Last updated: March 22, 2023
Contact: latychev@fas.harvard.edu

Disclaimer

We have made a comprehensive effort to exclude user's data corruption when running our codes. A code crash may happen, most often due to specifying an incorrect path, however, not a single incident affecting data has occurred to date. Nevertheless, we cannot assume responsibility for loss of data and strongly suggest having a backup. Working with huge arrays of data may cause the system to run out of memory, overheat, etc.

Conventions & Abbreviations

- Folder names are in **bold blue**.
- Terminal commands are in **bold green**,
- Web site addresses are in blue,
- File and program names are plain **bold.**
- Equations and table entries are highlighted in yellow.

- SL = Sea Level
- GMSL = Global Mean Sea Level
- CE = Common Era
- I/O = Input/Output
- GUI = Graphical User Interface

Contents

Conventions & Abbreviations....1
Overview....1
Code....2
Ice Model....3
Ice Model Examples....4
Sea Level....6
Sea Level Examples....7
Topography....11

Overview

Folder **Data** has three subfolders, **Ice**, **SL** and **Topo**, which contain data files for the ice model, sea level predictions and topography, respectively, where the latter also includes flooding maps.

Folder **Code** contains interactive utilities compiled for common operating system platforms, Linux, MacOS and Windows (both 32 & 64 bits), in respective subfolders **Linux**, **MacOS**,

Win32, **Win64** to manipulate data. Most common operations on data are linear transformation and interpolation & integration on a gridded portion of a spherical surface. These tools are running off terminal (command window). Log session examples (screenshots of terminal I/O) are located in folder **Code/Log** and can be followed up to perform similar tasks. Most folders have a file **Contents.txt** with a brief description of their contents.

Code

We provide a selection of utilities from the Seakon project and refer to them as Seakon Utilities (SKU). SKUs are research-grade tools with no GUI, designed to be ran in serial (single core/CPU), in a fully interactive mode. Each utility is self-contained and requires the user to specify input data (location and type), processing options and where to place the output. Below is their list and a summary of the major relevant functionality on (mostly) scalar data. All utilities provide on-screen explanations of their options

rq2xyz interpolation of data on quadrilateral structures, ***single grid – multiple data*

rqgen generation of custom quadrilaterals (including true lat-lon), some format conversion

gsll2xyz interpolation on global lat-lon grids (including Gauss-Legendre) and patches, generation of respective grid coordinates, *single grid - single data*

mcfile_manip linear operations on group(s) of files (up to four groups)

mcfile_amv adding/subtracting components of a vector to matrices

mcfile_mask basic operations on data with mask, in particular, converting data file(s) to mask(s) based on value

cvpti interpolation (both linear and non-linear), integration and smoothing data on triangular grids, ***single grid – multiple data*

trimanip *various transformation of data on triangular grid, e.g., custom grid cuts, boundary contour extraction, etc.*

** *multiple data files defined on a single (common) source grid --> respective number of data files on a single (common) target grid, where the data is saved separately from the grid*

To run an executable, copy it to a certain location, **type its name and follow on-screen prompts and samples of terminal sessions provided in log files, called **utility_name_log_xx.txt** in **Code/Log/,** where **utility_name** corresponds to the name of the executable and **xx** is a two-digit integer tag [01, 02, ..] if it is used for more than one task. Provide the full path to data files if they are not in current directory.

** *Linux/MacOS: executable name is usually preceded by* **./**, *e.g.,* ***$./utility_name***
Windows: may require to include ***start****, e.g.,* **$start utility_name** *(or use system launcher)*
occasionally, the screen prompts are reported to appear in another window

All utilities can optionally generate visualization scripts for Paraview https://www.paraview.org/. If this is desired, please install Paraview and, when producing a visualization script, attach the ***.inp** extension to the name, for example, **my_script.inp**. To render a plot, do not use Paraview's import data option. Instead, either issue a terminal command **$paraview my_script.inp** or launch the application and open *.inp file(s) from Paraview menu (File-Open). In either case, press the green button Apply to see your graphics. Figures 1B, 3A and 4A-E in the text are produced in Paraview.

Ice Model

The ice model is constructed as a sequence of time slices (frames) of effective ice height distribution relative to 0-equilibrium (< 1000 CE), sampled every 20 years from 1000 CE to 2000 CE.

All frames are defined on a common quadrilateral grid i**ceM_xyz.dat**, dimensioned 381 x 821, where the ordering is by rows with the second index varying faster, i.e., 381 rows of (column) length 821. The grid is saved separately from the data in the following format: first line contains dimensions {312801 381 821}, where 312801 = 381*821, and is followed by 312801 rows of the respective global X Y Z (with the center at R = 0) in meters.

A complementary description is via two single-column files **iceM_lat.dat** & **iceM_lon.dat**, listing latitudes [-90, 90] and longitudes [0, 360], respectively, in degrees. Note that the coordinate section in all three data files have the same row dimension of 381 x 821, listing Lat, Lon or (x,y,z) at every grid point.

Each ice data frame is a scaled version of the lateral master configuration or lateral unit surface mask (**iceM.dat**) normalized to have the minimum height of 0 and the maximum height of 1. The densities of ice and water are assumed to be 920 and 1000 kg/m^3, respectively. To realize 1 mm of GMSL equivalent, we require the extra scaling factor of 141.494705 *m* as the maximum ice height, which is attained towards the edge of the mask by construction (see Fig 1B in the text). For GMSL = 7 mm, we need to further multiply it by 7, etc.

If we denote this factor as H and keep it constant (to realize a prescribed GMSL at e.g., t=1840 CE), the time scaling function can be expressed as H*U(t), where U varies between 0 and 1, forming a time envelope. For the model shown in Fig. 1C in the text,

t (*CE*)	**1**000	1100	1200	1300	1400	1600	1840	2000
U(t)	0.01	0.289	0.493	0.640	0.75	0.896	1.0	0.0

Note an arbitrary small (1%) value assigned at 1000 CE at the beginning of the melt history. The rest of the frames at every 20-year time increment is obtained via a cubic spline from 1000 CE to 1840 CE and a linear downward slope from 1840 CE to 2000 CE.

Ice frames normalized for GMSL=1mm are gzipped and packed into a tarball **icedata_GMSL_1mm.tar**. On Linux/MacOS, use the commands:
$tar -xvf icedata_GMSL_1mm.tar followed by **$gunzip ice_*.gz** to extract individual files **ice_1000**, **ice_1020**, **ice_1040, ..**, **ice_2000**, where the 4-digit tag corresponds to the actual calendar time (CE).

To interpolate lateral ice distribution in files **ice_xxxx**, from the 381 x 821 quadrilateral to any other structure on the sphere, use SKU **rq2xyz**. An acceptable target structure must have the dimension as the first entry on the first line followed by XYZ coordinates. SKU **gsll2xyz** and /or **rqgen** can be used to create a structured surface grid or point set, for example, a full spherical grid (**gsll2xyz** only), true Lat-Lon patch or a custom quadrilateral (**rqgen** only). Note that if the target grid is outside the source domain, we are unable to interpolate and need to assign a specified numerical value.

Ice Model Examples

As an example, consider interpolation of the lateral unit surface mask **iceM.dat** generated on quadrilateral **iceM_xyz** to a Lat-Lon patch making its bounding box. Run SKU **fmmax** to get Lat-Lon bounds from **iceM_lat.dat** and **iceM_lon.dat**, respectively. See **fmmax_log_01.txt** for the Lat part.

Angular ranges of **iceM_xyz**:

Latitudes [60.8529549 61.7206230]

Colatitudes [28.2793770 29.1470451]

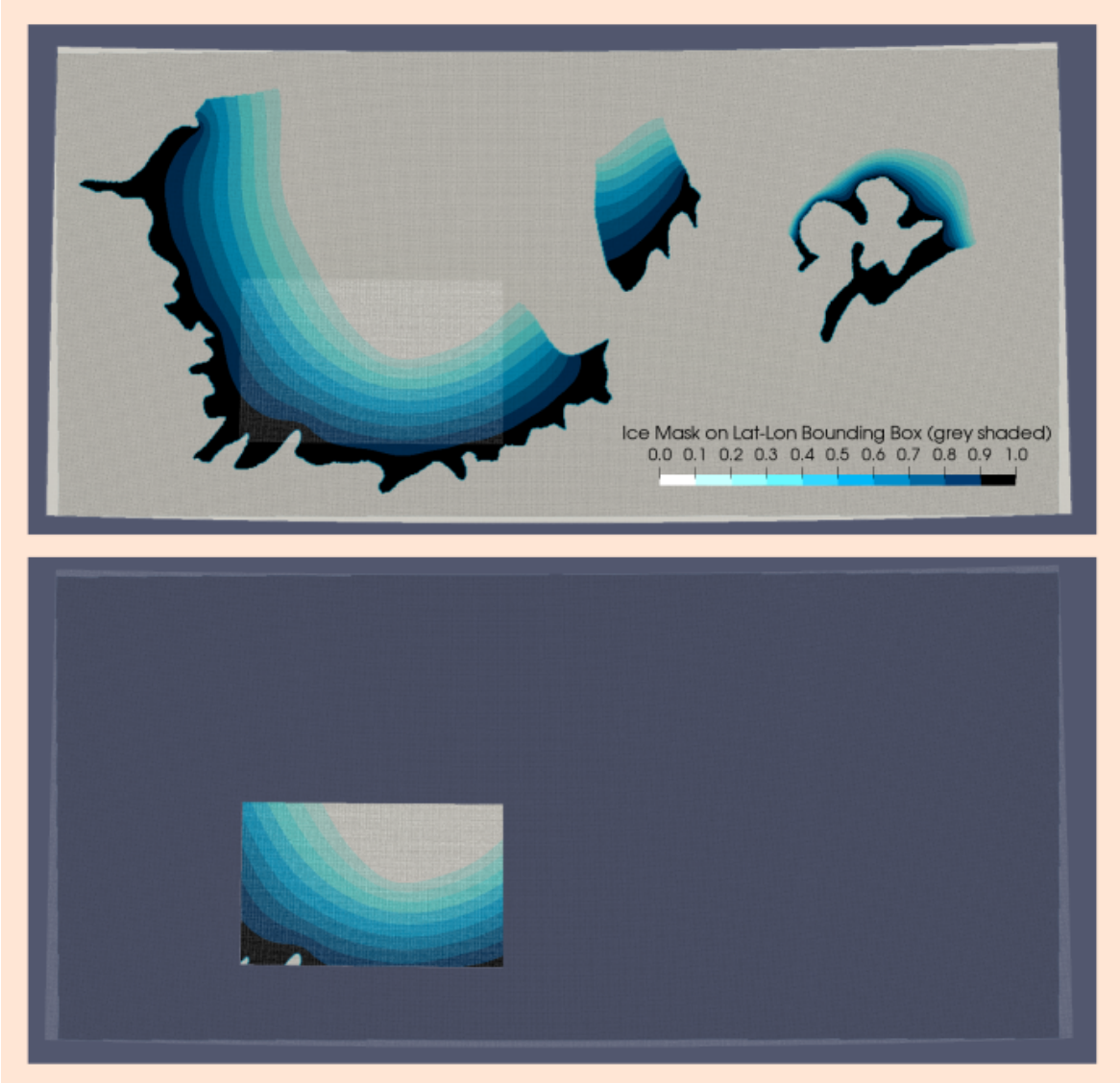
Longitudes [311.761261 315.651184]
or
[-48.238739 -44.348816]

For a closely matching resolution and bounding box dimensions 381 x 821, the angular increments in degrees for latitudes & longitudes are {0.002283337, 0.004743809} which corresponds to ~ 250 m of lateral resolution.
From the above table, we can create two small files listing latitudes (or colatitudes) and longitudes for the true Lat-Lon patch dimensioned 381 and 821, respectively, and call them **iceMLL_lat.dat** & **iceMLL_lon.dat.**

SKU **rqgen** can read them and create a full XYZ matrix as a target structure, required by SKU **rq2xyz**. See scripts **rqgen_log_01.txt** to create the target **iceMLL_xyz.dat** and **rq2xyz_log_01.txt** to produce the interpolated data output called **iceMLL.dat**. SKU **gsll2xyz** can be used to interpolate data on a Lat-Lon structure much faster than **rq2xyz**. It can also

generate XYZ coordinates for a patch without going through separate Lat & Lon files and SKU **rqgen**, but it requires entering explicit bounds. Note that we must specify colatitudes and then the actual ordering (either S --> N or N --> S). Script **gsll2xyz_log_01.txt** shows how to produce a Paraview script for **iceMLL.dat**, which can also be generated with SKU **rq2xyz** upon interpolation. Compare the resulting ice mask shown below to that in Fig 1B. We intentionally use the inverse colors to shade in grey the common area between the quadrilateral and its (generally larger) bounding box. The light grey margins are parts of the box outside the quadrilateral domain. The resulting unit ice mask on a Lat-Lon patch can be interpolated in a more efficient way to other structures with SKU **gsll2xyz**.

As an example, we run it first to create a smaller Lat-Lon test patch with Colat=[28.7, 29.0], Lon=[312.5, 313.5] sized 150 x 300, which are entered online and save it as **P_xyz.dat** (script **gsll2xyz_log_02.txt**), then run SKU **gsll2xyz** again and interpolate (script **gsll2xyz_log_03.txt**). The output **P.dat** can be visualized following the steps in **gsll2xyz_log_01.txt** for the entire ice mask.



Sea Level

Sea level output is saved on a part of the triangular computational surface grid, shown within the yellow contour in Fig 1B. A sample SL change from 1000 CE to 1450 CE on that grid is shown in Fig. 3A. The grid file (common to all SL time frames) is called **Tsurf**, an ASCII file with the following structure:

```
Number_of_nodes(=n)   Number_of_triangles(=nt)
X1  Y1  Z1                          <--- node 1
X2  Y2  Z2                          <--- node 2
. . . . . .
Xn  Yn  Zn                          <--- node n
J1_1   J1_2   J1_3                  <--- element 1
J2_1   J2_2   J2_3                  <--- element 2
. . . . . .
Jnt_1  Jnt_2  Jnt_3                 <--- element nt
```

where for any triangle k=[1, nt], its three anchors (Jk_1, Jk_2, Jk_3) are distinct grid nodes out of [1, n]. The numbering starts from 1. SL data is saved separately as a 1D column for each output time. The naming convention is similar to the ice data: **sl_1000**, **sl_1020**, **sl_1040**, ... **sl_2000**, totalling 51. These data files are gzipped and packed into **SL_GMSL_1mm.tar.**

Extensive test (GMSL = 1 ... 10 mm) have confirmed that we can save SL time history for GMSL=1 mm, as a template and scale it linearly for a prescribed GMSL value without re-running the 3D solver. This is because the response is linear wrt the load *and* impact of shoreline migration and assumed topography combined on the vertical (SL) predictions are negligible, correct to ~ 5 digits. This is certainly not true for the flooding area as discussed in the Topography Section.

For the production run, we set GMSL=7 mm and assume a background SL rise at a constant rate of 1.92 mm/y, therefore, in order to compute flooding maps subject to the assumed GMSL, background SL rise and present-day topography, we need to multiply the SL template matrices by 7 and add a linear function of time ranging from 0 at 1000 CE to 1.92 m (= 0.00192 m/y *1000 y) to account for the ambient SL rise. The latter may be considered as a vector **BkgSL**(1:51) and the operation can be formally written as

$$\mathbf{SL}(t\_k) = 7 * \mathbf{sl}(t\_k) + J * \mathbf{BkgSL}(k),$$

where k=[1,51] and J is a square 51 x 51 matrix of ones. This can be accomplished with SKUs **mcfile_manip** and **mcfile_amv**. See log scripts **mcfile_manip_log_01.txt** to scale by 7 and **mcfile_amv_log_01.txt** to add **BkgSL**, respectively. The resulting SL frames {**SL7b_1000, SL7b_1100, ... SL7b_2000} are** gzipped and packed into **SL_GMSL_7mm_BKG.tar**. These are used to compute time series of topography and flooding maps (see Topography Section). Note that t = 1450 CE falls in-between the actual output times at 1440 CE and 1460 CE and we obtain **SL7b_1450** by means of a cubic spline.

To handle data on triangular grid, we provide utilities **cvpti** and **trimanip**. Sea level is a reasonably smooth function and for many applications its native triangular grid **Tsurf** is fine enough (~ 1 km over the most of Eastern Settlement) to use a linear interpolation onto conventional structures such as Lat-Lon. A notable exception is the flooding map calculations where we require a much more refined topography grid (ideally sampled to meters rather than kilometers). In this case, a non-linear interpolation is employed.

The algorithm involves an iterative process of mid-edge values construction, followed by Laplace smoothing and, in general, may require some experimentation with a number of options. If an input field is discontinuous or, alternatively, an input grid is too coarse for that field, data over- and under-shots on the target grid are unavoidable, although they tend to be confined to a narrow margin tracing the discontinuity. For a well-behaved input data, effectively raising interpolation order renders a smooth function. This may be achieved by refining the input grid and data by injecting mid-edge points for a few levels first and then opting for interpolation (either linear or non-linear) or proceeding straight from source to target.

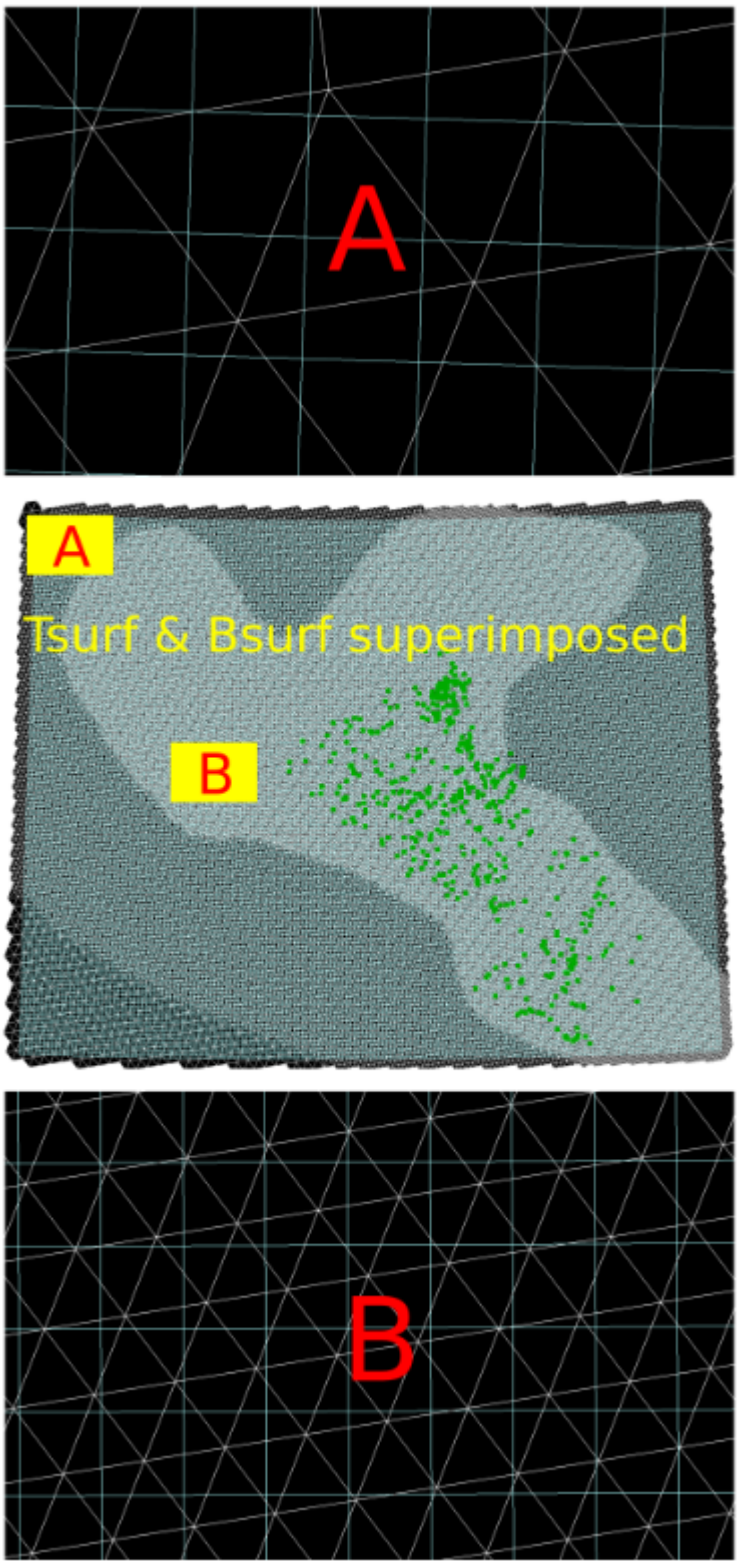
When working with multiple files, it is suggested to save interpolation map (to match target node and source element for a faster search), so that it is re-used for every other input file, thus saving CPU time. SKU **cvpti** is a potentially CPU/RAM-intensive.

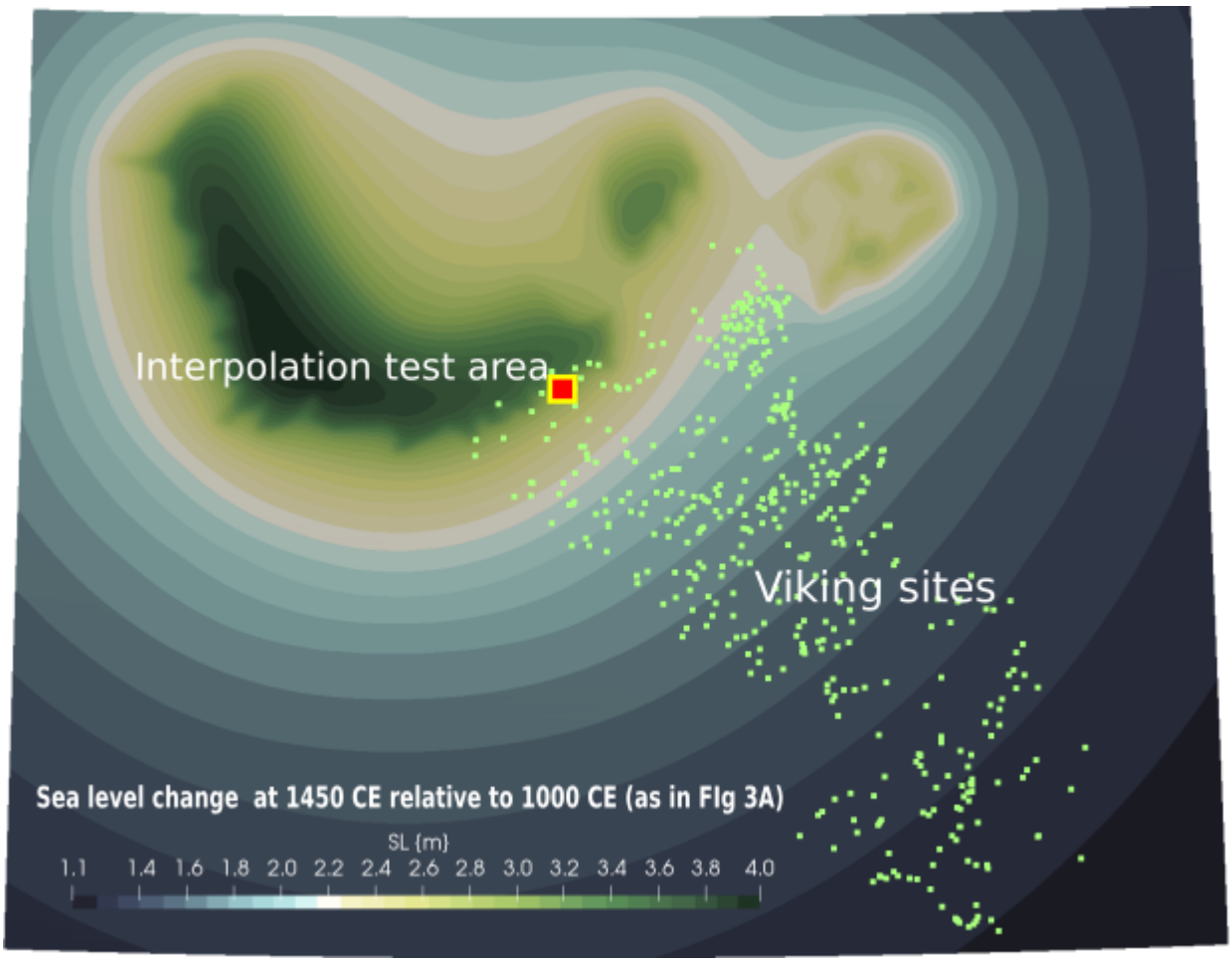
Sea Level Examples

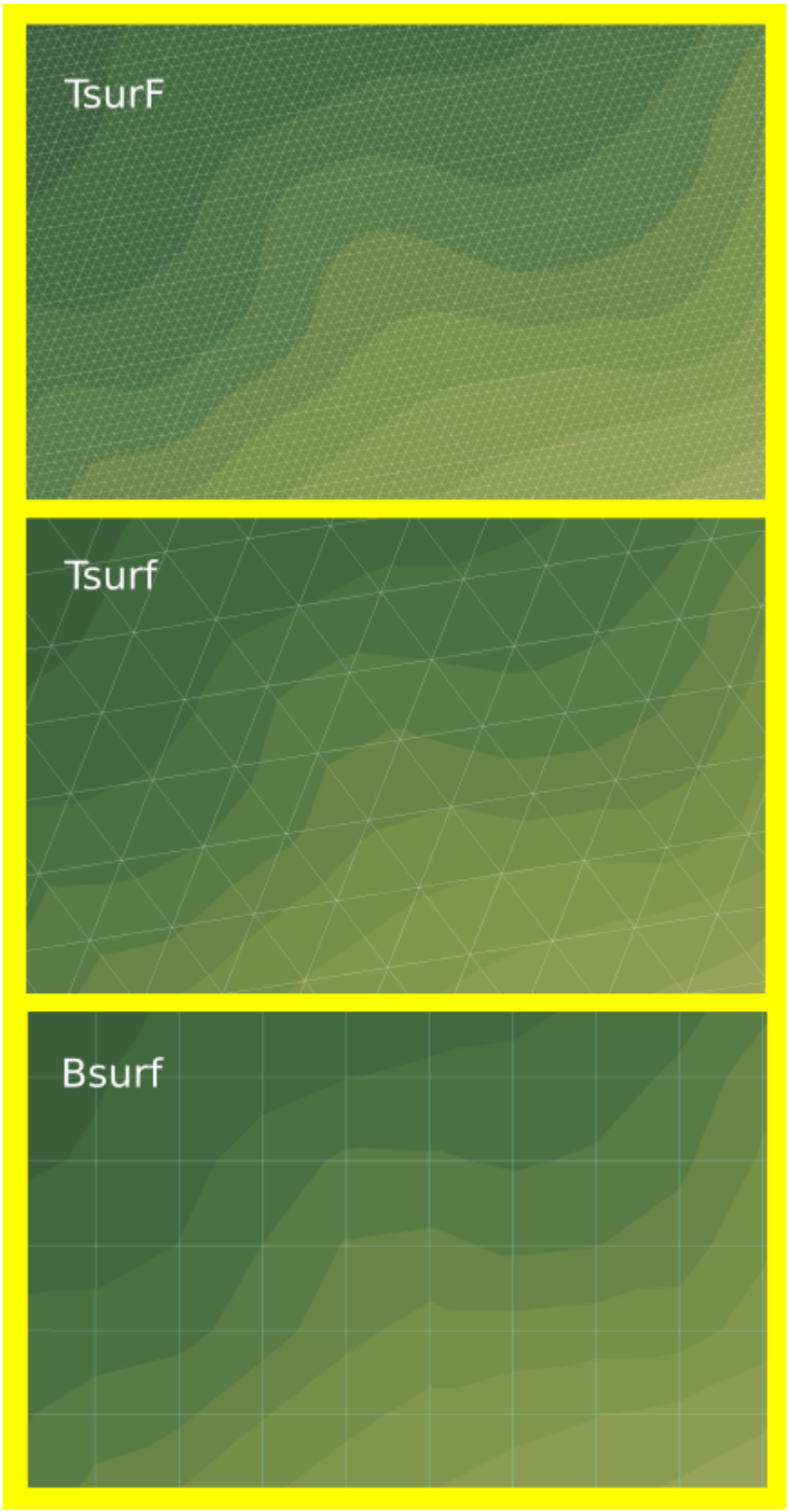
As a practical example, see script **cvpti_log_01.txt** on how to interpolate a SL frame (**SLb7_1450**) from triangular grid **Tsurf** to a Lat-Lon patch, **Bsurf**, created to be fully within the triangular grid domain. Fig 3A in the text displays the projected data **BSL7b_1450** on that Lat-Lon patch. The first figure below shows both grids superimposed - triangular (white lines) & Lat-Lon (blue lines) - and Viking sites (green dots). Two close-ups highlight two areas: *A* where both grids are of approximately the same resolution (~ 1 km) and *B*, where the triangular grid coarsens towards the boundary.

Note that due to variable resolution, **Tsurf** has 38159 nodes while **Bsurf** has 51657 (= 201 x 257). The next plot is largely a re-draw of Fig. 3A where we identify a small test area (yelow-red square) to test a non-linear interpolation by comparing SL on **Tsurf**, **Bsurf** and on an upsampled version of **Tsurf**, called **TsurF.** The latter is obtained by running **cvpti** three times successively with the option to refine grid and data (and saving both for the next restart). Grid **TsurF** has ~ 2.5 million nodes and the maximum resolution of ~ 100 m.

Depending on the application at hand, this process can be continued. Non-linear upsampling by mid-point injection is relatively fast, but interpolation from an upsampled source grid to any other target grid takes much longer than that from the original source.


A
A
Tsurf & Bsurf superimposed
B
B

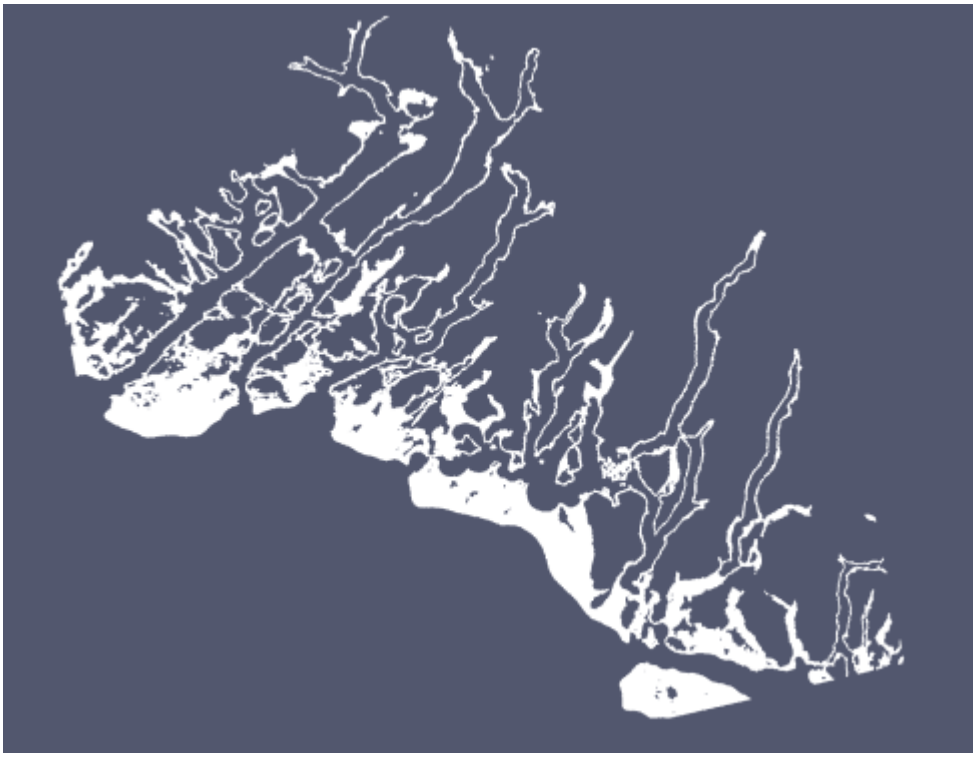

Interpolation test area
Viking sites
Sea level change at 1450 CE relative to 1000 CE (as in Fig 3A)
SL (m)
1.1 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0


TsurF
Tsurf
Bsurf

Topography

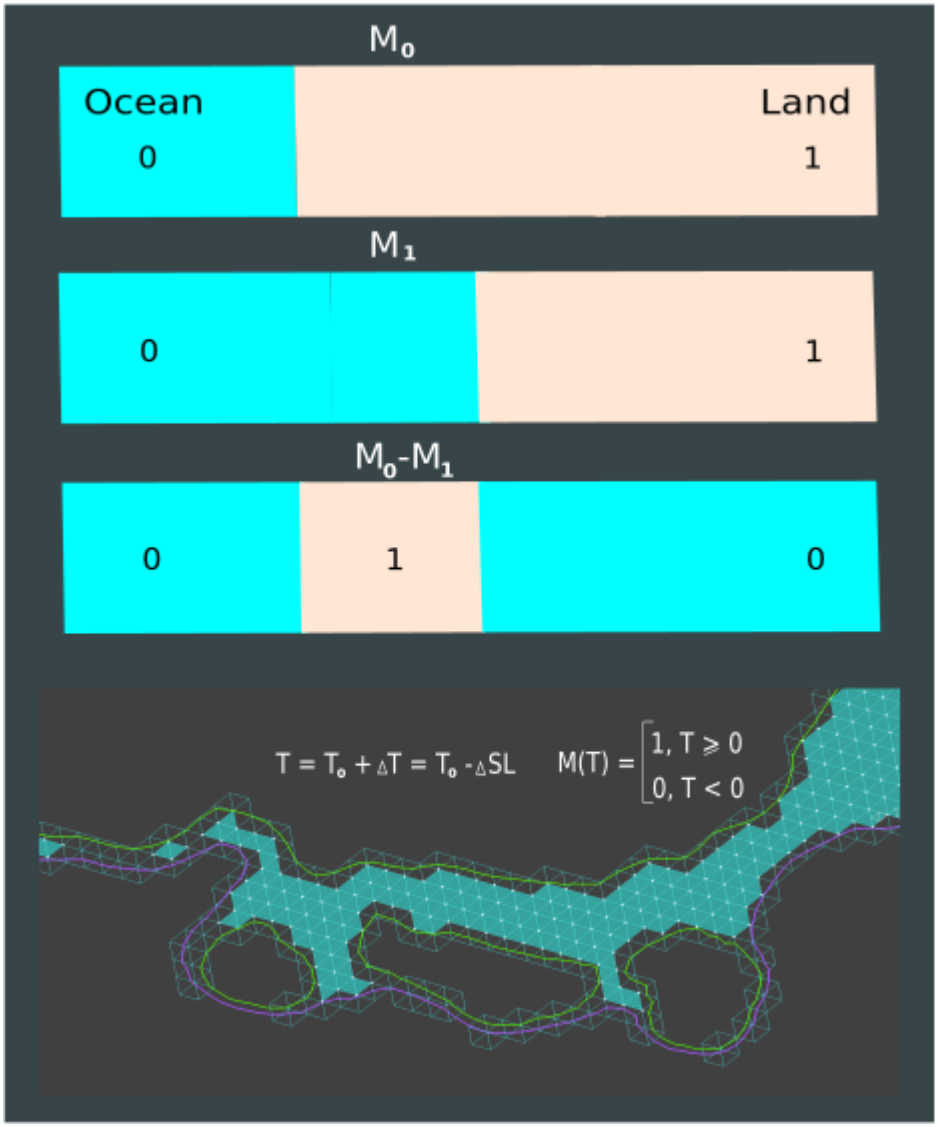
To run the SL solver, we require a sea floor bathymetry and land topography (further topography for either of the two). As mentioned above, it does not to be super fine, and, in fact, will be restricted by the computational grid resolution (~ 1 km at the max). Thus, an ~489 m resolution topography downloaded from GMRT site https://www.gmrt.org/GMRTMapTool/ for the Southern Greenland is sufficient for the SL simulations.

In order to reconstruct time series of the lateral flood area, accurate topography slopes play a critical role. Since we are interested in the shoreline area only and have SL predictions, we begin by constructing a suitable triangular grid covering the anticipated shoreline migration area. A rather laborious yet conceptually simple iterative process is outside the scope of these notes, but it can be completed with SKUs **gsll2xyz** (interpolation from Lat-Lon topography patches) and **trimanip** (cut triangular grid and data based on topography value to be within the expected dSL around zero plus some safe margin to avoid loss of data on steep cliffs). The topography grid is constructed iteratively, starting from a coarse grid–topography pair and followed up by grid-data refinement and cut (trimming up the grid to be closer to the shoreline), then repeating the previous step, while interpolating topography from a progressively finer topography patches (tiles) and increasing their number (often due to the size restriction set by the data providers). The final present-day, 12-million-point topography grid is called **y_15**. It has a near-uniform resolution of 15 meters, shown in white below. On the last iteration, the topography data is projected from nine 10-meter resolution tiles covering the Eastern Settlement with some overlaps. The resulting data file is called **yT_15**. Both **y_15** and **yT_15** are found in folder **Topo**.



Folder **Topo/Tiles** contains the 9 tiles downloaded from the NOAA site https://www.ncei.noaa.gov/maps/bathymetry/ along with a brief description, in **tiles_readme.txt,** including their angular bounds, suitable for entering into SKU **gsll2xyz** for interpolation onto **y15** to produce **yT_15**.

To generate flooding maps, we construct a time series of topography, convert them to ocean-land masks at the respective times and subtract from the reference mask at equilibrium (the start of simulations at 1000 CE). Integrating a mask as a surface function yields a square area (where mask = 1). Note that all these operations must be performed on the same grid (**y_15)**. To generate plots in Fig. 4 showing the flooded area, we cut the grid with the mask difference data and retain grid nodes corresponding to the mask difference = 1**.** The process is summarized below.



Note that cutting a grid has a principal uncertainty equal to the size of the grid element (triangle). A better answer is an “average” (between the green and purple bounding curves at the grid image above), but we cannot split an element without refinement. On the other hand, integration over the grid elements with the 0/1 mask to compute the area of flood is a more accurate process, where we can naturally account for fractions of the elements, depending on the quadrature rule. A simple (1/3, 1/3, 1/3) first-order triangular rule is used.

It is convenient to construct an initial topography (at 1000 CE) from the following relation, observing that dSL = -dT

$$T\_init = T\_present + dSL\_present,$$

where dSL_present is SL change at present with respect to the initial state, such as **SL7b_2000** in folder **SL**, except we must have the latter on the same grid **y_15**. Scripts **cvpti_log_02.txt** & **mcfile_manip_log_02.txt** provide details on how to perform interpolation and addition operations, respectively. Once the initial topography is obtained, we can construct a sequence of topography data files for each available SL output at time t

$$T(t) = T\_init – dSL(t).$$

Folder **Topo/T** contains topography data files from 1000 CE (**Ty_init**) to 1450 CE, where the 4-digit tag reflects the time.

Next, as outlined above, we convert **Ty_xxx** topography data to 0/1 integer masks **M_xxxx** based on value, assigning 0 if T <= 0 and 1 if T > 0. See script **mcfile_mask_log_01.txt** for an example on a single topography file, but it can be done on a number of files at once. Finally, we subtract those from initial topography mask **M_init** to get a mask difference (files **dM_xxxx)** suitable for computing the flooded area. Mask difference files are provided in folder **Topo/M**. A sample calculation for the flooded area can be found in script **cvpti_log_03.txt**.

For a basic manipulation and visualization of data on triangular grids, SKU **trimanip** can be used. Script **trimanip_log_01.txt** shows a process of reading **dM_1450**, selecting the points with dM=1, trimming the grid to a manageable size of 1 million points (down from the original 12) by removing zeroes and generating a visualization script to create Fig.4(A-E) in the text.