SCHISM v5.9 Manual

SCHISM development teams

August 15, 2021

Table of Contents

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/finite-volume formulation
- Unstructured mixed triangular/quadrangular grid in the horizontal dimension
- Hybrid SZ coordinates or new LSC^2 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 momentum advection
- Natural treatment of wetting and drying suitable for inundation studies
- Mass conservative and monotonicity-enforced transport: TVD^2
- **No bathymetry smoothing necessary**
- **Very tolerant of bad-quality meshes in the non-eddying regime**

1.2 Modeling system & application areas

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

1.3 Main differences between SCHISM and SELFE

1) Apache license

2) Vertical grid system (LSC^2, Zhang et al. 2015);

3) Mixed triangular-quadrangular horizontal grid;

4) Implicit vertical advection scheme for transport (TVD^2);

5) Higher-order transport in the horizontal dimension (3rd-order WENO);

6) Advection scheme for momentum: optional higher-order kriging with ELAD filter;

7) 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 two chapters; there is also a 'cheat sheet' near the end of this chapter. 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 the simple test cases in Chapter 5.5 and gradually progress toward more complex 3D baroclinic or coupled 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.nml), purple words to denote output file names (e.g. mirror.out), and green words to denote parameters specified in param.nml; e.g. nchi (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

*i*34(*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 mostly distributed via github.com. To download the source code:

- Make sure your system has git
- Clone schism repository into a directory where you want to store the code:

  git clone https://github.com/schism-dev/schism.git

  cd schism

  git checkout tags/v5.9.0

Note that you might need to add your system's ssh public key to your github.com account first before cloning. Alternatively, you can go to https://github.com/schism-dev and directly download the zip files.

General users also have access to **master** branch of the development repository (https://github.com/schism-dev/schism), but this manual is intended for the tag version only.

- Due to the large file size inside, the test suite used to test the code is still being distributed via svn. You'll need svn v1.8 or above (see http://svnbook.red-bean.com/ for a manual for using svn). Svn clients on linux/unix/windows/Mac should all work.

  svn co https://columbia.vims.edu/schism/schism_verification_tests

  Note that the test suite is used to test the **master** branch. Due to the differences between the release tags and master branch, you may need to modify some input files (e.g. param.nml) but we hope these tests show you how we set up models for different problems;

- The web site and this manual will be updated when a newer version/bug fixes becomes available.
- Useful user info can be found in src/Readme.beta_notes (including change of format for input files and bug fixes).

1.7 Compilation

Your system must have a FORTRAN and C compiler (and MPI wrapper like mpif90, mpicc), netcdf (version 4.4 or newer is required), python (version 2.7 and above) and perl installed.

1.7.1 Gnu make

You need the following two directories for gnu make.

- src/

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

- mk/

You need a 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.bora 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 (v4.4 and above preferred) on your local cluster (sometimes you may also need to add FORTRAN support for netcdf, e.g. –lnetcdff), and name of the executable (EXEC). Also the graph partitioning lib ParMETIS is compiled each time together with the SCHISM code, and you may 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). Make sure mk/sfmakedepend.pl and mk/cull_depends.py are executable (otherwise make them executable with chmod +x ….).

After all of these are done:

cd ../src
make clean
make pschism

You might get some errors if you did not use git to clone so the make cannot find the hash information; ignore it and proceed in making. The final name of executable has the cluster name and also additional suffixes if you turn on modules: pschism_* (note that you get a preview of this name when you do make clean so you know which modules you have turned on).

Note: the Makefile will automatically invoke Core/gen_version.py, which will generate Core/schism_version.F90, needed for querying the hash.

1.7.2 cmake

Alternatively, the cmake utility is a very powerful way of building the source code and utility script bundle. The main cmake files are found in cmake/. You'll need two essential files in this directory.

(1) SCHISM.local.build: used to toggle on/off optional modules (similar to *include_modules*);
(2) SCHISM.local.<cluster>: similar to Make.defs.local, this file specifies the most important environment variables like name/path of compiler, netcdf library etc. In general, cmake is quite adept at inferring some of these variables but sometimes you might need to overwrite the 'defaults' in this file. You can start from an existing file for a similar cluster e.g.
cp –L SCHISM.local.whirlwind SCHISM.local.myown

Once these two files are set:

mkdir ../build

cd ../build; rm -rf * (blow away old cache)

cmake -C ../cmake/SCHISM.local.build -C ../cmake/SCHISM.local.myown ../src/

The cmake is essentially a pre-processor for make, and it creates cache files (e.g. build/CMakeCache.txt, where you can inspect all env variables). After cmake is done, make can be executed in parallel or in serial mode:

make -j8 pschism (efficient parallel build)

or: make VERBOSE=1 pschism (serial build with a lot of messages)

The executable (pschism*) is found in build/bin/ and the compiled libraries are in build/lib/. If 'pschism' is omitted above, the main executable and all utility scripts will be built in bin/.

1.8 Bug fixes and major algorithmic changes from previous release

Svn versions:

(91) R5178: fix bug of saturate DO in ICM

(92) R5187: incorporated dry bnd treatment from bndry (c/o Jens Wyrwa). However, to get good volume conservation it's best to keep all open flow bnd's wet all the time as before;

(93) R5191: fixed a bug in iwind_form (accidentally re-init'ed after reading);

(94) R5202: fixed a bug on negative ICM tracer concentration - check negative values before sending back to hydro.

(95) R5204: a new option to control behavior of btrack when trajectory hots an open bnd (set vel=0 and exit). Goal is to eventually use this as default;

Git versions:

(96) 64b7181: fixed an efficiency issue and added nc checks in ptrack3 (c/o Marcel Rieker)

(97) 2138e77: Fixed a major bug introduced during reshuffling of _init for PDAF: need to call nodavel before sflux, as (uu2,vv20 are needed there.

(98) 5ff2198: revamped AGE module. B.C. now all should use 0. Only 0 and 1 should be used in AGE_hvar_[1:ntr/2].ic, where ntr is @ of age tracers, and the code will 'hold' concentration at 1 at those prisms (level specified in param.nml) that have 1 as i.c. I.C. in AGE_hvar_[ntr/2+1:ntr].ic should =0. Injecting '1' at dry elem's is allowed but results may be harder to interpret.

(99) 6cec698: bug fix for spherical coord in WWM (around dateline);

(100) c46c7bd (Feb 14, 2020): Fei tweaked WENO solver (wrt upwind);

(101) d3adb2c (Feb 14, 2020): changed to bilinear interp for HYCOM hot and nudging scripts (to speed up);

(102) 7506147 (Mar 1, 2020): add limiter for settling vel to avoid char line out of bnd;

(103) 5665418 (Mar 18, 2020): PR from Baptiste mengual for SED3D (sed_frition.F90 merged after the rest).

(104) eb00a00 (Mar 31, 2020): added 12th tracer model (USE_DVD of Klingbeil)

(105) aaf1306 (April 4, 2020): added hybrid ELM transport for efficiency (via branch hybrid_ELM);

(106) 40955ab (April 16, 2020): replaced the fatal error in nadv=2 (aptivity trap) with exit

(107) e703189 (May 6, 2020): reverted to old simple approach for depositional mass (c/o Baptiste Mengual).

(108) 14e201f (May 7, 2020): switched to ParMETIS v4.0.3;

(109) 33fb37f (Jun 30, 2020): fixed a bug in ielm_transport (Fei Y.).

(110) 55910d1 (July 31, 2020; via branch mem_leak_nc4): reverted the change in schism_io (close/open nc output after/b4 each write, to accommodate PDAF's flexible mode), as this has caused a lot of mem leaking (discovered by Nicole C. when testing ICM).

(111) eaf9093 (Aug 2020): Baptiste M. modified the computation of the bottom shear stress under combined waves and currents in SED3D

(112) 4ac56df (Aug 2020): Paul Ryan and Claire Trenham (CSIRO) fixed some init array issues in hydro and WWM.

(113) 7e9ff0a (Aug 20, 2020): Fei Y fixed a msource bug (init of msource at 0 at elem's not in source_sink.in led to ice rain;

(114) f311026 (Aug. 27, 2020): reverted init of msource to 0 for tracers other than T,S, c/o Nicole Cai. Using ambient values for other tracers can lead to artificial and additional accumulation of nutrients;

(115) 00b6f16 (Nov. 9, 2020): PR #21 from Kijin merged (SED): changed method to compute near bottom vel for LSC2.

(116) 8083cfa (Nov. 10, 2020): added option for transport solver only (itransport_only)

(117) 6dc44d3 (Dec 14, 2020): Fixed a bug, thanks to Kevin Kartin, on vortex formulism of wave force (missing a factor of area() in I_4);

(118) 28ff9d1 (Dec 16, 2020): added optional self-attraction loading tides. The option shares some constants with tidal potential: freq names and cut-off depth;

(119) b1bcaa0 (April 20, 2021): removed most of 'goto'. Remaining ones: harm.F90, lap.F90, WWM;

(120) 0cec024 (April 21, 2021): bug fix in misc_subs c/o of Fei (inunfl=1: nodeA in final extrap stage may be interface node);

(121) fb30239 (April 22, 2021): bug fix for ellipsoidal earth (tensor frames).

(122) fb79cdc (May 26, 2021): in interpolate_depth_structured2*, added an option to shift 1/2 cell for ll corner (c/o Charles Seaton);

(123) 5e87c24 (June 9, 2021): more changes in interpolate_depth_structured2* to extrap also into right/upper sides.

(124) 843c40f (19 June, 2021): fixed a bug in ptrack3 (pt_in_poly3; c/o Jilian Xiong) that affects quads;

(124) b2cf92b (29 June, 2021): fixed a bug in station outputs in basin scale cases ics=2 (local proj is not accurate if the station is far away from the local frame);

1.9 Changes in input and output format from previous release

The info below can also be found in src/Readme.beta_notes. Most changes are made in param.in (now renamed as param.nml).

- A e281d94 (25, June 2021): added a new option for SAL (Stepanov & Hughes 2004): iloadtide=3;
- bfb4afc (18 June, 2021): added an optional input 'shapiro_min.gr3' to be used with ishapiro=2;
- d07d75d (April 21, 2021): added T,S in required inputs for offline transport (to use hydro only results);
- b66e554 (April 14, 2021): added ishapiro=2 - Smagorinsky like filter option. In this option, shapiro0 is the coefficient;
- b43afea (April 2, 2021): changed x,y to double in nc outputs (for newer visIT);
- 9150d86 (Mar 31, 2021): added a new SAL option (iloadtide=2) using a simple scaling;
- 084e149 (Feb 25, 2021): Removed parameters: ibtrack_openbnd, iwindoff, dzb_decay (with hardwire); also removed option for negative roughness. Fixed race condition for marsh module.
- 3576c9c (Jan 24, 2021): added new option for source/sink input: if_source=-1 requires source.nc (which includes elem list inside; allows different time steps and # of records for volume/mass source/sinks. Also, now the source/sink values in .th must be single precision (not double);

- bcdfce6 (Jan 6, 2021): added main switch for nc output 'nc_out' (useful for other programs to control outputs);
- 28ff9d1 (Dec 16, 2020): added optional self-attraction loading tides; if iloadtide=1, need amp/phases in loadtide_[FREQ].gr3 (freq's shared with tidal potential);
- f8ba470 (Dec 3, 2020): Added a new option (meth_sink) to treat net sink: if an elem is dry with a net sink, vsource is reset to 0;
- f9043e2 (Nov. 12, 2020): merged with LRU branch wwm_lr; removed 'sav_cd' (isav=1 now requires sav_cd.gr3). New parameters related to WWM: fwvor_advxy_stokes,fwvor_advz_stokes, fwvor_advz_stokes,fwvor_breaking,cur_wwm,wafo_obcramp; Changes in WWM .nml: ALPBJ ->B_ALP, BRHD->BRCR.
- 8083cfa (Nov. 10, 2020): added option for offline transport solver only ('itransport_only');
- 7e9ff0a (Aug 20, 2020): added two options (selected by 'i_hmin_airsea_ex') for locally turning off heat/salt exchange.
  i_hmin_airsea_ex=1: exchange turned off if local grid depth<hmin_airsea_ex;
  i_hmin_airsea_ex=2: exchange turned off if local water depth<hmin_airsea_ex

  This replaces the change made by adde1aa (July 8, 2020): added a new parameter 'hmin_airsea_ex' (min total water depth for heat/salt exchange);
- 2f7bab1 (Jun 28,2020): reorganized ICM process (rates/fluxes) and added output flag in param.nml.
- 1477780 (May 8, 2020): changed sediment.in to sediment.nml (and made parameters lower case: BEDLOAD_COEFF, NEWLAYER_THICK,IMETH_BED_EVOL, SAND_SD50=>Sd50, SAND_ERATE=>Erate, SED_TYPE=>iSedtype, SAND_SRHO=>Srho, SAND_WSED=>Wsed, SAND_TAU_CE=>tau_ce, sed_morph_fac=>morph_fac, poro_cst=>porosity);

1.10 Known bugs

None known.

1.11 Some notes on launching parallel job on linux systems

You should consult the local system manual on how to prepare batch scripts and launch parallel jobs. Here are some general tips for some commonly encountered problems.

SCHISM uses distributed memory model via MPI. All global arrays are partitioned into MPI tasks (aka processing element or PE). Therefore you must make sure that there is enough memory to hold the local arrays. This is especially true if you invoke some modules that use large arrays (e.g. WWM). In general, simply launching the executable on the head node as

./pschism*

may not work due to memory limits (and system administrators do not like this anyway). Use parallel launchers (e.g. mpiexec etc) even if you only use 1 task.

A word on scalability: SCHISM is written to be scalable down to ~300 grid nodes per task. The model usually achieves good performance with < 1K nodes per task, which is what you may want to start with. Barotropic model is generally less scalable than the baroclinic model. For large core counts, you should consider using the PETSc solvers for better scalability.

Note that the hybrid openMP-MPI option is only available for the hydro module at the moment. You can still use this mode for other modules but the efficiency would not be good as only the master thread is doing all the tasks there. Hybrid mode outperforms pure MPI mode without PETSc on large core counts, provided that the hyper threading is turned on. Currently, it seems that pure MPI with PETSc is still the best approach for small to medium core count.

1.12 Typical work flows: a cheat sheet

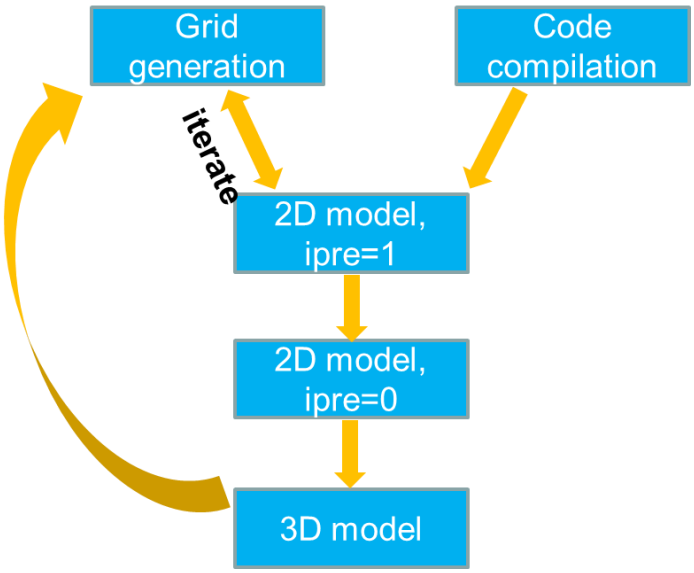


Fig. 1.1: Typical workflow with SCHISM modeling.

Grid generation

1. Generate grids in map projection, not lon/lat
2. Make sure major channels are resolved with at least 1 row of 'always wet' elements – no blockage of major channel flow
3. Always keep the map file and DEM sources and be willing to edit the grid, as often the model results (and sometimes performance) depend on the grid
4. Being an implicit model with ELM treatment of momentum advection, SCHISM has an 'operating range' for the time step. For field applications, the range is 100-400 sec for barotropic cases, 100-200 sec for baroclinic cases. If you have to reduce the time step, make sure you recheck the inverse CFL criterion as show below

5. First estimate the smallest Δt you'd anticipate (e.g., 100s for field applications), and then estimate the coarsest Δx at sample depths to make sure CFL>0.4 (cf. Table 5.1)
6. Resolving features is much easier with SCHISM – be game! Bathymetry smoothing is not necessary
7. Make sure all open boundaries stay wet during simulation (cf. Section 5.1.7). This condition has been relaxed from v5.8 but the code will still crash when an entire open boundary segment is dry
8. Implicit TVD^2 transport is very efficient, but horizontal transport is still explicit (and is the main bottleneck). Therefore beware of grid resolution in critical regions to avoid excessive sub-cycling; use upwind in areas of no stratification
9. Check the following things immediately after a grid is generated (via ACE/xmgredit5)
    - CFL>0.4 (at least in 'wet' areas). Note that you need to do this check in map projection (meters), not in lon/lat!
    - Minimum area: make sure there are no negative elements
    - Maximum skewness for triangle: use a generous limit of 13, mainly to find excessive 'collapsed' elements
    - Quad quality: use 0.5 (ratio of min and max internal angles) as threshold.

2D model: pre-processing

1. Check additional grid issues with a 2D barotropic model with ipre=1, ibc=1, ibtp=0
    - You can cheat without any open boundary segments during this step
    - Remember to mkdir outputs in the run directory
2. Iterate with gridgen step to fix any grid issues

2D model: calibration

1. Start from simple and then build up complexity
    - Simplest may be a tidal run with a constant Manning's *n*
2. Remember most of the outputs are on a per-core basis and need to be combined using the utility scripts; e.g., for global outputs (schout*.nc), use *combine_output11.f90* to get global netcdf outputs that can be visualized by VisIT; for hotstart, use *combine_hotstart7.f90* (cf. Section 5.4).
3. Examine surface velocity in animation mode to find potential issues (e.g. blockage of channels)
4. Negative river flow values for inflow
5. Check all inputs: 'junk in, junk out'

- There are several pre-processing scripts for this purpose
- Xmgredit5 or SMS is very useful also

3D model

1. The model needs velocity boundary condition at ocean boundary. There are two alternative approaches for this:
    a. Use FES 2014 tide package to generate tidal velocity, and use a global ocean model (e.g. HYCOM) to get sub-tidal velocity. Then use type '5' in bctides.in
    b. Use the one-way nesting capability to generate velocity boundary condition at the ocean boundary
        - You may use the 2D barotropic tidal model for this purpose, and then augment the tidal velocity with baroclinic velocity from a large-scale model like HYCOM
2. Avoid large bottom friction in shallow areas in 3D regions
3. Examine surface velocity in animation mode to find potential issues
4. Control the balance between numerical diffusion and dispersion (indvel, ihorcon…)
5. Transport solver efficiency may require some experience
6. LSC^2 grid requires some learning/experience, but is a very powerful tool (resembling unstructured grid in the vertical)
7. See for Section 5.7 commonly encountered issues in 3D setup

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, including the effects of :

Momentum equation: $\frac{D\boldsymbol{u}}{dt} = \mathbf{f} - g\nabla\eta + \boldsymbol{m_z} - \alpha|\boldsymbol{u}|\boldsymbol{u}L(x,y,z),$ (2.1)

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

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

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

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

Transport equations:

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

Equation of state:

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

where

∇	$\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y}\right)$
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]

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^{2}s^{-1}]

κ vertical eddy diffusivity, for tracers, in [m^{2}s^{-1}]

$\mathbf{F}_m$ horizontal viscosity [m^{2}s^{-1}]

F_h horizontal diffusion and mass sources/sinks [m^{2}s^{-1}]

Vegetation effects have been accounted for in Eq. (2.1). The main vegetation parameter is $\alpha(x,y) = D_v N_v C_{Dv}/2$ is a vegetation related density variable in m^{-1}, where Dv is the stem diameter, Nv is the vegetation density (number of stems per m^2), and C_{Dv} is the bulk form drag coefficient. Selection of C_{Dv} is the topic of other studies with values between 0 and 3 (Nepf and Vivoni 2000; Tanino and Nepf 2008), and is validated against reported lab study values. The underlying assumption used here is to treat the vegetation as arrays of solid cylinders, which is only a first-order approximation of the problem. Flexibility of the vegetation, sheltering effects within a cluster of vegetation can lead to one to two orders of reduction in the drag forces, and Gaylord et al. (2008) showed that the drag formulation is also species dependent. These additional complexities are outside the scope of the current study. In this paper, we assume C_{Dv} is a constant, but a vertically varying C_{Dv} (as suggested by Nepf and Vivoni 2000 and others) can be easily added as well; the latter can be used to approximate flexible stems (Nepf and Vivoni 2000; Luhar and Nepf 2011).

Since SCHISM allows 'polymorphism' with mixed 2D and 3D cells in a single grid (Zhang et al. 2016), we have different forms for the vertical eddy viscosity term

$$\boldsymbol{m}_z = \begin{cases} \frac{\partial}{\partial z}\left(\nu \frac{\partial \boldsymbol{u}}{\partial z}\right), & 3D\ cells \\ \frac{\boldsymbol{\tau_w} - \chi \boldsymbol{u}}{H}, & 2D\ cells \end{cases} \tag{2.1a}$$

and the vegetation term:

$$L(x,y,z) = \begin{cases} \mathcal{H}(z_v - z), & 3D \\ 1, & 2D \end{cases} \quad (2.1b)$$

where ν is the eddy viscosity, $\boldsymbol{\tau}_w$ is the surface wind stress, $H=h+\eta$ is the total water depth (with h being the depth measured from a fixed datum), $\chi = C_D|\boldsymbol{u}|$, C_D is the bottom drag coefficient, $\mathcal{H}()$ is the Heaviside step function

$$\mathcal{H}(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \quad (2.1c)$$

and z_v is the z-coordinate of the canopy. Note that $\boldsymbol{u}$ denotes the depth-averaged velocity in a 2D region.

2.2 Boundary conditions (B.C.)

The differential equations above need initial condition (I.C.) and B.C. In general, all state variables (η, **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. Note that these only apply to 3D cells; for 2D cells, Eq. (2.1a) has taken into account the B.C.

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

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

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 (**u** = w= 0) is replaced by a balance between the internal Reynolds stress and the bottom frictional stress,

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

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:

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

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

$$\mathbf{u} = \frac{\ln[(z+h)/z_0]}{\ln(\delta_b/z_0)}\mathbf{u_b},\ z_0 - h \le z \le \delta_b - h \tag{2.8}$$

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:

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

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

$$\nu\frac{\partial \mathbf{u}}{\partial z} = \frac{\kappa_0}{\ln(\delta_b/z_0)}C_D^{1/2}\,|\mathbf{u_b}|\mathbf{u_b},\ \ z_0 - \boldsymbol{h} \le \boldsymbol{z} \le \delta_b - h \tag{2.10}$$

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

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

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:

$$\frac{Dk}{Dt} = \frac{\partial}{\partial z}\left(\nu_k^{\psi}\frac{\partial k}{\partial z}\right) + \nu M^2 + \kappa N^2 - \epsilon + c_{fk}\alpha|\boldsymbol{u}|^3\mathcal{H}(z_v - z) \tag{2.12}$$

$$\frac{D\psi}{Dt} = \frac{\partial}{\partial z}\left(\nu_{\psi}\frac{\partial \psi}{\partial z}\right) + \frac{\psi}{k}\left[c_{\psi 1}\nu M^2 + c_{\psi 3}\kappa N^2 - c_{\psi 2}\epsilon F_{wall} + c_{f\psi}\alpha|\boldsymbol{u}|^3\mathcal{H}(z_v - z)\right] \tag{2.13}$$

where ν_k^{ψ} and ν_{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

$$\psi = \left(c_\mu^0\right)^p K^m \ell^n \tag{2.14}$$

where c_μ^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:

$$\begin{aligned} \nu &= \sqrt{2} s_m K^{1/2} \ell \\ \kappa &= \sqrt{2} s_h K^{1/2} \ell \\ \nu_k^\psi &= \frac{\nu}{\sigma_k^\psi} \\ \nu_\psi &= \frac{\nu}{\sigma_\psi} \end{aligned} \tag{2.15}$$

where the Schmidt numbers σ_k^ψ and σ_ψ are model-specific constants. The stability functions (sm and sh) are given by an Algebraic Stress Model (e.g.: Kantha and Clayson 1994, Canuto et al. 2001, or Galperin et al. 1988). Following Shimizu and Tsujimoto (1994; ST94 hereafter), we set $c_{fk} = 0.07$ and $c_{f\psi} = 0.16$.

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

$$K = \frac{1}{2} B_1^{2/3} |\boldsymbol{\tau_b}|, \text{ or } \frac{1}{2} B_1^{2/3} |\boldsymbol{\tau_w}| \tag{2.16}$$

$$\ell = \kappa_0 d_b \text{ or } \kappa_0 d_s, \tag{2.17}$$

where $\boldsymbol{\tau}_b$ is a bottom frictional stress, κ_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), which can be viewed here.

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:

$$\begin{cases} z = \eta(1+\sigma) + h_c\sigma + (\tilde{h} - h_c)C(\sigma) & (-1 \le \sigma \le 0) \\ C(\sigma) = (1-\theta_b)\dfrac{\sinh(\theta_f\sigma)}{\sinh\theta_f} + \theta_b \dfrac{\tanh\left[\theta_f(\sigma+1/2)\right] - \tanh(\theta_f/2)}{2\tanh(\theta_f/2)} & (0 \le \theta_b \le 1;\ \ 0 < \theta_f \le 20) \end{cases} \tag{3.1}$$

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 $\theta_f \to 0$, the *S* coordinates reduce to the traditional σ-coordinates:

$$z = \tilde{H}\sigma + \eta, \tag{3.2}$$

where $\tilde{H} = \tilde{h} + \eta$ is the restricted total water depth. For $\theta_f >> 1$, more resolution is skewed towards the boundaries, and the transformation becomes more nonlinear. If $\theta_b \to 0$, only the surface is resolved, not the bottom, while if $\theta_b \to 1$, both are resolved (Fig. 3.4). For typical coastal applications, we suggest θ_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.

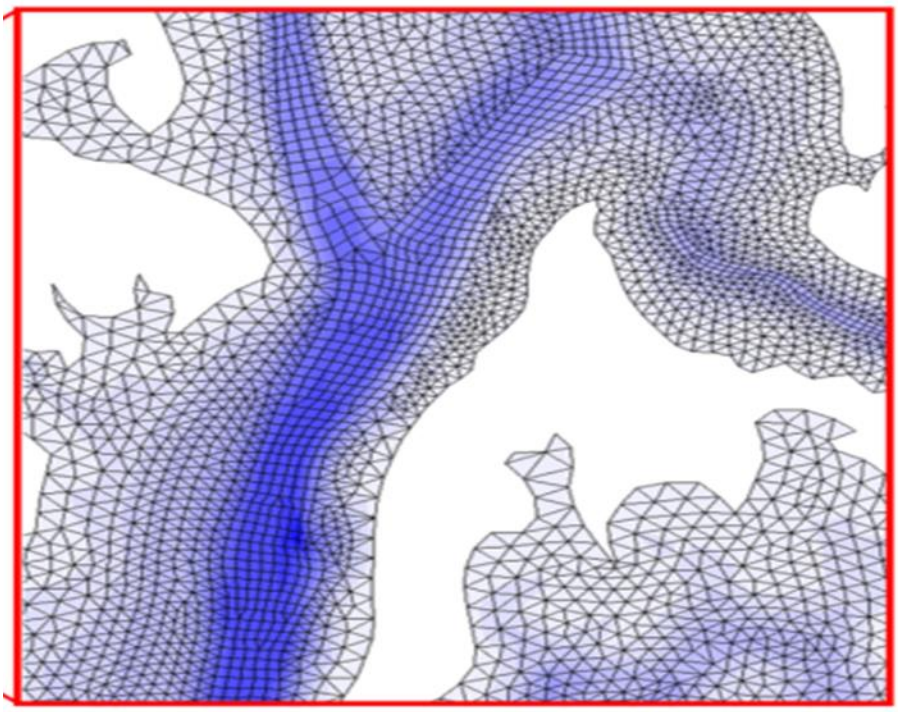


Fig. 3.1: Example horizontal grid (hgrid.gr3) of SCHISM.

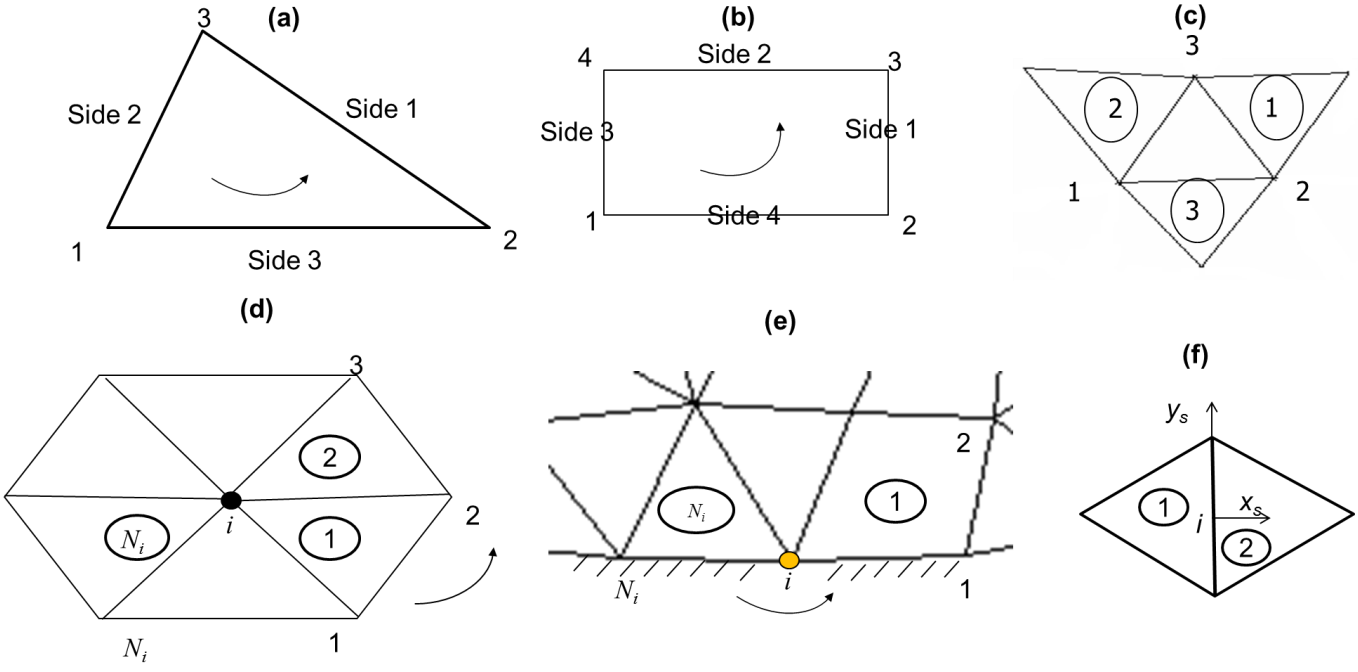


Fig. 3.2: Horizontal element convention. Local indices of nodes/sides for (a) triangular element, and (b) quad element. (c) Adjacent elements of an element (following side convention). (d,e): Local ball info (surrounding elements/nodes of node *i*) for (d) internal and (e) boundary node *i*. (f) shows 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.

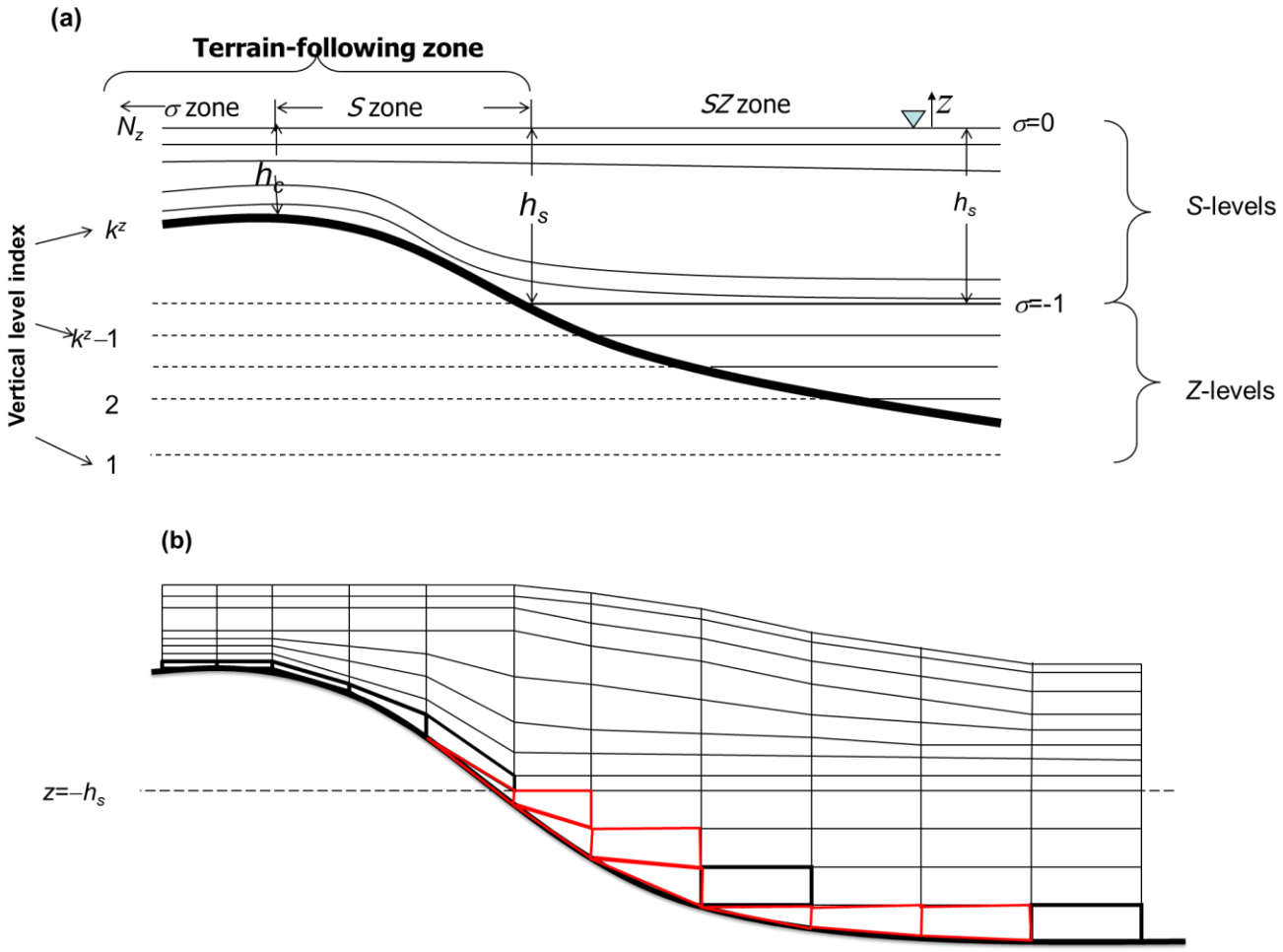


Fig. 3.3: Example of *SZ* grid. (a) Vertical coordinate systems (thick line is the bottom); (b) side view of a vertical transect resulted from *SZ* grid; the red cells are irregular/shaved cells.

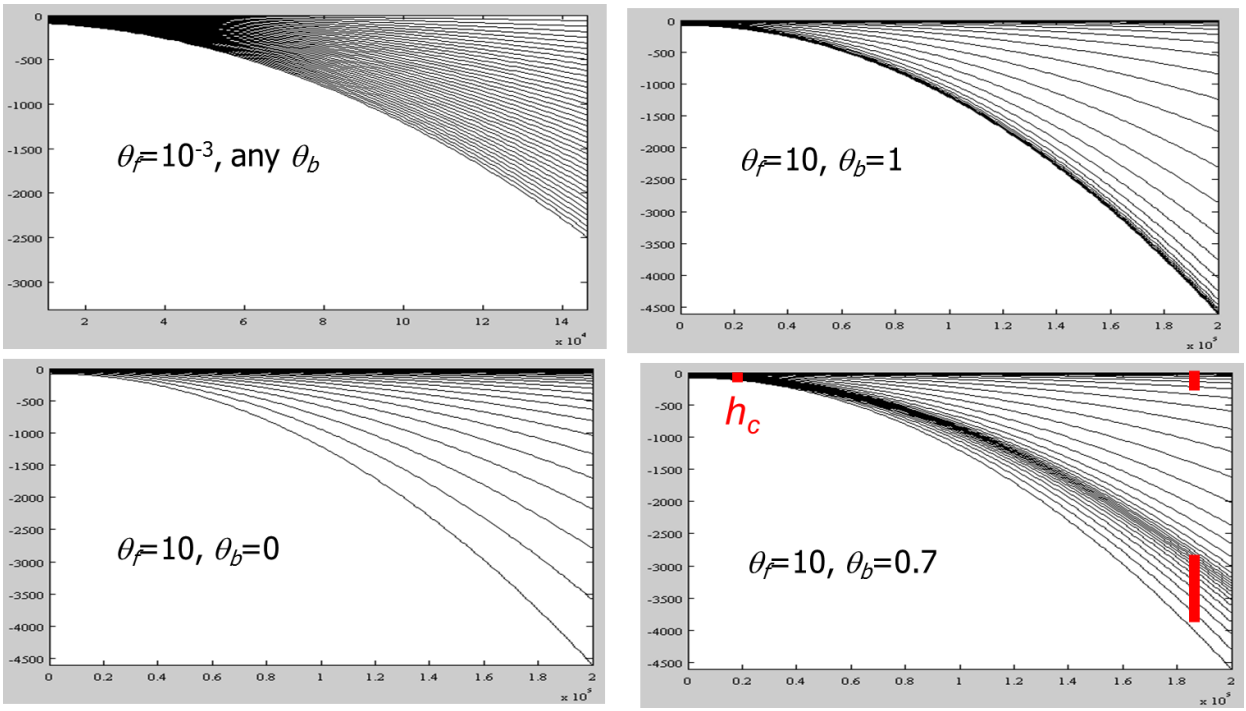


Fig. 3.4: Examples of *S* transformation.

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 (LSC^2). Although there are many ways of generating a LSC^2 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 references 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 LSC^2. As can be seen from Fig. 3.5, a major advantage of LSC^2, 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), LSC^2 is instrumental in reducing unphysical diapycnal mixing and pressure-gradient errors (PGE). A well-designed LSC^2 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 LSC^2 grid, and so beginners may want to start with the *SZ* grid.

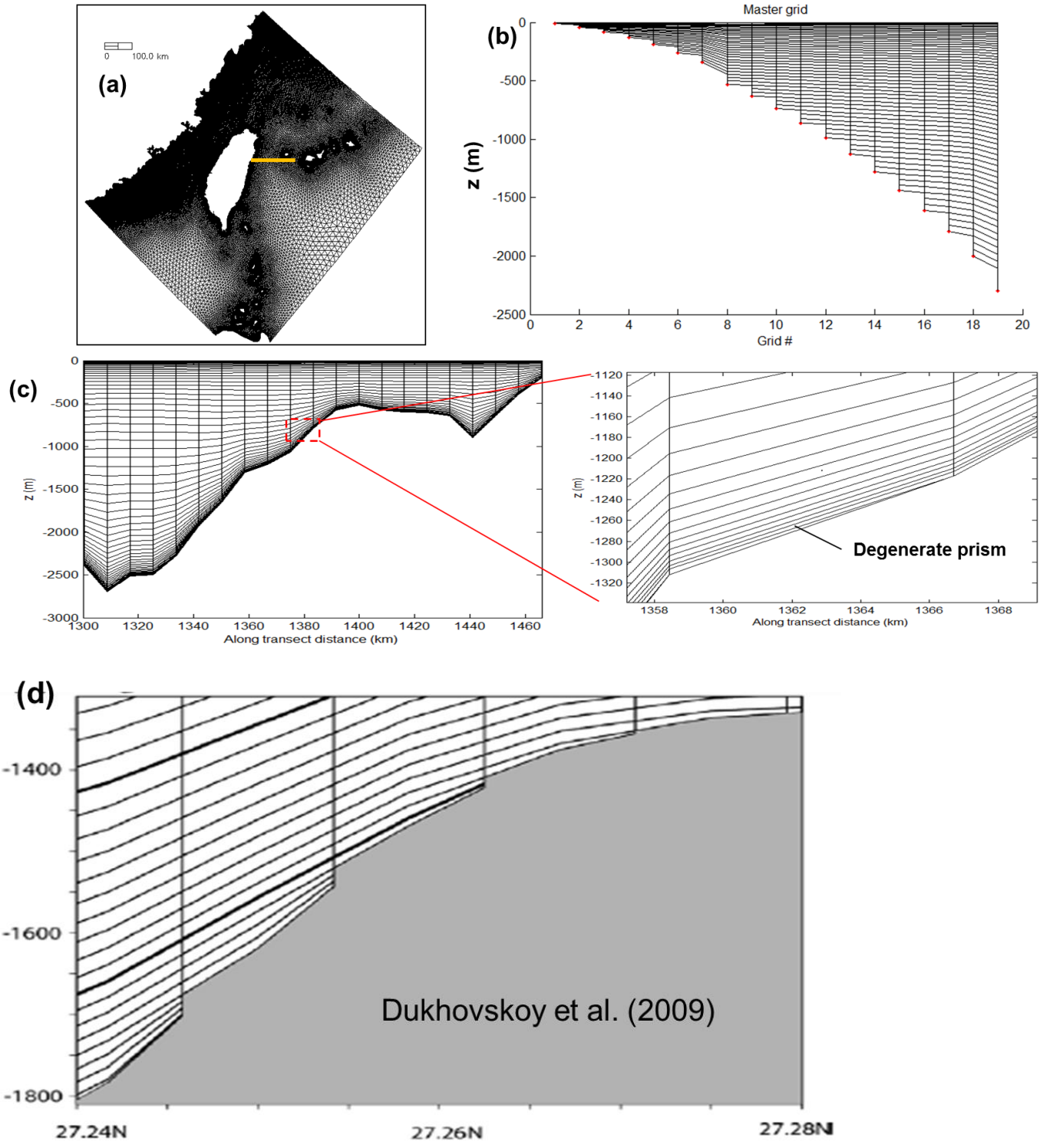


Fig. 3.5: LSC2 grid. (a) Horizontal grid showing the location of the transect (yellow line); (b) master grid at 19 reference depths; (c) corresponding vertical grid of the transect in (a), with degenerate prisms/shaved cells located near the bottom. (d) shows a typical VQS grid that contains staircases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

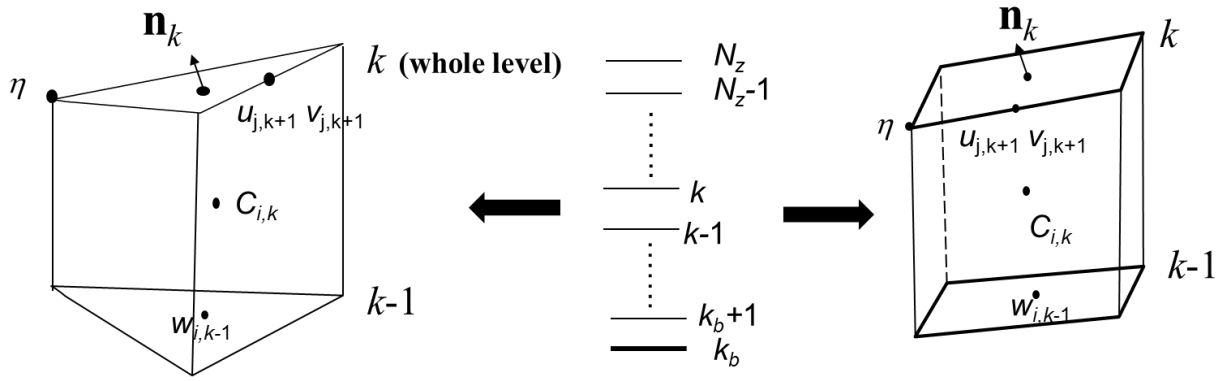


Fig. 3.6: Basic 3D computational unit in SCHISM where variables are staggered.

Polymorphism

The combination of LSC^2 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 LSC^2 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.

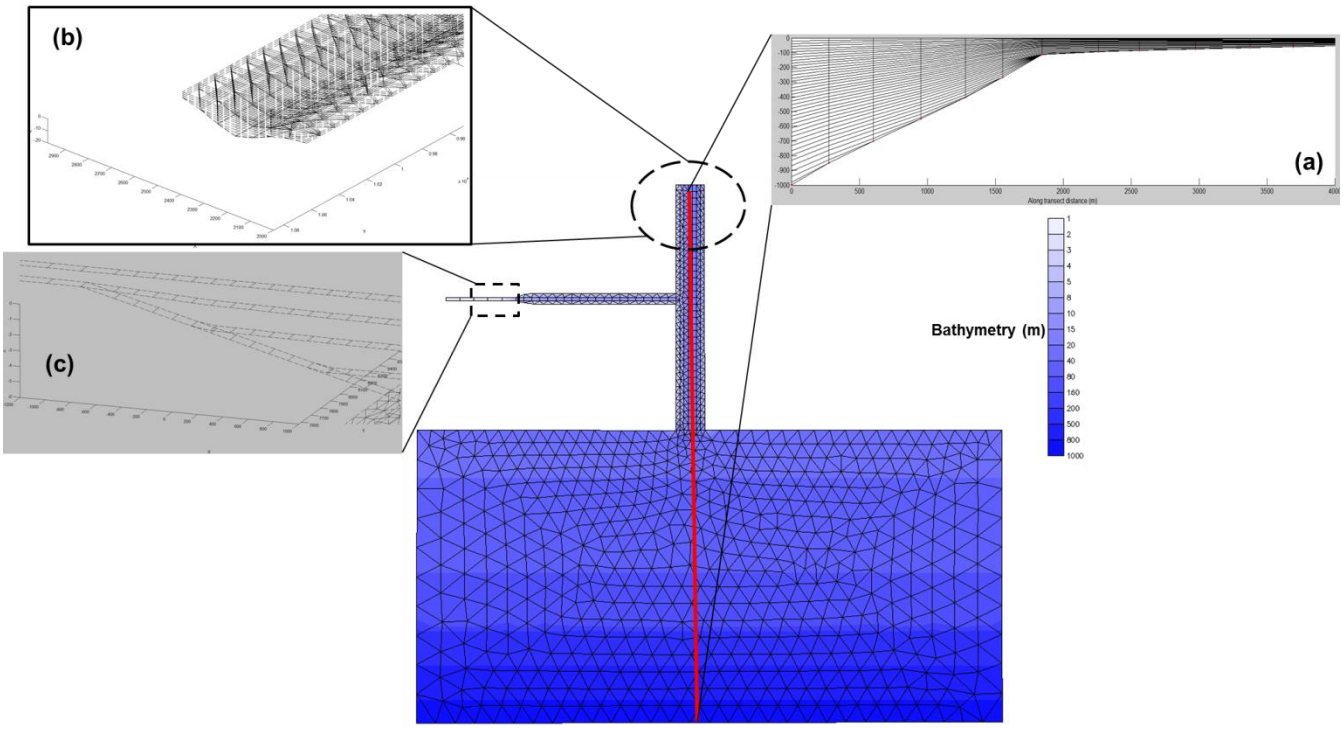


Fig. A3: Model polymorphism illustrated with a toy problem. The mixed triangular-quadrangular grid and the bathymetry are shown in the foreground. The vertical transect grid along the redline going from deep ocean into estuary ('shipping channel') is shown in insert (a). The 3D view of the grid near 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:

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

$$\frac{\boldsymbol{u^{n+1}-u^{*}}}{\Delta t}=\boldsymbol{f}-g\theta\nabla\eta^{n+1}-g(1-\theta)\nabla\eta^{n}+\boldsymbol{m_z^{n+1}}-\alpha|\boldsymbol{u}|\boldsymbol{u^{n+1}}L(x,y,z), \tag{3.4}$$

$$\text{for 3D cells: }\begin{cases}\nu^n\dfrac{\partial\mathbf{u}^{n+1}}{\partial z}=\boldsymbol{\tau}_w^{n+1}, & \text{at } z=\eta^n;\\ \nu^n\dfrac{\partial\mathbf{u}^{n+1}}{\partial z}=\chi^n\mathbf{u}_b^{n+1}, & \text{at } z=-h,\end{cases} \tag{3.5}$$

where superscripts denote the time step, $0\le\theta\le1$ 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:

$$\int_\Omega\phi_i\frac{\eta^{n+1}-\eta^n}{\Delta t}d\Omega+\theta\left[-\int_\Omega\nabla\phi_i\cdot\boldsymbol{U}^{n+1}d\Omega+\int_\Gamma\phi_iU_n^{n+1}d\Gamma\right]+(1-\theta)\left[-\int_\Omega\nabla\phi_i\cdot\boldsymbol{U}^{n}d\Omega+\int_\Gamma\phi_iU_n^{n}d\Gamma\right]=0,\ \ (i=1,\cdots,N_p) \tag{3.6}$$

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 Eq. (3.4) along the vertical dimension results in an equation for $\boldsymbol{U}^{n+1}$, and we discuss three scenarios below.

3.2.1 Locally 2D case

Since there is only one degree of freedom in the vertical dimension in this case, a straightforward integration of Eq. (3.4) gives:

$$\boldsymbol{U}^{n+1}=\breve{\boldsymbol{G}}-g\theta\Delta t\frac{H^2}{\tilde{H}}\nabla\eta^{n+1} \tag{3.7}$$

where $\breve{\boldsymbol{G}}$ incorporates explicit term:

$$\breve{\boldsymbol{G}} = \frac{H}{\widetilde{H}}[\boldsymbol{U}^* + \Delta t(\boldsymbol{F} + \boldsymbol{\tau}_w - g(1-\theta)H\nabla\eta^n)] \tag{3.8}$$

and $\boldsymbol{F}$ is the depth integrated term of $\boldsymbol{f}$, and $\widetilde{H}$ is the depth enhanced by the bottom friction and form drag:

$$\widetilde{H} = H + (\chi + \alpha|\boldsymbol{u}|H)\Delta t \tag{3.9}$$

The net effect of the vegetation is therefore equivalent to increases in the bottom drag, since the 3D structure of the flow/vegetation is not accounted for. Substituting Eq. (3.7) into (3.6) results in an integral equation for the unknown η^{n+1} alone.

3.2.2 Locally 3D case

We first integrate Eq. (3.4) from bottom to surface:

$$\frac{\boldsymbol{U}^{n+1}-\boldsymbol{U}^*}{\Delta t} = \boldsymbol{F} - gH\theta\nabla\eta^{n+1} - gH(1-\theta)\nabla\eta^n + \boldsymbol{\tau}_w - \chi\boldsymbol{u}_b^{n+1} - \alpha\overline{|\boldsymbol{u}|}\boldsymbol{U}^{\boldsymbol{\alpha}} \tag{3.10}$$

where we have performed linearization of the vegetation term:

$$\int_{-h}^{z_v}|\boldsymbol{u}|\boldsymbol{u}dz \cong \overline{|\boldsymbol{u}|}\int_{-\boldsymbol{h}}^{\boldsymbol{z_v}}\boldsymbol{u}dz \equiv \overline{|\boldsymbol{u}|}\boldsymbol{U}^{\boldsymbol{\alpha}} \quad (\boldsymbol{U}^{\boldsymbol{\alpha}} = \int_{-\boldsymbol{h}}^{\boldsymbol{z_v}}\boldsymbol{u}dz) \tag{3.11}$$

$$\overline{|\boldsymbol{u}|} = \frac{\boldsymbol{1}}{H^{\alpha}}\int_{-\boldsymbol{h}}^{\boldsymbol{z_v}}|\boldsymbol{u}^{\boldsymbol{n}}|dz \tag{3.12}$$

and $H^{\alpha} = z_v + h$ is the height of vegetation. Note that similar procedures have been used for other nonlinear terms (e.g. the quadratic bottom drag).

To eliminate $\boldsymbol{u}_b^{n+1}$ in Eq. (3.10), we invoke the discretized momentum equation at the bottom cell and utilize the fact that the Reynolds stress is constant within the boundary layer, as shown in Zhang and Baptista (2008):

$$\frac{\boldsymbol{u_b^{n+1}}-\boldsymbol{u_b^*}}{\Delta t} = \boldsymbol{f_b} - g\theta\nabla\eta^{n+1} - g(1-\theta)\nabla\eta^n - \alpha|\boldsymbol{u_b}|\boldsymbol{u_b^{n+1}} \tag{3.13}$$

from which $\boldsymbol{u_b^{n+1}}$ can be formally solved as:

$$\boldsymbol{u}_b^{n+1} = \frac{1}{1+\alpha|\boldsymbol{u}_b|\Delta t}[\boldsymbol{u}_b^* + \boldsymbol{f_b}\Delta t - g(1-\theta)\Delta t\nabla\eta^n] - \frac{g\theta\Delta t}{1+\alpha|\boldsymbol{u}_b|\Delta t}\nabla\eta^{n+1} \tag{3.14}$$

The subscript 'b' denotes the top of the bottom cell. Note that the main difference from the original formulation of Zhang and Baptista (2008) is the appearance of the vegetation term.

The remaining task is to find $\boldsymbol{U}^{\boldsymbol{\alpha}}$. We'll discuss two scenarios of submerged and emergent vegetation.

3.2.2.1 Emergent vegetation

When the vegetation is *locally* emergent, i.e., $H^{\alpha} \geq H$, we have $\boldsymbol{U}^{\alpha} = \boldsymbol{U}^{\mathrm{n+1}}$, and therefore $\boldsymbol{U}^{\mathrm{n+1}}$ can be found from Eqs. (3.10) and (3.14) as:

$$\boldsymbol{U}^{n+1} = \boldsymbol{G_1} - \frac{g\theta\widehat{H}\Delta t}{1+\alpha|\overline{\boldsymbol{u}}|\Delta t}\nabla\eta^{n+1} \tag{3.15}$$

where $\boldsymbol{G_1}$ contains explicit terms:

$$\boldsymbol{G}_1 = \frac{\boldsymbol{U}^{*}+(\boldsymbol{F}+\boldsymbol{\tau_w})\Delta t - g(1-\theta)\widehat{H}\Delta t\nabla\eta^{n} - \tilde{\chi}\Delta t(\boldsymbol{u}_b^{*}+\boldsymbol{f}_b\Delta t)}{1+\alpha|\overline{\boldsymbol{u}}|\Delta t} \tag{3.16}$$

$$\tilde{\chi} = \frac{\chi}{1+\alpha|\boldsymbol{u_b}|\Delta t} \tag{3.17}$$

and $\widehat{H}$ is a friction modified depth:

$$\widehat{H} = H - \tilde{\chi}\Delta t \tag{3.18}$$

Compared to the original formulation in Zhang and Baptista (2008), the only change in this depth is the vegetation term in $\tilde{\chi}$.

3.2.2.2 Submerged vegetation

When the vegetation is submerged, i.e., $H^{\boldsymbol{\alpha}} < H$, strong shear and turbulence develop between the vegetation and the overlying flow above it (ST94). Nepf and Vivoni (2000) demonstrated that there are two zones for submerged aquatic vegetation (SAV). In the upper canopy (called the 'vertical exchange zone'), mean shear at the top of the canopy produces vertical turbulent exchange with the overlying water, which plays a significant role in the momentum balance. The lower canopy ('longitudinal change zone' as in Nepf and Vivoni, 2000) communicates with surrounding water predominantly through longitudinal advection. The extent of the vertical exchange of momentum between the vegetation zone and overlying water is dependent on the submergence.

We first integrate the momentum equation (3.4) from the bottom to the top of canopy:

$$\boldsymbol{U}^{\alpha} = \boldsymbol{U}^{*\alpha} + \boldsymbol{F}^{\boldsymbol{\alpha}}\Delta t - g\theta H^{\alpha}\Delta t\nabla\eta^{n+1} - g(1-\theta)H^{\alpha}\Delta t\nabla\eta^{n} - \alpha\Delta t|\overline{\boldsymbol{u}}|\boldsymbol{U}^{\alpha} + \Delta t\nu\frac{\partial\boldsymbol{u}}{\partial z}\Big|_{-h}^{z_v} \tag{3.19}$$

where:

$$\boldsymbol{U}^{*\alpha} = \int_{-h}^{z_v}\boldsymbol{u}^{*}dz, \quad \boldsymbol{F}^{\boldsymbol{\alpha}} = \int_{-h}^{z_v}\boldsymbol{f}dz \tag{3.20}$$

The Reynolds stress at the top of canopy can be calculated from theory proposed by ST94. These authors found through lab experiments that the stress variation inside the vegetation layer approximately follows an exponential law:

$$\nu \frac{\partial \boldsymbol{u}}{\partial z} \equiv \overline{u'w'} = \boldsymbol{R}_0 e^{\beta_2 (z - z_v)}, \quad (z \le z_v) \tag{3.21}$$

where $\boldsymbol{R}_0$ is the stress at $z = z_v$, and β_2 is determined by an empirical formula:

$$\beta_2 = \sqrt{\frac{\sqrt{N_v}}{H^\alpha}} \left[-0.32 - 0.85 \log_{10} \left(\frac{H - H^\alpha}{H^\alpha} I \right) \right] \tag{3.22}$$

where I is an energy gradient:

$$I = \frac{\chi |\boldsymbol{u}_b|}{gH} \tag{3.23}$$

which is estimated from the previous time step in the model.

The stress term in Eq. (3.21) therefore becomes:

$$\nu \frac{\partial \boldsymbol{u}}{\partial z}\Big|_{-h}^{z_v} = \beta \chi \boldsymbol{u}_b^{n+1} \tag{3.24}$$

where $\beta = e^{\beta_2 (z_v - z_b)} - 1$, and z_b is the location of the top of the bottom grid cell. Substituting Eqs. (3.24) and (3.14) into (3.19) we can 'solve' for $\boldsymbol{U}^\alpha$ as:

$$\boldsymbol{U}^\alpha = \boldsymbol{G}_3 - \frac{g\theta \widehat{H}^\alpha \Delta t}{1 + \alpha \overline{|\boldsymbol{u}|} \Delta t} \nabla \eta^{n+1} \tag{3.25}$$

$$\boldsymbol{G}_3 = \frac{\boldsymbol{U}^{*\alpha} + \boldsymbol{F}^\alpha \Delta t + \beta \tilde{\chi} \Delta t \left(\boldsymbol{u}_b^* + \boldsymbol{f}_b \Delta t \right) - g(1-\theta) \widehat{H}^\alpha \Delta t \nabla \eta^n}{1 + \alpha \overline{|\boldsymbol{u}|} \Delta t} \tag{3.26}$$

$$\widehat{H}^\alpha = H^\alpha + \beta \tilde{\chi} \Delta t \tag{3.27}$$

Finally, substituting Eqs. (3.25) and (3.14) into (3.10) results in a relationship between $\boldsymbol{U}^{n+1}$ and η^{n+1}:

$$\boldsymbol{U}^{n+1} = \boldsymbol{G}_2 - g\theta \overline{\overline{H}} \Delta t \nabla \eta^{n+1} \tag{3.28}$$

$$\overline{\overline{H}} = H - \tilde{\chi} \Delta t - c \widehat{H}^\alpha \tag{3.29}$$

$$c = \frac{\alpha \overline{|\boldsymbol{u}|} \Delta t}{1 + \alpha \overline{|\boldsymbol{u}|} \Delta t} \tag{3.30}$$

$$\boldsymbol{G_2} = \boldsymbol{U}^* - c\boldsymbol{U}^{*\alpha} + (\boldsymbol{F} + \boldsymbol{\tau}_w)\Delta t - c\boldsymbol{F}^{\boldsymbol{\alpha}}\Delta t - \tilde{\chi}\Delta t(1+\beta c)(\boldsymbol{u}_b^* + \boldsymbol{f}_b\Delta t) - g(1-\theta)\overline{\overline{H}}\Delta t\nabla\eta^n \tag{3.31}$$

3.2.3 General case

In summary, the depth-integrated velocity can be expressed in compact form as:

$$\boldsymbol{U}^{n+1} = \boldsymbol{E} - g\theta\breve{H}\Delta t\nabla\eta^{n+1} \tag{3.32}$$

where

$$\breve{H} = \begin{cases} \frac{H^2}{\widetilde{H}}, & \text{2D} \\ \frac{\hat{H}}{1+\alpha|\overline{\overline{\boldsymbol{u}}}|\Delta t}, & \text{3D emergent} \\ \overline{\overline{H}}, & \text{3D submerged} \end{cases} \tag{3.33}$$

$$\boldsymbol{E} = \begin{cases} \breve{\boldsymbol{G}}, & \text{2D} \\ \boldsymbol{G_1}, & \text{3D emergent} \\ \boldsymbol{G_2}, & \text{3D submerged} \end{cases} \tag{3.34}$$

Substituting Eq. (3.32) back into (3.6) gives an equation for the unknown elevations alone:

$$I_1 = I_4 - \theta\Delta tI_3 - (1-\theta)\Delta tI_5 - \theta\Delta tI_{6,} \quad (i = 1, \cdots, N_p) \tag{3.35}$$

where:

$$I_1 = \int_\Omega \left[\phi_i\eta^{n+1} + g\theta^2\Delta t^2\breve{H}\nabla\phi_i \cdot \nabla\eta^{n+1}\right]d\Omega \tag{3.36}$$

$$I_4 = \int_\Omega \left[\phi_i\eta^n + \theta\Delta t\nabla\phi_i \cdot \boldsymbol{E} + (1-\theta)\Delta t\nabla\phi_i \cdot \boldsymbol{U}^n\right]d\Omega \tag{3.37}$$

$$I_3 = \int_{\Gamma_v} \phi_i\widehat{U}_{\boldsymbol{n}}^{\boldsymbol{n+1}}d\Gamma_v \tag{3.38}$$

$$I_5 = \int_\Gamma \ \phi_i U_{\boldsymbol{n}}^{\boldsymbol{n}}d\Gamma \tag{3.39}$$

$$I_6 = \int_{\overline{\Gamma}_v} \phi_i U_{\boldsymbol{n}}^{\boldsymbol{n+1}}d\overline{\Gamma}_v \tag{3.40}$$

Following standard finite-element procedures, and using appropriate essential and natural boundary conditions, SCHISM solves Eq. (3.35) to determine the elevations at all nodes. Note that the RHS terms $I_{3\text{-}6}$ are known; in the case of I_6, the integrals on $\overline{\Gamma}_v$ need not be evaluated since the essential boundary conditions are imposed by eliminating corresponding rows and columns of the matrix.

The matrix resulting from Eq. (3.35) is sparse and symmetric. It is also positive-definite as long as the depth $\breve{H}$ is non-negative (ihhat=1); numerical experiments indicated that even this restriction can be relaxed for many practical applications that include shallow areas. We show that the addition of vegetation does not introduce additional stability constraint. The effects of the vegetation on $\breve{H}$ are generally similar to the bottom friction. For the 2D case, $\breve{H}$ is always positive. For the 3D emergent case, the vegetation term in the denominator is positive and so does not change the sign. For the 3D submerged case, as $\alpha \to 0$, previous results of Zhang and Baptista (2008) are recovered. As $\alpha \to \infty$ (i.e. very dense vegetation), the friction term (the second term in Eq. (3.29)) dwarfs in comparison with the vegetation term (the third term in Eq. (3.29)), and therefore the friction is negligible under dense vegetation. Since $c \to 1$ as $\alpha \to \infty$, $\breve{H}$ approaches the submergence $H - H^{\alpha}$, which is positive. When the submergence is very small (i.e. almost emergent vegetation), $\breve{H} \to 0$ and the conditioning of the matrix would somewhat deteriorate but the model remains stable. Physically, this means that very strong shear will develop near the canopy.

It's important to notice that the friction-modified depth is different between 2D and 3D cases, which has implications in shallow depths. We show more details of the evaluation of the integrals I_i below. 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 SCHISM's polyphorphism 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). This simple matrix solver is implemented in SCHISM; alternatively, the efficient parallel matrix solver PETSc can be used for large matrices.

In the model, the decision on 2D/3D emergent/3D submerged scenarios is made at each side (where the velocity is defined) based on the total depths from the previous time step and the transition of regimes is handled in the model.

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):

$$I_3 = \sum_j \frac{L_{ij}}{2} \sum_{k=kbs}^{N_z-1} \Delta z_{j,k+1} \frac{\hat{u}_{j,k+1}^{n+1} + \hat{u}_{j,k}^{n+1}}{2} \tag{3.41}$$

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:

$$\hat{U}_n^{n+1} - \bar{U}_n = \sqrt{g/H}\,(\eta^{n+1} - \bar{\eta}), \tag{3.42}$$

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

$$I_3 = \sum_j \frac{L_{ij}(\bar{U}_n)_{ij}}{2} + \frac{L_{ij}\sqrt{g\,H_{ij}}}{6}\left[2(\eta_i^{n+1} - \bar{\eta}_i) + (\eta_j^{n+1} - \bar{\eta}_j)\right] \tag{3.43}$$

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.

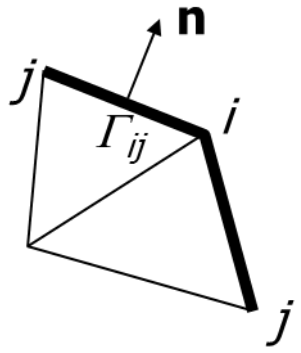


Fig. 3.7: Boundary node *i* with adjacent sides.

Integral I_5

Similar to I_3, we have:

$$I_5 = \sum_j \frac{L_{ij}}{2} \sum_{k=kbs}^{N_z-1} \Delta z_{j,k+1} \frac{u_{j,k+1}^{n} + u_{j,k}^{n}}{2} \tag{3.44}$$

Integral I_1

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

$$I_1 = \sum_{j=1}^{Nb(i)} \sum_{l=1}^{i34(j)} \eta_{j,l}^{n+1} \int_{A_j} \hat{\phi}_{i'} \hat{\phi}_l dA_j + g\theta^2 \Delta t^2 \sum_{j=1}^{Nb(i)} \bar{\bar{H}}_j \sum_{l=1}^{i34(j)} \eta_{j,l}^{n+1} \int_{A_j} \nabla\hat{\phi}_{i'} \cdot \nabla\hat{\phi}_l dA_j \tag{3.45}$$

Where *j* is a neighboring element of *i*, *i'* is the local index of node *i* inside element *j*, overbar in $\overline{\overline{H}}$ 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:

$$\int_{A_j} \hat{\phi}_{i'} \hat{\phi}_l dA_j = \frac{1+\delta_{i',l}}{12} A_j \tag{3.46}$$

$$\int_{A_j} \nabla\hat{\phi}_{i'} \cdot \nabla\hat{\phi}_l dA_j = \frac{\vec{\imath'} \cdot \vec{l}}{4A_j} \tag{3.47}$$

where $\vec{\imath'}$ and $\vec{l}$ are two vectors along side *i'* and *l* respectively, and is $\delta_{i',l}$ the Kronecker delta:

$$\delta_{i',l} = \begin{cases} 1, & i' = l \\ 0, & i' \neq l \end{cases}$$

Case II: Quad

We can analytically evaluate integral Eq. (3.46) as:

$$\int_{A_j} \hat{\phi}_{i'} \hat{\phi}_l dA_j = \frac{A_j}{16}\left(1 + \frac{1}{3}\xi_{i'}\xi_l\right)\left(1 + \frac{1}{3}\varsigma_{i'}\varsigma_l\right) + \frac{B_1}{96}\left(1 + \frac{1}{3}\varsigma_{i'}\varsigma_l\right)(\xi_{i'} + \xi_l) + \frac{B_2}{96}\left(1 + \frac{1}{3}\xi_{i'}\xi_l\right)(\varsigma_{i'} + \varsigma_l) \tag{3.48}$$

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

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

$$B_2 = (\overrightarrow{23} \times \overrightarrow{14})_k \tag{3.49}$$

The other integral Eq. (3.47) cannot be evaluated analytically and so we use the 4-point Gauss quadrature.

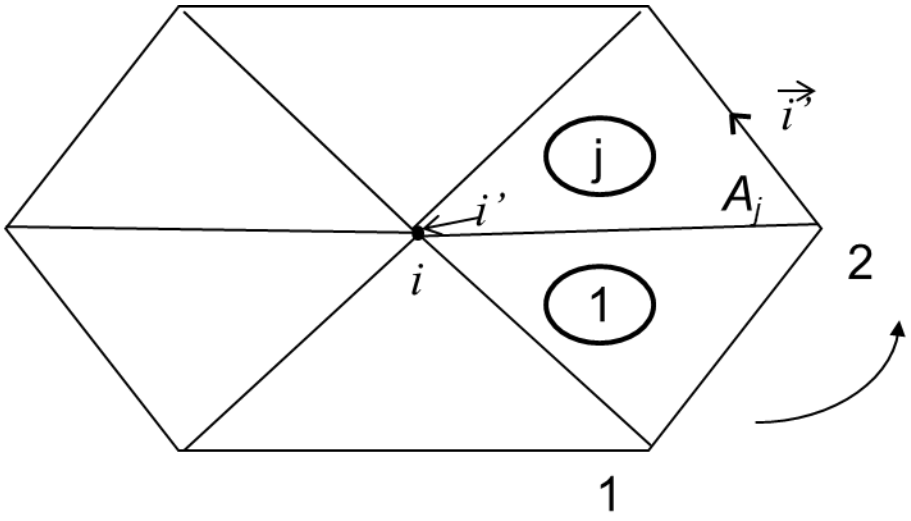


Fig. 3.8: Node ball used in calculating I_1.

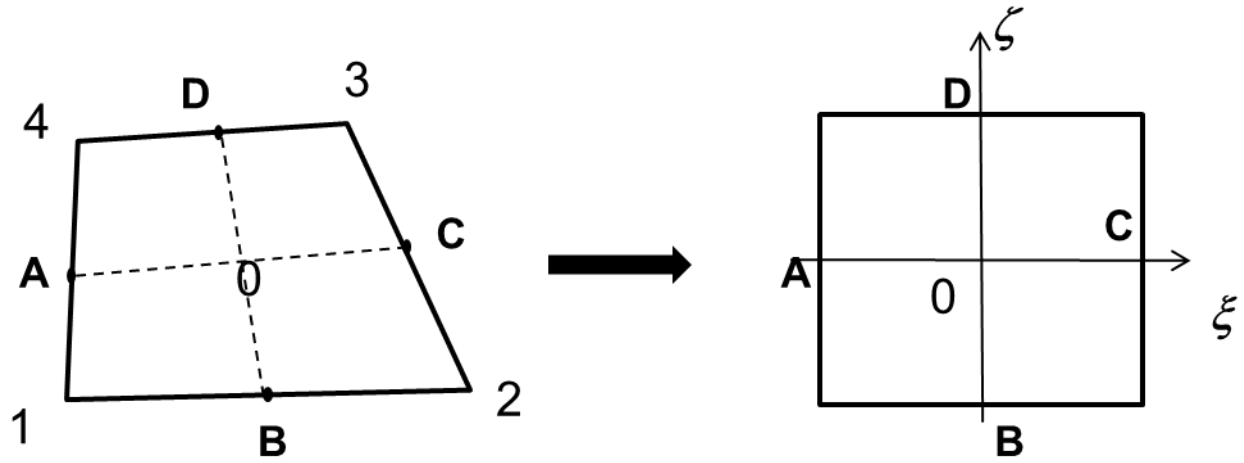


Fig. 3.9: Quad element.

Integral I_4

This integral contains most of the explicit terms. Most terms are straightforward to evaluate, e.g., using element averaging or analytical integration (in the case of volume sources/sinks); 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:

$$\frac{\partial \rho}{\partial x}(x_1 - x_0) + \frac{\partial \rho}{\partial y}(y_1 - y_0) = \rho_1 - \rho_0 \tag{3.50}$$

$$\frac{\partial \rho}{\partial x}(x_2 - x_0) + \frac{\partial \rho}{\partial y}(y_2 - y_0) = \rho_2 - \rho_0 \tag{3.51}$$

$$\frac{\partial \rho}{\partial x}(x_3 - x_0) + \frac{\partial \rho}{\partial y}(y_3 - y_0) = \rho_3 - \rho_0 \tag{3.52}$$

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., (3.50) with (3.51), (3.51) with (3.52), and (3.52) with (3.50). If the 3 centers happen to be co-linear, the equations have no solution and are discarded; however, at least 1 pair has a 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:

$$\frac{\partial \rho}{\partial y}(x_A - x_B) + \frac{\partial \rho}{\partial x}(y_B - y_A) = 0 \tag{3.53}$$

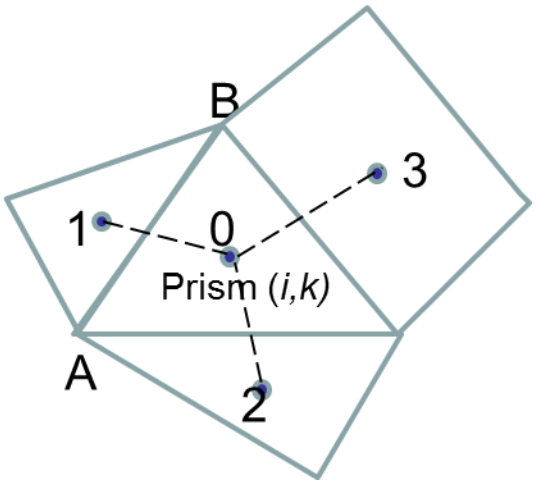


Fig. 3.10: Reconstruction method. *i* is the element index and *k* is the vertical index.

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):

$$\nabla\cdot(\mu\nabla u)\big|_0 = \frac{\mu_0}{\sqrt{3}A_I}(u_1+u_2+u_3+u_4-4u_0) \tag{3.54}$$

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:

$$\begin{aligned} -\lambda\nabla^4 u\big|_0 &= -\lambda\gamma_3(\nabla^2u_1+\nabla^2u_2+\nabla^2u_3+\nabla^2u_3-4\nabla^2u_0) = \\ &\frac{\gamma_2}{\Delta t}\left[7(u_1+u_2+u_3+u_4)-u_{1a}-u_{1b}-u_{2a}-u_{2b}-u_{3a}-u_{3b}-u_{4a}-u_{4b}-20u_0\right] \end{aligned} \tag{3.55}$$

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 $\gamma_2 \le 0.025$ is sufficient to suppress inertial spurious modes.

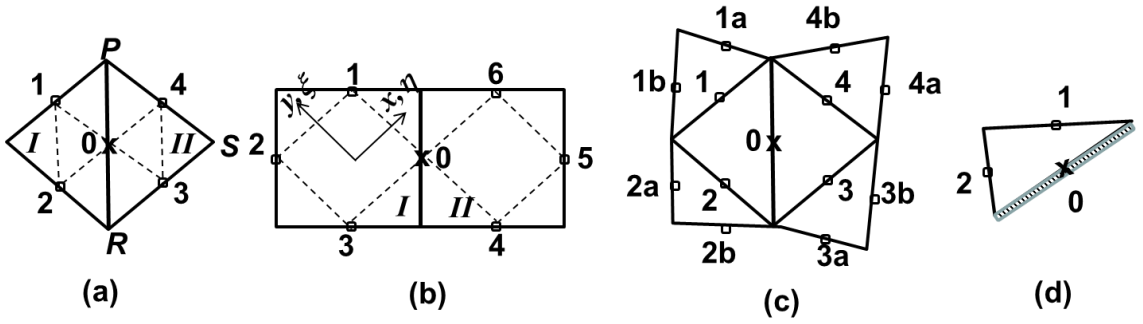


Fig. 3.11: Shapiro filters and viscosity stencil for (a) triangular and (b) quadrangular elements. '*I*' and '*II*' are 2 adjacent elements of side of interest ('0'). The extended stencil used in constructing bi-harmonic viscosity is shown in (c). The special case of a boundary side is shown in (d).

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:

$$\frac{D\mathbf{u}}{D\mathrm{t}} \cong \frac{\mathbf{u}(\mathbf{x},t^{n+1})-\mathbf{u}(\mathbf{x}^*,t^n)}{\Delta t} \tag{3.56}$$

where **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:

$$\frac{D\mathbf{x}}{D\mathrm{t}} = \mathbf{u} \tag{3.57}$$

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 (**x**), which is in our case a side center at whole level where the horizontal velocity **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).

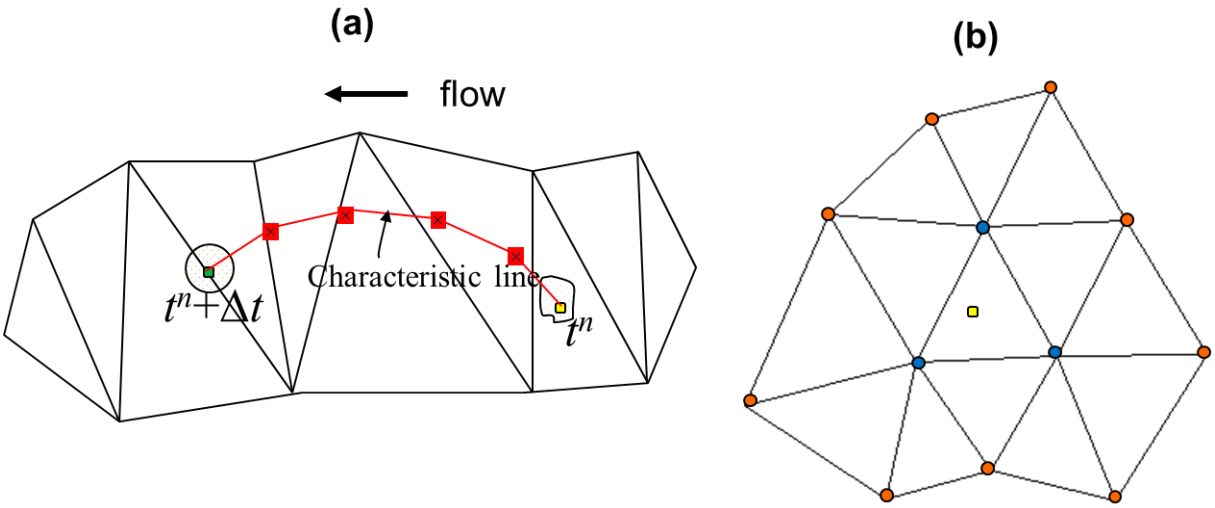


Fig. 3.12: Two steps in Eulerian–Lagrangian method. (a) The characteristic equation is integrated backward in space and time, starting from a side center (the green dot). The characteristic line is further subdivided into smaller intervals (bounded by the red dots), based on local flow gradients, and 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:

$$\tilde{\mathbf{u}}_0 = \mathbf{u}_0 + \frac{\gamma}{4}(\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 + \mathbf{u}_4 - 4\mathbf{u}_0), \tag{3.58}$$

with the strength usually set as γ=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 introduces 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.

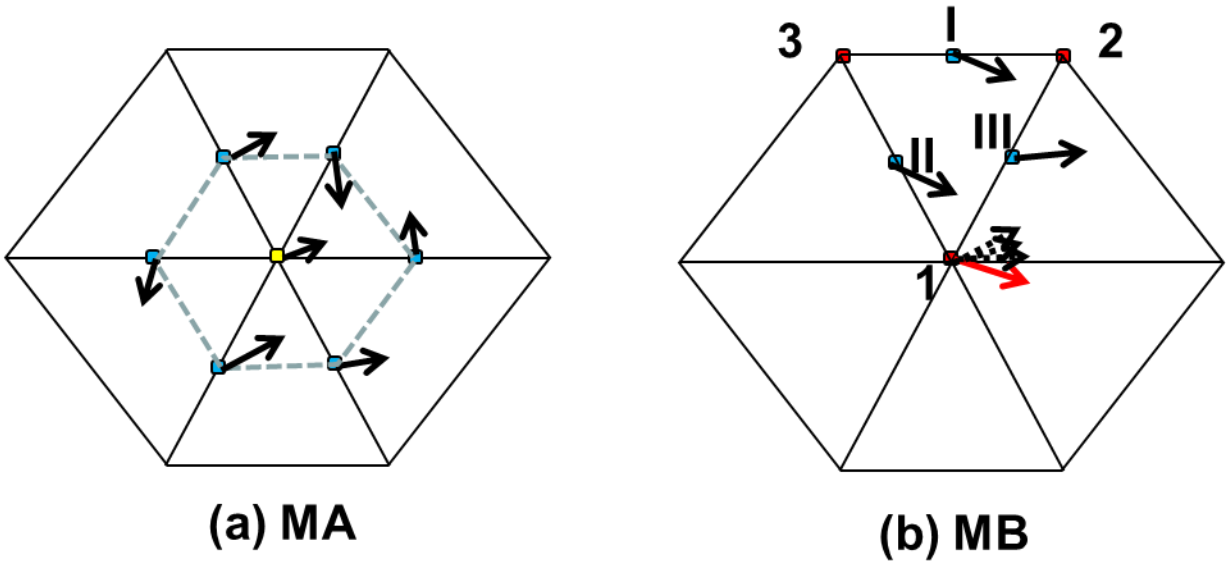


Fig. 3.13: Two methods of converting side velocities to a node velocity. (a) Inverse distance interpolation from sides (blue dots) to node (yellow dot); (b) use of FEM shape function to find the node velocity within each element first (the red arrow), i.e. $u_1 = u_{II} + u_{III} - u_I$, followed by 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):

$$f^h(x,y) = (\alpha_1 + \alpha_2 x + \alpha_3 y) + \sum_{i=1}^{N} \beta_i K(r_i) \qquad (3.59)$$

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):

$$K(r) = -r,\ r^2 \log(r),\ r^3,\ -r^5,\ r^7 \qquad (3.60)$$

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:

$$\begin{cases} f^h(x_i, y_i) = d_i, \;\; 1 \le i \le N \\ \sum_{i=1}^{N} \beta_i = 0 \\ \sum_{i=1}^{N} x_i \beta_i = 0 \\ \sum_{i=1}^{N} y_i \beta_i = 0 \end{cases} \tag{3.61}$$

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. (3.61) 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. (3.59).

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/smoother. 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:

$$\begin{aligned} u_{k,j}^{\max} &= \max_{l=1:i34,k=-1,0} u_{kf+k,im(l,nf)} \\ u_{k,j}^{\min} &= \min_{l=1:i34,k=-1,0} u_{kf+k,im(l,nf)} \end{aligned} \tag{3.62}$$

where im() enumerates all nodes of an element.

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

$$\varepsilon_{k,j}^{(1)} = \max\left[0, u_{k,j}^{n+1,1} - u_{k,j}^{\max}\right] + \min\left[0, u_{k,j}^{n+1,1} - u_{k,j}^{\min}\right] \tag{3.63}$$

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:

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \mu' \Delta t \nabla^2 \varepsilon_{k,j}^{(1)}, \ \forall j,k \tag{3.64}$$

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

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \frac{1}{8}\left[\varepsilon_{k,1}^{(1)} + \varepsilon_{k,2}^{(1)} + \varepsilon_{k,3}^{(1)} + \varepsilon_{k,4}^{(1)} - 4\varepsilon_{k,j}^{(1)}\right] \tag{3.65}$$

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. (3.65) 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. (3.65) is modified in order to maintain the conservation:

$$u_{k,j}^{n+1,2} = u_{k,j}^{n+1,1} + \frac{1}{8}\left[\varepsilon_{k,1}^{(1)} + \varepsilon_{k,2}^{(1)} - 2\varepsilon_{k,j}^{(1)}\right] \tag{3.66}$$

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. For 3D cells, we have:

$$\int_{\delta_b - h}^{\eta} \psi_l \left[\mathbf{u}^{n+1} - \Delta t \frac{\partial}{\partial z}\left(\nu \frac{\partial \mathbf{u}^{n+1}}{\partial z}\right)\right] dz = \int_{\delta_b - h}^{\eta} \mathbf{g}\psi_l dz, \quad (l = kbs + 1, \cdots, N_z) \tag{3.67}$$

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

$$\mathbf{g} = \mathbf{u}^* + \Delta t[\mathbf{f} - g\theta\nabla\eta^{n+1} - g(1-\theta)\nabla\eta^n] \tag{3.68}$$

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:

$$\frac{\Delta z_{l+1}}{6}(2\mathbf{u}_l^{n+1}+\mathbf{u}_{l+1}^{n+1})+\frac{\Delta z_l}{6}(2\mathbf{u}_l^{n+1}+\mathbf{u}_{l-1}^{n+1})-\nu_{l+1/2}\Delta t\frac{\mathbf{u}_{l+1}^{n+1}-\mathbf{u}_l^{n+1}}{\Delta z_{l+1}}+\nu_{l-1/2}\Delta t\frac{\mathbf{u}_l^{n+1}-\mathbf{u}_{l-1}^{n+1}}{\Delta z_l}=$$
$$\frac{\Delta z_{l+1}}{6}(2\mathbf{g}_l+\mathbf{g}_{l+1})+\frac{\Delta z_l}{6}(2\mathbf{g}_l+\mathbf{g}_{l-1}),\ (l=kbs+2,\cdots,N_z-1) \tag{3.69}$$

$$\frac{\Delta z_{l+1}}{6}(2\mathbf{u}_l^{n+1}+\mathbf{u}_{l+1}^{n+1})-\nu_{l+1/2}\Delta t\frac{\mathbf{u}_{l+1}^{n+1}-\mathbf{u}_l^{n+1}}{\Delta z_{l+1}}+\chi\Delta t\mathbf{u}_{kbs+1}^{n+1}=\frac{\Delta z_{l+1}}{6}(2\mathbf{g}_l+\mathbf{g}_{l+1}),\ (l=kbs+1) \tag{3.70}$$

$$\frac{\Delta z_l}{6}(2\mathbf{u}_l^{n+1}+\mathbf{u}_{l-1}^{n+1})+\nu_{l-1/2}\Delta t\frac{\mathbf{u}_l^{n+1}-\mathbf{u}_{l-1}^{n+1}}{\Delta z_l}=\boldsymbol{\tau}_w^{n+1}\Delta t+\frac{\Delta z_l}{6}(2\mathbf{g}_l+\mathbf{g}_{l-1}),\ (l=N_z) \tag{3.71}$$

The bottom velocity is:

$$\mathbf{u}_{kbs}^{n+1}=0, if\ \chi\neq 0, \tag{3.72}$$

$$\mathbf{u}_{kbs}^{n+1}=\mathbf{u}_{kbs+1}^{n+1}, if\ \chi=0 \tag{3.73}$$

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.

If a cell is 2D locally, the velocity is simply solved as:

$$\boldsymbol{u}^{n+1}=\frac{\bar{H}}{H}[\boldsymbol{u}^*+(\boldsymbol{f}+\boldsymbol{\tau}_w/H)\Delta t-g\theta\nabla\eta^{n+1}-g(1-\theta)\nabla\eta^n] \tag{3.74}$$

3.5 Vertical velocity

The vertical velocity serves as a diagnostic variable for local volume conservation[1], 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:

$$\hat{S}_{k+1}(\bar{u}_{k+1}^{n+1}n_{k+1}^x+\bar{v}_{k+1}^{n+1}n_{k+1}^y+w_{i,k+1}^{n+1}n_{k+1}^z)-\hat{S}_k(\bar{u}_k^{n+1}n_k^x+\bar{v}_k^{n+1}n_k^y+w_{i,k}^{n+1}n_k^z)+$$
$$\sum_{m=1}^{3}\hat{P}_{js(i,m)}(\hat{q}_{js(i,m),k}^{n+1}+\hat{q}_{js(i,m),k+1}^{n+1})/2=0,\ \ (k=k^b,...,N_z-1) \tag{3.75}$$

[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.

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 (imm≠0), the bed velocity is prescribed. A compact form for Eq. (3.75) is $\sum_{j\in S^+}|Q_j| = \sum_{j\in S^-}|Q_j|$, where Q_j is the facial fluxes outward of a prism *i*. This conservation will be utilized in the transport equation as the foundation for mass conservation and constancy.

3.6 Turbulence closure

Umlauf and Burchard's GLS model is:

$$\frac{DK}{Dt} = \frac{\partial}{\partial z}\left(\nu_k^{\psi}\frac{\partial K}{\partial z}\right) + \nu M^2 + \mu N^2 - \varepsilon \tag{3.76}$$

$$\frac{D\psi}{Dt} = \frac{\partial}{\partial z}\left(\nu_{\psi}\frac{\partial \psi}{\partial z}\right) + \frac{\psi}{K}\left(c_{\psi 1}\nu M^2 + c_{\psi 3}\mu N^2 - c_{\psi 2}F_w\varepsilon\right) \tag{3.77}$$

with natural B.C.:

$$\begin{cases} \nu_k^{\psi}\frac{\partial K}{\partial z} = 0, \;\; z = -h \; or \; \eta \\ \nu_{\psi}\frac{\partial \psi}{\partial z} = \kappa_0 n \nu_{\psi}\frac{\psi}{l}, z = -h \\ \nu_{\psi}\frac{\partial \psi}{\partial z} = -\kappa_0 n \nu_{\psi}\frac{\psi}{l}, z = \eta \end{cases} \tag{3.78}$$

and essential B.C.:

$$\begin{cases} K = \left(c_{\mu}^0\right)^{-2}\nu\left|\frac{\partial u}{\partial z}\right|, \\ l = \kappa_0\Delta, \qquad\qquad at \;\; z = -h \; or \; \eta \\ \psi = \left(c_{\mu}^0\right)^p K^m(\kappa_0\Delta)^n, \end{cases} \tag{3.79}$$

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

$$M^2 = \left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2 \tag{3.80}$$

$$N^2 = \frac{g}{\rho_0}\frac{\partial \rho}{\partial z} \tag{3.81}$$

$$\varepsilon = \left(c_{\mu}^0\right)^3 K^{1.5} l^{-1} \tag{3.82}$$

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:

$$\mathcal{H}(N_z-m)\left[\frac{\Delta z_{m+1}}{6}(2K_m^{n+1}+K_{m+1}^{n+1}-2K_m^n-K_{m+1}^n)-\Delta t\left(\nu_k^{\psi}\right)_{m+\frac{1}{2}}^n\frac{K_{m+1}^{n+1}-K_m^{n+1}}{\Delta z_{m+1}}\right]+\mathcal{H}(m-$$

$$kbp)\left[\frac{\Delta z_m}{6}(2K_m^{n+1}+K_{m-1}^{n+1}-2K_m^n-K_{m-1}^n)-\Delta t\left(\nu_k^{\psi}\right)_{m-\frac{1}{2}}^n\frac{K_m^{n+1}-K_{m-1}^{n+1}}{\Delta z_m}\right]=\mathcal{H}(N_z-$$

$$m)\Delta t\left[\begin{Bmatrix}\frac{\Delta z_{m+1}}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m+1/2}^n\\ \frac{\Delta z_{m+1}}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m+1/2}^n\frac{2K_m^{n+1}+K_{m+1}^{n+1}}{K_{m+1/2}^n}\end{Bmatrix}-\left(c_{\mu}^0\right)^3(K^{0.5}l^{-1})_{m+\frac{1}{2}}^n\frac{\Delta z_{m+1}}{6}(2K_m^{n+1}+K_{m+1}^{n+1})\right]+$$

$$\mathcal{H}(m-kbp)\Delta t\left[\begin{Bmatrix}\frac{\Delta z_m}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m-1/2}^n\\ \frac{\Delta z_m}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m-1/2}^n\frac{2K_m^{n+1}+K_{m-1}^{n+1}}{K_{m-1/2}^n}\end{Bmatrix}-\left(c_{\mu}^0\right)^3(K^{0.5}l^{-1})_{m-\frac{1}{2}}^n\frac{\Delta z_m}{6}(2K_m^{n+1}+K_{m-1}^{n+1})\right],$$

$$(l=kbp,\cdots,N_z) \tag{3.83}$$

$$\mathcal{H}(N_z-m)\left[\frac{\Delta z_{m+1}}{6}(2\psi_m^{n+1}+\psi_{m+1}^{n+1}-2\psi_m^n-\psi_{m+1}^n)-\Delta t\left(\nu_{\psi}\right)_{m+\frac{1}{2}}^n\frac{\psi_{m+1}^{n+1}-\psi_m^{n+1}}{\Delta z_{m+1}}\right]+\mathcal{H}(m-$$

$$kbp)\left[\frac{\Delta z_{m+1}}{6}(2\psi_m^{n+1}+\psi_{m-1}^{n+1}-2\psi_m^n-\psi_{m-1}^n)-\Delta t\left(\nu_{\psi}\right)_{m-\frac{1}{2}}^n\frac{\psi_m^{n+1}-\psi_{m-1}^{n+1}}{\Delta z_m}\right]+$$

$$\kappa_0 n\Delta t\left[\delta_{m,N_z}\left(\frac{\nu_{\psi}}{l}\right)_{N_z}^n\psi_{N_z}^{n+1}+\delta_{m,kbp}\left(\frac{\nu_{\psi}}{l}\right)_{kbp}^n\psi_{kbp}^{n+1}\right]=\mathcal{H}(N_z-$$

$$m)\Delta t\left[\begin{Bmatrix}\frac{\Delta z_{m+1}}{2}(c_{\psi 1}\nu_tM^2+c_{\psi 3}\nu_t^{\theta}N^2)_{m+1/2}^n(\psi/K)_{m+1/2}^n\\ \frac{\Delta z_{m+1}}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m+1/2}^n\frac{2\psi_m^{n+1}+\psi_{m+1}^{n+1}}{2K_{m+1/2}^n}\end{Bmatrix}-c_{\psi 2}\left(c_{\mu}^0\right)^3(K^{0.5}l^{-1}F_w)_{m+\frac{1}{2}}^n\frac{\Delta z_{m+1}}{6}(2\psi_m^{n+1}+\right.$$

$$\left.\psi_{m+1}^{n+1})\right]+\mathcal{H}(m-kbp)\Delta t\left[\begin{Bmatrix}\frac{\Delta z_m}{2}(c_{\psi 1}\nu_tM^2+c_{\psi 3}\nu_t^{\theta}N^2)_{m-1/2}^n(\psi/K)_{m-1/2}^n\\ \frac{\Delta z_m}{2}(\nu_tM^2+\nu_t^{\theta}N^2)_{m-1/2}^n\frac{2\psi_m^{n+1}+\psi_{m-1}^{n+1}}{2K_{m-1/2}^n}\end{Bmatrix}-\right.$$

$$\left.\left(c_{\mu}^0\right)^3(K^{0.5}l^{-1}F_w)_{m-\frac{1}{2}}^n\frac{\Delta z_m}{6}(2\psi_m^{n+1}+\psi_{m-1}^{n+1})\right],\quad(l=kbp,\cdots,N_z) \tag{3.84}$$

where {} indicates the alternative explicit/implicit schemes mentioned above, and $\mathcal{H}$ is a step function. We have applied the natural B.C. (3.78) in these equations, and after K and ψ are solved, the essential B.C. (3.79) 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:

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

where F_h includes vertical settling term (Section 3.7.4), source/sink and also horizontal viscosity terms. The vertical B.C. is:

$$\kappa\frac{\partial C}{\partial z}=\hat{C},\ \ z=\eta \tag{3.77}$$

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

Note that the 3D continuity equation ensures the constancy condition for the transport equation, i.e. *C*=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:

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

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)), and Q_j is the flux at face j *outward* of the prism. 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. With the upwind scheme, the face concentration is defined as:

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

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. The face concentration take different forms with higher-order schemes.

A note on mass conservation

Eq. (3.79) is the starting point of all FV solvers in SCHISM, from which a conservation statement can be derived. Assuming no zero fluxes at surface, bottom and lateral boundary, summing up over all prisms leads to:

$$\sum_i V_i\,C_i^{m+1}=\sum_i V_i\,C_i^m-\Delta t'\sum_{j\in FS}Q_jC_j \tag{3.80b}$$

where '*FS*' stands for free surface. Note that the volume V_i is evaluated at previous step *m*. The 2nd term in (3.79) cancels out at all faces (or vanish at lateral boundary) except at the free surface.

The 2nd term in (3.80b) represents the contribution from the surface movement and is supposed to account for the movement from *m* to *m*+1. However, this balance is not precise (time truncation error). In the case of TVD[2] or WENO (itr_met>2), there is also additional splitting error. Therefore mass conservation is only good up to time truncation error.

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

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

We have utilized the volume conservation:

$$\sum_{j\in S^+}|Q_j| = \sum_{j\in S^-}|Q_j| \tag{3.82}$$

Therefore the Courant condition is:

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

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:

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

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:

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

And the Courant condition is:

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

where:

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

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

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

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 TVD^2 as much as possible for efficiency/accuracy.

3.7.3 TVD^2

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:

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

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:

$$C\big|_j = C_{jup} + C_{jr}\,. \tag{3.90}$$

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:

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

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

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

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-cylcing 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 TVD^2. The origin of these two limiters is the approximation Eq. (3.90) via a Taylor expansion in both space *and* time (DB07):

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

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 $\tilde{\tilde{C}}$. Customary in a TVD method, we then replace the last 2 terms with limiter functions:

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

and so:

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

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:

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

where r_p and s_q are upwind and downwind ratios respectively:

$$\begin{aligned} r_p &= \frac{\sum_{q\in S_V^-}|Q_q|(\tilde{\tilde{C}}_q - \tilde{\tilde{C}}_i)}{|Q_p|(\tilde{\tilde{C}}_i - \tilde{\tilde{C}}_p)}, p \in S_V^+ \\ s_q &= \frac{(\tilde{\tilde{C}}_i - C_i^{M+1})\sum_{p\in S_V^+}|Q_p|}{|Q_q|(\tilde{\tilde{C}}_q - C_q^{M+1})}, q \in S_V^- \end{aligned} \tag{3.98}$$

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.:

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

$$1 + \frac{\Delta t}{2V_i}\sum_{j\in S_V^+}|Q_j|(\sum_{q\in S_V^-}\frac{\psi_q}{s_q} - \psi_j) \geq \delta > 0 \tag{3.100}$$

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 $\tilde{\tilde{C}}$. 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:

$$1 - \frac{\Delta t}{2V_i} |Q_j| \psi_j \geq \delta > 0, \; j \in S_V^+ \tag{3.101}$$

Therefore we choose the following form for the limiter:

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

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 ψ=0 everywhere, and fast convergence within a few iterations is usually observed.

Simple benchmark tests indicate that TVD[2] 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 Vertical movement

Many tracers have 'behaviors' in the form of vertical migration (upward or downward) in the water column. This is modeled with a 'settling' term:

$$\frac{\partial C}{\partial t} + \nabla \cdot (\mathbf{u}C) = \frac{\partial}{\partial z}\left(\kappa \frac{\partial C}{\partial z}\right) + \frac{\partial (w_s C)}{\partial z}, \tag{3.103}$$

where w_s is the settling velocity (positive *downward*). This term is treated implicitly to avoid stability issues; in particular, it's solved in the 3rd step together with the diffusion term in (3.93). The benefit of this approach is that often the settling term balances the diffusion at boundary (e.g., sediment).

3.7.5 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 (h_0). 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).

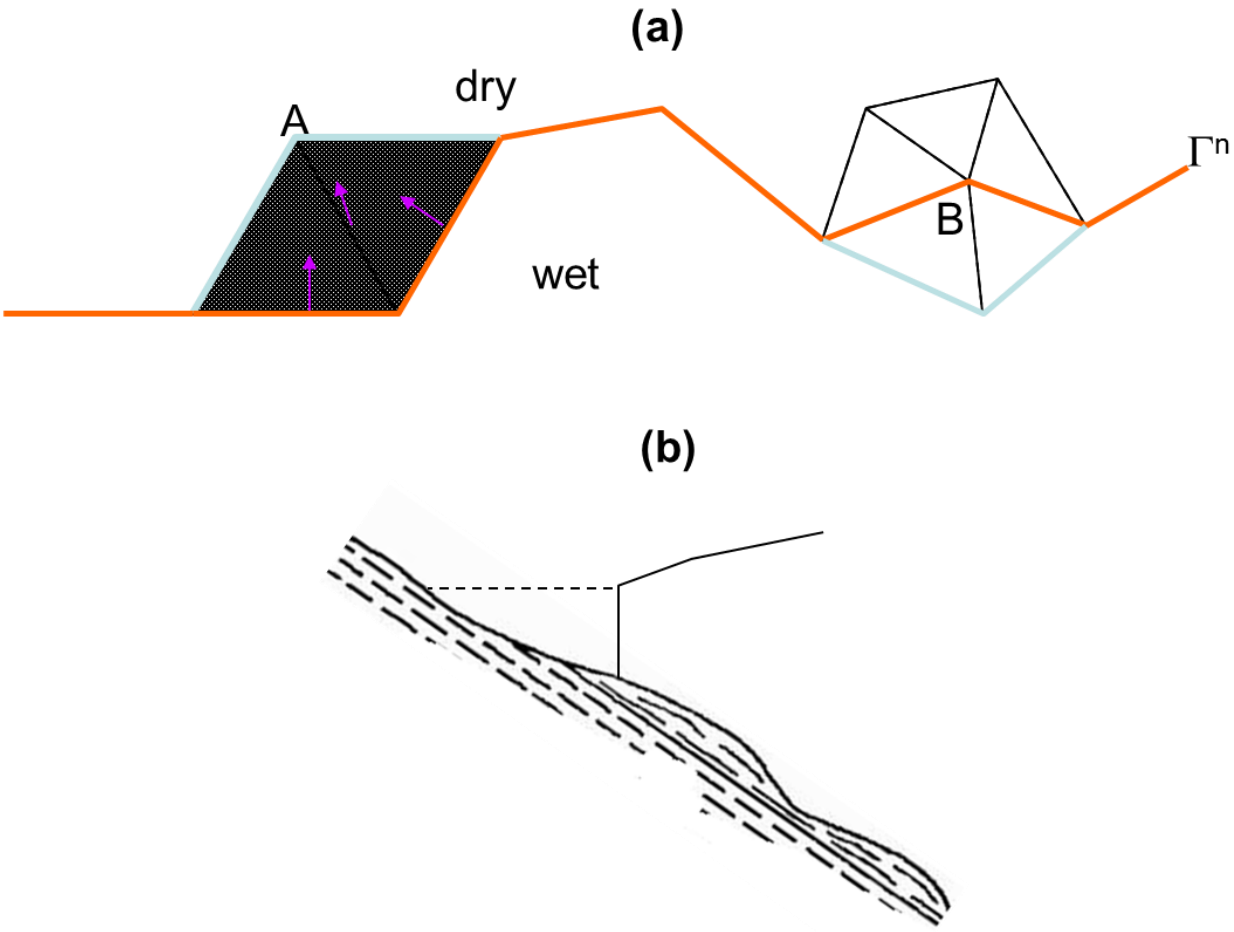


Fig. 3.14: Shoreline tracking algorithm. (a) The orange line is the shoreline from step *n*, and the light blue line is the updated portion since A is wetted and B becomes dry at step *n*+1. (b) At the end of the tracking, constant extrapolation of elevation is used at the new shoreline position, which 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:

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

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

  $$\boldsymbol{\lambda}^0 = -\sin\lambda \mathbf{i} + \cos\lambda \mathbf{j} \tag{3.104}$$

  $$\boldsymbol{\phi}^0 = -\cos\lambda \sin\phi\, \mathbf{i} - \sin\lambda \sin\phi\, \mathbf{j} + \cos\phi\, \mathbf{k} \tag{3.105}$$

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

where **i,j,k** 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:

- (xnd,ynd,znd), (xcj,ycj,zcj),(xctr,yctr,zctr) are *global* coordinates of node, side center and element centroid for ics=1 or 2. If ics=1 (Cartesian), znd=zcj=zctr= 0;
- (znl, zs, ze) are local z-coordinates measured vertically upward from the undisturbed surface (i.e. local frame when ics=2), at node, side, and element;
- eframe(1:3,1:3,1:nea) is the tensor for the element frame (w.r.t. global frame). The 2nd index indicates axid id (1=x; 2=y; 3=z) and the 1st index indicates tensor components. *sframe* and *pframe* are similar.

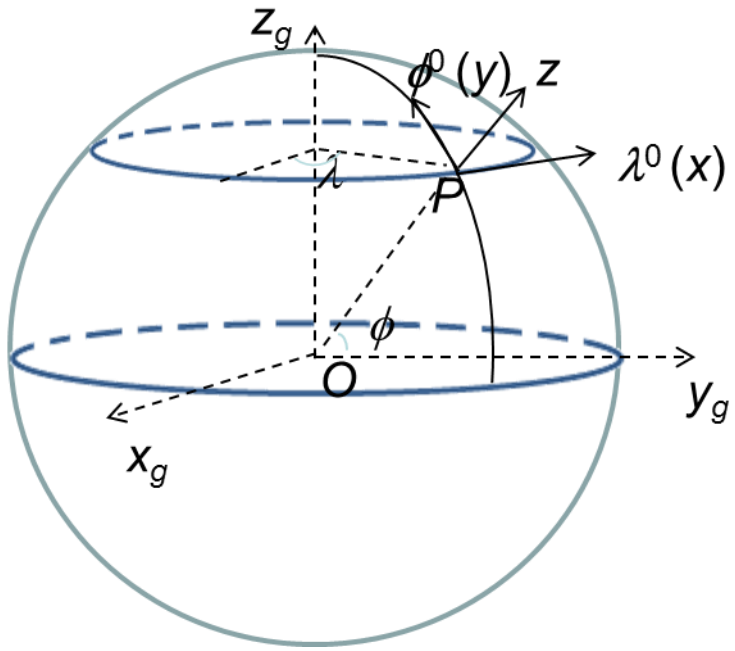


Fig. 3.15: Coordinate frames on a sphere.

Chapter 4 SCHISM I/O

Most SCHISM inputs can be visualized with ACE/xmgredit5 or xmgr5 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 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. *.nc: netcdf4 inputs, including time history (*.th.nc), hotstart (hotstart.nc), and nudging inputs (*_nu.nc);
6. *.nml: main parameter input (param.nml);
7. *.in: role-specific input files with individual formats. ASCII inputs include vertical grid (vgrid.in), B.C. input (bctides.in), and hydraulics.in (for hydraulics module) etc;
8. 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;
9. 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.nml)
4. B.C. input (bctides.in)
5. Bottom friction input (drag.gr3, or rough.gr3 or manning.gr3)

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

4.2.1 hgrid.gr3

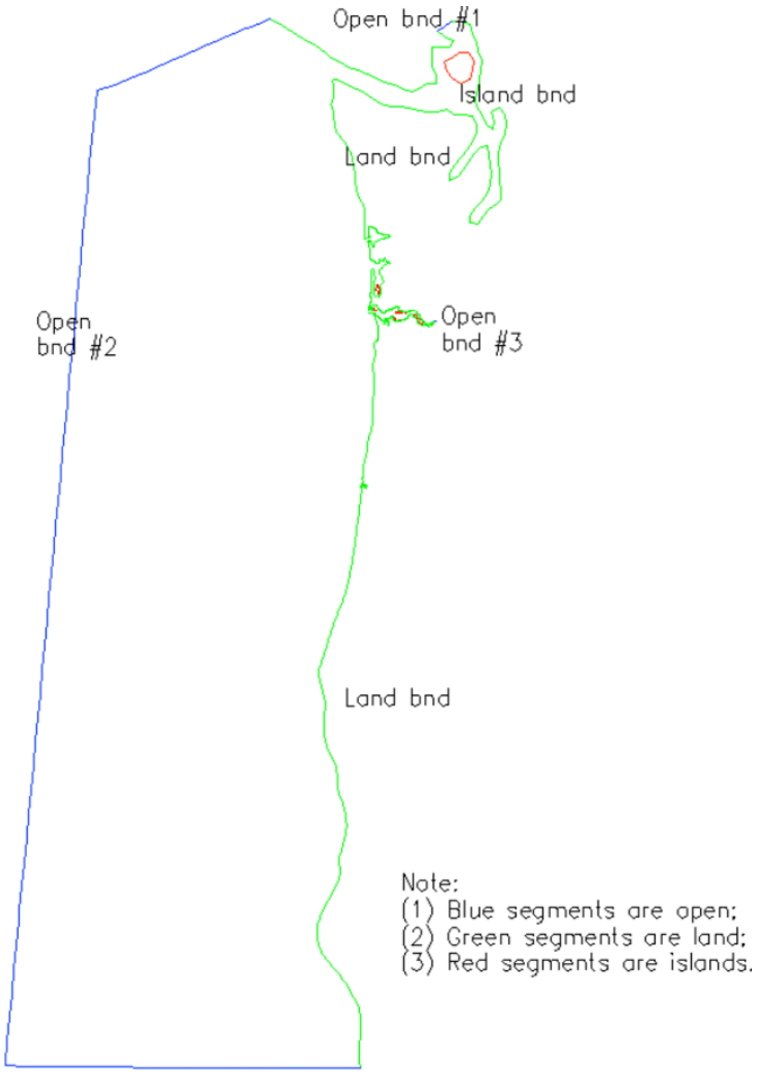


Fig. 4.1: Horizontal grid and boundary segments.

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 land 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;

(5) If WWM is used, the land boundary flags (cf. red texts above) are required, and also there must not be any open boundary segments on any island. Since WWM can only handle triangles, the mixed grid needs to be converted to a pure triangular grid for WWM using a pre-processing script.

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: LSC^2; 2: SZ)**

54 18 100. **!nvrt(=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 -h_s**

S levels **!S-levels below**

30. 0.7 10. **! constants used in S-transformation: h_c, θ_b, θ_f**

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 nvrt) 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 LSC^2 grid

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

1 **!ivcor (1: LSC^2; 2: SZ)**

39 **!nvrt(=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. 'N/A' denotes 'not available'.

Lateral b.c. and nudging — bctides.in — param.in

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

<you notes> **!Not used in code; write your own comments**

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), tamp(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.nc (netcdf);

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

time history of elevation is read in from elev2D.th.nc, 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

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) == 4 or -4) !3D input

time history of velocity (not discharge!) is read in from uv3D.th.nc (netcdf)

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 '3'

time history of velocity (not discharge!) is read in from uv3D.th.nc (netcdf) 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:N_z,*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.nc (netcdf)**

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.

(A note on the AGE module: the number of tracers inside this module (ntracer_age) must be an even number, and usually you only specify the first ntracer_age/2 tracers on some open boundaries. For example, suppose ntracer_age=4, you can set the B.C. flags as:

3 !nope

88 3 0 0 0 0 !ocean – no age tracer b.c. here

....

5 0 1 1 3 2 !Columbia River

1. !relax for T

1. !relax for S

1. 0. 0. 0. !inject age tracer #1 here

1. !relax for AGE

3 0 2 3 3 2 !Fraser River

1. !relax for T

1. !relax for S

0. 1. 0. 0. !inject age tracer #2 here

1. !relax for AGE

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.nml

Please refer to sample param.nml in the source code directory (sample_inputs/) while you read the following.

The file uses the FORTRAN namelist format. The order of input parameters is not important. Governing rules for this file are:

- lines beginning with "!" are comments; blank lines are ignored;
- the format for each parameter is: 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 (use ' ' (single quotes) for this); 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;
- if multiple entries for a parameter are found, the last one wins - please avoid this
- array inputs follow column major (like FORTRAN) and can spill to multiple line.

The namelist file is divided into 3 major sections: CORE, OPT and SCHOUT. CORE lists out all core parameters that *must* be specified by the user, i.e., no defaults are provided by the code. OPT and SCHOUT sections contain optional parameters and I/O flags, all of which have default values so the user does not have to specify any of these (the values shown in the sample are defaults unless otherwise stated). SCHISM will also echo the input values in the output file param_out.nml.

Most parameters (and their keywords) are explained as follows; some 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.nml). 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.nml. The parameters are listed out below in alphabetic order.

4.2.4.1 CORE

The following parameters have to be specified by the user; otherwise you'll get a fatal error.

1. ipre (int)

Pre-processing flag is very useful for checking integrity of the horizontal grid and some inputs. ipre=1: code will output centers.bp, sidecenters.bp, (centers build point, sidcenters build point), and mirror.out and stop. Check errors in fatal.error. IMPORTANT: ipre=1 only works for single CPU! ipre=0: normal run. Pre-processing flag (1: on; 0: off).

2. ibc (int), ibtp (int)

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

3. rnday (double)

Total simulation time in days.

4. dt (double)

Time step in seconds.

5. msc2 (int), mdc2 (int)

These two parameters are only used if the wave module WWM is invoked (USE_WWM is on, i.e. icou_elfe_wwm=1). The values represent the spectral resolution used in WWM and must match those in wwminput.nml;

6. 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.

7. nspool, ihfskip (int)

These two flags control the global netcdf outputs. Output is done every nspool steps, and a new output stack is opened every ihfskip steps.

4.2.4.2 OPT

The optional parameters below are explained in alphabetical order. The default values can be seen below and also in the sample file (sample_inputs/).

- dtb_min=10, dtb_max=30 (double)

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

- flag_ic(:)=1 (int array)

Options for specifying initial tracer field for cold start, where each array entry corresponds to individual tracer model (e.g. TEM, SAL, SED etc). If flag_ic=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 flag_ic=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.

- h0=0.01 (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.

- h[1,2]_bcc=50,100 (double)

Option on how the baroclinic gradient is calculated below bottom. The 'below-bottom' gradient is zeroed out if h>=h2_bcc (i.e. like Z) or uses constant extrapolation (i.e. like terrain-following) if h<=h1_bcc(<h2_bcc). A linear transition is used if the local depth h1_bcc<h<h2_bcc.

- ibcc_mean=0 (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=0 (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=0, iwbl=0 (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.

iwbl=1: modified Grant-Madsen formulation for wave enhanced boundary layer; =2: Soulsby (1997) formulation; =0: off.

If icou_elfe_wwm=1, additional parameters are:

nstep_wwm=1 (int): call WWM every this many time steps;

hmin_radstress=1.0 (double): min. total water depth used only in radiation stress calculation; the radiation stress is zero if local depth <hmin_radstress.

- ics=1 (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).

- i_hmin_airsea_ex, hmin_airsea_ex

Option to locally turn off heat/salt exchange.

i_hmin_airsea_ex=1: exchange turned off if local grid depth<hmin_airsea_ex

i_hmin_airsea_ex=2: exchange turned off if local water depth<hmin_airsea_ex

- ielm_transport = 0, max_subcyc = 10 (int)

Hybrid ELM-FV transport for performance; used only with itr_met>=3. If ielm_transport=1, the hybrid scheme is invoked and max_subcyc represents the max # of subcycling per time step in transport allowed; if the actual # of subcycling in a prism at a time step exceeds this threshold, more efficient ELM transport is used locally (at the expense of mass conservation, so make sure this option is used sparingly).

- ieos_type=0, ieos_pres=0 (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.

- iloadtide=0 (int)

Option to specify Self Attraction and Loading (SAL) tide, usually used for basin-scale applications. If iloadtide=0, SAL is off. If iloadtide=1, the SAL input is interpolated values from FES2014, given in loadtide_[FREQ].gr3, where [FREQ] are frequency names (shared with tidal potential, in upper cases like M2) and the two 'depths' inside are amplitude (m) and phases (degrees behind GMT). If iloadtide=2, a simple scaling is used to reduce the gravity. If iloadtide=3, the scaling is dependent on the local depth *a la* Stepanov & Hughes (2004).

- if_source=0 (int)

Point sources/sinks option (0: no; 1: on). If if_source=1, needs source_sink.in, vsource.th, vsink.th, and msource.th (see sample files in the source code directory src/ for their formats). If if_source=-1, the input is source.nc, which includes element list inside and allows for different time steps and # of records for volume/mass source/sinks.

- iflux=0 (int)

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

- iharind=0 (int)

Harmonic analysis flag. If iharind≠0, an input harm.in is needed.

- ihconsv=0, isconsv=0 (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=0 (int)

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

- ihot=0 (int)

Hot start flag. If ihot=0, cold start; if ihot≠0, hot start from hotstart.nc. 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.nc, and in this case, you do not need to adjust other inputs.

- ihydraulics=0 (int)

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

- imm=0, ibdef=10 (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=0 (int), ihorcon=0 (int), hvis_coef0=0.025 (double), ishapiro=1, niter_shap=1 (int), shapiro0=0.5 (double)

These parameters (and inter_mom below) 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 generally require a velocity B.C.

Due to spurious modes or dispersion (oscillation), viscosity/filter should 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. In addition to the viscosity, one can add the

Shapiro filter, which is specified by ishapiro =0,±1, 2 (turn off/on Shapiro filter). If ishapiro=1, shapiro0 specifies the Shapiro filter strength. If ishapiro=-1, an input called shapiro.gr3 is required which specifies the filter strength at each node. If ishapiro=2, a Smagorinsky-like filter is applied and shapiro0 is a coefficient (γ_0), which is on the order of 10^3:

$$\gamma = 0.5\tanh(\gamma_0 \Delta t \hat{\mu}) \tag{4.1}$$

$$\hat{\mu} = \sqrt{u_x^2 + v_y^2 + (u_y + v_x)^2/2}$$

If ishapiro/=0, niter_shap specifies the number of times the filter is applied. Note that for non-eddying regime applications (nearshore, estuary, river), an easiest option is: indvel=0, ishapiro=1 (shapiro0=0.5), ihorcon= inter_mom=0.

For applications that include the eddying regime, grid resolution in the eddying regime needs to vary smoothly (Zhang et al. 2016), and the user needs to tweak dissipation carefully. A starting point can be: indvel=ishapiro=inter_mom=0, ihorcon=2, hvis_coef0=0.025. If the amount of dissipation is insufficient in the non-eddying regime, consider using ishapiro=-1, with an appropriate shapiro.gr3 to turn on Shapiro filter locally to add dissipation, or use ishapiro=2 and shapiro0=1000.

- inter_mom=0, kr_co=1 (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=0, inu_uv=0 (int)

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

$$\tilde{\varphi} = (1-\gamma)\varphi + \gamma\varphi_{target} \tag{4.2}$$

which is a discrete analogue of the restoration equation:

$$\frac{\partial \varphi}{\partial t} = \frac{\gamma}{\Delta t}\left(\varphi_{target} - \varphi\right) \tag{4.3}$$

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_tr(:)=0 (int array), step_nu_tr=86400. (double)

Nudging flag for tracer models (e.g. temperature), and nudging step (in sec). When inu_tr=0, no nudging is done.

When inu_tr=1, relax back to initial conditions.

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

If inu_tr≠0, the horizontal relaxation factors are specified in [MOD]_nudge.gr3 (as depths info), and the vertical relaxation factors are specified as a linear function of depths with: vnh[1,2] (transitional depths) and vnf[1,2] (relaxation constants at the 2 depths). The final relaxation constant is the sum (horizontal+vertical relaxation factors) times dt.

- inunfl=0 (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.

- isav=0 (int)

Parameters for submerged or emergent vegetation. If isav=1 (module on), you need to supply 4 extra inputs: sav_cd.gr3 (form drag coefficient), sav_D.gr3 (depth is stem diameter in meters); sav_N.gr3 (depth is # of stems per m^2); and sav_h.gr3 (height of canopy in meters).

- itr_met=3 (int), h_tvd=5. (double)

Transport option for all tracers. itr_met=3 for TVD^2, and itr_met=4 for 3^{rd} order WENO. **Note that itr_met=1,2 (explicit) is only kept for comparison and benchmark purpose and so should NOT be used normally- use '3' or '4' instead**. If itr_met>=3, 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=0 (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, vdmin_pp1, tdmin_pp1, h2_pp, vdmax_pp2, vdmin_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: mid, stab, which specify the closure scheme and stability function used: mid= 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; 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.

- level_age(:)=-999 (int array)

If USE_AGE is on, this array specifies the vertical level indices used to inject age tracers. Use -999 to inject the tracer at all levels.

- meth_sink=0 (int)

Option for sinks. If meth_sink =1, the sink value is reset to 0 if an element is dry with a net sink value locally to prevent further drawdown of groundwater.

- nadv=1 (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.

- nchi=0 (int)

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

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

If nchi=1, one additional parameter is required: dzb_min (in meters). In this case the drag coefficients are calculated using the log drag law when the bottom cell thickness δ_b>=dzb_min. when δ_b<dzb_min, Cd=Cdmax, where Cdmax=Cd(δ_b=dzb_min). This is to avoid exaggeration of Cd in very shallow water.

- ncor=0 (int)

Coriolis option. If ncor=0 or -1, a constant Coriolis parameter is specified. If ncor=0, coricoef specifies the Coriolis factor. If ncor=-1, rlatitude 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 sfea0.

- nramp=1 (int), dramp=1. (double), nrampbc=0 (int), drampbc=1. (double)

Ramp options for the tides, B.C. or baroclincity, and ramp-up periods in days (not used if the corresponding ramp flag is 0). If ibc=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:

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

- nws=0, iwind_form=-1 (int), wtiminc=dt (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.

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=1 or 4, or nws=2 && ihconsv=0, or nws=2 && iwind_form=1, the stress is calculated from Hwang (2018) 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=10. (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.

- s1_mxbnt=0.5, s2_mxnbnt=3.5 (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.

- slam0=-124, sfea0=45 (double)

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

- start_year=2000, start_month=1, start_day=1 (int), start_hour=0, utc_start=8 (double)

Starting time for simulation. ‘utc_start’ is hours *behind* the GMT, and is used to adjust time zone. For example, utc_start=5 is US Eastern Time, and utc_start= –8 is Beijing Time.

- rho0=1000, shw=4184. (double)

Reference water density for Boussinesq approximation and specific heat of water in J/kg/K.

- thetai=0.6 (double)

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

4.2.4.3 SCHOUT

- iout_sta=0, nspool_sta=10 (int)

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

- nc_out =1(int)

Main switch to turn on/off netcdf outputs, useful for other programs to control outputs.

- nhot=0, nhot_write=8640 (int)

Hot start output control parameters. If nhot=0, no hot start output is generated. If nhot=1, hot start output is named outputs/hotstart_[process_id]_[time_step].nc every nhot_write steps, where it is the corresponding time iteration number. nhot_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.nc using combine_hotstart7.f90 (./combine_hotstart7 –h for help).

- iof_* (int)

Global output (in netcdf4 format) options, where * stands for module name (e.g. hydro, wwm etc). The frequency of global outputs is controlled by 2 parameters in CORE: 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 outputs/schout_[MPI process id]_[1,2,3,...].nc etc. The combine scripts are then used to gather each output variable across all MPI processes into a single output, e.g., schout_[1,2,3…].nc.

Each output variable is controlled by 1 flag in param.nml. We only show a few examples below; the rest are similar. Note that variables may be centered at nodes/sides/elements horizontally and whole/half levels vertically. However, at the moment most variables are centered at nodes and whole levels, and most post-processing FORTRAN scripts can only handle this type of outputs.

iof_hydro(1) = 1 **!global elevation output control. If** iof_hydro(1)**=0, no global elevation is recorded. If** iof_hydro(1)**= 1, global elevation for each node of a sub-domain is recorded in schout*.nc. 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. iof_sed(1) = 1 won't output the bottom depth change unless you turn on USE_SED in makefile.

Some 'native' variables (e.g., element- or side-centered) are:

iof_hydro(26) = 1 **!horizontal velocity defined at side [m/s]. These are the original velocity inside SCHISM**

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 nchi=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 m^2/s for diffmin and 1 m^2/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 2^{nd}-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 ≥dt.

In the case of wind.th (with nws=1 or 4), the time step specified inside must also match wtiminc in param.nml. 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. If you do not have good values for some tracers, use -9999. instead and the code will inject ambient concentration values for you.

4.3.3 .th.nc (netcdf4)

These include elev2D.th.nc, uv3D.th.nc, TEM_3D.th.nc, SAL_3D.th.nc, and [MOD]_3D.th.nc (where MOD is the tracer module name like 'COS'). The format can be found below (also in test suite (e.g. schism_verification_tests/Test_ICM_UB/)).

Note: (1) the time must start from 0, and the time step can be anything ≥dt.

(2) 'nOpenBndNodes' should be total number of nodes on all open boundary segments that require this input, and the values appear in same order as in bctides.in inside 'time_series'.

Sample elev2D.th.nc:

```
netcdf elev2D.th {

dimensions:

    time = UNLIMITED ; // (73 currently)
```

nOpenBndNodes = 748 ;

nLevels = 1 ;

nComponents = 1 ;

one = 1 ;

variables:

double time_series(time, nOpenBndNodes, nLevels, nComponents) ;

float time_step(one) ;

double time(time) ;

data:

time_series =…

Sample uv3D.th.nc:

netcdf uv3D.th {

dimensions:

nOpenBndNodes = 748 ;

one = 1 ;

time = UNLIMITED ; // (73 currently)

nLevels = 44 ;

nComponents = 2 ;

variables:

float time_step(one) ;

double time(time) ;

float time_series(time, nOpenBndNodes, nLevels, nComponents) ;

data:

time_step = 86400 ;

….

Sample TEM_3D.th.nc:

netcdf TEM_3D.th {

dimensions:

nOpenBndNodes = 30 ;

nLevels = 35 ;

nComponents = 1 ;

one = 1 ;

time = UNLIMITED ; // (371 currently)

variables:

float time_step(one) ;

time_step:long_name = "time step in seconds" ;

double time(time) ;

time:long_name = "simulation time in seconds" ;

float time_series(time, nOpenBndNodes, nLevels, nComponents) ;

data:

….

4.3.4 .prop

The element-centered inputs color each element with a 'property', which can be visualized with xmgredit5 (Special→Properties). The format follows xmgredit5 element property format:

1 -1 **!element number, element property**

2 0

3 2

4 5

....

Currently there are 2 .prop files:

tvd.prop

If TVD transport is used (itr_met>=2), user can explicitly specify horizontal regions where upwind or TVD/TVD^2 is used, based on the element property values (0: upwind; 1: TVD/TVD^2) in tvd.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 [m^3/s] from region '1' to '0', flow from region '2' to '1',…**

0.002778 0.8852E-02 0.2285E-01

…

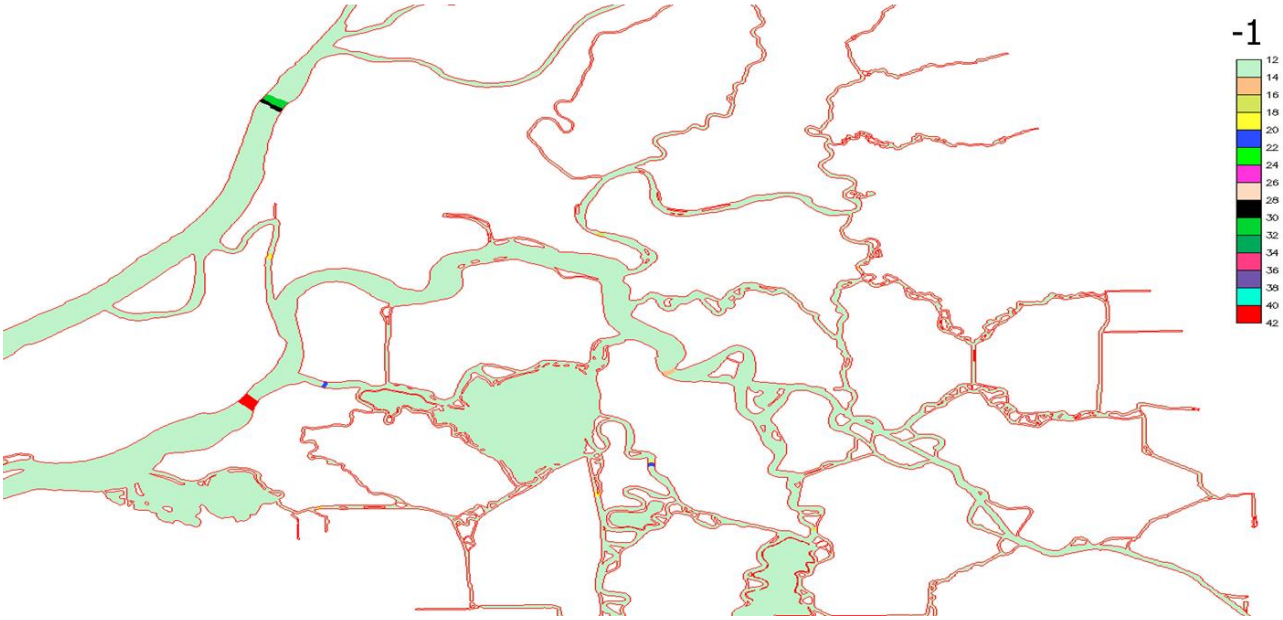


Fig. 4.2: Example of fluxflag.prop.

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.nc.

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 .nc

Beside the time series inputs, we have the other types of netcdf4 input files as follows. Sample files can be found in the test suite (e.g. schism_verification_tests/Test_ICM_UB/).

4.3.6.1 hotstart.nc

This input basically contains all major state variables defined at node/side/element.

*4.3.6.2 *_nu.nc*

This input is used for tracer nudging (inu_tr=2). You only need to specify values in the nudging zone and may use junk values -9999 inside (in this case the code will not nudge to the junk value). The mapping array 'map_to_global_node' is used to map the array indices to the global node indices.

4.3.7 .in

4.3.7.1 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.

4.3.6.2 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**

4.3.6.3 hydraulics.in

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

4.3.6.4 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.**

4.3.8 sflux/

This dir is required if nws=2. In this case, atmoserpic 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 asci file is a namelist; the command below shows its content:

$cat sflux_inputs.txt

&sflux_inputs ! file name

/

Notes: all parameters inside this input are optional. Advanced users may consult the source code for a complete list of parameters.

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

Note that 4 is only required if the heat exchange module is invoked via ihconsv=1, and 3 is only required if the salt exchange module is invoked via isconsv=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_1.0003.nc might be blended with a file called sflux_air_2.0003.nc. The ".0003" 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 (0001, 0002, 0003) produced every 12 hours in a forecast cycle.

Interpolation and prioritization

Using air as an example, it is assumed that the file sflux_air_2.0001.nc is more resolved or accurate than sflux_air_1.0001.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 ".0001". 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 sptatial 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_1.0001.nc --> narr_air.2004_06_10.nc

sflux_air_1.0002.nc --> narr_air.2004_06_11.nc

...

sflux_air_1.0011.nc --> narr_air.2004_06_20.nc

sflux_air_1.0012.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); we only use 1 set in this example.

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.

Since the time and space interpolation will be used to interpolate sflux info onto hgrid.ll at runtime, you need to make sure that the lon/lat grid in sflux_*_1.* covers hgrid.ll, and the union of time records in sflux*.nc covers the entire simulation period. The time zone info is given by utc_start, and you may pre-pend some records if this value is negative (eastern hemisphere) or append if it is positive. Even if utc_start=0, the code will need at least one time record beyond rnday for interpolation; simply duplicate the record at rnday if you do not have such info.

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. This is the time origin used in each file and the 'time' is then the offset (**in days**) from this origin;
- 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<=i<=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 time origin) 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

All SCHISM outputs can be found in outputs/.

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 =    3.000000000000000
```

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 ; average |eta|= 4.553235604713400E-005**

done solving momentum eq...

done solving w

done solving transport equation

done recomputing levels...

done density calculation...

TIME STEP= 1; TIME= 300.000000

….

The '**average |eta|**' highlighted above can be used as a quick and easy way to check if the run is progressing smoothly; it is the average of the absolute value of surface elevation at all nodes. If it's too large or NaN, you have a problem.

4.4.2 Global output

SCHISM netcdf4 output is emitted in a directory called outputs/. This directory must exist or you will get an immediate crash from the model.

An example file name is outputs/schout_0000_2.nc. More generally, the file name is: schout_[processor_no]_[time_block /stack #].nc

Processor number

The mpi_processor number starts at 0 and represents the MPI processor ID from the task that wrote the output.

Time block

The time blocks (‘stack’) 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-day long.

- "Neat" time lengths that will make meaningful analysis (e.g. daily, 10 days etc) are usually easiest later when you post-process;
- Some of the 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.
- 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 is very small in size. 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 empty block were not written out).

Variable names

The variables inside .nc correspond to 3D grid and state variables info at each output time step. The state variables (arrays) may have different centerings, e.g. at node/element/side. At the moment, most variables are centered around nodes (and most post-processing FORTRAN scripts also work on node-centered variables).

Combine outputs

The per-processor outputs need to be gathered into combined nc4 outputs first before you can visualize or post-process them. The script that does this is called *combine_output11.f90* (a simple perl script *autocombine_MPI_elfe.pl* exists to combine all available outputs transparently; you just need to update the path to the compiled *combine_output11* inside the script, and it can be launched before or after a run is done). See the header of combine_output11.f90 on sample compilation commands and usage. Once you are done combining, you should have nc4 files called something like schout_2.nc etc. Note that the stack # remains but the MPI process number is gone. There is no utility for gathering the outputs in stack/time; instead most post-processing tools are able to work with multiple stacks.

Note that SCHISM allows users to easily add more customized outputs, using the routine writeout_nc() inside schism_step. The combine scripts will automatically combine the additional outputs.

Visualization of outputs

You can visualize the combined nc4 outputs using VisIT (with SCHISM plug-ins). More info can be found in Section 5.3.

Other global outputs

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 (nonfatal_*) contains non-fatal warnings, while fatal message file (fatal.error) 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.

One important point to remember is that grid generation for SCHISM always starts from raw high-resolution DEMs; do NOT use computational grids from another model as DEM.

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

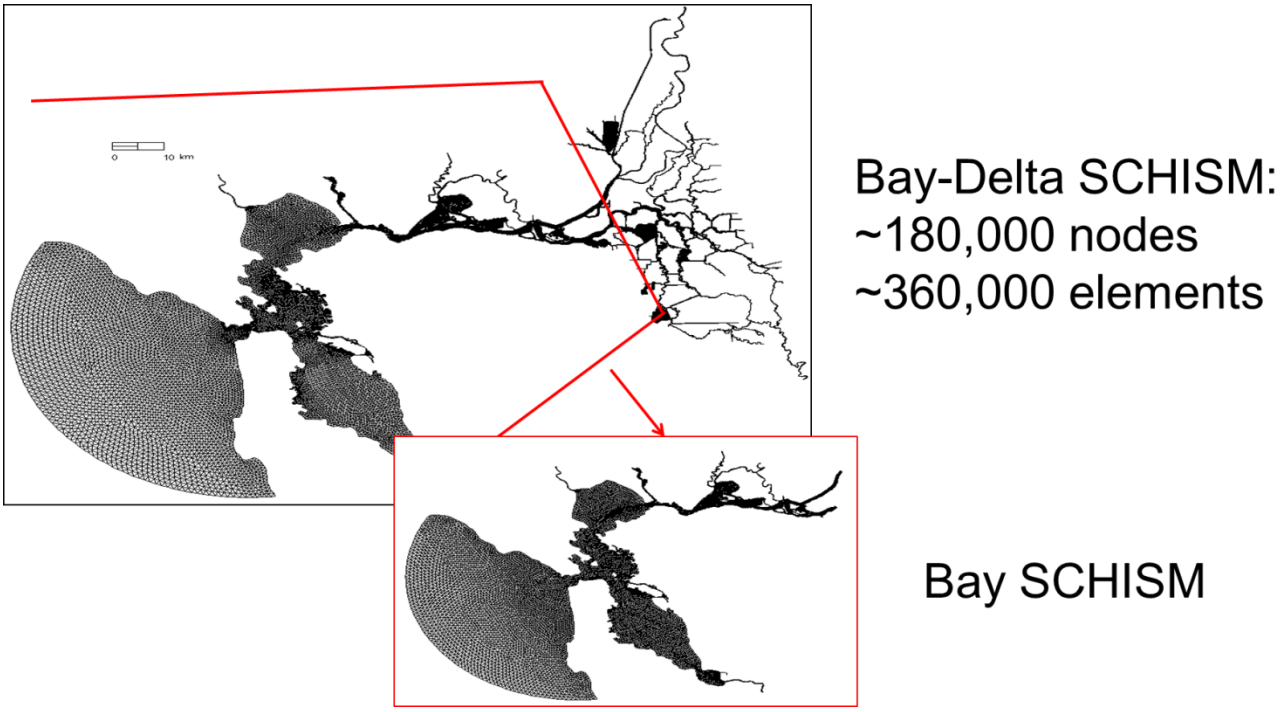


Fig. 5.1: San Francisco Bay & Delta grid (c/o CAL-Department of Water Resource (DWR)).

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 **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 $h<0.1$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.

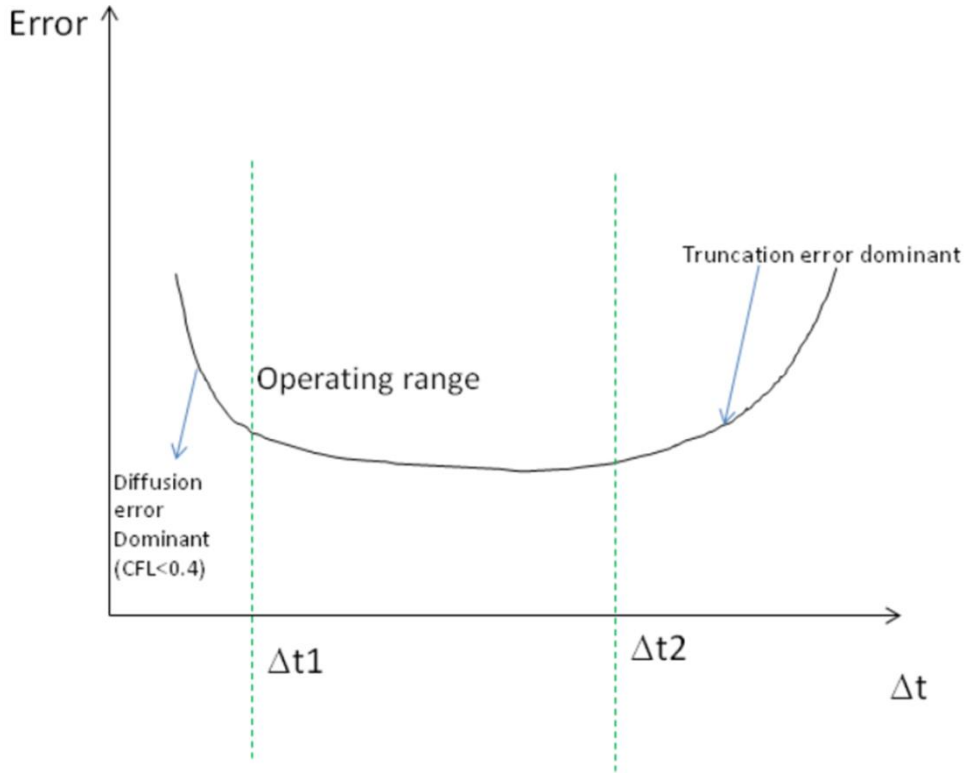


Fig. 5.2: Operational range for time step *for a given grid size*.

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 Δt=100s). 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 Δx, $\Delta t \rightarrow 0$ and $\Delta x/\Delta t$=constant.

Table 5.1: Coarsest grid resolution at sample depths, assuming a 'worse case' scenario of Δt=100s.

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 **u**, and a different form is used for CFL in the shallow h<0.1m, we should impose a min depth of 0.1m (where $\sqrt{gh}$=1m/s)). You can do this by:
    - Edit→Edit over grid/regions→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 (~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 superior stability means the model is very forgiving in grid quality; skewness of triangles (defined as the ratio between the max. side and the equivalent radius) >15 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. You can check quad quality in xmgredit5→Edit→Edit over grid/region→Quality check for quadrangles, and input 0.5 (which is the ratio between the min and max interior angles) for cutoff and 'Accept'. All violating elements will be highlighted.

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 serves 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 LSC^2, there is no need for bathymetry smoothing or other manipulation (and implicitness allows high resolution on steep slope and skew elements).

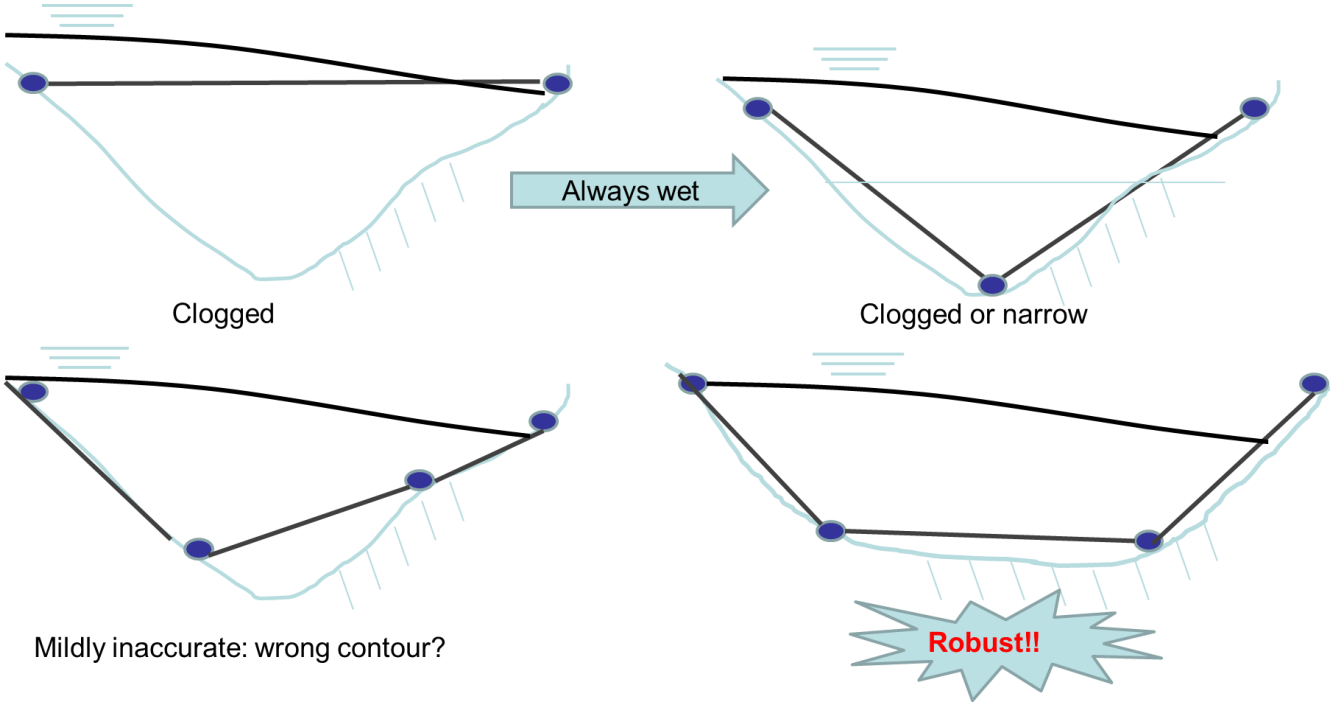


Fig. 5.3: Side view of channel representation. Remember SCHISM does not allow partial wet/dry.

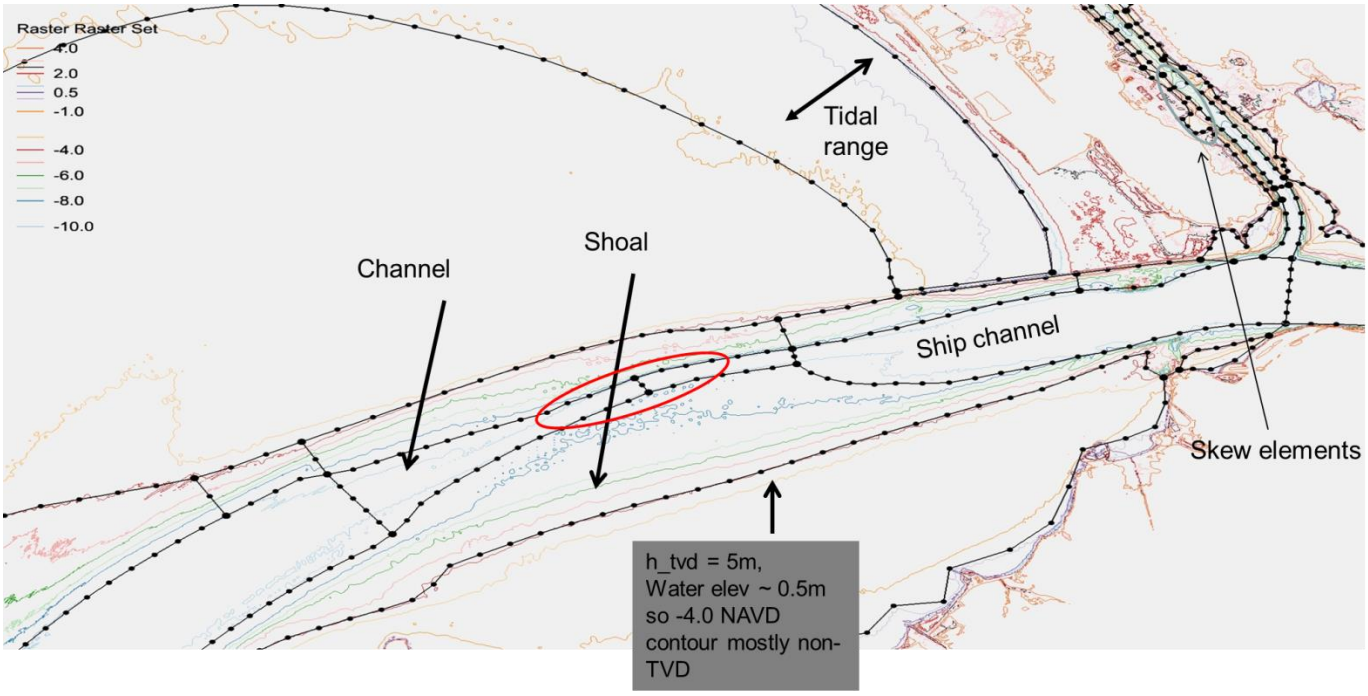


Fig. A5: SMS map in a stretch of San Francisco Bay.

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.

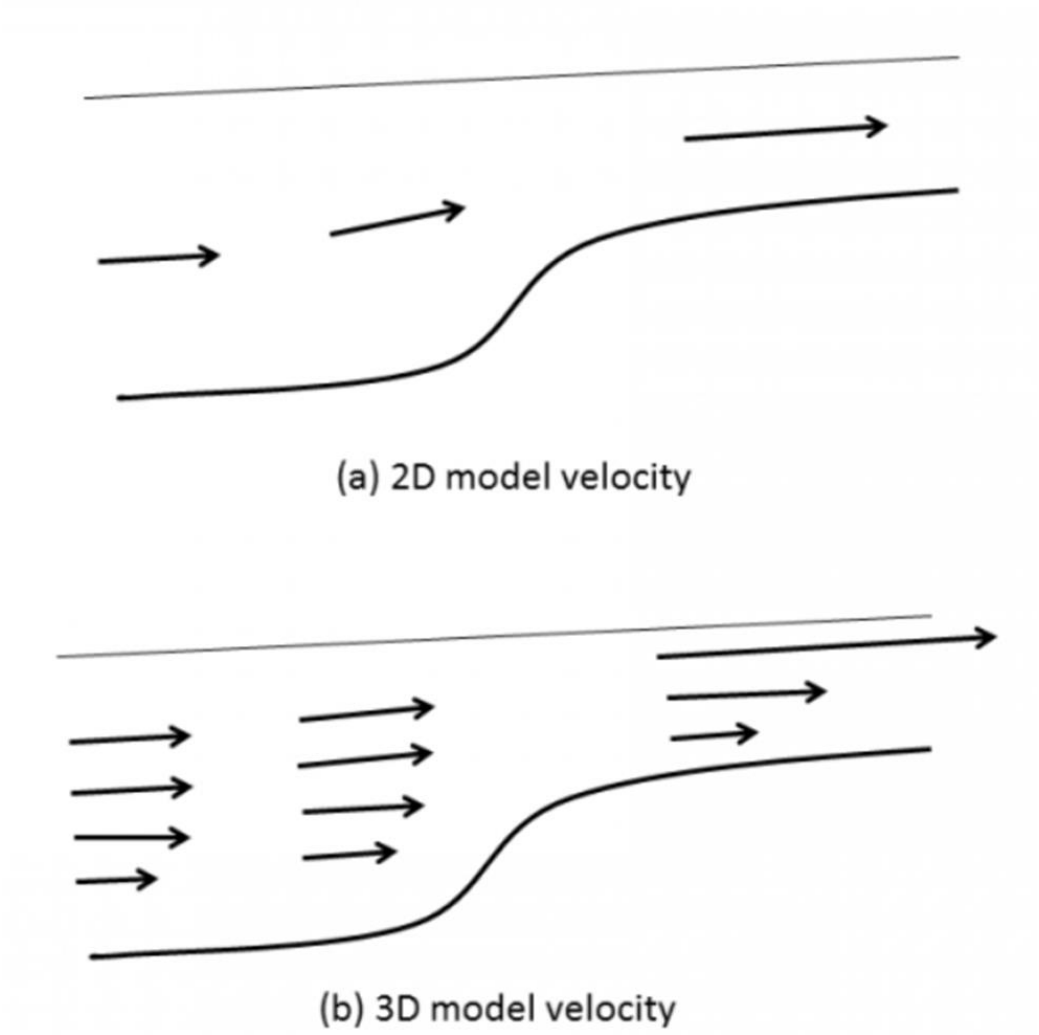


Fig. 5.4: Side view of channel flow for (a) 2D and (b) 3D cases. Volume conservation dictates larger velocities in shallow areas. Note that strong shear is possible in 3D model.

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.

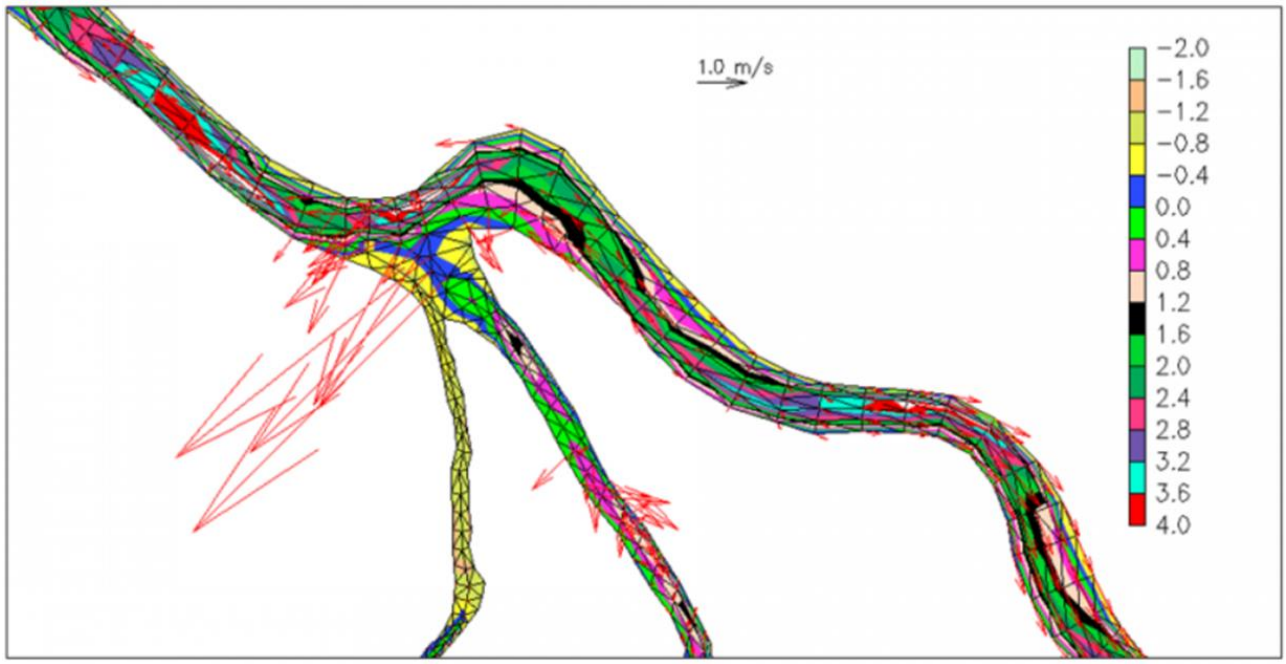


Fig.5.5: Noisy velocity field in shallow areas in SCHISM 3D.

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. 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 an *entire* open boundary segment to become dry at ANY time (while drying at individual nodes is fine). 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. Note that wet/dry is allowed to occur at land/island boundaries.

If you care about wetting and drying near the open boundary location, one option is to relocate 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. Another robust option is to use point sources (if_source=1): in this case no open boundary is required there so wet/dry can happen without crashing the code.

DWR tools

In the git 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, not for stability but to avoid distortion of eddies (Zhang et al. 2016). 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.

5.1.10 Periodic boundary condition

Implementing this type of B.C. in SCHISM introduces some challenges in the barotropic solver because it'd destroy the symmetry of the matrix and cause blowup if the conditioning is sufficiently bad. A workaround is to 'drape' the hgrid onto a sphere to avoid the corresponding open boundary segments altogether. A simple script to guide this process can be found in Utility/Pre-Processing/periodic_grid.f90.

5.2 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.

- LSC^2 scripts: gen_vqs.f90 and plot_VQS.m

The FORTRAN script can be used as a start for creating a LSC^2 grid and you need to use the matlab 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 through the FORTRAN carefully and understand the details.

- Nudging scripts: gen_nudge.f90, gen_nudge2.f90, gen_nudge_from_hycom.f90

The 1st two scripts generate a simple elliptic nudging zone or a zone with fixed distance from boundary as *_nudge.gr3. The 2nd script generates *_nu.nc from HYCOM (you may modify this for other gridded data sources from other structured-grid models).

- Gen_Hotstart/change_hotstart4.f90

This simple script shows you the internal structure of hotstart.nc and how to manipulate it.

- Gen_Hotstart /gen_hot_3Dth_from_hycom.f90

This script shows you an example of how to create hotstart.nc and *.th.nc 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.

5.3 Visualization

The easiest way to visualize SCHISM nc4 outputs is via VisIT (https://wci.llnl.gov/simulation/computer-codes/visit/downloads). California Dept. of Water Resource group has developed VisIT plug-ins for SCHISM and more info can be found at our Forum site: http://ccrm.vims.edu/w/index.php/Visualization_with_VisIT

5.4 Post-processing

You can find some useful post-processing tools in the src/Utility directory.

- Combining_Scripts/

combine_output11.f90 is used to combine process-specific netcdf to global netcdf. *combine_hotstart7.f90* is used combine process-specific hotstart outputs (outputs/hotstart_0*.nc) into hotstart.nc. *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/interpolate_variables7.f90

The purpose of this script is to generate elev2D.th.nc, SAL_3D.th.nc, TEM_3D.th.nc and/or uv3D.th.nc from a large-domain run to be used in a small-domain run. This is of limited utility now because uv3D.th.nc for the sub-tidal component can be generated using tidal packages such as FES.

To prepare for the nesting, first do a 2D barotropic run for a larger or same grid, with only elevation b.c. Note that 2D model is inherently more stable than 3D model, and to further enhance stability, make sure you use indvel=1 (ishapiro=ihorcon=0), thetai=1, and also use a large Manning's *n* (e.g., 0.025 or larger) near the boundary. Once this is done,

(1) use *interpolate_variables7.f90* to generate *[23]D.th.nc for the small-domain run;

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

Note that interpolate_variables.in in the directory are sample inputs for the script. A common mistake is that the parent elements of some open boundary nodes in fg.gr3 (i.e. 'small-domain' hgrid) become dry and the script then fails to find a wet element to interpolate from. So make sure all open boundary nodes in fg.gr3 are located in wet region of bg.gr3; this is especially important for those nodes near the coast. You can use xmgredit5 to ascertain this. If necessary, modify fg.gr3 by moving some nodes to deeper depths.

- Time series extraction with read_output*.f90

This group of scripts read multiple nc4 outputs and extract time series of a point, a slab, a transect etc. They share similar code structure and can be used to understand the nc4 output format as well as how to do your own processing. You may start from read_output*_xyz.f90. After you are familiar with these scripts, you can easily customize them for your own purpose.

5.5 Simple test cases

For beginners, it's important to start from simple set-up and progress toward more complex set-ups. 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/. Note that these tests are used to test the master branch and so you might need to change param.nml sometimes – please consult src/Readme_beta_notes or the sample param.nml for possible changes.

5.5.1 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.

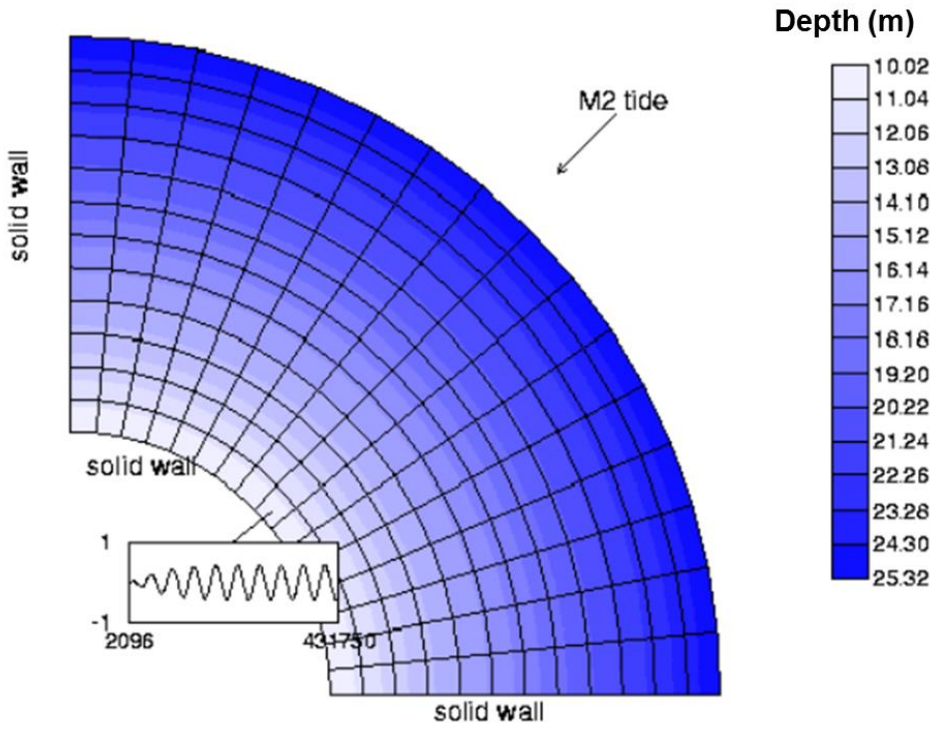


Fig. 5.6: Quarter annulus domain.

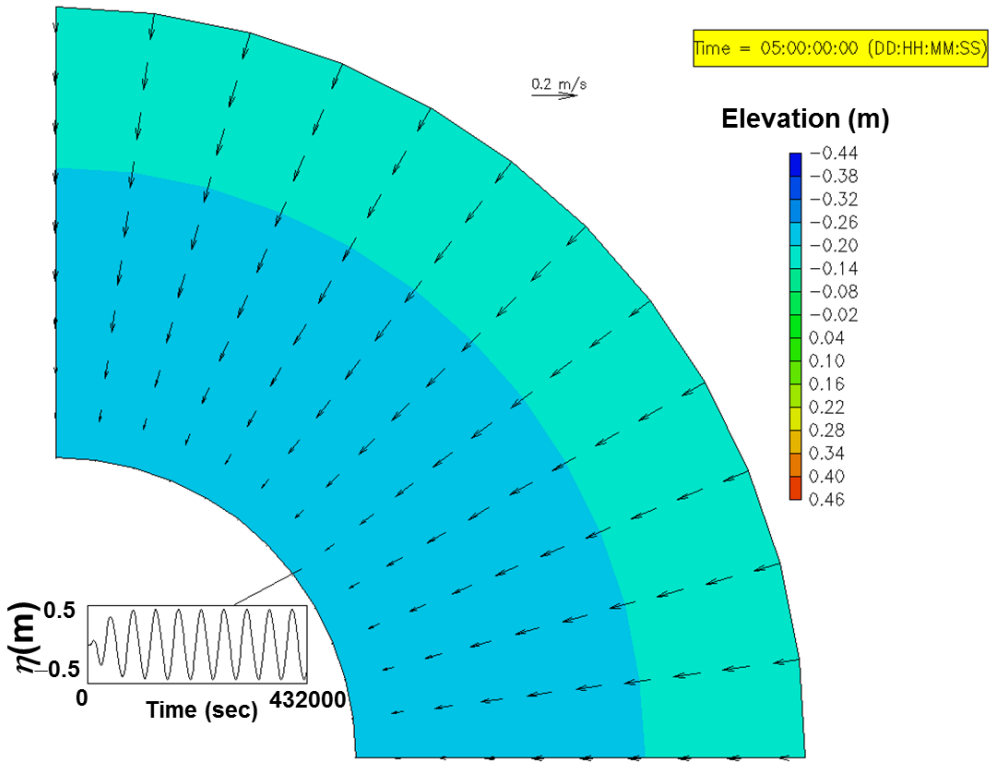


Fig. 5.7: Visualization of elevation and depth-averaged velocity, and time series of elevation near the interior boundary, using xmvis6.

5.5.2 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.

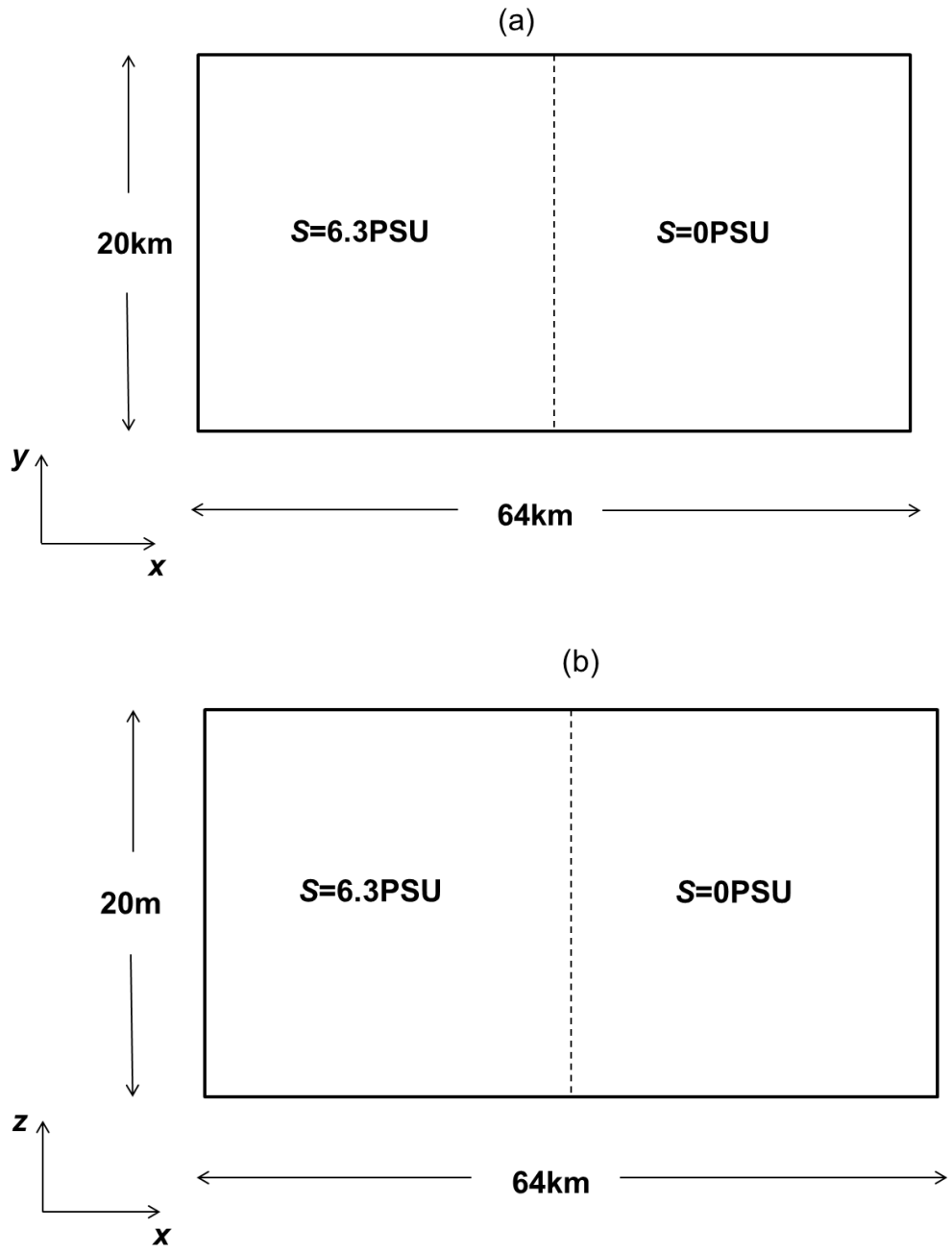


Fig. 5.8: (a) Top and (b) side view of domain.

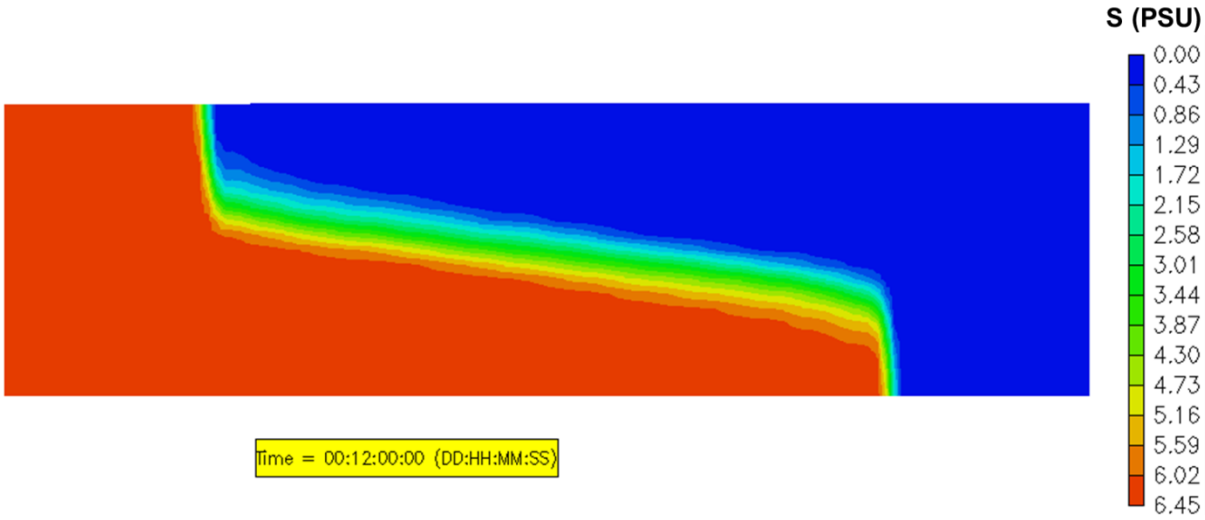


Fig. 5.9: xmvis6 transect view of the density front propagation at the end of simulation.

5.6 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'. A good resource for beginners is a mini live manual written on Overleaf by Ms. Christelle Auguste (U. of Tasmania).

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 fatal.error 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 fatal.error 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.nml:

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 VisIT. Usually you may see some large/noisy velocity somewhere, which may give you some hints on grid or forcing issues 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_output*) 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.

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 allows dry open-boundary nodes, but the entire open boundary segment cannot be dry at any given time if the velocity B.C. is imposed there. Either relocate the boundary or deepen some nodes to allow flow to come in.

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 largely consistent with each other. The velocity B.C. can be generated using global circulation model (HYCOM etc) plus global tidal model (FES etc).

9. fatal.error 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.

5.7 Clinical case studies

In this section we will discuss some common mistakes made by users.

5.7.1 Cross-scale applications that include eddying regime

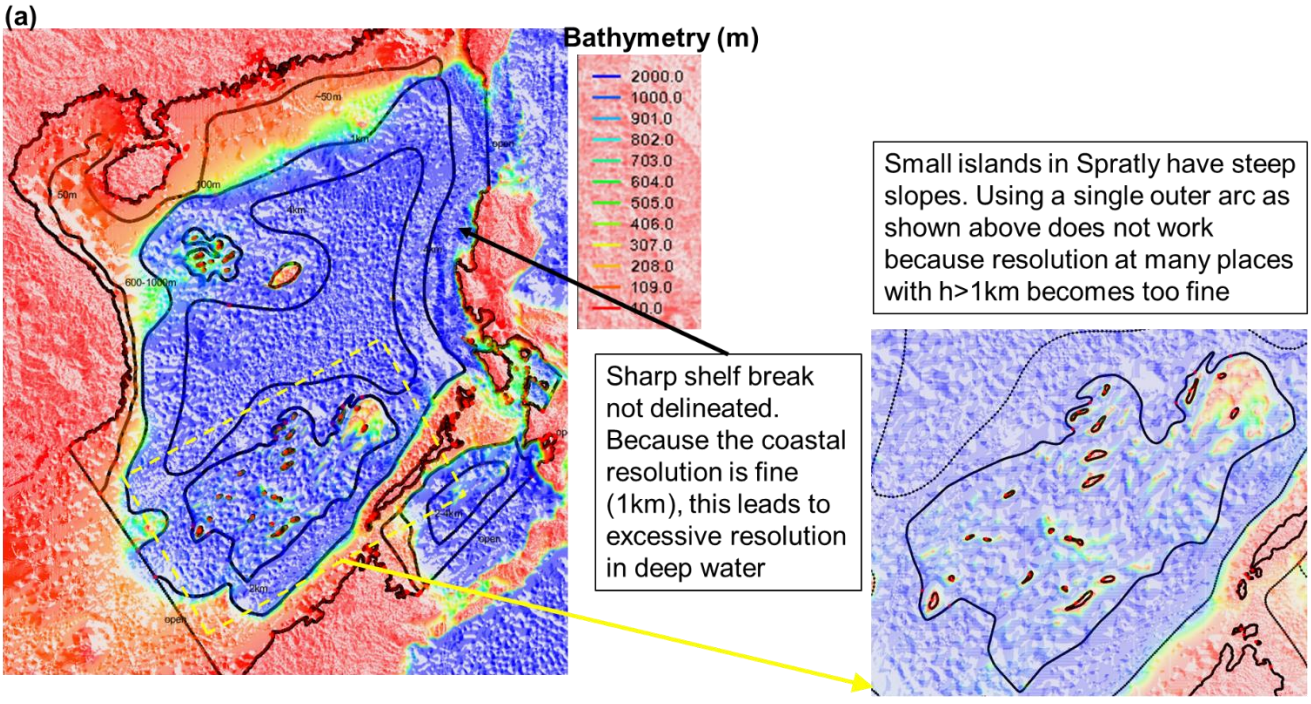
To get a good SCHISM baroclinic setup for this type of applications, one needs to pay attention to

1) Horizontal grid

2) Vertical grid: use LSC^2, and at least resolve the surface layer
3) Parameterization: especially important are parameters that control momentum dissipation (indvel, ihorcon, ishapiro, dt)

The case study on South China Sea (SCS) shown below illustrates some common mistakes made by users. A particularly severe challenge is a transitional regime (between eddying and non-eddying regimes) with steep slopes that tend to excite parasitic noises, and a common symptom for this manifests itself as spurious upwelling of cold water (Fig. 5.11).

For gridgen, start with *modest* and quasi-uniform resolution for eddying regime. High resolution (<=2km) in deep water (h>=1km) should be avoided (Fig. 5.10) (if this is absolutely required, one needs to reduce time step below 100 sec or apply some special parameterization such as niter_shap etc). At steep slopes, this means that an 'outer arc' (in SMS map) may need to be added to delineate the boundary between the 2 regimes (e.g. at shelf break; Fig. 5.12). Note that skew elements are fine in the shallow non-eddying regimes (e.g. coast). One may be tempted to omit this arc for small islands, but the grid generated this way would be too fine in deep water (Fig. 5.10), which leads to spurious noise (Fig. 5.11). Same procedure should be applied to narrow shelves e.g. west of Luzon (Fig. 5.12). With the corrected grid, the noise is greatly reduced: the remaining noise in southern Philippines can be rectified by adding outer arcs there (Fig. 5.13).



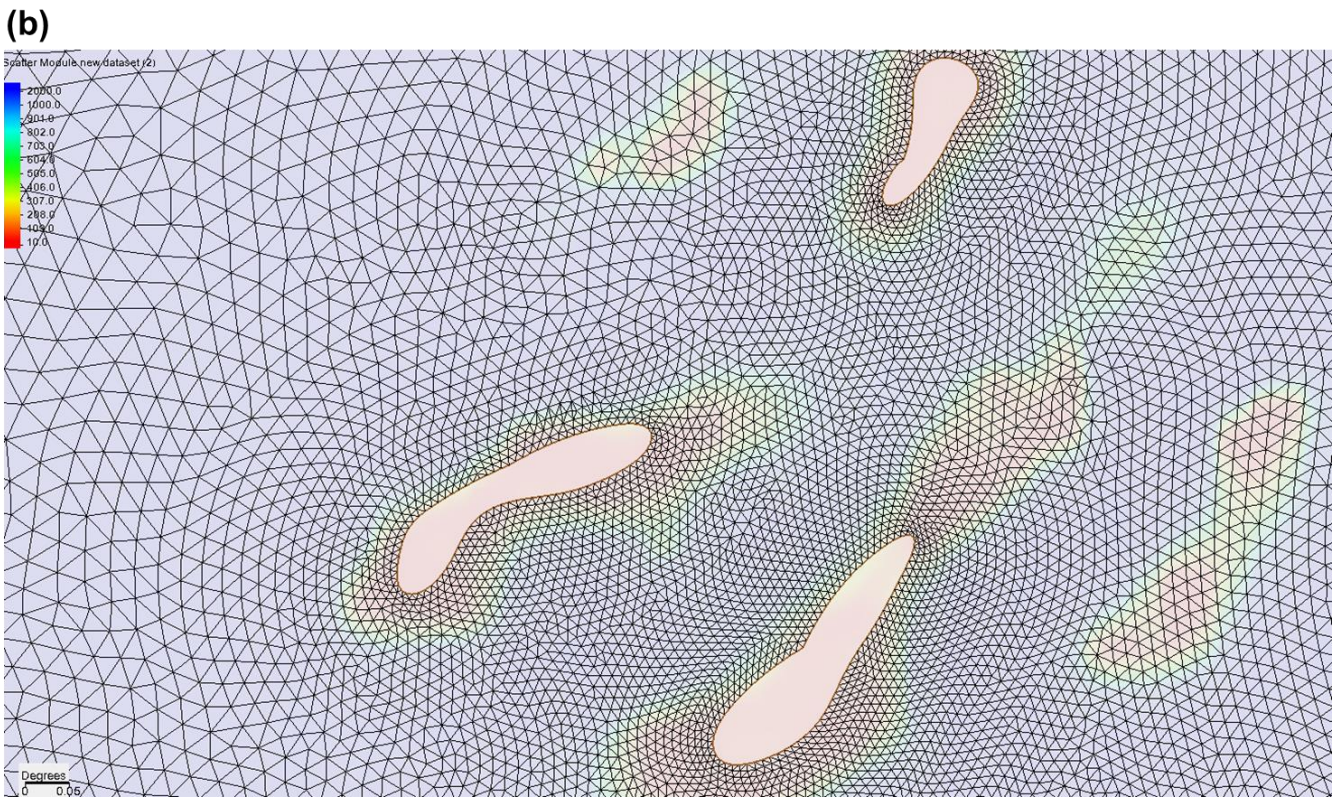


Fig. 5.10: (a) 'Wrong' map for SCS, showing multiple issues with grid design. Steep slopes are prevalent in this region, near small islands and continental shelf breaks. (b) grid near Spratly, showing excessively high resolution in deep water (0.01 degree ~ 1 km).

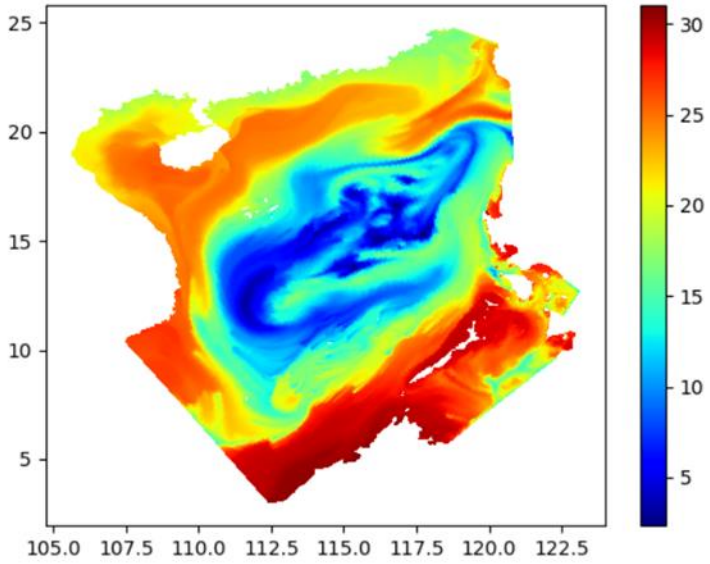


Fig. 5.11: Surface temperature (SST) resulted from the grid generated from Fig. 5.10, showing the excessive spurious upwelling of cold water.

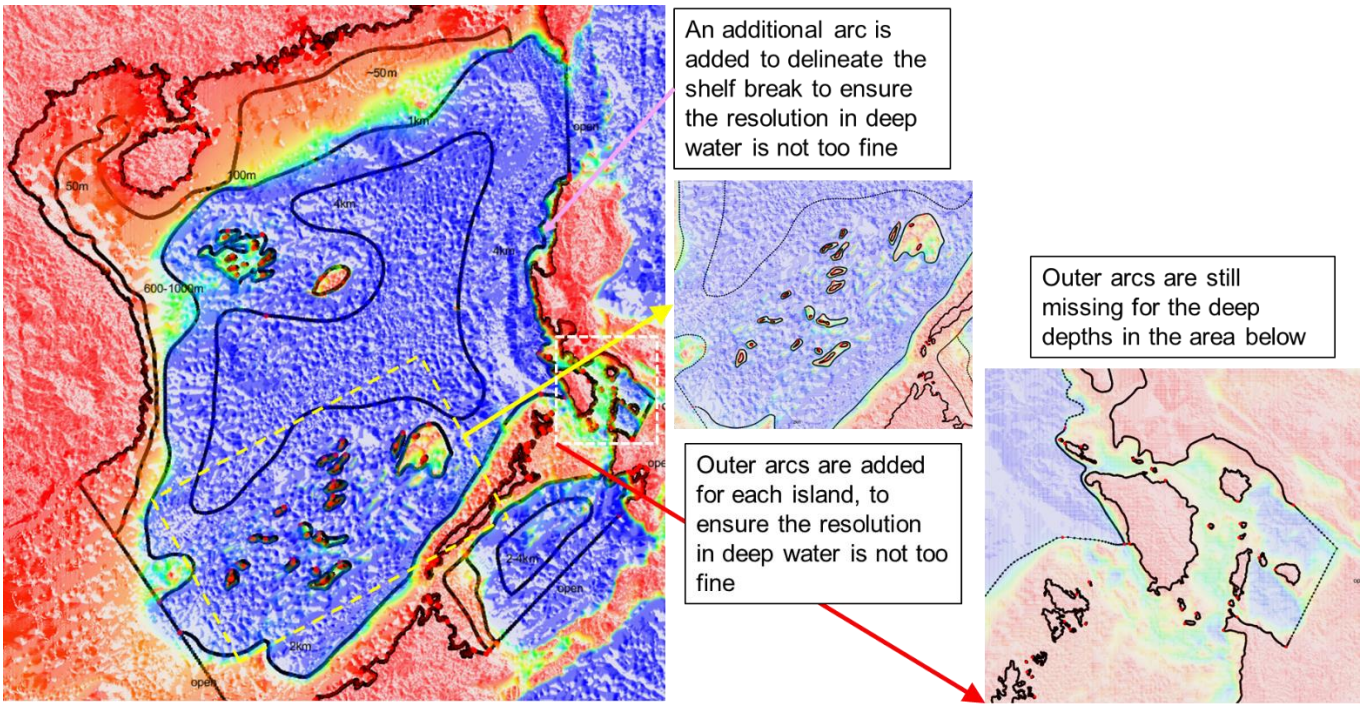


Fig. 5.12: A corrected SMS map file. Outer arcs are added near shelf break of west coast of Luzon and Spratly Islands, but are missing in the southern Philippines.

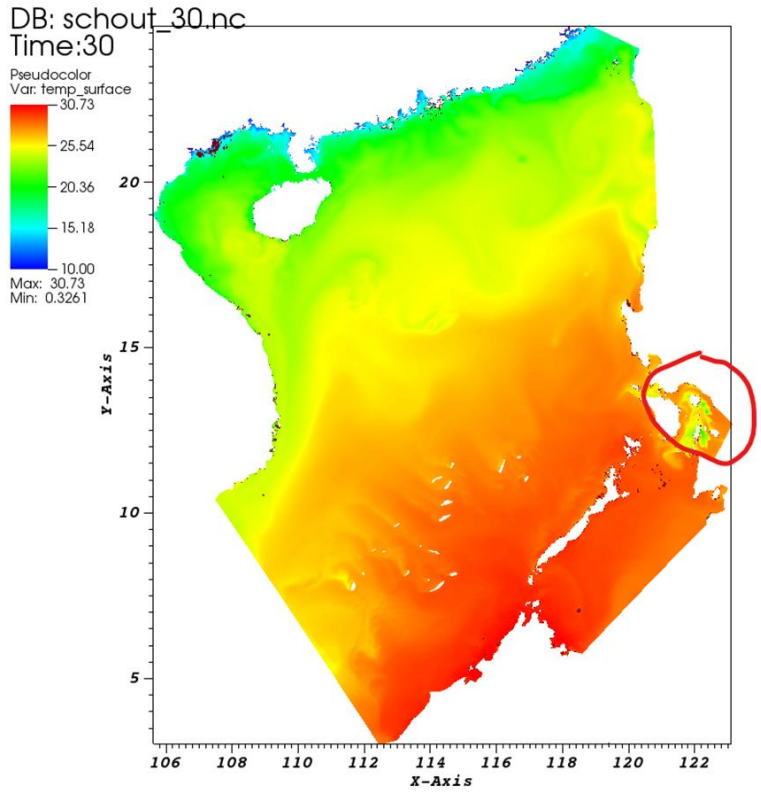


Fig. 5.13: SST calculated from the corrected grid. The remaining 'upwelling' in the circled area is due to the missing outer arcs in that area (cf. Fig. 5.12).

Parameters: dt=100s (slightly larger values like 120 s should also work), indvel=0, ihorcon=2, ishapiro= −1. shapiro.gr3 is then generated using gen_slope_filter2.f90 by using larger filter strengths near steep slopes (and small elsewhere). Fig. 5.14a shows the resultant input. In general, the momentum needs to be stabilized with larger viscosity near steep slopes, as bi-harmonic viscosity alone is not sufficient there. The I.C. and B.C. can be derived from HYCOM and FES 2014 (i.e. use type '5' for both elevation and velocity B.C.). In addition, tracers are relaxed to HYCOM values near open boundaries (Fig. 5.14b). Required pro-processing scripts can be found in Utility/.

Starting from v5.9, users can also try the new Smagorinsky–like filter option (ishapiro=2) with a proper shapiro0 (see Section 4.2.4.2).

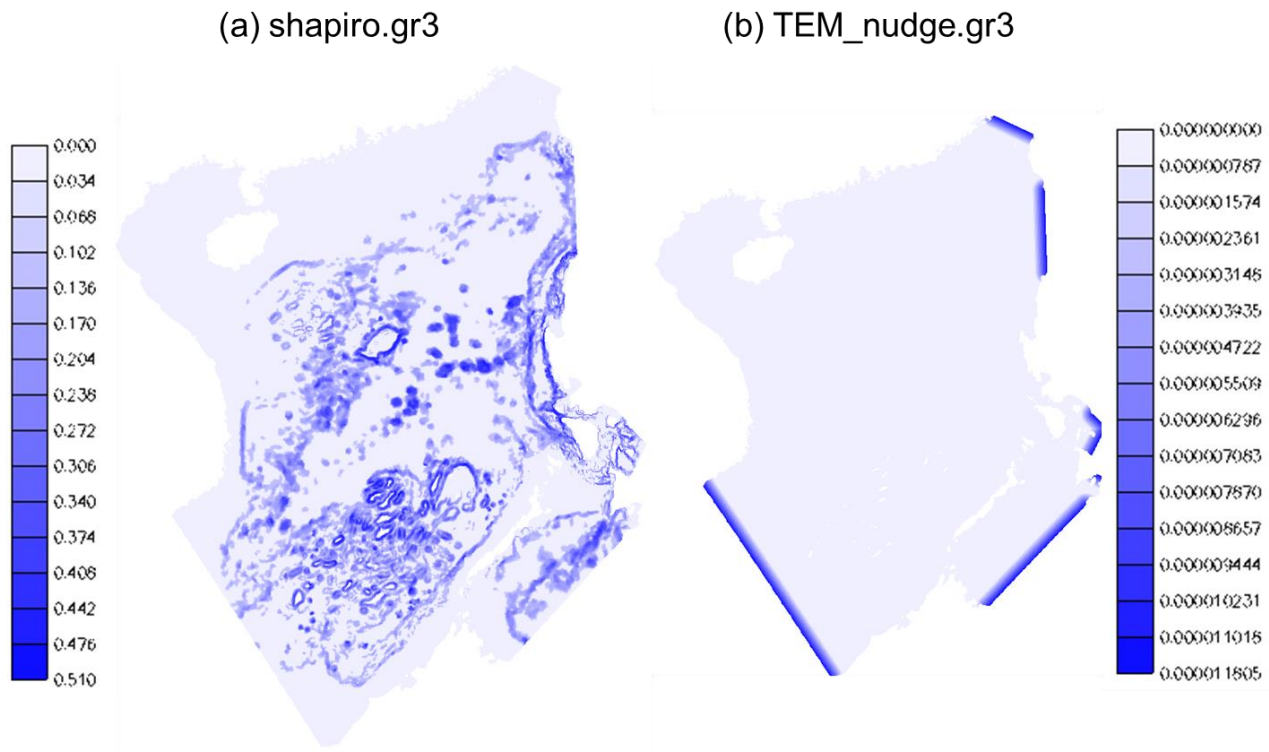


Fig. 5.14: (a) Filter strength input (shapiro.gr3). Larger values are used near steep slopes. (b) Nudging strength input (TEM_nudge.gr3, SAL_nudge.gr3), generated from gen_slope_filter2.f90, with maximum time scale of 1 day.

5.7.2 More on grid generation in cross-scale applications

Mesh generation is most challenging in applications that include eddying and transitional regimes (i.e. with open deep ocean water). This is because users often tend to over-refine in these regimes, thus violate the hydrostatic assumption (the horizontal scale >> vertical scale). For example, setting mesh resolution at 500m at 500m depth would likely lead to spurious upwelling discussed in the previous sub-section. If sub-mesoscale processes are of interest, users will need to apply some treatments such as reducing the time step and using iterative viscosity filter etc.

This challenge is particularly acute when one deals with islands sitting on top of steep slopes. Fig. 5.15 shows 3 attempts to add high resolution near Guam in the Pacific basin mesh. The mesh resolution in the surrounding deep ocean is ~5km, and we need to refine the mesh to ~30m around Guam. The Apra Harbor jetty that requires higher resolution happens to be located close to a steep slope. Combination of wet/dry and forced sharp transition in 'new14' and 'new15' led to spread of upwelled water there. Removing an inner arc in 'new15' helped smoother transition from the outer arc to the inner arc and thus alleviated the upwelling issue. In 'new16', we made more 'room' for transition by moving the inner arc away from the outer arc and coarsening it a little (to match the resolution of the outer arc), thus further reducing the upwelling. However, we had to add more internal arcs to provide adequate resolution nearshore (otherwise the nearshore area would be represented by a few skew elements that have side lengths of a few km's on the outside and 30m on the inside). Fig. 5.16 shows the details of 'new16' and final mesh. Note that in all three maps, we used 500m isobaths for the outer arc (as representation of start of eddying regime) but the arc veered to 1km isobaths near the harbor as a way to make more room for transition there.

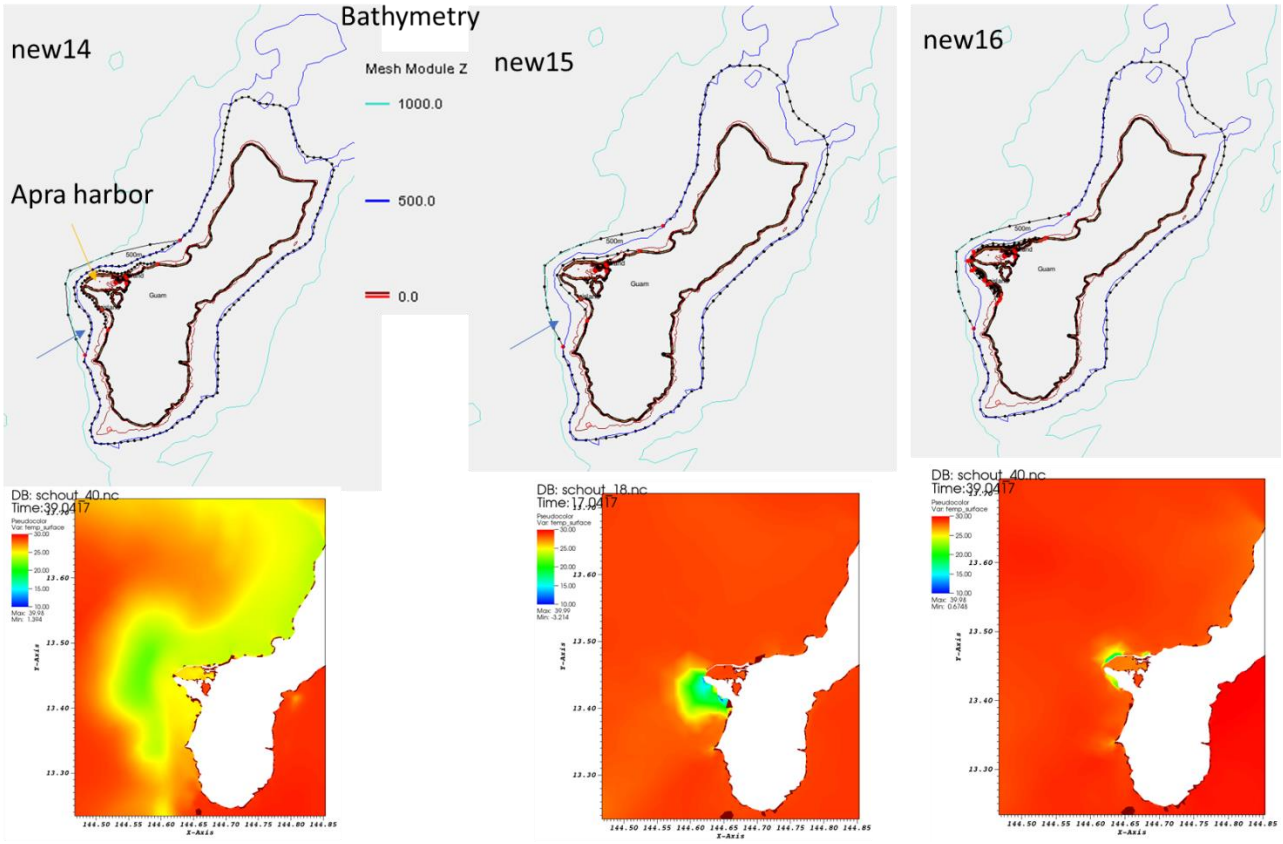


Fig. 5.15: Three SMS maps (top) and corresponding results for SST.

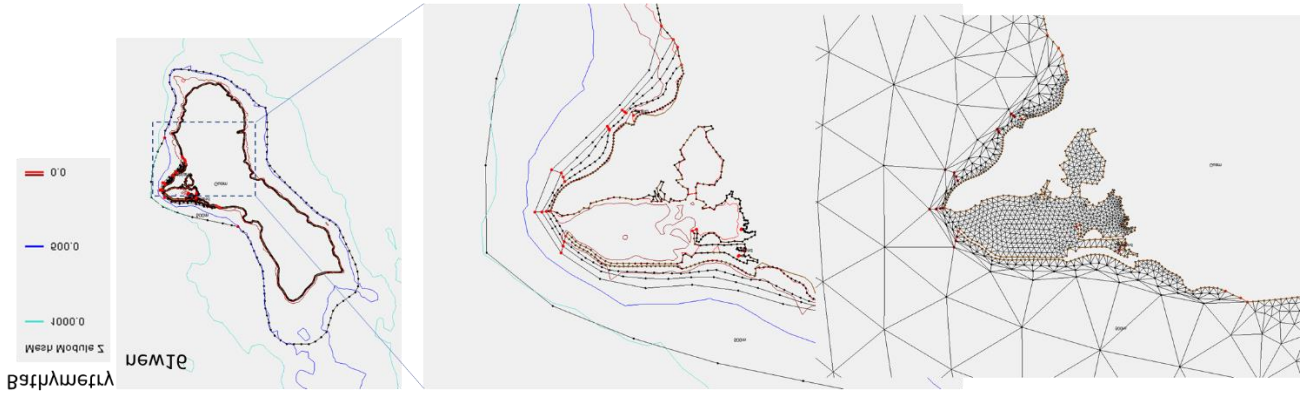


Fig. 5.16.: Details for map 'new16' and the final mesh around Apra Harbor. Skew elements nearshore are fine.

5.7.3 Hydrologic flow

Simulation of hydrologic flow in watershed, with bottom elevation above sea level (thus strong wetting and drying) and with complex river channel network, is challenging. The discussions below are taken from a training course on compound flooding simulation.

A common, convenient method for introducing river flows into watershed in SCHISM mesh is via the point source/sinks as shown in Fig. 5.17. Together with proper parameter choices (e.g., a small minimum depth of 10^{-5}m, a fully implicit scheme, a large bottom friction with proper vertical grid to allow for 2D representation in the watershed etc) this usually works fine for smaller rivers. For large rivers, the open-boundary approach (i.e., river channels as open boundary segments) is the preferred method. In general, the point source approach injects flow via the continuity equation alone without providing extra momentum (note that the volume sources/sinks are added to the RHS of Eq. (2.2) and (2.3) but not in the momentum Eq. (2.1)), and thus it will take the system some time to adjust internally to set up the flow from the pressure gradient. Because of this drawback, large elevations may be found near the injection points, especially during initial ramp up or during high and rapidly varying flow periods. This symptom can be exacerbated by the following missteps:

1. Pairs of source and sink in close proximity (Fig. 5.18a). Users should combine these pairs;
2. Steep slopes near boundary with coarse resolution (Fig. 5.18b);
3. Inverted bed slopes near the injection location (so the flow has to overcome gravity; Fig. 5.18c);
4. Poorly ventilated 'dead-end' (Fig. 5.18d);
5. Undulating channel ('water fall'; Fig. 5.18e).

The model is stable, but interpretation of results may be problematic in those cases. Besides more grid work in those spots, using open-boundary segments can help. Also one should really exclude transient responses during ramp-up period in computing the maximum elevation to allow the system time to adjust. Also don't forget that sometimes the rainfall (which can also be introduced as sources) on high mountains should result in high surface elevations there, which are realistic.

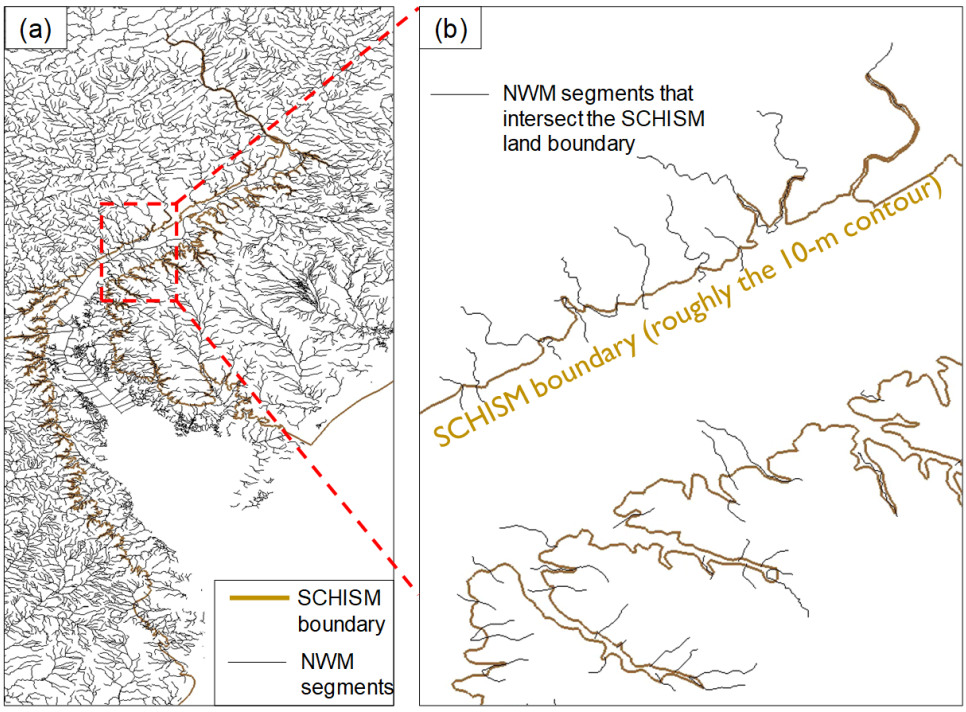


Fig. 5.17: Coupling of hydrologic model (National Water Model in this case) with SCHISM. The river network (lines) intersects SCHISM land boundary, and the river flow is introduced at the intersection points (sources for inflow and sinks for outflow).

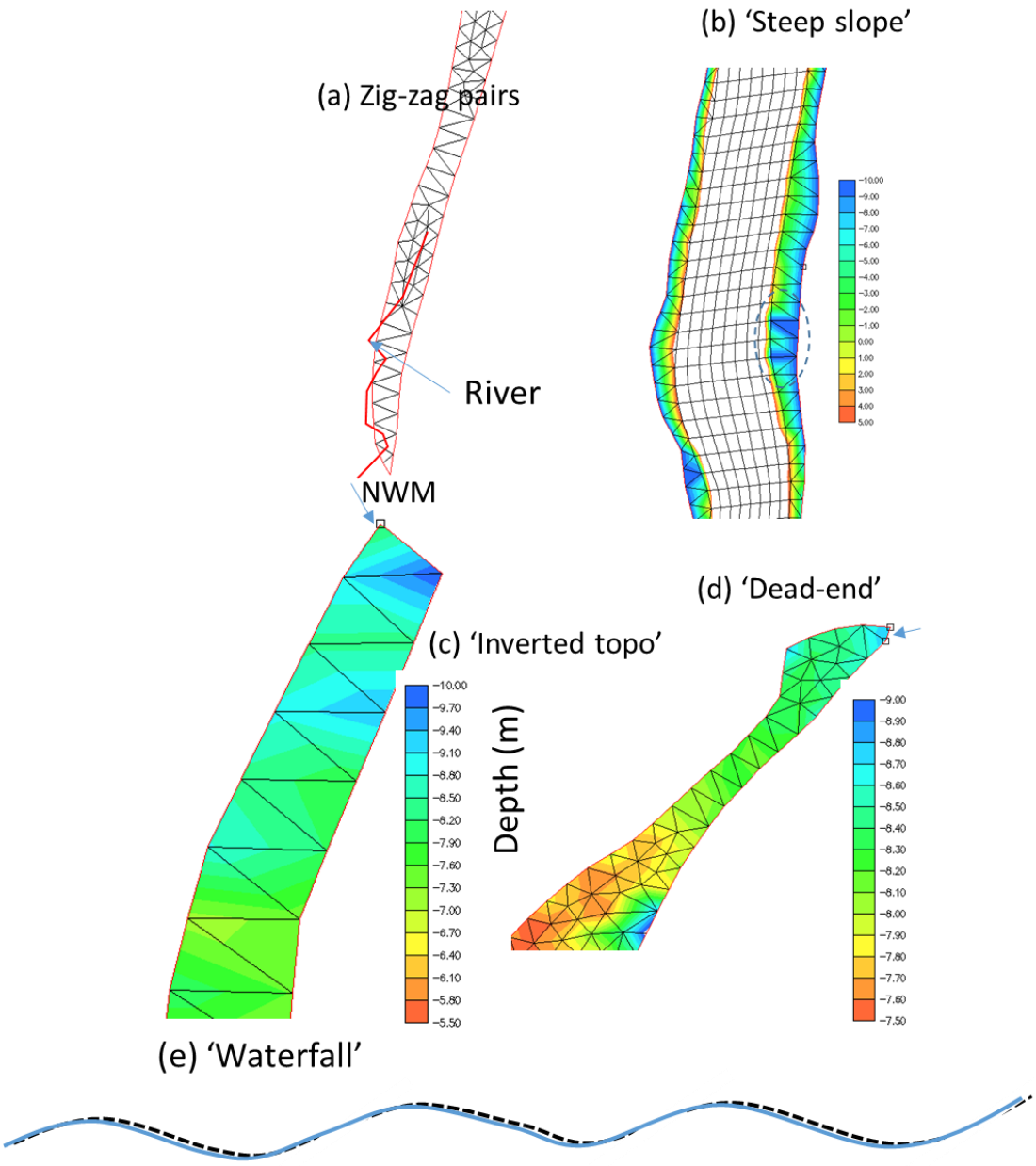


Fig. 18: Pathologic cases for hydrologic flows.

Chapter 6 SCHISM code

The information presented in this chapter is mostly intended for developers or anyone who wishes to work on the code.

6.1 General info

SCHISM was written in MPI FORTRAN 90. After svn revision 5225, we have migrated to github: https://github.com/schism-dev. Fig. 6.1 shows the directory structure of the SCHISM github repository schism. General public have access to all branches.

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.

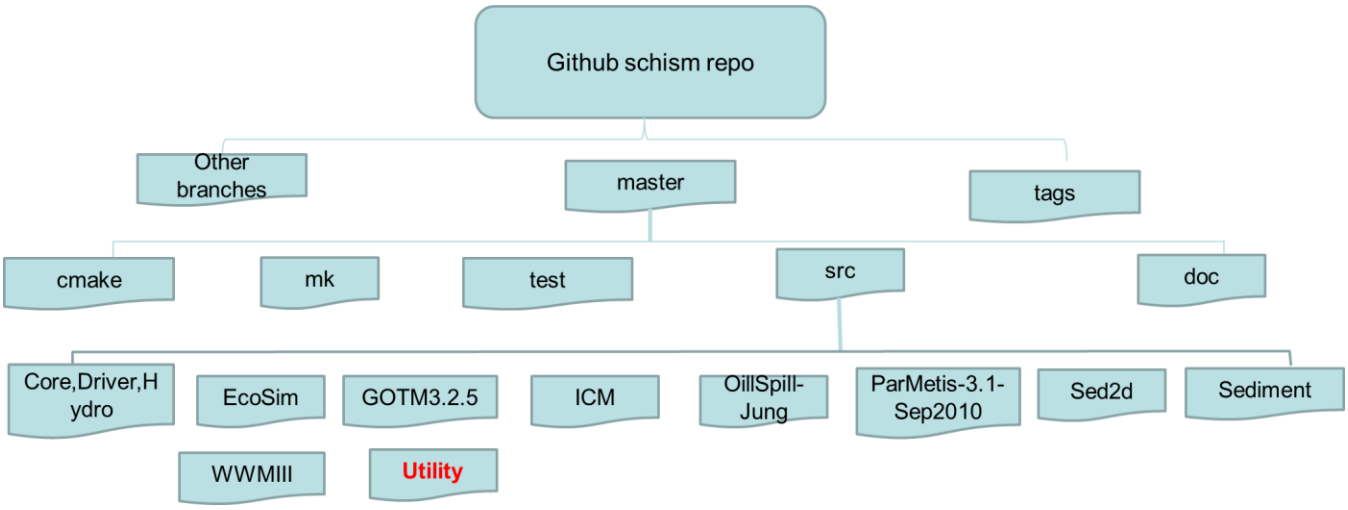


Fig. 6.1: SCHISM github repository.

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; the manual can be found on the web site.
- Combining_Scripts: FORTRAN and perl scripts used to gather outputs from SCHISM (e.g., outputs/schout_0*_[stack number].nc etc) into one binary file (schout_[stack nnumber].nc)

- 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.nc from e.g., netcdf outputs from a structured-grid model
- Pre-Processing: various scripts for pre-processing (checking, viz etc)
- OneWayNestScripts: scripts for 1-way nesting, by preparing *[23]D.th.nc (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. At the moment, these scripts have not been updated to handle LSC^2 grid.

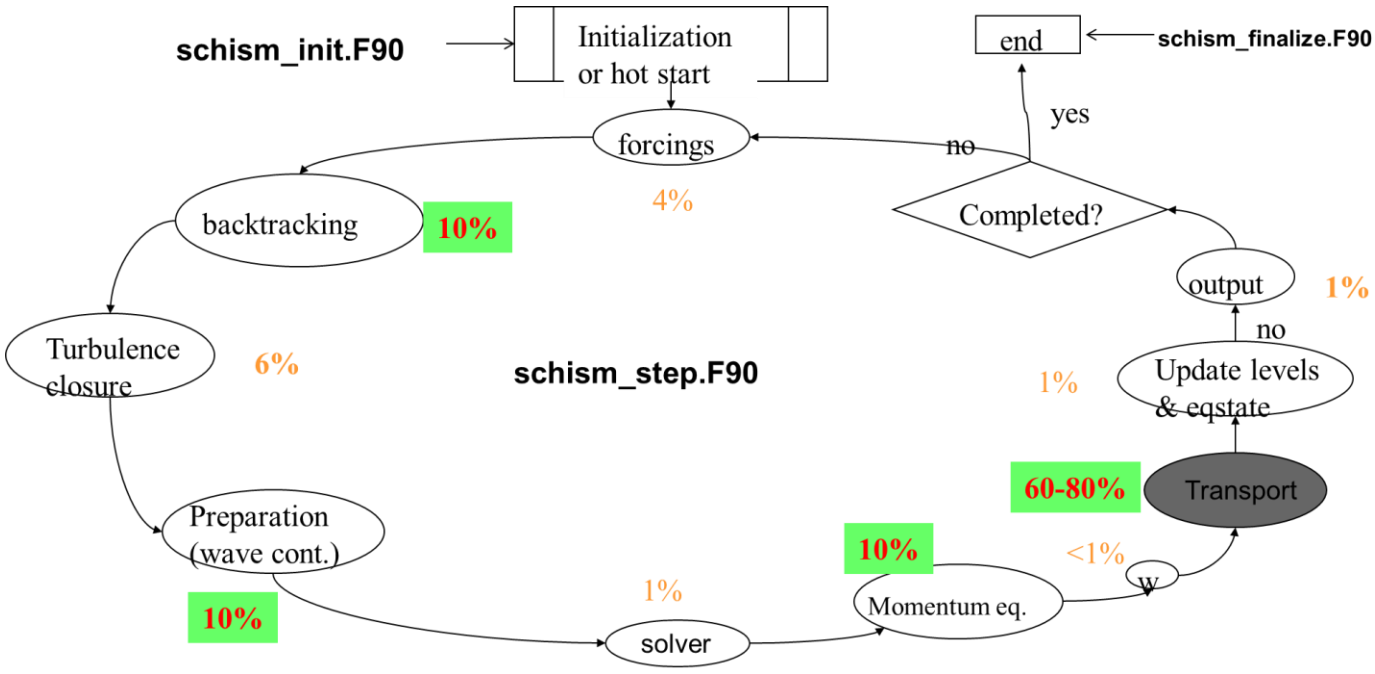


Fig. 6.2: SCHISM code work flow. The percentages are estimates from a test case with an earlier version and may not be up to date.

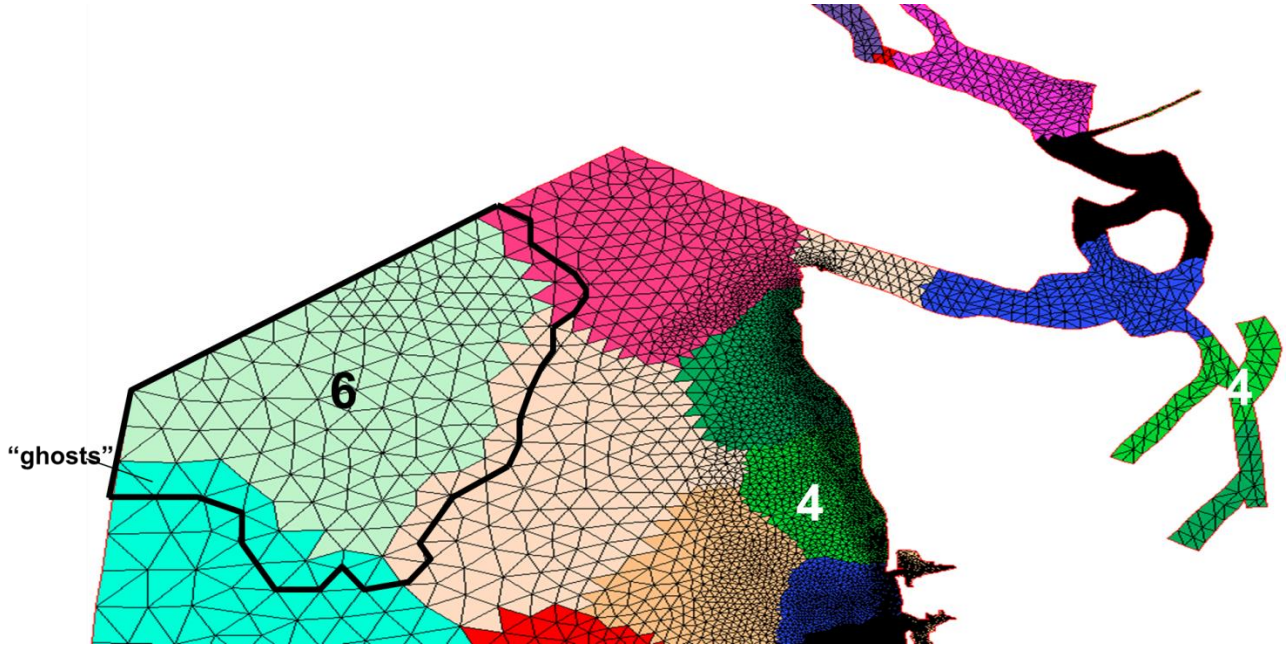


Fig. 6.3: Domain decomposition. Each color represents a sub-domain ('resident domain') taken by a MPI process and the thick black polygon represents the 'augmented' (=resident+ghost) domain of process 6.

6.2 Info on git

Please refer to CONTRIBUTING.md on SCHISM github site.

6.3 For developers working on the code

6.3.1 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;
3) Avoid using tabs in editor as they mess up with the appearance. Use space instead and strictly align code blocks for easy read.
4) Do not use automatic arrays of ≥2 dimensions. It's often tempting to grab dimensions directly from the module schism_glbl and use them to define arrays in a routine; e.g. vel_sd(nvrt,nsa). This causes trouble with some compilers and may result in segfault. Use either of the following 2 approaches instead: (1) allocatable arrays (and always remember to deallocate them at the end of the routine); (2) pass on the dimensions explicitly as dummy

arguments (e.g. subroutine routine1(nvrt2,nsa2, vel_sd..), where nvrt2=nvrt and nsa2=nsa, and then use these to define: vel_sd (nvrt2,nsa2)).

6.3.2 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.

6.3.3 Notes on SCHISM Code

6.3.3.1 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 aquire_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.

6.3.3.2 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 segements in the global domain.

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

   iopelg(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<=j<=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 segements 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 #); □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., *i*th elements from each processor are summed up and returned as *i*th 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_Allgatherv (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 discontiguous data blocks from multi- dimensional arrays.

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

$$nea\overbrace{\begin{cases} (1,1) & (2,1) & \cdots & (nvrt,1) \\ (1,2) & (2,2) & \cdots & (nvt,2) \\ \vdots & \vdots & \vdots & \vdots \\ (1,nea) & (2,nea) & \cdots & (nvrt,nea) \end{cases}}^{nvrt}$$

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 nvrt, and displacement of ith block is (iesend(i)-1)*nvrt.

- 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);
  recvcount- 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)

6.3.4 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.

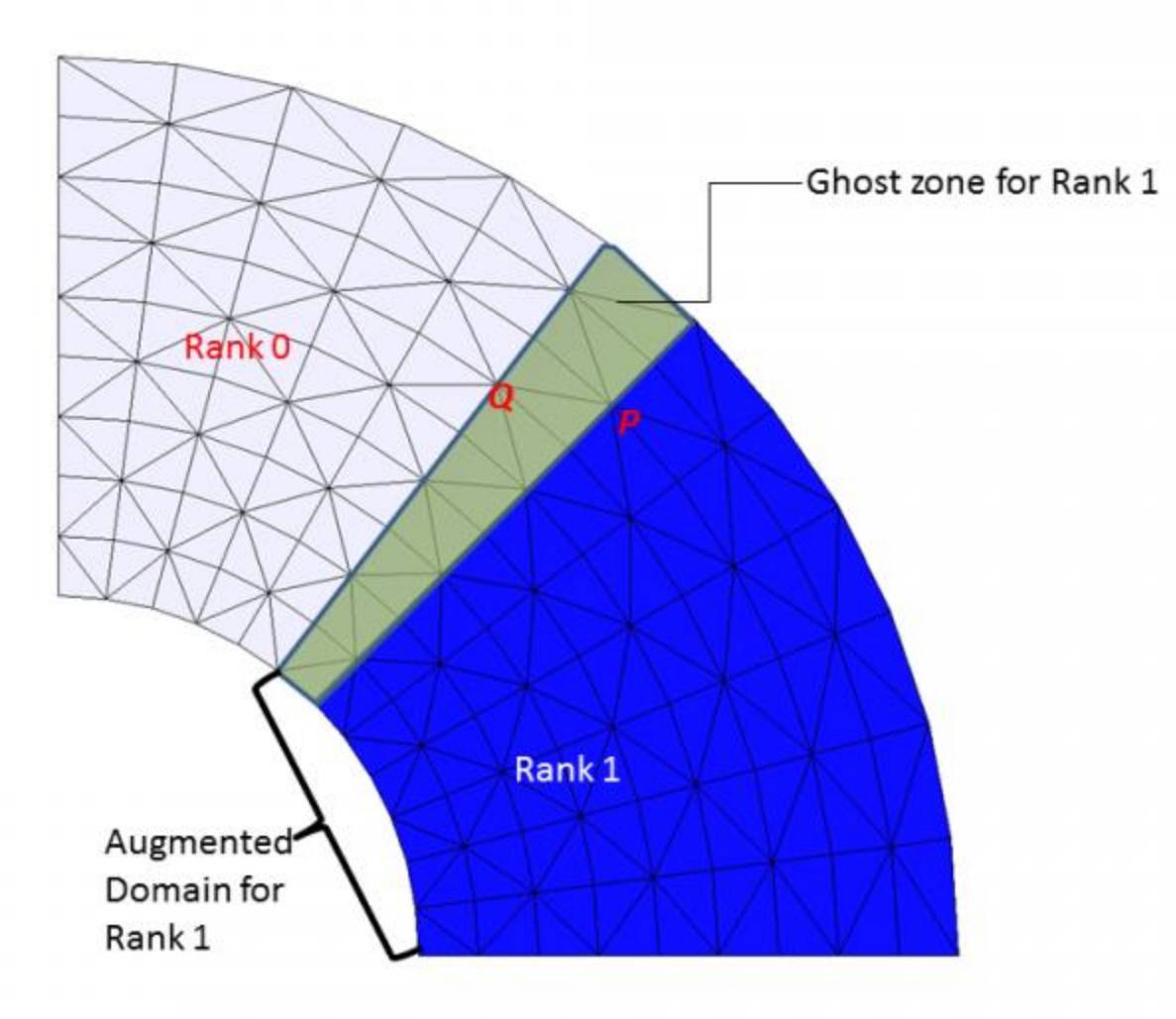


Fig. 6.4: Domain decomposition.

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(i)=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

Note that some modules are under active development and we will update the info as it becomes available in the future.

SCHISM modules can be broadly divided into two categories: tracer and non-tracer modules. The main difference is that tracer modules share more infrastructure with the main hydro code base, e.g. using the transport solver, with I.C. and B.C.'s that resemble those for the temperature and salinity, and sharing the source inputs (msource.th). Most modules also require additional inputs of their own (e.g. wwminput.nml for WWM).

There are 12 tracer modules and they are (in order of appearance and precedence in bctides.in; the names in brackets are used in input names; e.g. TEM_1.th etc):

1. Temperature [TEM]
2. Salinity [SAL]
3. Generic tracer [GEN]: generic tracer module with a settle velocity (gen_wsett in param.nml). The user can use this module as a template to add their own tracer behavior etc (by modifying the code sections bounded by USE_GEN);
4. AGE [AGE]: water age module of Delhez & Deleersnijder (2002) and Shen and Haas (2004);
5. SED3D [SED]: 3D non-cohesive sediment transport module;
6. EcoSim [ECO]: EcoSim of Paul Bissett;
7. ICM [ICM]: USACE's water quality model of CE-QUAL-ICM
8. CoSINE [COS]: Carbon, Silicate, Nitrogen Ecosystem model of Prof. Fei Chai (U. of Maine)
9. Fecalbacteria [FIB]: fecal indicating bacteria model;
10. TIMOR: not active at the moment
11. FABM [FBM]: Framework for Aquatic Biogeochemical Models, a flexible biogeochemical model framework;
12. DVD [DVD]: numerical mixing analysis of Klingbeit et al. (2014)

The B.C. flags for each invoked tracer module are specified in bctides.in. For example, if you invoked GEN, SED, and ICM, the boundary condition at an open segment may look like:

39 2 0 3 4 1 2 0 ![# of nodes], elev, vel, T,S, GEN, SED, ICM

0.5 !constant elev

0.1 !relax for T

0.1 !relax for S

0.5 !relax for GEN

0. !constant SED concentration

1.e-3 !relax for SED

[next segment…]

And in addition, you'll need to prepare inputs: SAL_3D.th.nc and GEN_*.th.

Similarly, msource.th should also include the tracer concentrations for all invoked modules; for the example shown above, a line in the msource.th should look like this (assuming 2 sources in source_sink.in):

86400. 10. 10. 0. 0. -9999. -9999. 0. 0. -9999. …-9999. !time (sec), T, S, GEN, SED, ICM

21x2 values for ICM

For some tracer modules the user also needs to specify number of tracer classes inside the module in param.nml:

ntracer_gen = 2 !user defined module (USE_GEN)

ntracer_age = 4 !age calculation (USE_AGE). Must be =2*N where N is # of age tracers

sed_class = 5 !SED3D (USE_SED)

eco_class = 27 !EcoSim

The output flags for all modules are iof_[name], where name is the lower case of the module name; e.g. iof_wwm(). See the sample param.nml for a complete list of output flags as well as the variable names that appear in the outputs schout*.nc. Some modules have additional parameters specified in param.nml; e.g., gen_wsett, flag_fib etc. See the sample param.nml for explanation.

7.1 Generic tracer module

We provide this module to users as template for potential addition of behavior etc. As is, the module simulates multiple classes of passive tracers with constant settling velocity gen_wsett (gen_wsett <0 =>swimming velocity), with no body forces and zero fluxes at surface and bottom. The number of generic tracers is specified as ntracer_gen. I.C. flag is flag_ic(3), and nudging flag is inu_tr(3) in param.nml. The output flags are iof_gen(1:ntracer_gen).

7.2 AGE

AGE module is different from other tracer modules in that it works with even number of tracers, with the 1st half being the age tracer concentration and the 2nd half being the corresponding 'ages' (Shen and Haas 2004). The code hardwires the I.C. flag for this module to be '1', i.e. horizontally variable conditions for all tracers specified in AGE_hvar_*.ic. Inside each .ic for the 2nd half of tracers, set a uniform 0 at all nodes. For tracers in the 1st half, the user specifies the

age concentration to be either 0 or 1, the latter in specific regions where the age tracers will be *continuously* injected into the domain. For this reason, we suggest setting all B.C. flags for this module to be '0' in bctides.in.

The vertical positions where the age tracers are injected are specified in param.nml in the array level_age(1: ntracer_age/2); e.g. for ntracer_age=4, we may specify:

level_age = 9, -999

to inject the 1st age tracer at level 9, and the 2nd tracer at all levels (and the code will reset the tracer concentrations to 1 at the same 'injection' elements and level(s) at each time step). The output flags for the water 'age' in days are specified for half of the total number; e.g., iof_age(1:2)=1, 0.

7.3 3D Sediment model

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. 2008). 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.

- 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.nml, 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.nc. The nudging inputs may be SED_nudge.gr3 and SED_nu.nc.

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 ICM

A preliminary user manual for this module can be found on the Manual site.

7.5 CoSiNE

A complete manual for this module can be found on the Manual site.

7.6 DVD

Klingbeit et al. (2014) proposed a theory of estimating numerically induced mixing coefficients. At the moment we have only implemented this module for 1 tracer (salinity).

The I.C. for this module is hardwired to be 0 and the code will set it automatically. The B.C. flags should also be 0 in general, so is the nudging flag. The main output is iof_dvd(1) which corresponds to numerical mixing for salinity in PSU^2/s.

7.7 Marsh migration

This module is not a tracer module and simulates the long-term migration of marshes under sea level rise. It is usually invoked together with SED (with optional morphological acceleration) and optionally with WWM; cf. Nunez et al. (2020).

The only parameter for this module is slr_rate (sea-level rise rate in mm/yr). The output flag is iof_marsh, which is an integer of either 0 (no marsh) or 1 (has marsh) at an element. Optionally, the user might also consider invoking the vegetation option in the code isav in conjunction with the marsh module to simulate the form drag and turbulence induced by the marsh vegetation.

Additional inputs are required to specify the I.C. for marsh extent (marsh_init.prop) as well as migration barrier info (marsh_barrier.prop); both use 0 or 1 to specify 'on/off'.

7.8 Analysis mode

Under this module the user can output several internal arrays inside the code, and this is useful for detailed analysis like momentum and tracer budget etc. We have implemented some commonly requested terms in the momentum and transport equations, which can be found in the code by searching for '_ANALYSIS' and also in param.nml under iof_ana. The user can also use this module to output additional terms if they are familiar with the source code. Note that invoking this module may slow down the code.

7.9 Hydraulics

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

7.10 Particle tracking

SCHISM supports a 3D particle tracking code in Utility/Particle_Tracking/ptrack3.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, and schout*.nc. The main parameters are set in particle.bp. Below is a sample (bold texts are comments ignored by the code):

```
1 screen outputs

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.nml)

0.1 8. 90. 10 960 9 !h0,rnday,dt,nspool,ihfskip,ndeltp (same as param.nml
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 ACE/xmvis6. Fig. 7.1 shows an example.

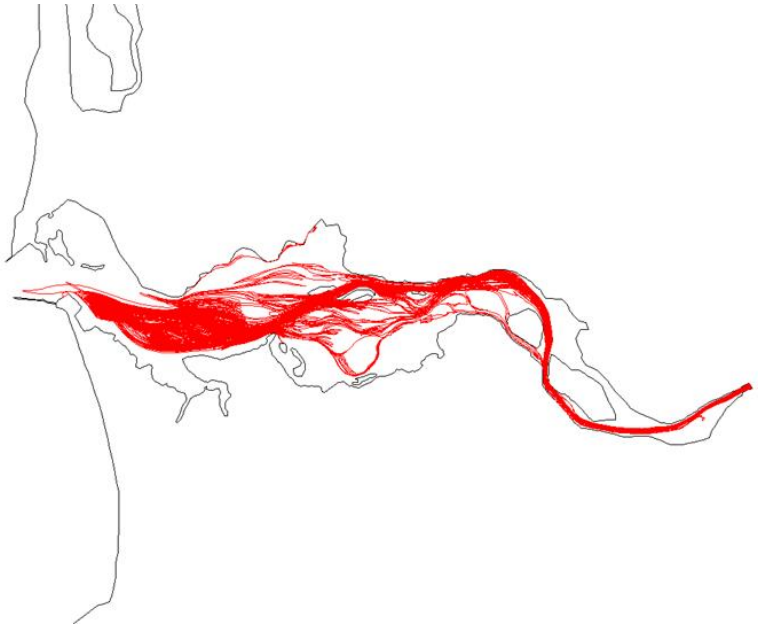


Fig. 7.1: Particle tracking with SCHISM.

7.11 2D Sediment model (SED2D)

The manual for this module can be found at:
http://ccrm.vims.edu/yinglong/wiki_files/Manual_SED2D.pdf

7.12 WWM

A preliminary manual for this module can be found in: src/WWMIII/Manual/manual.tex. For the detailed theory and numerical methods used in WWM, you can consult Dr. Aron Roland's PhD thesis.

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. Geophy. Res, 99(25), pp. 235-266.

Klingbeil, K., Mohammadi-Aragh, M., Gräwe, U., Burchard, H. (2014) Quantification of spurious dissipation and mixing – Discrete variance decay in a Finite-Volume framework, Ocean Modelling, https://doi.org/10.1016/j.ocemod.2014.06.001.

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.

Nunez, K., Zhang, Y., Herman, J., Reay, W. and Hershner, C. (2020) A multi-scale approach for simulating tidal marsh evolution, Ocean Dynamics, https://doi.org/10.1007/s10236-020-01380-6

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.

Shen, J. and Haas, L. (2004) Calculating age and residence time in the tidal York River using three-dimensional model experiments, Estuarine, Coastal and Shelf Science, https://doi.org/10.1016/j.ecss.2004.06.010.

Song, Y. and D. Haidvogel (1994) A semi-implicit ocean circulation model using a generalized topography-following coordinate system. J. Compt. 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.

Warner, J.C., Sherwood, C.R., Signell, R.P., Harris, C.K., Arango, H.G. 2008. Development of a three-dimensional, regional, coupled wave, current, and sediment-transport model. Comput. Geosci. 34 (10), 1284–1306. doi: 10.1016/j.cageo.2008.02.012

Wilcox, D.C. (1998) Reassessment of scale determining equation for advance turbulence models. AIAA J., 26, pp. 1299-1310.

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.

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.