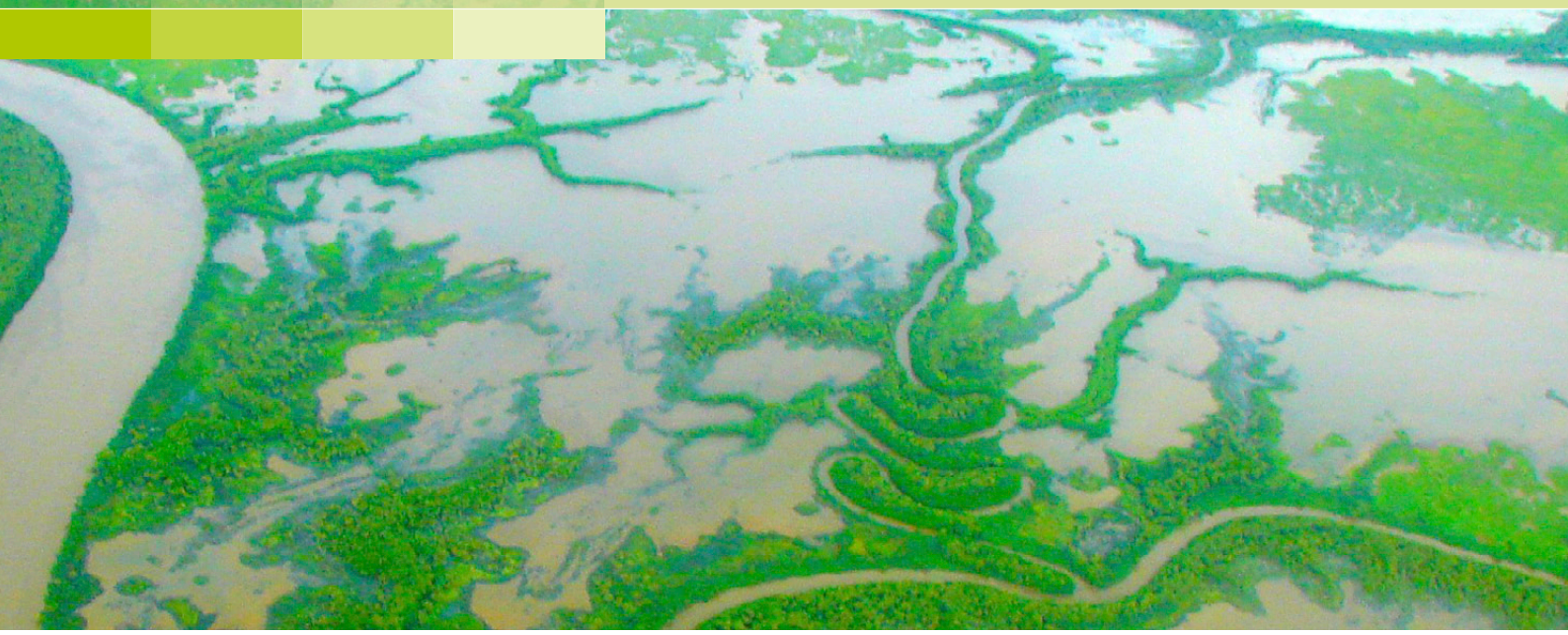
1D/2D/3D modelling suite for integral water solutions

D-Flow Flexible Mesh



Hydrodynamics

Deltares

Enabling Delta Life

User Manual

D-Flow Flexible Mesh

User Manual

Arthur van Dam
Herman Kernkamp
Sander van der Pijl
Wim van Balen

Version: 1.1.124
SVN Revision: 52614

April 15, 2019

D-Flow Flexible Mesh, User Manual

Published and printed by:
Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone: +31 88 335 82 73
fax: +31 88 335 85 82
e-mail: info@deltares.nl
www: https://www.deltares.nl

For sales contact:
telephone: +31 88 335 81 88
fax: +31 88 335 81 11
e-mail: software@deltares.nl
www: https://www.deltares.nl/software

For support contact:
telephone: +31 88 335 81 00
fax: +31 88 335 81 11
e-mail: software.support@deltares.nl
www: https://www.deltares.nl/software

Contents

List of Figures

List of Tables

!! See new UM on: http://content.oss.deltares.nl/delft3d/manuals/D-Flow_FM_User_Manual_new.pdf !!

1 Introduction to D-Flow Flexible Mesh

D-Flow Flexible Mesh (D-Flow FM, formerly Unstruc) is a 1D-2D-3D hydrodynamical simulation package that runs on flexible meshes. 'Flexible' here means the familiar curvilinear meshes from Delft3D combined with triangles, pentagons, etc. and 1D channel networks, all in one single mesh.

1.1 Current state

D-Flow FM is still in active development, which means that functionality is rapidly changing. The program is currently intended for the entire pipeline mesh generation–model setup–computation. Preprocessing in other packages is upcoming: the next release of RGFGRID will have full flexible mesh support. Postprocessing can be done in Delft3D-QUICKPLOT or directly in MATLAB (output is in NetCDF). The current version of D-Flow FM (1.1.124) roughly contains the following functionality:

Mesh:

- ◇ Curvilinear mesh generation based on splines
- ◇ Curvilinear mesh manipulation (add/delete/move/orthogonalise/smoothen/...)
- ◇ Triangular mesh generation
- ◇ Basic node/link manipulation (add/delete/move/...)
- ◇ Local refinement (halve cell sizes) within user-defined polygon.
- ◇ Refined aspect ratio driven by scalar sample values.
- ◇ Adaptive refinement based on bathymetry.
- ◇ Cutcell formulation on land boundaries (polygons).

Hydrodynamics:

- ◇ 2Dh-formulation.
- ◇ Bathymetry (on corners/centers/u-points), exact or by interpolation (cf. QUICKIN).
- ◇ Thin dams and thin dykes.
- ◇ Bottom friction: a.o. Chézy, Manning (space-varying roughness values possible).
- ◇ Wind forcing (uniform and/or location- and time-dependent).
- ◇ Boundary conditions: water levels (tidal components or time series), discharge, tangential/normal velocity. salinity (tracer).
- ◇ Initial waterlevels and salinity (in subdomains).
- ◇ Various numerical and physical enhancements: Smagorinsky&Elder horizontal turbulence, 2D conveyance, Piaczek-treatment for advection (allows for higher Courant numbers), faster matrix solver.
- ◇ 3D-modelling has started: with σ layers. Proof-of-concept: flexible (z and σ mixed).
- ◇ k-ϵ turbulence model.

Structures

- ◇ Weir
- ◇ Pump
- ◇ Controllable dam
- ◇ Controllable gate

1D-specifics

- ◇ 1D conveyance formulation (cross sectional profiles (pipe, rectangle, yz) in 1D channels à la SOBEK).

Monitoring:

- ◇ Observation stations (waterlevel + velocity timeseries)
- ◇ Moving observation stations (waterlevel + velocity timeseries)
- ◇ Cross sections (discharge, area, etc. timeseries + cumulative)

I/O:

- ◇ NetCDF readers/writers + basic plotting in MATLAB (in OpenEarth Toolbox).

Water quality

- ◇ Offline coupling with D-Water Quality.
- ◇ 2D Unstructured models (also 1D), non-aggregated.

1.2 Developments in the near future

(in no particular order)

- ◇ Parallel version for distributed memory clusters.
- ◇ Controllers and triggers (through new RTC-Tools).
- ◇ 1D network underneath 2D surface waters
- ◇ 3D temperature model

1.3 Basic mesh concepts

In D-Flow FM, meshes (sometimes denoted as 'networks') consist of netcells and are described by:

- ◇ netnodes: corners of a cell (triangles, quadrangles, ...),
- ◇ netlinks: edges of a cell, connecting netnodes,
- ◇ flownodes: the cell circumcentre, in case of triangles the exact intersection of the three perpendicular bisectors and hence also the centre of the circumscribing circle,
- ◇ flowlinks: a line segment connecting two flownodes.

This mesh topology is illustrated in Figure 1.1.

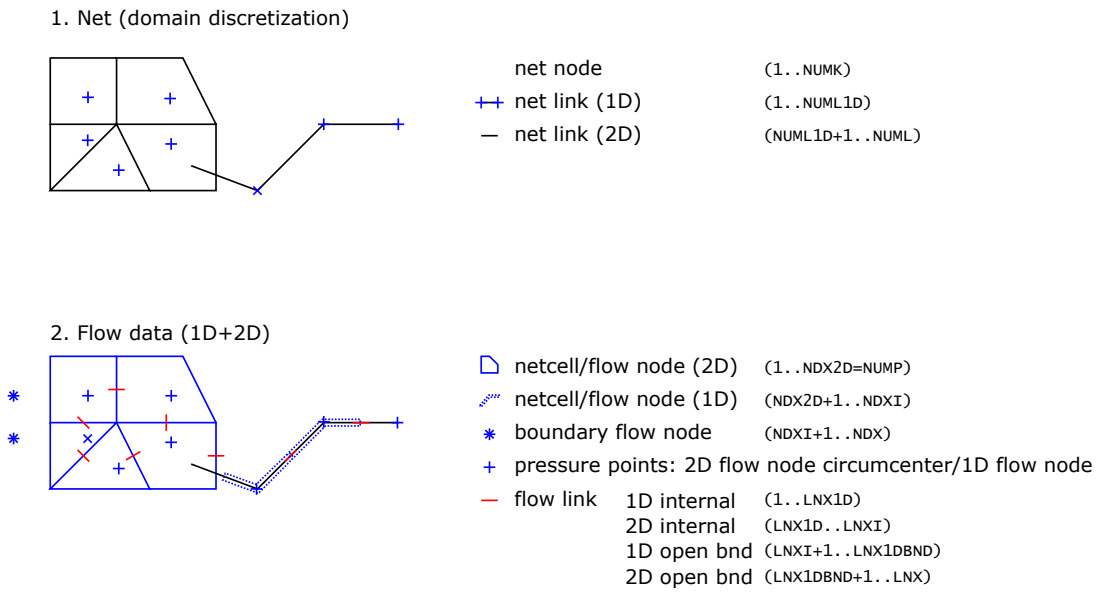


Figure 1.1: *Flexible mesh topology*

Important properties of the mesh are the *orthogonality* and *smoothness*. The *orthogonality* is defined as the cosine of the angle φ between a flowlink and a netlink. Ideally 0, angle $\varphi = 90^{\circ}$. The *smoothness* of a mesh is defined as the ratio of the areas of two adjacent cells. Ideally 1, the areas of the cells are equal to each other. A nearly ideal setup is shown in Figure 1.2.

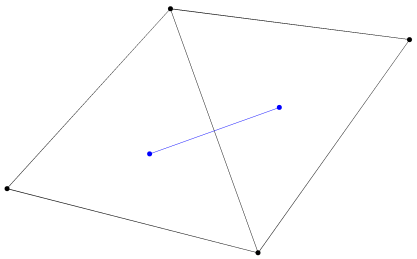
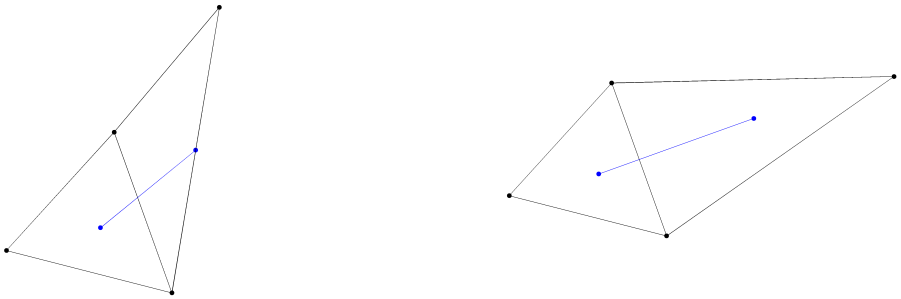


Figure 1.2: *Perfect orthogonality and nearly perfect smoothness along the edge connecting two triangles. Black lines/dots are network links/nodes, blue lines/dots are flow links/nodes.*

It is rather facile to generate meshes that violate the orthogonality and smoothness requirements. In Figure 1.3, two different setups of two gridcells are shown with different mesh properties.

The left picture of Figure 1.3 shows how orthogonality can be detoriated by skewing the right triangle with respect to the left triangle. While having the same area (perfect smoothness), the mutually oblique orientation results in poor orthogonality. In this particular case, the centre of the circumscribing circle is in principle located outside the right triangle. Such a triangle is denoted as an 'open' triangle, which is bad for computations.

The opposite is shown in the right picture of Figure 1.3 in which the right triangle has strongly been elongated, disturbing the smoothness property. However, the orthogonality is nearly perfect. Nonetheless, both meshes need to be improved to assure accurate model results.



(a) *Perfect smoothness, but poor orthogonality.* ***(b)*** *Perfect orthogonality, but poor smoothness*

Figure 1.3: *Poor mesh properties due to violating either the smoothness or the orthogonality at the edge connecting two triangles. Black lines/dots are network links/nodes, blue lines/dots are flow links/nodes.*

1.4 D-Flow FM workflow

Building and running a D-Flow FM model basically can be done in five steps:

	Step	Input	Tools	Output files
1.	Build a mesh	landboundaries, bathymetry	splines, polygons, polylines	<∗_net.nc>
2.	Impose boundary conditions	<∗_net.nc>	splines, polygons, polylines	<∗.pli>, <∗.cmp>, <∗.tim>, <∗.ext>
3.	Initialise the model	<∗_net.nc>, <∗.ext>		<∗.mdu>
4.	Perform the computation	<∗_net.nc>, <∗.mdu>		<∗_his.nc>, <∗_map.nc>
5.	View the outcomes	<∗_his.nc>, <∗_map.nc>	GoogleEarth, Quickplot, OpenEarth	<∗.kml>, etc.

Table 1.1: *Workflow for building and running a D-Flow FM model.*

1.5 Tutorials in this manual

In this manual, the various aspects of the use of D-Flow FM are illustrated by means of tutorials. One can follow these tutorials subsequently, in the order in which these are reported. One can also pick the tutorials of personal interest and use the files necessary for that particular tutorial. These necessary files are available for each tutorial within the provided directories. Each directory contains the results of the previous tutorial.

Section 9.1 explains the basic functionality of mesh generation. This section contains an overview of keyboard shortcuts in the D-Flow FM workspace as well as a short tutorial on basic mesh operations like placing, replacing and deleting nodes of a mesh.

Section 9.2 contains some more advanced mesh generation tutorials. This section deals both with curvilinear meshes and unstructured meshes. Although curvilinear meshes are

very familiar to Delft3D-users, one can practice with some newly implemented curvilinear mesh generation features. In this section, a mesh is elaborated for the Westerscheldt area, containing several types of curvilinear meshes as well as an unstructured mesh, and their mutual coupling.

Section 9.3 demonstrates how boundary conditions and initial conditions can be imposed, once one has composed a mesh in D-Flow FM. Thereafter, one can practise with actually running a hydrodynamic computation using D-Flow FM in section 9.4. Section 9.5 gives a very brief overview of the current postprocessing functionality in Google Earth and QuickPlot.

1.6 Quick learning

Following all the tutorials provided in this manual may take too much time for one afternoon. If you want to get yourself familiar with the basics of D-Flow FM, the following tutorials are recommended:

- Tutorial 1 on page 61: the very basics of drawing a mesh. Estimated duration: 10 minutes.
- Tutorial 2 on page 64: generating a curvilinear mesh in a Delft3D-like way. Estimated duration: 15 minutes.
- Tutorial 3 on page 66: generating a curvilinear mesh using significantly improved functionality. 15 minutes.
- Tutorial 4 on page 67: generating a triangular mesh. Estimated duration: 30 minutes.
- Tutorial 5 on page 69: connecting two meshes. Estimated time: 20 minutes.
- Tutorial 8 on page 77: interpolation of bottomlevel data on the mesh. Estimated duration: 10 minutes.
- Tutorial 9 on page 78: imposing boundary conditions. Estimated duration: 30 minutes.
- Tutorial 13 on page 83: starting the computation. Estimated duration: 20 minutes.

Hence, the total estimated duration of these tutorials is about 2.5 hours.

All the necessary files for each tutorial are provided in directories. For instance, the input data for tutorial 6 are given in the directory *.../tutorial06*. Not only the input data are provided, but also the output. In this way, it is possible to continue the tutorials if you could not manage the obtain the desired result.

The files that comprise the results of the mesh generation procedure, as well as the files that prescribe the boundary conditions, are found in the directory *.../tutorial13*. In tutorial 13, it is explained how the computation can actually be run.

2 Getting started

2.1 Installation

Installation of D-Flow FM currently comes down to copying all necessary program files into a single directory, for example: <c:\delft3d\w32\dflowfm\>.

Required files are: <unstruc.exe>, <unstruc.ini>, <interact.ini>, <isocolour.hls>, and a set of third-party DLLs, al contained in the release-zipfile. Optional is the <rununstruc.bat> batch file, which automates the installation of all necessary files.

Starting D-Flow FM from different work directories

It is recommended to start D-Flow FM from within your working directory. The batch file <rununstruc.bat> copies the necessary files to the current directory (e.g., your problem/work directory) and starts D-Flow FM. During installation, add the D-Flow FM installation directory to your Windows environment variable `PATH`.

- In Windows, click *Start* → *Control Panel* → *System*.
- In the System Properties window, click *Advanced* → *Environment Variables*.
- Under System variables, select the `Path`-line and click *Edit*.
- Append the text string `D:\unstruc;` at the front of the current `Path`-string. Change the directory if you have installed all D-Flow FM files elsewhere.

Associating MDU-files with D-Flow FM

The Master Definition Unstruc (MDU)-files can be associated with the D-Flow FM program to allow double clicking <∗.mdu> files. Below are the steps for association in Total Commander:

- In Total Commander click menu option *Files* → *Associate with...*.
- Enter 'mdu' in the file extension dropdown box.
- Click button *New type...*
- In the file dialog browse to the D-Flow FM install directory and select <startunstruc.bat>.
- Confirm with *OK*.

2.2 Starting D-Flow FM

Start <unstruc.exe> from the installation directory, or call <startunstruc.bat> from anywhere to automatically copy all beforementioned program files into your problem directory and start it from there. A window with a large blank work area should come up.

The directory where D-Flow FM was started will contain a diagnostics file <unstruc.dia>, which might help in case of warnings or errors.

To exit D-Flow FM, click the window's *close* button or *Files* → *Stop program*.

2.3 Getting into D-Flow FM

The setup of a model will be explained in detail in the following chapters. Some basic knowledge of the D-Flow FM graphical user interface is useful in advance, though. The D-Flow FM window basically is one big workspace, which you can operate with mouse clicks and command keys. Above it is a single menubar.

2.3.1 The workspace

When the workspace is active, your mouse cursor will display as a crosshair pointer. When the menubar is active, it is highlighted and the mouse cursor displays as a normal arrow pointer.

◇ Move your mouse around the workspace. Notice the coordinates displayed in the top right corner.

When a computation is not running, the workspace is always in *Edit*-mode.

◇ Click some points in the workspace, notice how a polyline is formed.

The default interactive mode is *Edit Polygon*-mode. You can switch to other modes (*Network*, *Spline*, *Grid* and *Samples*) through the *Edit* menu.

◇ Switch to splines through *Edit* → *Edit splines*. Click three or more points in the workspace, notice how a new spline is formed.

During all edit operations, you can always zoom in/out the visible part of the workspace by pressing z.

◇ With the polygon and spline still visible, press z. A wireframe shows up, which indicates the next zoom window. Confirm with a single left mouse click.
◇ The workspace was just zoomed in. Press z, do *not* click, and press z once more. The view is now zoomed out.
◇ Zoom in once again as in step 1.
◇ Press z, do *not* click and move the mouse/wireframe to the top/bottom/left or right boundary of the workspace. When the wireframe hits the window edge, the view shifts up/down/left or right, respectively. This is the way to move through your domain locally. For quicker navigation, zoom out several times (z+z), move your mouse to a different part of the domain and zoom back in several times (z+click).
◇ Press z and confirm with right mouse click. The workspace is now zoomed out to show all meshes, polygons, etc.
◇ To cancel a zoom operation (if you have accidentally pressed z, or to force a window redraw) simply press Esc.

As better alternative to z+Esc for redrawing is through menu item *Display* → *Redraw*.

2.3.2 The menu bar

D-Flow FM's menu bar offers six menus:

File	Open and save MDU files, various grid files, polygon, spline and land boundary files, observation, cross section and thin dam files, bathymetry and sample files, and restart files.
Operation	Various network operations (grid generation, local refinement, etc.), bathymetry and flow initialization, and starting flow computation.
Display	Configure the various mesh and flow quantities to be displayed, show/hide certain elements in the workspace and additional display settings.
Edit	Switch to a different interactive *Edit* mode (*Poygon*, *Network*, *Spline*, *Grid*, *Samples* and *Flow*).
Addsubdel	Delete various workspace items, set up waterlevels and other quantities, and interactive creation of observation stations, cross sections, thin dams, and sample points.
Various	Change parameter for physical and numerical model, geometry, co-

ordinates and interpolation, curvilinear and unstructured grids.

To exit a menu, simply click somewhere outside of it.

2.3.3 Keyboard shortcuts

Keyboard shortcuts are case insensitive unless denoted otherwise.

Table 2.1: *Keyboard shortcuts in the D-Flow FM workspace*

Key(s)	Functionality
Within *Workspace*, all modes	
Alt+P	Switch to *Edit Polygon*-mode
Alt+N	Switch to *Edit Network*-mode
Alt+S	Switch to *Edit Spline*-mode
Alt+G	Switch to *Edit Curvilinear grid*-mode
Alt+B	Switch to *Edit Samples (bathymetry)*-mode
Alt+F	Switch to *Edit Flow*-mode
A	Set anchor at current mouse position (used to measure distances).
Z	Zoom in (click to confirm, or Z again to zoom out, or move mouse to a workspace side to shift).
+	Enlarge the zoom frame (after Z-press), click to confirm).
−	Shrink the zoom frame (after Z-press), click to confirm).
Esc	Cancels current operation, restores last state (for either polygon, network, etc.).
Backspace	Deletes the selected (or all) items in the current mode.
In *Edit Polygon/Network/Flow*-mode for 3D models	
Home	Plot the next layer (one up).
End	Plot the previous layer (one down).
Page Down	Plot vertical profile at next flow node.
Delete	Plot vertical profile at previous flow node.
In *Edit Polygon*-mode	
Left click	Selects or puts a polygon point, depending on the following operations:
I	Start inserting new poly points after point currently under the mouse pointer (or start a new poly).
R	Move a poly point (next, click the point and place the point).
D	Delete a poly point (next, click the point).
X	Split a poly in two at the point currently under the mouse pointer.
E	Erase the polygon currently under the mouse pointer.
e	Erase *all but* the polygon currently under the mouse pointer.
F	Refine (part of) poly linearly (next, click begin and end point).
M	Connect two polys (next, click the two points to be connected).
L	Snap (part of) poly to nearest land boundary (next, click begin and end point).
N	Snap (part of) poly to nearest network boundary (next, click begin and end point).
w/W	Raise/lower water levels under mouse pointer/within active polygon.
b/B	Raise/lower bottom levels under mouse pointer/within active polygon.
;	Start a movie (create snapshots) of the current view, shifting the view window along the current polygon.
.	Flip orientation of polygon under mouse pointer, or of all polygons.
In *Edit Network*-mode	
Left click	Selects or puts a net node, depending on the following operations:
I	Start inserting new net nodes (click existing nodes to connect directly).
R	Move a net node (next, click the point and place the point).
V	Smooth ('field-')move a net node (next, click the point and place the point).
B	Smooth rotation of local network (next, click the point and click other point to set amount and direction of rotation).
D	Delete a net node (and its net links) or just a link (next, click the node or link).
M	Merge one net node with another (next, click the two points to be merged).
C	Delete a link by clicking its two net nodes (next, click the two nodes).

(continued on next page)

Key(s)	**Functionality** *(continued from previous page)*
X	Delete a node and connect its two neighbours (next click a node with two nbs).
s	Split a net link and its neighbouring cells, i.e., add a new net node in the middle (next click the link).
S	Insert a mesh line, i.e., an automatically repeated 's' split (next click a cross link that the new mesh line should go through).
1	Toggle 1D-2D type for net links (next, click the link(s)).
k	Kill cell currently under mouse by connecting all surround links to its center point
K	Derefine all grid cells (within polygon, if any) (Casulli-inverse).
G	Detect largest possible curvilinear grid under mouse pointer and convert it to a regular grid.
L	Snap network boundaries to nearby land boundaries.
E	Grow grid layers from a network boundary (next, click a network boundary node).
In *Edit Spline*-mode	
Left click	Selects or puts a spline point, depending on the following operations:
Right click	Ends the current spline (only in *insert*-mode).
I	Start inserting new spline points after point currently under the mouse pointer (or start a new spline).
R	Move a spline point (next, click the point and place the point).
D	Delete a poly point (next, click the point).
C	Copy a spline orthogonally (next, click a spline point and place the point).
M	Move a spline (next, click a spline point and place the point).
L	Snap spline to nearest land boundary (next, click a spline point).
X	Delete a spline (next, click a spline point).
In *Edit Grid*-mode	
I	Insert new grid cell by expanding a boundary cell (next click near an edge).
R	Move a grid point, in either *point*- or *field*-mode (next, click the point and place the point).
D	Delete a grid point (next, click the point).
In *Edit Samples*-mode	
Left click	Selects or puts a new sample point, depending on the following operations:
I	Start inserting new sample points.
R	Move a sample point (next, click the point and place the point).
D	Delete a sample point (next, click the point).
C	Change the z-value of a sample point (next, click the point).
m	Sets the sample coloring lower bound (next, click a sample point, or in empty space for reset to auto scaling).
M	Sets the sample coloring upper bound (idem).
Q	Create a highest-walk sample path (next, click a sample).
F	Flood fill (next, set water level).
In *Edit Flow*-mode	
Left click	Selects a flow node or flow link, depending on the following operations:
N	Display details at one flow node (next, click a flow node/cell).
L	Display details at one flow link (next, click a flow link).
m	Sets the flow node colouring lower bound (next, click a sample point, or in empty space for reset to auto scaling).
M	Sets the flow node colouring upper bound (idem).

3 Mesh generation

3.1 Introduction

Computations in D-Flow FM are performed on unstructured meshes, called *nets*. The mesh quality is critical to the accuracy of the simulation. Loosely speaking, one aims for meshes that are:

- ◇ orthogonal,
- ◇ smooth,
- ◇ sufficiently dense in regions of importance.

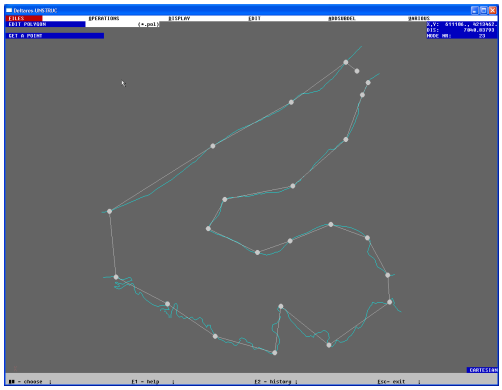
This chapter elaborates on the generation of such meshes. The first two sections cover the generation of an initial mesh in a specific subdomain (i.e., inside a polygon) and the subsequent mesh enhancement procedure. Refining the mesh locally and projecting the mesh onto a land boundary are described in the remainder of this chapter.

3.2 Creating an initial mesh inside a polygon

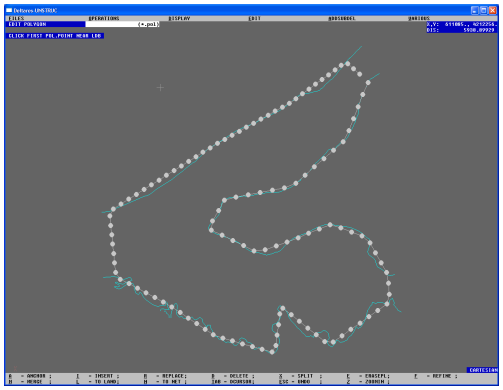
The initial mesh is created by firstly generating sample points inside a polygon and subsequently triangulating the samples to a mesh. This polygon could be based on land boundaries. In Figure 3.1 an example is given. The generation of the initial mesh comprises the following steps:

- ◇ Compose a polygon near the land boundary and indicate the objective mesh size with the begin and end segment of the polygon respectively, see Figure 3.1a for an example.
- ◇ Refine the polygon through *Operation* $\rightarrow$ *Refine Polygon*, see Figure 3.1b.
- ◇ Press `L` (and click start and end point) to snap the refined polygon to the land boundary, see Figure 3.1c.
- ◇ Define the maximum coarseness in the interior through *Various* $\rightarrow$ *Change network* $\rightarrow$ *TRIANGLESIZEFACTOR, MAX.INSIDE/ AV.EDGE*. This controls the growth of the triangle size from the boundary. In this particular example, it is set to 2.
- ◇ Create sample points inside the polygon through *Operation* $\rightarrow$ *Create samples in polygon*, see Figure 3.1d.
- ◇ Create the initial mesh through *Operation* $\rightarrow$ *Triangulate samples to net in polygon*, see Figure 3.1f.

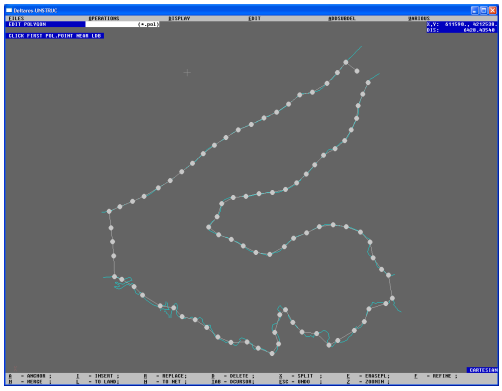
The initial mesh may not be of good quality immediately, in terms of orthogonality or smoothness. Orthogonality can be inspected by selecting *Display* $\rightarrow$ *Values at net links* $\rightarrow$ *Link Orthogonality COSPHI*. For orthogonal grids, COSPHI should be zero. In practice, however, one should aim to get these values as small as possible (< 0.001). This is achieved by enhancing the mesh, as explained in the following section.



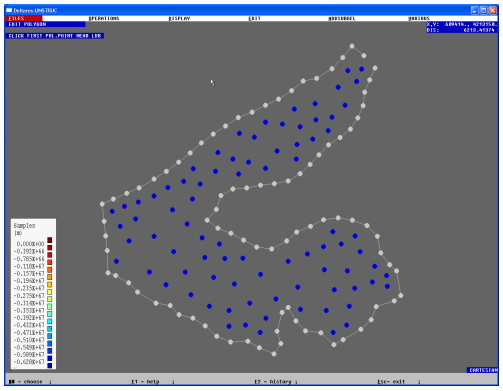
(a) *Initial polygon near the land boundary*



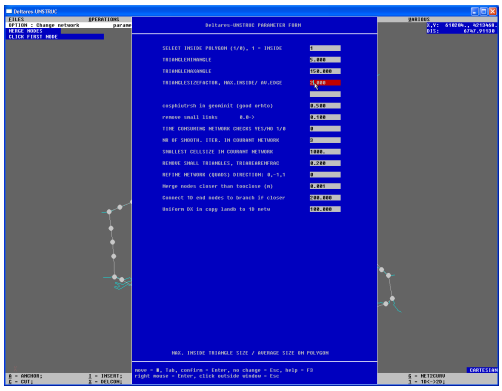
(b) *Refined polygon*



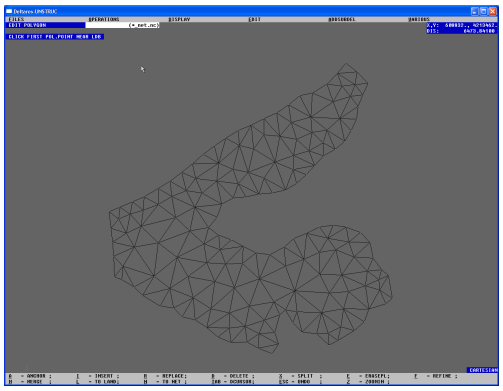
(c) *Polygon snapped to land boundary (*`L` *key)*



(d) *Samples*



(e) *Network parameters menu*



(f) *The initial mesh*

Figure 3.1: *Creating an initial triangular mesh inside a polygon*

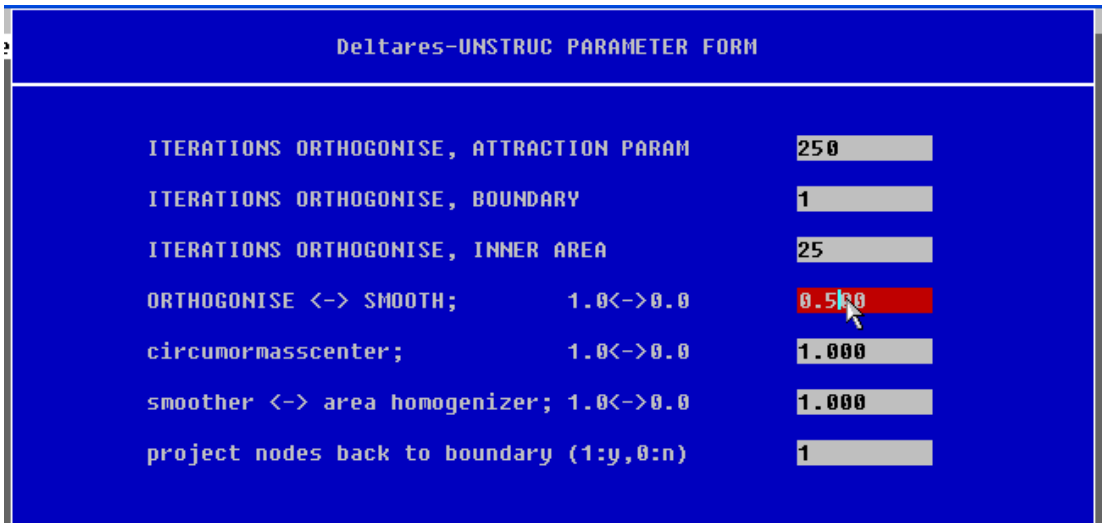


Figure 3.2: *Orthogonalization parameters menu; in this example the number of iterations is set to* 250 *and the orthogonalization parameter to* 0.500

3.3 Enhancing the mesh

The mesh smoothness and orthogonality can be improved simultaneously through *Operation* $\rightarrow$ *Orthogonalize net*. The parameters governing this procedure can be changed in *Various* $\rightarrow$ *Orthogonalization parameters*, see Figure 3.2. Two parameters are of particular interest:

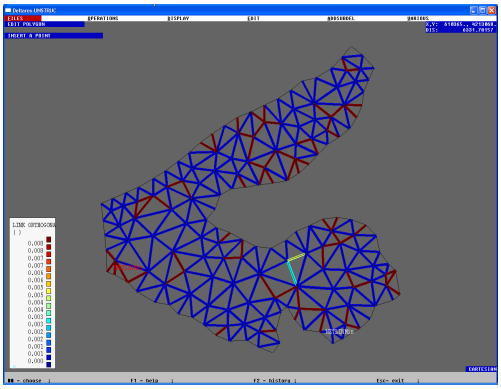
- `Iterations orthogonalize, attraction param`: the number of iterations in the orthogonalization. This number should be sufficiently large to allow mesh convergence.
- `Orthogonalize <-> Smooth`: the balance between mesh-smoothing (0.0) and mesh-orthogonalization (1.0). One has to keep in mind that mesh smoothing (ortho. param. $\rightarrow$ 0) will compromise mesh orthogonality. Sole orthogonalization (ortho. parameter=1) on the other hand, can cause highly distorted, non-smooth meshes, especially for meshes consisting of quadrilaterals. It is advised to choose a low orthogonalization parameter and repeat the process while gradually increasing the orthogonalization parameter at every repetition.

stage	1	2	3	4	5	6
ortho. parameter	0.5	0.8	0.9	0.99	0.999	0.9999

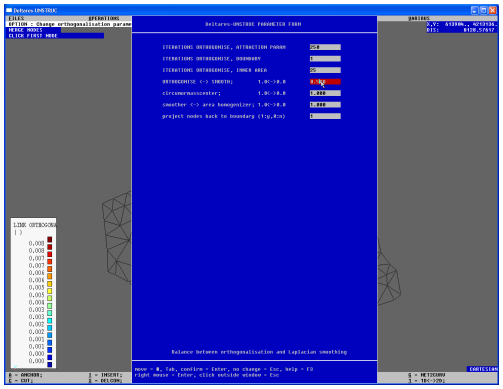
Table 3.1: *Multi-stage orthogonalization strategy*

In a repeated, or multi-stage, orthogonalization approach, the orthogonalization parameter is typically chosen as indicated in Table 3.1. At every stage, one has to ensure (by visual inspection) that the mesh is sufficiently converged and remains smooth. Mesh distortion is often a sign that the orthogonalization parameter was increased too rapidly. If mesh distortion appears during the iteration process, one can stop the iterations by pressing the right mouse-button and undo the orthogonalization through *Operations* $\rightarrow$ *Undo net*. Hereafter, the process can be resumed with an adjusted (smaller) orthogonalization parameter. One could also opt for orthogonalization in confined areas using polygons.

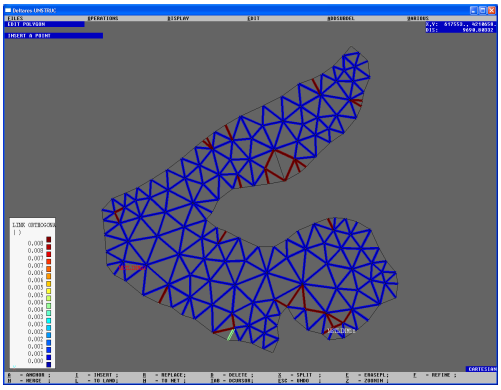
If mesh convergence within a stage is observed visually, before the maximum number of iterations is reached, or for any other reason, the iterations can be stopped by pressing the right mouse button at any time. The left mouse button toggles the mesh rendering on/off. This will speed up the orthogonalization process.



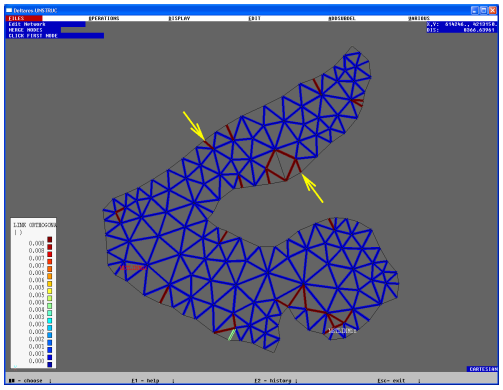
(a) *initial mesh; red edges have bad orthogonality, while blue edges are orthogonal in this color scale*



(b) *Orthogonalization parameters menu; the number of iterations is set to* 250 *and the orthogonalization parameter to* 0.500

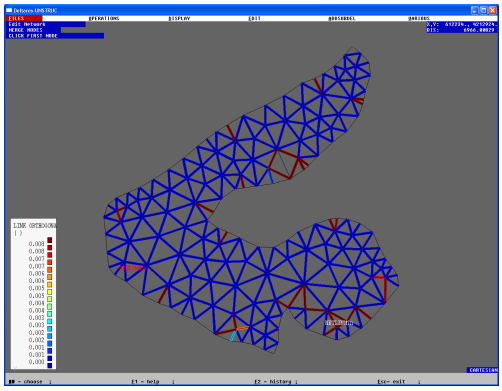


(c) *stage 1: orthogonalization parameter=*0.5*; orthogonality did improve as can be seen from the number of non-orthogonal ('non-blue') edges*

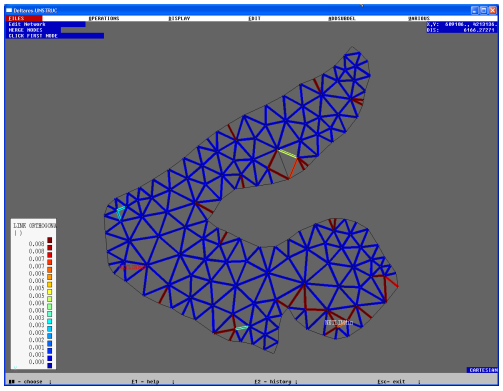


(d) *manual merge of colliding nodes after stage 1; yellow arrows indicate the merged nodes*

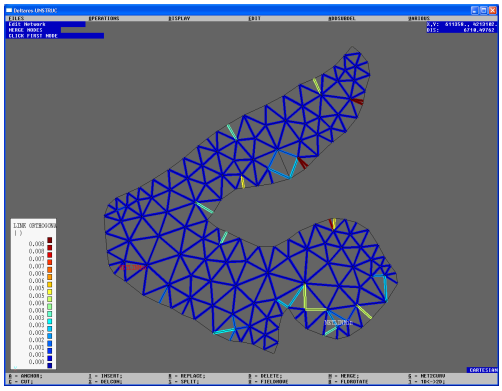
Figure 3.3: *Multi-stage mesh orthogonalization; colors indicate the mesh orthogonality, varying from* 0 *(blue) to* 0.008 *(red)*



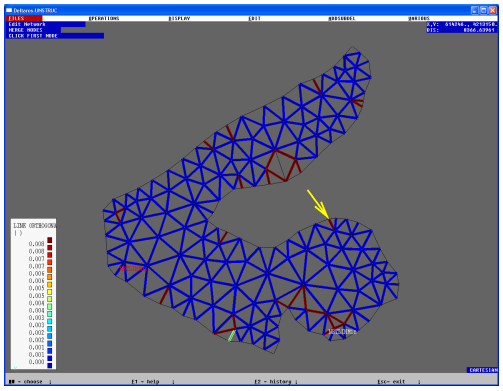
(e) *stage 2: param.=0.8; no dramatic changes are observed, but still it is advised to gradually increase the orthogonality parameter, especially for meshes that contain quadrilaterals (not shown here)*



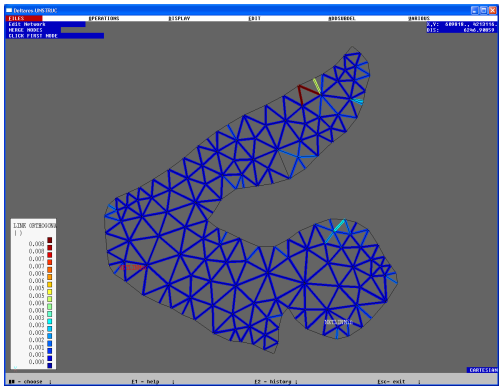
(f) *stage 3: param.=0.9; slight improvements in orthogonality are observed*



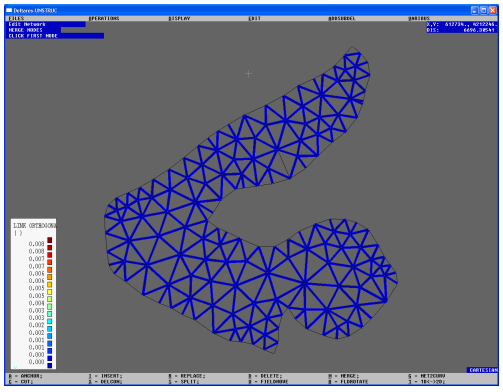
(g) *stage 4: param.=0.99; orthogonality has improved, as the color of the non-orthogonal edges changed from red to yellow/green/blue*



(h) *manual merge of colliding nodes after stage 4; the yellow arrow indicates the merged node*

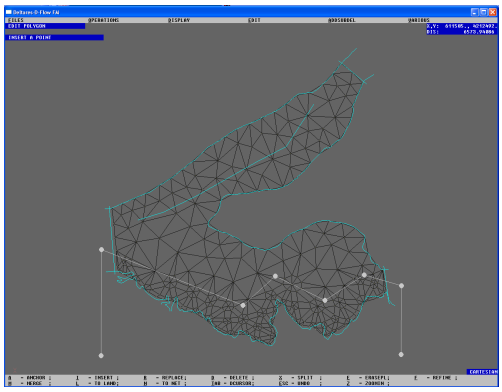


(i) *stage 5: param.=0.999; only a few non-orthogonal edges are distinguishable in this color scale*

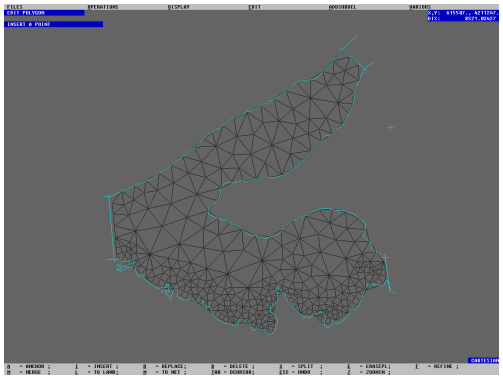


(j) *stage 6: param.=0.9999; all edges are orthogonal in this color scale, i.e. < 0.0005*

Figure 3.3: *continued*



(a) *Mesh after the Casulli-type refinement within a polygon*



(b) *Smoothed mesh; orthogonalization parameter=0.0; note that the polygon for marking the area of refinement has been deleted prior to this operation*

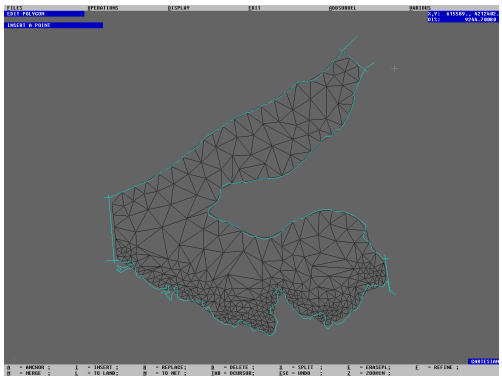
Figure 3.4: *Casulli-type mesh refinement within a region marked by a polygon*

In Figure 3.3 the initial mesh of our previous example is orthogonalized. The orthogonalization parameter is gradually increased in six stages from 0.5 to 0.9999. At each orthogonalization stage, the number of iterations "ATTRAC. PARAM" is set to 250 to ensure convergence within that stage. The iterations were stopped with the right mouse button when mesh convergence was observed visually at every stage. Figures 3.3d and 3.3h show that colliding boundary nodes are manually merged (*Edit Network*, M key). The resulting mesh in Figure 3.3j has an orthogonality smaller than 0.0005.

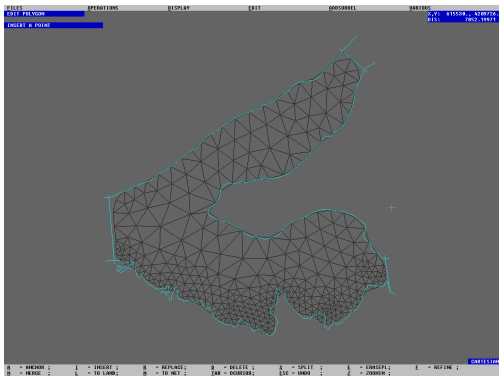
3.4 Local mesh refinement

Local mesh refinement can be achieved by so-called Casulli-type mesh refinement, available through *Operation* $\rightarrow$ *Refine cells factor 2 (Casulli-type)*. Figure 3.4a shows how this procedure is applied to our previous mesh. Note that the polygon defines the region of refinement. Note also that we have augmented the land boundary with three segments and created a, more or less, closed contour. The reason for this will become clear later on, when meshlines are projected onto the land boundary.

Figure 3.4 shows how the quality of the mesh has improved by smoothing it as described in the previous sections. Note that the selecting polygon used to mark the region of mesh refinement, was deleted prior to this operation through *Addsubdel* $\rightarrow$ *Delete polygon*. In this particular example the orthogonalization parameter has been set to 0.0, meaning that the mesh is fully smoothed and will in principal not be orthogonal. Before improving mesh orthogonality by increasing the orthogonalization parameter (again, see the previous sections), we will first improve the mesh by changing its *topology*, or connectivity, in the next section. Anticipating this, note that the Casulli-type mesh refinement resulted in both triangular and quadrilateral mesh cells as can be seen in Figure 3.4b.



(a) *Mesh after topology improvement with 'Flip links'*



(b) *Smoothed mesh; orthogonalization parameter=*0.0

Figure 3.5: *Improving mesh connectivity with 'Flip links'*

3.5 Improving mesh connectivity

The Casulli-type mesh refinement as described in the previous section resulted in highly non-uniform cell sizes, even after full mesh smoothing, see Figure 3.4. The reason for this lies in the way the nodes are interconnected. Nodes that are connected to more than, say, six other nodes, are typically enclosed by cells of highly non-uniform shape and wildly varying sizes. This motivates to improve the mesh connectivity by selecting *Operation* $\rightarrow$ *Flip links*.

Having selected 'Flip links', the user is asked in a pop-up screen whether the non-triangular cells need to be triangulated or not. Remember that Casulli-refinement resulted in both triangular and quadrilateral cells. Here we have chosen to do so, causing the quadrilateral cells to be triangulated before the 'link flipping' operation. One may choose *not* to triangulate the non-triangular cells when parts of the mesh originate from curvilinear grid generation.

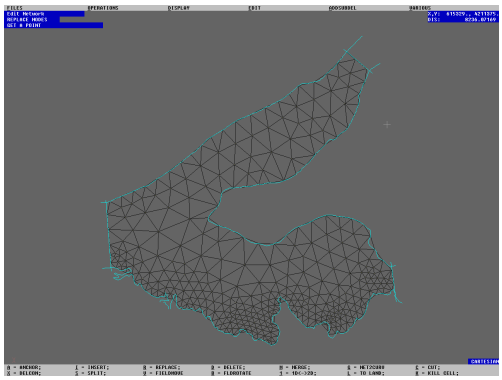
The next question that is prompted concerns the land boundaries. This topic will be discussed in the next section and for now we choose *not* to take the land boundaries into account.

The resulting mesh is depicted in Figure 3.5a. Although somewhat discouraging at first sight, the smoothed mesh (orthogonalization parameter=0.0) in Figure 3.5b clearly confirms the viability of mesh connectivity improvement with 'Flip links'. One has to remember that mesh smoothing (ortho. param < 1) is a necessary step after 'link flipping'. Orthogonalization of the mesh can now be performed as discussed in the previous sections. It will not be repeated here.

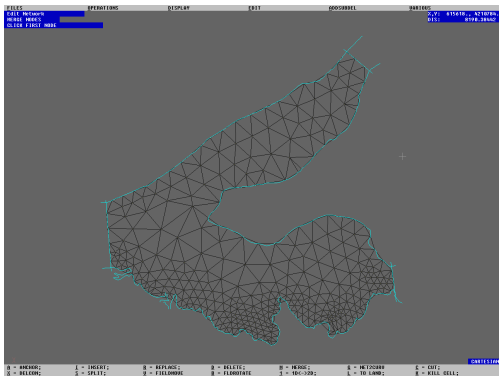
3.6 Projecting the mesh onto a land boundary

Although our efforts may have produced a satisfactory mesh so far, it certainly does not conform to the land boundary any longer, as can be observed from Figure 3.5. To overcome this, one has to possibility to project meshlines (not necessarily the mesh boundaries) onto the land boundary during orthogonalization/smoothing. In *Various* $\rightarrow$ *Orthogonalization parameters* the parameter 'project to (land)boundary' can be set according to:

0 do not project onto any boundary,
1 project the mesh boundaries onto the original mesh boundaries,
2 project the mesh boundaries onto the land boundary,
3 project the mesh boundaries onto the land boundary and (nearest) meshlines in the *whole* mesh onto the remaining parts of the land boundary,
4 project the nearest meshlines in the *whole* mesh onto all land boundaries.



(a) *Smoothed grid, i.e. orthogonalization parameter = 0.0*



(b) *Orthogonalized grid, i.e. orthogonalization parameter = 1.0*

Figure 3.6: *Smoothed and orthogonalized mesh with 'project to (land)boundary' parameter set to* 2*, i.e. projection of mesh boundaries to the land boundary*

The tolerances that determine whether a meshline (or mesh boundary) is close to a land boundary or not, and whether the meshline will be projected on that land boundary or not, can be controlled by specifying two 'snap-to-landbdy tolerance' parameters in *Various* $\rightarrow$ *Network parameters*. These are the tolerance for the 'netboundary' (projection parameter set to 2 or 3) and the 'inner network' (projection parameter set to 4) respectively. They are measured in typical mesh widths and their defaults are 5 and 1 mesh widths respectively.

Figure 3.6 shows the effect of full mesh smoothing (Figure 3.6a) and subsequent orthogonalization (Figure 3.6b) along the lines of the multi-stage approach of section 3.3. The mesh boundaries are projected onto the land boundary be setting the projection parameter to 2. The orthogonality of the final mesh in Figure 3.6b is smaller than 0.01.

Orthogonalization and smoothing with the projection parameter set to 3 or 4 is useful when meshlines need to be enforced at internal boundaries. We will encounter such a situation when we connect curvilinear grids in chapter 5.

4 Generation of curvi-linear grid from splines

4.1 Introduction

This chapter describes the generation of curvilinear grid from splines. The user is required to provide a spline, from which the grid is grown perpendicularly. Note that the grid can be grown from multiple center splines simultaneously. Per center spline, the extent of the grid and the heights of the grid layers can be controlled by supplementary splines and setting parameters as will be described shortly. Note that the grid generation process can be interrupted at any time by pressing a mouse button.

4.2 Supplying the splines

There are three types of splines that form the basis of the grid generation, see Figure 4.1. These are:

- a center spline (A),
- bounding splines (B_u and B_e), which exist pairwise. Such pairs are called sets. One set of bounding splines contain two splines, one on each side of the center spline. There may be up to two sets of bounding splines:
  - a middle set B_u, closest to the center spline, bounding the uniform grid part
  - an outer set B_e, bounding the exponentially growing grid part.

  Note that the center spline may be absent, in which case one of the splines B_u acts as center spline as well.
- cross splines (C). Splines are considered to cross one another only if they are nearly perpendicular. This can be set by "MINIMUM ABS. SINE OF CROSSING ANGLES" in the parameter screen. Setting this value to 0.95, for example, corresponds to a required minimum angle of about 70 degrees.

First a grid line will be generated on the center spline. Next the grid is grown perpendicularly into two parts:

- a uniform part from the center spline to the first set of bounding splines (B_u). The grid layer heights are uniform perpendicular to the spline (but not necessarily uniform along the spline). If no bounding splines are present, no uniform grid part will be created,
- an exponentially growing part from the middle bounding splines B_u towards the outer B_e. The user can prescribe the grow factor by setting the "GRID LAYER HEIGHT GROWTH

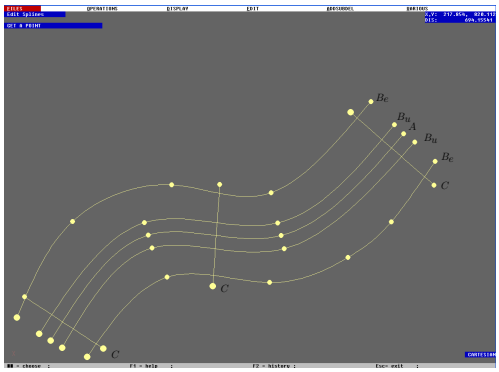


Figure 4.1: *A: center spline, B_u: middle bounding splines, B_e: outer bounding splines, C: cross spline*

Figure 4.2: *various combinations of splines and the resulting grid*

Figure 4.2: *continued*

FACTOR" in the parameter screen. If the outer set of bounding splines is absent, the length of the cross splines is used to determine the grid height.

The uniform part, if it exists, is grown first. The exponential part, if it exists, is grown subsequently. The following rules are obeyed:

- ◇ a cross spline contains two control points,
- ◇ a center or bounding spline contains three or more control points,
- ◇ the center and bounding splines are crossed by at least one cross spline,
- ◇ there should be equally many bounding splines on either side of the center spline,
- ◇ if no center spline is specified, one of the bounding splines in the middle set acts as center spline too. The uniform grid is only grown from the first middle bounding spline/center spline towards the other bounding spline in the middle set,
- ◇ if the outer set of bounding splines is absent, the total grid height is determined by the cross-spline control points,
- ◇ whether or not the exponential part of the grid is generated in the presence of a uniform part, is controlled by setting "GROW GRID OUTSIDE FIRST PART" in the parameter screen to "0" (no) or "1" (yes).

The various combinations of splines and the resulting grids are illustrated in Figure 4.2.

4.3 Parameter form

Generation of curvilinear grids from splines is performed through *Operations* → *Grow curvilinear grid from splines*. The user is presented with a parameter screen, in which the following variables can be set:

- ◇ maximum number of grid cells along spline: *upper bound* of the number of grid cells along the center spline. The actual number of grid cells is determined by the cell and spline lengths, but will not exceed this number. In practice one has to set this number sufficiently large,
- ◇ maximum number of grid cells perp. spline: *upper bound* of the number of grid layers that will be grown from the center spline. The actual number of grid layer is determined by the grid height (specified by the splines, see section 4.2), the height of the first grid layer and the grid layer growth factor, see below, but will not exceed this number. In practice one has to set this number sufficiently large,
- ◇ aspect ratio of first grid layer: the ratio of the grid cell height and length at the center spline. If a center spline is provided solely, the aspect ratio of the grid on either side of the center spline is determined by this variable, see **??** for an example,
- ◇ grid layer height growth factor: the fractional increase of grid layer heights in the exponentially growing part of the grid,
- ◇ maximum grid length at center spline: the maximum grid cell length. Note that the length decreases where the spline curvature increases,
- ◇ grow grid outside first part (0,1): create the exponentially growing grid supplementary to the uniform part (1) or not (0). This parameter has no effect if no uniform part is present, i.e. no bounding splines are provided. In that case the exponentially growing grid is the sole grid created.
- ◇ maximum number of gridcells perpendicular to the center spline in the uniform part: the

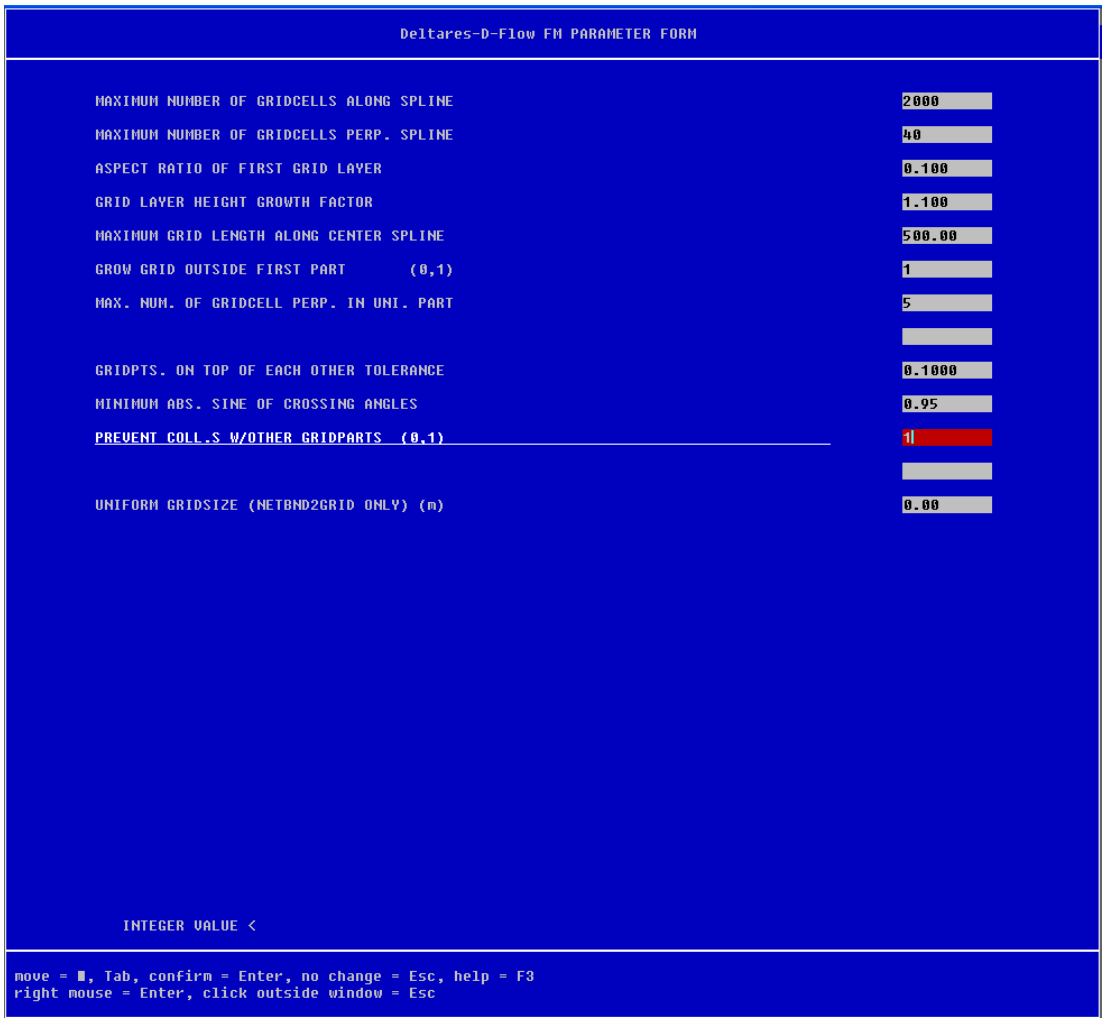


Figure 4.3: *Grow curvilinear grid form splines - parameter form*

number of gridcells will not exceed this number. If necessary, the cells will be enlarged, and the aspect ratio is disregarded.
- ◇ gridpoint on top of each other tolerance: a tolerance on merged gridlines; for expert users only.
- ◇ minimum absolute sine of crossing angles: if the absolute value of the sine of the cross angle is less than required, the crossing is disregarded.
- ◇ prevent collisions with other gridpart (0,1): setting this parameter to'0' will only detect merging grid lines, while setting it to '1' will also detect and prevent overlapping grids, but is more exhaustive.
- ◇ uniform gridsize (netboundary to grid only): this parameter only affects the growth of grid from netboundaries, which is not discussed here.

4.4 A posteriori grid operations

Once the grid generation has finished, the user is prompted whether to "remove skinny triangles" or not. This enables the user to have triangular grid cells with angels smaller than 20° automatically deleted. Hereafter, the user has the possibility to copy the center spline to a polygon. This is useful if the first grid line needs to be kept on the center spline. In this case, one could transform the land boundary into a polygon through *Addsubdel* $\rightarrow$ *copy polygon to* ... $\rightarrow$ *Copy polygon to land boundary* and proceed along the lines of section 3.6.

4.5 Spline operations

Splines can be copied *orthogonally* by pressing "C" in spline-mode, selecting a spline control point and putting it a desired distance. In this case the new spline control points are copied in a direction normal to the original spline. This is particularly useful for generating bounding splines from a center spline, vice versa. Pressing "M" instead will move the spline by just translating it.

5 Connecting curvi-linear grids

5.1 Introduction

In the previous chapter we have seen how curvi-linear grids can be generated from splines. This chapter will elaborate on the creation of junctions by connecting individual curvi-linear grids. To this end, we will consider the Mahakam delta, see Figure 5.1 and aim to generate a grid that covers its river and channels.

5.2 Creating the individual grids

Following the guidelines of the previous chapter, we first generate center, bounding and cross splines prior to growing the curvi-linear grid, see Figure 5.2. A curvi-linear grid is then grown from the splines as explained in the previous chapter. By selecting *Operation* → *Convert grid to net*, the grid is converted to an unstructured mesh as depicted in Figure 5.3. We will focus on the area marked by the polygon and aim to connect two disjunct meshes. Note that the polygon is intended for visualization purposes only. It is not a selecting polygon.

5.3 Joining the meshes

The actual connection of the individual meshes is performed by the following steps:

1. Divide the junction into *four* rectangular and *one* triangular block, see Figure 5.4,
2. Create additional block boundaries.
   The additional block boundaries are used to enforce meshlines at the block boundaries during the mesh smoothing/orthogonalization later on. They need to be (additional) land boundaries and can be created by clicking them as polylines, see Figure 5.5, and converting them through *Addsubdel* → *Copy polygon to ...* → *Copy polygon to land boundary*,
3. Create the mesh in the triangular block.
   We could this by hand. Alternatively, we could generate a uniform curvi-linear grid in a polygon, and split it in half, as indicated in Figure 5.6. We have to choose the dimensions of the grid such that it matches the dimensions of the adjacent rectangular blocks. Figure 5.7 shows the polygon and Figure 5.8 the clipped curvi-linear grid.
4. Manually connect the mesh in the triangular block with the other meshes.
   After converting the curvi-linear grid to an unstructured mesh, the nodes need to be manually merged, and in general, mesh lines may need to be inserted in the rectangular block, see Figure 5.9.
5. Smooth and orthogonalize the mesh. This will be explained in more detail in the next section.

5.4 Mesh smoothing and orthogonalization

It will be clear that the mesh so far may be connected, but is neither smooth nor orthogonal. We improve this by manually inserting meshlines and smoothing/orthogonalization the mesh as described in section 3.3. Note that:

- ◇ We will only operate on a small part of the mesh as indicted in Figure 5.10. Besides speeding up the process, we are guaranteed that no mesh parts are altered that may already be satisfactory smooth and orthogonal.
- ◇ If one is satisfied with the mesh inside the polygon, one may need to smooth/orthogonalize once again with an enlarged polygon. This new, enlarged polygon should include the parts of the mesh that was crossed by the original selecting polygon. These parts of the mesh would otherwise remain non-smooth and/or non-orthogonal.
- ◇ We enforce meshlines on the block boundaries by setting the projection parameter to 3,

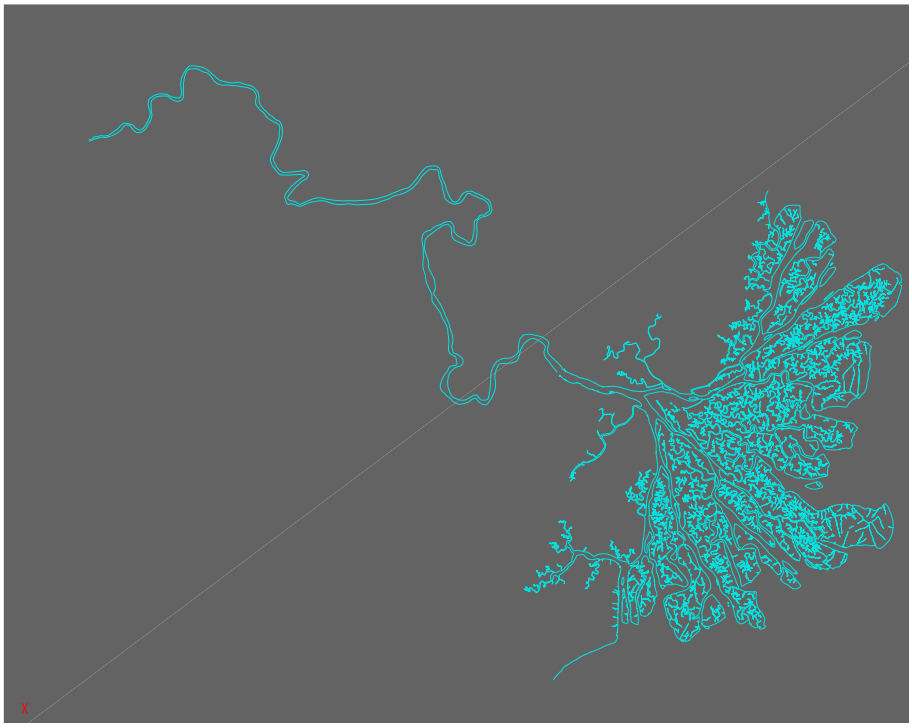


Figure 5.1: *Land boundaries in the Mahakam delta*

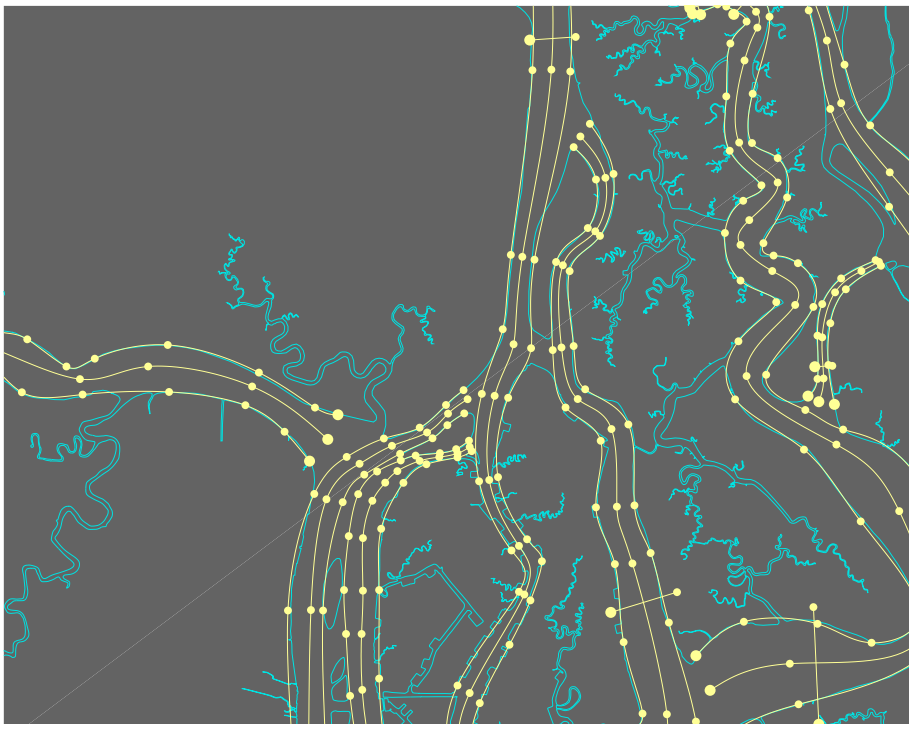


Figure 5.2: *Splines in the Mahakam delta (zoomed)*

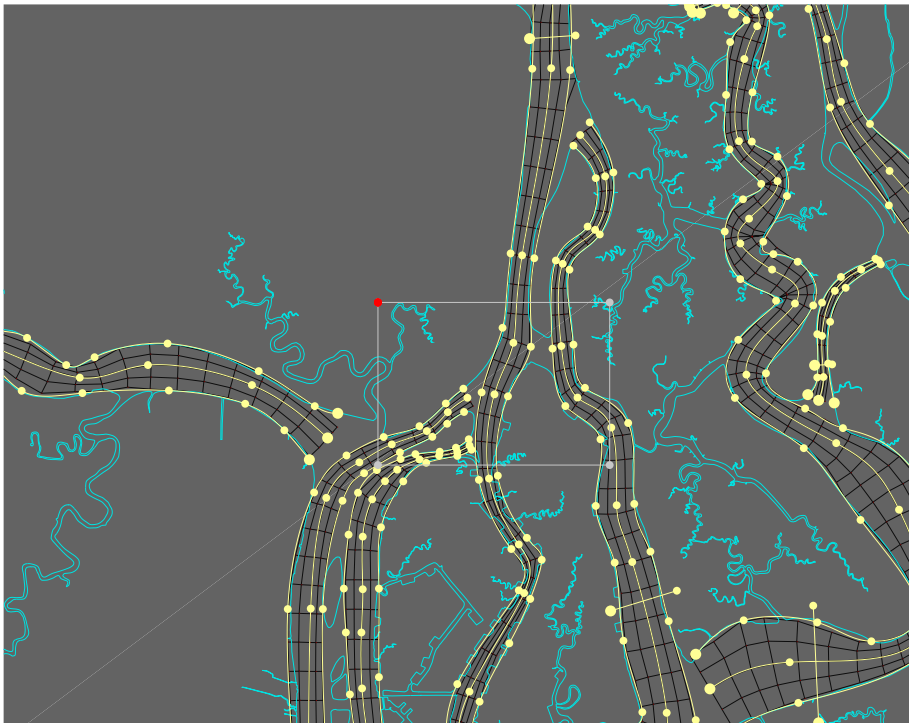


Figure 5.3: *Mesh grown from the splines; the polygon marks the area of interest*

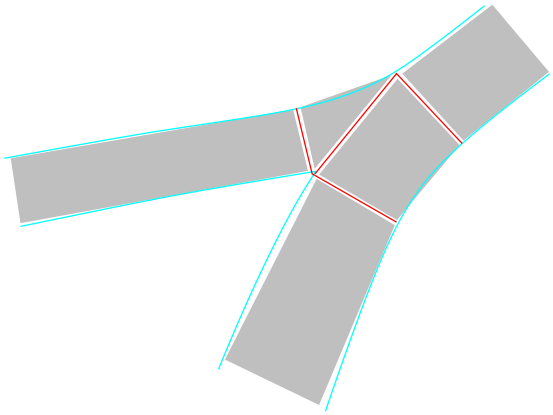


Figure 5.4: *Mesh-junction block layout; land boundaries are in blue, blocks are shaded gray, and additional block boundaries are in red*

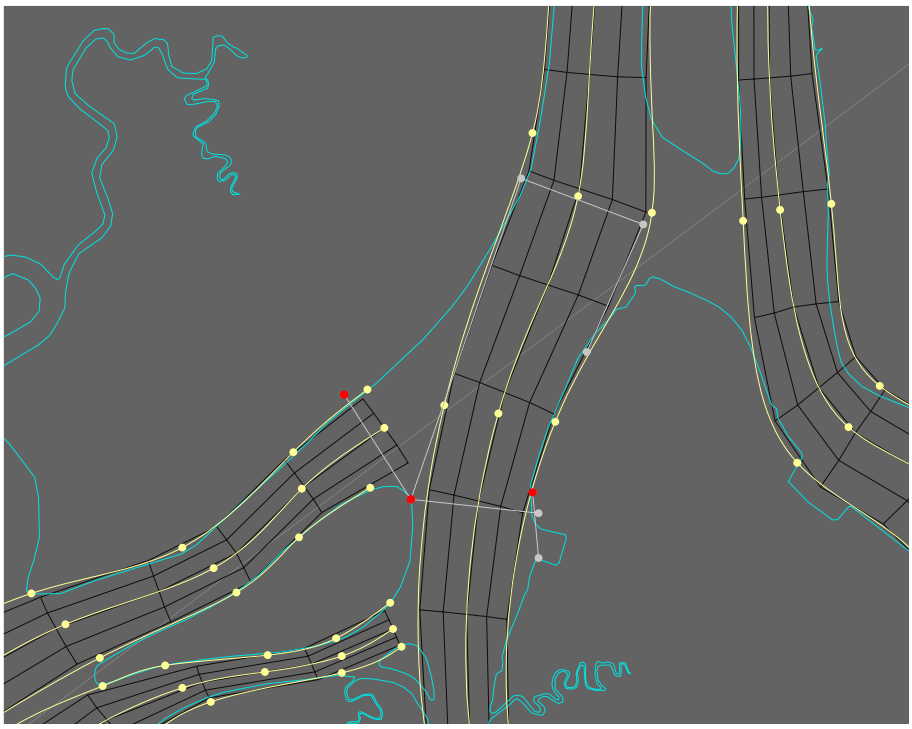


Figure 5.5: *Additional block boundaries clicked as polylines, just before the conversion to (additional) land boundaries*

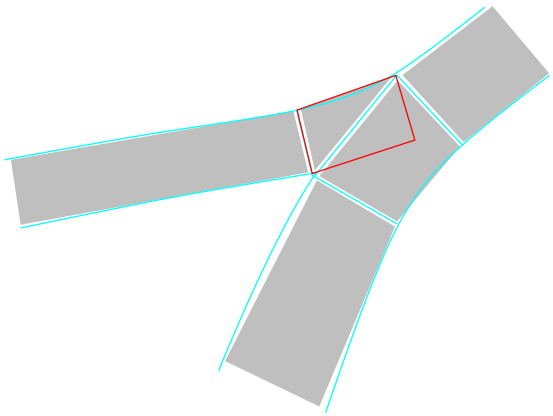


Figure 5.6: *Curvi-linear grid bounds in red, which is used to generate a mesh in the triangular block.*

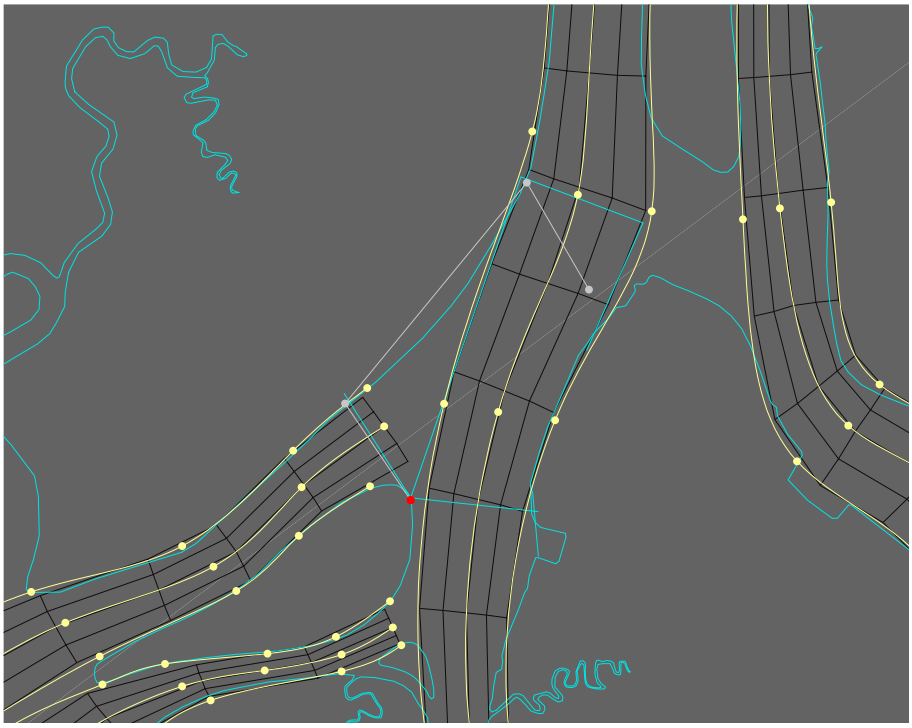


Figure 5.7: *Polygon used to generate a curvi-linear grid in the triangular block.*

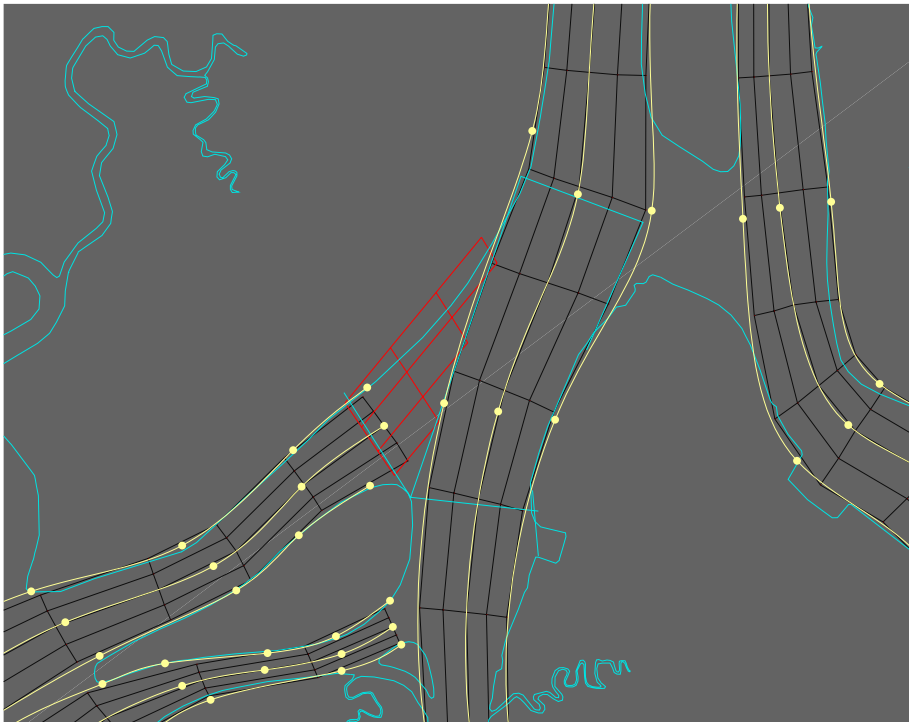


Figure 5.8: *Clipped curvi-linear grid in the triangular block*

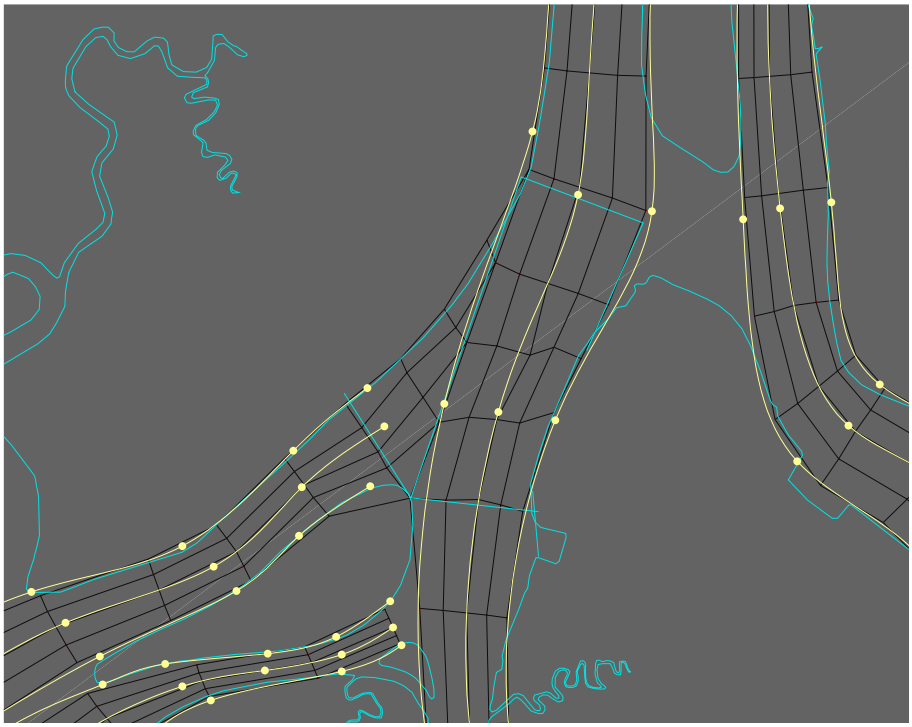


Figure 5.9: *Connected mesh in the triangular block*

see section 3.6.

The resulting mesh is shown in Figure 5.11. Repeating the steps of section 5.3 for all the junctions results in a mesh as shown in Figure 5.12.

5.5 Conversion of DD-type Delft3D grids to D-Flow FM nets

D-Flow FM facilitates the automatic coupling of typical DD-type neighbouring regular grids. Some additional manual finetuning can greatly improve the quality of the resulting nets.

5.5.1 Loading the curvilinear grids

- ◇ Load an RGFGRID <∗.grd> file through *Files* → *Add curvilinear grid*.
- ◇ Convert the grid to an unstructured net by *Operations* → *Convert grid to net*.
- ◇ Repeat the preceding two steps for all <∗.grd> files.

5.5.2 Connect DD-type curvilinear grids

- ◇ When all grids are loaded *and* converted to unstructured nets, perform the automatic coupling by *Operations* → *Connect curvilinear quads dd type*.
- ◇ For large nets, or when undesired couplings are created, limit the area of influence by clicking a masking polygon (*Edit* → *Edit polygons*). Include several rows and columns for the coupling to work properly.

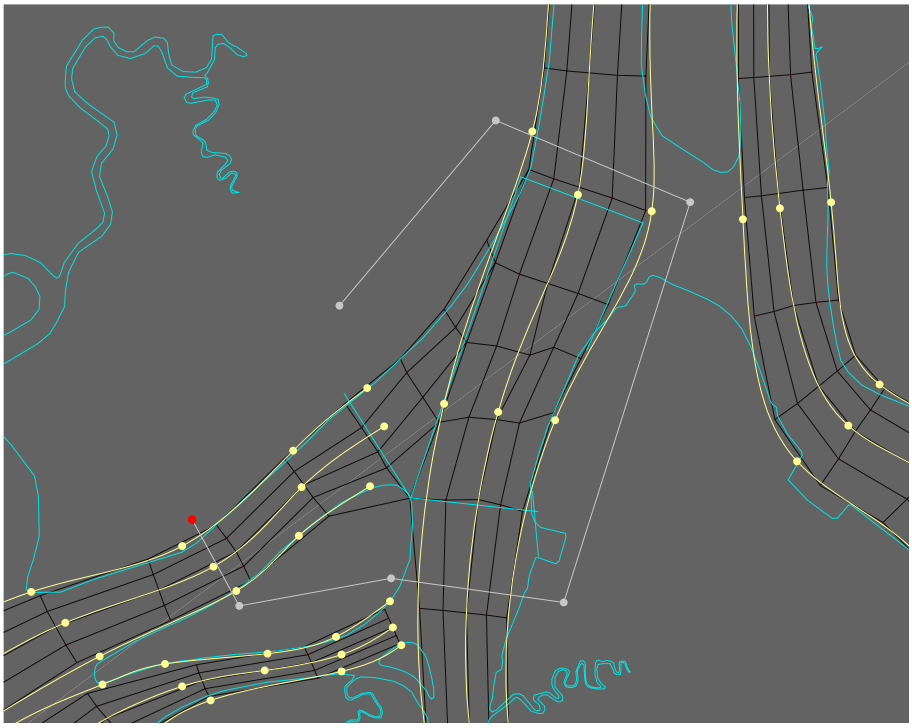


Figure 5.10: *Selecting polygon used for mesh orthogonalization/smoothing*

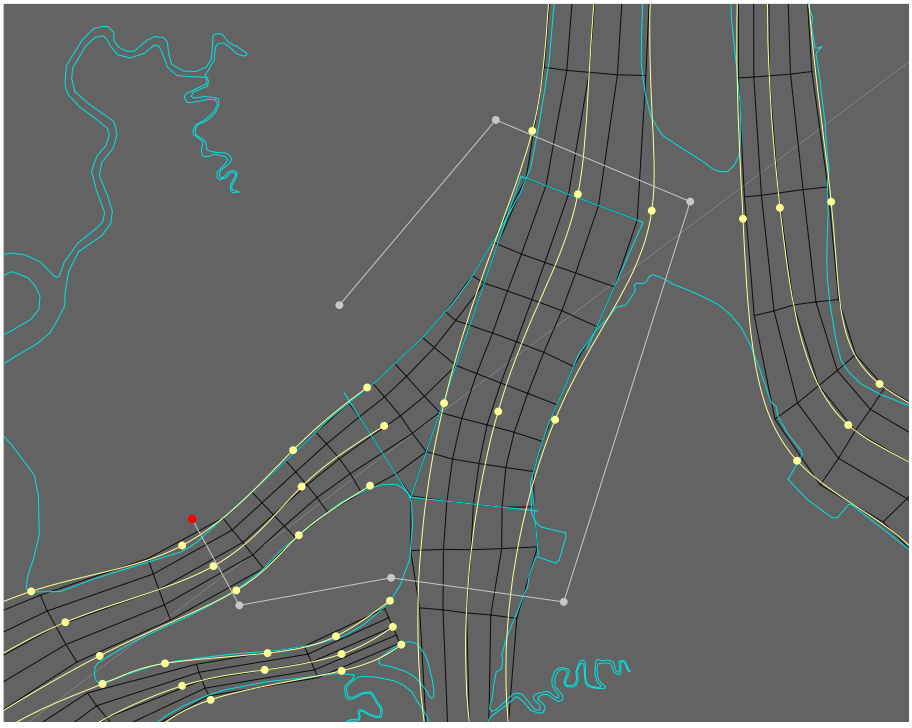


Figure 5.11: *Orthogonalized/smoothed mesh*

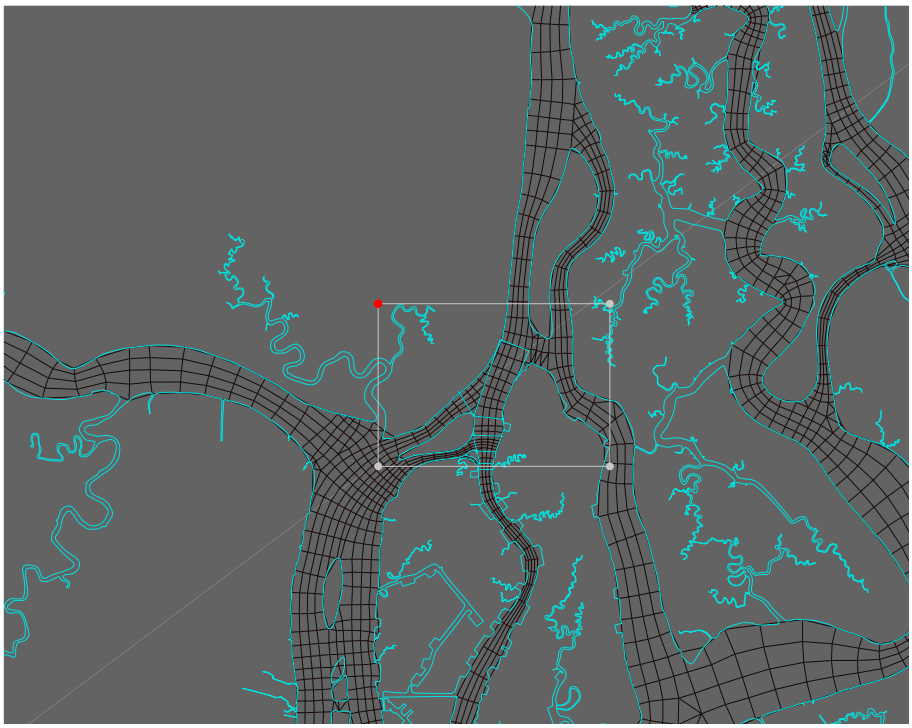


Figure 5.12: *Orthogonalized/smoothed whole mesh; the polygon marks the region we have worked on*

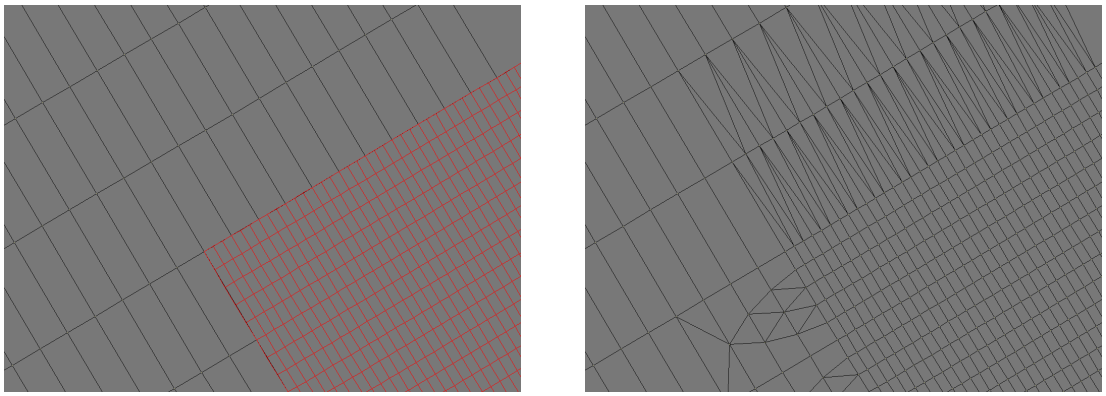


Figure 5.13: *Left: Loading curvilinear grids. Black lines are an already converted net, red lines are still a structured grid. Right: direct DD-couping results in bad aspect-ratio.*

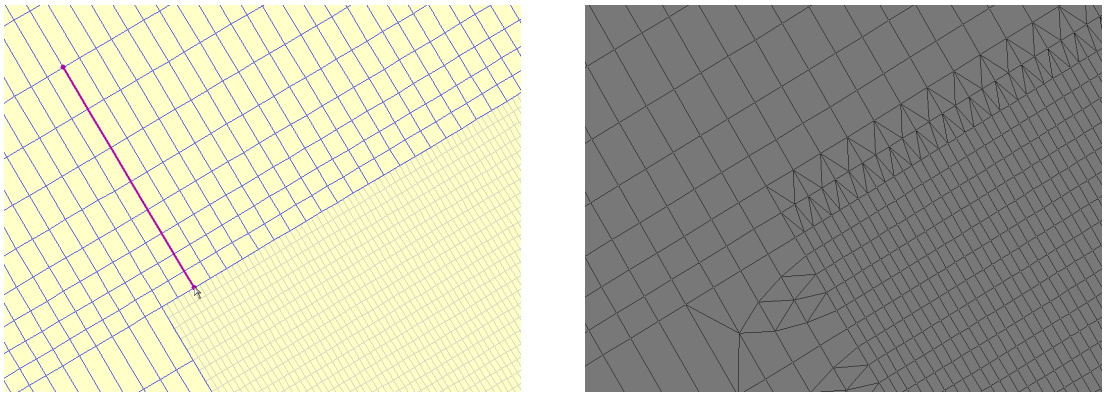


Figure 5.14: *Left: Coarse grid after two local refinement and one line smooth. Right: now the DD-couping results in good aspect-ratio.*

5.5.3 Improve results by preprocessing in RGFGRID

The DD-coupling operates along the boundary where the two grids meet. The resulting triangles can have aspect ratios several times larger than the original quads, which may be undesirable. See for example the right diagram in Figure 5.13. The solution is to perform some local refinement in RGFGRID in the direction *normal* to the grid boundary.

- ◇ For the coupling of two grids, open the coarse grid in RGFGRID through *File* → *Import* → *Grid*.
- ◇ Choose M- and N-refinement factors, e.g., 2 and 2, through *Settings* → *General...*
- ◇ Click *Edit* → *Grid* → *Refine Grid Locally*
- ◇ In the grid, click an arbitrary point on the DD-boundary.
- ◇ From this point, go in normal direction (along an M- or N-line) and click a second point one or more grid cells ahead.
- ◇ Optionally, repeat the last two steps for further refinement near the boundary.
- ◇ Click *Edit* → *Line* → *Smooth*, click a boundary point and in normal direction a second point sufficiently far ahead to smoothen the created refinement.
- ◇ Save the modified grid (*File* → *Export* → *Grid*) and load this file in D-Flow FM (*Files* → *Add curvilinear grid*).

Figure 5.14 shows the locally refined and line smoothened grid and the resulting DD-coupling with good aspect ratios.

5.6 Courant networks: automatic local refinement based on bathymetry

More information follows. Short instructions:

- ◇ Make sure to have a curvilinear Unstruc net loaded.
- ◇ Load bathymetry data (*Files* → *Load samples*)
- ◇ Set desired maximum timestep and CFL limit (*Various* → *Numerical parameters*)
- ◇ Have the Courant refinement automatically generated (*Operations* → *r*).

6 Parallel computing with D-Flow FM

6.1 Introduction

This chapter describes parallel computing with D-Flow FM based on the *Message Passing Interface* system (MPI). This can be run both on computing clusters with distributed memory as well as shared memory machines with multiple processors and/or multiple CPU cores. This chapter does not contain any further information on OpenMP-based parallelization.

Technical backgrounds on the parallel algorithms in D-Flow FM are described in the Technical Reference Manual D-Flow FM TRM (2015).

It is assumed here, that a properly compiled (Linux) MPI-version of D-Flow FM is at hand. Build instructions can be found on `http://publicwiki.deltares.nl/display/DFLOWFM/Building+on+Linux`.

Workflow of a parallel run

The goal of parallelization of D-Flow FM is twofold. We aim for faster computations on shared- or distributed-memory machines and the ability to model problems that do not fit on a single machine. To this end we partition the model and let separate processes solve the submodels that generate partitioned output. Given a whole model, the workflow is as follows:

1. partition the model (mesh and model definition file),
2. submit the parallel job to a queue on a computing cluster, and
3. visualize the results from partitioned output files.

6.2 Partitioning the model

In D-Flow FM a model is defined by the model definition file, the mesh file and external forcing/boundary condition files, et cetera. The latter are shared by all submodels and do *not* need to be partitioned, they should only be available to all processes. So, partitioning the model concerns partitioning of

- the mesh. This is achieved through the graphical user interface or by a command line option. Mesh files for every subdomain will be created, as well as one so-called partitioning polygon file that contains the subdomain bounding polygons;
- the model definition file. The partitioned model definition files will contain references to the subdomain mesh and the partitioning polygon file, all other information equals its sequential counterpart. The model definition files are partitioned with a script as will be explained later.

6.2.1 Partitioning the mesh

The mesh can be automatically partitioned with the METIS software package, see the Technical Reference Manual D-Flow FM TRM (2015) and references mentioned therein, or manually by supplying polygons that define the subdomains.

The mesh partitioning is available through the GUI or by the command line. The METIS partitioner or, alternatively, the user-supplied polygons, produce a cell coloring of the unpartitioned mesh. The coloring is used to decompose the mesh into subdomain meshes, which are augmented with ghost cells and saved to file. The coloring itself is not used directly in the parallel computations. Instead subdomain bounding polygons are used that are saved to file while partitioning the mesh. There exists only one such polygon file and we will refer to it as the *partitioning polygon* file. It is read by all parallel processes during the initialization of the

(parallel) computations.

The partitioning of the mesh will thus generate:

- ◇ subdomain mesh files, e.g. <example_NNNN_net.nc>, and
- ◇ the partitioning polygon file, e.g. <example_part.pol>.

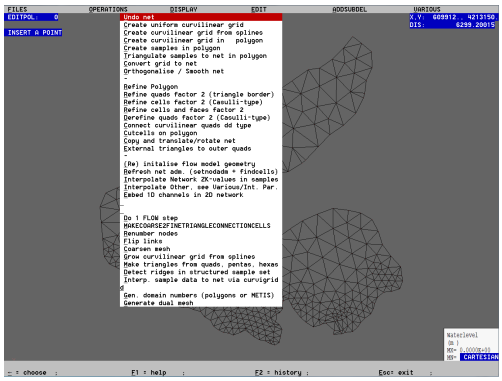
Partitioning the mesh with METIS using the GUI

We will firstly focus on the METIS partitioner. Partitioning the mesh with the graphical user interface is achieved by the following steps, see Figure 6.1 for an example:

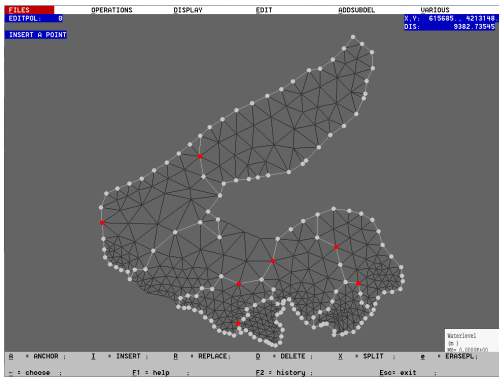
- ◇ load the mesh,
- ◇ generate the subdomain numbers with METIS by selecting *Operations* → *Generate domain numbers (polygon or METIS)*, see Figure 6.1a. You are prompted for
  - ▫ the number of subdomains,
  - ▫ contiguous domains or not. Contiguous domains are not necessary for parallel computations in D-Flow FM and often METIS produces contiguous subdomains without enforcing it. Still, some users may want to explicitly enforce contiguous domains. If you want to do so, make sure that your unpartitioned mesh is contiguous. Not being so may cause errors,
- ◇ inspect the partitioning polygons that are produced, see Figure 6.1b. Note that there is *no* polygon defining subdomain 0. Parts of the mesh not confined by any polygon are assumed to be in subdomain 0. In other words, there are at least $N-1$ polygons defining N subdomains.
- ◇ optionally, adapt the polygons to your liking by editing the polygons. In that case, regenerate the cell colors/domain numbers again by selecting *Operations* → *Generate domain numbers (polygon or METIS)*. Note that you will not be asked to specify the number of subdomains. The cell coloring/domain numbering is now based on the (modified) polygons and not produced by the METIS partitioner. Manual partitioning with user-specified polygons will be explained in the next section,
- ◇ optionally, visualize the cell domain numbers through *Display* → *values at net cells* → *partitioning info, domain number*, see Figure 6.1c,
- ◇ if satisfied, save the partitioning files through *Files* → *save partition files*. The entered mesh filename is a *basename* that will be used to derive the partitioning filenames, e.g. <example_net.nc> will produce:

  <example_part.pol>: partitioning polygon file
  <example_0000_net.nc>: meshfile of subdomain 0,
  . . .
  <example_NNNN_net.nc>: meshfile of subdomain 'NNNN',

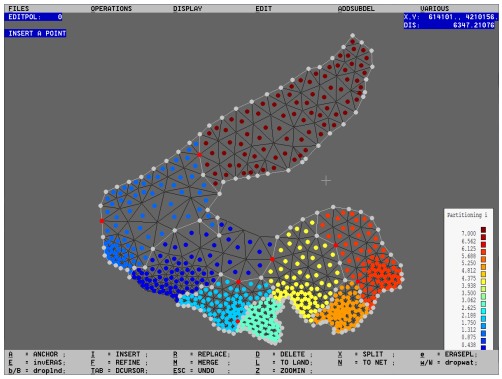
  with 'NNNN' referring to the number of subdomains minus one. Note that a meshfile with the specified basename <example_net.nc> will not be generated, it only serves to derive the partitioning file names.



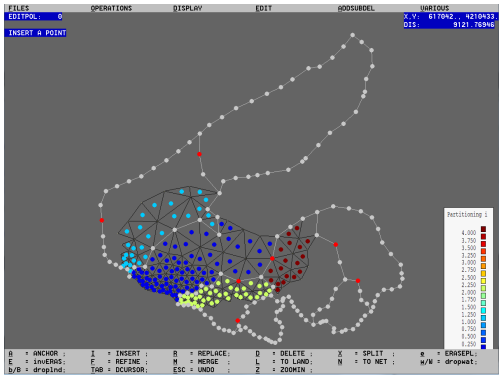
(a) *select* Operation → Generate domain numbers (polygon or METIS)



(b) *the subdomain bounding polygons*



(c) *the domain numbers*



(d) *mesh of subdomain 0 augmented with ghost cells*

Figure 6.1: *Partitioning the mesh with the GUI*

Partitioning the mesh manually

Alternatively, the mesh can be partitioned manually with user-supplied partitioning polygons. The partitioning obeys the following rules:

- ◇ if the polygons have a z-value specified, it is considered a subdomain number,
- ◇ if the polygons have no z-value specified, its order determines the corresponding subdomain number,
- ◇ if a cell is not inside at least one polygon, it is assigned to subdomain 0,
- ◇ if a cell is inside only one polygon, it is assigned to the subdomain defined by that polygon,
- ◇ if a cell is inside more than one polygon, it is assigned to the subdomain with the highest number.

In other words, the polygons may be overlapping and the largest subdomain number is taken in the overlapping regions. If the polygons have no z-value, the polygon order determines the corresponding subdomain number, i.e. the first polygon corresponds to subdomain 1 et cetera and there is no polygon defining subdomain 0.

Partitioning the mesh from the command line

Apart from using the graphical user interface, it is possible to perform the partitioning on the command line:

```
dflowfm --partition[:ndomains=n[:contiguous=j]] <meshfile> [polygonfile]
```

where `n` is the number of subdomains and `j` is 1 to enforce contiguous subdomains or 0 otherwise (default). If the number of subdomains is omitted, the polygons in the polygon file are used for manual partitioning. For the basename of the partitioning files the input mesh filename is used. For example, if we want to decompose the meshfile <example_net.nc> into 8 subdomain meshes and do not need contiguous domains:

```
dflowfm --partition:ndomains=8 example_net.nc
```

This will generate:

```
example_part.pol     example_0002_net.nc  example_0005_net.nc
example_0000_net.nc  example_0003_net.nc  example_0006_net.nc
example_0001_net.nc  example_0004_net.nc  example_0007_net.nc
```

The command to partition manually, based on a user-specified polygon file <userpols.pol>, is

```
dflowfm --partition example_net.nc userpols.pol
```

This will generate files with the same names as before.

6.2.2 Partitioning the MDU file

Having partitioned the mesh, the model definition file needs to be partitioned, as every submodel requires its own definition file with references to

- ◇ the partitioned mesh file, e.g. <example_0000_net.nc>,
- ◇ the partitioning polygon file, e.g. <example_part.pol>,
- ◇ an appropriate parallel Krylov solver, can be
  - 6: PETSc solver, recommended, or
  - 7: parallel CG with MILU block preconditioning,
- ◇ a unique snapshot directory, and
- ◇ optionally, a partitioned restart file.

The `generate_parallel_mdu.sh` script partitions a sequential model definition file automatically:

```
generate_parallel_mdu.sh <mdu-file> <nprocs> <partpol-file> <Icgsolver>
```

with

`mdu-file`:	sequential model definition file,
`nprocs`	number of subdomains/parallel processes,
`partpol-file`	partitioning polygon file,
`Icgsolver`	parallel Krylov solver, can be 6 or 7.

Note that the partitioned mesh filenames are derived from the mesh filename specified in the sequential model definition file. If the sequential model is never run, the sequential meshfile itself does not necessarily has to exist.

For example, if in the sequential mdu-file <example.mdu> contains a reference to meshfile <example_net.nc>, then the command

```
generate_parallel_mdu.sh example.mdu 8 example_part.pol 6
```

will generate eight partitioned mdu-files, namely <example_0000.mdu> to <example_0007.mdu> and the different entries in e.g. <example_0000.mdu> with respect to the sequential model definition file are

```
[geometry]
NetFile       = example_0000_net.nc
PartitionFile = example_part.pol

[numerics]
Icgsolver     = 6

[output]
SnapshotDir   = snapshots_0000
```

6.2.3 Remaining model input

A parallel run of a D-Flow FM model needs only partitioned <.mdu> and <_net.nc> files and a partitioning polygon file <_part.pol>. All other model input is the same as for a standalone run, e.g., meteo forcings, boundary conditions, observation stations and more. These are generally copied to the working directory by the parallel job submission script.

6.3 Running a parallel job

.. details differ per cluster.

.. short example submission script (lisa/gordon).

6.4 Visualizing the results of a parallel run

```
* history file (time series) is gathered in _0000_his.nc
* map files are partitioned in  _00dd_map.nc
* OpenEarthTools: dflowfm.plotMap, dflowfm.plotPartitions %\todo
```

7 Wind

7.1 Introduction

Various external influences can exert a force on the flow field. One of these influences is the wind. The force exerted by the wind is coupled to the flow equations as a shear stress. The wind field should be provided by means of an ascii-type file. This file should contain the grid on which the wind field is defined as well as the wind velocity vector(s).

D-Flow FM currently supports four types of wind field prescriptions[1], i.e. four grid types on which the wind field can be given. This wind grid does not need to be the same as the computational grid. The grid options to provide the wind data on are:

1. the computational grid – in this case, no specific wind grid is provided. The provided wind field is considered to be uniform over the entire model area. The wind field can be time dependent.
2. an equidistant grid – in this case, a wind field can be prescribed that varies both in space and in time. A Cartesian arcinfo-type grid should be provided on which the wind field is defined.
3. a curvilinear grid – this case is conceptually similar to the previous type (the equidistant grid) in the sense that a wind field can be imposed that both varies in space and time. However, a separate file should be provided in which a curvilinear grid is defined (a classic `.grd`-type file as known from Delft3D) on which the wind field is defined.
4. a spiderweb grid – this type of wind specification is specially devoted to cyclone winds and is only available in combination with computational grids that are of spherical type. In this case, a cyclone wind field is given on a polar grid with the center ('eye') of the cyclone being the origin of the polar coordinate system. The location of this eye and the associated wind field usually varies in time.

Each of these filetypes can be assigned through the entry in the external forcings file (the `.ext`-file) named `FILETYPE`. In this chapter, the various types of wind field specifications are highlighted subsequently. Each of the options is illustrated by means of an example.

7.2 Specification of wind on the computational grid

In D-Flow FM, the specification of the wind on the computational grid is equivalent to the specification of a uniform wind, since no separate wind grid is provided to the model. The specification of a uniform wind field can be done in two ways:

1. componentwise: as velocity in x-direction (m/s) and in y-direction (m/s) (cf. Figure 7.1 for a definition sketch) – the associated `FILETYPE` in the external forcings file is depicted as `uniform`, which has `FILETYPE=1`.
2. by magnitude (m/s) and direction (degN) – the associated `FILETYPE` in the external forcings file is depicted as `unimagdir`, which has `FILETYPE=2`.

These two types are treated below separately.

[1] The description of the wind formats, given in this chapter, is compatible with D-Flow FM version `1.1.90.31666`. Currently, the wind functionalities module is being replaced by the more general EC-module which is intended to not only support D-Flow FM, but also Delft3D and SOBEK. As a result, parts of this chapter might not be compatible with later versions of D-Flow FM that run on this new EC-module.

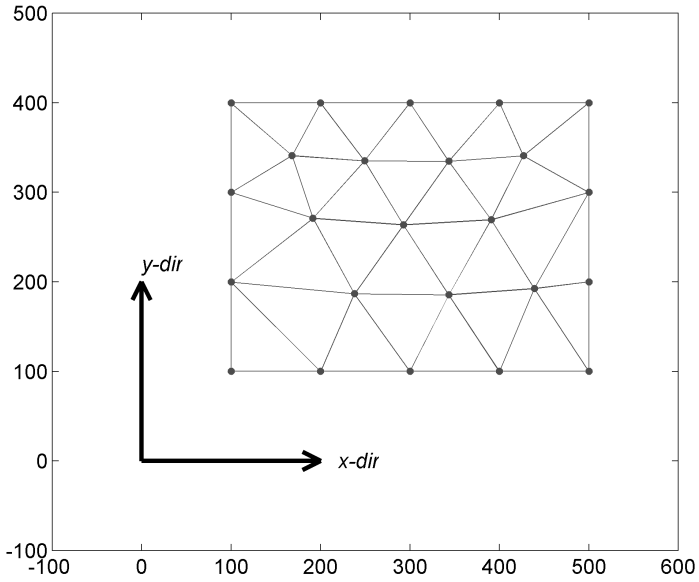


Figure 7.1: *Example grid with definitions of the wind directions (distances in meters).*

7.2.1 Specification of uniform wind through velocity components

Since no particular wind grid is used, only timeseries for the x-component and the y-component of the wind need to be specified. The specification of these timeseries can be done separately (one single file for the x-component and one single file for the y-component) or jointly (one single file containing the x-component and the y-component of the wind).

Uniform wind should be provided as an `.wnd`-file containing either 2 colums (in case of separate specification of the x-component and y-component of the wind) or 3 columns (in case of joint specification of the velocity components). In either case, the first column contains the time in minutes with respect to the overall reference time.

Example

As an example, a uniform wind field is applied to a model with a computational grid shown in Figure 7.1. The uniform wind is provided in a file named `windxdirydir.wnd`. The contents of this wind file are:

```
 0.00000   10.00000   10.00000
60.00000  -10.00000  -10.00000
```

The first column denotes the time in minutes with respect to the reference date (specified in the MDU-file). The second column denotes the wind velocity in x-direction, whereas the third column denotes the wind velocity in y-direction; both wind components are provided in one single file.

The connection with the flow model itself is laid through the external forcings file. The actual specification of the wind is in this case:

```
QUANTITY =windxy
FILENAME =windxdirydir.wnd
FILETYPE =1
METHOD   =1
OPERAND  =O
```

Since the two components are given in one single file, the `QUANTITY` is set to `windxy`. If two separate files would have been provided, the `QUANTITY` would have been set to `windx` and `windy` over two separate datablocks in the external forcings file.

7.2.2 Specification of uniform wind through magnitude and direction

Instead of specifying the separate components of the wind field, the uniform wind vector can also be prescribed through its magnitude and direction. This kind of specification should be done by means of one single file, containing three columns, representing the time (in minutes with respect to the reference date), the velocity magnitude (in m/s, not necessarily positive) and the wind direction (in degN).

Example

As an example, the previous uniform wind case is reformulated as a case with magnitude and direction of the wind field prescribed. The unimagdir wind is provided in a file named `windinput.wnd`. The contents of this file are:

```
 0.00000   14.14213562373095  225.00000
60.00000  -14.14213562373095  225.00000
```

The first column denotes the time in minutes with respect to the reference date (specified in the MDU-file). The second column denotes the wind velocity magnitude, whereas the third column denotes the wind direction. Note that there is a clear difference between the above case and a case in which the magnitude is kept positive (14.1421 m/s) and the direction varies (and hence *rotates*!) from 225 degN to 45 degN.

The connection with the flow model itself is laid through the external forcings file. The actual specification of the wind is in this case:

```
QUANTITY =windxy
FILENAME =windinput.wnd
FILETYPE =2
METHOD   =1
OPERAND  =O
```

7.3 Specification of wind on an equidistant grid

The vector components of the velocity vectors can also be specified on a distinct grid, either of equidistant type or of curvilinear type. In both cases, the characteristics of the grid should be provided. In case of an equidistant grid, the grid is specified in arcinfo-style. That means,

the constant grid sizes dx and dy should be specified such that a grid is spanned with respect to the location of the lower left corner of the grid (either the center of the lower left cel or the lower left corner of the lower left cell).

Example

As an example, a grid with $dx = dy = 100$ m is spanned, based on the center of the lower left cell, located at $x = y = 60$ m with respect to the origin (cf. Figure 7.1). The input data for the x-component and the y-component should be specified separately, in two distinct files. The input of the x-component data should be given in an `.amu`-type file, such as `windxdir.amu` as an example:

```
### START OF HEADER
### This file is created by Deltares
### Additional commments
FileVersion      =    1.03
filetype         =    meteo_on_equidistant_grid
NODATA_value     =    -9999.0
n_cols           =    5
n_rows           =    4
grid_unit        =    m
x_llcenter       =     60
y_llcenter       =     60
dx               =    110
dy               =    110
n_quantity       =    1
quantity1        =    x_wind
unit1            =    m s-1
### END OF HEADER
TIME = 0 hours since 2006-01-01 00:00:00 +00:00
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
TIME = 1 hours since 2006-01-01 00:00:00 +00:00
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
```

For the y-component data, a similar file (e.g. `windydir.amv`) should be provided. In addition, the pressure could be specified in a similar file (e.g. `pressure.amp`). Note that `x_llcorner` and `y_llcorner`, instead of `x_llcenter` and `y_llcenter`, are also supported.

Wind on an equidistant grid has been provided a filetype specification as `FILETYPE=4`. The connection with the flow model itself is laid through the external forcings file. The actual specification of the wind is in this case:

```
QUANTITY =windx
FILENAME =windxdir.amu
FILETYPE =4
METHOD   =2
OPERAND  =O

QUANTITY =windy
```

```
FILENAME =windydir.amv
FILETYPE =4
METHOD   =2
OPERAND  =O

QUANTITY =atmosphericpressure
FILENAME =pressure.amp
FILETYPE =4
METHOD   =2
OPERAND  =O
```

7.4 Specification of wind on a curvilinear grid

In analogy with the wind specification on an equidistant grid, the wind can be specified on a curvilinear grid. This curvilinear grid should be provided as a classic `.grd`-file as known from Delft3D. A difference with the equidistant grid wind is the necessity to compile all data blocks (i.e. x-component, y-component and pressure) in one single file. This file should have the extension `.apwxwy`. The sequence of this datablock is: pressure – x-velocity component – y-velocity component.

Example

As an example, a curvilinear grid named `meteo.grd` is present, providing the underlying coordinates of the wind data field. The input data, comprising the atmospheric pressure, the x-velocity component *and* the y-velocity component, are given in one single file (as is compulsory). The contents of the example `meteo.apwxwy`-file is:

```
### START OF HEADER
### This file is created by Deltares
### Additional commments
FileVersion      =    1.03
filetype         =    meteo_on_curvilinear_grid
NODATA_value     =    -9999.0
grid_file        =    meteo.grd
first_data_value =    grid_llcorner
data_row         =    grid_row
n_quantity       =    3
quantity1        =    apwxwy
unit1            =    Pa
### END OF HEADER
TIME = 0.0 hours since 2006-01-01 00:00:00 +00:00
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
10 10 10 10 10
TIME = 1.0 hours since 2006-01-01 00:00:00 +00:00
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
101325 101325 101325 101325 101325
-10 -10 -10 -10 -10
```

```
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
-10 -10 -10 -10 -10
```

Note that `grid_llcenter`, instead of `grid_llcorner`, is also supported. On the contrary, `grid_column` is *not* supported instead of `grid_row`.

Wind on a curvilinear grid has been provided a filetype specification as `FILETYPE=6`. The connection with the flow model itself is laid through the external forcings file. The actual specification of the wind is in this case:

```
QUANTITY =airpressure_windx_windy
FILENAME =meteo.apwxwy
FILETYPE =6
METHOD   =3
OPERAND  =O
```

Notice that `METHOD=3` is chosen for wind on a curvilinear grid, instead of `METHOD=2` in case of wind on an equidistant grid.

7.5 Specification of a cyclone wind on a spiderweb grid

Cyclone winds can be imposed by means of a polar grid that is spanned around the cyclone eye. In addition to the cyclone eye, quite probably varying in both space and time, the number of rows (discretisation in radial direction) and the number of columns (discretisation in angular direction) should be given, as well as the radius of the grid (in meters). The definitions of the cylcone wind grid (also depicted as spiderweb grid) is illustrated in Figure 7.2.

Cyclone winds can only be used in combination with a spherical computational grid. The location of the cyclone eye should be given as longitude (for x_{eye}) and latitude (for y_{eye}). At the eye the pressure *drop* should be prescribed. This pressure drop is taken relative to the atmospheric pressure as prescribed in the MDU-file. The extension of the spiderweb grid file is `.spw`. The contents of the spiderweb wind file should comprise the local atmospheric pressure drop, the wind velocity magnitude and the wind direction.

Example

As an example, a spiderweb grid named `spwsimple.spw` is present, providing the underlying coordinates of the wind data field. The input data, comprising the atmospheric pressure drops, the wind velocity magnitudes (in m/s) *and* the wind directions (in degN), are given in one single file (as is compulsory). The contents of the example `spwsimple.spw`-file is:

```
### Spiders web derived from TRACK file: gonu.trk
### This file is created by Deltares
### All text on a line behind the first # is parsed as commentary
### Additional commments
```

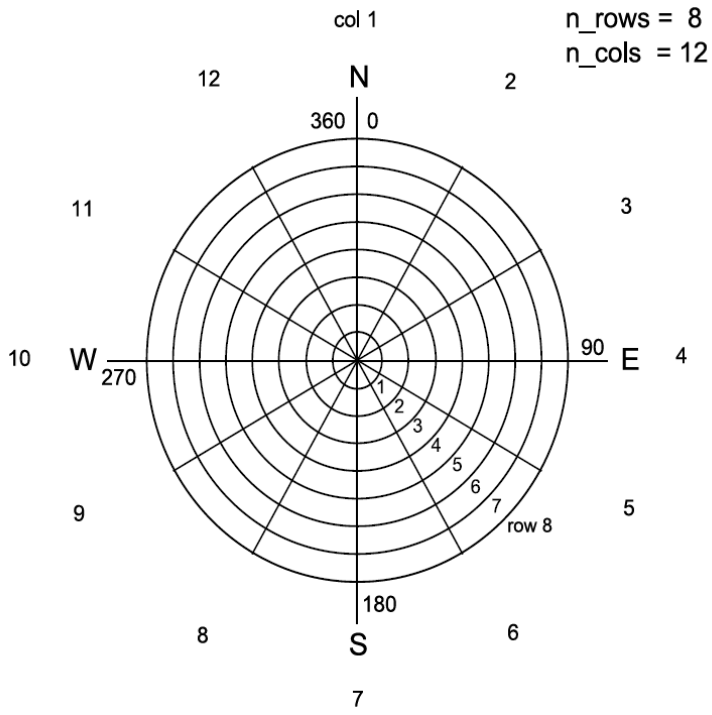


Figure 7.2: *Grid definition of the spiderweb grid for cyclone winds.*

```
FileVersion      = 1.03
filetype         = meteo_on_spiderweb_grid
### Spiders web derived from TRACK file: gonu.trk
### This file is created by Deltares
### All text on a line behind the first # is parsed as commentary
### Additional commments
NODATA_value     = -1001
n_cols           = 4
n_rows           = 4
grid_unit        = degree
spw_radius       = 600000.0
spw_rad_unit     = m
n_quantity       = 3
quantity1        = wind_speed
quantity2        = wind_from_direction
quantity3        = p_drop
unit1            = m s-1
unit2            = degree
unit3            = Pa
### END OF HEADER
TIME             = 340000.00   minutes since 2005-01-01 00:00:00 +00:00
x_spw_eye        =  265.00
y_spw_eye        =   33.00
pdrop_spw_eye    = 7000.000
  5.000000  5.000000  5.000000  5.000000
 10.000000 10.000000 10.000000 10.000000
 15.000000 15.000000 15.000000 15.000000
 20.000000 20.000000 20.000000 20.000000
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
   4000.00   4000.00   4000.00   4000.00
   3000.00   3000.00   3000.00   3000.00
   2000.00   2000.00   2000.00   2000.00
```

```
   1000.00   1000.00   1000.00   1000.00
TIME             = 380000.00   minutes since 2005-01-01 00:00:00 +00:00
x_spw_eye        =  275.00
y_spw_eye        =   18.00
pdrop_spw_eye    = 8000.000
  5.000000  5.000000  5.000000  5.000000
 10.000000 10.000000 10.000000 10.000000
 15.000000 15.000000 15.000000 15.000000
 20.000000 20.000000 20.000000 20.000000
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
    270.00      0.00     90.00    180.00
   4000.00   4000.00   4000.00   4000.00
   3000.00   3000.00   3000.00   3000.00
   2000.00   2000.00   2000.00   2000.00
   1000.00   1000.00   1000.00   1000.00
```

Note that in the header of the file, only the entry `unit3` could be chosen freely, i.e. either the unit Pa or the unit mbar could be chosen. The other entries are frozen and hence not available for free choices.

Wind on a spiderweb grid has been provided a filetype specification as `FILETYPE=5`. The connection with the flow model itself is laid through the external forcings file. The actual specification of the wind is in this case:

```
QUANTITY =windx_windy_airpressure
FILENAME =spwsimple.spw
FILETYPE =5
METHOD   =1
OPERAND  =O
```

Notice that `METHOD=1` is chosen for wind on a spiderweb grid, instead of `METHOD=2` in case of wind on an equidistant grid and `METHOD=3` in case of wind on a curvilinear grid.

7.6 Combination of several wind specifications

The combination of the various wind specification types can *only* be achieved if the `QUANTITY` of the winds to be combined is the same, for instance `QUANTITY=windx`. That means that the combination of a uniform wind with a cyclone cannot be combined, at the moment. The option `OPERAND=+` can be used to add a wind field to an existing wind field.

Example

If the uniform wind is to be combined with a wind specified on an equidistant grid, then the wind field could be assigned in the external forcings file as follows:

```
QUANTITY =windx
FILENAME =windxdir.wnd
FILETYPE =1
METHOD   =1
OPERAND  =O
```

```
QUANTITY =windy
FILENAME =windydir.wnd
FILETYPE =1
METHOD   =1
OPERAND  =O

QUANTITY =windx
FILENAME =windxdir.amu
FILETYPE =4
METHOD   =2
OPERAND  =+

QUANTITY =windy
FILENAME =windydir.amv
FILETYPE =4
METHOD   =2
OPERAND  =+
```

7.7 Masking of points in the wind grid from interpolation ('land-sea mask')

A mask can be supplied by the user to prevent selected points in the wind grid from contributing to the wind interpolation on velocity points, e.g. to exclude land points. This feature was included to conform to SIMONA and therefore implemented in the same way.

For each individual grid point for which to interpolate from the wind grid:

- ◇ Masked wind points are excluded from the interpolation.
- ◇ The total of the weight factors for the remaining wind points is determined.
- ◇ If this total falls below `1E-03`, the mask is ignored and the original bilinear weights are used.
- ◇ Otherwise, the weights for the remaining wind points are normalised again.

The effect of the mask, when applied as a land-sea mask, is that for velocity points close to shore the interpolated wind is no longer influenced by the wind over land (which would otherwise yield a zone of points with reduced wind near the shore).

Specification and format of the mask file

The name of the mask file, if any, is specified in the `.EXT` file, labelled `SOURCEMASK`, directly following the `FILENAME` specification, e.g.:

```
QUANTITY   =windxy
FILENAME   =meteo.wxwy
SOURCEMASK =meteo_mask.asc
FILETYPE   =6
METHOD     =3
OPERAND    =O
```

The mask file itself has the same layout as the wind file, though the number of required header fields is reduced, e.g.:

```
FileVersion      =    1.03
```

```
.
.
unit1             =    Pa
### END OF HEADER
1       1       1       1       1
1       1       1       0       0
1       1       1       0       0
1       1       0       0       0
```

The lines in the header are ignored. The number of columns and rows in the matrix of ones and zeroes should match those of a block (for a single variable and a single timestep) in the meteo files. Zeroes signify the position of rejected points (and ones those of the accepted points) in the wind grid.

8 Calibration and data assimilation

8.1 Introduction

A flow model in D-Flow FM will generally need *calibration* of its parameters to closely approximate observations. When the model also runs in an operational system, it will generally benefit from *data assimilation*: the incorporation of observations into the running model. For the automatic calibration and data assimilation, the open source toolbox **OpenDA** is available.

OpenDA basically comprises two elements: an optimization algorithm that actually performs the calibration or Kalman filtering, and communication routines for parsing information between OpenDA and D-Flow FM. For the communication between OpenDA and D-Flow FM (i.e. reading and writing of input-files and output-files), a so-called fully Black Box model wrapper, written in Java, for D-Flow FM has been realized.

This chapter contains a description of how the OpenDA toolbox could be deployed to apply automatic calibration and data assimilation. The OpenDA tools can run D-Flow FM models and analyze the model results. A more extensive design description of the D-Flow FM wrappers for OpenDA can be found in the D-Flow FM Technical Reference D-Flow FM TRM (2015). More information on OpenDA can be found on the website `http://www.openda.org`.

General information on the installation of OpenDA to get started is provided in section 8.2. section 8.3 described how a certain model can be configurated. Examples case for automized calibration and ensemble Kalman filtering are described in section 8.4.

8.2 Getting started with OpenDA

The required D-Flow FM wrapper is currently not part yet of the standard OpenDA release. The necessary files for D-Flow FM can be found on the D-Flow FM wiki-homepage:

`http://publicwiki.deltares.nl/display/DFLOWFM/D-Flow+Flexible+Mesh`

The following four elements (all available on the wiki) are needed for a calibration or data assimilation run with D-Flow FM:

1. A D-Flow FM installation. All OpenDA algorithms start D-Flow FM with a shell script `start_unstruc.sh` or a batch script `start_unstruc.bat`. Both scripts assume that D-Flow FM is installed somewhere on your system. The `.sh`-script expects an environment variable `$DFLOWMROOT$`. The `.bat`-script assumes that D-Flow FM can be started from the commandline. If you follow the instruction steps for installing D-Flow FM as provided on the wiki, this should be the case. Alternatively, you could also define a new environment variable `PATH` that directly points to the directory in which the executable of D-Flow FM is located.
2. An OpenDA installation including the wrapper code for D-Flow FM. For this, use the <deltares_bin_dddd.zip> provided at the D-Flow FM homepage, or download a fresh build from Teamcity: Configuration '*openda trunk, win32-2-Build Java*', artifact '*deltares_bin/*. Extract (only) the directory <deltares_bin>, rename it to <bin/>.
3. A JRE environment 1.6 (or higher) is neede to run OpenDA. It is recommended to use the JRE environment as provided on the D-Flow FM homepage. If you put the JRE-directory directly next to the bin-directory of the previous step, OpenDA will automatically use this JRE-environment. That means: the directory <bin/> and the directory <jre/> are in the same directory.
4. Some testcases: the sample configuration files that are described in the next section. The

.zip-file contains a directory <tests> with 5 different subdirectories (in general referred to as 'case' hereafter). They may be placed anywhere on your system, but make sure that the name of the installation path does not contain spaces.

8.3 Model configuration

For the configuration of an OpenDA run (either calibration or Kalman filtering), three different types of files need to be paid attention on:

- ◇ the OpenDA algorithm files with extension .oda,
- ◇ the wrapper files for the exchange items with extension .xml,
- ◇ the available observation files, provided in NOOS-format.

These three types of files are addressed subsequently in this section.

8.3.1 The algorithm files

If not stated differently, all files and directories mentioned in this section are relative to the directory <case>. In each <case> directory you will find main input files for OpenDA (xml files with extension .oda). You should open one of these files in the OpenDA-gui to start a run.

Almost all filenames and filedirectories relative to <case> are configurable through the main input file. However, the testcases have been configured identical as much as possible so, for the ease of reading, we will refer to the situation as it was configured.

The name of the .oda file corresponds to the algorithms executed by OpenDA:

- ◇ Simulation.oda: performs a regular D-Flow FM simulation run, only the executable is kicked off by OpenDA. This algorithm is useful to check the configuration. Output is always found in <./stochModel/work0>.
- ◇ Dud.oda: Dud (Doesn't Use Derivative) is one of the optimization algorithms available for calibration purposes. Output is found in <./stochModel/work*>, <work0> and <work1> are needed for initialisation purposes, <work2> and higher contain the results for succesive calibration runs.
- ◇ SequentialSimulation.oda: performs a D-Flow FM simulation run through which the executable is stopped and restarted by OpenDA at the moments for which observed data are available. For all cases, the configuration of the algorithm (through the file algorithm/SequentialSimulation.xml) is such that no stochastic noise is added to the model. Thus, the results of this run (found in ./stochModel/work0) compared to those of a regular run give an indication of the exactness of the restart functionality of D-Flow FM for this particular testcase.
- ◇ EnFK.oda: performs an EnKF simulation. Configuration of the algorithm, such as the number of ensembles N used, is found in ./algorithm/EnkfAlgorithm.xml. Results are found in <./stochModel/work1> up to <./stochModel/work$N$>. Directory <work0> is needed for initialisation purposes.

All .oda-files contain a line:

```
<timingSettings doTiming="true"></timingSettings>
```

With this option switched on, a timing report will be created at the end of a run to monitor the performance of the run.

8.3.2 The wrapper files

The wrapper code implements so-called ExchangeItems that allow OpenDA to manipulate specific parts of the input files of D-Flow FM depending on information read from the output files of D-Flow FM. In case of a calibration experiment, the output of run n is used to prepare the input for the next run $n + 1$, until some convergence criterion for the calibration process is met. In case of an Ensemble Kalman Filtering (EnKF) run, the D-Flow FM computations (one run for each ensemble member) is stopped each time an observation is available, the input files for each ensemble are modified according to the EnKF-algorithm, after which the D-Flow FM run is restarted.

The black box wrapper configuration usually consists of three `.xml` files. For D-Flow FM, these files can be found in directory <case/stochModel>. These are called:

- ◇ `dflowfmWrapper.xml`: this file specifies the actions to perform in order to run a D-Flow FM simulation and the files plus related readers and writers that can be used to let OpenDA interact with the model.
- ◇ `dflowfmModel.xml`: the specification of the deterministic part of the model. All available ExchangeItems are listed here.
- ◇ `dflowfmStochModel.xml`: the specification of the stochastic part of a D-Flow FM model. This file selects the ExchangeItems that may be manipulated by OpenDA in a specific configuration, it specifies the noise model used and the predictions, i.e. the values on observation locations as computed by D-Flow FM.

The D-Flow FM files that can be manipulated and the corresponding OpenDA class names to be used in the `dflowfmWrapper.xml` file are given in Table 8.1. There is no wrapper code

Table 8.1: *D-Flow FM files that can be manipulated and the corresponding OpenDA class names to be used in the `dflowfmWrapper.xml` file.*

D-Flow FM filetype	OpenDA className
`.xyz`	`org.openda.model_dflowfm.DFlowFMFricionCoefficientFile`
`.mdu`	`org.openda.model_dflowfm.DFlowFMTimeInfo`
`_his.nc`	`org.openda.exchange.dataobjects.NetcdfDataObject`
`_map.nc`	`org.openda.model_dflowfm.DFlowFMRestartFileWrapper`
`.tim`	`org.openda.model_dflowfm.DFlowFMTimeSeriesDataObject`

available for meteo files (wind fields) yet. Within OpenDA, ExchangeItems are identified by their ExchangeItemID. This ID is constructed from the information available in the D-Flow FM input directory. Possible ExchangeItems are:

- ◇ for the `.xyz`-file: sample files (e.g. the bed friction coefficients) are specified via the external forcings file. OpenDA expects a single file with an arbitrary number of lines containing three columns, separated by white spaces. For instance,

```
157671.5781250  428974.4062500  0.9994357525438934
156082.3906250  429480.0625000  0.9994357525438934
163667.1562500  434933.8750000  2.0021214673505600
```

  The first two columns represent (x, y)-coordinates and are neglected by OpenDA. The third column contains a real value that is used by OpenDA in two ways:

1 when initializing OpenDA, this value is used to group the lines. Lines with the same label belong to a polygon that circumvents a region with constant bed friction coefficient. Each group of lines corresponds with one exchange item,
2 during the callibration algorithm, this column is used to exchange the value of the bed friction coefficient between OpenDA and D-Flow FM.

◇ from the `.mdu`-file: `org.openda.model_dflowfm.DFlowFMTimeInfoExchangeItem`. Also see Table 8.2.

Table 8.2: *Possible ExchangeItems in the D-Flow FM MDU-file to OpenDA.*

D-Flow FM reference	OpenDA ID	remark
`TStart`	`start_time`	`RefDate` and `Tunit` needed for interpretation
`TStop`	`end_time`	`RefDate` and `Tunit` needed for interpretation

◇ for the `_his.nc`-file: `org.openda.exchange.NetcdfScalarTimeSeriesExchangeItem`. A `NetcdfDataObject` expects a NetCDF-file that contains dimensions `time` and `stations` plus a variable `station_id` of type string and dimension `stations`. For each variabele in this NetCDF-file with dimensions `(time,stations)` an exchange item is created, that can be referred to as `station_id(i).variablename`. For instance:

```
dimensions:
time = UNLIMITED ; // (2882 currently)
stations = 3 ;
station_name_len = 40 ;
variables:
char station_id(stations, station_name_len) ;
(containing strings Obs1, Obs2, Obs3)
double waterlevel(time, stations) ;
(containing the computed values of the waterlevel)
```

results in three exchange items (1D vector in this case) with ID's: `Obs1.waterlevel`, `Obs2.waterlevel` and `Obs3.waterlevel`.

◇ for the `_map.nc`-file: `org.openda.model_dflowfm.DFlowFMExchangeItem`. This file contains all information needed to restart a D-Flow FM computation. Not all variables are relevant for manipulation by OpenDA: variable names that start with "time", "NetLink", "BndLink", "FlowLink", "NetElem", "FlowElem", "NetNode", "wgs84" and "projected_coordinate_system" are ignored. Variables of other types than float or double are also ignored. For all remaining variables, an ExchangeItem is created. The name of the variable as used in the NetCDF is used as ExchangeItemID.

◇ for the `.tim`-files: `org.openda.exchange.timeseries.TimeseriesSet`. Creating an ExchangeItem for a `.tim`-file starts with reading the name of the external forcing file from the `.mdu`-file (key `ExtForceFile`). The `.ext`-file contains formatted blocks, one for each forcing. Forcings are defined along polylines, given in `.pli`-files. A `.pli`-file is accompanied by a `.cmp`- or a (number of) `.tim`-file(s). Noise is added by means of an extra block in the `.ext`-file. As an example, noise is added to a boundary with a discharge imposed as:

```
QUANTITY =dischargebnd
FILENAME =sw_east_dis.pli
FILETYPE =9
METHOD   =3
OPERAND  =O

QUANTITY =dischargebnd
FILENAME =sw_east_dis_noise.pli
```

```
FILETYPE =9
METHOD   =3
OPERAND  =+
```

The discharge is set by the first block (operand=O), the information in the .pli-files is identical and noise is added as a timeseries: the _noise.pli file is always accompanied by a (number of) .tim file(s). The location-information on the first line of the .pli-file combined with the quantity is used to construct an identifier: location.1.dischargebnd. The numbering is used to discern between multiple tim-files possibly linked to a single _noise.pli-file.

8.3.3 The observation files

Observations in all testcases are configured to be of NOOS-format and located in directory <./StochObserver>. All observations were extracted from an earlier D-Flow FM run. Java unit test DFlowFMRestartTest was used to convert the timeseries in a _his.nc file to a number of files (one for each observation location) in NOOS format. An example of (a part of) a NOOS file is:

```
#-------------------------------------------------------
#-------------------------------------------------------
# Location : station01
# Position : (0.0,0.0)
# Source : twin experiment DFlowFM
# Unit : waterlevel
# Analyse time: null
# Timezone : null
#-------------------------------------------------------
199101010000   1.0000
199101010100   0.8944
199101010200   0.6862
199101010300   0.5956
199101010400   0.3794
199101010500   0.1372
199101010600   -0.1300
199101010700   -0.3044
199101010800   -0.3963
199101010900   -0.3739
199101011000   -0.1930
...
```

8.4 Examples of the application of OpenDA for D-Flow FM

In this section, some examples are elaborated for both the calibration of a model and the ensemble Kalman filtering (abbreviated as 'EnKF') of a model.

8.4.1 Example 1: Calibration of the roughness parameter

The automatic calibration of a model needs two main choices from the user:

1 Which model parameters may be modified during the calibration process?
2 Which model results need to be compared to observations, to judge the model quality?

The remainder of this section will be in the form of a tutorial, to directly illustrate all steps in an example model. In this example you will use a small river model 'simple_waal' and use the bed roughness to calibrate this model for its three water level observation stations.

Step 1: Inspect the model

All the required model files can be found in the directory <exercices/tutorial17/>. Consider the following steps:

1 Navigate to <exercises/tutorial17/> and place your executable <unstruc.exe> in the subdirectory <simple_waal/stochModel/bin/>.
2 Start D-Flow FM (standalone) in directory <input_dflowfm/>.
3 Select *Files* → *Load MDU-file*.
4 Load the model: select <simple_waal.mdu>.
5 You can run the model if you like (right mouse button).

The basic model is built to simulate a simple two-dimensional river with a spatially varying bed friction coefficient. It is driven by two boundary conditions: an upstream discharge inflow at the eastern boundary and a downstream water level at the western boundary. Inspect the model forcing in the following way:

1 Open the external forcings file <simple_waal.ext> in a text editor.
2 Notice how, in addition to the two boundaries, there are two blocks for the frictioncoefficient. The first one is a spatially varying roughness field in a sample <∗.xyz> file. The second refers to a small sample file that will act as a multiplication factor for the original friction coefficients.
3 In D-Flow FM, select *Files* → *Load sample file* and select <sw_frcfact_all.xyz>.
4 Notice how the loaded samples have three distinct values 1, 2 and 3, which act as identifiers: they approximately define the corner points of three subdomains of the entire river stretch. For each subdomain, a different roughness can be calibrated.

Step 2: Select the model parameters

Currently, the only calibratable parameters are the time-independent parameters in the external forcings file that use the `.xyz` sample file format. The most obvious parameter is the bed friction coefficient.

Step 3: Select the model results

The example directory <simple_waal> contains all the necessary configuration files for the so-called Black Box model wrapper for D-Flow FM to run a "twin experiment". In a twin experiment a model setup with given solution (the synthetic observations) is perturbed after which OpenDA is applied to re-estimate the original settings. The effects of the parameter variations may be judged by comparing time series output in the `_his.nc` history file to observed data in NOOS time series format.

The D-Flow FM model simulates a 1D river flow. The input files for D-Flow FM are located in the directory <simple_waal/stochModel/input>. D-Flow FM allows the user to specify regions with a different bed friction coefficient (constant for each region) and is able to handle the interpolation of the coefficients between these regions. In the experiment we try to re-estimate the values of the bed friction coefficients of an earlier run. As observations, the waterlevel at three locations (stations) along the river is used. These results are written to the main output file (<∗_his.nc>) as timeseries.

In the directory <simple_waal>, there are two "main configuration files" of OpenDA present:

- `Simulation.oda`: runs a single run of the model, this configuration is mainly used to test the black box configuration files.
- `Dud.oda` runs a calibration experiment with algorithm DUD (Doesn't Use Derivative).

These files configure the main ingredients of an OpenDA run:

1. the stochObserver (`org.openda.observers.NoosTimeSeriesStochObserver`). Observations for this experiment were created by extracting the timeseries for all stations from the netcdf-file and convert them to NOOS format (script nchis2noos.sh)
2. the stochModelFactory (org.openda.blackbox.wrapper.BBStochModelFactory). A black box model configuration consist of three configuration files that are described in more detail below.
3. the algorithm (org.openda.algorithms.Dud). Dud is a well known algorithm in calibration experiments, more information about it can be found on the OpenDA website or in the literature.

The configuration files for these 3 components are located in different sub directories to reflect the Object Oriented architecture of OpenDA. The fourth block in the configuration file specifies the result writer (org.openda.resultwriters.MatlabResultWriter). The resulting m-file may be loaded in Matlab to visualize results of the OpenDA run.

In this example, the data exchange between OpenDA and D-Flow FM is limited to the bed friction coefficients and the computed waterlevel at observation locations. The waterlevel at a observation location is expected to be written to NetCDF-file with the following features

1. dimension 'time' and 'stations' are defined
2. there exists a variable 'station_id(stations)' defined that contains strings with the station_id

For NetCDF-files that satisfy these two conditions OpenDA creates an 'exchange item' for each variable that has the dimensions (time,stations). The exchange item is referred to as 'station id(nr)'.'name of variable'.

8.4.2 Example 2: EnKF with uncertainty in the inflow velocity

The geometry in this example is the same as for the calibration example: a two-dimensional river model, the initial waterlevel is zero, the river bed is filled gradually due to the boundary conditions. At the inflow boundary, a constant discharge is prescribed (along the line `sw_east_dis_0001.pli`), whereas at the outflow boundary, a constant waterheight is prescribed (along the line `sw_west_wlev_0001.cmp`).

There are 3 observation locations along the river (Obs01, Obs02 and Obs03). The simple matlab script `plot_results_hisfile.m` is available to plot the waterlevel as a function of time for these 3 stations. Simulation time span is 100 days (Start: 199208310000, End: 199212090000). The noise contribution is found in file `sw_east_dis_noise.pli`. Initially, no noise is present, so the file contains zeroes in directory <input_dflowfm>.

Compare the results of a sequential simulation to the observations. The results of running `SequentialSimulation.oda` show a considerable deviation from the results of running `Simulation.oda` or running a regular D-Flow FM run. This testcase seems to be very sensible to small errors in the restart functionality of D-Flow FM. The configuration of the EnKF algorithm is therefore not yet optimized.

8.4.3 Example 3: EnKF with uncertainty in the inflow condition for salt

The geometry in this example and the boundary conditions for waterlevel and velocity are exactly the same as in `simple_waal_kalman`. The transport of salt is added to the computation by a discharge boundary condition `sw_east_dis_sal_001.pli` and a noise component added to this boundary.

The results are hampered by the same restart problems as the `simple_waal`, so the configuration of the EnKF algorithm must still be improved upon.

8.4.4 Example 4: EnKF with uncertainty in the tidal components

The geometry for this testcase is the same as used in the Delft3D model example for calibration that was presented in a Deltares webinar (recording, slides and all configuration files for this example are available at the OpenDA website).

Regularly, D-Flow FM uses 1 component file to specify all tidal component (one component at a line). In order to add different noise models to different components, you must split the component file and add one `.tim`-files for noise for each component.

Again, all configuration files are available, but not much effort has been put into the exact configuration of the EnKF algorithm or the noise model specifications. The results of the SequentialSimulation show that this testcase suffers much less from the inexact restart. The whole workflow is highlighted in more detail below.

Step 1: Investigate the directory structure

The directory structure is set as the following:

<bin>
    oda_run_gui.bat
    [and a lot of other files]
<jre>
    [a lot of not directly relevant files]
<tests>
    <estuary_kalman>
        <algorithm>
            EnkfAlgorithm.xml
            SequentialSimulation.xml
            simulationAlgorithm.xml
        <stochModel>
            <bin>
                start_unstruc.bat
                start_unstruc.sh
            <input_dflowfm>
                [D-Flow FM input files]
            BoundaryNoiseM2.xml
            BoundaryNoiseS2.xml
            dflowfmModel.xml
            dflowfmStochModel.xml
            dflowfmWrapper.xml
        <stochObserver>
            waterlevel_station01.noos
            waterlevel_station02.noos
            waterlevel_station03.noos
            noosObservations.xml
        Enkf.oda
        SequentialSimulation.oda
        Simulation.oda
        clean.bat
        clean.sh
        [some Matlab scripts]

A few remarks are made:

- ◇ the directories <bin> and <jre> should be on the same level,
- ◇ the computation is started through running the file oda_run_gui.bat,
- ◇ the bare D-Flow FM model is located in the directory <input_dflowfm>,
- ◇ the observations are in `.noos`-format, and are located in the directory <stochObserver>.

Step 2: Start the EnKF computation

The OpenDA run is launched through the core `oda_run_gui.bat` file. Once having opened this file, a user interface appears. Within the user interface, an `.oda`-file can be opened from a certain case directory (in this case, we have <estuary_kalman>). One can choose `Enkf.oda`, `SequentialSimulation.oda` or `Simulation.oda`. In this case, we choose for `Enkf.oda`.

Step 3: Examine the applied noise

The basic necessary component of an EnKF computation comprises the noise applied to some variable. In this case, the noise is applied to the waterlevel boundary representing the tidal motion. Within the directory <input_dflowfm>, this noise is explicitly declared through a separate polyline and a separate data file for the boundary. The configation of the noise is accomplished through two `.xml`-files in the directory <stoch_model>.

Step 4: Run the EnKF computation

By means of the user interface, the EnKF computation can be launched. After having opened the file `EnKF.oda`, the 'Run'-button can be pressed. The computation is being performed. Along the computation's duration, multiple 'work' directories are generated in the directory <stochModel>, i.e. <work0>, <work1>, <work2>, etc.

Step 5: Evaluate the outcomes

After having run the computation, output files have been generated in Matlab-format. The relevant files are placed in the directory <estuary_kalman>. The data are stored in the Matlab file `Enkf_results.m`. Visualisation could be accomplished like shown in Figure 8.1.

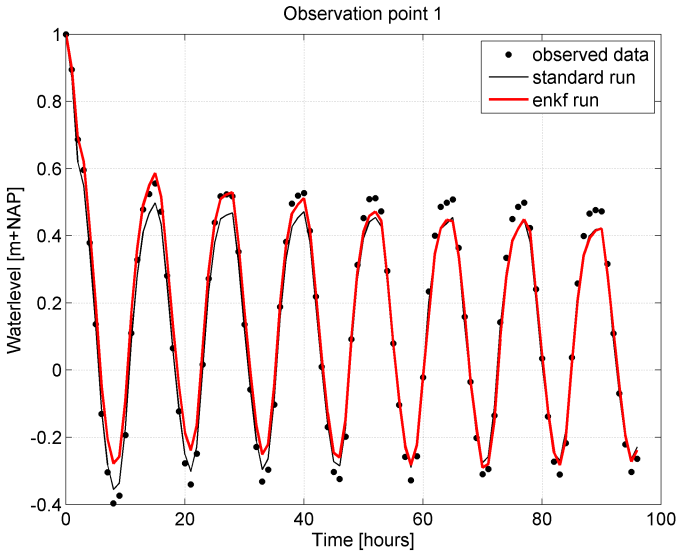


Figure 8.1: *Visualisation of the EnKF computation results from OpenDA for a certain observation point. The dots represent the observed data, the black line represents the original computation with D-Flow FM (without Kalman filtering) and the red line represents the D-Flow FM computation with Kalman filtering.*

9 Tutorials

In this chapter, the grid generation functionalities mentioned in the previous chapters are demonstrated within the context of a real modelling application. For this purpose, the Westernscheldt area is chosen.

9.1 Tutorial 1: Getting familiar with the network mode

To get yourself familiar with the network mode and its relative ease for users, a small exercise is included below. This exercise aims at getting you used to the clicking and keyboard facilities that have already been displayed in Table 2.1.

A net can be established as follows:

1. Open D-Flow FM and click on *Edit* → *Edit network*.
2. Press `i` and click several times around with the left cursor of the mouse. Each click will insert a netnode that is connected with the previously placed netnode via a netlink.
3. When you press `i` again, a new series of netnodes can be added, independent of the previously placed network. If you place a netnode in the very close vicinity of an already existing netnode, you will see a large dot appearing on that node. This large dot indicates that the two nodes have merged.
4. Click on *Addsubdel* → *Delete network* to delete the network.
5. Repeat the previous steps until you have achieved a network that is more or less similar to the network displayed in Figure 9.1.

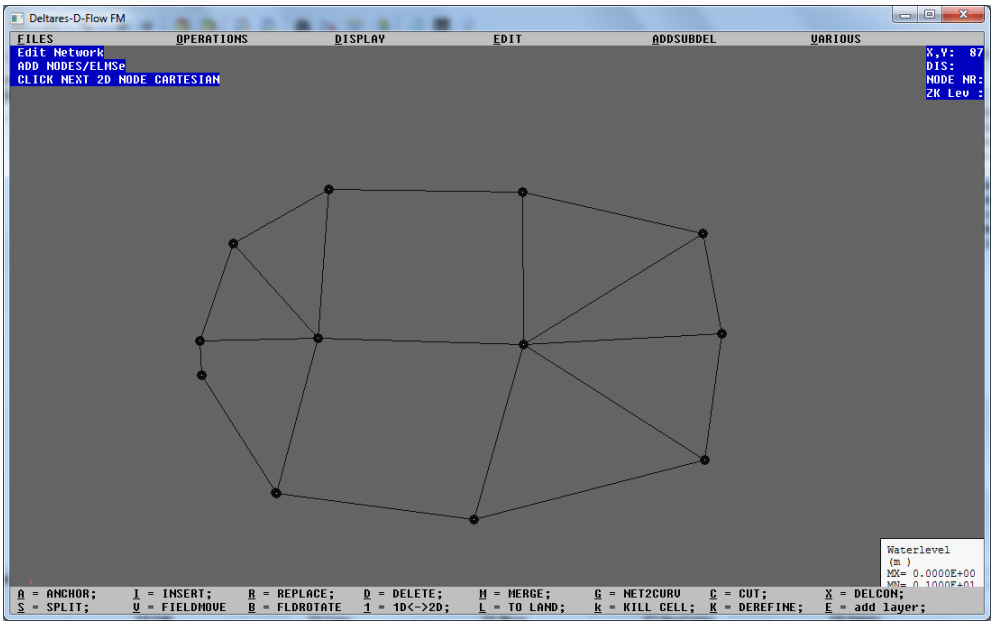


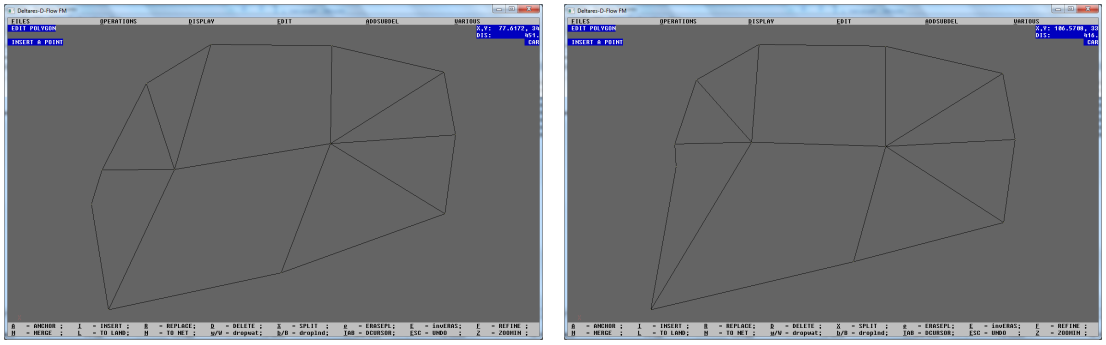
Figure 9.1: *Arbitrarily generated network by multiple left mouse clicks.*

You can also directly load the file `tutorial01_net.nc`. You can use this network for the following exercises.

Moving a netnode

A *netnode* can be moved in two different ways:

1 through the `V`-button – this button enables moving a node while adapting the coordinates of the nodes in its vicinity: press `V` and drag some point outwards,
2 through the `R`-button – this button facilitates moving one single node: press `R`, select a node and define a new location.



(a) *Field move through `V`* ***(b)*** *Local replacement through `R`*

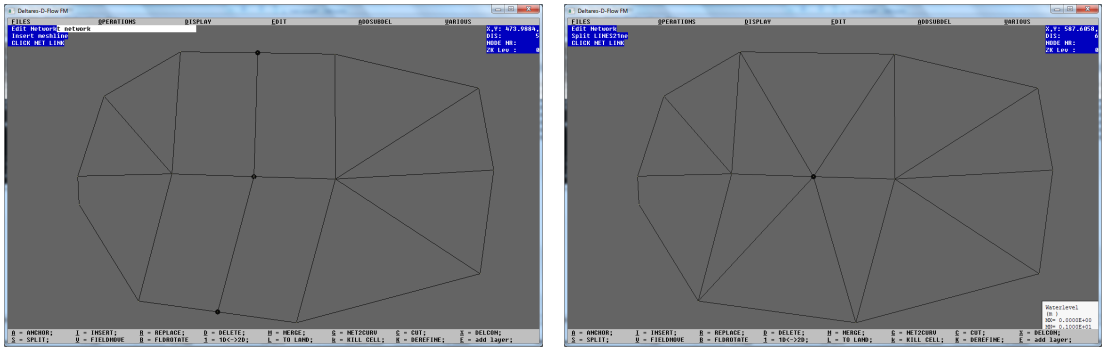
Figure 9.2: *Difference between the field move and the local replacement of a netnode.*

Notice the difference between both operations in Figure 9.2. In the left picture, the left side of the mesh has been deformed after the operation, whereas in the right picture just the single node has been replaced.

Splitting a netlink

A *netlink* can be split in two different ways:

1 through the `S`-button (i.e. `shift+s`) – through pressing this key combination, a netlink can be split along the entire width perpendicular to the netlink: try pressing `shift+s` on the centre netlink,
2 through the `s`-button – this button facilitates local splitting of a netlink: try pressing `s` on the centre netlink.



(a) *Global splitting through `shift+s`* ***(b)*** *Local splitting through `R`.*

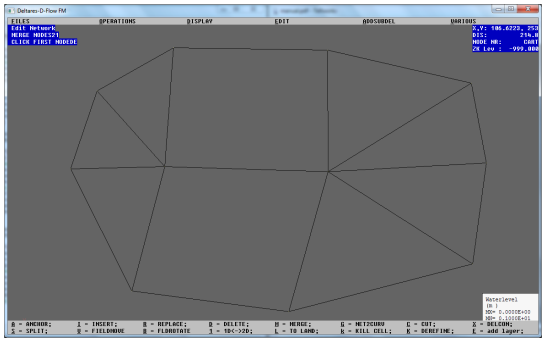
Figure 9.3: *Difference between global splitting and the local splitting of a netlink.*

Notice the difference between both operators in Figure 9.3. In the left picture, it has initiated a split along the entire width, in the right picture, the netlink has locally been split.

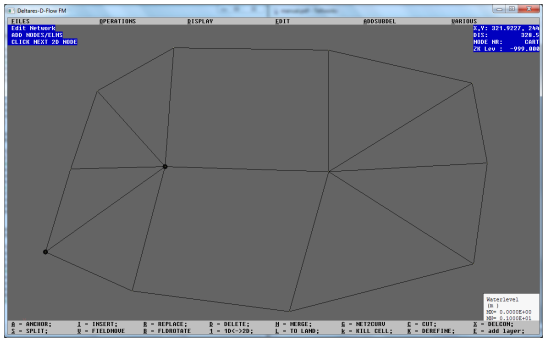
Getting rid of a netnode

There are several ways to get rid of a netnode. Look for instance at the left lower cell of the mesh, where a netnode connects only two netlinks. This netnode can either be part of a corner cell, but can also be an undesired irregularity. To get rid of this netnode, you can follow these options:

1 through the M-button – merge two netnodes: press M, left click on the netnode and see a large dot appear; then select a second node to merge the netnode with. The dent has disappeared.
2 through the X-button – this button facilitates deleting a netnode while keeping the adjacent two netlinks alive. Find out that this functionality only works for netnodes with two netlinks.
3 through the D-button – this button enables the deletion of a *netlink*: press D and left click on an adjacent netlink. The disadvantage of this approach is that still a merge should be done through M. However, in more complex cases, key D can be rather useful though.



(a) *Adapt the mesh through M, X or D*



(b) *Extend the mesh through R and I*

Figure 9.4: *Amendments of the mesh by either reducing the mesh (through M, X or D) or extending the mesh (through R and I).*

4 If the cell is meant to be a cornercell, then the netnode can be replaced towards a more favourable location through R. Optionally, one can add a new netlink through I.

The result of operations 1, 2 or 3 is shown in the left picture of Figure 9.4, the result of the operation 4 is shown in the right picture of Figure 9.4.

Other operations

Now you have seen quite some possibilities to manually amend your mesh. Still some features have yet been unmentioned. Take the opportunity to find out how (the combination of) multiple subsequent operations work out. The more you master these operators, the better you will be able to adapt the mesh towards your our demands as well as towards the demans of the computational core (i.e. orthogonality and smoothness). The next section offers an extensive exercise by which you can practise.

9.2 Generating a combined mesh

In order to illustrate the possibilities of D-Flow FM to generate meshes, the Scheldt river has been chosen for some tutorial exercises. The Scheldt river in the greater Antwerp area consists of a narrowing meandering river and some harbour areas adjacent to it. The region of interest for the exercises is shown in Figure 9.5.

First, the entire Scheldt harbour as well as the Westerscheldt (up to the North Sea) will be captured by a mesh. Thereafter, bottomlevel data will be projected onto the mesh and bound-

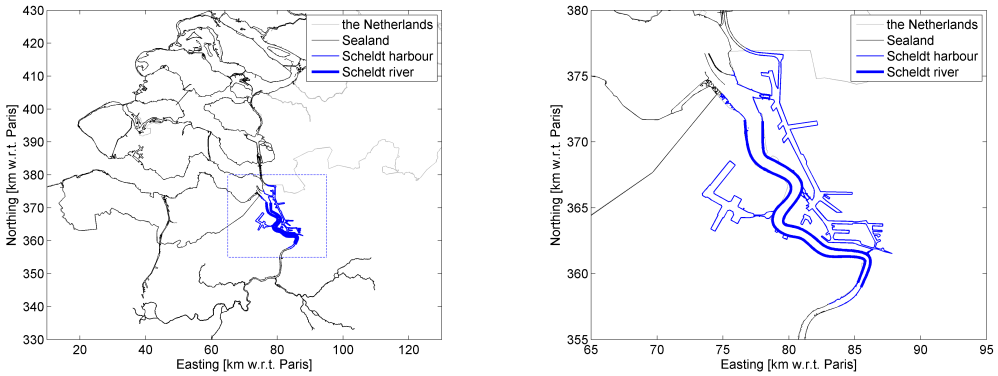


Figure 9.5: *The Scheldt river in the greater Antwerp area. The right figure is a detailed version of the area marked with blue dashed lines in the left figure.*

ary conditions will be imposed. At the end, the computation will be run and the output of it will be viewed.

9.2.1 Tutorial 2: The familiar curvilinear mesh method

D-Flow FM provides two different ways to generate curvilinear meshes. The first method is identical to the method RGFGRID has provided within the Delft3D-context. The second method enables mesh generation by means of an algorithm in which the mesh 'grows' from the centre of a channel. This second, alternative method is only available in D-Flow FM. Both methods are illustrated in this section.

A curvilinear mesh of the Scheldt river can be set up by means of the familiar method following these steps:

1 Load the provided file <scheldtriver.ldb> by choosing *Files* → *Load land boundary*.
2 Click *Edit* → *Edit splines* and press the `i`-key.
3 Use the left mouse button to insert a spline, consisting of multiple points, that follows somehow the lefthandside land boundary.
4 Press `L` (i.e. the 'to land' option down in the screen) and click on a point along the spline. Confirm with `yes` and continue until the spline collapses with the land boundary. You can see that the spline has evolved towards the land boundary automatically.
5 Again, press the `i`-key and insert a second spline that follows the righthandside land boundary.
6 Again, press `L` and click on a point along the spline. Confirm with *yes* and continue until the spline collapses with the land boundary.
7 Choose *Edit* → *Edit splines* and press `i`. Now place a cross-spline at the north exit as well as at the south exit of the river area, both perpendicular to the first two splines.
8 Choose *Files* → *Save splines* and save the four splines.
9 Change the settings of the curvilinear grid by clicking on *Various* → *Change curvilinear grid parameters*. Change the number in the upper two entries (the *M-refinement factor* and *N-refinement factor*) into *60* and *8*. This will create a 60×8 mesh.
10 Generate the mesh by clickin on *Operation* → *Create curvilinear grid from splines*. The thus resulting mesh does not look quite orthogonal.
11 Choose the option *Operation* → *Convert grid to net*. The curvilinear mesh is then transformed into an unstructured mesh (a network).
12 Visualise the orthogonality by clicking *Display* → *Values at net links* and subsequently clicking on *Link orthogonality cosphi*. The thus revealed picture is shown in Figure 9.6.

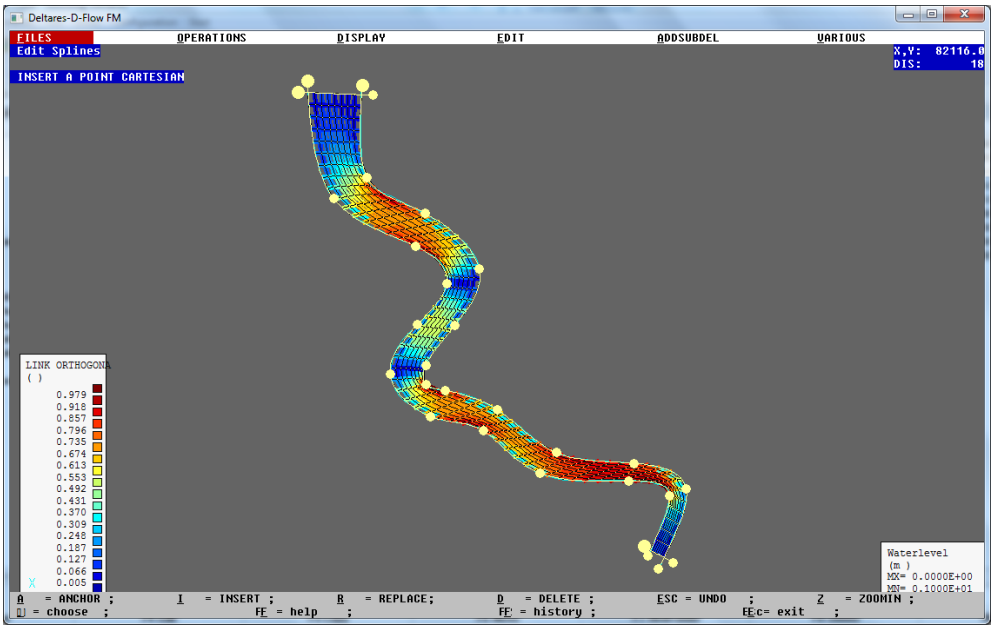


Figure 9.6: *Orthogonality of the generated curvilinear mesh following the familiar RGFGRID-procedure of Delft3D.*

The orthogonality is given by the cosine of the angle φ, which is the angle between a line connecting cell centres (i.e. 'flowlinks') and a line connecting network corners (i.e. 'netlinks'). Apparently, the mesh shown in Figure 9.6 contains angles between flowlinks and netlinks down to 14°. Ideally, all the angles equal 90° to assure accurate hydrodynamic computations.

Improvement of the mesh orthogonality can be established by adding multiple cross-splines. The approach is as follows:

1. Delete the current network by choosing *Addsubdel* → *Delete network* and confirm with *yes*.
2. Click on *Edit* → *Edit splines*. Press `i` and add 9 cross-splines perpendicular to the channel axis, dividing the channel into 10 parts.
3. The number of gridcells is imposed between two cross-splines. To achieve a mesh of about 60×8 cells, the settings of the curvilinear grid should be adapted. Therefore, choose *Various* → *Change curvilinear grid parameters* and change the number *60* into *6* (leave the N-refinementfactor at 8).
4. Generate the mesh by clicking on *Operations* → *Create curvilinear grid from splines*. Afterwards, change the mesh into a network via *Operation* → *Convert grid to net*.
5. Visualise the orthogonality by clicking *Display* → *Values at net links* and subsequentially clicking on *Link orthogonality cosphi*. The thus revealed picture is shown in Figure 9.7.

From Figure 9.7, it is clear that the orthogonality has significantly improved.

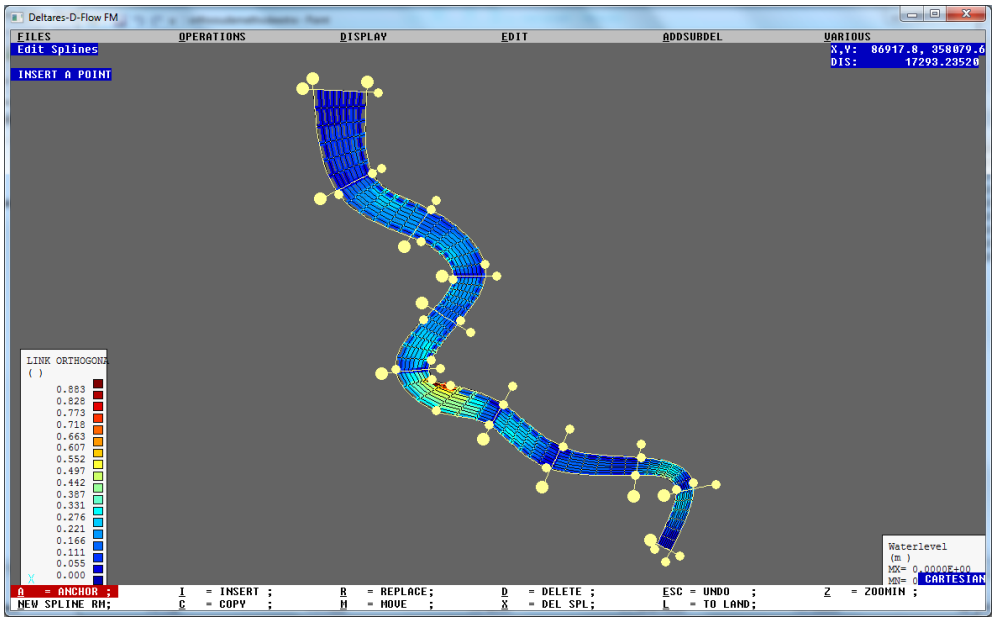


Figure 9.7: *Orthogonality of the generated curvilinear mesh using the additional cross-splines according to the familiar RGFGRID method from Delft3D.*

9.2.2 Tutorial 3: The alternative curvilinear mesh method

D-Flow FM provides an improved method to generate curvilinear meshes directly from splines. In this method, a mesh is gradually developed from the centreline of the channel towards the boundaries. This methods requires less actions by the user. This approach can be illustrated as follows:

1 If you have restarted the D-Flow FM environment, then load the four splines you have previously saved.
2 If you have not closed the D-Flow FM environment delete the current network and the intermediate cross-splines. Click *yes* as the answer to the appearing question.
3 Now click on *Operations* → *Grow curvilinear grid from splines*. A list of settings appears on the screen. The upper 7 entries should be adapted into the values shown in Table 9.1. Using the parameter `Max. num. of gridcells perp. in uni. part`,

`Maximum number of gridcells along spline`	`2000`
`Maximum number of gridcells perp. spline`	`40`
`Aspect ratio of first grid layer`	`0.5`
`Grid layer height growth factor`	`1`
`Maximum grid length along center spline`	`400`
`Grow grid outside first part (1/0)`	`0`
`Max. num. of gridcells perp. in uni. part`	`8`

Table 9.1: *Selected settings for the approach of a growing curvilinear mesh.*

the user can give an indication of the number of cells across the width between the longitudinal splines. By using the parameters `Maximum grid length along center spline`, the user can give an indication of the length of the cells in longitudinal direction. Based on the value of the parameter `Aspect ratio of first grid layer`, the algorithm establishes a suitable mesh, under the restrictions of the prevailing maximum numbers of gridcells (first two entries).
The option `Grid layer height growth factor` enables the user to demand for a non-equidistant mesh in cross-sectional direction. The value represents the width-ratio

of two adjacent cells. Using the option `Grow grid outside first part (1/0)`, one can extend a mesh outside the longitudinal splines, for instance to capture winterbed regions.

4 After entering the values of Table 9.1, press `Enter` which will deliver the mesh as shown in Figure 9.8. The smalles angle between netlink and flowlink is about 80°, which is quite good as a starting point for further orgonalisation of the mesh (see next section). This approach has considerable advantages compared to the familiar mesh generation approach regarding both the user's comfort and the orthogonality.

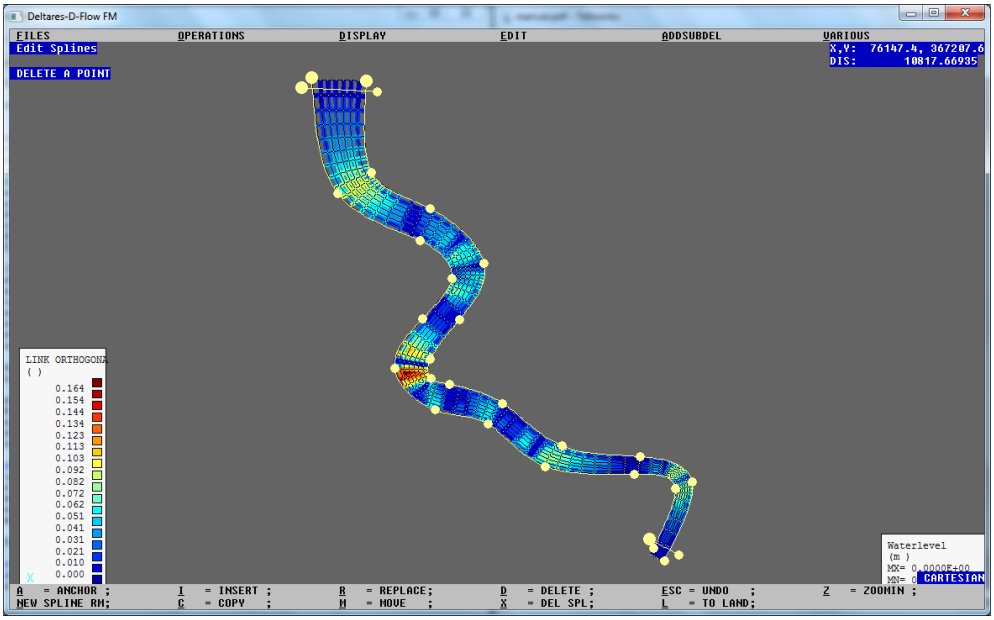


Figure 9.8: *Orthogonaly of the generated curvilinear mesh according to the new method of D-Flow FM.*

Now, it is illustrated how the orthogonalisation can further be improved in D-Flow FM.

1 Choose *Various* → *Change orthogonalisation parameters*. The fourth parameter in the list, *Orthogonalise - smooth; 1.0 - 0.0*, can be used to compromise between the orthogonality (angle between netlink and flowlink) of the grid and its smoothness (the ratio between two areas of two adjacent cells). In this case, we choose for strict orthogonality by setting the parameter on 1.
2 Click on *Operations* → *Orthogonalise / smooth net*. This action can be done twice. The thus resulting mesh is shown in Figure 9.9.

After the first orthogonalisation step, the value of $\cos\varphi$ is about 2.5%; after the second orthogonalisation step, this value has decreased to 2.3% which corresponds to an angle of about 89°. Locally, the mesh can further be improved by merging, moving or removing netnodes (i.e. connections between netlinks). These options will be illustrated in the next exercise.

9.2.3 Tutorial 4: Basic triangular mesh generation

From the previous section, a curvilinear mesh is available for the Scheldt river. The river is separated from the harbour, west of the river, by a sluice. The small area between the sluice and the Scheldt will benefit from an unstructured mesh because of its irregular geometry. This irregular geometry is meshed in this section first and afterwards connected to the existing Scheldt river mesh.

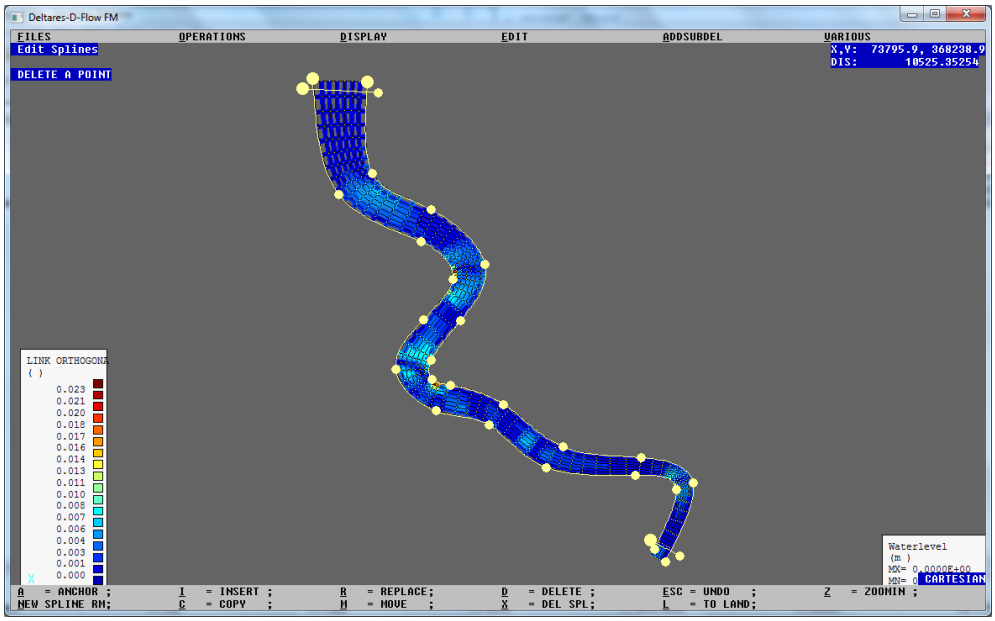


Figure 9.9: *Orthogonaly of the generated curvilinear mesh according to the new method of D-Flow FM after two orthogonalisations.*

The approach is as follows:

1 Load the files <scheldtharbour.ldb> (landboundaries of the harbour) and (if necessary) <tutorial03_net.nc> (curvilinear mesh from the previous section).
2 Click on *Edit* → *Edit polygon* and press on the `i` key. The intention is to mark the area of interest through a polygon.
3 Start drawing a polygon at a distance of the order of a gridcell away from the curvilinear mesh. Let the second point be at a relative small distance from the first one. This distance is later used as an indication of the size of the triangular gridcells to be placed.
4 Mark the elementary locations of the area and place the last point again at a distance of the order of a gridcell away from the river mesh. The result is shown in Figure 9.10.
5 Next, we choose *Operations* → *Refine polygon*. Now, the polygon is divided into a finer set of line elements. The distance between the points of the polygon is derived from the distance between the first two initial polygon points. Hence, the importance of the location of the first two points of the initial polygon.
6 Press `L` to let the polygon fit the landboundaries. To that end, press a first polygon point and a second polygon point in an area where to fit the landboundary. Again, choose *Operations* → *Refine polygon*.
7 Choose *Various* → *Choose network parameters* and change the fourth number (the *triangulesizefactor* into 1.5. This number represents the extent the triangles 'grow' from the boundaries towards the center of the polygon.
8 Choose *Operations* → *Create samples in polygon*. Now, red squares are placed inside the polygon as an aid to insert the triangles.
9 Insert the triangles by choosing *Operations* → *Triangulate samples to net in polygon*. The result is a screen showing the landboundaries and the triangular mesh.

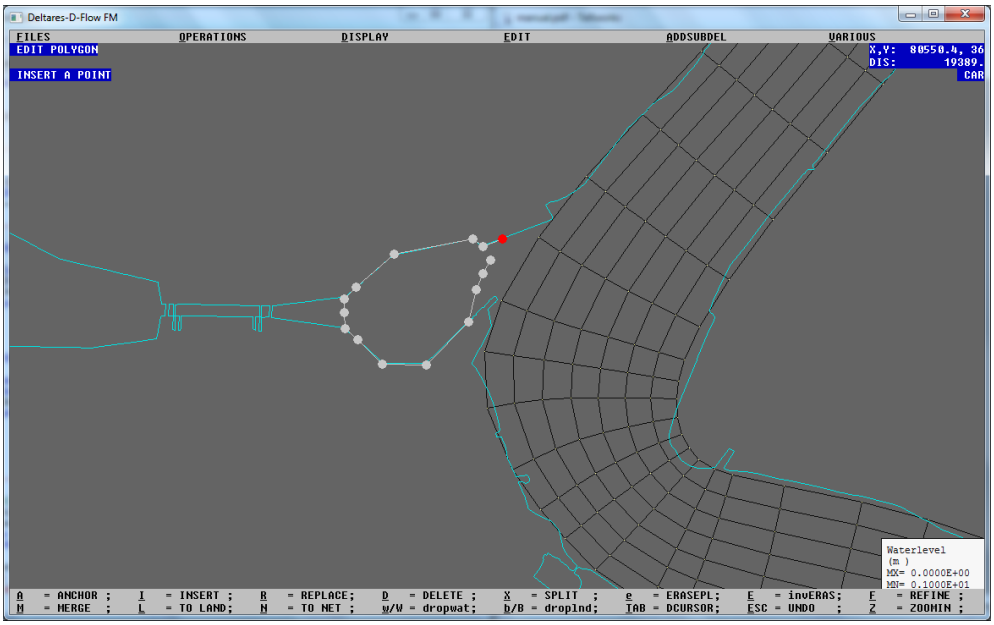


Figure 9.10: *Orthogonaly of the generated curvilinear mesh according to the new method of D-Flow FM after two orthogonalisations.*

9.2.4 Tutorial 5: Coupling two meshes

Next, we have to couple the triangular mesh to the curvilinear mesh.

1. First, we refine the curvilinear mesh near the connection. To that end, we choose *Edit* → *Edit network*.
2. At boundaries of the curvilinear net, we press `Shift+s` to split the mesh across the entire width.
3. Next, we press `i` and we connect the netnodes near the interface in a zigzag-like way. Try to achieve the picture that is shown in Figure 9.11. The black dots mark locations where new connections are added.

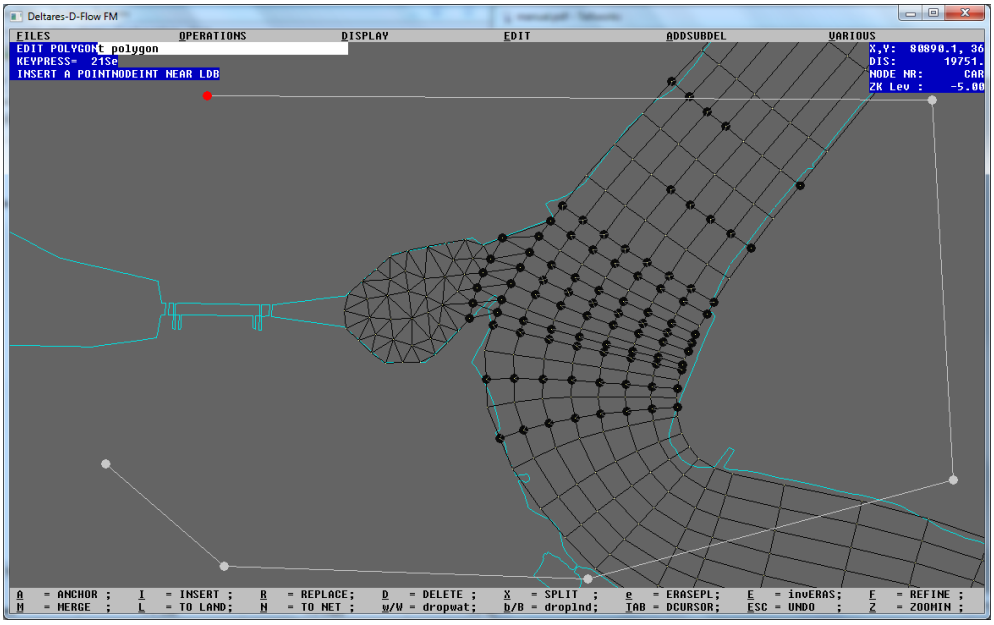


Figure 9.11: *Orthogonaly of the generated curvilinear mesh according to the new method of D-Flow FM after two orthogonalisations.*

4 The final thing that should be done now is the orthogonalisation. To that end, choose *Edit* → *Edit polygon* and draw a polygon that encloses a region covering the entire unstructured net as well as a considerable part of the curvilinear net.
5 Visualise the orthogonality of the net by choosing *Display* → *Values at net links* and subsequently choosing *Link orthogonality cosphi*.
6 The next thing to do is adapting the orthogonality setting by clicking *Various* → *Change orthogonalisation parameters*. The number in the fourth cell (*Orthogonalise — smooth*) should be 0.95.
7 Choose *Operations* → *Orthogonalise / smooth net*. The net now evolves towards a more orthogonal net. Repeat this operation. Ater some manipulation (probably, only the use of the replace-functionality will do, i.e. the R button), the picture shown in Figure 9.12 can be achieved.

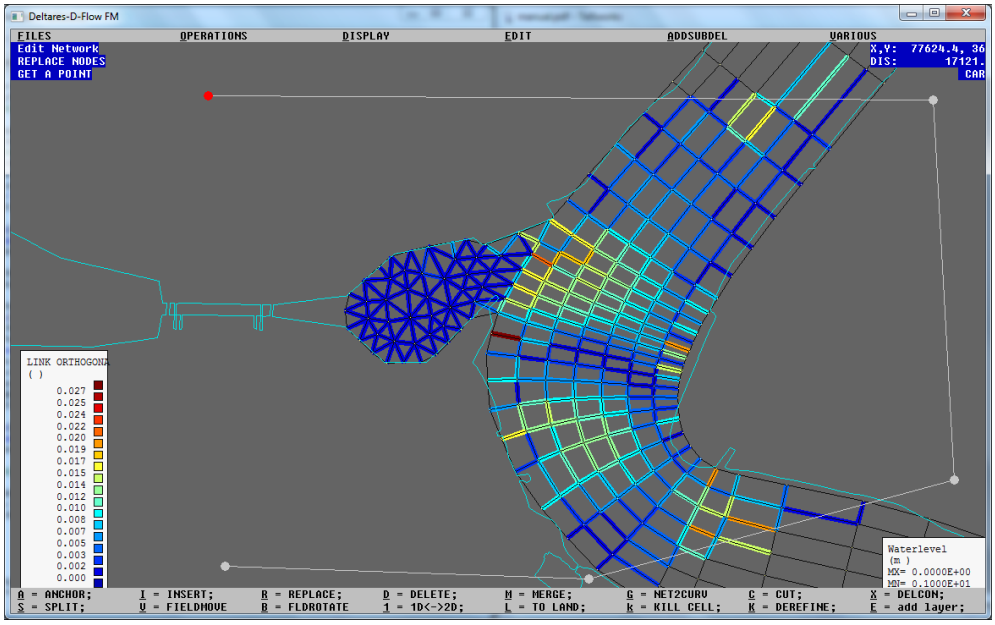


Figure 9.12: *Orthogonaly of the generated curvilinear mesh according to the new method of D-Flow FM after two orthogonalisations.*

It is the art to use the network tools, presented in the previous section, in such a way that these support better orthogonality and smoothness. To that end, use the keys I, R, D, X, etc. You could also try the option *Operations* → *Flip links* from the menu bar: this option facilitates crosswisely oriented flipping of a netlink within a cell.

Figure 9.12 on page 70 shows that the orthogonality of the mesh near the connection is generally below 3%. The highest orthogonality is found at the edge of the polygon. This is not surprising, since this netlink is surrounded by three cells located outside the polygon. These three cells are not adapted by the orthogonalisation/smoothing-algorithm.

9.2.5 Tutorial 6: Meshing the harbour domain

At this point, one mesh is available that captures the Scheldt river and the irregular basin towards the harbour. Yet, the sluice as well as the harbour itself are still left to be meshed. To cover the remainder of the harbour, we first treat the sluice.

Meshing the sluice

The sluice has a rather rectangular shape. Because of its regularity, a curvilinear mesh is prefered. In previous sections, we have discussed two methods for creating a curvilinear grid. The new method (the 'growing' mesh) is preferred if the geometry contains strong curvatures. Since we deal with a straight channel, no particular method is preferable. In any case, we use splines to capture the sluice geometry.

Creating of a regular mesh in the sluice area can be conducted as follows:

1. Choose *Edit* → *Edit splines*, press `i` and draw the contours of the sluice. In this example, we use four cross-splines.
2. Choose *Various* → *Change curvilinear grid parameters* and change the numbers in the upper two entries into 6 and 4, respectively. An area bounded by four splines will hence contain 6×4 gridcells. The result is shown in Figure 9.13.
3. Choose *Operations* → *Create curvilinear grid from splines*
4. Click on *Operations* → *Convert grid to net* and answer the appearing question *Merge nodes?* with *no*. The mesh will turn from red to black. If prefered, one can delete the splines for visibility reasons by clicking *Addsubdel* → *Delete splines*.
5. The next thing to do is connecting the sluice network to the existing network. To that end, choose *Edit* → *Edit network*.
6. Connect the networks in a smart way. It might be necessary to improve the mesh by choosing *Addsubdel* → *Merge nodes on top of each other* and *Addsubdel* → *Remove small flow links from network*. Although, these options are expected not to be necessary in this case, these commands might be useful in more complex networks.
7. Check the fourth parameter in the menu *Various* → *Change orthogonalisation parameters*. The number 0.95 is ok to maintain.
8. Visualise the orthogonality of the mesh by choosing *Display* → *Values at net links* and subsequently improve this by choosing *Operations* → *Orthogonalise / smooth net*. You can repeat this action, if you to do so. It might be helpful to replace some nodes to improve the orthogonality. Therefore, press `r` and follow the instructions on the screen.

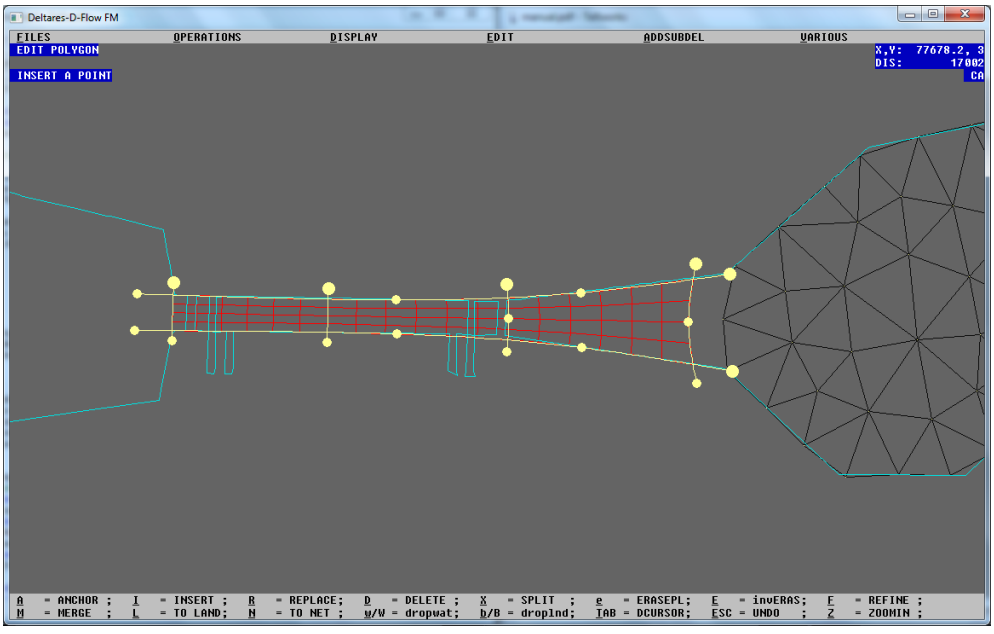


Figure 9.13: *In yellow: splines that follow the sluice geometry, in red: the generated curvilinear grid and in black: the existing network.*

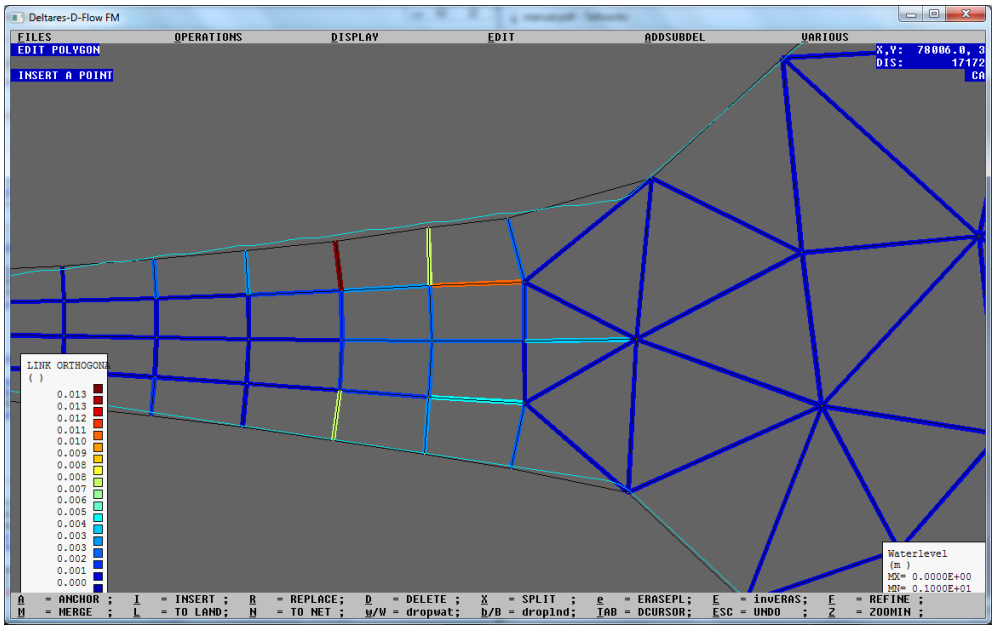


Figure 9.14: *Orthogonality of the mesh after coupling the two networks.*

The result of the operations is shown in Figure 9.14. The orthogonality is 0.009 at the maximum. If you zoom out (press z twice), you can check the overall state of the network.

Meshing the docks

The area behind the sluice still needs to be meshed. This dock area can be thought of as one main channel with four side channels. These five parts in total have quite a rectangular shape which makes curvilinear meshing preferable above unstructured meshing. The proposed meshing procedure is hence very much comparable to the procedure that has been followed for the sluice area. Therefore, the procedure how to apply a mesh to the five-dock area will only concisely be prescribed.

The general procedure *for each block subsequently* is the following:

1. Load the file $<$scheldtharbour.ldb$>$ and zoom to the region of interest.
2. Choose *Edit* $\rightarrow$ *Edit splines* and create elementary boundaries for the curvilinear grid to be generated. That means: for each particular block (A, B, etc.), draw one spline at each of the four boundaries. Each block will hence have four splines, following the boundaries.
3. Choose appropriate gridcell numbers via the menu *Various* $\rightarrow$ *Change curvilinear grid parameters*.
4. Click on *Operations* $\rightarrow$ *Convert grid to net*, remove the old splines and continue with creating new splines for the next mesh block. The result should be like indicated in Figure 9.15. Appropriate gridcell amounts are 40×8 for region A, 20×6 for region B, 6×6 for region C, 20×6 for region D and 6×6 for region E.

Figure 9.15 reveals three issues that are yet to be dealt with:

- ◇ the meshes are not yet connected to each other,
- ◇ the meshes do not very well fit the boundaries of the harbour,
- ◇ the meshes do not look quite orthogonal.

Now, follow this procedure:

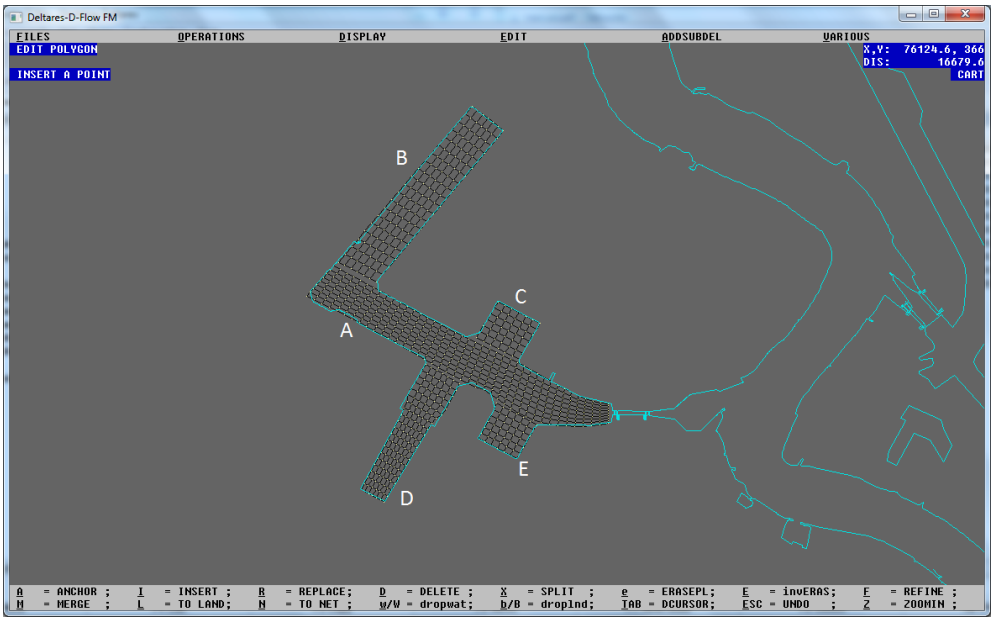


Figure 9.15: *Curvilinear meshes for the five parts of the harbour area.*

1 Meeting the first issue is easy to do. Enter the network-mode (menu *Edit* → *Edit network*) and manually connect netnodes by repeatedly pressing `i` and clicking the left mouse button.
2 The second and third issue can be solved integrally. Thereto, click *Various* → *Change orthogonalisation parameters*. The crucial parameter is *project to (land)boundary*, the eighth number in the list. Scroll down with the cursor arrow. If the cell colours red, then one can see an explanation of the several options down in the screen. Choose option 3.
3 To be more elaborative, choose the first option and the third option equal to a higher number, e.g. 250. This number represents the number of iterations to be carried out.
4 Click *Operations* → *Orthogonalise / smooth net*. With option 3 (see step 2) the mesh will evolve and strain towards a situation in which it fits the boundaries and is as orthogonal as possible. At some particular time during this procedure, the mesh can look like the one shown in Figure 9.16.

Figure 9.16 shows a mesh with is for the far most part orthogonal. Locally, the orthogonality can be improved. This can be done manually. Important means for improvement are the `Shift+s` (splitting cells across the width) and `s` (local splitting of netlinks) buttons, because it gives the mesh topology more freedom to move. Play around until the mesh shows satisfactory orthogonality (i.e. $\cos\varphi < 0.01$).

The highest value will probably occur near a boundary. In case of Figure 9.16, the highest value is found near the sluice, i.e. the region where the dock area is to be connected to the already generated mesh.

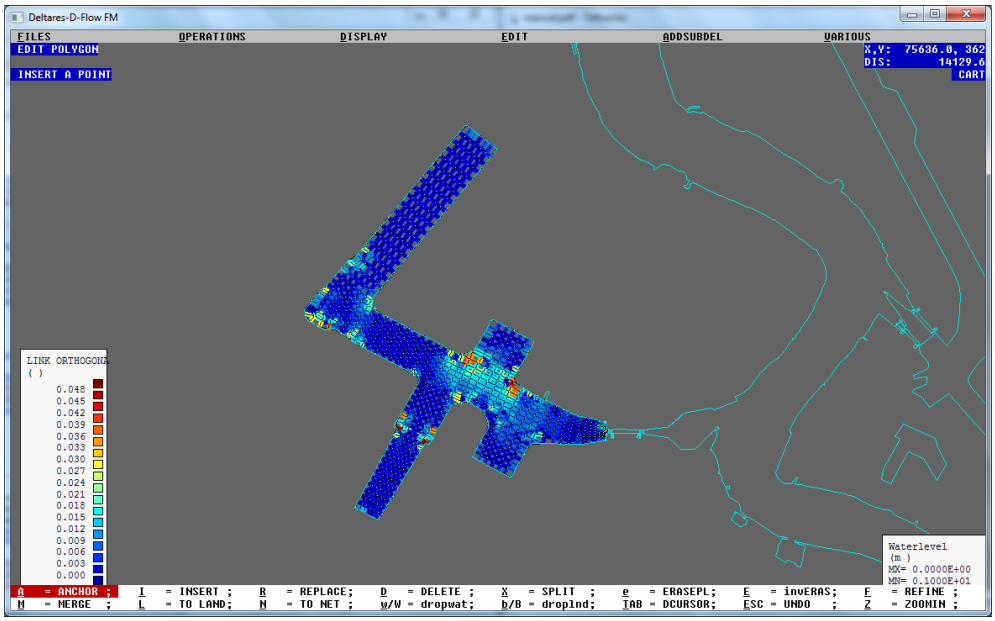


Figure 9.16: *Computational mesh at an intermediate step during the procedure to get the mesh fit the boundaries and be orthogonal.*

Finalising the mesh

The major part of the meshing activities has been done now. The four different subdomains (Scheldt river – sluice – transition region river/sluice – the docks in the harbour) have been covered by either a curvilinear or a unstructured mesh. Only one network connection should still be made: the connection between the sluice and the harbour region.

The procedure is as follows:

1 Load the following two files: <scheldtharbour.ldb> and <tutorial06withsluice_net.nc>.
2 Click on *Files* → *Add network* and select <tutorial06harbour_net.nc>.
3 Zoom to the area where the harbour area and the sluice area should connect. You will see that the two area overlap.
4 Draw a polygon (click *Edit* → *Edit polygon*), around the region of interest and return to the network mode (click *Edit* → *Edit network*).
5 At this time, you will probably be familiar with the split and merge options. Use these options, while monitoring the orthogonality, to achieve the picture shown in Figure 9.17.

As soon as you have attained the result shown in Figure 9.17, the mesh is ready to be finalised. Delete the polygon and zoom out until you can check the entire mesh region. You will probably see that at some points, the orthogonality image indicates that still some work should be done. The reason for this, is that at the boundaries of the polygon (see Figure 9.17) the mesh has not been orthogonolised.

To end up with a sound mesh, orthogonolise the mesh once again, but now for entire mesh. The result of this operation is shown in Figure 9.18 on page 75. The quality of the mesh can roughly be assessed against the following rules of fist: the maximum allowed orthogonality at the boundaries is 5% and the maximum allowed orthogonality in the internal parts of the domain is 1%. The mesh shown in Figure 9.18 seems to meet these rules of fist. On a local level, one could further adapt the mesh towards personal demands.

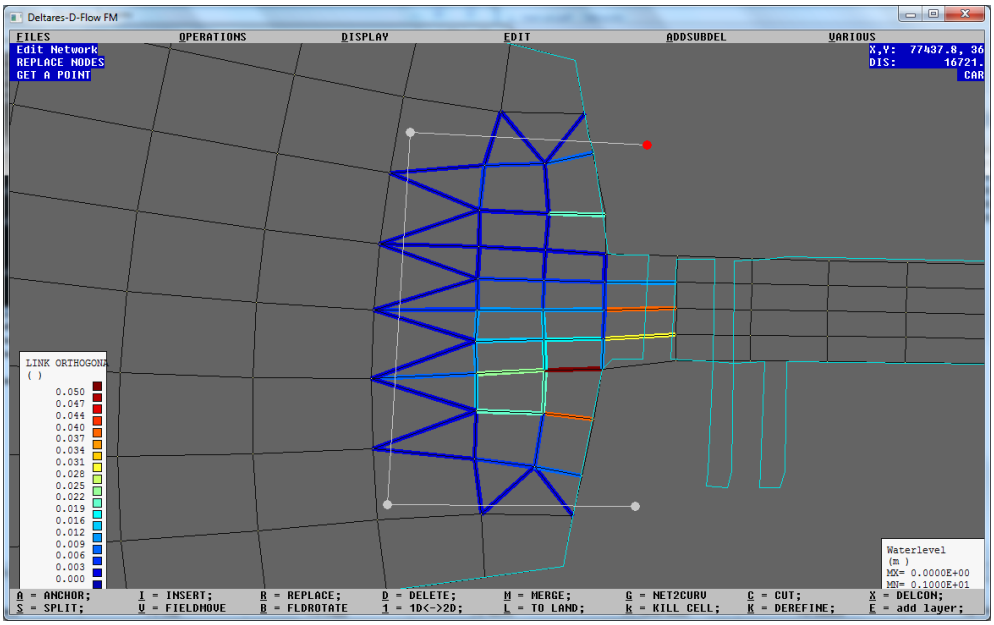


Figure 9.17: *Connection of the harbour area with the sluice area.*

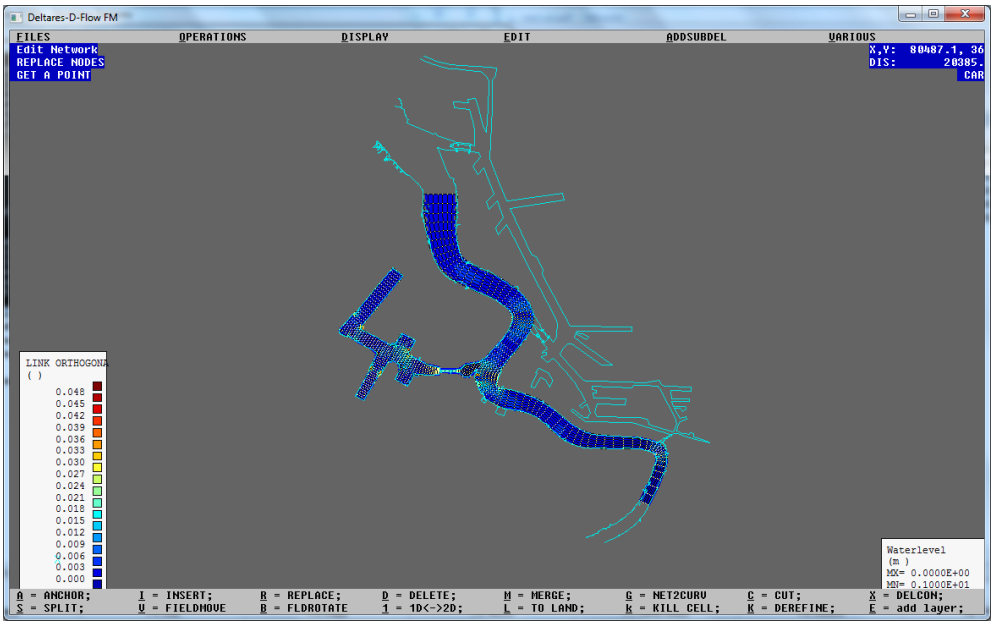


Figure 9.18: *The end result for the mesh. The colours show the orthogonality.*

9.2.6 Tutorial 7: Coupling with Delft3D-meshes

D-Flow FM can easily deal with meshes that have been generated using Delft3D (i.e. RGFGRID). As an example, the D-Flow FM-mesh of the Scheldt area is coupled to an existing mesh of the Westerscheldt (including a part of the Northsea) from Delft3D.

Follow the next procedure to see how the coupling works:

1 Load the files:

- ◇ landboundary file <sealand.ldb>,
- ◇ network file from the previous section <tutorial06_net.nc>,

- ◇ the gridfile <westerscheldt.grd> which contains a curvilinear grid capturing the Westerscheldt and a part of the North Sea.

2 Choose an appropriate location where to connect the two meshes. A suggestion is marked by the white arrow in Figure 9.19.

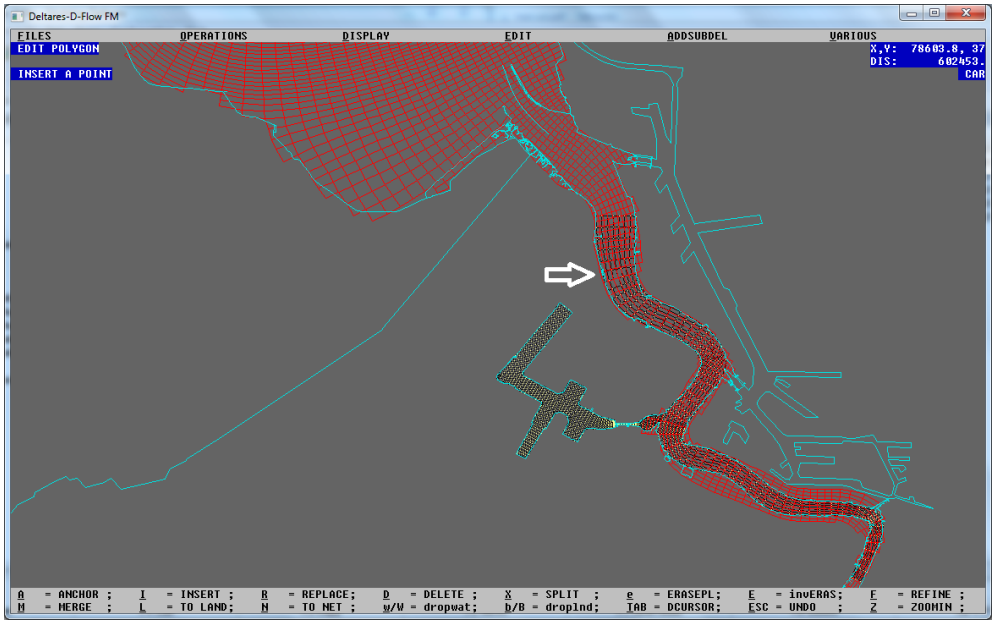


Figure 9.19: *Overlap region between the Westerscheldt and the Scheldt river near Antwerp. The white arrow marks a proper connection cross-section.*

3 Choose *Edit* → *Edit polygon* and draw a polygon that envelopes the 'red' mesh (the curvilinear mesh) south of the white arrow. Then choose *Addsubdel* → *Delete curvilinear grid*.
4 Choose *Edit* → *Edit polygon* and draw a polygon that envelopes the 'black' flexible mesh (the network) north of the white arrow. Then choose *Addsubdel* → *Delete network*.
5 Choose *Operations* → *Convert grid to net*.
6 Choose *Edit* → *Edit network* and press `m`. Now you are able to merge the netnodes from the original mesh with the netnodes of the Westerscheldt.
7 Draw a polygon that captures the connection region and improve the orthogonalisation using the now familiar methods. See Figure 9.20 for the result.
8 Zoom out and check the orthogonality of the overall mesh and carry out corrections if needed. Aim for less than 5% orthogonality near the boundaries and less than 1% orthogonality in the internal parts of the meshes.

In the next section, we will use the thus originated mesh to illustrate how to perform a computation with D-Flow FM on a mesh.

9.3 Preparing a computation

In the previous section, we have set up a mesh that captures the Westerscheldt from far in the North Sea towards the region south of Antwerp. In this section, we will adjust the mesh by adding the bottom topography and appropriate boundary conditions and initial conditions. The final mesh of the previous section reaches far into the North Sea. To save computational time, we cut off the mesh at the mouth of the Westerscheldt, since these exercises mainly serve educational purposes.

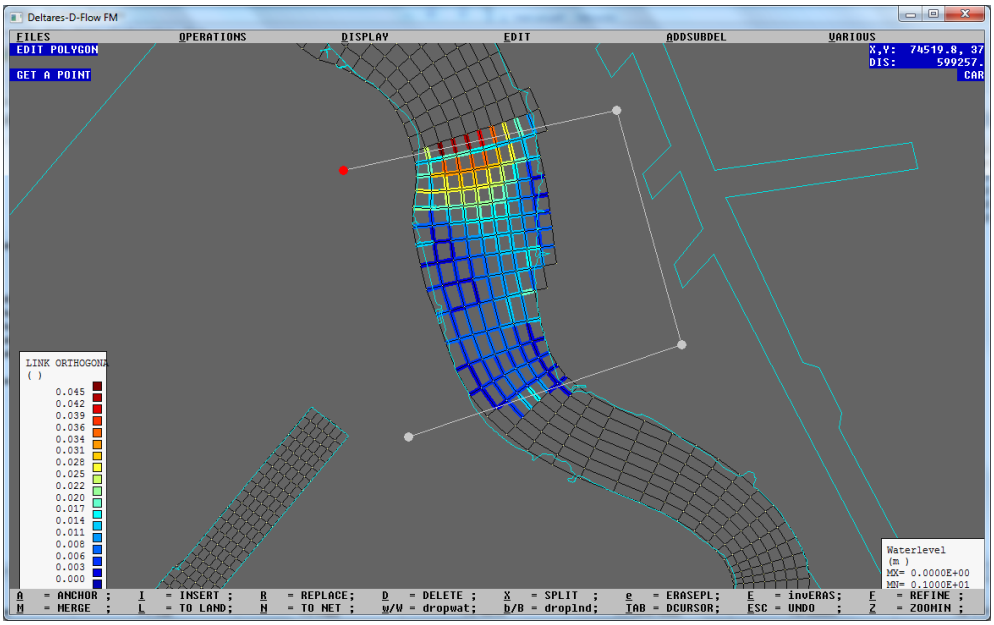


Figure 9.20: *Orthogonality of the new mesh that includes the Westerscheldt.*

9.3.1 Tutorial 8: Inserting the bathymetry

An amended version of the mesh is available from the source files: the file <tutorial07_net.nc> in the directory *tutorial08*. Bottom levels in the Westerscheldt area are available in the file <westerscheldt_bottom.xyz>. The file with bottom levels should first be projected onto the unstructured mesh by means of interpolation. The following actions should be done:

1 Load the files:
   - ◇ landboundary file <sealand.ldb>,
   - ◇ network file <tutorial07_net.nc> from the directory *tutorial08*,
   - ◇ the gridfile <westerscheldt_bottom.xyz> which contains bottom level data for the Westerscheldt from the North Sea up to Antwerp. For this file, choose *Files* → *Load samples*.

2 Choose *Addsubdel* → *Netw zk coordinates*. Then click the option *Field = missing value -999* and confirm with *yes*. This is necessary, since data are only interpolated to points where no data are present yet.

3 Change the visibility of the data. To that end, choose *Display* → *Display samples* → *No sample points*.

4 The bottom level data will be interpolated towards the netnodes. We want to monitor the bottom level values at those locations. To that end, choose *Display* → *Values at net nodes* → *Vertical level zk*. Before interpolation, the screen looks like shown in Figure 9.21.

5 Now we are going to project the bottom level data onto the netnodes. To that end, choose *Operations* → *Interpolate network zk-values in samples*. The result will look like Figure 9.22. If you do not see a colorbar, click *Display* → *Isoscale on or off* and select the option *Isoscale nodes on*.

6 As you can see from Figure 9.22, still a few netnodes have not been assigned a bottom level. These points still show a large purple dot instead of a coloured value. Find where the purple dots are and draw a polygon around these dots (first: *Edit* → *Edit polygon*, then press `i`).

7 Choose *Addsubdel* → *Netw zk coordinates* → *specify uniform value*. Here, you are about to insert the value you wish at the locations denoted by purple dots within the polygon. Choose an appropriate, reasonable value (for instance 3.5 m).

8 After having inserted the value, choose the option *Field = uniform value, only missing*. This imposes the inserted value at locations where no bottom data have been assigned.
9 Have a look at the sluice and the docks of the harbour. You can see that the bottom level in this area is unrealistically high. This unrealistic value has been assigned because of undesired extrapolation, since no bottom level data has been available in this area. To repair this, first draw a polygon that envelops the sluice and the docks.
10 Choose *Addsubdel* → *Netw zk coordinates* → *specify uniform value* and choose an appropriate value, for instance -10 m. Then click on *Field = uniform value, all points*. The result will look like the picture shown in Figure 9.23 on page 79.
11 Zoom out and save the network. The bottom data are now stored in the <∗_net.nc>-file.

9.3.2 Tutorial 9: Boundary conditions

Along both the sea boundary and the Scheldt river boundary, appropriate boundary conditions have to be imposed. The boundary conditions can be prescribed in many ways, for instance as waterlevels, velocities (also tangential and normal), discharge and Riemann. For the sake of simplicity, we are going to impose straightforward boundary conditions for the Westerscheldt:

◇ at the sea: a periodic waterlevel boundary, described as a harmonic signal with a mean 0.5 m+NAP, an amplitude 2.0 m and a period of 745 minutes,
◇ at the river: a constant discharge boundary, described by a constant 2500 cubic meters per second.

The way to define boundary conditions follows this strategy: first, draw a polyline along the boundary – second, fill a file with essential information – third, link the boundary files with the model setup. The boundary conditions can be imposed as follows:

1 Choose *Edit* → *Edit polygon*, press `i` and insert a polyline along the sea boundary.
2 Save the polyline by choosing *Files* → *Save polygon*. Enter a filename that ends with `.pli`, for instance <boundarysea.pli>. It is important to notice that, after having saved the polyline, multiple `.cmp` files have appeared in your working directory. For instance, the files <boundarysea_0001.cmp> up to <boundarysea_0010.cmp>. The number `10`

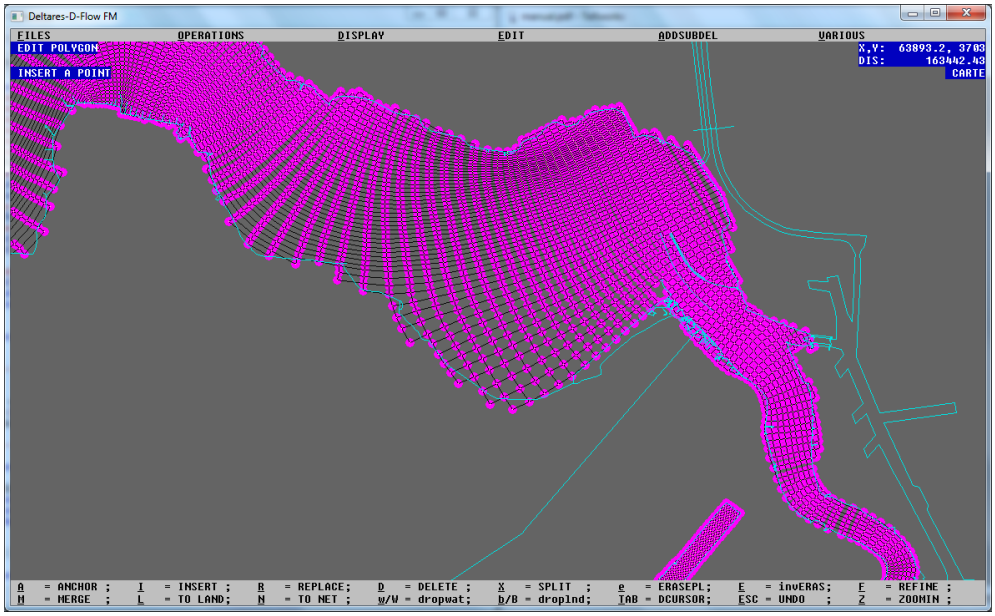


Figure 9.21: *Locations of the bottom levels at a part of the domain. The purple dots denote `zk`-values equal to -999 at the netnodes.*

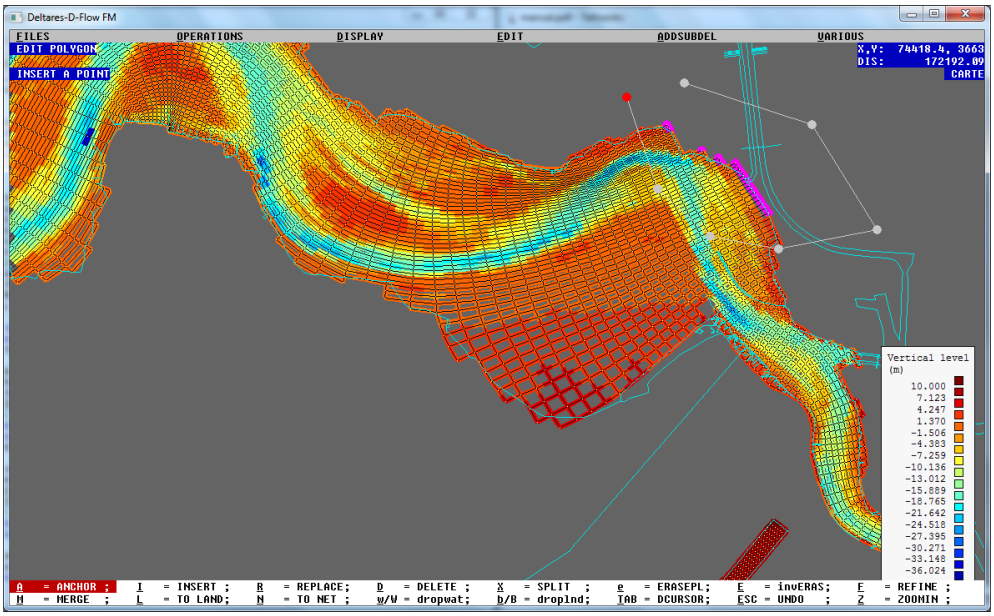


Figure 9.22: *Interpolated bottom levels values at the mesh.*

stems from the amount of points that span the polyline. The abbreviation `cmp` stems from 'component'. In a `cmp`-file, the components of a quantity are prescribed, for instance the period and amplitude of a wave signal. One can also use `tim`-files, which prescribe full timeseries of a quantity. For brevity, we only focus on components rather than on timeseries.

3 Choose *Addsubdel* → *Delete polygon*.

4 Draw another polyline (press `i`) at the river entrance south of Antwerp and save the polyline as <boundaryriver.pli>. Notice, that some <boundaryriver_000i.cmp>-files have appeared in your working directory.

5 Go to your working directory and delete the `cmp`-files, *except* <boundarysea_0001.cmp> and <boundaryriver_0001.cmp>. As a consequence, we will impose boundary condi-

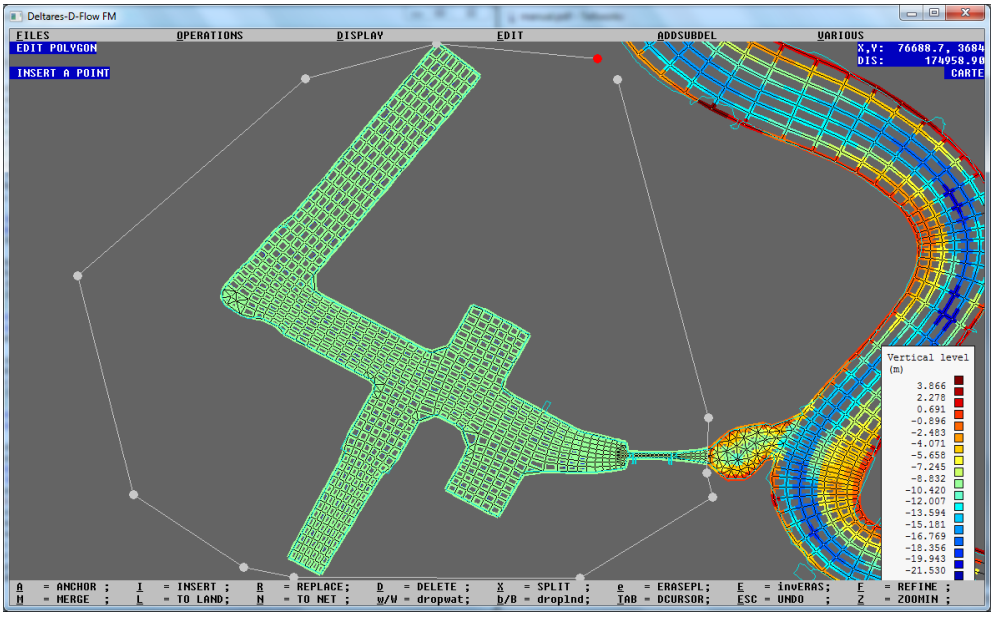


Figure 9.23: *Corrected bottom levels in the harbour area.*

tions that are constant along the polyline. If you keep the files <_0002.cmp> (etc.) files, you can demand for interpolation along polyline points.

6 Fill the file <boundarysea_0001.cmp> as shown in the upper panel of Figure 9.25. By means of this file, we intend to prescribe a harmonic waterlevel signal, consisting of two components. The first component has an amplitude equal to 0.5 m with a period of 0 minutes, i.e. a constant value of 0.5 m+NAP. The second component has an amplitude of 2.0 m with a period of 745 minutes, i.e. 12 hours and 25 minutes. Basically, the signal $h(t) = 2.0 \cdot \cos(2\pi t/745) + 0.5$ has been prescribed now, with h in meters w.r.t. NAP and t in minutes.

7 Fill the file <boundaryriver_0001.cmp> as indicated in the lower panel of Figure 9.25. The intention is to prescribe a constant discharge equal to 2500 m^3/s.

8 The next action comprises the coupling of the `cmp`-files with the model, via an `ext`-file. The `ext`-file has already been provided: <westerscheldt.ext>. In lines 1 up to 45, the contents of the file are explained. In the first lines, the several types of boundary conditions are listed. The key issue is the coupling of the polyline-file to the type of boundary condition. As long as the name of the `pli`-file is the same as the same of the `cmp`-file, the link is automatically laid between the polyline and the quantitative description of the boundary condition.

9.3.3 Tutorial 10: Initial conditions

Obviously, initial conditions are required when running a computation. For the sake of simplicity, we will just use an initial waterlevel of 0.5 m+NAP, being equal to the mean value of the sea water level boundary. The following actions should be undertaken:

1 Choose *Various* → *Change flow geometry parameters*. Insert the value 0.5 in the top cell (parameter *sini*).

2 Choose *Operations* → *(Re) initialise flow model geometry*. At the top of the screen, quite some computational parameters have appeared. The initial waterlevel has been set to 0.5 m+NAP.

3 Click on *Files* → *Save MDU-file* and choose for an appropriate filename. This `mdu`-file contains all the information that is required by the computational core to actually conduct the computation.

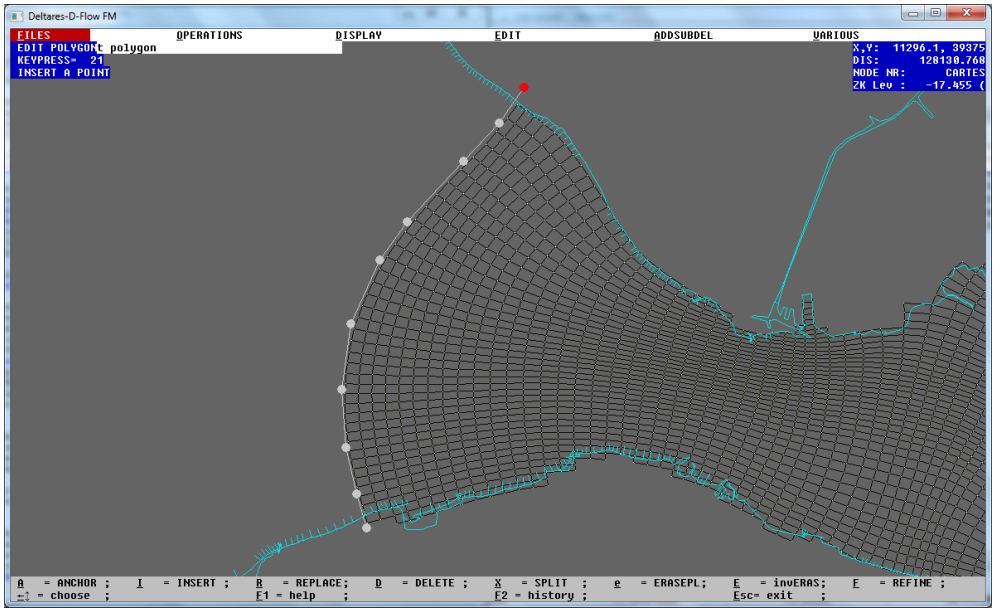


Figure 9.24: *Polyline along the sea boundary.*

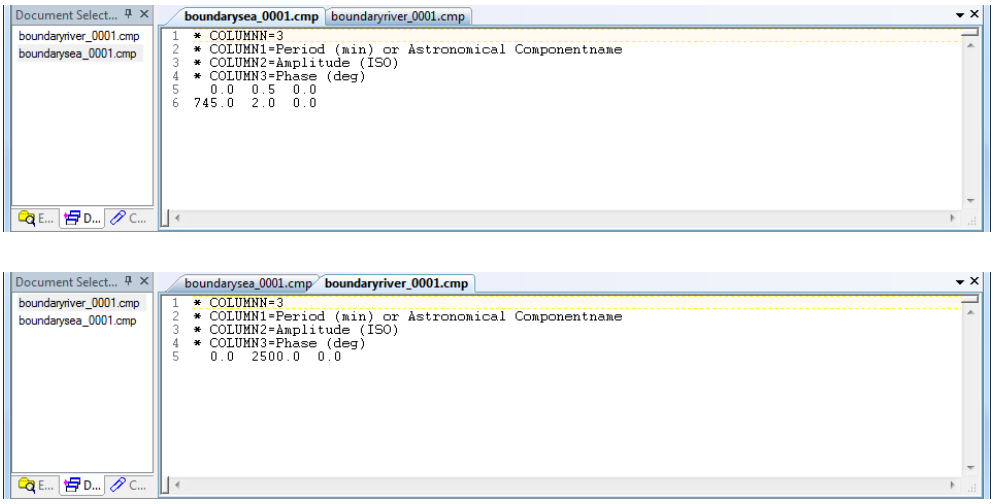


Figure 9.25: *Files with the prescription of boundary conditions, both for the sea (upper panel) and for the river (lower panel).*

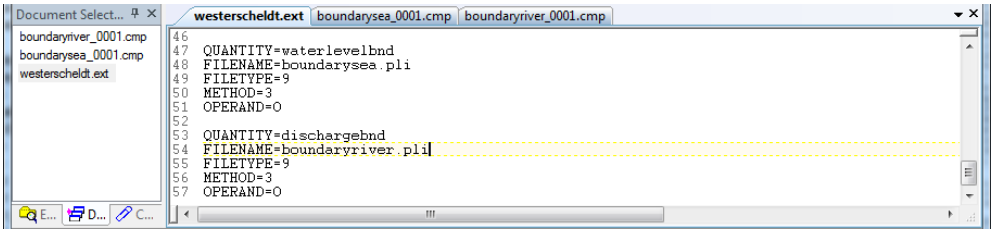


Figure 9.26: *File in which the boundary conditions are actually assigned.*

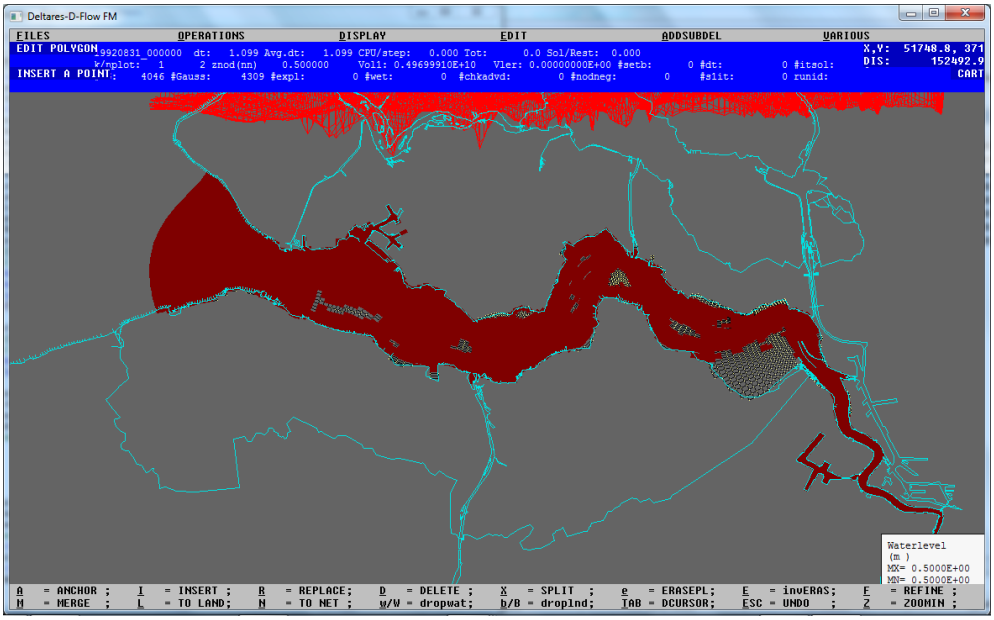


Figure 9.27: *Screen after having set the initial waterlevel.*

9.3.4 Tutorial 11: Cross-sections and observation points

Often, one would like to monitor computational outcomes at certain specific cross-sections or locations (i.e. 'observation points'). Inserting cross-sections and observation points is straightforward:

1 First the cross-sections. Choose *Edit* → *Edit polygon* and press the `i`-button. Draw as many cross-sections as you like. Separate each cross-section by pressing the `i`-button.
2 To save the cross-sections, choose *Edit* → *Save polygon* (so, *not* the *Save cross-sections* option!). Choose a filename and let it end with `_crs.pli`. In your working directory, several `cmp`-files have appeared. Those can be deleted as well, since obviously we do not deal with boundary conditions in these cases.
3 Choose *Addsubdel* → *Delete polygon*.
4 Then the observation points. Choose *Edit* → *Edit polygon* and press the `i`-button.
5 Draw a polyline. Each point of the polyline will turn to be an observation point.
6 Choose *Addsubdel* → *Copy polygon to ...* → *Copy polygon to observation points*.
7 Click on *Files* → *Save observation points*, choose an appropriate filename and let it end with `_obs.xyn`.
8 Choose *Addsubdel* → *Delete polygon*. Now the points of the polyline have turned into blue crosses.
9 Click on *Files* → *Load cross-sections* and choose the file you saved previously. Now, the cross-sections are loaded, the polylines inclusive. If you want so, you can delete the polylines via *Addsubdel* → *Delete polygon*. The cross-sections themselves will stay.

The resulting `_crs.pli`-file and `_obs.xyn`-file can be viewed with notepad, textpad or whatever editor you like. As an example, the contents of the file <crosssections_crs.pli> and the file <obslocations_obs.xyn> are shown in Figure 9.28.

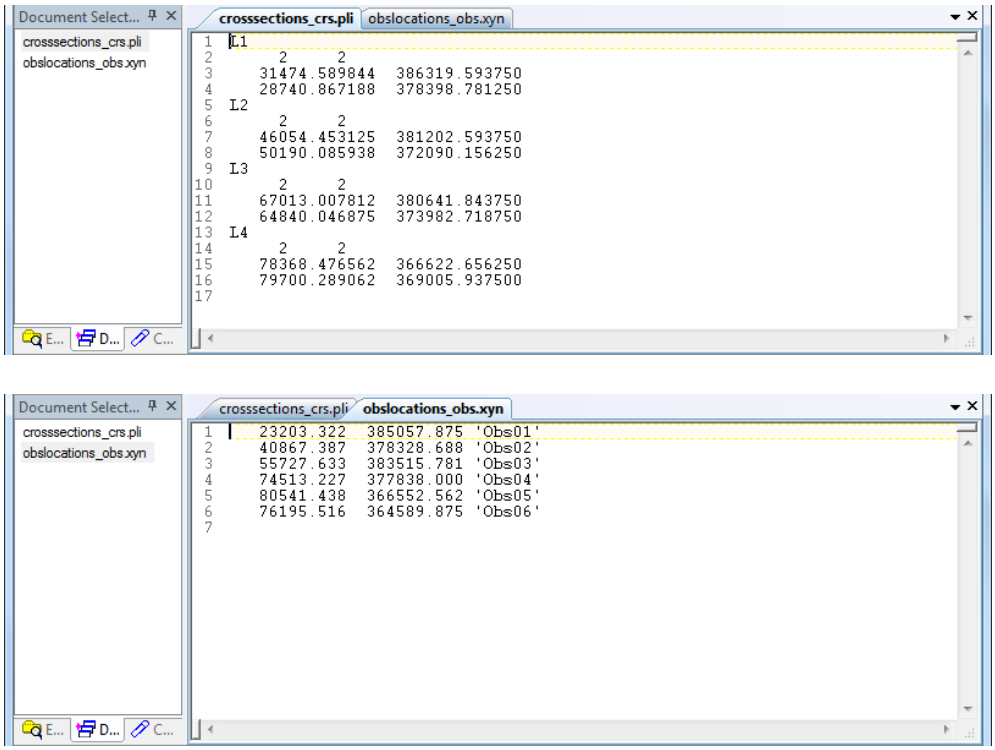


Figure 9.28: *Files with the information on the cross-sections and the observation points.*

The cross-sections file shows the coordinates of polylines that span the cross-sections. Obviously, each cross-sections can consist of more than 2 points. The observation points file just administrates the coordinates of each single observation point, including a specific name. Particularly the latter can be predefined manually instead of using the graphical interface.

9.3.5 Tutorial 12: The master definition file

We end this section with having a first look inside the master definition file, the `mdu`-file, we saved at the end of section 9.3.3. At this point, we only check the issues we have discussed this section. In the next section, we will check the master definition file in detail.

The master definition file is divided in several blocks, denoted by square brackets. Throughout this section, we have inserted boundary conditions, initial conditions and output locations (cross-sections and observation points). This information ends up at the following lines in the master definition file:

```
    [geometry]
10  NetFile             =  tutorial08_net.nc
13  LandBoundaryFile    =  sealand.ldb
19  WaterLevIni         =  0.5

    [external
    forcing]
71  ExtForceFile        =  westerscheldt.ext

    [output]
74  ObsFile             =  obslocations_obs.xyn
75  CrsFile             =  crosssections_crs.pli
```

Table 9.2: *Information in the `mdu`-file about the geometry, the boundary conditions, the initial conditions and the output files.*

Recall that the bathymetry data are stored in the network-file, i.e. <tutorial08_net.nc>. In the next section, we will fill in the rest of the master definition file with further information.

9.4 Running a computation

In the previous section, we have set up the hydrodynamic model. In this section, we will actually run the computation. In section 9.4.1, it is explained how the computation can be started. In section 9.4.2, it is briefly adressed how some output can be visualised within the D-Flow FM environment, developed so far. In section 9.5, it will be demonstrated how data can be visualised in external environments.

9.4.1 Tutorial 13: Starting the computation

In Table 9.2, a part of the master definition file has already been shown. First, we shall briefly discuss some input blocks of the `mdu`-file. The lower part of this file is shown in Figure 9.29.

The simulation timestep is computed by a CFL-based criterion. Subsequently, the parameter `DtUser` (line 59), given in seconds, prescribes the frequency with which snapshots are stored in the directory named in line 86.

In lines 63 and 64, the starting and ending times are specified. The unit in which these are specified can be inserted in line 58 via the parameter `Tunit`. In this case, minutes are chosen. Hence, the simulation will capture 745 minutes of simulated time, being 12 hours and 25 minutes, i.e. one tidal cicle.

The computation will deliver his-files and map-files. In his-files, timeseries are stored at the cross-sections (see line 75) and observation locations (see line 74), at a frequency specified in line 77 via the parameter `HisInterval`. The map-files collect data over the entire domain,

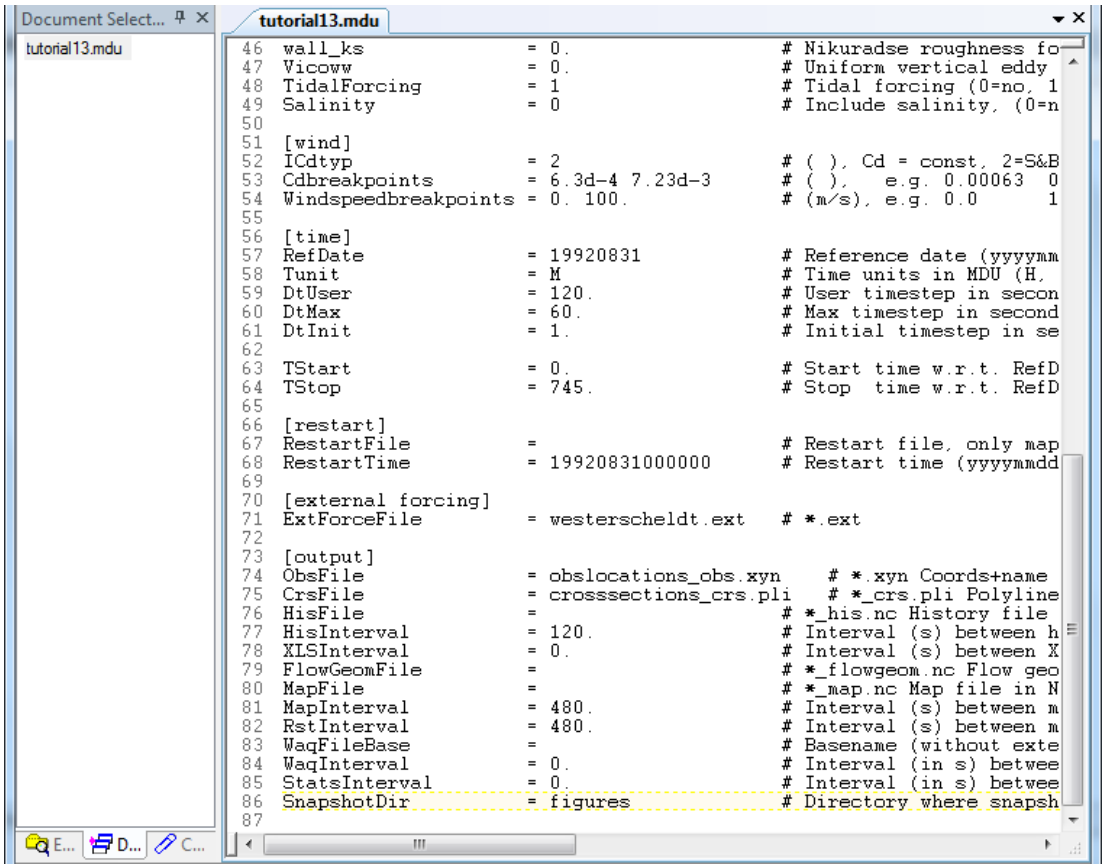


Figure 9.29: *The lower part of the* `mdu`*-file.*

at a frequency specified in line 81 via the parameter `MapInterval`. Be aware that the periods specified in lines 77 and 81 are clipped by the parameter `DtUser`. That means, if `DtUser` > `HisInterval`, then the period with which his-files are written is given by `DtUser`.

Restart files are written every `RstInterval` seconds (see line 82). This period is not clipped by `DtUser`. If you want to restart a computation from a certain time, you can also use map-files. If you want to do so, specify the name of `RestartFile` in line 67 and the value of `RestartTime` in line 68. Use the mentioned conventions. Be aware of the fact that a restart-file (extension `.rst`) is an ascii-type file, whereas his-files and map-files (extension `_his.nc` and `_map.nc`) are netcdf-files.

Computations can be launched as follows:

1. Double click on the `mdu`-file in your working directory, or load this file within the D-Flow FM environment.
2. Simply *right*-click on your mouse.
3. If you want to stop the simulation, double click no the *left* mouse button.
4. If you want to continue your computation, again *right*-click on your mouse. An alternative is to use the *spacebar*. The spacebar enables you to carry out the computation stepwise, i.e. just 1 timestep is done.

After a short time, the screen will typically look like shown in Figure 9.30. This figure shows waterlevels at the time `19920831_004051`, i.e. on August 31, 1992, at 40 minutes and 51

seconds after midnight (shown in the top left part of the screen).

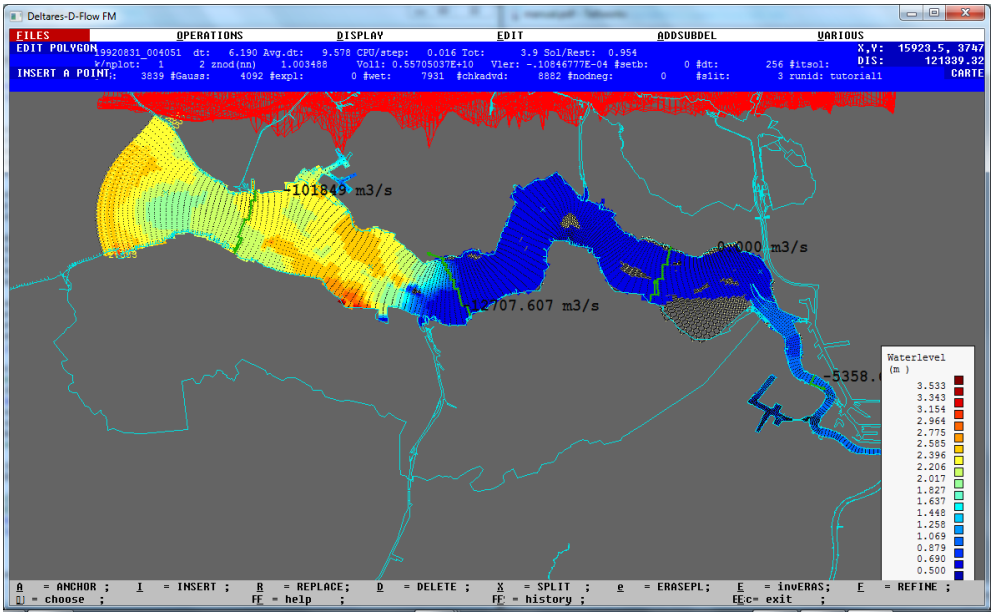


Figure 9.30: *Screendump of the D-Flow FM environment after a short time.*

9.4.2 Tutorial 14: Viewing the output within the shell environment

The menu *Display* provides quite some functionalities as regards viewing the data. When viewing the data, it is important to know *where* the several output values are computed. In Figure 1.2, we have introduced you into the concept of netnodes, netlinks, flownodes and flowlinks.

To get you familiar with the visualisation functionality of D-Flow FM, do the following:

1. Waterlevels are given at flownodes (cell centres). Disable their visualisation by choosing *Display* → *Values at flow nodes* → *NO*.
2. Choose *Display* → *Show all flow links in white* → *Show all flows*. The result is shown in Figure 9.31 .
3. You can again disable the visualisation of the flowlinks via *Display* → *Show all flow links in white* → *Do not show flowlinks*.
4. The red lines at the top of the screen display a sideview of the bathymetry and the water-level. Disable this view by choosing *Display* → *Show sideview* → *No rai*.
5. You can visualise the velocity in many ways. One way is to opt for *Display* → *Values at flow nodes* → *Node velocity magnitude*. You can also play around with the colourbar ticks via *Display* → *Change isocolour parameters*. See Figure 9.32 on page 86 for an example.

Notice that a his-file, a map-file and plenty of rst-files have appeared in your working directory.

9.5 Viewing the outcomes

In this section, we will briefly discuss the possibilities to visualise D-Flow FM data in other external environments, such as Google Earth and QuickPlot. In section 9.5.1, a coupling with Google Earth is laid. In section 9.5.2 demonstrate how the output can be read using QuickPlot.

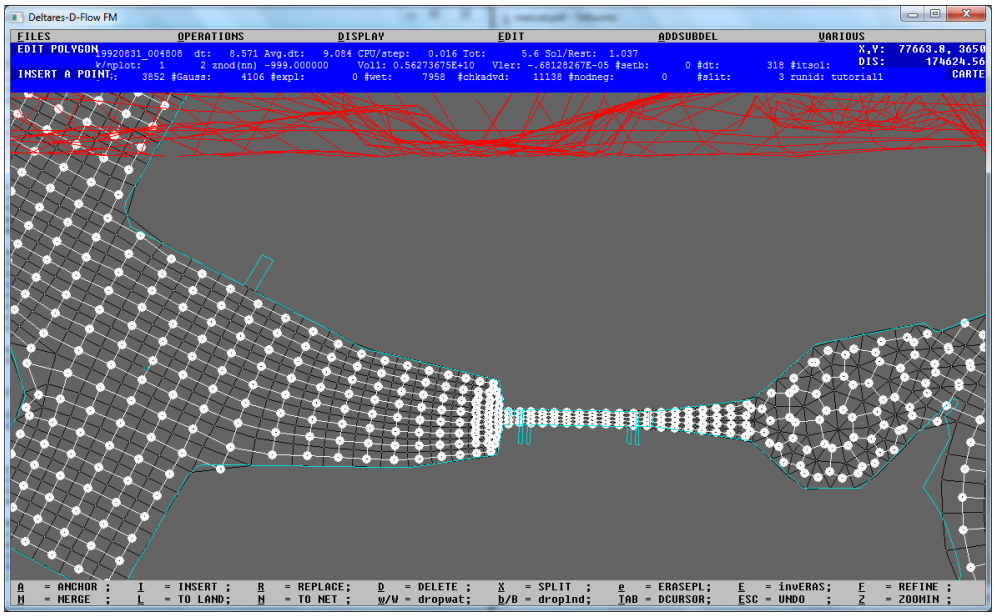


Figure 9.31: *The computational mesh. In black: netlinks, in white: the flowlinks (the lines) and the flownodes (the dots).*

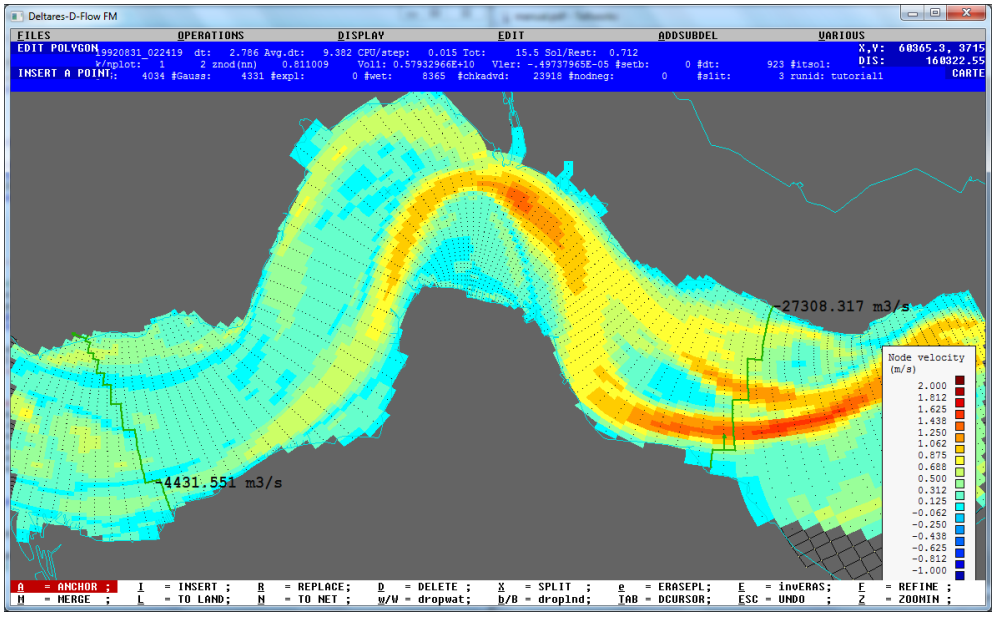


Figure 9.32: *The computational mesh. In black: netlinks, in white: the flowlinks (the lines) and the flownodes (the dots).*

9.5.1 Tutorial 15: Visualisation of the bathymetry in Google Earth

In the D-Flow FM environment, it is possible to export the mesh together with the bathymetry in `kml`-format, i.e. the format of Google Earth. The way to establish this, is as follows:

1. Load the network-file, i.e. the file that ends with `_net.nc`.
2. Apply a coordinate transformation. This is necessary, since D-Flow FM works with coordinates in meters with respect to Paris, whereas Google Earth works with longitudes and latitudes. Choose *Various* → *Coordinate transformation*.
3. The first entry, *Type of map projection*, has to be changed into `3`. Confirm by pressing `Enter`.

4 Choose *Files* → *Save network for Google Earth* and insert an appropriate filename.

Figure 9.33: *Visualisation of the bathymetry in Google Earth.*

The resulting `kml`-file can be opened and viewed in Google Earth. A part of the mesh, as well as the bottom levels, are shown in Figure 9.33. A funny thing is that it appears that a new dock has recently been built in the Antwerp harbour. Apparently, our landboundary file has not been up to date.

9.5.2 Tutorial 16: Viewing the output in QuickPlot

The output data from the computation is available within the directory *tutorial16*. The `_his.nc`-file contains timeseries, the `_map.nc` contains areal data, both in netcdf-format. The QuickPlot utiliy, known from Delft3D, has already been made suitable to read this D-Flow FM output.

The data can be viewed as follows:

1 Open QuickPlot and navigate to the directory *.../tutorial16*.
2 In the Station-listbox, you can select either an observation or a cross-section. Unless you have specified certain names to the output locations, the locations with names `Obs01`, etc. are available.
3 If you select the waterlevel, a typical visualisation picture will look like Figure 9.34. The timeseries is displayed over the the full 745 minutes, i.e. 12 hours and 25 minutes. The cosine signal for the waterlevel at the sea boundary is clearly transferred throughout the domain.
4 To visualise a `_map.nc`-file, open the file <tutorial13_map.nc>, also available in the directory *.../tutorial16*. Click around, and try to visualisize the velocity magnitude. The resulting picture will look like Figure 9.35. Figure 9.35 shows that the highest velocities are found near the sluice, which has to do with the narrowing width. The waterlevels can also be visualised. One should be aware that the bottomlevels are shown instead of waterlevels at dry locations.

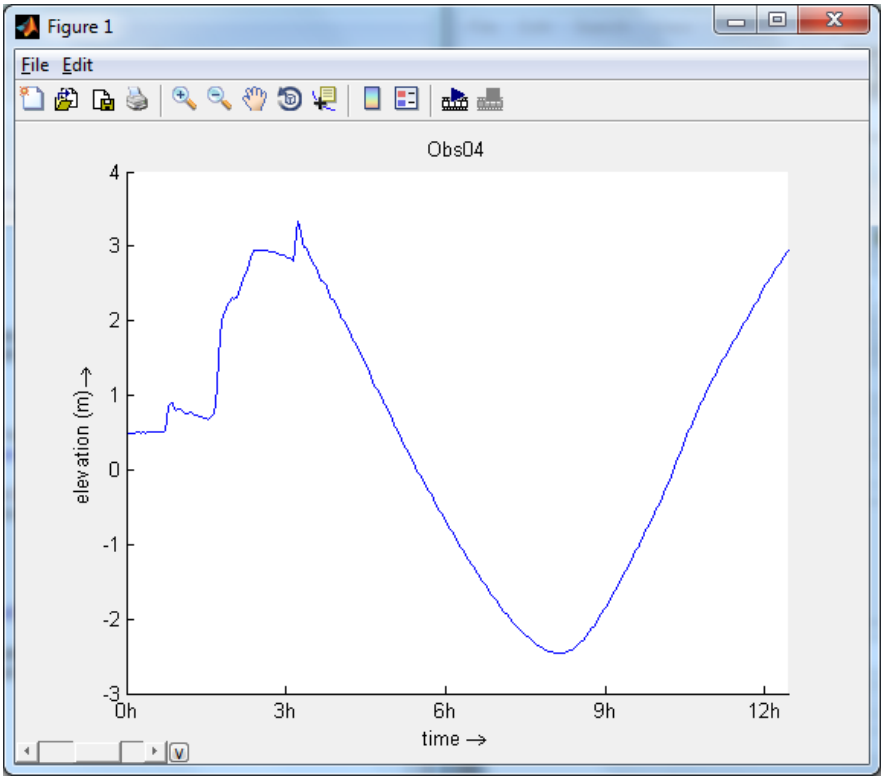


Figure 9.34: *Waterlevel timeseries at a certain location.*

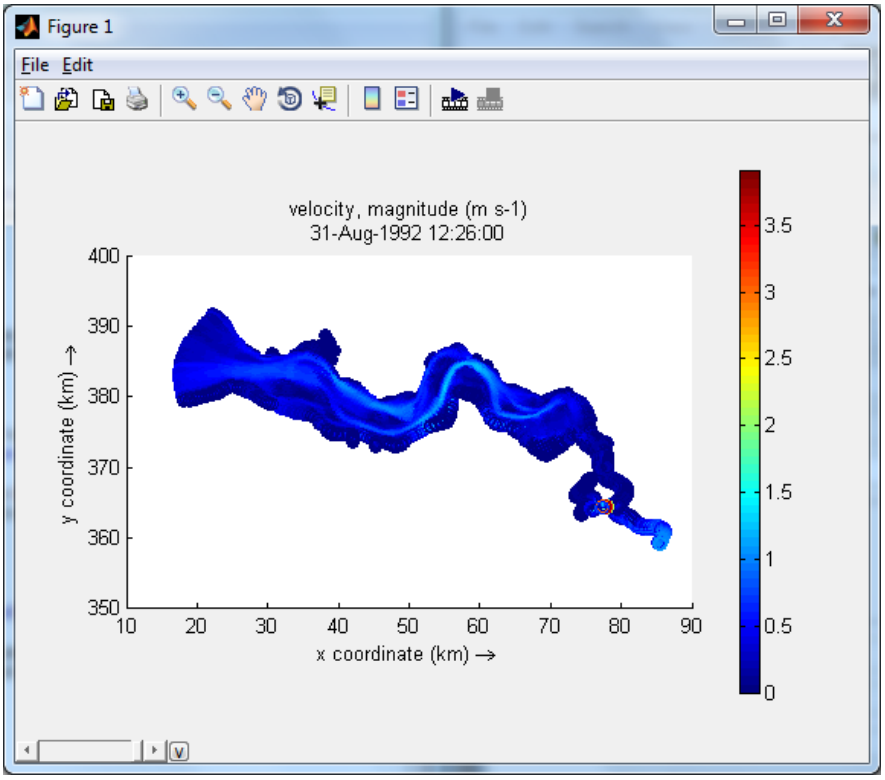


Figure 9.35: *Velocity map at a certain moment in time.*

10 Grid generation in RGFGRID

In the previous chapter, the key grid generation functionalities have been demonstrated through tutorials. These exercises have been established for the development GUI of D-Flow FM. Since the main building blocks of the grid generation engine have been incorporated in RGFGRID now, some elements of the tutorials are repeated in this chapter, but now for RGFGRID.

The following elements are highlighted in this chapter:

- ◇ Generating a curvilinear grid through 'grow curvilinear grid from splines'. This functionality has already been demonstrated in section 9.2.2 for the D-Flow FM development GUI, but is elaborated for RGFGRID in section 10.1.
- ◇ Generating a triangular grid. In section 9.2.3, it is described how triangular grids can be generated in the D-Flow FM development GUI by means of using samples. In section 10.2, its RGFGRID is discussed. For unstructured grids, RGFGRID uses a so-called moving front method (through the engine SEPRAN), thus different from the sample approach of the D-Flow FM development GUI.
- ◇ Coupling multiple distinc grids. In section 9.2.4, it is discussed how two grids – one structured, one unstructured – can be coupled. section 10.3 shows how grids can be put together in RGFGRID.

10.1 Generating a curvilinear grid from splines

D-Flow FM provides an improved method to generate curvilinear meshes directly from splines. In this method, a mesh is gradually developed from a base line of the channel towards the boundaries. This method requires less actions by the user and provides better orthogonality. As in the D-Flow FM development GUI, this functionality is present in RGFGRID. In this section, it is shown how to develop a grid in such a way by means of RGFGRID.

This approach can be illustrated as follows:

1. Load the land boundary file <scheldtharbour.ldb>.
2. Draw two cross-splines, intending to mark the inflow and outflow cross-section of the river part, through *Edit* → *Spline* → *New*.
3. Draw two additional splines, intended to loosely follow the riverbanks, in longitudinal direction.
4. Select one of the longitudinal splines and select the option *Edit* → *Spline* → *Spline to Land Boundary (new)*. The spline is now snapped to the land boundary. Repeat this action until you are satisfied with the result. For the second longitudinal spline, the actions can be repeated. The result of these actions is provided in the directory as <scheldtsplines.spl>.
5. To generate a curvilinear grid, choose *Settings* → *Grow grid from splines*. You will be able to set several settings of the operator. The upper 7 entries should be adapted into the values shown in Figure 10.1.
6. To be able to further amend the grid, choose *Operations* → *Convert grid* → *Regular to irregular*. Strictly, the grid is now not curvilinear anymore, but unstructured.
7. Choose *View* → *Grid property* → *Coloured edge* and then *Operations* → *Grid properties* → *Orthogonality*. Now, the orthogonality of the mesh is shown.
8. To improve the orthogonality, choose *Operations* → *Orthogonalise Grid*. The settings for the orthogonalisation procedure can be manipulated through the menu *Settings* → *Orthogonalisation (irregular)*. The result of the procedure is shown in Figure 10.2.

Figure 10.2 shows that the maximum value of the orthogonality is 0.056. You can change the model setup or orthogonalisation settings to further improve the orthogonality of the grid.

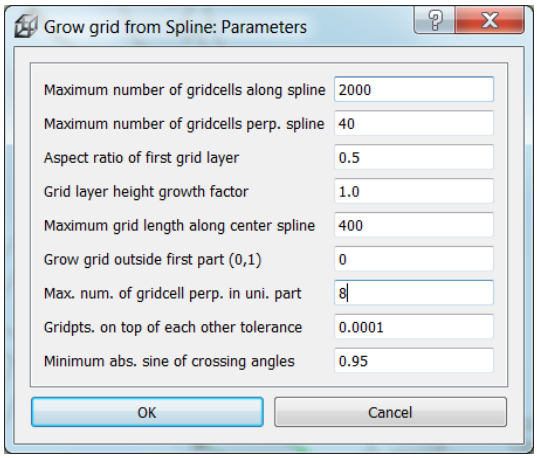


Figure 10.1: *Settings for the 'grow grid from splines' procedure.*

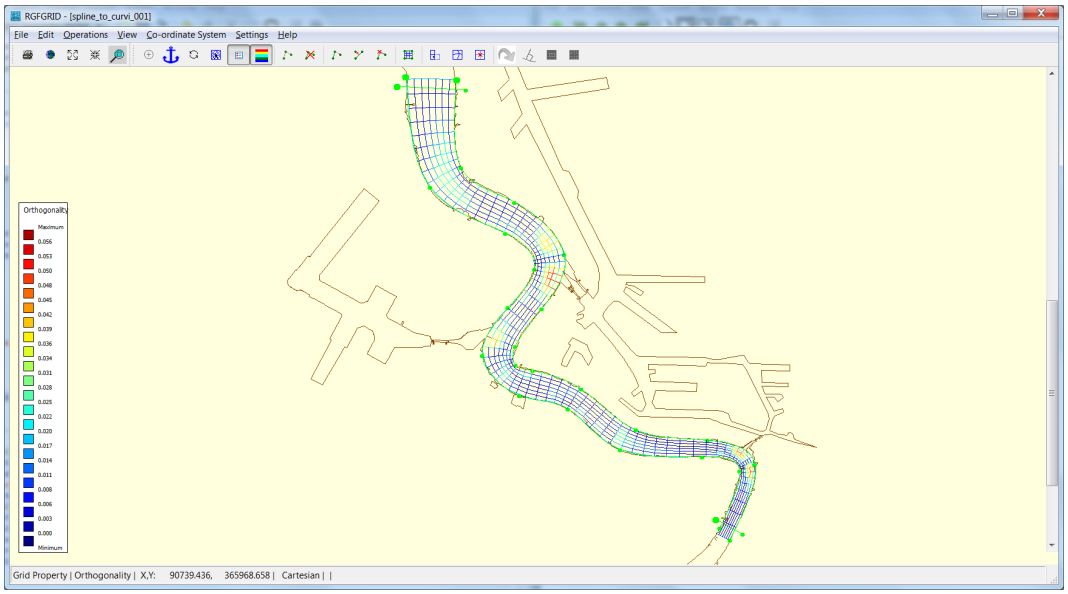


Figure 10.2: *Orthogonality of the generated curvilinear mesh after the new 'grow grid from splines' procedure.*

10.2 Generating a triangular grid

From the previous section, a curvilinear mesh is available for the Scheldt river. The river is separated from the harbour, west of the river, by a sluice. The small area between the sluice and the Scheldt will benefit from an unstructured mesh options of RGFGRID because of its irregular geometry. This irregular geometry is meshed in this section first and afterwards connected to the existing Scheldt river mesh.

The approach is as follows:

1 Click on *Edit* → *Polygon* → *New*. The intention is to mark the area of interest (i.e. the area that should be captured by the grid) through a polygon.
2 Start drawing a polygon at a distance of the order of a grid cell away from the curvilinear mesh. Let the second point be at a relatively small distance from the first one. This distance is later used as an indication of the size of the triangular grid cells to be placed.
3 Mark the elementary locations of the area (land boundary) and place the final point again at a distance of the order of a grid cell away from the river mesh.
4 Next, we choose *Edit* → *Polygon* → *Refine* and click on *two* points at the righthandside,

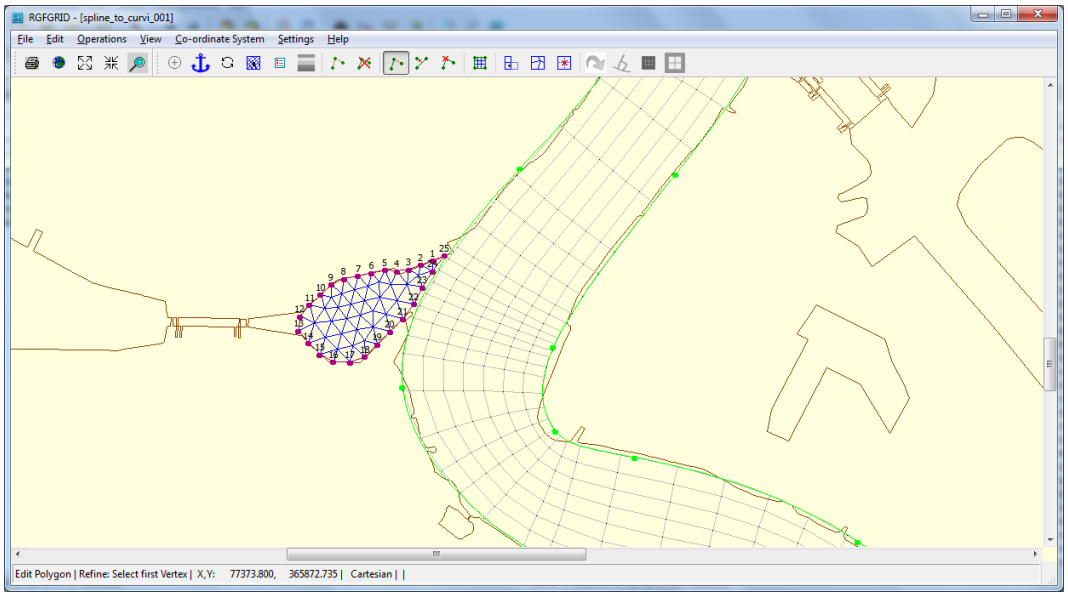


Figure 10.3: *Generated irregular grid within a polygon.*

located close to each other, and click on the *right mouse button*. Now, the polygon is divided into a finer set of line elements. Some remarks:

- ◇ The distance between the points of the polygon is derived from the distance of the two polyline segments at both sides of the *selected* segment. The length of the polyline segments varies linearly from the segment length at the one side of the selected segment towards the segment length at the other side of the selected segment.
- ◇ You can play around to see how this works. If needed, you can add extra polyline points by choosing *Edit* → *Polygon* → *Insert point*. Choose *Edit* → *Polygon* → *Move point* if a point move would make sense.
- ◇ You can snap the refined polygon to the land boundary through *Edit* → *Refine equidistant*. Then double-click on the polygon and press the `l`-key (lower case). The result is shown in Figure 10.3.

5 Choose *Operations* → *Create grid from polygon*. The result is shown in Figure 10.3.
6 Improve the orthogonality through *Operations* → *Orthogonalise grid*.
7 To further orthogonalise the grid by manipulating the settings, choose *Settings* → *Orthogonalisation (irregular)*, and then choosing *Operations* → *Orthogonalise grid* once again.

10.3 Coupling two distinct grids

The two yet distinct grids from the previous steps should properly be integrated into one single grid. Before we can couple the two grids, we should first make sure that the typical gridsize is of the same order of magnitude for both grids at the location where the connection is to be laid. Hence, basically two operations are to be done:

- ◇ Split the grid cells in the Scheldt river grid over the full width. Hence, the gridcell size in the river will match the grid cell size of the unstructured grid.
- ◇ Merge the two grids and put connections in between of these.

The splitting can be established as follows:

1 Select the river grid through *Edit* → *Select domain* and clicking the river grid.
2 Choose *Edit* → *Grid* → *Split row or column*.
3 Select the grid lines that should be split. Start at the left boundary, and apply multiple line

split operations towards the other side of the Scheldt river.

4 Try to achieve the picture shown in Figure 10.4 as regards the typical grid size in the curved area.

The merging part of the coupling schedule can be done as follows:

1 Choose *Edit* → *Allow Multi Select*. By now, you enable the option to select multiple grids.
2 Choose *Edit* → *Select domain* and click on the triangular part of the grid. As soon as you have clicked on it, both meshes are highlighted blue.
3 Merge the two seperate grids through *Operations* → *Merge grids*. Now, the grids have been merged.

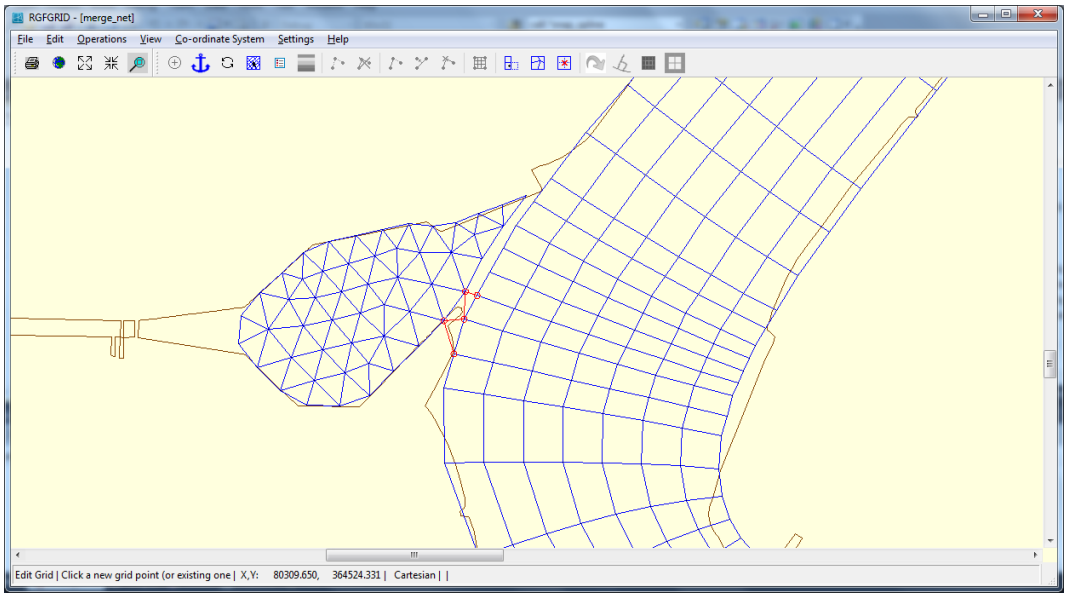


Figure 10.4: *Coupling of the two grids (regular and irregular, in blue) through manually inserting connecting grid lines (in red lines) between the two grids.*

4 As soon as the grids have been merged, new connections can be laid. Hence, choose *Edit* → *Grid* → *Edit* and then *Edit* → *Grid* → *Point insert*. Insert new gridlines in a zigzag-like style: see the red grid lines in Figure 10.4. Now, you will benefit from the (more or less) equal resolution in the river region as in the unstructured region.

11 Coupling with D-Water Quality (Delwaq)

11.1 Introduction

D-Water Quality is a multi-dimensional water quality model framework developed by Deltares over the past decades. It solves the advection-diffusion-reaction equation on a predefined computational grid and for a wide range of model substances. D-Water Quality offers flexible configuration of the substances to be included, as well as the processes to be evaluated. D-Water Quality is not a hydrodynamic model, so information on flow fields is obtained from hydraulic models such as Delft3D-FLOW or SOBEK (**??**) and also from D-Flow FM.

11.2 Offline versus online coupling

Two types of couplings between the hydrodynamics and water quality are distinguished: *offline* and *online* coupling. An offline coupling runs the entire hydrodynamic simulation first, i.e., separately from the entire water quality simulation. The file-based hydrodynamic output serves as input for the water quality simulation. As such, the hydrodynamic flow drives the water quality simulation, but there is no feedback from the water quality processes on the hydrodynamic simulation. For many applications, this is good practice.

An online coupling, on the other hand, exchanges data between hydrodynamics and water quality every time step. This tight coupling allows for direct feedback of the various processes onto one another. This coupling is scheduled for a future release of D-Flow FM.

11.3 Creating output for D-Water Quality

Creating output for D-Water Quality by D-Flow FM can be enabled in the GUI by changing `WaqInterval` under menu *Various* → *Change flow time parameters*, or by changing `[output]`, `WaqInterval` in the MDU-file. The `WaqInterval` should be a whole number of seconds, and must be a multiple of `dt_user`. D-Flow FM will create a special output folder named <DFM_DELWAQ_⟨mdident⟩>. In this folder a <.hyd>-file will be created that gives an overview of the coupling with references to the files containing the hydrodynamic exchange data.

Load the <.hyd>-file in the D-Water Quality GUI from Delft3D to prepare the input for a water quality calculation. For further information on how to run D-Water Quality please consult its user manual **?** and use a standard installation of Delft3D-FLOW with D-Water Quality.

11.4 Current limitations

The coupling between D-Flow FM and D-Water Quality has currently been tested with good results for 2D and 3D meshes without domain decomposition. A future release will contain some minor improvements for regions with wetting and drying mesh cells.

The Delft3D D-Water Quality GUI fully supports coupling with results from D-Flow FM. Post-processing still has some limitations: producing map plots of 3D results in QuickPlot or GPP is not directly possible yet. A temporary alternative is to convert the 3D D-Water Quality result into a 2D result with a small tool <map3d22d.exe>. The resulting 2D map file has an additional number of substances representing each layer (counting from the top to the bottom).

Usage: `map3d22d.exe <name of the 3d input map-file> <name of the 2d output map-file> <number of layers in 3D>`

It can be convenient to create a small batch file to do this for you. The resulting 2D map-file

can be analyzed in QuickPlot, however moving through vertical layers or vertical profiles are not yet available.

11.5 Aggregation

It can be useful to model the water quality processes on a coarser mesh than the hydrodynamic grid, and for this grid aggregation can be used in the near future. An upcoming release of D-Flow FM will contain further details on this topic.

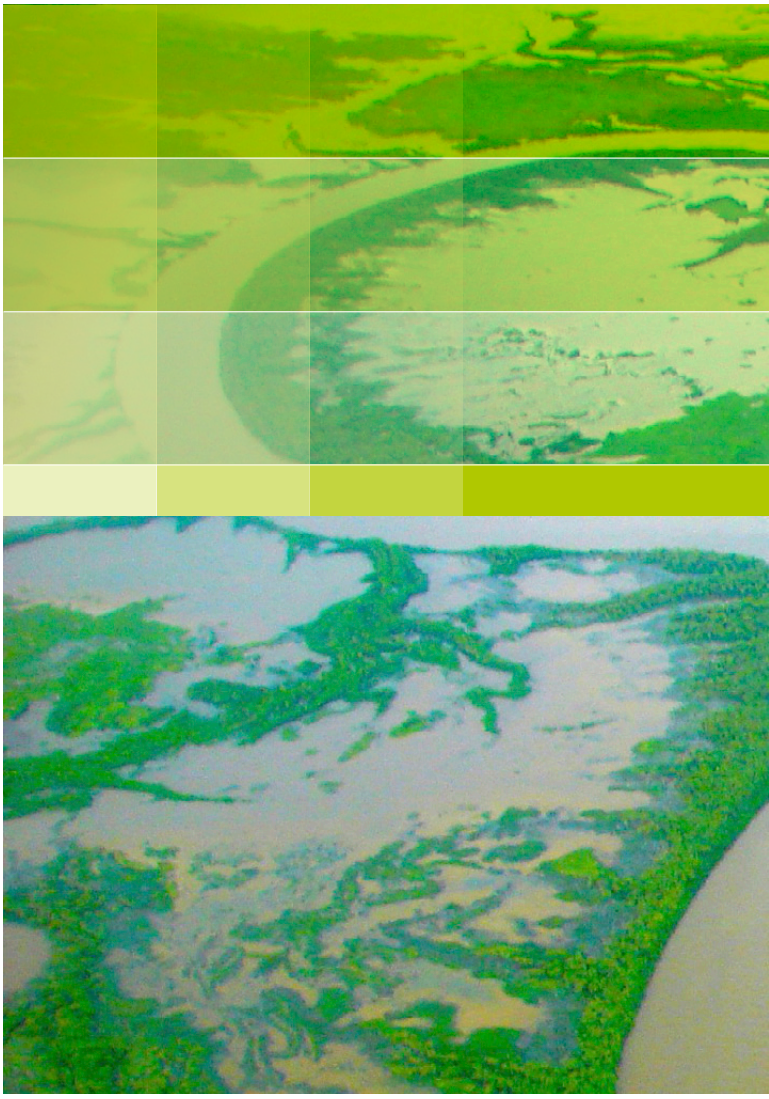
References

D-Flow FM TRM, 2015. *D-Flow Flexible Mesh Technical Reference Manual*. Deltares, Delft, 1.1.124 ed.

D-WAQ UM, 2013. *D-Water Quality User Manual*. Deltares, 4.02 ed.

Delft3D-FLOW UM, 2013. *Delft3D-FLOW User Manual*. Deltares, 3.14 ed.

SOBEK UM, 2013. *SOBEK User Manual*. Deltares, 0.00 ed.



Deltares

PO Box 177
2600 MH Delft
Rotterdamseweg 185
2629 HD Delft
The Netherlands

T +31 (0)88 335 82 73
F +31 (0)88 335 85 82

dflowfm.support@deltares.nl
http://public.deltares.nl/display/dflowfm