ERDC/CHL CHETN-VI-47
March 2018

US Army Corps of Engineers®

Automatic Mesh Generation and Element Filtering: The AutoMesh Tool

by Drew A. Loney, Elissa M. Yeates, and Brent H. Hargis

PURPOSE: The purpose of this Coastal and Hydraulics Engineering Technical Note (CHETN) is to provide guidance for automated finite-element mesh generation. This is implemented in the Automatic Mesh Generation (AutoMesh) tool within the Rapid Operational Access and Maneuver Support (ROAMS) platform. The automatic meshing procedure was developed to analyze scatter set data and produce logical finite-element mesh node connectivity without user interaction. The resultant element structure can then be used to perform hydraulic analysis in Adaptive Hydraulics (AdH) or other finite-element tools. Automatically generating computational meshes reduces model creation effort, lessens required user skill level, and often quickens the process compared to manual mesh creation. The automatic mesh algorithm is a critical component within the ROAMS platform, which produces rapid solutions to coastal hydrodynamics and vessel routing questions with limited information and user input. AutoMesh enables military decision makers to quickly access relevant information during logistical and combat operations in water-adjacent environments within the ROAMS platform.

BACKGROUND: The ROAMS platform was originally developed at the U.S. Army Engineer Research and Development Center, Coastal and Hydraulics Laboratory, to provide predictive route and logistics modeling in remote access areas with scarce physical data. The platform supports force projection operations and logistics movements such as humanitarian assistance/disaster relief efforts. The ROAMS platform computes hydrodynamic conditions including water depth, wave height, and water velocity for a region to assist in determining optimum entry timing and predicting the accessibility of various vessels to the region (Farthing et al. 2014; Loney et al. [2017]). A major effort within ROAMS is developing process automation to minimize input time, effort, and expertise required by the user. The finite-element meshing algorithm underlying the AutoMesh tool was created to address the need for further automation within ROAMS.

The AutoMesh tool produces unstructured triangular finite-element meshes, which can be used as inputs for subsequent calculations. Meshes of this type are required to solve the physical shallow water, groundwater, and Navier-Stokes equations used for hydraulic and hydrologic analysis. Within ROAMS, AdH is a finite-element engine developed to solve the physical equations to perform water depth and velocity calculations. The ROAMS package also implements a routing toolbox, which uses the nodal connectivity of a finite-element mesh to determine an initial vessel route as part of an A* search between user-specified locations. AutoMesh was designed to produce depth-node meshes compatible with AdH and with the routing toolbox to enhance the usability of the ROAMS platform. However, the methods described within this CHETN are generalizable to mesh generation for other types of scatter set data and have many applications outside of predictive routing.

METHODS: Automesh produces a finite-element mesh from a node scatter set using three core processes: detecting the scatter set boundary, generating the mesh elements, and filtering the mesh to eliminate elements that erroneously cross the boundary edge.

Each step in the AutoMesh generation process is presented in detail in the Methods subsections. The detailed steps are as follows:

1. Read in a depth node scatter set for a port access area.
2. Perform initial boundary edge detection to product a rough boundary.
3. Adjust the boundary segments to *tighten* the boundary around the set.
4. Triangulate between the depth nodes to create mesh elements.
5. Filter out mesh elements crossing the identified boundary.
6. Format the resultant list of elements for compatibility with ROAMS.

1. Read in a Depth Node Scatter Set

The AutoMesh process requires an input scatter set with *x*-, *y*-, and *z*-coordinates. The AutoMesh tool uses an object-oriented structure implemented in Python. Scatter set information is read into an empty finite-element mesh object as the mesh nodes, along with information about the number of nodes in the set and the units and geographic zone associated with the set. The mesh nodes are stored as a numpy matrix within the finite-element mesh object to facilitate later computations.

2. Detect Initial Boundary Nodes to Produce a Rough Boundary

Drawing an accurate exterior boundary around a scatter set facilitates mesh generation. A defined exterior boundary allows the removal of mesh elements that inappropriately generate outside of the boundary and that have no physical significance. Identifying the nodes that are on the edge of the set and then drawing boundary edges between selected nodes produces the scatter set boundary. This process is done successively in clockwise fashion around the scatter set. In future iterations, this process may be parallelized for more efficient computing if conflicts between polygon segments can be effectively handled. Boundary identification is currently the most computationally intensive part of the automatic mesh generation process (see Timing Benchmarks in the Results section for detail). The algorithm enacted in the boundary node detection process is described and illustrated in detail below.

An empty set of boundary nodes is created for the scatter set object. The minimum and maximum *x*- and *y*-location values are determined from all of the nodes in the scatter set, identifying those nodes located on minimum and maximum values (Figure 1).

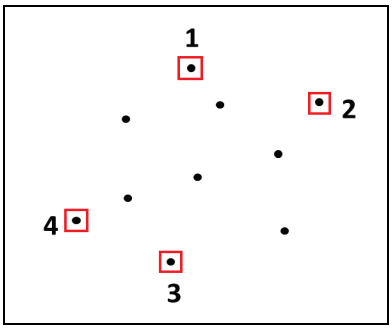

Figure 1. Detection of minimum and maximum node locations.

These identified nodes are entered into the boundary set and used to form a bounding rectangle around the scatter set. The rectangle is formed by extending lines with a slope of zero from the nodes with the minimum and maximum *y*-locations and lines with a slope of one from the nodes with the minimum and maximum *x*-locations (Figure 2). The four vertices formed by this bounding rectangle are entered into the boundary set as synthetic corner nodes (nodes a, b, c, and d in Figure 2). The nodes in the boundary set are ordered clockwise in the array to form a boundary polygon. Duplicate synthetic corner nodes, if they exist, are removed. This could occur if one node has both a minimum or maximum x and y value. That node would exist on the corner, and the corner node generated by the line intersections would duplicate that node.

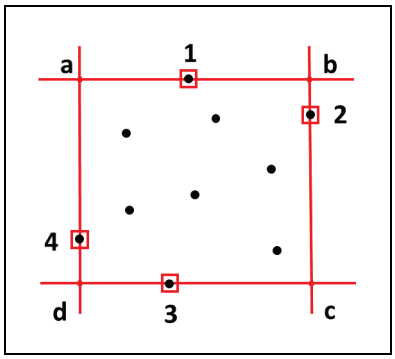

Figure 2. Drawing a bounding rectangle around the set.

If no nodes in the scatter set are on a corner of the bounding rectangle, those corners are then cut (Figure 3). Corners are cut by connecting the nodes that lie on the horizontal and vertical boundary lines closest to the corner, beginning with the upper-left corner node and proceeding clockwise.

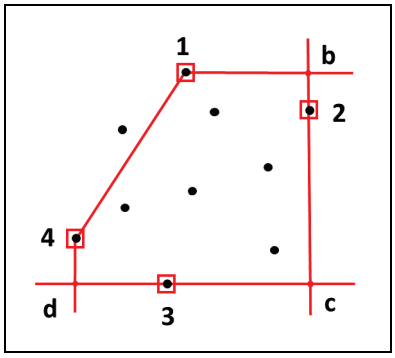

Figure 3. Cutting the boundary corners.

Cutting the corners might move the boundary in a way that excludes nodes in the scatter set, making those nodes exterior to the polygon. In Figure 4, Node 5 is found to be exterior to the bounding polygon after the first corner node is cut.

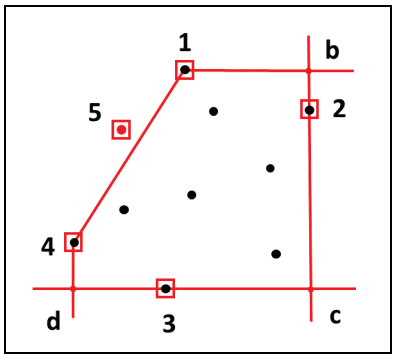

Figure 4. Node 5 is external to the boundary.

This external node is added to the boundary node set (Figure 5). Every node that is added to the boundary node set is removed from the scatter set array to eliminate duplication within the algorithm. The set is checked to ensure that no nodes lie exterior to those that have been identified as boundary nodes.

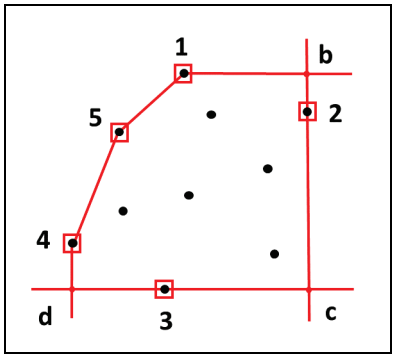

Figure 5. Adding Node 5 to the boundary set.

If multiple nodes are found to be exterior to a cut corner, the node with the greatest perpendicular distance from the corner is identified as the node to be included in the boundary node set (For example, Node 6 in the panels of Figure 6 is included to the boundary). This process is repeated until no external nodes remain.

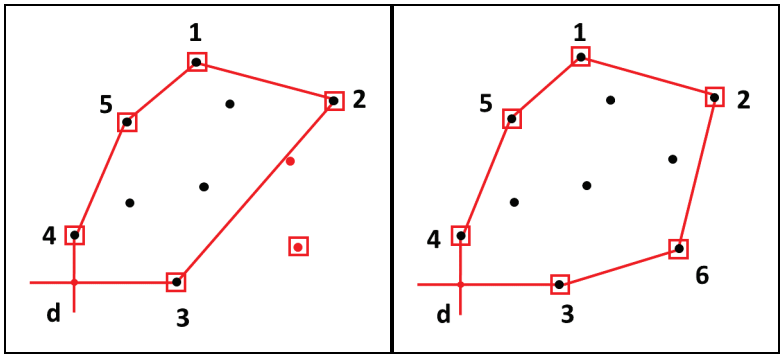

Figure 6. Adding Node 6 to the boundary set.

Once no nodes are found to be exterior to a corner, the algorithm proceeds to the next corner (Figure 7). Once all corners have been cut, the boundary node detection algorithm is finished, and AutoMesh proceeds to the adjustment of each individual boundary segment.

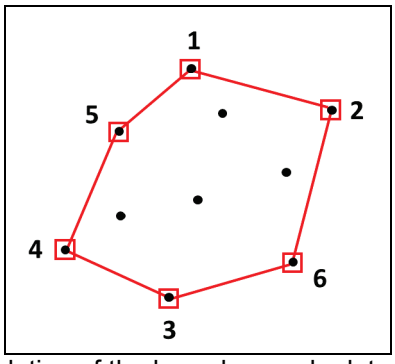

Figure 7. Completion of the boundary node detection algorithm.

3. Perform Boundary Segment Adjustment to Tighten the Boundary

The initial boundary-drawing algorithm produces a rough boundary, but for the mesh to be correctly defined, the boundary must be tightened around the scatter set. This is done via the boundary segment adjustment process clockwise from the upper left around the bounding polygon. Each segment is iteratively analyzed to determine if an improved fit, one that includes a greater number of nodes and conforms more closely to the scatter set shape, can be found. The paneled illustrations in this section show this process occurring over a detailed portion of a depth node scatter set.

To limit the boundary segment adjustment algorithm, several steps are taken to cut the number of nodes that are considered for inclusion to the boundary set. This reduces computation time since the boundary segment adjustment process can occur thousands of times on the segments of a typical scatter set. In the analysis of a single segment, the node starting the segment in the clockwise direction is designated the *backward node*, and the node ending the segment is designated the *forward node* (Figure 8).

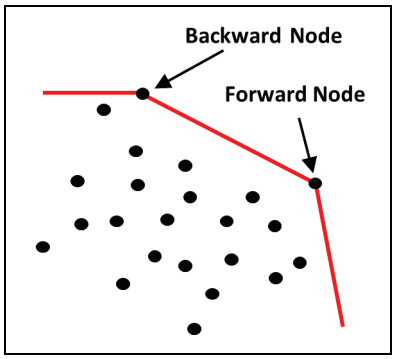

Figure 8. Backward and forward nodes of a segment under analysis.

The first step in limiting potential boundary nodes is to exclude nodes that are farther than one segment length away from the segment under adjustment (Figure 9). Segment length ($\mathcal{L}$) is defined as the Euclidean distance between the forward and backward nodes of the segment. This

step treats the segment as an extended line; all nodes within a segment length ($\mathcal{L}$) distance perpendicular to that line are included in the set of nodes to be considered. In the example shown in Figure 9, the lighter-colored nodes have been excluded from analysis by this step because they exceed length $\mathcal{L}$ in perpendicular distance from the line that includes the segment. In most cases, this step will exclude a large percentage of all nodes in the scatter set from consideration for the boundary set.

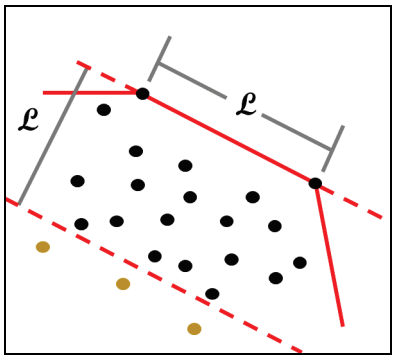

Figure 9. Limiting the depth of the node field to be analyzed.

In the next step the set of potential boundary nodes is limited further. Lines perpendicular to the segment are extended from the backward and forward nodes (Figure 10). Nodes that lie outside of the area bounded between these lines are removed from consideration for the boundary. Figure 10 shows four nodes excluded by this step. In most cases, this step will appreciably reduce the nodes eligible for inclusion to the boundary.

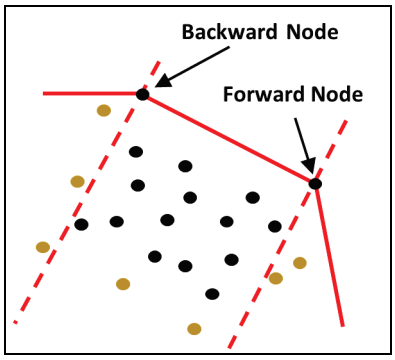

Figure 10. Limiting the width of the node field to be analyzed.

Each remaining potential boundary node is then evaluated to determine its perpendicular distance from the segment (d_p) and the location of that intersection with the segment (Figure 11, left panel). The distances from the point of intersection to the backward node (d_b) and to the forward node (d_f) are calculated. The node under analysis is excluded from the set of potential boundary nodes if d_p is greater in magnitude than either d_b or d_f. This condition excludes nodes outside of the resulting 45-degree triangular area from the potential boundary set (Figure 11, right panel). This step is taken after the previous two set-reducing conditions to limit computational intensity in the distance calculation.

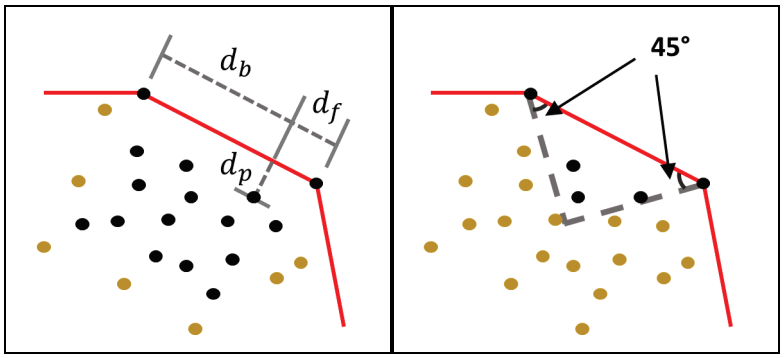

Figure 11. Reducing the potential boundary node set to a triangular area.

Once the set of possible nodes is reduced by the previous conditions, the remaining nodes in the set are examined for inclusion in the boundary. The node that has the smallest perpendicular distance to the segment is considered first (these perpendicular distances were calculated and temporarily stored during the step shown in Figure 11). The algorithm then proceeds to the node with the next smallest distance from the segment, until either a node is found to be an acceptable boundary node or until all potential boundary nodes have been examined. A node is acceptable for the boundary if including it would not render other nodes external to the boundary. In Figure 12, new nodes are added to the boundary. The added node becomes the new forward node, and the boundary segment adjustment process begins again with the same backward node and this new forward node. The algorithm proceeds clockwise until no segments require adjustment.

If no nodes are within the triangular area shown in Figure 11, then the boundary segment under analysis may not require adjustment, and the algorithm proceeds into failsafe mode. Figure 13 shows a segment that would be analyzed with the failsafe routine, given the lack of nodes that could be added to the boundary in the area of analysis.

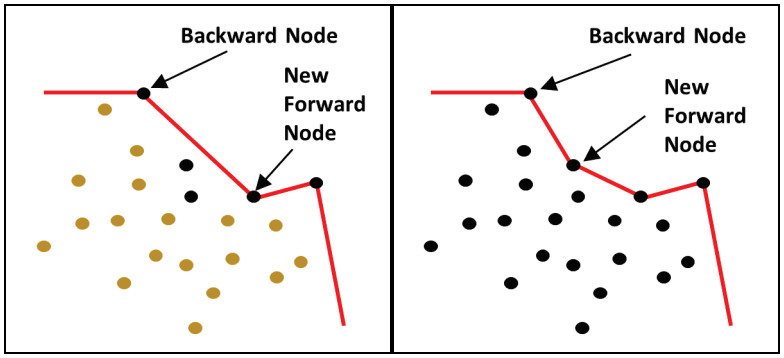

Figure 12. Adding nodes to the boundary.

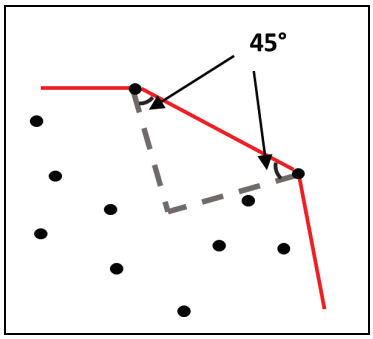

Figure 13. Condition to enter the failsafe mode.

The failsafe mode ensures that the boundary segments are an appropriate length given the scatter set characteristics and employs a different segment adjustment algorithm if necessary. A boundary segment is an acceptable length if it is less than or equal to twice the average length of all the nearby node connections. To determine this, the forward node, backward node, and all nodes within one segment length distance from the segment under adjustment are analyzed. For each of these nodes, the distance to each of its ten nearest neighbors is calculated, and the average of this internodal spacing is found (Figure 14). This is compared to the length of the boundary segment. If the length of the segment is less than twice the average nearby internodal length, then the segment adjustment process is considered finished for that segment. The forward node for the segment becomes the backward node for the next segment in the clockwise direction.

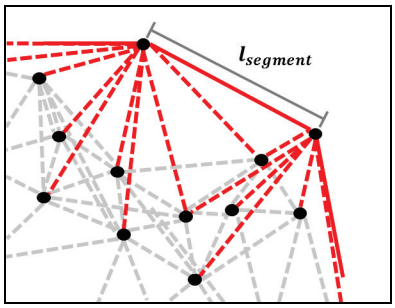

Figure 14. Calculating segment length and average internodal spacing in failsafe mode.

If the segment is longer than twice the average internodal spacing, the failsafe routine considers the segment too long to be a boundary segment, and it must find another nearby node to add to the boundary. If this is the case, the next step is to generate a node on the midpoint of the segment, between the forward and backward nodes (Figure 15). The distances from this new midpoint node and each of the nodes within one segment length from the segment are calculated.

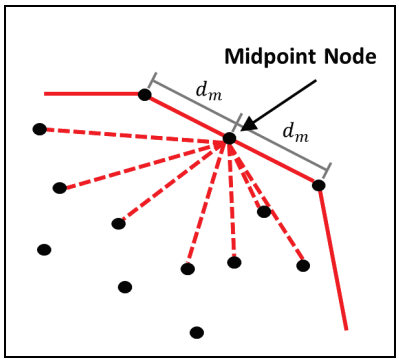

Figure 15. Generating the midpoint node and calculating distances in failsafe mode.

The node with the shortest distance to this newly generated midpoint node is then accepted into the boundary (Figure 16). The midpoint node is deleted. The algorithm checks to ensure that adding this new node into the boundary does not result in any other nodes being made external to the boundary. If not, then the newly added node is designated as the new forward node for the segment, and this newly adjusted segment goes back into the segment adjustment process routine until it passes the appropriate segment length condition.

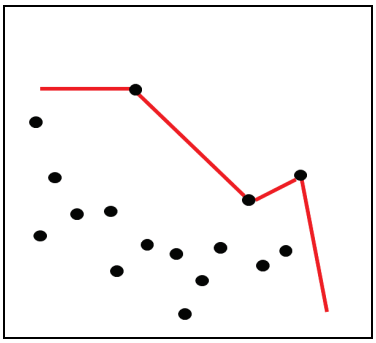

Figure 16. Adding the new node to the boundary in failsafe mode.

In some rare cases with complex inlet geometries, adding the new node to the boundary can result in making another node or nodes external to the boundary. In this event, the node that is excluded from the boundary polygon is accepted as the new boundary node. If more than one node is made external to the boundary, those nodes are associated with the left or right side of the new boundary segment. The node on the left side and the node on the right side that are closest to the midpoint node are accepted into the boundary.

Once the set of boundary nodes for the scatter set is determined, the finite-element mesh boundary edges are calculated. The boundary edges connect the boundary nodes into a polygon. The edge connection indices are paired with the boundary node indices and reordered to match the clockwise order of the boundary nodes.

4. Triangulate between the Nodes to Produce Mesh Elements

After the boundary edge is finalized, a triangulation function is called to produce triangular mesh elements connecting all nodes in the scatter set. The AutoMesh algorithm uses the Delaunay triangulation method (described in Lee [1980]). Delaunay triangulation is implemented from the Python open source library of scientific tools, SciPy, using the spatial code library (SciPy

v0.14.0 Reference Guide [2009]). The Delaunay triangulation process works to maximize the minimum angles in the triangular elements, reducing the occurrence of long, thin triangles. This preserves more physical accuracy within the triangulation.

The mesh generation through triangulation routine typically takes much less time to perform than the boundary edge detection routine. Processing time is on the order of hundredths of a second to one or two seconds, depending on the size and density of the node set.

5. Filter and Delete Elements That Exist outside of the Mesh Boundary

Creating node connectivity within a scatter set via Delaunay triangulation results in some mesh elements being generated outside of the scatter set boundary. For application within ROAMS and other programs, it is important that no elements of the mesh exist outside of this boundary as exterior elements artificially expand the domain. These elements could also result in incorrect interpolation values if barycentric interpolation is utilized. It is therefore necessary to remove the external mesh elements produced by the triangulation. Every node in each element in the mesh is examined to determine if it is included in the set of boundary nodes. If an element contains three boundary nodes, it is likely outside of the mesh boundary, and that element is deleted from the mesh.

Figure 17 shows the result of this process for a bathymetry scatter set of the New River Inlet region. The left panel shows the New River Inlet AutoMesh result before the filtering routine described in this section. The Delaunay process constructs a triangulation, which results in many elements that connect across different inlets of the domain. These elements contain three nodes on the boundary and are outside of the natural boundary of the inlet. The right panel shows the mesh after those elements are deleted from the mesh. The routine succeeds in eliminating the majority of these improper mesh elements.

The bottom-right feature of the port mesh highlighted in the right panel of Figure 17 shows an area where the mesh cleaning routine did not perform well. The boundary node detection process does not completely define shallow pinched inlets such as this one, so the mesh filtration routine does not identify the elements within this inlet as containing boundary nodes.

The mesh cleaning routine results in some nodes being disconnected from the mesh (for example, the top-left feature of the port mesh highlighted in the right panel of Figure 17). This occurs when a node at the edge of the mesh does not connect to any internal nodes, only to other nodes on the boundary. In this case, when the mesh element cleaning routine filters out those elements, the nodes are left unconnected to any other node. This is an area of future development of the AutoMesh tool. Adapting the filtering routine to consider the location of the centroid of a triangular mesh element can fix this problem. Triangles with centroids internal to the boundary will be maintained, even if all nodes are boundary nodes.

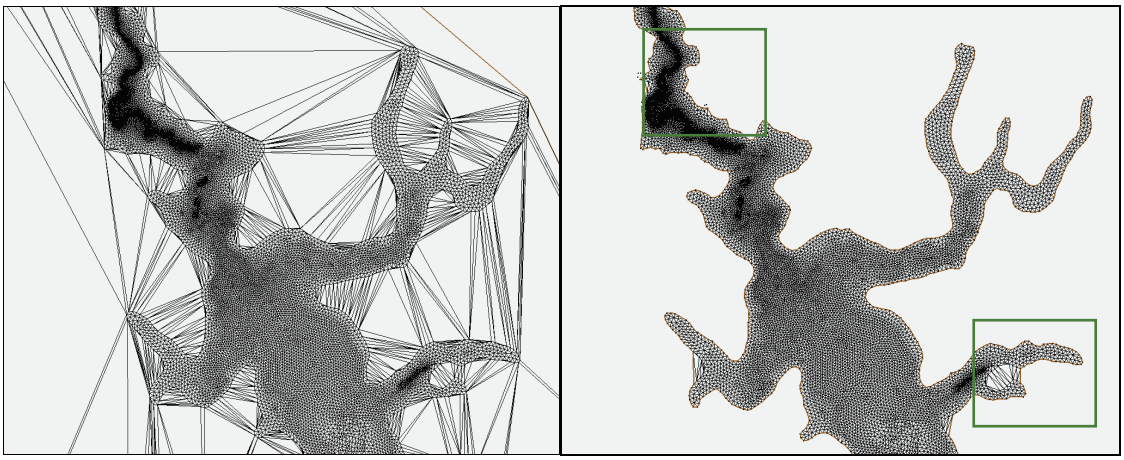

Figure 17. New River Inlet detail, before (left) and after (right) mesh filtration. Boxed areas show examples of mesh filtration performance problems: disconnected nodes and shallow inlet webbing.

6. Format the Resultant Mesh for Compatibility with Adaptive Hydraulics (AdH)

The final step is to format the connectivity to ensure functionality as input files to AdH or other hydrodynamic analysis applications. AutoMesh adds sequential element numbers to every element in the mesh and adds a placeholder value to function as the material number at the end of the row for each element in the array. This can be altered if a user needs varied material numbers, but that is not necessary for its present use within ROAMS.

RESULTS: The AutoMesh generation process was executed for the North Carolina New River Inlet, which includes the port area at the U.S. Marine Corps Base Camp Lejeune. Automesh was also executed on a bathymetry set from the General Bathymetric Chart of the Oceans database (GEBCO 2017) and on randomized scatter sets of varying densities. Results from these processes are presented in this section. Automesh was benchmarked on a desktop computer with a quad-core four GHz Intel Core i7 processor and 32 gigabytes of DDR3 1,867 MHz RAM.

New River Inlet. The New River Inlet was chosen as a test case for the AutoMesh tool because the riverine and coastal geographies are complex, posing a challenge to the process, and because a human-generated mesh existed for this location for comparison. The AutoMesh tool required 9.6 minutes for boundary node detection with an additional 15 seconds for the subsequent mesh generation. The human-generated mesh was created in the Surface-water Modeling System (SMS) and took approximately 8 hours to create to the shown level of detail. The SMS is a commercial model support platform created by Aquaveo that is commonly utilized to facilitate the creation of AdH models (Aqueveo 2016). The additional time was necessary to review the output of the SMS meshing utilities, remove the excess triangles at the boundary, and manually refine the mesh to improve the mesh quality. Note that AutoMesh does not currently refine the mesh to improve its quality. Figure 18 shows the human-generated mesh using SMS. Figure 19 shows the AutoMesh-generated mesh.

The majority of the elements in both meshes were triangulated within reasonable proximity to one another. The AutoMesh-generated boundary is loosely defined around some areas of the mesh, resulting in some additional external webbing around shallow inlets, as discussed in Step 5 in the Methods section.

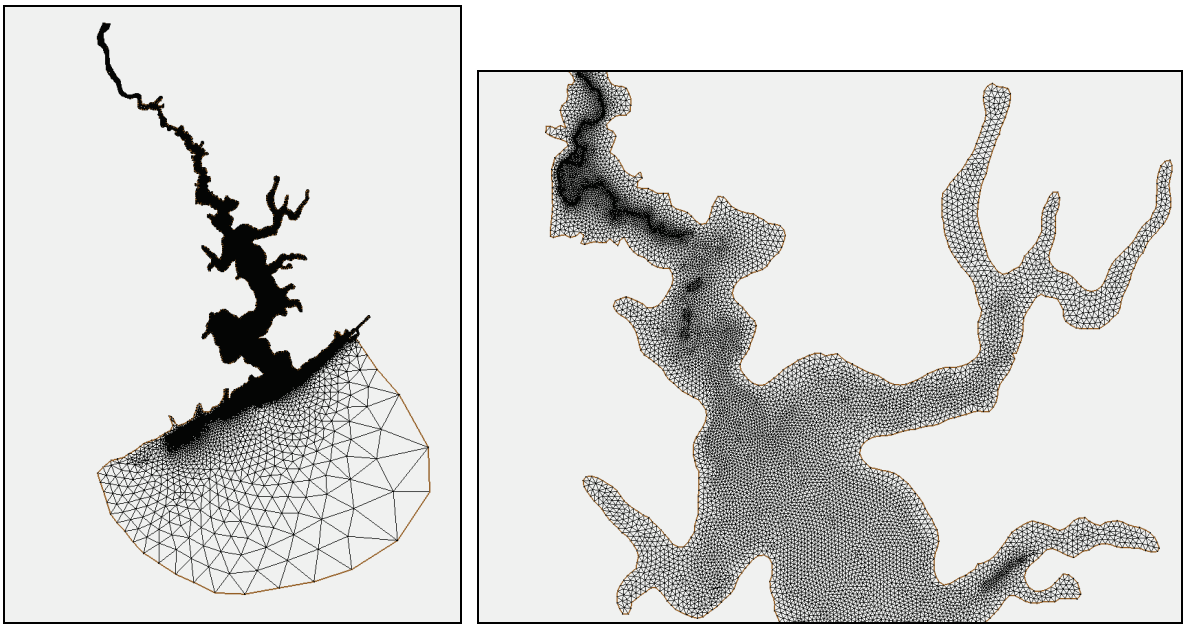

Figure 18. Human-generated New River Inlet mesh, entire (left) and detail (right).

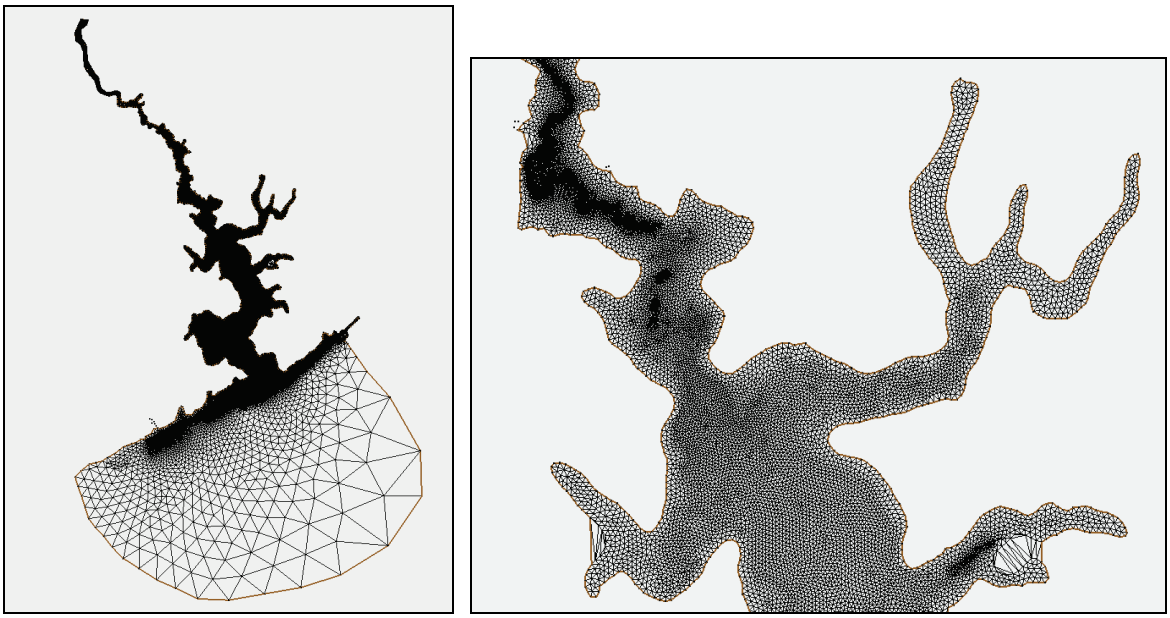

Figure 19. AutoMesh New River Inlet mesh, entire and detail.

GEBCO Dataset. A gridded dataset from GEBCO (2017) was used to evaluate the efficacy of AutoMesh on gridded data. The dataset appears slightly skewed due to the projection of the area shown (Figure 20, top panel). Due to this skew, the boundary tightening process and the Delaunay triangulation resulted in a layer of long, thin triangles along the top of the grid (Figure 20, bottom-left panel). Deleting the elements that consist of three boundary nodes eliminated this problem but resulted in the top-left node being dropped from the mesh (Figure 20, bottom-right panel). This dropping of some boundary nodes in the element filtration process was discussed in Step 5 of the Methods section.

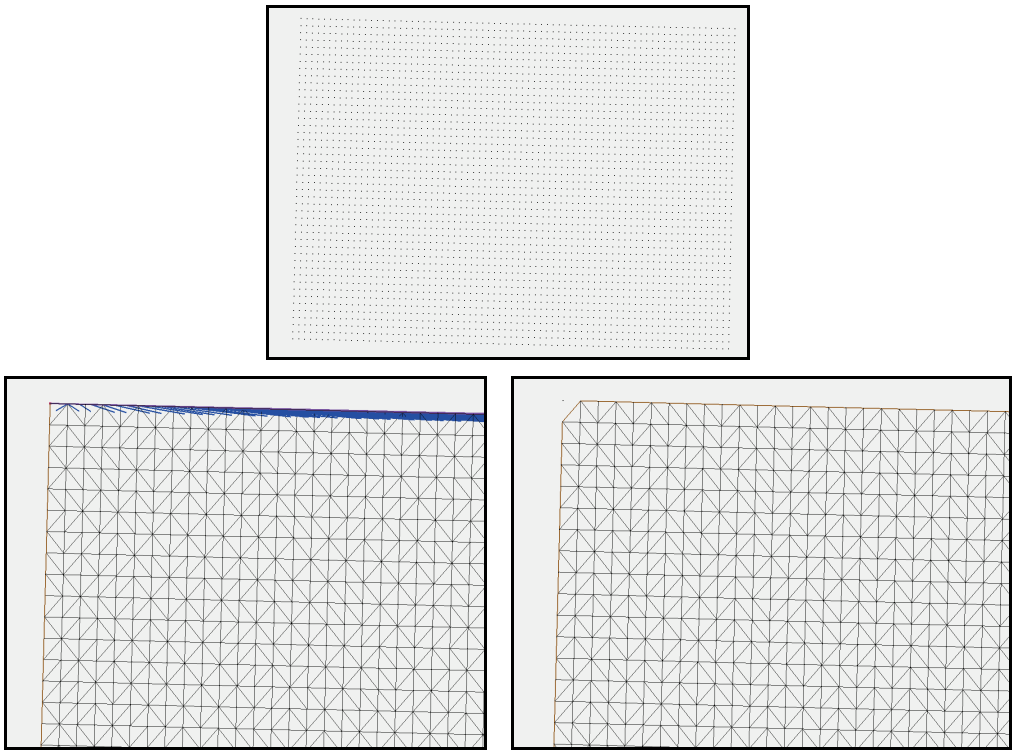

Figure 20. GEBCO dataset nodes (top); GEBCO mesh detail after Delaunay triangulation (bottom left); GEBCO mesh detail after Delaunay triangulation and deletion of boundary-crossing elements (bottom right).

Randomly Generated Scatter Sets. Eight randomly generated scatter sets with node densities increasing by a factor of two were used as AutoMesh inputs. The meshes were run on the same computer to compare computation times. The first four node densities, 64, 128, 256, and 512, are shown in Figure 21. Lower-density scatter sets displayed a tendency to drop a small number of nodes when filtering out external elements. These nodes only connected to other boundary nodes, rather than connecting to internal nodes, and were dropped from the mesh by the cleanup routine outlined in Methods: Step 5. This issue becomes less prominent as node density increased. Since the method may be used in data-sparse areas, addressing this node-dropping problem by incorporating the centroid method discussed in Methods: Step 5 should be an area of future development.

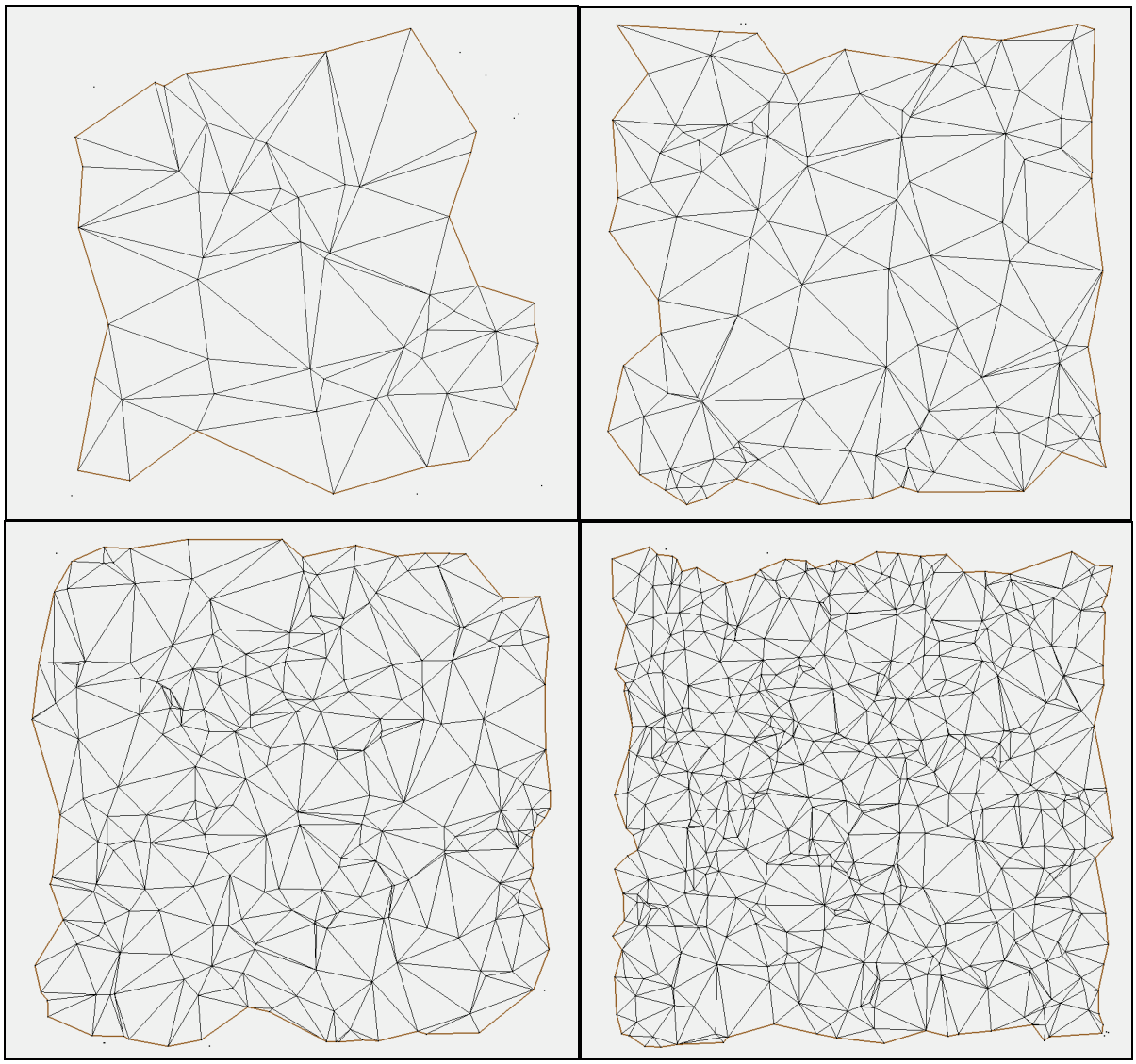

Figure 21. AutoMesh meshes generated from random scatter set. Top left: node density 64. Top right: node density 128. Bottom left: node density 256. Bottom right: node density 512.

For datasets tested, the computation time for the AutoMesh mesh generation process correlated linearly with the size of the dataset. This indicates that AutoMesh is efficiently capturing the change in boundary effects for denser datasets without spending proportionally more computation time handling the growth in dataset size. The filtering mechanisms are efficiently handling the growing number of internal nodes with increasing node density.

For all datasets analyzed, the boundary node identification process took much more time than the mesh generation and filtration process. Figure 22 shows boundary detection and mesh generation computation times as a function of node density size. Note that mesh element generation is graphed along the secondary right vertical axis and is several orders of magnitude smaller than boundary detection time. The most complex meshes took under an hour to generate with AutoMesh.

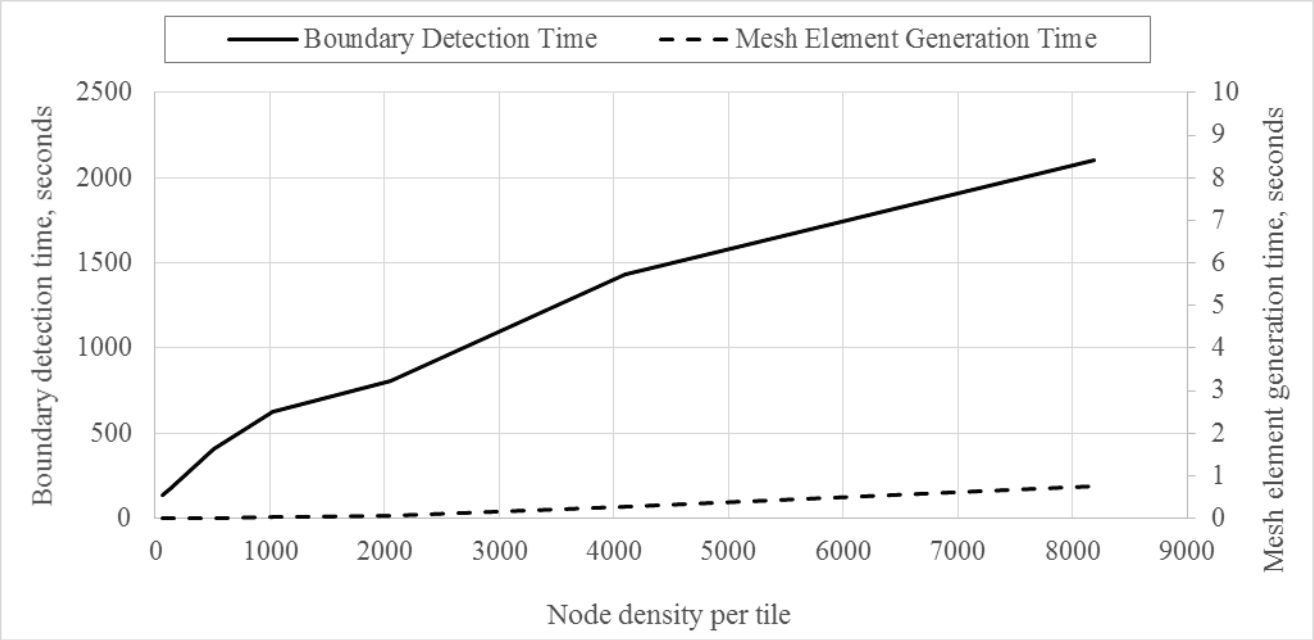

Figure 22. AutoMesh computation time versus node density.

CONCLUSION: In this CHETN, guidance was provided to automate finite-element mesh generation. This was demonstrated through the functionality of the AutoMesh tool within the roams platform. The methods section described each step in the mesh generation process. Automesh was performed on the New River Inlet scatter set of interest, three-structured GEBCO bathymetry scatter sets, and eight randomly generated scatter sets. The mesh generated for the New River Inlet closely matched the human-generated mesh that was previously in use. The results section presented these examples as well as timing benchmarks and areas for further development of the AutoMesh tool. The resultant meshes can be used to perform hydraulic analyses, which are used in ROAMS to produce vessel-routing solutions during logistical and combat operations. Meshes produced with the AutoMesh tool are also generalizable to many other applications.

POINT OF CONTACT: The point of contact for technical inquiries for this CHETN is Drew Allen Loney, PhD. Contact via email at *Drew.A.Loney@erdc.dren.mil* or by phone at 601-634-3490. This CHETN should be cited as follows:

> Loney, Drew A., Elissa M. Yeates, and Brent H. Hargis. 2018. *Automatic Mesh Generation and Filtering: The AutoMesh Tool*. ERDC/CHL CHETN-VI-47. Vicksburg, MS: U.S. Army Engineer Research and Development Center. *http://dx.doi.org/10.21079/11681/26530*

REFERENCES

Aquaveo. 2016. SMS Introduction | Aquaveo.com. *http://www.aquaveo.com/software/sms-surface-water-modeling-system-introduction*.

Farthing, M. W., K. D. Winters, A. Ahmadia, T. J. Hesser, S. E. Howington, B. D. Johnson, J. N. Tate, and C. E. Kees. 2014. “Rapid Prototyping of Hydrologic Model Interfaces with IPython.” In *American Geophysical Union Fall Meeting Abstracts*. Vol. H44D-08. San Francisco, California.

GEBCO: General Bathymetric Chart of the Oceans. 2017. *The GEBCO_2014 Grid*, version 20141103. http://www.gebco.net/data_and_products/gridded_bathymetry_data/version_20141103/.

Lee, Der-Tsai, and Bruce J. Schachter. 1980. "Two algorithms for constructing a Delaunay triangulation." *International Journal of Computer & Information Sciences* 9(3): 219–242.

Loney, Drew, Kimberly Pevey, Jennifer McAlpin, Benjamin Nelsen, and Brent Hargis. 2017. *Rapid Operational Access and Maneuver Support (ROAMS) Platform for Improved Military Logistics Lines of Communication and Operational Vessel Routing*. ERDC/CHL-CHETN-IX-45. Vicksburg, MS: U.S. Army Engineer Research and Development Center. http://dx.doi.org/10.21079/11681/22642.

SciPy v0.14.0 Reference Guide – Spatial algorithms and data structures (scipy.spatial). 2009. http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.Delaunay.html.