PPUTILS

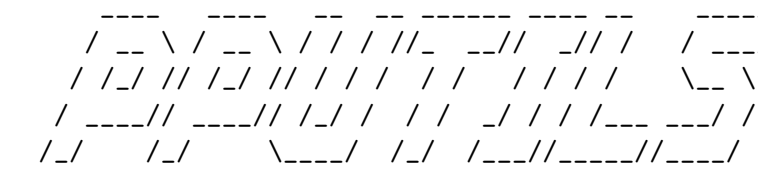
A PRACTICAL TOOLKIT FOR TERRAIN,
FREE SURFACE FLOW AND WAVE MODELING

BY

PAT PRODANOVIC, PH.D., P.ENG.
London, Ontario, Canada

web: https://github.com/pprodano/pputils
email: pprodano@gmail.com

APRIL 7, 2017



Welcome to Pat Prodanovic's utilities for terrain,
free surface flow and wave modeling!

Revision history

Date	Version	Comment
April 7, 2015	PPUTILS v1.07	Initial release of the manual.

Contents

List of Tables

List of Figures

Preface

The idea behind open source software fascinated me ever since my first introduction to the subject in the fall of 2002. Over the years I have become a heavy user of open source software. I use it at home and at work. As I have benefited significantly from open source software over the years, I thought it important to offer some of my work to the open source community. PPUTILS is my such contribution.

The PPUTILS project was initially conceived after I completed a number of key tools that I used in completing different river and coastal engineering modeling projects. PPUTILS includes a toolbox of simple scripts that ultimately allows one to complete entire numerical modeling projects from start to finish by using only free and open source software.

I have developed most of the tools within PPUTILS during my spare time. Prior to having PPUTILS, I had to rely on many proprietary software packages to complete tasks associated with my own engineering projects. But I always wanted to avoid using proprietary tools. I searched the web trying to find open source alternatives to assist me in numerical modeling tasks, but I couldn't find much. So, I decided to begin developing simple scripts, each that would solve a particular task. Eventually, with many such simple scripts in hand, I found that I had a collection of tools that allowed me to complete entire engineering projects from start to finish. This indicated to me that PPUTILS have reached a status of maturity, and could be shared with others. I decided to package the project on Github, and offer it to rest of the open source community.

Please note that scripts in PPUTILS will always be free (as in freedom), but will also be free of warranty. Should you find errors or bugs with any one of the tools within PPUTILS, I strongly encourage you to report them back to me.

The assumptions I have made while working on the PPUTILS project reflect my understanding of the subject matter, which I recognize are biased given my professional training, work experience, and personal preferences. In writing this manual it is my intention to share with the open source community the knowledge I have acquired on the subject of digital surface modeling, geometric mesh generation, and output visualization. I only hope others will find what I have produced useful.

I hope you enjoy using PPUTILS on your projects!

Pat Prodanovic
London, Ontario
April 2017

1 Introduction to PPUTILS

PPUTILS consist of a number of scripts that assist the user in completing environmental modeling projects using nothing but free and open source tools. Tasks that PPUTILS excels at are: development of digital surface models through creation of Triangular Irregular Networks (TINs), development of quality meshes and model grids for use in 2D and 3D numerical flow and wave modeling, and pre- and post-processing of model output for use in open source Geographic Information Systems (GIS) and other model viewers. The PPUTILS project is designed to be used with the open source TELEMAC modeling system. TELEMAC is a 2D/3D finite element environmental modeling system used in the study river and coastal hydraulics, wave generation and transformation, sediment transport, water quality, and other studies. A number of scripts in PPUTILS also deal with the pre- and post-processing for the SWAN numerical model. SWAN is an open source numerical model used for propagation and generation of waves in coastal areas.

Since all scripts in PPUTILS are in open source, modification of its source code (with simple format conversions) allow it to be extended to any other modeling system that requires use and processing of numerical model meshes and grids.

1.1 Governing principles

When the PPUTILS project was first conceived, the goal was for all code to:

1. Be entirely in open source,
2. Work on all common platforms,
3. Have minimal dependencies and easy installation,
4. Be computationally efficient, and
5. Be executed using standard command line.

Many tools used in the field of environmental flow and wave modeling intentionally do not adhere to the above principles. Sometime the tools are proprietary, and the developer expects to be paid for writing them. In other cases, a tool may be free and its code is entirely available, but its license is not open source. Triangle mesh generator (Shewchuk, 1996) is an example of a piece of code that is in the public domain (and is available to use and modify), but is not in open source according to any one open source licensing models available.

The user of environmental flow and wave modeling should posses a set of tools that allows completion of entire modeling projects using nothing but open source tools. This has, and always will be, the main focus of the PPUTILS project.

PPUTILS will always adhere to the above principles. By doing so, the author believes the PPUTILS project will ensure its long term survival.

1.2 Pre-requisite knowledge

In order to reach the full benefit of the tools in the PPUTILS project, the user should be familiar with the following:

1.2.1 Terminal use

As all of the tools part of PPUTILS are executed from the terminal (or command prompt), a basic familiarity of its usage is necessity. For example, the user should be familiar with the terminal, and know how to launch it from the particular operating system. The user should also know some terminal basics (i.e., how to change directories, get a listing of files, and how to execute simple commands).

1.2.2 Geographic Information Systems

Basic knowledge of GIS is also required for maximum benefit of the PPUTILS tools. Presently, a number of open source GIS packages are available, including QGIS (2017), GRASS (2017), SAGA (Conrad et al., 2015), GDAL (2017), gvSIG (2015), and others. Some familiarity of GIS is necessary in the construction of terrain, wave and flow meshes and grids using PPUTILS. The user should know, for example, how to construct a polyline and polygon objects within a GIS environment that have vertices spaced a certain distance. Different GIS packages allow the user to do this quite routinely. Oftentimes, it is as simple as selecting a polyline, specifying vertex spacing in the correct dialog box, and getting a new polyline with the desired spacing set. The user should also be familiar with basics of vector and raster data, and should know how to create, edit and modify such files in a GIS environment.

Note that the PPUTILS manual is not a tutorial for any one GIS platform or any of its numerous packages or plugins. There are many excellent tutorials that allow new users to rapidly learn GIS basics. The basic requirement for the PPUTILS user is to know how to import, create and export vector based data from a GIS environment. The PPUTILS project also creates raster files for use in GIS, which is especially useful for preparation of report style figures of modeling outputs.

1.2.3 Digital terrain modeling

In order to appreciate the terrain modeling set of tools within PPUTILS, the user has to have some basic knowledge of creating and manipulating digital surfaces. For example, the user should know what contours are, Digital Elevation Models (DEMs), Triangular Irregular Networks (TINs), and how each is created and modified. These tasks can all be done within a GIS environment. Traditionally, such tasks were always done in Computer Aided Design (CAD) environments.

A pre-requisite of any environmental flow or wave modeling is the development of a numerical mesh or grid that is developed to resolve the physics in question. After the mesh is developed, its nodes have to be assigned with proper elevation

values, mimicking real world topography and/or bathymetry. In PPUTILS, digital surfaces created by means of TINs are used to assign elevation values to the numerical meshes or grids, extract cross sections, compute volumes, etc. The TIN is therefore used as a basic digital surface in PPUTILS.

1.2.4 Pre- and post-processing for numerical modeling

The PPUTILS project offers a great many tools for pre-processing of numerical modeling inputs and outputs. For example, PPUTILS allows a user to create meshes and grids for flow and wave modeling application using the TELEMAC and SWAN models.

There are two types of mesh generation used in PPUTILS. One deals with generating triangular meshes for use in digital surfaces, while the other deals with quality meshes developed for use in numerical flow and wave modeling. PPUTILS does not include new or original mesh generation code. Rather, the PPUTILS project uses mesh generation code written by world's experts in the field of computational geometry. The meshing program Triangle (Shewchuk, 1996) is used for the development on TINs, while the program GMSH (Geuzaine and Remacle, 2009) is used exclusively for development of quality triangular meshes for use in numerical flow and wave modeling. For the development of regular grids, PPUTILS reads TIN models and creates regular grids (with grid size specified as input) using Python's Numpy, Matplotlib and Scipy dependencies.

After the simulations are complete, a number of the scripts allow the user to post-process the modeling output (extract data, plot graphs, or otherwise visualize the output using various open source tools). In many cases PPUTILS will create data to be read by CAD, GIS, and other numerical model viewer programs. In this sense, the PPUTILS tools are simple format converters that provide its users with freedom and flexibility in using their numerical model outputs however they wish. For some users it may be sufficient to obtain a screen grab of the output, while for others simulation output may need to be imported into a GIS environment for further processing and presentation of report style figures.

1.3 Scope of this manual

Prior to getting started with the individual tools, it is important to answer the question of what is, and what is not, included in this manual. Details are provided below.

1.3.1 What is not included in this manual

First, this manual does not present to its reader the fundamentals of GIS, digital surface modeling, river hydraulics, coastal engineering, or numerical modeling in general. Fundamentals in these areas are best obtained through university courses, and/or specialized training. The user should have some familiarity on

these subjects. Second, this manual refers to, and relies on, a number of different GUI software for visualization, but does not explicitly provide tutorials for any one of these. It will be up to the user to experiment and learn these applications. Third, this manual does not guide the PPUTILS user with the setup of simulation steering files (or definition of boundary or initial conditions) in numerical modeling applications. It will be up to the user to figure this out on their own.

In becoming proficient with numerical modeling the user needs to become familiar with the intricacies and nuances of the particular model. The best way to achieve this familiarity is to practice, starting with the most of simple models (for which analytical solution exists), and gradually increasing to more and more complex models. One of the main difficulties in applying flow and wave numerical models is associated with development of meshes and grids that property resolve the physics of the problem under study. Online help forums are littered with users complaining about models crashing under routine simulations. In many such instances the causes of the crash are improperly resolved numerical mesh or grid.

1.3.2 What is included in this manual

The scope of this manual is on i) preparation of geometry files for use in numerical flow and wave modeling applications, and ii) visualization of simulation outputs using model viewers and GIS platforms. The PPUTILS project allows its users to use public domain and open source software to construct the necessary inputs, and visualize outputs, in environmental free surface modeling applications. Specifically, the focus is on:

1. Development of digital surface models,
2. Generation of triangular meshes,
3. Visualization of the numerical simulation output.

The basic structure of the digital surface models and quality triangular meshes in the PPUTILS project is an ascii format easily understood by a variety of numerical models. The PPUTILS project provides conversion scripts to assist users in preparing necessary inputs for TELEMAC and SWAN numerical models.

1.4 Intended audience

The intended audience of this manual is the person undertaking terrain, environmental flow and wave modeling projects. This may include students, researchers, scientists and engineers practicing on a wide range of subjects dealing with river and coastal projects.

2 Getting PPUTILS

The PPUTILS project is hosted on Github and has the following address:

```
https://github.com/pprodano/pputils
```

By going to the above web address, the user can easily download the latest stable release of the code. Under releases, the user should select the latest version of the code. A change-log is included in the code which summarizes what has changed and/or what are the new additions to code.

Adventurous users are also encouraged to try the latest development snapshots, also available through Github. As this is work in progress, these work in progress snapshots may not be the most stable.

2.1 Installing PPUTILS

The PPUTILS project is written mostly in the Python programming language, but there are a few tools written in Fortran and C. The Fortran and C programs have been compiled for Windows, Linux 32-bit and Linux 64-bit, and are included as binaries in the PPUTILS distribution. Python scripts in the PPUTILS project call these binaries when required. The source code for these binaries is included should the user require it.

One of the reasons Python was selected was due to its readily available tools for pre- and post-processing numerical simulation outputs. The scripts in PPUTILS require installation of the Python programming language. All scripts work in Python 2 (v2.7 or later) and Python 3 (v3.4 or later), so either version installed will work. PPUTILS depend on three major Python packages: Matplotlib, Numpy and Scipy. These must be installed in order for PPUTILS to work. If these dependencies are not installed, some of the scripts will just not work.

2.1.1 Installation under Linux

The first step is to make sure Python is installed. Under the popular Linux distributions the Python programming language comes already pre-installed. If not, Python must be installed first.

There are a number of different ways in which the user can install dependencies required by PPUTILS. For Debian based Linux distributions under Python 2, the required dependencies are installed with:

```
$ sudo apt-get install python-matplotlib python-numpy python-scipy
```

Similarly, for Python 3 the installation is:

```
$ sudo apt-get install python3-matplotlib python3-numpy python3-scipy
```

For testing purposes, having both Python 2 and Python 3 is recommended.

2.1.2 Installation under Windows

Under Windows, installing Python and the required dependencies is likewise straight foreword. There are open source pre-compiled products that provide binary versions of the required dependencies. A simplest way is to install QGIS (2017) (which also installs of Python along with Numpy, Matplotlib and Scipy). After QGIS is installed, the user has access to the required dependencies through the OSGeo4W command prompt.

Note that there are other ways in Windows to install Python and the required dependencies for use in the PPUTILS project. For example, there are a number of Python pre-compiled distributions that include the required dependencies for PPUTILS, such as Python(x,y), Anaconda, or Canopy. No doubt, other methods exist too.

2.2 Checking the installation

To check if the dependencies are properly installed, the use has to open the terminal (or the command prompt), change the directory to the PPUTILS root folder, and launch the script called `check_dependencies.py`. All this script does is prints out the version of Numpy, Matplotlib and Scipy to the screen (if they are installed, and exits if they are not). If the dependencies are not installed, most of the PPUTILS scripts will not work.

2.3 Supplemental programs

The main strength of PPUTILS is the pre- and post-processing of numerical model inputs and outputs. All GUI based processing is achieved using tools that are free and in open source. PPUTILS relies on the following tools:

- A GIS platform (QGIS (2017), GRASS (2017), SAGA (Conrad et al., 2015), GDAL (2017), or gvSIG (2015)) for vector and raster based data manipulation,
- Triangle (Shewchuk, 1996), for generation of TIN models,
- GMSH (Geuzaine and Remacle, 2009), for generation of model meshes, and
- Paraview (Ayachit, 2017), for numerical model data analysis and visualization.

To fully utilize the PPUTILS set of tools, the user is highly urged to download the latest binary versions of a GIS platform, GMSH, and Paraview programs. A compiled version of Triangle is included in the PPUTILS project, along with the source code supplied by its author. When required, scripts in PPUTILS call the compiled version of Triangle using Python's subprocess module.

2.4 Summary

For optimal use of the PPUTILS scripts, the user should carry out the following:

1. Download the PPUTILS project code,
2. Install a GIS platform of choice (used for vector/raster data editing, digital surfaces and model output visualization), and preparation of report style figures,
3. Install GMSH (used for numerical model mesh generation), and
4. Install Paraview (used for visualization of digital surfaces and model output).

Note that the PPUTILS code can be entirely executed using Python (with Numpy, Matplotlib, and Scipy dependencies). No other dependencies are required. But a GIS platform is recommended for input data creation, and visualization of certain outputs. Paraview is highly recommended for output visualization.

3 The PPUTILS file format

The PPUTILS project aims to provide its users with an efficient set of tools to accomplish tasks required in typical terrain, free surface flow and wave numerical modeling projects. In being able to execute such tasks, PPUTILS has to be able to read and write vector-based data (such as point clouds, lines and polygons, meshes, and other formats).

The PPUTILS project does not provide its users with a GUI to manipulate geometry files. Instead the PPUTILS tools rely on open source GIS platforms for all geometric editing required for environment modeling projects. Adopting such a methodology means that development of PPUTILS can focus on algorithms that perform processing tasks efficiently while leaving the intricacies of GUIs to programs that are already great at this. (There are many open source GIS applications with excellent and well developed GUIs.) The downside to this approach is that the user is required to perform one extra step and convert output from a GUI program to a format understood by the PPUTILS project.

The basic formats used in the PPUTILS project are custom formats for point clouds, lines and polygons, TINs and meshes. PPUTILS project uses simple ascii based formats that are human readable and easy to understand. This means the user can open them in a text editor or a spreadsheet program and easily examine its contents. The formats processed by PPUTILS are presented next.

3.1 Points file format

The points file format is a simple comma separated ascii format that includes x,y,z values. The format is such that column headings are not included. All values must be comma delimited.

Given the scale of typical river and coastal engineering projects, a projected coordinate systems are used (such as the Universal Transverse Mercator (UTM) projection). PPUTILS stores the x and y coordinates in Numpy's double precision format, so the user need not shift coordinates for fear of truncation errors. In its core code, PPUTILS retains only three digits after the decimal point. Anything after the third decimal point is ignored.

It is recognized that retaining thee digits after the decimal may be a problem for those working in the geographic coordinate system. Should additional digits after the decimal be needed, the user can easily go into the code and adjust as required.

An example of a points file is provided below:

```
378722.83,4656782.6,-5.24
378725.94,4656785.42,-5.06
378728.91,4656788.83,-4.79
378731.81,4656792.32,-5.35
378734.66,4656795.54,-4.76
378737.76,4656798.68,-5.32
```

```
378740.97,4656801.75,-5.56
378744.4,4656804.71,-5.26
378748.09,4656807.35,-5.56
378751.65,4656809.96,-5.44
```

Please note that the point cloud file must be comma separated. If there are spaces between the comma and the value, the file will not be recognized and Numpy's read function will throw an exception error. Of course, the interested user can modify the original source code and expand formatting capability as necessary.

3.2 Lines file format

Similar to the point cloud format, PPUTILS has a simple ascii based file that represents open and closed polylines. For PPUTILS, polygons are defined as polylines that start and end with the same vertex (i.e., the starting and ending vertex must be identical). A polygon is therefore a closed polyline. In the sections that follow, the nomenclature used defines lines as open and closed polylines.

The basic structure of the lines file format in PPUTILS is as follows: shapeid,x,y. Shapeid is an integer identifier of a distinct line, and x and y are its vertices. An example of a lines file is the following:

```
1,379025.0,4657545.0
1,379023.5,4657532.0
1,379022.1875,4657523.0
2,378424.047583,4656843.12896
2,378516.137257,4656808.4285
2,378503.009662,4656779.30165
2,378491.663632,4656754.12764
2,378424.047583,4656843.12896
3,379028.59375,4657511.5
3,379023.427215,4657501.08541
3,379016.862906,4657487.85321
```

By inspecting the above file, we see that it has three lines: line 1 is an open polyline with three vertices, line 2 is a closed polygon (the first and the last vertex are identical) having a total of five vertices, and line 3 is an open polyline having three vertices.

There is no limit to the number of lines that can be included in the line file. As before, the values have to be comma separated, with no spaces between the delimiter and the individual value.

Digital surface modeling using TINs requires use of breaklines that have an elevation value associated with an individual vertex. In PPUTILS, breakline vertices get their elevation values from the master points file that includes x,y,z values of both the masspoints and breakline vertices. An efficient search mechanism (using Scipy's KDTree algorithm) assigns elevation values to each breakline vertex at time

of processing. PPUTILS also includes features to export masspoints and breaklines data to traditional *.dxf and *.shp formats used typically in the engineering and GIS communities. More details on this will be provided in subsequent chapters.

3.3 Triangulation (TIN and mesh) format

For storing triangulation data, such as data used by TINs and meshes, the PPUTILS project relies on the ascii based format used by the ADCIRC model. The ADCIRC format was selected as it is a simple text based format which is easy to for a person to read and understand. By using a simple text file, the user can easily open the mesh file in a simple text processor or a spreadsheet and inspect its contents.

Any triangulation format needs to store individual vertices (i.e., x,y,z values), as well as information on how each triangle is defined (i.e., which three nodes make a given triangle). An example of an ADCIRC triangulation with two elements is provided below:

```
ADCIRC
2 4
1 10.000 0.000 1.000
2 20.000 0.000 3.000
3 20.000 10.000 2.000
4 10.000 10.000 4.000
1 3 4 1 2
2 3 2 3 4
```

The ADCIRC format specification is defined next. First line in the file is a text string, indicating the format of the triangulation. In PPUTILS, the first line (containing a text string) is ignored. The second line in the ADCIRC format indicates the number of elements (two in above), and the number of nodes (four in above), delimited by a single space character. Starting on the third line, the ADCIRC format lists all nodes in the file - each line consists of node id, its x, y and z coordinate. The values are separated by a single space. After the listing of the nodes (four nodes in above), the ADCIRC file includes a list of the elements and describes how they are connected. Each line includes an element id, a number 3 (to indicate three node triangles), and a tupple (a list of three integers) describing how each element is assembled. In the above example, element 1 is created by connecting nodes 4, 1, and 3, while element 2 is created with nodes 2, 3, and 4. When generating ADCIRC meshes, PPUTILS checks that all elements are oriented in CCW fashion. If they are not, their orientation is switched to ensure this criterion is met. As before, the values have to be delimited by a single space.

3.4 Creating PPUTILS formatted files

3.4.1 Point files

Creating, editing, importing and exporting comma separated xyz files is something that GIS platforms do extremely well. Open source GIS platforms can easily do this, as can a variety of others. It is assumed in this manual that the PPUTILS user knows how to graphically create, edit, import and export comma separated ascii xyz files.

3.4.2 Lines files

As alluded to earlier, the user should also be familiar in creating, editing, importing and exporting vector files (polylines and polygons) in a GIS environment. In the context of GIS, these files are typically represented by the ESRI shapefile format. All open source GIS platforms provide the user with tools to create, edit, import and export ESRI Ehapefile formatted files.

The best way of working with vector files in a GIS environment is to work with the ESRI Shapefile format. Let us assume the user has a single file that represents the domain boundary as a closed polygon within one of the open source GIS platforms, and that the file is named `boundary.shp`. Note that the ESRI Shapfile format also includes `boundary.dbf`, `boundary.prj` and `boundary.shx` files.

There is a script in PPUTILS called `shp2csv.py` that converts the ESRI Shapefile format to the ascii based format recognized by PPUTILS. The script uses Joel Lawhead's `pyshp` module (Lawhead, 2013). The `shp2csv.py` script recognizes 2D and 3D ESRI Shapefile types and thus works with geometry files created and edited by various GIS platforms. For example, to convert the boundary polygon from an ESRI Shapefile (created in GIS) to the PPUTILS format, the user would invoke the conversion as follows:

```
python3 shp2csv.py -i boundary.shp -o boundary.csv
```

where the -i is the flag for the input boundary file, and -o is the output PPUTILS file. Note, the same script also works for converting points files from the ESRI Shapefile format to the ascii xyz format used by PPUTILS.

The script `shp2csv.py` recognizes the following ESRI Shapefile types: POINT, POINTZ, POLYLINE, POLYLINEZ, POLYGON, POLYGONZ. The PPUTILS user is thus restricted to the above ESRI Shapefiles types. If other types are used, the converter will most probably not work.

The PPUTILS user is cautioned that before using `shp2csv.py` script the skeleton geometry must have at least one numeric field defined in the input ESRI Shapefile. This means that each line in the file should have a numeric attribute (i.e., a line id number). These are only needed to prevent `pyshp` module writing 'NULL' to the *.csv files when a field value is not present.

3.4.3 Triangulation files

Creation of triangulation files with PPUTILS is covered in detail in the next two chapters. Creation of TINs is dealt with in Chapter 4, while the creation of numerical modeling mesh is presented in Chapter 5.

4 Digital surface modeling

4.1 Introduction

The topic of digital surface modeling presented here focuses on practical aspects on how to create digital surfaces efficiently. The focus of this chapter will not be on the theoretical aspects of digital surface creation nor will it include all of the different ways to create such surfaces. Theoretical aspects of digital surface modeling are covered in texts on the subject (Li et al., 2015). Rather, the focus in this manual is to present the reader means with which to create and modify digital surfaces for use in a range of projects, including numerical modeling. What is presented here is one way of undertaking the creation of digital surface models. No doubt, other means exists too.

In the PPUTILS project, the basic digital surface is a Triangular Irregular Network (TIN) model. A TIN model is a digital representation of a surface that is made up of irregular points in three dimensional space, often constrained by individual lines. In TIN modeling, the irregular points are referred to as masspoints or nodes or point clouds, while the constraint lines are referred to as breaklines. The digital surface in TIN models is constructed by a triangulation algorithm that creates non-overlapping triangles out of the nodes in the data set while respecting the constraints imposed by breaklines.

Applications of digital surface modeling can be found in a variety of subject areas, ranging from geophysics, engineering, geomatics, geography, archaeology and others. As a way of introducing the subject, consider an example from civil engineering. Let us assume that a new design is required for construction of new civil works (a building, a bridge, a dam, a wharf, etc). As the first step in the process a surveyor goes to the site and collects the necessary terrain data using instrumentation at his or her disposal. In the engineering industry, a typical way of communicating the results of the survey is to produce a drawing depicting existing conditions of the project area.

After collection the survey data is typically processed and cleaned to obtain a consistent set of valid data. Data cleaning is necessary to avoid including artificially high or low spots (or other inconsistencies) resulting from measurement or equipment errors.

If the survey data set is rather sparse (as would the case of a typical topographic survey), the user is required to connect individual elevation nodes in order to delineate features in the data (such as top and toe of slopes, water's edge of rivers, crests of dunes along coastlines, etc). Topographic surveys are often visualized and presented on drawings by creating breaklines and/or contours of the area in question. Contours from topographic surveys are created from a digital surface that is generated from masspoints (individual elevations collected) and breaklines (polylines joining similar topographic features). A digital surface is created using an external program that uses masspoints and breaklines as input. In the engineering industry, digital surfaces are typically a TIN, or a model that connects indi-

vidual masspoints with non-overlapping triangles, while respecting the constraints induced by the breaklines.

If the survey data consists of points that is rather dense (as would be the case of multi-beam echo sounder or a lidar point cloud), the user is simply required to triangulate the masspoints and create a TIN model. The TIN model represents a digital surface from which all further processing can be accomplished (i.e., extracting cross sections, calculating volumes, interpolating elevations onto a numerical model mesh or grid, etc).

Note that TINs are not the only means with which to create digital surfaces, but they are the ones covered in this manual. Alternative methods exists that can create digital surface via other methods (i.e., kriging, nearest neighbour interpolation, inverse distance methods, etc.). Open source GIS software packages have such algorithms built into their core. The user simply has to load the masspoints (some even allow for breaklines as inputs), and a desired grid spacing, and the algorithm produces a gridded digital surface (i.e., a DEM). The interested user should consult relevant open source GIS manuals for steps in the creation of gridded digital surfaces from raw data like point clouds.

4.2 TIN modeling engine

For the creation of TIN models the PPUTILS projects uses the engine from a meshing code Triangle (Shewchuk, 1996). According to Shewchuk (1996):

> Triangle is a C program for two-dimensional mesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations, and Voronoï diagrams. Triangle is fast, memory-efficient, and robust; it computes Delaunay triangulations and constrained Delaunay triangulations exactly. ... Features [of Triangle] include user-specified constraints on angles and triangle areas, user-specified holes and concavities, and the economical use of exact arithmetic to improve robustness (p. 203).

The original source code of Triangle is included in the PPUTILS project code. Note that Triangle is a piece of code that is freely distributed, but is not in open source (to the best knowledge of the author). In the PPUTILS project, pre-compiled binaries for Windows and Linux are included in the distribution. A number of PPUTILS scripts call the pre-compiled binaries of Triangle when meshing for TINs is required. Following the creation of the TIN file, PPUTILS scripts use Matplotlib Triangulation library to process the TINs (i.e., perform the necessary interpolations and other data processing tasks).

PPUTILS also gives the user the option of using Triangle for quality mesh generation (for use in numerical modeling). More on the implementation details is covered in later sections.

4.3 Understanding TIN modeling

In order for a user to create a TIN two basic inputs are required: A closed polygon boundary, and masspoints (or embedded nodes). TIN's require a boundary to encapsulate the points from which a digital surface will be created. The masspoints are defined as xyz point cloud file, or nodes that are to be embedded into the TIN. Digital surface modeling with TINs also allows user to specify two additional (and optional) inputs: breaklines and/or islands. Breaklines are drawn by connecting individual nodes in the point cloud in order to constrain where the triangulation algorithm can and can not form triangles. Optionally, TINs can also include islands (or holes), which simply represent regions that are to be excluded from the surface modeling. For example, a project that needs to capture the relationship between storage volume vs. flood stage of a large reservoir would require a TIN that excludes islands (or dry overland areas) from the calculations.

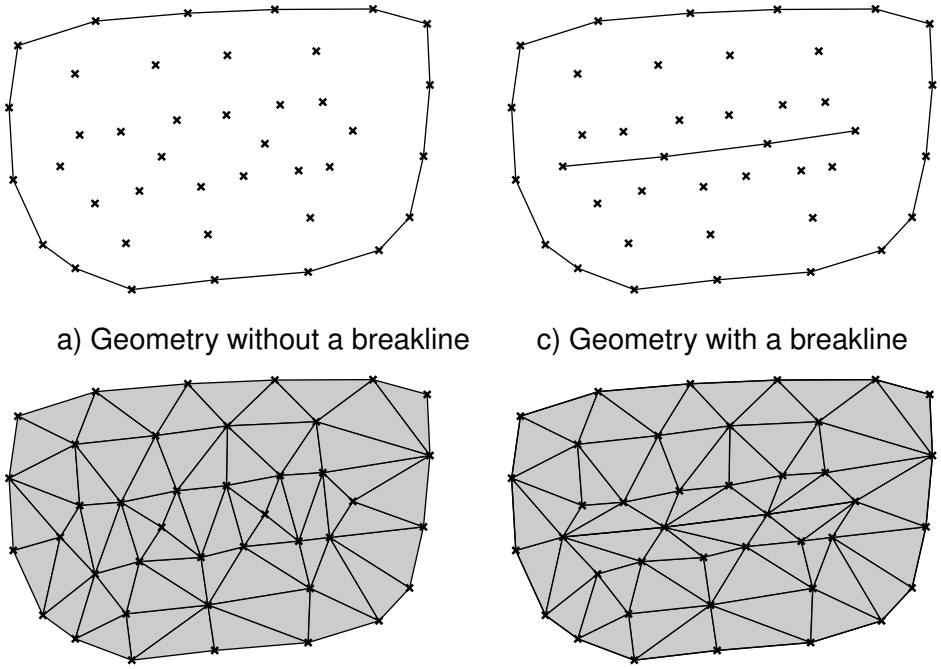
To illustrate the creation of TIN, consider a set of arbitrary point cloud shown in Figure 1a), together with closed boundary polygon. TINs in the PPUTILS project require the user to specify a boundary polygon (in the PPUTILS lines format). Invoking the triangulation program with inputs data from from Figure 1a) results in a TIN shown in Figure 1b), where the masspoints within the boundary polygon are simply connected to form non-overlapping triangles and thus generate a digital surface. For this simplistic example the elevation attribute, z, is ignored (actually, it is assigned a value of zero to create a surface with no relief).

Consider now what happens when a single breakline is introduced to the input, as shown in Figure 1c). The breakline is drawn to connect a number of masspoints within the problem boundary. Note that each vertex within the breakline may have a different z value. A TIN created from this scenario is shown in Figure 1d). The breakline in this example forces the triangulation mechanism to constrain how it generates non-overlapping triangles within the problem domain. By introducing a breakline, the triangulation program is told that it can not form triangles that cross any given breakline.

How and where the user introduces breaklines to the problem will have an impact in the digital surface that is eventually created. For example, breaklines are required to pick up certain physical features in the domain, such as water's edge, top and toe of bank, position of ridges, dunes or other readily identifiable physical features of interest.

Not including breaklines in a TIN model could result in an improper representation of the physical feature, and thus might create a digital surface that does not properly capture the terrain geometry. Breaklines could come from topographic (on the ground) surveys, and/or collected from aerial surveys via photogrametry techniques. Regardless where the breaklines come from, their assembly and use is critical in properly capturing digital surfaces for use in various kinds of projects.

Consider the example where the user is require to create a TIN for a river flow modeling project, where the digital surface must capture the topography of the land



a) Geometry without a breakline

c) Geometry with a breakline

b) TIN without a breakline

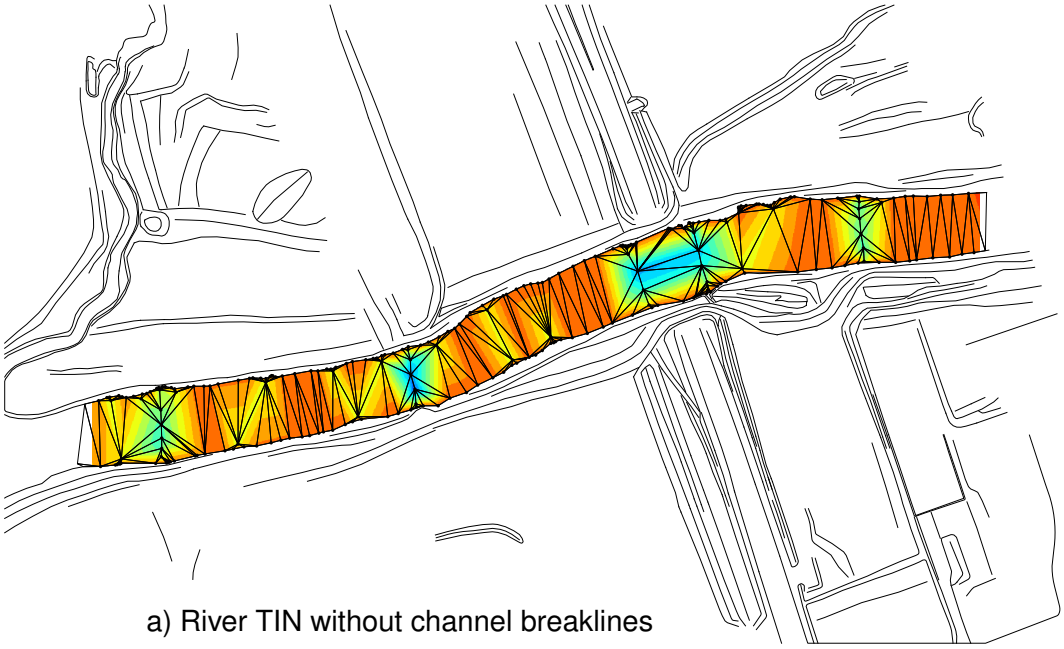
d) TIN with a breakline

Figure 1: Example of a TIN with and without a breakline

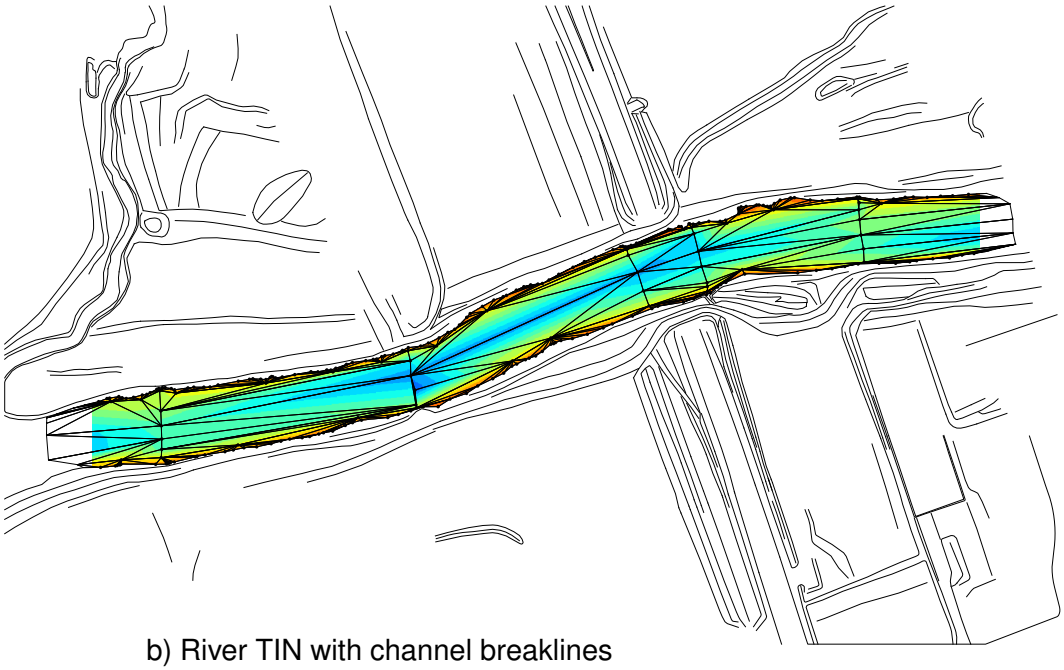
(terrain above water) and bathymetry of the river bed (surface below water). A project like this recently completed by the author used a combination of locally collected topographic and bathymetric data coupled with data provided by an aerial survey. Local survey data was provided as ascii points in xyz format, while the results from an aerial survey were supplied as a set of ESRI Shapefiles.

Simply importing all of the available data, and generating a TIN model produces a result shown in Figure 2a). Upon inspection of the resulting TIN model, it is readily evident that bathymetry of the river bed is not properly captured. This (improperly represented TIN) shows local deep spots where the river cross section data was collected, with high spots between the cross sections. By not specifying breaklines within the river bed, the triangulation mechanism connects water's edge nodes on one side of the river to water's edge nodes on the other side, thus creating localized high spots in the digital surface that are not representative of local conditions.

The correct TIN model for the above case is shown in Figure 2b). This version of the TIN model is generated by inspecting the elevation data in the river cross section data and connecting nodes longitudinally along the river bed to form breaklines inside the channel of the river. The breaklines placed inside the river channel make sure the TIN model properly captures river elevation data between the cross sections. This TIN model correctly shows the thalweg (or deepest part) of the river along the entire reach shown. Addition of breaklines is particularly important when



a) River TIN without channel breaklines



b) River TIN with channel breaklines

Figure 2: Digital surface in a typical river flow modeling project

a meander bend exists between the collected river cross sections. In this case, intermediate vertices (with appropriate elevations) are required to be inserted to ensure river bathymetry is properly captured in the TIN model.

In summary, breaklines in TIN modeling are needed to ensure digital surfaces are properly resolved and are able to correctly capture the geometry in question if the data collected is sparse (as in a case of topographic survey using spot elevations). Having a TIN model that properly resolves the terrain geometry is necessary to produce numerical models and grids for use in numerical modeling, which are the subject of subsequent chapters.

4.4 TIN modeling rules

Before the user can successfully create TIN models for use in real life projects, a number of rules in TIN modeling must be adhered to. In the PPUTILS project, the user should abide by the following:

1. A closed boundary must encapsulate all masspoints and breaklines.
2. A breakline should not cross any other breakline in the TIN*.
3. Breaklines should not be co-linear with other breaklines*.
4. Starting node of a breakline should not lie exactly on a segment of another breakline*.
5. Holes or islands are to be represented with closed polygons.

Strictly speaking, the rules above should be followed whenever possible, as they ensure that the triangulation mechanism will generate a geometrically valid TIN. A valid TIN in this manual is defined as one which is generated by strict adherence to the above rules.

In the PPUTILS project it is also possible to generate invalid TINs, or TINs where the rules marked with an asterisk (*) are ignored. Invalid TINs may produce triangles that have zero areas (generated from two breaklines that are co-linear), or have nodes that are on the segment of another element. (To the best knowledge of the author, PPUTILS does not create TINs in which triangles overlap.) Post-processing invalid TINs within PPUTILS is certainly possible, but scripts are not as flexible as those for processing valid TINs.

A way for a user to ensure that a valid TIN is generated is to ensure that input topology (i.e., boundary, masspoints and breaklines) are geometrically correct and always abide by the above rules. It is possible that input data provided to the user may not follow the above rules. In such instances the user has two options: i) clean the input data and produce input topology that meets the above requirements, or ii) accept that an invalid TIN will be generated. More on the subject of invalid TINs is given in a subsequent section.

4.5 Creating inputs for TIN models in PPUTILS

The way TIN generation in the PPUTILS works is as follows: The user creates the required inputs (boundaries, breaklines, masspoints, and islands, if any) and simply runs the correct Python script to create the TIN. The inputs have to be in the PPUTILS format (see Chapter 3). The output TIN is created in the ADCIRC format, which can then be read by a number of PPUTILS scripts to carry out various operations (like interpolating, extracting cross sections, visualizing, etc.).

It is recommended that the user follows the above rules, which will in turn ensure that a valid TIN will be generated by the triangulation algorithm. Working with a valid TIN in PPUTILS is very efficient, as the scripts use Matplotlib to read in the triangulation and complete the required processing (usually interpolation). Matplotlib uses a Trapezoidal Map algorithm (de Berg et al., 2001) for a point in polygon test, which is an advanced and a computationally efficient algorithm (it can handle TINs with millions of nodes in seconds). The downside with Matplotlib is that it requires the TIN to be valid (i.e., it must be generated by following the above rules). A number of scripts in PPUTILS exist that can work with invalid TINs, but these are much less computationally efficient.

The rest of this section describes the required and optional inputs that can be used by the scripts in the PPUTILS project. The formatting described in Chapter 3 is used for the points and lines files necessary in developing skeleton geometry for use in TIN modeling. The rest of this section sketches out user requirements for the successful generation of TIN models.

4.5.1 Model boundary

For use in TIN modeling with the PPUTILS project, the user is required to develop a closed model boundary. The closed boundary must be in the PPUTILS line format, and have the same (x,y) coordinate for its starting and ending node. The user can easily verify this by opening the ascii file of the boundary and inspecting the coordinates of the first and last vertex. The boundary file must have only one boundary in the file.

The recommended way of creating a TIN boundary is to use a GIS platform and create an ESRI Shapefile encompassing the data. All of the data (masspoints and breaklines) must lie within the closed boundary. After the boundary is generated, the ESRI Shapefile is converted to the PPUTILS file format using the script `shp2csv.py`. Note that upon completion of the `shp2csv.py` script, boundary nodes are automatically written to an ascii file (if needed in a subsequent step).

Sometimes the user may create a TIN boundary using a region manually drawn which may not have elevations associated with its vertex nodes. In such a case, a script called `interpBreakline_from_pts.py` may be called in order to assign to each TIN boundary vertex an elevation from the closest node in a xyz point cloud file.

Model boundary is a required input.

4.5.2 Breaklines

Similar to the TIN model boundary, breaklines in the PPUTILS project are represented using the lines file format defined in Chapter 3. Breaklines file can contain any number of breaklines (the maximum number depends on the computer's internal memory). Each breakline vertex is made up by connecting a line between two vertices from the master nodes file (which is simply an xyz ascii text file). By virtue of this definition, the breaklines in the PPUTILS project do have z values, but rather obtain their z values from the master nodes file during the computations. Scripts in PPUTILS use a sophisticated search mechanism (using Scipy's KDTree) and assign to each breakline vertex an elevation value from the master nodes file. Thus, breaklines in PPUTILS need only have the (x,y) coordinates of each breakline vertex. If z values are in the breaklines file, they are ignored as PPUTILS assigns z values to breakline vertices by searching the master nodes file.

An efficient way managing breaklines in PPUTILS is to store them in an ESRI Shapefile format (so that they may be visualized and edited using a GIS platform). Once the user is satisfied with the breaklines, they have to be converted to the PPUTILS lines format by using `shp2csv.py`.

Sometimes the user may wish to use breaklines data files that already have elevation values embedded in its vertices (such as ESRI Shapefile of type POLYLINEZ or POLYGONZ). These are sometimes referred to as 2.5D or 3D ESRI Shapefiles. If such data is to be used in the TIN with PPUTILS, the user must first convert such files to a PPUTILS breaklines file (using `shp2csv.shp`). This conversion script will also create an xyz listing of the nodes, which are then to be merged into the master nodes file.

Of course, the user may choose to construct breaklines in any other manner, as long as the formatting of Chapter 3 is respected.

Breaklines are an optional input.

4.5.3 Masspoints

Masspoints are a collection of xyz points stored in a comma separated ascii file, and are the nodes to be embedded into a particular TIN. The masspoints are thus simply a listing of a set of coordinates with their corresponding elevation values that will be used in the construction of a TIN model. Note that masspoints, to be taken into account, have to be included into the master nodes file (see below).

Masspoints, or embedded nodes, are an optional input.

4.5.4 Master nodes file

The PPUTILS project uses the concept of a master nodes file which is defined as an xyz file that includes a listing of: i) boundary vertices, ii) breaklines vertices (if any), and iii) masspoints (if any). The master nodes file is simply a listing of all nodes to be used in the TIN model.

A simple way of creating the master nodes file is to take the xyz nodes of the boundary, xyz vertices of the breaklines (if any), and the xyz of the masspoints (if any), and simply merge them in one file. For TINs having in the order of 100,000 nodes this operation can easily be done by copying and pasting ascii text in a text editor. For larger models, command line concatenate functions can be used to merge the ascii comma separated files and thus efficiently generate a master nodes file for use in the PPUTILS project.

The master nodes file is the one that has xyz values of all nodes to be used in the TIN. During the TIN generation, PPUTILS uses the master nodes file to assign z values to boundary and breakline vertices.

Master nodes file is a required input.

4.5.5 Island or hole polygons

The first step in creating islands or holes in a TIN is achieved by creating closed regions in the breaklines file (i.e., making sure that each island forms a closed breakline in the breaklines file). The breaklines representing island outlines are saved in the breaklines file, as above. The second step in the creation of holes is to specify in the ascii text holes file a listing of xy coordinate points within the boundary of the island breakline. The specification of the xy coordinates tells the the triangulation mechanism to 'eat away' the elements generated within the specified closed boundary (i.e., create holes). Note that the hole xy coordinate is not an embedded point but only an indicator of where the hole is to be created.

The holes are thus specified two fold: i) by including closed polygon breaklines in the breaklines file, and ii) by placing xy coordinates in a holes file which tell the triangulation mechanism to create the holes out of the closed breaklines. An example in the next section will illustrate and clarify how all inputs in PPUTILS TIN modeling are to be created.

Holes are an optional input.

4.6 Generating a TIN model

Let's suppose that a user has prepared the necessary input files for the generation of a TIN model with PPUTILS. The input files would typically be:

1. Master nodes file (`master_nodes.csv`)
2. Boundary file (`boundary.csv`)
3. Breaklines file (`breaklines.csv`)
4. Holes file (`holes.csv`)

The above files are required to be comma separated ascii files in the PPUTILS format (see Chapter 3). To generate a TIN with PPUTILS, the user invokes the script `gis2tin.py` as follows:

```
python3 gis2tin.py -n master_nodes.csv -b boundary.csv
     -l breaklines.csv -h holes.csv -o tin.grd
```

where -n stands for the nodes, -b for boundaries, -l for line constraints (or breaklines), -h for holes, and -o for the output TIN (in ADCIRC format). If a TIN is to be generated without -l or -h flags, these must be specified as 'none'. For example, to generate a TIN without line constraints (no breaklines) and without holes (no islands) the user would invoke the call to `gis2tin.py` as follows:

```
python3 gis2tin.py -n master_nodes.csv -b boundary.csv -l none
     -h none -o tin.grd
```

The `gis2tin.py` script works in the following manner: First, it reads the comma separated input files and produces a *.poly file for use by the Triangle mesh generator. Triangle is then called by the Python's subprocess module using the input *.poly file, which in turn generates node and element files in Triangle's format. The last part of the `gis2tin.py` script simply converts the mesh from Triangle's format to the ADCIRC format used in the PPUTILS project.

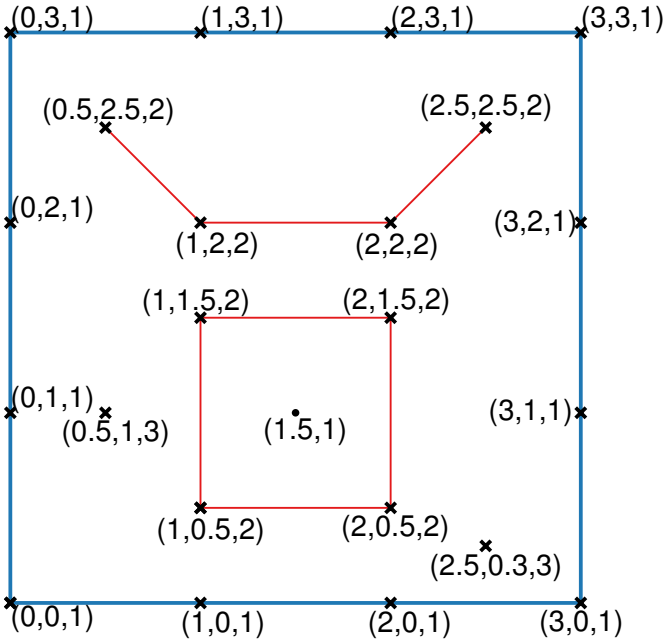
4.7 A simple example of a TIN model

In order to illustrate the generation of a TIN model in the PPUTILS project consider the following simple example. The example is provided only to illustrate the mechanics of TIN creation.

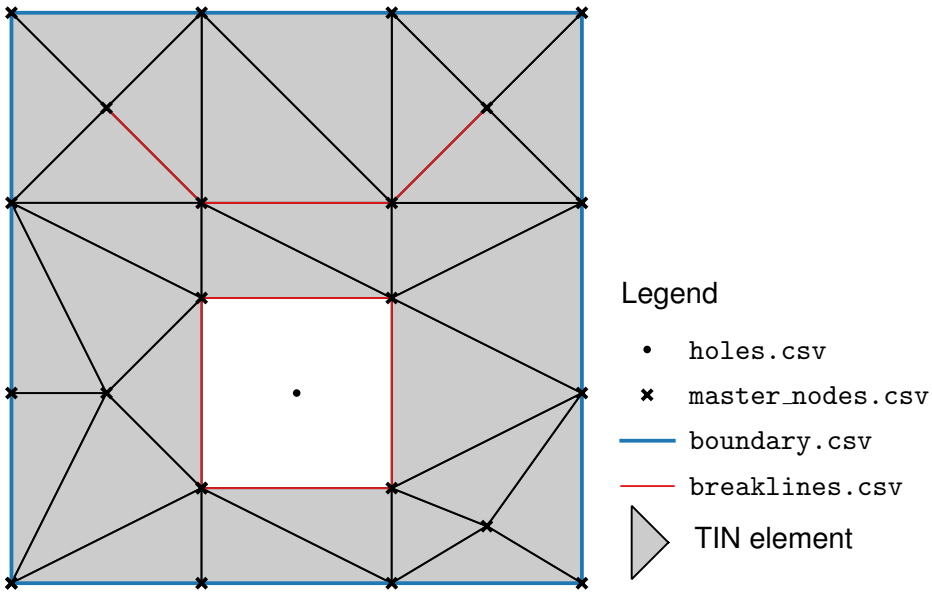
A graphical sketch of the boundary, breaklines, embedded nodes, a hole, and the master nodes file is shown in Figure 3a) for this hypothetical example TIN. Complete listing of the input files (in PPUTILS file format) is provided in Table 1 to assist the user in visualizing the required inputs.

By inspecting the input in Figure 3a) and listing of files in Table 1 we can make the following important observations:

1. The `master_nodes.csv` file contains elevations data for the TIN,
2. Each vertex on the boundary polygon has a z value of 1,
3. The boundary of the TIN is a closed polygon,
4. There are two breaklines in the file, one open and one closed,
5. Breaklines do not cross each other,
6. Both breaklines have z values of 2 for their vertices,
7. There is one hole assigned to the closed breakline (and is intended to create an island in the TIN),
8. There are two embedded nodes in the TIN, located at (0.5,1) and (2.5,0.3).
9. Both embedded nodes have a z value of 3.



a) TIN modeling inputs



b) TIN modeling output

Figure 3: Inputs and outputs to a simple TIN model generated by PPUTILS

Table 1: Listing of input files for the creation of a simple TIN

master_nodes.csv	boundary.csv	breaklines.csv	holes.csv
0,0,1	0,0,0	0,1,0.5	1.5,1.0
1,0,1	0,1,0	0,2,0.5	
2,0,1	0,2,0	0,2,1.5	
3,0,1	0,3,0	0,1,1.5	
3,1,1	0,3,1	0,1,0.5	
3,2,1	0,3,2	1,0.5,2.5	
3,3,1	0,3,3	1,1,2	
2,3,1	0,2,3	1,2,2	
1,3,1	0,1,3	1,2.5,2.5	
0,3,1	0,0,3		
0,2,1	0,0,2		
0,1,1	0,0,1		
0,0,1	0,0,0		
1,0.5,2			
2,0.5,2			
2,1.5,2			
1,1.5,2			
1,0.5,2			
0.5,2.5,2			
1,2,2			
2,2,2			
2.5,2.5,2			
0.5,1.0,3			
2.5,0.3,3			

In order to construct the TIN for the above inputs, the `gis2tin.py` script is called as follows:

```
python3 gis2tin.py -n master_nodes.csv -b boundary.csv
     -l breaklines.csv -h holes.csv -o tin.grd
```

The above script outputs a TIN within the file name `tin.grd`. The TIN is in the ADCIRC file format. The plot of the generated TIN model is shown in Figure 3b).

The TIN produced in the example is rather simple, having only 24 nodes. However, the true power of the TIN generation within the PPUTILS project is for the creation of TIN models that have a large number of embedded nodes and/or large number of breaklines (in the hundreds of thousands or even millions). The need for such TIN models exists in real life engineering projects, where topographic and/or bathymetric features need to be properly resolved to be included in digital surface generation of large areas.

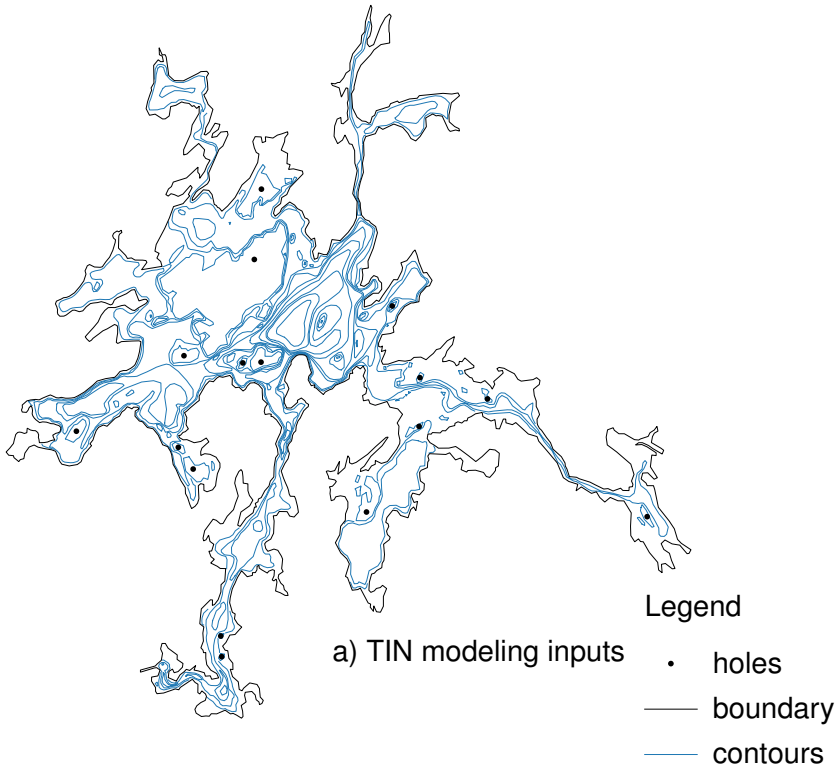
4.8 Examples of real world TIN models

The following section provides example where the PPUTILS project was used to create TIN models in real world engineering projects. The TIN models range in the number of nodes from thousands to millions. In each case the requirement of the project was to produce a digital surface surface to assist in completing a particular project related task (estimating reservoir storage volume, determining flood lines, identifying river currents, etc.).

4.8.1 TIN model of Lake Manitouwabing

A project previously undertaken by the author required construction of the TIN model of Lake Manitouwabing, Ontario. The lake was created following the construction of a dam. The TIN model was needed for hydraulic analyses undertaken as part of the permitting process related to dam's maintenance. The shoreline of the lake was traced from aerial imagery to create the TIN boundary. Elevations of the TIN boundary were assigned a z value of zero (i.e., a zero depth contour). Previous bathymetric survey results (available as depth contours) were used as internal constraint lines in the TIN modeling (i.e., breaklines). Each contour line was assigned a constant elevation value, and saved as a breakline file (i.e., a contour is a breakline with vertices having the same z values). Sixteen islands were specified over an equal number of closed polygons. There were no embedded nodes specified in the TIN model.

Prior to producing the TIN model, every effort was made to abide by the rules given in Section 4.4, as doing so produces valid TINs. Undertaking a visual inspection of the TIN input in Figure 4a) demonstrated (in this case) that the rules were followed. The inputs to the TIN model are a TIN boundary (i.e., the shoreline), the breaklines (the contours of the lake), and nodes indicating where islands should be created. The output of the TIN is shown in Figure 4b). The final TIN model consisted of 5,428 nodes and 9,149 triangular elements.



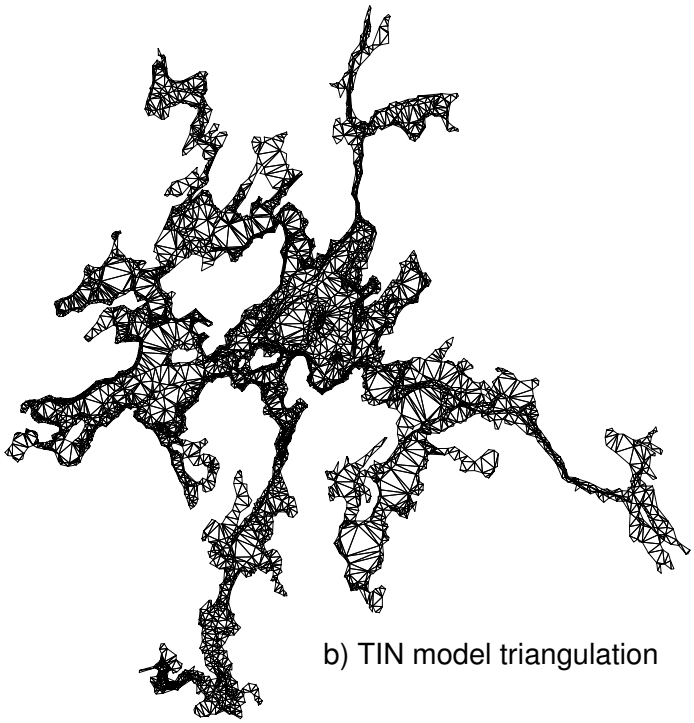


Figure 4: TIN model of Lake Manitouwabing

4.8.2 TIN model of North Branch of the Thames River

A previous river hydraulic modeling project looking at impact of various hydraulic structures on flooding and erosion characteristics required constructing a TIN model of the main river channel and floodplain of the North Branch of the Thames River, located in London, Ontario. The TIN model in this example was required to resolve the geometry of the main channel and the topography of the floodplain such that 2D hydraulic modeling could be carried out. The background elevation data used for construction of the TIN model was a digital terrain model (available as breaklines and masspoints) produced from data collected during an annual aerial survey of the city. The background data was supplemented with bathymetric (below water) and topographic (above water) surveys collected by a local land surveyor using GPS instrumentation.

The data was merged together in a master data set. The masspoints and breaklines from the aerial survey were checked for consistency using on ground topographic data. Minor adjustments were made to the aerial survey data where necessary in order to make them consistent with on the ground survey observations. Additional breaklines were added inside the channel of the river as bathymetric data was available only at select cross section locations. Addition of breaklines inside the river channel was necessary to accurately represent underwater portions of the riverbed between the surveyed cross sections (a condition absolutely critical for 2D floodplain modeling). Features relevant to hydraulic modeling were also included into the digital surface, such as other dykes in the study reach, bridge embankments, roadways (which could get over topped and act as weirs during heavy flooding), small creeks, and others. After all manipulations were completed, a final set of master nodes and breaklines was created and used to generate a TIN. The script `gis2tin.py` was used to create the TIN model for the project.

Subsequent processing of the TIN model revealed that certain breaklines intersected other breaklines and/or each other, while other breaklines had nodes on a segment of other breaklines. This meant the input data did not implicitly follow the TIN modeling rules from Section 4.4, and that PPUTILS generated an invalid TIN. Recall that Trapezoidal Map algorithm implemented by Matplotlib (and relied on by PPUTILS for speed) can only work with TIN models that are created by following the rules in Section 4.4.

Given the size of the modeling domain (and the fact that there were in the order of 13,000 individual breaklines having a total length of 652 km) manually detecting locations where TIN modeling rules were not followed was simply not possible. To proceed forward two options were available: i) keep the invalid TIN and use the less efficient scripts that can process invalid TINs, or ii) clean the input topology such that TIN modeling rules are followed, and re-create a valid TIN. For this project, the breaklines from the aerial survey were cleaned to ensure rules in Section 4.4 were obeyed. The generated TIN had 107,142 nodes with 208,875 triangular elements. TIN inputs from are shown in Figure 5, while the output triangulation is depicted in Figure 6. A 2 m x 2 m grid, created from the triangulation is shown in Figure 7.

Figure 5: North Branch Thames River TIN model inputs

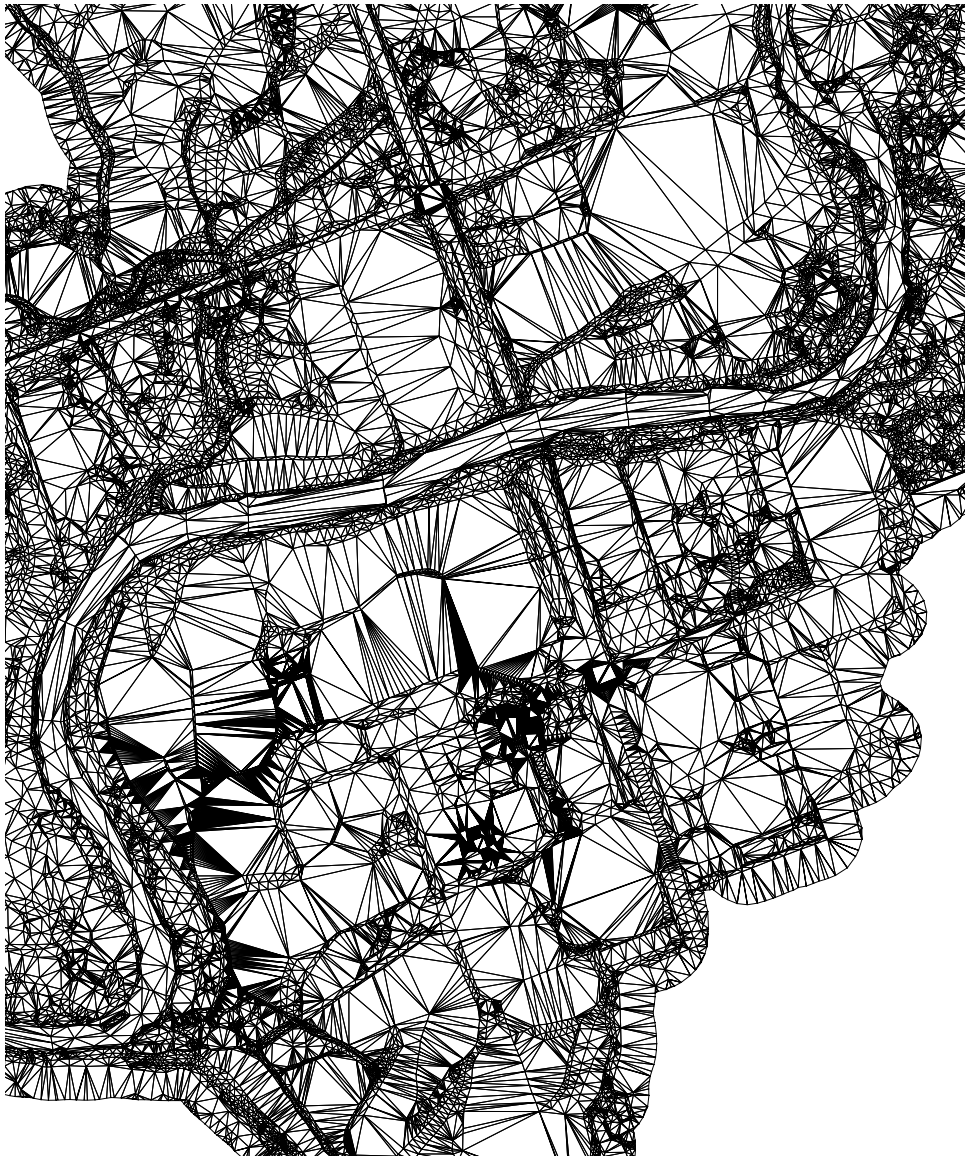


Figure 6: North Branch Thames River TIN model triangulation

Figure 7: North Branch Thames River 2 m x 2 m grid

4.9 Cleaning bad topology in TIN input data

The TIN modeling rules presented in Section 4.4 ensure the triangulation generated by PPUTILS (which uses Triangle as the mesh generation engine) is valid. Conversely, if the rules are not followed the TIN will still be generated, but it may not be able to be processed using Matplotlib's implementation of the Trapezoidal Map algorithm. Using Matplotlib's Trapezoidal Map algorithm provides the user an extremely efficient way of processing triangulations (interpolation, gridding, etc.). Of course, PPUTILS does allow the user to work with invalid triangulations, but with a significantly increased computation time. For small TINs the increased processing time may be acceptable. However, for very large TINs a better strategy is to clean the input data to ensure the rules in Section 4.4 are obeyed.

In cleaning the input topology, the user has two options: i) manually identify where in the input data the rules are not followed, and adjust the inputs as necessary, or ii) use an external tool to automatically clean the input data. The benefit of option i) is that it can be done rather quickly if TIN has a small number of breaklines. However, for a TIN having thousands of breaklines as input (when they are obtained from an external data source), an automatic procedure is required.

Unfortunately, PPUTILS does not have a script to automatically clean breaklines which would guarantee generation of valid TIN models. PPUTILS include in its scripts ways to eliminate duplicate nodes. However, methods that remove coincident breaklines are not yet part of PPUTILS, nor are methods that prevent a breakline to start on a segment of another breakline. For the time being, data cleaning of a breakline file is best achieved using the v.clean module from GRASS GIS. The procedure to clean breaklines in GRASS GIS is the following:

1. Convert the PPUTILS breakline file to a 2D ESRI Shapefile using the script `breaklines2shp.py`, making sure the option -t 2d is specified (-t flag stands for the type of ESRI Shapefile to be generated). Ensure the resulting ESRI Shapefile does not have z values as part of its geometry (i.e., probe the file with `probeshp.py` script, and verify that the file type is either POLYGON or POLYLINE).
2. Load 2D ESRI Shapefile into GRASS GIS.
3. Apply the v.clean module with the snap tool, and provide a small threshold (i.e., 0.1 m or less). The snap tool snaps vertices to another vertex not farther than the threshold distance.
4. Apply the v.clean module, using the break option. In case the breaklines cross each other, the break tool will split each offending breakline and insert a new vertex at the intersection point.
5. Apply the v.clean module, using the rmdupl option. This tool removes duplicate geometries, and should be applied after the break tool.
6. Save the cleaned breaklines file from GRASS GIS to an ESRI Shapefile.

7. Convert the cleaned ESRI Shapefile to the PPUTILS lines format using the `shp2csv.py` script.

The above topology cleaning procedure will generate additional nodes at locations where breaklines intersect. These nodes will automatically be assigned a z value of the intersecting node from the master nodes file using the closest node of the master file to the vertex of the intersection point. Based on the tests carried out by the author, applying the above cleaning procedure to the input breaklines seems to produce valid triangulations. The truth value of this statement needs to be strengthened by additional user testing however.

Note that the above procedure does not eliminate coincident lines in the breakline file. During TIN model generation, Triangle will display a warning message when coincident lines are detected in the breaklines file. The coincident lines do not seem to pose problems in processing.

It is recognized that data cleaning outlined in this section should be part of PPUTILS (i.e., the cleaning should be embedded into the source code and the user need not be bothered with the task). The topology cleaning is a feature that will be part of a future release.

4.10 Visualizing and processing TIN models

Above sections focuses on the procedure for generating a TIN model. A detailed set of descriptions was provided so that the user could develop the necessary inputs required to produce and manipulate digital surfaces using PPUTILS. In this section it is assumed that the user has produced a valid TIN model via PPUTILS. The triangulation is assumed to be stored in the ADCIRC ascii mesh format (see Chapter 3).

4.10.1 Gridded colour coded rasters

A useful way of visualizing a TIN is to convert the PPUTILS TIN (from the ADCIRC format) to a gridded file to be visualized in GIS. PPUTILS use two different file formats in gridded files: i) ascii based ESRI *.asc format, and ii) binary based ESRI *.flt format.

If the PPUTILS user wants to convert the entire TIN to an *.asc file, the following command should be invoked:

```
python3 adcirc2asc.py -i tin.grd -s 10 -o tin.asc
```

where the -i is the TIN in the ADCIRC format, -s is the grid spacing (10 m x 10 m in above example), and -o is the output grid file (in the ESRI *.asc format). The generated *.asc file can then be easily loaded into GIS and visualized.

Similarly, the user can produce the binary ESRI *.flt format by invoking:

```
python3 adcirc2flt.py -i tin.grd -s 10 -o tin.flt
```

where the -i and -s arguments are the same as above, and the -o is the output file grid file (in the ESRI *.flt format).

Suppose now that the user wishes to create a grid from only a portion of the TIN, this can be achieved by invoking the following:

```
python3 adcirc2asc_bnd.py -i tin.grd -b bnd.csv -s 10 -o tin.asc
```

where the -i, -s, and -o arguments are as above, while the -b argument is the closed boundary in the PPUTILS lines file. The output *.asc file is created only within the bounds of the closed boundary that is specified as input.

Similarly, the script that generates a gridded *.flt file for a portion of the TIN is invoked as follows:

```
python3 adcirc2flt_bnd.py -i tin.grd -b bnd.csv -s 10 -o tin.flt
```

where the -i, -b, -s, and -o arguments are as above.

The above PPUTILS scripts rely on Matplotlib's Trapezoidal Map algorithm to perform interpolation of surface data onto a specified grid. The algorithms are extremely fast, and are able to generate grids having millions of nodes in a manner of seconds. Such a feature allows creation of fine resolution grids which can then be imported it to a GIS platform. Once in GIS, the user is given a myriad of tools for raster editing and manipulation (smoothing, contours, volumes, etc.).

An example of a TIN model that is converted to a 2 m x 2 m regular grid is shown in Figure 7.

4.10.2 TIN nodes and elements in GIS

Suppose the user wishes to visualize TIN nodes and elements in GIS, and overlay it with a colour coded raster and/or an aerial photograph. There is a script in PPUTILS called `adcirc2shp.py` that converts the TIN file (in ADCIRC format) to an ESRI Shapefile that could be viewed in GIS. The conversion is achieved by executing the following:

```
python3 adcirc2shp.py -i tin.grd -o tin.shp
```

where -i is the flag for the input TIN, and -o is the flag for the output ESRI Shapefiles. The above script will actually produce two different ESRI Shapefiles: `tin_n.shp` and `tin_e.shp`. The former is an output of TIN nodes while the latter is the output of the triangulation (where each triangle is written as an individual polygon). Note that PPUTILS will also write *.dbf and *.shx files. As PPUTILS does not manage projections, the *.prj file is not written. The projection will have to be specified in the GIS platform when the file is loaded.

The PPUTILS user is also given another option to view nodes and elements of a TIN with a GIS platform. The script `adcirc2wkt.py` converts the TIN model to a Well Known Text (WKT) file format. The conversion is completed as follows:

```
python3 adcirc2wkt.py -i tin.grd -o tin.csv
```

Similarly, the above script produces `tin_n.csv` and `tin_e.csv` outputting node and element files, respectively.

Should the user wish to convert the TIN model to the *.dxf format, the conversion is done as follows:

```
python3 adcirc2dxf.py -i tin.grd -o tin.dxf
```

where -i is the flag for the input TIN, and -o is the flag for the output *.dxf file. The output is an ascii based *.dxf file that can be read by many different CAD packages.

Triangulations shown in Figures 1, 2, 3b), 4b), and 6 were all created by converting a TIN model from the ADCIRC format to the ESRI Shapefile and visualized in a GIS package.

4.10.3 Relief visualization with Paraview

An alternate way of visualizing the TIN is via Paraview, an open source package for data analysis and visualization. The strength of Paraview is its ability to quickly load and visualize large data sets. In this Section the focus will be on getting the TIN output to Paraview. Subsequent section will demonstrate how to get TELEMAC model output to Paraview.

Similar to above scripts, to get the TIN (in ADCIRC format) to Paraview, the following script in PPUTILS is invoked:

```
python3 adcirc2vtk.py -i tin.grd -o tin.vtk
```

where -i is the flag for the input TIN, and -o is the flag for the output *.vtk file. The *.vtk file is Paraview's legacy ascii format. To load the output TIN in Paraview, the user simply has to launch the application and open the *.vtk file. For very large TIN models, it is the author's experience that the *.vtk file should be converted to Paraview's binary format. The binary files are loaded by Paraview much more quickly than ascii based files. A simple way to convert to the Paraview binary format is to open the genererated ascii file, and simply save it as a binary file. A future version of PPUTILS will include scripts to write Paraview's binary files.

A particularly useful feature of Paraview are its filters. When visualizing digital surface models over large spatial area using exaggeration of the vertical may provide the user a better perspective. Paraview has a filter called 'Warp by Scalar' which the user can use to exaggerate the vertical. For domains in free surface flow and wave modeling application, the exaggeration factor of 10 typically gives appreciable visual appeal.

4.11 Cross sections and profiles from a TIN model

A useful way of visualizing terrain data is by extracting cross sections and profiles from the TIN. In the PPUTILS project such extractions are achieved by executing the following command:

```
python3 interpBreakline.py -t tin.grd -l lines.csv -o lines_z.csv
```

where -t is the input TIN in ADCIRC format, -l is the lines file in PPUTILS format containing cross sections and/or profile data, and -o is the output lines file where the input vertices have been assigned elevation values from the TIN. The `interpBreakline.py` script works by taking an existing TIN and a lines file, and assigns an elevation attribute to each vertex in the lines file. The lines file can have one or multiple cross sections. Extracting multiple cross sections or profiles is particularly useful for projects that require cross sections along a length of a river, slope, dyke, etc. The final output of the script (`lines_z.csv` in the above example) is a PPUTILS formatted lines file, but also has a station attribute. The station attribute is a distance along the cross section line starting with zero at the beginning vertex. The user can take the output file and plot the data using their own favourite visualization program (spreadsheet, CAD, GIS, Matplotlib, Gnuplot, etc.).

a) line with two vertices b) line with ten vertices

Figure 8: Distribution of vertices for cross section and profile extractions

A limitation of the `interpBreakline.py` script is that it works by interpolating the elevations for the vertices of the cross section. This means if the cross section is specified by two points (Figure 8a), the interpolation will be performed only for the two end points (elevations along the line segment between the two points are ignored). To extract a more meaningful cross section or a profile from the TIN the user should discretize each cross section into a large number of points (as in Figure 8b), and then carry out the interpolations. Discretizing the cross sections or profile lines will generate an elevation value along for every vertex on the polyline. Note Figure 8b) is shown having 10 points along its length, but cross sections and profiles in real world project may require 100 or more points. Such discretization of the input cross sections can easily be accomplished using a GIS platform. (QGIS has a tool called ”Densify geometries given an interval” that can accomplish this with one click on a button).

It is important to note that the above also works when cross sections or profile data consist of line segments that change direction (i.e., a profile along the thalweg of a river). To obtain a meaningful profile plot, the user should discretize the input (the thalweg in this case), and then carry out the interpolation.

The main output from the `interpBreakline.py` script is the PPUTILS lines file with the following columns id,x,y,z,sta; the id the identifier for the line, x,y,z are the vertices of the line, and the sta column is the station variable that is used to measure the distance along the line.

The `interpBreakline.py` script also outputs an ascii formatted *.csv file that contains the following field values: River, Reach, RS, X, Y, Z. This is a geometry

file that can easily be imported into the HEC-RAS hydraulic model. Therefore, given an existing TIN model and an appropriately re-sampled cross sections file the user can generate a geometry file that can be used in 1d hydraulic modeling using HEC-RAS. Interested users are encouraged to explore this feature of PPUTILS.

4.12 Summary

This Chapter provided a description of tools that allows the user to create and manipulate digital surface by using with only free and open source software. In completing this Chapter, the user should:

1. Understand what a digital surface is,
2. Be aware of different types of digital surfaces (TINs, grids),
3. Appreciate the rules to be followed when building TIN models,
4. Know when and where to add breaklines to generate a proper TIN,
5. Recognize bad TIN input topology, and know how to clean it,
6. Know how to prepare inputs for TIN modeling,
7. Be able to generate a TIN,
8. Know how to visualize a TIN, and
9. Know how to extract cross sections and profiles from a TIN.
10. Recognize that a properly resolved TIN is critical in numerical modeling.

5 Numerical model mesh generation

5.1 Introduction

This Chapter deals with the topic of triangular mesh generation for environmental free surface modeling projects (where the phenomena under study ranges from sub-meter to hundreds of meters or even thousands of kilometers). The problems studied in such projects range from small flume studies (which a researcher uses to understand a particular phenomena) to flood studies (river floods or coastal storm surges) to studies looking at wave generation and propagation of entire seas or parts of oceans. These studies are nowadays completed using 2D and 3D numerical models which require careful discretization of the study area into large number of discrete computational elements. For this work discretization is assumed to be in terms of 2D triangular elements.

Generating regular grids with PPUTILS is achieved by building a digital surface of the area first (see Chapter 4) and then converting the digital surface (or a portion of the digital surface) into to a regular grid. The discretization of the regular grid (i.e., its spacing) is a parameter to be specified by the user.

In order to discretize the study area into a mesh of triangular elements a mesh generator program is required. For 2D triangular mesh generation the PPUTILS project uses existing meshing tools available in the public domain. A number of such meshing engines exist. The meshing engines were developed to be independent of scale of the phenomena under study (i.e., they equally apply to the study nano scale as they do to studies encompassing hundreds of kilometers). This is what make them attractive to the environmental free surface modeling field.

A difficulty in using meshing tools in the public domain relate to the learning curve its users must subject themselves to in becoming proficient with their use. Each meshing tool in the public domain has a custom scripting language and its own formatting. The toolkit developed in the PPUTILS project is an attempt to provide the environmental free surface modeler access to existing mesh generation engines using GIS as the basic interface for constructing geometry. The GIS geometry is saved and then converted by PPUTILS to inputs understood by the meshing engines. This work thus assumes that the environmental flow modeler is familiar with at least one GIS platform. As noted earlier, there are a number of excellent GIS platforms in open source such as QGIS (2017), GRASS (2017), SAGA (Conrad et al., 2015), GDAL (2017), and gvSIG (2015).

The ultimate goal of this Chapter is to illustrate how to build meshes to be used in simulations of environmental flow modeling problems using only free and open source tools. This Chapter explicitly does not assist the user in producing numerical simulation results using any modeling system. The tools part of the PPUTILS project assist users only with the generation of quality meshes. It will be up to the individual user to figure out how to set up and execute numerical models of their choice and generate results from them.

5.2 Mesh generation engine

The PPUTILS project uses the GMSH mesh generation engine for construction of numerical modeling meshes (Geuzaine and Remacle, 2009). GMSH is an all purpose 2D and 3D finite element mesh generator (more than just triangular meshes can be generated), with a built-in GUI CAD engine for pre- and post-processing. GMSH's GUI is developed using FLTK GUI toolkit, making it fast, light while providing its users advanced graphical input and visualization features. GMSH is available as open source (pre-compiled binaries are available for all major platforms). The GMSH GUI has different modules for geometry, mesh, solver and post-processing. Geometry for GMSH can be created interactively using the GUI, or be imported from external files using a number of different formats (unfortunately, GIS formats are not yet included). Input geometry can also be developed using GMSH's text based steering files. GMSH can be used through its GUI, or through the command line. Both of these features make it appealing to the PPUTILS project.

For the generation of triangular meshes the PPUTILS user is provided with scripts that convert topology built using GIS to the GMSH steering language format. This means that the user can use GIS to produce skeleton geometry for GMSH, which, after applying an appropriate conversion script, can then be imported to the GMSH GUI where meshes for numerical models can be generated.

The PPUTILS project assists its user in getting the geometry from GIS to GMSH for the purposes of numerical model mesh generation. Once the geometry is imported to the GMSH GUI, the user will be directed how to generate meshes (inside the GUI, or via the command line). The user should be aware that this work is not a manual for GMSH, its GUI, or its many options. The PPUTILS user is thus encouraged to explore and learn the intricacies of the GMSH tool and its GUI. By doing so, the user will be able to generate better and more appropriate meshes for their own study areas.

The benefits of using GMSH for numerical model mesh generation are many. The way in which GMSH manages the growth from small to large elements is quite efficient, as are its different methods in generating model meshes. For example, GMSH offers its users different methodologies for generating different meshes using the same base input data. More on this topic is presented in subsequent sections.

5.3 Understanding mesh generation

The content of this section parallels a section of the same heading written in Chapter 4 that relates to TIN models. When it comes to constructing meshes for numerical simulations, the user needs to ensure that the mesh i) is refined enough to pick up features of interest in the study, and ii) consists of as little elements as possible to limit computation time. In other words, the mesh can not be arbitrarily fine (that would pick up all features in the topography), as such a fine mesh would lead to unreasonable computation times in a numerical model. The mesh can also

not be too coarse as it will not be able to pick up features relevant to the modeling of the problem at hand. A fine balance must therefore be struck between mesh resolution and numerical model's computation time when constructing numerical modeling meshes.

In the paragraphs that follow, a brief introduction is provided on construction of two types of meshes in the environmental free surface modeling, namely those associated with project studying hydraulics of i) inland rivers, and ii) coastal areas.

5.3.1 Meshes in river and floodplain studies

When constructing a mesh for river and floodplain studies, the numerical modeler needs to be aware a prori which features need to be included in the mesh (i.e., which features need to be resolved in the mesh, and which features can safely be neglected without influencing model output). As an example consider the mesh used in the floodplain modeling study shown in Figure 9. The numerical model mesh is overlaid with the 2 m x 2 m colour coded raster generated from a TIN model. By inspecting the numerical model mesh it is readily apparent that roadway and its approach embankments were resolved in the mesh. Resolving the roadway (running north-south) is absolutely critical as the roadway may act as a weir in the case that flood waters on the eastern part of the domain rise and over top the road. Feature lines were constructed (and appropriately discretized) to represent the toe and the top of the roadway on each side of the floodplain. The roadway in this case acts a dyke running perpendicular to the direction of the flow in the main channel.

Another feature included into the mesh in Figure 9 is the main channel of the river. The river channel was resolved using an orthogonal mesh generated along the study reach using six elements between the left and right banks. Such a fine mesh between the banks of the river is needed to properly resolve numerics during typical low flows (when the water's surface elevation is much lower than the top of bank). Using orthogonal mesh for the main channel of the river ensures that both low and high flows will be properly resolved in unsteady simulations of a typical flood wave, while minimizing the number of channel nodes by using elongated triangular elements.

Figure 9 also shows a number of features in the TIN that were explicitly selected not to be resolved in the numerical model mesh. For example, the topography in the TIN shows an area of higher elevation west of the roadway (depicted in blue). The mesh in this region does not resolve the toe or top of slope but rather simply discretizes the area using mesh constraints imposed by the modeler. The mesh also does not resolve the creek that runs along the toe of slope of the higher elevation area shown in blue (see Figure 7). Dykes surrounding the water treatment plant south of the roadway are also not resolved in this mesh.

Knowing which features to capture in the numerical model mesh depends on a number of factors, including i) experience of the modeler, ii) knowledge of local hydraulics, iii) topography of the area, and iv) specifics of the problem to be investigated. For example, a detailed resolution of upstream tributaries need not

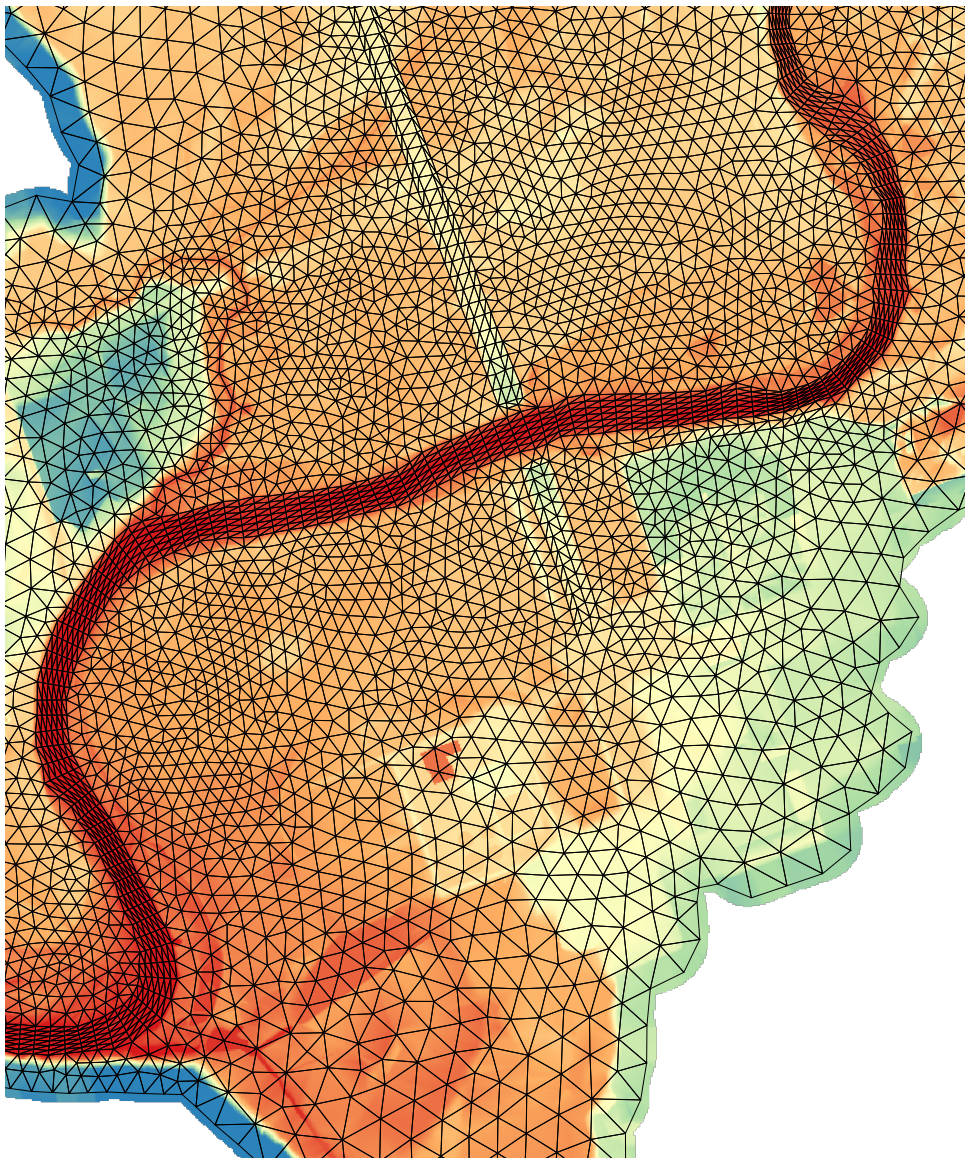


Figure 9: North Branch Thames River mesh

be included if the goal of the study is to examine flooding at a site much further downstream).

5.3.2 Meshes in coastal studies

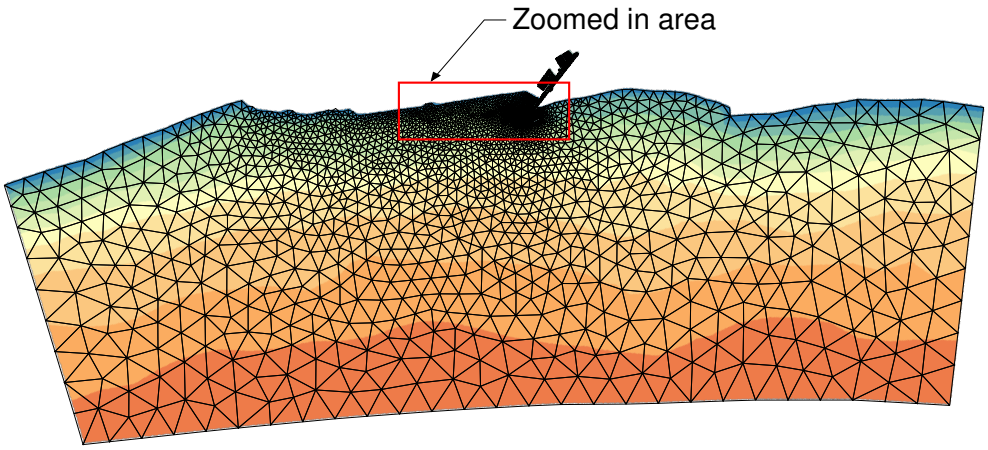
Prior to constructing a mesh for a coastal numerical modeling study the user needs to understand the strengths and limitations of the numerical models itself. For example it is reasonable to expect that two studies of the same geographic area (say an existing harbour) will require different meshes as long as different numerical models will be used. Consider that coastal studies for an existing harbour are required to quantify i) wave agitation inside the harbour limits, and ii) estimation of wind driven currents at a site where dredging is contemplated. Wave agitation studies would likely require a phase resolving model of the harbour and its main entrance structures, with the requirement that the numerical modeling mesh has at least 5-10 nodes per wavelength of the incident wave. For the coastline on the Great Lakes this amounts to a mesh spacing of about 5 m. A coastal circulation model of the same harbour may be sufficient with node spacing of 20 or 50 m (depending on the harbour geometry), but may require a much larger modeling domain.

An example of a coastal mesh generated using the PPUTILS project is shown in Figure 10. The mesh was generated as part of a coastal study used in previous harbour dredging project. One of the tasks of the study were to estimate the current speed and direction adjacent to the harbour entrance in response to dominant wind and wave generated currents. Of particular note in this example is the way in which the mesh was discretized to suit the problem needs. The model requirements were such that the harbour entrance needed to be discretized with a fine mesh in order to resolve the currents between the shoreline and the offshore breakwater, while the size of the remaining elements could be much larger. Element edges between the harbour entrance and the outer edges of the domain vary by an order of magnitude (20 m near the harbour entrance, and 350 m at the edges of the domain).

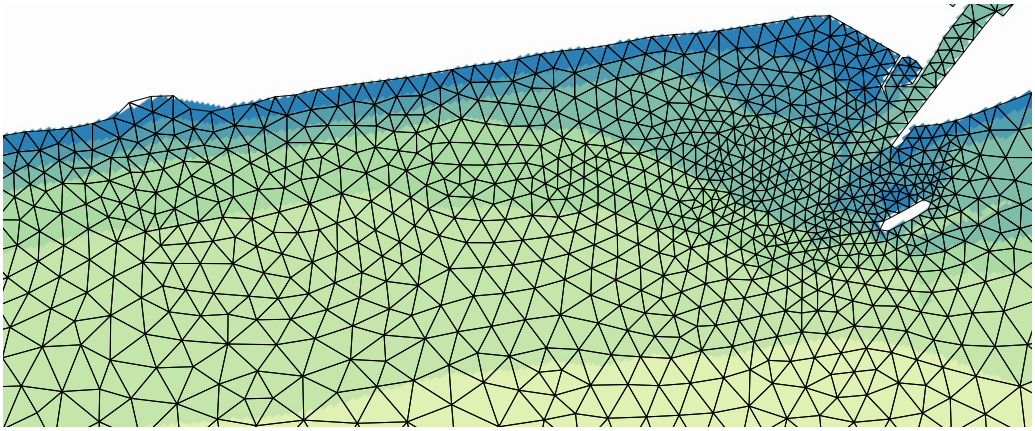
The mesh for this example was created by re-sampling segments of the shoreline to the desired mesh spacing. Polyline segments near the harbour were re-sampled with a 20 m node spacing. Polyline of the shoreline segments further away from the harbour were re-sampled gradually to 50 m, 100 m, 200 m and eventually 350 m at the offshore most boundary. The mesh was generated using GMSH. The growth in the element edge lengths were controlled by the spacing of the boundaries and/or internal constraint lines. An example of how to achieve this using PPUTILS is provided in a later section.

5.4 Numerical model mesh modeling rules

The content of this section loosely parallels Section 4.4, except that hard and fast rules can not be provided for numerical model mesh generation (as they were for TIN modeling). The reason being is that there are too many different numerical models to warrant application of a single set of standard rules. In other words, the



a) Wheatley model mesh



b) Close up near harbour

Figure 10: Wheatley Harbour hydrodynamic model mesh

rules for numerical model mesh generation are model (and application) dependent. This means that each model requires its users to adhere to a slightly different set of rules when generating a 'good' mesh. A 'good' mesh for one model, may not be suitable for numerical modeling with another model. Despite of this difficulty, the following set of remarks are offered for consideration when making numerical modeling meshes:

1. Understand the strengths and limitations of the numerical model,
2. Be aware of the model's scope of application,
3. Use the numerical model only for its intended purpose,
4. Choose limits of the numerical model mesh (i.e., mesh boundary) such that the boundaries do not influence results,
5. Develop a digital surface model (i.e., a TIN) that encompasses the entire model boundary,
6. Identify relevant features in the digital surface model that can impact model results (islands, shoals, slopes, dykes, roads, bridges, etc.),
7. Decide which features need to be included in the mesh, and which could safely be ignored,
8. Select node spacing for different areas of the domain,
9. Discretize the mesh boundary and the mesh constraint lines to a desired mesh spacing,
10. Adjust discretization of the mesh boundary and mesh constraint lines to ensure smooth transitions in zones where mesh spacing changes,
11. Generate a numerical model mesh, and
12. Assign properties to nodes the numerical model mesh using previously built TIN models (bottom elevations, spacially varying friction coefficients, etc.).

Mesh generation for use in numerical modeling application tends to be iterative. After initial construction, the mesh will be used in a numerical model simulation to obtain a preliminary set of results. Upon inspection of the initial results, the user may notice areas where the mesh is either too coarse (and does not resolve features of interest) and/or is too fine (thus being computationally inefficient). At this step, the user is required to carry out some local adjustments to the mesh (i.e., adjusting boundary and/or constraint lines). After the necessary adjustments to the inputs, the mesh is refined to address the inconsistency noted by the modeler. After a number of such iterations, the mesh will be ready for further simulations.

5.5 Discretizing (re-sampling) vector geometries using GIS

For the purposes of this work, it is assumed that node spacing on the boundaries and constraint lines are the mechanism that ultimately controls element spac-

ing of the resulting mesh. There are a number of ways to discretize (or re-sample) vector geometry using a GIS environment. For example, QGIS has a tool called 'Create points along lines' that will re-sample a polyline to a set of points spaced x units apart, where x is an input parameter. The set of re-sampled points can then be used to re-create a re-sampled polyline using a 'Points2One' tool that links the points to create the re-sampled polyline. The re-sampled polyline can then be manually edited and adjusted as necessary, particularly at transition zones where mesh spacing changes. The above represents one way to accomplish discretization of vector geometry using GIS. There are other methods as well. It is left up to the user to experiment and learn alternate re-sampling strategies that suit their favourite GIS platform.

After re-sampling the polyline (closed boundary and/or internal constraint lines) using GIS, the resulting file needs to be converted to the PPUTILS lines format. It is assumed that the user understands how to accomplish this task. If not, please re-read Chapter 3.

5.6 Creating inputs for mesh generation in PPUTILS

The necessary inputs for generating numerical modeling meshes are strikingly similar to the inputs used to generate a TIN (see Chapter 4). There is one main difference however –discretization (or re-sampling) of the boundaries and constraint lines. When carrying out TIN modeling, the user does not need to think about node spacing at the boundaries and/or the breaklines, so long as these have their proper elevation values and are appropriate for the TIN. Mesh generation for use in numerical modeling often use node spacing on the boundaries and internal constraint lines to control the element spacing of the resulting mesh.

The user should be aware that different mesh generation engines may use different mechanisms to control mesh spacing in addition to using node spacing along the boundaries and constraint lines. For instance, GMSH can use a variety of attractor algorithms to control growth of the mesh, while a script in the PPUTILS project (`gis2mesh2.py`) uses Triangle to generate a mesh that is refined based on a criteria extracted from a TIN model. These alternate mechanism are not covered here as satisfactory results can be obtained simply by controlling node spacing of the boundaries and constraint lines. The interested users are encouraged to explore these and other ways of controlling mesh spacing for their own projects.

The rest of this section describes inputs that are necessary when developing quality meshes for use in numerical simulations.

5.6.1 Model boundary

Similar to generating a TIN model, the user generating a mesh for use in numerical modeling requires a definition of the model boundary. In the PPUTILS project the model boundary must be a closed polyline (it must have the same coordinate at the beginning and end). This is the same criteria used in the generation of TIN

models. As outlined above, to control mesh spacing the user must discretize (or re-sample) the model boundary to a desired node spacing before using it for mesh generation. It is assumed that the re-sampled model boundary is generated in GIS, with the output saved as an ESRI Shapefile. The boundary is then converted to the PPUTILS file format using the script `shp2csv.py`, which generates ascii files of the boundary, and its listing of nodes (required in a subsequent step).

Model boundary is a required input.

5.6.2 Internal constraint lines

Internal constraint lines in numerical mesh generation are akin to breaklines in TIN models (except they need not contain z values). Internal constraint lines are used to implicitly add features to the numerical model mesh that that must be resolved. As with model boundary, the internal constraint lines must be discretized (or re-sampled) to a desired node spacing, which ultimately control growth and spacing of the elements in the mesh. Producing a mesh that grows from say 20 m to 100 m between line A and line B can simply be achieved by placing A and B a set distance apart and re-sampling them with 20 m and 100 m spacing. There is no limit to the number of internal constraint lines to be used in mesh generation.

The user is cautioned that all constraint lines must lie within the limits of the model boundary. If constraint lines lie outside this limit, PPUTILS scripts will not be able to produce a valid mesh. Also, similar to breaklines, internal constraint lines can not intersect each other. If they do, garbage (or possibly invalid) results will be generated. If the user intends for the constraint lines to cross, a node must be manually inserted at the intersection point on both intersecting lines. By having polyline nodes at intersection points avoids topological difficulties and allows the mesh generator to produce a geometrically consistent result.

As with model boundary, it is assumed that internal constraint lines are created in a GIS environment. After saving the constraint lines as an ESRI Shapefile, the user is then required to use `shp2csv.py` script to generate a PPUTILS lines file (and its listing of nodes).

Internal constraint lines are an optional input.

5.6.3 Island or hole polygons

When the numerical model mesh has islands that are part of its domain, these features have to be manually specified. In the PPUTILS project, islands (or holes) are internal constraint lines that are closed polygons, which have to be placed in a separate file. Placing the island polygons in a separate file is required for the PPUTILS scripts that prepare input for the GMSH engine.

The user is cautioned that in constructing a TIN model (using the Triangle engine) the user is asked to specifying islands differently than specifying islands in the

numerical mesh. This difference in inputs is a direct consequence of using a different meshing engine (the PPUTILS project uses Triangle for generation of TINs and GMSH for generation of numerical model mesh). An example in the subsequent section is presented that illustrates this difference.

Islands (or holes) are an optional input.

5.6.4 Embedded nodes

Should a user wish to place mesh vertices at specific nodes, these can be manually inserted as embedded nodes. Embedding a node or nodes in a mesh ensures that the resulting mesh will generate a node at that exact location (which could be important for placing flow sources or sinks for example). The criteria to note when embedding nodes in a mesh is that i) embedded nodes must lie within the model boundary, and ii) embedded nodes can not lie on the boundary or constraint lines segment. Abiding by the above rules will ensure that a valid mesh is produced.

Embedded nodes are an optional input.

5.6.5 Master nodes file

Similar to the generation of TINs, mesh generation in PPUTILS also require a master nodes file. This file is an xyz ascii file that consists of listing of vertices from i) boundary polygon, ii) internal constraint lines (if any), iii) island (or hole) polygons (if any), and iv) embedded nodes (if any). The master nodes file is simply a merged set of all input vertices that are used to create the mesh.

When the user converts the inputs from an ESRI Shapefile to the PPUTILS file format, corresponding xyz files (with the `*_nodes.csv` in the file name) are automatically generated. To produce the master nodes file the user simply has to copy and paste the generated xyz nodes files into one. A simple text editor often suffices for this task. Alternately, a concatenate command in the terminal can also be used to merge the xyz nodes files.

In the creation of the master nodes file, any value of z suffices. This is because a digital surface in the form of a TIN will be used to assign elevations to the generated mesh at a subsequent step. An example is presented in a later section that shows how interpolation from a TIN is achieved using PPUTILS.

Master nodes file is a required input.

5.7 Generating a triangular mesh using GMSH

Let's assume the PPUTILS user has prepared the necessary skeleton geometry in a GIS package as vector geometry (i.e., polylines and/or polygons). Also assume that the boundary, the internal constrain lines and islands were re-sampled to a desired node spacing, and saved as individual ESRI Shapefiles, which were then

converted to the PPUTILS format using the `shp2csv.py` script. The list of input files for a typical mesh would include the following:

1. Master nodes file (`master_nodes.csv`)
2. Boundary file (`boundary.csv`)
3. Constraint lines file (`lines.csv`)
4. Holes file (`holes.csv`)

The above files are required to be in the PPUTILS file format (see Chapter 3). To create the GMSH steering file, invoke the `gis2gmsh.py` script as follows:

```
python3 gis2gmsh.py -n master_nodes.csv -b boundary.csv
     -l lines.csv -h holes.csv -o mesh.geo
```

where -n stands for the nodes listing file, -b for the boundary, -l for the constraint lines, -h for the holes, and -o for the GMSH steering file. If the GMSH steering file is to be generated without the -l and -g flags, the user must specify 'none' as inputs. For example, to generate a GMSH steering file without using -l and -h flags, the `gis2gmsh.py` script is invoked as:

```
python3 gis2gmsh.py -n master_nodes.csv -b boundary.csv
     -l none -h none -o mesh.geo
```

The `gis2gmsh.py` script works by constructing a GMSH steering file (`mesh.geo` in the previous example), which is then used by the GMSH mesh engine to produce a mesh. The `mesh.geo` steering file is an ascii based text file which the user can inspect before going to the GMSH program.

To generate a mesh, the user is required to launch the GMSH program, open the `mesh.geo` steering file, and generate a 2d mesh. GMSH has a number of different algorithms for generating irregular 2d triangular meshes. Before generating a mesh (or after generating an initial mesh), the user can explore the alternate 2d meshing algorithms (Adaptive, Delaunay and Frontal). The user is encouraged to generate a different mesh using each of the meshing algorithms and explore which mesh best suits the needs of the project. After the user is satisfied with the mesh, the file must be saved in the native ascii GMSH file format (using the `*.msh` extension). In the above example case, it is assumed that the user saves the generated mesh as `mesh.msh`.

To convert the generated mesh from the GMSH file format to the ADCIRC format the user applies the following script:

```
python3 gmsh2adcirc.py -i mesh.msh -o mesh.grd
```

where -i is the input mesh generated by GMSH, and -o is the converted mesh in the ADCIRC format. Note however, that the generated mesh does have dummy elevation values, which will be assigned proper values at a subsequent step.

5.8 A simple example of a numerical model mesh

To illustrate the mechanics of generating a mesh for use in numerical modeling using the PPUTILS project, consider a simple example of a box model boundary with some arbitrary islands and constraint lines. This example is not intended to be used in numerical modeling work; it's purpose is only to take the user through the required steps in producing a mesh using tools in PPUTILS. The graphical sketch of the skeleton geometry produced in GIS is shown in Figure 11a). Inspecting the input data the following observations can be made:

1. The master nodes file contains all nodes from the input files,
2. The mesh boundary is a closed polygon,
3. Node spacing varies along the boundary polygon,
4. There are three islands in the mesh,
5. Each island is a closed polygon (with a defined node spacing),
6. There are three line constraints (two open and one closed),
7. There are no embedded nodes in the mesh,

In order to generate a mesh using inputs in Figure 11a), the `gis2gmsh.py` script is invoked as follows:

```
python3 gis2gmsh.py -n master_nodes.csv -b boundary.csv
     -l lines.csv -h islands.csv -o mesh.geo
```

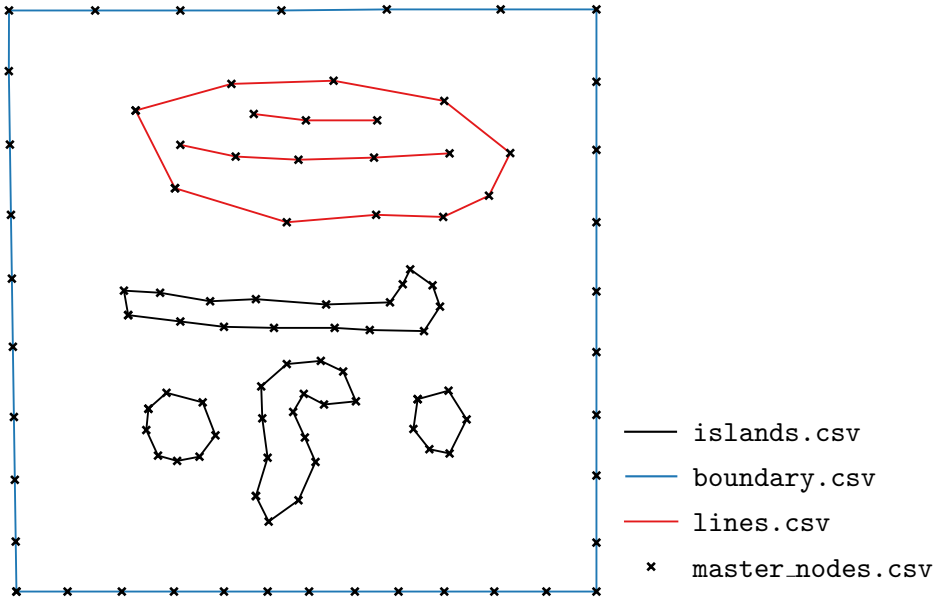
where the -n, -b, -l and -o flags are as explained above.

The script `gis2gmsh.py` produces a GMSH steering file, which is ready to be used by GMSH. To generate the mesh, the user is simply required to open the GMSH GUI, and load the steering file. Then the user generates the 2d mesh by clicking on the appropriate button in the GUI. The mesh generated by GMSH has to be saved as in the GMSH native ascii format (with the `*.msh` extension). Suppose the user saves the mesh as `mesh.msh`. This mesh is now converted to ADCIRC format by invoking the `gmsh2adcirc.py` script:

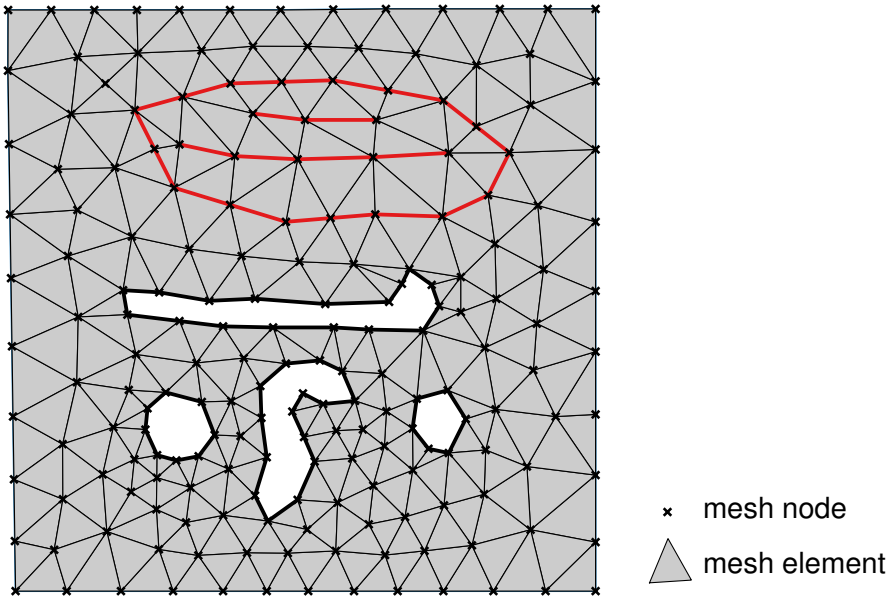
```
python3 gmsh2adcirc.py -i mesh.msh -o mesh.grd
```

where the `mesh.grd` is the mesh in ADCIRC format (used by PPUTILS).

The generated mesh in this example is shown in Figure 11b). Note that specified node spacing along the top boundary of the mesh (and along the closed constraint line) was too large in this example mesh, which forced GMSH to insert additional nodes along the boundary/lines to ensure generation of quality triangle in these areas. This constraint could be relaxed by adjusting the element size factor parameter in GMSH. It is up to the user to adjust this and other meshing parameters in GMSH.



a) Small mesh example inputs



b) Small mesh example output

Figure 11: Inputs and outputs to a simple mesh model generated by PPUTILS

For illustration purposes, let us explore what the generated mesh would look like if the constraint lines and the islands were not included. This example will also serve to illustrate three different meshing algorithms that are available for 2d triangular meshes (Adaptive, Delaunay, and Frontal).

In this instance, the `gis2gmsh.py` is invoked as follows:

```
python3 gis2gmsh.py -n boundary_nodes.csv -b boundary.csv
     -l none -h none -o mesh2.geo
```

where `mesh2.geo` is the revised GMSH steering file without lines and without island constraints. This example requires nodes listing only from the boundary polygon. Had the `master_nodes.csv` from the previous example been used instead (which also had island and constraint lines nodes) embedded nodes representing islands and constrain lines would be unnecessarily be included in the mesh. As part of this exercise, a different mesh is generated using the same skeleton geometry for each of the three meshing algorithms (Adaptive, Delaunay, Frontal) available in GMSH. The resulting meshes for the three cases are shown in Figure 12.

By inspecting the results in Figure 12 differences in the mesh generation algorithms are readily apparent. The three meshes in this example use exactly the same input geometry, with identical node spacing along the outer model boundary. The growth of the element lengths from the bottom to the top results from the node spacing of the boundary, but different mesh generating options also have an impact as well.

Another important observation from the results of Figure 12 relates to the number of nodes and elements in the generated mesh (which ultimately have an effect on computational time). Of note in this example is a feature of the GMSH's Adaptive meshing algorithm which has the effect of producing a mesh that has about 20% less nodes than the alternates, while satisfying the same geometric constraints. This finding is relevant to the modeler who is always concerned about computation time. Having a mesh that meets the constraints of the input geometry that has 20% less nodes is highly desirable.

5.9 Assigning properties to a mesh

At this point the PPUTILS user has been presented with means how to construct a digital surface for the domain of interest (i.e., a TIN), and a quality model mesh for use in numerical simulations. The next step in the tool chain is to assign properties to the numerical model mesh. In this work the properties are bottom elevations and friction coefficients as these are most often used in free surface flow modeling. In the subsections to follow, the PPUTILS user is presented with tools to assign properties to the mesh. The properties are stored as individual ADCIRC files where mesh nodes are assigned the value of the property in question.

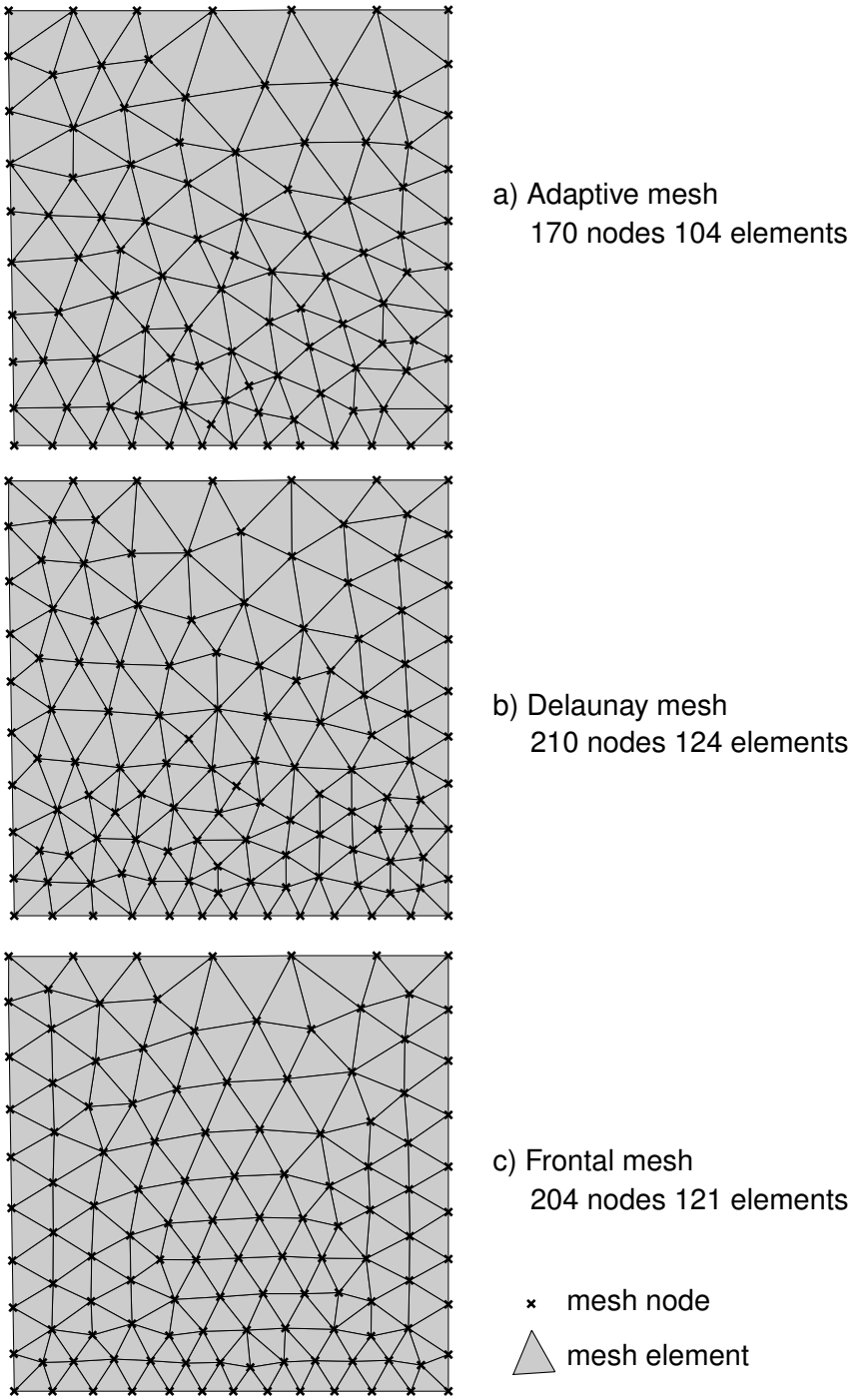


Figure 12: Small example mesh, alternate meshing algorithms

5.9.1 Bottom elevations

The best way to assign bottom elevation to a numerical model mesh is to use a digital surface (i.e., a TIN) and interpolate its values to the mesh nodes. The user is directed to Chapter 4 for more details on the construction of TINs. The PPUTILS TIN model is stored in the ADCIRC format, same as the model mesh. The task now is to assign the elevations to the mesh from the TIN model. In the PPUTILS project the process of assigning elevations from a TIN to the mesh is achieved by linearly interpolating the TIN surface using Matplotlib's Trapezoidal Map algorithm (de Berg et al., 2001). The Trapezoidal Map algorithm uses an extremely efficient way of finding a TIN element that encompasses each mesh node. Once the algorithm finds the right TIN element, a simple triangular shape function constructed from the TIN element is used to assign the mesh node an interpolated value.

Suppose the PPUTILS user has constructed a TIN model of the study area which is stored in the file `tin.grd`. Chapter 4 provides the background on how to achieve this. Also suppose that the user has constructed a mesh for use in numerical modeling that is stored in a file `mesh.grd`. Both `tin.grd` and `mesh.grd` are assumed to be ADCIRC files. To assign the bottom elevations to the mesh using a TIN, the following PPUTILS script is invoked:

```
python3 interp.py -i tin.grd -m mesh.grd -o mesh_bottom.grd
```

where -i is the input TIN, -m is the input mesh, and -o is the output mesh (in the ADCIRC format) that has its bottom elevations interpolated from the mesh.

In some cases the user may also use a dense point cloud (having comma separated xyz values) to assign elevations to the mesh instead of using a TIN. This method is recommended only when the point cloud is much denser than the resulting numerical model mesh. The interpolations in this case are carried out using Scipy's KDTree library that uses an efficient way of finding appropriates nodes in the point cloud to use for assigning elevations for the mesh. To assign bottom elevations used a dense point cloud, the following PPUTILS script is executed:

```
python3 interp_from_pts.py -p points.csv -m mesh.grd
     -o mesh_bottom2.grd -n 10
```

where -p is the xyz point cloud (comma separated), -m is the mesh to be interpolated (ADCIRC format), -o is the output mesh (also in ADCIRC format), and -n is the number of nearest neighbours to use in the point cloud search. If -n is given the value of 1, the actual nearest node in the point cloud to the mesh node in question will be used to assign its elevation value to the mesh. If -n of 10 is used for example, 10 nearest neighbours of the point cloud file will be averaged for each mesh node. The larger the number for -n, the more smoothing will result in the interpolated bottom values.

5.9.2 Friction values

The above example shows how to use a TIN and a point cloud to assign bottom elevations to an existing numerical model mesh. Using TINs and point clouds (xyz) values are appropriate when dealing with topography and/or bathymetry. When dealing with bottom friction values (or other spatially varying parameters), it is best to assign mesh values based on polygons.

The PPUTILS project has scripts that allow its users to assign properties (like bottom friction) to the mesh using boundary polygons. To achieve this the user first loads the numerical model mesh in a GIS platform as background, and then constructs an ESRI Shapefile. The ESRI Shapefile must have a field defined for the property in question (when dealing with bottom friction a field value defined as 'friction' suffices). Then, the user is required to draw boundary polygons and assign to each polygon the appropriate value in the defined field (i.e., assign a 'friction' value to each polygon). The user does not have to manually draw the friction polygons; they can easily be imported from other projects, and modified as necessary. Let us assume that the user has an ESRI Shapefile called `friction.shp` that consists of a number of polygons (each encompassing a different region of the mesh), and that each polygon has a numerical value of a field valued called 'friction'. The ESRI Shapefile file must be of type POLYGON. The script `shp2csv.py` is called first to convert the ESRI Shapefile to a PPUTILS lines file as follows:

```
python3 shp2csv.py -i friction.shp -o friction.csv
```

where -i is the input ESRI Shapefile, and -o is the resulting PPUTILS lines file that assigns each polygon a particular field value. Next, the task is to assign the field values to the nodes of the mesh. In PPUTILS, this is achieved as follows:

```
python3 assign.py -i mesh.grd -b friction.csv -o mesh_friction.grd
```

where -i is the numerical mesh (ADCIRC format), -b is the boundary polygon (PPUTILS lines format), and -o is the output mesh (ADCIRC format) that has its nodes populated with field values from the polygons.

Figure 13 shows an example where a background mesh is overlaid over three polygons that encompass part of the domain. Each polygon has a different value of the friction field, implying that mesh nodes within the polygon will be assigned the specified value. In the example in Figure 13 some mesh nodes are not encompassed by any of the polygons. Those nodes are assigned a default value of zero. To avoid this result, the user should always make sure that all nodes are included in the coverage with the polygons. Or, the user can include the first polygon that encompasses the entire domain, and assign a default value to the nodes. Then, secondary polygon should be drawn to cover portions of the numerical model domain and assign spatially varied field values (friction in this case). Or, the user can simply change in the source code the default hard coded value.

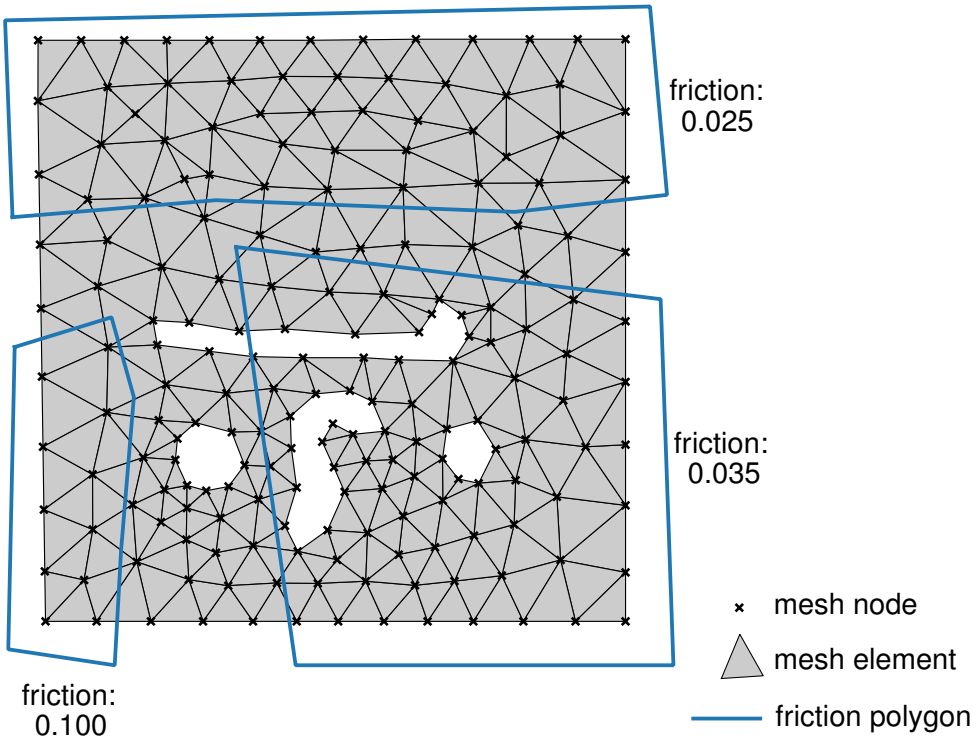


Figure 13: Small example mesh, assigning friction values

5.10 Visualizing a mesh

After generating a numerical model mesh, and after assigning bottom elevations, friction values or other field attribute, the user needs means to visualize the mesh. Visualization of the mesh can come in two forms: i) visualizing the nodes and elements (and their id numbers), and ii) visualizing the field values (i.e., bottom elevations, friction, etc.). The PPUTILS user is given a number of different means to accomplish visualization tasks using different open source GUI programs. A number of such GUIs are described below.

5.10.1 GIS

The PPUTILS user is provided with tools to visualize the generated numerical model mesh using existing GIS platforms should the user require it. The generated ADCIRC mesh file (having the `*.grd` file extension) can be converted to a number of formats recognized by any GIS platform for complete mesh visualization capability.

Mesh nodes and elements are visualized by converting the mesh to an ESRI Shapefile as follows:

```
python3 adcirc2shp.py -i mesh.grd -o mesh.shp
```

where -i is the input ADCIRC mesh and -o is the output ESRI Shapefile. Us-

ing the above script produces separate files for nodes (mesh_n.shp) and elements (mesh_e.shp). The output files include fields for nodes and element numbers, so that these attributes of a mesh can easily be visualized in a GIS platform. Being able to visualize node and element numbers of the mesh will assist the user in assigning boundary conditions to the mesh.

Similar to above, the user can also use an ascii based Well Known Text (WKT) as the format for visualizing the mesh. Similar to ESRI Shapefiles, the mesh can be converted to a WKT ascii format by invoking the following script:

```
python3 adcirc2wkt.py -i mesh.grd -o mesh.csv
```

where -i is the input mesh, and -o is the output WKT ascii text file. As with above, two different WKT files are generated –one for nodes (mesh_n.csv) and one for elements (mesh_e.csv). Identical to the ESRI Shapefile output, the WKT format also can be used to visualize node and element numbers.

In order to visualize the field values of the mesh (i.e., bottom elevations, friction values, etc.) the mesh can be converted to a regular grid that can be loaded into a GIS platform. Recall that the same tools (adcirc2asc.py, and adcirc2flt.py) were also presented in Chapter 4 to visualize TINs. To convert a mesh to a gridded ascii file, the user invokes the following:

```
python3 adcirc2asc.py -i mesh.grd -s 10 -o mesh.asc
```

where -i is the input mesh, -s is the desired grid spacing (10 m in the above example), and -o is the output ascii grid. Similarly, binary grid file can also be generated with:

```
python3 adcirc2flt.py -i mesh.grd -s 10 -o mesh.flt
```

where the parameters are identical to above.

After converting a mesh to a vector and raster based formats, the mesh (with all its attributes) can easily be visualized using GIS. For complete visualization using GIS, vector based files are loaded first (ESRI Shapefile or WKT from above) for geometry, followed by raster based files (*.asc or *.flt) for field properties.

Producing vector and raster based outputs from a TIN and a mesh, and loading the corresponding files as layers in GIS allows the modeler to compare how well the mesh represents the geometry of the TIN. This is a tool that a numerical modeler can use to evaluate whether the mesh satisfactorily captures features in the terrain model.

5.10.2 Paraview

Another useful way of visualizing the mesh is to use Paraview, a tool particularly useful for large models and their outputs. The PPUTILS project has a tool that allows both geometry and field values to be visualized using Paraview. The adcirc2vtk.py is invoked as:

```
python3 adcirc2vtk.py -i mesh.grd -o mesh.vtk
```

where -i is the input mesh (ADCIRC format) and -o is the output in Paraview's legacy ascii *.vtk format. The user is encouraged to load the *.vtk files with Paraview and inspect the mesh and its properties (such as bottom elevations, friction, etc.).

5.10.3 Meshlab

Sometimes the PPUTILS user may wish to have additional tools in modifying and editing the generated mesh. The program Meshlab has a myriad of tools for editing and fixing an existing mesh. To load the generated mesh from the ADCIRC format in Meshlab, the mesh has to be converted to Meshlab's *.ply format:

```
python3 adcirc2ply.py -i mesh.grd -o mesh.ply
```

where -i is the input mesh (in ADCIRC format), and -o is the output mesh in Meshlab's *.ply format (ascii based). The user can then launch Meshlab, and carry out appropriate mesh cleaning and editing as necessary. The resulting mesh has to be saved with Meshlab's *.ply format. To get the mesh back into the ADCIRC format, the user is required to launch the following:

```
python3 ply2adcirc.py -i mesh_modified.ply -o mesh_modified.grd
```

where -i is the modified mesh (in *.ply format) and -o is the output in the ADCIRC format.

The user is cautioned that Meshlab does not handle well double precision x and y coordinates of the mesh. To deal with this, `adcirc2ply.py` script shifts the mesh coordinates close to origin, and writes the x and y shift in the file. The `ply2adcirc.py` then uses the x and y shift to bring back the mesh to its original coordinates.

5.11 Creating a triangular mesh for numerical models

The last step in the tool chain is to take the generated mesh (in ADCIRC format), and convert it to the format used by the numerical model. In this subsection two such numerical models are covered –TELEMAC and SWAN.

5.11.1 TELEMAC

The TELEMAC numerical modeling system uses its binary SELAFIN format to store the mesh geometry. (The SELAFIN format is also used to store numerical model output.) The PPUTILS project includes a SELAFIN reader/writer class written in the Python programming language. This class is available under the ./ppmodules sub-directory of the PPUTILS project; it has been made to work with Python 2 and Python 3. The SELAFIN reader/writer class forms the foundation of Python scripting for TELEMAC using PPUTILS. The class provides users with access to all data stored in a SELAFIN file format.

The script `adcirc2sel.py` takes an ADCIRC mesh and converts it to the binary SELAFIN format recognized by TELEMAC. It is executed as follows:

```
python3 adcirc2sel.py -i mesh.grd -p single -o mesh.slf
```

where -i is the input mesh (in ADCIRC format), -p is the precision type of the SELAFIN file (single or double), and -o is the resulting SELAFIN mesh file. The `adcirc2sel.py` file also generates a `mesh.cli` file, the text based boundary file required by TELEMAC.

Suppose that the PPUTILS user has created an ADCIRC mesh file containing bottom elevations (`mesh_bottom.grd`) and also the same ADCIRC file that contains the information on friction (`mesh_friction.grd`). The two mesh files have the same structure (same x and y coordinates and same element connectivity). The two mesh files (one containing bottom elevations and one containing friction values) can be appended to create a single SELAFIN file having two variables:

```
python3 append_adcirc.py -b mesh_bottom.grd -f mesh_friction.grd
     -p single -o mesh_merged.slf
```

where the -b is the bottom elevation mesh, -f is the friction mesh, -p is the precision of the SELAFIN file (single or double), and -o is the resulting SELAFIN file containing two variables (bottom elevations and friction).

Note the `append_adcirc.py` script has been set up to take in the bottom elevation mesh and the friction mesh (in that order) and create the SELAFIN file containing elevation and friction variables in a single file. Should the user wish to create additional variables, or carry out different scripting tasks with SELAFIN files, use of the SELAFIN reader/writer class will be required.

Note that both `adcirc2sel.py` and `append_adcirc.py` produce TELEMAC's ascii boundary conditions file (i.e., the file with the *.cli extension). If the user understands precisely how boundary conditions are defined in the TELEMAC modeling system, all that is required is a text editor to produce a valid boundary conditions file for use in real life simulations. Of course, a GIS platform is needed to visualize mesh node numbering, which a user can use to construct the necessary boundary conditions file using a text editor. Other means of assigning boundary conditions for the models in the TELEMAC suite exist, and the user is encouraged to explore these as well.

5.11.2 SWAN

SWAN is a third generation spectral wave model used for propagation and generation of waves. SWAN provides its users the ability to use ADCIRC mesh files in its simulations, as well as gridded `*.asc` files (see Chapter 4). This section covers the preparation of the generated mesh using the PPUTILS project for use in SWAN simulations.

Prior to using the ADCIRC meshes generated by PPUTILS the user is required to append to the mesh additional information on the boundary conditions. At this

time the PPUTILS project does not handle ADCIRC's boundary conditions format. However, a FORTRAN script `bnd_extr.f` (written by ADCIRC's main author, Rick Luettich) is available to take a take the PPUTILS ADCIRC mesh and append to it the required boundary information. The `bnd_extr.f` script is included as part of the PPUTILS project under the ./boundary/src directory. After the `bnd_extr.f` script is executed, the resulting file is ready for simulations with SWAN. It will be up to the user to define incident wave conditions via SWAN's steering file.

6 Visualizing numerical model output

This Chapter presents deals with the topic of visualizing numerical model output. The output from the open source numerical modeling system TELEMAC is used in to illustrate how the PPUTILS project can assist the user in visualizing numerical modeling output. It is envisioned that as other environmental modeling systems become released in open source, and as they gain user base, the core PPUTILS utilities could be modified and eventually applied to these modeling systems as well. All that is required is a reader/writer class that is able to handle specific modeling outputs (usually written as binary files).

The PPUTILS project relies on existing software for visualization of modeling data; as such, no custom GUI's are provided. The PPUTILS manual is not a tutorial for these external visualization programs. The scripts in the PPUTILS project provide the user with means to convert modeling output to a number of different formats which can be read by a number of external visualization applications. It will be up to the user to learn these visualization applications on their own, and select one that best suits their needs.

There are two common ways to visualize numerical modeling output: i) using a numerical modeling output visualization package, and/or ii) using an existing GIS platform. Both ways are complementary; using each provides users the freedom and control in finding the best way of presenting outputs from simulation results. Inevitably, there will be circumstances where one will be preferred over the other. It will be up to the user to decide which visualization tool to use for the given circumstance.

It is envisioned that the user will use a numerical model output visualization package first to view the global modeling output. At this stage the user will inspect field and vector outputs for various time steps saved in the simulations. After the user is satisfied that the modeling output is valid, it is most likely required to be communicated to a wider audience. This communication is achieved by preparation of report style figures that aid in describing the phenomena under study. Given the scale of the domains in environmental free surface numerical modeling applications, a GIS platform is recommended for this task. At this stage the user is directed to using PPUTILS to extract out of the modeling output snapshots required for preparation of figures and maps for reporting purposes.

The PPUTILS project uses Paraview (Ayachit, 2017) as the open source application to view numerical modeling output on the global scale. The scripts in PPUTILS provide the user with means to convert modeling output to the Paraview format. For the preparation of report style figures, the user is recommended to use an existing GIS platform. There are many GIS platforms available in open source: QGIS (2017), GRASS (2017), SAGA (Conrad et al., 2015), GDAL (2017), and gvSIG (2015). The output produced by PPUTILS will work in any one of them (open source or otherwise). The PPUTILS project assists the user in extracting the required modeling output snapshot and using it in the GIS platform of their choice. It will then be up to users to create custom output figures to suit their projects.

6.1 Probing model output files

Upon obtaining the modeling output from a simulation, the user is recommended to probe the output file and see its meta data. The probing step is not required, but is simply a means to glance at the data before fully loading it with visualization software. In the PPUTILS project, the probing of an output file provides the user with type of the output (2d or 3d), precision of the output data (single or double precision), listing of the variable names and their units, and the time steps for which the data is saved.

Let's us assume that a result file from an existing TELEMAC simulation is named `result.slf`. The probe of this result file is done by the following script:

```
python3 probe.py -i result.slf
```

where -i is the input file in SELAFIN format. A custom class written for the PPUTILS projects is used to manage reading and writing of SELAFIN files. The class has been made to work in Python 2 and Python 3, and has been tested with 2d and 3d files having its output saved either as single or double precision.

A sample text based output from the `probe.py` script is given below (using an existing 2d flow hydrodynamic solution of a river reach):

```
################################
The input file being probed: results.slf
Precision: single
File type: 2d

################################
Variables in result.slf are:
--------------------------------
     v     variable        unit
--------------------------------
     0 --> VELOCITY U       [M/S]
     1 --> VELOCITY V       [M/S]
     2 --> WATER DEPTH      [M]
     3 --> FREE SURFACE     [M]
     4 --> BOTTOM           [M]
     5 --> SCALAR FLOWRATE  [M2/S]
     6 --> COURANT NUMBER   []
     7 --> FRICTION VEL.    [M/S]

################################
     t        time (s)
--------------------------------
     0 -->       0.0
     1 -->    3600.0
     2 -->    7200.0
```

```
     3 -->   10800.0
          ......
     168-->  604800.0
################################
```

After getting the output from the `probe.py` script, the user is made aware that the `result.slf` file above holds 8 variables (indexed from 0 to 7), each with having data for 169 time steps (indexed from 0 to 168). The time step output in the file is 3600 s (or 1 hr).

6.2 Visualizing output with Paraview

The easiest way of visualizing the output from SELAFIN files is via Paraview, an industry standard package used for numerical modeling output visualization. It allows its users with multitude ways of visualizing field and vector variables (in 2d and 3d).

The user is made aware that there is a way for the Paraview program to native read SELAFIN files. The drawback to this method is that it requires the user to compile Paraview from scratch (a not so simple of an undertaking). The PPUTILS project takes a simpler approach. A script named `sel2vtk.py` is available that takes a SELAFIN result file and converts it to a legacy Paraview file in the *.vtk format. The script does not rely on external libraries, other than Python's Numpy, Matplotlib, and Scipy.

It is acknowledged that converting binary (TELEMAC's SELAFIN) to ascii (Paraview's *.vtk) formats can be inefficient for large data sets. This amounts to converting TELEMAC's modeling output from binary to ascii format. A future version of PPUTILS will include a script that will convert TELEMAC's SELAFIN to Paraview's binary *.vtk format. For the time being, converting a SELAFIN result file to Paraview's ascii *.vtk format in PPUTILS is achieved as follows:

```
python3 sel2vtk.py -i result.slf -o result.vtk
```

where -i is the SELAFIN result file, and -o is the output in Paraview's *.vtk format. The `sel2vtk.py` uses Python to write output for each variable, for each time step, to an individual *.vtk file. In Paraview's *.vtk format, one individual file is required for each time step of the output. Should the PPUTILS user wish to convert a subset of the time steps from the SELAFIN result file, the following should be executed:

```
python3 sel2vtk.py -i result.slf -o result.vtk -t_start 23 -t_end 35
```

where -i and -o flags are same as before, and -t_start and -t_end are the starting and ending time step indices (see output from `probe.py` script). In the above example, Paraview output is written for time step indices from 23 to 35. Using -t_start and -t_end flags is a way to limit the number of individual *.vtk files that are written in the conversion. A sample output created in Paraview is shown in Figure 14.

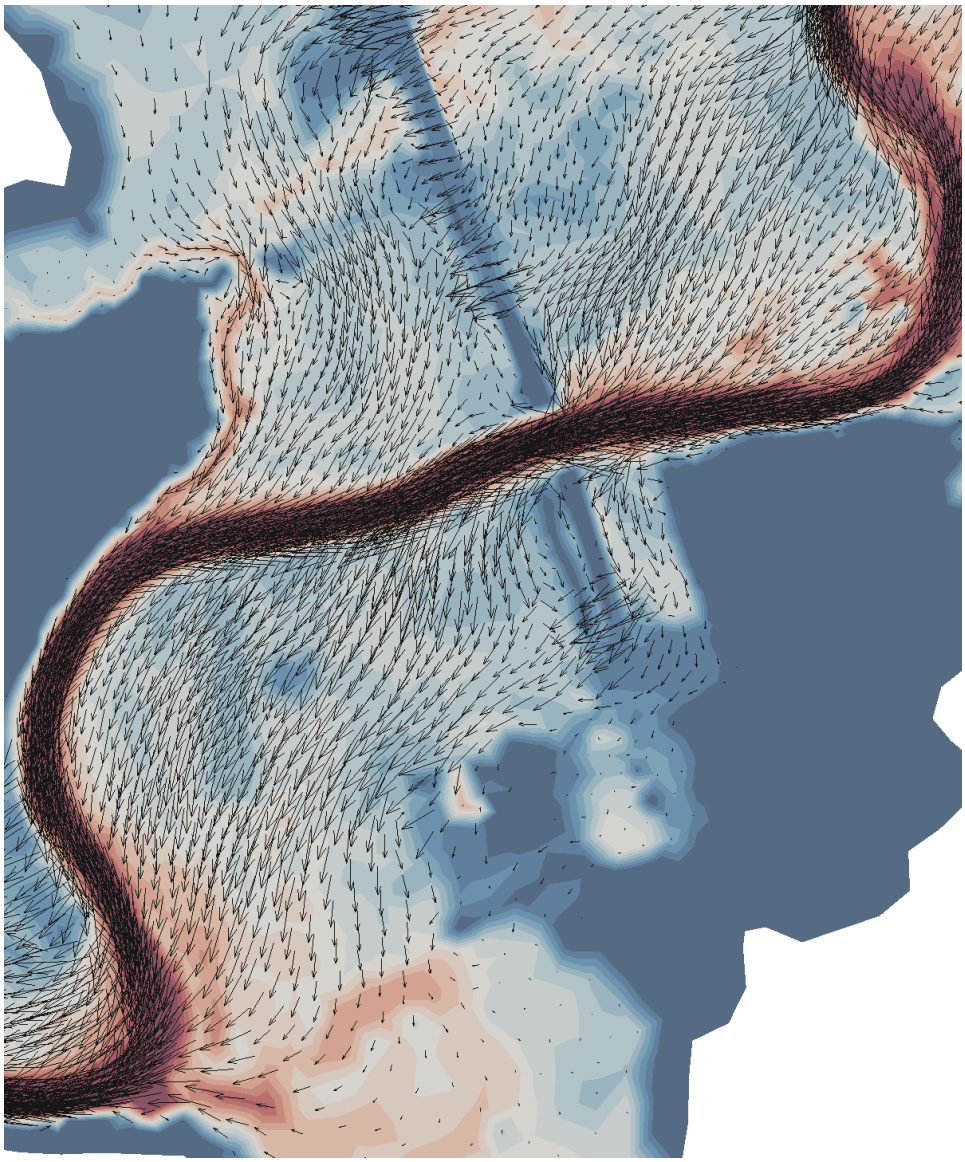


Figure 14: Sample numerical output created in Paraview

An important feature of the `sel2vtk.py` script is the automatic construction of vector variables for typical vector quantities used in the TELEMAC modeling system. For example, the `sel2vtk.py` searches among the available variables for flow velocity variables typical in TELEMAC-2D and TELEMAC-3D result files, as well as TOMAWAC's and ARTEMIS's wave direction variables. Presently, the restrictions are the following: TOMAWAC's `MEAN DIRECTION` variable has to be defined clockwise from geographic north, while ARTEMIS's `WAVE INCIDENCE` variable has to be defined as counter clockwise with respect to the Cartesian x-axis. These are the default values in TOMAWAC and ARTEMIS, respectively. The TELEMAC's default direction conventions have been hard coded into the `sel2vtk.py` script. In the case different conventions are used in the output, the original source code can easily be modified to accommodate the adopted convention.

6.3 Visualizing output with GIS

Once the output SELAFIN result file has been obtained from the simulations, the PPUTILS user is provided with a number of tools to visualize its output using an existing GIS platform. The PPUTILS user is cautioned that visualizing modeling output via existing GIS packages is never meant to replace visualization using Paraview. Paraview's main purpose is to assist users in visualizing modeling output; it does this efficiently, and extremely well. Visualization of model output using GIS via PPUTILS is only meant to take individual model snapshots and make them available in GIS for the production of publication quality figures. The user is still required to use an external tool of their choosing and decide which snapshot to display using GIS.

6.3.1 Field output

To visualize field output (a model output variable approximated over a continuous region) using GIS, the user is required to convert model output to a format recognized by the GIS platform. Field variables are best displayed by a GIS platform if they are converted to a raster format. Raster formats are ascii or binary gridded data files representing a particular quantity (like topography, water depth, velocity magnitude, wave height, etc.). The PPUTILS user is given a tool to convert a snapshot of the model output to a raster file.

To convert model output to an ascii based gridded file, the `sel2asc.py` script is called as follows:

```
python3 sel2asc.py -i result.slf -v 4 -t 0 -s 10 -o result_bottom.asc
```

where -i is the SELAFIN result file, -v is the index of the variable to extract, -t is the index of the time step to extract, -s is the grid spacing (in meters), and -o is the gridded raster file in *.asc file. Indices of the -v and -t flags are obtained by running the `probe.py` script first.

Similarly, to create a binary gridded raster file, the `sel2flt.py` script is called:

```
python3 sel2flt.py -i result.slf -v 4 -t 0 -s 10 -o result_bottom.flt
```

where -i, -v, -t, -s, and -o flags are same as above, with the only exception that the output raster is defined as a binary *.flt file. The binary rasters tend to be smaller in size than their ascii counterparts, but are not easily visualized by a person (i.e., the user can not open the file in a text editor, and inspect the output).

The above scripts work by reading the mesh structure from the input SELAFIN file, and then creating a grid with a defined spacing (-s flag). The scripts then use Matplotlib's Trapezoidal Map algorithm to interpolate the intended field variable (-v flag), for the intended time step (-t flag), and create the output raster file (-o flag). The output raster can then be loaded in the user's GIS platform of choice, and used for preparation of publication quality figures and maps.

It is recommended that the user starts with a large grid spacing parameter first (-s flag), and then refine the spacing as required. By doing so, the user will be in a position to decide on the compromise between file size and grid resolution of the raster output.

6.3.2 Vector output

Vector output is defined as output that is associated with variables that have magnitude and direction (such as flow velocity), or sometimes with direction only (such as direction of wave propagation). Vector output is typically displayed with arrows, with magnitude that control the length of the arrows, and direction controlling their orientation.

To the best knowledge of the author, existing GIS platforms presently have limited abilities to display vector based output. As with raster based output displayed in GIS, the PPUTILS user is required to extract from the SELAFIN file a snapshot of the output from a specified time step. The script `extract.py`, described below, allows the user to extract a comma separated file (having column based output) for the specified time step.

Let us assume that the user has successfully executed `extract.py` script from a SELAFIN file that has at least one vector variable. The extracted point cloud data file (where each node holds a different field value corresponding to the file's output variables) can then loaded with GIS and displayed as a vector quantity using vectors. The user has to ensure that appropriate variables are used for magnitude and directions. Note that each open source GIS platform has slightly different terminology and different means to achieve this task. The user is required to learn how to accomplish this with their GIS platform of choice. An example of such an output produced by the QGIS platform is provided in Figure 15.

6.4 Miscellaneous utilities for extracting modeling output

There will be times when the numerical modeler will require to extract out of the result file simple text based output as part of a given task (a calibration exercise

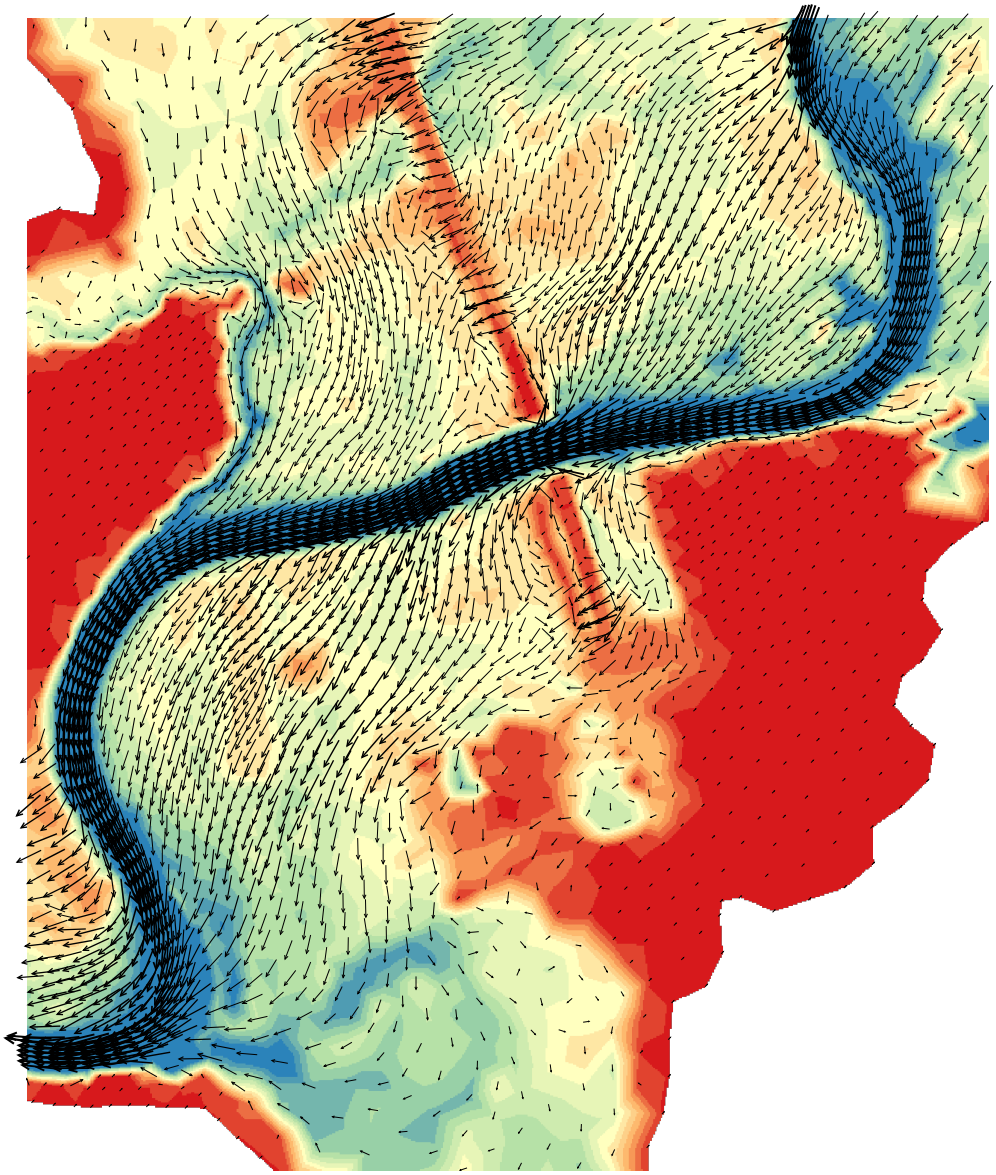


Figure 15: Sample numerical output created in QGIS

for example). There are a number of miscellaneous utilities in the PPUTILS project that can assist the user in this task. Some are described below:

6.4.1 Extracting output time series at a point

After carrying out simulations that have large number of time steps, sometimes the user may wish to extract time series output at a single node point and visualize the time series. This is particularly important for large SELAFIN result files that could take a long time to load with visualizing software. To extract time series at a single point from the result SELAFIN file, the user would do the following:

```
python3 extract_pt.py -i result.slf -x 300 -y 150 -o node_ts.csv
```

where -i is the simulation result file in SELAFIN format, -x and -y are the coordinates of the extraction point, and -o is the output ascii text file. The `extract_pt.py` script uses Scipy's KDTree algorithm to select the node in the result file that is closest to the input coordinate given with -x and -y parameters. The output file is produced as a comma separated ascii file for all variables, for all time steps in the input SELAFIN file. Time is written on the first column of the file, followed by results for each variable. The `extract_pt.py` script works for extracting output at a point from 2d and 3d files. Sample output is shown below.

```
TIME   VELOCITY U  VELOCITY V  WATER DEPTH  FREE SURFACE  COURANT NO
0       0.000      0.000       3.321        176.200       0.000
3600   -0.436      0.939       3.539        176.419       0.123
7200   -0.457      0.982       3.517        176.397       0.129
10800  -0.462      0.993       3.515        176.394       0.130
14400  -0.463      0.995       3.514        176.394       0.131
18000  -0.462      0.994       3.515        176.394       0.131
21600  -0.463      0.995       3.514        176.394       0.131
25200  -0.463      0.995       3.515        176.394       0.131
28800  -0.463      0.995       3.514        176.394       0.131
```

The output ascii file can then be used by an external plotting program to visualize the results, or be simply viewed in the text editor.

6.4.2 Extracting model output at snapshots

The PPUTILS user is given a tool to extract comma separated ascii text output for an identified time snapshot from the SELAFIN output file. The `extract.py` script will extract values from all variables, for all nodes, for an identified time step from the input SELAFIN result file. The script is called as follows:

```
python3 extract.py -i result.slf -t 81 -o result_81.csv
```

where -i is the SELAFIN result file, -t is the index of the time step to extract (see output from `probe.py` script for values), and -o is the comma separated ascii file that includes coordinates and all output variables as columns).

The snapshot from the SELAFIN result file may be loaded in a GIS platform as a point cloud file, and thus visualize the values of individual (or multiple) variables. The `extract.py` script works for 2d and 3d SELAFIN files, its use is not recommended for 3d output files (as all output for each node, for each variable and for each plane will be written as a text file). Viewing 3d result files at snapshots is better achieved using Paraview and/or the POSTEL-3D module of the TELEMAC modeling system.

6.4.3 Extracting output cross sections and profiles

The PPUTILS user may be required to extract cross section and profile data from an existing simulation run of a 2d result file. (Extracting cross sections or profiles from 3d result files is best achieved using TELEMAC's POSTEL-3D module and/or Paraview). The extraction of longitudinal profiles are particularly relevant in 2d river hydraulic studies, where the user investigates how water surface elevations change through out the modeling domain. Extraction of results at individual transects is also relevant in coastal studies, when, for example, the user wishes to plot the transformation of a wave as it moves thorough the harbour structures. At other times the user may wish to see the variation of a particular variable (like wave height) as it varies across the structure that is to be designed. Being able to extract output from cross sections and profiles, and to have it readily available as an ascii text file, it believed to be valuable.

In the PPUTILS project, the user is required to construct a polyline for the locations where the output is to be extracted. In the same manner as when extracting cross sections and/or profiles from TIN models, the PPUTILS user is required to re-sample each polyline with an appropriate number of nodes (as the interpolation mechanism works by assigning values to each node on the polyline). The user is thus required to produce a PPUTILS lines file (appropriately re-sampled) before using it for extraction of model output.

There are two different scripts in PPUTILS that allow the user to extract model results along a polyline. The script `extract_line_t.py` is a script that is executed as follows:

```
python3 extract_line_t.py -i result.slf -t 81 -l profile.csv
     -o profile_results.csv
```

where -i is the SELAFIN results file, -t is the time step index (see output of `probe.py` for values) of the extracted data, -l is the input polyline (re-sampled with enough points to make the output meaningful), and -o is the extracted output along the polyline for the specified time step, for all variables in the file. Each variable is written as a separate column in the ascii comma separated text file. In the example output (shown below), only three such variables are shown.

```
id   x        y       sta     VELOCITY U   VELOCITY V  DEPTH
0    77.176  131.195  0.000  -0.397       -0.887       3.708
0    78.954  130.279  2.000  -0.405       -0.901       3.922
```

```
0    80.733  129.364  4.000  -0.413       -0.915       4.136
0    82.511  128.449  6.000  -0.421       -0.929       4.351
0    84.289  127.533  8.000  -0.429       -0.943       4.565
1    86.067  126.618  0.000  -0.437       -0.957       4.780
1    87.846  125.703  2.000  -0.445       -0.971       4.994
1    89.624  124.788  4.000  -0.453       -0.985       5.209
1    91.402  123.872  6.000  -0.461       -0.999       5.423
1    93.181  122.957  8.000  -0.469       -1.013       5.638
```

Suppose that the user wishes to extract results for all time steps, for a particular variable along an input polyline. Such result extraction is achieved by executing the following:

```
python3 extract_line_v.py -i result.slf -v 2 -l profile.csv
     -o profile_results_depth.csv
```

where -i is the SELAFIN results file, -v is the index of the variable to extract (see output of `probe.py` for mapping of variables to numeric indexes), -l is the input polyline (re-sampled appropriately), and -o is the extracted output along the polyline for the specified variable, for all time steps in the file. Each time step is written (for the variable indexed at 2) as a separate column in the ascii comma separated file (see below).

```
id   x        y        sta    0        3600     7200     10800
0    77.176  131.195   0.000  176.200  176.424  176.403  176.406
0    78.954  130.279   2.000  176.200  176.424  176.403  176.407
0    80.733  129.364   4.000  176.200  176.425  176.404  176.408
0    82.511  128.449   6.000  176.200  176.425  176.404  176.406
0    84.289  127.533   8.000  176.200  176.426  176.404  176.405
0    86.067  126.618  10.000  176.200  176.426  176.405  176.406
0    87.846  125.703  12.000  176.200  176.426  176.405  176.408
0    89.624  124.788  14.000  176.200  176.427  176.405  176.408
0    91.402  123.872  16.000  176.200  176.427  176.406  176.408
0    93.181  122.957  18.000  176.200  176.427  176.406  176.408
```

The above scripts that extract output along a polyline work whether the user has a single or multiple polylines in the input file. The extracted output can be easily read by a text editor or a spreadsheet program, or otherwise used by the user as they see fit (i.e., automated by a plotting script using Matplotlib, Gnuplot, etc.).

6.4.4 Converting from SELAFIN to ADCIRC

Let us suppose that a user receives an existing SELAFIN result files (generated by others), and wishes to extract its mesh, and visualize it with tools provided in the previous chapter. There is a script in PPUTILS that extracts from a SELAFIN file an ADCIRC file for a particular variable, for a particular time step. Suppose that the `result.slf` is provided to the user. If we assume that the user probes the file provided, which reveals the meta data presented in the previous section. Should

the user wish to extract the mesh geometry from the file, the script `sel2adcirc.py` would be called:

```
python3 sel2adcirc.py -i result.slf -v 4 -t 0 -o result_mesh.grd
```

where -i is the SELAFIN output file, -v is the index of the variable to be extracted, -t is the index of the time step to extract, and -o is the output mesh (in ADCIRC) format. The indexes for -v and -t are obtained from running the `probe.py` script (see above for example). When the output is in ADCIRC format, it can be visualized with any one of the tools mentioned in Chapter 4 and Chapter 5.

7 Putting it all together - St. Clair River model

The purpose of this Chapter is to provide the user a practical example of how tools in the PPUTILS project can be used to complete a real life modeling project from start to finish using the TELEMAC modeling system. The focus of the Chapter is on the construction of the model mesh, and visualization of its simulation results. Recall that the scope of this manual covers only the front- and back-end tasks associated with numerical modeling (pre- and post-processing). Developing steering files and producing simulation results is left up to the user. The user is directed to TELEMAC's vast set of validation cases where many such steering files are provided, for all models in its suite. The user should study the validation cases, and select (and appropriately modify) cases that most closely resemble their problems and study areas.

As part of this example the following topics are covered:

1. Defining the study boundary,
2. Obtaining river bathymetry,
3. Constructing a digital surface model from bathymetric data,
4. Developing a numerical model mesh,
5. Visualizing i) digital surface mesh and ii) numerical model mesh,
6. Extracting ascii output at nodes, snapshots, and cross sections, and
7. Displaying model output via visualization software and GIS platforms.

7.1 Background

The example case selected for demonstration of the PPUTILS project is the upper portion of the St. Clair River, located at the outlet of Lake Huron (one of the five North American Great Lakes). The St. Clair River forms the international border between Canadian Province of Ontario and the US State of Michigan. The same reach of the St. Clair River has been studied by the International Joint Commission (IJC, 2009), who was tasked to investigate issues related to water levels in the upper Great Lakes. During the course of the IJC (2009) study a number of different scientific reports were produced on the subjects of hydrology, river hydraulics, sediment transport, morphology, etc. There was even a TELEMAC-2D model that was developed to answer certain study questions. Most importantly, the background data collected during the technical studies were made publicly available. The publicly available data are used in this demonstration example.

7.2 St. Clair River study area

The study area in this example includes the upper portion of the St. Clair River, extending from the outlet of Lake Huron to approximately 7.5 km downstream. The flow of the river is southward, from Lake Huron to Lake St. Clair. The St. Clair River

eventually discharges into Lake St. Clair, approximately 65 km from the southern end of Lake Huron. The river drops in slope approximately 2 m from Lake Huron to Lake St. Clair, is between 250 m and 800 m wide, more than 10 m deep, with an average discharge of 5,200 m^3/s. The upper reach of the St. Clair River has a strong hydraulic gradient, and influences (and is influenced by) the water levels levels on Lake Huron. Canadian city of Sarnia, and the US city of Port Huron share the upper portion of the river. Bluewater Bridge crosses the narrowest part of the St. Clair River, and provides a link between Canada and the US.

7.2.1 Bathymetry

As part of the IJC (2009) study multi-beam bathymetry was collected of the upper part of the St. Clair River. The bathymetry was collected by the US Army Corps of Engineers, Detroit District. The hydrographic survey system included a Reson 8125 multi-beam at 455 kHz with a .5 degree beam width, a velocity profiler, and a GPS/POS MV positioning and heave, pitch and roll compensation.

The bathymetric data set made available includes a single text file, approximately 500 MB in size, having a total of 13 million individual points. Shown below is a small sample set of the original bathymetric data.

```
'X'          'Y'          'Z'   'DATUM' 'ELEVATION'
4151328.30   126368.43     9.12   174.56   165.44
4151328.36   126380.99     9.79   174.56   164.77
4151328.38   126387.28    10.37   174.56   164.19
4151328.45   126486.34     8.27   174.56   166.29
4151328.69   126456.40    10.71   174.56   163.85
4151328.83   126487.82     8.14   174.56   166.42
4151329.09   126464.16    10.56   174.56   164.00
4151329.15   126476.72     9.40   174.56   165.16
4151329.47   126465.63    10.53   174.56   164.03
4151329.66   126509.62     4.57   174.56   169.99
...
```

The meta data provided in the data set noted that X and Y coordinates are in NAD 83 Michigan Plane South (meters). Depths, datum and elevations are provided in meters. The elevations are referenced to the IGLD1985 vertical datum, with depths below datum referenced to the St. Clair River step down planes.

7.2.2 Shoreline outline

For the purpose of this demonstration project, shoreline outline traced from existing satellite and/or aerial mapping is deemed sufficient. Ontario's Ministry of Natural Resources and Forestry (MNRF) provides RAW LAS data set for its ortho-rectification projects, at 0.3 m resolution. In this demonstration project, the RAW LAS data was converted to a black and white ortho-rectified image, which was then

used to trace out the shoreline within the study area. The reconstructed ortho-rectified image was in the NAD83(CSRS) horizontal reference system, which is commonly used in Canada.

Alternatively, open source GIS applications have plugins that allow users to automatically load satellite and/or aerial maps available from various online databases. In this demonstration example the shoreline was traced using ortho-rectified aerial imagery, but it could have been traced out using satellite or aerial imagery available from online databases. The shoreline tracing was done using QGIS, and saved as an ESRI Shapefile with the file name `shoreline_poly.shp`. The shoreline polygon was then converted to the PPUTILS line format using the `shp2csv.py` script as follows:

```
python3 shp2csv.py -i shoreline_poly.shp -o shoreline_poly.csv
```

The shoreline outline used in this example is shown in Figure 16.

7.3 Development of a TIN model

Before developing a numerical model mesh for use in hydraulic modeling simulations, a properly resolved digital surface is required. To build a TIN model of the study area, the bathymetry was processed to retain only the `X, Y, ELEVATION` columns. A simple Python script was written to read the original bathymetry data line by line, and retain only the desired columns of data. Following the column cropping, the coordinate system of the data was converted from NAD83 Michigan Plane South to NAD83(CSRS). After coordinate conversion (and after removing the header text), the file was saved as `bathymetry.csv`.

Next, the bathymetry data was cropped using the model boundary polygon as some bathymetric points of the original data set lie outside the present study area. GIS packages have methods to crop point clouds using polygon boundaries. In this example project a point cloud cropping script part of the PPUTILS project was used instead. Using the script avoids waiting for GIS to carry out the required tasks for such a large data set (especially when progress during execution is not provided to the user). To crop the bathymetry to the model boundary polygon, the `crop_pts.py` script was used:

```
python3 crop_pts.py -i bathymetry.csv -p shoreline_poly.csv
     -o bathymetry_cr.csv
```

where -i is the original bathymetry (converted to the NAD(CSRS) coordinate system), -p is the traced out boundary polygon, and -o is the cropped bathymetric data set. The cropped bathymetry file consisted with 1.7 million points (the original data set included a larger area and consisted of 13 million points). All inputs in the `crop_pts.py` script have to be in the PPUTILS file format. Similarly, the PPUTILS project also has a `crop_lns.py` script that crops a set of lines to a boundary of a polygon. This script was not used in this project, but may be useful when cropping large sets of breaklines.

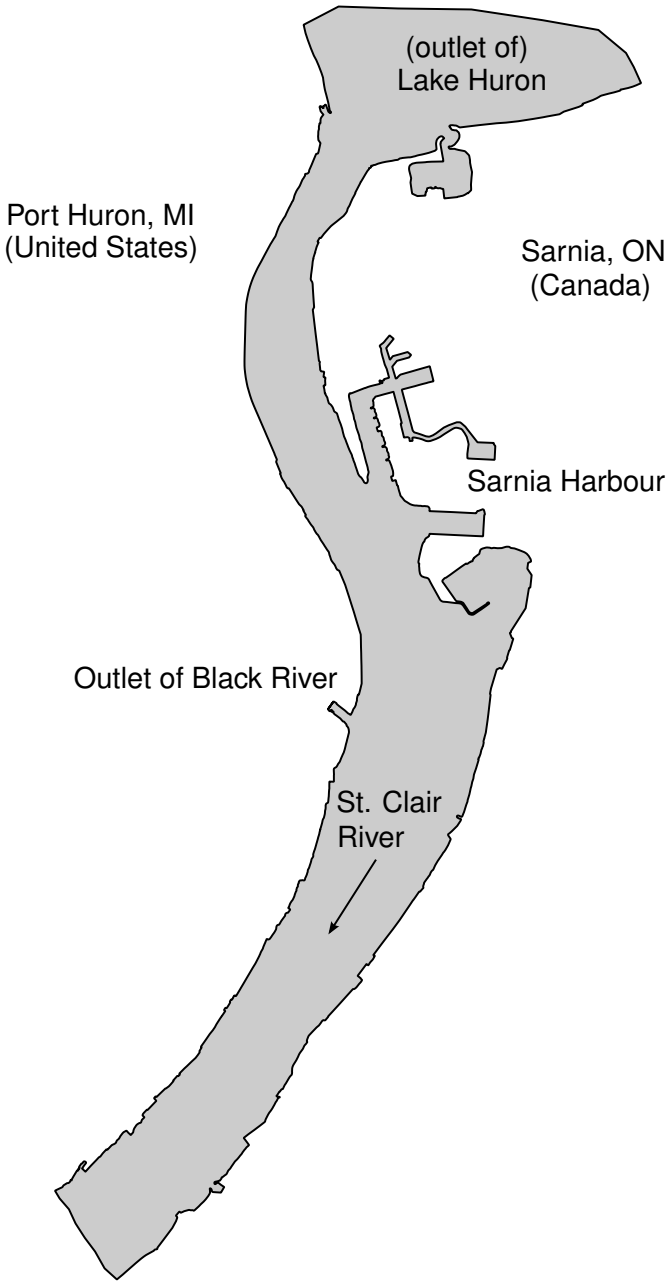


Figure 16: St. Clair River model boundary

To produce a TIN model that has reasonable values near the model boundaries, some manual insertion of bathymetry is typically required. In the present case, the multi-beam echo sounder did not approach locations too close to shore; in these instances, a number of manually inserted bathymetric data points were stored in a file called `artificial_bathymetry.csv`. As an aside, the methodology of manually inserting bathymetric points to a given data set was a point of great debate during the IJC (2009) study. A number of different methodologies were used to to manually insert bathymetry in order to capture river's bathymetry close to shore. In the IJC (2009) study it was discovered that different methods of inserting bathymetry near the shoreline were responsible for producing different results in numerical modeling. To the best knowledge of the author, the differences in the methodologies were not resolved in the IJC (2009) study. Regardless, the user will have to explore and find appropriate ways of inserting bathymetry adjacent to shorelines.

After manual insertion of the bathymetric points, the shoreline boundary polygon was reduced (to trim the lake portion of the domain), and re-sampled to a 20 m node spacing. This task was performed in QGIS, and the final TIN model boundary saved as `shoreline_poly_20m.shp`, indicating the node spacing in the polygon file is 20 m. The re-sampled TIN model boundary was then converted to the PPUTILS lines format using `shp2csv.py`, which produced `shoreline_poly_20m.csv` file. The nodes of the re-sampled boundary polygon do not have elevation values (or have a dummy values assigned during the tracing exercise). For a proper TIN model the boundary polygon elevations were interpolated using the `bathymetry_cr.csv` data as follows:

```
python3 interpBreakline_from_pts.py -p bathymetry_cr.csv
     -l shoreline_poly_20m.csv -o shoreline_poly_20m_interp.csv -n 1
```

where -p is the point cloud file, -l is the input line file (re-sampled shoreline boundary in this case), -o is the interpolated output PPUTILS line file, and -n is the number of nearest neighbours to average in proximity of the closest node. The script works by creating a Scipy KDTree object from the input bathymetry, and then assigns to each node in the input lines file (re-sampled shoreline in our case) an elevation value. The -n parameter is the number of nearest neighbour nodes (one in our case) from which to average over. The above script automatically creates a *_nodes.csv file (in our case, `shoreline_poly_20m_interp_nodes.csv`).

The `master_nodes.csv` file was created by merging the following *.csv files:

- Cropped bathymetry file: `bathymetry_cr.csv`,
- Manually inserted bathymetry file: `artificial_bathymetry.csv`, and
- Re-sampled shoreline nodes: `shoreline_poly_20m_interp_nodes.csv`.

After obtaining the merged xyz point cloud, the TIN digital surface was created as follows:

```
python3 gis2tin.py -n master_nodes.csv -b shoreline_poly_20m.csv
     -l none -h none -o tin.grd
```

where -n is the master (i.e., merged) nodes file, -b is the re-sampled shoreline boundary, and -o is the resulting TIN model, all in the PPUTILS file format (see Chapter 3).

7.3.1 Visualizing the TIN model

There are two ways to visualize the TIN model using PPUTILS, using: i) Paraview (model output visualization package), and ii) GIS (by converting the TIN to a gridded raster file to visualize bathymetry). Each are covered next:

To create the Paraview file from the TIN, the user executes the following:

```
python3 adcirc2vtk.py -i tin.grd -o tin.vtk
```

where -i is the input TIN model (in ADCIRC format), and -o is the Paraview *.vtk file. The user is encouraged to load the `tin.vtk` file in Paraview, and explore its visualization features (i.e., extract cross sections and profiles, change bathymetric colour coding, check TIN for general consistency, etc.). The St. Clair River TIN model, as visualized in Paraview is shown in Figure 17.

In order to create a gridded raster file that can be loaded in a GIS platform, the user can execute the following:

```
python3 adcirc2asc.py -i tin.grd -s 10 -o tin.asc
```

where -i is the input TIN model, -s is the desired grid spacing (10 m in the above example), and -o is the output raster (in ESRI *.asc file format). Sister script `adcirc2flt.py` works in the same way, except rather than producing an ascii based gridded file, it produces a binary gridded file (ESRI *.flt file format). The gridded file is loaded in GIS, where the user can perform various tasks provided by the GIS application of their choice (such as extraction of cross sections and profiles, computation of volumes, overlaying with aerial images, etc.). The St. Clair River TIN model, as visualized in QGIS is shown in Figure 18.

After the user is satisfied that the generated TIN is appropriate for use in the modeling project, the task of generation of a numerical model mesh can start. This is described next.

7.4 Development of a quality model mesh

The mesh for the St. Clair River model used in this example is for demonstration purposes, but it will be used to generate hydrodynamic model simulations using the TELEMAC-2D numerical model. In the example explicit features are not built in the mesh (i.e., re-sampled internal constraint lines are not deemed necessary). In other words, internal mesh constraint lines are not considered necessary to capture the hydrodynamics of the St. Clair River in this example.

For the purposes of this demonstration example, the quality model mesh will be produced using the GMSH mesh generator, using mesh node spacing of 50 m on the Canadian, and 100 m on the US shoreline. In order to accomplish this task the

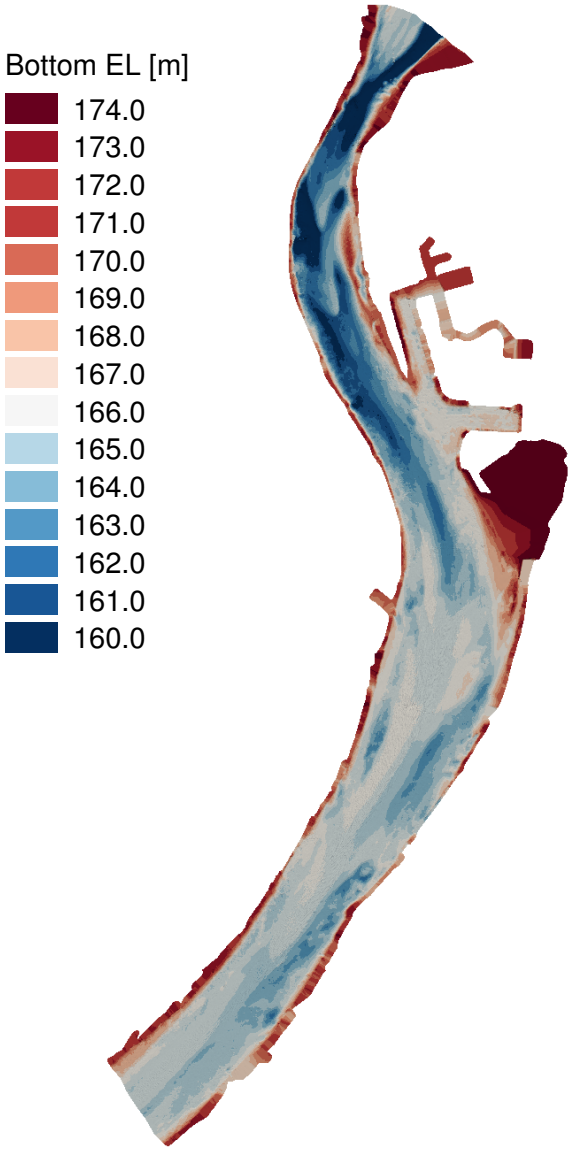


Figure 17: St. Clair River TIN model visualized in Paraview

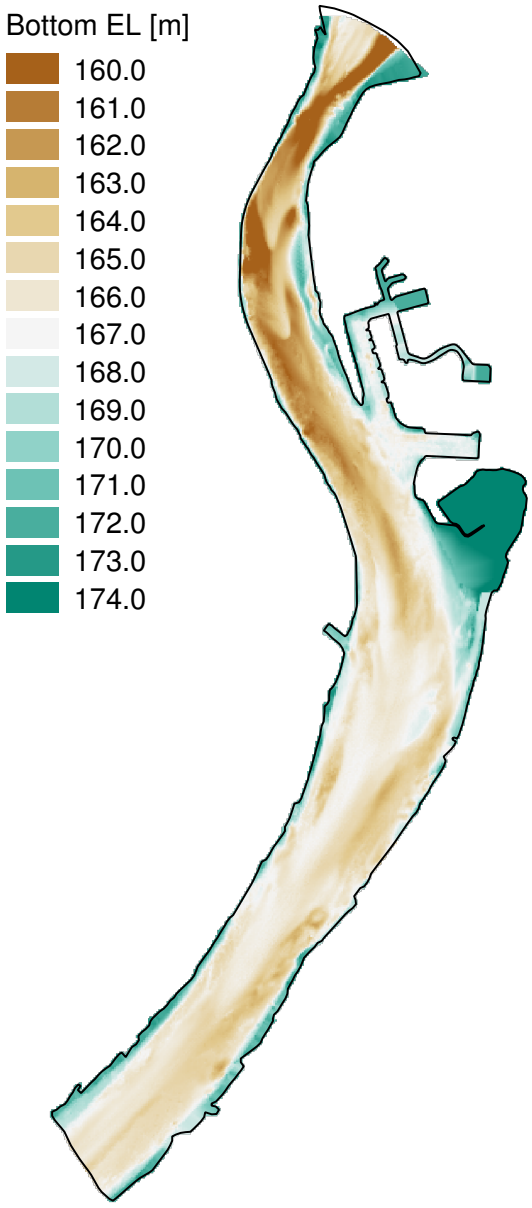


Figure 18: St. Clair River TIN model visualized in QGIS

original traced shoreline (in the ESRI Shapefile format) is loaded into the GIS platform. The original polygon is split into a polyline. The polyline is then broken into Canadian and US portions, and re-sampled to the 50 m and 100 m interval, respectively. The up- and downstream boundary segments of the river are re-sampled to a 50 m node spacing. The re-sampled polylines of the various segments are then merged to a single polygon, and nodes spacing manually adjusted (where required). The numerical mesh outline is then saved as `model_outline.shp`, and converted to the PPUTILS format using `shp2csv.py` script:

```
python3 shp2csv.py -i model_outline.shp -o model_outline.csv
```

where -i is the model outline (in ESRI Shapefile format), and -o is the model outline (in PPUTILS ascii text format). The above script automatically produces the nodes file (in this case `model_outline_nodes.csv`) for use in the generation of the mesh. There are no islands in the study boundary, so island polygons (re-sampled to an appropriate spacing) are not needed.

To generate the GMSH steering file, the following script is required to be executed:

```
python3 gis2gmsh.py -n model_outline_nodes.csv -b model_outline.csv
     -l none -h none -o mesh.geo
```

where -n is the master nodes file (in this case the nodes of the model outline polygon), -b is the boundary file containing the model outline, -l is the constraint lines file (none in this case), -h is the hole file (none in this case), and -o is the GMSH steering file produced by PPUTILS. After generating the GMSH steering file, the user is then required to launch GMSH, and load the `mesh.geo` steering file. Once in the GMSH program, it will be up to the user to select the type of the mesh (Adaptive, Delaunay, or Frontal), and other mesh generation parameters (if desired). Please note that the PPUTILS manual is not a tutorial for GMSH, or its many options. It will be up to the user to figure out how to use the GMSH program. The last step in the process is to generate the mesh, and save the file in GMSH's file format. In this example, it is assumed that the file is saved as `mesh.msh`.

To get the GMSH output to the ADCIRC format, executing the following script is required:

```
python3 gmsh2adcirc.py -i mesh.msh -o mesh.grd
```

where -i is the input mesh (in GMSH format), and -o is the output mesh (in ADCIRC format). To visualize the generated mesh in GIS for example, a conversion to either WKT format (via `adcirc2wkt.py`) or ESRI Shapefile format (via `adcirc2shp.py`) is required. The mesh, as visualized in QGIS, is shown in Figure 19. Note that the generated mesh file has no elevation values assigned. Assigning elevations to the nodes of the mesh is discussed next.

Figure 19: St. Clair River mesh visualized in QGIS

7.4.1 Assigning bathymetry to mesh

Having developed a TIN digital surface model of the study area (which encapsulates the model mesh entirely) it is a fairly straightforward task to assign elevations to the mesh nodes from the TIN. Assigning elevations to the generated mesh is accomplished as follows:

```
python3 interp.py -t tin.grd -m mesh.grd -o mesh_bathy.grd
```

where -t in the TIN digital surface model, -m is the numerical model mesh, and -o is the numerical model mesh with its nodes interpolated from the TIN model. All of the inputs in the above script are in the ADCIRC mesh format (used by the PPUTILS project).

After assigning elevation nodes to the mesh, a check is recommended to verify how well the interpolated mesh resolves the model bathymetry of the TIN. This check is typically performed by comparing cross sections and profiles between the TIN and the mesh. This step can be accomplished two ways: i) converting the mesh to Paraview's *.vtk format, and ii) converting the mesh to a raster for viewing with GIS. For the example mesh in this demonstration project, this check is shown in Figure 20, where the TIN and the interpolated mesh are shown in GIS on a colour coded plot showing bottom elevations. Mesh spacing that varies between 50 m and 100 m may be too course to capture all of the details of relevant river hydraulics in this example. Using the above mesh spacing the large sand deposit at the re-circulation zone north of Sarnia Harbour is not captured. Should the user require a mesh for real life numerical simulations to pick up the above re-circulation feature, mesh spacing would have to be reduced. Reducing mesh spacing will bring about a closer match between bathymetry in the TIN and the interpolated bathymetry of the mesh.

7.4.2 Assigning friction zones to mesh

After the user develops the mesh (and successfully assigns bottom elevations) the next step in the process in river hydraulic studies is to assign spacially variable friction to the mesh. In the PPUTILS project this task is completed by constructing in a GIS platform a polygon for different friction zones in the modeling domain, and saving the file as an ESRI Shapefile. The TELEMAC-2D model that was developed as part of the IJC (2009) study provided spacially variable friction zones that were calibrated to water levels from a number of different gauges on the Canadian and US shorelines. The previously developed friction map is used in this demonstration project. A single ESRI Shapefile containing a number of polygons was created in a GIS package, where each polygon was assigned a single value for friction. The polygon friction file was saved as `friction.shp`, and then converted to `friction.csv` using `shp2csv.py` script. The `friction.csv` file is a PPUTILS lines file with the following columns of data: id,x,y,friction, where each shape id has its own friction value. Figure 21 shows the spacially variable bottom friction used in this example project.

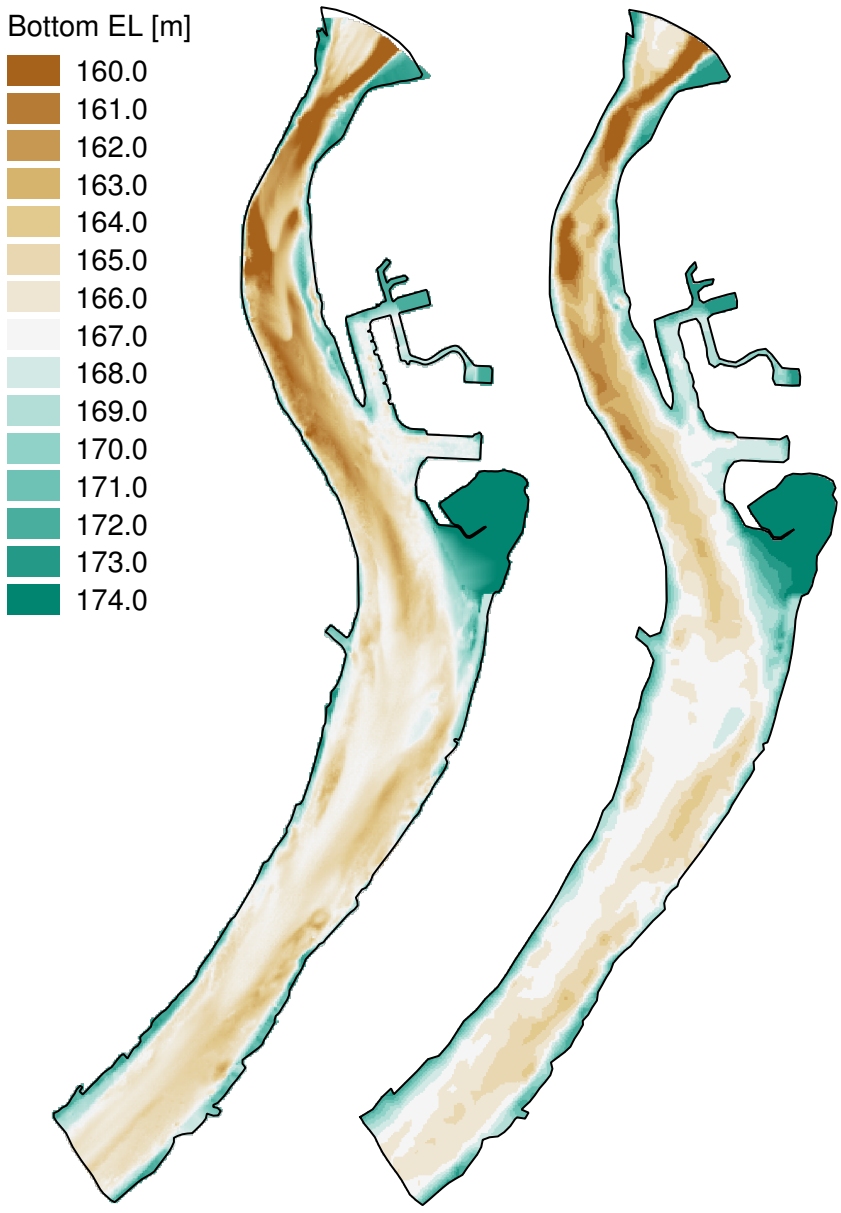


Figure 20: Comparison between St. Clair River TIN and interpolated mesh

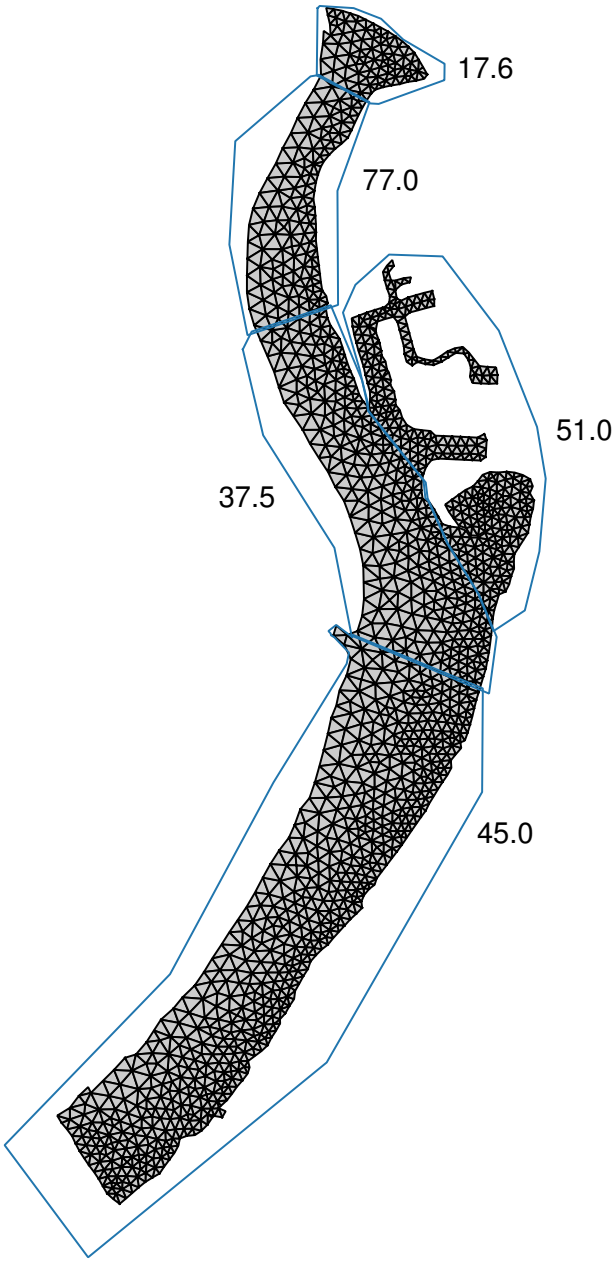


Figure 21: St. Clair River bottom roughness values (Strickler's coefficient)

To assign the friction to the generated mesh, the PPUTILS user is required to produce an ADCIRC mesh file that (instead of bottom elevations) contains friction values for each node. Essentially, the `assign.py` script searches the nodes that are inside the defined polygon, and assigns to those nodes the field values (in this case friction) specified in the file. The `assign.py` is executed as follows:

```
python3 assign.py -i mesh.grd -b friction.csv -o mesh_friction.grd
```

where -i is the numerical model mesh previously generated, -b is the friction boundary file where each polygon has assigned a friction value, and -o is the output ADCIRC file that has friction values instead of elevation values. The user is encouraged to inspect the `mesh_bathy.grd` and `mesh_friction.grd` in a text editor to see the difference. Of course, any one of the mesh visualization tools can be used to verify that the intended values of friction are assigned to different regions in the mesh.

7.4.3 Converting mesh for TELEMAC modeling

The PPUTILS project uses a custom SELAFIN reader/writer class written in the Python programming language to perform the basic I/O tasks with SELAFIN files. Most simple way of converting an ADCIRC mesh to the SELAFIN format is done by calling the `adcirc2sel.py` script:

```
python3 adcirc2sel.py -i mesh_bathy.grd -p single -o mesh_bathy.slf
```

where -i is the input ADCIRC file, -p is the precision of the SELAFIN file (single or double), and -o is the converted SELAFIN file. However, as the St. Clair River model example has spacially variable friction, the `append_adcirc.py` script is used that produces a SELAFIN file with `BOTTOM` and `BOTTOM FRICTION` variables used in the TELEMAC modeling system. The `append_adcirc.py` script is called as follows:

```
python3 append_adcirc.py -b mesh_bathy.grd -f mesh_friction.grd
     -p single -o mesh.slf
```

where -b is the bottom elevation ADCIRC mesh, -f is the friction ADCIRC mesh, -p is the precision of the SELAFIN file (single or double), and -o is the resulting SELAFIN file holding both `BOTTOM` and `BOTTOM FRICTION` variables. To verify the generated SELAFIN mesh file has both variable, the user should launch the `probe.py` script, and inspect the meta data output. The `mesh.slf` is is the file used in the numerical model simulations in this example. Note that the boundary conditions text file `mesh.cli` is also generated by the above script, which can be used by the user in preparing the simulations. Given that the user understands how boundary conditions are set in the TELEMAC modeling system, all that is required is to open the `mesh.cli` file in a text editor, and change the default fields to desired values.

7.5 Visualizing simulation results

It is assumed that the PPUTILS user is in the position to take the mesh geometry and boundary condition files generated above and prepare valid TELEMAC simu-

lations. It is assumed in this section that `result2d.slf` file has been generated using TELEMAC-2D numerical model using the above developed mesh. Different ways of visualizing the output files created in the St. Clair River example model are presented in this section. Before visualizing the output, it is useful to probe the generated result file and look at the basic meta data of the output. This is achieved as follows:

```
python3 probe.py -i result2d.slf
```

where -i is the result file. The output of the `probe.py` script is shown below.

```
################################
The input file being probed: result2d.slf
Precision: single
File type: 2d

################################
Variables in result2d.slf are:
--------------------------------
     v     variable        unit
--------------------------------
     0 --> VELOCITY U       [M/S]
     1 --> VELOCITY V       [M/S]
     2 --> WATER DEPTH      [M]
     3 --> FREE SURFACE     [M]
     4 --> BOTTOM           [M]
     5 --> COURANT NUMBER   []

################################
     t        time (s)
     --------------------------------
     0 -->       0.0
     1 -->    3600.0
     2 -->    7200.0
     3 -->   10800.0
              ...
     8-->    28800.0
################################
```

The above meta data output provides the user with a quick glance of what is actually stored in the TELEMAC-2D result file. By inspecting the meta data it is apparent that the output file holds 6 variables (indexed 0 to 5), with a total of 9 time steps (indexed 0 to 8). The time step of the output is 3600 sec or 1 hr.

7.5.1 Model output via Paraview

The Paraview application, having been developed specifically for the purpose of visualizing model output, is used for viewing model output of the St. Clair River

example model. Whether the simulations are produced in 2D or 3D variants of TELEMAC, the conversion to Paraview's *.vtk format is the same (this example only uses 2d results):

```
python3 sel2vtk.py -i result2d.slf -o result2d.vtk
```

where -i is the TELEMAC's result file, and -o is the Paraview output. Once the results are loaded into Paraview, the user can visualize the following: i) mesh geometry and bathymetry, ii) geometry using an exaggeration in the vertical, iii) colour coded output of various model output variables, iv) magnitude and direction vectors for velocity variable, and a variety of others. The user can also i) draw cross sections directly in Paraview, and obtain plots on an adjacent view port, ii) probe any one location within the model domain, and obtain numeric values of the output, iii) select a particular node and obtain a time series plot of the output, for any (or all) of its variables. A particularly useful feature of Paraview is its ability to step through the simulated time, and obtain animated outputs of the selected variable(s) in its model space. The animated outputs can even be saved a movie file, to be used in project presentations. A sample output of the colour coded velocity magnitude and its vectors is shown in Figure 22. As suspected, the re-circulation zone north of Sarnia Harbour is not well resolved with the mesh used. If this feature is of interest to the modeler, the mesh will have to refined and simulations repeated.

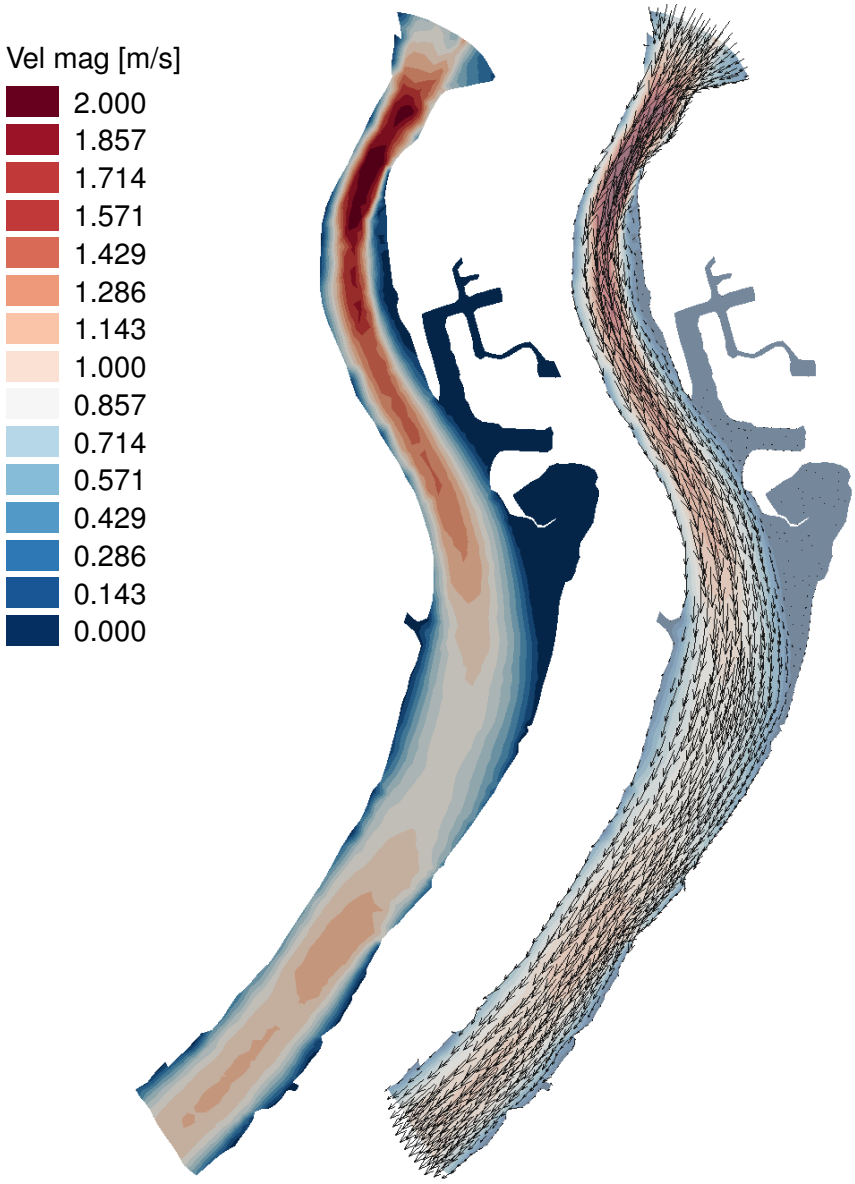
7.5.2 Model outputs using a GIS platform

In the PPUTILS project it is envisioned that when the TELEMAC model output is converted to an output for viewing via a GIS platform, the user does so for the purposes of producing a report style figure from the model output. Given such large model extents (the upper reach of the St. Clair River model is 7.5 km long) viewing both model inputs and outputs via GIS is also desired. A very useful feature is to load aerial or satellite imagery into a GIS platform, and then superimpose model input (geometry) and/or model output (results produced by a simulation).

Suppose the PPUTILS user wishes to bring in to a GIS platform the model results at specified snapshots. In the PPUTILS project Paraview is recommended as the first application to view the global model output, and decide on which snapshot would be required to export to a GIS platform. In the St. Clair River hydraulic model, the required snapsnot is at the last time step of the results file, indexed at time 8 (see the output of `probe.py` script).

Suppose that the output variable to display in GIS is the velocity magnitude. (The same procedure outlined below is applied for any other variable.) The next task is to extract flow velocities (TELEMAC-2D variables names `VELOCITY U` and `VELOCITY V`, representing x- and y- components of the velocity), overlaid them over the mesh geometry. First, the velocity components (for time step 8) are extracted from the results file (`results2d.slf`) as follows:

```
python3 sel2asc.py -i results2d.slf -v 0 -t 8 -s 10 -o velu.asc
python3 sel2asc.py -i results2d.slf -v 1 -t 8 -s 10 -o velv.asc
```



a) Velocity magnitude b) Velocity vectors

Figure 22: St. Clair River model results visualized in Paraview

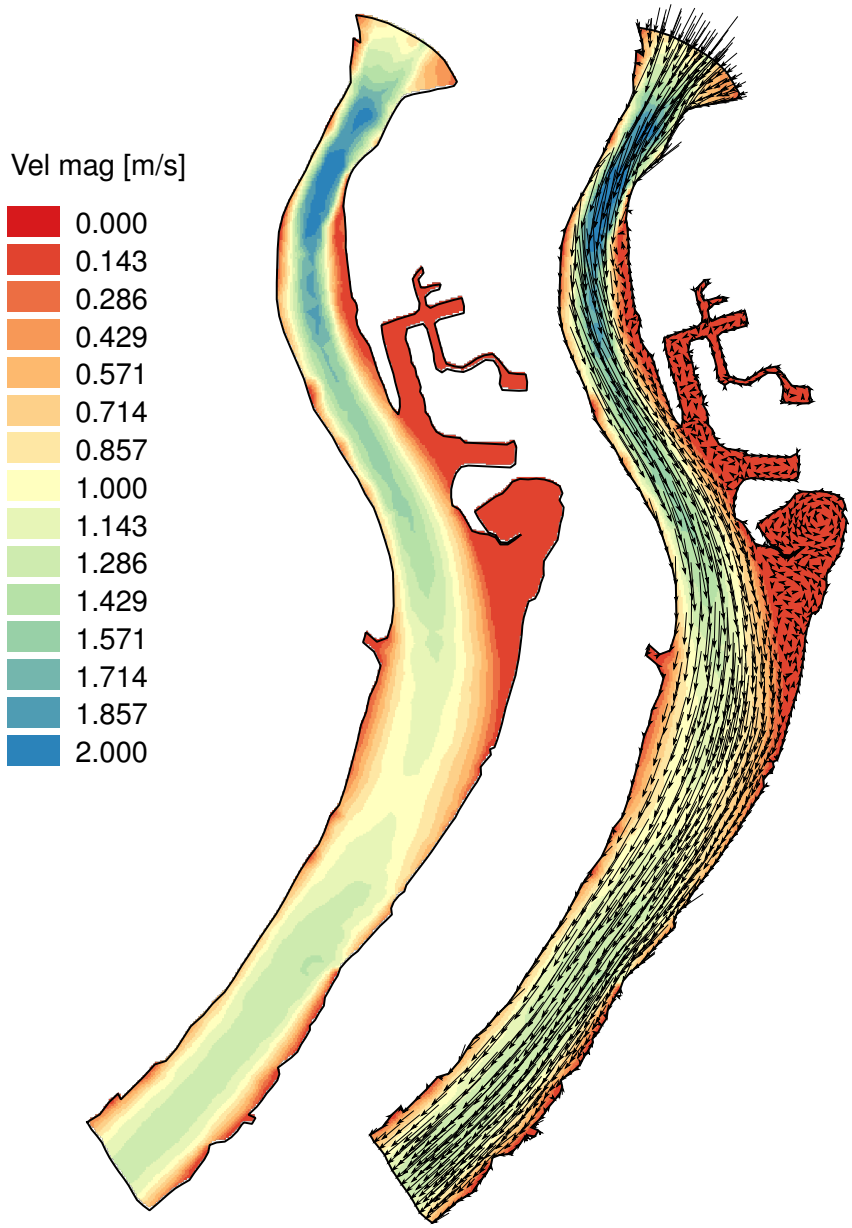
where -i is the TELEMAC-2D results file, -v is the variable to extract (see output from `probe.py`), -t is the time step index, -s is the grid spacing (in meters), and -o is the gridded output file. Rather than displaying components of the velocity in GIS, a velocity magnitude variable is often preferred. To generate a raster of the velocity magnitude from the outputs above, a raster calculator is required to compute the velocity magnitude for every grid cell. GIS platforms have such calculators readily available. The plot of the velocity magnitude, as generated in a QGIS, is shown in Figure 23a).

It is also assumed that velocity vectors at mesh nodes are required to be displayed in GIS. To achieve this the `extract.py` script is called to write to a text file the nodal coordinates and all variables in the results file:

```
python3 extract.py -i results2d.slf -t 8 -o results2d_5.csv
```

where -i is the SELAFIN results file, -t is the time step index, and -o is the comma separated ascii text file of the output for the specified time step. A plot in Figure 23b) shows the velocity magnitude, overlaid with velocity vectors (re-constructed in QGIS using the `results2d_5.csv` file).

A variety of scripts from the PPUTILS project can be used to extract ascii based output from the `result2d.slf` file. For example, the user can select a coordinate in the mesh, and use `extract_pt.py` script to extract an ascii file and view in a text editor temporal evolution of all model variables at that node. The user can also construct a single (or multiple) cross sections, re-sample that cross section, and use either `extract_line_t.py` (to get a snapshot of all variables at a specified time step), or `extract_line_v.py` (to get temporal variation for a single variable for all nodes on the line). The combinations of the tools used will depend on the user preference, and/or project requirements.



a) Velocity magnitude b) Velocity vectors

Figure 23: St. Clair River model results visualized in QGIS

8 Closure

The purpose of this manual is to illustrate to the user how the PPUTILS scripts can be used to complete real life numerical modeling projects from start to finish using only public domain and open source software. The primary focus of the PPUTILS project is in constructing geometry files for terrain, free surface flow, and wave modeling applications. The secondary focus is to provide users with tools that allow visualization of numerical model output in a number of different ways. To this end, the PPUTILS project accomplishes these objectives.

There are many scripts that are part of the PPUTILS project that are not described in this manual; and there will be many more that will be added to the collection as time goes on. It is my hope to expand this manual (and keep it up to date) with the latest developments as they become available.

References

Ayachit, U. (2017). *The ParaView Guide Community Edition: A Parallel Visualization Application*. Kitware.

Conrad, O., Bechtel, B., Bock, M., Dietrich, H., Fischer, E., Gerlitz, L., Wehberg, J., Wichmann, V., and Böhner, J. (2015). "System for Automated Geoscientific Analyses (SAGA) v. 2.1.4." *Geoscientific Model Development*, 8, 1991–2007.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2001). *Computational Geometry, Algorithms and Applications*. Springer Publishing, Berlin, Germany.

GDAL (2017). *Geospatial Data Abstraction Library (GDAL): Version 2.0.0, Open Source Geospatial Foundation*. http://gdal.osgeo.org.

Geuzaine, C. and Remacle, J. (2009). "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities." *International Journal for Numerical Methods in Engineering*, 79(11), 1309–1331.

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

gvSIG (2015). *gvSIG User Guide, gvSIG Association*. http://www.gvsig.com.

IJC (2009). *International Upper Great Lakes Study, Impacts on upper Great Lakes water levels: St. Clair River*. Report prepared by the International Joint Commission, Ottawa, Canada and Washington, DC.

Lawhead, J. (2013). *Learning Geospatial Analysis with Python*. Packt Publishing, Bermingham, UK.

Li, Z., Zhu, Q., and Gold, C. (2015). *Digital Terrain Modeling: Principles and Methodology*. CRC Press, Boca Raton, Florida.

QGIS (2017). *QGIS Geographic Information System, Open Source Geospatial Foundation Project*. http://qgis.osgeo.org.

Shewchuk, J. R. (1996). "Triangle: Engineering a 2d quality mesh generator and delaunay triangulator." *Applied Computational Geometry: Towards Geometric Engineering*, M. Lin and D. Manocha, eds., Springer-Verlag, Berlin, 266–290.

Colophon

This manual was completed entirely on a Debian 8.2 GNU/Linux system. Document processing system used was pdfTeX, version 3.14159265-2.6-1.40.15 (TeX Live 2015 distribution). BibTeX, version 0.99d was used to manage references. Page size set is 8.5 x 11 in; text font used is Helvetica, 11 pt. The text of the document was written using GNU Emacs, version 24.4.1. Figures were generated using Paraview, version 4.1.0 and QGIS, version 2.14.1, then brought to Inkscape, version 0.48.5 for final editing and exporting. pdfTeX was used to render text of the illustrations, thus ensuring consistency with the main document.