

SCHISM v5.3 Manual

June 5, 2016

Table of Contents

SCHISM v5.3 Manual	1
Chapter 1 Overview	5
1.1 Major Characteristics of SCHISM	5
1.2 Modeling system & application areas.....	5
1.3 Main differences between SCHISM and SELFE	5
1.4 How to read this manual.....	6
1.5 Notations used in this manual	6
1.6 Download SCHISM and compilation	7
1.7 Compilation.....	8
Chapter 2 Physical Formulation	9
2.1 Governing equations	9
2.2 Boundary conditions (B.C.)	10
2.3 Turbulence closure.....	11
2.4 Air-sea exchange model.....	12
Chapter 3 Numerical formulation.....	13
3.1 Geometry and discretization	13
3.2 Barotropic solver.....	19
3.3 Eulerian-Lagrangian Method (ELM)	28
3.4 Momentum equation	33
3.5 Vertical velocity.....	34
3.6 Turbulence closure.....	34
3.7 Transport equation	36
3.7.1 Upwind.....	36
3.7.2 TVD (explicit).....	37
3.7.3 TVD ²	38
3.7.4 Horizontal B.C. for transport	41
3.8 Updating the levels/inundation	41
3.9 Spherical coordinates	43
Chapter 4 SCHISM I/O.....	45
4.1 Type of inputs for SCHISM.....	45

4.2 Mandatory inputs	45
4.2.1 hgrid.gr3.....	46
4.2.2 vgrid.in.....	48
4.2.3 bctides.in.....	50
4.2.4 param.in	54
4.3 Optional inputs	63
4.3.1 .gr3, hgrid.ll.....	63
4.3.2 .th (ASCII)	63
4.3.3 .th (binary).....	64
4.3.4 .prop	64
4.3.5 .ic.....	65
4.3.6 .in	66
4.3.7 sflux/	67
4.4 Outputs	70
4.4.1 Run info output (mirror.out)	70
4.4.2 Global output	72
4.4.3 Station outputs	76
4.4.4 Warning and fatal messages.....	77
Chapter 5 Using SCHISM	78
5.1 Grid generation	78
5.1.1 Beware of CFL number.....	78
5.1.2 Check CFL number in xmgredit5	80
5.1.3 Grid quality.....	81
5.1.4 General rules	81
5.1.5 Channels.....	82
5.1.6 Grid near wetting and drying	83
5.1.7 Post SMS checks.....	85
5.1.8 Barotropic simulation	86
5.1.9 Baroclinic simulation.....	86
5.3 Pre-processing.....	86
5.4 Visualization	87
5.5 Post-processing	87

5.6 Simple test cases	88
5.6.1 Quarter Annulus.....	88
5.6.2 Lock exchange	90
5.7 Trouble shooting and FAQ	91
Chapter 6 SCHISM code structure.....	93
6.1 General info	93
6.2 For developers working on the code	95
6.2.1 Rules for contributing your code to be included in the SCHISM package	95
6.2.2 I/O channels in SCHISM.....	95
6.2.3 Notes on SCHISM Code	96
6.2.4 Working on your own code.....	106
Chapter 7 Modules	109
7.1 Hydraulics	109
7.2 Particle tracking	109
7.3 3D Sediment model (SED3D).....	110
7.4 2D Sediment model (SED2D).....	111
7.5 WWM	111
References.....	112

Chapter 1 Overview

SCHISM (Semi-implicit Cross-scale Hydroscience Integrated System Model) is a 3D seamless cross-scale model grounded on unstructured grids (UG) for hydrodynamics and ecosystem dynamics. SCHISM is a derivative work from the original SELFE model (v3.1dc as of Dec. 13, 2014). SCHISM has been implemented by Dr. Joseph Zhang (College of William & Mary) and other developers around the world, and licensed under Apache. SELFE was developed at the Oregon Health Sciences University. However, there are now significant differences between the two models.

1.1 Major Characteristics of SCHISM

- Finite element/volume formulation
- Unstructured mixed triangular/quadrangular grid in the horizontal dimension
- Hybrid SZ coordinates or new LSC² in the vertical dimension
- Polymorphism: a single grid can mimic 1D/2DV/2DH/3D configurations
- Semi-implicit time stepping (no mode splitting): no CFL stability constraints → numerical efficiency
- Robust matrix solver
- Higher-order Eulerian-Lagrangian treatment of advection
- Natural treatment of wetting and drying suitable for inundation studies
- Mass conservative and monotonicity-enforced transport: TVD²

1.2 Modeling system & application areas

- 3D baroclinic cross-scale lake-river-estuary-plume-shelf-ocean circulations
- Tsunami hazards
- Storm surge
- Sediment transport
- Ecology & water quality
- Oil spill
- Short wave-current interaction

1.3 Main differences between SCHISM and SELFE

- 1) Vertical grid system (LSC², Zhang et al. 2015);

- 2) Mixed triangular-quadrangular horizontal grid;
- 3) Implicit advection scheme for transport (TVD²);
- 4) Advection scheme for momentum: optional higher-order kriging with ELAD filter;
- 5) A new horizontal viscosity scheme (including bi-harmonic viscosity) to effectively filter out inertial spurious modes without introducing excessive dissipation.

Citation

We suggest the following language for citing the model:

SCHISM is a derivative product built from the original SELFE (v3.1dc; Zhang and Baptista 2008), with many enhancements and upgrades including new extension to large-scale eddying regime and a seamless cross-scale capability from creek to ocean (Zhang et al. 2016).

- Zhang, Y. and Baptista, A.M. (2008) SELFE: A semi-implicit Eulerian-Lagrangian finite-element model for cross-scale ocean circulation", *Ocean Modelling*, 21(3-4), 71-96.
- Zhang, Y., Ye, F., Stanev, E.V., Grashorn, S. (2016). Seamless cross-scale modeling with SCHISM, *Ocean Modelling*, 102, 64-81. doi:10.1016/j.ocemod.2016.05.002

1.4 How to read this manual

This manual contains detailed information on physical and numerical formulations, as well as usage for SCHISM. For beginners, we suggest you familiarize yourself with the basic notations in Chapters 2 and 3 but skip some details in those 2 chapters. Chapters 4 & 5 describe how to set up the model, including grid generation, and so should be read carefully, in consultation with appropriate sections in the previous 2 chapters.

Since SCHISM is quite a sophisticated package, we strongly recommend you start from simple test cases in Chapter 5.5 and gradually progress toward more complex 3D baroclinic applications.

1.5 Notations used in this manual

We will use bold characters to denote vectors and matrices, and unbold characters to denote scalars in mathematical equations. In addition, superscripts usually denote time step and subscripts denote spatial locations. E.g., $T_{i,k}^{n+1}$ may mean the temperature at step $n+1$ (i.e., new time step) and prism (i,k) , where i is the element number and k is the (whole) vertical index. We will use red words to denote input file names (e.g. **param.in**), purple words to denote output file

names (e.g. **mirror.out**), and green words to denote parameters specified in param.in; e.g. **bfric** (the bottom friction option flag). Code/pseudo-code fragments are written in *italic*.

Below are some notations used in this manual:

N_p : # (number) of nodes

N_s : # of sides

N_e : # of elements

N_z : maximum # of vertical levels

$i34(j)$: type of an element j (3: triangle; 4: quad)

$Nb(i)$: # of surrounding elements of a node i ;

$kbp(i)$: bottom index as seen by a node i

$kbs(i)$: bottom index as seen by a side i

$kbe(i)$: bottom index as seen by an element i

A : area of an element

Δz : layer thickness (at a node, side or elem.)

δ_{ij} : Dirac's Delta function (=1 when $i=j$; 0 otherwise)

1.6 Download SCHISM and compilation

SCHISM modeling system is distributed via svn. To download the source code:

- Make sure your system has svn v1.6 or above (see <http://svnbook.red-bean.com/> for a manual for using svn). Linux/unix svn works best;
- cd into a directory where you want to store the code, and issue the following command:
svn co <http://www.ccrm.vims.edu/schism/branches/v5.3>
user name: anonymous
passwd: you don't need passwd as this is an anonymous site
- You also have free access to the following directories:
 1. http://www.ccrm.vims.edu/schism/schism_verification_tests: test suite we use to test the **trunk** version. Due to differences between the release branch and trunk, you may need to modify some input files (e.g. **param.in**) but we hope these tests show you how we set up models for different problems;
- Remember to svn update to get the latest. We'll send out an announcement via mailing list when a newer version becomes available.

- useful user info can be found in src/beta_notes (including change of format for input files and bug fixes).

1.7 Compilation

You need the following two directories for gnu make. In addition, your system must have a FORTRAN and C compiler (and MPI wrapper like mpif90, mpicc), netcdf (version 3.x preferred) and perl installed.

- src/

Makefile is the main makefile, and in general you should NOT change this file.

- mk/

src/Makefile needs Make.defs.local in this directory. To help you design this file, we have put Make.defs.* which are from various clusters around the world. You should copy one of these to Make.defs.local; e.g., if your MPI compiler is Intel based ifort, you may try:

```
cp Make.defs.tsunami Make.defs.myown
```

```
ln -sf Make.defs.myown Make.defs.local
```

Then you need to edit Make.defs.local for e.g. the MPI compiler name, path names for netcdf library on your local cluster, and name of the executable (EXEC). Also the graph partitioning lib ParMETIS is compiled each time together with the SCHISM code, and you need to update the MPI C compiler names in src/ParMetis-3.1-Sep2010/Makefile.in (consult also "INSTALL" inside the ParMETIS directory). Lastly, turn on/off modules in *include_modules* (note that *TVD_LIM* should always have a valid value).

After this is done:

```
cd ../src
make clean
make
```

The final name of executable may have additional suffixes if you turn on modules.

Chapter 2 Physical Formulation

2.1 Governing equations

We will focus only on the hydrostatic solver in side SCHISM. Under this mode, we solve the standard Navier-Stokes equations with hydrostatic and Boussinesq approximations:

$$\text{Momentum equation: } \frac{D\mathbf{u}}{Dt} = \frac{\partial}{\partial z} \left(\nu \frac{\partial \mathbf{u}}{\partial z} \right) - g\nabla\eta + \mathbf{f}, \quad (2.1)$$

$$\mathbf{f} = f(v, -u) - \frac{g}{\rho_0} \int_z^\eta \nabla \rho d\zeta - \frac{\nabla p_A}{\rho_0} + \alpha g \nabla \Psi + \mathbf{F}_m + \text{other}$$

Continuity equation in 3D and 2D depth-integrated forms:

$$\nabla \cdot \mathbf{u} + \frac{\partial w}{\partial z} = 0, \quad (2.2)$$

$$\frac{\partial \eta}{\partial t} + \nabla \cdot \int_{-h}^\eta \mathbf{u} dz = 0, \quad (2.3)$$

Transport equations:

$$\frac{\partial C}{\partial t} + \nabla \cdot (\mathbf{u}C) = \frac{\partial}{\partial z} \left(\kappa \frac{\partial C}{\partial z} \right) + F_h, \quad (2.4)$$

Equation of state:

$$\rho = \rho(S, T, p)$$

where

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right)$$

D/Dt	material derivative
(x,y)	horizontal Cartesian coordinates
z	vertical coordinate, positive upward
t	time
$\eta(x, y, t)$	free-surface elevation in [m]
$h(x, y)$	bathymetric depth (measured from a fixed datum) [m]
$\mathbf{u}(x, y, z, t)$	horizontal velocity, with Cartesian components (u, v) [m/s]
$w(x, y, z, t)$	vertical velocity [m/s]

p	hydrostatic pressure [Pa]
p_A	atmospheric pressure reduced to MSL [Pa]
ρ, ρ_0	water density and reference water density [kg/m^3]
\mathbf{f}	other forcing terms in momentum (baroclinic gradient, horizontal viscosity, Coriolis, earth tidal potential, atmospheric pressure, radiation stress). These are usually treated explicitly in the numerical formulation
g	acceleration of gravity, in [ms^{-2}]
C	tracer concentration (e.g., salinity, temperature, sediment etc)
ν	vertical eddy viscosity, in [m^2s^{-1}]
κ	vertical eddy diffusivity, for tracers, in [m^2s^{-1}]
\mathbf{F}_m	horizontal viscosity [m^2s^{-1}]
F_h	horizontal diffusion and mass sources/sinks [m^2s^{-1}]

2.2 Boundary conditions (B.C.)

The differential equations above need initial condition (I.C.) and B.C. In general, all state variables (η, \mathbf{u}, C) are specified at $t=0$ as I.C. and these are also specified at all open *lateral* boundary segments (open ocean, rivers etc). However, not all variables need to be specified at all boundary segments and we'll revisit this in Chapter 4.

The vertical B.C. for (2.1-2.4) are described in detail below as these impact the numerical scheme.

At the sea surface, SCHISM enforces the balance between the internal Reynolds stress and the applied shear stress:

$$\nu \frac{\partial \mathbf{u}}{\partial z} = \boldsymbol{\tau}_w, \text{ at } z = \eta \quad (2.5)$$

where the stress $\boldsymbol{\tau}_w$ can be parameterized using the approach of Zeng et al. (1998) or the simpler approach of Pond and Pickard (1998). (If the Wind Wave Model is invoked, it can also be calculated from the wave model).

Because the bottom boundary layer is usually not well resolved in ocean models, the no-slip condition at the sea or river bottom ($\mathbf{u} = \mathbf{w} = 0$) is replaced by a balance between the internal Reynolds stress and the bottom frictional stress,

$$\nu \frac{\partial \mathbf{u}}{\partial z} = \boldsymbol{\tau}_b, \text{ at } z = -h \quad (2.6)$$

The specific form of the bottom stress $\boldsymbol{\tau}_b$ depends on the type of boundary layer used and here we will only discuss the turbulent boundary layer below (Blumberg and Mellor 1987), given its prevalent usage in ocean modeling. The bottom stress is then:

$$\boldsymbol{\tau}_b = C_D |\mathbf{u}_b| \mathbf{u}_b. \quad (2.7)$$

The velocity profile in the interior of the bottom boundary layer obeys the logarithmic law:

$$\mathbf{u} = \frac{\ln[(z+h)/z_0]}{\ln(\delta_b/z_0)} \mathbf{u}_b, \quad z_0 - h \leq z \leq \delta_b - h \quad (2.8)$$

which is smoothly matched to the exterior flow at the top of the boundary layer. In Eq. (2.8), δ_b is the thickness of the bottom computational cell (assuming that the bottom is sufficiently resolved in SCHISM that the bottom cell is inside the boundary layer), z_0 is the bottom roughness, and \mathbf{u}_b is the velocity measured at the top of the bottom computational cell. Therefore the Reynolds stress inside the boundary layer is derived as:

$$\nu \frac{\partial \mathbf{u}}{\partial z} = \frac{\nu}{(z+h) \ln(\delta_b/z_0)} \mathbf{u}_b \quad (2.9)$$

Utilizing the turbulence closure theory discussed below, we can show that the Reynolds stress is constant inside the boundary layer:

$$\nu \frac{\partial \mathbf{u}}{\partial z} = \frac{\kappa_0}{\ln(\delta_b/z_0)} C_D^{1/2} |\mathbf{u}_b| \mathbf{u}_b, \quad z_0 - h \leq z \leq \delta_b - h \quad (2.10)$$

and the drag coefficient is calculated from Eqs. (2.6), (2.7), and (2.10) as:

$$C_D = \left(\frac{1}{\kappa_0} \ln(\delta_b/z_0) \right)^{-2} \quad (2.11)$$

which is the drag formula as discussed in Blumberg and Mellor (1987). Eq. (2.10) also shows that the vertical viscosity term in the momentum equation vanishes inside the boundary layer. This fact will be utilized in the numerical formulation.

2.3 Turbulence closure

Eqs. (2.1-2.4) are not closed and must be supplemented by turbulence closure equations for the viscosity/diffusivity. We use the Generic Length-scale (GLS) model of Umlauf and Burchard (2003), which has the advantage of encompassing most of the 2.5-equation closure models (k - ε (Rodi 1984); k - ω (Wilcox 1998); Mellor and Yamada 1982). In this framework, the transport, production, and dissipation of the turbulent kinetic energy (K) and of a generic length-scale variable (ψ) are governed by:

$$\frac{DK}{Dt} = \frac{\partial}{\partial z} \left(v_k^\psi \frac{\partial K}{\partial z} \right) + \nu M^2 + \mu N^2 - \varepsilon \quad (2.12)$$

$$\frac{D\psi}{Dt} = \frac{\partial}{\partial z} \left(v_\psi \frac{\partial \psi}{\partial z} \right) + \frac{\psi}{K} (c_{\psi 1} \nu M^2 + c_{\psi 3} \mu N^2 - c_{\psi 2} F_w \varepsilon) \quad (2.13)$$

where v_k^ψ and v_ψ are vertical turbulent diffusivities, $c_{\psi 1}$, $c_{\psi 2}$ and $c_{\psi 3}$ are model-specific constants (Umlauf and Burchard 2003), F_w is a wall proximity function, M and N are shear and buoyancy frequencies, and ε is a dissipation rate. The generic length-scale is defined as

$$\psi = (c_\mu^0)^p K^m \ell^n \quad (2.14)$$

where $c_\mu^0 = 0.3^{1/2}$ and ℓ is the turbulence mixing length. The specific choices of the constants p , m and n lead to the different closure models mentioned above. Finally, vertical viscosities and diffusivities as appeared in Eqs. (2.1, 2.4) are related to K , ℓ , and stability functions:

$$\begin{aligned} \nu &= \sqrt{2} s_m K^{1/2} \ell \\ \kappa &= \sqrt{2} s_h K^{1/2} \ell \\ v_k^\psi &= \frac{\nu}{\sigma_k} \\ v_\psi &= \frac{\nu}{\sigma_\psi} \end{aligned} \quad (2.15)$$

where the Schmidt numbers σ_k^ψ and σ_ψ are model-specific constants. The stability functions (s_m and s_h) are given by an Algebraic Stress Model (e.g.: Kantha and Clayson 1994, Canuto et al. 2001, or Galperin et al. 1988).

At the free surface and at the bottom of rivers and oceans, the turbulent kinetic energy and the mixing length are specified as Dirichlet boundary conditions:

$$K = \frac{1}{2} B_1^{2/3} |\boldsymbol{\tau}_b|^2, \quad (2.16)$$

$$\ell = \kappa_0 d_b \text{ or } \kappa_0 d_s, \quad (2.17)$$

where $\boldsymbol{\tau}_b$ is a bottom frictional stress, $\kappa_0 = 0.4$ is the von Karman's constant, B_1 is a constant, and d_b and d_s are the distances to the bottom and the free surface, respectively.

2.4 Air-sea exchange model

We use the bulk aerodynamic module of Zeng et al. (1998).

Chapter 3 Numerical formulation

3.1 Geometry and discretization

As a FE model, SCHISM uses flexible UG in the horizontal and flexible vertical coordinate systems in the vertical dimension. Hybrid triangular-quad elements are used in the horizontal, to take advantage of the superior boundary-fitting capability of triangles and efficiency/accuracy of quads in representing certain features like channels. Fig. 3.1 shows an example of horizontal grid. Fig. 3.2 shows our convention of numbering local nodes/sides etc.

In the vertical dimension, SCHISM supports the following 2 types of coordinates.

Hybrid SZ grid

The 1st type uses a hybrid S (terrain-following generalized s -coordinates; Song and Haidvogel 1994) and *shaved* z coordinates, with the latter always being placed underneath the former at a prescribed demarcation depth h_s . Note that the use of shaved z layers is optional and the users can use a pure S grid with a choice of h_s greater than the maximum depths in the grid. Fig. 3.3 shows a SZ grid.

The transformation from S to Z is given by:

$$\left\{ \begin{array}{l} z = \eta(1 + \sigma) + h_c \sigma + (\tilde{h} - h_c)C(\sigma) \quad (-1 \leq \sigma \leq 0) \\ C(\sigma) = (1 - \theta_b) \frac{\sinh(\theta_f \sigma)}{\sinh \theta_f} + \theta_b \frac{\tanh[\theta_f(\sigma + 1/2)] - \tanh(\theta_f/2)}{2 \tanh(\theta_f/2)} \quad (0 \leq \theta_b \leq 1; \quad 0 < \theta_f \leq 20) \end{array} \right. \quad (3.1)$$

where $\tilde{h} = \min(h, h_s)$ is a “restricted” depth, h_c is a positive constant dictating the thickness of the bottom or surface layer that needs to be resolved, and θ_b and θ_f are constants that control the vertical resolution near the bottom and surface. As $\theta_f \rightarrow 0$, the S coordinates reduce to the traditional σ -coordinates:

$$z = \tilde{H}\sigma + \eta, \quad (3.2)$$

where $\tilde{H} = \tilde{h} + \eta$ is the restricted total water depth. For $\theta_f \gg 1$, more resolution is skewed towards the boundaries, and the transformation becomes more nonlinear. If $\theta_b \rightarrow 0$, only the surface is resolved, not the bottom, while if $\theta_b \rightarrow 1$, both are resolved (Fig. 3.4). For typical coastal applications, we suggest $\theta_b=0$. Unfortunately, the S coordinate system becomes invalid in shallow depth $\tilde{h} < h_c$; under such circumstance, the traditional σ coordinates (3.2) are used.

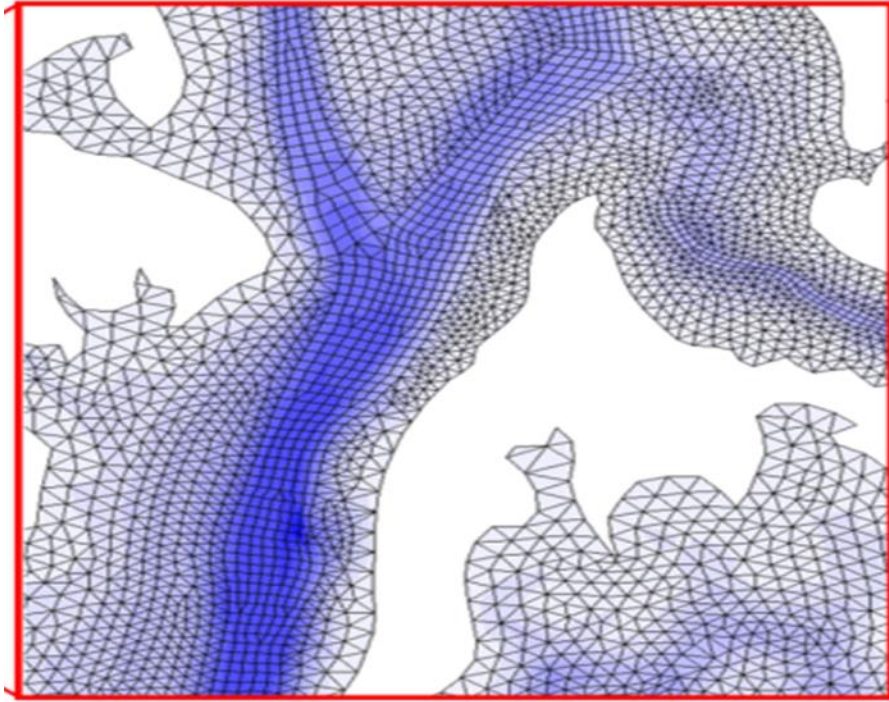


Fig. 3.1: Example horizontal grid (hgrid.gr3) of SCHISM.

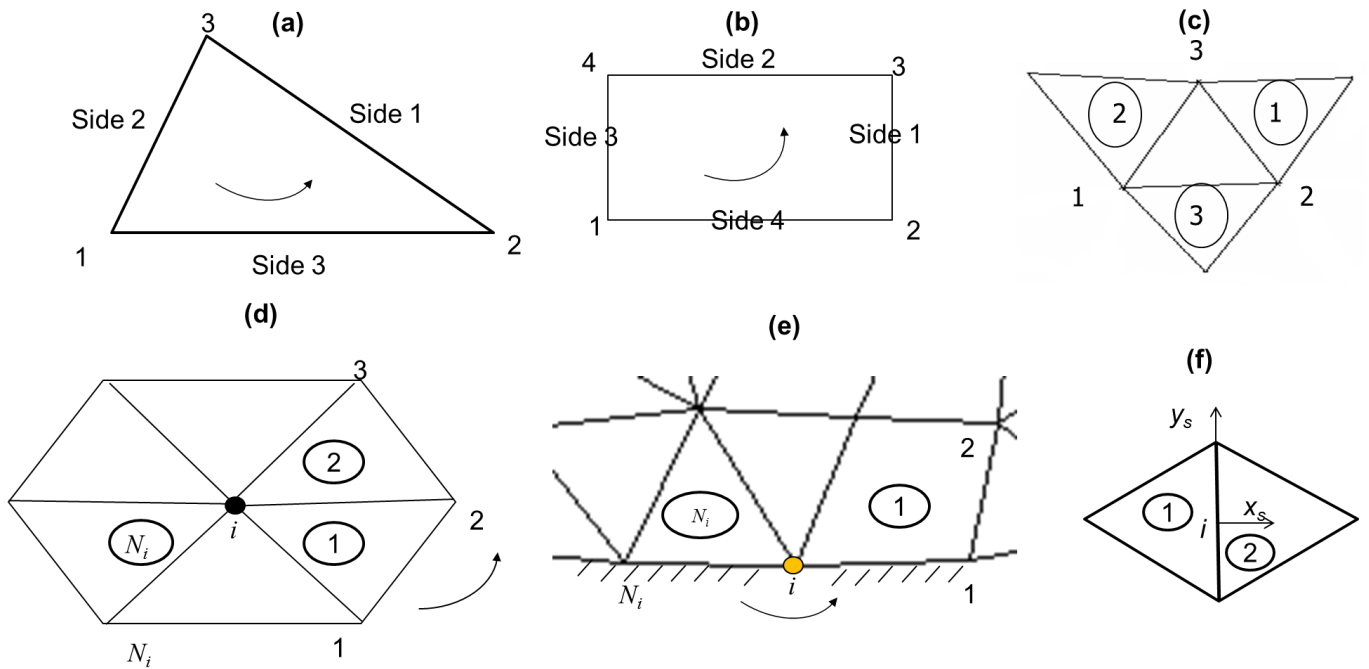


Fig. 3.2: Horizontal element convention. Local indices of nodes/sides for (a) triangular element, and (b) quad element. (c) Adjacent elements of an element (following side convention). (d,e): Local ball info (surrounding elements/nodes of node i) for (d) internal and (e) boundary node i . (f) shows a local side center-based frame used to calculate the normal fluxes; the (global)

element number of “1” is smaller than “2”, and x_s axis always points out of “1”. We follow counter-clockwise convention/right-hand rule.

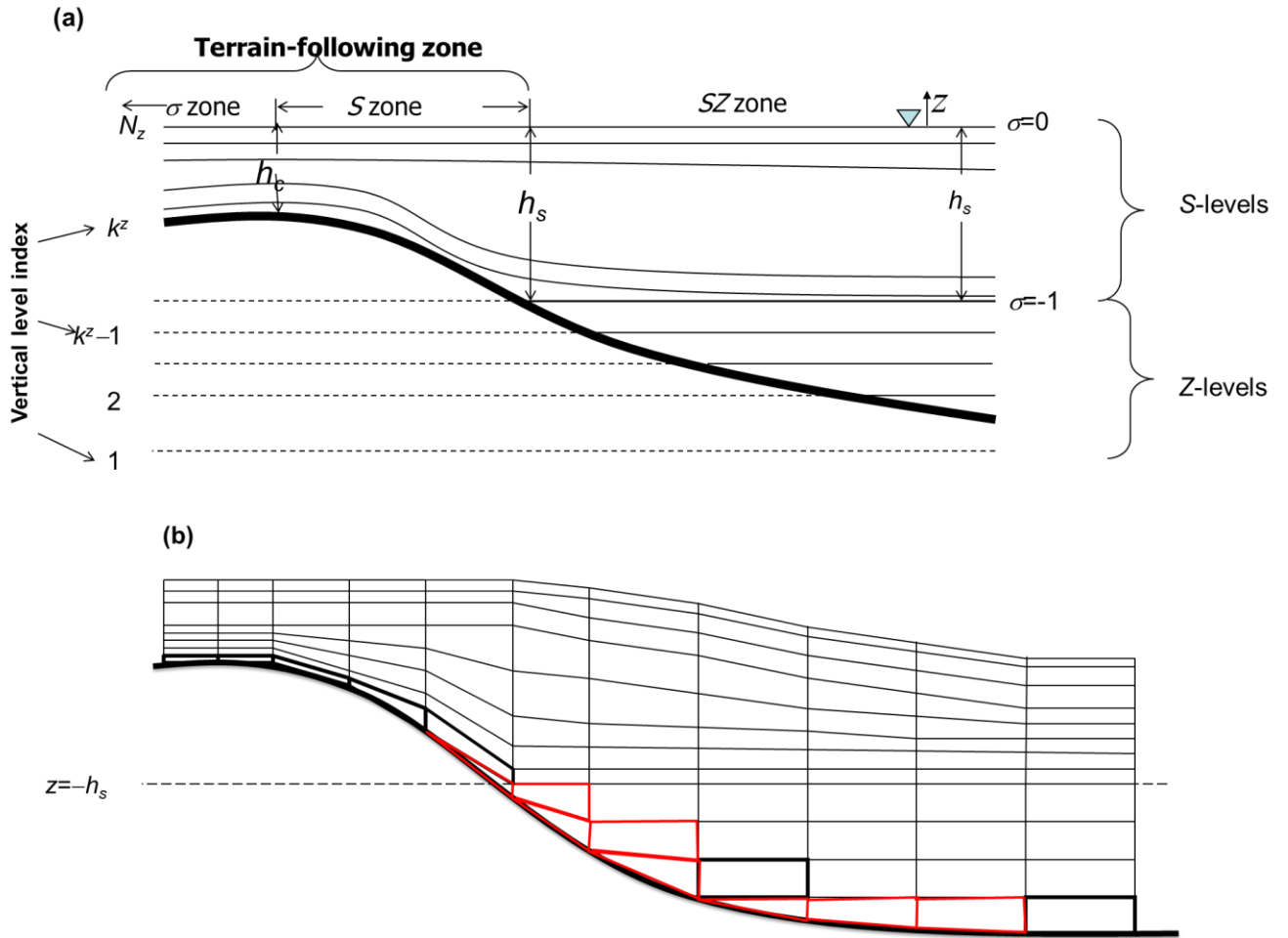


Fig. 3.3: Example of SZ grid. (a) Vertical coordinate systems (thick line is the bottom); (b) side view of a vertical transect resulted from SZ grid; the red cells are irregular/shaved cells.

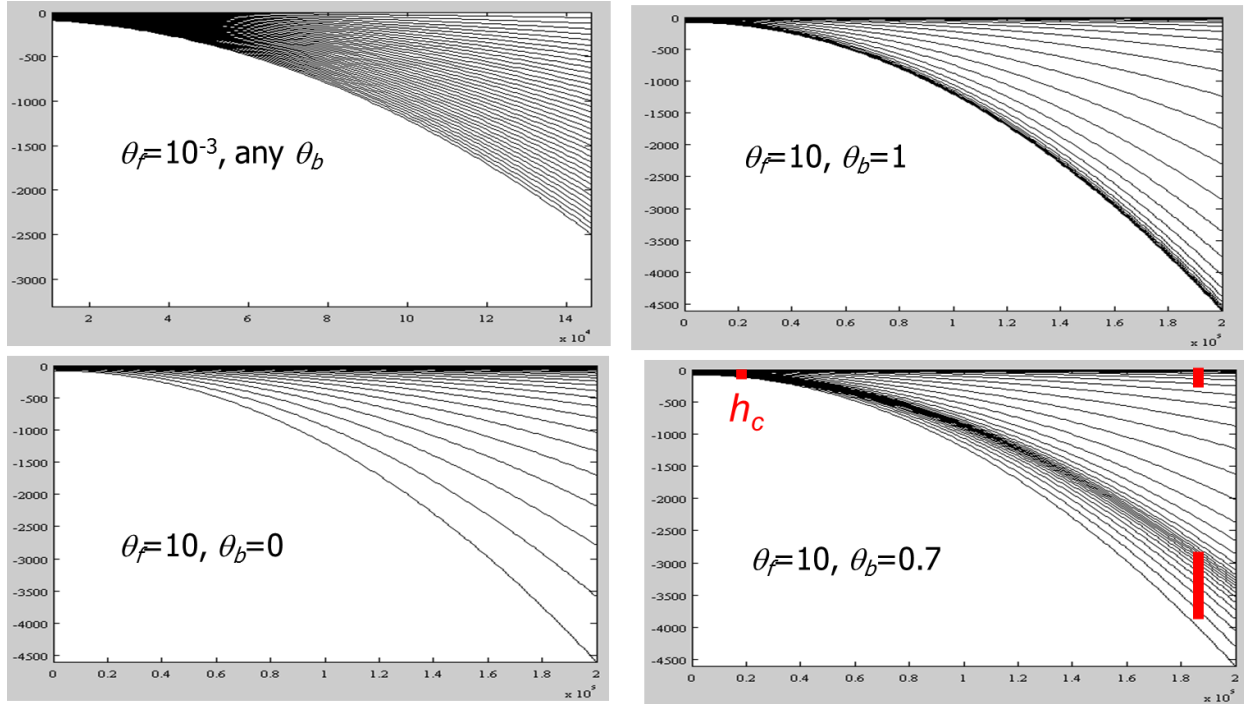


Fig. 3.4: Examples of S transformation.

LSC² grid

The 2nd type of vertical coordinate uses localized σ . Each grid node has its own vertical grid with a set of σ coordinates for maximum flexibility; the mismatch of # of vertical levels between adjacent nodes is treated with shaved cells near the bottom, thus the name Localized Sigma Coordinates with Shaved Cells (LSC²). Although there are many ways of generating a LSC² grid, a convenient start is the Vanishing Quasi Sigma (VQS) suggested by Dukhovskoy et al. (2009). The essence of VQS is to first design a master (vertical) grid at selected reference depths, and then use this master grid to interpolate the local z coordinates at a grid node based on the local depth and 2 reference depths that contains the depth (Fig. 3.5). The original VQS also requires masking of thin layers near the bottom, and is not free of Z-like staircases near the bottom (Fig. 3.5d). Since SCHISM is an implicit model, we do not need such special treatment, and furthermore, ‘fill in’ unmatched levels with shaved cells to get LSC². As can be seen from Fig. 3.5, a major advantage of LSC², besides being very economical (as fewer/more # of levels are used in shallow/deep), is that it resembles a Z grid near the surface and interior of the water column while behaving like terrain-following grid near the bottom. As demonstrated in Zhang et al. (2015, 2016ab), LSC² is instrumental in reducing unphysical diapycnal mixing and pressure-gradient errors (PGE). A well-designed LSC² grid completely removes the need for bathymetry manipulation (e.g. smoothing/clipping), thus enabling a faithful representation of the bathymetry/topography, as demonstrated in those papers. However, some experience is required in designing a good LSC² grid, and so beginners may want to start with the SZ grid.

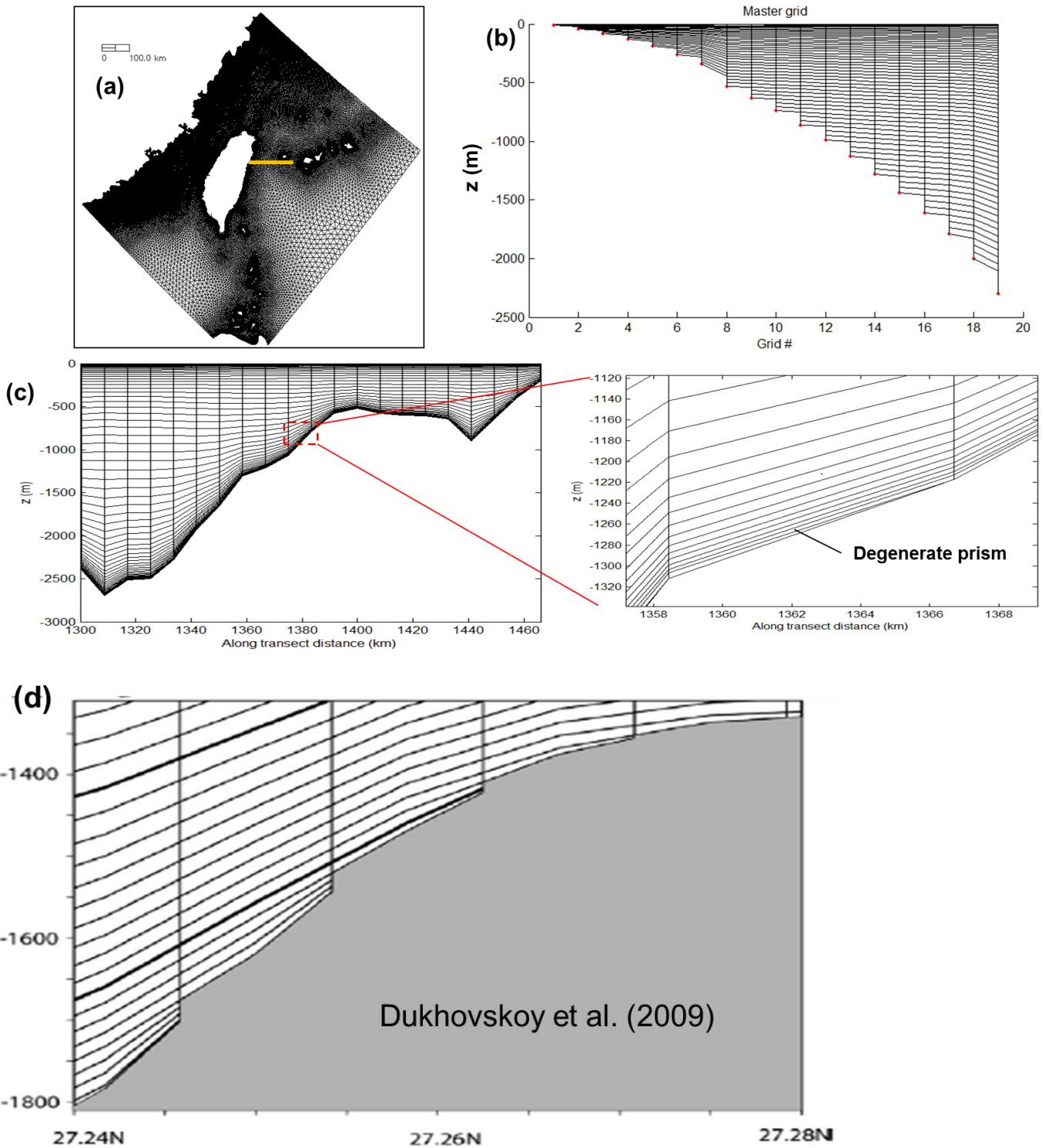


Fig. 3.5: LSC2 grid. (a) Horizontal grid showing the location of the transect (yellow line); (b) master grid at 19 reference depths; (c) corresponding vertical grid of the transect in (a), with degenerate prisms/shaved cells located near the bottom. (d) shows a typical VQS grid that contains staircases near the bottom.

Prisms

Regardless which type of *vgrid* is chosen, the basic 3D computational unit in SCHISM is a triangular or quad prism, with 3 or 4 vertical faces and uneven top and bottom faces (Fig. 3.6). SCHISM solves all equations in the original Z space for consistency; even with spherical coordinates (lon/lat), the equations are not transformed but instead multiple coordinate frames are used (see below). A staggering scheme *a la* Arakawa-CD grid is used to define variables. The surface elevations are defined at the nodes. The horizontal velocities are defined at the side centers and whole levels. The vertical velocities are defined at the element centers and whole levels, and the tracer concentration is defined at prism center, as they are solved with a finite-volume method. The linear shape functions are used for elevations and velocities (the latter has a vertical linear shape function as well). For quad elements, the bi-linear shape function is used.

At the end of each time step, the vertical levels are updated to account for free-surface movement. Although method like Arbitrary Lagrangian Eulerian (ALE) can be used to march the vertical levels in time, this is not done at the moment. Therefore, the state variables are simply moved to the new vertical location without re-interpolation.

In the model sometimes vertical interpolation is required (e.g., calculation of baroclinic force). We use cubic spline interpolation in the vertical to minimize diffusion.

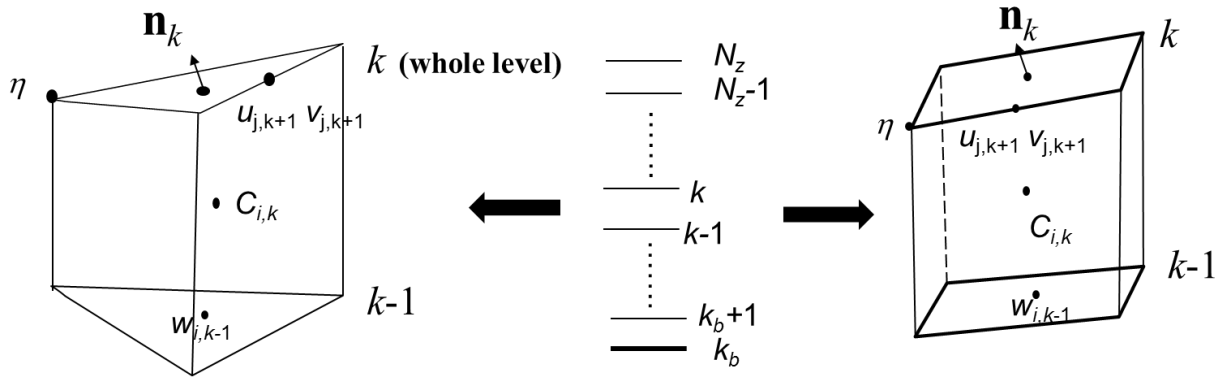


Fig. 3.6: Basic 3D computational unit in SCHISM where variables are staggered.

Polymorphism

The combination of LSC² vertical grid (Zhang et al. 2015) and horizontal mixed-element grids results in an extremely flexible grid system that has great practical applications. We demonstrate this with a toy problem for coastal ocean-estuary-river system depicted in Fig. 8. Since the tracer concentrations are defined at the prism centers, a row of quads and 1 vertical layer resembles a 1D model (Fig. A3c). Similarly, a row of quads with multiple vertical layers leads to 2DV configuration (Fig. 8c). Some parts of the shoals that are sufficiently shallow are discretized

using 1 vertical layer (Fig. A3b), which is a 2DH configuration. The deeper part of the domain is discretized using full 3D prisms, but with a larger number of layers in the deeper depths than in the shallow depths, in a typical LSC² fashion (Fig. A3a; Zhang et al. 2015). Different types of grids are seamlessly welded into a single SCHISM grid, resulting in greatest efficiency. With some care taken of the consistent bottom friction formulations across 1D, 2D and 3D, the model results show no discontinuity across different types of grids. The use of 1D or 2D cells in shallow areas also enhances numerical stability, as they are well suited and more stable for inundation process than 3D cells; e.g., the crowding of multiple 3D layers in the shallow depths is not conducive to stability.

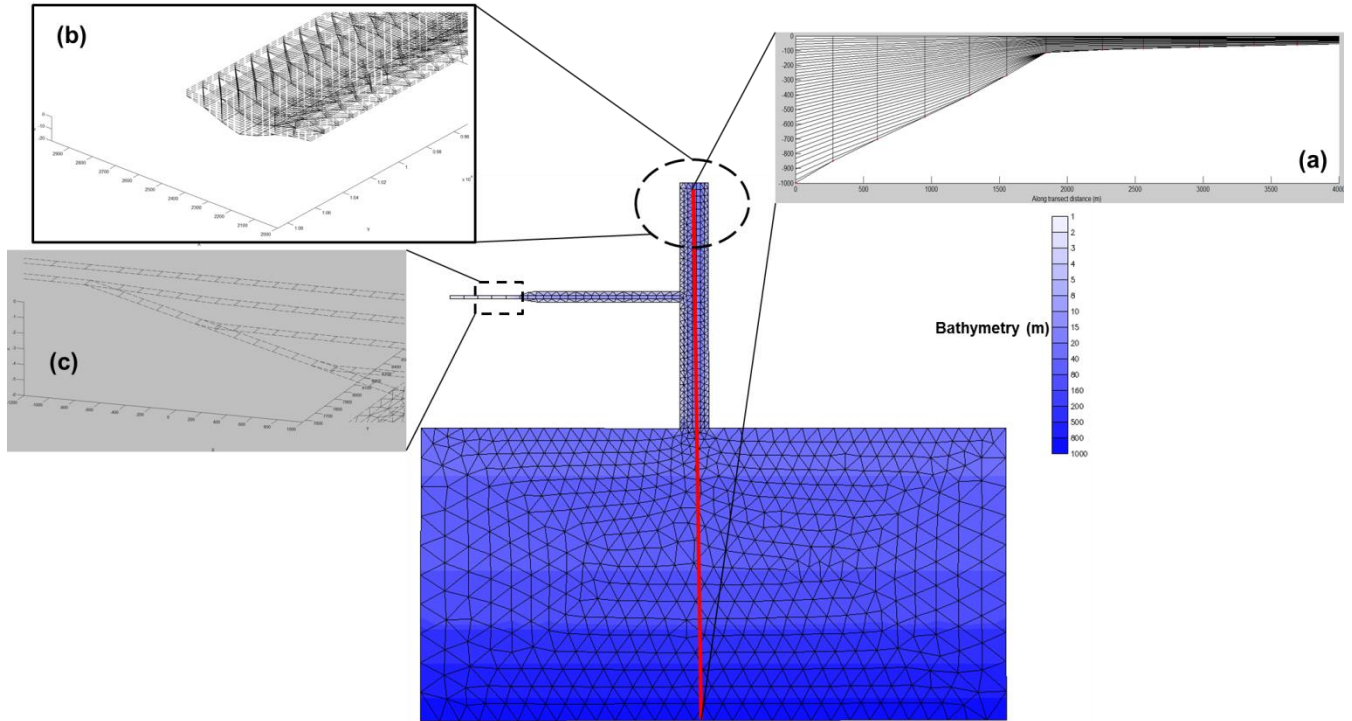


Fig. A3: Model polymorphism illustrated with a toy problem. The mixed triangular-quadrangular grid and the bathymetry are shown in the foreground. The vertical transect grid along the redline going from deep ocean into estuary (‘shipping channel’) is shown in insert (a). The 3D view of the grid near the head of estuary is shown in insert (b), with few layers on the shallow shoals. The grid near the upstream river is shown in insert (c), where transition from 2DV to 1D grid can be seen. In the test, a M2 tide is applied at the ocean boundary, and fresh water discharges are imposed at the heads of the river and estuary.

3.2 Barotropic solver

SCHISM solves the barotropic Eqs. (2.1-2.3) first, as the transport and turbulent closure equations lag one time step behind (in other words, the baroclinic pressure gradient term in the momentum equation is treated explicitly in SCHISM). The transport and turbulent closure equations will be discussed later. Due to the hydrostatic approximation, the vertical velocity w is

solved from the 3D continuity equation after the horizontal velocity is found. To solve the coupled Eqs. (2.1 & 2.3), we first discretize them and the vertical boundary conditions semi-implicitly in time as:

$$\frac{\eta^{n+1} - \eta^n}{\Delta t} + \theta \nabla \int_{-h}^{\eta} \mathbf{u}^{n+1} dz + (1-\theta) \nabla \int_{-h}^{\eta} \mathbf{u}^n dz = 0 \quad (3.3)$$

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}_*}{\Delta t} = \mathbf{f}^n - g\theta \nabla \eta^{n+1} - g(1-\theta) \nabla \eta^n + \frac{\partial}{\partial z} \left(\nu^n \frac{\partial \mathbf{u}^{n+1}}{\partial z} \right), \quad (3.4)$$

$$\begin{cases} \nu^n \frac{\partial \mathbf{u}^{n+1}}{\partial z} = \boldsymbol{\tau}_w^{n+1}, & \text{at } z = \eta^n; \\ \nu^n \frac{\partial \mathbf{u}^{n+1}}{\partial z} = \chi^n \mathbf{u}_b^{n+1}, & \text{at } z = -h, \end{cases} \quad (3.5)$$

where superscripts denote the time step, $0 \leq \theta \leq 1$ is the implicitness factor, $\mathbf{u}_*(x,y,z,t^n)$ is the back-tracked value calculated with Eulerian-Lagrangian Method (ELM; more on this later), and $\chi^n = C_D |\mathbf{u}_b^n|$. The elevations in the 2nd and 3rd terms of Eq. (3.3) are treated explicitly, which effectively amounts to a linearization procedure.

A Galerkin weighted residual statement in the weak form for Eq. (3.3) reads:

$$\int_{\Omega} \phi_i \frac{\eta^{n+1} - \eta^n}{\Delta t} d\Omega + \theta \left[- \int_{\Omega} \nabla \phi_i \cdot \mathbf{U}^{n+1} d\Omega + \int_{\Gamma_v} \phi_i \hat{U}_n^{n+1} d\Gamma_v + \int_{\bar{\Gamma}_v} \phi_i U_n^{n+1} d\bar{\Gamma}_v \right] + (1-\theta) \left[- \int_{\Omega} \nabla \phi_i \cdot \mathbf{U}^n d\Omega + \int_{\Gamma} \phi_i U_n^n d\Gamma \right] = 0, \quad (i = 1, \dots, N_p) \quad (3.6)$$

where N_p is the total number of nodes, $\Gamma \equiv \Gamma_v + \bar{\Gamma}_v$ is the boundary of the entire domain Ω , with Γ_v corresponding to the boundary segments where natural boundary conditions are specified, $\mathbf{U} = \int_{-h}^{\eta} \mathbf{u} dz$ is the depth-integrated velocity, U_n is its normal component along the boundary, and \hat{U}_n is the boundary condition. In SCHISM, linear shape functions are used (area coordinates for triangles and bi-linear functions for quads); thus, ϕ_i are familiar ‘‘hat’’ functions.

Integrating the momentum Eq. (3.4) along the vertical direction leads to:

$$\mathbf{U}^{n+1} = \mathbf{G}^n - g\theta H^n \Delta t \nabla \eta^{n+1} - \chi^n \Delta t \mathbf{u}_b^{n+1} \quad (3.7)$$

with

$$\mathbf{G}^n = \mathbf{U}_* + \Delta t \left[\mathbf{F}^n + \boldsymbol{\tau}_w^{n+1} - g(1-\theta)H^n \nabla \eta^n \right], \quad (3.8)$$

$$H^n = h + \eta^n, \quad \mathbf{F}^n = \int_{-h}^{\eta^n} \mathbf{f} dz, \quad \mathbf{U}_* = \int_{-h}^{\eta^n} \mathbf{u}_* dz$$

Note that Eq. (3.7) involves no vertical discretization as it is merely an analytical integration of Eq. (3.4).

To eliminate the unknown \mathbf{u}_b^{n+1} in Eq. (3.7), we invoke the discretized momentum equation, as applied to the top of the bottom cell:

$$\frac{\mathbf{u}_b^{n+1} - \mathbf{u}_{*b}}{\Delta t} = \mathbf{f}_b^n - g\theta \nabla \eta^{n+1} - g(1-\theta) \nabla \eta^n + \frac{\partial}{\partial z} \left(\nu^n \frac{\partial \mathbf{u}^{n+1}}{\partial z} \right), \quad \text{at } z = \delta_b - h. \quad (3.9)$$

However, since the viscosity term vanishes inside the bottom boundary layer (Eq. (2.10)), the bottom velocity can be formally solved as:

$$\mathbf{u}_b^{n+1} = \hat{\mathbf{f}}_b^n - g\theta \Delta t \nabla \eta^{n+1}, \quad (3.10)$$

where:

$$\hat{\mathbf{f}}_b^n = \mathbf{u}_{*b} + \mathbf{f}_b^n \Delta t - g\Delta t(1-\theta) \nabla \eta^n. \quad (3.11)$$

Note that although the vertical viscosity is not explicitly present in Eq. (3.10), it is indirectly involved through terms \mathbf{u}_{*b} and the Coriolis term in \mathbf{f}_b^n .

Substituting Eq. (3.9) into (3.7) results in:

$$\mathbf{U}^{n+1} = \hat{\mathbf{G}}^n - g\theta \hat{H}^n \Delta t \nabla \eta^{n+1}, \quad (3.12)$$

where:

$$\hat{\mathbf{G}}^n = \mathbf{G}^n - \chi^n \Delta t \hat{\mathbf{f}}_b^n, \quad \hat{H}^n = H^n - \chi^n \Delta t. \quad (3.13)$$

It is interesting to note from Eq. (3.13) that the bottom friction reduces the total depth by an amount that is proportional to the drag coefficient, the bottom velocity and the time step.

For simplicity the Coriolis terms are treated explicitly in SCHISM; experiments with implicit treatment or the 3rd-order Adam-Bashforth method indicate little improvement over the explicit treatment of this term.

Since SCHISM uses linear shape functions for the elevations, the two components of the horizontal velocity, u and v , are solved from the momentum equation independently from each other after the elevations are found. This approach has important implications as far as the Coriolis is concerned, and is different from that used in ELCIRC (Zhang et al., 2004). As a

matter of fact, special treatment must be made to find the tangential velocity components in UnTRIM-like models after the normal velocities are found to carefully maintain geostrophic balance, as discussed in Zhang et al. (2004) and Ham et al. (2005). On the other hand, the geostrophic balance is easily achieved with SCHISM.

Finally, substitution of Eq. (3.12) into (3.6) leads to an equation for elevations alone:

$$I_1 = I_4 - \theta\Delta t I_3 - (1 - \theta)\Delta t I_5 - \theta\Delta t I_6, \quad (i = 1, \dots, N_p) \quad (3.14)$$

where:

$$I_1 = \int_{\Omega} [\phi_i \eta^{n+1} + g\theta^2 \Delta t^2 \check{H} \nabla \phi_i \cdot \nabla \eta^{n+1}] d\Omega \quad (3.15)$$

$$I_4 = \int_{\Omega} [\phi_i \eta^n + (1 - \theta)\Delta t \nabla \phi_i \cdot \mathbf{U}^n + \theta\Delta t \nabla \phi_i \cdot \hat{\mathbf{G}}^n] d\Omega \quad (3.16)$$

$$I_3 = \int_{\Gamma_v} \phi_i \hat{U}_n^{n+1} d\Gamma_v \quad (3.17)$$

$$I_5 = \int_{\Gamma} \phi_i U_n^n d\Gamma \quad (3.18)$$

$$I_6 = \int_{\bar{\Gamma}_v} \phi_i U_n^{n+1} d\bar{\Gamma}_v \quad (3.19)$$

(the forms of \check{H} and $\hat{\mathbf{G}}^n$ will be given below for 3D and 2D cases)

Following standard finite-element procedures, and using appropriate essential and natural boundary conditions, SCHISM solves Eq. (3.14) to determine the elevations at all nodes. Note that the RHS terms I_{3-6} are known; in the case of I_6 , the integrals on $\bar{\Gamma}_v$ need not be evaluated since the essential boundary conditions are imposed by eliminating corresponding rows and columns of the matrix. Natural boundary conditions are used to evaluate the integral on Γ_v on the left-hand side of Eq. (3.17).

The matrix resulting from Eq. (3.14) is sparse and symmetric. It is also positive-definite if a mild restriction is placed on the friction-reduced depth in the form of $\hat{H}^n \geq 0$ ($\hat{H}^n=1$); numerical experiments (not shown) indicated that even this restriction can be relaxed for many practical applications that include shallow areas. Also note that the corresponding constant for 2D prisms is always positive (see below) and therefore the 2D mode is inherently more stable than 3D mode; consequently, judiciously leveraging the polyphororphism enhances stability near the wetting and drying interface. The matrix can be efficiently solved using a pre-conditioned Conjugate Gradient method (Casulli and Cattani 1994).

The above formulation applies to 3D prisms only because the bottom boundary layer (BBL) is utilized. In the case of a 2D prism, the final integral equation can be easily obtained and shares a similar form as (3.14), with different definitions for \tilde{H} and $\hat{\mathbf{G}}^n$:

For 3D prisms:

$$\tilde{H} = \hat{H}^n = H - \chi^n \Delta t \quad (3.20)$$

$$\hat{\mathbf{G}}^n = \mathbf{U}^* + \Delta t [\mathbf{F} + \boldsymbol{\tau}_w^{n+1} - \chi(\mathbf{u}_b^* + \Delta t \mathbf{f}_b) - g(1 - \theta) \hat{H}^n \nabla \eta^n] \quad (3.21)$$

For 2D prisms:

$$\tilde{H} = \frac{H^2}{\hat{H}}, \quad \tilde{H} = H + \chi \Delta t \quad (3.22)$$

$$\hat{\mathbf{G}}^n = \frac{H}{\tilde{H}} [\mathbf{U}^* + \Delta t [\mathbf{F} + \boldsymbol{\tau}_w^{n+1} - g(1 - \theta) H \nabla \eta^n]] \quad (3.23)$$

It's important to notice that the friction-modified depth is different between 2D and 3D cases (Eq. 3.20 vs 3.22), which has implications in shallow depths. We show more details of the evaluation of the integrals I_i below.

Integral I_3

This is a boundary integral that only need to be evaluated when the node i is located on an open boundary segment where Neuman-type B.C. is prescribed. Since the unknowns vary linearly along any side, we have (cf. Fig. 3.7):

$$I_3 = \sum_j \frac{L_{ij}}{2} \sum_{k=kbs}^{N_z-1} \Delta z_{j,k+1} \frac{\hat{u}_{j,k+1}^{n+1} + \hat{u}_{j,k}^{n+1}}{2} \quad (3.24)$$

where the outer sum is carried out along the 2 adjacent open side j , L_{ij} is the side length, $\Delta z_{j,k+1}$ is the layer thickness along side j , and kbs is the local bottom index.

If a Flather-type radiation condition (Flather 1987) needs to be applied, it can be done in the following fashion:

$$\hat{U}_n^{n+1} - \bar{U}_n = \sqrt{g/H} (\eta^{n+1} - \bar{\eta}), \quad (3.25)$$

where \bar{U}_n and $\bar{\eta}$ are specified mean incoming current and mean elevation. Then:

$$I_3 = \sum_j \frac{L_{ij}(\bar{U}_n)_{ij}}{2} + \frac{L_{ij} \sqrt{g H_{ij}}}{6} [2(\eta_i^{n+1} - \bar{\eta}_i) + (\eta_j^{n+1} - \bar{\eta}_j)] \quad (3.26)$$

In this case, the unknown η^{n+1} need to be moved to the LHS, and the diagonal (associated with η_i^{n+1}) is enhanced as a result and matrix symmetry is preserved.

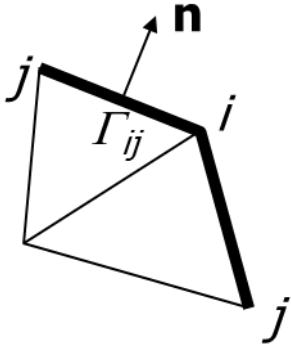


Fig. 3.7: Boundary node i with adjacent sides.

Integral I_5

Similar to I_3 , we have:

$$I_5 = \sum_j \frac{L_{ij}}{2} \sum_{k=kbs}^{N_z-1} \Delta z_{j,k+1} \frac{u_{j,k+1}^n + u_{j,k}^n}{2} \quad (3.37)$$

Integral I_1

This is the only implicit term. Referring to Fig. 3.8, we have:

$$I_1 = \sum_{j=1}^{Nb(i)} \sum_{l=1}^{i34(j)} \eta_{j,l}^{n+1} \int_{A_j} \hat{\phi}_{i'} \hat{\phi}_l dA_j + g\theta^2 \Delta t^2 \sum_{j=1}^{Nb(i)} \bar{H}_j \sum_{l=1}^{i34(j)} \eta_{j,l}^{n+1} \int_{A_j} \nabla \hat{\phi}_{i'} \cdot \nabla \hat{\phi}_l dA_j \quad (3.38)$$

Where j is a neighboring element of i , i' is the local index of node i inside element j , overbar in \bar{H} denotes element averaging, and $\hat{\phi}$ is the local linear shape function. We discuss the case of a triangle and quad element.

Case I: triangle

The 2 integrals can be evaluated analytically as:

$$\int_{A_j} \hat{\phi}_{i'} \hat{\phi}_l dA_j = \frac{1+\delta_{i',l}}{12} A_j \quad (3.38)$$

$$\int_{A_j} \nabla \hat{\phi}_{i'} \cdot \nabla \hat{\phi}_l dA_j = \frac{\vec{i}' \cdot \vec{l}}{4A_j} \quad (3.39)$$

where \vec{i}' and \vec{l} are two vectors along side i' and l respectively.

Case II: Quad

We can analytically evaluate 1 integral as:

$$\int_{A_j} \hat{\phi}_i \hat{\phi}_l dA_j = \frac{A_j}{16} \left(1 + \frac{1}{3} \xi_{i'} \xi_l\right) \left(1 + \frac{1}{3} \zeta_{i'} \zeta_l\right) + \frac{B_1}{96} \left(1 + \frac{1}{3} \zeta_{i'} \zeta_l\right) (\xi_{i'} + \xi_l) + \frac{B_2}{96} \left(1 + \frac{1}{3} \xi_{i'} \xi_l\right) (\zeta_{i'} + \zeta_l) \quad (3.40)$$

Where (ξ, ζ) are local coordinates, and B_1 and B_2 are 2 geometric constants (cf. Fig. 3.9):

$$B_1 = (\overline{12} \times \overline{43})_k = (x_2 - x_1)(y_3 - y_4) - (x_3 - x_4)(y_2 - y_1)$$

$$B_2 = (\overline{23} \times \overline{14})_k \quad (3.41)$$

The other integral cannot be evaluated analytically and so we use the 4-point Gauss quadrature.

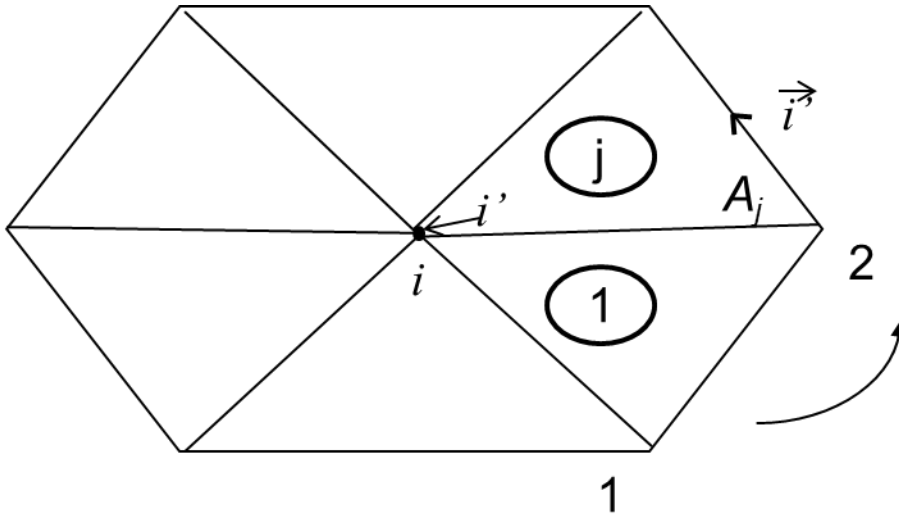


Fig. 3.8: Node ball used in calculating I_1 .

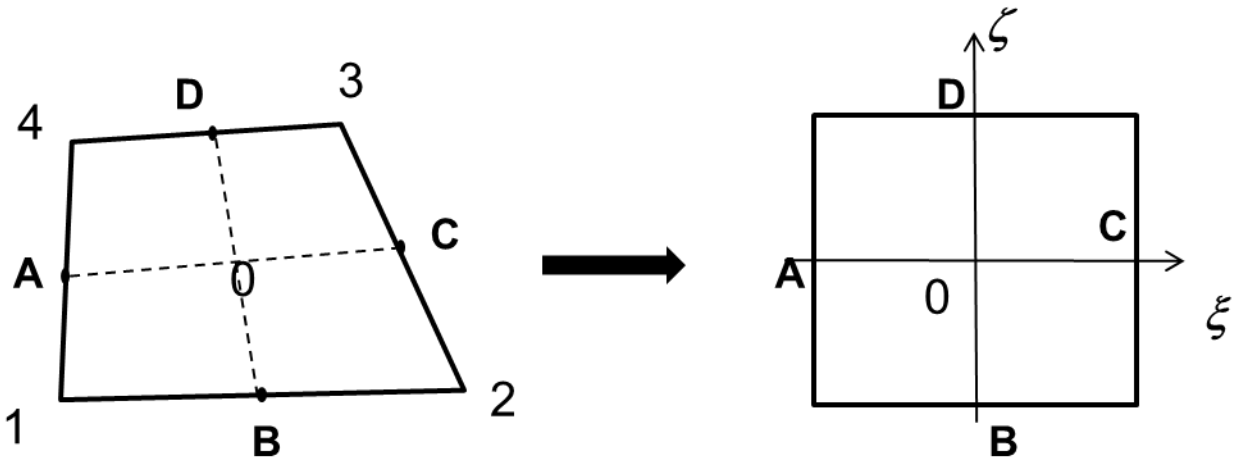


Fig. 3.9: Quad element.

Integral I_4

This integral contains most of the explicit terms. Most terms are straightforward to evaluate, e.g., using element averaging; the integrals involving the shape function or its derivative can be calculated using the similar method as in I_1 . Therefore we will only discuss the a few terms below given their importance in SCHISM.

Baroclinicity

Since a FVM is used to solve the tracers (including T,S) at the prism center, we evaluate the density gradient at prism center via a reconstruction method. Referring to Fig. 3.10, given a prism center '0', we first project the gradient onto vectors connecting adjacent prism centers:

$$\frac{\partial \rho}{\partial x}(x_1 - x_0) + \frac{\partial \rho}{\partial y}(y_1 - y_0) = \rho_1 - \rho_0 \quad (3.42)$$

$$\frac{\partial \rho}{\partial x}(x_2 - x_0) + \frac{\partial \rho}{\partial y}(y_2 - y_0) = \rho_2 - \rho_0 \quad (3.43)$$

$$\frac{\partial \rho}{\partial x}(x_3 - x_0) + \frac{\partial \rho}{\partial y}(y_3 - y_0) = \rho_3 - \rho_0 \quad (3.44)$$

after a cubic spline interpolation has been performed to calculate the density at prism '1' at the same vertical location as '0' (i.e. (i,k)). Note that if the element i is a quad, we will have 4 equations. We then solve pairs of equations to find for $\nabla \rho$, i.e., (3.42) with (3.43), (3.42) with (3.44), and (3.43) with (3.44). If the 3 centers happen to be co-linear, the equations have no solution and are discarded; however, at least 1 pair has valid solution. If a neighbor does not exist (boundary) or is dry, we replace the corresponding equation with the no-flux B.C.; e.g., if AB is such a side, then:

$$\frac{\partial \rho}{\partial y}(x_A - x_B) + \frac{\partial \rho}{\partial x}(y_B - y_A) = 0 \quad (3.45)$$

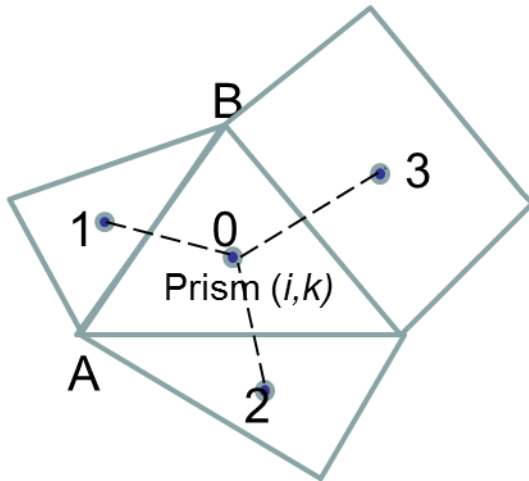


Fig. 3.10: Reconstruction method. i is the element index and k is the vertical index.

After the density gradients are found at prism centers, a simple linear interpolation in the vertical is used to calculate the gradients at side centers (and half levels). Then the trapezoidal rule is used to compute the baroclinic term: $-\frac{g}{\rho_0} \int_z^\eta \nabla \rho dz$. This will also be utilized in the solution of the momentum equation.

Horizontal viscosity

Momentum stabilization is an important consideration in designing advection and viscosity schemes. Zhang et al. (2016) demonstrated that the standard Laplacian viscosity is equivalent to the 5-point Shapiro filter (see below) on uniform grids; however, on non-uniform grids, it may behave like an ‘amplifier’ and therefore the filter form should be used instead (cf. Fig. 3.11ab):

$$\nabla \cdot (\mu \nabla u) \Big|_0 = \frac{\mu_0}{\sqrt{3}A_I} (u_1 + u_2 + u_3 + u_4 - 4u_0) \quad (3.46)$$

where all velocities have been interpolated onto a horizontal plane using linear interpolation in the vertical. The bi-harmonic viscosity is often superior to the Laplacian viscosity as it is more discriminating in removing sub-grid instabilities without adversely affecting the resolved scales of flow (Griffies and Hallberg 2000). The bi-harmonic viscosity can be implemented by applying the Laplacian operator twice. Referring to Fig. 3.11c, we have:

$$\begin{aligned} -\lambda \nabla^4 u \Big|_0 &= -\lambda \gamma_3 (\nabla^2 u_1 + \nabla^2 u_2 + \nabla^2 u_3 + \nabla^2 u_4 - 4\nabla^2 u_0) = \\ & \frac{\gamma_2}{\Delta t} [7(u_1 + u_2 + u_3 + u_4) - u_{1a} - u_{1b} - u_{2a} - u_{2b} - u_{3a} - u_{3b} - u_{4a} - u_{4b} - 20u_0] \end{aligned} \quad (3.47)$$

where λ is a hyper viscosity in m^4/s , $\gamma_3 = 1/(\sqrt{3}A_I)$ and $\gamma_2 = \lambda \gamma_3^2 \Delta t$ is a diffusion-number-like dimensionless constant. We found that in practice $\gamma_2 \leq 0.025$ is sufficient to suppress inertial spurious modes.

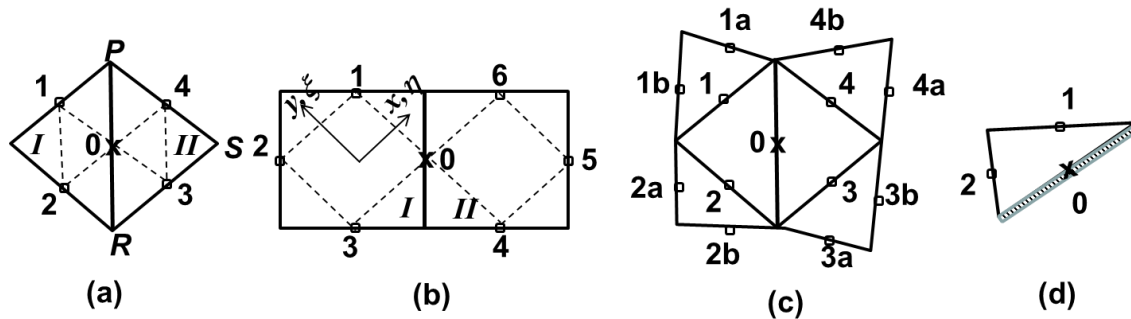


Fig. 3.11: Shapiro filters and viscosity stencil for (a) triangular and (b) quadrangular elements. ‘I’ and ‘II’ are 2 adjacent elements of side of interest (‘0’). The extended stencil used in

constructing bi-harmonic viscosity is shown in (c). The special case of a boundary side is shown in (d).

3.3 Eulerian-Lagrangian Method (ELM)

The default option for the momentum advection in SCHISM is ELM, which is one of few explicit methods that are unconditionally stable. Under this method, the momentum advection is approximated as:

$$\frac{D\mathbf{u}}{Dt} \cong \frac{\mathbf{u}(\mathbf{x}, t^{n+1}) - \mathbf{u}(\mathbf{x}^*, t^n)}{\Delta t} \quad (3.48)$$

where \mathbf{x} is a shorthand for (x, y, z) , and \mathbf{x}^* is the location of the foot of characteristic line (FOCL), calculated from the characteristic equation:

$$\frac{D\mathbf{x}}{Dt} = \mathbf{u} \quad (3.49)$$

The location \mathbf{x}^* is found via a backtracking step, standard in an ELM, via backward integration of Eq. (3.49) starting from a *given* location (\mathbf{x}), which is in our case a side center at whole level where the horizontal velocity \mathbf{u} is defined. The fixed starting location (Eulerian framework) followed by a Lagrangian tracking step gives the name Eulerian-Lagrangian method. Therefore the ELM consists of two major steps: a backtracking step (Fig. 3.12a) and an interpolation step at FOCL (Fig. 3.12b). We further sub-divide the tracking step into smaller intervals (based on local flow gradients), and use a 2nd-order Runge-Kutta method (mid-point method) within each interval, in order to accurately track the trajectory. Although exact integration methods have been proposed (Ham et al. 2006), their implementation is complicated for a 3D (triangular and quadrangular) prism and in the exceptional cases of wetting and drying interfaces. The interpolation step serves as an important control for numerical diffusion/dispersion in the ELM, and we therefore experimented with several options as shown below. However, before we get to this, we first explain how SCHISM converts the velocities at sides to the velocities at nodes, as the latter are required in the interpolation of the velocities along the characteristic line and at the FOCL (Fig. 3.12).

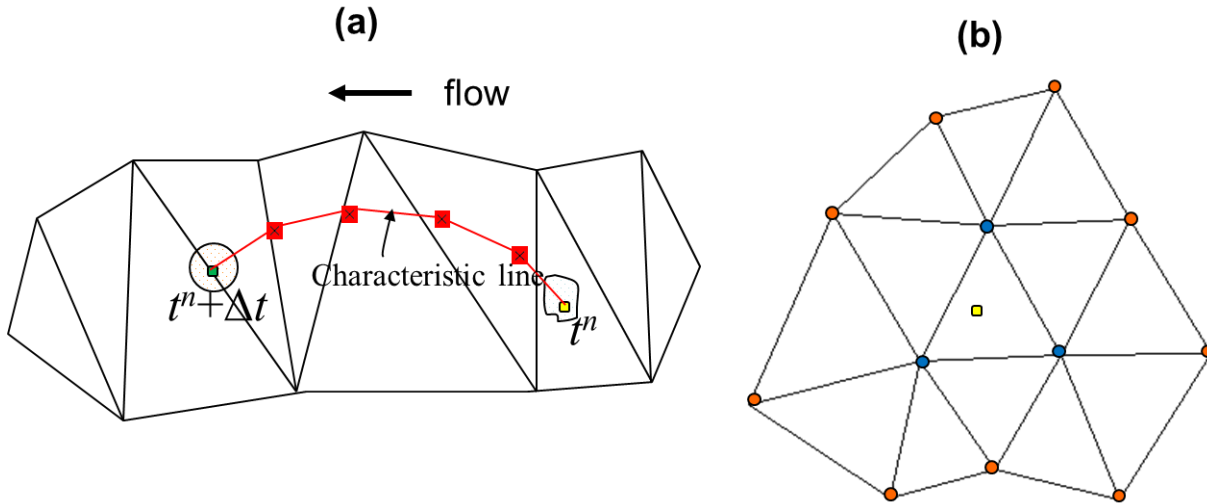


Fig. 3.12: Two steps in Eulerian–Lagrangian method. (a) The characteristic equation is integrated backward in space and time, starting from a side center (the green dot). The characteristic line is further subdivided into smaller intervals (bounded by the red dots), based on local flow gradients, and a 2nd-order Runge–Kutta method is used within each interval. The foot of characteristic line is marked as a yellow dot. Note that the vertical position of the trajectory is also changing and so the tracking is in 3D space. (b) Interpolation is carried out at FOCL (yellow dot), based on either the nodes of the containing elements (blue dots), or the 2-tier neighborhood (blue plus red dots; the latter are the neighbors of the blue dots) using a dual kriging method. Proper linear vertical interpolation has been carried out first to bring the values at each node onto a horizontal plane before the horizontal interpolation is done.

As explained by Danilov (2012), the conversion method used bears important ramifications: judicious averaging (e.g., from side to elements or to node etc.) may greatly reduce the need later on for filters to remove the inertial spurious modes while still keeping the inherent numerical dissipation low. In fact, one could have used the discontinuous velocity calculated within each element to carry out the backtracking, but this would introduce insufficient amount of dissipation to suppress the inertial modes.

In the first approach (‘MA’ hereafter; `indvel=1`), we use inverse distance weights to interpolate from velocities at surrounding sides onto a node (Fig. 3.13a). This introduces diffusion which may be excessive in our experience, and therefore no further stabilization (via filters or viscosity) is required for this approach (see the discussion of stabilization in Danilov 2012). This approach works well in shallow waters especially for the inundation process, as numerical stability often trumps the order of accuracy there. The 2nd approach (‘MB’ hereafter; `indvel=0`) is more elegant and utilizes the (linear) shape function in FEM within each element to calculate the node velocities. This is equivalent to using the P^{NC} non-conformal shape function (Le Roux et al. 2005) as one essentially interpolates based on information at sides (Fig. 3.13b). Because

each element produces a velocity vector at each of its nodes, the final node velocity is the simple average of the values calculated from all of the surrounding elements (Fig. 3.13b). This approach introduces much less dissipation, but does exhibit inertial spurious modes. As a result, further stabilization is required. To this end, SCHISM uses a 5-point Shapiro filter (Shapiro 1970) as illustrated in Fig. 3.11ab; the velocity at a side ‘0’ is filtered as:

$$\tilde{\mathbf{u}}_0 = \mathbf{u}_0 + \frac{\gamma}{4}(\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 + \mathbf{u}_4 - 4\mathbf{u}_0), \quad (3.50)$$

with the strength usually set as $\gamma=0.5$. It’s obvious that the filter is analogous to the Laplacian viscosity implementation in the previous section. It proves to be very effective in removing the sub-grid scale inertial spurious modes; however, it tends to introduce too much dissipation in the eddying regime, and therefore should be used only in the non-eddying regime, i.e. shallow waters. Since it’s equivalent to the Laplacian viscosity, we may remove this filter in the future options and ask users to use `ihorcon=1` instead.

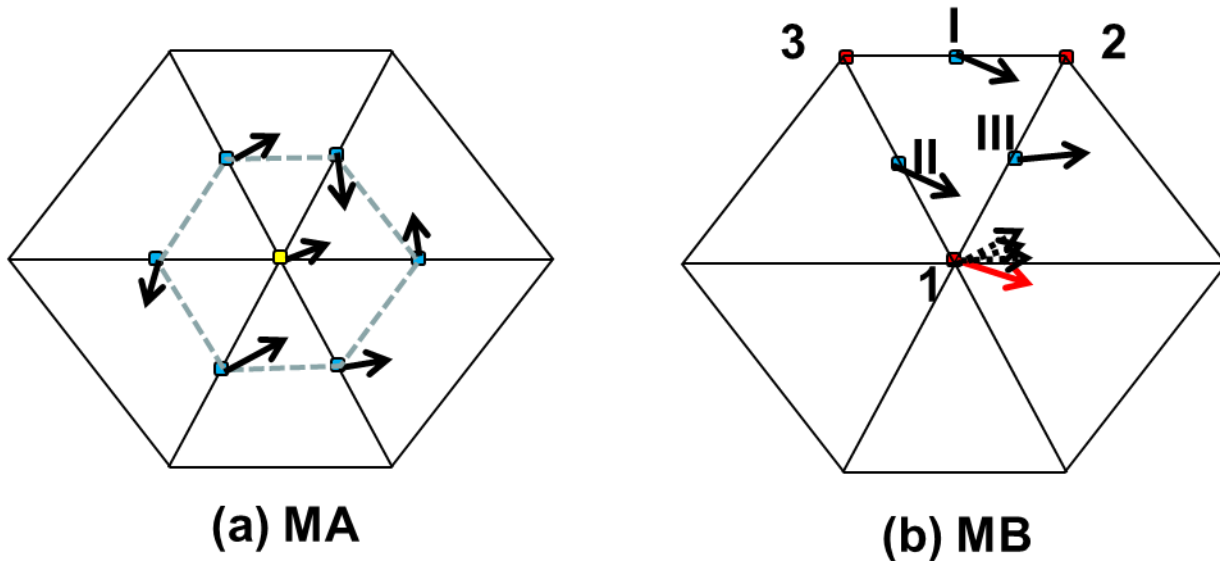


Fig. 3.13: Two methods of converting side velocities to a node velocity. (a) Inverse distance interpolation from sides (blue dots) to node (yellow dot); (b) use of FEM shape function to find the node velocity within each element first (the red arrow), i.e. $u_I = u_{II} + u_{III} - u_I$, followed by a simple averaging method to calculate the final value from all of its surrounding elements (dashed arrows).

Once the node velocities are found via MA or MB, the interpolation at the FOCL is carried out in 3D space. A simple linear interpolation is used in the vertical dimension as the results from the cubic-spline interpolation turned out to be similar, due to more confined spatial scales and smaller grid sizes in the vertical. The horizontal interpolation can be done using either a simple

linear shape function based on all of the nodes of the containing element ('LI' hereafter), or a higher-order dual kriging method ('KR' hereafter) suggested by Le Roux et al. (1997) (Fig. 3.12b). The latter requires larger stencil around the FOCL, and for best parallel efficiency we use a 2-tier neighborhood as shown in Fig. 3.12b. Given a total of N nodes available in the 2-tier neighborhood, the interpolation function is constructed as (Le Roux 1997):

$$f^h(x, y) = (\alpha_1 + \alpha_2 x + \alpha_3 y) + \sum_{i=1}^N \beta_i K(r_i) \quad (3.51)$$

where the first 3 RHS terms inside the parentheses represent a mean drift (modeled as a linear function), and the 2nd term is the fluctuation part, α_j , β_i are unknown coefficients, and r_i is the distance between (x, y) and (x_i, y_i) , with i being a node. The following forms of the generalized covariance function are commonly used (Le Roux et al. 1997):

$$K(r) = -r, r^2 \log(r), r^3, -r^5, r^7 \quad (3.52)$$

with increasing dispersion for the higher-degree functions; therefore in practice, the last two functions are seldom used. In the following we will refer to the first 3 functions as 'KR1', 'KR2' and 'KR3' respectively.

The equations to solve for the unknown coefficients are:

$$\left\{ \begin{array}{l} f^h(x_i, y_i) = d_i, \quad 1 \leq i \leq N \\ \sum_{i=1}^N \beta_i = 0 \\ \sum_{i=1}^N x_i \beta_i = 0 \\ \sum_{i=1}^N y_i \beta_i = 0 \end{array} \right. \quad (3.53)$$

where d_i are given data at each node. The 1st equation in (3.53) indicates that the dual kriging is an exact interpolator, and the other 3 equations are derived from minimization of the variance of estimation error (Le Roux et al. 1997). Note that the matrix of Eq. (3.53) is dependent only on geometry and therefore can be inverted and stored before the time stepping loop to achieve greater efficiency. After the coefficients are found, the interpolation at FOCL is done via Eq. (3.51).

The smaller stencil used here compared to that used by Le Roux et al. (1997) leads to larger numerical dispersion. Therefore an effective method must be found to control the dispersion, and we use the ELAD scheme of Shchepetkin and McWilliams (1998) for this purpose. The essence of ELAD is to iteratively diffuse the *excess field*, instead of the original signal, using a diffusion operator/smoothing. The viscosity scheme presented in the previous sub-section is used as the diffusion operator. The procedure is summarized as follows:

- 1) Find the local max/min at FOCL. Assuming that the prism at FOCL starting from a side j and level k is (kf, nf) , where nf is the element index and kf is the vertical index, the max/min are found in the prism (kf, nf) as:

$$\begin{aligned} u_{k,j}^{\max} &= \max_{l=1:i34, k=-1,0} u_{kf+k, im(l, nf)} \\ u_{k,j}^{\min} &= \min_{l=1:i34, k=-1,0} u_{kf+k, im(l, nf)} \end{aligned} \quad (3.54)$$

where $im()$ enumerates all nodes of an element.

- 2) The excess field associated with (k, j) is:

$$\varepsilon_{k,j}^{(1)} = \max\left[0, u_{k,j}^{n+1,1} - u_{k,j}^{\max}\right] + \min\left[0, u_{k,j}^{n+1,1} - u_{k,j}^{\min}\right] \quad (3.55)$$

where $u_{k,j}^{n+1,1}$ is the interpolated value at FOCL.

- 3) Apply a global diffusion operator to ε to obtain estimated velocity at the next iteration:

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \mu' \Delta t \nabla^2 \varepsilon_{k,j}^{(1)}, \quad \forall j, k \quad (3.56)$$

and we use the 5-point filter with maximum strength:

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \frac{1}{8} \left[\varepsilon_{k,1}^{(1)} + \varepsilon_{k,2}^{(1)} + \varepsilon_{k,3}^{(1)} + \varepsilon_{k,4}^{(1)} - 4\varepsilon_{k,j}^{(1)} \right] \quad (3.57)$$

where subscripts 1-4 are the 4 adjacent sides of j (Fig. 3.11ab);

- 4) Calculate the new excess field using $u_{k,j}^{n+1,2}$ in 2) and apply the filter 3) again to find the velocity at the next iteration $u_{k,j}^{n+1,3}$. Iterate until the excess field falls below a prescribed threshold. In practice, 10 iterations are usually sufficient to bring the excess field below an acceptable level (10^{-4} m/s); the remaining excess field is then further smoothed with the viscosity.

The filter in Eq. (3.57) is conservative in the sense that it only redistributes excess mass and does not introduce any additional mass. This is similar in spirit to the conservative scheme of Gravel and Staniforth (1994) but appears simpler in implementation. At a boundary side j , Eq. (3.57) is modified in order to maintain the conservation:

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \frac{1}{8} \left[\varepsilon_{k,1}^{(1)} + \varepsilon_{k,2}^{(1)} - 2\varepsilon_{k,j}^{(1)} \right] \quad (3.58)$$

where subscripts ‘1’ and ‘2’ are the 2 adjacent sides of j (Fig. 3.11d). Note that since the linear interpolation scheme (LI) does not introduce local extrema, ELAD is not applied there.

The various schemes presented above can be freely combined, resulting in schemes like ‘MA-LI’, ‘MB-KR2’ etc. Experiments indicate that overall the best scheme is MB-LI for both eddying and non-eddying regimes (Zhang et al. 2016).

3.4 Momentum equation

After the elevations are found, SCHISM solves the momentum Eq. (2.1) along each vertical column at side centers. A semi-implicit Galerkin finite-element method is used, with the barotropic pressure gradient and the vertical viscosity terms being treated implicitly, and other terms treated explicitly:

$$\int_{\delta_b-h}^{\eta} \psi_l \left[\mathbf{u}^{n+1} - \Delta t \frac{\partial}{\partial z} \left(\nu \frac{\partial \mathbf{u}^{n+1}}{\partial z} \right) \right] dz = \int_{\delta_b-h}^{\eta} \mathbf{g} \psi_l dz, \quad (l = kbs + 1, \dots, N_z) \quad (3.59)$$

where ψ is the hat function in the vertical dimension, δ_b is the bottom cell thickness, and

$$\mathbf{g} = \mathbf{u}^* + \Delta t [\mathbf{f} - g\theta \nabla \eta^{n+1} - g\theta \nabla \eta^n] \quad (3.60)$$

The two terms that are treated implicitly would have imposed the most severe stability constraints. The explicit treatment of the baroclinic pressure gradient and the horizontal viscosity terms, however, does impose mild stability constraints.

The final FEM equations are:

$$\begin{aligned} \frac{\Delta z_{l+1}}{6} (2\mathbf{u}_l^{n+1} + \mathbf{u}_{l+1}^{n+1}) + \frac{\Delta z_l}{6} (2\mathbf{u}_l^{n+1} + \mathbf{u}_{l-1}^{n+1}) + \nu_{l+1/2} \Delta t \frac{\mathbf{u}_{l+1}^{n+1} - \mathbf{u}_l^{n+1}}{\Delta z_{l+1}} + \nu_{l-1/2} \Delta t \frac{\mathbf{u}_l^{n+1} - \mathbf{u}_{l-1}^{n+1}}{\Delta z_l} = \\ \frac{\Delta z_{l+1}}{6} (2\mathbf{g}_l + \mathbf{g}_{l+1}) + \frac{\Delta z_l}{6} (2\mathbf{g}_l + \mathbf{g}_{l-1}), \quad (l = kbs + 2, \dots, N_z - 1) \end{aligned} \quad (3.61)$$

$$\frac{\Delta z_{l+1}}{6} (2\mathbf{u}_l^{n+1} + \mathbf{u}_{l+1}^{n+1}) + \nu_{l+1/2} \Delta t \frac{\mathbf{u}_{l+1}^{n+1} - \mathbf{u}_l^{n+1}}{\Delta z_{l+1}} + \chi \Delta t \mathbf{u}_{kbs+1}^{n+1} = \frac{\Delta z_{l+1}}{6} (2\mathbf{g}_l + \mathbf{g}_{l+1}), \quad (l = kbs + 1) \quad (3.62)$$

$$\frac{\Delta z_l}{6} (2\mathbf{u}_l^{n+1} + \mathbf{u}_{l-1}^{n+1}) + \nu_{l-1/2} \Delta t \frac{\mathbf{u}_l^{n+1} - \mathbf{u}_{l-1}^{n+1}}{\Delta z_l} = \boldsymbol{\tau}_w^{n+1} \Delta t + \frac{\Delta z_l}{6} (2\mathbf{g}_l + \mathbf{g}_{l-1}), \quad (l = N_z) \quad (3.63)$$

The bottom velocity is:

$$\mathbf{u}_{kbs}^{n+1} = 0, \text{ if } \chi \neq 0, \quad (3.64)$$

$$\mathbf{u}_{kbs}^{n+1} = \mathbf{u}_{kbs+1}^{n+1}, \text{ if } \chi = 0 \quad (3.65)$$

which is consistent with the bottom BL formulation we used.

After the velocities at all sides are found, the velocity at a node, which is needed in ELM, is evaluated using scheme MA or MB as discussed above.

3.5 Vertical velocity

The vertical velocity serves as a diagnostic variable for local volume conservation¹, but is a physically important quantity, especially when a steep slope is present (Zhang et al. 2004). To solve the vertical velocity, we apply a finite-volume method to a typical prism, as depicted in Fig. 3.6, assuming that w is constant within an element i , and obtain:

$$\hat{S}_{k+1}(\bar{u}_{k+1}^{n+1}n_{k+1}^x + \bar{v}_{k+1}^{n+1}n_{k+1}^y + w_{i,k+1}^{n+1}n_{k+1}^z) - \hat{S}_k(\bar{u}_k^{n+1}n_k^x + \bar{v}_k^{n+1}n_k^y + w_{i,k}^{n+1}n_k^z) + \sum_{m=1}^3 \hat{P}_{js(i,m)}(\hat{q}_{js(i,m),k}^{n+1} + \hat{q}_{js(i,m),k+1}^{n+1})/2 = 0, \quad (k = k^b, \dots, N_z - 1) \quad (3.66)$$

where \hat{S} and \hat{P} are the areas of the prism surfaces (Fig. 3.6), (n^x, n^y, n^z) , are the normal vector (pointing upward), \bar{u} and \bar{v} the averaged horizontal velocities at the top and bottom surfaces, and \hat{q} is the outward normal velocity at each side center. The vertical velocity is then solved from the bottom to the surface, in conjunction with the bottom boundary condition $(u, v, w) \cdot \mathbf{n} = 0$. In the case of earthquake module ($\text{imm} \neq 0$), the bed velocity is prescribed.

3.6 Turbulence closure

Umlauf and Burchard's GLS model is:

$$\frac{DK}{Dt} = \frac{\partial}{\partial z} \left(v_k \psi \frac{\partial K}{\partial z} \right) + \nu M^2 + \mu N^2 - \varepsilon \quad (3.67)$$

$$\frac{D\psi}{Dt} = \frac{\partial}{\partial z} \left(v_\psi \frac{\partial \psi}{\partial z} \right) + \frac{\psi}{K} (c_{\psi 1} \nu M^2 + c_{\psi 3} \mu N^2 - c_{\psi 2} F_w \varepsilon) \quad (3.68)$$

with natural B.C.:

$$\begin{cases} v_k \psi \frac{\partial K}{\partial z} = 0, & z = -h \text{ or } \eta \\ v_\psi \frac{\partial \psi}{\partial z} = \kappa_0 n v_\psi \frac{\psi}{l}, & z = -h \\ v_\psi \frac{\partial \psi}{\partial z} = -\kappa_0 n v_\psi \frac{\psi}{l}, & z = \eta \end{cases} \quad (3.69)$$

and essential B.C.:

$$\begin{cases} K = (c_\mu^0)^{-2} \nu \left| \frac{\partial u}{\partial z} \right|, \\ l = \kappa_0 \Delta, \\ \psi = (c_\mu^0)^p K^m (\kappa_0 \Delta)^n, \end{cases} \quad \text{at } z = -h \text{ or } \eta \quad (3.70)$$

where K is the TKE, l is the mixing length, $c_{\psi*}$ are constants, $\psi = (c_\mu^0)^p K^m l^n$ is a generic length-scale variable, and Δ is the distance to the boundary (surface or bottom). The turbulence production and dissipation terms are:

¹ Although other definitions of volume/mass exist, we define volume/mass in the finite-volume sense throughout this paper and measure conservation based on this definition.

$$M^2 = \left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2 \quad (3.71)$$

$$N^2 = \frac{g}{\rho_0} \frac{\partial \rho}{\partial z} \quad (3.72)$$

$$\varepsilon = (c_\mu^0)^3 K^{1.5} l^{-1} \quad (3.73)$$

In the code, the natural B.C. is applied first (see the FEM formulation below), and the essential B.C. is then used to overwrite the boundary values of the unknown, as suggested by GOTM. In the FEM formulation, K , ψ are defined at nodes and whole levels. Furthermore, the sums of M^2 and N^2 are treated explicitly/implicitly depending on the sign. The final equations look similar to those for the momentum equation:

$$\begin{aligned} & \mathcal{H}(N_z - m) \left[\frac{\Delta z_{m+1}}{6} (2K_m^{n+1} + K_{m+1}^{n+1} - 2K_m^n - K_{m+1}^n) - \Delta t (v_k^\psi)_{m+\frac{1}{2}}^n \frac{K_{m+1}^{n+1} - K_m^{n+1}}{\Delta z_{m+1}} \right] + \mathcal{H}(m - \\ & kbp) \left[\frac{\Delta z_m}{6} (2K_m^{n+1} + K_{m-1}^{n+1} - 2K_m^n - K_{m-1}^n) - \Delta t (v_k^\psi)_{m-\frac{1}{2}}^n \frac{K_m^{n+1} - K_{m-1}^{n+1}}{\Delta z_m} \right] = \\ & \mathcal{H}(N_z - m) \Delta t \left[\left\{ \frac{\Delta z_{m+1}}{2} (v_t M^2 + v_t^\theta N^2)_{m+1/2}^n \right\} - (c_\mu^0)^3 (K^{0.5} l^{-1})_{m+\frac{1}{2}}^n \frac{\Delta z_{m+1}}{6} (2K_m^{n+1} + \right. \\ & \left. K_{m+1}^{n+1}) \right] + \\ & \mathcal{H}(m - kbp) \Delta t \left[\left\{ \frac{\Delta z_m}{2} (v_t M^2 + v_t^\theta N^2)_{m-1/2}^n \right\} - (c_\mu^0)^3 (K^{0.5} l^{-1})_{m-\frac{1}{2}}^n \frac{\Delta z_m}{6} (2K_m^{n+1} + K_{m-1}^{n+1}) \right], \\ & (l = kbp, \dots, N_z) \end{aligned} \quad (3.74)$$

$$\begin{aligned} & \mathcal{H}(N_z - m) \left[\frac{\Delta z_{m+1}}{6} (2\psi_m^{n+1} + \psi_{m+1}^{n+1} - 2\psi_m^n - \psi_{m+1}^n) - \Delta t (v_\psi)_{m+\frac{1}{2}}^n \frac{\psi_{m+1}^{n+1} - \psi_m^{n+1}}{\Delta z_{m+1}} \right] + \mathcal{H}(m - \\ & kbp) \left[\frac{\Delta z_m}{6} (2\psi_m^{n+1} + \psi_{m-1}^{n+1} - 2\psi_m^n - \psi_{m-1}^n) - \Delta t (v_\psi)_{m-\frac{1}{2}}^n \frac{\psi_m^{n+1} - \psi_{m-1}^{n+1}}{\Delta z_m} \right] + \\ & \kappa_0 n \Delta t \left[\delta_{m, N_z} \left(\frac{v_\psi}{l} \right)_{N_z}^n \psi_{N_z}^{n+1} + \delta_{m, kbp} \left(\frac{v_\psi}{l} \right)_{kbp}^n \psi_{kbp}^{n+1} \right] = \\ & \mathcal{H}(N_z - m) \Delta t \left[\left\{ \frac{\Delta z_{m+1}}{2} (c_{\psi 1} v_t M^2 + c_{\psi 3} v_t^\theta N^2)_{m+1/2}^n (\psi/K)_{m+1/2}^n \right\} - \right. \\ & \left. c_{\psi 2} (c_\mu^0)^3 (K^{0.5} l^{-1} F_w)_{m+\frac{1}{2}}^n \frac{\Delta z_{m+1}}{6} (2\psi_m^{n+1} + \psi_{m+1}^{n+1}) \right] + \end{aligned}$$

$$\mathcal{H}(m - kbp)\Delta t \left[\left\{ \begin{array}{l} \frac{\Delta z_m}{2} (c_{\psi_1} v_t M^2 + c_{\psi_3} v_t^\theta N^2)_{m-1/2} (\psi/K)_{m-1/2}^n \\ \frac{\Delta z_m}{2} (v_t M^2 + v_t^\theta N^2)_{m-1/2} \frac{2\psi_m^{n+1} + \psi_{m-1}^{n+1}}{2K_{m-1/2}^n} \end{array} \right\} - \right. \\ \left. (c_\mu^0)^3 (K^{0.5} l^{-1} F_w)_{m-\frac{1}{2}}^n \frac{\Delta z_m}{6} (2\psi_m^{n+1} + \psi_{m-1}^{n+1}) \right], \quad (l = kbp, \dots, N_z) \quad (3.75)$$

where $\{ \}$ indicates the alternative explicit/implicit schemes mentioned above, and \mathcal{H} is a step function. We have applied the natural B.C. (3.69) in these equations, and after K and ψ are solved, the essential B.C. (3.70) is then used to overwrite the boundary values.

3.7 Transport equation

SCHISM supports a few FV solvers for the transport equation. All of the tracers, including T,S, sediment (if invoked) etc are solved simultaneously for efficiency.

The transport equation for a generic tracer C is given by:

$$\frac{\partial C}{\partial t} + \nabla \cdot (\mathbf{u}C) = \frac{\partial}{\partial z} \left(\kappa \frac{\partial C}{\partial z} \right) + F_h, \quad (3.76)$$

where F_h includes source/sink and also horizontal viscosity terms. The vertical B.C. is:

$$\kappa \frac{\partial C}{\partial z} = \hat{C}, \quad z = \eta \quad (3.77)$$

$$\kappa \frac{\partial C}{\partial z} = \hat{C}_b, \quad z = -h \quad (3.78)$$

Note that the 3D continuity equation ensures the constancy condition for the transport equation, i.e. $C=\text{const}$ initially will remain so in the absence of sinks/source.

3.7.1 Upwind

Since most of the variables below are defined at prism center, we will use shorthand like i etc to denote a prism at level $k+1/2$ when there is no confusion. Also we often omit the superscript ' n ' in the explicit terms for brevity.

A F.V. discretization of (3.76) for prism i is:

$$C_i^{m+1} = C_i^m - \frac{\Delta t'}{V_i} \sum_{j \in \mathcal{S}} Q_j T_{j*} + (F_h)_i^n \Delta t' + \frac{A_i \Delta t'}{V_i} \left[\kappa_{i,k} \frac{C_{i,k+1}^{m+1} - C_{i,k}^{m+1}}{\Delta z_{i,k+\frac{1}{2}}} - \kappa_{i,k-1} \frac{C_{i,k}^{m+1} - C_{i,k-1}^{m+1}}{\Delta z_{i,k-\frac{1}{2}}} \right], \quad (k = kbe + 1, \dots, N_z) \quad (3.79)$$

where $\Delta t' \neq \Delta t$ is the transport time step (subject to Courant condition below), V_i is the volume of the prism, C_i is a shorthand for $C_{i,k}$ (i.e., concentration at prism (i,k)). Note that we have treated the diffusion term implicitly. For the sake of brevity we'll drop the source and diffusion terms from now on and focus on the advection term. The upwind concentration is defined as:

$$C_{j*} = \begin{cases} C_i, & j \in S^+ \\ C_j, & j \in S^- \end{cases} \equiv C_{up} \quad (3.80)$$

where we have used shorthand for concentration at prism j (i.e. the prism adjacent to (i,k) from face j), and S^+ and S^- are outflow and inflow faces respectively.

Eq. (3.79) then becomes (only retaining the advection term):

$$C_i^{m+1} = C_i \left(1 - \frac{\Delta t'}{V_i} \sum_{j \in S^+} |Q_j| \right) + \frac{\Delta t'}{V_i} \sum_{j \in S^+} |Q_j| C_j \quad (3.81)$$

We have utilized the volume conservation:

$$\sum_{j \in S^+} |Q_j| = \sum_{j \in S^-} |Q_j| \quad (3.82)$$

Therefore the Courant condition is:

$$1 - \frac{\Delta t'}{V_i} \sum_{j \in S^+} |Q_j| \geq 0 \quad (3.83)$$

SCHISM uses this eq. to dynamically adjust the time step for transport for each step. Moreover, to improve efficiency, the vertical flux terms in (3.81) are treated implicitly, and the corresponding terms are then removed in the Courant condition (3.83). This is allowable because upwind is a linear method.

3.7.2 TVD (explicit)

The only difference between TVD and upwind schemes lies in the evaluation of the interfacial concentration:

$$C_{j*} = C_{up} + \frac{\varphi_j}{2} (C_{jD} - C_{up}) \quad (3.84)$$

where C_{up} is given in (3.80), C_{jD} is the downstream concentration, and $0 \leq \varphi_j \leq 2$ is a limiter function. TVD scheme nominally approaches 2nd order accuracy due to the anti-diffusion term.

After some algebraic manipulation, the final eq. for TVD is:

$$C_i^{m+1} = C_i + \frac{\Delta t'}{V_i} \sum_{j \in S^-} |Q_j| (C_j - C_i) + \frac{\Delta t'}{V_i} \sum_{j \in S^+} |Q_j| \frac{\varphi_j}{2} (C_i - C_j) + \text{source} + \text{diffusion} \quad (3.85)$$

And the Courant condition is:

$$\Delta t' \leq \frac{V_i}{\sum_{j \in S^-} |Q_j| (1 - \varphi_j/2 + \delta_i)} \quad (3.86)$$

where:

$$\delta_i = \sum_{p \in S^+} \frac{\varphi(r_p)}{2r_p} \quad (3.87)$$

and the upwind ratio, which involves upwind of upwind neighboring prism, is given by:

$$r_p = \frac{\sum_{m \in S^-} |Q_m| (C_m - C_i)}{|Q_p| (C_i - C_p)}, \quad p \in S^+ \quad (3.88)$$

In eqs. (3.85) and (3.86), the faces S , S^+ , and S^- need to exclude the locations where upwind is applied: all horizontal and vertical boundaries, and interfaces between wetting and drying. In those places, $\varphi_j = \delta_i = 0$. Again SCHISM automatically calculates the time step according to the Courant condition (3.86); the sub-time step used is the minimum of all prisms. The choices for the limiter function include: MINMOD, OSHER, van Leer, Super Bee etc.

Since TVD is a nonlinear method, we cannot treat the vertical fluxes implicitly, and so all fluxes have to be treated explicitly. TVD method is therefore more expensive than upwind. A hybrid upwind/TVD, with TVD being used in the deeper depths and upwind in the shallow depths, has been implemented in SCHISM to improve efficiency. The user can also manually specify upwind/TVD zones in the domain via `tvd.prop`. You are encouraged to use TVD² as much as possible for efficiency/accuracy.

3.7.3 TVD²

The TVD scheme shown above is explicit in 3D space and thus subject to the Courant condition, which comprises of horizontal and vertical fluxes across each of the prism faces (Casulli and Zanolli 2005). The restriction related to the vertical fluxes is especially severe due to smaller grid size used in the vertical dimension, and therefore a large number of sub-cycles within each time step are usually required. To partially mitigate the issue, a hybrid upwind-TVD approach can be used in which the more efficient upwind scheme, with an implicit treatment of the vertical fluxes, is used when the flow depth falls below a given threshold (with the assumption that stratification is usually much smaller in the shallows). However, this approach does not work in deeper depths of eddying regime, as large vertical velocities are not uncommon along steep bathymetric slopes. Together with the fact that a large number of vertical levels are usually required in the eddying regime, the explicit scheme leads to subpar computational performance and usually takes over 90% of the total CPU time.

We therefore develop an implicit TVD scheme in the vertical dimension in SCHISM. We start from the FVM formulation of the 3D transport equation at a prism i :

$$C_i^{n+1} = C_i^n - \frac{\Delta t}{V_i} \sum_{j \in S^-} |Q_j| (C_i - C_j) - \frac{\Delta t}{V_i} \sum_{j \in S} Q_j C_{jr} + \frac{A_i \Delta t}{V_i} \left[\left(\kappa \frac{\partial C}{\partial z} \right)_{i,k} - \left(\kappa \frac{\partial C}{\partial z} \right)_{i,k-1} \right] + \frac{\Delta t}{V_i} \int_{V_i} F_h dV \quad (3.89)$$

where C_j is the concentration at the neighboring prism of i across a prism face $j \in S = S^+ \cup S^-$, with S^+/S^- denoting outflow/inflow faces (which can be horizontal or vertical) respectively, V_i is the prism volume, A_i is the area of the associated surficial element, and Q_j is the flux at a face. In

Eq. (3.89) we have utilized the volume conservation in a prism (which is enforced by the solution of the vertical velocity): $\sum_{j \in S^-} |Q_j| = \sum_{j \in S^+} |Q_j|$. We have also approximated the concentration at a face as the sum of an upwind and a correction part as:

$$C|_j = C_{jup} + C_{jr}. \quad (3.90)$$

Note that in the 2nd term of RHS of Eq. (3.89), we have $C_j = C_{jup}$ as j is an inflow face. In addition, we have intentionally left out the time level in some terms in (3.89) as they will be treated explicitly or implicitly in the following.

We split the solution of Eq. (3.89) into 3 sub-steps:

$$C_i^{m+1} = C_i^m + \frac{\Delta t_m}{V_i} \sum_{j \in S_H} |Q_j| (C_j^m - C_i^m) - \frac{\Delta t_m}{V_i} \sum_{j \in S_H} Q_j \hat{\psi}_j^m, \quad (m = 1, \dots, M) \quad (3.91)$$

$$\tilde{C}_i = C_i^{M+1} + \frac{\Delta t}{V_i} \sum_{j \in S_V^-} |Q_j| (\tilde{C}_j - \tilde{C}_i) - \frac{\Delta t}{V_i} \sum_{j \in S_V^-} Q_j (\Phi_j + \Psi_j), \quad (j = kbe + 1, \dots, N_z) \quad (3.92)$$

$$C_i^{n+1} = \tilde{C}_i + \frac{A_i \Delta t}{V_i} \left[\left(\kappa \frac{\partial C}{\partial z} \right)_{i,k}^{n+1} - \left(\kappa \frac{\partial C}{\partial z} \right)_{i,k-1}^{n+1} \right] + \frac{\Delta t}{V_i} \int_{V_i} F_h^n dV, \quad (k = kbe + 1, \dots, N_z) \quad (3.93)$$

The 1st step solves the horizontal advection part (for all 3D prisms i), the 2nd step deals with the vertical advection part (where k_b is the bottom level index and N_z is the surface level index), and the last step tackles the remaining terms. We could have combined the 1st and 3rd steps into a single step at the expense of efficiency, because sub-cycling is used in the 1st step. In Eq. (3.91), sub-cycling in M sub-steps is required because of the horizontal Courant number condition, Δt_m is the sub-time step used, and $\hat{\psi}_j^m$ is a standard TVD limiter function. Eq. (3.91) is then solved with a standard TVD method. The last step (3.93) requires the solution of a simple tri-diagonal matrix. So we will only focus on the 2nd step.

Following Duraisamy and Baeder (2007, hereafter DB07), we use two limiter functions in Eq. (3.92): Φ_j is the space limiter and Ψ_j is the time limiter – thus the name TVD². The origin of these two limiters is the approximation Eq. (3.90) via a Taylor expansion in both space *and* time (DB07):

$$C_j^{n+1/2} = C_{jup}^{n+1} + \Phi_j + \Psi_j = C_{jup}^{n+1} + \mathbf{r} \bullet [\nabla C]_{jup}^{n+1} - \frac{\Delta t}{2} \left[\frac{\partial C}{\partial t} \right]_{jup}^{n+1} \quad (3.94)$$

Note that the interface value is taken at time level $n+1/2$ to gain 2nd-order accuracy in time. The vector \mathbf{r} points from prism center jup to face center j . Due to the operator splitting method, C^{n+1} now actually corresponds to \tilde{C} . Customary in a TVD method, we then replace the last 2 terms with limiter functions:

$$C_j^{n+1/2} = \tilde{C}_{jup} + \frac{\phi_j}{2} (\tilde{C}_{jD} - \tilde{C}_{jup}) - \frac{\psi_j}{2} (\tilde{C}_{jup} - C_{jup}^{M+1}) \quad (3.95)$$

and so:

$$\Phi_j = \frac{\phi_j}{2} (\tilde{C}_{jD} - \tilde{C}_{jup}), \Psi_j = -\frac{\psi_j}{2} (\tilde{C}_{jup} - C_{jup}^{M+1}) \quad (3.96)$$

where ‘ jD ’ stands for the downwind prism of i along the face j , and ϕ_j and ψ_j are 2 limiter functions in space and time respectively. Note that $\phi_j = \psi_j = 1$ leads to 2nd-order accuracy in both space and time.

Substituting Eq. (3.96) into (3.92) and after some algebra we obtain a nonlinear equation for the unknown concentration:

$$\tilde{C}_i + \frac{\frac{\Delta t}{V_i} \sum_{j \in S_V^-} |\mathcal{Q}_j| \left[1 + \frac{1}{2} \left(\sum_{p \in S_V^+} \frac{\phi_p}{r_p} - \phi_j \right) \right] (\tilde{C}_i - \tilde{C}_j)}{1 + \frac{\Delta t}{2V_i} \sum_{j \in S_V^+} |\mathcal{Q}_j| \left(\sum_{q \in S_V^-} \frac{\psi_q}{s_q} - \psi_j \right)} = C_i^{M+1} \quad (3.97)$$

where r_p and s_q are upwind and downwind ratios respectively:

$$r_p = \frac{\sum_{q \in S_V^-} |\mathcal{Q}_q| (\tilde{C}_q - \tilde{C}_i)}{|\mathcal{Q}_p| (\tilde{C}_i - \tilde{C}_p)}, p \in S_V^+ \quad (3.98)$$

$$s_q = \frac{(\tilde{C}_i - C_i^{M+1}) \sum_{p \in S_V^+} |\mathcal{Q}_p|}{|\mathcal{Q}_q| (\tilde{C}_q - C_q^{M+1})}, q \in S_V^-$$

DB07 showed that a sufficient TVD condition for Eq. (3.97) is that the coefficient of the 2nd LHS term be non-negative, i.e.:

$$1 + \frac{1}{2} \left(\sum_{p \in S_V^+} \frac{\phi_p}{r_p} - \phi_j \right) \geq 0 \quad (3.99)$$

$$1 + \frac{\Delta t}{2V_i} \sum_{j \in S_V^+} |\mathcal{Q}_j| \left(\sum_{q \in S_V^-} \frac{\psi_q}{s_q} - \psi_j \right) \geq \delta > 0 \quad (3.100)$$

where δ is a small positive number. Eq. (3.99) can be satisfied with any choice of standard limiter functions in space, and Eq. (3.100) must be solved together with Eq. (3.97) iteratively, because ψ and s_q are functions of \tilde{C} . We need to discuss 3 scenarios for prism i :

(1) vertically convergent flow: in this case, the outer sum in Eq. (3.100) is 0, so the inequality is always true;

- (2) divergent flow: the numerator of the 2nd LHS term in Eq. (3.97) is 0, and so $\tilde{C}_i = C_i^{M+1}$;
- (3) uni-directional flow (either upward or downward): in this case, prism i has exactly 1 inflow and 1 outflow face vertically, so a sufficient condition for Eq. (3.100) is:

$$1 - \frac{\Delta t}{2V_i} |Q_j| \psi_j \geq \delta > 0, j \in S_v^+ \quad (3.101)$$

Therefore we choose the following form for the limiter:

$$\psi_j = \max \left[0, \min \left[1, \frac{2(1-\delta)V_i}{|Q_j| \Delta t} \right] \right], j \in S_v^+ \quad (3.102)$$

where we have imposed a maximum of 1 in an attempt to obtain 2nd-order accuracy in time. Note that the limiter is a function of the vertical Courant number: it decreases as the Courant number increases. Eqs. (3.97) and (3.102) are then solved using a simple Picard iteration method starting from $\psi=0$ everywhere, and fast convergence within a few iterations is usually observed.

Simple benchmark tests indicate that TVD² is accurate for a wide range of Courant numbers as found in typical geophysical flows. It works equally well in eddying and non-eddying regimes, from very shallow to very deep depths, and is thus ideal for cross-scale applications. You are encouraged to use this option as much as possible.

3.7.4 Horizontal B.C. for transport

In either upwind or TVD schemes, the concentration at the neighboring prism T_j at the open boundary is known. For outflow, $T_j=T_i$ and the signal is advected out of the domain without hindrance. For incoming flow, T_j is specified by the B.C. (either in [bctides.in](#) or [*.th](#)), and SCHISM nudges to this value with a relaxation constant (specified in [bctides.in](#)), in order to prevent sharp gradient there. For a complete list of horizontal B.C. supported by SCHISM, see Table 4.1.

3.8 Updating the levels/inundation

After all variables are solved at the new step, the vertical grid is updated to reflect the newly solved elevations. An important aspect of this update is the treatment of inundation (wetting/drying). SCHISM supports 2 options for inundation. Note that the elevations are solved at *all* nodes including dry nodes at the previous time step; in other words, the elevations at the dry nodes – they are simply below the local bottom and can be thought as inactive ground water.

The 1st option ([inunfl=0](#)) is the default one that should be used in most applications. It uses a simple book-keeping algorithm to mark node/side/element as wet/dry based on a threshold depth ([h₀](#)). SCHISM does not support partial wetting and drying, and so the rule inside the code is: an

element is wet if all of its nodes and sides are wet, and is dry if any of its nodes or sides becomes dry. A node/side is wet iff (if and only if) at least 1 of its surrounding element is wet. For newly rewetted element/node/side, some state variables may be re-initialized based wet neighbors.

The 2nd option (`inunfl=1`) uses a shoreline tracking algorithm which requires finer resolution to be effective; otherwise the extrapolation procedure below may over-predict the inundation. In realistic cases, <5m resolution is usually sufficient for this purpose. Starting from the shoreline position at step n , each node on this line is checked for wetting and drying, and the local portion of the line is updated accordingly (Fig. 3.14a). The process is iterative and multi-layer wetting and drying is possible. After the new shoreline position is found at step $n+1$, a constant extrapolation of surface elevation is used from this position projected onto the nearest ‘dry’ node, in order to enhance the stability of the wetting and drying front (a technique advocated by the tsunami MOST model).

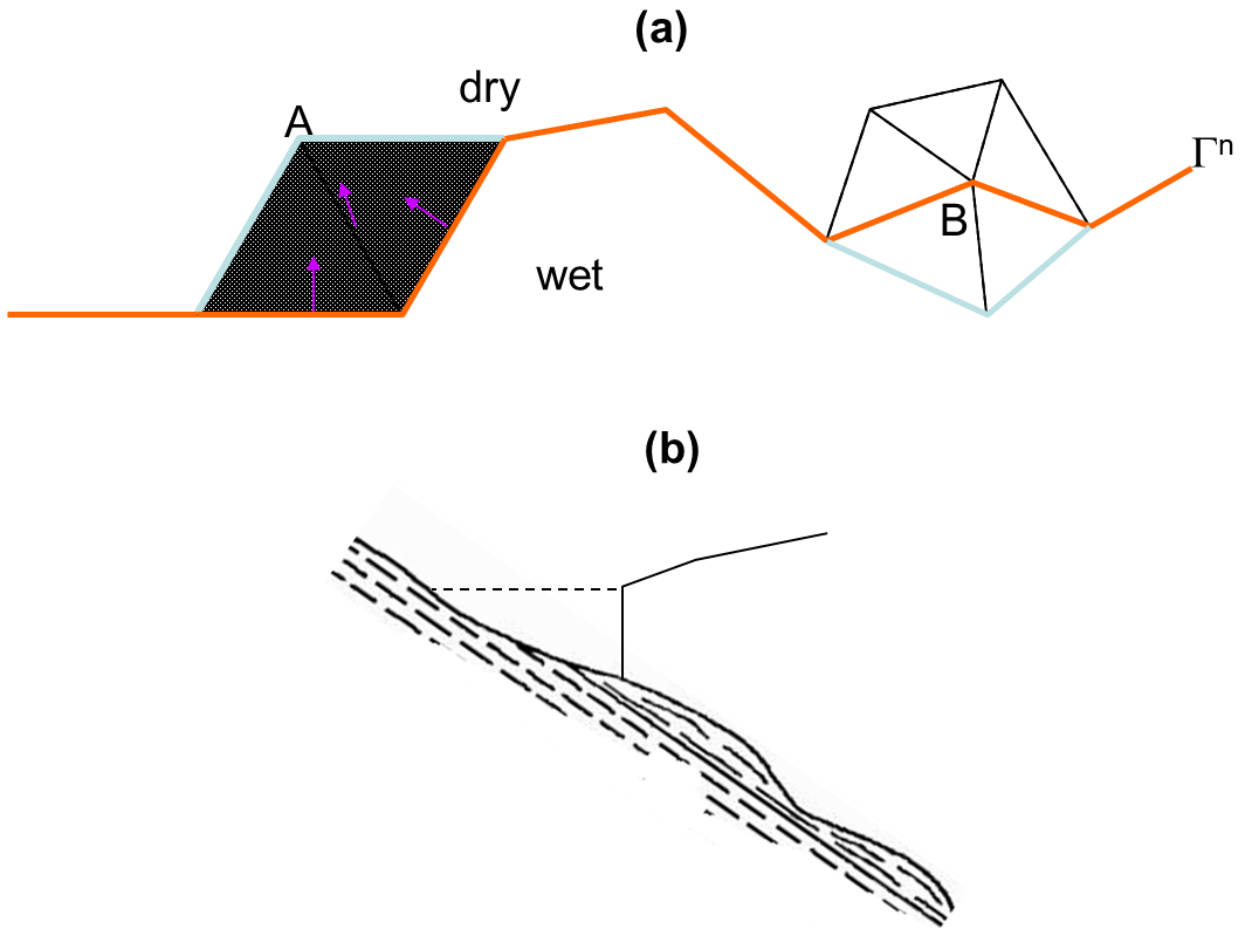


Fig. 3.14: Shoreline tracking algorithm. (a) The orange line is the shoreline from step n , and the light blue line is the updated portion since A is wetted and B becomes dry at step $n+1$. (b) At the

end of the tracking, constant extrapolation of elevation is used at the new shoreline position, which may flood the next dry node.

3.9 Spherical coordinates

We used the approach of Comblen et al. (2009) and transform the coordinates instead of the equations. Since the unstructured grids work naturally on a sphere, the polar singularity is avoided. Most of the work is done inside local frames. There are 2 frames used (cf. Fig. 3.15); note that all frames rotate with the earth.

- Global frame (x_g, y_g, z_g) (Fig. 3.15). The origin is located at center of the earth (assumed to be a sphere) of radius R_0 , x_g axis pointing to prime meridian, z_g to the north pole. The coordinates are related to the longitude and latitude of a point on the spherical surface:

$$\begin{cases} x_g = R_0 \cos \phi \cos \lambda \\ y_g = R_0 \cos \phi \sin \lambda \\ z_g = R_0 \sin \phi \end{cases} \quad (3.103)$$

- Local frame, located at a point on sphere (e.g., node/side center/element centroid): The 3 axes of this frame are: λ^0 (zonal), ϕ^0 (meridional North), and $\mathbf{r}^0 = \lambda^0 \times \phi^0$ (radial). The relationship between this frame and global frame is:

$$\lambda^0 = -\sin \lambda \mathbf{i} + \cos \lambda \mathbf{j} \quad (3.104)$$

$$\phi^0 = -\cos \lambda \sin \phi \mathbf{i} - \sin \lambda \sin \phi \mathbf{j} + \cos \phi \mathbf{k} \quad (3.105)$$

The frame $(\lambda^0, \phi^0, \mathbf{r}^0)$ is then the local (x,y,z) frame.

where \mathbf{ijk} are unit vectors of the global frame. Strictly speaking this frame is undefined at the 2 poles, but we can still use it there as long as the axes are unique.

Further assumption is made that when the origins are close to each other, the z -axes also coincide with each other, e.g. in the back-tracking part etc. This is reasonable as the relevant distances are much smaller than R_0 .

With the aid from all these frames, the equations can be solved in a very similar way as in Cartesian frame. The main difference is in the evaluation of the vectors in the local frame. The changes to the code are therefore minimal.

Below are some important info about arrays used in the code that are affected by lon/lat frames:

- $(x_{nd}, y_{nd}, z_{nd}), (x_{cj}, y_{cj}, z_{cj}), (x_{ctr}, y_{ctr}, z_{ctr})$ are *global* coordinates of node, side center and element centroid for **ics**=1 or 2. If **ics**=1 (Cartesian), $z_{nd}=z_{cj}=z_{ctr}=0$;
- (z_{nl}, z_s, z_e) are local z -coordinates measured vertically upward from the undisturbed surface (i.e. local frame when **ics**=2), at node, side, and element;

- $e_{frame}(1:3,1:3,1:nea)$ is the tensor for the element frame (w.r.t. global frame). The 2nd index indicates axis id (1=x; 2=y; 3=z) and the 1st index indicates tensor components. $sframe$ and $pframe$ are similar.

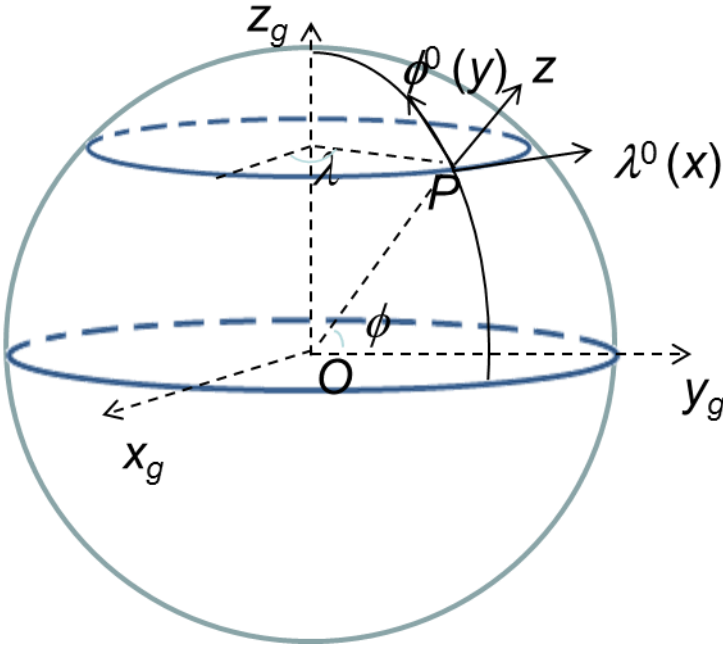


Fig. 3.15: Coordinate frames on a sphere.

Chapter 4 SCHISM I/O

Most SCHISM inputs/outputs can be visualized with ACE tools; see src/Utility/ACE for instructions on how to install these tools. Other useful tools will be mentioned in the later chapters.

4.1 Type of inputs for SCHISM

SCHISM input files can be broadly categorized into following 7 groups:

1. *.gr3, hgrid.ll: node centered spatial data and mesh connectivity. These file can be visualized using ACE/xmgredit5;
2. *.th: time history files in ASCII or binary format. The ASCII files can be visualized using ACE/xmgr5;
3. *.ic: initial condition files. Some of these files use .gr3 format, while others have simple ASCII format;
4. *.prop: element-centered spatial data and properties; can be visualized using ACE/xmgredit5;
5. *.in: role-specific input files with individual formats. ASCII inputs include vertical grid (vgrid.in), main parameter (param.in), B.C. input (bctides.in), and hydraulics.in (for hydraulics module) etc. Binary inputs include hotstart (hotstart.in), and nudging inputs (TEM_nu.in etc);
6. sflux/: atmospheric and heat flux files in netcdf format (CF convention v1.0). These files can be visualized using standard tools like ncview, ferret etc;
7. Inputs from modules: .nml, .inp etc.

4.2 Mandatory inputs

These inputs are required for all SCHISM simulations:

1. Horizontal grid (**hgrid.gr3**)
2. Vertical grid (**vgrid.in**)
3. Parameter input (**param.in**)
4. B.C. input (**bctides.in**)
5. Bottom friction input (**drag.gr3, rough.gr3 or manning.gr3**)

We'll explain these inputs in detail below. Comments/explanations are in **bold**.

4.2.1 hgrid.gr3

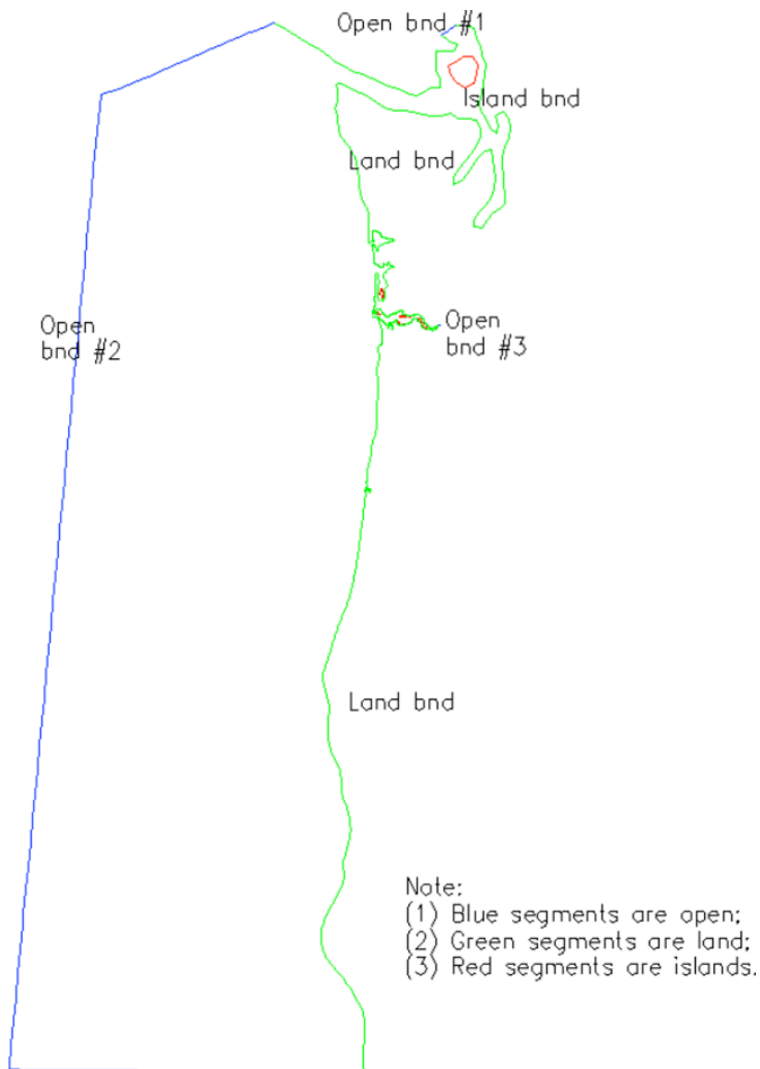


Fig. 4.1: Horizontal grid and boundary segments.

The format of this file is shown below (cf. Fig. 4.1):

`hgrid.gr3 ! alphanumeric description; ignored by code`

`60356 31082 ! # of elements and nodes in the horizontal grid`

Followed by list of node info:

`1 402672.000000 282928.000000 2.0000000e+01 ! node #, x,y, depth`

`2 402416.000000 283385.000000 2.0000000e+01`

`3 402289.443000 282708.750000 2.0000000e+01`

`4 402014.597000 283185.897000 2.0000000e+01`

.....
31082 331118.598253 112401.547031 2.3000000e-01 !last node

Following is the connectivity table:

1 4 1 2 3 101 ! element #, element type (triangle or quad), nodes 1-4

2 3 2 4 3

3 3 4 5 3

.....
60356 3 26914 30943 26804 !last element

Following is list of open and land boundary segments (needed for hgrid.gr3 only; not needed for other .gr3)

3 =Number of open boundaries

95 = Total number of open boundary nodes

3 = Number of nodes for open boundary 1

29835 ! first node on this segment

29834 ! 2nd node on this segment

.....
30001 !last node on this segment

90 =Number of nodes for open boundary 2

.....
16 =number of land boundaries (including islands)

1743 =Total number of land boundary nodes

753 0 = Number of nodes for land boundary 1 ('0' means the exterior boundary)

30381 ! first node on this segment

.....
1 !last node on this segment

741 0 ! Number of nodes for land boundary 2 ('0' means the exterior boundary)

.....
10 1 = Number of nodes for island boundary 1 ('1' means island)

29448 ! first node on this island

29451

.....

29449 **!last node on this island (note this is different from the first node '29448' above)**

.....

Note: (1) B.C. part can be generated with xmgredit5 → GridDEM → Create open/land boundaries; it can also be generated with SMS;

(2) if you have no open boundary, you can create two land boundary segments that are linked to each other. Likewise, if you have no land boundary, you should create two open boundary segments that are connected to each other;

(3) although not required, we recommend you follow the following convention when generating the boundary segments. For the exterior boundary (open+land), go in counter-clockwise direction. With xmgredit5, the island boundaries are automatically created once you have finished designating all open and land segments on the exterior boundary.

(4) note that this format is the same as fort.14 of ADCIRC.

4.2.2 vgrid.in

See Chapter 3.1 for details of different types of vgrid supported in SCHISM.

4.2.2.1 An example of SZ grid

2 **!ivcor (1: LSC²; 2: SZ)**

54 18 100. **!nvert(=N_z); kz (# of Z-levels); h_s (transition depth between S and Z)**

Z levels **!Z-levels in the lower portion**

1 -5000. **!level index, z-coordinates**

2 -2300.

3 -1800.

4 -1400.

5 -1000.

6 -770.

7 -570.

8 -470.

9 -390.

```

10 -340.
11 -290.
12 -240.
13 -190.
14 -140.
15 -120.
16 -110.
17 -105.
18 -100. !z-coordinate of the last Z-level must match -hs
S levels !S-levels below
30. 0.7 10. ! constants used in S-transformation: hc, θb, θf
18 -1. !first S-level (σ-coordinate must be -1)
19 -0.972222 !levels index, σ-coordinate
20 -0.944444
.....
54 0. !last σ-coordinate must be 0

```

Notes:

- Global outputs are from the bottom (*kbp*, variable in space) to surface (level *nvert*) at each node;
- The code will crash if the surface elevation falls below $-h_c$ so make sure h_c is sufficiently large (there is a hardwired lower bound for this around 5m in the code).

4.2.2.2 An example of pure S grid

If a "pure S" model is desired, use only 1 Z-level and set h_s to a very large number (e.g., 1.e6) above. For example, using *vgrid.in* below leads to a 2D model:

```

2 !ivcor
2 1 1.e6
Z levels
1 -1.e6
S levels

```

40. 1. 1.e-4

1 -1.

2 0.

4.2.2.3 An example of LSC² grid

This type of grid requires some user experience and can be generated using scripts (e.g., gen_vqs.f90).

1 !ivcor (1: LSC²; 2: SZ)

39 !nvrvt(=N_z)

1 29 -1. -0.893491 -0.793885 -0.694399 -0.595016 -0.495718 -0.396491 -0.297320 -0.198191
-0.099089 0. !node ID, bottom level index, sigma coordinate from bottom (-1) to surface
(0)





....

19520 12 -1.0 -0.985926 -0.918190 -0.854562 -0.794749 -0.738477 -0.685488 -0.635540 -
0.588405 -0.543870 -0.501733 -0.461804 -0.423903 -0.387860 -0.353515 -0.320714 -0.289311 -
0.259169 -0.230153 -0.202137 -0.174997 -0.148616 -0.122878 -0.097671 -0.072887 -0.048417 -
0.024156 0.0 !@last node

4.2.3 bctides.in

Please refer to sample bctides.in in the source code directory when you read this. Table 4.1 summarizes all horizontal B.C. and nudging options supported by SCHISM.

Table 4.1: Horizontal B.C. and nudging options in SCHISM. ‘X’ denotes ‘not enabled’.

Variable	bctides.in							param.in
	Type 1 (*.th)	Type 2:	Type 3	Type 4 (*[23]D.th)	Type 5	Type -1	Type -4, -5 (uv3D.th); nudging	
η	elev.th: Time history; uniform along bnd	constant	Tidal amp/phases	elev2D.th: time- and space- varying along bnd	elev2D.th: combination of 3&4	Must =0		inu_elev=1
<i>S&T, tracers</i>	[MOD].th: relax to time history (uniform along bnd) for inflow	Relax to constant for inflow	Relax to i.c. for inflow	[MOD]_3D. th: relax to time- and space- varying values along bnd during inflow				inu_[MOD]= 1 or 2
u, v	flux.th: via discharge (<0 normally!)	Via discharge (<0 normally!)	Tides (uniform amp/phase along bnd)	uv3D.th: time- and space- varying along bnd (in lon/lat for ics=2)	uv3D.th: combination of 3&4 (but tidal amp/phases vary along bnd)	Flather ('0' for η)	Relax to uv3D.th (2 separate relaxations for in & outflow)	inu_uv=1

04/01/2013 00:00:00 GMT !48-character start time info string. This is only used for visualization with xmis6

ntip tip_dp !# of constituents used in earth tidal potential; cut-off depth for applying tidal potential (i.e., it is not calculated when depth < tip_dp).

For $k=1, ntip$

talpha(k) !tidal constituent name

jspc(k), tamps(k), tfreq(k), tnf(k), tear(k) !tidal species # (0: declinational; 1: diurnal; 2: semi-diurnal), amplitude constants, angular frequency, nodal factor, earth equilibrium argument (in degrees);

end for

nbfr !total # of tidal boundary forcing frequencies

For $k=1, nbfr$

alpha(k) !tidal constituent name

amig(k), ff(k), face(k) !angular frequency (rad/s), nodal factor, earth equilibrium argument (in degrees) for constituent

end for

nope !# of open boundary segments

For $j=1, nope$

neta(j), iettype(j), ifltype(j), itetype(j), isatype(j), (optional) itrtype(j) **!# of nodes on the open boundary segment j (corresponding to hgrid.gr3), B.C. flags for elevation, velocity, temperature, and salinity, and (optionally) for each tracer module invoked (in the order of GEN, AGE, SED3D, EcoSim, ICM, CoSiNE, FIB, and TIMOR)**

Elevation b.c. section

if (iettype(j) == 1) !time history of elevation on this boundary

no input in bctides.in; time history of elevation is read in from elev.th (ASCII);

else if (iettype(j) == 2) !this boundary is forced by a constant elevation

ethconst !constant elevation value for this segment

else if (iettype(j) == 3) !this boundary is forced by tides

for k=1, nbfr

alpha(k) !tidal constituent name

for i=1, nond(j) !loop over all open boundary nodes on this segment

emo((j,i,k) efa(j,i,k) !amplitude and phase for each node on this open boundary

end for i

end for k

else if (iettype(j) == 4) !space- and time-varying input

no input in this file; time history of elevation is read in from elev2D.th (binary);

else if (iettype(j) == 5) !combination of '3' and '4'

time history of elevation is read in from elev2D.th, and then added to tidal B.C. specified below

for k=1, nbfr

alpha(k) !tidal constituent name

for i=1, nond(j) !loop over all open boundary nodes on this segment

emo((j,i,k) efa(j,i,k) !amplitude and phase for each node on this open boundary

end for i

end for k

else if (iettype(j) == 0)

elevations are not specified for this boundary (in this case the velocity must be specified).

endif

Velocity b.c. section

if (ifltype(j) == 0) !vel. not specified

no input needed

else if (ifltype(j) == 1) !time history of discharge on this boundary

no input in this file; time history of discharge is read in from flux.th (ASCII)

```

else if (ifltype(j) == 2) !this boundary is forced by a constant discharge
    vthconst !constant discharge (note that a negative number means inflow)
else if (ifltype(j) == 3) !vel. (not discharge!) is forced in frequency domain
    for k=1, nbfr
        vmo(j,k) vfa(j,k) !uniform amplitude and phase along each boundary segment
    end for;
else if (ifltype(j) == 4 or -4) !3D input
    time history of velocity (not discharge!) is read in from uv3D.th (binary)
    if ifltype(j) == -4
        rel1 rel2 !relaxation constants for inflow and outflow (between 0 and 1 with 1 being strongest nudging)
    endif
else if (ifltype(j) == 5) !combination of '4' and tides
    time history of velocity (not discharge!) is read in from uv3D.th (binary) and then added to tidal velocity specified below
    for k=1, nbfr
        alpha(k) !tidal constituent name
        for i=1, nond(j) !loop over all open boundary nodes on this segment
            umo(j,i,k) ufa(j,i,k) vmo(j,i,k) vfa(j,i,k) !amplitude and phase for (u,v) at each node on this open boundary
        end for i
    end for k
else if (ifltype(j) == -1) !Flanther type radiation b.c. (iettype must be 0 in this case)
    eta_mean !mean elevation below
    for i=1, nond(j) !loop over all nodes
        eta_m0(i) !mean elev at each node
    end for i
    vn_mean !mean normal velocity
    for i=1, nond(j)
        qthcon(1:Nz,i,j) !mean normal velocity at the node (at all levels)
    end for i
endif

Temperature b.c. section
if (itetype(j) == 0) !temperature not specified

```

no input needed

else if (itetype(j) == 1) !time history of temperature on this boundary

tobc !nudging factor (between 0 and 1 with 1 being strongest nudging) for inflow; time history of temperature will be read in from TEM_1.th (ASCII)

else if (itetype(j) == 2) !this boundary is forced by a constant temperature

tthconst !constant temperature on this segment

tobc !nudging factor (between 0 and 1) for inflow

else if (itetype(j) == 3) !initial temperature profile for inflow

tobc !nudging factor (between 0 and 1) for inflow

else if(itetype(j) == 4) !3D input

tobc !nudging factor (between 0 and 1); time history of temperature is read in from TEM_3D.th (binary)

endif

Salintiy B.C. part is similar to temperature:

if (isatype(j) == 0) !salinity not specified

.....

endif

If any tracer module is invoked, you also need the corresponding B.C. part for each tracer module, and the structure is similar to temperature.

end for !j: open boundary segment

Notes: the tidal amplitudes and phases can be generated using utility scripts shown on the web.

4.2.4 param.in

Please refer to sample param.in in the source code directory while you read the following.

The file uses free format, i.e. the order of input parameters is not important (but if you specify >1 instances of a parameter, the 1st instance will be used). Governing rules for this file are:

- Lines beginning with "!" are comments; blank lines are ignored;
- one line for each parameter in the format: keywords=value; keywords are case sensitive; spaces allowed between keywords and "=" and value; comments starting with "!" after value are ignored;

- value is an integer, double, or 2-char string (no single quotes needed for string input); for double, any of the format is acceptable: 40 or 40. or 4.e1 but the last 2 are preferred. Use of decimal point for integers is discouraged.

Most parameters (and their keywords) are explained as follows in alphabetical order; others are ‘developers handles’ that should not be tweaked usually. Also the sample has suggested values for many parameters. Note that you do not have to follow the order below (cf. param.in.sample). In some cases we have grouped some parameters for easier explanation, but you should specify them on separate lines. Also you'll find additional useful info in the comments of param.in.sample.

- **bfri**c (int)

Bottom friction option. If **bfri**c=-1, and Manning's n is specified in **manning.gr3**. If **bfri**c=0, spatially varying drag coefficients are read in from **drag.gr3** (as depth info). For **bfri**c=1, bottom roughnesses (in meters) are read in from **rough.gr3**.

If **bfri**c=-1, an additional parameter is required: **hmin_man** (in meters) which sets the minimum depth used in the Manning formulation.

If **bfri**c=1, two additional parameters are required: **dzb_min** (in meters) and **dzb_decay** (nondimensional). In this case the drag coefficients are calculated using the log drag law when the bottom cell thickness $\delta_b \geq \text{dzb_min}$. when $\delta_b < \text{dzb_min}$, $Cd = Cd_{max} * \exp[\text{dzb_decay} * (1 - \delta_b / \text{dzb_min})]$, where $Cd_{max} = Cd(\delta_b = \text{dzb_min})$. This is to avoid exaggeration of Cd in very shallow water.

- **consv_check** (int)

Parameter for checking volume and salt conservation. If turned on (=1), the conservation will be checked in regions specified by **fluxflag.prop**.

- **cpp_lon, cpp_lat** (double)

Centers of projection used to convert lat/lon to Cartesian coordinates. These are used if a variable Coriolis parameter is employed (**ncor**=1).

- **dt** (double)

Time step (in seconds)

- **dtb_min, dtb_max** (double)

Min/max sub-steps allowed in btrack; actual sub-steps are calculated based on local gradients.

- **eco_class, ntracer_gen, ntracer_age, sed_class** (int)

These parameters set the # of tracer ‘classes’ for each tracer module (EcoSim, GEN, AGE and SED3D), and are required if these modules are invoked in makefile. Note that other tracers modules (ICM, CoSiNE) set their own # of classes.

- global output options (int)

Global output (in machine-dependent binary) options. The frequency of global outputs is controlled by 2 parameters: `nspool` and `ihfskip`. Output is done every `nspool` steps, and a new output stack is opened every `ihfskip` steps. Therefore the outputs are named as `[1,2,3,...]_[MPI process id]_salt.63` etc. The combine scripts are then used to gather each output variable across all MPI processes into a single output, e.g., `[1,2,3...]_salt.63`.

Each output variable is controlled by 1 flag in `param.in`. We only show a few examples below; the rest are similar.

`elev.61 = 1` !global elevation output control. If `elev.61 =0`, no global elevation is recorded. If `elev.61= 1`, global elevation for each node of a sub-domain is recorded in `[1,2,3...]_[process id]_elev.61` in binary format. The output is either starting from scratch or appended to existing ones depending on `ihot`.

Some outputs are conditional upon you turn on certain module; e.g. `SED_1.63 = 1` won't output the concentration of sediment class 1 if you did not turn on `USE_SED` in makefile.

The outputs above are all node-centered ('standard'). In addition, the user can output any internal arrays using `schism_output_custom`. Some 'non-standard' outputs (e.g., element- or side-centered) are:

`hvel.67 = 1` !horizontal velocity defined at side [m/s]. These are the original velocity inside SCHISM

`vert.69 =1` !vertical vel. at centroids [m/s]. These are original vertical velocity inside SCHISM

- `h0` (double)

Minimum depth (in m) for wetting and drying (recommended value: 1cm). When the total depth is less than `h0`, the corresponding nodes/sides/elements are considered dry. It should always be positive.

- `hotout, hotout_write` (int)

Hot start output control parameters. If `hotout=0`, no hot start output is generated. If `hotout=1`, hot start output is named `outputs/it_[process id]_hotstart` every `hotout_write` steps, where `it` is the corresponding time iteration number. `hotout_write` must be a multiple of `ihfskip`. If you want to

hot start a run from step `it`, you need to combine all process-specific hotstart outputs into a `hotstart.in` using `combine_hotstart6.f90` (`./combine_hotstart6 -h` for help).

- `hw_depth`, `hw_ratio` (double)

These 2 parameters should only be adjusted for extreme under resolution near shelf break. Usually set: `hw_depth = 1.e6`, `hw_ratio = 0.5` to turn off the check. Also they only matter if `ibcc=0`.

The under resolution occurs often when you do not have interest or resource to resolve every steep slope near shelf break, and in this case, the depth changes from a few meters to a few km's between 2 adjacent elements. In this case, it's best to zero out the baroclinic force below the shallower depth.

The under resolution is detected using the Hannah-Wright-like ratio (`hw_ratio`) & depth (`hw_depth`), i.e., if $\max(h_1, h_2) \geq \text{hw_depth}$ and $|h_1 - h_2| \geq \text{hw_ratio} * \max(h_1, h_2)$, where h_1, h_2 are the depths at 2 adjacent elements. In this case, the baroclinic force is zeroed out below the higher bottom $\min(h_1, h_2)$.

- `ibcc` (int), `itransport` (int), `nrampbc` (int), `drampbc` (double)

Barotropic/baroclinic flags. If `ibcc=0`, a baroclinic model is used and regardless of the value for `itransport`, the transport equation is solved. If `ibcc=1`, a barotropic model is used, and the transport equation may (when `itransport=1`) or may not (when `itransport=0`) be solved; in the former case, S and T are treated as passive tracers.

If `ibcc=0`, the ramp-up function is specified with `nrampbc`, `drampbc`: ramp option flag and ramp-up period (in days). If `nrampbc=0`, `drampbc` is not used. The ramp function is a hyperbolic tangent function:

$$f(t) = \tanh(2t/86400/drampbc)$$

- `ibcc_mean` (int)

Mean T,S profile option. If `ibcc_mean=1` (or `ihot=0` and `icst=2`), mean T/S profile is read in from `ts.ic`, and will be removed when calculating baroclinic force. No `ts.ic` is needed if `ibcc_mean=0`.

- `ic_elev` (int)

Elevation initial condition flag for cold start only (`ihot=0`). If `ic_elev=1`, `elev.ic` (in *.gr3 format) is needed to specify the I.C. Otherwise elevation is initialized to 0 everywhere (cold start only).

- `icou_elfe_wwm` (int)

Coupler flag with WWM; needed if `USE_WWM` pre-processor is enabled. `icou_elfe_wwm = 0`: no feedback from WWM to SCHISM (decoupled); `1`: coupled SCHISM-WWM.

If `icou_elfe_wwm=1`, additional parameters required are:

`nstep_wwm (int)`: call WWM every this many time steps;

`msc2, mdc2 (int)`: these must match those in `wwminput.nml`;

`hmin_radstress (double)`: min. total water depth used only in radiation stress calculation; the radiation stress is zero if local depth $<$ `hmin_radstress`.

- `ics (int)`

Coordinate frame flag. If `ics=1`, Cartesian coordinates are used; if `ics=2`, both `hgrid.ll` and `hgrid.gr3` use degrees latitude/longitude (and they should be identical to each other in this case).

- `ic_[MOD] (int)`

Options for specifying initial tracer field for cold start, where ‘MOD’ is the name of the tracer model (e.g. TEM, SAL, SED etc). If `ic_[MOD]=1`, a vertically homogeneous but horizontally varying initial tracer field is specified in inputs like `temp.ic`, `salt.ic`, `[MOD]_hvar_[class #].ic` etc. If `ic_[MOD]=2`, a horizontally homogeneous but vertically varying initial tracer field, prescribed in a series of z-levels, is specified in inputs like `ts.ic`, `[MOD]_vvar_[class #].ic`. For more general 3D initial tracer fields, use the hot start option.

- `ieos_type, ieos_pres (int)`

By default, use the nonlinear equation of state: `ieos_type==0`. If the potential temperature is used, the pressure effect has been accounted for: `ieos_pres=0`.

- `if_source (int)`

Point sources/sinks option (0: no; 1: on). If `if_source=1`, needs `source_sink.in`, `vsource.th`, `vsink.th`, and `msource.th`.

- `iharind (int)`

Harmonic analysis flag. If `iharind \neq 0`, an input `harm.in` is needed.

- `ihconsv, isconsv (int)`

Heat budget and salt conservation models flags. If `ihconsv=0`, the heat budget model is not used. If `ihconsv=1`, a heat budget model is invoked, and a number of netcdf files for radiation flux input are read in from `sflux/sflux_rad*.nc`. If `isconsv=1`, the evaporation and precipitation model is evoked but the user needs to turn on the pre-processing flag `PREC_EVAP` in `makefile` and recompile. In this case, `ihconsv` must be 1, and additional netcdf inputs for precipitation (`sflux/sflux_prc*.nc`) are required.

- `ihdif (int)`

Flag to use non-zero horizontal diffusivity. If `ihdif=0`, it is not used. If `ihdif \neq 0`, input `hdif.gr3` is needed.

- `ihot` (int)

Hot start flag. If `ihot=0`, cold start; if `ihot≠0`, hot start from `hotstart.in`. If `ihot=1`, the time and time step are reset to zero, and outputs start from $t=0$ accordingly (and you need to adjust other inputs like `.th` etc). If `ihot=2`, the run (and outputs) will continue from the time specified in `hotstart.in`, and in this case, you do not need to adjust other inputs.

- `ihydraulics` (int)

Hydraulic model option. If `ihydraulics≠0`, `hydraulics.in` is required.

- `imm, ibdef` (int)

Bed deformation option. Default: 0 (no bed deformation); 1: with bed deformation (needs `ibdef` (# of steps during which deformation occurs), `bdef.gr3`); 2: 3D bottom deformation (need to interact with code).

- `indvel` (int), `ihorcon` (int), `hvis_coef0` (double), `ishapiro` (int), `shapiro` (double)

These parameters control the numerical dissipation in momentum solver; see SCHISM paper (Zhang et al. 2016) for details.

`indvel` determines the method of converting side velocity to node velocity. If `indvel=0`, the node velocity is allowed to be discontinuous across elements and additional viscosity/filter is needed to filter out grid-scale noises (spurious 'modes'). If `indvel=1`, an inverse-distance interpolation procedure is used instead and the node velocity is continuous across elements; this method requires no additional viscosity/filter unless kriging ELM is used (`inter_mom` >0). In general, `indvel=0` leads to smaller numerical dissipation and better accuracy, but does require a velocity B.C.

In the presence of spurious modes or dispersion (oscillation), viscosity/filter can be applied. `ihorcon=0`: no horizontal viscosity; =1: Laplacian (implemented as a filter); =2: bi-harmonic. For `ihorcon=0`, `hvis_coef0` specifies the non-dimensional viscosity. The filter is specified by `ishapiro` =0,1: turn off/on Shapiro filter. If `ishapiro=1`, `shapiro` specifies the Shapiro filter strength. If `ishapiro=-1`, an input called `shapiro.gr3` is required which specifies the filter strength at each node.

- `inter_mom, kr_co` (int)

Interpolation method at foot of characteristic line during ELM. `inter_mom=0`: linear interpolation; =1: dual kriging method. If `inter_mom=-1`, the depth in `krvel.gr3` (0 or 1) will determine the order of interpolation (linear or kriging). If the kriging ELM is used, the general covariance function is specified in `kr_co`: 1: linear $f(h)=-h$; 2: $(h^2*\log(h))$; 3: (cubic h^3); 4: ($-h^5$).

In general, `indvel=0` should be used with `inter_mom=0` or 1 to avoid large dispersion (with additional viscosity/filter also). `indvel=1` can be used with any covariance function without viscosity/filter.

- `inu_elev, inu_uv (int)`

Sponge layer for elevation and velocity. In SCHISM, relaxation/nudging of a generic variable is implemented as:

$$\tilde{\varphi} = (1 - \gamma)\varphi + \gamma\varphi_{target} \quad (4.1)$$

which is a discrete analogue of the restoration equation:

$$\frac{\partial\varphi}{\partial t} = \frac{\gamma}{\Delta t}(\varphi_{target} - \varphi) \quad (4.2)$$

If `inu_elev=0`, no relaxation is applied to elevation. If `inu_elev=1`, relaxation constants are specified in `elev_nudge.gr3` (depth=0 means no relaxation, depth=1 means strongest nudging) and the elevations are relaxed toward 0. Similarly for `inu_uv` (with input `uv_nudge.gr3`)

- `inu_[MOD] (int), step_nu_tr (double)`

Nudging flag for tracer model [MOD] (e.g. "TEM" for temperature), and nudging step (in sec). When `inu_[MOD]=0`, no nudging is done.

When `inu_[MOD]=1`, relax back to initial conditions.

When `inu_[MOD]=2`, nudge to values specified in `[MOD]_nu.in`, which has a time step of `step_nu_tr`.

If `inu_[MOD]≠0`, the horizontal relaxation factors are specified in `[MOD]_nudge.gr3` (as depths info), The final relaxation constant is the product of the horizontal relaxation factor and `dt`.

- `inunfl (int)`

Choice of inundation algorithm. `inunfl=1` can be used if the horizontal resolution is fine enough, and this is critical for tsunami simulations. Otherwise use `inunfl=0`.

- `iout_sta, nspool_sta (int)`

Station output flag. If `iout_sta≠1`, an input `station.in` is needed. In addition, `nspool_sta` specifies the spool for station output.

- `ipre (int)`

Pre-processing flag is very useful for checking grid errors. `ipre=1`: code will output `centers.bp`, `sidecenters.bp`, (`centers` build point, `sidecenters` build point), and `mirror.out` and `stop`. Check errors in `fort.11`. IMPORTANT: `ipre=1` only works for single CPU! `ipre=0`: normal run.

- `itr_met (int), h_tvd (double)`

Transport option for all tracers. `itr_met=1` for the upwind option, `itr_met=2` for the 2nd-order mass-conservative (explicit) TVD scheme, and `itr_met=3` for TVD². If `itr_met>=2`, 1 extra parameter is needed: `h_tvd` which specifies the transition depth (in meters) between upwind and TVD schemes; i.e. more efficient upwind is used when the local depth < `h_tvd`. Also in this case,

additional toggle between upwind and TVD is specified in `tvd.prop`. The TVD limiter function is specified in `TVD_LIM` in `mk/include_modules` (for efficiency purpose). If `itr_met=3`, 2 tolerances are also required (use recommended values).

- `itur` (int)

Turbulence closure model selection.

If `itur=0`, constant diffusivities are used for momentum and transport, and the diffusivities are specified in `dfv0`, `dfh0`.

If `itur=-2`, vertically homogeneous but horizontally varying diffusivities are used, which are read in from `hvd.mom` and `hvd.tran`.

If `itur=-1`, horizontally homogeneous but vertically varying diffusivities are used, which are read in from `vvd.dat`.

If `itur=2`, the zero-equation Pacanowski and Philander closure is used. In this case, a few extra parameters are required: `h1_pp`, `vdmax_pp1`, `vmin_pp1`, `tdmin_pp1`, `h2_pp`, `vdmax_pp2`, `vmin_pp2`, `tdmin_pp2`. Eddy viscosity is computed as: $vdiff = vdiff_max / (1 + rich)^2 + vdiff_min$, and diffusivity $tdiff = vdiff_max / (1 + rich)^2 + tdiff_min$, where `rich` is a Richardson number. The limits (`vdiff_max`, `vdiff_min` and `tdiff_min`) vary linearly with depth between depths `h1_pp` and `h2_pp`.

If `itur=3`, then the two-equation closure schemes from the GLS model of Umlauf and Burchard (2003) are used. In this case, 2 additional parameters are required: `turb_met`, `turb_stab`, which specify the closure scheme and stability function used: `turb_met=MY` is Mellor & Yamada; `=KL` is GLS as k-kl; `KE` is GLS as k-ε; `=KW` is GLS as k-ω; `=UB` is Umlauf & Burchard's optimal; `turb_stab=GA` is Galperin's clipping (only for MY); `=KC` is Kantha & Clayson's stability function) Also the user needs to specify max/min diffusivity/viscosity in `diffmax.gr3` and `diffmin.gr3`, as well as a surface mixing length scale constant `xlsc0`.

If `itur=4`, GOTM turbulence model is invoked; the user needs to compile the GOTM libraries first (see README inside GOTM/ for instructions), and turn on pre-processing flag `USE_GOTM` in makefile and recompile. In this case, the minimum and maximum viscosity/diffusivity are still specified in `diffmin.gr3` and `diffmax.gr3`. In addition, GOTM also requires an input called `gotmturb.inp`. There are some ready-made samples for this input in the source code bundle. If you wish to tune some parameters inside, you may consult `gotm.net` for more details.

- `nadv` (int)

Advection on/off option. If `nadv=0`, advection is selectively turned off based on the input file `adv.gr3`. If `nadv=1` or `2`, advection is on for the whole domain, and backtracking is done using either Euler or 2nd-order Runge-Kutta scheme.

- `ncor` (int)

Coriolis option. If `ncor=0` or `-1`, a constant Coriolis parameter is specified. If `ncor=0`, `coriolis` specifies the Coriolis factor. If `ncor=-1`, `latitude` specifies the mean latitude used to calculate the Coriolis factor.

If `ncor=1`, a variable Coriolis parameter, based either on a beta-plane approximation (`ics=1`) or on the latitude-dependent Coriolis (`ics=2`), is used, with the lat/lon coordinates read in from `hgrid.ll`. For `ics=1`, the center of beta-plane approximation must be correctly specified in `cpp_lon` and `cpp_lat`.

- `nramp` (int), `dramp` (double)

Ramp option for the tides and B.C., and ramp-up period in days (not used if `nramp=0`).

- `nws` (int), `wtiminc` (double)

Wind forcing options and the interval (in seconds) with which the wind input is read in. If `nws=0`, no wind is applied (and `wtiminc` becomes unused). If `nws=1`, constant wind is applied to the whole domain at any given time, and the time history of wind is read in from `wind.th`. If `nws=2` or `3`, spatially and temporally variable wind is applied and the input consists of a number of netcdf files in the directory `sflux/`. The option `nws=3` is only for checking heat conservation and needs `sflux.th`. If `nws=4`, the required input `wind.th` specifies wind and pressure at each node and at each time step $n * wtiminc$.

If `nws>0`, the ramp-up option is specified with `nrampwind` and `drampwind`. These are ramp flag and period (in days) for wind. In addition, the user can scale the wind by using `iwindoff`. If `iwindoff=0`, wind is applied as is. If `iwindoff=1`, the scaling factors are the depths in `windfactor.gr3`. The wind stress formulation is selected with `iwind_form`. If `nws=1` or `4`, or `nws=2` && `ihconsv=0`, or `nws=2` && `iwind_form=-1`, the stress is calculated from Pond & Pichard formulation. If `nws=2`, `ihconsv=1` && `iwind_form=0`, the stress is calculated from heat exchange routine. If WWM is enabled and `icou_elfe_wwm > 0` and `iwind_form=-2`, stress is calculated by WWM; otherwise the formulations above are used.

- `rmaxvel` (double)

Maximum velocity. This is needed mainly for the air-water exchange as the model may blow up if the water velocity is above 20m/s.

- `rnday` (double)

Total simulation time in days

- `s1_mxbnt`, `s2_mxbnt` (double)

Dimensioning parameters used in inter-subdomain backtracking. Start from `s[12]_mxnbt=0.5` 3, and increase them (gradually) if you get a fatal error like “btrack: overflow”. Accuracy is not affected by the choice of these two parameters.

- `thetai` (double)

Implicitness parameter (between 0.5 and 1). Recommended value: 0.6.

4.3 Optional inputs

4.3.1 .gr3, hgrid.ll

The format is the same as hgrid.gr3, except that the B.C. part is not necessary. For hgrid.ll, the x,y of each node are replaced by lon/lat coordinates.

The only difference between .gr3 files is that the ‘depth’ means different variables. We show some examples below.

Bottom drag input ([drag.gr3](#) or [rough.gr3](#) or [manning.gr3](#))

The ‘depth’ means C_d , bottom roughness in meters, or Manning’s n . The 3 files correspond to [bfric=0,1,-1](#).

[diffmin.gr3](#) and [diffmax.gr3](#)

The ‘depth’ specifies the min/max of the viscosity or diffusivity. Needed if [itur=3](#) or 4. Suggested value: $1.e-6$ m²/s for diffmin and 1 m²/s (or larger) for diffmax.

[adv.gr3](#)

Required if [nadv=0](#). The ‘depth’ is either 0, 1 or 2 in this case, corresponding to where the momentum advection is turned off, using backward Euler, or using 2nd-order Runge-Kutta method at a node.

[krvel.gr3](#)

Required if [inter_mom=-1](#). The ‘depth’ is either 0, 1, corresponding to kriging ELM bring turned off/on. The generalized covariance function is specified by [kr_co](#).

[bdef.gr3](#)

Required if [imm=1](#). The ‘depth’ specifies the amount of bed deformation (in meters) during time step 1 to [ibdef](#), with positive values being uplift.

4.3.2 .th (ASCII)

This includes [elev.th](#), [flux.th](#), [TEM_1.th](#), [SAL_1.th](#) etc, which share same ASCII structure. Below is a sample [flux.th](#) (note that *negative values mean inflow!*):

0. -1613.005 -6186.0 **!time (in sec), discharge at the 1st boundary segment that has ifltype=1 (in bctides.in), discharge at the 2nd boundary segment with ifltype=1,...**

300. -1613.05005 -6186.60156

600. -1611.37854 -6208.62549

900. -1609.39612 -6232.22314

.....

Note that the time must start from 0, and the step can be anything $\geq dt$.

In the case of `wind.th` (with `nws=1` or `4`), the time step specified inside must also match `wtiminc` in `param.in`. If `nws=4`, the format of `wind.th` is different: each line specifies the wind `u,v`, and atmospheric pressure (in Pa) at all nodes.

In the case of `msource.th` (invoked with `if_source=1`), the values after time stamp are tracer values at each `source` element (specified in `source_sink.in`), and the order of tracers is: T,S, followed by each tracer module invoked.

4.3.3 .th (binary)

These include `elev2D.th`, `uv3D.th`, `TEM_3D.th`, `SAL_3D.th`, and `[MOD]_3D.th` (where MOD is the tracer module name like 'COS'), which share similar binary structure. The format of `uv3D.th` is:

```
do it=1,nt !all time steps
```

```
  read(12,rec=it)time,((uth(k,i),vth(k,i),k=1,nvrt),i=1,nodes)  !loop over all open-boundary
nodes that need this type of B.C. (cf. bctides.in)
```

```
end do!it
```

Note: the time must start from 0, and the time step can be anything $\geq dt$.

4.3.4 .prop

The element-centered inputs color each element with a 'property', which can be visualized with `xmgrid5` (Special \rightarrow Properties). The format follows `xmgrid5` element property format:

```
1 -1 !element number, element property
```

```
2 0
```

```
3 2
```

```
4 5
```

```
....
```

Currently there are 2 .prop files:

[tv.d.prop](#)

If TVD transport is used (`itr_met` ≥ 2), user can explicitly specify horizontal regions where upwind or TVD/TVD² is used, based on the element property values (0: upwind; 1: TVD/TVD²) in [tv.d.prop](#).

[fluxflag.prop](#)

The element property flags (integers from -1,0,1,...) specify the 'region number' for each element. The code will only compute the flow across a side if (1) the flags at its 2 adjacent elements differ by 1, and (2) neither flag is -1 (cf. Fig. 4.2). The output `flux.dat` is a simple ASCII file with the format:

```
0.001389  0.1004E-01  0.1934E-01  !time (days), flow [m3/s] from region '1' to '0', flow
from region '2' to '1',...
0.002778  0.8852E-02  0.2285E-01
...
```

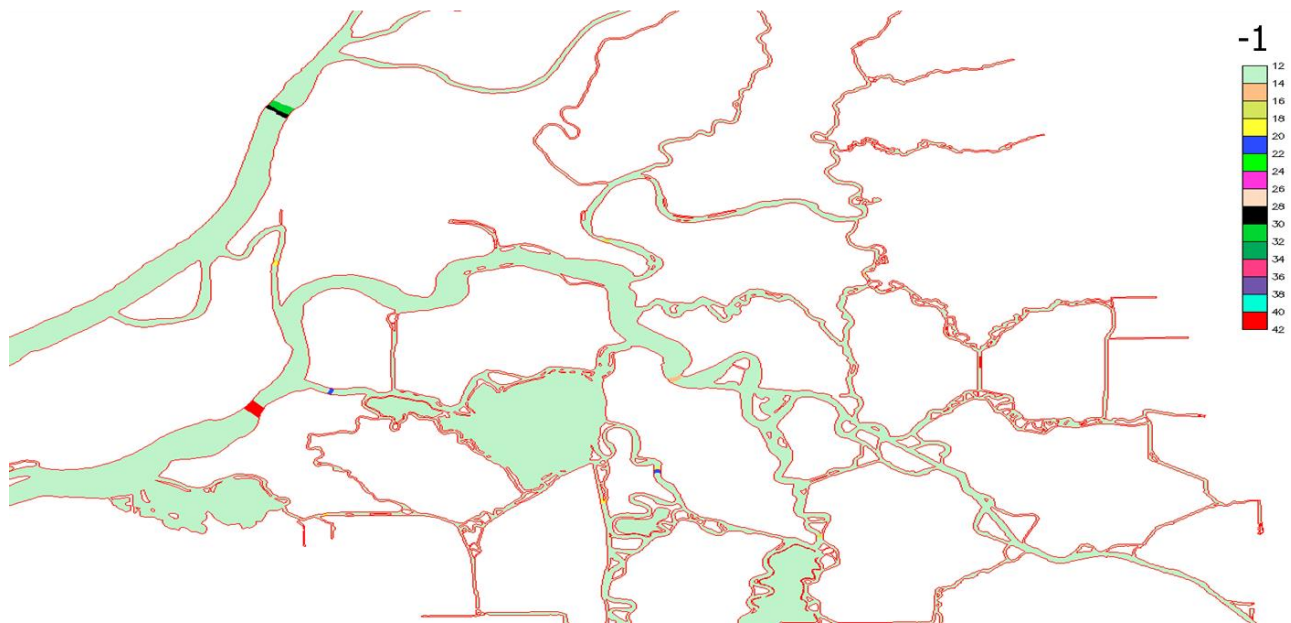


Fig. 4.2: Example of `fluxflag.prop`.

4.3.5 .ic

The I.C. inputs include the initial condition for elevation or tracers (`salt.ic`, `temp.ic`, and `ts.ic`). For most generic form of I.C. (variable in space and time), use `hotstart.in`.

`elev.ic` is a .gr3 file that specifies the initial elevation at each node.

Depending on the values of `icst`, T,S I.C. inputs have different format.

- If `icst` = 1, `salt.ic` and `temp.ic` take the .gr3 format;
- If `icst` = 2, `ts.ic` takes the following simple format:

43 !total # of vertical levels

1 -2000. 4. 34. !level #, z-coordinates, temperature, salinity

2 -1000. 5. 34.

....

Similar format is used for other tracers.

4.3.6 .in

4.3.6.1 Hotstart.in

See the scripts in Utility/Gen_Hotstart/ for the format of this input. It basically contains all major state variables defined at node/side/element.

4.3.6.2 station.in (.bp format)

This file is needed if `iout_sta=1` and is in a build point format (essentially.gr3 without the connectivity table):

```
1 1 1 1 1 1 1 1 1 !on (1)|off(0) flags for elev, air pressure, windx, windy, T, S, u, v, w
nsta    !# of stations
do i=1,np
    i,xsta(i),ysta(i),zsta(i) !zsta(i) is z-coordinates (from vertical datum; <0 is below)
enddo
```

Also see station.in.sample in the source bundle.

4.3.6.3 source_sink.in

This input is invoked if `if_source=1`, and specifies the element #'s for each (volume and mass) source and sink. The format is:

```
2 ! total # of elements with sources
100 ! element # of 1st source
101 ! element # of 2nd source
    !A blank line for readability; below are sinks
3 ! total # of elements with sinks
99 ! element # of 1st sink
100
105 ! element # of 3rd sink
```

4.3.6.4 hydraulics.in

This input is invoked if `ihydraulics=1`, and is the main input for the hydraulics module. See hydraulics module manual for details.

4.3.6.5 *harm.in*

This file is needed if `iharind=1`. Harmonic analysis capabilities were introduced in SCHISM by Andre Fortunato, using the routines of ADCIRC. These routines were developed by R.A. Luettich and J.J. Westerink, who are hereby acknowledged, and were used with written permission by R.A. Luettich. Note that only analysis on elevations at all nodes can be done at the moment.

The file has the following format (text adapted from the ADCIRC user's manual):

1. **NFREQ = number of frequencies included in harmonic analysis of model results.**

2. *for k=1 to NFREQ*

NAMEFR(k) = an alphanumeric descriptor (i.e. the constituent name) whose length must be <=16 characters

HAFREQ(k), HAFF(k), HAFACE(k) = frequency (rad/s), nodal factor, equilibrium argument (degrees)

end k loop

3. **THAS, THAF, NHAINC, FMV = the number of days after which data starts to be harmonically analyzed, the number of days after which data ceases to be harmonically analyzed, the number of time steps at which information is harmonically analyzed (information every NHAINC time steps after THAS is used in harmonic analysis), fraction of the harmonic analysis period (extending back from the end of the harmonic analysis period) to use for comparing the water elevation and velocity means and variances from the raw model time series with corresponding means and variances of a time series resynthesized from the harmonic constituents. This comparison is helpful for identifying numerical instabilities and for determining how complete the harmonic analysis was. Examples: FMV = 0. - do not compute any means and vars. FMV = 0.1 - compute means and vars. over final 10% of period used in harmonic analysis FMV = 1.0 - compute means and vars. over entire period used in harmonic analysis.**

4. **NHAGE, NHAGV = flags that indicate whether or not harmonic analysis is performed: NHAGE= 0 no harmonic analysis is performed for global elevations; NHAGE = 1 harmonic analysis is performed for global elevations (output on harme.53); NHAGV is for velocity which is not active right now.**

4.3.7 *sflux/*

This dir is required if `nws=2`. In this case, atmospheric forcings include wind, air pressure and temperature, precipitation, humidity and longwave and shortwave fluxes. These are specified in the netcdf files inside *sflux/* dir, and conform to the NetCDF Climate and Forecast (CF) Metadata Convention 1.0.

There are 4 types of files in sflux/dir; see this site for sample files.

1. **sflux_inputs.txt**

This ascii file is a namelist; the command below shows its content:

```
$cat sflux_inputs.txt
```

```
&sflux_inputs ! file name
```

```
start_year = 1998, ! start year
```

```
start_month = 09, ! start month
```

```
start_day = 01, ! start day
```

```
start_hour = 0.0, ! start hour
```

```
utc_start = 0.0, ! offset from UTC in hours, with positive numbers for western hemisphere
```

```
/
```

2. **sflux_air_1.0*.nc**: netcdf files that have time, wind speed at 10m above MSL (u,v), air temperature and pressure and specific humidity;
3. **sflux_prc_1.0*.nc**: netcdf files that have precipitation data;
4. **sflux_rad_1.0*.nc**: netcdf files that have downward long and short (solar) wave radiation fluxes.

Note that 3&4 are only required if the heat exchange module is invoked via **ihconsv=1**. We have NARR sflux files from 1979-present, but cannot upload all of them to the web due to disk space limitation. You can find some samples at http://ccrm.vims.edu/yinglong/wiki_files/NARR/.

Two sources of data are allowed for each type of .nc files, and the relative priority is fixed by the file name. For instance **sflux_air_1.003.nc** is might be blended with a file called **sflux_air_2.003.nc**. The ".003" component of the name represents the order of the file within the stack of provided input files. For instance, there might be a new file (001,002,003) produced every 12 hours in a forecast cycle.

Interpolation and prioritization

Using air as an example, it is assumed that the file **sflux_air_2.001.nc** is more resolved or accurate than **sflux_air_1.001.nc**. The two will be blended in the model in a way that favors the ‘_2’ file. This blending of the fields is only adjustable in the code as described in notes below. The default in sflux_9c.F90 is a 99:1 blend of the ‘_2’ file to the ‘_1’ file.

As was remarked above, the files are arranged temporally in a stack of files starting with ".001". Given the sequence of forecasting and analysis, it is common for atmospheric files to overlap. A file might begin with a brief period of data assimilation plus a few days of forecast. SCHISM assumes that a new file indicates the injection of information, so when it encounters overlap, it advances to the later file.

Using NARR files for your simulation (North America only)

First, make sure the NARR grid covers your hgrid.ll to ensure proper spatial interpolation.

In your run directory, mkdir sflux and inside it, create symbolic links to the NARR files. e.g., if you run starts from June 10, 2004 and ends June 20, 2004, then

```
sflux_air_1.001.nc --> narr_air.2004_06_10.nc
```

```
sflux_air_1.002.nc --> narr_air.2004_06_11.nc
```

...

```
sflux_air_1.011.nc --> narr_air.2004_06_20.nc
```

```
sflux_air_1.012.nc --> narr_air.2004_06_21.nc (extra day to account for time zone difference)
```

Similarly for `sflux_rad_*.nc` and `sflux_prc_*.nc`. As described above, the number "1" after "air_" denotes the first data set used, with the second set taking priority; you can use up to 2 sets in SCHISM (which combines them with some given weights set in `sflux_subs.F90`); here you only use 1 set.

Preparing your own sflux inputs

After familiarize yourself with the NARR files and their format, you may embark on creating your own nc files. The best way is to modify existing matlab scripts (`src/Utility/Sflux_nc/readnc*.m`) included in the source code bundle, which have extensively in-line comments to guide you along the way.

Below are some conventions for .nc-files in sflux directory and additional files needed for sflux run:

wind: *u*-component is eastward, *v*-comp. is northward (normal math convention, not compass convention)

`windrot_geo2proj.gr3`: rotates winds in case they do not align with coordinate axes, i.e. lat/lon

`watertype.gr3`: 6 is clear water, 7 is muddiest. Search in `schism_init.F90` for different water types. Required only if `ihconsv=1`.

A few details

Below are some details from `sflux_9c.F90`, near the beginning of the code.

- Of all attributes in nc file, only 'base_date' is required;
- The grids for air, rad and prc can be different (but must be the same within each type and each source). Additional requirements for the structured grid in .nc: [lon,lat](nx,ny) give

x,y coord., nx is # of pts in x. Suppose a node in the grid is given by (i,j) ($1 \leq i \leq nx$), then **the quad (i,j), (i+1,j), (i+1,j+1,i,j+1) must be along counter-clockwise direction (otherwise you'll get an error suggesting that no parents can be found for a grid ndoe);**

- Search for "relative_weight" (inside netcdf_io) to change relative weights of the 2 sources for air, rad and prc if needed. All weights must > 0!
- in case of 2 sources/grids for a variable, use "1" as larger grid (i.e. encompassing hgrid.ll) and "2" as smaller grid. The code will calculate weights associated with the 2 grids, and if some nodes in hgrid.ll fall outside grid "2" the interpolation will be done on grid "1" only (see combine_sflux_data, in particular, bad_node_ based on area coordinates outside [0,1]). Both grids must start from stack 1 and have same # of stacks for each variable. However, within each nc file # of time steps can vary;
- air_1_max_window_hours (etc) are set in netcdf_io to define the max. time stamp (offset from start time in each) within each nc file. Besides those in netcdf_io, max_file_times (max. # of time records in each nc file) in routine get_times_etc() may need to be adjusted as well.

4.4 Outputs

Most of the SCHISM outputs can be found in outputs/, while the rest are in the same directory as [hgrid.gr3](#).

4.4.1 Run info output (mirror.out)

This is a mirror image of now-defunct screen output. Below is a sample:

Barotropic model without ST calculation

```
# of tracers in each module:      1      1      0
      0      0      0      0      0      0      0
Total # of tracers=              2
Index ranges of each module:      1      1      2
      2      3      2      3      2      3
      2      3      2      3      2      3
      2      3      2      3      2
# of global outputs=              27
done reading param.in; s2_mxnbt in param.in = 3.0000000000000000
lhas_quad= T
mnei, mnei_p =          4      9
lhas_quad= T
Global Grid Size (ne,np,ns,nvrt):  108   130   237   2
*****Augmented Subdomain Sizes*****
rank  nea  ne  neg  nea2  neg2  npa  np  npg  npa2  npg2  nsa  ns  nsg  nsa2  nsg2
```

0	30	14	16	40	10	43	24	19	43	0	72	37	35	72	0
1	28	14	14	38	10	40	23	17	40	0	67	36	31	67	0
2	26	13	13	32	6	38	23	15	38	0	63	35	28	63	0
3	23	13	10	30	7	34	22	12	34	0	56	34	22	56	0
4	23	13	10	30	7	34	22	12	34	0	56	34	22	56	0
5	28	14	14	38	10	40	23	17	40	0	67	36	31	67	0
6	26	13	13	32	6	38	23	15	38	0	63	35	28	63	0
7	30	14	16	40	10	43	24	19	43	0	72	37	35	72	0

*****Global Boundary Sizes*****

nope	neta	nland	nvel
1	13	1	31

*****Augmented Subdomain Boundary Sizes*****

rank	nope	neta	nland	nvel
0	0	0	1	7
1	0	0	1	7
2	0	0	1	10
3	0	0	1	10
4	1	4	1	7
5	1	7	0	0
6	1	5	1	6
7	1	7	0	0

Max. & min. sidelength= 19934.91537849107 7973.938973963872

done init (1)...

done init. tracers..

done initializing cold start

Done initializing variables

Done initializing outputs

done computing initial vgrid...

done computing initial nodal vel...

done computing initial density...

time stepping begins... 1 1440

done adjusting wind stress ...

```
done flow b.c.  
done hvis...  
done backtracking  
done 1st preparation  
done 2nd preparation  
done solver; etatot= 3.1245774014922456E-002  
done solving momentum eq...  
done solving w  
done solving transport equation  
done recomputing levels...  
done density calculation...  
TIME STEP=      1; TIME=      300.000000  
....
```

The 'etatot' highlighted above can be used as a quick and easy way to check if the run is progressing smoothly; it is the sum of the absolute value of surface elevation at each node, from which you can estimate the averaged elevation at each step. If it's too large or NaN, you have a problem.

4.4.2 Global output

SCHISM binary state output is emitted in a directory called `outputs/`. This directory must exist or you will get an immediate warning from the model. Depending on your MPI configuration, the outputs directory may exist in a central location (this is more common) or each processor may have an instance in which case you need to collect together the contents into a central location.

An example file name is `9_0000_elev.61`. More generally, the file name is: `[time_block /stack #]_[processor_no]_[variable_name].[fortran_unit]`

Variable name

The variable name is transparent and covered in the documentation.

Time block

The time blocks start from 1 and are sequential. The model buffers and writes data occasionally. Every `ihfskip` time steps it opens a new stack. For instance, if the time step is 120 seconds and `ihfskip = 10080` each stack will be 14 days long.

- "Neat" time lengths that will make meaningful analysis (e.g. 14 days) are usually easiest later when you postprocess;

- Some of the output post-processing scripts will run a lot better if the length of your simulation is an even multiple of `ihfskip`. This can be done by altering `ihfskip` or the simulation length - at the risk of lengthening the simulation a bit, the latter often produces a neater result.
- If your simulation length is not an even multiple of the time block length, the last time block will be truncated on the last block. This will cause some minor errors and warnings in the post-processing tools. In addition, if you then restart the run it is best to repeat and overwrite the truncated block - the post-processing tools do not work well with blocks that grow and shrink in the middle of the run.
- Even if the output blocks match the end of the simulation very neatly, the model (at the time of writing) will open a new block that will be unused. This is useful for the `autocombine_MPI_elfe.pl`, as the latter always waits until a new block to come out before starting to combine the previous block (and so it'd hang if the last block were not written out).

Processor number

The `mpi_processor` number starts at 0 and represents the MPI processor id from the task that wrote the output. If your processor writes to a shared directory, the files for different processors will be collocated, but since clusters are not always set up the same way and each process is writing locally this isn't always the case. The per-processor outputs are usually gathered into a global binary. The scripts that do this are called `combine_output*.f90` (a simple perl script `autocombine_MPI_elfe.pl` exists to combine all available outputs transparently). Once you are done, you will have a binary file called something like `9_elev.61`. The stack # and the variable name remain. There is no utility for gathering the outputs in stack/time, instead most post-processing tools are able to work with multiple files.

Combine outputs of element- or side-centered variables

Some outputs are currently located in side/element centers. To combine these variables, a ".gr3" type file named "`sidecenters.gr3`" or "`centers.gr3`" is needed, which can be generated by these steps:

- Run model with `ipre = 1` (inside `param.in`), then build point files "`sidecenters.bp`" and "`centers.bp`" will be generated.
- Triangulate ".bp" with `xmgredit5` or `Triangle` (for very large grid) to generate a ".gr3" file. The mesh quality does not matter much if you can tolerate some visual artifacts later;
- Put "`sidecenters.gr3`" or "`centers.gr3`" inside your work directory (where `hgrid.gr3` is) and use the perl script `autocombine_MPI_elfe.pl` to combine the outputs.

- Visualize the combined outputs with xmvis6 or visIT.

The triangulated files .gr3 may have some elements outside the mesh domain. It's not necessary to remove the extra elements for combining and visualizing, but a cleaner triangulated ".gr3" is good for better visualization. A cleaner triangulated grid can be obtained by first reading hgrid.gr3 in xmgredit5, computing its boundary, and only then read in and triangulate the [sidecenters.bp](#) and [centers.bp](#). However, make sure no points are lost during the triangulation.

Combined outputs

After the process-specific outputs have been combined using combine_output8.f90, there are 4 types of standard output (i.e., node centered) in SCHISM, which correspond to the following 4 types of suffixes:

1. *.61: 2D scalar - no vertical structure (elevation and 9 other variables used in the heat exchange model);
2. *.62: 2D vector - no vertical structure: depth-averaged velocity (dahv.62), wind speed (u,v) and stress (tauxz,tauyz);
3. *.63: 3D scalar - has vertical structure (vertical vel., temperature, salinity, density, z-coordinates, diffusivity, turbulent kinetic energy and mixing length);
4. *.64: 3D vector - has vertical structure: horizontal vel.

All output variables are defined on [hgrid.gr3](#), i.e. @ nodes and in binary format, and can directly visualized using xmvis6 or visit (with SCHISM plug-in).

Format of combined outputs

Please consult the script read_output*.f90 for a complete description of the format.

The header part contains grid and other useful info:

1. Data format description (char*48) **!e.g., 'DataFormat v5.0'**
2. version (char*48) **!version of SCHISM (usually ignored)**
3. start_time (char*48) **!start time of the run**
4. variable_nm (char*48) **!variable name**
5. variable_dim (char*48)
6. # of time steps per stack (int)
7. output time step (real) **!=dt*nspool**
8. **nspool** (int) **!as in param.in**
9. ivs **!scalar (1) or vector (2)**

10. i23d !2D (2) or 3D(3)

Vertical grid part:

1. nvrt !total # of vertical levels

2. kz !# of Z-levels

3. h0, h_s, hc, theta_b, theta_f !constants used in defining SZ hybrid grid (not used in LSC²)

4. do k=1,kz-1

 ztot(k) !z-coordinates of each Z-level (not used in LSC²)

 enddo

5. do k=1,nvrt-kz+1

 sigma(k) !sigma-coordinates of each S-level (not used in LSC²);

 enddo

Horizontal grid part:

1. np !# of nodes

2. ne !# of elements

3. do m=1,np

 x(m) !x-coord

 y(m)

 h(m) !depth

 kbp00(m) !initial bottom indices

 enddo

4. do m=1,ne

 i34 !element type

 do mm=1,i34

 nm(m,mm) !node # (connectivity table)

 enddo

 enddo

The header is followed by time iteration part:

do it=1,ntime

1. time (real)

```

2. it !iteration #
3. do i=1,np
   eta(i) !surface elevation of node i
   enddo
4. do i=1,np
   if(i23d=2) then !2D output
   if(ivs.eq.1) out1
   if(ivs.eq.2) out1,out2 !2 vector components
   else !i23d=3 !3D output
   do k=max(1,kbp00(i)),nvrt
   if(ivs.eq.1) out1
   if(ivs.eq.2) out1,out2
   enddo !k
   endif
   enddo !i
enddo !it

```

These structures can also be seen in the simple I/O utility code `read_output*.f90` included in the package.

In addition, SCHISM also allows users to output any internal arrays in the code using the routine 'schism_output_custom'; see `Hydro/misc_modules.F90` for details. The side- and element-centered outputs are done this way.

Lastly, the user may be interested in some maximum quantities. At the moment, SCHISM outputs two max files for elevation and depth-averaged velocity (`outputs/maxelev_*` and `outputs/maxdahv_*`). These files can be combined using `Utility/Combining_Scripts/combine_gr3.f90` to generate `maxelev.gr3` and `maxdahv.gr3`.

4.4.3 Station outputs

These outputs are invoked with `iout_sta=1`, and are found in `outputs/staout_[1..9]`, corresponding respectively to elev, air pressure, wind u, wind v, T, S, u, v, w. Each output has a simple ASCII format:

Time(sec), variable @ station 1,2,... (specified in `station.in`)

4.4.4 Warning and fatal messages

Warning message (`outputs/nonfatal_*`) contains non-fatal warnings, while fatal message file (`fort.11`) contains fatal errors. In addition, you'd also check the system error outputs from your parallel job.

Chapter 5 Using SCHISM

5.1 Grid generation

UG modelling starts with grid generation, and the latter is often an *iterative* process. Here lies the greatest strength and challenges of using an UG model like SCHISM. Fortunately the gridgen tools have come a long way since 2000s and at the moment we are routinely generating large UGs with millions of nodes and ever higher resolution, all within a few hours (after the Digital Elevation Model (DEM) has been assembled).

We will only cover SCHISM specific aspects of gridgen using SMS (aquaveo.com) in this chapter; you are referred to schism.wiki for other info related to DEM preparation etc. Please also refer to Dr. Wood's thesis (http://ccrm.vims.edu/yinglong/wiki_files/Wood_MScThesis_2012-SMS-WMS-Chesapeake&DelawareBays-Grid.pdf) for some insight info on using SMS.

Remember, with some practice and care, you can generate great-looking UG like Fig. 5.1.

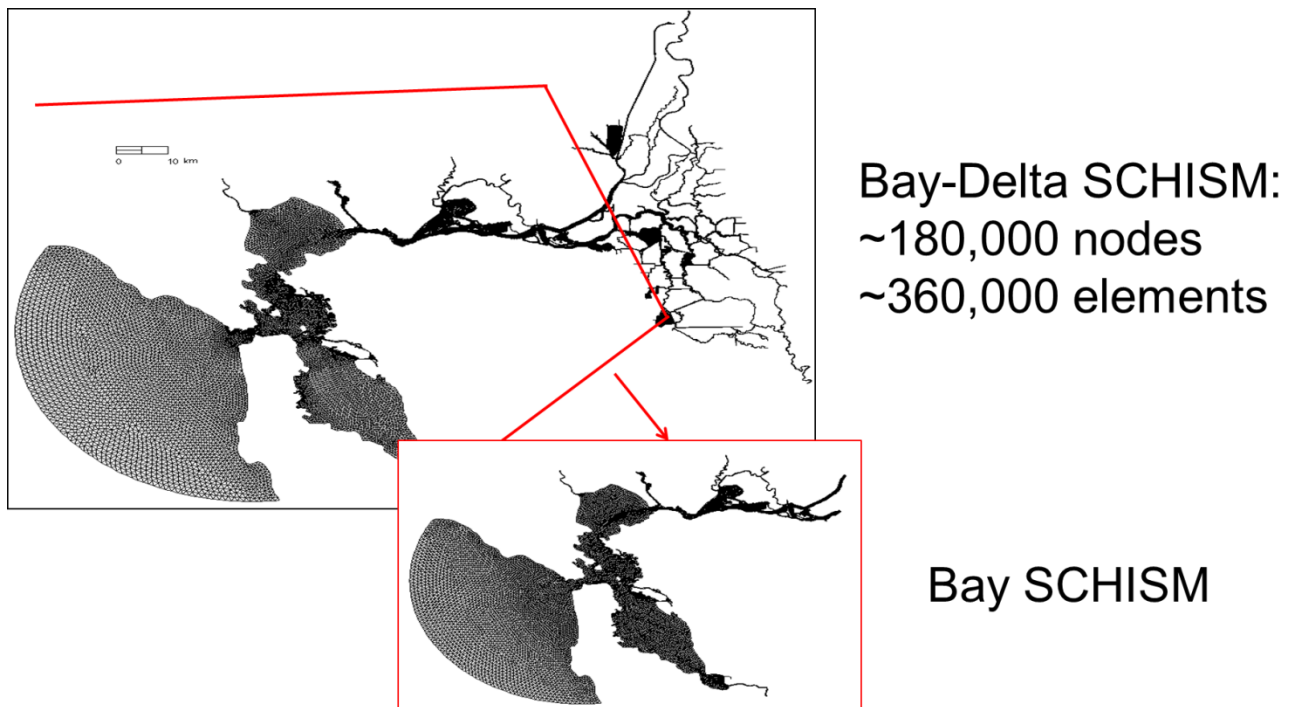


Fig. 5.1: San Francisco Bay & Delta grid (c/o CAL-Department of Water Resource (DWR)).

5.1.1 Beware of CFL number

You may be familiar with the CFL restriction associated with explicit (mode-splitting) models $CFL < 1$, where the CFL number is defined as

$$CFL = \frac{(|\mathbf{u}| + \sqrt{gh})\Delta t}{\Delta x} \quad (5.1)$$

where h is the local water depth, and \mathbf{u} is the flow velocity. Note that the surface wave celerity term is undefined when $h < 0$, and in the shallow area it's best to drop this term and assume $|\mathbf{u}| \sim 1$, and therefore $CFL \approx \frac{\Delta t}{\Delta x}$ when $h < 0.1\text{m}$.

Being an implicit model using ELM, SCHISM has a somewhat opposite requirement:

$$CFL > 0.4 \quad (5.2)$$

(you may be able to get away with $CFL > 0.2$ in some applications like tsunami). Therefore care must be taken in the grid generation process; otherwise numerical diffusion in ELM would ruin your results, which may manifest itself in the form of either noise or dissipation. For a given grid, the errors changes with Δt in a nonlinear manner, as shown in Fig. 5.2.

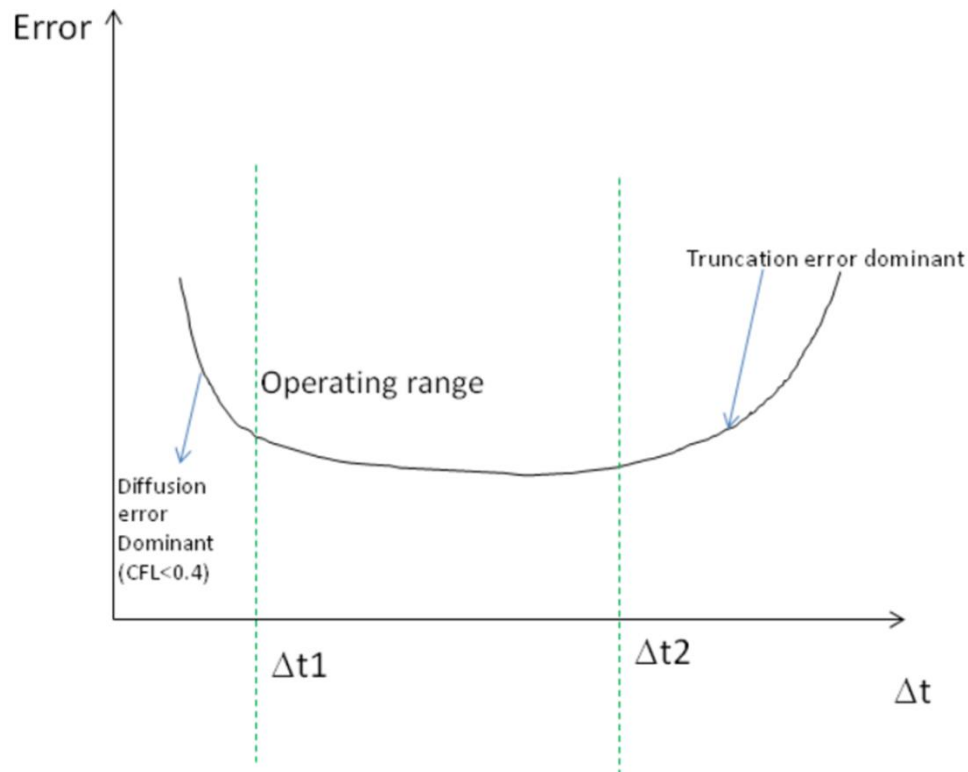


Fig. 5.2: Operational range for time step for a given grid size.

The gridgen process for SCHISM therefore starts with a range of time step for your application, We found the following ranges work for field applications: 100-400s step for barotropic applications, and 100-200s for baroclinic applications. Eq. (5.2) is then used to calculate the coarsest grid size at each depth (in anticipation of the smallest possible $\Delta t = 100\text{s}$). Table 5.1 shows some examples. You can see that the inverse CFL restriction is not restrictive.

Three immediate consequences of Eq. (5.2) are that (1) you do not need to reduce Δt if you refine grid (yay!); (2) if you have to coarsen grid, you will have to recheck (5.2); (3) if you have to reduce the time step for some reason, you may have to refine grid. The first consequence embodies the greatest strength (efficiency) of SCHISM as an implicit model. If you are doing a convergence study, you need to keep the CFL number fixed while reducing the time step (which means you have to reduce the grid spacing), as is done with explicit models. Therefore both explicit and implicit models converge (and are consistent) as $\Delta x, \Delta t \rightarrow 0$ and $\Delta x/\Delta t = \text{constant}$.

Table 5.1: Coarsest grid resolution at sample depths, assuming a ‘worse case’ scenario of $\Delta t = 100\text{s}$.

h (m)	Δx_{max} (m)
1	790
10	2500
50	5.5e3
100	7.9e3
500	1.7e4
1000	2.5e4
4000	5e4

5.1.2 Check CFL number in xmgredit5

With xmgredit5, you can very easily visualize CFL for the entire grid.

- `xmgredit5 -belel -1.e-10 hgrid.gr3` (note: '-belel -1.e-10' is used mainly to increase precision for lat/lon grid. Note: if your hgrid.gr3 is in lat/lon, you need to first project it, as the grid size dx in lat/lon is not in meters)
- Since the CFL inside ACE is calculated without \mathbf{u} , and a different form is used for CFL in the shallow $h < 0.1\text{m}$, we should impose a min depth of 0.1m (where $\sqrt{gh} = 1\text{m/s}$). You can do this by:
 - Edit \rightarrow Edit over grid/regions \rightarrow Evaluate, and then in the dialogue box, type `depth=max(depth,0.1)`. (Note that the Evaluate function is also very useful for generation of other .gr3 files needed by SCHISM)

- Special → Dimensionless numbers, and then in the right half of the dialogue box, type in 100s (Δt), Warning value (say 0.8), Unacceptable value (say 0.4) and depress 'Display filled' button. The color will appear in the main window, with red indicating good CFL, and green for bad CFL, and orange somewhere in between. You may also 'Display Courant number' but this may take a while to refresh, and so you may want to zoom into a small region first.
- Small patches of 'green' are OK with SCHISM, especially if they are in shallow area. However, avoid large patches of green in deeper area.
- Revise your grid accordingly if necessary.

In tsunami simulations, Δt has to be small ($\sim 1s$) due to small wavelength, and you can bypass the CFL condition by turning off advection in deeper depths as the advection is negligible there. You also need to make sure that each wavelength is resolved by at least 20 grid points.

5.1.3 Grid quality

SCHISM's stability means the model is very forgiving in grid quality; skewness (defined as the ratio between the max. side and the equivalent radius) of >12 can be used without problem (the skewness can be easily checked with `xmgredit5: Evaluate` → Acceptable skewness). However, for baroclinic applications, mesh quality may be important especially in critical regions. Also note that quality quads are required; split quads otherwise.

5.1.4 General rules

Unlike explicit models, you'll find gridgen for SCHISM is more 'intuitive' and 'freer'. Implicit model allows you to focus on physics instead of numerics. You are freer to resolve important features (e.g. jetties) without worrying about cost/instability. SCHISM is not picky about grid quality (except for quads). While the physics generally suggests that coarser resolution be used in deeper depths, this is not always the best practice. E.g., you may want to resolve the channel to more accurately represent the salt intrusion process.

Gridgen process for baroclinic applications requires more effort and **is often an iterative process, and so it's important to establish a good work flow from the start**. In the eddying regime, quasi-uniform grid or gradual transition in resolution should be used as much as possible.

While there are many methods for creating conceptual maps in SMS, we typically create representative isobaths (e.g., isobath at the highest gradient that represents slope) first and specify resolution along each arc based on Table 5.1.

We will cover some important aspects of gridgen for SCHISM below.

5.1.5 Channels

Channels serve as the main conduit for fresh and ocean water flow, and thus are important features for gravitational circulation. When gridding channels, try to use ‘patch’ method to generate quads as much as possible for better efficiency but do not round corners (Fig. A5). Use arcs to follow the thalweg and the steep slopes (highest gradient zone for isobaths). Avoid artificial blocking of channels (Fig. 5.3), which may lead to noisy flow field, but try to represent the channel cross section as faithfully as possible. With LSC², there is little need for bathymetry smoothing or other manipulation (and implicitness allows high resolution on steep slope and skew elements).

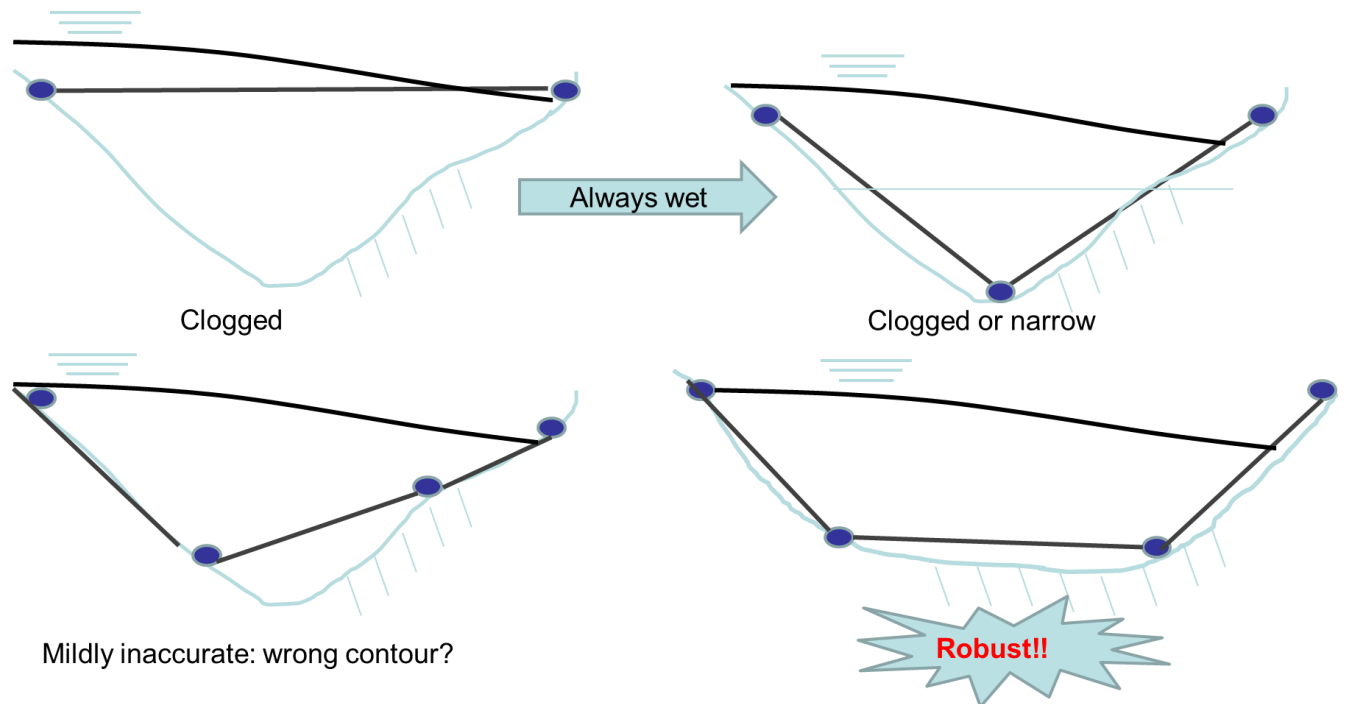


Fig. 5.3: Side view of channel representation. Remember SCHISM does not allow partial wet/dry.

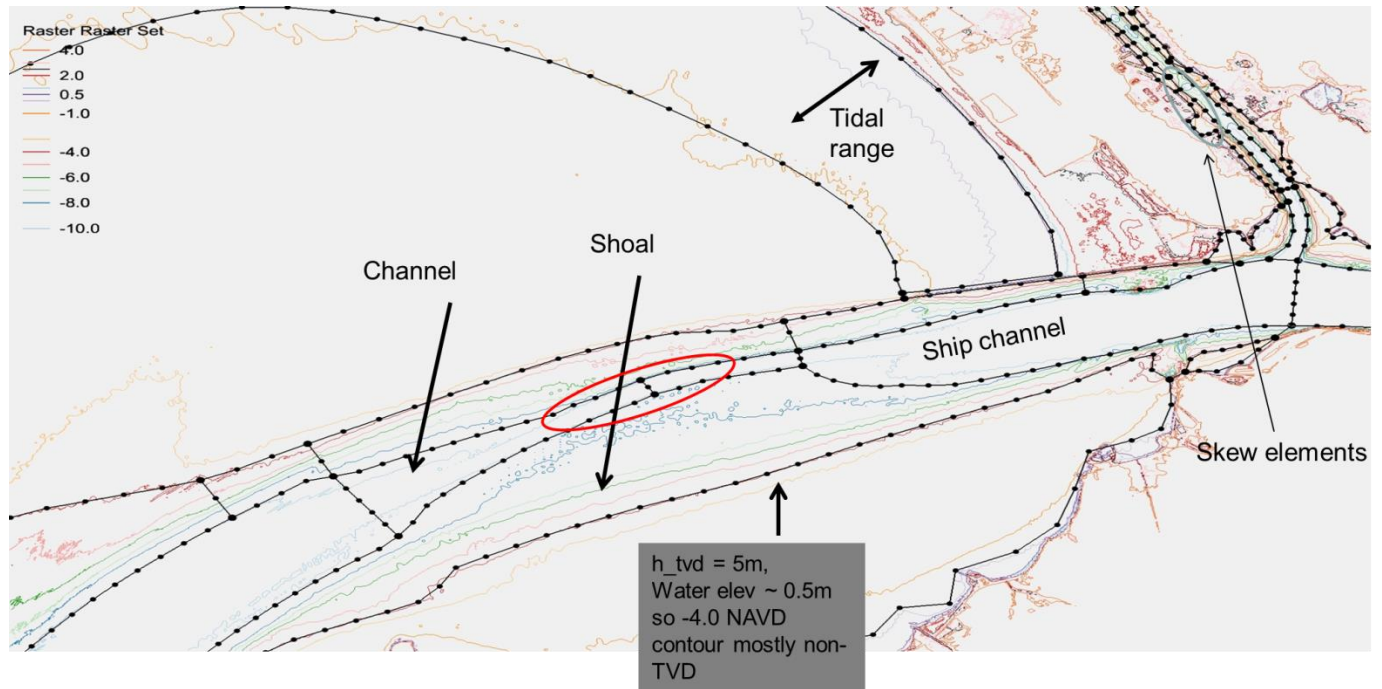


Fig. A5: SMS map in a stretch of San Francisco Bay.

5.1.6 Grid near wetting and drying

You may want to have an arc follow the initial shoreline. Reasonable grid transition should be done from shoreline to dryland. Use comparable or finer grid resolution in the dryland that is expected to be wetted, and then transition to coarser resolution beyond (to account for rare inundation).

Simulating wetting and drying well with SCHISM requires some effort. A critical parameter in shallow area is the bottom friction. Note that the bottom friction parameterizations are very different between 2D and 3D prisms.

From a physical point of view, the 2D and 3D models behave very differently. Consider a straight channel with variable depths, with flow coming from deeper part and going into shallower part. Fig. 5.4 shows the side views of 2D and 3D velocities.

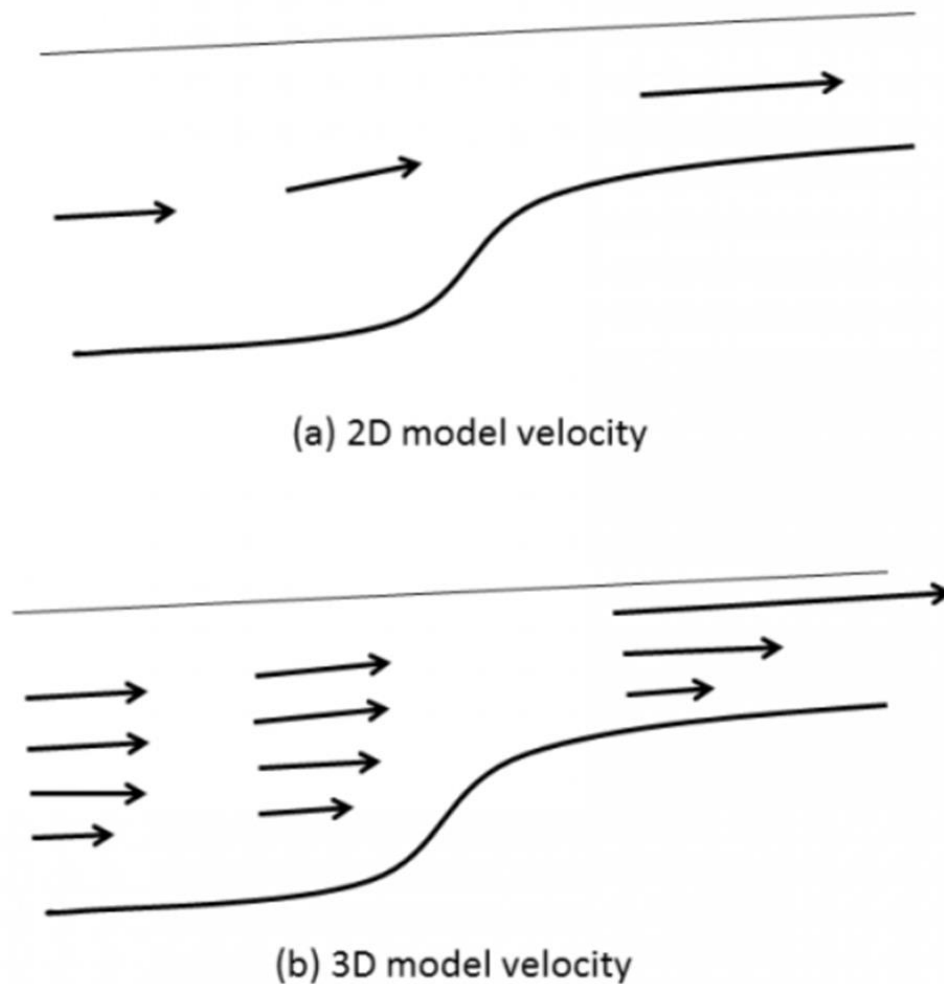


Fig. 5.4: Side view of channel flow for (a) 2D and (b) 3D cases. Volume conservation dictates larger velocities in shallow areas. Note that strong shear is possible in 3D model.

In 2D model, the velocity is depth-averaged and vertical shear is not represented. Strong friction merely translates into reduced velocity. In 3D model however, a large friction will lead to strong shear, although the depth integrated velocity value matches that from the 2D model. This problem is exacerbated by the exaggeration of C_d in the shallow if the roughness formula is used (since the bottom cell thickness is very small). A classical pathological velocity field obtained with SCHISM is seen in Fig. 5.5.

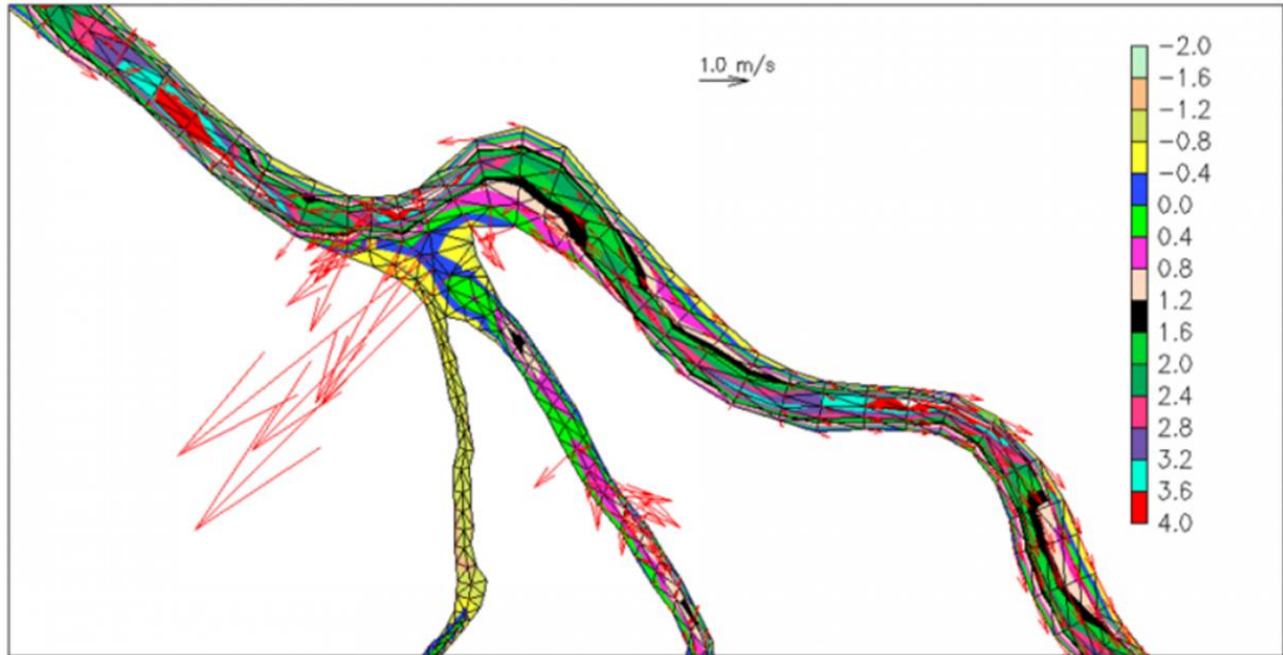


Fig.5.5: Noisy velocity field in shallow areas in SCHISM 3D.

There are a few approaches to resolve this issue. First, make sure the channel is not blocked. Second, try to use 2D prisms in shallows. Third, use a negative roughness (which tells the code that the value there is C_d not roughness), e.g., -0.001, to avoid large friction there. Lastly, using a larger `thetai` would also stabilize the wetting and drying fronts.

5.1.7 Post SMS checks

Dredging open boundary

A very common crash is related to the wet/dry near the open boundary. SCHISM does NOT allow the open boundary segments to become dry at ANY time. Therefore you need to make sure the depths there are deep enough compared to expected tidal range. An easy way is to impose a minimum depth near those segments (which can be done using `xmgredit5`) if the accuracy near the boundary is not of importance.

Since the wet/dry rule inside SCHISM is element-based, a node becomes dry if any of its surrounding elements becomes dry. As a result, you need to impose a minimum depth a couple of rows of elements into the domain, not just at open boundary nodes. This ensures that water can come into the domain without being blocked at the open boundary. On the other hand, wet/dry is allowed to occur at land/island boundaries.

If you care about wetting and drying near the open boundary location, we suggest you consider relocating the open boundary elsewhere. Also for upstream rivers where depths become negative

and you do not want to dredge depths there, you can use the bed deformation option (`imm=1`): start with a dredged boundary, and then gradually move the bed back to its original location.

DWR tools

In the svn directory of BayDeltaSCHISM you will find a number of useful python tools for DEM preparation and for aiding the gridgen.

5.1.8 Barotropic simulation

Grid quality requirement is relatively lax for barotropic simulations. Besides the considerations above, you mainly need to use appropriate resolution based on physics (e.g., generally coarser resolution in deeper depths and finer resolution for shallow depths). Remember: you are on implicit UG territory, and so you are free to resolve features as you wish!

5.1.9 Baroclinic simulation

The transport process is influenced by your choice of grid, and so the gridgen for baroclinic simulations needs some attention. The grid quality may need to be better in some critical areas (otherwise you may see noise). Quasi-uniform grid needs to be used in the eddying regime. Avoiding excessive resolution in high-flow area would speed up the transport TVD solver. Establish a good work flow and be willing to revise the grid.

5.3 Pre-processing

All `.gr3` and `.prop` inputs can be visualized/generated using `xmgredit5`. For other inputs, you can find some useful pre-proc tools in the `src/Utility` directory. In general, all FORTRAN scripts have a header that explains its purpose, how to use it (inputs/outputs) and sample compilation command.

- `2dm2gr3_m2m.pl` and `grd2sms.pl`

These 2 perl scripts are used to convert between `.2dm` (SMS) and `.gr3`.

- `interpolate_unstructured.f90`

This efficient script can be used to interpolate depths from `.gr3` onto another `.gr3` quickly. It uses a bucket search algorithm along either x or y direction.

- LSC² scripts: `gen_vqs.f90` and `plot_VQS.m`

The FORTRAN script can be used as a start for creating a LSC² grid and you need to use the `m-lab` script to plot out transects to see if the vertical grid makes sense. You may lump multiple transects into 1 transect.bp. If you want to use this type of vgrid, make sure you go thru the `.f90` carefully and understand the details.

- [read_3Dth.f90](#)

This script can be used to check *[23]D.th binary inputs; it extracts time series at a node and a level (3D).

- [Nudging scripts: gen_nudge.f90, gen_nudge_from_nc.f90](#)

The 1st script generates a simple elliptic nudging zone as *_nudge.gr3. The 2nd script generates *_nu.in based on gridded outputs.

- [change_hotstart3.f90](#)

This simple script shows you the internal structure of hotstart.in.

- [gen_hot_3Dth_from_nc.f90](#)

This script shows you an example of how to create hotstart.in from gridded outputs. It essentially consists of 3D interpolations.

- [Sflux_nc/](#)

The m-lab scripts inside this dir show you the structure of sflux*.nc as well as how to generate your own files.

5.4 Visualization

The easiest way to viz SCHISM outputs *.6? is via xmvis6. However, some functions in xmvis6 have become outdated due to the addition of LSC² (including: sample, slab at a fixed z level). Other functions (surface/bottom, transect, time history) still work OK. Therefore we strongly recommend you install ACE tools on your system following the instruction in ACE/.

DWR has developed a visIT plug-in for SCHISM and more info can be found on the wiki site.

5.5 Post-processing

You can find some useful pre-proc tools in the src/Utility directory.

- [Combining Scripts/](#)

combine_output8.f90 is used to combine process-specific binary to binary or netcdf (note that *combine_output9.f90* is the latest and it does not split quads into pair of triangles and xmvis6 has issues with that). *combine_hotstart6.f90* is used combine process-specific hotstart outputs (outputs/*_0*_hotstart) into hotstart.in. *combine_gr3.f90* is used to combine process-specific *maxelev_** and *maxdahv_** (ASCII) into *maxelev.gr3* or *maxdahv.gr3*. *autocombine_MPI_elfe.pl* is a simple perl wrapper script that automatically combines all available outputs during or after the run.

- [OneWayNestScripts/](#)

The purpose is to generate elev2D.th, SAL_3D.th, TEM_3D.th and/or uv3D.th from a large-domain run to be used in a small-domain run. This is especially useful as uv3D.th is usually hard to generate from other models.

(1) use interpolate_variables6.f90 to generate *[23]D.th for the small-domain run;

(2) use the new *[23]D.th for the small-domain run.

Note that *.in in the dir are sample inputs for these scripts.

- Time series extraction with read_output*.f90

This group of scripts read multiple binary outputs *.6? and extract time series of a point, a slab, a transect etc. They share similar code structure and can be used to understand the binary output format as well as how to do your own processing. You may start from read_output7_xyz.f90. After you are familiar with these scripts, you can easily customize them for your own purpose.

5.6 Simple test cases

For beginners, it's important to start from simple set-up and progress toward more complex set-up. As soon as you generate a grid, you'd run the pre-processor (ipre=1) to check any gridding error. Therefore we will explain a few simple test cases below; the set-up can be found in the svn directory of schism_verification_tests: Test_QuarterAnnulus/, Test_Flat_Adjust/.

5.6.1 Quarter Annulus

This is a simple 2D barotropic test for tidal propagation in a quarter annulus domain shown in Fig. 5.6. The outer boundary is open and the other 3 sides are land. A simple M2 tide is imposed on the open boundary; both elevation and velocity tidal B.C. are specified according to analytical solution. The velocity B.C. is important especially for indvel=0. A sample visual of outputs is shown in Fig. 5.7.

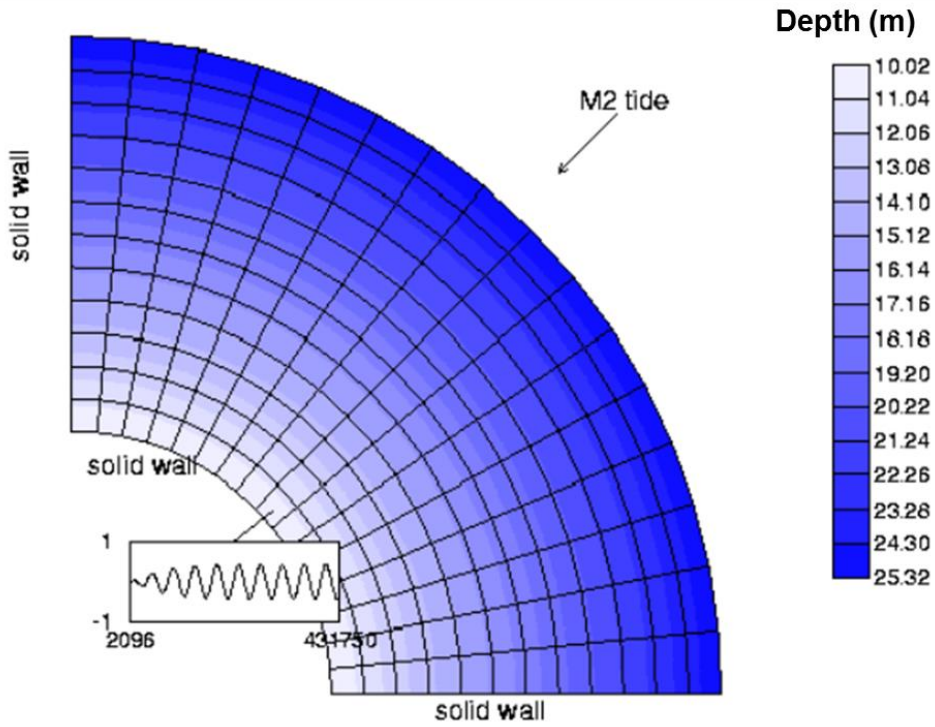


Fig. 5.6: Quarter annulus domain.

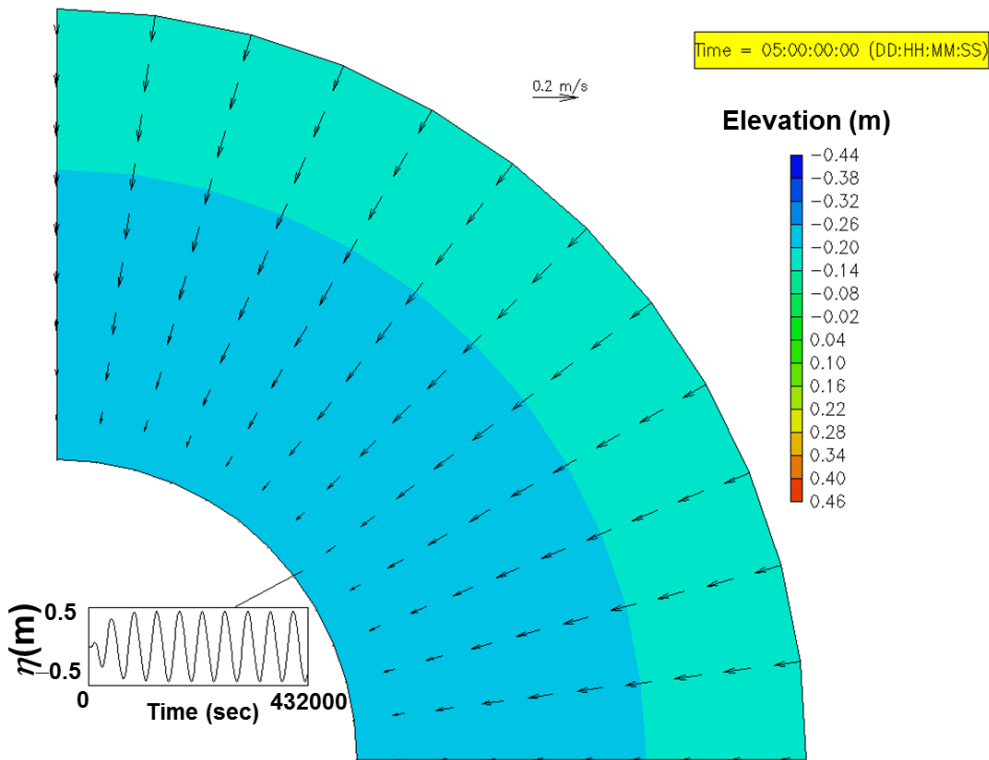


Fig. 5.7: Visualization of elevation and depth-averaged velocity, and time series of elevation near the interior boundary, using xmvis6.

5.6.2 Lock exchange

This is the simplest baroclinic test. Two fluids of different density (via salinity) are placed on each half of a rectangular box initially (Fig. 5.8), resulting in a density front propagating in both directions (Fig. 5.9). This test has been frequently used to test numerical dissipation of a model (cf. Zhang et al. 2016). The results shown in Fig. 5.9 are obtained with `indvel=1`, and you can explore further with the ‘eddying regime’ option of `indvel=0`, possibly with a smaller Δt .

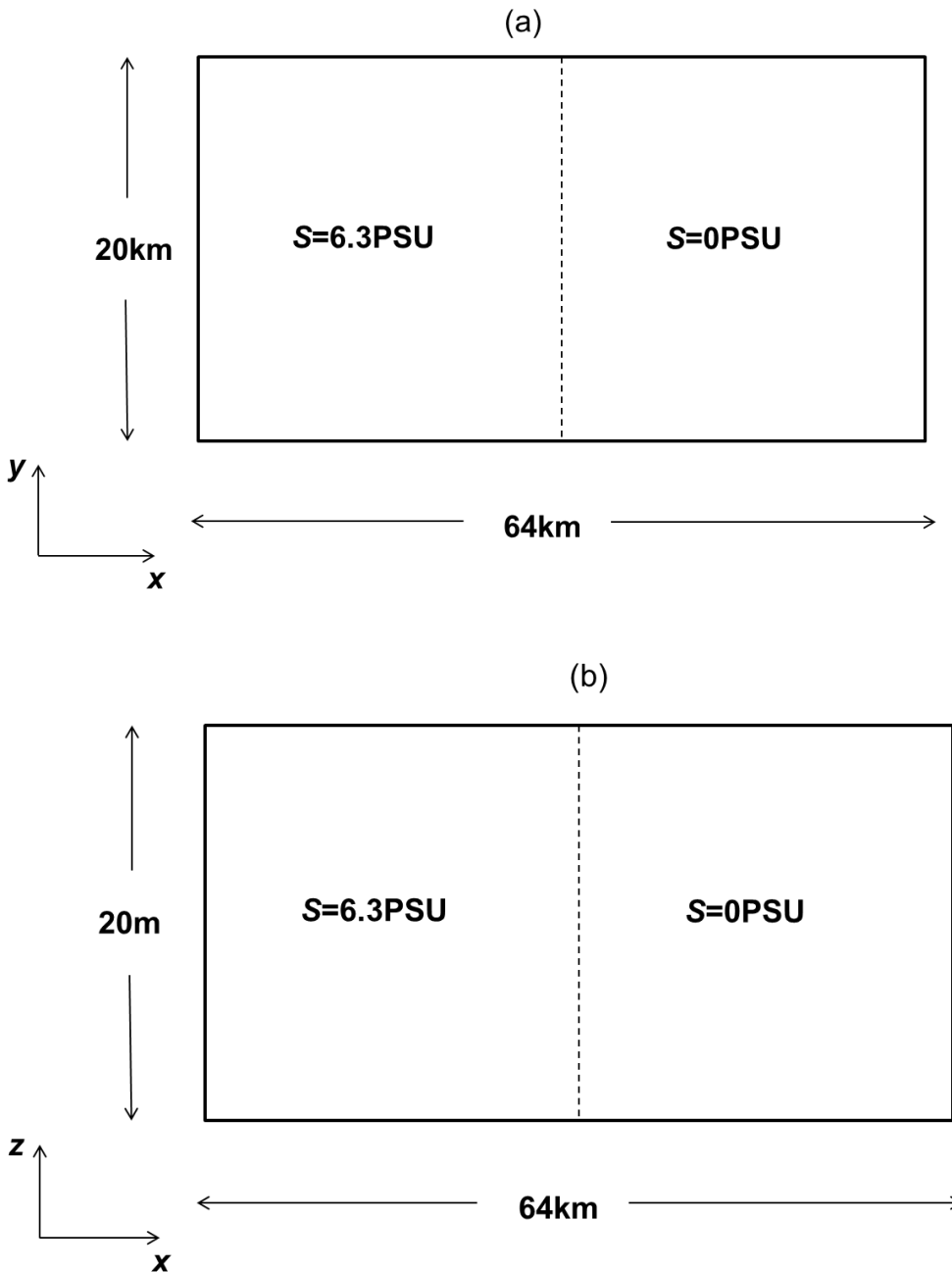


Fig. 5.8: (a) Top and (b) side view of domain.

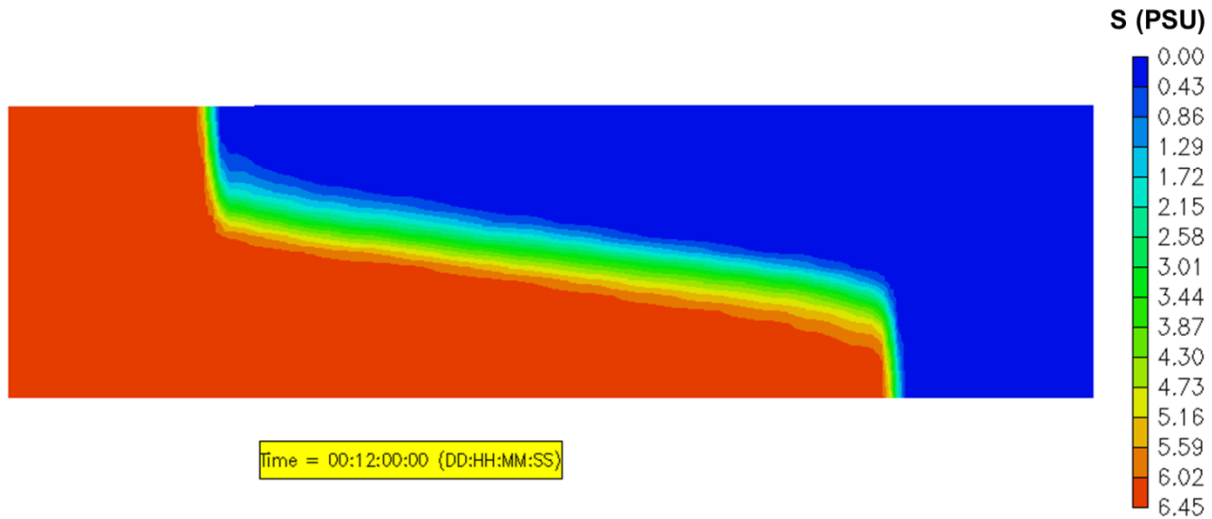


Fig. 5.9: xmvis6 transect view of the density front propagation at the end of simulation.

5.7 Trouble shooting and FAQ

Besides the following info, you may also search archived messages from the SCHISM mailing list on the same web where you registered yourself for the mailing list. Just log in, and select 'Archive' from the left of the screen, and then on right side of screen select 'Advanced search'.

1. My results show platform/compiler/CPU dependency

Due to some intricate differences between different compilers/platforms some minor differences in results are expected. Bit-by-bit match of results using different numbers of CPUs is impossible also. But the differences should be small and should stabilize over time iteration.

2. Run crashed with a fort.11 error message "QUICKSEARCH: no intersecting edge...."

First you need to viz the results before the crash (e.g. surface velocity) to see if anything is outrageously wrong. If the results look reasonable, contact the developers for more info.

3. Run crashed with a fort.11 error message "bktrk_subs: overflow..." or "MAIN: nbtrk > mxnbt"

The backtracking (ELM) in SCHISM is parallelized across MPI processes, and some arrays need to be allocated to store trajectories that exited the augmented domain. The dimension of those arrays is defined in *mxnbt* and *mnbt*, which are proportional to local # of side and # of levels, and the proportionality constants are defined in param.in:

`s1_mxnbt = 0.5`

`s2_mxnbt = 3.5`

(Gradually) increasing these (to say 1, and 4) will solve the problem. Note that these constants only affect memory consumption and not accuracy.

4. How to do a tidal simulation with SCHISM?

The simplest way is to use SCHISM 2D. If you have to use SCHISM 3D, make sure the vertical grid is reasonable.

5. My run crashed; how can I find out why?

The best way to find out the reason for a crash is to visualize the surface velocity with ACE/xmvis6. Usually you may see some large/noisy velocity somewhere, which may give you some hints on forcing etc.

Sometimes you want to visualize the problem right before the crash. You cannot use *autocombine_MPI_elfe.pl* as the last stack of output is incomplete. But you can use the FORTRAN combine script (e.g., *combine_output8*) to directly combine an incomplete stack. Just follow the instruction in the header of this script to prepare the inputs and run. Then visualize the combined outputs with xmvis6.

6. How to impose river discharge if the depth is negative there?

See Chapter 5.1.7.

7. Run crashes with a dry boundary/depth error.

SCHISM does not allow any open-boundary nodes to become dry at any time; see Chapter 5.1.7.

8. I have large velocity at open boundary

In hydrostatic models the incoming velocity should be specified at open boundary. Over-specification i.e. elevation+velocity B.C. there usually avoids the problem, but the two B.C.'s must be consistent with each other.

In reality it's often difficult to find velocity B.C., and so a useful way is to use 1-way nesting approach

9. fort.11 indicates "Could not find suitable element in input"

This is likely because your *sflux*.nc* is not prepared properly: either the grid in *.nc* does not cover *hgrid.ll*, or the structured grid in *.nc* is not ordered counter-clockwise.

Chapter 6 SCHISM code structure

6.1 General info

SCHISM was written in MPI FORTRAN 90. Fig. 6.1 shows the directory structure of the SCHISM svn repository. General public have access to the release branch.

Fig. 6.2 shows its main work flow and major code blocks. Two files inside src/ (Hydro/schism_init.F90 and Hydro/schism_step.F90) represent bulk of the hydro code, and are driven by a master program (Driver/schism_driver.F90). The global variables are defined in Core/schism_glbl.F90, and the message passing routines are defined in Core/schism_msgp.F90.

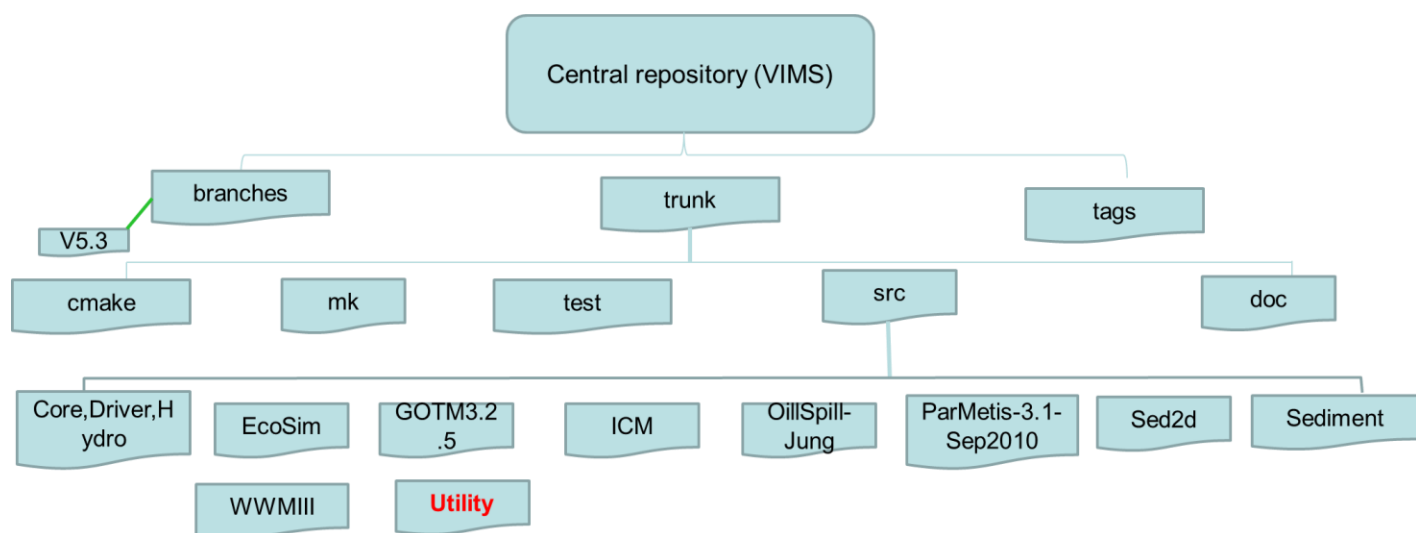


Fig. 6.1: SCHISM svn repository.

The domain decomposition is done using ParMETIS graph partitioning library (included in the src/). Fig. 6.3 shows an example of sub-domains generated by this lib.

The Utility dir has a lot of useful utility scripts for pre- and post-processing. The header of each program generally has info on its purpose and instructions on how to use it.

- ACE: source code and install notes for ACE tools; see wiki for more details.
- Combining_Scripts: FORTRAN and perl scripts used to gather outputs from SCHISM (e.g., [outputs/?_0000_elev.61](#) etc) into one binary file ([?_elev.61](#))
- Grid_Scripts: FORTRAN codes to interpolate depths from DEM files in either structured grid (raster) or unstructured grid format. While ACE/xmgredit5 can do similar things, these scripts are mostly for interpolating from very large DEM files.
- Gen_Hotstart: scripts for preparing hotstart.in from e.g., netcdf outputs from a structured-grid model
- Pre-Processing: various scripts for pre-processing (checking, viz etc)

- NCOM: scripts to read NCOM outputs and prepare hotstart.in and salt_nu.in, temp_nu.in, and Z0 components on the boundary in bctides.in.
- OneWayNestScripts: scripts for 1-way nesting, by preparing *[23]D.th (elevation, horizontal velocity, salinity and temperature boundary condition) that can be used for the 'small-domain' run.
- Particle_Tracking: particle tracking code that uses SCHISM's outputs for 3D tracking
- Post-Processing-Fortran: FORTRAN codes for extracting time series at selected 3D points (including transects). You can modify these codes for your own purposes.
- Sflux_nc: matlab scripts useful for preparing your own .nc files for sflux/. NARR_util/ has scripts to prepare .nc files from NCEP's NARR products.
- SMS: scripts to convert between .2dm of SMS and .gr3
- Vis_Matlab: matlab scripts for viz. We suggest you use this for presentations/publications, and use ACE tools or VisIT for daily research. At the moment, these scripts have not been updated to handle LSC² grid.

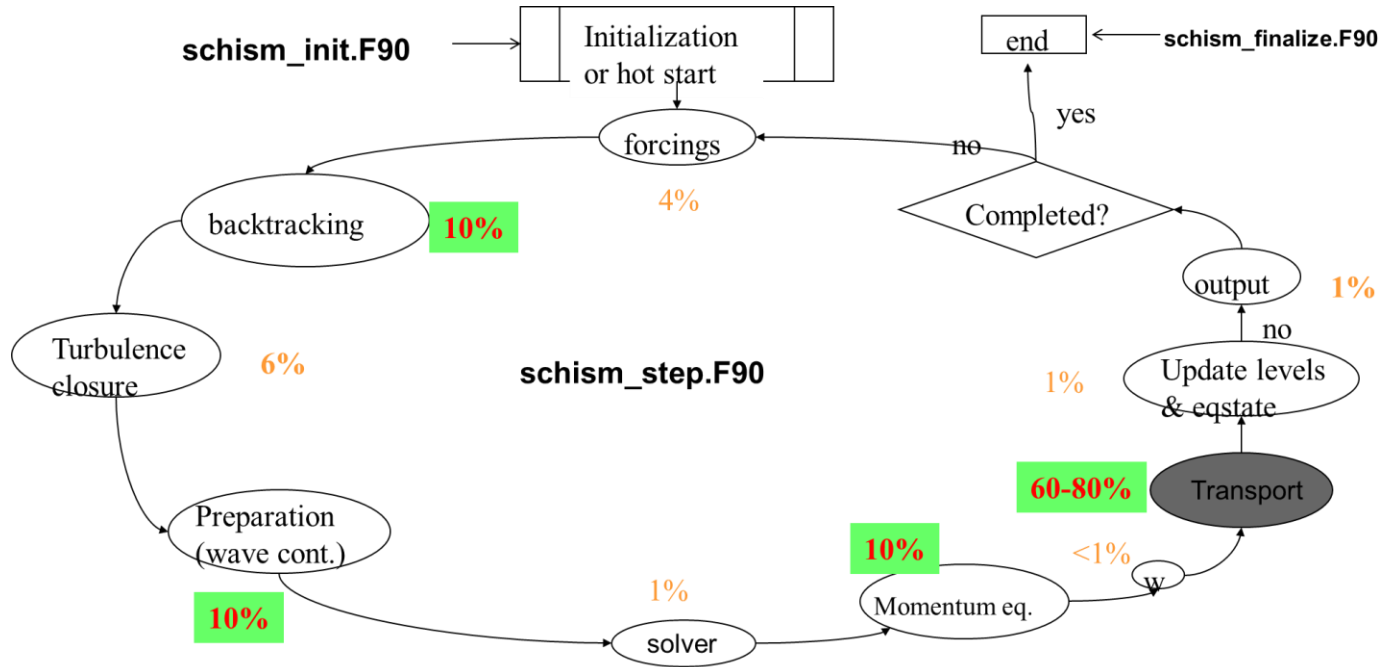


Fig. 6.2: SCHISM code work flow. The percentages are estimates from a test case with an earlier version and may not be up to date.

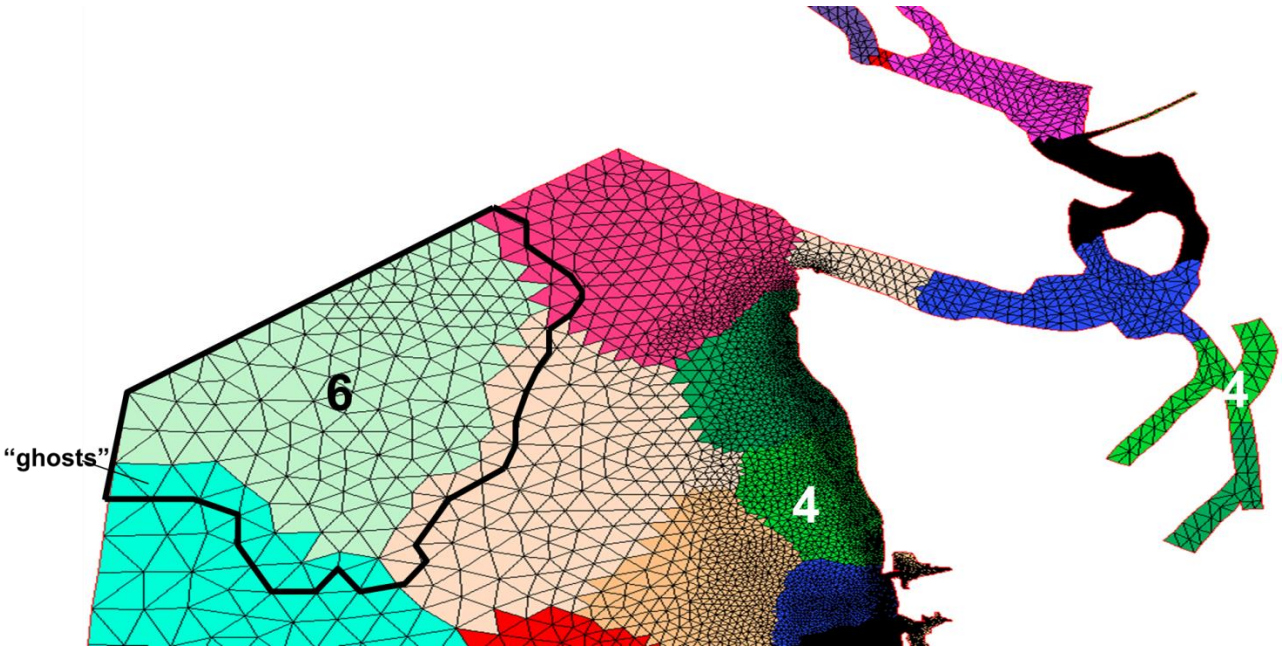


Fig. 6.3: Domain decomposition. Each color represents a sub-domain (‘resident domain’) taken by a MPI process and the thick black polygon represents the ‘augmented’ (=resident+ghost) domain of process 6.

6.2 For developers working on the code

6.2.1 Rules for contributing your code to be included in the SCHISM package

Here are some house rules for preparing your own code:

- 1) No spaces between “#” (pre-processor) and if/else/end for CPP flag;
- 2) Try to use the I/O channel number directly, e.g., read(61, etc instead of assigning a number to a variable (e.g. read(ich,). This'd facilitate others searching for conflicts.

6.2.2 I/O channels in SCHISM

You need to exercise caution when dealing with parallel I/O especially for writing. For writing outputs, you’d generally let only 1 process do the job, e.g.

```
if(myrank==0) write(10,*).....
```

If you do need to have all processes write e.g. debug messages, you’d consider using channel 12 which has been pre-set to point to multiple files from each MPI process (see below).

Here are all I/O channel numbers currently used in different sub-models of SCHISM (and so you’d avoid using them). A good way to find out if a channel is available is to issue the following cmd from src/:

grep "(61" */*.F90 -->Looks for '(61'

Please contact lead developer after you have selected a channel number to use in your module.

1. Hydro/
Channels between 8 and 200 are used by various codes for I/O. In particular:
 - a) **101 to 100+noutputs** (inclusive of both): reserved for global outputs (including from tracers like sediment, EcoSim, ICM, as well as WWM);
 - b) **201-250: non-standard outputs (e.g. at sidecenters, prism centers);**
 - c) **251 to 259**: reserved for station outputs
 - d) **16**: this channel points to mirror.out (on rank 0), the main message output about the run. You should use it with *if(myrank==0)*
2. WWM
 - a) **1100 to 1200**: handles for inputs/outputs etc
3. EcoSim
 - a) **600**: outputting some messages
4. ICM
 - e) **301 to 323**: reading channels for non-point source inputs for ICM
5. Sediment (SED and SED2D)
 - a) **26, 2626**

The following channels can be used:

- a) 10, 31, 32: used for one-off I/O – can be used by other sub-models as long as you close them immediately after use
- b) 12: this channel is initialized by different processes to point to files *outputs/nonfatal_XXXX*, where “XXXX” are the process IDs. Therefore it’s very useful for debugging purpose; you can use it anywhere in your part of the code to dump messages to these files.

6.2.3 Notes on SCHISM Code

6.2.3.1 Domain partitioning

The domain is first portioned into non-overlapping sub-domains (in element sense; see Fig. 6.3). Then each sub-domain is augmented with 1 layer of ghost elements. This is accomplished by the call `partition_hgrid()` early in the main program. After calling `acquire_hgrid(.true.)` immediately after that, the elements, nodes, sides in each augmented and non-augmented (i.e., without ghosts) domains are shown in Fig. 6.3. Intuition is typically followed although there are exceptions as described below.

6.2.3.2 Arrays and constants

	Global	Local non-augmented	Ghost	Augmented
Elements	ne_global	ne	neg	nea=ne+neg
Nodes	np_global	np	npg	npa=np+npg
Sides	ns_global	ns	nsg	Nsa=ns+nsg

1. `llsit_type :: iegl(iegb)`
`iegb` is a global element #. If the element is resident (not ghost), `iegl(iegb)%rank=myrank`, and `iegl(iegb)%id = local element index`, and `iegl%next=null`. If `iegb` is a ghost, then `iegl` list has two entries: `myrank` and the rank where `iegb` is resident. All processors have this info, but the lists are different (1st entry is usually `myrank` etc).
2. `llsit_type :: ipgl(ipgb)`
`ipgb` is a global node #. Use this list only when the node is resident (not ghost); it's confusing when `ipgb` is ghost. If `ipgb` is resident, `ipgl(ipgb)%rank=myrank`, and `ipgl(ipgb)%id = local node index`. `ipgl%next%next%next...` is the linked list, with ranks in ascending order. Unless `ipgb` is an interface node (i.e., resident in more than 1 process), the list has only 1 entry. All processors have this info, but the lists are different (1st entry is usually `myrank` etc).
3. `llsit_type :: isgl(isgb)`
`isgb` is a global side #. Similar to `ipgl`, if the side is resident (not ghost), `isgl(isgb)%rank=myrank`, and `isgl(isgb)%id = local side index`. `isgl%next%next...` is the list, with ranks in ascending order. All processors have this info, but the lists are different (1st entry is usually `myrank` etc).
4. `int :: ielg(ie), iplg(ip), islg(isd)`
The global element index of local element `ie` in the aug. domain. Similar for the other two (nodes/sides).
5. `int :: iegrpv(iegb)`
The rank # for global element `iegb` (before augmenting the domain). Used mainly in partitioning the grid.
6. Arrays that have similar meaning between global and local *aug.* domains, i.e., they do not have problem of getting outside the aug. domain: `i34`, `elnode` (old name: `nm`), `elside` (`js`), `ssign`, `snx`, `sny` ...

7. Arrays that need special attention in the aug. domain:

int :: ic3(1:i34(), ie) – positive if the neighbor element is inside the aug. domain as well (and in this case it is the local index of the neighbor element); 0 if (global) boundary; negative if the neighbor element is outside the aug. domain and in this case, the absolute value is the *global* element index.

int :: nne(ip) – total # of neighbor elements around local node ip, including those outside the aug. domain (i.e., same as nnegb()).

int :: indel(1: nne(ip),ip) – surrounding element indices. If inside aug. domain, this is the *local* element index; if outside, this is the negative of the *global* element index.

int :: iself(1: nne(ip),ip) – same as global iselfgb, i.e., the elemental local index for node ip in neighbor element indel() (even if it is outside).

int :: nnp(ip) – total # of surrounding nodes for node ip, excluding all nodes outside the aug. domain. For SCHISM, include all nodes outside.

int :: indnd(1: nnp(ip),ip) – list of surrounding nodes, excluding all nodes outside the aug. domain. For SCHISM, all nodes outside will have negative global index returned.

int :: isdel(1:2,isd) & isidenode(1:2,isd) – order of the two adjacent elements follows global indices, and so the vector from node 1 to 2 in isidenode(1:2,isd) forms local y-axis while the vector from element 1 to 2 in isdel(1:2,isd) forms local x-axis. Therefore either of isdel(1:2,isd) can be negative. The element index is local (positive) if it is inside the aug. domain; otherwise the minus of global element index is returned. The local side isd is on the boundary if and only if isdel(2,isd)=0, and in this case, isdel(1,isd) must be inside the aug. domain (i.e., positive) (if isd is resident). If isd is resident and not ghost, isdel() has the same meaning as serial code, i.e., is(1,isd)>0 (inside the aug. domain), and isdel(2,isd)>=0, and isd is on the boundary if and only if isdel(2,isd)=0.

double :: delj(isd) – meaningful only if isd is resident.

8. Boundary arrays:

Most arrays point to global bnd segment #. Most B.C. arrays are global as well.

nope_global – total # of open bnd segments in the global domain.

nope – total # of open bnd segments in the aug. domain.

iopegl(1:nope) – returns global open bnd segment # for a local open bnd segment.

iopegl(0,k) - # of local fragmentations of global open bnd segment k.

iopegl(j,k) - local segment # of jth ($1 \leq j \leq \text{iopegl}(0,k)$) fragmentation of global open bnd segment k.

nond(1:nope) – total # of open bnd nodes on each segment. The corresponding global array is nond_global().

iond(nope, 1: nond(1:nope)) – list of local node indices on each open bnd segment. The corresponding global array is iond_global().

nland - total # of land bnd segments in the aug. domain. nland_global is global.

nlnd(1:nland) - total # of land bnd nodes on each segment.

ilnd(nland, 1: nlnd(1:nland)) – list of local node indices on each land bnd segment.

nosd(nope) (ELCIRC) - # of open bnd sides on each segment.

iosd(nope, 1: nosd(nope)) (ELCIRC) – list of open bnd side on each segment.

noe(nope) (ELCIRC) - # of open bnd elements on each segment.

ioe(nope, 1: noe(nope)) (ELCIRC) – list of open bnd elements on each segment.

isbe(2,1:nea) (ELCIRC) – if the element is on the *local* open bnd, this returns the local segment # and element #. 0 otherwise.

isbs() (ELCIRC) – similar to isbe.

isbnd(-2:2,ip) (SCHISM) - If ip is on 1 open bnd only, isbnd(1,ip) points to the *global* segment # of that open bnd and isbnd(2,ip)=0; if ip is on 2 open bnds, isbnd(1:2,ip) point to the *global* segment #s for the 2 open bnds. If ip is on land bnd only (i.e., not on open bnd), isbnd(1,ip)=1 and isbnd(2,ip)=0. If ip is an internal node, isbnd(1:2,ip)=0.

Therefore, ip is on open bnd if isbnd(1,ip)>0 (and in this case isbnd(2,ip) may also be positive, even though isbnd(2,ip) may be outside the aug. domain), on land bnd (not on any open bnd) if isbnd(1,ip)=1, and an internal node if isbnd(1,ip)=0. If on

open bnd, isbnd(-2:-1,ip) are global index (i.e., isbnd(-1,ip)th node on the isbnd(1,ip)th open bnd);

isbs(nsa) - positive if a local side is on the global open bnd (in this case, isbs() is the *global* segment #); \square 1 if it is on land bnd; 0 if internal side.

iettype, ifltype, itetype, and isatype all take global bnd segment as argument; other b.c. arrays (eth etc) are also global.

uth(nvrt,nsa),vth(nvrt,nsa) – local b.c. for ifltype/=0 for a local side.

uthnd(nvrt,mnond_global, nope_global) vthnd() – global arrays.

elbc(ip) – ip is a local node.

9. Arrays defined in `elfe_msgp.F90`

`nnbr` - # of neighbor processors (excluding myrank).

`nbrrank(nnbr)` – rank of each neighbor processor.

`int :: ranknbr(0:nproc-1)` – neighbor # for each processor (0 if not neighbor).

`nerecv(nnbr)` - # of elements to be received from each neighbor.

`ierecv(1: nerecv(nnbr),nnbr)` – list of element indices (ghost in myrank) to be received from each neighbor (where the elements are resident and not ghost).

`nesend(nnbr)` - # of elements to be sent to each neighbor.

`iesend(1: nesend(nnbr),nnbr)` – list of element indices (local resident in myrank) to be sent to each neighbor (where the elements are ghost).

Similar for nodes/side (`nprecv`, `iprecv` etc).

A note on ghost exchange: since the message exchanges between processors have to wait for each other in order to communicate collectively, it's not necessary to synchronize the processes.

10. ParMetis routine:

```
call ParMETIS_V3_PartGeomKway(vtxdist,xadj,adjncy,vwgt,adjwgt,wgtflag, &  
                               numflag,ndims,xyz,ncon,nproc,tpwgts,ubvec,options, &  
                               edgecut,part,comm)
```

My notes from manual:

(p : # of processors;

n : total # of vertices (local) in graph sense;

m : total # of neighboring vertices ("edges"); double counted between neighboring vertice u and v.

ncon : # of weights for each vertex.)

int(in) vtxdist(p+1) : Processor j stores vertices vtxdist(j):vtxdist(j+1)-1

int (in) xadj(n+1), adjncy(m) : locally, vertex j's neighboring vertices are adjncy(xadj(j):xadj(j+1)-1). adjncy points to global index;

int(in) vwgt(ncon*n), adjwgt(m) : weights at vertices and "edges". Format of adjwgt follows adjncy;

int(in) wgtflag : 0: none (vwgt and adjwgt are NULL); 1: edges (vwgt is NULL); 2: vertices (adjwgt is NULL); 3: both vertices & edges;

int(in) numflag : 0: C-style numbering from 0; 1: FORTRAN style from 1;

int(in) ndims: 2 or 3 (D);

float(in) xyz(ndims*n) : coordinate for vertex j is xyz(j*ndims:(j+1)*ndims-1) (C style; ndims*(j-1)+1: ndims*j for FORTRAN);

int(in) nparts: # of desired sub-domains (usually nproc);

float(in) tpwgts(ncon*nparts) : =1/nparts if sub-domains are to be of same size for each vertex weight;

float(in) ubvec(ncon) : imbalance tolerance for each weight;

int(in) options : additonal parameters for the routine (see above);

int(out) edgecut : # of edges that are cut by the partitioning;

int(out) part() : array size = # of local vertices. It stores indices of local vertices.

11. Important MPI routines (C code; FORTRAN code just adds a last argument as status)

- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
 Inputs:
 count - maximum number of elements in receive buffer (integer);
 datatype - datatype of each receive buffer element (handle);
 source - rank of source (integer);
 tag - message tag (integer);
 comm - communicator (handle).
 Outputs:
 buf - initial address of receive buffer (choice);
 status - status object (Status).
- `int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)` – nonblock receive.
 Inputs:
 buf - initial address of receive buffer (choice);
 count - number of elements in receive buffer (integer);
 datatype - datatype of each receive buffer element (handle);
 source - rank of source (integer);
 Tag - message tag (integer);
 comm - communicator (handle).
 Output:
 request - communication request (handle)
- `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
 Inputs:
 buf - initial address of send buffer (choice);
 count - number of elements in send buffer (nonnegative integer);
 datatype - datatype of each send buffer element (handle);
 dest - rank of destination (integer);
 tag - message tag (integer);
 comm - communicator (handle).
- `int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)` – non-block send
 Inputs:
 buf - initial address of send buffer (choice);
 count - number of elements in send buffer (integer);
 datatype - datatype of each send buffer element (handle);
 dest - rank of destination (integer);
 tag - message tag (integer);

comm - communicator (handle).

Output:

request - communication request (handle).

- int MPI_Allreduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm) – Combines values from all processes and distribute the result back to all processes

Inputs:

sendbuf - starting address of send buffer (choice);

count - number of elements in send buffer (integer). Also the size of the output (i.e., *ith* elements from each processor are summed up and returned as *ith* element of output);

datatype - data type of elements of send buffer (handle);

op - operation (handle) (e.g., MPI_SUM, MPI_LOR);

comm - communicator (handle).

Output:

recvbuf - starting address of receive buffer (choice).

- int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm) – only difference from MPI_Allreduce is that the result is sent to rank "root".
- int MPI_Gather (void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm) - Gathers together values from a group of processes.

Inputs:

sendbuf - starting address of send buffer (choice)

sendcount - number of elements in send buffer (integer)

sendtype - data type of send buffer elements (handle)

recvcount - number of elements for any single receive (integer, significant only at root)

recvtype - data type of recv buffer elements (significant only at root) (handle)

root - rank of receiving process (integer)

comm - communicator (handle)

Output:

Recvbuf - address of receive buffer (choice, significant only at root). The received values are stacked according to the rank number (i.e., first recvcount are from rank 0 etc).

- int MPI_Allgather (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm) - Gathers data from all tasks and deliver it to all.

Inputs:

sendbuf - starting address of send buffer (choice);

sendcount - number of elements in send buffer (integer)

sendtype - data type of send buffer elements (handle);

recvcounts - integer array (of length group size) containing the number of elements that are received from each process;

displs - integer array (of length group size). Entry i specifies the displacement (relative to recvbuf) at which to place the incoming data from process i;

recvtype - data type of receive buffer elements (handle);

comm. - communicator (handle).

Output:

recvbuf - address of receive buffer (choice).

- int MPI_Type_indexed(int count, int blocklens[], int indices[], MPI_Datatype old_type, MPI_Datatype *newtype) - Creates an indexed datatype; the corresponding routine in MPI2 is mpi_type_create_indexed_block().

Inputs:

count - number of blocks -- also number of entries in indices and blocklens;

blocklens - number of elements in each block (array of nonnegative integers);

indices - displacement of each block in multiples of old_type (array of integers);

old_type - old datatype (handle).

Output:

newtype - new datatype (handle)

Notes: the new MPI type treats multi-dimensional arrays in FORTRAN as 1D array, expanding with 1st index varying before 2nd etc. So this routine can be used to grab discontinuous data blocks from multi- dimensional arrays.

So if a 2D array is a(nvrt,nea) the corresponding 1D array is illustrated below:

$$\begin{array}{c}
\overbrace{\hspace{10em}}^{nvt} \\
nea \left\{ \begin{array}{cccc}
(1,1) & (2,1) & \cdots & (nvt,1) \\
(1,2) & (2,2) & \cdots & (nvt,2) \\
\vdots & \vdots & \vdots & \vdots \\
(1,nea) & (2,nea) & \cdots & (nvt,nea)
\end{array} \right.
\end{array}$$

Now suppose we need to grab all ghost elements `iesend(1:nesend)`, these will correspond to rows `iesend(i)` of the table. In this case the # of blocks is `nesend`, block length is `nvt`, and displacement of `i`th block is `(iesend(i)-1)*nvt`.

- `int MPI_Barrier (MPI_Comm comm)` - Blocks until all process have reached this routine.

Input: `comm.` - communicator (handle)

- `int MPI_Type_struct(`
`int count,`
`int blocklens[],`
`MPI_Aint indices[],`
`MPI_Datatype old_types[],`
`MPI_Datatype *newtype)` - Creates a struct datatype.

Inputs:

`count` - number of blocks (integer) -- also number of entries in arrays `array_of_types` , `array_of_displacements` and `array_of_blocklengths`;

blocklens - number of elements in each block (array);

indices – byte displacement of each block relative to the start of the type (array);

old_types - type of elements in each block (array of handles to datatype objects).

Output:

newtype - new datatype (handle)

- `int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,`
`int recvcnt, MPI_Datatype recvtype,`
`MPI_Comm comm)` - Sends data from all to all processes.

Inputs:

sendbuf - starting address of send buffer (choice);

sendcount - number of elements to send to each process (integer);

sendtype - data type of send buffer elements (handle);

recvcnt- number of elements received from any process (integer);

recvtype - data type of receive buffer elements (handle);

comm. - communicator (handle).

Outputs

recvbuf - address of receive buffer (choice)

6.2.4 Working on your own code

If you are working on the code you may be confused about the exchanges inside SCHISM. When should you use these?

The first thing you need to remember when writing MPI code with domain decomposition is that a rank (or MPI 'process') only has info in its 'augmented' (=resident + ghost) domain, and knows absolutely *nothing* outside this region.

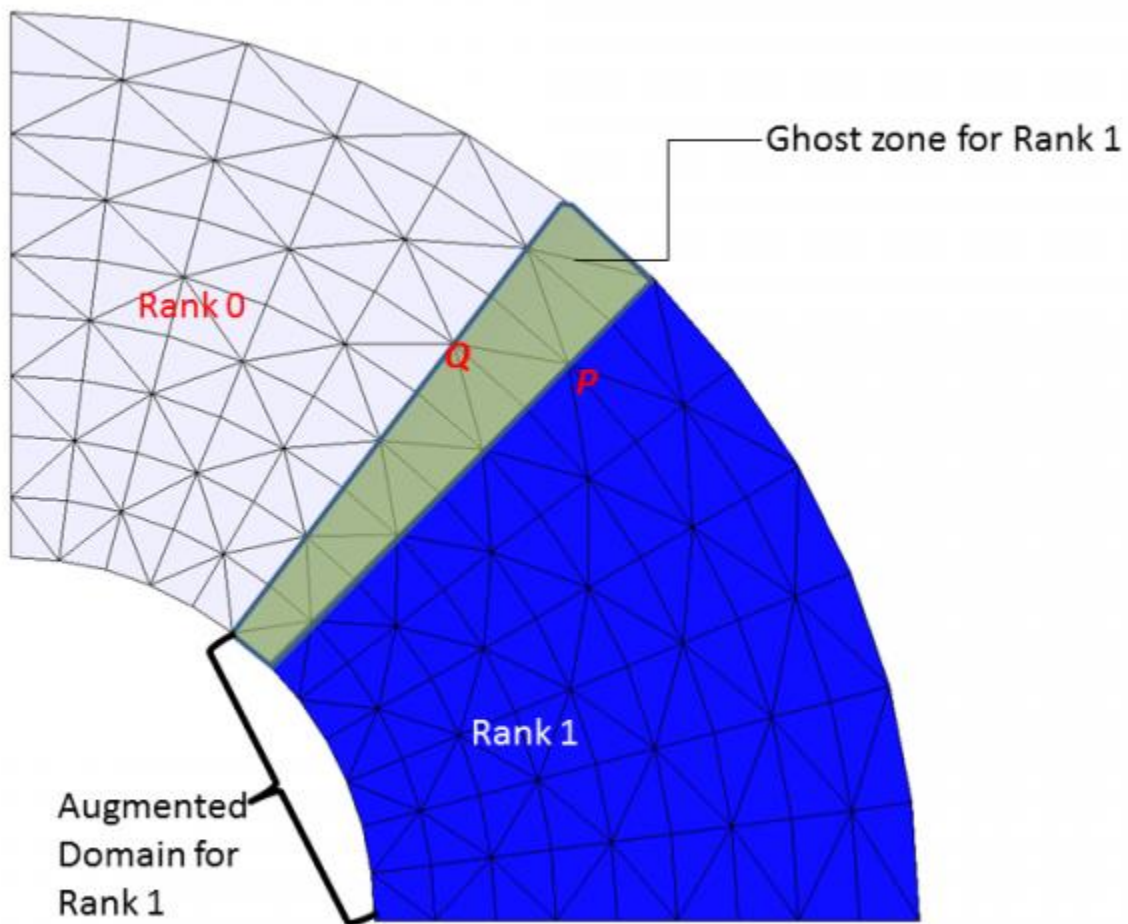


Fig. 6.4: Domain decomposition.

Consider Fig. 6.4. For example, you want to do averaging at each node of the sub-domain around its ball of elements.

```
do i=1,np !not 'npa'  
  sum1=0  
  sum_a=0  
  do j=1,nne(i)  
    ie=indel(j,i)  
    sum1=sum1+stokes(ie)*area(ie)  
    sum_a=sum_a+area(ie)  
  enddo !j  
  stokes_nd=sum1/sum_a  
enddo !i  
call exchange_p2d(stokes_nd)
```

....

Notice that you'd use 'np' instead of 'npa' (=np+npg; augmented domain) here. For any resident node 'P', indel() is well defined (because again, Rank 1 has info in its augmented domain including the ghost zone), and so the loops make sense. As long as all ranks have same info in each others' ghost zones (which is the purpose of exchange routines), 'stokes_nd' at an interface node (e.g. 'P') will be same across ranks.

However, for a ghost node 'Q', some surrounding elements are located outside the augmented domain (and in this case, indel() are actually negative!), and so if you use the same operation, erroneous values at ghost nodes would be generated. Therefore you cannot use npa in the outer loop.

Now, after this loop is executed, what happens to the ghost nodes like 'Q'? Since they are excluded on Rank 1, the 'stokes_nd' will be wrong there. Fortunately, some neighboring ranks have the correct values for these nodes, which are resident nodes in those neighboring ranks; e.g., 'Q' is a resident node of Rank 0. So now different ranks will have different values at some overlapping nodes, and this needs to be avoided. In order to make sure each rank has correct (and up-to-date) values in its augmented domain, you need to follow this loop with an exchange routine. Remember that for the exchange routines to work, you need to define the exchanged array in the augmented domain:

```
allocate(stokes_nd(npa)) !not 'np'
```

in order to allow each rank to receive info in the ghost zone. Description of all exchange routines used in SCHISM can be found in schism_msgp.F90.

The above is just one example of when exchanges are needed. Usually this involves some kind of queries into a neighborhood, but beware that there are other circumstances where exchanges are necessary. The most important thing to remember is, again,

A rank only has info in its 'augmented' domain, and knows absolutely nothing outside this region.

and

In MPI code, it's crucial to make sure all ranks have identical values in overlapping zone.

Now let's consider an example where no exchanges are needed. Say you want to simply go thru all elements and update some arrays defined at elements:

```
do i=1,nea !not 'ne'  
    qdl_e(i)=cde(i)*dte/area(i)  
enddo !i
```

Since all ranks have info of 'cde' and 'area' locally available, 'qdl_e' will be correctly calculated even inside the ghost zones, and its values are the same cross ranks there (since 'cde' and 'area' are same cross ranks in the overlapping zones). So in this case you'd use 'nea' instead of 'ne' in the loop. Of course, there is nothing wrong with using 'ne' in the loop followed by an exchange routine, but doing so would be less efficient and incur unnecessary communication cost.

Chapter 7 Modules

Manual for modules at the moment is lacking, and also some modules are under active development. We will add more info as it becomes available in the future.

7.1 Hydraulics

The manual for this module can be found at: http://ccrm.vims.edu/yinglong/wiki_files/structs_main.pdf.

7.2 Particle tracking

SCHISM supports a 3D particle tracking code in Utility/Particle_Tracking/ptrack2.f90, which can also be used for simple oil spill simulation. The code can do forward/backward tracking and has simple dispersion and beach retention processes for oil spill.

The inputs to this code are: **hgrid.gr3**, **vgrid.in**, **particle.bp**, ***hvel.64**, ***vert.63**, ***elev.61**. The main parameters are set in **particle.bp**. Below is a sample (bold texts are comments ignored by the code):

```
1 nscreen
0 mod_part (0: passive; 1: oil spill)
1 ibf (forward (=1) or backward (=-1) tracking)
1 istiff (1: fixed distance from free surface)
1 -124 46.25 ics slam0 sfea0 (see param.in)
0.1 8. 90. 10 960 9 !h0,rnday,dt,nspool,ihfskip,ndeltp (same as param.in
except for the last one, which is # of sub-divisions within each step)
16 !# of particles
1 84600. 385000 510000 -5. !particle id, start time (s), starting x,y, and
z relative to the instant surface (<=0)
2 84610. 300000 450000 0.
.....
[Oil spill parameters]
```

The main output is **particle.pth**, which uses the **drogue** format that can be visualized with **xmvis6**. Fig. 7.1 shows an example.

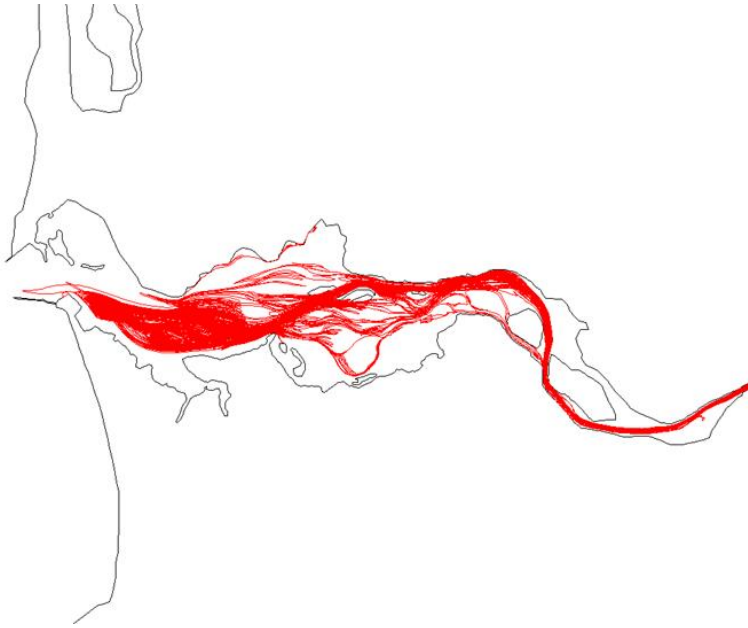


Fig. 7.1: Particle tracking with SCHISM.

7.3 3D Sediment model (SED3D)

The 3D sediment model inside SCHISM (USE_SED, also referred to as ‘SED3D’ sometimes to differentiate it from the 2D sediment module ‘SED2D’) is adapted from Community Sediment Transport Model (Warner et al. 2005). The algorithm is implemented on UGs and we have also reworked several components and added a morphological module (Exner equation). Detailed are reported in Pinto et al. (2012). Also note that for best results this module should be run in conjunction with the wave model (WWM) to account for the wave-enhanced bottom stress.

The main parameter input is `sediment.in`, which is a free-format file. The sample file has comments /explanations for each parameter; a few important ones are explained below.

- `sed_morph`

This parameter controls active morphology. Always start from off (0). If active morphology is required, make sure you adjust `depo_scale` to avoid excessive depositional flux at the bottom.

- `Nbed`

The # of bed layers affects the stability of the sorting, and `Nbed=1` works robustly with `sed_morph>0`.

Other inputs include `sed_class` and output flags in `param.in`, and a few .ic files: `bedthick.ic`, `bed_frac_[1,2..].ic`, and I.C. for concentrations of all classes (`*_[hvar]_[1,2...].ic` or

(*_[vvar]_[1,2...].ic). The B.C. inputs may include SED_[1,2..].th, SED_3D.th. The nudging inputs may be [SED_nudge.gr3](#) and [SED_nu.in](#).

The outputs from SED3D can be combined and visualized just as other SCHISM outputs.

A common crash occurs when the active morphology is on. At a river inflow boundary, often the depths will decrease over time due to deposition there and eventually the boundary will become dry/blocked. A work-around is to use a combination of ‘bare-rock’ bed around the boundary (specified in [bedthick.ic](#)), and ‘clear-water’ inflow as B.C. (sediment concentration =0 in [bctides.in](#)), and then input the incoming sediment concentration as point sources ([msource.th](#)) a distance away from the boundary.

7.4 2D Sediment model (SED2D)

The manual for this module can be found at:

http://ccrm.vims.edu/yinglong/wiki_files/Manual_SED2D.pdf

7.5 WWM

The manual for this module can be found at:

http://ccrm.vims.edu/yinglong/wiki_files/Manual_WWM.pdf

References

- Blumberg, A.F. and G.L. Mellor (1987) A description of a three-dimensional coastal ocean circulation model. In: *Three-Dimensional Coastal Ocean Models*, vol. 4, *Coastal and Estuarine Studies*, N. Heaps, editor, Washington, D.C.: AGU, pp. 1-16.
- Comblen, R., Legrand, S., Deleersnijder, E., and Legat, V. (2009) A finite element method for solving the shallow water equations on the sphere. *Ocean Mod.*, 28, 12-23.
- Canuto, V.M., A. Howard, Y. Cheng and M.S. Dubovikov (2001) Ocean turbulence I: one-point closure model. Momentum and heat vertical diffusivities. *J. Phys. Oceano.*, 31, pp. 1413-1426.
- Casulli, V. and E. Cattani (1994) Stability, accuracy and efficiency of a semi-implicit method for 3D shallow water flow. *Computers & Mathematics with Applications*, 27, pp. 99-112.
- Casulli, V. and P. Zanolli (2005) High resolution methods for multidimensional advection–diffusion problems in free-surface hydrodynamics. *Ocean Modelling*, 10, pp.137-151.
- Danilov, D. (2013) Ocean modeling on unstructured meshes, *Ocean Mod.*, 69, 195-210.
- Dukhovskoy, D.S., Morey, S.L., O’Brien, J.J., Martin, P.J., and Cooper, C. (2009) Application of a vanishing quasi-sigma vertical coordinate for simulation of high-speed deep currents over the Sigsbee Escarpment in the Gulf of Mexico, *Ocean Mod.*, 28, 250–265.
- Duraisamy, K. and J.D. Baeder (2007), Implicit Scheme For Hyperbolic Conservation Laws Using Nonoscillatory Reconstruction In Space And Time, *Siam J. Sci. Comput.* 29(6), 2607–2620.
- Flather, R.A. (1987) A tidal model of Northeast Pacific. *Atmosphere-Ocean*, 25, pp. 22-45.
- Galperin, B., L. H. Kantha, S. Hassid and A. Rosati (1988) A quasi-equilibrium turbulent energy model for geophysical flows. *J. Atmos. Sci.*, 45, pp. 55-62.
- Gravel, S. and A. Staniforth (1994), A mass-conserving semi-Lagrangian scheme for the shallow-water equations, *Mon. Wea. Rev.*, 122, 243-248.
- Griffies, S.M. and R.W. Hallberg (2000), Biharmonic Friction with a Smagorinsky-Like Viscosity for Use in Large-Scale Eddy-Permitting Ocean Models, *Monthly Weather Review*, 128, 2935-46.
- Ham, D.A., Pietrzak, J., and G.S. Stelling (2006), A streamline tracking algorithm for semi-Lagrangian advection schemes based on the analytic integration of the velocity field, *Journal of Computational and Applied Mathematics* 192, 168–174.

- Kantha, L.H. and C.A. Clayson (1994) An improved mixed layer model for geophysical applications. *J. Geophys. Res.*, 99(25), pp. 235-266.
- Le Roux, D.Y., Lin, C.A., Staniforth, A. (1997), An accurate interpolating scheme for semi-Lagrangian advection on an unstructured mesh for ocean modelling, *Tellus*, 49A, 119–138.
- Le Roux, D.Y., Sène, A., Rostand, V., and E. Hanert (2005), On some spurious mode issues in shallow-water models using a linear algebra approach. *Ocean Modelling* 10, 83–94.
- Mellor, G.L. and T. Yamada (1982) Development of a turbulence closure model for geophysical fluid problems. *Rev. Geophys.*, 20, pp. 851-875.
- Pinto, L., Fortunato, A.B., Zhang, Y., Oliveira, A. and Sancho, F.E.P. (2012) Development and validation of a three-dimensional morphodynamic modelling system, *Ocean Mod.*, 57-58, 1-14.
- Pond, S. and G.L. Pickard (1998) *Introductory Dynamical Oceanography*, Butterworth-Heinmann.
- Rodi, W. (1984) *Turbulence models and their applications in hydraulics: a state of the art review*. Delft, The Netherlands, International Association for Hydraulics Research.
- Shapiro, R. (1970), Smoothing, filtering and boundary effects, *Rev. Geophys. Space Phys.* 8 (2), 359–387.
- Shchepetkin, A.F. and J.C. McWilliams (1998), Quasi-Monotone Advection Schemes Based on Explicit Locally Adaptive Dissipation, *Monthly Weather Review*, 126, 1541-80.
- Song, Y. and D. Haidvogel (1994) A semi-implicit ocean circulation model using a generalized topography-following coordinate system. *J. Comput. Phys.*, 115, pp. 228-244.
- Umlauf, L. and H. Burchard (2003) A generic length-scale equation for geophysical turbulence models. *J. Mar. Res.*, 6, pp. 235-265.
- Warner, J.C., Sherwood, C.R., Arango, H.G. Signell, R.P., 2005. Performance of four turbulence closure models implemented using a generic length scale method. *Ocean Modelling*, 8, 81-113.
- Wilcox, D.C. (1998) Reassessment of scale determining equation for advance turbulence models. *AIAA J.*, 26, pp. 1299-1310.
- Zhang, Y., Baptista, A.M. and Myers, E.P. (2004) "A cross-scale model for 3D baroclinic circulation in estuary-plume-shelf systems: I. Formulation and skill assessment". *Cont. Shelf Res.*, 24: 2187-2214.
- Zhang, Y. and Baptista, A.M. (2008) "SELFE: A semi-implicit Eulerian-Lagrangian finite-element model for cross-scale ocean circulation", *Ocean Modelling*, 21(3-4), 71-96.
- Zhang, Y., Ateljevich, E., Yu, H-C., Wu, C-H., and Yu, J.C.S. (2015) A new vertical coordinate system for a 3D unstructured-grid model, *Ocean Modelling*, 85, 16-31.

Zhang, Y., Ye, F., Stanev, E.V., Grashorn, S. (2016). Seamless cross-scale modeling with SCHISM, *Ocean Modelling*, 102, 64-81. doi:10.1016/j.ocemod.2016.05.002

Zhang, Y., Stanev, E.V. and S. Grashorn (2016) Unstructured-grid model for the North Sea and Baltic Sea: validation against observations, *Ocean Modelling*, 97, 91-108.

Zeng, X., M. Zhao and R.E. Dickinson (1998) Intercomparison of bulk aerodynamic algorithms for the computation of sea surface fluxes using TOGA COARE and TAO data. *J. Clim.*, 11, pp. 2628-2644.