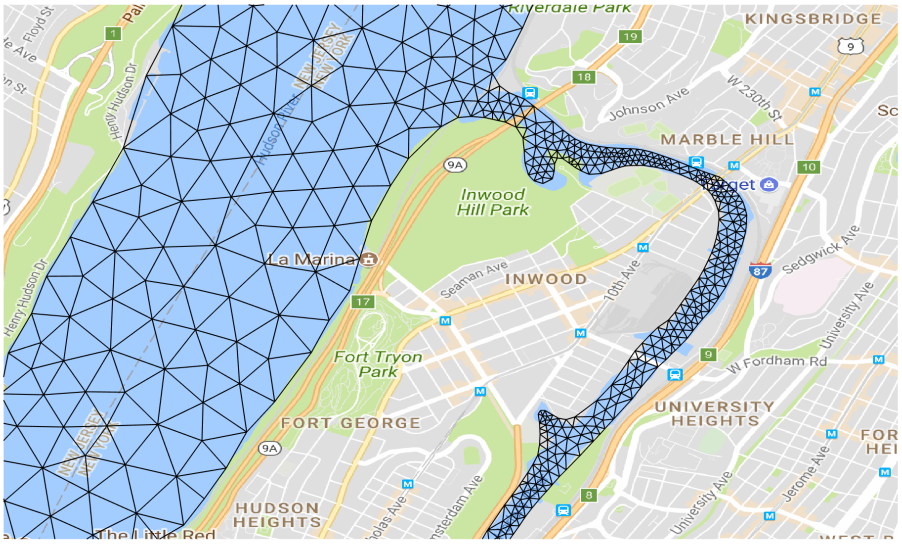
- The PostSandy ESRI Shapefile and DEM (download from https://drive.google.com/open?id=1LeQJFKaVCM2K59pKO9jDcB02yjTmJPmL) were obtained by merging four Post-Sandy NCEI tiles around the New York/Long Island/Jamaica Bay area and then extracting a ESRI Shapefile from the DEM using GRASS's r.contour module with a cut parameter of 150. Then this contour was manually closed following Section 2.6.

```
%% STEP 1: set mesh extents and set parameters for mesh. New York high resolution
bbox      = [-74.5 -73.8
               40.5  40.9];
min_el    = 30;          % minimum resolution in meters.
max_el    = 1e3;         % maximum resolution in meters.
max_el_ns = 240;         % maximum resolution nearshore.
dt        = 2;           % Ensure mesh is stable at a 2 s timestep.
grade     = 0.20;        % mesh grade in decimal percent.
R         = 3;           % Number of elements to resolve feature.

%% STEP 2: specify geographical datasets and process the geographical data to be ...
    used later with other OceanMesh classes...
coastline = 'PostSandyNCEI'; dem  = 'PostSandyNCEI.nc';
gdat = geodata('shp',coastline,'dem',dem,'bbox',bbox,'h0',min_el);
% NOTE: You can plot the dem with shapefile and bounding box by using the ...
    overloaded plot command:
%plot(gdat,'dem');

%% STEP 3: create an edge function class
fh = edgefx('geodata',gdat,'fs',R,'max_el',max_el,'max_el_ns',max_el_ns,...
            'dt',dt,'g',grade);

%% STEP 4: Pass your edgefx class object along with some meshing options and build ...
    the mesh...
mshopts = meshgen('ef',fh,'bou',gdat,'plot_on',1);
% now build the mesh with your options and the edge function.
mshopts = mshopts.build;

%% STEP 5: Plot it and write a triangulation fort.14 compliant file to disk.
plot(mshopts.grd,'tri');
write(mshopts.grd,'NY_HR');
```

Figure 10: This example uses a high resolution DEM and coastline shapefile to mesh the New York area. The DEM bathymetry is used to ensure the mesh is stable with a 2 second time. Estimated time to completion: about 5 minutes.



(a)



(b)

Figure 11: Result: Convergence after 40 iterations in approximately 5 minutes on a PC. There are 19,907 vertices, 33,212 elements, $\overline{q_E} = 0.956$.

5.3 Large domains with *locally* high refinement: *Example_3_ECGC.m*

This example illustrates that large sized domains (around 50° by 50°) can be handled by the software in relatively short times (< 30 minutes) on a personal computer with small regions of locally high refinement. Reading the coastline and prepping the edgelength functions can take about half the total time. Once prepped, convergence in `msh.build` can be quick. Note that you can create multiple instances of the `edgefx` and `geodata` classes and pass them all as a cell-array to `meshgen`. Since it is in general the case that two adjacent `edgefx` classes have disparate mesh size functions, the `meshgen` class automatically calls the `smooth_outer` command to ensure the transitions in and out of the smaller domains are graded (smooth) to the user defined local grade. The insets must be placed completely within the bounding box of the coarsest outer domain (the first entry in the cell array), however there is little restriction on how the insets are placed otherwise. Insets can be recursively nested any number of times and/or placed in different locations without overlap.

Data sources:

- GSHHS: https://www.ngdc.noaa.gov/mgg/shorelines/
- topo15_compressed.nc: ftp://topex.ucsd.edu/pub/srtm15_plus/
- PostSandyNCEI: same dataset as prior example.

```
%% STEP 1: set mesh extents and set parameters for mesh. The greater US East Coast ...
    and Gulf of Mexico region
bbox      = [-100 -50; 10  60]; % lon min lon max; lat min lat max
min_el    = 1e3;        % minimum resolution in meters.
max_el    = inf;        % maximum resolution in meters.
wl        = 60;         % 60 elements resolve M2 wavelength.
dt        = 2;          % Ensure mesh is stable at a 5 s timestep.
grade     = 0.20;       % mesh grade in decimal percent.
R         = 3;          % Number of elements to resolve feature.
%% STEP 2: specify geographical datasets and process the geographical data to be ...
    used later with other OceanMesh classes...
coastline = 'GSHHS_f_L1'; dem  = 'topo15_compressed.nc';
gdat1 = geodata('shp',coastline,'dem',dem,'bbox',bbox,'h0',min_el);
%% STEP 3: create an edge function class
fh1 = edgefx('geodata',gdat1,'fs',R,'wl',wl,'max_el',max_el,...
             'dt',dt,'g',grade);
%% Repeat STEPS 1-3 for a high resolution domain for High Res New York Part
min_el    = 30;         % minimum resolution in meters.
max_el    = 1e3;        % maximum resolution in meters.
max_el_ns = 240;        % maximum resolution nearshore.
coastline = 'PostSandyNCEI'; dem = 'PostSandyNCEI.nc';
% Bounding box is automatically taken from the DEM file
gdat2 = geodata('shp',coastline,'dem',dem,'h0',min_el);
fh2 = edgefx('geodata',gdat2,'fs',R,'wl',wl,'max_el',max_el,...
             'max_el_ns',max_el_ns,'dt',dt,'g',grade);
%% STEP 4: Pass your edgefx class object along with some meshing options and build ...
    the mesh...
mshopts = meshgen('ef',{fh1,fh2},'bou',{gdat1,gdat2},'plot_on',1);
mshopts = mshopts.build;
%% STEP 5: Plot and save the msh class object/write to fort.14
m = mshopts.grd; % get out the msh object
m = makens(m,'auto',gdat1); % make the nodestring boundary conditions
plot(m,'bd',1,'Mollweide'); % plot on Mollweide projection with nodestrings
save('ECGC_w_NYHR.mat','m'); write(m,'ECGC_w_NYHR');
```


Figure 12: Similar to example two but now embeds the area of high resolution around New York from the previous example seamlessly into a larger outer domain. Estimated time to completion: 20-30 minutes.

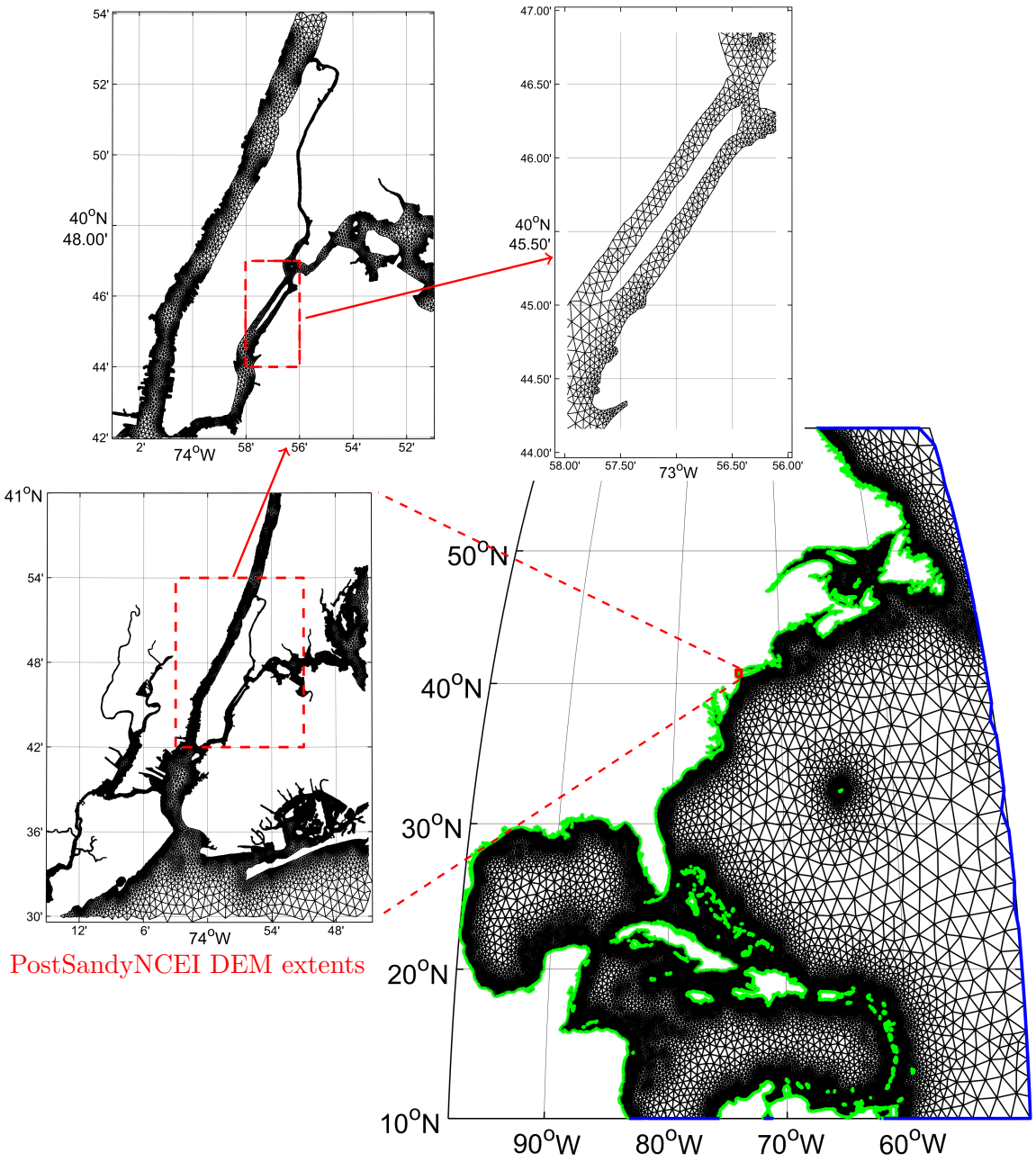


Figure 13: Desired result: Convergence after 50 iterations. There are 237,755 vertices, 421,329 elements, $\overline{q_E} = 0.967$.

6 Post-processing functions

Here we describe some of the most useful post-processing functions we've found in our studies and then some of them in the following section through an example of a typical workflow.

6.1 utilities/ValidateTides.m

```
ValidateTides(fort53,stadat.csv,constituents)
```

Given a harmonic output file from an ADCIRC simulation in NetCDF format (i.e., fort.53.nc), a station database CSV file (stadat.csv) created from calling GatherStationsFromKML.m, and the desired tidal harmonics to be validated, this script can compute the complex root mean square error (RMSE) major semi-diurnal and diurnal constituents and/or the total complex RMSE of all or some combination of constituent labels (M_2, S_2, N_2, K_2, K_1, O_1, P_1, Q_1, 'all'). This subroutine outputs a .txt file called Harm.txt that contains a list of the complex RMSE for each station.

6.2 utilities/HarmonicsCompare

```
HarmonicsCompare()
```

Given the Harm.txt file produced by ValidateTides.m, this subroutine plots and saves to disk various 1-to-1 scatter plots of the tidal constituents and a variety of error metrics. See and edit the header of this script to change these options.

6.3 utilities/Make_f15

```
msh_obj=Make_f15(msh_obj, ts, te, dt, varargin )
```

This subroutine creates the control file for an ADCIRC simulation. Input the msh_obj, the simulation start time (e.g., '01-Jan-2018 03:00'), end times, timestep (in seconds) and any other optional arguments relevant to the numerical simulation and get back a msh_obj with the f15 (i.e., fort.15) structure populated. The f15 sub-structure inside the msh_obj can be modified through MATLAB's dot-structure syntax. `help Make_f15` provides a more detailed description of what is possible with this subroutine.

6.4 utilities/Calc_f13_inpoly

```
Calc_f13_inpoly(msh_obj,attribute,polys,cfvals,inverse,cf_vals_on_mesh)
```

This subroutine takes a `msh_obj` and puts on a specified f13 `attribute` inside (or outside) of polygonal regions defined in a cell-array (`polys`) with the corresponding vector of `cfvals`. Currently the following `attribute`(s) are supported:
`'Cf'` (quadratic_friction_coefficient_at_sea_floor) with a default value of 0.0025;
`'EV'` (average_horizontal_eddy_viscosity_in_sea_water_wrt_depth) with a default Smagorinksy coefficient of 0.2.

`'mannings_n_at_sea_floor'` Manning's n type friction coefficient with a default value of 0.02.

The optional `inverse` argument is a vector of 0s and 1s corresponding to selecting the inside (0), or outside (1) of the polygon to apply the `cfval` for each polygonal region (the inside option is applied by default without supplying this argument). The optional `cf_vals_on_mesh` argument is

a vector of the attribute values (with length equal to the number of mesh vertices) that is simply applied into the f13 struct format.

6.5 utilities/Calc_tau0

```
msh_obj=Calc_tau0(msh_obj,varargin)
```

Input a msh_obj with bathymetric data on it and get back the tau0 primitive weighting for continuity equation for the Generalized Wave Continuity Equation in the f13 struct of the msh_obj. Optional name/value pairs (varargin) are:

1. 'distance' - some positive scalar

   the mean distance to neighboring nodes at which to switch between depth-based tau0 (below) and the default tau0 = 0.03 (default distance is 2 [km])

2. 'depth' - some positive scalar

   the depth at which to switch between tau0 = 0.005 and tau0 = 0.02 (default depth is 10 [m])

6.6 utilties/Calc_IT_Fric

```
msh_obj=Calc_IT_Fric(msh_obj,N_filename,varargin)
```

Input a msh_obj with bathymetry and the bathymetric slope interpolated onto it (see interp function), and get back the internal_tide_friction ADCIRC fort.13 attribute populated into the f13 structure of the msh_obj. The basic theory and method of computation of this attribute (internal tide energy conversion) is detailed in Pringle et al.. Its application and sensitivity is detailed in Pringle et al. [2018]. N_filename is the path and filename of a '.mat' file containing the gridded data of buoyancy frequencies covering at least the region of the msh_obj. This can be generated with the *Make_Gridded_N.m* function found at https://github.com/WPringle/ADCIRC_MATLAB/tree/master/Internal_tide. Optional name/value pairs (varargin) are:

1. 'type' - 'local_dir' (default), 'local_sca' or 'nonlocal'

   First two options are the local (directional or scalar, respectively) internal tide generation assumption types Pringle et al.. For those wishing to use the 'nonlocal' internal tide generation option Pringle et al., the *Calc_dJ_Nyc_Unstruc.m* and its subsidiary functions found at https://github.com/WPringle/ADCIRC_MATLAB/tree/master/Internal_tide are required.

2. 'crit' - 'Nb' (default), or 'Nmw'

   The buoyancy frequency definition used ('Nb' is the value at the seabed, 'Nmw' is the weighted depth-averaged value) to define the the degree of criticality of the slope. The dissipation coefficients are saturated out when criticality > 1 Pringle et al..

3. 'Cit' - some positive scalar value (0.20 is default value)

   This is a scalar multiplication factor of the dissipation coefficients.

4. 'cuttoff_depth' - some positive scalar value (100 m is default value)

   This is the depth above which the dissipation coefficients are set to zero.

6.7 utilties/Calc_Sponge

```
msh_obj = Calc_Sponge(msh_obj,W,generator,varargin)
```

Input a `msh_obj` with bathymetry interpolated onto it (see *GridData.m*), and get back the sponge_generator_layer ADCIRC fort13 attribute (values of absorbing coefficients used in the sponge layer) populated into the f13 structure of the `msh_obj`. A basic description of the absorption-generation sponge layer lateral boundary condition can be found in Pringle et al. [2018]. `'W'` is the width of the sponge layer specified in degrees, or it can have two entries [period frac], where period is the wave period (s) of a specific long wave, and frac is the fraction of the wavelength that is desired for the sponge width (0.1 is common). `'generator'` is 0 for a fully absorbing sponge layer, `'generator'` is 1 for an absorption-generation (tides only) sponge layer (in which case you must also generate a f5354 structure using *Make_f5354.m*), and `'generator'` is -1 for an absorption-generation (arbitrary wave) sponge layer (in which case you must also generate a f2001 structure using *Make_f2001.m*). Optional name/value pairs (`varargin`) are:

1. `'F' - some positive scalar value (20 is default value)`

   the magnitude that a function will decreased by at the position `'dis_ratio'` into the sponge.

2. `'dis_ratio' - some positive scalar value (0.50 is default value)`

   This is the position in terms of the sponge width ratio in which the function will be decreased `'F'` fold.

3. `'spngtype' - 'poly' (default) or 'hyper'`

   the function type used for computing the absorbing coefficients in the sponge. `'poly'` is a second order polynomial, and `'hyper'` is a hyperbolic function.

6.8 utilities/BuildFort24

```
msh_obj = Build
```

6.9 Example of Typical Workflow

Once a mesh is generated, it is then necessary to create the input files for a simulation. This can be a tedious process so we have provided the user the ability to script this. In the code below, we first interpolate the bathymetry using a cell-averaged approach (interp) and then ensure the mesh will be stable with a 2-s timestep by bounding the Courant number to under 0.50. Conversely, one could also run CalCFL for a variety of timesteps to see what DT would give the maximum Courant number under 0.50. Following this, we create the nodal attributes (internal tide friction, primitive weighting in the continuity equation, self-attraction and loading), specify the boundary conditions (no-flux island and land boundaries), add 8 tidal constituent elevation specified boundary conditions from TPXO9, and set some various output options for the control file (fort.15). We have also provided the ability to automatically determine all the NOAA CO-OP, NDBC and blah stations within the meshes extents. By passing the "sta_database" name-value pair, the method makef15 will automatically populate the fort.15 with the stations that fall inside the meshes extents. We conclude the script by using the overloaded `write` method, which will write all the input files for simulation. The user can then simulate ADCIRC with the resulting input files. It is recommended to save the .mat `msh` object that contains all the created input files for simulation as a compact representation of the model.

Data sources:

- TPXO9.1: http://volkov.oce.orst.edu/tides/global.html
- Gridded_N_Values: Make your own '.mat' file using *Make_Gridded_N.m* function found at https://github.com/WPringle/ADCIRC_MATLAB/tree/master/Internal_tide.
- topo15_compressed.nc: SRTM global seafloor dataset.

```
% Post-processing script to take a msh and create all the necessary
% input files for an ADCIRC simulation.
%
%....This example continues on from the Large Domains with locally high refinement ...
    example above....
mshopts   = meshgen('ef',{fh1,fh2},'geodata',{gdat1,gdat2},'plot_on',1);
mshopts   = mshopts.build;
m         = mshopts.grd; % unpack the mesh object

% Specify your simulation parameters
DT        = 2 ;                    % Stable timestep
TS        = '01-Aug-2012 00:00' ; % start time of simulation
TE        = '31-Nov-2012 00:00' ; % end time of simulation
DEMFILE   = 'topo15_compressed.nc' ;
BUOYFILE  = 'Gridded_N_values.mat';
TPXO9     = 'tpxo9_netcdf/h_tpxo9.v1.nc';
CONST     = 'major8' ;
%%
m = interp(m,DEMFILE,'type','depth') ;

m = CheckTimestep(m,DT) ;

m = renum(m) ;

m = makens(m,'auto',gdat{1}) ; % geodata instance from the example
% plot(m,'bd',0); % check your boundary conditions here
m = Calc_tau0(m) ;

m = interp(m,DEMFILE,'type','slope') ;

m = Calc_IT_Fric(m,'Nfname',BUOYFILE) ;

m = Make_f15( m, TS, TE, DT, 'tidal_database', TPXO9, 'const', ...
    {CONST},'sta_database',{'CO-OPS','NDBC',[1]} ) ;
m.f15.dramp = 30;                  % ramp period
m.f15.nramp = 1;                   % ramp type
m.f15.outge = [5 30.0 31.0 3600]; % global elevation
m.f15.ntip=2;                      % SAL + normal tidal potential
m.f15.oute   = [5 30.0 35.0 360]; % station output frequency
m.f15.outhar = [30 120 360 0];     % THAS, THAF, NHAINC, FMV
m.f15.outhar_flag = [0 0 5 0] ;    % NHASE, NHASV, NHAGE, NHAGV

write( m ) ;                       % write all files to disk
```

7 Appendix: Class Prototypes

Here will describe the class properties and the class methods.

7.1 msh: data storage, visualization, and manipulation class.

7.1.1 Properties

- title: the title of the model (less than 20 characters please).
- p: np x 1 array of vertices of the mesh.
- t: nt x 3 triangles of the mesh as a connectivity matrix.
- b: the bathymetry on the mesh.
- bd: NBVV a matrix where each column denotes a no-flux boundary condition as one would "walk" along it.
- op: NVEL same as NBVV but for elevation specified boundary conditions.
- bx: the bathymetric slope in the x-direction (populated through the method in GridData.m).
- by: the bathymetric slope in the y-direction (populated through the method in GridData.m).
- f11: Contains the relevant attributes of an initial density file for ADCIRC (fort.11)
- f13: Contains the relevant attributes for an ADCIRC nodal attributes file (fort.13).
- f15: Contains the relevant attributes for an ADCIRC control file (fort.15).
- f19: Contains the relevant attributes for an ADCIRC non-periodic elevation boundary condition.
- f20: Contains the relevant attributes for an ADCIRC non-perioidic flux/elevation radiation boundary condition.
- f24: Contains the relevant attributes for an ADCIRC Self-attraction and Loading file.

7.1.2 Methods

- `msh`

  `msh_obj = msh(fname,type)`

  This method is the class constructor for the `msh_obj`. It can be formed by passing a filename of a fort.14 , fort.13, or fort.15 compliant ADCIRC files. Note that `fname` does not contain the post-fix denoting the file type. By default, it assumes `fname` is a fort.14 file. If you would like to populate the fort.13 or fort.15 structures in the `msh_obj`, you must additionally put the post-fix type in the `type` argument.

- `plot`

  `[] = plot(msh_obj,type,proj,projection)`

This method takes a msh_obj and plots it according to the type with or without a projection of your choosing. The current options for type are:

- tri: plot the triangulation.
- bd: plot the triangulation with nodestring boundary conditions.
- ob: plot the triangulation with outer boundary vertices.
- b: plot the bathymetry
- slp: plot the slope of the bathymetry.
- reso: plot the mesh resolution (calculated as the radius of the circumcircle).
- resodx: plot the grade.
- tau0: plot the primitive weights in the continuity equation.
- itfric: plot the internal tide energy conversion coefficient.
- cfvals: plot spatially varying quadratic bottom coefficient values.
- sponge: plot the sponge layer coefficients.
- transect: instructs the user to draw a line, then plots the depth along that line.

By default, proj = 1 and projection = 'Transverse Mercator'. Set proj = 0 to plot without a projection. Set projection to another option for a different projection which may be useful on large domains (type m_proj('set') to see available projections; some do not work).

- write

  ```
  [] = write(msh_obj,fname,type)
  ```

  This method takes a msh_obj and writes it to disk with the filename fname. By default, write will write all the populated input files in the msh_obj; however, the user can optionally decide to write only a select few of these files by passing a comma-delimited list of the post-fix name types (e.g., 14,15 to denote fort.14 or fort.15). **Do not name the msh_obj with the name "msh"**.

- msh_obj = interp(geodata,msh_obj,VARARGIN)

  This method takes a msh_obj and interpolates topo/bathymetric data onto it using a cell-averaging approach. It requires the supply of geodata which is either a *geodata* class object/a cell array of *geodata* class objects, or the filename of a NetCDF digital elevation model ('DEM-FILE.nc'). This interpolation method (cell-averaging) leverages the grid-size with the underlying horizontal resolution of the DEM to minimize the aliasing effect that would otherwise occur with nearest neighbor and linear interpolation schemes when there are resolution differences between the mesh and the DEM. The scheme resorts to a nearest neighbor approach when the grid scales are roughly equivalent and a board stencil when they are not.

  By default, interp will interpolate both bathymetry and the bathymetric slope data onto the mesh but the user can control this by passing a name/value pair: 'type','depth' to interpolate only bathymetric depth; or 'type','slope' to interpolate only the bathymetric slope (the slope is required for the Calc_IT_Fric routine).

  Additionally, one can switch to any of MATLAB's default interpolation schemes through the interp name/value pair. The full variety of options in this routine can be shown by typing help msh.interp. In the case that you have many datasets, such as when you constructed

a mesh using the multi-grid nesting approach, you can pass a cell-array of *geodata* instances to this method where interpolated data will be replaced in areas of overlap with the last most entry.

- `bound_con_int`

  `msh_obj = bound_con_int(msh_obj,MaxValency)`

  This method accepts a `msh_obj` and bounds the vertex-to-vertex connectivity (otherwise known as valency) to

  `MaxValency`

  using the method documented in Massey [2015].

- `CalcCFL`

  `CFL = CalcCFL(msh_obj,dt)`

  This method accepts a `msh_obj` and a desired simulatable *dt* and outputs a vector *np* x 1 which contains the CFL condition at each vertex of the mesh assuming the flow is determined by the shallow water wave speed plus the orbital velocity nearshore (see section). If you don't pass it a *dt* it will output the theoretical maximum possible stable *dt* in the domain given the mesh size.

- `CheckTimestep`

  `msh_obj = CheckTimestep(msh_obj,dt,NoIt)`

  This method takes as input a `msh_obj` and a desired *dt* and locally decimates vertices that lead to CFL violations giving back an updated `msh_obj`. It operates only in the connected elements to the violating vertices and will not alter any other data. This method is iterative in nature and will only exit until either the desired CFL condition is met or when the number of iterations (NoIt) is exhausted. This enables the mesh to be time marched with the desired *dt without* instabilities. **This method will delete your boundaries, but they can easily be recreated with** `makens`

- `renum`

  `msh_obj = renum(msh_obj)]`

  This method takes as input a `msh_obj` and renumbers the numbering of vertices using Reverse Cuthill Mckee (RCM) algorithm so to minimize the bandwidth of the finite element coefficient matrix. This routine will renumber all nodal attributes and the boundaries, but requires you recreate the fort.15 control file again after renumbering.

- `makens`

  `msh_obj = makens(msh_obj,type,dir)`

  This method takes as input a `msh_obj` and adds nodestring boundary conditions to it. It categorizes the boundaries as either islands or the outer boundary in which the latter is further broken down into either mainland boundaries or ocean boundaries.

  The following `type`(s) are available:

1. `'auto'`

   Applies all island nodestring boundary conditions, and automatically breaks the outer boundary into mainland and ocean boundary nodestring boundary conditions. This method relies on supplying a *geodata* class as the third argument (in place of `dir`) in order to split up the outer boundary. It assumes a default minimum length and depth below sea level to classify a boundary segment as an ocean boundary, but you can change. CHECK YOUR BOUNDARIES WITH the command plot(m,'bd',0).

2. `'islands'`

   Applies all island nodestring boundary conditions

3. `'outer'`

   Manually break up the outer boundary into mainland and ocean nodestring boundary conditions by entering data cursor mode. Need to supply a direction of travel (`dir`) as 0 (anticlockwise) or 1 (clockwise). **Do one nodestring at a time**

4. `'weirs'`

   Applies a weir boundary condition for all inputted weirs to the `geodata` class constructor that was used to build the mesh. For each weir, it will ask you to enter a crestline height. Currently, the software only supports constant crestline heights for each weir.

- `plus operator` The plus (+) operator has been overloaded to facilitate automatic merging of partially or fully overlapping triangulations represented each as msh classes. The user simply adds the two msh class objects together setting it equal to the name of the combined triangulation represented also a msh class object.

  ```
  msh_obj = msh_obj1 + msh_obj2
  ```

  Note the merging process is not communicative since the contents from the first msh object will be retained in areas of overlap. In other words the contents of the second msh object will be overwritten in areas of overlap with the first msh object. This scheme uses MATLAB's Bowyer-Watson algorithm to incrementally add the triangulations together and then attempts to clean invalid and degenerate elements that may arise due to this operation. This method works best when each of the two meshes where they are being merged are fairly straight. Thus, it can be advantageous to strategically build your domains so that the two domains slightly overlap in an area **away** from the shoreline. Also note that the mesh merging process will preserve constraints in each of the msh objects that are merged together.

- `GatherStationsFromKML.m`

  ```
  GatherStationsFromKML(msh_obj,kmlfile,ofname,VARARGIN)
  ```

  Given a `msh_obj` and a KML file downloaded from our tidal database [Pringle, 2017], this method extracts all the tidal observations that fall within the extents of the mesh's boundaries and then saves them as a .csv file for later use with other codes (see `ValidateTides.m` below). The input argument VARARGIN controls what source of stations are to be retained. More information on this method can be returned by typing `help GatherStationsFromKML`.

- `minus operator` Given two overlapping triangulations described in msh objects obj1 and obj2, remove the triangles in obj1 with centroids that are in the boundary of obj2 and returns this in a new msh object instance. Here the notion of "in" is defined as being with the polygon that is the boundary of obj2.

```
msh_obj3 = msh_obj2 - msh_obj1
```

Note: you cannot subtract and then add back two meshes and achieve the original mesh.

- In the following list we list the allowable parameters for 'name' in this call: plot(fh_obj,'name','proj'):
- Note that the argument proj determines whether or not the data is plot in a projected coordinate system or not.

7.2 edgefx: the mesh size function class

7.2.1 Properties

- dis: the decimal percent the edgelength changes in space.
- wl: the number of triangles per wavelength of M_2 semi-diurnal tidal constituent.
- slp: a non-dimensional number directly proportional to the number of triangles per bathymetric slope and inversely proportional to the bathymetric depth.
- fl: the low and high pass Gaussian moving-average filter bounds in meters.
- g: max allowable triangle-to-triangle transition rate in the mesh.
- dt: theoretical stable timestep given Courant number ¡ 0.5.
- fs/R: number of triangles to resolve a feature width.
- ch: A non-dimensional number that represents how the resolution nearby channels scales with bathymetry.
- F: a MATLAB gridded interpolant in WGS84 that contains the minimum mesh size of all supplied mesh size functions.

7.2.2 Methods

- disfx: build linear distance mesh size function.
- featfx: build feature size mesh size function.
- wlfx: build wavelengh mesh size function.
- slpfx: build the slope mesh size function.
- chfx: build the channel mesh size function.
- finalize: take the minimum of all used mesh size functions, grade the mesh size function, and optionally, enforce the CFL condition.
- In the following list we list the allowable parameters for 'name' in this call: plot(fh_obj,'name'):
  - dis: plot the distance mesh size function.
  - fs: plot the feature size mesh size function.

- wl: plot the wavelength mesh size function.
- ch: plot the channel mesh size function.
- no name given: plot the minimum mesh size function.

7.3 meshgen

7.3.1 Properties

- bou (**required input**): A geodata class object.
- ef (**required input**): An edgefx class object.
- grd (output): A msh_obj that contains the triangulation from the mesh generator.
- p: An array of x,y coordinates representing points that you would like the mesh generator to start from rather than from a random seeding of the meshing domain.
- nscreen: How frequently to output mesh information to screen.
- plot_on: Show the triangulation as it is incrementally modified.
- dj_cutoff: In decimal percent (0 by default). Removes disjoint components of mesh that are smaller than dj_cutoff percent.
- cleanup: Either 1 default on or 0 to turn off topological cleanup routine.
- pfix: An array of x,y coordinates representing vertices you do not want to move in the mesh.
- egfix: An array of indices that map into pfix, which represent connections between pfix. In other words, edges that you want to fix in the mesh.

7.3.2 Methods

- mshgen: class constructor for meshgen. Requires you pass an edgefx class instance with your various mesh size options and a geodata class that describes your boundary. These are the only two required arguments all other arguments are optional by default (e.g., plot_on, nscreen).
- build: this method calls the modified force equilibrium mesh generation algorithm with the options contained in the edgefx class and the boundary described in the geodata class.

7.4 msh

7.4.1 Properties

7.4.2 Methods

clean: Applies the topological cleaning routine described in Section 4.3 on the populated msh class. Note that this method considers fixed points *pfix* and will not edit these points unless they alter the traversability of the mesh's boundary.

References

Randolph E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*. Society for Industrial and Applied Mathematics, 1 1998. ISBN 978-0-89871-409-8. doi: 10.1137/1.9780898719635. URL http://epubs.siam.org/doi/book/10.1137/1.9780898719635.

P Brufau, P Garcã, and M E Vã. Zero mass error using unsteady wetting-drying conditions in shallow flows over dry irregular topography. *International Journal for Numerical Methods in Fluids*, 45:1047–1082, 2004. doi: 10.1002/fld.729. URL https://doi.org/10.1002/fld.729.

Darren Engwirda. *Locally optimal Delaunay-refinement and optimisation-based mesh generation*. PhD thesis, University of Sydney, 2014. URL http://hdl.handle.net/2123/13148.

GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2*. Open Source Geospatial Foundation, 2017. URL http://grass.osgeo.org.

Jonas Koko. A Matlab mesh generator for the two-dimensional finite element method. *Applied Mathematics and Computation*, 250:650–664, 2015. ISSN 00963003. doi: 10.1016/j.amc.2014.11.009. URL http://dx.doi.org/10.1016/j.amc.2014.11.009.

Florent Lyard, Fabien Lefevre, Thierry Letellier, and Olivier Francis. Modelling the global ocean tides: modern insights from FES2004. *Ocean Dynamics*, 56(5-6):394–415, 12 2006. ISSN 1616-7341. doi: 10.1007/s10236-006-0086-x. URL http://link.springer.com/10.1007/s10236-006-0086-x.

T. Chris Massey. Locally constrained nodal connectivity refinement procedures for unstructured triangular finite element meshes. *Engineering with Computers*, 31(2):375–386, Apr 2015. ISSN 1435-5663. doi: 10.1007/s00366-014-0357-y. URL https://doi.org/10.1007/s00366-014-0357-y.

Per Olof Persson. Mesh size functions for implicit geometries and PDE-based gradient limiting. *Engineering with Computers*, 22(2):95–109, 2006. ISSN 01770667. doi: 10.1007/s00366-006-0014-1.

Per-olof Persson and Gilbert Strang. A Simple Mesh Generator in MATLAB. *SIAM Rev.*, 46: 2004, 2004. doi: 10.1137/S0036144503429121. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.7905.

William J. Pringle. Major tidal constituents for the Indian Ocean and Western Pacific Basin, 2017. URL http://dx.doi.org/10.17632/tjyjn56jbf.1.

William J. Pringle, D. Wirasaet, and Joannes J Westerink. Modifications to Internal Tide Conversion Parameterizations and Implementation into Barotropic Ocean Models. *MethodsX*, page submitted.

William J. Pringle, Damrongsak Wirasaet, Andika Suhardjo, Jessica Meixner, Joannes J. Westerink, Andrew B. Kennedy, and Shangyao Nong. Finite-Element Barotropic Model for the Indian and Western Pacific Oceans: Tidal Model-Data Comparisons and Sensitivities. *Ocean Modelling*, 2018. ISSN 14635003. doi: 10.1016/j.ocemod.2018.07.003. URL https://linkinghub.elsevier.com/retrieve/pii/S146350031830026X.

J. J. Westerink, R. A. Luettich, A. M. Baptists, N. W. Scheffner, and P. Farrar. Tide and Storm Surge Predictions Using Finite Element Model. *Journal of Hydraulic Engineering*, 118(10):1373–1390, 10 1992. ISSN 0733-9429. doi: 10.1061/(ASCE)0733-9429(1992)118:10(1373). URL http://ascelibrary.org/doi/10.1061/%28ASCE%290733-9429%281992%29118%3A10%281373%29.

D. Wirasaet, S.R. Brus, C.E. Michoski, E.J. Kubatko, J.J. Westerink, and C. Dawson. Artificial boundary layers in discontinuous Galerkin solutions to shallow water equations in channels. *Journal of Computational Physics*, 299:597–612, 10 2015. ISSN 00219991. doi: 10.1016/j.jcp.2015.07.015. URL http://www.sciencedirect.com/science/article/pii/S002199911500460X.