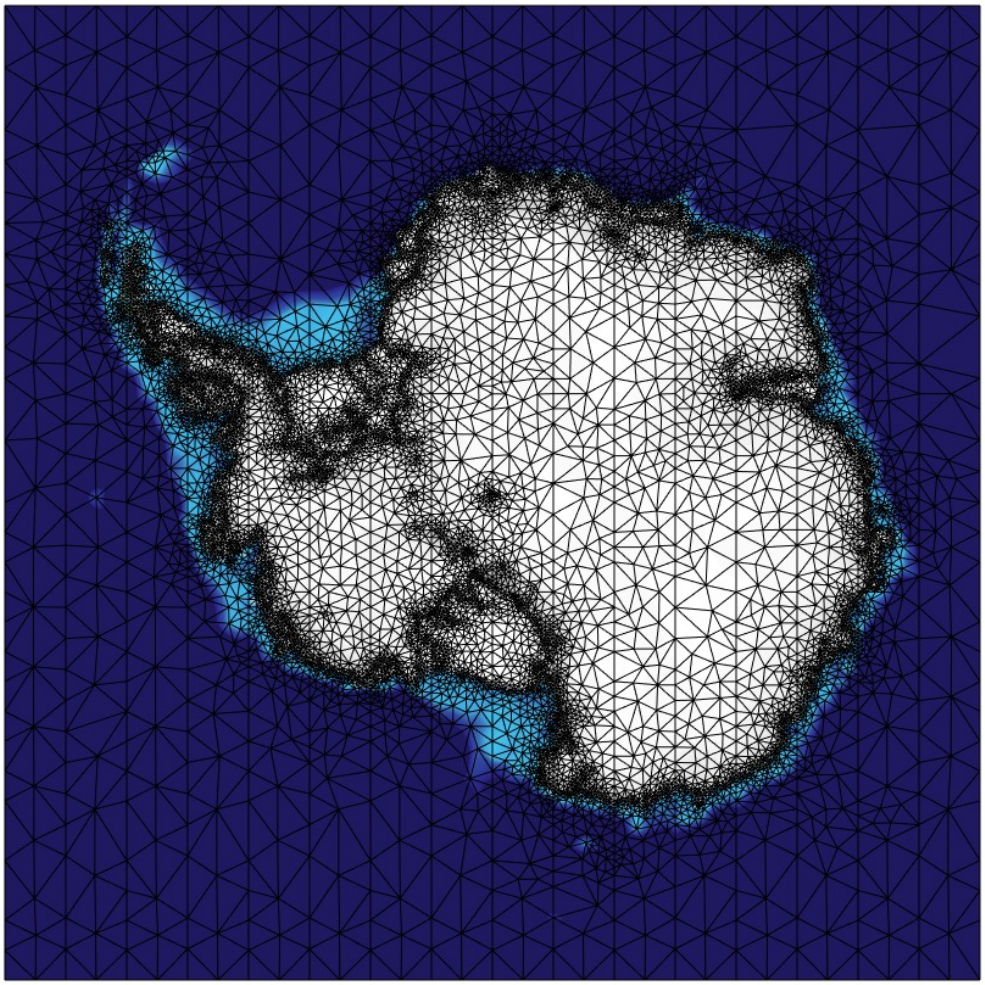
UFEMISM - documentation

dr. C. J. (Tijn) Berends

2021-04-03



Abstract The Utrecht Finite voluMe Ice-Sheet Model (UFEMISM) is an ice-sheet-shelf model that solves the hybrid SIA/SSA ice-dynamical equations on a dynamic adaptive grid. It has been developed at IMAU since 2020. While intended mainly for palaeo-applications, the model is very versatile, and can be used for future projections as well. This document contains an in-depth description of the model, including a derivation of the discretisations of the underlying equations, and a description of the data and code structure.. In theory, this should give the user enough information to be able to freely use and adapt UFEMISM for their own research.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Objectives . . . 1
- 1.3 Status quo . . . 2
  - 1.3.1 Features . . . 2
  - 1.3.2 Benchmark experiments . . . 2
  - 1.3.3 Performance . . . 3

2 Model structure **4**
- 2.1 Coupler . . . 4
- 2.2 Ice model . . . 5
- 2.3 Data structure . . . 6
  - 2.3.1 Shared memory . . . 7
  - 2.3.2 Parallelisation . . . 8
- 2.4 Output . . . 8
  - 2.4.1 Text files . . . 8
  - 2.4.2 NetCDF files . . . 9
- 2.5 Computer resources . . . 9

3 Ice dynamics and thermodynamics **10**
- 3.1 Shallow Ice Approximation . . . 10
- 3.2 Shallow Shelf Approximation . . . 10
  - 3.2.1 Discretisation and solution . . . 10
  - 3.2.2 Grounding-line flux . . . 12
  - 3.2.3 Parallelisation . . . 13
- 3.3 Mass continuity . . . 14
- 3.4 Thermodynamics . . . 15
- 3.5 Physical properties . . . 17

4 Climate **18**
- 4.1 Temperature . . . 18
- 4.2 Precipitation . . . 19
- 4.3 Data structure . . . 21

5 Mass balance **23**
- 5.1 Surface mass balance . . . 23
- 5.2 Basal mass balance . . . 24

6 Glacial isostatic adjustment **25**
- 6.1 ELRA . . . 25
  - 6.1.1 Theory . . . 25
  - 6.1.2 Implementation . . . 25

7 Unstructured grid **26**
- 7.1 Basic concepts . . . 26
  - 7.1.1 Nomenclature . . . 26
  - 7.1.2 Voronoi cells and Delaunay triangulation . . . 26
- 7.2 Data structure . . . 28
- 7.3 Staggered mesh . . . 31
- 7.4 Mesh refinement . . . 33
- 7.5 Paralellisation . . . 36
- 7.6 Discretisation . . . 38
  - 7.6.1 First-order partial derivatives . . . 38
  - 7.6.2 Second-order partial derivatives . . . 39
  - 7.6.3 First-order derivatives on staggered vertices . . . 42
  - 7.6.4 Least-squares discretisation scheme . . . 43
  - 7.6.5 Convergence . . . 45
- 7.7 Remapping . . . 46
  - 7.7.1 Theory . . . 46
  - 7.7.2 Implementation . . . 47

8 Personal note **51**

9 References **52**

1 Introduction

1.1 Background

The Institute for Marine and Atmospheric research Utrecht (IMAU) has a long, rich history of ice-sheet modelling. In the early 2000's, GRICE (GReenland ICE-sheet model) was created: an SIA-only model for studying the evolution of the Greenland ice sheet during glacial cycles. Several years later this model was adapted for the Antarctic ice sheet, including a module that solved the SSA for floating ice shelves, thus becoming ANICE (ANtarctic ICE-sheet model). Around 2010 this was extended into the coupled model ANICE-SELEN, which included four copies of ANICE to simulate the large continental Pleistocene ice-sheets (North America, Eurasia, Greenland, and Antarctica), as well as the sea-level equation solver SELEN. In early 2021 ANICE was replaced by its spiritual successor IMAU-ICE; still a square-grid hybrid SIA/SSA model, but thoroughly cleaned up and restructured, so as to enable a new generation of PhD's and postdocs to use and develop it without having to first know all the details and peculiarities of the work of their numerous predecessors.

In the late 00's and early 10's, several studies were published that showed that the phenomenon of grounding-line migration was much more important for large-scale ice-sheet evolution then previously thought, particularly in Antarctica. They also showed that the increasingly common hybrid SIA/SSA ice models (like ANICE and IMAU-ICE) performed poorly at capturing this phenomenon, partly due to their often coarse resolution. The first few of these studies suggested that, in order to solve the problem, a resolution of 100 m or less was required. This is completely unachievable for palaeo-ice-sheet models; they must be able to simulate tens or sometimes hundreds of thousands of years, which is why researchers typically use resolutions of 10 - 40 km. However, several later studies developed clever "heuristic" solutions to this problem, which allowed them to use much coarser resolutions (ranging from 1 km to 20 km, depending on which study you believe) and still get good results. Aside from these numerical issues with resolution, it also became clear that small-scale topographical features like fjords and underwater hills could significantly affect large-scale ice-sheet dynamics, implying that even the "heuristic" models would still need a resolution of a few kilometres or less.

However, while 20 km is still manageable, 10 km is already pushing the limits of what can reasonably be done with a square-grid model, and the even finer numbers required for resolving bed topography are simply not feasible for long palaeo-simulations. The obvious solution to this, in our view, was not to use a square grid. The studies that investigated the resolution problem already noted that in order to get good results, a high resolution was required only at the grounding line, not everywhere on the ice sheet. This intuitively makes sense; surface curvature (the leading term in the truncation error of any first-order discretisation scheme) is highest near the grounding line, as are the strain rates in the ice. Also, since grounded ice is thinnest and fastest there, this is where the topographical features of the underlying bedrock become most important. By creating a grid that has a high resolution only at the grounding line and nowhere else, you can "skip" a lot of non-essential calculations in the interior (determining velocities for essentially stagnant ice), freeing up computation time for the rest of the model. This line of reasoning led to the inception of UFEMISM: the first palaeo-ice-sheet model to use a dynamic adaptive grid.

1.2 Objectives

When the development of UFEMISM first began, the following objectives were set:

- **Dynamic adaptive grid**: a key difference between palaeo-simulations and future projections is that the changes in ice-sheet geometry are much larger in the former. Since it is not feasible to create a single mesh that has a high resolution everywhere the grounding line might be at some point in time (which would defeat the purpose of the unstructured grid!), the mesh must be able to adapt to the ice sheet.
- **Flexibility**: a large part of palaeoglaciological research consists of inventing new features to add to ice-sheet models, just to see what they do. This is one of the main reasons why most palaeoglaciologists use simple square-grid models; not only are they fast enough for such long simulations, they're also very easy to build by hand, enabling the individual researcher to adapt the model to suit the needs of whatever project they're working on. We wanted UFEMISM to have that same flexibility.
- **Performance**: since the whole point of creating UFEMISM was to be able to run very long simulations at a high resolution without requiring excessive amounts of computer time, good computational performance was an important goal.

The first objective, that of the dynamic adaptive grid, immediately created the biggest challenge, since no other ice-sheet model in existence has this feature. The second objective, that of flexibility, led to the decision to create a dedicated mesh-generation algorithm for UFEMISM, rather than to build upon existing

external software. While there are several packages available, developed both academically and commercially, to generate meshes and solve equations (using finite elements or other methods), from the point of view of a palaeoglaciologist these are "black boxes": they work, but you won't know how, or how to make them do something else if you need them to. Since this conflicts strongly with the objective of flexibility, we decided not to rely on such external packages, but instead develop our own codes for mesh generation and PDE solving from scratch.

1.3 Status quo

1.3.1 Features

Currently (April 2021), the following features in UFEMISM are **operational**, in development, and *planned*:

- **Hybrid SIA/SSA ice dynamics**
- **Adaptive mesh**
- **Shared-memory (single node) MPI parallelisation**
- **Simultaneous, coupled simulation of four continental ice sheets**
- **Glacial index / inverse-$\delta^{18}O$ forcing**
- **Matrix-method climate model**
- **Insolation-temperature-based SMB**
- **Basic temperature-depth-based BMB**
- **ELRA GIA model**

- Coupling to sea-level equation model SELEN

- *Multi-node MPI parallelisation*
- *Englacial tracers ($CO_2, \delta^{18}O$)*
- *Improved thermodynamics (enthalpy-based, energy-conserving)*
- *Basal hydrology*
- *Eigencalving, cliff failure, shelf crevasse failure*
- *Improved SMB and BMB parameterisations*

1.3.2 Benchmark experiments

Analytical solutions Experiments reproducing the analytical solutions by Halfar (1981) and Bueler et al. (2005) are hard-coded into UFEMISM. Halfar (1981) derived an analytical solution for the SIA, for the case of a circular symmetric ice sheet lying on flat bedrock, having a uniform flow factor (i.e. no thermomechanical coupling), and zero mass balance. Bueler et al. (2005) extended this solution for a simple parameterised mass balance. The results of these experiments are presented by Berends et al. (2021), who show that UFEMISM reproduces the analytical solutions well across a range of resolutions.

EISMINT-1 The six experiments from the first EISMINT intercomparison exercise (Huybrechts et al., 1996) are all hard-coded into UFEMISM. These experiments verify the SIA and the (uncoupled) thermodynamics, using a schematic ice sheet and forcing. The results of these experiments are presented by Berends et al. (2021), who show that the ice sheet geometry and temperature match with the results of the other models in Huybrechts et al. (1996).

MISMIP A slightly modified version of the first MISMIP experiment (Pattyn et al., 2012) is hard-coded into UFEMISM. This experiment describes a schematic ice sheet lying on a coastal slope, so that a shelf forms. It is meant to investigate grounding-line migration under repeated forced advance-retreat cycles, using the hybrid SIA/SSA ice dynamics. The original experiment is a 1-D flowline set-up; following the approach of Pattyn (2017), it is adapted into a 2-D plan-view set-up by assuming the 1-D flowline is a radial transect of a circular symmetric ice-sheet. The results of this experiment are presented by Berends et al. (2021), who show that the grounding-line flux condition results in a grounding-line position that is independent of model resolution or forcing history (i.e. no hysteresis), matching the other models in Pattyn et al. (2012).

1.3.3 Performance

Currently (March 2021), a simulation of all four ice sheets over the entire last glacial cycle (120 kyr) with a grounding line resolution of 10 km takes about 40 wall clock hours.

2 Model structure

UFEMISM is intended mainly for simulations of glacial cycles. For this reason, it has the ability to simultaneously simulate the evolution of ice sheets in Greenland and Antarctica, as well as North America and Eurasia (where large ice sheets existed during the Pleistocene glacial cycles). It also includes an elaborate mix of modelled data and extrapolations for both climate and mass balance, as well as an ELRA GIA model (the coupling to the sea-level/GIA model SELEN is being worked on, but is currently not yet operational). UFEMISM has been designed from the start to include all these components, and the code/data structure has been created so as to make this easy to achieve.

2.1 Coupler

The Fortran90 program UFEMISM_PROGRAM.F90 contains the coupler: the overarching routine that calls the different regional ice-sheet models, updates the global climate forcing, and calls the sea-level equation solver SELEN (in progress). This is done in a loop using the coupler time step dt_coupling. The general structure of the coupler looks as follows:

```
PROGRAM UFEMISM_program

! Initialise the ice model regions
CALL initialise_model( [North America] )
CALL initialise_model( [Eurasia] )
CALL initialise_model( [Greenland] )
CALL initialise_model( [Antarctica] )

! Initialise the global climate forcing
CALL initialise_climate_forcing

t_coupling = t_start

DO WHILE (t_coupling < t_end)

        ! Run each ice model region from t_coupling to t_coupling + dt_coupling
        CALL run_model( [North America] )
        CALL run_model( [Eurasia] )
        CALL run_model( [Greenland] )
        CALL run_model( [Antarctica] )

        ! Advance the coupler time
        t_coupling = t_coupling + dt_coupling

        ! Update the global climate forcing
        CALL update_climate_forcing

        ! Solve the sea-level equation
        IF (time_since_last_SELEN > dt_SELEN) THEN
                CALL run_SELEN
        END IF

END DO

END PROGRAM UFEMISM_program
```

Note that, since we generally want to update the global climate more often than the sea-level equation, SELEN is given its own seperate time step. Also note that model initialisation has been separated from model running, a practise that we have adopted everywhere in the model.

2.2 Ice model

The main ice model of UFEMISM is contained in the subroutine RUN_MODEL in the module UFEMISM_MAIN_MODULE.F90. This subroutine is called from the coupler and is provided one of the model regions as an in/output argument. When called, it will run that model region up till the next coupling moment. The general structure of the ice model looks as follow:

```
SUBROUTINE run_model( [region] )

DO WHILE (region%time <= t_end)

        IF (do_GIA) run_ELRA_model( [region] )
       CALL update_ice_thickness( [region] )
       IF (mesh fitness < threshold) update_mesh( [region] )
       IF (do_SIA) CALL update_SIA_velocities( [region] )
       IF (do_SSA) CALL update_SSA_velocities( [region] )
       IF (do_climate) CALL run_climate_model( [region] )
       IF (do_SMB) CALL run_SMB_model( [region] )
       IF (do_BMB) CALL run_BMB_model( [region] )
       IF (do_thermo) CALL run_thermodynamics( [region] )
       IF (do_output) CALL write_output( [region] )
       region%time = region%time + region%dt

END DO

END SUBROUTINE run_model
```

An important feature of UFEMISM is the asynchronous time stepping of the different model components. All components have their own independent time step. The model time step REGION%DT is determined in every loop to equal the difference between the current model time, and the first time one of the components requires an update. This is visualised below.

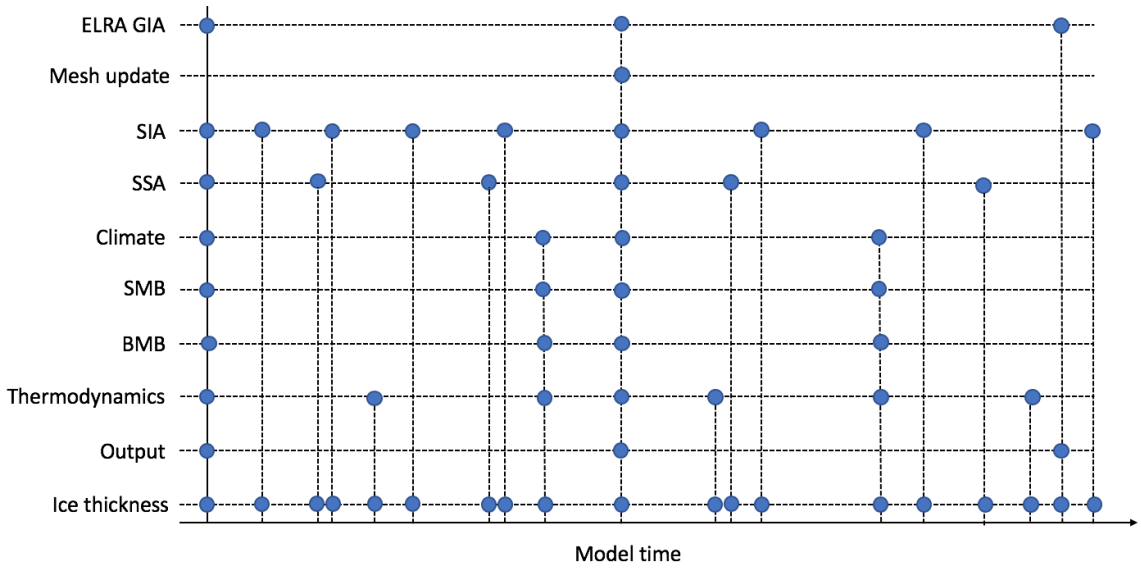


Figure 1: Asynchronous time stepping in UFEMISM.

The time steps dt_{SIA}, dt_{SSA} of the SIA and SSA are calculated dynamically using their own respective stability criteria (eqs. (4) and (17)). All the other model components have fixed time steps, which are set through the config. At the end of every model loop, the subroutine DETERMINE_TIMESTEPS_AND_ACTIONS determines which model component(s) should be run next, and advances the model time to that particular moment. The mesh fitness is checked after the ice-sheet geometry has been updated. If the fitness lies below the prescribed

fitness threshold, a mesh update is triggered. When this happens, only ice thickness and englacial temperature are remapped from the old to the new mesh; all other model data is calculated by running all of the model components.

The order of the different model components is such that output written at time t exactly represents the modelled ice thickness and velocity fields at that time. In order to achieve this, in the very first iteration of the first time RUN_MODEL is called, bedrock deformation, ice thickness, and englacial temperature are updated with a time step of zero. By the time output is written, the ice velocities, climate, SMB, and BMB are all given exactly for the initial model state at $t = 0$.

2.3 Data structure

Data fields in UFEMISM have been organised into Fortran90 TYPEs. This main advantages of this approach are that it makes the header sections of subroutines a lot smaller (since instead of passing 10 or 20 different arrays as arguments, only the type containing them needs to be passed), and that it makes it impossible to accidentally use different names for the same variables in different subroutines.

Different data fields related to the same model component have been grouped into several TYPEs called "submodels", such as TYPE_ICE_MODEL for ice dynamics and TYPE_SMB_MODEL for the surface mass balance. These submodels are grouped together into TYPE_REGION, which contains all the model data describing one of the four model regions. In this context, the mesh too is treated as a submodel.

- TYPE_REGION
  - TYPE_MESH
    - REAL(DP), DIMENSION(:,:) :: V
    - INTEGER, DIMENSION(:) :: nC
    - INTEGER, DIMENSION(:,:) :: C
    - INTEGER, DIMENSION(:) :: niTri
    - INTEGER, DIMENSION(:,:) :: iTri
    - ...
  - TYPE_ICE_MODEL
    - REAL(DP), DIMENSION(:) :: Hi
    - REAL(DP), DIMENSION(:) :: Hb
    - REAL(DP), DIMENSION(:) :: Hs
    - REAL(DP), DIMENSION(:,:) :: Ti
    - ...
  - TYPE_SMB_MODEL
    - REAL(DP), DIMENSION(:,:) :: Albedo
    - REAL(DP), DIMENSION(:,:) :: Snowfall
    - REAL(DP), DIMENSION(:,:) :: Rainfall
    - ...
  - ...

All these different data TYPEs are defined in the DATA_TYPES_MODULE. This prevents interdependency issues, and also keeps the modules containing the subroutines clean and easy to read.

The different submodels gathered in TYPE_REGION are:

- MESH (TYPE_MESH): All the data particular to the mesh: the basic mesh geometry and connectivity data described in Sect. XXX, as well as all the different neighbour functions.
- ICE (TYPE_ICE_MODEL): All the data required to calculate ice dynamics and thermodynamics: ice thickness, bedrock elevation, surface elevation, geoid elevation, partial derivatives to (x, y, z, t) of all of these, ice velocities (SIA and SSA), englacial temperature, temperature-dependent ice physical properties (thermal conductivity, heat capacity, flow factor), different masks, etc.
- CLIMATE (TYPE_CLIMATE_MODEL): All the different climate fields mapped to this regions grid, divided into "subclimates" containing present-day observations, GCM snapshots, and the final applied climate. This is explained in more detail in Sect. XXX.

- SMB (TYPE_SMB_MODEL): All the different components of the surface mass balance: snowfall, rainfall, melt, refreezing, runoff, albedo, etc.
- BMB (TYPE_BMB_MODEL): All the different components of the basal mass balance (currently only the yearly melt).
- PD (TYPE_PD_DATA_FIELDS): The present-day ice-sheet geometry, used to calculate different anomalies: surface load for the GIA model, change in surface elevation for the parameterised climate, etc. This data is stored both on the current mesh, and on the original x,y-grid on which it is provided (so that, after a mesh update, it can be projected onto the updated model mesh from its original grid, to prevent numerical diffusion from accumulating).
- INIT (TYPE_INIT_DATA_FIELDS): The initial ice-sheet geometry at the start of the simulation. Stored on both the mesh and the original data grid, just as the PD reference data.

2.3.1 Shared memory

The current version of UFEMISM has been parallelised using MPI shared memory, allowing the model to run in parallel on any number of cores that have direct access to the same memory chip. MPI uses its own routines to allocate shared memory and associate pointers with it. These have been wrapped in different routines in the PARALLEL_MODULE:

- ALLOCATE_SHARED_DP_0D
- ALLOCATE_SHARED_BOOL_1D
- ALLOCATE_SHARED_INT_2D
- ...

The naming convention of these routines is "ALLOCATE_SHARED_[TYPE]_[DIM]". With the exception of the 0D variants (which allocate scalars rather than arrays), they require the dimensions to be allocated as input arguments, and return two output arguments: a POINTER to the memory space (which is accepted by Fortran subroutines just like the array it points to), and an MPI WINDOW (really just an INTEGER) associated with this memory space. The WINDOW is needed to deallocate the memory later. The subroutine ALLOCATE_SHARED_INT_2D looks like this:

```
SUBROUTINE allocate_shared_int_2D( n1,n2,p,w)

! Dimension(s) of memory to be allocated
INTEGER, INTENT(IN) :: n1, n2

! Pointer to memory space
INTEGER, DIMENSION(:,:), POINTER, INTENT(OUT) :: p

! MPI window to the allocated memory space
INTEGER, INTENT(OUT) :: w

...
(allocate shared memory for the (n1-by-n2) array p
...

END SUBROUTINE allocate_shared_int_2D
```

The MPI windows generated by the ALLOCATE_SHARED routines are all stored in the same TYPEs that hold the variables the point to, and have the same name as the variable prefixed by the letter W. For example, ice thickness is stored in the type TYPE_ICE_MODEL like:

- REAL(DP), DIMENSION(:), POINTER :: HI
- INTEGER :: WHI

This shared memory can be deallocated by simply calling DEALLOCATE_SHARED, e.g.:
CALL DEALLOCATE_SHARED(ICE%WHI)

2.3.2 Parallelisation

All subroutines in UFEMISM, apart from those involved in mesh generation, are parallelised by simple domain decomposition. This is achieved by assigning to each processor a range of vertices that it is allowed to operate on. These are stored as non-shared integers v_1, v_2 in TYPE_MESH. For example, suppose that we wish to calculate the surface slope of the model region. This will then look something like this:

```
DO vi = mesh%v1, mesh%v2
  ice%dHs_dx( vi) = ddx( ice%Hs, vi)
END DO
CALL sync
```

If, for example, the mesh has 1000 vertices and the model is running on 4 processors. Then the vertex ranges for the different processors will be:

Processor	**v1**	**v2**
0	1	250
1	251	500
2	501	750
3	751	1000

These ranges are assigned during mesh generation using the PARTITION_LIST subroutine. Note that processors are indexed from 0; this is the MPI standard. Note also the CALL SYNC statement after the DO loop. This is simply a wrapper for the MPI_BARRIER command, which ensures synchronisation. Although the work load of each processor is nearly the same, one of them might still be finished a little faster than the others. The SYNC statement ensures that the program is only allowed to continue once each processor has finished its work and therefore the entire data field (in this case, ICE%DHS_DX) has been updated.

This kind of parallelisation works only for "embarassingly parallel" problems, where operations on an element on an array do not depend on operations on any other elements. This is the case for the vast majority of calculations done in the ice model. There are a few exceptions where great care must be taken to ensure that the program will behave properly. For example, the SOR iteration in the SSA needs to stop when the smallest residual velocity across the entire mesh has decreased below the specified tolerance. This must be checked after all processes have finished updating values in their range, and the smallest value must be found for the entire mesh. Another problem that is not easily parallelised is the flood-fill algorithm that is used to determine the ocean mask; instead, this is done only by the master process, while all other processors simply wait around.

Mesh generation is parallelised in a very different manner. This is explained in Sect. XXX.

2.4 Output

2.4.1 Text files

UFEMISM creates a number of ASCII text files containing scalar output data. These are given the prefix "AA_" so that they will always be the first to show up in the results directory (since long simulations with frequent mesh updates can produce a large number of NetCDF files, see next section).

The file "GENERAL_OUTPUT.TXT" lists some global scalars such as global mean sea level, atmospheric CO_2, deep-sea temperature, regional ice volume, contributions to the benthic $\delta^{18}O$ signal, etc.

The regional "GENERAL_OUTPUT" files list regional scalars such as ice-sheet area, volume, volume above flotation, mean surface temperature, and the integrated mass balance components.

The regional "TIME_LOG" files list the computation time used by the different model components used by each region.

Lastly, the Point-Of-Interest (POI) files list ice thickness and englacial temperature over time for the different POIs that have been specified through the config, to enable easy comparison with ice core data.

2.4.2 NetCDF files

Data fields are written to a number of NetCDF files. Each model region has its own set of output files, which are divided into "RESTART" and "HELP_FIELDS" files, and into "MESH" and "GRID" files.

RESTART files, as their name suggests, contain everything that is needed to restart a simulation (either in the event of a crash, or when it is convenient to split a long simulation into parts): ice thickness, bed topography, englacial temperature, the depth of the overlying firn layer, and the amount of melt that occurred during the previous year (used in the albedo parameterisation).

HELP_FIELDS files can be configured by the user to contain a wide selection of data fields. Ice velocities (surface, basal, 3D), mass balance components (yearly or monthly), climate fields (temperature, precipitation, wind; yearly or monthly), etc. A full list of all the data fields that can be chosen can be found in the routine CREATE_HELP_FIELD in the NETCDF_MODULE (where additional options can also easily be added).

GRID files contain the data on a regular square grid. The resolution of this grid can be specified through the config; UFEMISM internally generates this grid, and whenever output is written, it projects the specified model data to this grid and writes it to the grid output file. Note that here too, a distinction is made between RESTART and HELP_FIELDS files, but that the grid restart files cannot be used to restart the model. They are intended mainly for fast and easy output inspection, for example with NcView.

MESH files contain the data on the model mesh, as well as all the data describing that mesh. Since the mesh is irregularly updated during simulations, changing the number of vertices, this data cannot be contained within a single NetCDF file. Instead, whenever the mesh is updated, a new NetCDF file is created. The UFEMISM Github repository contains a number of Matlab scripts that can display mesh data. The MESH files also contain a list of vertex indices and weights for creating transects, which are very useful for benchmark experiments.

2.5 Computer resources

The following results were obtained with a "typical" simulation of the last glacial cycle (120 kyr, all four ice sheets) with a grounding-line resolution of 10 km, and 32 km for the ice margin, calving front, and coastline.

Region	**nV**
North America	8,000
Eurasia	6,000
Greenland	5,500
Antarctica	18,000

As was shown by Berends et al. (2021), the number of vertices scales with the desired resolution to about order 1.2 - 1.4. UFEMISM uses about 100 kB of RAM per vertex, adding up to a total of about 3.6 Gb for the above simulation (the maximum memory use during a simulation is written in the GENERAL_OUTPUT.TXT file).

Computation time depends on a number of factors, the most important one being (again) resolution. BLA

3 Ice dynamics and thermodynamics

UFEMISM uses the hybrid SIA/SSA approximation to the stress balance developed by Bueler and Brown (2009). In this approach, the SIA is used to determine deformational velocities over land, while the SSA provides both deformational velocities for floating ice, and sliding velocities over land. Here, we will briefly list the equations describing these approximations. Built into UFEMISM are options to run the EISMINT-I benchmark experiments (Huybrechts et al., 1996) for benchmarking the SIA and the thermodynamics, and a modified, 2D version of the first MISMIP experiment (Pattyn et al., 2012) for benchmarking the hybrid SIA/SSA. The results of these benchmark experiments are described in detail by Berends et al., 2021.

3.1 Shallow Ice Approximation

The SIA assumes that all velocity gradients inside the ice sheet are negligible except for the vertical gradients of the horizontal velocities. This assumption simplifies the Navier-Stokes equation to a form that expresses the horizontal ice velocities solely in terms of local quantities (ice thickness, surface gradient, ice viscosity). This means no differential equation has to be solved numerically to obtain the velocity field, which makes it a very computationally efficient approach. The general consensus is that the SIA is accurate for the interior of an ice sheet, but cannot accurately describe ice flow near the ice margin, near the grounding line, or in other areas where the local ice thickness is no longer negligibly small compared to the horizontal length scale of the local topography.

The SIA expresses the horizontal ice velocities $u(z), v(z)$ in the vertical column in terms of the depth-dependent ice diffusivity $D(z)$:

$$D(z) = 2(\rho g H)^n |\nabla H|^{n-1} \int_0^z A\left(T^*(\zeta)\right) \zeta^n d\zeta \tag{1}$$

$$u(z) = D(z) \frac{\partial h}{\partial x} \tag{2}$$

$$v(z) = D(z) \frac{\partial h}{\partial y} \tag{3}$$

The depth-dependent ice diffusivity $D(z)$ is defined as a function of the ice density ρ, gravitational acceleration g, ice thickness H, surface gradient ∇h, Glen's flow law exponent n, and the temperature-dependent ice flow factor $A\left(T^*(z)\right)$. For a comprehensive derivation of these equations, see e.g. Bueler and Brown (2009).

In most ice sheet models a staggered grid is used for either the ice velocities, defining ice thickness on the regular grid, or vice versa. Early experiments with SIA ice models have shown that this approach strongly increases numerical stability, so that much larger time steps can be used. In UFEMISM this is done using the staggered mesh described in Sect. XXX. The ice thickness H and the surface elevation h are defined on the regular mesh, whereas the surface gradient ∇h and the ice velocities u, v are defined on the staggered mesh. This also makes it very straightforward to calculate ice thickness change over time using a finite-volume approach (see Sect. XXX).

SIA ice velocities are updated with a dynamical time step according to the CFL-criterion, which relates the largest time step that does not cause numerical instability to the (local) resolution R and the SIA diffusivity D. For this calculation, these quantities are defined on the staggered vertices c:

$$dt_{SIA} < \min_c \frac{R_c^2}{6D} \tag{4}$$

3.2 Shallow Shelf Approximation

3.2.1 Discretisation and solution

The SSA assumes that all velocity gradients inside the ice sheet are negligible except for the horizontal gradients of the horizontal velocities. This means that the gravitational driving stress is balanced by the longitunal (or "membrane") stresses, plus a (small) basal shear stress for ice on land. A concrete form of this stress balance is given by Bueler and Brown (2009), based on the work by MacAyeal (1989) and Weis et al. (1999):

$$\frac{\partial}{\partial x}\left[2\overline{\eta}H\left(2\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right]+\frac{\partial}{\partial y}\left[\overline{\eta}H\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]-\frac{\tau_c u}{|\mathbf{u}|}=\rho g H\frac{\partial h}{\partial x}, \tag{5}$$

$$\frac{\partial}{\partial y}\left[2\overline{\eta}H\left(2\frac{\partial v}{\partial y}+\frac{\partial u}{\partial x}\right)\right]+\frac{\partial}{\partial x}\left[\overline{\eta}H\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right]-\frac{\tau_c v}{|\mathbf{u}|}=\rho g H\frac{\partial h}{\partial y}. \tag{6}$$

The first two terms on the left-hand sides of these equations describe the membrane stresses, in terms of the horizontal ice velocities u, v, the vertically averaged ice viscosity $\overline{\nu}$ and the ice thickness H. The last terms on the left-hand side describe the basal shear stress, related to the ice velocity through a Coulomb-type sliding law (where the magnitude of the stress is independent of the magnitude of the velocity). These stresses are balanced out by the driving stress, which is described on the right-hand sides in terms of the ice thickness H and the surface slopes h_x and h_y. The vertically averaged ice viscosity $\overline{\nu}$ is described by MacAyeal (1989) as a function of ice velocity:

$$\overline{\eta}=\frac{A}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2+\frac{\partial u}{\partial x}\frac{\partial v}{\partial y}+\frac{1}{4}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)^2\right]^{\frac{1-n}{2n}} \tag{7}$$

In UFEMISM, the SSA is further simplified following the approach by Determann (1991) and Huybrechts (1992), where the lateral variations of the effective strain rate $\frac{\partial \eta}{\partial x}, \frac{\partial \eta}{\partial y}$ are neglected. Determann (1991) and Huybrechts (1992) show that, since these terms are small compared to variations of the individual strain rates, this does not significantly affect the solution, while improving numerical stability and computational efficiency. Using this simplification, setting the basal shear stress $\tau_b = \frac{\tau_c}{|\mathbf{u}|}$, and using subscript notation for partial derivatives $u_x = \frac{\partial u}{\partial x}$ simplifies to:

$$4u_{xx}+u_{yy}+3v_{xy}-\frac{\tau_b u}{\overline{\eta}H}=\frac{\rho g h_x}{\overline{\eta}}, \tag{8}$$

$$4v_{yy}+v_{xx}+3u_{xy}-\frac{\tau_b v}{\overline{\eta}H}=\frac{\rho g h_y}{\overline{\eta}}. \tag{9}$$

These equations are discretised on the mesh using the neighbour functions derived in Sect. XXX:

$$u^i\left(4N^i_{xx}+N^i_{yy}-\frac{\tau^i_b}{\overline{\eta^i}H^i}\right)+3v^iN^i_{xy}+\sum_{c=1}^{n}\left[u^c\left(4N^c_{xx}+N^c_{yy}\right)+3v^cN^c_{xy}\right]=\frac{\rho g h^i_x}{\overline{\eta^i}}, \tag{10}$$

$$v^i\left(4N^i_{yy}+N^i_{xx}-\frac{\tau^i_b}{\overline{\eta^i}H^i}\right)+3u^iN^i_{xy}+\sum_{c=1}^{n}\left[v^c\left(4N^c_{yy}+N^c_{xx}\right)+3u^cN^c_{xy}\right]=\frac{\rho g h^i_y}{\overline{\eta^i}}. \tag{11}$$

This system of linear equations can be solved using any preferred method. In UFEMISM, this is done with successive over-relaxation (SOR). This approach was chosen because the boundary conditions of the SSA (i.e. ice-sheet geometry and viscosity) tend to change very slowly over time, so that the velocity field at time t is usually very similar to that at time $t+\delta t$. This means that an iterative solver using the solution from the previous time step as an initial guess will generally be more efficient than a direct solver. The choice for SOR as the particular iterative method is one of convenience, since it is very easy to implement without even having to construct a matrix representation of the system of equations.

Let a system of linear equations be of the general form $a_i f_i + \sum_{j\neq i} b_j f_j = c_i$. The SOR iteration scheme to find the solution f then looks like:

$$f_i^{k+1}=f_i^k-\omega\frac{a_i f_i^k+\sum_{j\neq i}b_j f_j^k-c_i}{a_i} \tag{12}$$

Applying this general form to the SSA yields the following SOR iteration scheme:

$$u_i^{k+1}=u_i^k-\frac{\omega}{e_{u,i}}\left(u_i^k e_{u,i}+3v_i^kN_{xy,i}+\sum_{c=1}^{n}\left[u_c^k\left(4N_{xx,c}+N_{yy,c}\right)+3v_c^kN_{xy,c}\right]-\frac{\rho g h_{x,i}}{\overline{\eta_i}}\right), \tag{13}$$

$$v_i^{k+1}=v_i^k-\frac{\omega}{e_{v,i}}\left(v_i^k e_{v,i}+3u_i^kN_{xy,i}+\sum_{c=1}^{n}\left[v_c^k\left(4N_{yy,c}+N_{xx,c}\right)+3u_c^kN_{xy,c}\right]-\frac{\rho g h_{y,i}}{\overline{\eta_i}}\right). \tag{14}$$

The centre coefficients $e_{u,i}, e_{v,i}$ are defined as:

$$e_{u,i} = 4N_{xx,i} + N_{yy,i} - \frac{\tau_{b,i}}{\overline{\eta_i} H_i}, \tag{15}$$

$$e_{v,i} = 4N_{yy,i} + N_{xx,i} - \frac{\tau_{b,i}}{\overline{\eta_i} H_i}. \tag{16}$$

Currently, no boundary conditions are applied at the calving front. Instead, following the approach by Bueler and Brown (2009), a nearly-infinitely thin (10 cm) ice shelf extends all the way to the model domain, where Neumann boundary conditions are applied.

The over-relaxation parameter ω is currently set to $\omega = 1.1$. Higher values generally yield faster convergence, but risk numerical instability. Preliminary experiments indicate that in this case, divergence tends to occur at specific vertices, and that using a spatially variable value of ω (which is low only at these vertices and high elsewhere) can significantly decrease the number of iterations required to find an accurate solution. However, determining rules for how ω is to be defined as a function of mesh geometry such that numerical instability is guaranteed to never occur, has turned out to be difficult, and a robust implementation of this solution is left as a future project.

SSA ice velocities are updated with a dynamical time step according to the CFL-criterion, which relates the largest time step that does not cause numerical instability to the (local) resolution R and the SSA ice velocities u, v D. For this calculation, these quantities are defined on the staggered vertices c:

$$dt_{SSA} < \min_c \frac{R_c}{|u_c| + |v_c|} \tag{17}$$

3.2.2 Grounding-line flux

Solving the discretised equations of the SSA is not particularly difficult. However, when the resulting velocity field is used to integrate the ice-sheet model through time, several problems become apparent in the solution. This is particularly obvious in experiments where a periodic step-wise forcing is applied, such that the grounding line responds (or rather, should respond) with repeated advance-retreat cycles. Early work by Weertman, as well as later work by Schoof, showed that the SSA predicts a unique steady-state solution for a given bed geometry and mass balance. However, Pattyn et al., (2012) demonstrated that numerical ice-sheet models that solve the SSA produce significant hysteresis in these experiments. Furthermore, the magnitude of the hysteresis often depends strongly on model resolution, and vanishes only when model resolution (at least at the grounding line) is ¡ 100 m (a value that is completely unfeasible for palaeosimulations). A solution to this problem that has become common practise is to apply a "semi-analytical solution" to the grounding-line flux as some sort of boundary condition to the SSA, or to the mass continuity integration. Schoof (2007) and Tsai et al. (2015) produced such semi-analytical solutions for the cases of Weertman-type and Coulomb-type sliding laws, respectively, relating the flux at the grounding line to the ice thickness at the grounding line, regardless of the geometry of the rest of the ice sheet.

Since UFEMISM uses a (regularised) Coulomb-type sliding law, the solution by Tsai et al. (2015) is used. Rather than using this to "overrule" the ice flux in the ice mass continuity integration, as is sometimes done (which requires some sort of heuristic to choose where to apply the flux, and thereby violates conservation of mass), we apply it directly as a boundary condition to the SSA velocity field.

The semi-analytical solution by Tsai et al. (2015) reads:

$$q_g = Q_0 \frac{8A(\rho_i g)^n}{4^n tan\phi} \left(1 - \frac{\rho_i}{\rho_w}\right)^{n-1} H_g^{n+2} \tag{18}$$

Here, the ice flux at the grounding line q_g is related to the ice flow factor A, the till friction angle ϕ (used in the Coulomb sliding law), and the ice thickness at the grounding line H_g. $Q_0 = 0.61$ is a tuning factor derived by Tsai et al. (2015).

UFEMISM solves the SSA on both the regular and the staggered mesh simultaneously (see Sect. XX). At grounding-line vertices (defined as staggered vertices separating a floating and a non-floating pixel), the semi-analytical solution is calculated. To get accurate values, the ice flow factor A, the till friction angle ϕ and the ice thickness H are evaluated at the sub-grid grounding line position by linearly interpolating the thickness above flotation $T = H - (S - b)\,\frac{\rho_w}{\rho_i}$ to zero. The resulting grounding-line flux q_g is divided by the ice thickness H_g

to find the magnitude u of the ice velocity. The direction of the ice velocity is assumed to be perpendicular to the grounding-line, which means it is anti-parallel to the gradient of the thickness above flotation. This yields separate values for u and v, which are applied to the grounding-line pixels and subsequently left unchanged during the SOR iteration that solves the SSA.

This solution is only valid when there is no significant buttressing. While this condition holds for the schematic MISMIP_mod benchmark experiment for which UFEMISM is validated, it does not hold for most of the actual Antarctic grounding line. Different approaches have been proposed to "adapt" this solution for buttressing, which have so far achieved very limited success. No solution for buttressing has so far been implemented in UFEMISM. This means that realistic experiments can only be run by setting the config option USE_ANALYTICAL_GL_FLUX to FALSE.

3.2.3 Parallelisation

The SOR solver used to solve the SSA is parallelised using a five-colour scheme. This is similar to the red-black scheme commonly used on square grids. Since SOR calculates the new solution f_i^{k+1} on vertex i based on the old solution f^k on i and the direct neighbours of i, problems can occur when two processors attempt to update values on two connected vertices at the same time. To prevent this, the five-colour theorem is used:

The vertices of any planar graph can be coloured such that no two connected vertices are ever of the same colour, using at most five different colours.

In this context, a square grid presents a special case of a planar graph where only 2 colours are necessary: a red-black scheme. For the unstructured mesh used in UFEMISM, five colours are sufficient (the stronger *four-colour theorem* has been boggling the minds of mathematicians for almost two centuries; although the consensus now is that it is true, no easily implementable algorithms to calculate four-colourings for a mesh are yet available, so for now we shall stick to five colours). The five-colouring of a mesh is calculated using the algorithm presented by Williams (1985), which is easy to implement, uses very little memory, and is fast enough that it requires negligibly little computation time compared to the rest of the mesh generation routines.

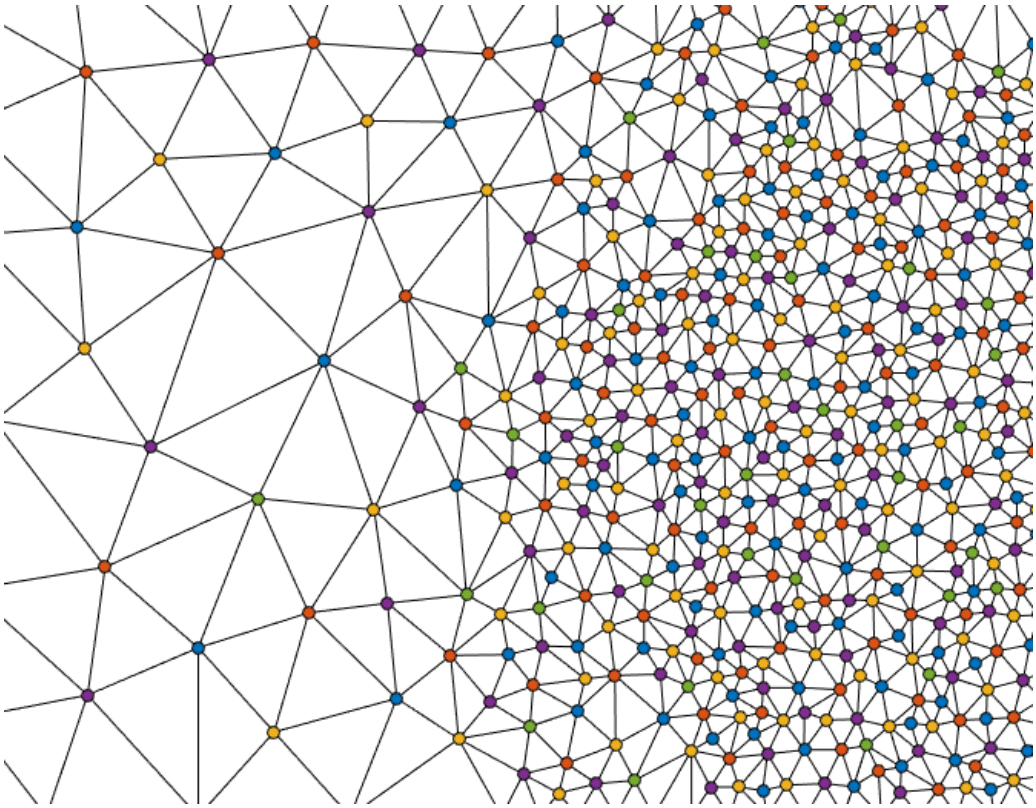


Figure 2: Part of a mesh, showing the five-colouring scheme.

The colours are stored in mesh data as two arrays:

- COLOUR: the colour of each vertex (1 - 5)

- COLOUR_VI: list of vertices of each colour (size nV-by-5)

There are nV_c vertices of each colour. These five groups of vertices are all partitioned over the processors. In each SOR iteration, an outer loop loops over the five colours, while an inner loop loops of the range of vertices of that colour assigned to each processor:

```
! Loop over the five colours
DO ci = 1, 5

   ! Find process domains for this colour
   vii1 = mesh%colours_v1( ci)
   vii2 = mesh%colours_v2( ci)

   ! Loop over all vertices of this colours assigned to this process
   DO vii = vii1, vii2

      vi = mesh%colours_vi( vii,ci)

      ! Calculate new solution for this vertex using SOR
      ...
      ...
      !

   END DO
   CALL sync

END DO
```

3.3 Mass continuity

In UFEMISM, ice thickness changes over time are calculated using a finite volume approach, which is conceptually very similar to the more commonly used combination of finite differencing with a staggered grid. Consider the conservation law for flowing ice, equating the time derivative of the ice thickness H to the (two-dimensional) divergence of the ice flux (being the product of H and the vertically averaged ice velocity $\mathbf{u}$), and the mass balance M:

$$\frac{\partial H}{\partial t} = -\nabla \cdot (\mathbf{u}H) + M \tag{19}$$

By applying the divergence theorem, this can be rewritten as:

$$\frac{\partial \overline{H_\Omega}}{\partial t} = \left[\frac{-1}{A_\Omega} \oint_{\partial\Omega} (\mathbf{u}H \cdot d\hat{\mathbf{n}})\right] + \overline{M} \tag{20}$$

Here, Ω is some arbitrary 2-D region (the control volume after which the finite volume approach is named), enclosed by the 1-D curve $\partial\Omega$ with outward unit normal vector $\hat{\mathbf{n}}$. The unstructured triangular mesh partitions the 2-D domain into Voronoi cells, which function as the control volumes (see).

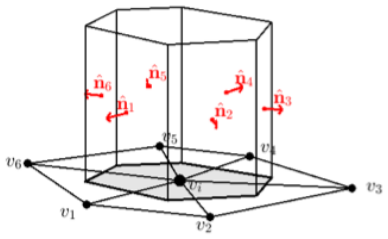


Figure 3: The Voronoi cell (grey) of a vertex serves as the control volume in the finite volume approach. Ice flows through the vertical faces of the volume, while the surface and basal mass balance add or remove ice from the top and bottom faces, respectively

The conservation law for a single Voronoi cell reads:

$$\frac{\partial \overline{H_i}}{\partial t} = \left[\frac{-1}{A_i} \sum_{c=1}^{n} \int_{\partial_c} (\mathbf{u}_c H_c \cdot d\hat{\mathbf{n}}_\mathbf{c}) \right] + \overline{M_i} \tag{21}$$

Here, the Voronoi cell of vertex i shares the boundary ∂_c with that of neighbouring vertex c. The equation is then discretised in space by assuming that the ice velocity $\mathbf{u}$ and the ice thickness H are constant on ∂_c, so that the line integral becomes a simple multiplication with the length L_c of the shared boundary ∂_c:

$$\frac{\partial \overline{H_i}}{\partial t} = \left[\frac{-1}{A_i} \sum_{c=1}^{n} (L_c \mathbf{u}_c H_c \cdot d\hat{\mathbf{n}}_\mathbf{c}) \right] + \overline{M_i} \tag{22}$$

Lastly, the equation is discretised explicitly in time:

$$H_i^{t+\Delta t} = H_i^t + \Delta t \left(\left[\frac{-1}{A_i} \sum_{c=1}^{n} (L_c \mathbf{u}_c H_c \cdot d\hat{\mathbf{n}}_\mathbf{c}) \right] + \overline{M_i} \right) \tag{23}$$

In order to solve this equation, we need to know the ice velocities $\mathbf{u}_c$ on the Voronoi cell boundaries. Both the SIA and the SSA are therefore solved on the staggered vertices described earlier. The staggered ice thickness H_c is determined using an up-wind scheme ($H_c = H_i$ if ice flows from i to j, $H_c = H_j$ if it flows fromj to i). This means that the finite volume approach is essentially identical to the "mass-conserving up-wind finite difference scheme" used in PISM (Winkelmann et al., 2011), and very similar to the combination of finite differences with a staggered grid used in many other ice-sheet models.

A simple flux-correction scheme is applied to prevent negative ice thicknesses from occurring. After the fluxes across all Voronoi cell boundaries have been calculated, a check is done to find vertices where $H_i^{t+\Delta t} < 0$. There, all outgoing fluxes are reduced by a scaling factor so that $H_i^{t+\Delta t} = 0$. Since this rescaling is applied to the staggered fluxes before the final ice thickness update is applied, this scheme is still mass-conserving.

3.4 Thermodynamics

UFEMISM uses the approach developed for SICOPOLIS (Greve et al., 1997) to solve the heat equation in a flowing medium:

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c_p} \nabla^2 T - \mathbf{u} \cdot \nabla T + \frac{\Phi}{\rho c_p}. \tag{24}$$

Here, the first term $\frac{k}{\rho c_p} \nabla^2 T$ represents diffusion, the second term $\mathbf{u} \cdot \nabla T$ represent advection, and the third term $\frac{\Phi}{\rho c_p}$ represents internal heating. For ice on land, internal heating is related to the vertical shear strain rates:

$$\Phi = 2 \left(\dot{\epsilon_{xz}} \tau_{xz} + \dot{\epsilon_{yz}} \tau_{yz} \right). \tag{25}$$

Furthermore, since the horizontal dimensions of the ice sheet are so much larger than the vertical dimension, horizontal diffusion of heat is neglected, simplifying the heat equation to:

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c_p} \frac{\partial^2 T}{\partial z^2} - \mathbf{u} \cdot \nabla T + \frac{\Phi}{\rho c_p}. \tag{26}$$

This equation is discretised on an irregular grid in the vertical direction. All vertical derivatives are discretised implicitly, whereas horizontal derivatives are discretised explicitly. This mixed approach, which was first used in the ice-sheet model SICOPOLIS (Greve, 1997), is numerically stable (since both the steepest gradients and shortest grid distances are in the vertical direction), relatively easy to implement, and fast to compute. Using 15 layers in the vertical direction, a time step of 10 years is typically sufficient to maintain numerical stability, independent of the horizontal resolution.

In order to calculate a depth-dependent temperature distribution, a vertical spatial discretisation is required. Following the approach used by many ice-sheet models, we adopt a scaled vertical coordinate:

$$\zeta = \frac{h - z}{H} \tag{27}$$

This guarantees that the top and bottom of the vertical ice column always coincide with the first and last vertical grid point, 5 respectively. This coordinate transformation results in the appearance of a few extra terms in the heat equation:

$$\frac{\partial T}{\partial t}+\frac{\partial T}{\partial \zeta}\frac{\partial \zeta}{\partial t}=\frac{k}{\rho c_p}\frac{\partial^2 T}{\partial \zeta^2}\left(\frac{\partial \zeta}{\partial z}\right)^2-u\left(\frac{\partial T}{\partial x}+\frac{\partial T}{\partial \zeta}\frac{\partial \zeta}{\partial x}\right)-v\left(\frac{\partial T}{\partial y}+\frac{\partial T}{\partial \zeta}\frac{\partial \zeta}{\partial y}\right)-w\left(\frac{\partial T}{\partial \zeta}\frac{\partial \zeta}{\partial z}\right)+\frac{\Phi}{\rho c_p}. \tag{28}$$

The different spatial derivatives of ζ follow from (27):

$$\frac{\partial \zeta}{\partial t}=\frac{1}{H}\left(\frac{\partial h}{\partial t}-\zeta\frac{\partial H}{\partial t}\right), \tag{29}$$

$$\frac{\partial \zeta}{\partial x}=\frac{1}{H}\left(\frac{\partial h}{\partial x}-\zeta\frac{\partial H}{\partial x}\right), \tag{30}$$

$$\frac{\partial \zeta}{\partial y}=\frac{1}{H}\left(\frac{\partial h}{\partial y}-\zeta\frac{\partial H}{\partial y}\right), \tag{31}$$

$$\frac{\partial \zeta}{\partial z}=\frac{-1}{H}. \tag{32}$$

The partial derivatives $\frac{\partial T}{\partial \zeta}, \frac{\partial^2 T}{\partial \zeta^2}$ can be discretised on the irregular vertical grid using Taylor expansions, yielding expressions of the form:

$$\frac{\partial T^k}{\partial \zeta}=a_\zeta T^{k-1}+b_\zeta T^k+c_\zeta T^{k+1}, \tag{33}$$

$$\frac{\partial^2 T^k}{\partial \zeta^2}=a_{\zeta\zeta} T^{k-1}+b_{\zeta\zeta} T^k+c_{\zeta\zeta} T^{k+1}. \tag{34}$$

Note that these expressions are very similar to the neighbour functions we saw earlier. We will not include their derivation here. Using these expressions to discretise the vertical derivatives of T, and using an implicit time-discretisation for the vertical derivatives, yields:

$$\begin{aligned}\frac{T_k^{t+\Delta t}-T_k^t}{\Delta t}+&\frac{\partial \zeta}{\partial t}\left(a_\zeta T_{k-1}^{t+\Delta t}+b_\zeta T_k^{t+\Delta t}+c_\zeta T_{k+1}^{t+\Delta t}\right)=\\ &\frac{k}{\rho c_p H^2}\left(a_{\zeta\zeta} T_{k-1}^{t+\Delta t}+b_{\zeta\zeta} T_k^{t+\Delta t}+c_{\zeta\zeta} T_{k+1}^{t+\Delta t}\right)\\ &-u\left[\frac{\partial \zeta}{\partial x}\left(a_\zeta T_{k-1}^{t+\Delta t}+b_\zeta T_k^{t+\Delta t}+c_\zeta T_{k+1}^{t+\Delta t}\right)+\frac{\partial T}{\partial x}\right]\\ &-v\left[\frac{\partial \zeta}{\partial y}\left(a_\zeta T_{k-1}^{t+\Delta t}+b_\zeta T_k^{t+\Delta t}+c_\zeta T_{k+1}^{t+\Delta t}\right)+\frac{\partial T}{\partial y}\right]\\ &-w\left[\frac{-1}{H}\left(a_\zeta T_{k-1}^{t+\Delta t}+b_\zeta T_k^{t+\Delta t}+c_\zeta T_{k+1}^{t+\Delta t}\right)\right]+\frac{\Phi}{\rho c_p}.\end{aligned} \tag{35}$$

This can be rearranged to read:

$$\begin{aligned}&T_{k-1}^{t+\Delta t}\left[a_\zeta\left(\frac{\partial \zeta}{\partial t}+u\frac{\partial \zeta}{\partial x}+v\frac{\partial \zeta}{\partial y}-\frac{w}{H}\right)-\frac{a_{\zeta\zeta}k}{\rho c_p H^2}\right]\\ &+T_k^{t+\Delta t}\left[b_\zeta\left(\frac{\partial \zeta}{\partial t}+u\frac{\partial \zeta}{\partial x}+v\frac{\partial \zeta}{\partial y}-\frac{w}{H}\right)-\frac{b_{\zeta\zeta}k}{\rho c_p H^2}-\frac{1}{\Delta t}\right]\\ &+T_{k+1}^{t+\Delta t}\left[c_\zeta\left(\frac{\partial \zeta}{\partial t}+u\frac{\partial \zeta}{\partial x}+v\frac{\partial \zeta}{\partial y}-\frac{w}{H}\right)-\frac{c_{\zeta\zeta}k}{\rho c_p H^2}\right]\\ &=\frac{T_k^t}{\Delta t}-u\frac{\partial T}{\partial x}-v\frac{\partial T}{\partial y}+\frac{\Phi}{\rho c_p}.\end{aligned} \tag{36}$$

If the vertical direction is discretised into n unevenly spaced layers, this system of equations can be represented by the matrix equation $AT^{t+\Delta t}=\delta$, where the lower diagonal α, central diagonal β and upper diagonal γ of the tridiagonal n-by-n matrix A and the right-hand side vector δ are given by:

$$\alpha = a_\zeta \left(\frac{\partial \zeta}{\partial t} + u \frac{\partial \zeta}{\partial x} + v \frac{\partial \zeta}{\partial y} - \frac{w}{H} \right) - \frac{a_{\zeta\zeta} k}{\rho c_p H^2}, \tag{37}$$

$$\beta = b_\zeta \left(\frac{\partial \zeta}{\partial t} + u \frac{\partial \zeta}{\partial x} + v \frac{\partial \zeta}{\partial y} - \frac{w}{H} \right) - \frac{b_{\zeta\zeta} k}{\rho c_p H^2} - \frac{1}{\Delta t}, \tag{38}$$

$$\gamma = c_\zeta \left(\frac{\partial \zeta}{\partial t} + u \frac{\partial \zeta}{\partial x} + v \frac{\partial \zeta}{\partial y} - \frac{w}{H} \right) - \frac{c_{\zeta\zeta} k}{\rho c_p H^2}, \tag{39}$$

$$\delta = \frac{T_k^t}{\Delta t} - u \frac{\partial T}{\partial x} - v \frac{\partial T}{\partial y} + \frac{\Phi}{\rho c_p}. \tag{40}$$

This matrix equation can be solved for every individual grid cell independently, making this an embarrassingly parallel problem. In UFEMISM, this is done with the Fortran package LAPACK; in Matlab, it can be done with the backslash method: T = A \delta. We apply a Dirichlet boundary condition at the top of the column, keeping ice temperature equal to surface air temperature (limited to melting point). At the base, a Neumann boundary condition is applied, keeping the vertical temperature gradient fixed to a value dictated by the geothermal heat flux and the frictional heating from sliding. Ice temperature throughout the vertical column is limited to the depth-dependent pressure melting point (a non-energy-conserving approach).

3.5 Physical properties

The ice flow factor A, specific heat C_p, and thermal conductivity K are calculated based on parameterised relations to the englacial temperature T.

The flow factor A is parameterised using an Arrhenius relation, following Huybrechts (1992):

$$A = A_0 e^{\frac{-Q}{R \cdot T}} \tag{41}$$

$$A_0 = \begin{cases} 1.14 \cdot 10^{-5} & T < 263.15 \\ 5.47 \cdot 10^{10} & T \geq 263.15 \end{cases} \tag{42}$$

$$Q = \begin{cases} 6.0 \cdot 10^4 & T < 263.15 \\ 13.9 \cdot 10^4 & T \geq 263.15 \end{cases} \tag{43}$$

Here, $R = 8.314$ J mol^{-1} K^{-1} is the gas constant. Before being used in the SIA and SSA, the flow factor A is multiplied by an enhancement factor e, with different values used in the SIA and the SSA. This can be interpreted as a correction for the (overly) simplified parameterised expression for the flow factor, wth the different values representing different degrees of anisotropy present in slowly deforming land ice and fast-flowing floating ice. The default values are $e_{SIA} = 5.0$, $e_{SSA} = 0.7$, which are taken from ANICE, (based on tuning experiments by Bas de Boer, which seem to agree with literature values, e.g. Ma et al., 2017).

The specific heat C_p is calculated according to Pounder (1965), and the thermal conductivity K according to Ritz (1987):

$$C_p = 2115.3 + 7.79293 \, (T - T_0) \tag{44}$$

$$K = 3.101 \cdot 10^8 \cdot e^{-0.0057 \cdot T} \tag{45}$$

4 Climate

UFEMISM's climate model is the same as that in IMAU-ICE, which in turn is same as in its predecessor ANICE2.1. It uses the matrix method developed by Berends et al. (2018). Quoting that paper: "A climate matrix, as defined by Pollard (2010), is a collection of output data from different steady-state GCM simulations that differ from each other in one or more key parameters or boundary conditions, such as prescribed atmospheric pCO_2, orbital configuration, or ice-sheet configuration. At every point in time during the simulation, the location of the model state within this matrix is extracted from the matrix by interpolating between its constituent precalculated climate states."

Berends et al. (2018) developed a way to use modelled ice-sheet geometry as a spatially variable interpolant between two GCM snapshots, respectively describing the PI and the LGM (Singarayer and Valdes, 2010). The ice-albedo feedback and the orographic forcing of precipitation are parameterised by this interpolation, resulting in a modelled ice-sheet geometry at LGM that agrees better with geomorphological evidence than was the case with earlier versions of the IMAU ice-sheet models. Since temperature and precipitation (which together constitute the entirety of the "climate" that's needed to calculate the surface mass balance) are affected by ice-sheet geometry in different ways, two separate interpolation schemes are used for these quantities.

4.1 Temperature

The equations described here are contained in the subroutine RUN_CLIMATE_MODEL_MATRIX_PI_LGM_TEMPERATURE of the CLIMATE_MODULE.

During the LGM, temperatures in the high latitudes of the northern hemisphere were between 10 and 15 K colder than pre-industrial, much more than the global mean cooling of 5 K. This temperature change has three main causes: a lower CO2 concentration in the atmosphere (global), a much higher albedo (regional), and a much higher surface elevation (local).

The temperature interpolation scheme of the climate matrix is based on the absorbed insolation I_{abs}, which is defined as the product of insolation at the top of the atmosphere Q_{TOA} with the inverse of the surface albedo a:

$$I_{abs}(x,y) = \sum_{m=1}^{12} Q_{TOA}(x,y,m) \cdot (1 - a(x,y,m)) \tag{46}$$

The absorbed insolation is calculated for the modelled ice-sheet geometry at time t using the insolation at the top of the atmosphere at that point in time, using the solution by Laskar et al. (2004). Absorbed insolation fields are also calculated for the two GCM snapshots, so that we have three fields: $I_{abs,mod}$, $I_{abs,PI}$, $I_{abs,LGM}$. For all three, the albedo a is modelled with UFEMISM's own SMB model. The insolation-based weighting field $w_{ins}(x,y)$ is calculated by scaling $I_{abs,mod}$ between the two reference fields:

$$w_{ins}(x,y) = \frac{I_{abs,mod}(x,y) - I_{abs,LGM}(x,y)}{I_{abs,PI}(x,y) - I_{abs,LGM}(x,y)} \tag{47}$$

To account for the fact that a change in albedo will affect temperatures not only locally but also regionally, a Gaussian smoothing filter F with a smoothing radius of 200 km and a region-wide average value are added to find the weighting field w_{ice}:

$$w_{ice}(x,y) = \frac{1}{7} w_{ins}(x,y) + \frac{3}{7} F\left(w_{ins}(x,y)\right) + \frac{3}{7} \overline{w_{ins}(x,y)} \tag{48}$$

NOTE: preliminary attempts at writing a proper (i.e. conservative, monotonic, efficient) smoothing algorithm on an unstructured mesh have not been successful. Instead, a fixed square grid is included for these operations. The weighting field w_{ins} is mapped from the mesh to the square grid, smoothing is applied on the grid, and the smoothed field is mapped back to the mesh. The two mapping operations generate a certain amount of numerical diffusion, which fortunately is not a problem for a smoothing operation.

Lastly, the atmospheric CO_2 concentration is included to yield the final weighting field w_{tot}:

$$w_{CO_2} = \frac{CO_2 - 190}{280 - 190} \tag{49}$$

$$w_{tot}(x,y) = \frac{w_{CO_2} + w_{ice}(x,y)}{2} \tag{50}$$

This weighting field is then used to interpolate between the two GCM snapshots:

$$T_{ref,GCM}(x,y) = w_{tot}(x,y) \cdot T_{PI}(x,y) + (1 - w_{tot}(x,y)) \cdot T_{LGM}(x,y) \tag{51}$$

Since the native resolution of the GCM snapshots is much lower than that of the ice-sheet model, this interpolated temperature is still very "smooth". To correct for this, an elevation-based downscaling scheme is used. First, the reference orography of the GCM snapshots is interpolated with the same weighting field. Then, the GCM-derived lapse rate is derived to find the ice-sheet model temperature:

$$h_{ref,GCM}(x,y) = w_{tot}(x,y) \cdot h_{PI}(x,y) + (1 - w_{tot}(x,y)) \cdot h_{LGM}(x,y) \tag{52}$$

$$T_{mod}(x,y) = T_{ref,GCM} + \lambda_{GCM}(x,y) \cdot (h_{mod}(x,y) - h_{ref,GCM}(x,y)) \tag{53}$$

Rather than using a constant lapse rate (typical values range from 6 to 8 K / km), a GCM-derived, spatially variable lapse rate $\lambda_{GCM}(x,y)$ is calculated from the temperature and orography fields of the two GCM snapshots:

$$\lambda_{LGM}(x,y) = \frac{T_{LGM}(x,y) - (T_{PI}(x,y) + \Delta T_{LGM})}{h_{LGM}(x,y) - h_{PI}(x,y)} \tag{54}$$

The temperature offset ΔT_{LGM} is the mean difference in GCM-calculated temperature between the LGM and PI fields over the ice-free area in the respective model region (either North America or Eurasia) at the LGM. The rationale behind this approach is explained in detail by Berends et al. (2018); in short, it ensures that, during the inception phase of the glacial cycle, the ice-free parts of the continent experience an appropriate amount of cooling so that the ice-sheet can expand.

4.2 Precipitation

The equations described here are contained in the subroutine RUN_CLIMATE_MODEL_MATRIX_PI_LGM_PRECIPITATION of the CLIMATE_MODULE.

At present, Antarctica is the driest desert on Earth, with the interior of East Antarctica receiving barely a centimetre of precipitation per year. The two main reasons for this are the extreme cold, which strongly reduces the moisture carrying capacity of the atmosphere, and the geometry of the ice-sheet. The dome shape of the Antarctic (or indeed any) ice sheet results in relatively steep slopes at the margin. As the wind blows moist air from the sea towards the ice sheet, the air has to move up this slope. Moving upwards means the pressure and temperature decrease, reducing the moisture carrying capacity, and so most of the moisture precipitates onto the ice-sheet margin. Any air that reaches the plateau of the ice-sheet interior is very dry, and so hardly and precipitation occurs there. This "plateau desert" effect is also expected to have existed over the North American and Eurasian ice sheets during the LGM.

This geometry-induced precipitation change is captured in the matrix method by using a different way to calculate the weighting field. Instead of absorbed insolation, ice thickness is used an interpolant:

$$w_{ice}(x,y) = \frac{H_{mod}(x,y) - H_{PI}(x,y)}{H_{LGM}(x,y) - H_{PI}(x,y)} \cdot \left(\frac{V_{mod} - V_{PI}}{V_{LGM} - V_{PI}} \right)^2 \tag{55}$$

Here, $H(x,y)$ is the local ice thickness, while $V = \iint H dA$ is the total regional ice volume. Note that (55) is slightly different from Eq. 12 in Berends et al. (2018); the version in the paper is wrong, in the actual model code the scaled local ice thickness is multiplied with the square of the scaled ice volume.

The final weighting field $w_{tot}(x,y)$ is obtained by applied a Gaussian smoothing filter F with a radius of 200 km:

$$w_{tot}(x,y) = 1 - F(w_{ice}(x,y)) \tag{56}$$

Note that the weight is inverted so that a value of 1 corresponds to PI conditions.

The GCM precipitation fields are then interpolated with this weighting field:

$$P_{ref,GCM}(x,y) = w_{tot}(x,y) \cdot P_{PI}(x,y) + (1 - w_{tot}(x,y)) \cdot P_{LGM}(x,y) \tag{57}$$

As with the temperature, the resulting precipitation is still very "smooth" because of the low GCM resolution. Here too, an orography-based downscaling scheme is used to transform the precipitation from the smooth reference orography to the "fine" model orography. This is done using the temperature/orography precipitation model developed by Roe and Lindzen (2001), which models precipitation as a function of surface temperature, wind speed and direction, and surface slope (parameterising both the effects of temperature on atmospheric moisture carrying capacity, and of orographic forcing of precipitation).

$$e_{sat} = e_0 e^{\frac{c_1(T-T_0)}{c_2+T-T_0}} \tag{58}$$

$$w_{vv} = max(0, u_{wind}\frac{\partial h}{\partial x} + v_{wind}\frac{\partial h}{\partial y}) \tag{59}$$

$$f(w'_{vv}) = \frac{1}{N} e^{-\left(\frac{w'_{vv}-w_0}{\alpha}\right)^2} \tag{60}$$

$$dP_{RL} = e_{sat} max(0, a + bw'_{vv}) f(w'_{vv}) dw'_{vv} \tag{61}$$

Here, e_{sat} is the saturation vapour pressure at the surface, which is a good proxy for the moisture content of the overlying air column. It is described by a Clausius-Clapeyron relation using the monthly mean surface temperature T, where e_0 = 6.112 mbar, c_1 = 17.67, and c_2 = 243.5 K. The vertical wind velocity w_{vv} is calculated from the 850 hPa wind $[u_{wind}, v_{wind}]$and the surface gradient. The precipitation P_{RL} is related to vertical wind velocity w_{vv} through a probability distribution $f(w')dw'$, which is the probability that w_{vv} lies between w'_{vv} and $w'_{vv} + dw'_{vv}$. α = 1.15 cm s^{-1} is the measure of variability in the vertical wind velocity (Roe, 2002). The precipitation dP_{RL} is given by Eq. XX, where the constants a = 2.5 · 10-11 kg^{-1} s^2 m and b = 5.9 · 10^{-9} s^3 kg were obtained by tuning to observations of Greenland (Roe, 2002). This equation is solved analytically using error functions (Roe and Lindzen, 2001), resulting in the following expression:

$$x_0 = \frac{a}{b} + w_{vv} \tag{62}$$

$$P_{RL} = be_{sat}\left[\frac{x_0}{2} + \frac{x_o^2 erf(a|x_0|)}{2|x_0|} + \frac{e^{-\alpha^2 x_0^2}}{2\sqrt{\pi}\alpha}\right] \tag{63}$$

The Roe and Lindzen precipitation model is applied to both the "smooth" interpolated GCM orography $h_{ref,GCM}$ (calculated with the weighting field for precipitation, which is not the same as the interpolated orography used in the temperature scheme) and to the "fine" modelled orography to calculate a correction factor with which the interpolated GCM precipitation $P_{ref,GCM}$ is adjusted:

$$P_{mod} = P_{ref,GCM}\frac{P_{RL,mod}}{P_{RL,GCM}} \tag{64}$$

4.3 Data structure

The current implementation of the matrix method uses the PI and LGM climates produced by Singarayer and Valdes (2010) as GCM snapshots, plus the ERA-40 reanalysis as a present-day reference state.

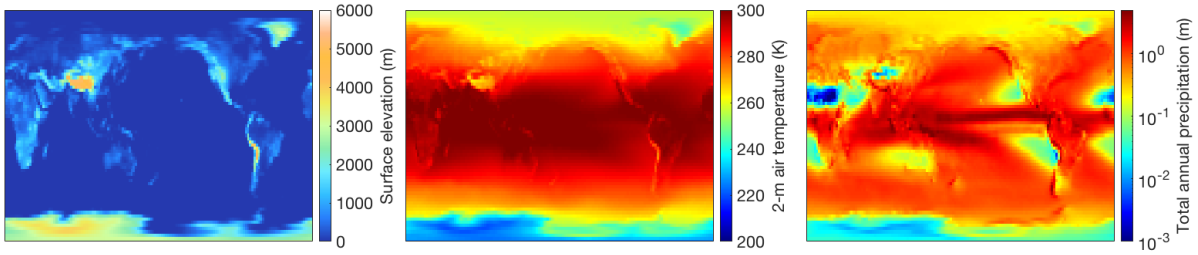


Figure 4: The ERA-40 reanalysis data.

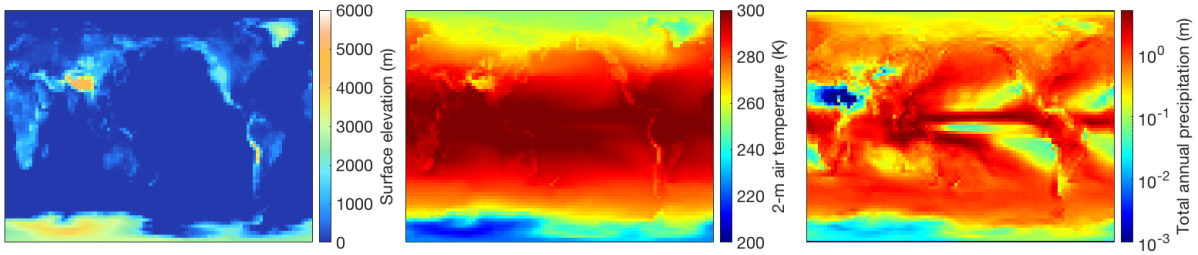


Figure 5: The PI GCM snapshot by Singarayer and Valdes (2010).

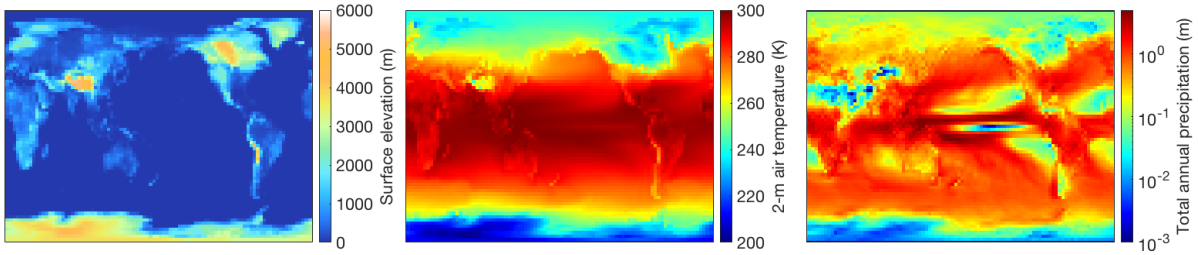


Figure 6: The LGM GCM snapshot by Singarayer and Valdes (2010).

The data structure of UFEMISM's climate model has been created specifically to accommodate the matrix method. The TYPE_CLIMATE_MODEL is subdivided into several "subclimates", which describe either the present-day observed climate PD_OBS (typically the ERA-40 reanalysis), the different GCM snapshot used in the matrix, or the final applied climate resulting from the matrix interpolation. This structure looks as follows:

- TYPE_CLIMATE_MODEL
  - PD_OBS (TYPE_SUBCLIMATE_REGION)
    - REAL(DP), DIMENSION(:,:) :: T2M ! Monthly mean 2-m air temperature $[K]$
    - REAL(DP), DIMENSION(:,:) :: PRECIP ! Monthly total precipitation $[m]$
    - REAL(DP), DIMENSION(:) :: Hs ! Reference surface elevation $[m]$
    - REAL(DP), DIMENSION(:) :: WIND_WE ! Zonal wind $[m/s]$
    - REAL(DP), DIMENSION(:) :: WIND_SN ! Meridional wind $[m/s]$
    - REAL(DP), DIMENSION(:) :: WIND_LR ! Wind in regional grid x-direction $[m/s]$
    - REAL(DP), DIMENSION(:) :: WIND_DU ! Wind in regional grid y-direction $[m/s]$
    - REAL(DP), DIMENSION(:) :: I_ABS ! Yearly total absorbed insolation $[J/m^2]$
    - ...
  - GCM_PI (TYPE_SUBCLIMATE_REGION)
    - ...

- GCM_LGM (TYPE_SUBCLIMATE_REGION)
  - * ...
- APPLIED (TYPE_SUBCLIMATE_REGION)
  - * ...

All data fields in the TYPE_CLIMATE_MODEL are defined on the mesh. To easily handle mesh updates, global climates (on their respective original lat/lon grids) are stored in memory as well. These are kept in the TYPE_CLIMATE_MATRIX, which is declared in UFEMISM_PROGRAM and passed to the different models as an argument. The TYPE_CLIMATE_MATRIX has the same structure as the TYPE_CLIMATE_MODEL, with subclimates containing the present-day observed climate and the GCM snapshots. When the mesh is updated, the different subclimates are remapped from the global fields to the new mesh.

5 Mass balance

The surface mass balance is modelled in UFEMISM using IMAU's in-house insolation-temperature model IMAU-ITM (Berends et al., 2018). This model took part in the GrSMBMIP intercomparison exercise (Fettweis et al., 2021), where it was shown to perform well in comparison to other parameterised SMB models when simulating the different components of the mass balance of the Greenland ice sheet during the satellite era.

The basal melt currently consists of the temperature/depth-dependent sub-shelf melt parameterisation by Martin et al. (2011), combined with the glacial/interglacial parameterisation by Pollard and DeConto (2012), and the subtended-angle/distance-to-open-ocean parameterisation also by Pollard and DeConto (2012). Basal melt underneath grounded ice is currently not included. All of these parameterisations are currently the same as in IMAU-ICE and its predecessor ANICE2.1.

5.1 Surface mass balance

The surface mass balance is modelled in UFEMISM using IMAU's in-house insolation-temperature model IMAU-ITM (Berends et al., 2018; Fettweis et al., 2021). This 0-D model takes monthly values of surface temperature and precipitation, and insolation at the top of the atmosphere, to separately calculate different components of the surface mass balance.

First, the snow fraction of precipitation is calculated according to the parameterization by Ohmura (1999), so that the monthly precipitation P is partitioned into the monthly snowfall S and the monthly rainfall R (all expressed in metres of water equivalent):

$$f_{snow} = \frac{1 - 0.796 \cdot tan^{-1}\left(\frac{T - T_0}{3.5}\right)}{2} \tag{65}$$

$$S = P \cdot f_{snow} \tag{66}$$

$$R = P \cdot (1 - f_{snow}) \tag{67}$$

The monthly snowmelt A is parameterised as a linear function of the 2-m air temperature T, the albedo a, and the insolation at the top of the atmosphere Q_{TOA} following the approach by Bintanja et al. (2002):

$$A = c_1(T - T_0) + c_2(Q_{TOA}(1 - a)) - c_3 \tag{68}$$

The refreezing F is calculated from the available liquid water content L and the (parameterised) superimposed water content L_{sup} following the approach by Huybrechts and de Wolde (1999) and Janssens and Huybrechts (2000):

$$L = R + A \tag{69}$$

$$L_{sup} = 0.012 \cdot max(0, T_0 - T) \tag{70}$$

$$F = min(L, L_{sup}, P) \tag{71}$$

The surface mass balance M_s is equal to the sum of snowfall, (minus) ablation, and refreezing, and converted from metres of water equivalent to metres of ice equivalent:

$$M_s = (S - A + F)\frac{\rho_w}{\rho_i} \tag{72}$$

In order to calculate the albedo used in the snowmelt parameterisation, the thickness of the firn layer H_f is modelled as well, using a simple no-compaction model where any snowfall is assumed to immediately become firn with the density of ice. The firn layer thickness is limited to 10 m. The albedo a is then parameterised according to the thickness of the firn layer H_f and the amount of snowmelt A_{prev} that occurred during the previous year:

$$a = a_{snow} - (a_{snow} - a_{surf})e^{-15H_f} - 0.015A_{prev} \tag{73}$$

Here, $a_{snow} = 0.85$ is the albedo of fresh snow, and a_{surf} is the albedo of the surface underneath the firn layer (either 0.5 for bare ice, or 0.2 for bare soil).

5.2 Basal mass balance

The basal melt currently consists of the (linear) temperature/depth-dependent sub-shelf melt parameterisation by Martin et al. (2011), combined with the glacial/interglacial parameterisation by Pollard and DeConto (2009), and the subtended-angle/distance-to-open-ocean parameterisation also by Pollard and DeConto (2009).

Pollard and DeConto (2009) provide a highly parameterised relation for sub-shelf melt, which depends only on horizontal shelf geometry. They provide three uniform values for sub-shelf melt underneath "protected", "exposed", and "deep ocean" shelves. A weighted average between these values is calculated as follows:

$$z_e = max[0, min[1, \frac{A - 80}{30}]]e^{\frac{-D}{100}} \tag{74}$$

$$z_d = max[0, min[1, \frac{h - h_{deep}}{200}]] \tag{75}$$

$$M = (1 - z_d)\left[(1 - z_e)M_p + z_e M_e\right] + z_d M_d \tag{76}$$

Here, the protected/exposed weighting factor z_e is calculated based on the angle A (degrees) subtended by the set of all straight lines from the point in question that reach open ocean without encountering land or grounded ice, and the shortest linear distance D (km) from the shelf point to the open ocean. The deep-ocean weighting factor z_d is calculated from the bathymetry h, scaled with respect to a threshold depth h_{deep} which represents the transition from the continental shelf to the oceanic basin (configurable per model region).

The three uniform melt rates for protected shelves M_p, exposed shelves M_e and deep-ocean shelves M_d are calculated as weighted averages from fixed values for the present-day, and hypothetical "cold" and "warm" ocean states. The values for these states were derived by Bas de Boer for ANICE somewhere in the early 2010' to provide reasonable patterns of Antartic advance/retreat. They are configurable per model region, with the default values copied from ANICE. The interpolation between these fixed values is done based on CO_2 and high-latitude insolation.

As in ANICE and IMAU-ICE, the melt rate for protected shelves is replaced with the parameterisation by Martin et al. (2011). Here, the sub-shelf melt rate M_p is parameterised as a linear function of the (mean) ocean temperature T_{Om} according to Martin et al. (2011):

$$T_{freeze} = -1.9011 - 7.64 \cdot 10^{-4} H \frac{\rho_i}{\rho_w} \tag{77}$$

$$M_p = \rho_w c_{pO} \gamma_T c_M \frac{T_{Om} - T_{freeze}}{L_{fus} \rho_i} \tag{78}$$

Here, the freezing temperature T_{freeze} of ocean water at the base of the shelf is expressed as a linear function of the water depth (which is simply the ice thickness H times the ratio of densities). The sub-shelf melt rate M is then found from the heat capacity $C_{pO} = 3.97 \cdot 10^3 Jkg^{-1}K^{-1}$ of the ocean, the thermal exchange velocity $\gamma_T = 10^{-4} ms^{-1}$, a tuning factor c_M (default value 0.005), the mean ocean temperature T_{Om}, and the heat of fusion $L_{fus} = 3.335 \cdot 10^J kg^{-1}$ for liquid water.

6 Glacial isostatic adjustment

Currently, glacial isostatic adjustment (GIA) is calculated with a simple ELRA (Elastic Lithosphere, viscously Relaxed Asthenosphere) model according to Huybrechts (1992). In future improvements, the option to instead use bedrock deformation rates supplied by SELEN will be included as well.

6.1 ELRA

6.1.1 Theory

The ELRA model uses three scalar values to completely describe the response of the solid Earth to changes in loading: the flexural rigidity $D = 10^5$ kg m^2 s^{-2} of the lithosphere, and the relaxation time $\tau = 3000$ yr and density $\rho_m = 3300$ kg m^{-3} of the viscous mantle (all configurable). These are used to calculate the 2D flexural profile F (i.e. the equilibrium deformation in response to a unit point load), using the Kelvin function Klv:

$$L_r = \left(\frac{D}{\rho_m g} \right)^{\frac{1}{4}} \tag{79}$$

$$F(r) = Klv(\frac{r}{L_r}) \tag{80}$$

Here, L_r (m) is the influence radius of lithospheric rigidity, and r is the linear distance (m) from the point load. The equilibrium bedrock deformation δh_{eq} in response to a load distribution $P(x, y)$ is calculated by 2D convolution of the load with the flexural profile:

$$\delta h_{eq}(x, y) = F(r) * P(x, y) \tag{81}$$

The lithospheric deformation rate $\frac{\partial h}{\partial t}$ is proportional to the difference between the modelled deformation :

$$\frac{\partial h}{\partial t}(x, y, t) = \frac{\delta h(x, y, t) - \delta h_{eq}(x, y)}{\tau} \tag{82}$$

6.1.2 Implementation

Due to the highly diffusive nature of glacial isostatic adjustment, the ELRA model needs neither a high spatial resolution nor a small time step. Therefore, the calculation of the deformation rate $\frac{\partial h}{\partial t}$ has been given its own separate time step DT_ELRA (configurable, default value 100 yr). Note that the bedrock elevation itself is updated in every model loop to prevent "jumps". Since the ELRA model does not require a high spatial resolution, and also since performing a proper 2D convolution on an unstructured mesh is not easy, the ELRA model is run on a fixed square grid (resolution configurable, default value 100 km). The surface load (relative to the present-day load, which includes both ice and liquid water) is calculated on the mesh, and then projected to this square grid. The 2D convolution is performed on the grid, and the resulting deformation rate is mapped back to the mesh. While the two mapping operations induce some small amount of numerical diffusion, this does not significantly affect the results due to the already highly diffusive nature of the ELRA model itself.

A similar square grid construction is being developed for the coupling to SELEN.

7 Unstructured grid

The core of UFEMISM is the unstructured triangular grid (the *mesh*) upon which the ice-dynamical equations are discretised and solved. The data structure used to describe the mesh (based on connectivity lists rather than matrices) has been created specifically for UFEMISM.

7.1 Basic concepts

Before we get started, we will briefly review some important concepts in mesh geometry.

7.1.1 Nomenclature

Consider the simple mesh shown in Fig. **??**.

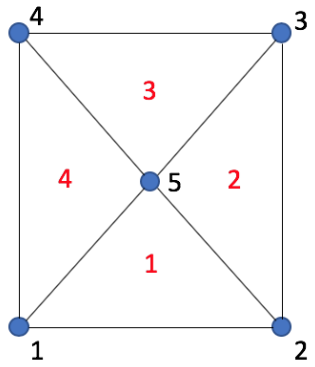


Figure 7: A simple mesh.

This *mesh* consists of five *vertices*, each connected to several others by *lines* and/or *segments*, which together span four *triangles*. Note: a connection between two vertices is called a line when the connection is a shared border of two triangles. Connections that are part of the domain boundary (and thus form the border of only a single triangle) are called segments. Vertices 1 and 2 are connected by a segment, vertices 1 and 5 are connected by a line.

7.1.2 Voronoi cells and Delaunay triangulation

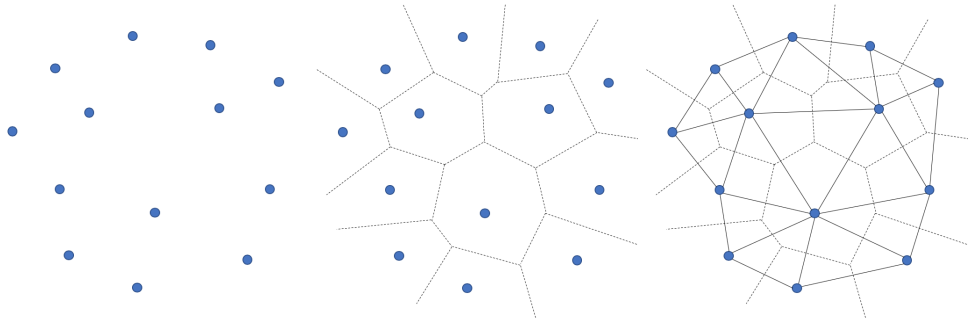
For a given set of points in $\mathbb{R}^2$, a *triangulation* is a way to connect the points with straight lines, such that each point is connected to at least two others by lines spanning triangles (and nothing else), without any intersecting lines. There generally exist many different triangulations for any set of points, but there is always a single unique one, called the *Delaunay triangulation*, which exhibits several useful properties. Formally, the Delaunay triangulation is defined as the triangulation where the circumcircle of any triangle contains does not contain any points inside it other than the three points spanning the triangle. However, this definition is not very easy to comprehend. Instead, it is more intuitive to define the Delaunay triangulation from the concept of Voronoi cells.
For a given set of points in $\mathbb{R}^2$, each point can be assigned a region of space that is closer to that point than to any other point. This region is called the *Voronoi cell* belonging to that point. The Delaunay triangulation is the triangulation where a point is connected to another point if and only if their Voronoi cells share a boundary. This is illustrated in Fig. **??**.

While we will not prove them, some useful properties of the Delaunay triangulation are:

- The line connecting two points is always perpendicular to the shared boundary of their respective Voronoi cells.
- The Delaunay triangulation minimises the smallest internal angle of all triangles.

These two properties are especially useful in the context of ice-sheet models (and other fluid dynamics models). The first property is very useful when using a finite volume approach to solve the mass conservation equation, while the second one is useful for improving numerical stability. As far as I've been able to find, all models that use unstructured triangular grids use the Delaunay triangulation.

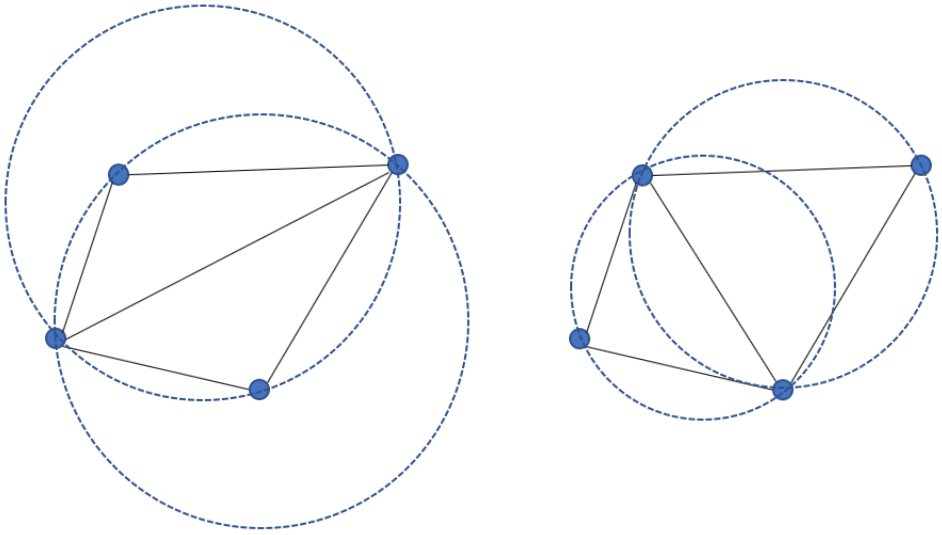
It can be proven mathematically (if one were of a mind to do so) that any triangulation can be changed into the Delaunay triangulation by iteratively finding triangle pairs that violate the local Delaunay criterion



(a) (b) (c)

Figure 8: (a) A collection of points in $\mathbb{R}^2$, (b) the Voronoi cells of the points, (c) the Delaunay triangulation.

(i.e. where the fourth vertex lies inside the circumcircle of the other three), and "flipping" them. This is illustrated in Fig. **??**.



(a) (b)

Figure 9: (a) This pair of triangles violates the local Delaunay criterion, (b) the pair has been "flipped", and now satisfies the criterion.

7.2 Data structure

Theoretically, all that is needed to uniquely and completely describe a mesh is a list of the coordinates of its vertices. Everything else flows from there; their position in space determines the geometry of their Voronoi cells, which determines the Delaunay triangulation, which determines the connectivity. However, deriving all this information from the vertex coordinates can be a lengthy process, so instead several different forms of the mesh connectivity are stored in memory as well.

A mesh consisting of nV vertices and $nTri$ triangles is stored in memory in the form of vertex data and triangle data:

Table 1: Vertex data

Name	**Description**	**Size**
V	Vertex [x,y] coordinates	DOUBLE(nV,2)
nC	Number of connected vertices	INT(nV)
C	Indices of connected vertices	INT(nV,32)
niTri	Number of triangles containing this vertex	INT(nV)
iTri	Indices of triangles containing this vertex	INT(nV,32)
edge_index	Edge index	INT(nV)

Table 2: triangle data

Name	**Description**	**Size**
Tri	Indices of the three vertices spanning this triangle	INT(nTri,3)
Tricc	[x,y] coordinates of the triangle's circumcentre	DOUBLE(nTri,2)
TriC	Indices of connected triangles	INT(nTri,3)
Tri_edge_index	Edge index	INT(nTri)

As the name "unstructured mesh" suggests, there is no particular order to the indices of the vertices or triangles, and no relation between their indices and their position. However, the following rules apply:

- **C**: neighbouring vertices are listed in counterclockwise order, and only the first nC entries of the 32 values per vertex are used; all other entries are zero. For vertices lying on the domain boundary, the list may not "jump" outside the domain; if, for example, this vertex lies on the eastern border, the first entry must be the neighbour lying north of the vertex on the boundary, and the last one must be the neighbour lying south of the vertex on the boundary.
- **iTri**: triangles containing the vertex are listed in counterclockwise order, and only the first $niTri$ entries of the 32 values per vertex are used; all other entries are zero. For boundary vertices, the same rule applies as for **C**.
- **Tri**: the three vertices spanning a triangle are listed in counterclockwise order. There are no rules for which vertex is listed first, but it must correspond to the entries in **TriC**.
- **TriC**: the three neighbouring triangles are sorted such that the first neighbouring triangles lies opposite of the first vertex, the second one across from the second vertex, etc. If no neighbouring triangle exists (i.e. when this side of the triangle is part of the domain boundary), a zero is listed.
- **edge_index**: the edge index describes if a vertex lies on the domain boundary: 1 = northern boundary, 2 = north-east corner, 3 = eastern boundary, etc., 0 = not on any boundary.
- **Tri_edge_index**: similar to the edge index of vertices. The corner values (2,4,6,8) are used when a triangle is situated such that two if its sides lie on different parts of the domain boundary.

We will illustrate these variables with an example. Consider the simple mesh shown in Fig. **??**.

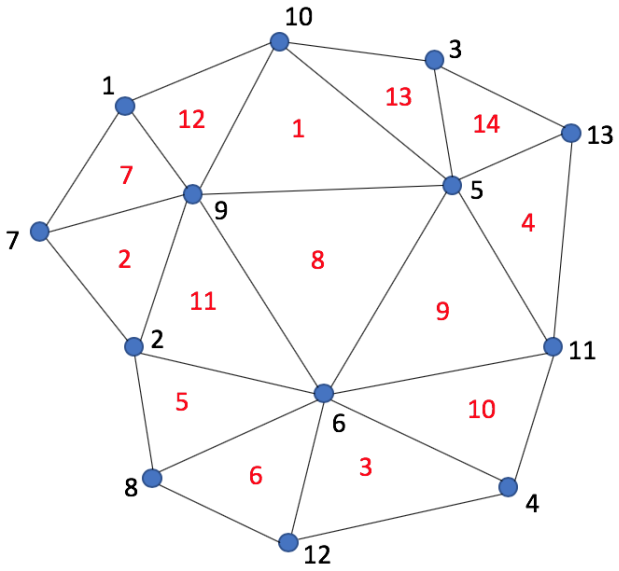


Figure 10: Another simple mesh.

The mesh data for this mesh is shown in Tables **??** and **??**. Vertex coordinates **V** and triangle circumcentre coordinates **Tricc** are omitted for brevity.

Table 3: Vertex data

vi	**V**	**nC**	**C**	**niTri**	**iTri**
1	...	3	7,9,10	2	7,12
2	...	4	8,6,9,7	3	5,11,2
3	...	3	10,5,13	2	13,14
4	...	3	11,6,12	2	10,3
5	...	6	13,3,10,9,6,11	6	8,9,4,14,13,1
6	...	7	9,2,8,12,4,11,5	7	3,10,9,8,11,5,6
7	...	3	2,9,1	2	2,7
8	...	3	12,6,2	2	6,5
9	...	6	10,1,7,2,6,5	6	2,11,8,1,12,7
10	...	4	1,9,5,3	3	12,1,13
11	...	4	13,5,6,4	3	4,9,10
12	...	3	4,6,8	2	3,6
13	...	3	3,5,11	2	14,4

Table 4: Triangle data

ti	**Tri**	**Tricc**	**TriC**
1	9,5,10	...	13,12,8
2	9,7,2	...	0,11,7
3	12,4,6	...	10,6,0
4	13,5,11	...	9,0,14
5	2,8,6	...	6,11,0
6	12,6,8	...	5,0,3
7	7,9,1	...	12,0,2
8	9,6,5	...	9,1,11
9	6,11,5	...	4,8,10
10	4,11,6	...	9,3,0
11	9,2,6	...	5,8,2
12	10,1,9	...	7,1,0
13	5,3,10	...	0,1,14
14	13,3,5	...	13,4,0

7.3 Staggered mesh

In order to apply the finite volume method, we need to define ice fluxes on the boundaries of the Voronoi cells of the vertices. This means we need a way to efficiently keep track of these boundaries. This is done by creating a "staggered mesh", which is conceptually very similar to the Arakawa C mesh used in square-grid models.

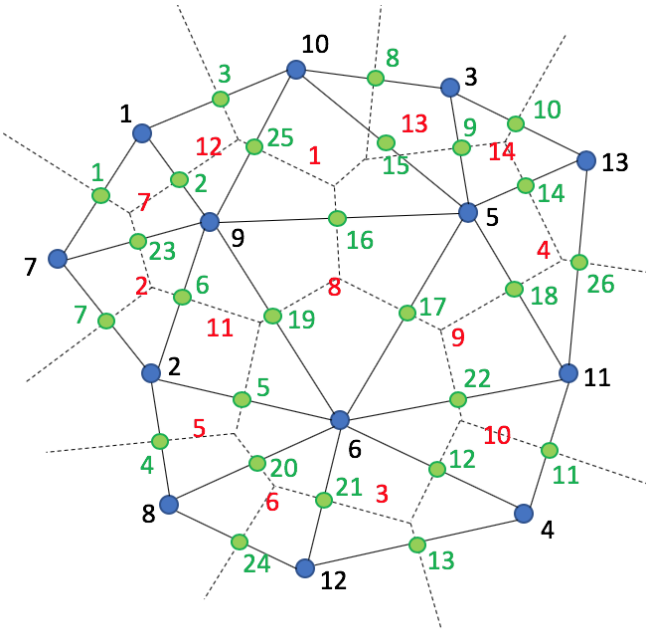


Figure 11: The staggered vertices (green) correspond to the connections between the regular vertices (blue).

Each connection between two regular vertices is given a unique index, and represented by a staggered vertex lying halfway along the connection. Two lists are created, **Aci** and **iAci**, which relate regular vertices to staggered vertices, and vice versa. For the simple mesh shown above, these lists look like this:

vi	**iAci**	**ai**	**Aci**
1	1,2,3	1	1,7,9,0
2	4,5,6,7	2	1,9,10,7
3	8,9,10	3	1,10,0,9
4	11,12,13	4	2,8,6,0
5	14,9,15,16,17,18	5	2,6,9,8
6	19,5,20,21,12,22,17	6	2,8,7,6
7	7,23,1	7	2,7,0,9
8	24,20,4	8	3,10,5,0
9	25,2,23,6,19,16	9	3,5,13,10
10	3,25,15,8	10	3,13,0,5
11	26,18,22,11	11	4,11,6,0
12	13,21,24	12	4,6,12,11
13	10,14,26	13	4,12,0,6
		14	5,13,3,11
		15	5,10,9,3
		16	5,9,6,10
		17	5,6,11,9
		18	5,11,13,6
		19	6,9,2,5
		20	6,8,12,2
		21	6,12,4,8
		22	6,11,5,4
		23	7,9,1,2
		24	8,12,6,0
		25	9,10,1,5
		26	11,13,5,0

iAci lists the staggered vertices surrounding each regular vertex. The ordering corresponds to that of **C**, so that the j'th staggered vertex listed for vertex i lies halfway along the connection between v_i and its j'th neighbour. **Aci** lists the regular vertices along whose connection each staggered vertex lies. Additionally, it lists the two regular vertices lying to the left and right of this connection, so that the four columns in a row of **Aci** read $[v_i, v_j, v_l, v_r]$, with v_l lying to the left of the line from v_i to v_j, and v_r lying to the right. If either v_l or v_r does not exist (which can happen when v_i and v_j lie on the domain boundary), the entry is zero. Keeping track of v_l and v_r is very useful for calculating derivatives on the staggered vertices, as we will see in Sect. XXX on discretisation.

The SSA is solved on both the regular and the staggered mesh at the same time. To make this easier, a "combined Aa-Ac mesh" is created as well. This is done by adding all the staggered vertices as regular vertices to the regular mesh. Applying this to the mesh in Fig. **??** results in the following combined Aa-Ac mesh:

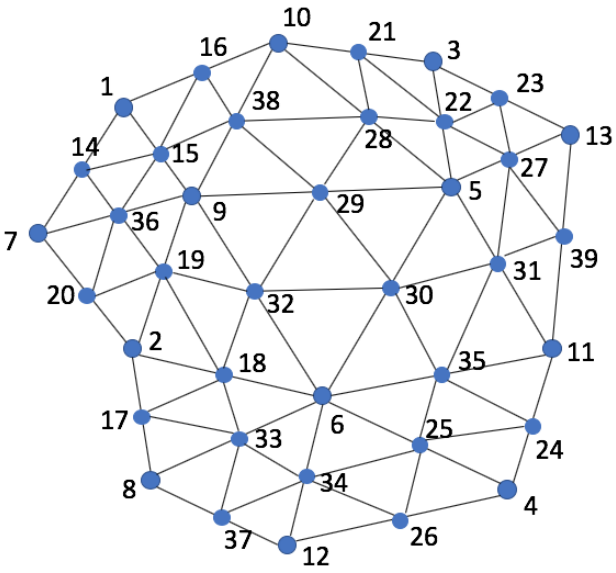


Figure 12: The combined Aa-Ac mesh.

Note that the indices of the regular vertices are unchanged, while the indices of the staggered vertices have been increased by nV. This makes it very easy to convert data between the regular/staggered mesh and the combined Aa-Ac mesh. Since the combined Aa-Ac mesh is a complete mesh, neighbour functions can be calculated for it using the same subroutine as for the regular mesh.

7.4 Mesh refinement

UFEMISM uses an iterative mesh refinement approach to create the model mesh. The basis for this is Ruppert's algorithm (Ruppert, 1995), which iteratively "splits" triangles (i.e. adds new vertices at their circumcentres) whose smallest internal angle lies below a certain prescribe threshold value (typically 25°) In pseudo-code form, this reads as follows:

WHILE (bad triangles exist)
    Find next bad triangle
    Add new vertex at this triangle's circumcentre
    Update Delaunay triangulation (i.e. flip triangle pairs if needed)
END WHILE

Ruppert (1995) proved that this algorithm converges (i.e. produces a mesh with no angles below the threshold value, no excessively small triangles, and a generally limited number of vertices) as long as the prescribed mesh boundary contains no sharp angles. Since the mesh boundary in UFEMISM is a simple rectangle, this is not a problem.

For clarity, we will give a brief example of triangle "splitting". Consider the simple mesh shown in Fig. **??**. The four panels show the steps of "splitting" triangle 8 by adding a new vertex at its circumcentre, and flipping some of the newly formed triangle pairs to ensure the new mesh satisfies the Delaunay criterion.

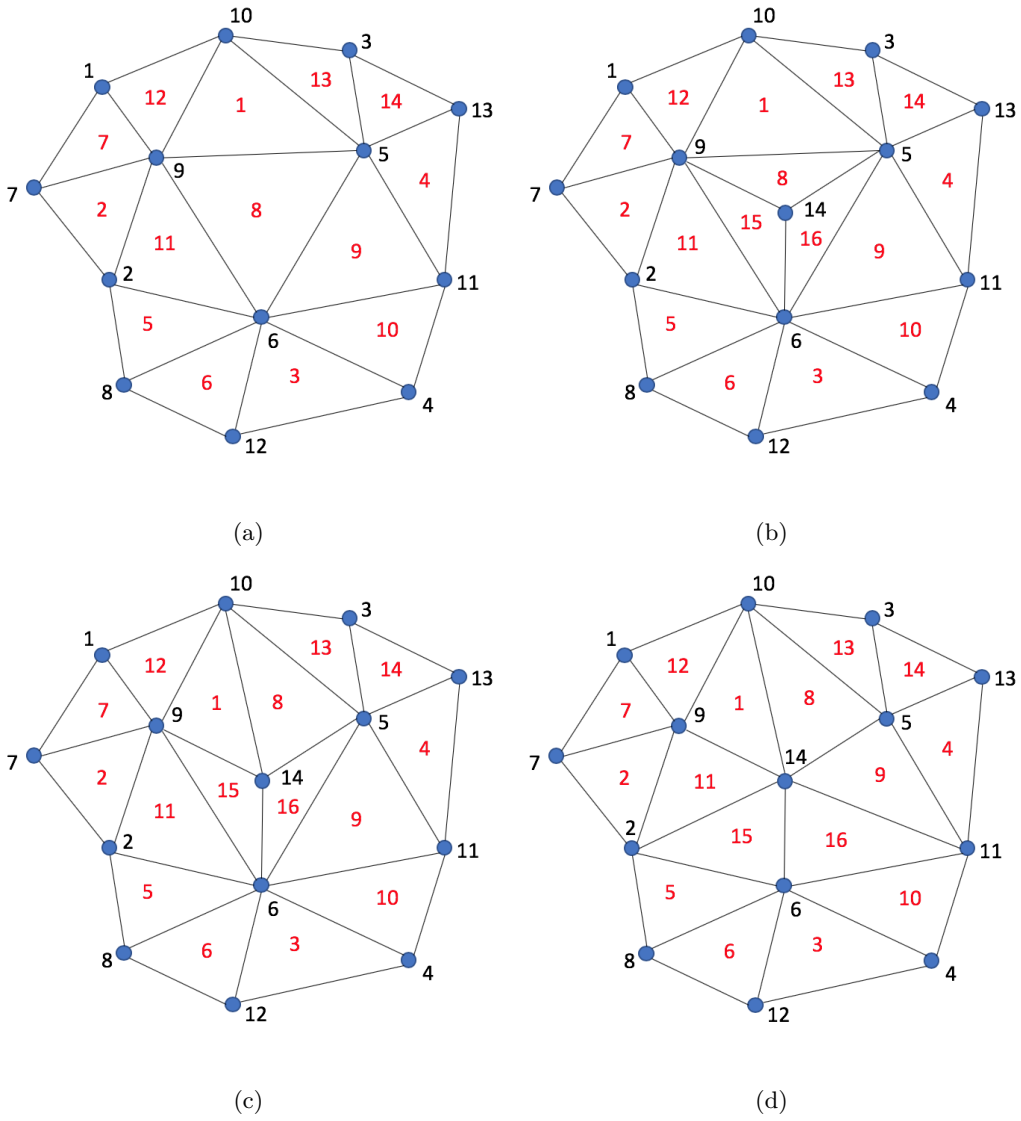


Figure 13: (a) Triangle 8 is marked for splitting. (b) A new vertex is added at the circumcentre of triangle 8, replacing it by three new triangles. (c) New triangle pair 8-1 is found to violate the local Delaunay criterion, and is flipped. (d) New triangle pairs 15-11 and 16-9 were also flipped; the mesh now once again is a Delaunay triangulation.

In UFEMISM, Ruppert's algorithm has been extended to include more conditions that can cause a triangle to be marked as "bad" than just the smallest internal angle, which are based on the ice model data. Currently, the following conditions are included:

Table 5: mesh refinement conditions

config parameter	**Description: triangle is marked as bad if...**
ALPHA_MIN	...any of its three internal angles is smaller than this value (original condition in Ruppert's algorithm).
RES_MAX	...any of its three sides exceeds 2 * this length.
RES_MAX_GL	...any of its three sides exceeds 2 * this length and the grounding line passes through it.
RES_MAX_CF	...any of its three sides exceeds 2 * this length and the calving front passes through it.
RES_MAX_MARGIN	...any of its three sides exceeds 2 * this length and the ice margin passes through it.
RES_MAX_COAST	...any of its three sides exceeds 2 * this length and the coastline passes through it.
POI_RESOLUTIONS	...any of its three sides exceeds 2 * this length and it contains a POI.

The mesh is refined step-wise, with the maximum resolutions being halved in each step. This means that generally, resolution "contrasts" (differences in resolution between adjacent vertices) that occur during mesh refinement are less dramatic, and that therefore the maximum number of neighbours of any vertex in the mesh (which, for the final mesh, is limited by the maximum internal angle from Ruppert's algorithm, but can be larger during the refinement process) doesn't become as high as would otherwise be the case. This saves on how much memory needs to be allocated, and also makes mesh alignment (see Sect. XX on mesh generation parallelisation) a lot easier.

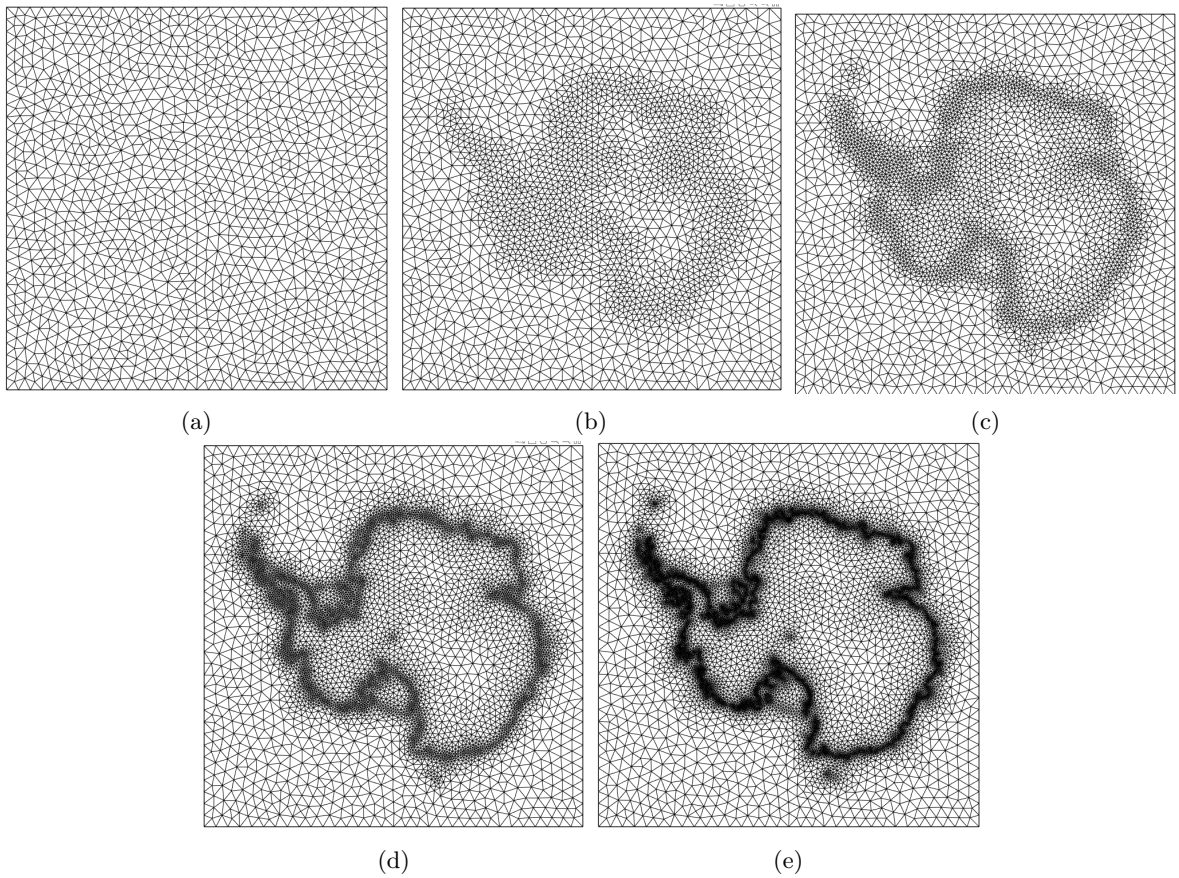


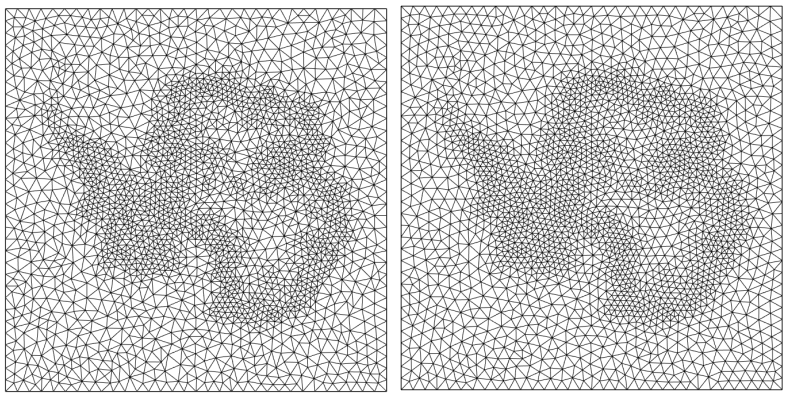
Figure 14: Step-wise refinement of a mesh, with the maximum resolution decreasing from 200 km (a) to 12.5 km (e).

The code that performs this step-wise mesh refinement looks more or less like this:

```
! Initialise the five-vertex dummy mesh
CALL initialise_dummy_mesh( mesh)
```

```
res_min_inc = C%res_max * 2._dp

DO WHILE (res_min_inc > C%res_min)

   ! Increase resolution
   res_min_inc = res_min_inc / 2._dp

   ! Determine resolutions of regions of interest
   mesh%res_min = MAX( C%res_min, res_min_inc)
   mesh%res_max_margin = MAX( C%res_max_margin, res_min_inc)
   mesh%res_max_gl = MAX( C%res_max_gl, res_min_inc)
   etc.

   ! Refine the process submesh
   CALL refine_mesh( mesh)

   ! Align with neighbouring submeshes
   CALL align_all_submeshes( mesh)

   ! Split any new triangles (added during alignment) that are too sharp
   CALL refine_submesh_geo_only( mesh)

   ! Smooth the submesh using Lloyd' algorithm
   CALL Lloyds_algorithm_single_iteration_submesh( mesh)

END DO
```

The meaning of the "alignment" step is explained in Sect. XX on the parallelisation of mesh generation. Lloyd's algorithm moves all vertices in a mesh to the geometric centres of their Voronoi cells. Theoretically this is an iterative approach, because moving a vertex alters the shape not only of its own Voronoi cell, but also those of all its neighbours. This procedure tends to "smooth" the mesh, resulting in nicer triangles (i.e. with internal angles that get closer to 60 degrees) and lower resolution gradients. This greatly improves the stability of the SOR solver used in the SSA.



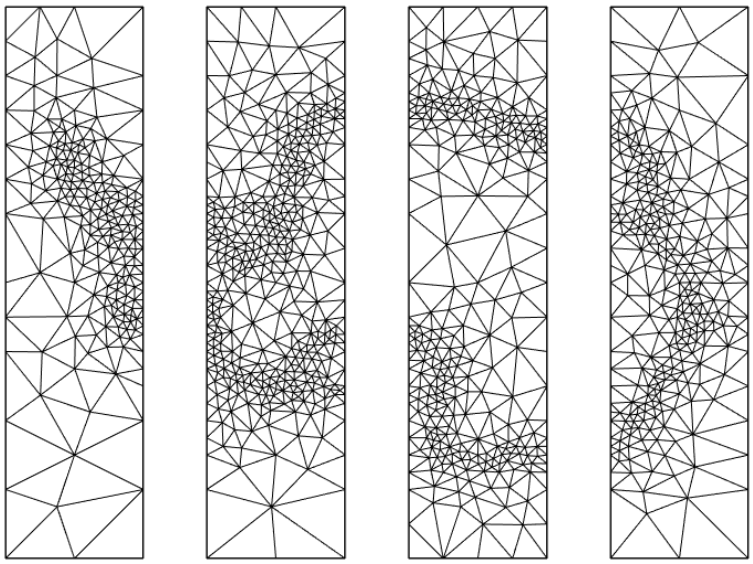
(a) (b)

Figure 15: (a) An unsmoothed, 100 km resolution mesh. (b) The same mesh after applying one single iterations of Lloyd's algorithm.

If run for infinitely many iterations, the mesh would approach a regular hexagonal structure, since this is the only way for vertices to coincide with the geometric centres of their Voronoi cells. However, to achieve this requires many iterations; only applying a single iteration during each refinement step, as is done here, achieves a good balance between smoothing the mesh without smoothing the resolution gradients too much (which would unnecessarily increase the number of vertices).

7.5 Paralellisation

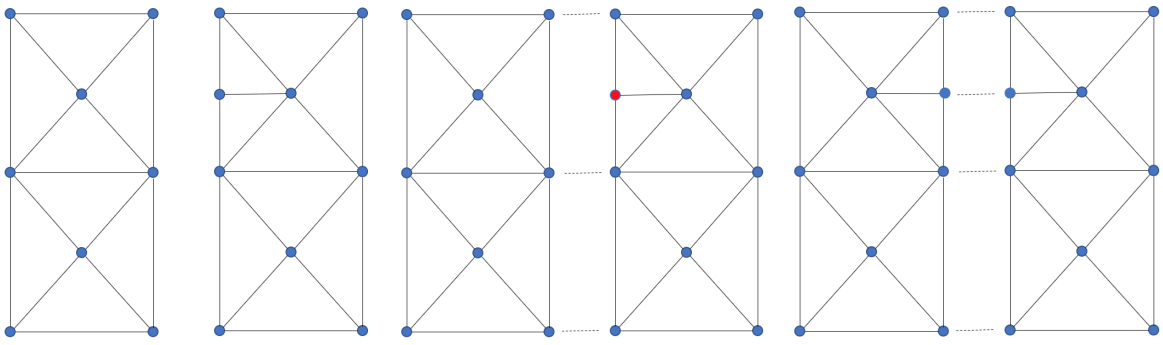
Mesh generation has been parallelised using a form of domain decomposition. Each processor is assigned a portion of the total model domain, and will generate a "submesh" for that portion only. Once this is done, the different meshes are "merged" to create one single mesh covering the whole model domain.



(a) (b) (c) (d)

Figure 16: Four submeshes for Antarctica.

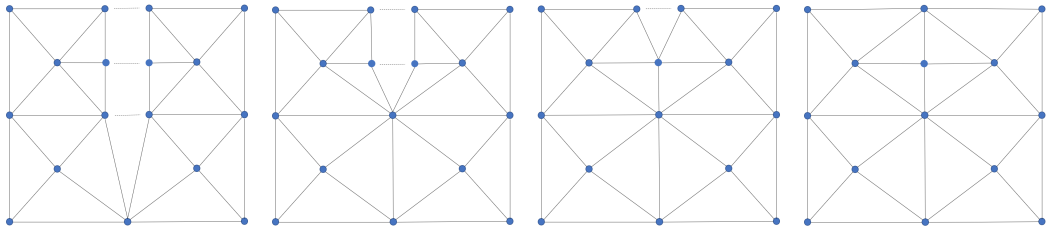
The process of mesh merging hinges on the fact that, when Rupperts algorithm refines a boundary triangle whose circumcentre lies outside the (sub)mesh domain, a new vertex is added at the midpoint of the triangle's segment. This means that boundary vertices always lie at integer sums of power-of-two fractions of a boundary edge (e.g. $\frac{1}{2}$, $\frac{3}{4}$, $\frac{1}{2}+\frac{1}{8}+\frac{13}{64}$, etc.). Looking carefully at submeshes a) and b) in the figure, we can see that many vertices on the eastern boundary of submesh a) lie at the same y-coordinate as vertices on the western boundary of submesh b). These vertices can then be merged. Generally, a small number of vertices on either boundary will have no corresponding vertex on the opposite boundary. During submesh merging, additional vertices are added first to ensure that all vertices on either boundary have corresponding vertices on the opposite boundary (a process called "submesh alignment").



(a) (b) (c)

Figure 17: Aligning two simple submeshes: a) before, b) a single non-overlapping (red) vertex is detected, c) after.

The figure illustrates how two very simple submeshes are aligned. First, the vertices lying on the "seam" are checked, and those that overlap are marked. The right-hand submesh has one vertex that doesn't overlap with any vertex in the left-hand submesh. This is remedied by adding a new vertex to the left-hand mesh. All seam vertices now match, and the submeshes can be merged. This is done by merging the overlapping vertices one by one, moving along the seam like a zipper, as illustrated in Fig. Xx.



(a) (b) (c) (d)

Figure 18: Merging two simple submeshes. Note that no actual deformation is occurring; the vertices along the seam have the same x-ccordinate, and have only been moved apart for this illustration.

Since only two submeshes can be merged at a time, submesh merging is an iterative process. For example, the four submeshes in Fig. XX would be merged in two phases. In the first phase, processor a) would be given access to the data of the submesh created by processor b). It allocates new memory that can accomodate the new (merged) submesh ab), and then merges submeshes a) and b) into this new memory. The old memory for submeshes a) and b) is then deallocated. At the same time, processor c) will do the same with submeshes c) and d), creating a new submesh cd). In the second phase, processor a) will merge submeshes ab) and cd) into the final mesh.

The four submeshes shown above all have equal domain widths. This is generally not the case; domain widths for mesh generation are determined such that each processor will have (roughly) the same number of vertices. This is done by looking at the vertex coordinates of the previous mesh, which is assumed to be a reasonable approximation of the new mesh.

7.6 Discretisation

Here, we will derive a way to discretise the first and second partial derivatives of a function defined on a mesh.

7.6.1 First-order partial derivatives

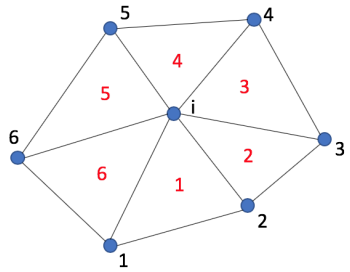


Figure 19: A simple mesh.

Consider the simple mesh in Fig. **??**. Let f be a function defined on the vertices of the mesh. Let vertex i be connected to n neighbouring vertices, thus spanning n triangles (in Fig. **??**, $n = 6$). In order to derive the first partial derivatives f^i_x, f^i_y of f on vertex i, we will first derive the partial derivatives on the triangles surrounding i. Since these triangles are plane sections in $\mathbb{R}^3$, they have well-defined first partial derivatives, which can be expressed using the normal vector to the plane. Let vertex i be described by a vector $\mathbf{v}^i = \left[x^i, y^i, f^i\right]$. Triangle t is then spanned by $\mathbf{v}^i$, $\mathbf{v}^t$ and $\mathbf{v}^{t+1}$, and its normal vector is given by:

$$\mathbf{n}^t = (\mathbf{v}^t - \mathbf{v}^i) \times (\mathbf{v}^{t+1} - \mathbf{v}^i) = \begin{bmatrix} f^i(y^{t+1} - y^t) + f^t(y^i - y^{t+1}) + f^{t+1}(y^t - y^i) \\ f^i(x^t - x^{t+1}) + f^t(x^{t+1} - x^i) + f^{t+1}(x^i - x^t) \\ (x^t - x^i)(y^{t+1} - y^i) - (y^t - y^i)(x^{t+1} - x^i) \end{bmatrix}. \tag{83}$$

Note that indices of the "next neighbour" $(t+1)$ need to be taken $mod(n)$ so that if $t = n$, then $t + 1 = 1$.

The first partial derivative $f^t_{x,tri}$ of f on t is given by:

$$f^t_{x,tri} = \frac{-n^t_x}{n^t_z}. \tag{84}$$

This can be written as a linear combination of the values of f on the three vertices spanning triangle t:

$$f^t_{x,tri} = f^i N^{t,1}_{x,tri} + f^t N^{t,2}_{x,tri} + f^{t+1} N^{t,3}_{x,tri}. \tag{85}$$

The linear coefficients $N^{t,1}_{x,tri} = \frac{y^t - y^{t+1}}{n^t_z}$, $N^{t,2}_{x,tri} = \frac{y^{t+1} - y^i}{n^t_z}$, $N^{t,3}_{x,tri} = \frac{y^t - y^i}{n^t_z}$ follow from (83) and depend only on mesh geometry, which means they only need to be calculated once. Such linear coefficients, which combine function values into a function derivative, we shall call "neighbour functions". The coefficients in (85) are the triangle neighbour functions for the first partial derivative on triangle t.

Since the vertex f represents the point where n plane sections meet, the gradient of the resulting piecewise smooth surface is undefined here. We solve this by averaging the gradients of the surrounding the triangles:

$$\begin{aligned} f^i_x &= \frac{1}{n} \sum_{t=1}^{n} f^t_{x,tri} \\ &= \frac{1}{n} \sum_{t=1}^{n} \left[f^i N^{t,1}_{x,tri} + f^t N^{t,2}_{x,tri} + f^{t+1} N^{t,3}_{x,tri} \right] \\ &= \frac{f^i}{n} \sum_{t=1}^{n} N^{t,1}_{x,tri} + \frac{1}{n} \sum_{t=1}^{n} \left[f^t N^{t,2}_{x,tri} + f^{t+1} N^{t,3}_{x,tri} \right]. \end{aligned} \tag{86}$$

Since the sum index t is periodic in n (i.e. if $t = n$ then $t + 1 = 1$), we can use the following identity:

$$\sum_{t=1}^{n} f^{t+1} g^t = \sum_{t=1}^{n} f^t g^{t-1}$$

This simplifies (86) to:

$$\begin{aligned} f_x^i &= \frac{f^i}{n}\sum_{t=1}^{n} N_{x,tri}^{t,1} + \frac{1}{n}\sum_{t=1}^{n}\left[f^t N_{x,tri}^{t,2} + f^{t+1} N_{x,tri}^{t,3}\right] \\ &= \frac{f^i}{n}\sum_{t=1}^{n} N_{x,tri}^{t,1} + \frac{1}{n}\sum_{t=1}^{n} f^t (N_{x,tri}^{t,2} + N_{x,tri}^{t-1,3}). \end{aligned} \tag{87}$$

To further simplify this expression, we introduce the vertex neighbour functions N_x^i for the first partial derivative on vertex i:

$$\begin{aligned} N_x^i &= \frac{1}{n}\sum_{t=1}^{n} N_{x,tri}^{t,1}, \\ N_x^t &= \frac{1}{n}(N_{x,tri}^{t,2} + N_{x,tri}^{t-1,3}). \end{aligned} \tag{88}$$

This simplifies (87) to:

$$f_x^i = f^i N_x^i + \sum_{t=1}^{n} f^t N_x^t. \tag{89}$$

7.6.2 Second-order partial derivatives

In the previous section, we expressed the first partial derivative f_x^i on vertex i as the average of the values on the surrounding triangles. In order to derive an expression for the second partial derivatives, we view the geometric centres of those triangles as staggered vertices, where the values of the first partial derivatives constitute new function values. We then construct a new set of subtriangles, spanned by vertex i and these staggered vertices, as illustrated in Fig. A3. Since we know the values of $f_{x,tri}^t$ and $f_{y,tri}^t$ on these new vertices, we can use the same approach as before to calculate the second derivatives on the sub-triangles, and average them to get the value on vertex i. Preliminary experiments showed that using the geometric centre instead of the circumcentre yields more stable solutions when using these to solve differential equations. A mathematical proof of why this is the case might be interesting, but lies beyond the scope of this study.

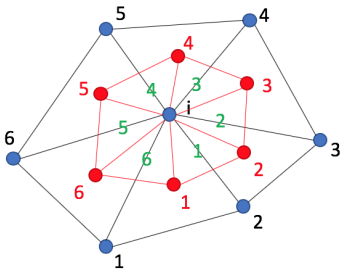


Figure 20: A simple mesh, with the regular vertices shown in blue. The temporary "staggered" vertices, used to derive an expression for the second partial derivatives, are shown in red. They span a new set of "subtriangles", shown in green.

A new, staggered vertex $\mathbf{v}_x^t$ is created in triangle t, using the horizontal coordinates of the geometric centre of t, and the first partial derivative $f_{x,tri}^t$ of f on t as the vertical coordinate:

$$\mathbf{v}_x^t = \left[\frac{x^i + x^t + x^{t+1}}{3}, \frac{y^i + y^t + y^{t+1}}{3}, f_{x,tri}^t\right]. \tag{90}$$

The subtriangle s is spanned by $\mathbf{v}_x^i$, $\mathbf{v}_x^s$ and $\mathbf{v}_x^{s+1}$, where $\mathbf{v}_x^i = \left[x^i, y^i, f_x^i\right]$, and f_x^i given by (89). The normal vector $\mathbf{n}_x^s$ to subtriangle s is then given by:

$$\begin{aligned} \mathbf{n}_x^s &= (\mathbf{v}_x^s - \mathbf{v}_x^i) \times (\mathbf{v}_x^{s+1} - \mathbf{v}_x^i) \\ &= \frac{1}{3}\begin{bmatrix} f_x^i(y^{s+2} - y^s) + f_{x,tri}^s(2y^i - y^{s+1} - y^{s+2}) + f_{x,tri}^{s+1}(y^s + y^{s+1} - y^i) \\ f_x^i(x^s - x^{s+2}) + f_{x,tri}^s(x^{s+1} + x^{s+2} - 2x^i) + f_{x,tri}^{s+1}(2x^i - x^s - x^{s+1}) \\ \frac{1}{3}(x^s + x^{s_1} - 2x^i)(y^{s+1} + y^{s+2} - 2y^i) - \frac{1}{3}(y^s + y^{s+1} - 2y^i)(x^{s+1} + x^{s+2} - 2x^i) \end{bmatrix}. \end{aligned} \tag{91}$$

The second partial derivatives $f^s_{xx,sub}, f^s_{xy,sub}$ of f on s are then given by:

$$\begin{aligned} f^s_{xx,sub} &= \frac{-n^s_{x,x}}{n^s_{x,z}}, \\ f^s_{xy,sub} &= \frac{-n^s_{x,y}}{n^s_{x,z}}. \end{aligned} \tag{92}$$

We then introduce the neighbour functions $N^s_{x,sub}$ for the first partial derivative on subtriangle s (note that, since the function we're applying them to right now is already a first partial derivative, the result is actually a second partial derivative):

$$\begin{aligned} N^s_{x,sub} &= \frac{1}{n^s_{x,z}} \left[(y^s - y^{s+2}), (y^{s+1} + y^{s+2} - 2y^i), (2y^i - y^s - y^{s+1}) \right], \\ N^s_{y,sub} &= \frac{1}{n^s_{x,z}} \left[(x^{s+2} - x^s), (2x^i - x^{s+1} - x^{s+2}), (x^s + x^{s+1} - 2x^i) \right]. \end{aligned} \tag{93}$$

This simplifies (92) to:

$$\begin{aligned} f^s_{xx,sub} &= f^i_x N^{s,1}_{x,sub} + f^s_{x,tri} N^{s,2}_{x,sub} + f^{s+1}_{x,tri} N^{s,3}_{x,sub}, \\ f^s_{xy,sub} &= f^i_x N^{s,1}_{y,sub} + f^s_{x,tri} N^{s,2}_{y,sub} + f^{s+1}_{x,tri} N^{s,3}_{y,sub}. \end{aligned} \tag{94}$$

We then substitute (85) and (89) into (94). From here on, only the xx-derivative is shown xy and yy follow similar derivations:

$$\begin{aligned} f^s_{xx,sub} &= N^{s,1}_{x,sub} \left(f^i N^i_x + \sum_{t=1}^{n} f^t N^t_x \right) \\ &\quad + N^{s,2}_{x,sub} \left(f^i N^{s,1}_{x,tri} + f^s N^{s,2}_{x,tri} + f^{s+1} N^{s,3}_{x,tri} \right) \\ &\quad + N^{s,3}_{x,sub} \left(f^i N^{s+1,1}_{x,tri} + f^{s+1} N^{s+1,2}_{x,tri} + f^{s+2} N^{s+1,3}_{x,tri} \right) \\ &= f^i \left(N^{s,1}_{x,sub} N^i_x + N^{s,2}_{x,sub} N^{s,1}_{x,tri} + N^{s,3}_{x,sub} N^{s+1,1}_{x,tri} \right) \\ &\quad + f^s \left(N^{s,2}_{x,sub} N^{s,2}_{x,tri} \right) \\ &\quad + f^{s+1} \left(N^{s,2}_{x,sub} N^{s,3}_{x,tri} + N^{s,3}_{x,sub} N^{s+1,2}_{x,tri} \right) \\ &\quad + f^{s+2} \left(N^{s,3}_{x,sub} N^{s+1,3}_{x,tri} \right) \\ &\quad + N^{s,1}_{x,sub} \sum_{t=1}^{n} f^t N^t_x. \end{aligned} \tag{95}$$

Again, we approximate the second derivative f^i_{xx} of f on vertex i by averaging over the surrounding subtriangles:

$$\begin{aligned} f^i_{xx} &= \frac{1}{n} \sum_{s=1}^{n} f^s_{xx,sub} \\ &= \frac{f^i}{n} \sum_{s=1}^{n} \left[N^{s,1}_{x,sub} N^i_x + N^{s,2}_{x,sub} N^{s,1}_{x,tri} + N^{s,3}_{x,sub} N^{s+1,1}_{x,tri} \right] \\ &\quad + \frac{1}{n} \sum_{s=1}^{n} \left[f^s \left(N^{s,2}_{x,sub} N^{s,2}_{x,tri} \right) \right] \\ &\quad + \frac{1}{n} \sum_{s=1}^{n} \left[f^{s+1} \left(N^{s,2}_{x,sub} N^{s,3}_{x,tri} + N^{s,3}_{x,sub} N^{s+1,2}_{x,tri} \right) \right] \\ &\quad + \frac{1}{n} \sum_{s=1}^{n} \left[f^{s+2} \left(N^{s,3}_{x,sub} N^{s+1,3}_{x,tri} \right) \right] \\ &\quad + \frac{1}{n} \sum_{s=1}^{n} \left[N^{s,1}_{x,sub} \sum_{t=1}^{n} f^t N^t_x \right]. \end{aligned} \tag{96}$$

First, we isolate the neighbour function N^i_{xx} of vertex i itself:

$$N^i_{xx} = \frac{1}{n}\sum_{s=1}^{n}\left[N^{s,1}_{x,sub}N^i_x + N^{s,2}_{x,sub}N^{s,1}_{x,tri} + N^{s,3}_{x,sub}N^{s+1,1}_{x,tri}\right]. \tag{97}$$

Then, we rearrange the sum terms containing f^{s+1} by lowering the summing index by one:

$$\begin{aligned}&\sum_{s=1}^{n}\left[f^{s+1}\left(N^{s,2}_{x,sub}N^{s,3}_{x,tri} + N^{s,3}_{x,sub}N^{s+1,2}_{x,tri}\right)\right]\\ =&\sum_{s=1}^{n}\left[f^{s}\left(N^{s-1,2}_{x,sub}N^{s-1,3}_{x,tri} + N^{s-1,3}_{x,sub}N^{s,2}_{x,tri}\right)\right].\end{aligned} \tag{98}$$

We do the same for the term containing f^{s+2}:

$$\begin{aligned}&\sum_{s=1}^{n}\left[f^{s+2}\left(N^{s,3}_{x,sub}N^{s+1,3}_{x,tri}\right)\right]\\ =&\sum_{s=1}^{n}\left[f^{s}\left(N^{s-2,3}_{x,sub}N^{s-1,3}_{x,tri}\right)\right].\end{aligned} \tag{99}$$

Then, by observing that the inner sum in the last term in (96) does not depend on the index of the outer sum, we rearrange this term to:

$$\sum_{s=1}^{n}\left[N^{s,1}_{x,sub}\sum_{t=1}^{n}f^tN^t_x\right] = \sum_{s=1}^{n}\left[f^sN^s_x\sum_{t=1}^{n}N^{t,1}_{x,sub}\right]. \tag{100}$$

Substituting (97), (98), (99) and (100) into (96) yields:

$$\begin{aligned}f^i_{xx} &= f^iN^i_{xx} + \sum_{s=1}^{n}\left[\frac{f^s}{n}\left(N^{s,2}_{x,sub}N^{s,2}_{x,tri} + N^{s-1,2}_{x,sub}N^{s-1,3}_{x,tri}\right.\right.\\ &\left.\left.+N^{s-1,3}_{x,sub}N^{s,2}_{x,tri} + N^{s-2,3}_{x,sub}N^{s-1,3}_{x,tri} + N^s_x\sum_{t=1}^{n}N^{t,1}_{x,sub}\right)\right]\\ &= f^iN^i_{xx} + \sum_{s=1}^{n}\left[\frac{f^s}{n}\left(N^{s,2}_{x,tri}\left(N^{s,2}_{x,sub} + N^{s-1,3}_{x,sub}\right)\right.\right.\\ &\left.\left.+N^{s-1,3}_{x,tri}\left(N^{s-1,2}_{x,sub} + N^{s-2,3}_{x,sub}\right) + N^s_x\sum_{t=1}^{n}N^{t,1}_{x,sub}\right)\right].\end{aligned} \tag{101}$$

As we see, this simplifies into the same form as (89):

$$f^i_{xx} = f^iN^i_{xx} + \sum_{s=1}^{n}f^sN^s_{xx}. \tag{102}$$

The neighbour functions for the second partial derivative f^i_{xx} (and, following the same derivation, f^i_{xy} and f^i_{yy}) of f on vertex i are therefore given by:

$$N^i_{xx} = \frac{1}{n}\sum_{s=1}^{n}\left[N^{s,1}_{x,sub}N^i_x + N^{s,2}_{x,sub}N^{s,1}_{x,tri} + N^{s,3}_{x,sub}N^{s+1,1}_{x,tri}\right], \tag{103}$$

$$N^i_{xy} = \frac{1}{n}\sum_{s=1}^{n}\left[N^{s,1}_{y,sub}N^i_x + N^{s,2}_{y,sub}N^{s,1}_{x,tri} + N^{s,3}_{y,sub}N^{s+1,1}_{x,tri}\right], \tag{104}$$

$$N^i_{yy} = \frac{1}{n}\sum_{s=1}^{n}\left[N^{s,1}_{y,sub}N^i_y + N^{s,2}_{y,sub}N^{s,1}_{y,tri} + N^{s,3}_{y,sub}N^{s+1,1}_{y,tri}\right], \tag{105}$$

$$\begin{aligned} N^s_{xx} = \frac{1}{n}\Big(&N^{s,2}_{x,tri}\left(N^{s,2}_{x,sub} + N^{s-1,3}_{x,sub}\right) \\ &+N^{s-1,3}_{x,tri}\left(N^{s-1,2}_{x,sub} + N^{s-2,3}_{x,sub}\right) + N^s_x\sum_{t=1}^{n}N^{t,1}_{x,sub}\Big), \end{aligned} \tag{106}$$

$$\begin{aligned} N^s_{xy} = \frac{1}{n}\Big(&N^{s,2}_{x,tri}\left(N^{s,2}_{y,sub} + N^{s-1,3}_{y,sub}\right) \\ &+N^{s-1,3}_{x,tri}\left(N^{s-1,2}_{y,sub} + N^{s-2,3}_{y,sub}\right) + N^s_x\sum_{t=1}^{n}N^{t,1}_{y,sub}\Big), \end{aligned} \tag{107}$$

$$\begin{aligned} N^s_{yy} = \frac{1}{n}\Big(&N^{s,2}_{y,tri}\left(N^{s,2}_{y,sub} + N^{s-1,3}_{y,sub}\right) \\ &+N^{s-1,3}_{y,tri}\left(N^{s-1,2}_{y,sub} + N^{s-2,3}_{y,sub}\right) + N^s_y\sum_{t=1}^{n}N^{t,1}_{y,sub}\Big). \end{aligned} \tag{108}$$

7.6.3 First-order derivatives on staggered vertices

In order to solve the SIA, we require a way to discretise the first-order partial derivatives of a function on the staggered vertices described in Sect. XXX. While it is possible to just take the average value of the derivatives on the vertices themselves, as defined by (89), the resulting value would be a linear combination of the function values on all the neighbours of these two vertices, resulting in an undesirable degree of numerical diffusion.

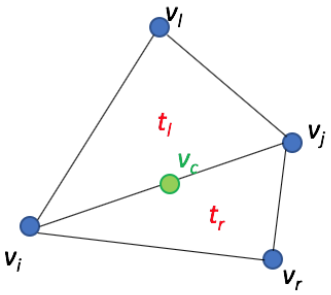


Figure 21: A very simple mesh, showing the staggered vertex v_c (green) connecting regular vertices v_i and v_j.

Consider the simple mesh in Fig. **??**, consisting of four vertices spanning two triangles. On the two adjacent triangles t_l and t_r, the first-order partial derivatives f^l_x, f^l_y, f^r_x and f^r_y are well-defined:

$$f^l_x = \frac{-n^l_x}{n^l_z} = \frac{f^i(y^j - y^l) + f^j(y^l - y^i) + f^l(y^i - y^j)}{x^i(y^j - y^l) + x^j(y^l - y^i) + x^l(y^i - y^j)}, \tag{109}$$

$$f^l_y = \frac{-n^l_y}{n^l_z} = \frac{f^i(x^l - x^j) + f^j(x^i - x^l) + f^l(x^j - x^i)}{x^i(y^j - y^l) + x^j(y^l - y^i) + x^l(y^i - y^j)}, \tag{110}$$

$$f^r_x = \frac{-n^r_x}{n^r_z} = \frac{f^i(y^r - y^j) + f^j(y^i - y^r) + f^r(y^j - y^i)}{x^i(y^r - y^j) + x^j(y^i - y^r) + x^r(y^j - y^i)}, \tag{111}$$

$$f^r_y = \frac{-n^r_y}{n^r_z} = \frac{f^i(x^j - x^r) + f^j(x^r - x^i) + f^r(x^i - x^j)}{x^i(y^r - y^j) + x^j(y^i - y^r) + x^r(y^j - y^i)}. \tag{112}$$

As before, we define the derivative f^c_x of f on v_c as the average of the values on the two adjacent triangles:

$$f_x^c = \frac{1}{2}(f_x^l + f_x^r), \tag{113}$$

$$f_y^c = \frac{1}{2}(f_y^l + f_y^r). \tag{114}$$

The purpose of the staggered mesh is to separate velocities on cell boundaries into components parallel and orthogonal to those boundaries. The component orthogonal to the boundary carries a flux from one cell into another, while the parallel component does not. On the mesh, this is complicated by the fact that the orientation of boundaries is not fixed. Let $\mathbf{u}$ be the vector pointing from v_i to v_j:

$$\mathbf{u} = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \tag{115}$$

The parallel and orthogonal derivatives f_{par}^c and f_{ort}^c of f on v_c are then given by:

$$f_{par}^c = f_x^c \frac{u_x}{|\mathbf{u}|} + f_y^c \frac{u_y}{|\mathbf{u}|}, \tag{116}$$

$$f_{ort}^c = f_x^c \frac{-u_y}{|\mathbf{u}|} + f_y^c \frac{u_x}{|\mathbf{u}|}. \tag{117}$$

It can be shown that the expression for f_{par}^c results in:

$$f_{par}^c = \frac{f_j - f_i}{|\mathbf{u}|}. \tag{118}$$

This is, of course, simply the slope of the line between v_i and v_j.

7.6.4 Least-squares discretisation scheme

An alternative scheme for discretising derivatives on an unstructured triangular grid, based on a (weighted) least-squares approximation, is given by Syrakos et al. (2017). This discretisation is arrived at by expressing the function values on the surrounding vertices as a Taylor expansion around v_i, and then solving the resulting (generally over-determined) matrix equation for the different derivatives. A derivation of the theoretical order of convergence is also given by Syrakos et al. (2017), showing that this discretisation is second-order convergent for the first-order partial derivatives. Here, we provide an extended derivation which also includes the second-order partial derivatives. This discretisation scheme is not included in UFEMISM, but is investigated here only for comparison.

The value $f(G_c)$ of f on the geometric centre G_c of the Voronoi cell of neighbouring vertex v_c, which is separated from that of vertex i by the vector $[\Delta x_c, \Delta y_c]$, can be expressed as a second-order Taylor expansion of f around v_i as follows (where $f_i = f(G_i)$, $f_{x,i} = f_x(G_i)$, etc., dropping the argument G_i for ease of notation):

$$f_c = f_i + \Delta x_c f_{x,i} + \Delta y_c f_{y,i} + \frac{1}{2}\Delta x_c^2 f_{xx,i} + \Delta x_c \Delta y_x f_{xy,i} + \frac{1}{2}\Delta y_c^2 f_{yy,i} + \mathcal{O}\left(\Delta x_c^3 . \Delta y_c^3\right). \tag{119}$$

As vertex v_i has n neighbours, this results in the following system of n linear equations (defining $\Delta f_c \equiv f_c - f_i$, dropping the truncation error $\mathcal{O}\left(\Delta x_c^3 . \Delta y_c^3\right)$, and introducing the vertex weights w_c for the weighted least-squares approximation:

$$\underbrace{\begin{bmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_n \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} \Delta x_1 & \Delta y_1 & \frac{1}{2}\Delta x_1^2 & \Delta x_1 \Delta y_1 & \frac{1}{2}\Delta y_1^2 \\ \Delta x_2 & \Delta y_2 & \frac{1}{2}\Delta x_2^2 & \Delta x_2 \Delta y_2 & \frac{1}{2}\Delta y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \Delta x_n & \Delta y_n & \frac{1}{2}\Delta x_n^2 & \Delta x_n \Delta y_n & \frac{1}{2}\Delta y_n^2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} f_{x,i} \\ f_{y,i} \\ f_{xx,i} \\ f_{xy,i} \\ f_{yy,i} \end{bmatrix}}_{z}. \tag{120}$$

Using matrix notation, this equation reads $Wb = WAz$, which is solved for z by writing:

$$z = \left(A^T W^T W A\right)^{-1} A^T W^T W b = M \beta_b. \tag{121}$$

Here, we have grouped the A and W terms into $M = \left(A^T W^T W A\right)^{-1}$ and $\beta_b = A^T W^T W b$. The symmetric matrix $A^T W^T W A$, which needs to be inverted to find M, reads:

$$A^T W^T W A = \sum_{c=1}^{n} w_c^2 \begin{bmatrix} \Delta x_c^2 & \Delta x_c \Delta y_c & \frac{1}{2}\Delta x_c^3 & \Delta x_c^2 \Delta y_c & \frac{1}{2}\Delta x_c \Delta y_c^2 \\ & \Delta y_c^2 & \frac{1}{2}\Delta x_c^2 \Delta y_c & \Delta x_c \Delta y_c^2 & \frac{1}{2}\Delta y_c^3 \\ & & \frac{1}{4}\Delta x_c^4 & \frac{1}{2}\Delta x_c^3 \Delta y_c & \frac{1}{4}\Delta x_c^2 \Delta y_c^2 \\ & & & \Delta x_c^2 \Delta y_c^2 & \frac{1}{2}\Delta x_c \Delta y_c^3 \\ & & & & \frac{1}{4}\Delta y_c^4 \end{bmatrix}. \tag{122}$$

The second term, β_b, is expressed as:

$$\beta_b = \sum_{c=1}^{n} w_c^2 \begin{bmatrix} \Delta x_c \Delta f_c \\ \Delta y_c \Delta f_c \\ \frac{1}{2}\Delta x_c^2 \Delta f_c \\ \Delta x_c \Delta y_c \Delta f_c \\ \frac{1}{2}\Delta y_c^2 \Delta f_c \end{bmatrix}. \tag{123}$$

The matrix M, which contains only information about mesh geometry, can be inverted in Matlab using the *inv* function, or in Fortran using the LAPACK linear algebra package. Once M has been calculated, the first partial derivative $f_{x,i}$ of f on i can be expressed as:

$$f_{x,i} = M(1,1)\sum_{c=1}^{n}\left(w_c^2 \Delta x_c \Delta f_c\right) + M(1,2)\sum_{c=1}^{n}\left(w_c^2 \Delta y_c \Delta f_c\right) + \frac{1}{2}M(1,3)\sum_{c=1}^{n}\left(w_c^2 \Delta x_c^2 \Delta f_c\right) + M(1,4)\sum_{c=1}^{n}\left(w_c^2 \Delta x_c \Delta y_c \Delta f_c\right) + M(1,5)\sum_{c=1}^{n}\left(w_c^2 \Delta y_c^2 \Delta f_c\right). \tag{124}$$

Since we defined that $\Delta f_c \equiv f_c - f_i$, we can once again introduce the neighbour functions N_x and rewrite this expression to read:

$$f_{x,i} = f_i N_{x,i} + \sum_{c=1}^{n} f_c N_{x,c}. \tag{125}$$

The neighbour functions are then given by:

$$N_{x,i} = -\sum_{c=1}^{n} w_c^2 t_{x,c}, \qquad N_{x,c} = w_c^2 t_{x,c}, \tag{126}$$

$$N_{y,i} = -\sum_{c=1}^{n} w_c^2 t_{y,c}, \qquad N_{y,c} = w_c^2 t_{y,c}, \tag{127}$$

$$N_{xx,i} = -\sum_{c=1}^{n} w_c^2 t_{xx,c}, \qquad N_{xx,c} = w_c^2 t_{xx,c}, \tag{128}$$

$$N_{xy,i} = -\sum_{c=1}^{n} w_c^2 t_{xy,c}, \qquad N_{xy,c} = w_c^2 t_{xy,c}, \tag{129}$$

$$N_{yy,i} = -\sum_{c=1}^{n} w_c^2 t_{yy,c}, \qquad N_{yy,c} = w_c^2 t_{yy,c}. \tag{130}$$

The constant terms $t_{x,c}$, $t_{y,c}$, $t_{xx,c}$, $t_{xy,c}$ and $t_{yy,c}$ are given by:

$$t_{x,c} = M(1,1)\Delta x_c + M(1,2)\Delta y_c + \frac{1}{2}M(1,3)\Delta x_c^2 + M(1,4)\Delta x_c \Delta y_c + \frac{1}{2}M(1,5)\Delta y_c^2, \tag{131}$$

$$t_{y,c} = M(2,1)\Delta x_c + M(2,2)\Delta y_c + \frac{1}{2}M(2,3)\Delta x_c^2 + M(2,4)\Delta x_c \Delta y_c + \frac{1}{2}M(2,5)\Delta y_c^2, \tag{132}$$

$$t_{xx,c} = M(3,1)\Delta x_c + M(3,2)\Delta y_c + \frac{1}{2}M(3,3)\Delta x_c^2 + M(3,4)\Delta x_c \Delta y_c + \frac{1}{2}M(3,5)\Delta y_c^2, \tag{133}$$

$$t_{xy,c} = M(4,1)\Delta x_c + M(4,2)\Delta y_c + \frac{1}{2}M(4,3)\Delta x_c^2 + M(4,4)\Delta x_c \Delta y_c + \frac{1}{2}M(4,5)\Delta y_c^2, \tag{134}$$

$$t_{yy,c} = M(5,1)\Delta x_c + M(5,2)\Delta y_c + \frac{1}{2}M(5,3)\Delta x_c^2 + M(5,4)\Delta x_c \Delta y_c + \frac{1}{2}M(5,5)\Delta y_c^2. \tag{135}$$

Syrakos et al. (2017) prove that the first-order partial derivatives $f_{x,i}, f_{y,i}$ are at least first-order accurate on any unstructured grid, and approach second-order accuracy when all neighbouring vertices are separated by

equal angles, and when the exponent q in the weights $w_c = \frac{1}{r_c^q}$ (with $r_c = \sqrt{\Delta x_c^2 + \Delta y_c^2}$) is chosen to be $q = \frac{3}{2}$. Although we will not attempt it ourselves, we very strongly suspect that their proof can be easily extended to show that, under these circumstances, the second-order partial derivatives approach first-order accuracy.

7.6.5 Convergence

Syrakos et al. (2017) provide a theoretical proof that their least-squares-based discretisation of the first-order partial derivatives is second-order accurate on "nice" vertices (i.e. vertices whose neighbours are separated by roughly the same angles, and whose distances to the vertex don't vary too much). If these conditions are not met, the convergence decreases to first-order. While they do not investigate the second-order partial derivatives, it makes sense intuitively that these would be first-order convergent with resolution, just as they are on a regular grid (which is a special case of a mesh, where all vertices are perfectly "nice").

We investigated the convergence behaviour of both the averaged-gradients discretisation scheme used in UFEMISM, and the least-squares approach from Syrakos et al. (2017). This was done by evaluating a simple analytical function of the form $f = \sin x \cos y$ on every vertex and its neighbours, and comparing the discretised derivatives resulting from both approaches to the analytical value. This was done on a mesh that was created by UFEMISM for the present-day geometry of the Antarctic ice sheet. The results of this experiment are shown in **??**. Panels a-e show the discretisation errors of all first- and second-order partial derivatives for the averaged-gradients discretisation used in UFEMISM, while panels f-j show the errors for the least-square approach from Syrakos et al. (2017). As can be seen, both results show generally the same results, with the discretisation errors being smallest where the mesh resolution is finest (i.e. at the grounding line).

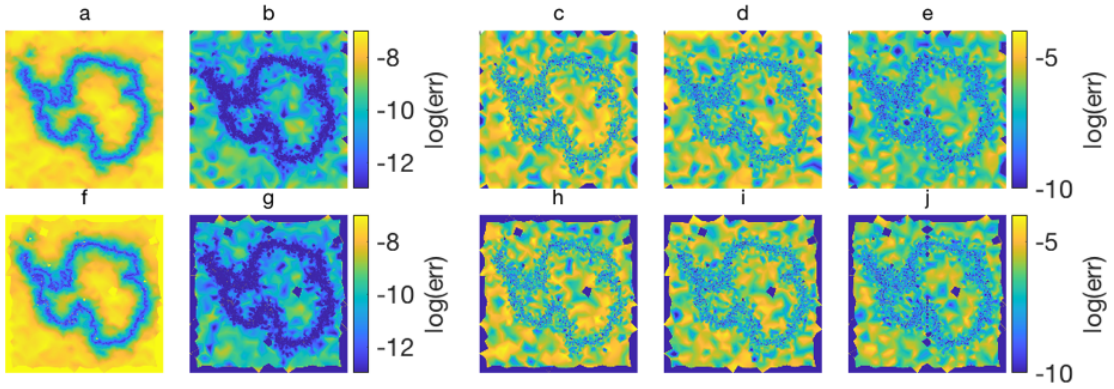


Figure 22: Panels a-e: discretisation errors for the averaged-gradients approach used in UFEMISM for, from left to right, f_x, f_y, f_{xx}, f_{xy} and f_{yy}. Panels f-j: the same for the least-squares approach from Syrakos et al. (2017).

To determine the order of convergence, discretisation errors are plotted against mesh resolution in **??**, with the panels ordered the same as in **??**. Again, the results for both discretisation methods are virtually identical, with both showing a clear second-order convergence for the first-order partial derivatives, and first-order convergence for the second-order partial derivatives.

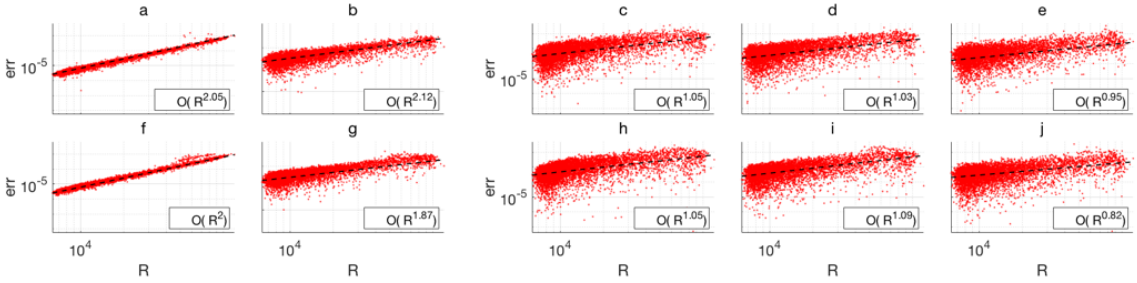


Figure 23: Panels a-e: discretisation errors vs. mesh resolution for the averaged-gradients approach used in UFEMISM for, from left to right, f_x, f_y, f_{xx}, f_{xy} and f_{yy}. Panels f-j: the same for the least-squares approach from Syrakos et al. (2017). Logarithmic fits have been made for all graphs, with the order of convergence shown in the labels.

7.7 Remapping

Whenever the mesh is updated, some of the modelled data fields need to be remapped from the old to the new mesh. While several generic mapping toolboxes are available, none of them are well-suited for UFEMISM. Mesh updates occur irregularly, with the frequency depending on the simulation (specifically on how fast the ice-sheet geometry evolves) such that a single glacial cycle simulation can already contain several hundred mesh updates. Mesh resolution can vary dramatically in space and time, as well as between simulations. This means that the remapping scheme must be both fast enough not to significantly affect model speed, and robust enough to handle all possible mesh combinations.

Following the objective of **flexibility** set out at the start, we decided to create a remapping scheme specifically for UFEMISM. The following section will give a brief description of how this has been done.

7.7.1 Theory

Whenever UFEMISM updates the mesh, (some of) the model data must be remapped from the old to the new mesh. Extensive preliminary experiments have shown that the his to be done using a conservative remapping scheme. If, for example, simpler schemes like nearest-neighbour, linear, quadratic, or even cubic interpolation are used, significant errors show up in the results of the schematic benchmark experiments. In the Halfar experiment, the ice thickness tends to "flatten out". In the EISMINT experiment, the basal temperature at the ice divide shows a significant warm bias. All of these anomalies disappear when a conservative remapping scheme is used. The conservative remapping scheme developed for UFEMISM is based on the work of Jones (1999). The mathematical principles behind this approach are quite straightforward, but creating an implementation that is both robust and fast enough to handle meshes of hundreds of thousands of vertices was surprisingly difficult.

Let there exist two meshes that both cover the same domain: a "source" mesh (indicated henceforth by the subscript s) and a "destination" mesh (subscript d). Suppose the source mesh is the one that existed before a mesh update, and the destination mesh is the newly generated mesh, so that we want to find a way to remap a data field f_s from the source mesh to the destination mesh. In the conservative remapping approach proposed by Jones (1999), the values f_d on the vertices of the destination mesh are found by integrating f_s over the regions of overlap A_{sd} between the Voronoi cells A_s of the source mesh, and the Voronoi cells A_d of the destination mesh:

$$f_d = \frac{1}{|A_d|} \sum_s \iint_{A_{sd}} f_s dA. \tag{136}$$

Since the Voronoi cells of a mesh generally are irregular polygons, these surface integrals are not very straightforward to solve. Instead, Jones (1999) uses the divergence theorem to convert the surface integrals $\int_{A_{sd}}$ over the region of overlap A_{sd} into a line integral around the perimeter C_{sd} of this region. However, Jones (1999), having an implementation in general circulation models in mind, derived his expressions in spherical coordinates. UFEMISM uses Cartesian coordinates, so the solution derived here will look slightly different.

In order to define the surface integral in (136), the function f_s is "undiscretised" using a simple first-order Taylor expansion:

$$f_s(x,y) = \overline{f_s} + \left(\frac{\partial f}{\partial x}\right)_s (x - x_s) + \left(\frac{\partial f}{\partial y}\right)_s (y - y_s). \tag{137}$$

Here, $\mathbf{r}_s = [x_s, y_s]$ is the geometric centre of the Voronoi cell of vertex s (which is generally not the same as the vertex itself!), and $\overline{f_s}$ is the discrete value of f on source vertex s. Substituting (137) into (136) yields:

$$f_d = \frac{1}{|A_d|} \sum_s \left[\iint_{A_{sd}} dA \left(\overline{f_s} + \left(\frac{\partial f}{\partial x}\right)_s (x - x_s) + \left(\frac{\partial f}{\partial y}\right)_s (y - y_s) \right) dA \right] \tag{138}$$

$$= \frac{1}{|A_d|} \sum_s \left[\overline{f_s} \iint_{A_{sd}} + \left(\frac{\partial f}{\partial x}\right)_s \left(\iint_{A_{sd}} x dA - x_s |A_{sd}| \right) + \left(\frac{\partial f}{\partial y}\right)_s \left(\iint_{A_{sd}} y dA - y_s |A_{sd}| \right) \right]. \tag{139}$$

Using the divergence theorem, we rewrite the three remaining surface integrals as line integrals:

$$\iint_{A_{sd}} dA = \oint_{C_{sd}} xdy, \tag{140}$$

$$\iint_{A_{sd}} xdA = -\oint_{C_{sd}} xydx, \tag{141}$$

$$\iint_{A_{sd}} ydA = \oint_{C_{sd}} xydy. \tag{142}$$

Substituting these into (138) yields:

$$f_d = \sum_s \left[w_0 \overline{f_s} + w_{1x} \left(\frac{\partial f}{\partial x} \right)_s + w_{1y} \left(\frac{\partial f}{\partial x} \right)_s \right]. \tag{143}$$

The three remapping weights w_0, w_{1x}, and w_{1y} are defined as:

$$w_0 = \frac{1}{A_d} \oint_{C_{sd}} xdy, \tag{144}$$

$$w_{1x} = \frac{-1}{A_d} \oint_{C_{sd}} xydx, \tag{145}$$

$$w_{1y} = \frac{1}{A_d} \oint_{C_{sd}} xydy. \tag{146}$$

7.7.2 Implementation

The basic theory of conservative remapping described in the previous section is fairly straightforward. Creating an implementation that is both robust (being able to handle any possible combination of meshes, including the possibility of overlapping vertices or lines) and fast (being able to remap data between meshes counting tens of thousands of thousands of vertices without significantly slowing down the model) turned out to be a surprisingly difficult problem, which took almost three months of full-time work to solve.

The main routines for remapping are CREATE_REMAPPING_ARRAYS_CONSERVATIVE (for creating the remapping weights, stored in the TYPE_REMAPPING_CONSERVATIVE) and REMAP_CONS_2ND_ORDER_2D/3D (for using them to remap a data field), which are located in the MESH_MAPPING_MODULE. A basic program that uses these routines to remap some data fields between a source mesh MESH_SRC and a destination mesh MESH_DST looks like this:

```
PROGRAM remap_data_between_meshes

! Declare the source and destination meshes
TYPE(type_mesh) :: mesh_src
TYPE(type_mesh) :: mesh_dst
! Declare the original (source) and remapped (destination) data fields
REAL(dp), DIMENSION(:), POINTER :: d1_src, d2_src, d3_src
REAL(dp), DIMENSION(:), POINTER :: d1_dst, d2_dst, d3_dst
INTEGER :: wd1_src, wd2_src, wd3_src, wd1_dst, wd2_dst, wd3_dst
! Declare the remapping arrays
TYPE(type_remapping_conservative) :: map

! Create the two meshes
...

! Create the remapping arrays
CALL create_remapping_arrays_conservative( mesh_src, mesh_dst, map)

! Allocate memory for the data fields
CALL allocate_shared_dp_2D( mesh_src%nV, d1_src, wd1_src)
CALL allocate_shared_dp_2D( mesh_src%nV, d2_src, wd2_src)
```

```
CALL allocate_shared_dp_2D( mesh_src%nV, d3_src, wd3_src)
CALL allocate_shared_dp_2D( mesh_dst%nV, d1_dst, wd1_dst)
CALL allocate_shared_dp_2D( mesh_dst%nV, d2_dst, wd2_dst)
CALL allocate_shared_dp_2D( mesh_dst%nV, d3_dst, wd3_dst)

! Remap the data
CALL remap_cons_2nd_order_2D( mesh_src, mesh_dst, map, d1_src, d1_dst)
CALL remap_cons_2nd_order_2D( mesh_src, mesh_dst, map, d2_src, d2_dst)
CALL remap_cons_2nd_order_2D( mesh_src, mesh_dst, map, d3_src, d3_dst)

! Deallocate the remapping arrays
CALL deallocate_remapping_arrays_conservative( map)

END PROGRAM remap_data_between_meshes
```

In the previous section, we showed that the values of f on the vertices of the destination mesh consist of linear combinations of the values of f on the vertices of the source mesh. As with the neighbour functions, we can calculate the linear coefficients (i.e. the remapping weights in eqs. (144)) once for a pair of meshes, then apply them to any and all data fields that need to be remapped. These remapping weights are expressed using line integrals around the perimeter C_{sd} of the area of overlap A_{sd} between the Voronoi cells of source vertex s and destination vertex d. Since the perimeter C_{sd} is made up of subsets of the perimeters of the two Voronoi cells, we can therefore calculate the line integrals around the Voronoi cells of all vertices of both meshes, keeping track of the Voronoi cells of the opposite mesh through which they pass. The results of these two calculations can then be combined to give the line integrals around each of the areas of overlap, which will then be used to give the remapping weights defined in (144).

Consider the (zoomed-in selections of) two meshes in **??**, panels A and C. The Voronoi cells of these meshes are shown in panels B and D, respectively.

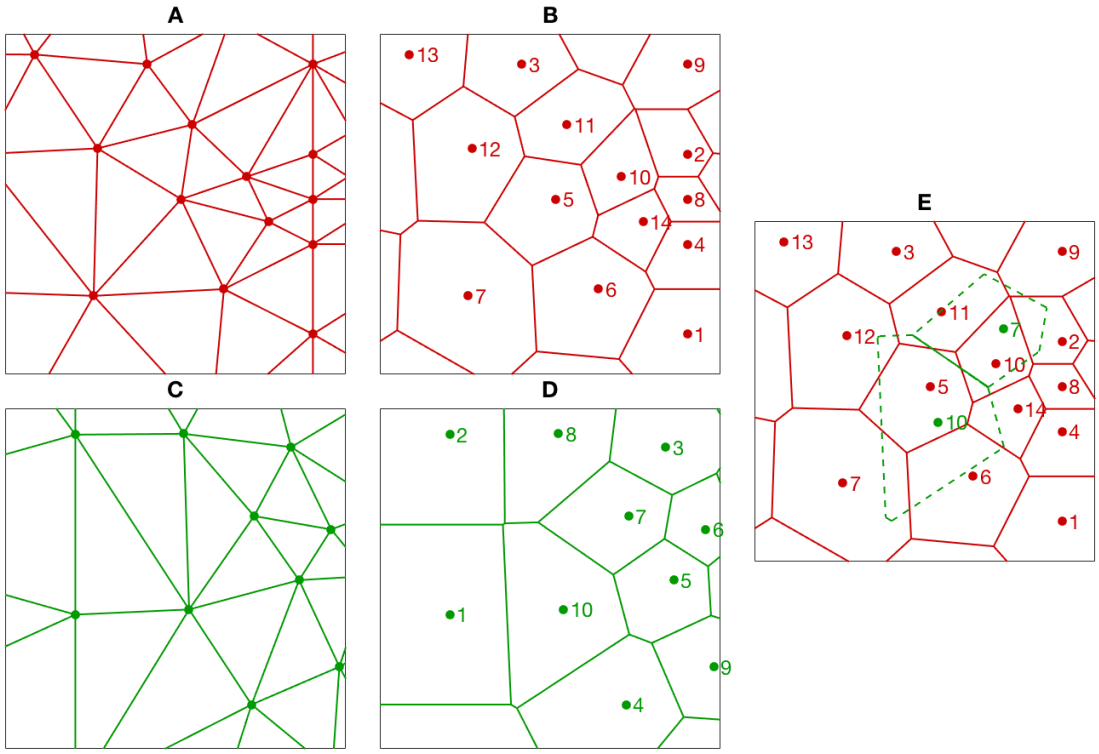


Figure 24: A) the source mesh, B) the Voronoi cells of the source mesh, C) the destination mesh, D) the Voronoi cells of the destination mesh, E) two Voronoi cells of the destination mesh overlaid on the source mesh

Suppose we want to calculate the three line integrals from (140) over the shared boundary between the Voronoi cells of destination (green) mesh vertices 7 and 10, shown by the solid green line in **??**, panel E. This

line passes through the source (red) mesh Voronoi cells of vertices 10, 5 and 11. In order to integrate over all the individual areas of overlap, the line integrals must be calculated over the three subsections of this line, and the results stored separately, recording the indices of both the source and destination vertices, lying on both the left-hand and right-hand sides of the line subsections. For this particular line, the results look like this:

Left-hand source vertex	Right-hand source vertex	Left-hand destination vertex	Right-hand destination vertex	$\int xdy$	$\int xydx$	$\int xydy$
10	10	10	7	...	...	...
5	5	10	7	...	...	...
11	11	10	7	...	...	...

The number of Voronoi cells of the opposite mesh through which a line passes is not generally known in advance. Finding the results shown above therefore requires an iterative algorithm which "traces" a line through a mesh; starting at one of the endpoints of a line, it will find the first point along the line where it crosses a Voronoi cell boundary of the mesh. The section of the line between the endpoint and the crossing point is integrated over, and is removed from the line, so that the crossing point now becomes the new end point. This is repeated until the entire line has been integrated over.

For the three sections of this particular line, the left-hand and right-hand source vertex indices are the same. This is not the case when Voronoi cell boundaries of the two meshes coincide something that, due to the deterministic nature of Rupperts algorithm, occurs quite frequently in UFEMISM. If the line integrals over coinciding line sections are calculated for both meshes, they will be double-counted when calculating the closed loop integrals, and the resulting surface integral will be incorrect. To prevent this, a flag must be raised when a coincidence is detected. When this is the case, only one of the two meshes is allowed to calculated the line integrals; the other one will simply list zero.

The subroutine CREATE_REMAPPING_ARRAYS_CONSERVATIVE, which calculates the remapping weights, is structured like this:

```
SUBROUTINE create_remapping_arrays_conservative

    ! Find the coordinates and relevant indices of the Voronoi boundary lines.
    CALL find_Voronoi_boundary_lines( mesh_src)
    CALL find_Voronoi_boundary_lines( mesh_dst)

    ! Integrate over all Voronoi cell boundary lines for both meshes
    CountCoincidences = .FALSE.
    CALL integrate_over_Voronoi_boundaries( mesh_src, mesh_dst, CountCoincidences)
    CountCoincidences = .TRUE.
    CALL integrate_over_Voronoi_boundaries( mesh_dst, mesh_src, CountCoincidences)

    ! Rearrange integral contributions from Aci to vertices
    CALL rearrange_contributions_from_lines_to_vertices( mesh_src)
    CALL rearrange_contributions_from_lines_to_vertices( mesh_dst)

    ! Integrate around domain boundary
    CALL integrate_around_domain_boundary( mesh_dst, mesh_src)

    ! Add contributions from mesh_src to mesh_dst
    CALL add_contributions_from_opposite_mesh( mesh_dst, mesh_src)

    ! Finish incomplete mesh_dst vertices
    CALL finish_incomplete_vertices( mesh_dst, mesh_src)

    ! Convert line integrals to remapping weights
    CALL calculate_remapping_weights_from_line_integrals( mesh_dst, mesh_src, map)

    ! Check if everything worked
    CALL check_if_remapping_is_conservative( mesh_src, mesh_dst, map)
```

END SUBROUTINE create_remapping_arrays_conservative

The different subroutines called here perform the following operations:

- FIND_VORONOI_BOUNDARY_LINES: First, the coordinates of the endpoints of all the line segments constituting the Voronoi cell boundaries are determined for both meshes. Most of the time these are simply the circumcentres of the triangles of the mesh; the only exceptions occur for the Voronoi cells of vertices lying on the domain boundary.

- INTEGRATE_OVER_VORONOI_BOUNDARIES: Then, the line integrals over all subsections of these line segments are calculated, as illustrated in Fig. **??**. It is possible that a Voronoi cell boundary of MESH_SRC will (partly) coincide with one of MESH_DST. In that case, that line must be integrated over only once, to prevent double-counting. The line-tracing algorithm that is used by INTEGRATE_OVER_VORONOI_BOUNDARIES can detect such coincidences; if the flag COUNTCOINDIDENCES is set to FALSE, the line integral over that segment will be listed as zero.

- REARRANGE_CONTRIBUTIONS_FROM_LINES_TO_VERTICES: Once the line integrals have been calculated, they are stored "per line". However, we want to have them available per vertex, so that they can be added together to give the loop integrals in Eqs. (144). This rearrangement step is done here.

- INTEGRATE_AROUND_DOMAIN_BOUNDARY: Since the Voronoi boundaries were defined using staggered vertices, the boundary sections lying on the domain boundary are not included. These are treated separately by this subroutine.

- ADD_CONTRIBUTIONS_FROM_OPPOSITE_MESH: The perimeter of a region of overlap between Voronoi cells of MESH_SRC and MESH_DST is made up of Voronoi cell boundaries of both meshes. In order to calculate the loop integral, the line integral sections from both meshes must be combined, which is done in this subroutine.

- FINISH_INCOMPLETE_VERTICES: Sometimes the Voronoi cell of a MESH_DST vertex will be completely enclosed by that of a MESH_SRC vertex, or vice versa. These relatively rare exceptions are handled separately by this routine.

- CALCULATE_REMAPPING_WEIGHTS_FROM_LINE_INTEGRALS: In this last step, the loop integrals are filed into (144) to calculate the remapping weights.

- CHECK_IF_REMAPPING_IS_CONSERVATIVE: To make sure everything works as intended, the remapping weights are applied to a dummy data field. On very rare occasions (estimated 1 in 100,000 vertices), minor errors occur when a vertex of one mesh lies exactly on the Voronoi cell boundary of a vertex of the other mesh. These erroneous vertices are detected by this routine, and their remapping weights are replaced by those representing simple bilinear interpolation.

8 Personal note

By Tijn Berends, February 2021.

In early 2019, the last year of my PhD at IMAU, I found myself with some spare time, sitting around and waiting for some particularly long ANICE simulations to finish (simulating the Lake Agassiz outburst flood during the last deglaciation, I believe). For some reason, the idea popped into my head that I should write a little Matlab program to create an unstructured grid. Not unusual for me - over the course of my PhD I wrote three different multi-purpose plotting programs, two text adventures, and a program for making 3D-printed ice sheets (I still have one lying on my desk!), all with the excuse of "my model's running!". Got to do something to keep the brain gears from rusting.

The first few attempts didn't look like much, but after a day or two I figured out a nice way to describe a triangular grid data-wise. A quick internet search and some late nights spent drawing triangles on paper got me to write out a very basic refinement scheme. By then the idea that I'd use this as the basis for a new ice-sheet model had already crept up on me. It just seemed so very elegant; triangles can fit together any way you want them to, so you can make them small or large wherever you please. The benefits of this seemed obvious to me, though I hadn't yet started digging into the literature on grounding-line migration or other resolution-related issues. Creating a proper algorithm for mesh adaptation and writing out the discretisation of the model equations took me almost half a year. Around July I had a version that could solve the SIA, but creating a solution for the SSA took until early December. At that point I had a model that could do basic simulations of the Antarctic ice sheet; reading in input data for geometry and climate, calculating a mass balance, solving the ice dynamics, and updating the mesh whenever the ice geometry changed. And it looked damn cool.

So, in January 2020 I put together a few nice, sleek-looking animations and went to visit Roderik (then my promotor, now still my supervisor). "Do you have a moment for me somewhere this week? I have something nice to show you.", I mailed late on Sunday evening. Not ten minutes later came the reply: "If it's really nice I have time first thing tomorrow morning." So, on Monday morning I spent just over an hour describing my pet project to Roderik. He asked a handful of questions throughout, otherwise just leaning back and observing. When I was finished, he was quiet for a moment, and he said to me, "Well, I think it's interesting.". He's not a very wordy man. However, early the next day he came to my office again, together with Heiko (Goelzer, then a postdoc at IMAU). "Show him what you showed me", Roderik told me. So I did. The two of them were very enthusiastic this time, and we agreed that I should finish up what I had into a paper, and they'd see if they could find a good way to put it to use. A few weeks later, on 31 January 2020, I defended my PhD thesis. In the laudatio, Roderik stated that the most memorable moment of my PhD happened at the very end of it when, "having already finished the thesis, one day Tijn showed up and said he had something nice to show me - and I nearly fell from my chair with amazement!". Not exactly how I remember that conversation, but I'll take it.

In May 2020 we wrote and submitted (I wrote, Roderik submitted) a proposal to NWO, asking them to fund a project to further developing UFEMISM and apply it to some nice palaeoglaciological research. While initially the plan was to use the funding (if we got it) to hire myself for a three-year postdoc, another position opened just after we'd submitted the proposal. Rather than gamble on a proposal that would take another 8 months at least to be processed, I decided to take the other position instead, and use the funding (if we got it) to hire someone else to carry out my ideas. In early February 2021, just over a year after I first showed my creation to anyone and almost two years since I started working on it, we received word that the proposal was accepted. That meant that UFEMISM was now officially, as they say, a thing. Although I'd maintained some very basic "documentation" just for myself, I've now set myself to the task of writing out a complete description of everything I've created. That will be the rest of this document; this final page right here I wrote simply to please my own ego. Both the paper and the documentation are, as is tradition, phrased as "we developed", "we chose", we, we, we, and I wanted to have just a little bit of "I" in there. Sue me.

9 References

Berends, C. J., de Boer, B., and van de Wal, R. S. W.: Application of HadCM3@Bristolv1.0 simulations of paleoclimate as forcing for an ice-sheet model, ANICE2.1: set-up and benchmark experiments, Geoscientific Model Development 11, 4657-4675, 2018

Berends, C. J., de Boer, B., Dolan, A. M., Hill, D. J., and van de Wal, R. S. W.: Modelling ice sheet evolution and atmospheric CO2 during the Late Pliocene, Climate of the Past 15, 1603-1619, 2019

Berends, C. J., de Boer, B., and van de Wal, R. S. W.: Reconstructing the evolution of ice sheets, sea level, and atmospheric CO2 during the past 3.6 million years, Climate of the Past 17, 361-377, 2021

Bintanja, R., van de Wal, R. S. W., and Oerlemans, J.: Global ice volume variations through the last glacial cycle simulated by a 3-D ice-dynamical model, Quaternary International 95-96, 11-23, 2002
Bueler, E. and Brown, J.: Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model, Journal of Geophysical Research 114, 200

Determann, J.: Numerical modelling of ice shelf dynamics, Antarctic Science 3, 187-195, 1991

Fettweis, X., Hofer, S., Krebs-Kanzow, U., Amory, C., Aoki, T., Berends, C. J., Born, A., Box, J. E., Delhasse, A., Fujita, K., Gierz, P., Goelzer, H., Hanna, E., Hashimoto, A., Huybrechts, P., Kapsch, M.-L., King, M. D., Kittel, C., Lang, C., Langen, P. L., Lenaerts, J. T. M., Liston, G. E., Lohmann, G., Mernild, S. H., Mikolajewicz, U., Modali, K., Mottram, R. H., Niwano, M., Nol, B. P. Y., Ryan, J. C., Smith, A., Streffing, J., Tedesco, M., van de Berg, W. J., van den Broeke, M. R., van de Wal, R. S. W., van Kampenhout, L., Wilton, D., Wouters, B., Ziemen, F., and Zolles, T.: GrSMBMIP: intercomparison of the modelled 1980-2012 surface mass balance over the Greenland Ice Sheet, The Cryosphere 14, 2020

Greve, R.: Application of a Polythermal Three-Dimensional Ice Sheet Model to the Greenland Ice Sheet: Response to Steady-State and Transient Climate Scenarios, Journal of Climate 10, 901-918, 1997

Huybrechts, P.: The Antarctic ice sheet and environmental change: a three-dimensional modelling study, Berichte zur Polarforschung 99, 1992

Huybrechts, P. and de Wolde, J.: The Dynamic Response of the Greenland and Antarctic Ice Sheets to Multiple-Century Climatic Warming, Journal of Climate, 12, 21692188, 1999

Janssens, I. and Huybrechts, P.: The treatment of meltwater re- tention in mass-balance parameterizations of the Greenland ice sheet, Ann. Glaciol., 31, 133140, 2000

Jones, P. W.: First- and Second-Order Conservative Remapping Schemes for Grids in Spherical Coordinates, Monthly Weather Review 127, 2204-2210, 1999

Laskar, J., Robutel, P., Gastineau, M., Correia, A. C. M., and Levrard, B.: A long-term numerical solution for the insolation quantities of the Earth, Astronomy and Astrophysics 428, 261-285, 2004

MacAyeal, D. R.: Large-Scale Ice Flow Over a Viscous Basal Sediment: Theory and Application to Ice Stream B, Antarctica, Journal of Geophysical Research 94, 4071-4087, 1989

Martin, M. A., Winkelmann, R., Haseloff, M., Albrecht, T., Bueler, E., Khroulev, C., and Levermann, A.: The Potsdam Parallel Ice Sheet Model (PISM-PIK) - Part 2: Dynamic equilibrium simulation of the Antarctic ice sheet, The Cryosphere 5, 727-740, 2011

Ohmura, A.: Precipitation, accumulation and mass balance of the Greenland ice sheet, Zeitschrift fur Gletscherkunde und Glazialgeologie, 35, 120, 1999

Pattyn, F., Schoof, C., Perichon, L., Hindmarsh, R. C. A., Bueler, E., de Fleurian, B., Durand, G., Gagliardini, O., Gladstone, R. M., Goldberg, D., Gudmunsson, G. H., Huybrechts, P., Lee, V., Nick, F. M., Payne, A. J., Pollard, D., Rybak, O., Saito, F., and Vieli, A.: Results of the Marine Ice Sheet Model Intercomparison Project, MISMIP, The Cryosphere 6, 573-588, 2012

Pollard, D. and DeConto, R. M.: Modelling West Antarctic ice sheet growth and collapse through the past

five million years, Nature 458, 329-332, 2009

Pollard, D.: A retrospective look at coupled ice sheet-climate modeiing, Climatic Change 100, 173-194, 2010

Roe, G. H.: Modeling precipitation over ice sheets: an assessment using Greenland, Journal of Glaciology 48, 70-80, 2002

Roe, G. H. and Lindzen, R. S.: The Mutual Interaction between Continental-Scale Ice Sheets and Atmospheric Stationary Waves, Journal of Climate 14, 1450-1465, 2001

Ruppert, J.: A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation, Journal of Algorithms 18, 548-585, 1995

Schoof, C.: Ice sheet grounding line dynamics: Steady states, stability, and hysteresis, Journal of Geophysical Research 112, , 2007

Singarayer, J. S. and Valdes, P. J.: High-latitude climate sensitivity to ice-sheet forcing over the last 120 kyr, Quaternary Science Reviews 29, 43-55, 2010

Syrakos, A., Varchanis, S., Dimakopoulos, Y., Goulas, A., and Tsamopoulos, J.: A critical analysis of some popular methods for the discretisation of the gradient operator in finite volume methods, Physics of Fluids 29, 2017

Tsai, V. C., Stewart, A. L., and Thompson, A. F.: Marine ice-sheet profiles and stability under Coulomb basal conditions, Journal of Glaciology 61, 205-215, 2015

Weis, M., Greve, R., and Hutter, K.: Theory of shallow ice shelves, Continuum Mechanics and Thermodynamics 11, 15-50, 1999

Williams, M. H.: A Linear Algorithm for Colouring Planar Graphs with Five Colours, The Computer Journal 28, 78-81, 1985

Winkelmann, R., Martin, M. A., Haseloff, M., Albrecht, T., Bueler, E., Khroulev, C., and Levermann, A.: The Potsdam Parallel Ice Sheet Model (PISM-PIK) - Part 1: Model description, The Cryosphere 5, 715-726, 2011