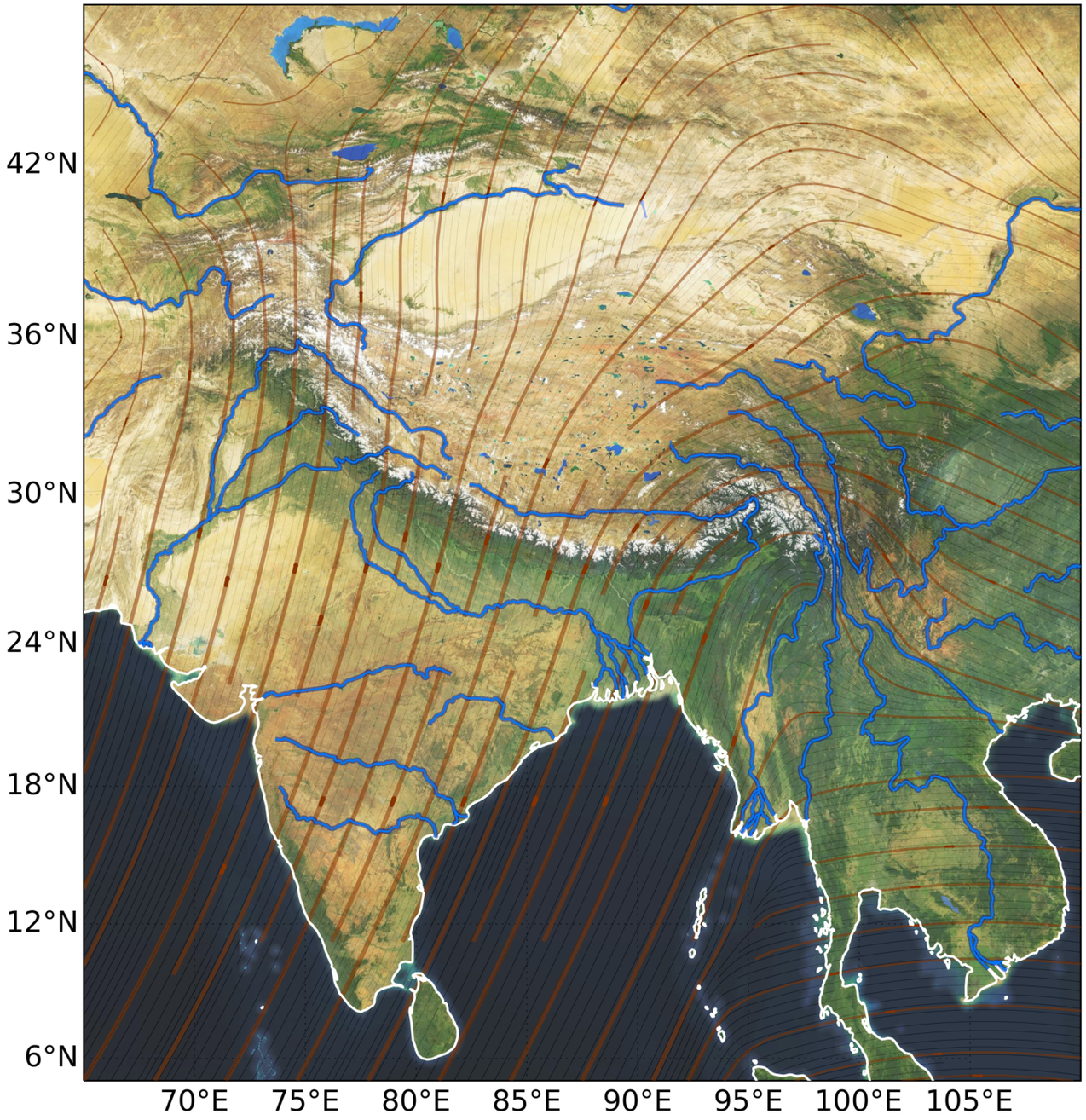
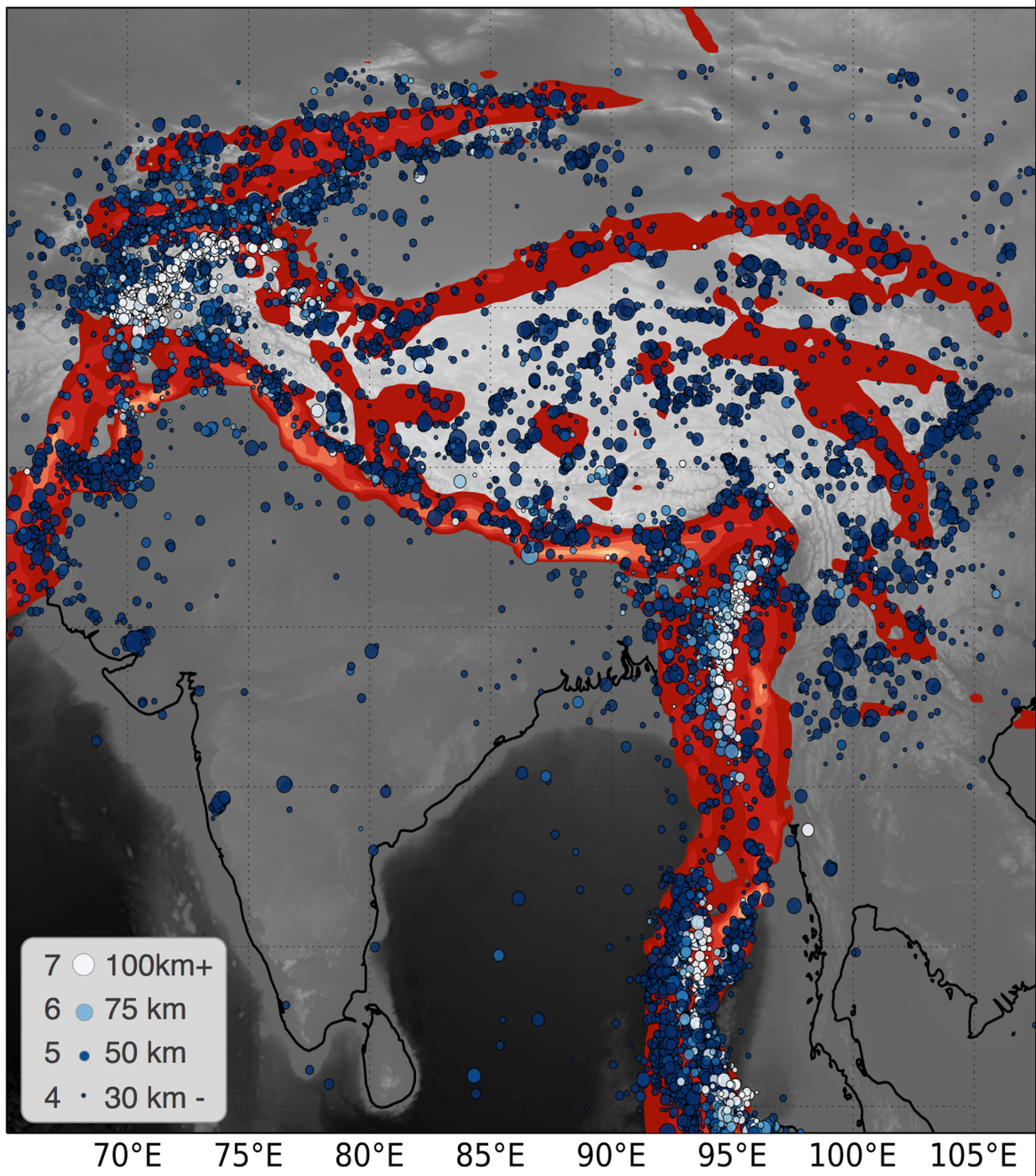
THE UNIVERSITY OF MELBOURNE

Parallel Landscape Evolution Models for Coupling to Large Scale Tectonics (Intro to Quagmire algorithm)

Louis Moresi, Ben Mather, Romain Beucher …

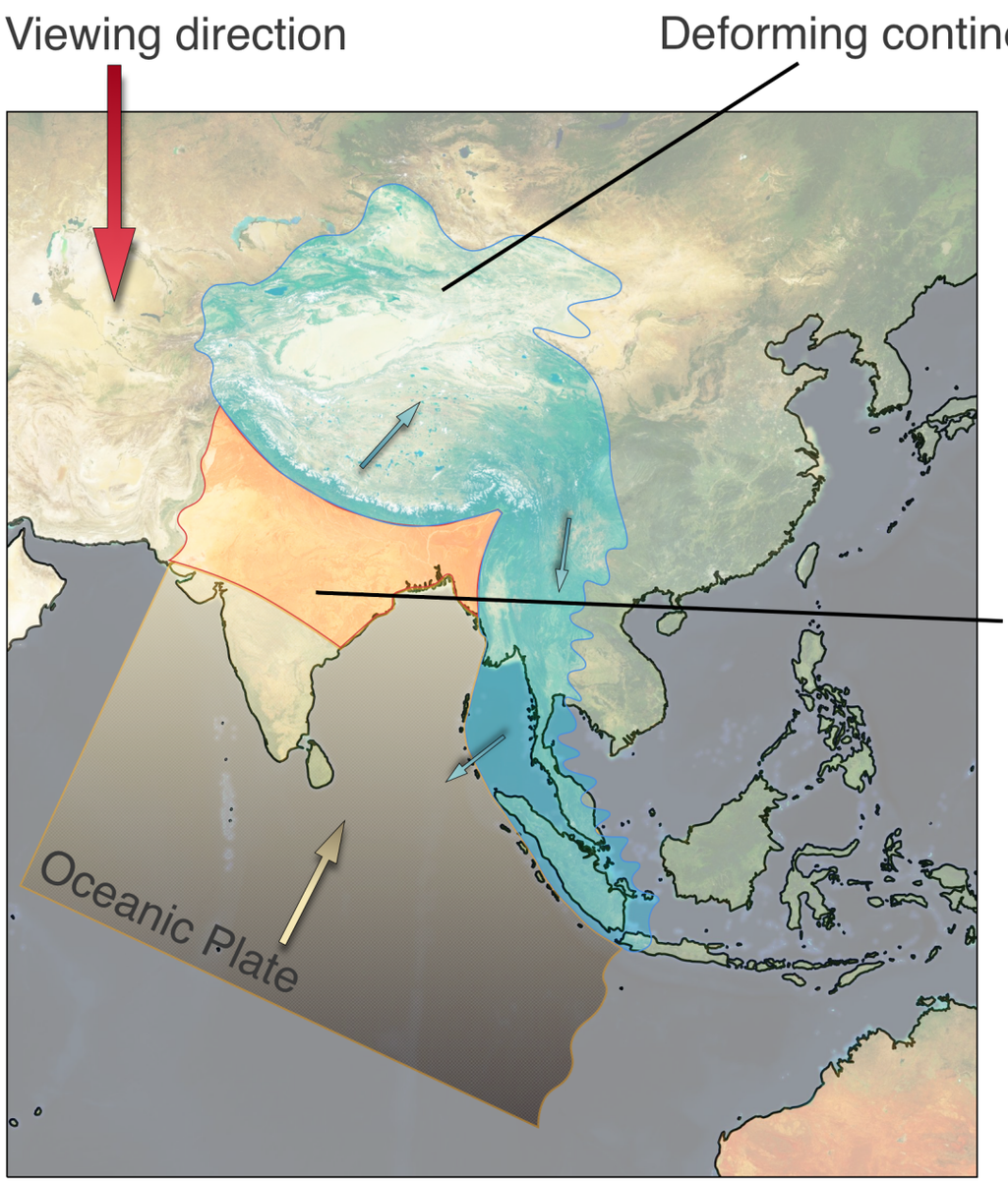
Inspiration



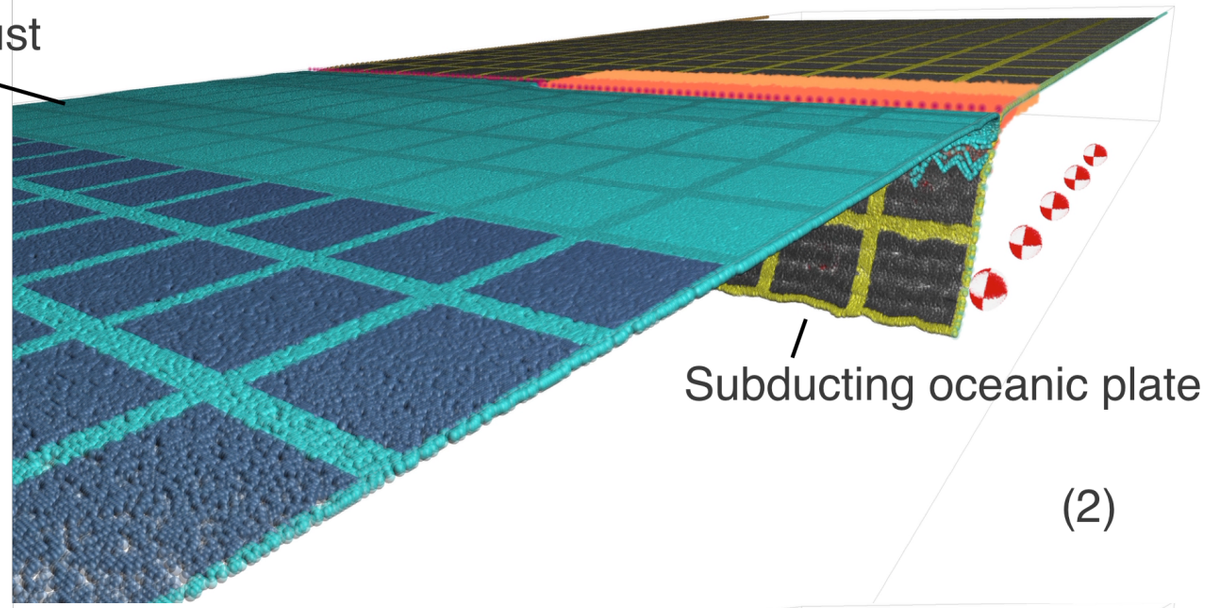


River systems in the largest collision zone are clearly out of equilibrium with the long-wavelength topography of the region. They carry information about the history of the deformation. The eroded material is deposited in marine sediments after long-range transport

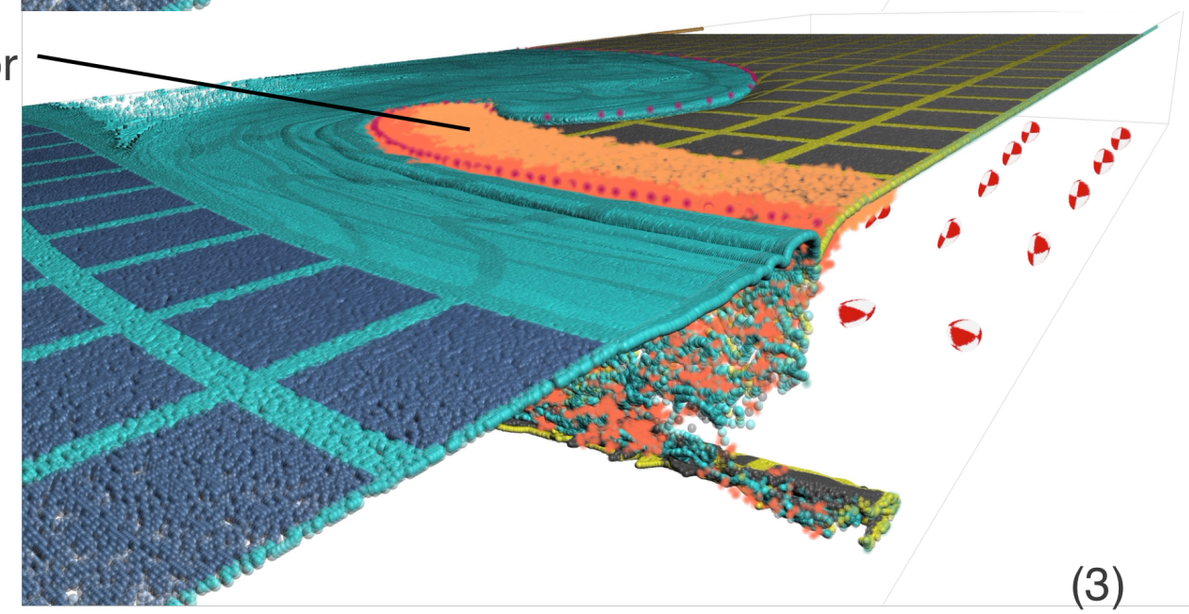
Tectonic models



(1)



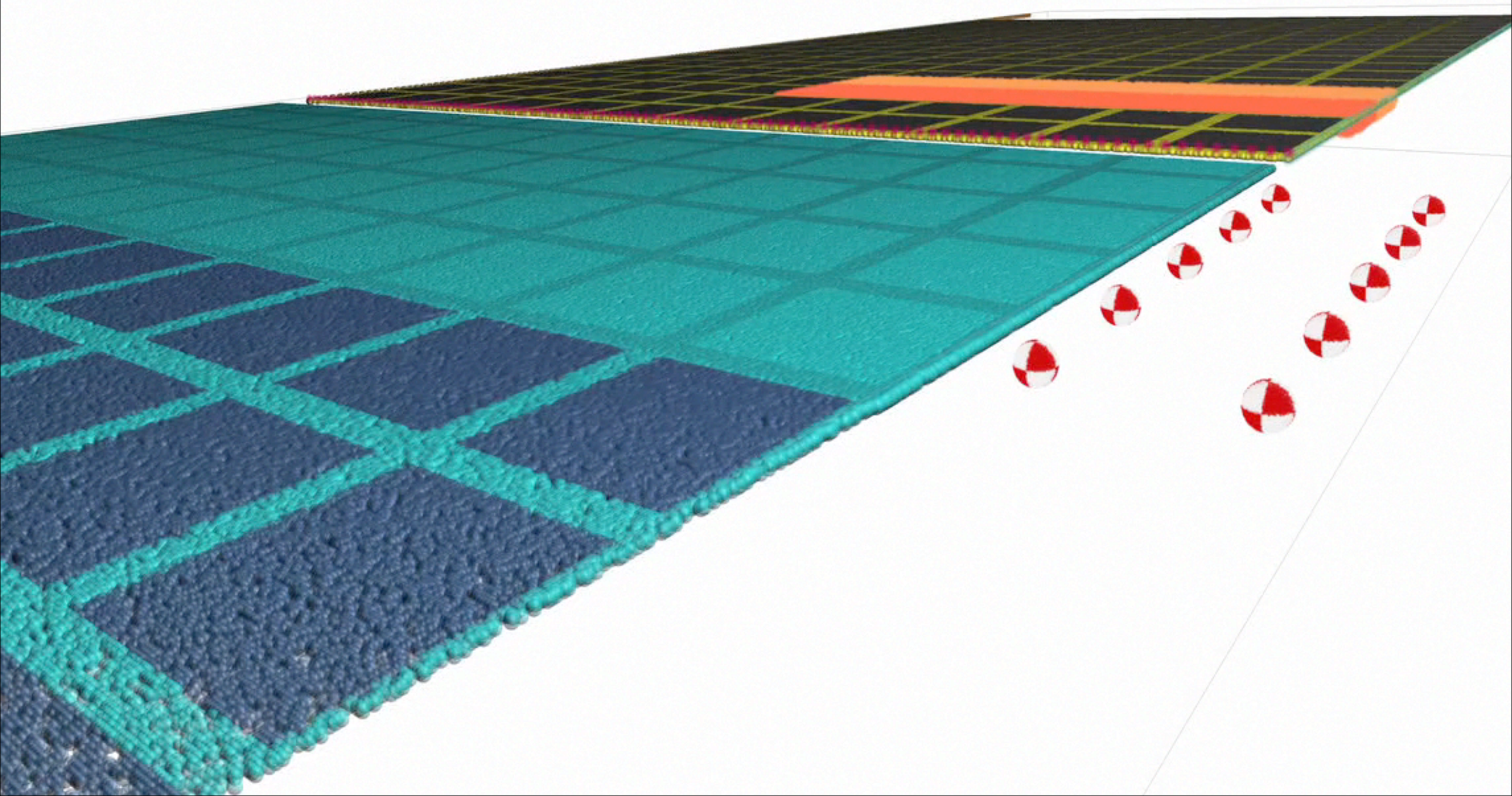
(2)



(3)

We can build sophisticated models of the tectonic deformation. We can also build sophisticated models of river formation and sediment transport. Coupled models are a work in progress. The challenges in building general models of river processes are interesting with some similarities to localisation problems in continuum mechanics.

Tectonic Models

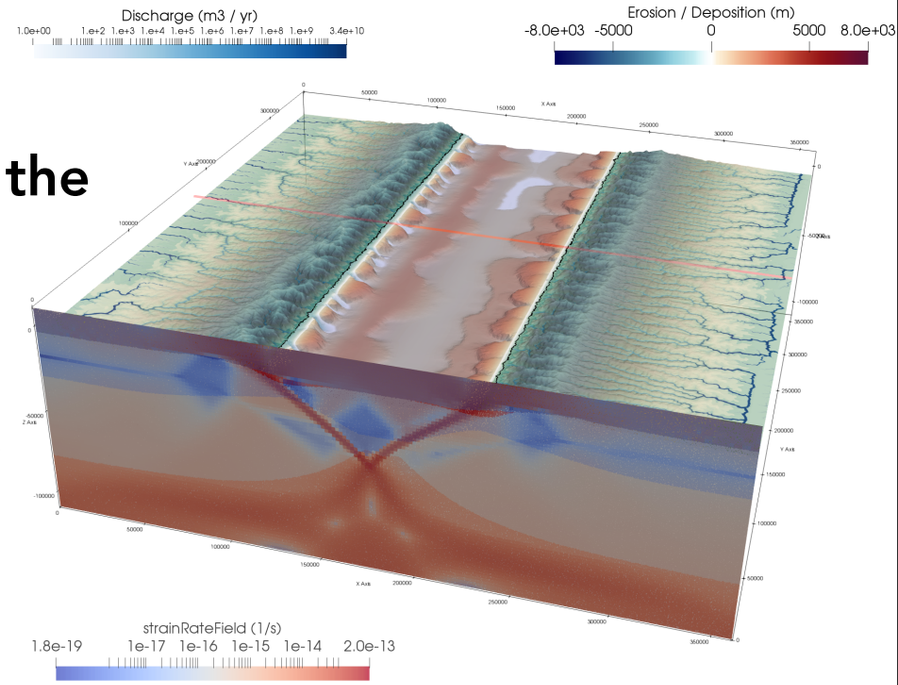


Modelling challenges & approaches

There are 2 different “regimes” where modelling challenges arise —

Models which attempt to match the pattern of rifting and forward model stratigraphy as an observable constraint on the modelling.

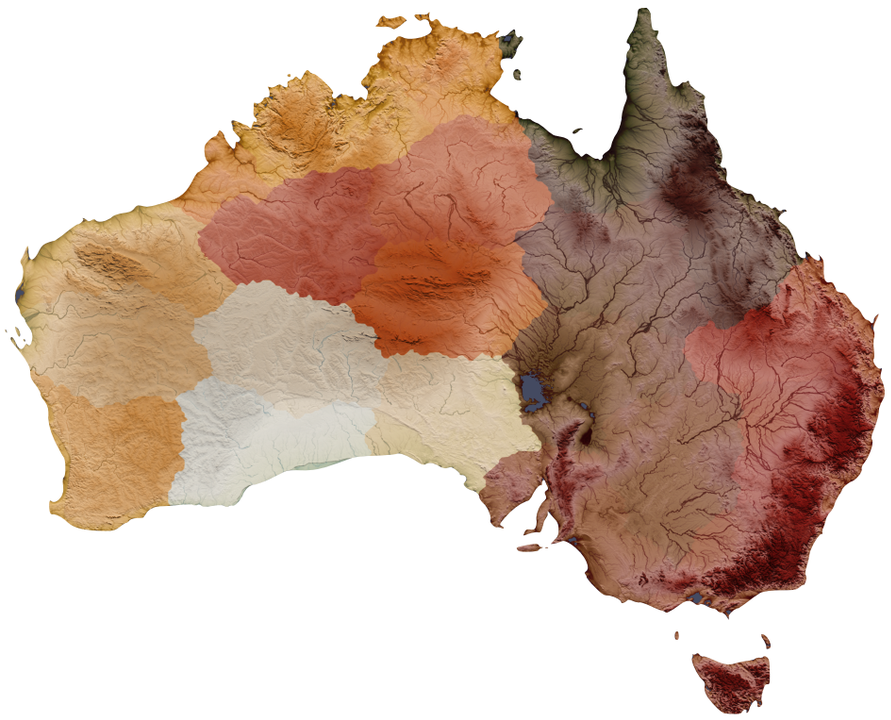
We use the Basin Genesis Hub python / numpy based code `pyBadlands` coupled to our tectonic forward modelling code `Underworld` for this work (Beucher & Salles). (https://github.com/badlands-model/pyBadlands).



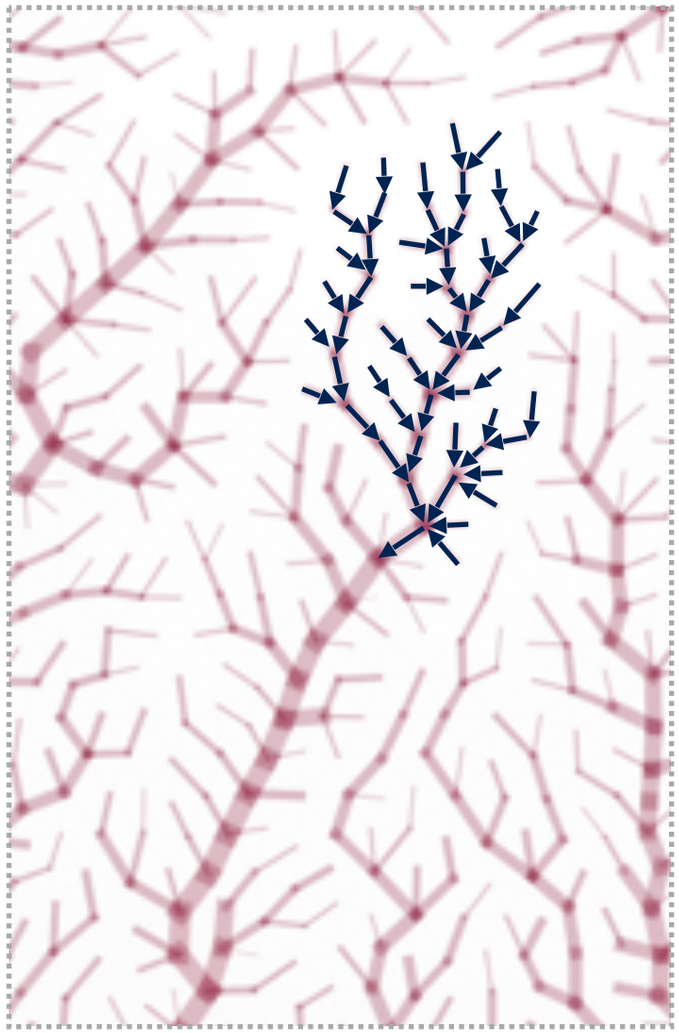
Models which focus on very large tectonic length and timescales and consider the generic, coupled surface / tectonics problem.

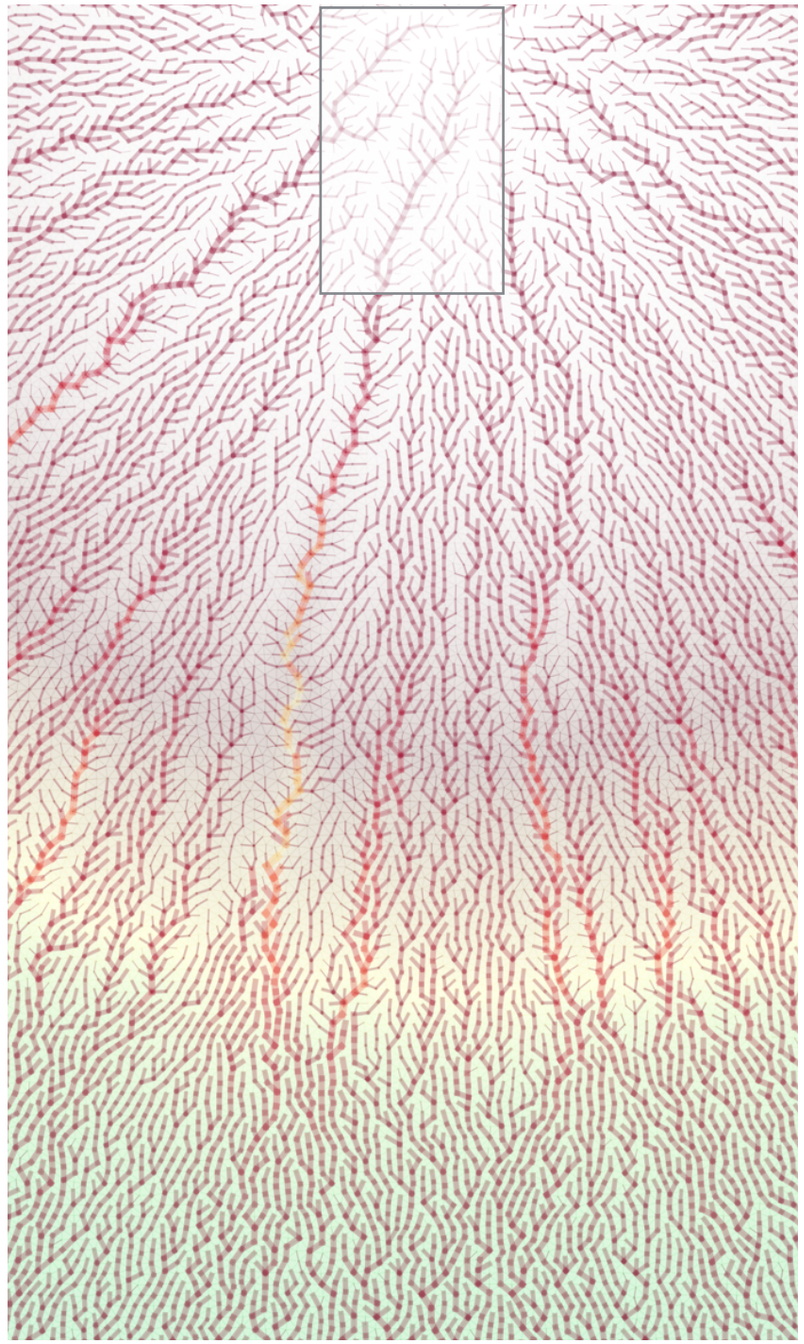
This problem poses interesting challenges in algorithms for parallel computation in models where there is likely to be significant lateral deformation and catchment changes.

We have an algorithm that copes with this problem and will release the `Quagmire` code which implements this



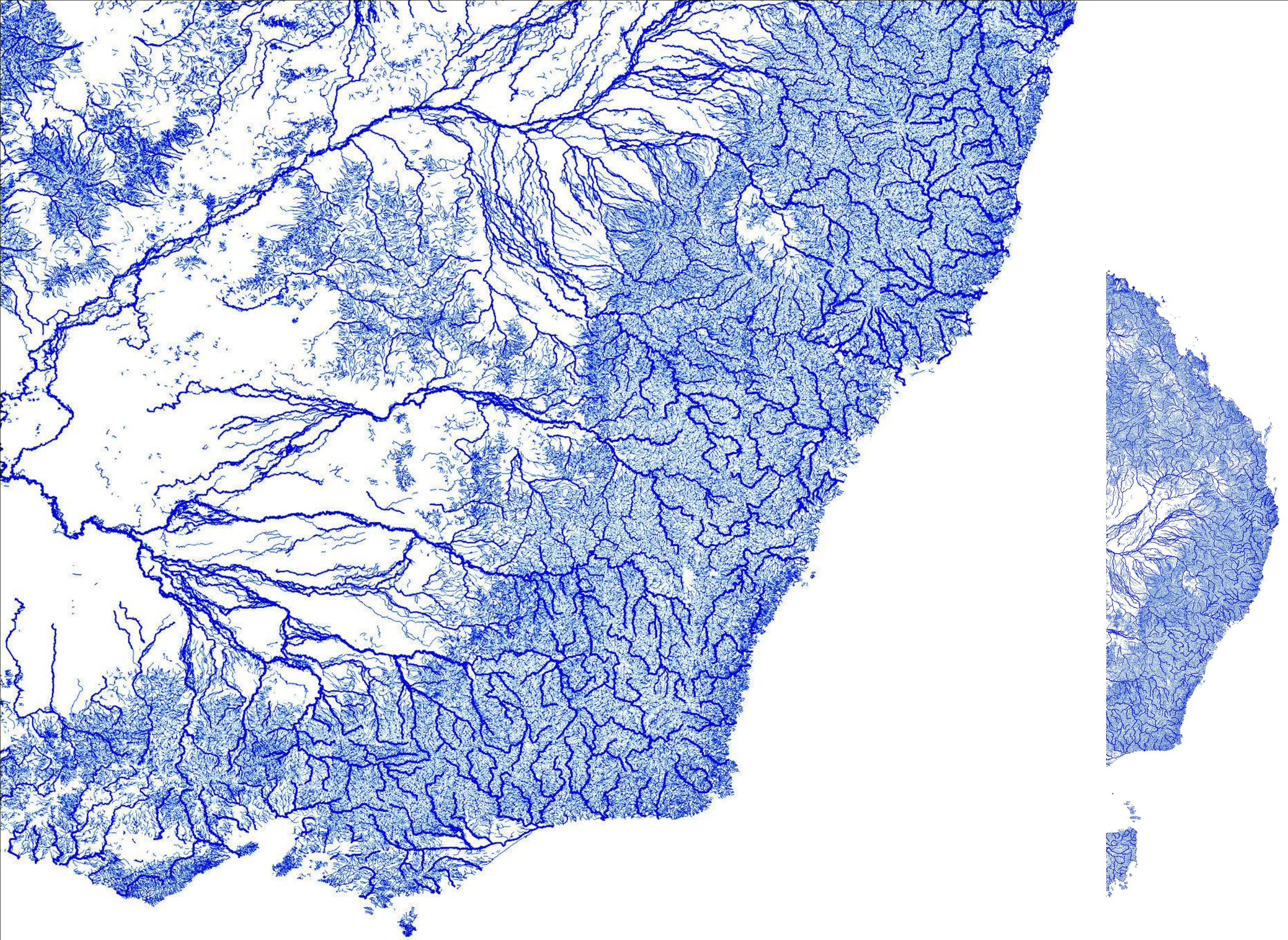
Downstream flow calculations 1.x dimensions





Downstream aggregation terms (integrals over everything upstream of a given point can be regarded as a tree structure with directions pointing downhill.
Very well behaved (non cyclic etc) if only steepest descent.

Or they can be viewed as independent paths which converge as they track through the landscape.



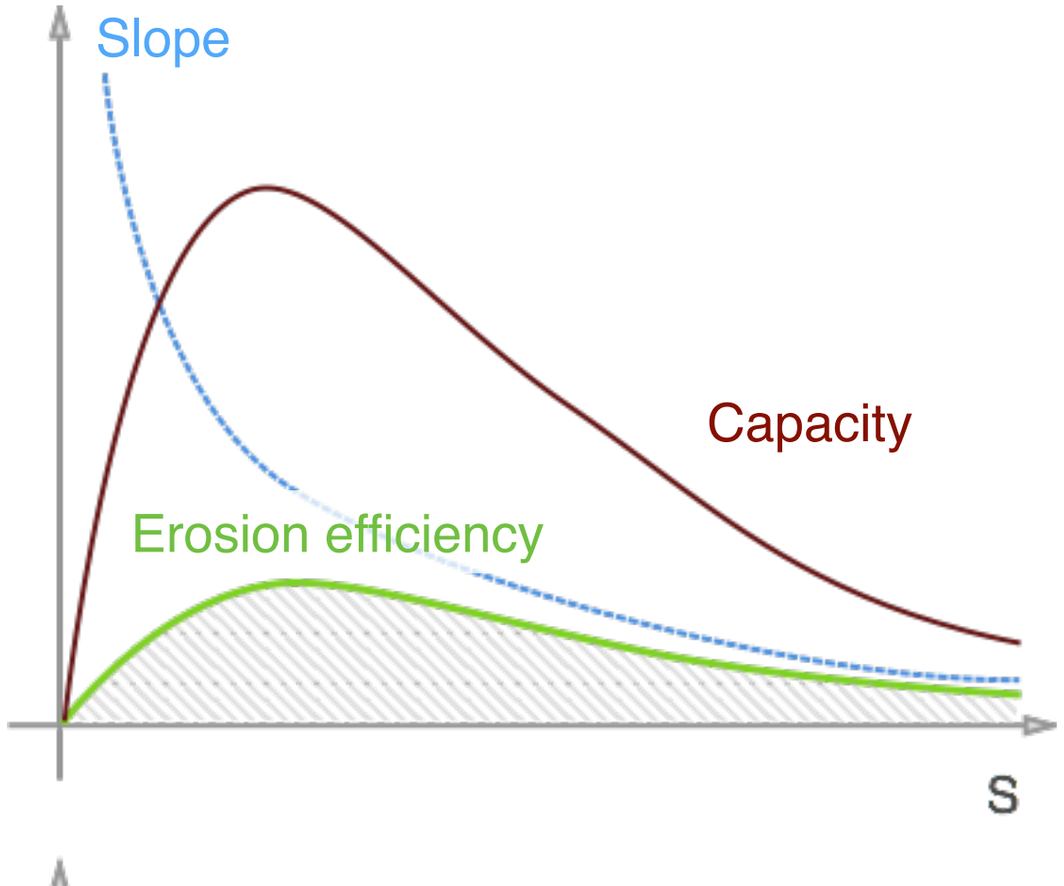
Characteristics of the surface evolution problem

Myriad contributions to the rate of change of surface height at many different scales.

Some can be characterised as local such as diffusion-like terms to describe slope evolution and these tend to be “easy” numerically.

Other terms include “non-local” transport dependencies … in cartoon form:

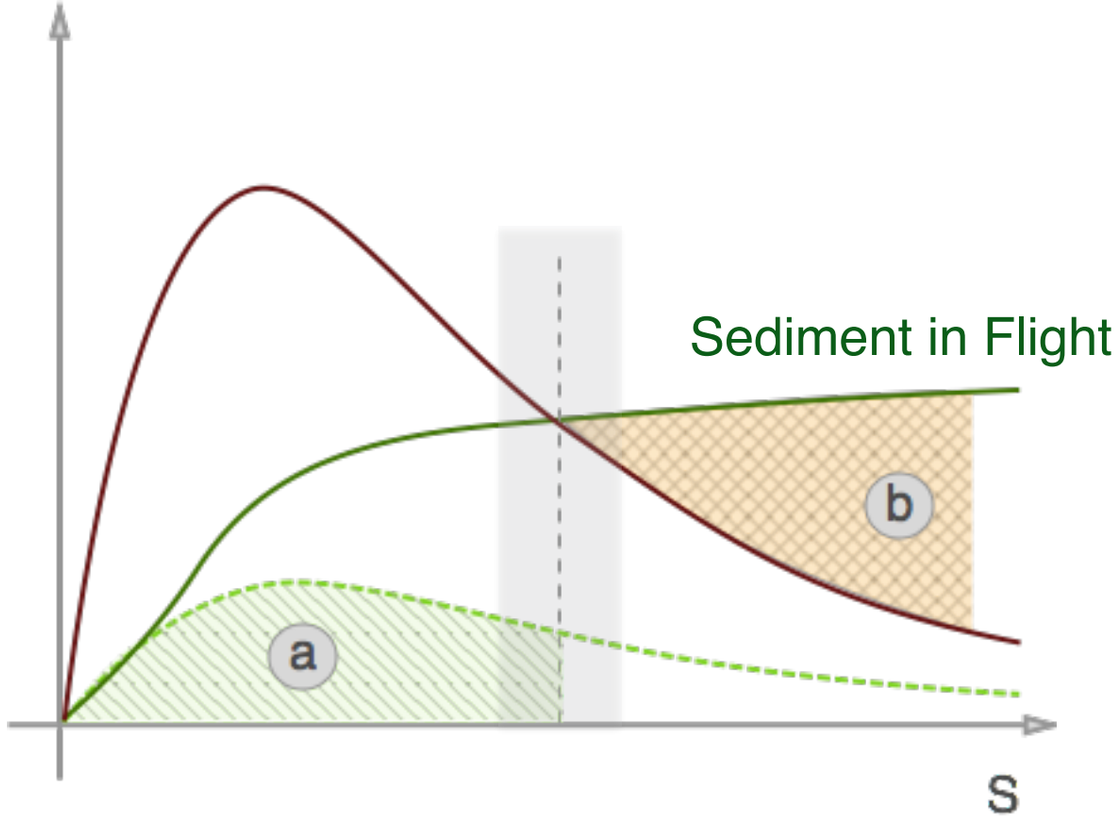
$$\frac{Dh}{Dt} = \nabla\left(\kappa(h)\nabla h\right) - KA^m\left|\nabla z\right|^n + \dot{h}_{\text{tectonic}}$$



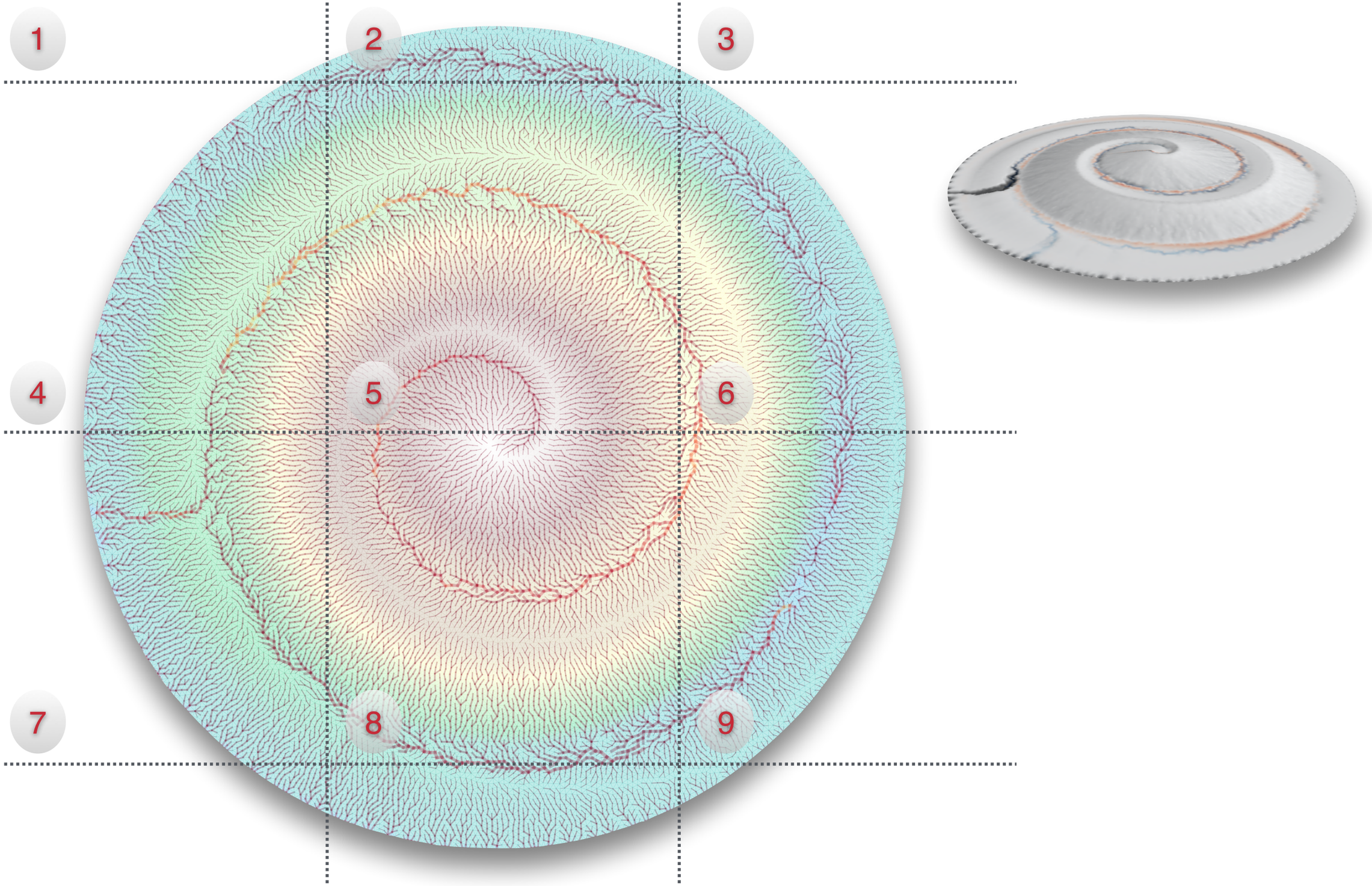
The non-local terms account for the fact that information ‘only’ propagates downstream.

- Total river runoff
- Sediment carried past any point (accumulated load)

Erosion terms are strongly localising with all obvious numerical caveats. Deposition terms have the opposite property.



Decomposition and Data Structures for Surface (forward) Modelling



Flow of information **8** ➛ 5 ➛ **8** ➛ 6 ➛ 5 ➛ 4 ➛ 7 ➛ **8** ➛ 9 ➛ 6 ➛ 5 ➛ 2 ➛ 1 ➛ 4 ➛ 7 ➛ **8** …

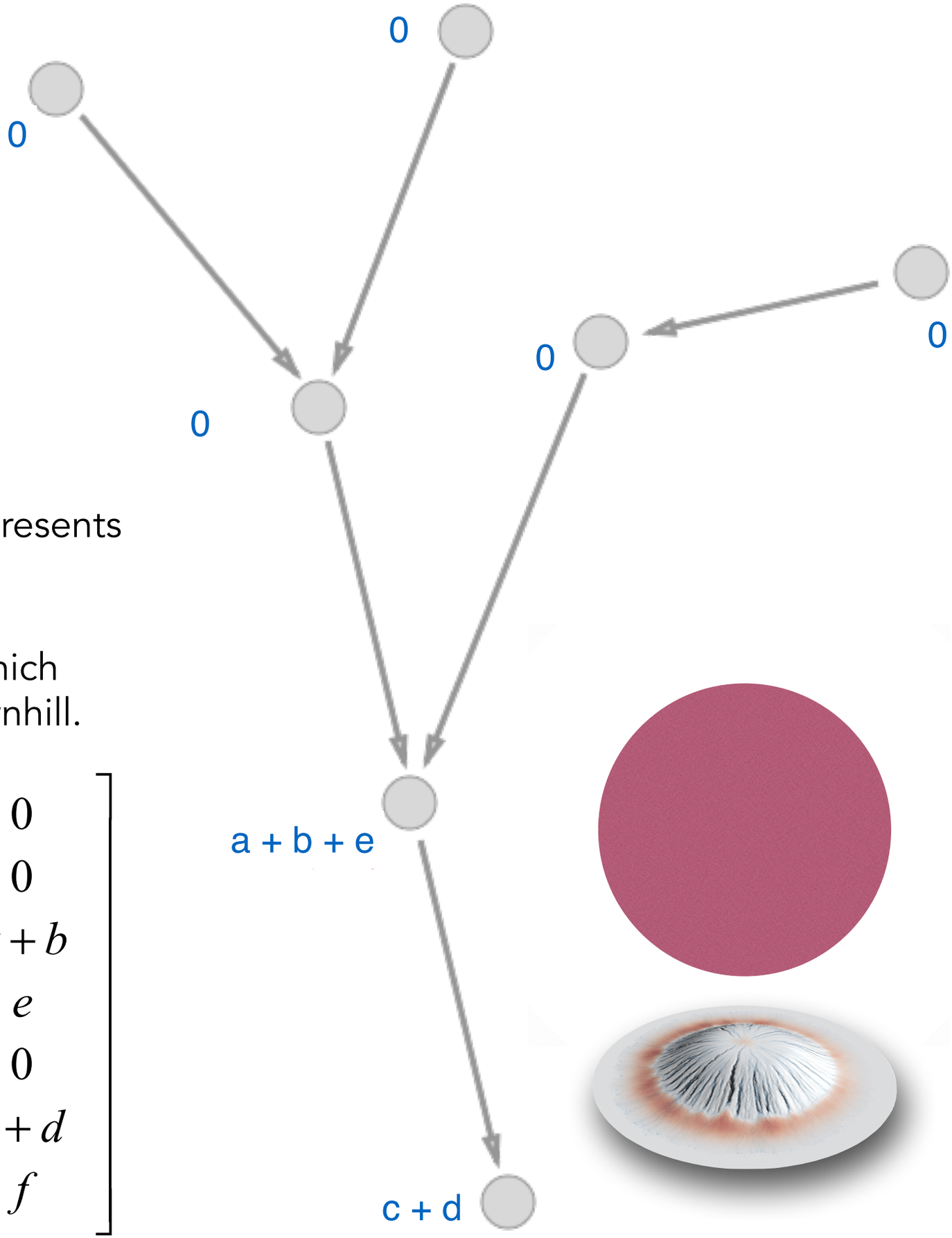
Although this is a contrived example, cyclic dependencies are always possible

Adjacency Matrix Approach

$$A(s) = \int_{\text{upstream}} \alpha(\xi) d\Omega \rightarrow \quad A_g = \sum_{i=0\ldots g} \alpha_i$$

$$A_g = \alpha_g + \alpha_f + \sum_{i=a,b,c} \alpha_i + \sum_{i=d,e} \alpha_i$$

These terms are simple to calculate if we can explicitly construct the graph (tree) which represents the connectivity.



We can do the same thing with a matrix **D** which operates to move information one node downhill.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ a+b \\ e \\ 0 \\ c+d \\ f \end{bmatrix}$$

VERY SPARSE MATRIX

Adjacency Matrix Approach

The matrix **D** which operates to move information one node downhill. $\mathbf{D}^2$ moves information downhill by 2 increments and so on …

VERY SPARSE MATRIX

$$A = (I + D + D^2 + D^3 + \ldots + \mathrm{D}^N)\alpha$$

$$A = D_N \alpha$$

NOT A SPARSE MATRIX

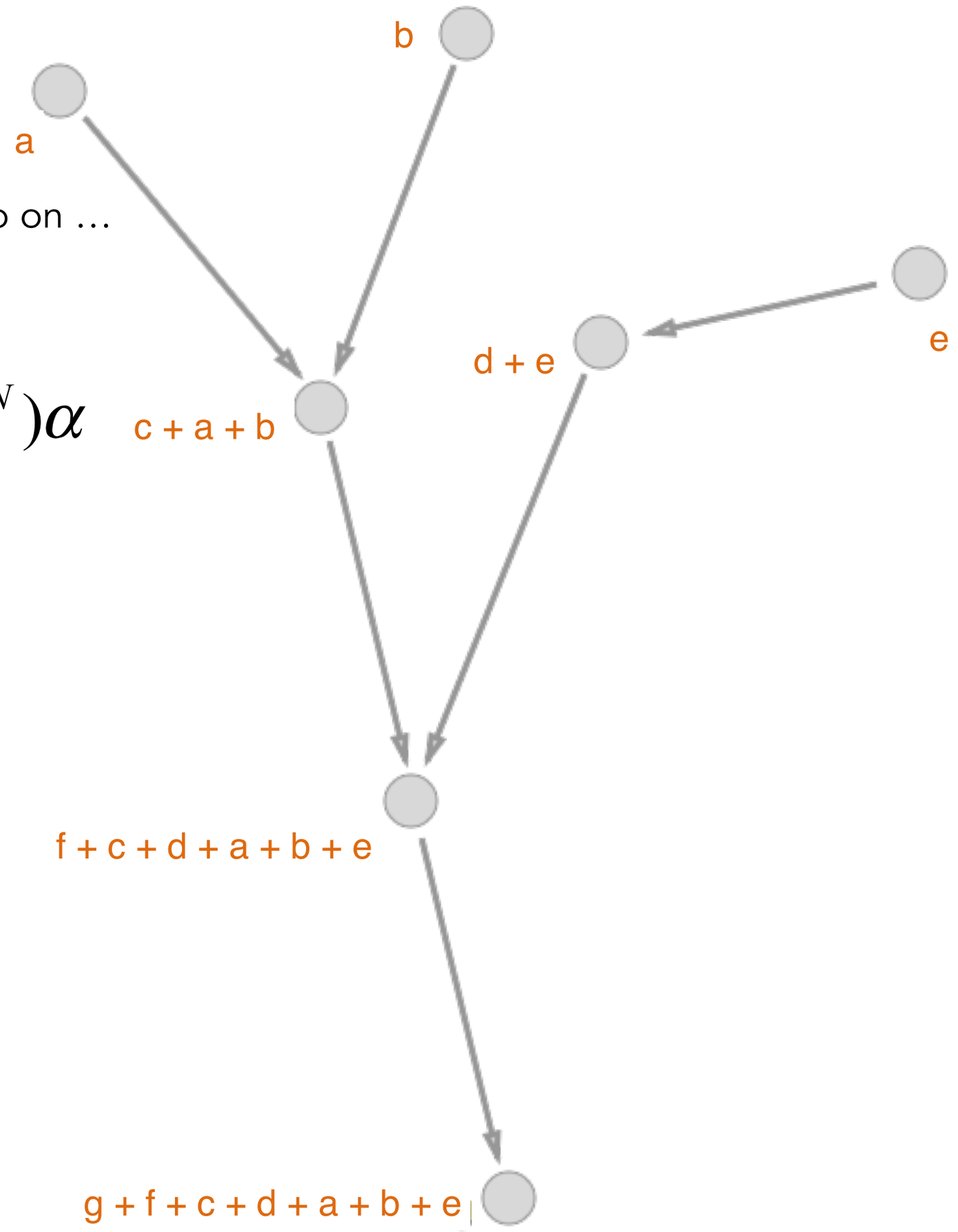
(N is the length of the longest chain)

Parallelism is straightforward … implementation via recursion is also straightforward …

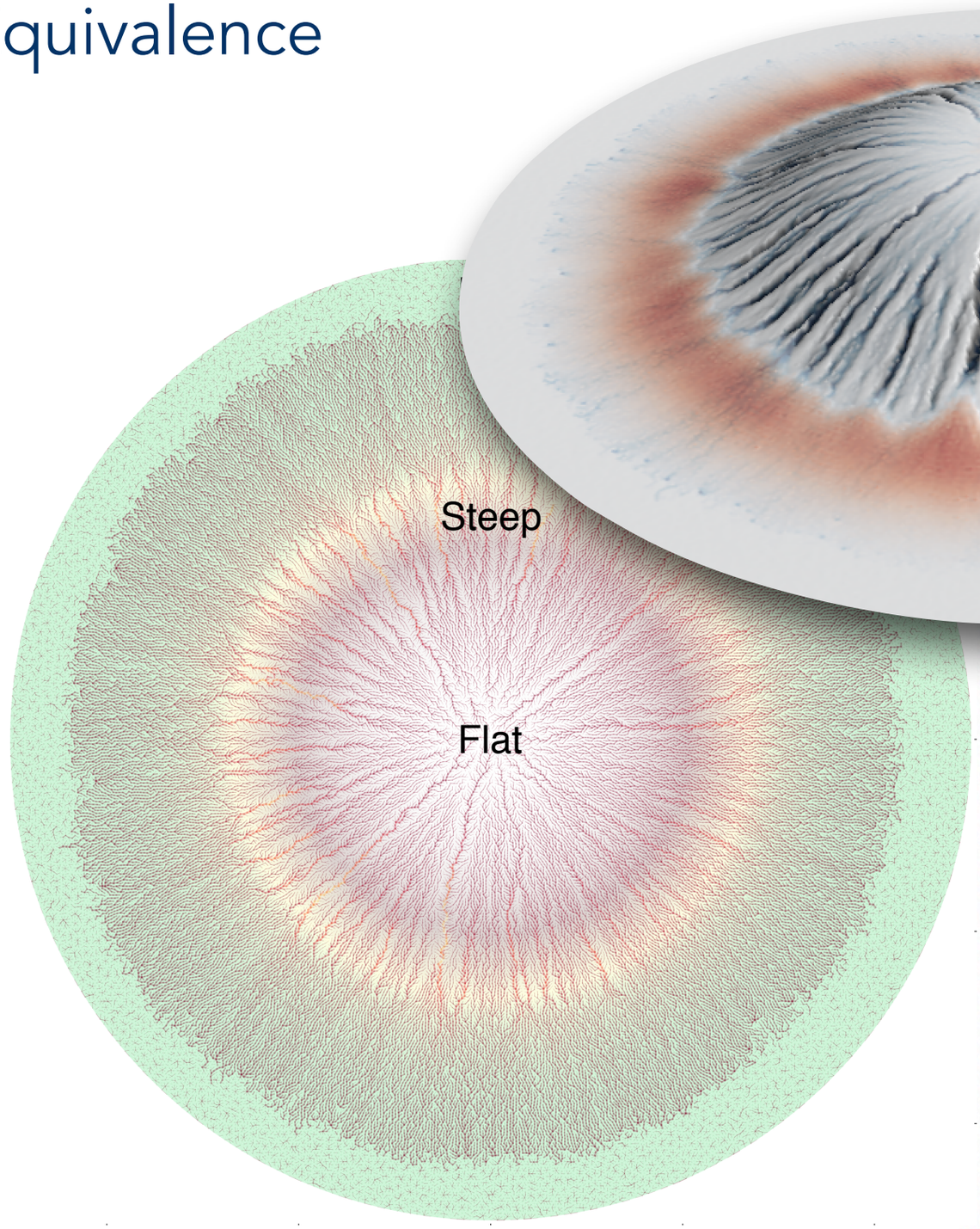
$$A_N^* = D^N \alpha = D A_{N-1}^*$$

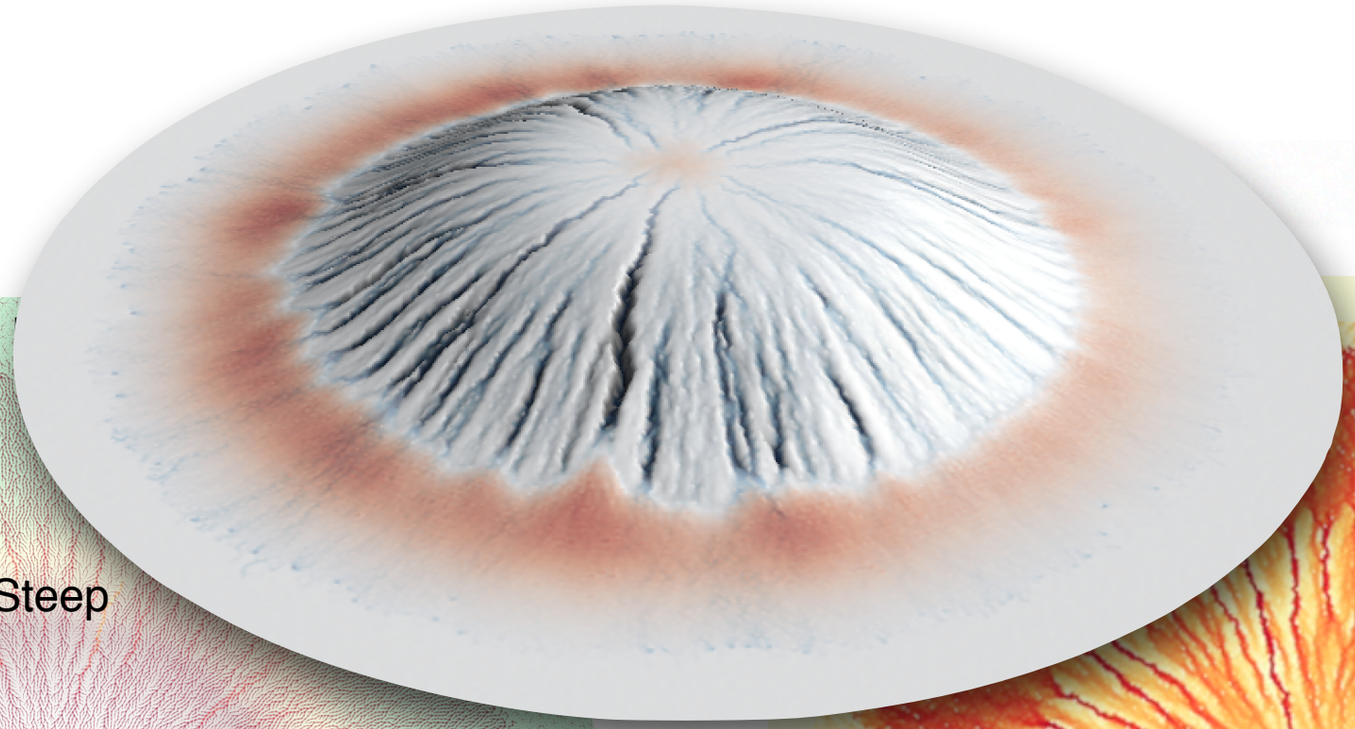
$$A_N = A_{N-1} + A_N^*$$

$$A_N = A_{N-1} + D A_{N-1}^*$$



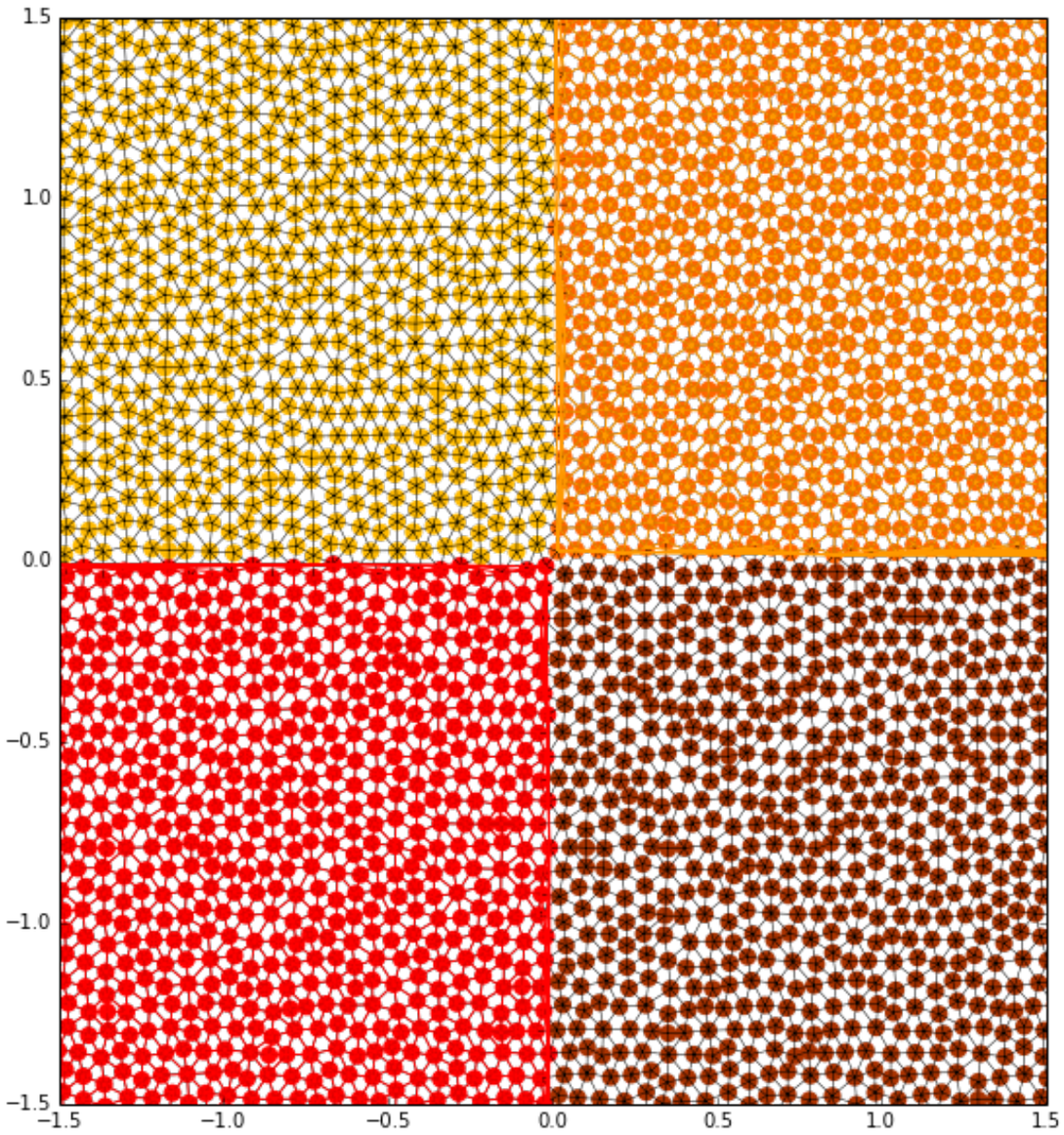
Equivalence

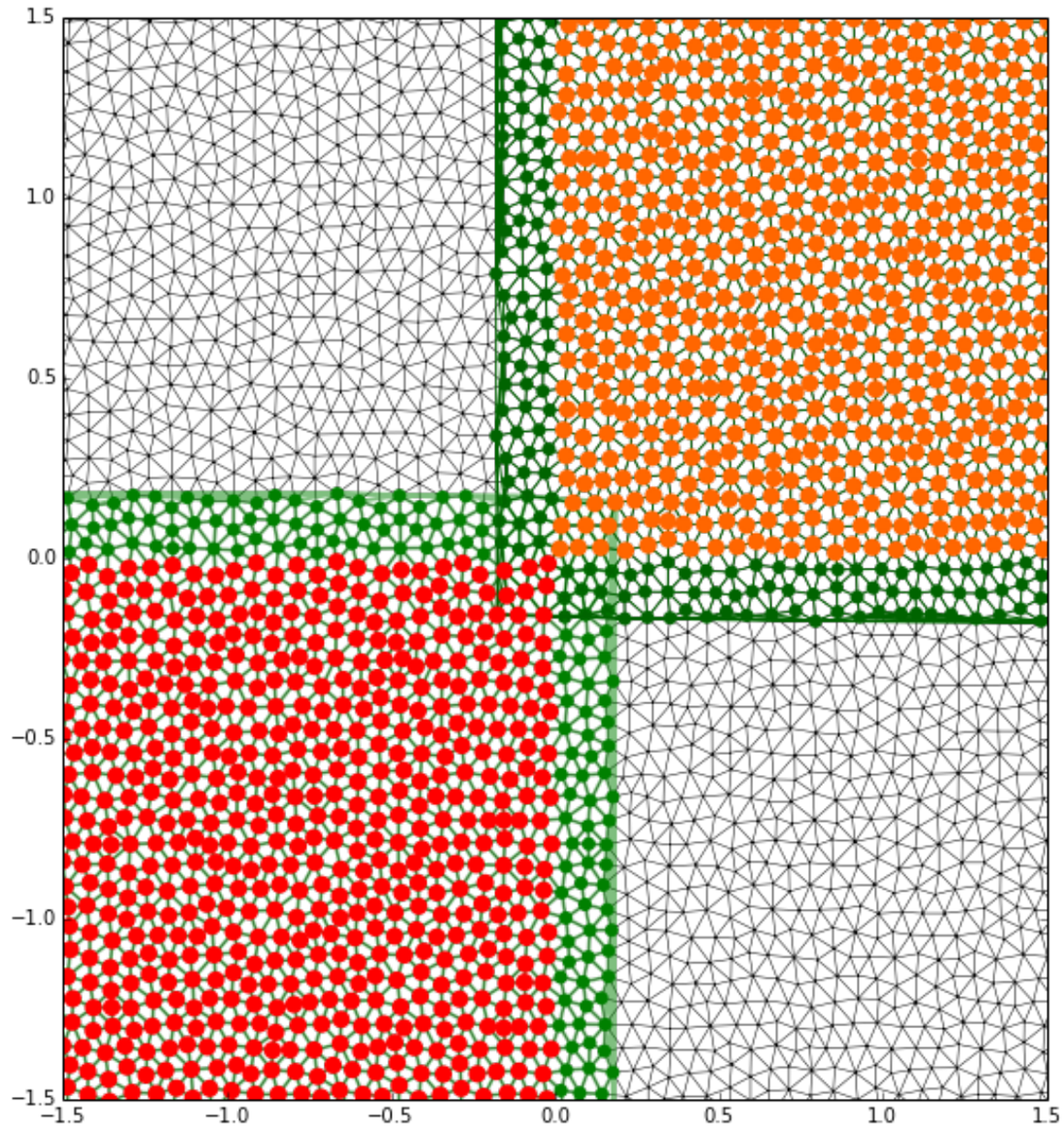




The two representations are entirely equivalent and produce identical results node by node for a given discretisation. Use whatever works !

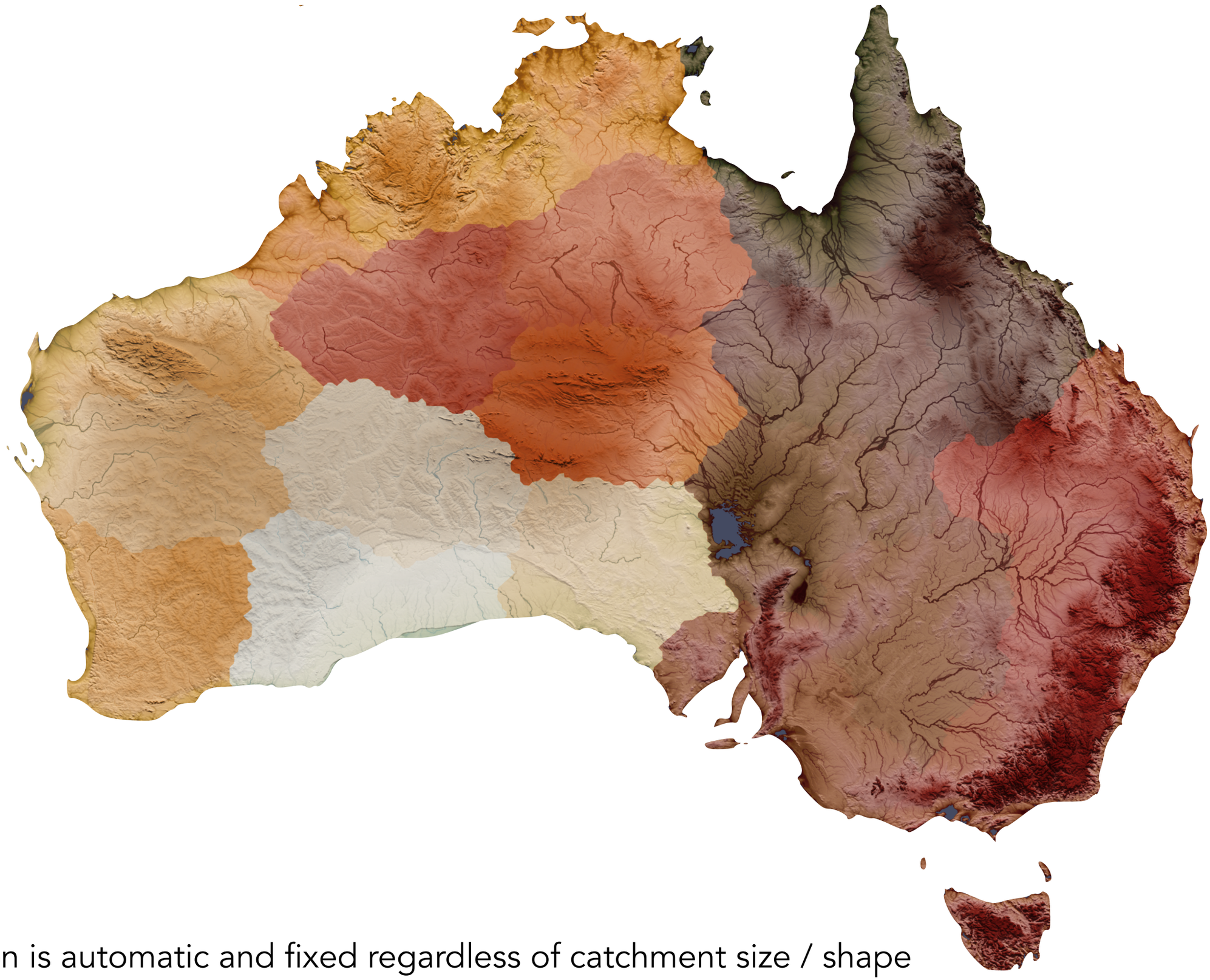
Domain decomposition





If we include shadow regions, then the local triangulation is identical to a subset of the global triangle set and the matrix $\mathbf{D}_d$ is identical to a sub-block of the global **D**. From this point on, the distributed matrix problem is well understood (i.e. **there are libraries such as PETSc to do everything "automatically" !**)

Example



Decomposition is automatic and fixed regardless of catchment size / shape

Matrix approach … other tools we can use

Upstream propagation — we can compute information in the upstream direction using

$$U = D^T$$

though, of course, this is not a one to one mapping any more. We can use this to find catchments for a given outflow of the domain.

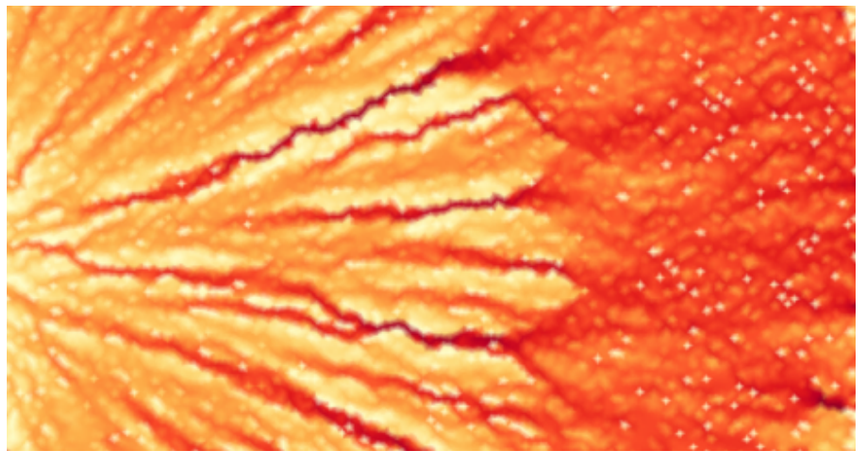
Multiple flow pathways can also be accommodated

$$D = w_1 D_1 + w_2 D_2 + \ldots$$

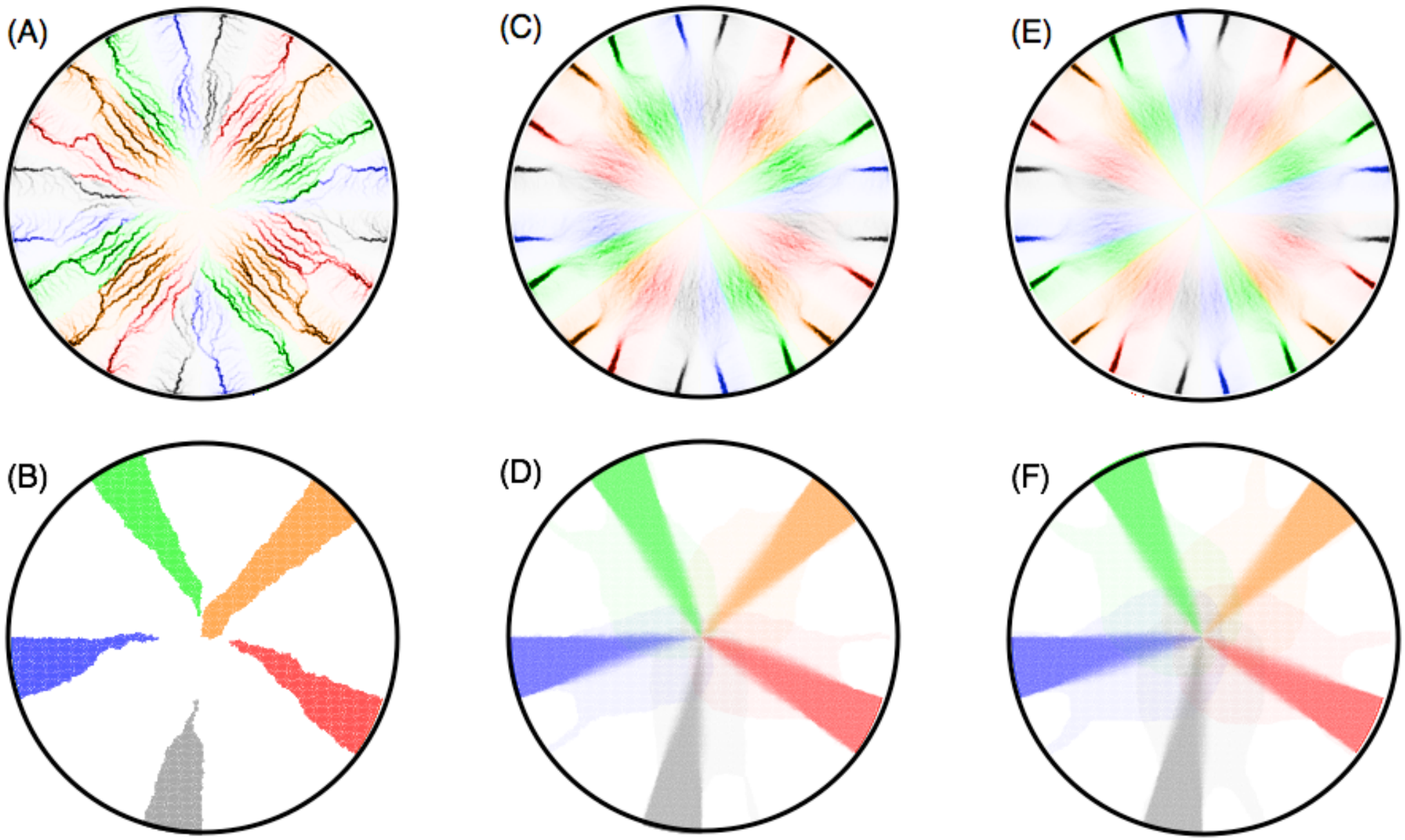
$\mathbf{D}_1$ is sparse, $\mathbf{D}_2$ is just as sparse (more sparse in a channel), $\mathbf{w}$ is a diagonal weight matrix either to weight the flow by gradient or to switch $\mathbf{D}_2$ on / off e.g. for flat areas.

$\mathbf{D}^N$ is pretty much a dense matrix.

$\mathbf{D}^T$ still runs information upstream but does not uniquely define a catchment any more.

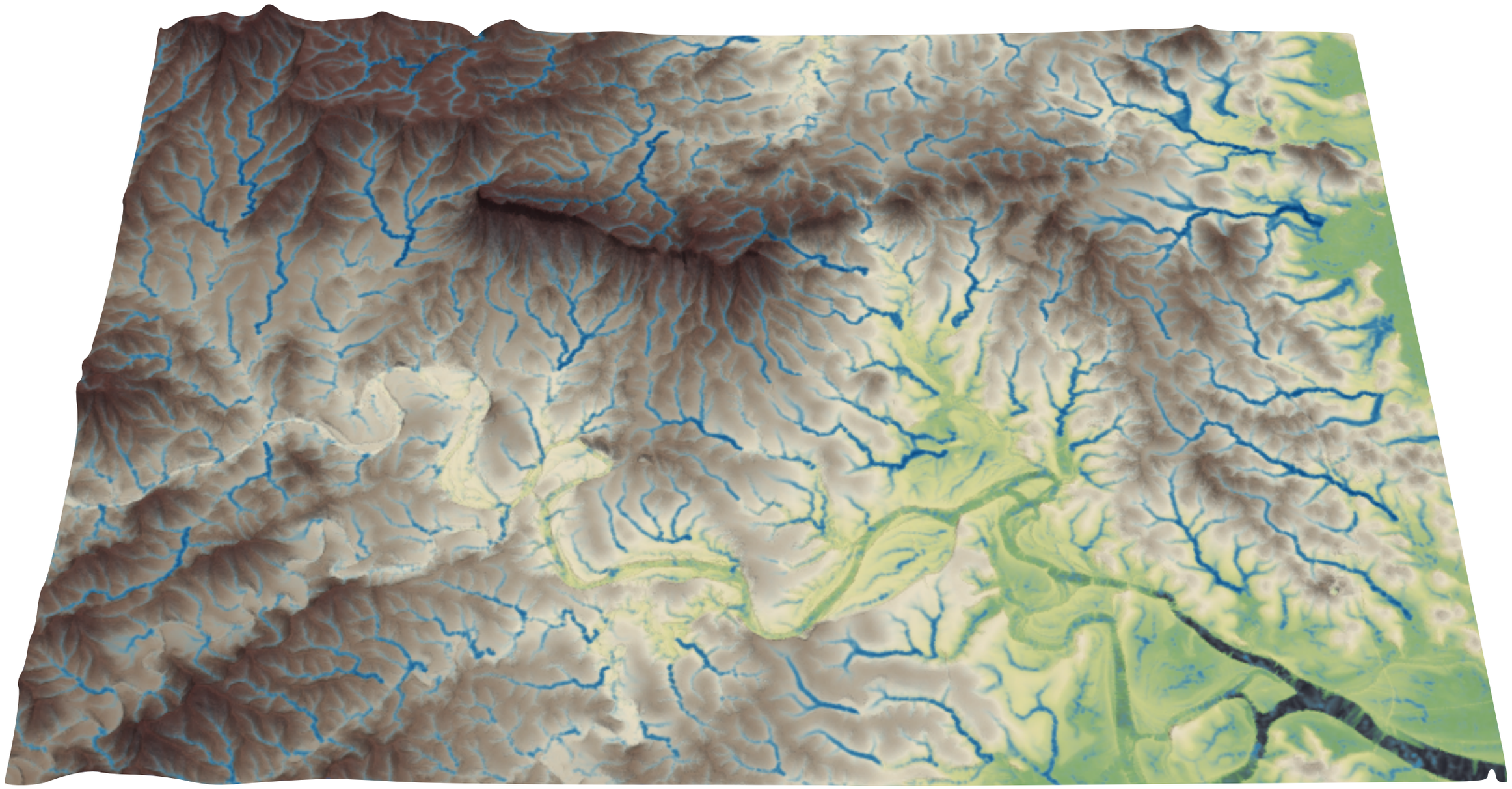


Mesh sensitivity



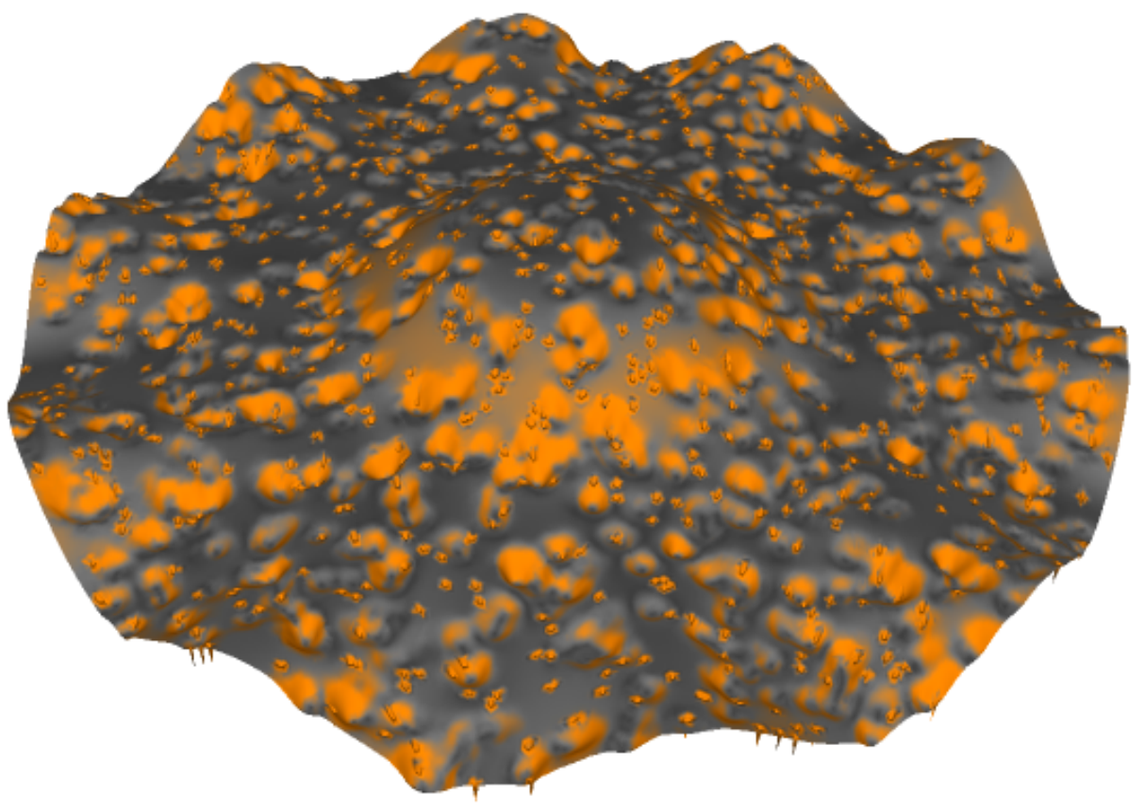
The multiple-pathway approach helps in situations where triangulation influences structure - for example when a flat or smooth area is modelled.

Real landscapes

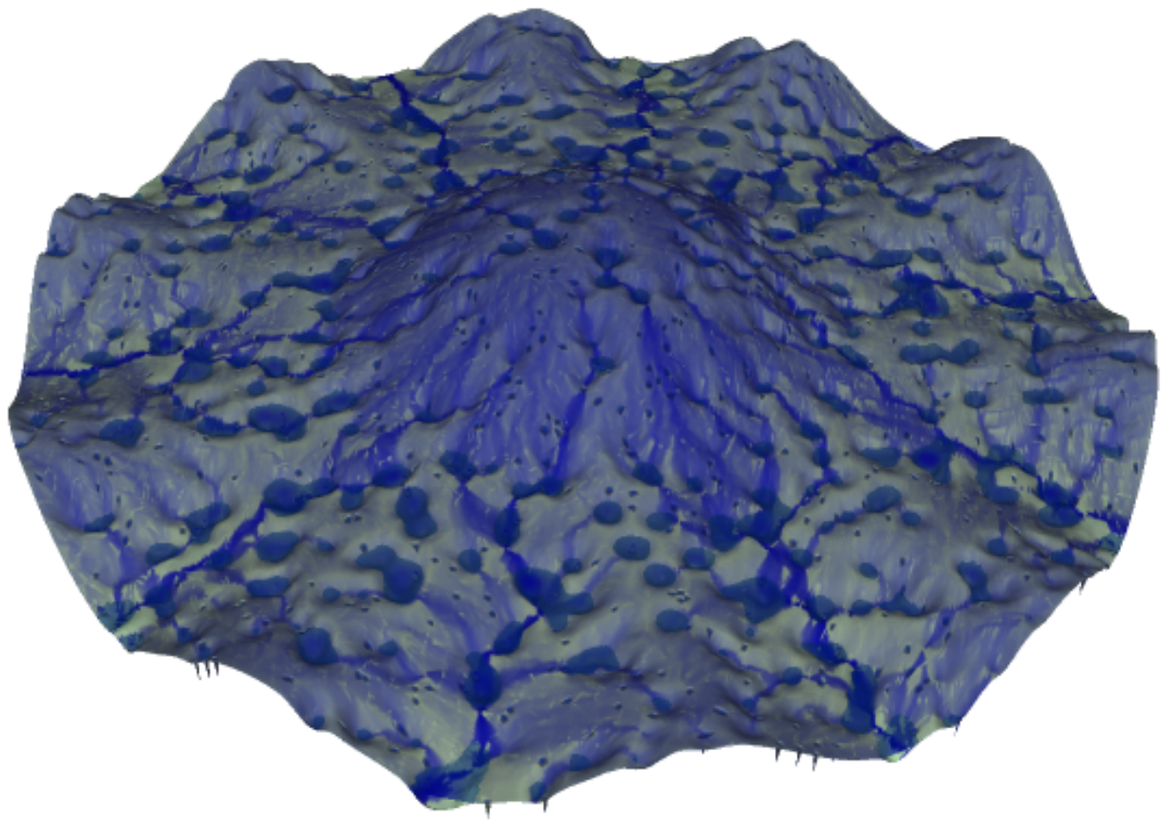


Another challenge (similar to mesh sensitivity) is the discretisation error and measurement error. If a system is sensitive to a few metres change, then the landscape models needs to account for or ignore vegetation and buildings, can be thrown by the fact that river channels are full of water, and needs to consider what to do with a lake / swamp / dam or other build up of water in the flow pathway.

Fill-the-swamp algorithm



Horribly cratered landscape



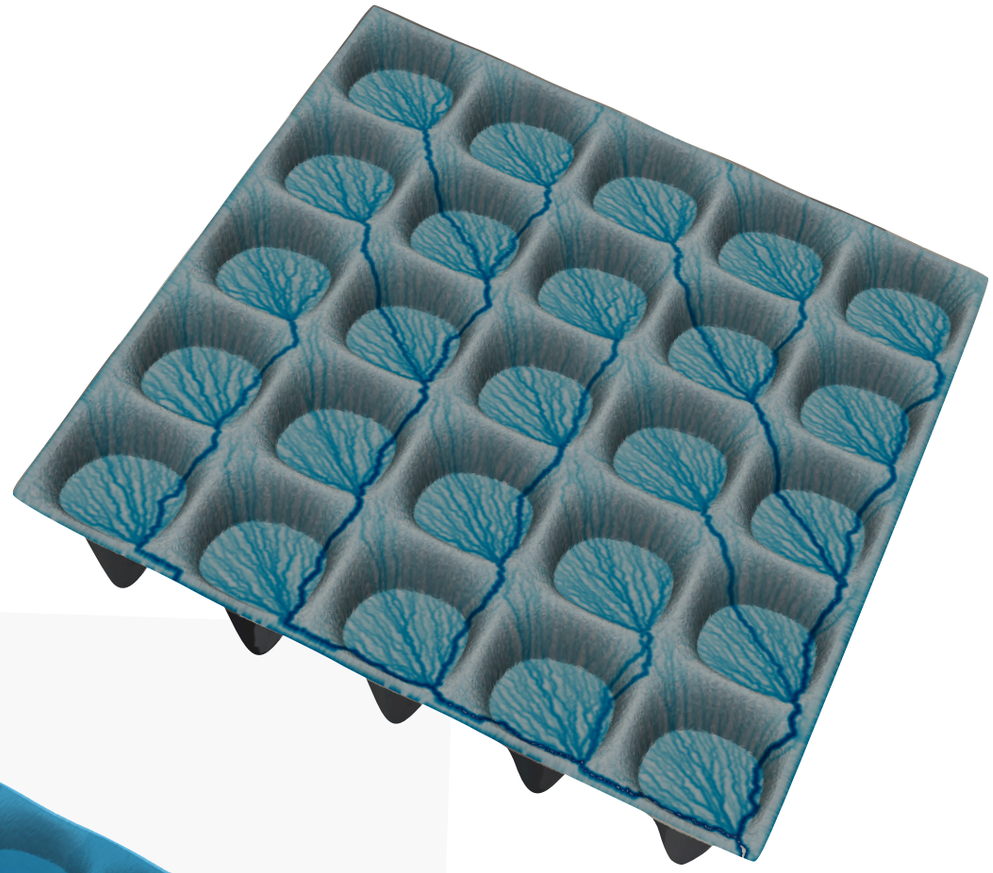
Horribly cratered landscape + lakes

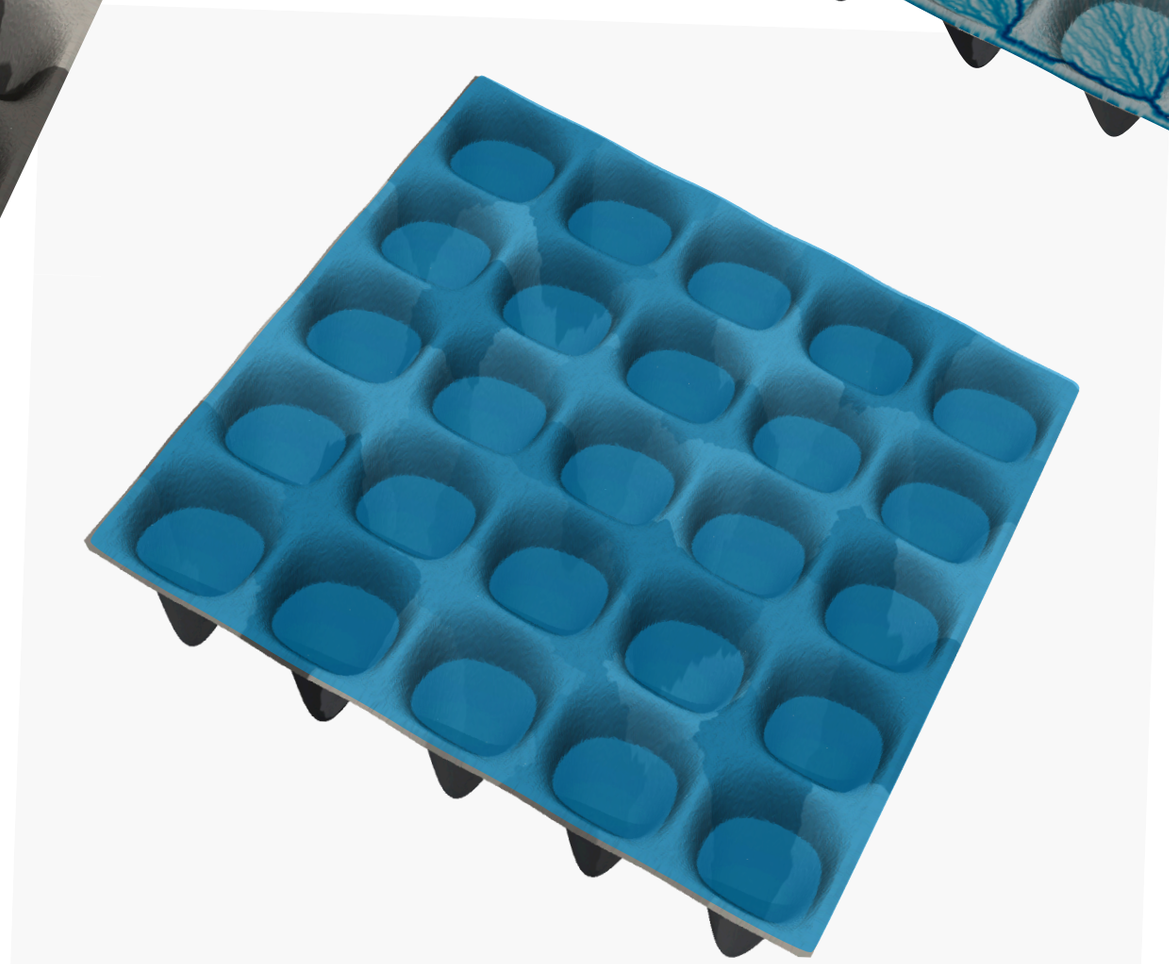
Identify catchments for any internal drainages, and then find the boundary points. The lowest boundary saddle point should be the spill for this catchment. This requires some parallel coordination as different processors identify different spills.

This can be a light-touch algorithm and may get stuck before finding a boundary outflow - fair enough as multiple craters can end up under one large swamp.

It helps to have a local patch-fill preprocessing step and repeat while searching for saddle points.

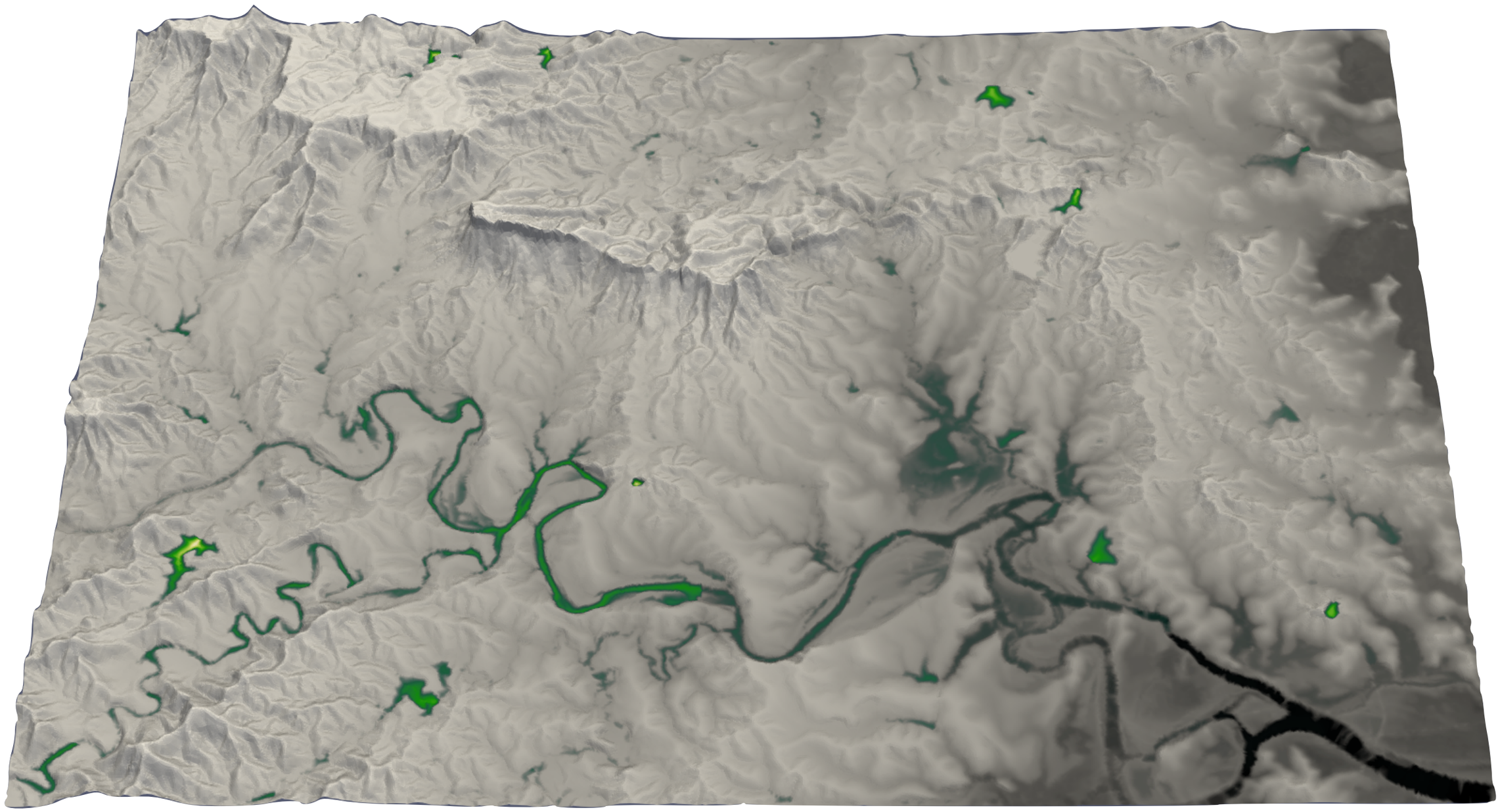
Tilted eggbox





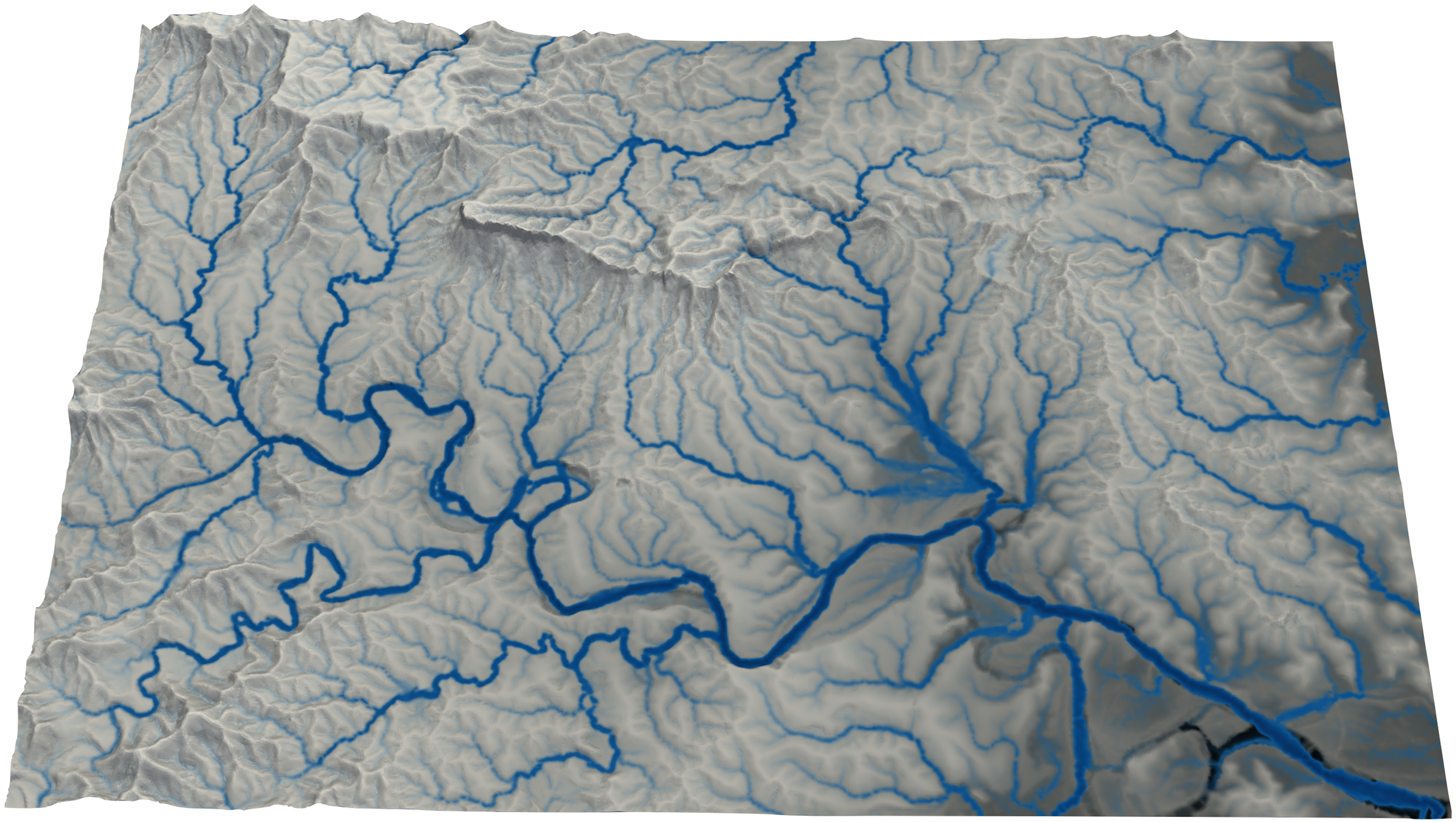
Or adding syrup to a waffle …
Or making ice in ice-cube trays …

Processed Landscape



Swamp filling algorithm finds: 1) a large number of point fills that are meshing-related, and 2) a fill of the river channel and flood plain that directs flow along a connected path.

Processed Landscape



Flow paths found this way are generally not unique - a more useful way to analyse this is to think of the ensemble of paths that can be found with small perturbations to the landscape. This seems to find flood plains, old drainages etc. Also roads, forest clearings, reservoirs etc !!

Raw surface

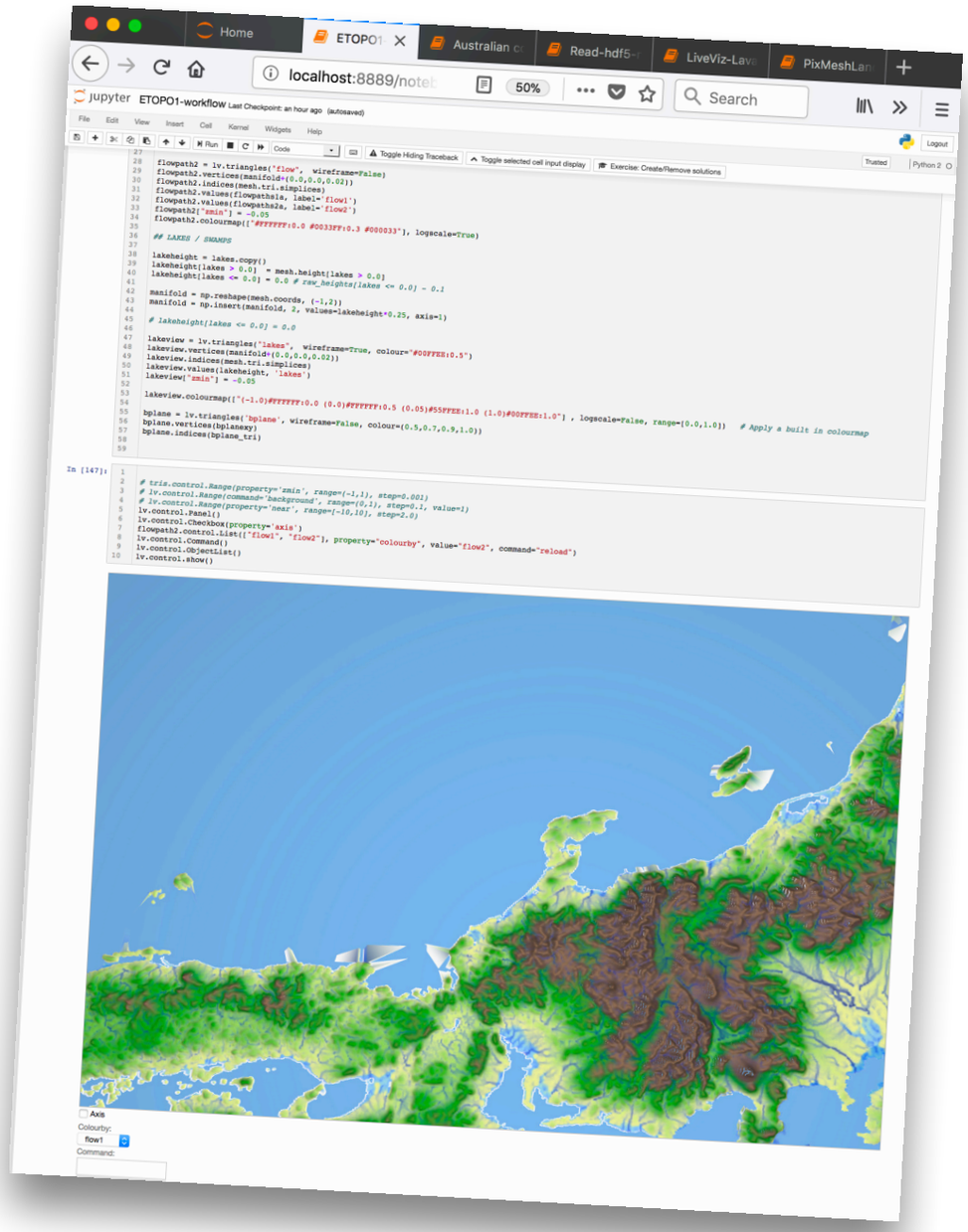
Flood-filled Landscape

Quagmire

Quagmire is a toolkit for exploring the interplay between large-scale dynamic models and surface process modification of the topography.

- python based but using `PETSc / petsc4py` to make it fast and 'trivially' parallel
  - DMPlex and DMDA (for triangulation v. pixels)
- provides classes for meshing based on `stripy`
  (based on `TRIPACK / STRIPACK / SRFPACK / SSRFPACK`)
- object oriented - parallel implementation largely "hidden"
- operators for downstream transport and upstream integration
- operators for flood-filling
  - Matrix form
  - Graph form (not scalable / not parallel)
- Pit filling, swamp flooding, stream power computation supplied

Live documentation in the form of jupyter notebooks



Quagmire — the philosophy

Goal: provide a grab bag of efficiently-implemented, python classes that people can quickly use to implement their favourite LEM (or just use some pieces in a python workflow).

- Let the science questions dictate what the codes do and not the other way around
- Let anyone in the community define their version of the problem and support their choices
- Open source, open community
- Leverage PETSc and other open-source tools
- Easy interoperability with python codes such as Underworld
- Available through pip / spack for easy installation

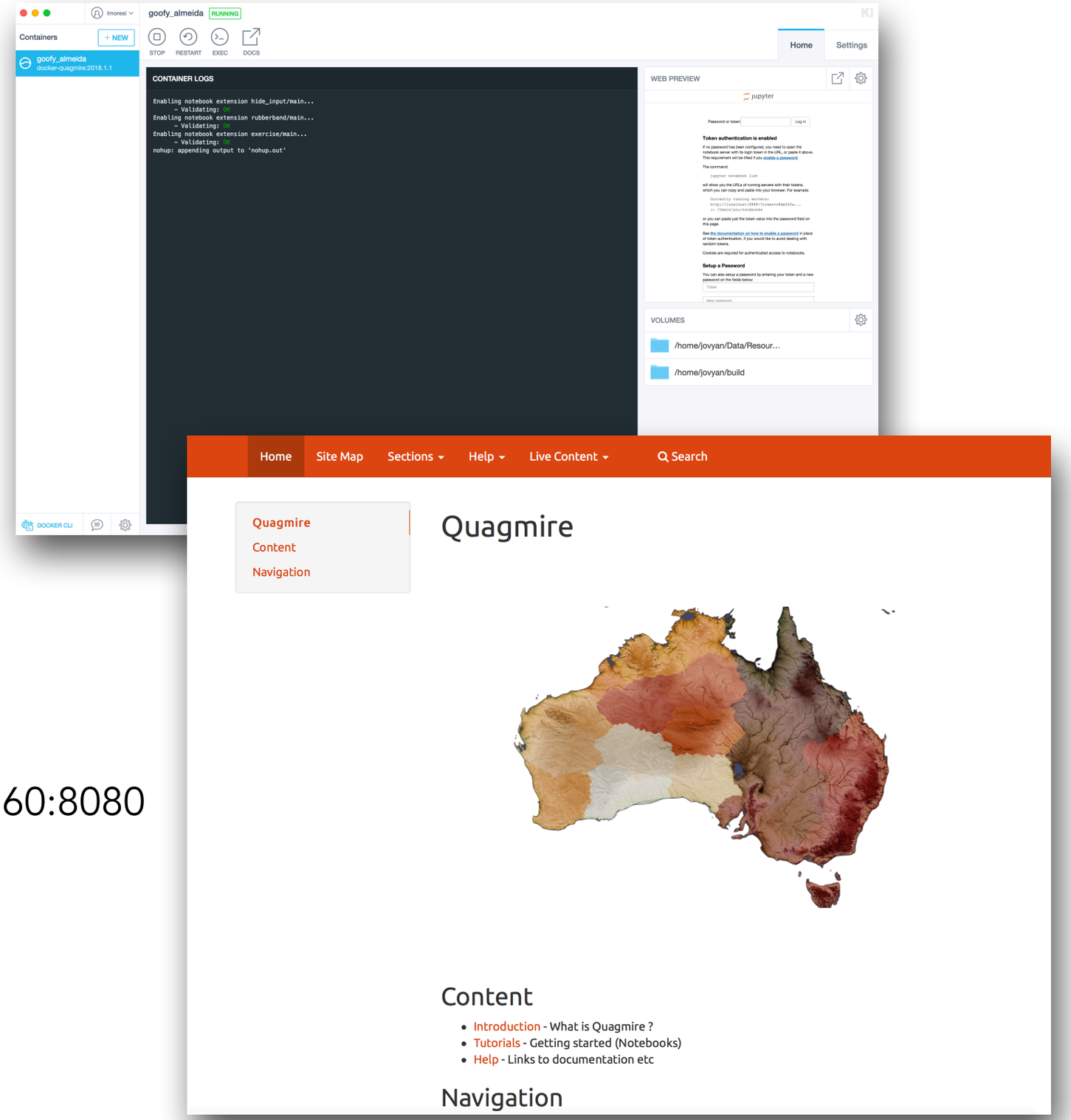
Quagmire is not a shrink wrapped LEM code that does anything very much other than showing a few examples. It's a typical python project that provides a way to do some things reasonably well.

Open source, so it may just end up as a template for other people to take and improve.

Try out Quagmire

docker: lmoresi/docker-quagmire:2018.1.1

for a limited time only … http://43.240.97.160:8080



github: https://github.com/University-of-Melbourne-Geodynamics/quagmire

Underworld Material point method

Fixed mesh with moving “particles”

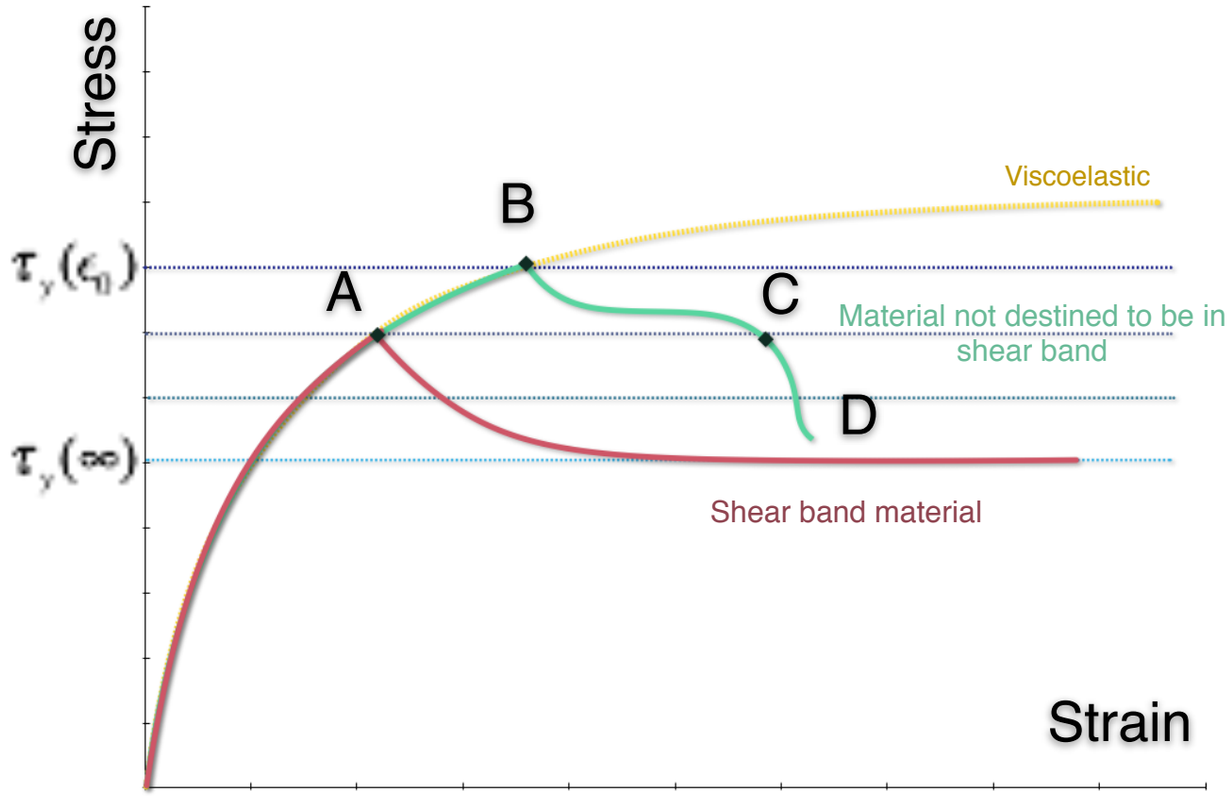
- Regular Eulerian mesh for momentum equation (efficient solvers)
- Lagrangian reference frame for:
  - Compositional tracking
  - Stress-history tensor
  - Plastic strain history (scalar / tensor)

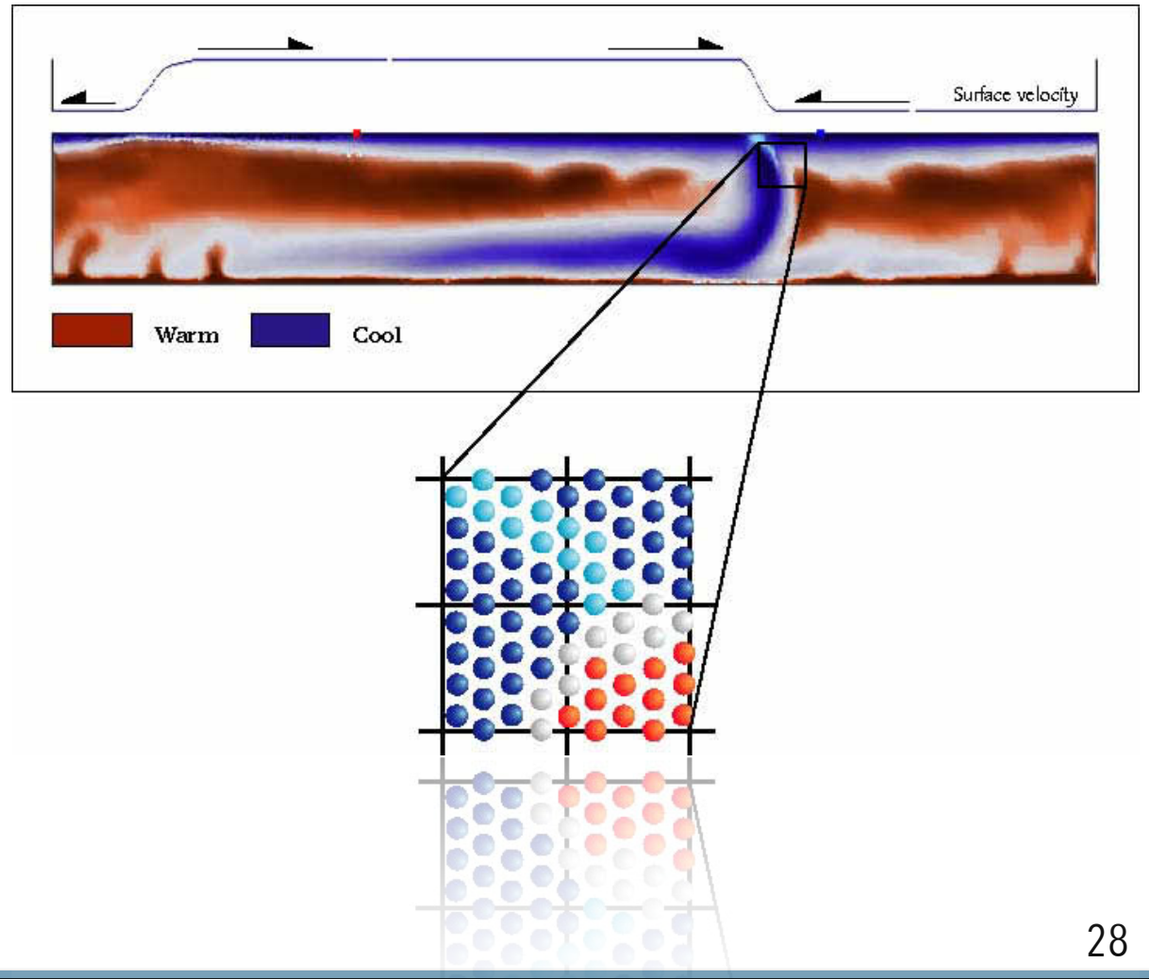
Finite element formulation

- robust, versatile
- very simple to go back and forth between particle and mesh representation

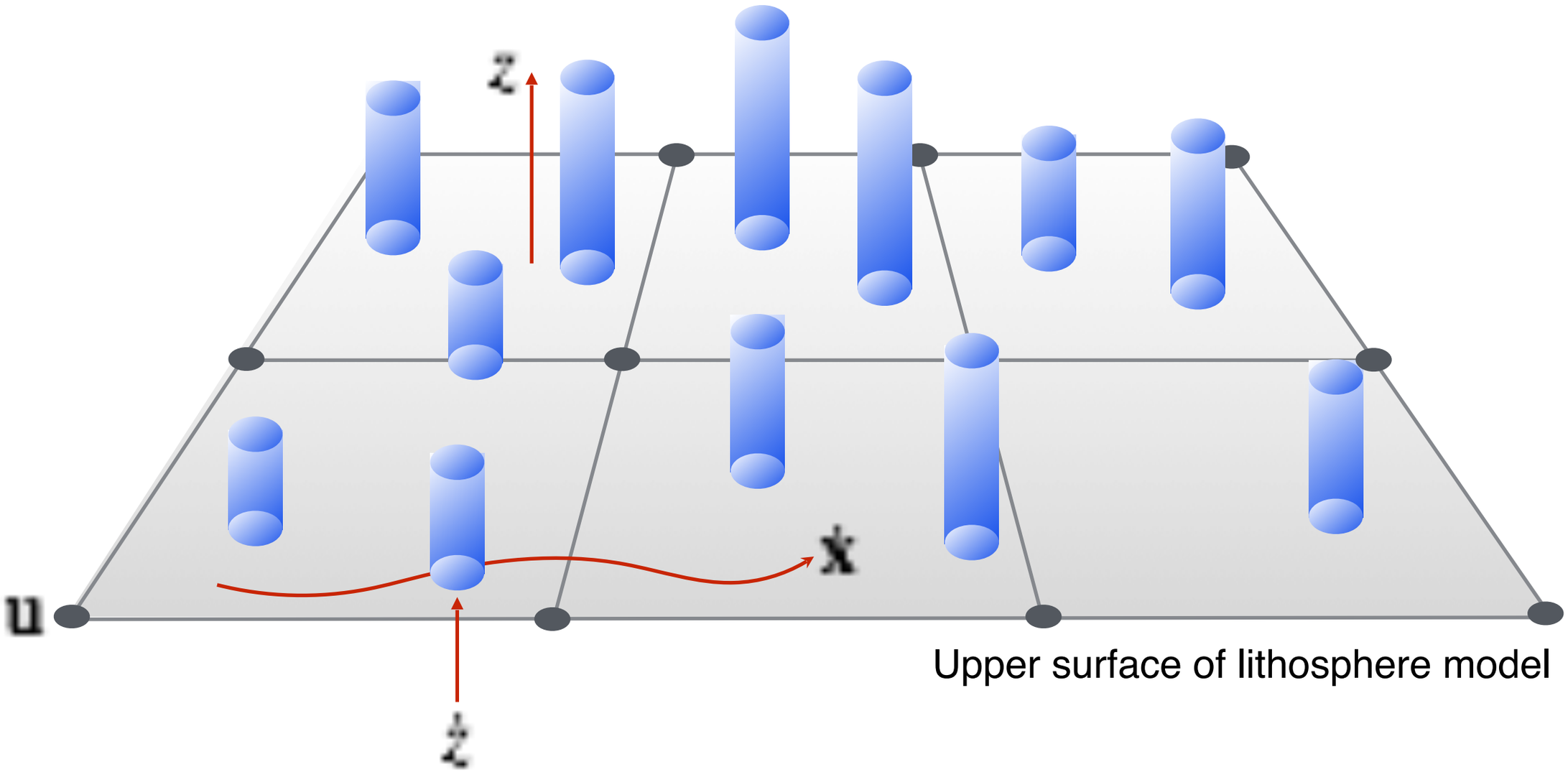
$$\mathbf{K}^E = \int_{\Omega_E} \mathbf{B}^T(\mathbf{x})\mathbf{C}(\mathbf{x})\mathbf{B}(\mathbf{x})d\Omega$$

$$\mathbf{K}^E = \sum_p w_p \mathbf{B}_p^T(\mathbf{x}_p)\mathbf{C}_p(\mathbf{x}_p)\mathbf{B}_p(\mathbf{x}_p)$$





Lagrangian mesh & information transport



Pseudo free-surface approach —

- surface tracking points hover above / below the flat free-slip boundary
- vertical loads applied to account for positive / negative topography (cf. isostasy !)
- lateral deformation from the deformation of basement