Response to Reviewer 1

Dear Reviewer,

Structurally, the paper will benefit from extensive modifications.

Extensive modifications were made to the paper to improve its presentation: sections were restructured and then edited to better reflect their new order. In many sections, the text was trimmed to be made more terse and precise, and thoroughly checked for grammatical mistakes. Overly formulaic descriptions were simplified and replaced with visual depictions of our algorithmic contributions through additional figures. The description of reproducibility in the context of the software was removed *in lieu* of the concepts of automation and efficiency.

Please see the tracked-changes manuscript highlighting line-by-line changes that we have made to the text.

In particular, the manuscript would benefit from presenting the test cases graphically. The three areas summarised by table 2 could be introduced more effectively through a figure with multiple panels showing the regions, in cartographic form and at various scales. The figure should also indicate data sources in different colours. Such a figure should be placed early in the paper to make it more appealing, and its discussion can be placed in a separate small section, outlining the cases, their location and reasons for selection. The figure would directly showcase the capabilities of OceanMesh2D, regarding handling multiple data sources. Also, the figure will be an "anchor point" facilitating later discussions. Table 2 could thus only summarise the meshing parameters, Mesh quality and Iterations, making it smaller, easier to typeset, more digestible to the reader, and could be placed later in the paper.

A new figure (Figure 1) was created that presents the example problems used throughout the text in cartographic form, at various scales indicating the datasets used and the minimum mesh resolutions. This figure is cited throughout the text and now serves as an "anchor point". Further, columns from table 1 (was table 2) were removed to make it more digestible to the reader (specifically each example's domain extents, and the datasets that were used to create the example). A new subsection 1.1 "Example Problems" was created to explain the example problems and why they were selected for this work upfront.

The description of the software modules would also benefit from restructuring. Section 2 could be renamed "Architecture overview", as the term framework has a different meaning in computer science. While conceding that this reviewer is now focusing on semantics, an architecture overview section will enhance the broader description of the software modules and show how their design and mutual interaction was conceived to address the specific problems outlined in earlier sections.

The description of the software modules was restructured and Section 2 was

renamed to "Architecture Overview". In this section, the verbiage describing object-oriented programming and how it is used were removed from the text, as the reviewer suggested. The notion of use of OOP programming producing reproducible workflows was removed instead for: *"..the use of OOP leads to automation and promotes the usage of efficient workflows."*

The details of specific classes should be placed in a section named "Component design" where each class is presented in a separate subsection. Note, currently the msh class is described in section 2, while all other classes are described in a separate section each. The authors could consider placing section 5.2 in "Architecture overview" or "Component design" as it repeats points made in those sections. In general, the authors are encouraged to revise the manuscript and avoid making the same arguments multiple times. Also, the meshgen class should be presented first, rather than last. The meshgen class is the core of the package. Therefore, it seems appropriate to describe it first, followed by the description of other classes. Thus the need for sections to refer to later sections is eliminated. The only exception to this is section 5.2, as discussed above.

A new section called "Component Design" was created in which each of the four classes are now presented as subsections. The order of the classes have been modified to present, first, the *meshgen* class, then the *geodata*, then the *edgefx* and finally a new section 3.4 termed "Mesh container: *msh* class".

The new location of the *meshgen* section, at the beginning of the component design section (Section 3.1), eliminates the repetition of points and arguments in multiple places (e.g., high resolution structured DEMs are memory inefficient but useful to build mesh size functions) in the manuscript and better highlights the functionality of the software earlier on in the manuscript. The motivation, description, and overview of the mesh improvement strategies were made more concise and more general enabling other readers to copy and employ a similar mesh cleaning algorithm in their mesh generators. The multiscale meshing capability was moved from the end of the manuscript to the end of Section 3.1 (Section 3.1.4), and was heavily revised for concision and to more clearly indicate our algorithmic contributions to the *DistMesh2D* algorithm to enable this ability.

We have also decided to restructure the *edgefx* class section. A new section 3.3.5, which describes the possible combinations of mesh size functions in the software and how they are finalized is included. The section titled "Mesh size gradation" was renamed to "Mesh size transitions" and moved further to the end of Section 3.3 *edgefx* section (Section 3.3.6) as the size grading is performed after the other mesh size functions are constructed.

The mathematical description of the signed distance function (previously in Section 3.2) was removed. Instead, we have described these mathematical definitions in plain English.

Two further points relevant to the meshgen class description are: i) The way Algorithm 2 is presented could leave readers uninterested, as it clouds the algo-

rithm's aim with data assignment and other operators. The pseudo-algorithm presentation is better suited to the User Manual. A figure with multiple panels and a more straightforward representation of the algorithm will be more effective at making the same point. ii)

The previous presentation of Algorithm 2 in a psuedo-code format was removed because, as the reviewer pointed out, it clouded the contribution in an article-style document and was better suited for a user manual. A new multi-panel figure (Figure 4) was created instead to illustrate the application of the the mesh cleaning function visually.

In page 27 the description of various methods (or are those functions?) is also poorly presented. The identifiers of the methods could be listed in a table, while a couple of figures could showcase the problematic cases that are targeted by the functions. It seems substantial effort has been invested into developing the methods outlined in page 27, and the present description will leave readers uninterested.

Section 3.4, at the end of Component Design, was created to describe the *msh* class with a focus on three select methods (*plot*, *makens*, *interp*), using a multipanel figure (Figure 15). The table that had previously described all the mesh utilities and methods was removed and curious readers are redirected to the user guide to see the complete list of methods and utilities. A new paragraph was introduced in Section 3.4 to describe Figure 15. The authors believe the *msh* class is an important contribution as it standardizes the interaction with the mesh generator and helps improve the efficiency and efficacy of coastal mesh generation workflows.

Note that we describe the above as methods (rather than functions) because they are unique to the *msh* class and cannot be used with any other class or arbitrary input.

Far more important than the above structural problems, the paper makes statements on reproducibility that are not supported. Reproducibility is conflated with automation and replication. In the article, reproducibility seems to be defined as the ability to produce a given output, with given inputs exactly. However, the output of mesh-generation algorithms can vary, due to differences between various platforms. Seeding point coordinates with a random number generator is one such example. Unless the authors have built a system that eliminates such variations, claiming this type of reproducibility is invalid...

The statements that made claims about mesh reproduciblity throughout the text were moved and instead replaced with claims regarding automation and workflow efficiency. The concepts of research data management were not explored in this manuscript, but the concepts of automation and workflow efficiency were.

Also, the term reproducibility is today explored within the context of provenance and attribution, aiming to accurately define the processes and inputs leading to specific outputs, while disseminating outputs, processes and citing data, collectively termed Research Data Management (RDM). The software presented in the paper makes no explicit effort to distribute, trace provenance and make attributions. While a script will inherently contain information on the mesh generation process, such pointers are weak as they do not adequately describe the environment or other inputs and processes. Also, the authors state that the distribution of the script file with other supplementary data helps establish reproducibility (line 5 page 5). However, one of the main obstacles targeted by RDM practices is the distribution of such complementary data trough persistent and open platforms. If the authors are suggesting that Revision Control Systems (RCS) could be used to address reproducibility, then that is an effect of RCS, rather than the software described in the paper. Besides, even with an RCS reproducibility in the context of RDM is not immediately achieved, often requiring more steps. For example pointing to very large datasets that are impossible to distribute, due to size or licence restrictions.
It seems that the concept of reproducibility has been added to the manuscript as an afterthought, with citations peppered in the text, making the same point in multiple locations. However, this is not the best feature of the presented software, is not a novel idea and does not address well-known issues in RDM: The 2011 paper by Roger D. Peng (doi:10.1126/science.1213847) outlines the basic concepts of RDM as well as obstacles to the consistent use of RDM in academic research. In that article, three steps are proposed as an incremental approach to RDM. To their credit, the authors have provided the source code under a permissive licence on an appropriate repository. Steps two and three are characterised as more difficult in the article, but since 2011 various open, citable repository services have been launched, where uploading code and data in citable forms is possible. Zenodo and Figshare are popular examples. The 2012 report by the Royal Society (https://royalsociety.org/topicspolicy/projects/science-public-enterprise/report/) reiterates and expands many of the points made in Peng 2011. Section 4.3 will be of interest to the authors. In particular, the sections on Provenance and Citation in Appendix 2 are the basis for many of the reviewer comments. Since 2012, attempts at automating RDM procedures have appeared: In the context of mesh generation in geophysical domains, the articles by Jacobs et al. (2015, https://arxiv.org/abs/1509.04729) and Avdis et al. (2018, https://doi.org/10.1016/j.renene.2017.09.058) present RDM approaches integrated into GIS and mesh generation and will be of interest to the authors.

Additional text was added into the first paragraph of the discussion section to describe the limitations of our approach highlighting that reproducibility in terms of provenance attribution could be improved through more formal Research Data Management practices as highlighted in Avdis et al., (2018) and Candy and Pietrzak (2018).

Response to Reviewer 2

Dear Reviewer,

A new mesh generation library: OceanMesh2D is described, focusing on the construction of multi-scale unstructured triangulations for applications in coastal ocean modelling. Adapting the well-known DISTMESH algorithm (Persson and Strang), and building on top of other open-source contributions for various mesh-based and geo-spatial processing tasks, the authors present a MATLAB-based meshing library designed to automate the unstructured grid generation workflow for coastal ocean modelling configurations.
In addition to a description of their MATLAB-based implementation, the authors present a variety of mesh-resolution heuristics to control element size throughout the domain. As well as a number of existing resolution functions appropriate for coastal modelling (distance-to-coast, barotropic wave-length scaling, etc) a set of new metrics (Rossbyradius filtered bathymetric gradients, channel thalweg scaling, etc) are introduced — focusing on better resolution of various dynamical processes and/or topographic features in unstructured models.
While much useful information is contained in the paper, I am overall somewhat unsure what its focus is or should be. Currently, I feel the authors have provided a detailed description of their MATLAB-based implementation, with much specific discussion of various classes and routines to be found in the OceanMesh2D code-base. To me, this reads a little like a software user manual.
If the authors intend to focus on algorithmic innovations, I suggest that a higher level and more mathematically-focused description of the algorithms be presented. While detailed discussions of various MATLAB functionality and the availability of open source code may undoubtedly be useful to model users, I do not feel that algorithmic discussions need to be focused on any particular implementation, and that in fact to do so may diminish adoption and re-implementation by other authors. If algorithmic innovations are to be the focus of this paper, I suggest it may be necessary to better compare against (and demonstrate improvement over) existing coastal meshing strategies and packages — highlighting the impact of any new algorithmic techniques.

In the revised manuscript, we have restructured the order of the sections to highlight the key algorithmic contributions to the coastal mesh generation problem that we made:

1. Section 3.1 second paragraph highlights how our approach to mesh generation eliminates the need for shoreline approximation pre-processing. Instead the boundary is implicitly defined through the mesh size function.

2. The multiscale meshing technique has been moved to Section 3.1.4 (previously was in Section 5.2) is now better clarified as to how the *meshgen* class is modified to assist with memory management.

3. Instead of Algorithm 2, the post-processing mesh routine to obtain traversabil-

ity (*Make_Mesh_Boundaries_traversable*), was highlighted through a new Figure 4 illustrating the elements that are deleted in the process. Three additional mesh improvement strategies after mesh generation are also illustrated through a new Figure 5. These four strategies are now outlined in Section 3.1.3 more concisely than previously without the hard to follow Algorithmic pseudocode (previously Algorithm 2).

4. In a number of sections we have removed details that are more suited for a user guide.

These changes to the manuscript were made to encourage adoption and re-implementation by other authors, as the reviewer was concerned about. For example the four strategies employed in Section 3.1.3 could be used to improve the quality and validity of any existing coastal mesh without using the mesh generation capabilities contained in *OceanMesh2D*. Overall, the open-source nature of the package and algorithms help encourage adoption and this point is now stressed in the revised Discussion and Conclusions section.

However, compromising with Reviewer 1's suggestions we have decided to focus on visualization and concise descriptions, which we feel will be more digestible to our intended audience than a dense mathematical focus.

Please see our tracked-changes manuscript for the full line-by-line changes and restructuring to the manuscript.

I feel the discussion of mesh-resolution heuristics would be much enhanced by actual simulation results and comparisons. The authors have introduced a number of new mesh scaling functions based on, for example, filtered bathymetric gradients and channel thalweg resolution. While these ideas are interesting, and may be expected to improve model skill under certain conditions, it would be beneficial to prove this was actually the case in practice and to document the impact of mesh resolution selection and design on model output. Without studying the effect on model physics, I feel it is difficult to judge the performance or utility of any particular mesh resolution heuristic. It may be possible to undertake several multi-mesh comparison studies: demonstrating that simulations run on meshes generated using the new resolution heuristics compare more favourably with high resolution numerical studies or observational data.

We certainly agree that assessing the impact of these mesh resolution heuristics on model physics (and numerics) is important, but this is not the direction we would like to go with this manuscript. In fact, we are currently in preparation of a paper with the exact aim as you are suggesting here. The aim of this GMD manuscript is straightforwardly a "Model description" of our mesh generator software and how it can improve the model development process. We hope that the structural changes and improved clarity in the algorithmic contributions better shows off the model and its attributes.

Overview of Changes

1. The second paragraph of the introduction was revised to reflect colloquial statements about "code complexity".

2. A new section 1.1 called "Example Problems" was created describing the example problems used throughout the paper. A new Figure 1 was created illustrating the overview of the domain and the location of the example problems in cartographic form at multiple scales. This figure is referred back to throughout the text and eliminates the need for the old figures that had individually described the PRVI (Fig. 13) and JBAY (Fig. 4) examples previously.

3. Section 2 was renamed to "Architecture Overview" to reflect reviewer 1's suggestions and trimmed to avoid the repetition of points made throughout the manuscript and the description of object-oriented program. Further, in this section and throughout the manuscript the notion of reproducibilty was removed in favor of the concepts of automation and efficient workflows to reflect Reviewer 1's concern.

4. Section 3 was heavily restructured and called "Component Design". Section 3.1 now describes the mesh generator tools first; specifically the improvements to the *DistMesh2D* algorithm that together enable mesh generation for coastal ocean models. Overly dense and formulaic descriptions of the *DistMesh2D* algorithm were removed in favor of more easily digestible language.

5. Section 3.1.1 on the termination criterion was rewritten to make it more clear about why we decided to use this criterion.

6. The sections describing the mesh improvement strategies that occur during (Section 3.1.2) and after (Section 3.1.3) mesh generation, to facilitate mesh generation for coastal ocean models, were made largely re-written and trimmed. Two new figures Figure 4 and Figure 5 were created to explain these mesh improvement strategies visually. The psuedo-code description of Algorithm 2 was removed in favor of Figure 4, reflecting Reviewer 1's suggestions

7. Section 3.1.4 now describes the multiscale meshing capability and the text was focused more on the changes to the *DistMesh2D* algorithm that enabled the capability to include a variety of digital elevation models into the mesh generation process concurrently. This edit was done to reflect reviewer's 2 concern about our algorithmic contributions being too heavily tied to our software.

8. The last paragraph of Section 3.2.2 describing the *geodata* class was rewritten to reflect how the algorithms we have developed can largely automate and improve coastal mesh generation workflow.

9. All subsections describing the automatic mesh size functions were largely trimmed and checked for grammatical mistakes. Specifically the function describing the channel mesh size function was largely edited.

10. To reflect reviewer 2's comment on how the paper read like a user guide we removed mesh size function parameter choices and implementation details which are better suited to the user guide.

11. The subsection that previously described the mesh size interactions between the automatic mesh size functions was positioned as Section 3.3.6 (Mesh size transitions). This rearrangement to further down in the manuscript better reflects the order in which the automatic mesh size functions are created.

12. A new subsection termed "3.3.5. Finalizing Mesh Size functions" was created to clarify exactly how the mesh size function is finished and the various combinations of mesh size functions that can be achieved.

13. What is now Figure 14 was corrected. Panels (b) and (d) were in the wrong order.

14. A new section 3.4 was created that highlights the mesh storage class (termed the *msh* class) and a new Figure 15 was created showcasing three methods of the *msh* class. The location of the *msh* class at the end of the "Component Design" section reflects reviewer 1's suggestions.

15. Statements in the section describing the wall-clock times when generating the example problems were trimmed and how they could be reduced in the future was removed.

16. To reflect reviewer 1's in-depth comment on reproducibility, two additional sentences were added to the end of the first paragraph of the Section titled "Discussions and Conclusions" to reflect how the overall tool could be improved by incorporating Research Data Management practices to maintain a more absolute notion of reproducibility.

17. A new paragraph at the end of the "Discussions and Conclusions" section was added that illustrates how the software's individual components may be used in a different order or for alternative purposes (i.e., to obtain a simplified shoreline boundary or to solely construct a high-resolution mesh size function).

List of changes

OceanMesh2D 1.0: MATLAB-based software for two-dimensional unstructured mesh generation in coastal ocean modeling

Keith J. Roberts[1], William J. Pringle[1], and Joannes J. Westerink[1]

[1]Department of Civil and Environmental Engineering and Earth Sciences, University of Notre Dame, 156 Fitzpatrick Hall, Notre Dame, IN

Correspondence: Keith J. Roberts (krober10@nd.edu)

Abstract. OceanMesh2D is a set of MATLAB functions with pre- and post-processing utilities to generate two-dimensional (2D) unstructured meshes for coastal ocean circulation models. ~~These~~ Mesh resolution is controlled~~generated~~ according to a variety of feature driven geometric and topo-bathymetric ~~mesh size~~ functions.~~, which are controlled by user-defined parameters.~~ Mesh generation is achieved through a force-balance algorithm to locate vertices and a number of topological improvement strategies aimed at increasing~~improving~~ the minimum~~worst case~~ triangle quality. The placement of vertices along the mesh boundary is adapted automatically according to a~~the~~ mesh size function eliminating the need for contour simplification algorithms. The software expresses the mesh design and generation process via an objected-oriented framework that facilitates efficient workflows that are flexible and automatic.~~The objective of this paper is to describe the functionality of OceanMesh2D.~~ This paper illustrates the various capabilities of the software and demonstrates its utility in realistic applications by producing high-quality, multiscale, unstructured meshes.

1 Introduction

Many phenomena in the coastal ocean, such as tides, tsunamis and storm surges, can be accurately modeled by the shallow water equations. Unstructured meshes are often used for numerical simulations of the coastal ocean ~~circulation~~ because they can ~~naturally~~ resolve the large range of horizontal length scales necessary for accurate hydrodynamic predictions and can conform well to complicated shoreline boundaries. The accuracy and the associated computational expense of the mesh are in direct conflict, which makes the mesh design process challenging. Computational work is governed by the distribution of vertices (mesh resolution) and accuracy is determined, in part, by the representation of relevant geometrical and bathymetric features that may influence the simulation~~of the shallow water physics~~. Due to this balance between accuracy and computational work, the prescription of the mesh resolution often leads to a highly subjective ~~and non-reproducible~~ mesh generation process. To address this issue, the ocean modeling community have developed approaches and tools to build unstructured meshes for coastal circulation problems (Hagen et al., 2002; Bilgili et al., 2006; Gorman et al., 2006, 2008; Lambrechts et al., 2008; Conroy et al., 2012; Engwirda, 2017; Candy and Pietrzak, 2018). Most works have either tried to minimize topo-bathymetric interpolation error on the mesh (e.g., Gorman et al., 2006) or construct the mesh based on resolving relevant physical processes in the domain and/or preserving the geometry of the shoreline boundary (e.g., Conroy et al., 2012; Engwirda, 2017). An iterative *a posteriori*

method which aims to keep the local truncation error constant throughout the mesh has also been employed (Hagen et al., 2002).

~~Often mesh generators are written in languages like C or C++ and use expert language and complex mathematical ideas, which may require a steep-learning curve to operate the software and can be challenging to adapt to the user's needs.In contrast,~~ Modern interpreter-based programming environments such as MATLAB and Python are attractive to many users to develop mesh generators because they include a plethora of built-in or community developed functions, toolboxes, and packages that are freely available. For instance, a simple and easily adaptable mesh generator based on the concept of force equilibrium and written in a few dozen MATLAB lines is *DistMesh2D* (Persson and Strang, 2004). The simplicity of the force-equilibrium algorithm makes it attractive as a general-purpose mesh generator by allowing users and developers to adapt it for various applications (e.g., Engsig-Karup et al., 2008; Liu, 2009; Nguyen et al., 2010; Wang et al., 2014). However, due to the general nature of *DistMesh2D*, it tends to be computationally inefficient for the large and highly multi-scale geophysical domains that are encountered in coastal ocean hydrodynamic modeling problems. Additionally, there are a number of pre-processing steps that must be performed to prepare the geospatial data for meshing and a number of post-processing steps to make sure the mesh is amenable for simulation.~~and post-processing steps that must be performed to prepare the geospatial data for meshing and operate on the mesh so that it is amenable for numerical simulation.~~ For instance, one must obtain a shoreline boundary that will lead to a mesh that is practical to simulate with. By integrating the tools to pre-process the geospatial data into the mesh generator directly, it reduces the time spent performing these essential tasks and largely automates the mesh~~model~~ development process.~~ performs the model develop process in a self-consistent, reproducible way.~~

In a related previous work, the Advanced Mesh generator (ADMESH; Conroy et al., 2012) implemented a *DistMesh2D* based coastal ocean mesh generator in MATLAB. In this work, we build~~ and improve~~ on many of the ideas described in *ADMESH* with the following primary improvements: a) a focus on computational efficiency to enable the software to become practically useful even for~~ exceedingly~~ large geophysical datasets (e.g., $\sim$1 km resolution global topo-bathy) in the MATLAB scripting language; b) the inclusion of pre- and post-processing workflows; c) a greater variety of mesh size functions and flexibility in their application which offers more control over mesh resolution placement; and critically: d) code written in an open-source environment for the benefit of the community. The codes place emphasis on facilitating automatic mesh design workflows that lead to the creation of~~ the~~ meshes and~~ also~~ the necessary model inputs for a numerical simulation~~following many of the ideas described in Candy and Pietrzak (2018)~~. These mesh generation workflows (i.e., a user-specified MATLAB control script) are typically represented by a few lines of MATLAB code and take between minutes to an hour to generate relatively large, multiscale, high-fidelity meshes (potentially global-to-channel scale) and their auxiliary components automatically.~~ on a personal computer.~~

The software is written in an objected-oriented framework that is divided into a set of standalone classes~~ (Table 2). related to: 1) processing geospatial datasets used in the mesh generation process; 2) computing mesh size functions; 3) performing the mesh generation; 4) storing, visualizing, and post-processing the mesh output.~~ Special attention has been made to ensure that only open-source functions~~paid MATLAB toolboxes~~ are~~ not~~ required to generate a mesh. Further, in its current state the software contains a number of post-processing functions specific to the ADvanced CIRCulation model (ADCIRC; Luettich

Table 1. The parameters (defined in Sect. 3.3) that are used to generate the three example meshes for this paper, ~~and~~ which were released with the toolkit. The final mesh quality (defined in Sect. 3.1.1) and the number of iterations in the mesh generator to achieve this are noted.

Region	Meshing Parameters								Mesh Quality		Iterations
	$h0$ (m)	h_{max} (m)	α_R	α_{wl}	α_{slp}	α_g	α_{ch}	Δ_t (s)	$\overline{q_E}$	$q_{E_{min}}$	$\ni q_{L3\sigma} > 0.75$
JBAY	15	1,000	3	–	–	0.15	–	2	0.97	0.60	38
GBAY	60	1,000	3	–	–	0.25	0.10	–	0.97	0.53	71
PRVI	10 & 30 & 1,000*	10,000	5	30	15	0.2	–	0**	0.97	0.45	30

*: Different values of $h0$ are used for each separate mesh size function domain as indicated in Fig. 1 (PRVI)
**: Setting $\Delta t = 0$ invokes the automatic time step selector option (see section 3.3.7)

and Westerink, 2004), but these can be ~~naturally~~ adapted to other solvers in the future. The rest of this paper is structured as follows: we begin by introducing the framework and organization of the code followed by a detailed description of each of the four standalone classes, and ending with a discussion on how the software can be useful for coastal ocean model development.

1.1 Example problems

To demonstrate the overall workflow and the design of the classes, three examples located along the East Coast and Gulf Coast of the United States of America are documented (Fig. 1). The first example produces a mesh of the Jamaica Bay estuary in New York (JBAY), demonstrating the utility of the software in incorporating high-resolution (∼1/9-arc second or approximately 3-m horizontal resolution) LIght Detection And Ranging (LIDAR) datasets with fine (∼15-m) resolution triangular elements nearshore. The second example meshes the Galveston Bay in Texas (GBAY), demonstrating the utility of a new mesh size function that can be used to target resolution along deep-draft marine navigation and tidal channels. The third example demonstrates how the software can produce truly multiscale unstructured meshes in less than one hour by building a mesh of the Western Atlantic Ocean with focused refinement around Puerto Rico and the U.S. Virgin Islands (PRVI). See Table 1 for details of the various options/parameters that were used to generate these example meshes.

2 Architecture Overview

The automated generation of geophysical-use unstructured meshes often requires a number of user defined parameters and a variety of geospatial data as inputs. As a result, the ~~resulting~~ mesh is strongly related to the algorithms and data that were used to create it. These task- and object- specific properties of the mesh generation process provide the motivation behind the development of an objected-oriented programming (OOP) approach. In this software, the use of OOP leads to automation and promotes the usage of efficient workflows. ~~since data and the methods used are tightly coupled together in OOP (Register, 2017).~~

OceanMesh2D is composed of four classes (*geodata*, *edgefx*, *meshgen*, and *msh*, see Table 2 for a brief description of each class) and a utilities directory containing various standalone functions. The *geodata* class is used as a pre-processor to mesh generation and creates an appropriate meshing boundary from user-supplied geospatial datasets and inputs. The *edgefx* class enables the user to build standardized mesh size functions with a variety of parameters ~~different options~~ and constraints. The

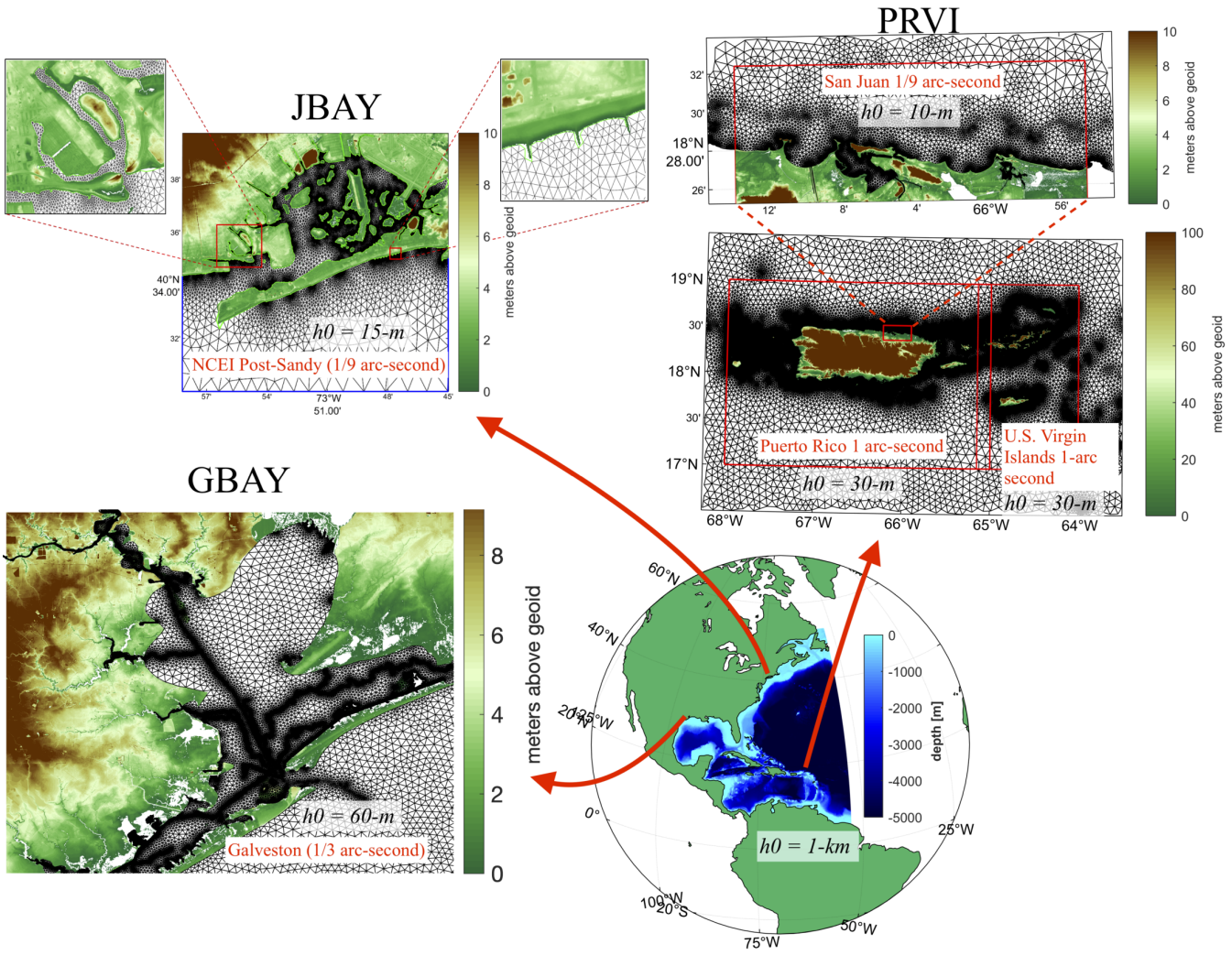

Figure 1. The geographical location and triangulation of the three meshes used as examples in this work. The minimum mesh sizes (*h0*) are annotated in black text and the names of the digital elevation models (DEMs) used in the construction of the mesh size functions are annotated in red text on each panel. The colormap indicates topographical data (bathymetric data was removed for production of this figure) in the DEMs, which are freely available through the NOAA Bathymetric data viewer website (https://coast.noaa.gov/dataviewer).

Table 2. Classes in OceanMesh2D.

Class name	Purpose
geodata	Process geospatial data: mesh boundaries (e.g., shoreline~~coastline~~) & digital elevation data.
edgefx	Compute the mesh size functions based on geospatial datasets~~data~~ and parameters.
meshgen	Mesh generator + descriptor for parameters and geospatial datasets~~data~~ used in mesh generation process.~~to build mesh.~~
msh	Store, visualize, and operate on the~~Store and visualize~~ mesh ~~topology~~ and associated attributes.

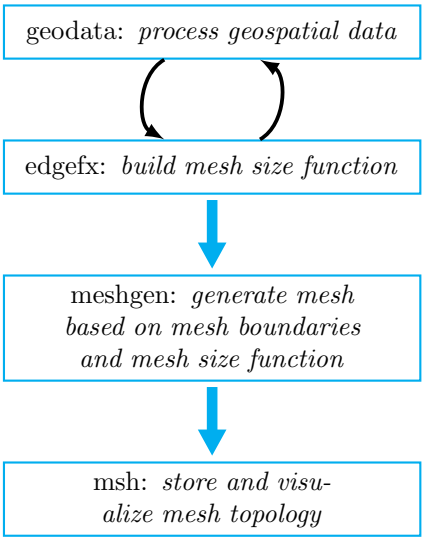

Figure 2. Standard workflow in OceanMesh2D.

meshgen ~~and *msh*~~ class is~~are~~ associated with mesh generation inheriting~~and inherit~~ various options from the *geodata* and *edgefx* classes. The *msh* class is a data storage class for the mesh and related attributes.~~These four classes are constructors for creating specific instances of each class otherwise known as objects. All classes are activated through the use of name-value pairs where the "name" represents an option and the "value" is the parameter relevant to that option. The following sections will detail each class and its various methods.~~ Additional technical information can be found in the user guide (Roberts and Pringle, 2018).

Although each individual class is standalone, there exists a specific workflow that is typically followed to build coastal ocean meshes with the *OceanMesh2D* software (Fig. 2).~~ In this workflow, each object is passed to the subsequent constructor that inherits various properties of the previous object (in addition to other user-defined parameters unique to that constructor). As demonstrated in section 3.1.4,~~ The structure of this workflow enables the user to create numerous instances of the *geodata* and *edgefx* classes that can be subsequently passed to the *meshgen* class. Numerous instances of the *geodata* and *edgefx* classes can be combined to seamlessly mesh high-resolution insets contained within wider coverage geospatial datasets. The ability to incorporate datasets over a wide-range of scales~~that are associated with coarser mesh size functions.~~ is particularly useful and pragmatic given the finite computational memory and highly-variable horizontal resolution of available high-resolution topo-bathymetric data.

3 Component Design

In the following section, each of the four classes that comprise *OceanMesh2D* are described.

3.1 Mesh generation : *meshgen* class

Mesh generation is achieved through the use of the *DistMesh2D* algorithm with a number of modifications to help improve the quality of the final triangulation, the speed of the mesh generation, and the memory footprint of the overall application. It's noted that the architecture of *OceanMesh2D* software could additionally support other mesh generation packages besides *DistMesh2D*, such as *JIGSAW-GEO* (Engwirda, 2017). Thus the purpose of the class is to configure mesh generation related options and subsequently call the mesh generator algorithm. IBut in its current state, the class is a wrapper function around the *DistMesh2D* algorithm that automatically uses classes that describe the meshing domain and the mesh size functions $h(\boldsymbol{X})$. However, it can be used as a standalone 2D mesh generator with a polygonal boundary and mesh size function.

For coastal mesh generation, a key advantage of using the *DistMesh2D* smoothing-based algorithm over Delaunay refinement and/or Frontal Delaunay mesh generation algorithms is that the boundary is implicitly defined. The implicit definition of the mesh boundary enables the vertices to move during mesh generation in accordance with the mesh size function. Thus, by using a set of mesh improvement strategies, the need for shoreline approximation pre-processing (e.g. Gorman et al., 2007) to define the mesh boundary as required by Delaunay refinement and/or Frontal Delaunay mesh generation approaches (Gorman et al., 2008) is eliminated. In this section, we document the mesh improvement strategies that occur during the execution of the *DistMesh2D* algorithm that lead to a high-quality approximate representation of the domain and are in congruence with mesh size functions.

The quality of any mesh can be often significantly enhanced through the use of mesh improvement strategies. Approaches for mesh improvement generally can be characterized in three ways: vertex relocation, connectivity adjustments, and addition/deletion of vertices. We present methodologies for both during and after the mesh generation process that combine these three general classes of techniques to improve the final quality of the mesh.

3.1.1 Termination criterion

In the *DistMesh2D* algorithm Persson and Strang (2004) proposed a termination criterion based on convergence to a state configuration of vertices where in which negligible movement of the mesh vertices would occur with additional meshing iterations. In practice, this our studies have found that a configuration of vertices with negligible movement is difficult to achieve within reasonable hundreds of meshing iterations for realistic coastal ocean mesh domains give because of the fractal complex shoreline boundary and desired heterogeneous mesh size functions. Thus, we propose an alternative highly-achievable termination criterion based on the distribution of element quality exceeding a triangle-quality threshold:.

The notion of what constitutes a goodhigh quality mesh is application dependent. Mesh quality can be viewed as a combination of geometric element quality measures, and application dependencies, and numerics (Shewchuk, 2002). For 2D shallow water flows, a high quality mesh is often determined by geometric quality measures (i.e., nearly all equilateral triangles) with a lower bound on the minimum element quality and the majority of vertices having nearly six edges connected to it (Babuška and Aziz, 1976; Babuška and Aziz, 1993; Babuška and Aziz, 2002).When most elements have a high-degree of regularity in the vertex-to-vertex connectivity, it improves the mesh size transitions (gradation) are smoother, the condition number of finite element coefficient matrices is reduced, and potentially the memory footprint of the finite element solver can be reduced (Massey, 2015).A high degree of regularity in the vertex-to-vertex connectivity also tends to coincide with a mesh

~~with many equilateral or nearly equilateral triangles.Smoothing-based mesh generation approaches, like *DistMesh2D*, have no theoretical guarantees of minimum triangle quality and thus may take a long time to, or may never, reach a desired quality.~~ A geometric measure of triangle equilateral-ness~~triangle quality (measure of *equilateral-ness*)~~ can be calculated through:

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

where A_E is the area of the triangle and $(\lambda_E)_i$ is the length of the i^{th} edge of the triangle. q_E = 1 corresponds to an equilateral triangular element and q_E = 0 indicates a triangle that degenerates to a line. A mesh with a sufficiently high minimum bound on q_E is often desired (Shewchuk, 2002; Persson and Strang, 2004; Engwirda, 2017). However, we find that a minimum bound on q_E is a strict measure for large domains with millions of elements and complex shoreline features, and difficult to achieve within the modified *DistMesh2D* algorithm (Fig. 3). Instead, we use the following termination criterion:

$$q_{L3\sigma} \equiv \overline{q}_E - 3\sigma_{q_E} > 0.75 \tag{2}$$

where the over-line and σ denote the mean and standard deviation respectively, and $q_{L3\sigma}$ is the "three-sigma lower control limit" element quality, used as a proxy for the minimum element quality.

Upon termination through the above criterion ~~we find that the distribution of the element equality is often Gaussian and that by ensuring Eq. (2)~~ the vast majority (>95%) of ~~the~~ triangles are of adequate quality~~ once this criteria is met. Ideally, we would wish to bound the minimum element quality but, the minimum element quality can be a poor measure of the overall mesh quality for large domains with millions of elements the minimum element quality is a poor indicator of overall mesh quality~~. Moreover, in ~~out~~ the approach ~~paradigm~~ used here, a number of mesh cleaning steps are performed *after* this mesh generation termination criterion has been met (Sect. 3.1.3) in order to improve a typically small number of the worst ~~-~~quality (Fig. 3).

3.1.2 Mesh improvement strategies during mesh generation

~~We find that~~Approximately every ten meshing iterations ~~or so~~ the $q_{L3\sigma}$ element quality starts to saturate ~~and~~. The termination criteria can be met more quickly ~~To accelerate convergence during the mesh generation process~~ by relying on the following mesh improvement strategies ~~are~~ that are conducted every ten~~10~~ iterations (except item 4 which is executed every meshing iteration):

1. Edges in the mesh that are greater than two times the length as given by the mesh size function (at the midpoint of the edge) are bisected. ~~$h(\boldsymbol{x}_i)$ (where $\boldsymbol{x}_i$ is the mid point of the edge) are bisected. This creates a new vertex in the center of the bisected edge.~~

2. Edges that are half as short as their intended length ~~($h(\boldsymbol{x}_i)$)~~ are deleted.

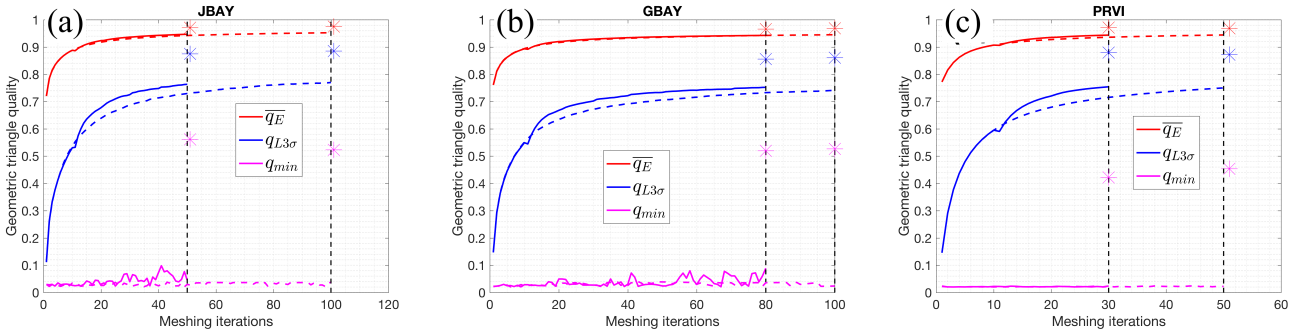

Figure 3. The geometric triangle quality q, Eq. (1), as a function of iterations in the mesh generation process for the three mesh examples (Fig. 1 and Table 1). The dotted ~~(solid) red (mean), blue (lower third sigma), and magenta (minimum)~~ and solid lines indicate the progression of quality metrics with the mesh improvement strategies turned off ~~(on)~~ and on, respectively, during mesh generation. At the end of mesh generation, a secondary round of mesh improvement strategies are applied and the resulting quality after this step is indicated by the colored asterisk. In each panel, the dotted vertical black line demarcates when the mesh generation process finished.

3. A vertex not on the mesh boundary that is connected to less than or equal to four vertices is deleted (this is also performed when the termination criterion is satisfied).

4. Triangles with exceedingly thin angles ($< 5°$) and large angles ($> 175°$) are removed every iteration.

Improvement strategies one and two add and delete vertices when they are part of edges that are too long and short, which produces a set of new edges that more closely approximate the mesh size function~~$h(\boldsymbol{X})$~~. Improvement strategy three directly reduces the occurrence of low vertex-to-vertex connectivity (valency of three or four) where a valency of six is ideal (Canann et al., 1993). Note that improvement strategy one also helps to reduce high vertex-to-vertex connectivity indirectly by avoiding ~~large~~ steep transitions ~~of~~ in the element size where ~~larger~~ valencies greater than six tend to develop. The fourth improvement strategy ~~four~~ removes ~~flat~~ triangles with small and large angles~~obtuse triangles~~ allowing neighboring ~~points~~ vertices~~to fill these gaps leading~~ to ~~new~~ form a triangulation that has a better geometric quality.

We demonstrate the benefit from using these mesh improvement strategies through the three example meshes (Fig. 1, Table 1). The time evolution of the geometric ~~element~~ quality demonstrates the benefit directly from these mesh improvement strategies. Figure 3 illustrates that in all three examples the mesh improvement strategies lower the number of iterations necessary to achieve the termination criterion. Further, the rate at which $q_{L3\sigma}$ increases is accelerated ~~in all examples~~ when mesh improvement strategies are enabled. For the development of large multi-scale meshes, 20-50 iterations can often save between 5-20 minutes for the problems ~~here~~ to reach the termination ~~quality~~ criterion. Based on the termination criterion and the improvements listed here, we generally find that complex coastal ocean meshes are generated in approximately 30-100 iterations. Thus, the maximum allowed number of iterations is commonly set to 100, which typically takes a few minutes to half an hour to compute depending primarily on the geometric complexity of the boundary ~~manifoldness~~ and the ratio of domain size to minimum element size.

3.1.3 Mesh improvement strategies after mesh generation

After mesh generation has terminated, ~~we apply~~ a secondary round of mesh improvement strategies is applied that is focused towards improving the geometrically worst quality triangles that often occur near the boundary of the mesh and can make simulation impossible (e.g., Fig. 4(a)). ~~Poor~~ Low quality triangles can~~typically~~ occur near the mesh boundary because the geospatial datasets used may contain features that~~narrow channels and restricted connections between bodies of water may be difficult or impossible to resolve~~ have horizontal lengthscales smaller than the minimum mesh resolution. To handle this issue, a set of~~we instead employ a handful of~~ algorithms are applied that iteratively~~exhaustively~~ address the vertex connectivity problems~~problems.that typically arise.~~ The application of the following mesh improvements strategies results in a~~end result is a~~ simplified mesh boundary that conforms ~~well~~ to the user-requested minimum element size. ~~and can be used for simulation.~~

~~In order to simulate with a given mesh, It is essential to that there are no~~ Topological defects in the mesh ~~that make the simulation impossible~~ can be removed by ensuring that it is valid, defined as having the following properties:~~We call a mesh with none of the following defects valid. A valid mesh in our work is defined by having the following properties:~~

1. The vertices of each triangle are arranged in the counter-clockwise order.

2. Conformity~~There mesh is conformal~~: a triangle is not allowed to have a vertex of another triangle in its interior.

3. Traversability: the number of boundary segments are equal to the number of boundary vertices, which guarantees a unique path along the mesh boundary.~~The boundary of the mesh can only be traversed from any starting point on it by only visiting *unique* boundary points that are connected together by an segment.~~

Properties one and two are handled with the *fixmesh.m* function that was provided with the original *DistMesh2D* package. Property three (traversability) is often not satisfied upon termination of the mesh generator because a simplification of the shoreline was not applied. Fragmented patches of triangles may appear~~result~~ near the shoreline boundary destroying traversability~~and can destroy the uniqueness of the mesh's boundary path, which we call traversability~~ (Fig. 4). ~~Note that property three can be more simply expressed as requiring the number of boundary segments to be equal to the number of boundary vertices.~~

A function, called *Make_Mesh_Boundaries_traversable*, ~~*Make_Mesh_Boundaries_traversable* function, containing the *Delete_Exterior*~~ was~~were~~ developed to iteratively remove ~~fragmented~~ patches of elements that are either disconnected from the major portion of the mesh ~~(that may or may not affect the mesh boundary traversability),~~ or are not disconnected but prevent~~prevented the mesh boundary~~ traversability. The former set of offending~~fragmented~~ elements are defined as being "exterior" disjoint components of the mesh where, starting from a random seed element in the mesh, the total area of a connected set of elements (i.e., elements that share an edge) is smaller than a user-defined threshold μ_{co}, which is defined in terms of either the total mesh area-fraction or an absolute area cutoff (by default we set $\mu_{co} = 0.25$ which is equivalent to a 25% total mesh area-fraction cutoff). These patches are identified and removed~~deleted by the *Delete_Exterior_Elements* sub-function~~ through the use of a breadth-first search (BFS) (Fig. 4(a)-(b)). The latter set of offending~~fragmented~~ elements are defined as being "interior" elements of the mesh that interfere with the traversability of the mesh boundary path ~~that are not caught by the *Delete_Exterior_Elements* sub-function. The *Delete_Interior_Elements* sub-function deals with identifying and deleting these~~

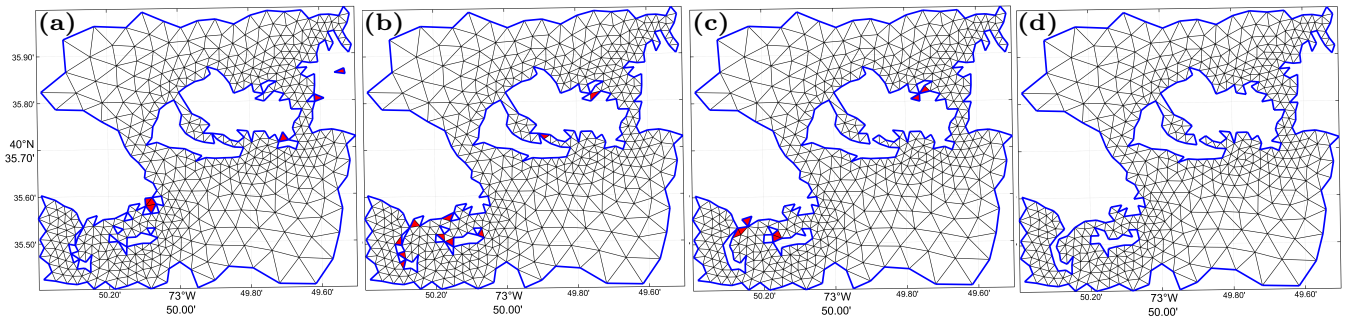

Figure 4. Mesh triangulation within the JBAY example before and after different stages of the *Make_Mesh_Boundaries_Traversable* function enabling mesh traversability. (a) After initial mesh generation (before entry to function); (b) After deleting offending exterior elements; (c) After deleting offending interior elements; (d) After exit of function once all offending exterior and interior elements are deleted and traversability is obtained. The thick blue line indicates the mesh boundary at each stage, and red patches indicate the elements that are deleted between stages (sub-plots).

~~elements. In this sub-function,~~. First, an offending vertex ~~is first identified~~ that has more than two connecting boundary edges is identified. One of the elements connected to this vertex is chosen to be deleted based on a hierarchy of, first, triangles that have two boundary edges, and second, triangles with the lowest quality, q_E (Fig. 4(b)-(c)). Offending exterior and interior elements are deleted~~The application of *Delete_Exterior_Elements* followed by the *Delete_Interior_Elements* is conducted~~ iteratively until traversability is achieved (Fig. 4(c)-(d)). ~~Further details of *Make_Mesh_Boundaries_traversable* and examples of the elements that it deletes is included in the user guide.~~

After ensuring traversability, three additional functions, depicted visually in Fig. 5, are applied to the mesh in the following order to improve mesh quality:

1. *Fix_single_connec_edge_elements*: elements that share an edge with only one other element (singly connected elements) poorly approximate geospatial datasets and are thus removed from the mesh iteratively (Fig. 5(a),(d)).

2. *bound_con_int*: bounds the vertex-to-vertex connectivity (e.g., Fig. 5(b),(e)) in the mesh to a user-defined value in order to improve mesh quality and gradation, and also increase solution accuracy and computational speed (Massey, 2015).

3. *direct_smoother_lur*: provides additional improvement to the mesh quality by moving non-boundary vertices based on a single-step implicit operation (Balendran, 1999) (Fig. 5(c),(f)). The application of this function significantly enhances the statistical distribution of q_E (Fig. 3).

~~the mesh's boundary is made traversable, boundary triangles that are connected to just a single neighboring triangle are removed exhaustively by the *Fix_single_connec_edge_elements* sub-function. The singly-connected triangles are removed because they are not well-constrained during mesh generation and often poorly represent the shoreline geometry and bathymetry. Further, singly-connected triangles can artificially constrict the exchange of water through narrow water courses when logic-based wetting/drying algorithms are used.~~

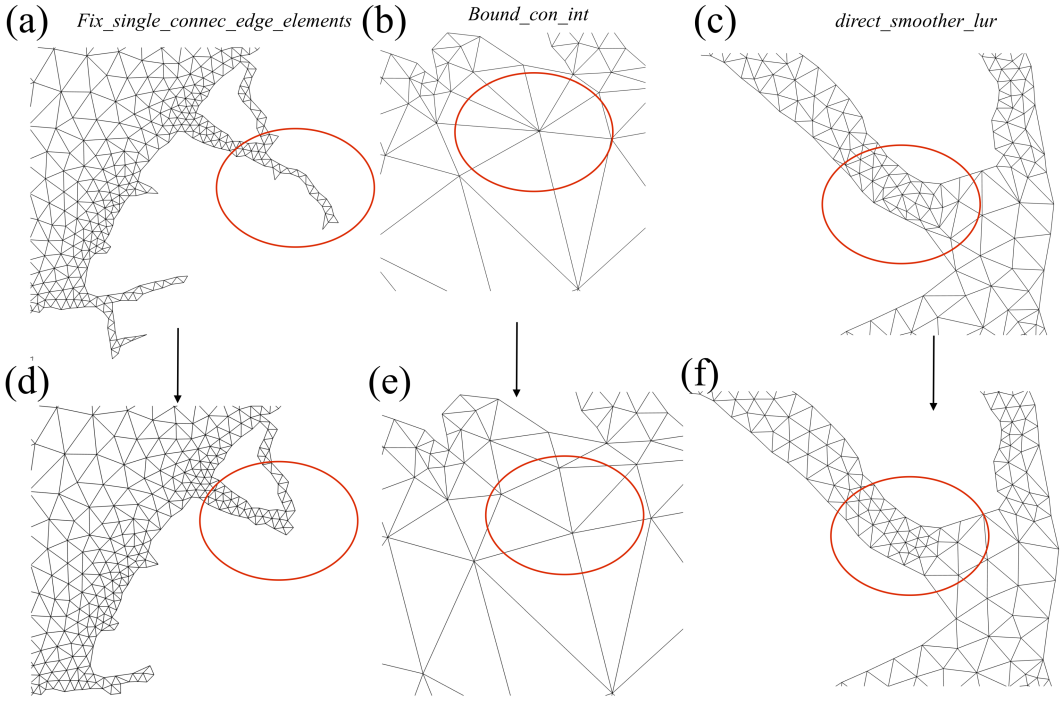

Figure 5. The mesh improvement strategies that are applied in sequence from left to right after mesh generation, with the red ovals denoting areas of change in the connectivity along with the function's name that performs the operation. The top row indicates various regions in the mesh before the improvement strategy, and the bottom row after improvement. Panels (a) and (d) indicate the deletion of elements that share an edge with only one other element (singly-connected elements); panels (b) and (e) illustrate the reduction of the vertex-to-vertex connectivity to an upper bound of six using the algorithms documented in Massey (2015); and panels (c) and (f) illustrate the single-step implicit smoothing operation (Balendran, 1999) that is used to maximize the overall mesh quality.

~~While non-boundary vertices connected to four or less vertices are deleted during the mesh generation process and on termination, vertices with high vertex-to-vertex connectivity are not directly dealt with in the generation process. However, we do find that the periodic splitting of long edges (compared to $h(x)$) and the smoothing of the mesh size function ensures that almost all of the vertices in the mesh have good vertex-to-vertex connectivity (i.e., close to six) with a typical upper bound of eight. Although connectivity higher than eight is less common, the *bound_con_int* function that was described and originally coded by Massey (2015) is used after the *Fix_single_connec_edge_elements* function to bound the vertex-to-vertex connectivity in the entire mesh to eight. It is possible to use the *bound_con_int* function to bound the vertex-to-vertex connectivity to seven, but convergence may take a considerable amount of time so this is excluded from the standard procedure. Instead, the user may invoke the *bound_con_int* function as an additional procedure in an attempt to bound the vertex-to-vertex connectivity to seven.~~

~~The final function that is applied to the mesh in the cleaning process is *direct_smoother_lur*, which provides additional smoothing to the non-boundary vertices through a single-step implicit operation (Balendran, 1999). Applying this function can often dramatically enhance the statistical distribution of q_E. This can be seen in the time series of the geometric element~~

quality where upon execution of the clean-up routine, the minimum triangle quality improves from <5% to 40-60% (Fig. 3). One drawback is that the implicit smoother does not take into account the mesh size distribution. Thus, it is important to ensure that enough iterations of the *DistMesh* algorithm have been conducted so that the triangles are mostly of high quality and conform well to the $h(\boldsymbol{X})$ before applying *direct_smoother_lur*. Note that for best results using *direct_smoother_lur*, elongated mesh *boundary* elements that have poor element quality (we choose q_E <0.5) are deleted before beginning the post-processing mesh improvement steps.

3.1.4 Multiscale meshing approach

The geospatial data used to generate mesh size functions often have varying horizontal resolutions and non-overlapping coverage interspersed with large areas where only coarser global datasets are available. Typically Users often to to mesh localized regions down to the estuarine-lengthscales with comparatively finer resolution (where high quality geospatial datasets are available) and use coarser resolution elsewhere. In these situations, the Cartesian mesh size function approach employed in the The *DistMesh2D* algorithm uses largely memory inefficiently for the development of multiscale regional and global meshes of the coastal ocean because it requires a uniform vertexminimum spacing to initialize to form the mesh size function and. The memory inefficiency becomes especially problematic when employing high-resolution elements locally to fully incorporate the information contained in high-resolution geospatial datasets while using coarser mesh resolution elsewhere. in the *DistMesh2D* algorithm. More fundamentally, for shallow water flow it is unnecessary to use fine mesh resolution in deeper waters because the dominant length scale grows according to the Rossby radius of deformation to thousands of kilometers (LeBlond, 1991). To reduce the memory overhead when constructing regional coastal meshes using the *DistMesh2D* algorithm, the *meshgen* class has been specifically developed to allow the user to pass multiple instances of the boundary description (*geodata*) and mesh size (*edgefx*) classes to the *meshgen* class, an approach that we term 'multiscale meshing'. Instances of these classes are defined within axis-aligned bounding boxes (rectangles) that reflect the available geospatial dataset coverage and can be partially or fully nested any number of times with largely disparate mesh sizes between nests (Figs. 1 (PRVI) and 6). Examples of the multiscale meshing technique are shown in Figs. 1 (PRVI) and 6 in which the mesh sizes seamlessly transition between the different DEM extents. The mesh size function of an *edgefx* instance is updated in areas of overlap using the mesh size function of comparatively higher resolution. The mesh size function of the coarser *edgefx* instance that was updated is subsequently smoothed using the *limgradStruct.m* function.

The major requirement when using this approach is that the instances must be provided in order of *decreasing* minimum mesh size (i.e., coarse to⇒ fine). Note that overlapping regions of the same minimum mesh size are also allowed. An additional requirement is that the coarsest (largest) instance of these classes, which acts as the absolute mesh boundary, must completely encompass all finer instances in space. A key computational benefit of this multiscale meshing approach is that it still enjoys the simplicity and speed associated with structured mesh size function grids as we mentioned in Sect. 3.3 but offers the user more control over how the available geospatial datasets arcis used in the mesh generation process. The OOP framework (Sect. 2) is integral to enabling this approach because multiple instances of the *geodata* and *edgefx* classes can be generated independently from each other.

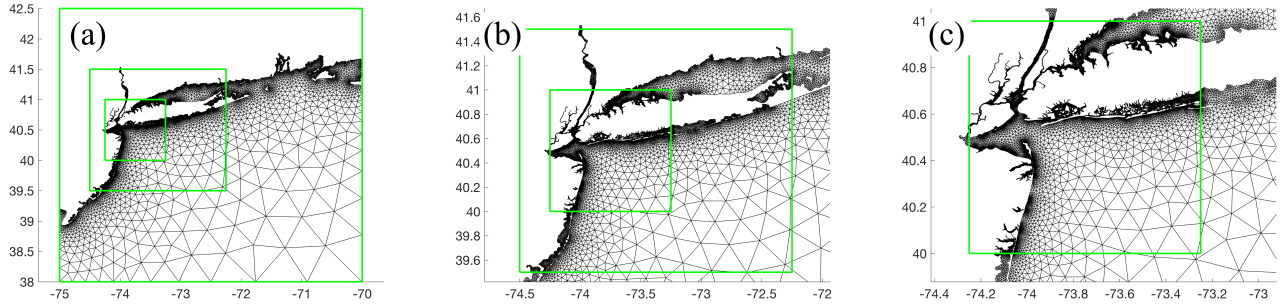

Figure 6. An example of the multiscale meshing technique applied to a set of domains around the New York/Long Island area. The green boxes are specified by the user. The minimum resolution of the outermost green box in each panel is different: (a) it is 1 km, (b) 500 m, and (c) 35 m. Notice how the regions of overlap gradually transition into each other.

Only minor modifications to the *DistMesh2D* algorithm were involved with enabling the multiscale meshing capability. The nested domains are evaluated in a loop inside *DistMesh2D* in a hierarchical~~hiearchial~~ order from comparatively coarser to finer resolution minimum mesh sizes. The hierarchical~~hiearchial~~ evaluation of the force function enables vertices of the mesh to move between the nested boxes so long as the outer box fully encloses the inner box. Since the finest local meshing boundary and mesh size function take precedent within each nested box, it enables many variable resolution geospatial datasets to be included into the mesh generation process simultaneously. In order for the multiscale meshing capability to work, it requires smooth mesh size transitions between nests. A routine (*smooth_outer.m*) was developed~~ during mesh generation the execution of *meshgen()* when multiple *edgefx* classes are present was developed~~ to ensure a smooth resolution transition occurs between nested boxes~~instances of disparate mesh resolution~~ by using ~~a~~the marching method ~~*limgradStruct.m*,~~ (Persson, 2006) that has been adapted~~adapated~~ for structured grids.

The multiscale meshing capability~~This approach~~ is similar to the multi-grid nesting technique employed by ocean models (e.g., Debreu et al., 2012; Brown et al., 2016; Pringle et al., 2018) but in the finite element framework without the need for a coupling paradigm. The application of this method allows for the construction of a single seamless unstructured mesh with mesh size~~edgelength~~ transitions that are bounded by the user-defined allowable limit ~~α_g to be generated~~, while the resolution is not significantly altered~~but does not alter resolution significantly in the insets~~ away from the boundaries of their nests~~bounding boxes~~. The multiscale approach~~This makes our approach~~ is ~~particularly~~ beneficial over traditional structured multi-grid nesting approaches employed by ocean models because it avoids issues associated with interpolation and smoothing at the interfaces between disparate resolution grids that ultimately reduce numerical accuracy.

~~Figure **??** illustrates an example of using multiple *edgefx* and *geodata* instances to construct a locally high-resolution mesh (minimum resolution of 10-30 m) around Puerto Rico and the U.S. Virgin Islands (PRVI), embedded within a significantly larger western North Atlantic mesh domain (minimum resolution of 1 km); see Table 1. The resulting seamless mesh is generated from one global DEM (SRTM15_PLUS, 30 arc-second resolution) and three local DEMs. Two of the local DEMs have 1 arc-second resolution and cover the general PRVI region. The other local DEM covers a small region around San Juan,~~

~~Puerto Rico with 1/9 arc-second resolution. high-resolution coastlines are extracted from the local DEMs using the r.contour module in GRASS GIS. Note how the mesh seamlessly transitions between the extents of DEMs that are used to create each local *geodata* and *edgefx* class (Fig. **??**).~~

3.2 Geospatial data: *geodata* class

The *geodata* class is a pre-processor to the mesh generator. It is used to create an appropriate mesh boundary description from user supplied input files. The *geodata* class also stores the region of the digital elevation model (DEM) that overlaps with the desired meshing domain efficiently in memory. This DEM data is used in the construction and computation of a number of mesh size functions (see *edgefx* class) and *msh* methods. The following section describes the methodologies to prepare the mesh boundary description.

~~The discrete representation of the boundary is a critical step in ocean modeling applications. A common problem with defining a mesh boundary along a highly irregular, fractal shoreline is that it may require unfeasibly small mesh resolution to correctly capture its complexity. To tackle this problem, a number of works have developed shoreline simplification algorithms to rearrange mesh boundary vertices so to balance the accuracy of the discrete shoreline given the user requested mesh resolution (e.g., Gorman et al., 2007; , 2008; **?**). In this work, through the adoption of the *DistMesh* algorithm, we avoid this issue by automatically resolving or de-resolving the shoreline via the mesh size function and a series of post-processing steps. The post-processing steps remove regions of the mesh that are invalid as a result of inadequately prescribed resolution in the vicinity of shoreline complexities. The end result is a mesh boundary that is simplified to closely match the desired mesh size function without the need to edit the shoreline beforehand. These post-processing steps will be explained in Sect. **??**.~~

3.2.1 Projections

Mesh resolution sizes need to be accurately distributed according to the mesh size function~~Users often want an ability to bound placement needs to be accurate in mesh resolution sizes in certain parts of a large coastal modeling domain~~. In order to accurately enforce these constraints on the Earth, a projection from the spherical geometry of the Earth to a planar one ~~$\mathbb{R}^3 \rightarrow \mathbb{R}^2$~~ is necessary. In this software, the mesh is generated and output in the World Geographic Coordinate System (WGS84).~~; therefore, all geospatial data must be supplied in this geographical coordinate system.~~ For the formation of some mesh size functions that rely on bathymetric gradients and distances, we use a simple relationship between WGS84 degrees and planar meters to calculate the underlying grid spacing:

$$\delta^*_{lon} = \delta_{lon} \frac{\pi R_E}{180} cos\phi, \qquad \delta^*_{lat} = \delta_{lat} \frac{\pi R_E}{180} \tag{3}$$

where δ_{lon} and δ_{lat} define the DEM resolution in WGS84 degrees between meridians and parallels, respectively, R_E is the mean radius of the Earth (≈ 6378 km), δ^*_{lon} and δ^*_{lat} are the distances between meridians and parallels in meters, and ϕ is the latitude in radians. To enforce mesh resolution constraints, we use the Haversine formula to convert between WGS84 and meters. An assumption is made that the length in geographic degrees forms a horizontal (i.e., latitude parallel) edge starting

at the point it is defined at. The distance between the start and end point of this edge are converted to Great Circle distances using the Haversine method and then, later, we invert the Haversine formula and solve for WGS84 degrees by assuming that the distance between latitudes is zero:

$$h_d = 2\arcsin\left(\sec\phi \sin\left(\frac{h^*}{2R_E}\right)\right) \tag{4}$$

where h^* is the length of the edge in meters, and h_d is the length of the edge in WGS84 degrees. The assumption that the edgelength extrudes along a latitude parallel is reasonable in practice because the mesh size function constraints matter mostly in areas of relatively high mesh refinement and, in these locations, the variation in ϕ is small.

3.2.2 Automatic mesh boundary definition ~~and shoreline data~~

Since a coastline is often approximated by a series of piecewise linear segments, the mesh boundary is often unbounded on the ocean-side and is not a polygon (i.e., first point does not equal the last). Thus, the user often has to turn their segments that represent the shoreline into a closed polygon for any meshing algorithm to work properly. To make this process ~~self-consistent and~~ automatic, we enable the user to specify the meshing region as a rectangular box, $bbox$. The mesh domain is then defined as the intersection of the area enclosed by the $bbox$ and the area enclosed by the shoreline polygon. ~~The distance function is signed as negative if a point is inside the mesh domain and positive if it is outside of it.~~ The boundary of the meshing domain is implicitly defined through the use of a signed distance function, d, whereby the distance to the nearest coastline point is zero (Persson and Strang, 2004). Note that a negative value of the signed distance function indicates a point within the mesh boundary, and a positive value of the signed distance function indicates a point outside the mesh boundary.

In addition to defining the meshing boundary ~~$\partial\Omega$~~, the signed distance function is also used to form mesh size functions (see Sect. 3.3) and is used during the execution of the *DistMesh2D* meshing algorithm. To ensure the calculation of the signed distance is computationally efficient, the calculation relies on a combination of the MATLAB class version ~~(Bagon, 2009)~~ of the Approximate Nearest Neighbor (ANN) method (Arya and Mount, 1993; Mount and Arya, 2006) (to obtain the absolute distance) and Dr. Darren Engwirda's points-in-polygon test (*inpoly.m*; available from https://www.mathworks.com/matlabcentral/fileexchange/10391-fast-points-in-polygon-test) function (to get the sign). The ANN method has high computational efficiency with a negligible memory footprint in comparison to the *dsegment.m* function available in *DistMesh2D*. Further, the *inpoly.m* function is several hundred-fold quicker ($O(\log N)$ vs. $O(n^2)$) than MATLAB's built-in *inpolygon.m* function.

In our methodology, the shoreline polygon is internally partitioned into mainland and island polygons (this categorization is defined below). New vertices are added to the shoreline polygon so that it conforms to the user-requested minimum mesh resolution ($h0$) inside the $bbox$. Vertices are decimated outside $bbox$ to save both memory and time during the mesh generation process since the calculation of signed distance function is proportional to the number of shoreline vertices.

1. The segments of ~~S~~ shoreline polygon that intersect with ~~B~~ $bbox$ are read in to memory.

2. The segments of ~~S~~ shoreline polygon are classified into three types: mainland, inner, or outer.

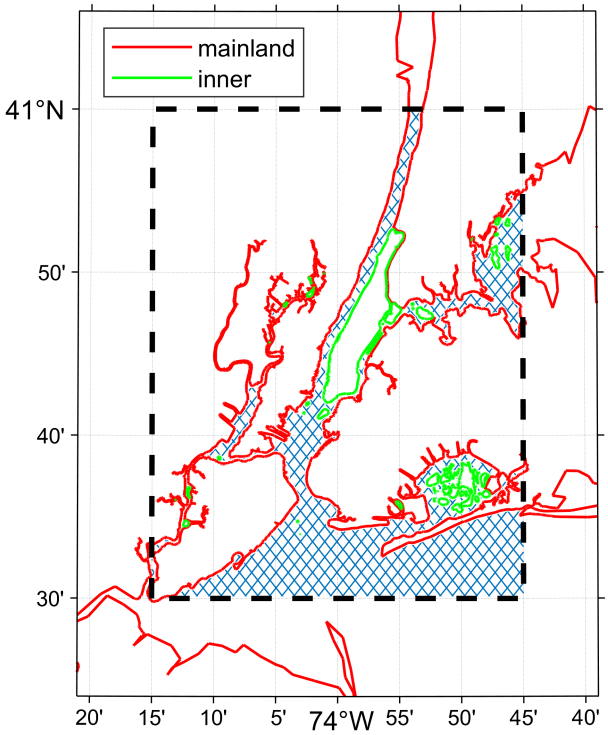

Figure 7. Example of boundary treatment in and around New York, United States; the bounding box of the mesh domain $bbox$ is indicated by the thick dashed black line, the meshing ~~region~~ domain ~~Ω~~ is hatched in blue, and the categorization of land boundary types are indicated according to the colored lines.

(a) The mainland category contains segments that are not totally enclosed inside the $bbox$ ~~$\mathcal{B}$~~.

(b) The inner (i.e., islands) category contains polygons totally enclosed inside the $bbox$ ~~$\mathcal{B}$~~.

(c) The outer category is the union of the mainland, inner, and $bbox$ ~~$\mathcal{B}$~~.

3. New vertices are added on these segments so that no two consecutive vertices along it are further than $\frac{h0}{2}$ apart. This is necessary for accurate re-projection of points that exit the meshing domain during the execution of the *DistMesh2D* algorithm (Persson and Strang, 2004).

4. All segments are smoothed using a n-point moving average. Simultaneously, small islands that have an area less than $(p * h0)^2$ are removed where n and p are user-specified integers ($n = 5$, $p = 4$ by default).

As an example, the following steps are applied to a shoreline extracted from a NCEI Post-Sandy DEM (JBAY in Fig. 1) leading to a classification of shoreline points that is crucial for correct automatic meshing of the complicated coastal domain it describes without human intervention (Fig. 7).

The capability to use geometric contours extracted directly from geospatial datasets in the mesh generation process without the need for external GIS shoreline simplification algorithms or external shoreline datasets improves workflow efficiency and

automation. Further, by using a geometric contour, the resulting shoreline boundary in the mesh is consistent with the topo-bathymetric dataset that is subsequently interpolated onto the mesh vertices. Since many coastal mesh generators rely on the Global Self-consistent Hierarchical High-resolution Shorelines (GSHHS) dataset (Wessel and Smith, 1996), we consider the automatic geospatial data processing algorithms to represent a significant step forward towards more comprehensive coastal modeling efforts. For example, the GSHHS dataset is largely insufficient for meshes with a desired resolution finer than 100-m as it often misses critical connections between water bodies (Fig. 8).

~~Perhaps the most accurate *global* shoreline database suitable for automatic mesh generation is the Global Self-consistent Hierarchical High-resolution Shorelines (GSHHS) (Wessel and Smith, 1996). This database is guaranteed to not have any coastlines that self-intersect and it has a resolution of about 50-m in most areas, which makes it useful for the generation of greater than approximately 100-m minimum resolution meshes. However, for meshes with a desired resolution finer than 100-m or so, the GSHHS is largely insufficient as it often misses critical connections between water bodies that are far finer than 50-m (Fig. 8). A more accurate representation of the meshing domain can often be obtained by extracting a geometric contour that represents the topo-bathymetric height (e.g., 0-m, 5-m, 10-m) above a datum from a DEM. By utilizing a geometric contour, the resulting shoreline boundary in the mesh will be fully consistent with the topo-bathymetric data, which we find can be helpful in producing stable meshes. However, a geometric contour extracted directly from a gridded dataset may not be a polygon by default and may contain breaks or gaps. If the vector shoreline data is not a polygon, we have included a *geodata* method 'close' that implements a flood-fill algorithm in order to obtain a suitable polygonal boundary for our software. The implementation is borrowed from Dr. Darren Engwirda's *JIGSAW-GEO* package (Engwirda, 2017). Artificial gaps in the contour that may originate from noise in the DEM can be largely eliminated by using the r.contour module in GRASS GIS (GRASS Development Team, 2017) with a cut parameter generally between 50 and 200. See the user guide (Roberts and Pringle, 2018) for more details on these aspects of creating self-consistent coastlines.~~

3.3 Automatic mesh size function: *edgefx* class

The careful placement of mesh resolution is critical to create meshes that lead to accurate but efficient simulations. There are a number of heuristics used to design unstructured meshes for shallow water flow applications. A review of some common resolution heuristics utilized in coastal ocean modeling can be found in Greenberg et al. (2007). We have considered a variety of constraints involved in the formation of the mesh size functions by integrating and adapting past work on the topic. The various mesh size functions are detailed in this section.

Mesh resolution is distributed in the domain according to a mesh size function. The mesh size functions~~$h(\boldsymbol{X}) : \mathbb{R}^2 \rightarrow \mathbb{R}$~~ are constructed on a *structured* grid that relates every point in the meshing domain ~~Ω~~ to a desired mesh size h, or more precisely, a triangular edgelength (hence *edgefx*).~~There are many techniques to form mesh size functions that vary principally around the methodology to form the background grid on which the mesh sizes are calculated (Persson, 2006). For example, the simplest approach is to use a Cartesian or structured grid for the mesh size function.~~ Defining the mesh size function on a structured grid has ~~some~~ advantages over an unstructured one (Conroy et al., 2012; Engwirda, 2017) in relation~~related~~ to computational efficiency when storing, querying, interpolating, and performing calculations. Further, bathymetric data is often defined on

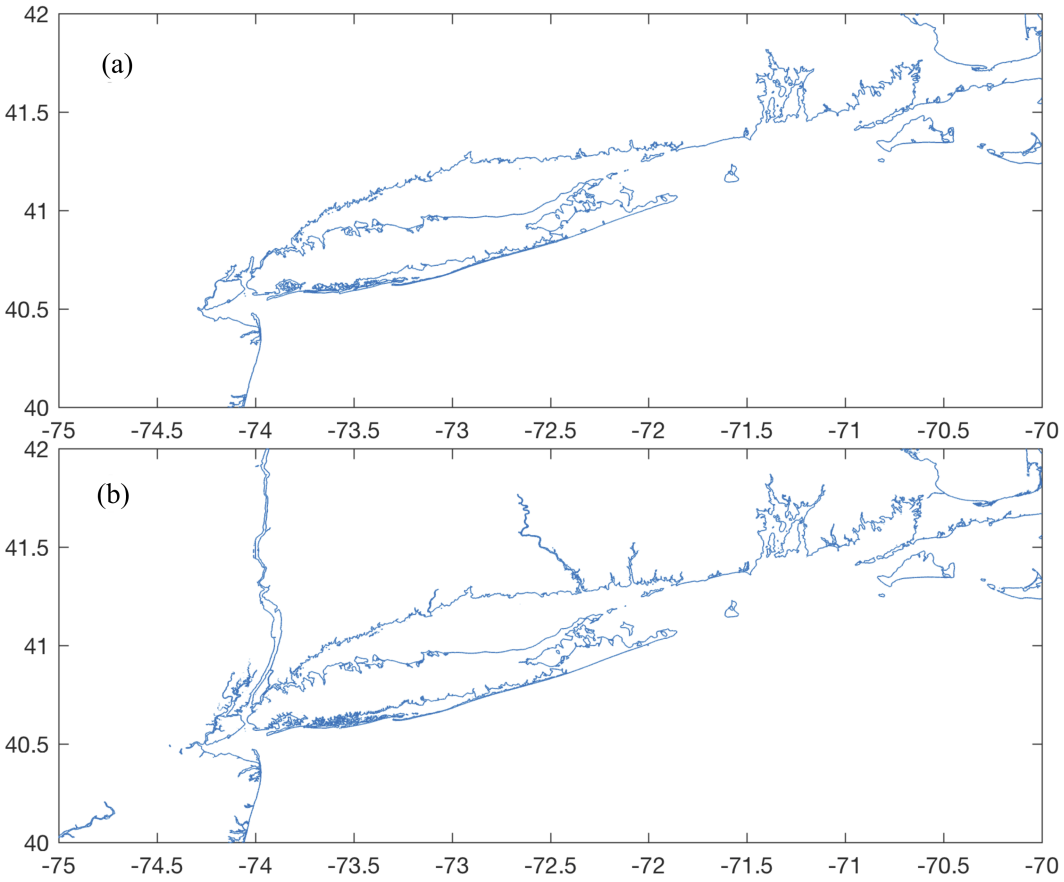

Figure 8. (a) The GSHHS fine (i.e., GSHHS_f) shoreline centered around New York, United States; (b) a shoreline extracted from mosaicing NCEI Post-Sandy DEM~~LiDAR~~ tiles with the GRASS GIS software.

structured grids, and in these cases, ~~it makes sense to~~ computing the mesh size function directly on the same grid can minimize seabed interpolation error for the mesh size function calculation. Given these reasons, we calculate our mesh size functions on Cartesian grids defined in geographical coordinates (i.e., WGS84). A major drawback to this approach is that the entire domain must be uniformly refined which becomes particularly severe for relatively large meshing domains. This impacts the scalability of the subsequent mesh generation process for regional and global coastal mesh generation, but the multiscale mesh capability (Sect. 3.1.4) alleviates this problem. ~~We present a solution to this problem in Sect. 3.1.4.~~

~~The mesh size function $h(\boldsymbol{X})$ is finalized by taking the minimum of the potentially multiple mesh size functions and applying, if specified, user defined minimum and maximum bounds on mesh resolution in meters that can be specified within depth ranges.~~ Each individual mesh size function is based on either shoreline data and/or the bathymetric datasets that were passed to the *edgefx* class constructor. Currently, the software supports a variety of mesh size functions that are used in the ocean modeling community: wavelength-to-grid-size (Westerink et al., 1994; Luettich and Westerink, 2013), topographic length scale (Greenberg et al., 2007; Lambrechts et al., 2008), Euclidean distance from the shoreline (Persson and Strang, 2004), approxi-

mate feature size of the shoreline (Persson, 2006; Koko, 2015), thalweg/polyline (Heinzer et al., 2012), and Courant-Friedrichs-Lewey (CFL)-limiting (Bilgili et al., 2006). Each mesh size function can either be incorporated or omitted based on the user's requirements. ~~Finally, $h(\boldsymbol{X})$~~ The mesh size function is graded using a marching algorithm (Persson, 2006) to ensure that the triangle-to-triangle change in edgelength is bounded below a user-defined percent, α_g. ~~The *limgradStruct.m* function that performs this grading has been specially modified for the structured grid mesh size functions used in this software.~~

3.3.1 Distance and feature size

A high degree of refinement is often necessary near the shoreline boundary to capture its geometric complexity~~mesh boundary $\partial\Omega$ to capture the geometric complexity of the shoreline~~. If mesh resolution is poorly distributed, critical conveyances may be missed leading to larger scale errors in the nearshore circulation (Greenberg et al., 2007). Thus, a mesh size function that is equal to a user-defined minimum mesh size $h0$ along the shoreline boundary, growing~~$\partial\Omega$ and grows~~ as a linear function of the signed distance d from it~~the nearest $\partial\Omega$ point~~ may be appropriate:

$$h_{dis} = h0 - \alpha_d d \tag{5}$$

where α_d is the percent change of mesh size with distance from the shoreline boundary~~S~~. Eq. (5) is what we call the *distance* mesh size function and was~~is~~ originally presented in the *DistMesh2D* algorithm (Persson and Strang, 2004).

One major drawback of the *distance* mesh size function is that the minimum mesh size will be placed even along straight stretches of shoreline~~coastline~~. If the distance mesh size function generates too many vertices~~this is undesirable~~, a *feature* mesh size function that places resolution according to the geometric width of the shoreline~~coastline~~ should instead be employed (Conroy et al., 2012; Koko, 2015). In this function, the feature size (e.g., the width of channels/tributaries, and the radius of curvature of the shoreline) along the coast is estimated by computing distances to the medial axis of the shoreline geometry. Here we have implemented an approximate medial axis method closely following Koko (2015). This involves finding local extrema in the gradient of the d, which in practice, amounts to defining a medial point as a location where $||\nabla d|| < 0.9$ and $d < 0$ (Koko, 2015). Sometimes due to the configuration of the mesh size function grid juxtaposed on the shoreline geometry, medial points inside small channels may be lost~~aliased~~. These medial points can be recovered by classifying mesh size function grid points as medial points if both adjacent~~their~~ neighbors (in the north-south or east-west directions) are outside of the domain but the mesh size function point under question is within the domain. Once the medial points are computed, the local feature size h_{lfs} is calculated as:

$$h_{lfs} = \frac{2(d_{MA} - d)}{\alpha_R} \tag{6}$$

where α_R is the user specified number of desired elements per local feature size (commonly $2 \leq \alpha_R \leq 6$), and d_{MA} is the absolute distance to the nearest medial point. Since the medial axis is an approximation, the identification of the full set of medial points depends on the horizontal resolution of the mesh size function. This implies that the *feature* mesh size calculation

will work best when computed on a structured grid of resolution similar orof finer than the horizontal resolution of the supplied geophysical datasets.data.

To demonstrate the efficacy of the *feature* mesh size function, we use a 1/9 arc second (∼3-m) topo-bathy DEM (Table 1) to generate an approximate 10-m minimum element size mesh of Jamaica Bay in New York, United States (JBAY; Fig. 1), with $\alpha_R = 3$. Relatively coarse resolution is placed along linear regions of the sand bar, while the dark patches indicate where higher resolution is automatically placed around points of high curvature in the coastline and through channels. For example, two closeups are shown where higher resolution is placed along a narrow constriction and around perpendicular coastal groins along a beach.

3.3.2 Wavelength

In shallow water theory, the wave celerity, and hence the wavelength λ, is proportional to the square-root of the depth of the water column. This relationship indicates that more mesh resolution at shallower depths is required to resolve waves that are shorter than those in deep water. With this considered, a mesh size function h_{wl} that ensures a certain number of elements are present per wavelength (usually of the M_2 dominant semi-diurnal tidal species) can be deduced:

$$h_{wl} = \frac{\lambda_{M2}}{\alpha_{wl}} \tag{7}$$

$$h_{wl} = \frac{T_{M2}}{\alpha_{wl}}\sqrt{gb} \tag{8}$$

where λ_{M2} and T_{M2} are the wavelength and period (≈12.42 hours) of the M_2 tidal wave, g is the acceleration due to Earth's gravity, b is the bathymetric depth, and α_{wl} is the user specified number of elements chosen to resolve the wavelength. If the M_2 wavelength is sufficiently captured, the diurnal species will also be sufficiently resolved since their wavelengths are approximately twice as large as the M_2. In general, the wavelength parameter α_{wl} is set to a value somewhere between 25 and 100 (Westerink et al., 1994; Le Provost and Lyard, 1997).

3.3.3 Topographic length scale

The distance, feature size, and/or wavelength mesh size functions can lead to coarse mesh resolution in deeper waters that under resolve and smooth over the sharp topographic gradients that characterize the continental shelf break.Large mesh resolution can emerge in deeper waters with solely the usage of the distance, feature size, and/or wavelength mesh size functions. Larger mesh resolution in deeper waters can lead to under resolving the sharp topographic gradients that characterize the continental shelf break. These slope features can be important for coastal ocean models in order to capture dissipative effects driven by the internal tides, transmissional reflection at the shelf break that control the astronomical tides, and trapped shelf waves (Huthnance, 1995). The scaling of the slope parameter, commonly called the topographic length scale, is usually represented

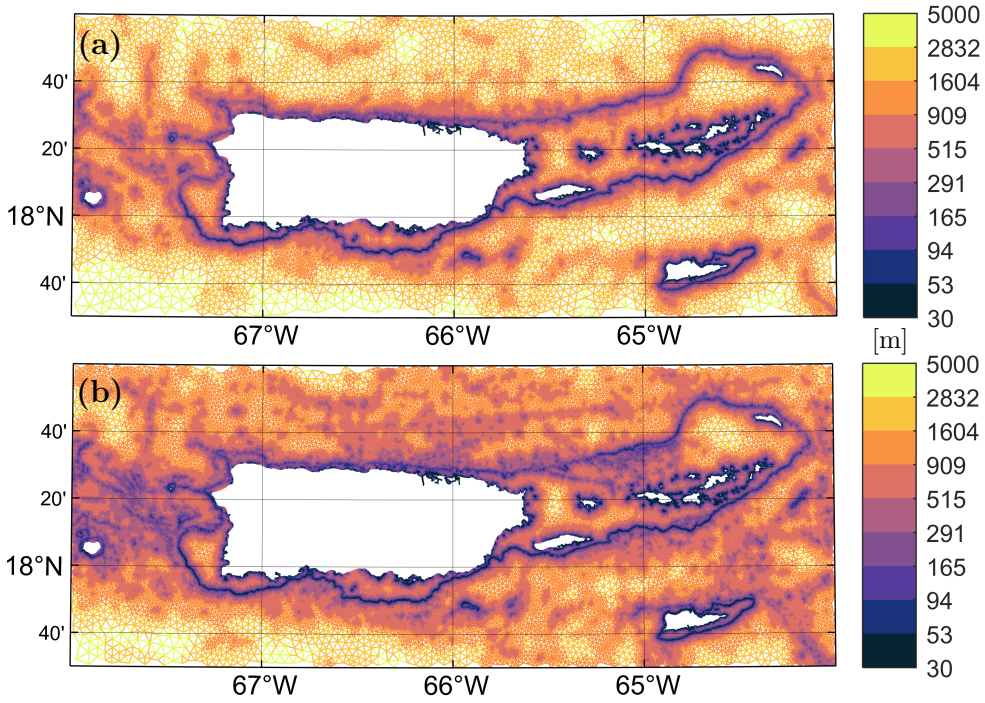

Figure 9. ~~Local~~Mesh resolution (defined as the local element circumradius [m]) in the PRVI example (see Table 1) around the Puerto Rico and U.S. Virgin Island inset region, with (a) and without (b) the Rossby radius slope filter applied. The 'thermal' color palette from cmocean (Thyng et al., 2016) is used in this figure.

by the following:

$$h_{slp} = \frac{2\pi}{\alpha_{slp}} \frac{b}{|\nabla b|} \tag{9}$$

where $2\pi/\alpha_{slp}$ is the number of elements that resolve the topographic slope, and ∇b is the gradient of the bathymetry evaluated on a structured grid of horizontal resolution $h0$. The 2π factor is a convention introduced by Lyard et al. (2006) so that α_{slp} can be set to a value similar in magnitude to α_{wl}, e.g., around 10-30.

Typically the gradient of the bathymetry often contains a high degree of noise, which results in high mesh refinement with the application of h_{slp} despite the fact that small features have marginal effects on shallow water flow, particularly in deep water (LeBlond, 1991). We would like to filter bathymetric features that are not relevant to the underlying shallow water processes, like the astronomical tides. Therefore, we propose to low-pass filter the bathymetry before calculating the gradient ~~by default~~. The ~~In this low-pass filter, we propose a~~ filter cutoff length is based on an estimate of the *local* Rossby radius of deformation:

$$L_R = \frac{\sqrt{gb}}{f} \tag{10}$$

where f is the Coriolis force. By *local* we mean that we discretely bin values of L_R in the meshing domain~~Ω~~ and apply a low-pass filter to those binned grid points with a cutoff set to L_R at the bin midpoint. For this approach to work correctly, partitioning the meshing domain is critical because meshing domain~~Ω can~~ often spans large regions of latitude with highly varying f. Here, the PRVI example (Fig. 1 and Table 1~~and Sect. 3.1.4 for more details on this example~~) is used to demonstrate the effect of the Rossby radius slope filter (Fig. 9). The mesh with the Rossby radius slope filter focuses mesh resolution at large and relatively shallow features such as the continental shelf break avoiding~~avoids~~ the placement of fine resolution over deep and small scale features that are not comparable to L_R. As a result, the ~~filtered~~ mesh with the filtered seabed has 27% fewer vertices in the illustrated region.

3.3.4 Channel thalwegs/polylines

Closer to the shoreline, the width of the nearshore geometry through which water must flow eventually becomes the dominant length scale instead of L_R. Thus, constraints imposed by continuity normally become more important than dynamic balances in determining spatial scales in estuaries~~the estuary~~ (LeBlond, 1991). Following this logic, the representation of the cross-sectional area of the channel that connects the ocean to the estuary is important~~critical~~ in order to simulate an accurate exchange of water.

The predominant conveyance through a watercourse is often ~~is~~ approximated by a series of neighboring points that connect local minimums in bathymetric depth. These locations are referred to collectively as a thalweg and are represented as polylines in the GIS framework. The level of mesh refinement near and around the thalweg can affect the bathymetric representation in the mesh through aliasing local minimums in bathymetric depth. Often the associated length scale of these features is too small to efficently~~successfully~~ resolve through the other~~topographic length scale or wavelength~~ mesh size functions.~~so~~ Instead we propose a mesh size function ~~similar to the feature size constraint function proposed by Heinzer et al. (2012). The purpose of this mesh size function is~~ to locally enhance mesh refinement around thalwegs.~~that would otherwise be under-resolved with the previously proposed mesh size functions (e.g., the wavelength mesh size function which would *reduce* mesh resolution along a deep-draft channel).~~

Thalwegs can be located by thresholding upslope area (O'Callaghan and Mark, 1984) in~~using~~ a DEM ~~along~~ with GIS software such as GRASS. One difficulty with thresholding upslope area to identify submerged channels is that it may produce spurious non-physical channel networks, especially in areas of flat bathymetry. ~~GIS software like GRASS enables the interaction with the vector data and it is encouraged to utilize modules within the GIS software framework (e.g., r.stream.extract, v.stream.network) to ensure that the network is representative of reality prior to~~accurate ~~before meshing.~~

Similar to the feature-constraint algorithm (Heinzer et al., 2012) this mesh size function treats the thalwegs as a set of connected vertices that form polylines and operates on~~ly~~ the polylines that intersect with the meshing domain~~*bbox* polygon~~. The mesh resolution is distributed as follows:~~This mesh size function distributes resolution as follows:~~

1. A circular region in the mesh size function is formed on each thalweg point with a diameter, dia, equal to:

$$dia = 2b\tan(\theta) \tag{11}$$

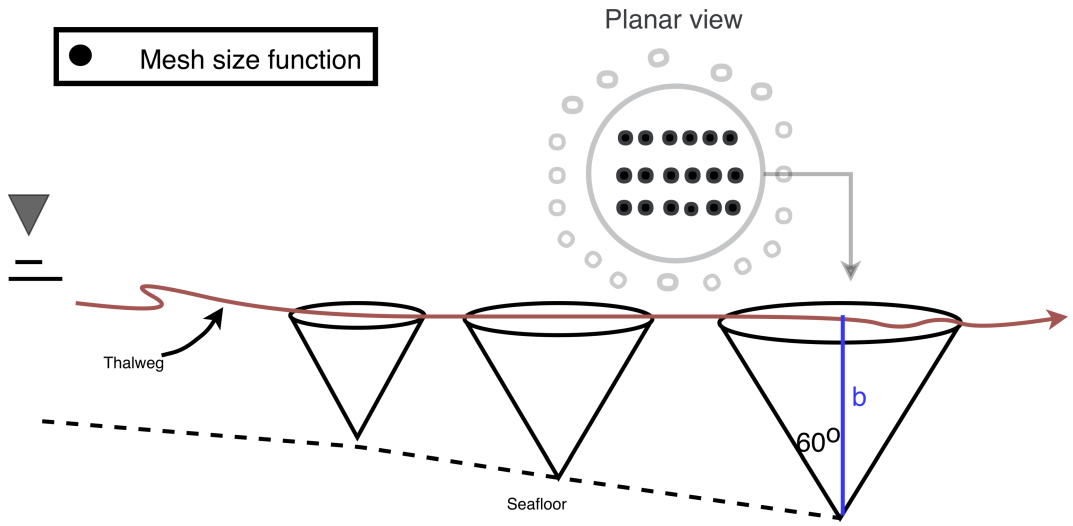

Figure 10. A schematic illustrating various aspects of the channel mesh size function. The thalweg is depicted by~~the~~ the maroon line. Along the ~~some points along~~ thalweg, cones are drawn to depict the regions ~~inside the mesh size function~~ where the mesh size function is altered.

where θ is the angle of reslope.

2. In each circular region, the mesh size function is assigned resolution by

$$h_{ch} = \frac{b}{\alpha_{ch}} \tag{12}$$

This assumes the thalweg has a cross-sectional area that resembles a v-shape with a bank angle of θ (which is set to 60° by default) and that the stencil becomes larger as ~~the~~ b increases (Fig. 10).

As the water column deepens, the horizontal length scale greatly enlarges, which implies that the dynamical effects from small-scale features like thalwegs should weaken. This dynamic is qualitatively captured through Eq. (11) by the enlargement of the thalweg region in the mesh size function as the water depth increases. Additionally, the quotient α_{ch} in Eq. (12) alters how the resolution scales with bathymetric depth to further reflect the fact that the horizontal length scale tends to grow as the water becomes substantially deeper, thus reducing the resolution around thalwegs.

As an example of this mesh size function, a mesh is built in and around Galveston Bay, Houston (GBAY; Fig. 1 and Table 1), ~~. Galveston Bay, located in the vicinity of Houston, Texas, is~~ a shallow-estuary with a heavily trafficked shipping channel along its centerline. In this example, we have provided the thalweg points by thresholding the Galveston DEM (Fig. 1~~Table 1~~) with an upslope area of 10,000 cells using GRASS GIS. Visually, the mesh generated using the channel mesh size function clearly captures the bathymetric feature of the Houston Ship Channel to a higher degree of accuracy (Fig. 11). Without the use of this mesh size function, the model design would be forced to use an extremely high degree of refinement everywhere to capture the Houston Ship Channel and its adjacent features, or to hand-edit the mesh resolution, which, in both cases, are inefficient methodologies.

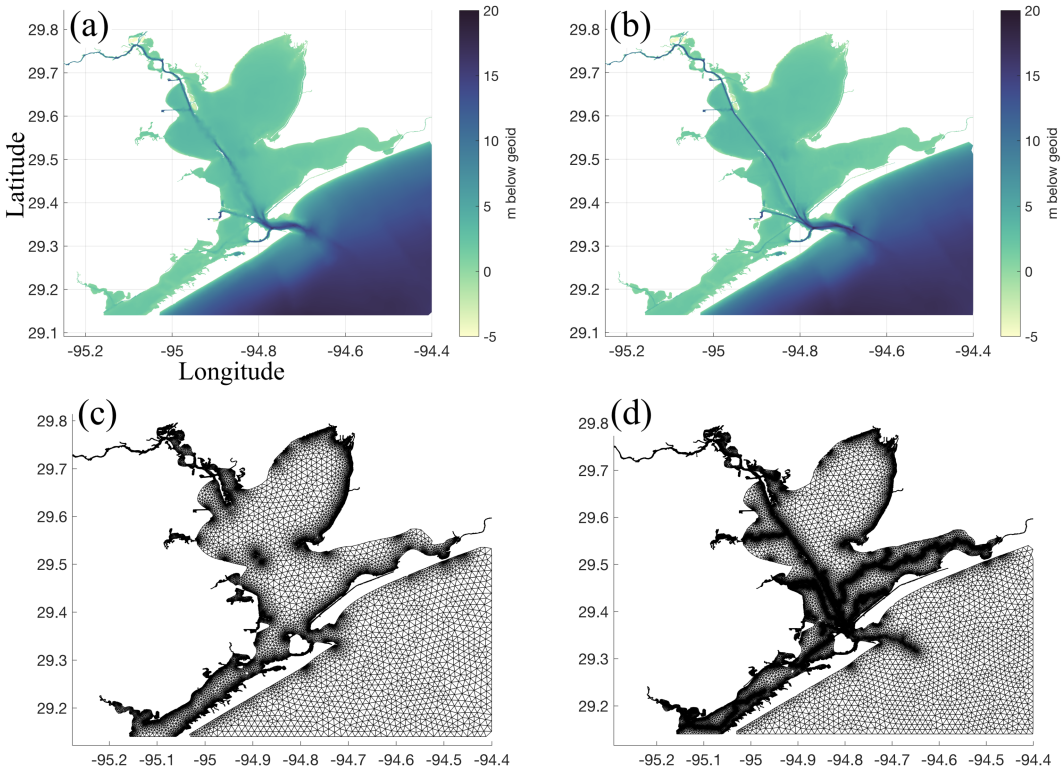

Figure 11. Panels (a) and (c) show the bathymetry and mesh connectivity in the GBAY example (Fig. 1 and Table 1) created without the thalweg mesh size function enabled; panels (b) and (d) are with the thalweg mesh size function enabled. The 'deep' color palette from cmocean (Thyng et al., 2016) is used in palettes (a) and (b).

3.3.5 Finalizing the mesh size function

The final mesh size function, h, is determined by applying the minimum function to the set of individual local mesh size functions, i.e.,

$$h = \min\left[(h_{dis} \text{ or } h_{lfs}), h_{wl}, h_{slp}, h_{ch}\right] \tag{13}$$

Note that it is possible to operate on any given subset of mesh size functions. Following this h is further refined based on mesh size transition bounds (Sect.3.3.6), Courant-Friedrichs-Lewey limiting (Sect.3.3.7), and global user-defined maximum (h_{max}) and minimum ($h0$) mesh size bounds.

3.3.6 Mesh size transitions~~Grading and mesh size interactions~~

It is necessary to ensure a size smoothness limit α_g such that for any two adjacent vertices $\boldsymbol{x_i}$, $\boldsymbol{x_j}$ connected by an edge, the local increase in mesh size is bounded above such that:

$$h(\boldsymbol{x_j}) \leq h(\boldsymbol{x_i}) + \alpha_g ||\boldsymbol{x_i} - \boldsymbol{x_j}|| \tag{14}$$

The mesh size gradation is enforced with the marching method that was introduced in Sect. 3.1.4. A smoothness criteria is essential to produce a mesh that can simulate physical processes with a practical time step as sharp gradients in mesh resolution typically lead to triangles with highly skewed angles that results in low~~inevitably resulting in poor~~ numerical accuracy (Shewchuk, 2002). ~~We adopt a marching method to smooth the mesh size function $h(\boldsymbol{X})$, originally proposed by Persson (2006) and later adapted by Engwirda (2014, 2017), that preserves the original gradients of the mesh size function while bounding~~ ~~We have further adapted this algorithm for support on structured grids in MATLAB (implemented in *limgradStruct.m* function).~~ In general, a smoother edgelength function is congruent with a higher overall triangle quality but with more triangles in the mesh. ~~Generally, setting $0.2 \leq \alpha_g \leq 0.3$ produces good results without over-resolving the mesh.~~ It is important to note that the rate of mesh resolution increase is bounded above by the grade; therefore, if the distance parameter in Eq. (5) is set to a value lower than the grade ($\alpha_d < \alpha_g$), then grading should have no effect on the mesh size function.

Here we demonstrate the relative effects of the *distance* and *feature* mesh size functions and their interaction with the grade. To illustrate this mesh size function, we built ~~a~~mesh is created over an estuary-like geometry with *distance* ($\alpha_d = 0.15$ and $\alpha_d = 0.35$) and *feature* ($\alpha_R = 3$ and $\alpha_R = 6$) mesh size functions, each using two different gradation bounds~~grading parameters~~ ($\alpha_g = 0.15$ and $\alpha_g = 0.35$) (Fig. 12). The *distance* mesh size function focuses resolution on the mesh boundary, which is often not necessary to resolve areas that are geometrically simple. Further, the use of a *distance* mesh size function often results in comparatively larger triangles in the center of the geometry; especially with a relatively high grade (i.e., 35%; Fig. 12(d)). In shallow estuaries, this can be undesirable as the bathymetric representation of high conveyance channels in the center of the estuary will be inaccurate, aliasing the transported mass and energy. In contrast, the *feature* mesh size function places~~tries to place~~ a uniform number of triangles across the widest axis of the feature (Fig. 12(e)-(f)). It focuses mesh resolution on regions that are narrow and/or where the shoreline has high curvature. The net result is a comparatively smaller number of vertices than the *distance* mesh size function (for $\alpha_R < 20$ in this example). Depending on the selection of α_R in the local feature size equation Eq. (6), the size of the mesh resolution in the center of the estuary can be bounded even when using a relatively high mesh grade ($\alpha_g > 0.25$). This is advantageous because a higher grade can dramatically lower the overall vertex count. Conversely, a relatively low grade ($\alpha_g < 0.20$) can hinder the *feature* mesh size function from expanding efficiently, and may be somewhat counter-productive to its purpose.

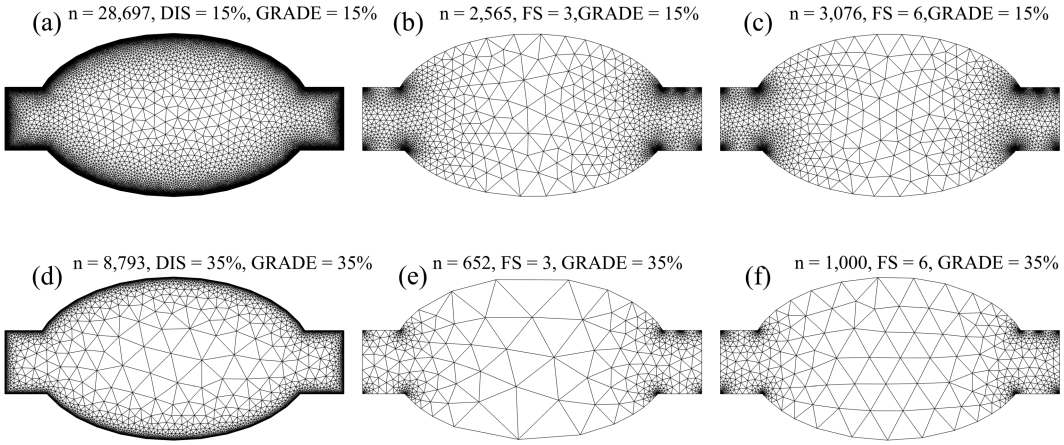

Figure 12. Depiction of mesh resolution interactions between the grade (α_g), *distance* (DIS), and *feature* (FS) mesh size functions. Panels (a)-(c) depict the resolution with a grade equal to 15% (α_g = 0.15), panels (d)-(f) with a grade equal to 35% (α_g = 0.35). The first column depicts how mesh resolution is distributed with a *distance* mesh size function and the second and third columns show how the mesh size varies with the *feature* mesh size function with α_R equal to 3 and 6, respectively. In the title of each panel, the number of vertices n in the triangulation is shown.

3.3.7 Courant-Friedrichs-Lewey (CFL)-limiting

The computational expense of a computational model and associated code is significantly affected by the time step that must be used to ensure stability and accuracy. For models that use explicit time stepping schemes, a necessary condition for numerical stability is determined by the Courant-Friedrichs-Lewey (CFL) condition. Although this is not a sufficient condition, it is a practical way to gauge the success of a new mesh. The CFL condition states that the Courant number (Cr) must be less than or equal to 1. Stricter conditions may be relevant for different numerical schemes and due to nonlinearities in the governing equations ~~and the amount~~ (Brufau et al., 2004). Additionally, the accuracy of a numerical scheme is ~~also~~ impacted by the Cr as high values tend to give poorer accuracy even in implicit solvers. ~~In a similar approach to Bilgili et al. (2006),~~ For the application of solving the shallow water equations the Cr can be estimated and bounded in the mesh (Bilgili et al., 2006). We define an estimate of Cr at the vertices of the mesh by adding the shallow water wave speed with an estimate of the anticipated flow speed:

$$Cr = \frac{(u + \sqrt{gH})\Delta t}{\Delta X} \tag{15}$$

where u is the magnitude of the flow, g is the acceleration due to gravity, H is the total water depth, Δt is the time step, and ΔX is the element size or the shortest connected edge to each vertex. Since the wave speed is a function of depth and ΔX is equivalent to the mesh size, ~~h~~~~(X)~~, the user can estimate the CFL condition *a priori* for a given Δt. Note that to obtain this *a priori* estimate of Cr in Eq. (15), we set $H \approx b$, and approximate the flow speed with the long wave linear orbital velocity, i.e., $u \approx \eta\sqrt{g/b}$, where η is the wave amplitude which we set to 1 m by default. Applying these approximations and rearranging

gives the following CFL-limiting condition on the mesh sizeedgelength:

$$h \geq \frac{(\eta\sqrt{g/b}+\sqrt{gb})\Delta t}{Cr} \tag{16}$$

where b is set to a minimum of 1 m to allow for the CFL condition to be satisfied overland in the case inundation were to occur.

Thus, the user can specify a value of Δt to bound the mesh resolution based on some value of $Cr < 1$. The aim of CFL-limiting is to help facilitate a mesh to be simulated with a certain time step when using explicit time stepping numerical models. However, this often comes with a loss of mesh resolution that may be critical for resolving important conveyances, so the user must consider reasonable values of Δt based on the minimum edgelength. To avoid this choice, we have also implemented an option (see the user guide for details on how to invoke) that automatically selects a suitable Δt that satisfies the condition Eq. (16) for the h_{dis} or h_{lfs} (whichever is induced) mesh size functions. The purpose of this is to preserve the nearshore resolution while applying the CFL-limiting to other mesh size functions that may give higher refinement offshore.

To demonstrate the CFL-limiting option, we return to the JBAY example (Fig. 1 and Table 1)mesh of Jamaica Bay in New York, United States, generated using the *feature* mesh size function. In one rendition of the mesh, no CFL-limiting is used (T_{woCFL}), in another rendition, CFL-limiting with Δt = 2 s (T_{wCFL}) is invoked. In the generation of T_{wCLF}, the mesh size function is conservatively bounded by $Cr = 0.5$ to allow a buffer for the effects of nonlinearities, bathymetric interpolation, and mesh smoothing. After the mesh is generated, the NCEI Post-Sandy DEM is interpolated onto each vertex using a cell-averaging approach (see *interp* method in Sect. 3.4function in user guide), and the resulting CFL is calculated by Eq. (15) with $\Delta t = 2$ s. The use of the CFL-limiter acts to shift the distribution of Cr to smaller values (Fig. 13). The maximum Cr decreases from 3.64 to 0.96 and the mean Cr shifts from 0.68 to 0.36. In the mesh with the CFL-limiting, there are no vertices that violate the CFL condition as compared to 10,492 in the mesh without it. CFL-limiting thus reduces the number of vertices by locally coarsening *hh(X) in fact* (T_{wCFL} has 45.6% fewer vertices than T_{woCFL}). Again, the user must be careful when selecting Δt as CFL-limiting may lead to choke points in small channels nearshore which are generally the first areas that violate the CFL (Fig. 14). Depending on the application this may or may not be tolerable.

Although the above example demonstrates that the Cr of all vertices is reduced to under 1 when using the CFL-limiting mesh size function, the maximum Cr is still 0.96 for a $\Delta t = 2$ s, which may be too close to the theoretical condition to simulate without instabilities. Based on our experience we need to impose a stricter CFL condition such as $Cr < 0.5$ to ensure numerical stability, accuracy, and to minimize numerical artifacts. To ensure that this more conservative condition is fully satisfied, we propose the *CheckTimestep* post-processing function (Algorithm 1). This function iteratively deletes vertices in the mesh associated with edges that violate the CFL. With each deletion, the vertices on the outer fan containing all the connected elements are smoothed using the Laplacian operator. The algorithm relies on MATLAB's implementation of the Bowyer-Watson incremental Delaunay triangulation to preserve the mesh connectivity outside of the modification patch. For example, in the JBAY example with CFL-limiting, *CheckTimestep* converged after 5 iterations resulting in a mesh with approximately 2,240 less vertices but one that fully satisfied $Cr < 0.5$ everywhere for $\Delta t = 2$ s. In addition to ensuring the CFL condition is

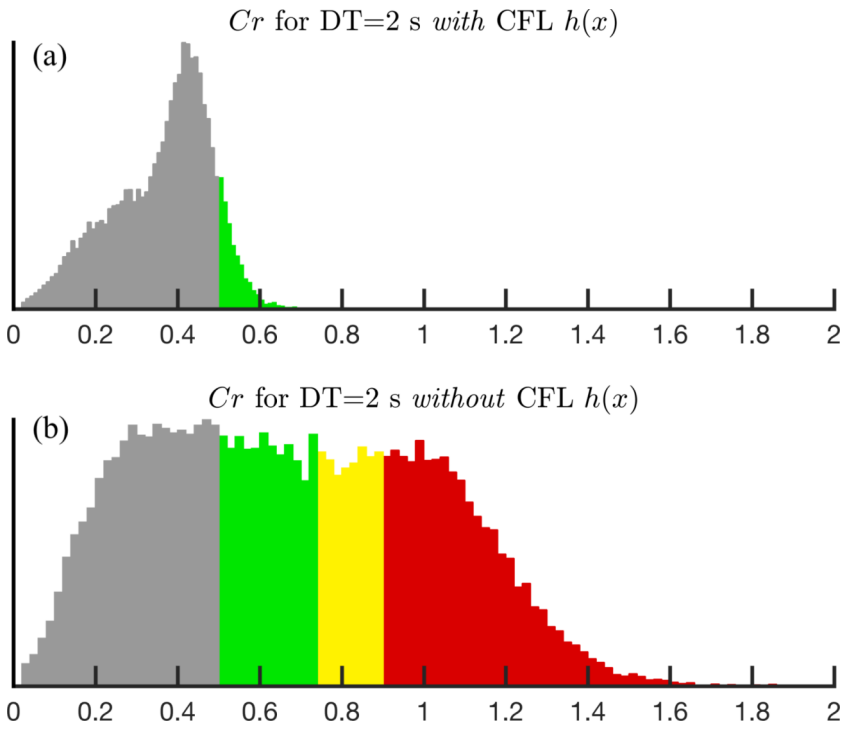

Figure 13. Panel (a) illustrates the effect of CFL-limiting on the Courant number Cr when constructing the JBAY example (Fig. 1 and Table 1) and (b) without it. The colored bars indicate the vertices with $Cr > 0.5$ and are shown to assist in the comparison of histograms.

fully met, *CheckTimestep* in practice is often used to explore how the mesh would have to be modified in order to achieve a stable simulation for a particular Δt.

Algorithm 1 Incrementally adapts a triangulation of points p and connectivity matrix t with bathymetric data b defined at p to a given timestep Δt in seconds through vertex decimation.

1: **Function** (p,t) = $CheckTimestep(p, t, b, \Delta t)$
2: Form nearest neighbor bathymetric interpolant with p, t, and b.
3: Calculate Cr at all vertices using Eq. (15) given b, Δt, and the shortest connected edge to the vertex.
4: If $Cr \leq 0.5 \; \forall \; p$, then exit.
5: Otherwise, determine point set p_v with $Cr > 0.5$
6: Incrementally delete p_v from t using Bowyer-Watson algorithm.
7: Apply Laplacian operator to the patch around each p_v containing the connected triangles.
8: Re-interpolate b onto p using nearest-neighbor interpolant and proceed back to 3.

3.4 Mesh container: *msh* class

~~It is often difficult, if not impossible, to recreate the vertex locations since the geospatial data used to define the mesh domain and the description of how resolution is distributed is often not stored with the mesh (Candy and Pietrzak, 2018). A scripting-based~~

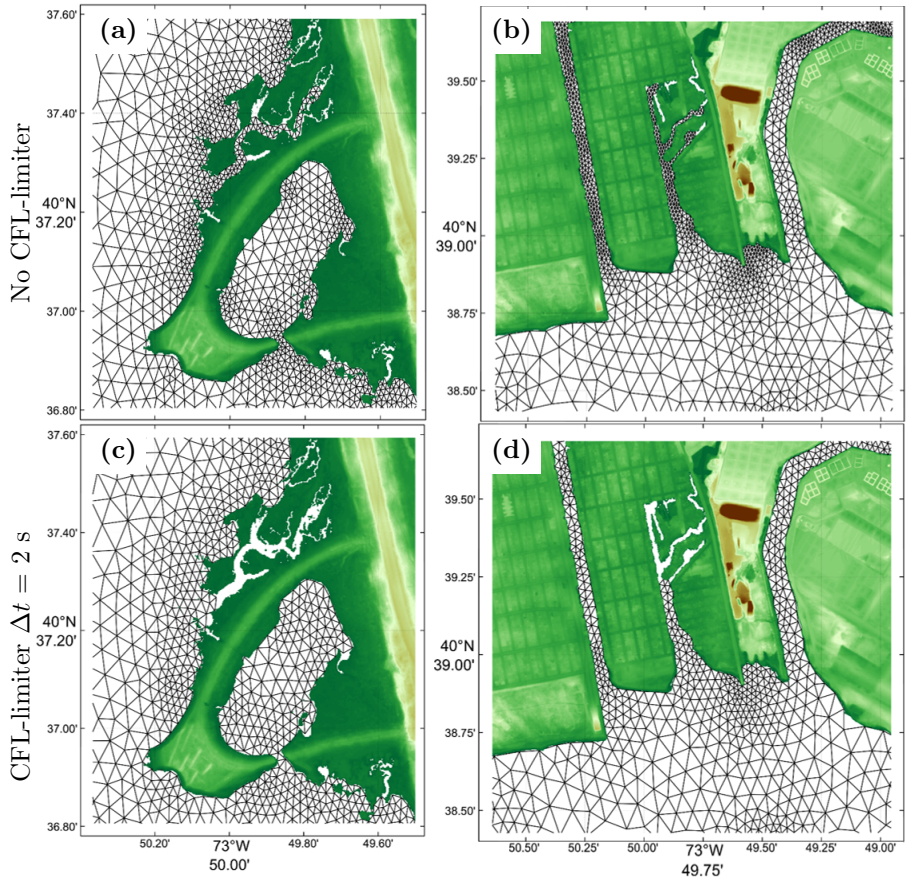

Figure 14. Selected closeup regions in Jamaica Bay, New York (left (a) and (c): West Pond, Queens; right (b) and (d): Old Howard Beach, Queens) of the mesh connectivity built with the JBAY example script (Fig. 1 and Table 1). Top panels (a) and (b) show the mesh connectivity without invoking the CFL-limiter, and the bottom panels (c) and (d) show it when using the CFL-limiting option with $\Delta t = 2$ s.

~~approach to mesh generation forces the user to create an input file with the options, parameters, and geospatial data that was used in the mesh generation process. Thus, since the mesh can be built on any machine with resolution identically, we envision that the script file used to build the mesh can be simply provided as supplementary material with any related published work in order to promote automation and transparency.~~

To store the triangulation and related files, the *msh* data storage class contains triangulation-related attributes and support for solver-specific input files~~The format of the *msh* class uses MATLAB's dot-structure syntax that naturally enables option-hierarchy, organizes the numerous associated data files in one place, and simplifies the interaction with the underlying data by creating a set of standardized methods(Table ??). Upon termination of the mesh generator, a *msh* class object containing the triangulation is returned and can be saved efficiently to hard disk as a MATLAB .mat file.~~. While the underlying purpose of the *msh* class is to store the mesh data, the OOP framework enables specific methods to be associated with it~~the class~~. This

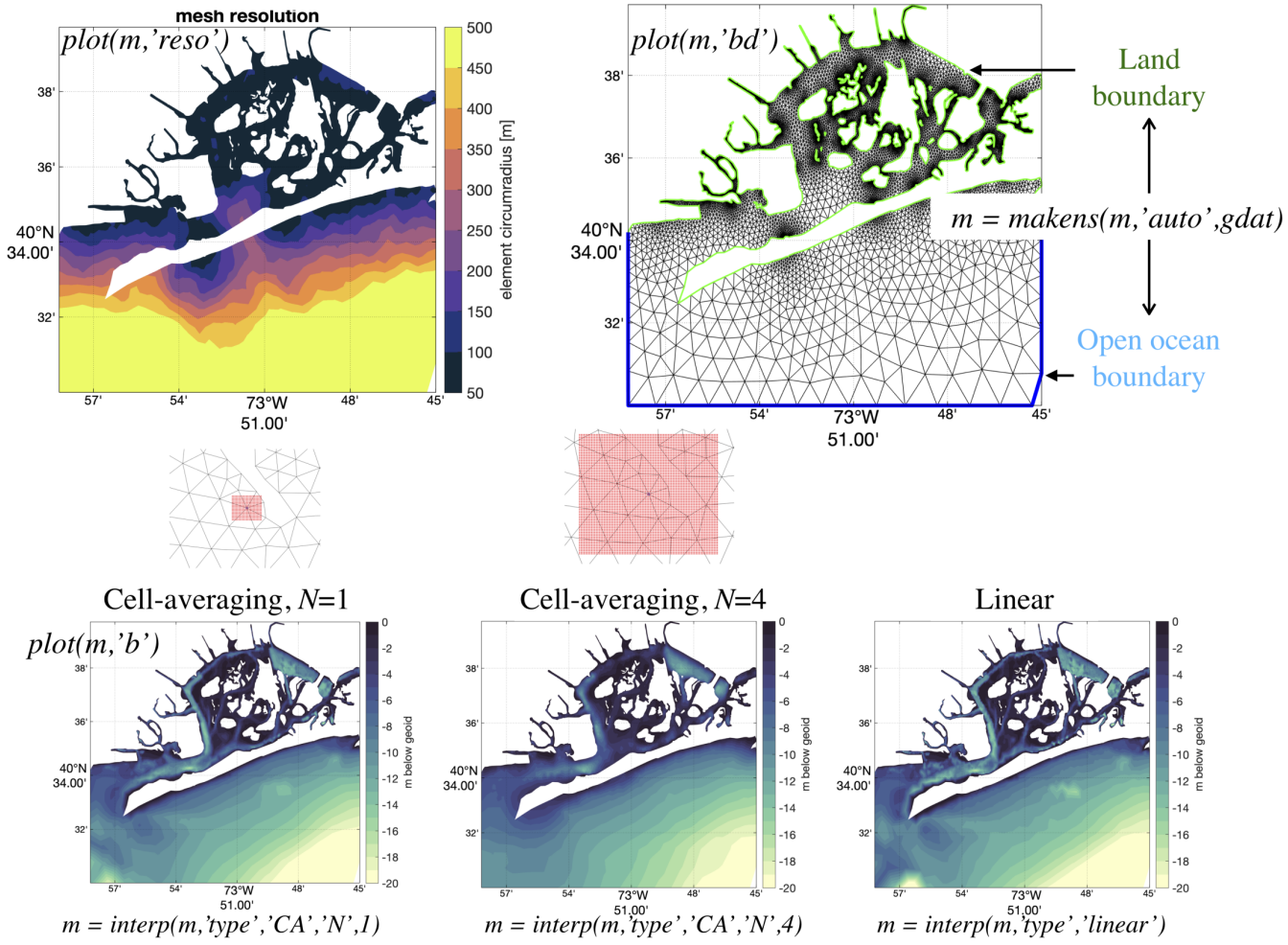

Figure 15. Illustration of key *msh* methods: *plot* can be used to visualize mesh resolution (top-left), mesh triangulation and boundary types (top-right), and seabed topography (bottom row); *interp* interpolates seabed topography onto the mesh using cell-averaging or built-in *griddedInterpolant* methods (bottom row); *makens* classifies mesh boundary vertices into land and open ocean types automatically using the native *geodata* class (top-right).

enables the *msh* class to act as an intermediary between the numerical solver and the user to assist the creation of solver-specific files and perform common data-driven operations on the mesh.

A substantial effort is often required after the triangulation is constructed to enable simulation with a coastal ocean solver such as~~like~~ ADCIRC, FVCOM, SELFE, or SCHISM. For example, the mesh often needs to be visualized and quality checked, boundary conditions must be specified, and seabed topography must be interpolated onto the mesh~~es~~ vertices (Fig. 15). 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 that can be edited by everyone using a version control software.

Figure 15 illustrates a few of the key methods associated with the *msh* class (see the user guide for a complete list) that we have implemented, such as the visualization of mesh triangulation, resolution, seabed topography, and boundary types. Further, a standardized method for interpolating seabed topography, which employs a generalized cell-averaging approach by default, has been developed. The method can also be used as a wrapper to the built-in MATLAB *griddedInterpolant* function with nearest, linear, and various higher-order interpolation methods. Comparison of the mesh seabed topography using cell-averaging and linear interpolation methods is shown along the bottom row in Fig. 15. Also included is a *msh* method to automatically classify mesh boundary vertices into open ocean, enclosed islands, and mainland types based on the native *geodata* class (top-right in Fig. 15).

~~For instance, there are a set of procedures that must be applied to a mesh to ensure it's suitable for numerical simulation such as renumbering the vertices, visual inspection of the mesh connectivity, topo-bathymetric interpolation, boundary condition application, and control file generation the user guide(see Table ?? for the complete list of *msh* methods). 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 that can be edited by everyone using a version control software.~~

4 Mesh Generation Wall-Clock Time

The total and component-based wall-clock times for generating each of the three examples presented in this study is shown in Table 3. Overall, the small examples (JBAY and GBAY) complete in under 2 minutes, and the large PRVI example takes approximately 45 minutes. Consistently for all examples vertex relocation consumes slightly more time than Delaunay triangulation, and the mesh cleaning (post-processing improvement strategies) accounts for approximately 6% of the total time. The relative balance between mesh generation and pre-processing times depends on the resolution of the shoreline and the size of the meshing domain. For example, in the small domain problems (JBAY and GBAY), the pre-processing time makes up roughly a quarter of the total time. In contrast, in the PRVI example which meshes most of the north western Atlantic ocean using four separate *geodata* and *edgefx* classes, the pre-processing time accounts for 64% of the total time.~~Therefore, while it is likely possible to speed-up the mesh generation process through e.g., parallel Deluanay triangulation and/or different approaches to the initial point rejection in the *DistMesh* algorithm, for large and complex meshes intelligent use of the multiscale meshing approach combined with parallelization of the construction of the individual *edgefx* and *geodata* classes is likely to result in the greatest speedup.~~

5 Discussion and conclusions

A self-contained model development toolkit to automate the generation of two-dimensional (2D) triangular~~reate~~ unstructured meshes ~~composed of two-dimensional (2D) triangular elements~~ for coastal ocean models was developed. The overarching goal of the software is to reduce the complexity and hours spent constructing real-world unstructured meshes to the degree that it allows one to more carefully and systemically study the impact on the coastal circulation.~~by modifying the~~

Table 3. Wall-clock time in seconds (% of total) for the steps involved in mesh generation. The pre-processing includes reading and processing the geospatial data (in *geodata*) and forming the mesh size functions (in *edgefx*). The vertex relocation timings include the time elapsed in the initial point rejection, vertex re-projection back into the meshing domain~~Ω~~, vertex movement, and mesh improvement strategies during mesh generation (Sect. 3.1.2~~in *meshgen.build*~~). The cleaning category includes the time spent on mesh improvement strategies after mesh generation (Sect. 3.1.3)~~in the methods in *meshgen.clean*~~).

		Mesh generation time			
Example	**Pre-processing time**	**Vertex relocation**	**Triangulation**	**Cleaning**	**Total Time**
JBAY	13.7 (22.5%)	24.9 (41.0%)	18.3 (30.0%)	3.83 (6.25%)	60.8
GBAY	23.9 (25.1%)	34.1 (35.8%)	31.4 (33.0%)	5.79 (6.09%)	95.2
PRVI	1,770 (64.1%)	498 (18.1%)	316 (11.5%)	174 (6.31%)	2,760

~~*DistMesh2D* mesh generation algorithm for the type of geophysical domains encountered in coastal ocean problems.~~ This is achieved through a standardized scripted workflow comprising pre- and post- processing steps of geospatial datasets and mesh properties, which are performed by four dedicated classes.~~This software is expressed as an objected-oriented approach composed of four MATLAB classes. that complement~~Each class was designed other to simplify the necessary pre- and post-processing procedures for mesh generation leading to a self-contained model development tool.The overarching goal of the software is to reduce the complexity and hours spent constructing real-world unstructured meshes to the degree that it allows one to more carefully and systemically study the impact on the coastal circulation.attributed to mesh refinement.~~ While the scripting based approach used to generate meshes promotes automation and approximate reproducibility, the pointers contained within the script do not adequately describe provenance attribution of the geospatial datasets and computing environment used. In the future, employing formal Research Data Management practices in the context of geophysical mesh generation (Avdis et al., 2018; Candy and Pietrzak, 2018) into *OceanMesh2D* would be beneficial to heighten reprodubility.

A set of common coastal ocean relevant mesh size functions were built into the mesh size function class (*edgefx*) that can handle a variety of user-based constraints and facilitate~~ensure~~ the approximate reproducibility of mesh vertex locations~~mesh connectivity can be approximately reproduced on a personal computer~~. The implementation of these mesh size functions were largely borrowed from pre-existing literature with some minor enhancements. We presented a polyline mesh size function to locally enhance resolution around and near marine navigation channels and deep-draft channels (i.e., thalwegs). These features are~~were~~ found by thresholding upslope-area calculated from a digital elevation model (DEM) using GIS software. The polyline mesh size function may have interesting future applications for the development of overland meshes that seamlessly mate with ocean meshes. For example, the user could provide a set of lines that characterize overland ridges so that the polyline mesh size function can be used to locally enhance mesh resolution to better capture the local maximums in the topographic heights. Since the representation of the inter-tidal and floodplain zone in the mesh is critical for coastal flooding applications, ensuring overland features like hills and levees are correctly represented in the mesh is a important feature. In its current state, the toolbox is able to constrain piecewise linear segments that may represent e.g., a series of levees; however, if there is a large degree of disparity between the point spacing on the constraints and the mesh size function, then the resulting mesh will be of poor quality.

To ensure that a mesh ~~is would be~~ computationally stable with a user-requested time step (relevant when simulating with explicit/semi-implicit numerical models), a CFL-limiting mesh size function similar to Bilgili et al. (2006) was introduced. In this approach, we estimate the Courant (Cr) number based on shallow water wave theory and ensure that the final mesh size function satisfies the CFL condition ($Cr < 1$). Although applying CFL-limiting to the mesh size function was shown to help encourage stability by lowering the Cr, the resulting unstructured mesh may not necessarily satisfy the CFL condition due to that fact that bathymetric interpolation from the DEM is not easily constrained. Thus, an iterative algorithm to be applied *after* the mesh was developed (*CheckTimestep*) to locally alter the connectivity by decimating vertices that violate the CFL condition. Depending on the users choice of time step and the various mesh size constraints, the algorithm decimates vertices in certain regions (e.g., small constricted channels) that may or may not be tolerable for the problem at hand. In such regions, anisotropic mesh elements (e.g., Piggott et al., 2009) that could be generated using mesh size functions which include a directional component may be more beneficial than isotropic equilateral elements. Thus, implementing anisotropic mesh size functions into the software, along with the testing of the resultant meshes in real coastal ocean problems, is an interesting direction for future work.

We emphasized the expensive nature of building large-scale high-fidelity mesh size functions which motivates the use of a multiscale meshing approach. This approach reflects the often sparse spatial coverage and heterogeneous nature of freely available digital elevation data that are often used in the construction of the mesh size functions. Multiscale meshing allows the user to build (extremely) high-resolution local mesh size functions that are embedded in larger scale ocean domains. The end result is a mesh that seamlessly transitions from the high refinement region to coarser elements outside the region of interest. This is practically useful to accurately model coastal flooding in small regions (e.g., a city or a small island – here we show an example of the approach with the mesh refinement region around Puerto Rico and the U.S. Virgin Islands) that may be susceptible to storms and tropical cyclones (TC) passing over it. For large-scale TC-driven storm surge events, it has been shown that a large model domain is essential to capture the pre-event conditions that can alter the modeled severity of the event (Blain et al., 1994). In forecasting scenarios, the multiscale meshing approach could be used to mesh around the predicted land-falling region based on the cone of uncertainty of the path of the storm to locally higher resolution. This approach could generate meshes for the prediction of coastal flooding on-the-fly as new forecast data becomes available. Given the local nature of the mesh refinement in this approach, these meshes could be computationally more efficient with smaller minimum element sizes than pre-existing ones, e.g., the U.S. National Ocean Service's (NOS) Hurricane Storm Surge Operational Forecast System (HSSOFS) mesh (Technology Riverside Inc. and AECOM, 2015), which cover entire swaths of coastline with medium level resolution.

The objected-oriented structure of the software enables each component to be used in isolation and/or under workflows different to that presented here. For instance, mesh size functions constructed through the *edgefx* class could be used with other mesh generators to distribute vertices. Furthermore, the ability of *OceanMesh2D* to automatically adapt user-supplied shoreline datasets to a mesh size function is a new feature to the authors' knowledge. This ability could be used as a standalone feature to produce polygonal boundaries that approximate the shoreline with a variety of spatial constraints for other mesh generator packages or GIS applications.

Three examples were used for demonstration in this study (Fig. 1 and Table 1). A further three separate examples are illustrated in the user guide (Roberts and Pringle, 2018). All six examples are released with this version of the OceanMesh2D package. They can be used to become familiar with the software, for testing purposes, and as templates for scripts used to generate the user's custom mesh.

Code availability. The OceanMesh2D mesh generator toolbox is hosted on the following GitHub page: https://github.com/CHLNDDEV/OceanMesh2D. The version release presented in this paper is available as a Zenodo archive: https://doi.org/10.5281/zenodo.1341385. The software requires no paid MATLAB toolboxes to generate meshes; however, some auxiliary functions (e.g., those that create ADCIRC input files) not used in primary workflows may. A user guide (Roberts and Pringle, 2018) and a suite of examples is available from the main GitHub page. All components of the OceanMesh2D toolbox are free software, being released under the GNU General Public License version 3.0. Full details of the license, including the compatible copyright notices of third party routines included in the package, are provided in the LICENSE file in the source distribution.

Author contributions. KR designed the framework of the software. KR and WP equally contributed to the coding and development of the software, design and testing of the provided examples, and the preparation of the manuscript and its associated figures. JW provided the research environment and intellectual discussion necessary for the software's development and eventual realization of this manuscript.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. We thank Dr. Darren Engwirda at Columbia University in the City of New York for the useful functions that are available through the MathWorks website. Our gratitudes to Dr. Chris Massey at US Army Corps of Engineers ERDC Coastal and Hydraulics Laboratory for his function to bound the vertex connectivity. The m_map (https://www.eoas.ubc.ca/~rich/map.html) and cmocean (Thyng et al., 2016) (https://matplotlib.org/cmocean/) toolboxes are widely used in plotting and mapping related routines within OceanMesh2D, for which we are grateful for the authors' work. We appreciate the valuable feedback provided by Dr. Darren Engwirda and Professor Per-Olof Persson at the University of California, Berkeley. The authors wish to thank Dr. Damrongsak Wirasaet at the University of Notre Dame for many helpful discussions and his comments on an initial draft that lead to an improvement in the quality of this manuscript. This work was supported in part by the National Science Foundation under grant ACI-1339738.

References

Arya, S. and Mount, D. M.: Approximate Nearest Neighbor Queries in Fixed Dimensions, in: Proc. 4th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 271–280, 1993.

Avdis, A., Candy, A. S., Hill, J., Kramer, S. C., and Piggott, M. D.: Efficient unstructured mesh generation for marine renewable energy applications, Renewable Energy, 116, 842 – 856, https://doi.org/https://doi.org/10.1016/j.renene.2017.09.058, http://www.sciencedirect.com/science/article/pii/S0960148117309205, 2018.

Babuška, I. and Aziz, A.: On the Angle Condition in the Finite Element Method, SIAM Journal on Numerical Analysis, 13, 214–226, https://doi.org/10.1137/0713021, 1976.

Bagon, S.: Matlab class for ANN, available at http://www.wisdom.weizmann.ac.il/~bagon/matlab.html, 2009.

Balendran, B.: A Direct Smoothing Method for Surface Meshes, in: Proceedings of the 8th International Meshing Roundtable, South Lake Tahoe, California, USA, October 10-13, 1999, pp. 189–193, http://imr.sandia.gov/papers/abstracts/Ba142.html, 1999.

Bilgili, A., Smith, K. W., and Lynch, D. R.: BatTri: A two-dimensional bathymetry-based unstructured triangular grid generator for finite element circulation modeling, Computers and Geosciences, 32, 632 – 642, https://doi.org/10.1016/j.cageo.2005.09.007, 2006.

Blain, C. A., Westerink, J. J., and Luettich, R. A.: The influence of domain size on the response characteristics of a hurricane storm surge model, Journal of Geophysical Research, 99, 467–479, https://doi.org/10.1029/94JC01348, 1994.

Brown, J. M., Norman, D. L., Amoudry, L. O., and Souza, A. J.: Impact of operational model nesting approaches and inherent errors for coastal simulations, Ocean Modelling, 107, 48–63, https://doi.org/10.1016/j.ocemod.2016.10.005, 2016.

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

Canann, S. A., Stephenson, M. B., and Blacker, T.: Optismoothing: An optimization-driven approach to mesh smoothing, Finite Elements in Analysis and Design, 13, 185 – 190, https://doi.org/10.1016/0168-874X(93)90056-V, 1993.

Candy, A. and Pietrzak, J.: Shingle 2.0: Generalising self-consistent and automated domain discretisation for multi-scale geophysical models, Geoscientific Model Development, pp. 213–234, https://doi.org/10.5194/gmd-11-213-2018, 2018.

Conroy, C. J., Kubatko, E. J., and West, D. W.: ADMESH:An advanced, automatic unstructured mesh generator for shallow water models, Ocean Dynamics, 62, 1503–1517, https://doi.org/10.1007/s10236-012-0574-0, 2012.

Debreu, L., Marchesiello, P., Penven, P., and Cambon, G.: Two-way nesting in split-explicit ocean models: Algorithms, implementation and validation, Ocean Modelling, 49-50, 1–21, https://doi.org/10.1016/j.ocemod.2012.03.003, 2012.

Engsig-Karup, A. P., Hesthaven, J. S., Bingham, H. B., and Warburton, T.: DG-FEM solution for nonlinear wave-structure interaction using Boussinesq-type equations, Coastal Engineering, 55, 197 – 208, https://doi.org/10.1016/j.coastaleng.2007.09.005, 2008.

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

Engwirda, D.: JIGSAW-GEO (1.0): Locally orthogonal staggered unstructured grid generation for general circulation modelling on the sphere, Geoscientific Model Development, 10, 2117–2140, https://doi.org/10.5194/gmd-10-2117-2017, 2017.

Gorman, G., Piggott, M., Pain, C., de Oliveira, C., Umpleby, A., and Goddard, A.: Optimisation based bathymetry approximation through constrained unstructured mesh adaptivity, Ocean Modelling, 12, 436 – 452, https://doi.org/10.1016/j.ocemod.2005.09.004, 2006.

Gorman, G., Piggott, M., Wells, M., Pain, C., and Allison, P.: A systematic approach to unstructured mesh generation for ocean modelling using GMT and Terreno, Computers and Geosciences, 34, 1721–1731, https://doi.org/10.1016/j.cageo.2007.06.014, 2008.

Gorman, G. J., Piggott, M. D., and Pain, C. C.: Shoreline approximation for unstructured mesh generation, Computers and Geosciences, 33, 666–677, https://doi.org/10.1016/j.cageo.2006.09.007, 2007.

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

Greenberg, D. A., Dupont, F., Lyard, F. H., Lynch, D. R., and Werner, F. E.: Resolution issues in numerical models of oceanic and coastal circulation, Continental Shelf Research, 27, 1317 – 1343, https://doi.org/10.1016/j.csr.2007.01.023, recent Developments in Physical Oceanographic Modelling: Part IV, 2007.

Hagen, S. C., Horstmann, O., and Bennett, R. J.: An Unstructured Mesh Generation Algorithm for Shallow Water Modeling, International Journal of Computational Fluid Dynamics, 16, 83–91, https://doi.org/10.1080/10618560290017176, 2002.

Heinzer, T. J., Williams, M. D., Dogrul, E. C., Kadir, T. N., Brush, C. F., and Chung, F. I.: Implementation of a feature-constraint mesh generation algorithm within a GIS, Computers and Geosciences, 49, 46 – 52, https://doi.org/10.1016/j.cageo.2012.06.004, 2012.

Huthnance, J. M.: Circulation, exchange and water masses at the ocean margin: the role of physical processes at the shelf edge, Progress in Oceanography, 35, 353 – 431, https://doi.org/10.1016/0079-6611(95)80003-C, 1995.

Koko, J.: A Matlab mesh generator for the two-dimensional finite element method, Applied Mathematics and Computation, 250, 650–664, https://doi.org/10.1016/j.amc.2014.11.009, 2015.

Lambrechts, J., Comblen, R., Legat, V., Geuzaine, C., and Remacle, J.-F.: Multiscale mesh generation on the sphere, Ocean Dynamics, 58, 461–473, https://doi.org/10.1007/s10236-008-0148-3, 2008.

Le Provost, C. and Lyard, F.: Energetics of the M2 barotropic ocean tides: an estimate of bottom friction dissipation from a hydrodynamic model, Progress in Oceanography, 40, 37–52, https://doi.org/10.1016/S0079-6611(97)00022-0, 1997.

LeBlond, P. H.: Tides and their Interactions with Other Oceanographic Phenomena in Shallow Water (Review), in: Tidal hydrodynamics, edited by Parker, B. B., pp. 357–378, John Wiley & Sons, Inc., New York, USA, 1991.

Liu, J.: Open and traction boundary conditions for the incompressible Navier–Stokes equations, Journal of Computational Physics, 228, 7250 – 7267, https://doi.org/10.1016/j.jcp.2009.06.021, 2009.

Luettich, R. A. and Westerink, J. J.: Formulation and Numerical Implementation of the 2D/3D ADCIRC Finite Element Model Version 44.XX, Tech. rep., http://www.unc.edu/ims/adcirc/adcirc_theory_2004_12_08.pdf, 2004.

Luettich, R. A. and Westerink, J. J.: Continental Shelf Scale Convergence Studies with a Barotropic Tidal Model, chap. 16, pp. 349–371, American Geophysical Union (AGU), https://doi.org/10.1029/CE047p0349, 2013.

Lyard, F., Lefevre, F., Letellier, T., and Francis, O.: Modelling the global ocean tides: modern insights from FES2004, Ocean Dynamics, 56, 394–415, https://doi.org/10.1007/s10236-006-0086-x, 2006.

Massey, T. C.: Locally constrained nodal connectivity refinement procedures for unstructured triangular finite element meshes, Engineering with Computers, 31, 375–386, https://doi.org/10.1007/s00366-014-0357-y, 2015.

Mount, D. M. and Arya, S.: ANN: A Library for Approximate Nearest Neighbor Searching, version 1.1.1, available at http://www.cs.umd.edu/~mount/ANN/, 2006.

Nguyen, V.-T., Peraire, J., Khoo, B. C., and Persson, P.-O.: A discontinuous Galerkin front tracking method for two-phase flows with surface tension, Computers & Fluids, 39, 1 – 14, https://doi.org/10.1016/j.compfluid.2009.06.007, 2010.

O'Callaghan, J. F. and Mark, D. M.: The extraction of drainage networks from digital elevation data, Computer Vision, Graphics, and Image Processing, 28, 323 – 344, https://doi.org/10.1016/S0734-189X(84)80011-0, 1984.

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

Persson, P. O. and Strang, G.: A Simple Mesh Generator in MATLAB, SIAM Review, 46, 2004, https://doi.org/10.1137/S0036144503429121, 2004.

Piggott, M. D., Farrell, P. E., Wilson, C. R., Gorman, G. J., and Pain, C. C.: Anisotropic mesh adaptivity for multi-scale ocean modelling, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 367, 4591–4611, https://doi.org/10.1098/rsta.2009.0155, 2009.

Pringle, W. J., Yoneyama, N., and Mori, N.: Multiscale Coupled Three-dimensional Model Analysis of the Tsunami Flow Characteristics around the Kamaishi Bay Offshore Breakwater and Comparisons to a Shallow Water Model, Coastal Engineering Journal, https://doi.org/10.1080/21664250.2018.1484270, 2018.

Register, A. H.: A Guide to MATLAB Object-Orientated Programming, Chapman and Hall/CRC, 2017.

Roberts, K. J. and Pringle, W. J.: OceanMesh2D: User guide - Precise distance-based two-dimensional automated mesh generation toolbox intended for coastal ocean/shallow water, https://doi.org/10.13140/RG.2.2.21840.61446/2, 2018.

Shewchuk, J. R.: What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures, Tech. rep., In Proc. of the 11th International Meshing Roundtable, 2002.

Technology Riverside Inc. and AECOM: Mesh Development, Tidal Validation, and Hindcast Skill Asessment of an ADCIRC Model for the Hurricane Storm Surge Operational Forecast System on the US Gulf-Atlantic Coast, Tech. rep., National Oceanic and Atmospheric Administration/Nation Ocean Service, Coast Survey Development Laboratory, Office of Coast Survey, https://doi.org/10.7921/G0MC8X6V, 2015.

Thyng, K. M., Greene, C. A., Hetland, R. D., Zimmerle, H. M., and DiMarco, S. F.: True colors of oceanography: Guidelines for effective and accurate colormap selection, Oceanography, 29, 9–13, https://doi.org/10.5670/oceanog.2016.66, 2016.

Wang, Q., Danilov, S., Sidorenko, D., Timmermann, R., Wekerle, C., Wang, X., Jung, T., and Schröter, J.: The Finite Element Sea Ice-Ocean Model (FESOM) v.1.4: formulation of an ocean general circulation model, Geoscientific Model Development, 7, 663–693, https://doi.org/10.5194/gmd-7-663-2014, 2014.

Wessel, P. and Smith, W. H. F.: A global, self-consistent, hierarchical, high-resolution shoreline database, Journal of Geophysical Research: Solid Earth, 101, 8741–8743, https://doi.org/10.1029/96JB00104, 1996.

Westerink, J. J., Luettich, R. A., and Muccino, J.: Modelling tides in the western North Atlantic using unstructured graded grids, Tellus A, 46, 178–199, https://doi.org/10.1034/j.1600-0870.1994.00007.x, 1994.