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

28 February 2018

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 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.

The mesh generation is accomplished by using documented [1] and original improvements to the seminal force equilibrium algorithm, *DistMesh2D* [2]. 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 *DistMesh2D*. However, in contrast to *DistMesh2D* the mesh generator class has been tuned to converge quickly for complex polygonal regions and highly heterogeneous edgelength functions typically associated with meshes for geophysical shallow water flows.

In automated mesh generation, the edgelength function determines how resolution is distributed in space. While the topic of mesh generation is rich and well-studied, research on the impact of mesh resolution on the shallow-water equations 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 simple through abstraction and overloading. These properties reduce the complexity of function calls and the number of lines the user needs to type.

There are four classes: `geodata`, `msh`, `meshgen`, and `edgefx`. 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. Help documentation can be accessed by typing for example:

```
help geodata
```

And all the methods of each class can be inspected by typing

```
methods('classname')
```

Here we list some of the basic requirements used for this software.

- MATLAB (tested on versions post 2015a up to 2017b) plus the following toolboxes:
  - paid: mapping toolbox

  - free: m_map toolbox v1.4 (https://www.eoas.ubc.ca/~rich/map.html)
  - free: ann_wrapper (http://www.wisdom.weizmann.ac.il/~bagon/matlab.html)
  - free: MEX compiler

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

The m_map and ann_wrapper packages must be placed inside the utilities folder.

2.1 Installation

Unzip the contents sent to you. Navigate to the 'OceanMesh2D/utilities' directory. Download and place the m_map and ann_wrapper packages in this directory. Still within the utilities directory, the user must execute 'ann_class_compile.m'. This script compiles some of the MEX files associated with the ann_wrapper package used throughout the code. Be sure to add the 'OceanMesh2D/utilities' and 'OceanMesh2D/datasets' directories and sub-directories to your MATLAB path, e.g., 'addpath(genpath('utilities/'))'.

2.2 Mesh

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 in the Continuous Galerkin Finite Element framework. A *valid* mesh for use with the Continuous Glakerin 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 [3]:

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

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.

A restriction on the smoothness or the grade g of the triangulation is applied. Sharp gradients in triangular resolution may produce large numerical errors in these regions. Smooth transitions constrain the size of the error so that the solution does not noticeably degrade. We generally find that setting $0.2 < g < 0.3$ produces reasonable solutions.

An empty mesh for use within *OceanMesh2D* can be constructed by typing: `mesh = msh()` or, by populating it with a ADCIRC compliant fort.14 file,
`mesh = msh('fname.14')`.

2.3 Domain

The region to be meshed is identified using a rectangular box referred to as a bounding box (bbox) with a constant grid spacing $h0$. The variable $h0$ represents the minimum grid spacing or edgelength that is desired in your mesh.

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

where x and y are in horizontal WGS84 coordinates.

2.4 Digital Elevation Model

A digital elevation model (DEM) is a structured grid that contains x,y,z data at each grid point. Here the coordinate z represents the height in meters *above* a geodetic datum (e.g., mean sea level/MSL). 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, and channels). 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. The correct NetCDF format for our software can be achieved by typing (regardless if the file is already in NetCDF format):

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

This results in the variables inside the Netcdf being compliant and sized correctly for it to work with the software. 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.

2.5 Boundary

Features in *OceanMesh2D* are represented using polygons. A polygon is a 2-tuple of x,y coordinates with features separated by a NaN. For use within the ocean modeling framework, the boundary provided to *OceanMesh2D* can be a shapefile or a NaN-delimited vector.

One simple way to obtain a shapefile of the coastline for use within *OceanMesh* is to perform the following operation on your DEM dataset:

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

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 to the `geodata` class constructor, and bbox:

```
data = geodata('shp',filename.shp,'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.

3 Edgelength functions

Here we discuss how the mesh resolution is distributed in 2D space. All edgelength functions are encapsulated within the `edgefx` class. After constructing the `edgefx` class with arguments, the minimum of all edgelength functions is computed to form the final edgelength function.

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. All edgelength functions are WSG84 decimal degrees.

3.1 Distance

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:
   $$f_{h_{dis}} = h0 + \alpha_d d \tag{2}$$
   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',αd)
   ```

   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** $f_{h_{dis}}$.
   $$f_{h_{fs}} = \frac{W}{R} \tag{3}$$
   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 [1] 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 (3) becomes
   $$f_{h_{fs}} = \frac{2(d_{MA} + d)}{R} \tag{4}$$
   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)
   ```

3.2 Wavelength

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 [4].

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:

$$f_{h_{wl}} = \frac{\lambda_{M2}}{\alpha_{wl}} \tag{5}$$
$$f_{h_{wl}} = \frac{T_{M2}}{\alpha_{wl}}\sqrt{gH} \tag{6}$$

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. The wavelength function is converted to decimal degrees by assuming a latitude of 0°. To enable this option:

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

3.3 Slope

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:

$$f_{h_{slp}} = \frac{2\pi}{\alpha_{slp}} \frac{h}{|\nabla h^*|} \tag{7}$$

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 [5] 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 https://www.mathworks.com/matlabcentral/fileexchange/61003-filt2-2d-geospatial-data-filter, 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. Alternatively, a low-pass filter length is automatically deduced based on the λ_{M2} at a depth of 100 m and the α_{slp} parameter:

$$fl_{low} = \frac{\lambda_{M2}(h = 100)}{\alpha_{slp}}, \qquad fl_{high} = 0 \tag{8}$$

this has the physical meaning that features larger than a $1/\alpha_{slp}$ fraction of the M2 wavelength (based on the 100 m depth level) should be captured, e.g. $fl_{low} = 46.7$ km for $\alpha_{slp} = 30$. Furthermore, $f_{h_{slp}}$ can become unnecessarily large nearshore and override the feature size variations, thus we set $f_{h_{slp}} = 0$ in depths less than 100 m. The decision to cutoff the slope parameter and deduce the filter length at 100 m depth is driven by the expectation that shelf breaks begin just beyond 100 m and any pertinent topographic slope-based features shallower than this should be captured by the channel finder algorithm in §3.4. The slope edgelength function is enabled by typing:

```
hh = edgefx('geodata',geodata_obj,'slp',αslp,'fl',[fllow,flhigh])
```

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.

3.4 Channels (ch)

There are often both man-made dredged channels and submerged river basins (e.g. Hudson river valley) in the nearshore region that can alter the near shore circulation. For instance, in submerged channels in the ocean, the bottom friction is locally reduced due to their deeper depth relative to neighboring regions. This acts to enhance the flow along the centerline of the channel. Overland, there are channel networks where rivers drain into the ocean basins. The overland channel networks are typically the first locations where flooding occurs during storm surge events. Considering this, it is useful to resolve overland and submerged channels in the mesh in order to simulate the correct movement of water and adequately resolve the overland regions were flooding would likely occur.

The details of the channel finder are beyond the scope of the user guide but a curious reader can read OUR PAPER CITATION on it. Once the algorithm has calculated the channel network, it creates a circular region around each channel point and assigns a percentile of the slope edge function in this region. The circular region is determined by assuming the channel has a v-shaped profile with the bathymetric depth at that particular channel's point and an angle of reslope of 60^o.

Essentially, we are incorporating channels into the mesh that have steep banks with higher resolution. This reduces the number of falsely identified channels if one were to solely use an upslope area threshold to identify them. Since we are using the slope edge function to determine mesh resolution here, the only argument the user needs to specify is α_{ch}, which is the percentile of the neighboring slope edge function to be used to place resolution along the channel points. The user can enable the channel function by typing::

```
hh = edgefx('geodata',geodata_obj,'ch',αch)
```

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:

$$f_h(\boldsymbol{x_j}) \leq f_h(\boldsymbol{x_i}) + \alpha_g||\boldsymbol{x_i} - \boldsymbol{x_j}|| \qquad (9)$$

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 [6] and later adapted by [7]. 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 [6, 7].

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 to the `edgefx` class.

```
fh = edgefx(varargin,'g',αg)
```

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 [8]:

$$Cr = \frac{(|\boldsymbol{u}| + \sqrt{gH})\Delta t}{f_E} \tag{10}$$

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 [9].

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:

$$|\boldsymbol{u}| = \eta\sqrt{\frac{g}{h}} \tag{11}$$

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 (10) to find the minimum edgelength, f_h possible for a given h and Δt, based on some value of $Cr \leq 1$ (such as $Cr = 0.8$, which provides a 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 (11) 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 (10) at the coast is used to limit f_h everywhere else. This is determined by:

$$\Delta t = \min\left[\frac{f_{h_d} Cr}{|\boldsymbol{u}| + \sqrt{gH}}\right] \tag{12}$$

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 (12) 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 §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 *Distmesh2D* program. Property number 4, referring to ensuring that there are only two traversal paths along the mesh boundary, is handled through the `fix_bad_edges_and_mesh` function described below.

The `fix_bad_edges_and_mesh` 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 `fix_bad_edges_and_mesh` 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 removes 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

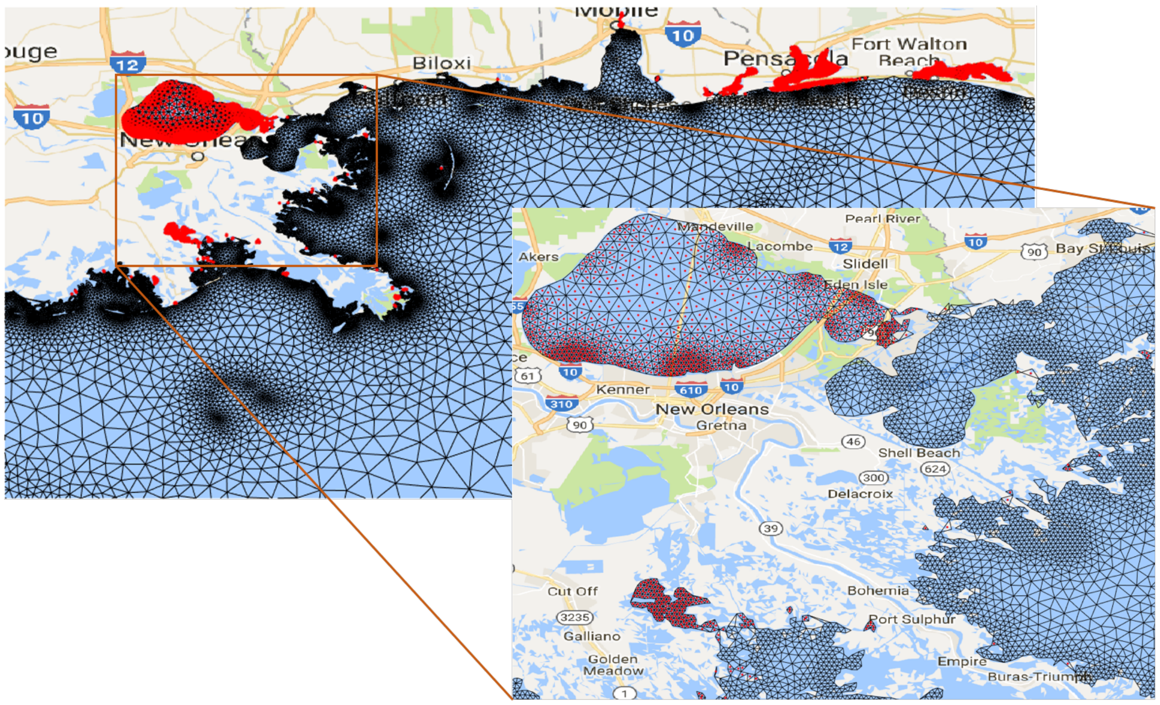


Figure 1: Red dots indicate the elements that are deleted on the first pass of the `delete_exterior_elements` sub-function contained within the `fix_bad_edges_and_mesh` routine. As shown on the blown-up graphic, Lake Pontchartrain in New Orleans is disconnected from the main grid and represents a very small portion of the overall graph (the mesh spans the entire U.S. East and Gulf Coasts), so it is deleted.

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 fraction, μ_{co} of the total graph, which is set to be 0.01 (1%) by default. The μ_{co} fraction 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. Thus, a user may need to set μ_{co} as a parameter in the `meshgen` class depending on the requirements and conditions:

```
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 1. 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 2 along the Pensacola Beach barrier island. Because the barrier

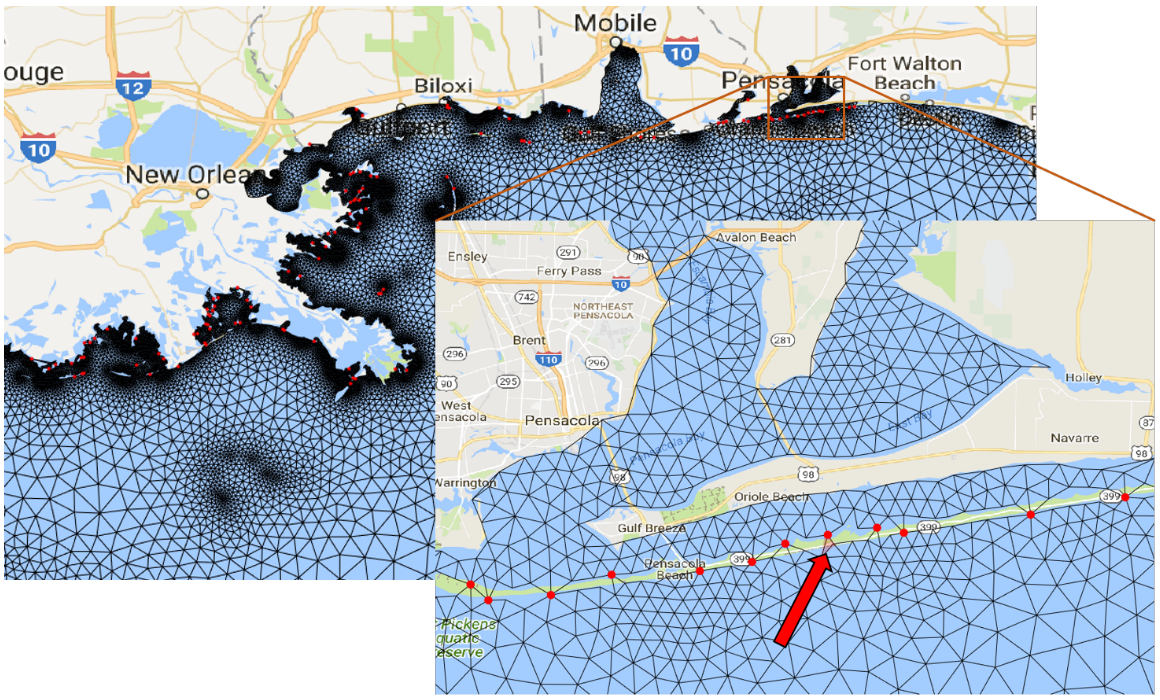


Figure 2: Red dots indicate the elements that are deleted on the first pass of the `delete_interior_elements` sub-function contained within the `fix_bad_edges_and_mesh` routine. As shown on the blown-up graphic, a number of nodes along the thin Pensacola Beach barrier island are connected to more than 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.

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 2 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 menthat have two boundary edges in which there are many examples contained within Figure 2. 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 un-traversable after deleting one of the elements, but `fix_bad_edges_and_mesh` iteratively calls `delete_interior_elements` until this condition is met.

5 Example of use

Mesh generation is accomplished with a standalone class that technically only requires a boundary, but has a number of optionally useful input arguments. 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(varargin)
```

Then the user calls the mesh generation algorithm by typing:

```
mesh = build(mesh)
```

or

```
mesh = mesh.build()
```

Which displays the mesh (if `plot_on= 1`) as it is incrementally modified at `nscreen` intervals. After convergence or reaching the maximum number of iterations `itmax`, it produces a `mshgen` class object in which the user can access the triangulation's points and triangle table (mesh.grd) and all the options the mesh was built with. There are a 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 shapefile has coverage in that region**. It's recommended that the user create a 'datasets 'directory in the rootdir of where they installed this software and place their 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 `msh.build`! In just a few iterations you will see the mesh start to improve.

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 §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

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.

```
%% 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.shp';
gdat = geodata('shp',coastline,...
               'bbox',bbox,...
               'h0',min_res);
% 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...
msh = meshgen('h0',min_el,'bbox',bbox,'ef',fh,...
              'bou',gdat,'nscreen',1,'plot_on',1,'itmax',100);
% now build the mesh with your options and the edge function.
msh = msh.build;

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

Figure 3: 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 10 minutes.

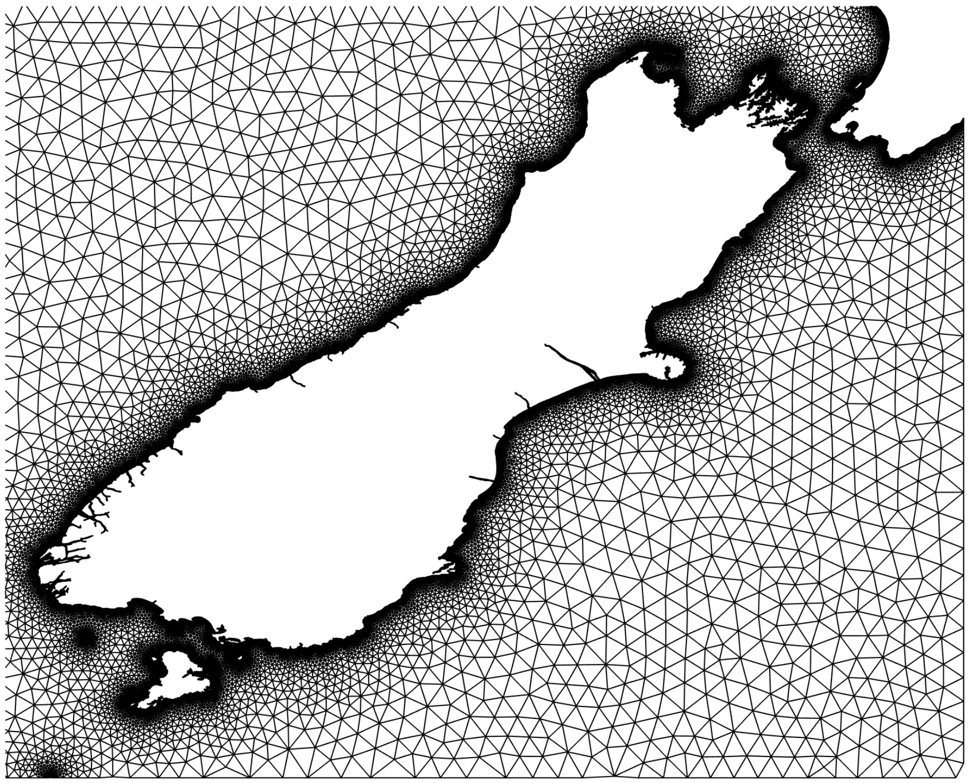


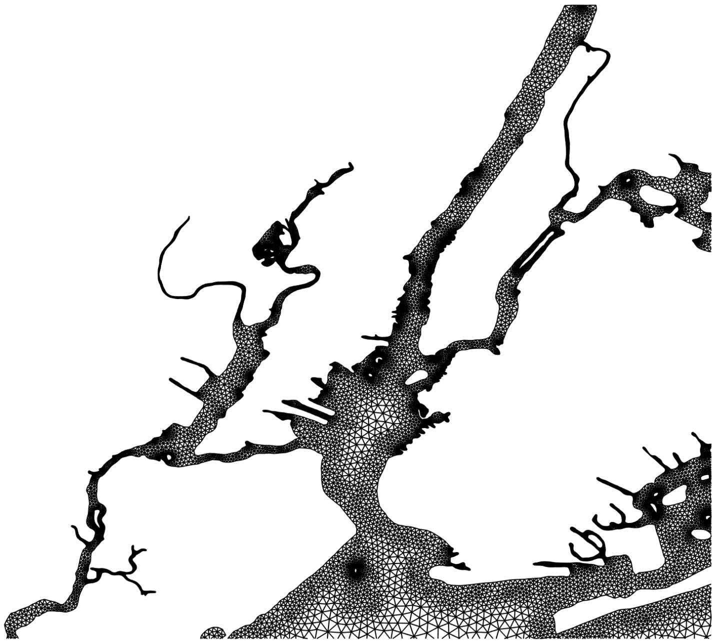
Figure 4: Result: Convergence after 30 iterations in 4 wall-clock minutes on PC, 51,812 vertices, 90,869 elements, $\overline{q_E} = 0.958$

5.2 High fidelity

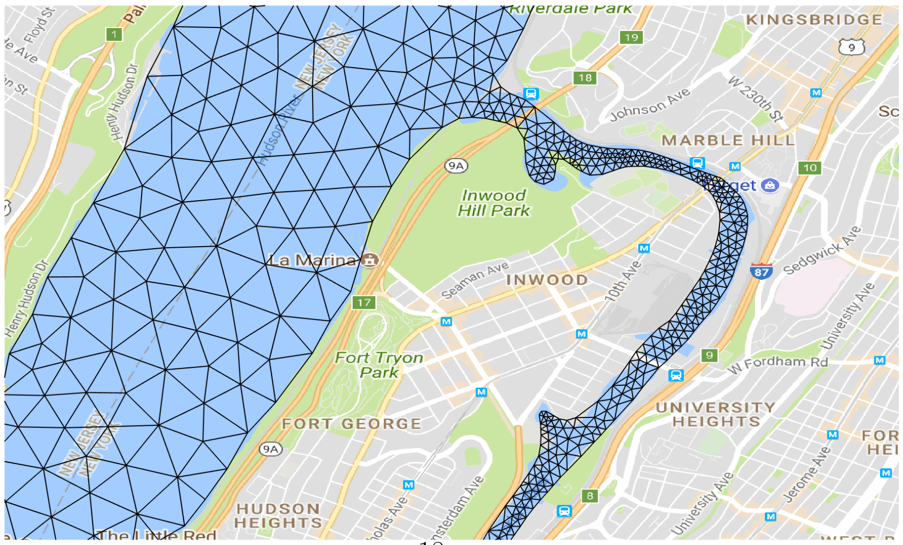
Using higher resolution digital elevation data to create the coastline boundary will dramatically improve the results. See the example below that uses the shapefile “combined2.shp” and the “combined2_subset.nc” DEM, which was created by interpolating SRTM15+, the Coastal Relief Model and then the Post-Sandy NCEI DEMs on a 90-m uniform grid using the software gdal. This example illustrates the ability to capture small regions with high fidelity.

```
%% 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 = 'combined2.shp';
dem       = 'combined2_subset.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...
msh = meshgen('h0',min_el,'bbox',bbox,'ef',fh,...
              'bou',gdat,'nscreen',1,'plot_on',1,'itmax',100);
% now build the mesh with your options and the edge function.
msh = msh.build;

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

Figure 5: 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 15 minutes.



(a)



(b)

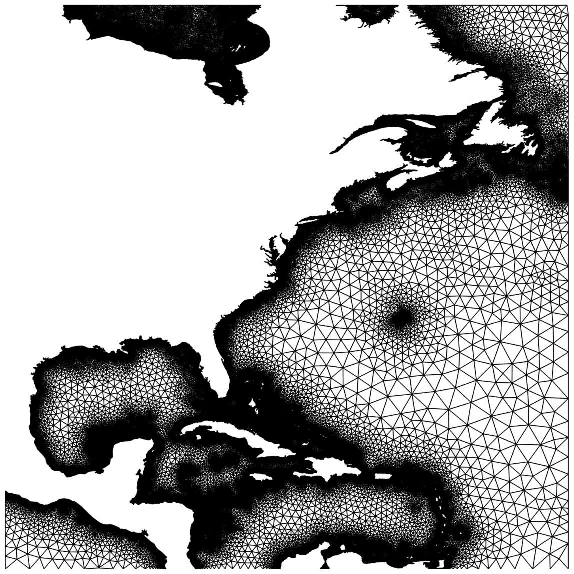
Figure 6: Result: Convergence after 40 iterations in approximately 15 minutes on a PC. There are

5.3 Large domains

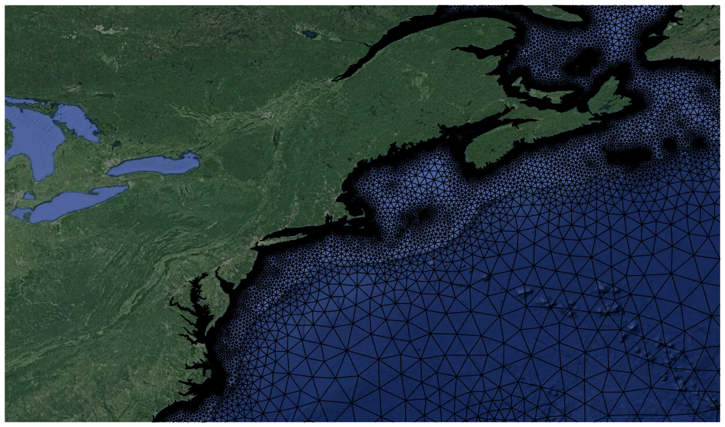
This example illustrates the size of domains (50° by 50°) the program can handle in relatively short times (15-20 minutes) on a personal computer. Reading the coastline and prepping the edgelength functions can take about half the total time. Once prepped, convergence in msh.build can be quick.

```matlab
%% STEP 1: set mesh extents and set parameters for mesh. The greater US East Coast ...
    and Gulf of Mexico region
bbox      = [-100 -50;  % lon_min lon_max
               10  60];  % lat_min lat_max
min_el    = 1e3;        % minimum resolution in meters.
max_el    = inf;        % maximum resolution in meters.
max_el_ns = 5e3;        % maximum resolution nearshore.
wl        = 60;         % 60 elements resolve M2 wavelength.
dt        = 5;          % 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.shp';
dem       = 'topo15_compressed.nc';
gdat = geodata('shp',coastline,...
               'dem',dem,...
               'bbox',bbox,...
               'h0',min_res);
% 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,...
            '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...
msh = meshgen('h0',min_el,'bbox',bbox,'ef',fh,...
                  'bou',gdat,'nscreen',1,'plot_on',1,'itmax',100);
% now build the mesh with your options and the edge function.
msh = msh.build;

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

Figure 7: Similar to example two but also uses DEM data to build the wavelength function edge function and limit the CFL. Note how we set the max_el to inf allowing the wavelength to control the resolution in the deep ocean. Estimated time to completion: 20 minutes.



(a)



(b)

Figure 8: Desired result: Convergence after 20 iterations. There are 486,299 vertices, 905,605 elements, $\overline{q_E} = 0.967$. Notice in (b) how the wavelength parameter affects mesh resolution over the Georges Bank.

6 Post-processing functions

6.1 `msh.calcCFL`

```
CFL = CalcCFL(msh_obj,dt)
```

This method accepts a `msh_obj` and a desired *dt* and outputs a vector *np* x 1 CFL 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.

6.2 `msh.checkTimestep`

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

This method takes as input a `msh_obj` and a desired *dt* and iteratively decimates vertices and triangles that result in CFL violations giving back an updated `msh_obj`. This enables the mesh to be time march with the desired *dt* *without* instabilities. **This method will delete your boundaries! They can be recreated with `msh.makeNS`**

6.3 `msh.renum`

```
msh_obj = renum(msh_obj)]
```

This method takes as input a `msh_obj` and renumbers the vertices using Reverse Cuthill Mckee algorithm so to minimize the bandwidth of the mass matrix. **The Reverse Cuthill Mckee algorithm requires the graph toolbox to use!**.

6.4 `msh.makeNs`

```
msh_obj = makeNs(msh_obj,dir,'Islands','Outer',)
```

This method takes as input a `msh_obj` and adds either island or outer boundary (either mainland boundaries or ocean boundaries) nodestrings to an existing `msh_obj`. If you would like to add island nodestrings, you would use the 'Islands' argument to the method call. If you would like to add outer boundaries, you would use the 'Outer' argument to the method call. **Do one at a time**. The argument 'dir' is either 0 or 1 and indicates the traversal direction (0 for clockwise and 1 for counter clockwise) and only matters for determining the traversal path along outer boundaries.

6.5 + 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.

6.6 Example of workflow

When creating high fidelity triangulations that are highly multiscale in nature, it is necessary for the sake of memory to partition the domain into partially overlapping rectangular boxes. Besides relaxing the memory requirement, this also allows one to script the mesh generation process and possibly trivially parallelize it. See the script below.

After the meshes are created, we will merge them together automatically using the plus operator as seen in the script below.

Once the final merged mesh is created, it is then necessary to create the input files for a simulation and determine the stable timestep. 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 and then ensure the mesh will be simulatable with a 2 second timestep. Conversely, one could also run CalCFL for a 2 second timestep and then see if the maximum CR was under 0.75. Following this, we create the nodal attributes (internal tide friction, primitive weighting in the continuity equation, self-attraction and loading), specify the boundary conditions, 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 triangulation.

```matlab
clearvars; close all;
% Post-processing script to take a msh and create all the necessary
% input files for an ADCIRC simulation
%%
MSHFILE   = 'step1.14' ;          % filename of mesh
DT        = 2 ;                   % goal timestep
TS        = '01-Aug-2012 00:00' ; % start time of simulation
TE        = '31-Nov-2012 00:00' ; % end time of simulation
%%
DEMFILE   = 'combined_wpr_wphilly.nc' ;
BUOYFILE  = 'Gridded_N_values.mat';
TPXO9     = 'tpxo9_netcdf/h_tpxo9.v1.nc';
CONST     = 'major8' ;
%%
m = msh(MSHFILE) ;

m = GridData(DEMFILE,m) ;

m = CheckTimestep(m,DT) ;

m = renum(m) ;

m = makens(m,'islands') ;

m = makens(m,'outer',1) ;

m = makens(m,'outer',0) ;

m = Calc_tau0(m) ;

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 ) ;
```

7 Appendix: Class Prototypes

Here will list the class prototypes and a brief description of each method.

7.1 `edgefx`

```
edgefx(h0,bbox,
```

7.2 `geodata`

7.3 `msh`

7.4 `mshgen`

References

[1] J. Koko, “A Matlab mesh generator for the two-dimensional finite element method,” *Applied Mathematics and Computation*, vol. 250, pp. 650–664, 2015.

[2] P.-o. Persson and G. Strang, “A Simple Mesh Generator in MATLAB,” *SIAM Rev.*, vol. 46, p. 2004, 2004.

[3] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*. Society for Industrial and Applied Mathematics, 1 1998.

[4] 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*, vol. 118, pp. 1373–1390, 10 1992.

[5] F. Lyard, F. Lefevre, T. Letellier, and O. Francis, “Modelling the global ocean tides: modern insights from FES2004,” *Ocean Dynamics*, vol. 56, pp. 394–415, 12 2006.

[6] P. O. Persson, “Mesh size functions for implicit geometries and PDE-based gradient limiting,” *Engineering with Computers*, vol. 22, no. 2, pp. 95–109, 2006.

[7] D. Engwirda, *Locally optimal Delaunay-refinement and optimisation-based mesh generation*. PhD thesis, University of Sydney, 2014.

[8] D. Wirasaet, S. Brus, C. Michoski, E. Kubatko, J. Westerink, and C. Dawson, “Artificial boundary layers in discontinuous Galerkin solutions to shallow water equations in channels,” *Journal of Computational Physics*, vol. 299, pp. 597–612, 10 2015.

[9] P. Brufau, P. Garcã, and M. E. Vã, “Zero mass error using unsteady wetting – drying conditions in shallow ows over dry irregular topography,” *International Journal for Numerical Methods in Fluids*, vol. 1082, no. May, pp. 1047–1082, 2004.