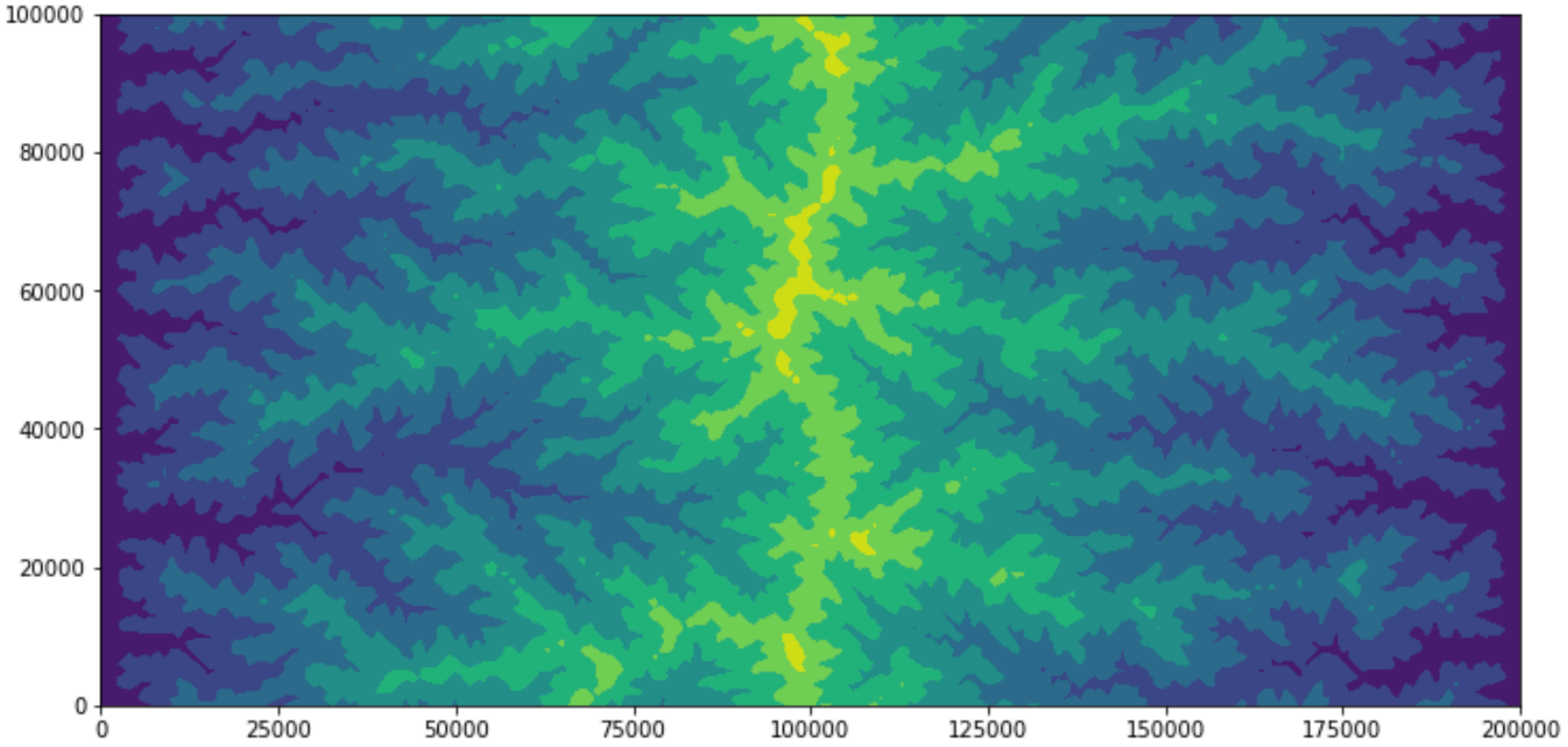
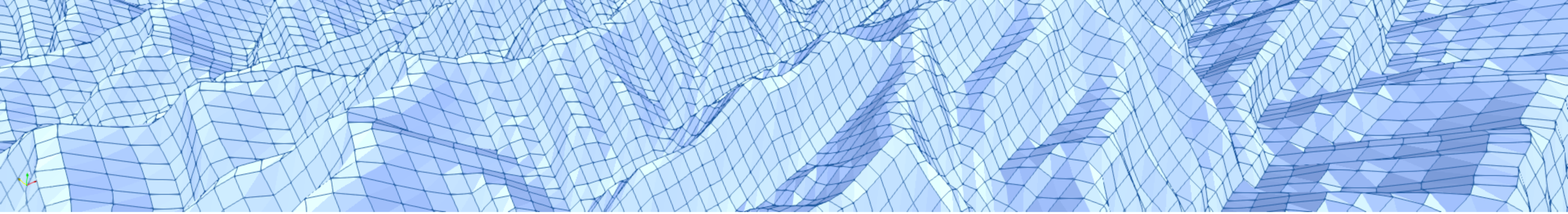
Universität Potsdam

Earth Surface Process Modelling course

FastScape: an O(n) and implicit solver for the SPL equation

Jean Braun 2020


100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
150000
175000
200000
5600
4800
4000
3200
2400
1600
800
0



FastScape

Basic equation

We wish to solve the SPL equation:

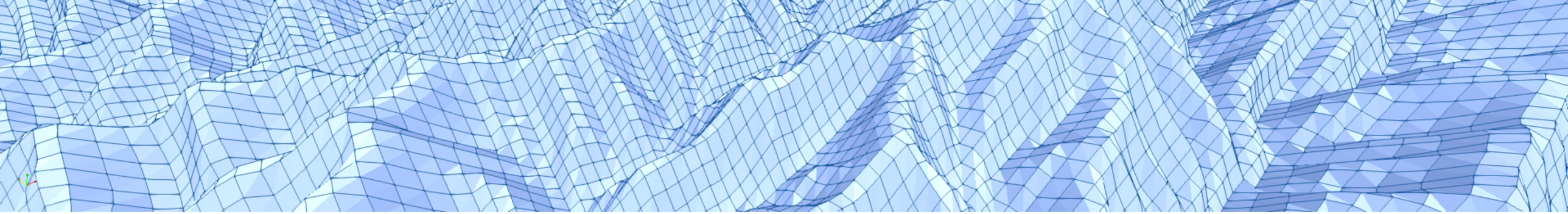
$$\partial_t h = U - KA^m S^n$$

in 2D where h is topographic elevation, U uplift rate, A contributing drainage area, S slope in the direction of water flow and K, m and n are constant. The topography is discretised on a rectangular grid of dimension $n_x \times n_y$ at a spatial resolution of $\Delta x \times \Delta y$ such that $L_x = (n_x - 1)\Delta_x$ and $L_y = (n_y - 1)\Delta y$, where L_x and L_y are the domain dimensions. We will assume that water flow from each node to the steepest of its eight neighbours on the rectangular grid.

The initial topography is assumed to be a small (1 m high) white random noise. All four boundaries are assumed to be at base level, i.e. fixed at $h = 0$.

To store the topography and other variables (such as drainage area, uplift rate, etc.) we will use single indexed arrays, h_i for $i = 1, \cdots, n_x \times n_y$, for example. The reasons for this will become clearer later.

The FastScape algorithm consists in defining an optimum node order or stack to compute drainage area first and then to solve the equation. To compute this order, one first needs to compute the list of receiver nodes r_i (each node's steepest neighbour) and the list of donor nodes $d_{i,k}$ (list of neighbouring nodes that have i as a receiver), noting that each node can only have one receiver but can have more than one donor.

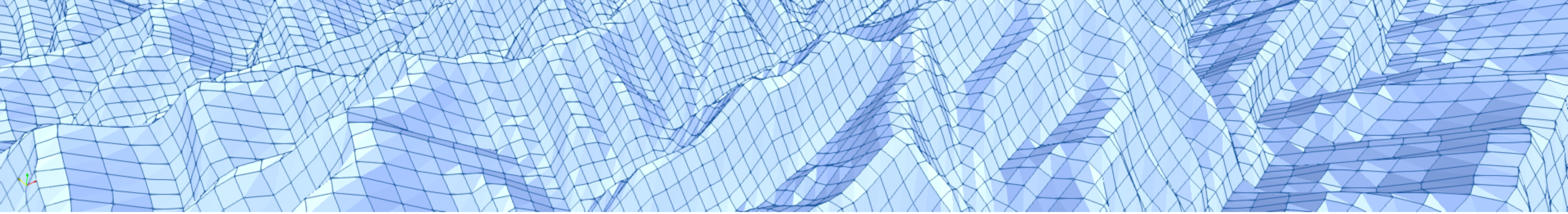


FastScape

Algorithm summary

The algorithm can be split into 5 steps:

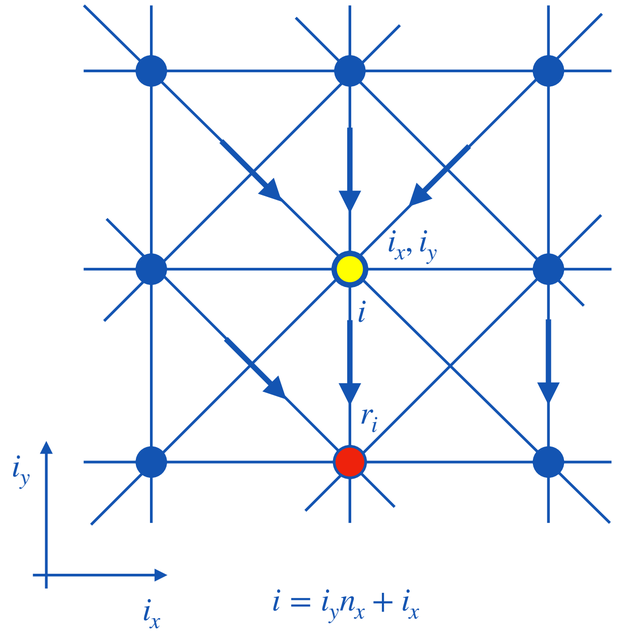
1. Compute receivers
2. Compute donors
3. Compute stack order
4. Compute drainage area
5. Add tectonic uplift
6. Solve equation

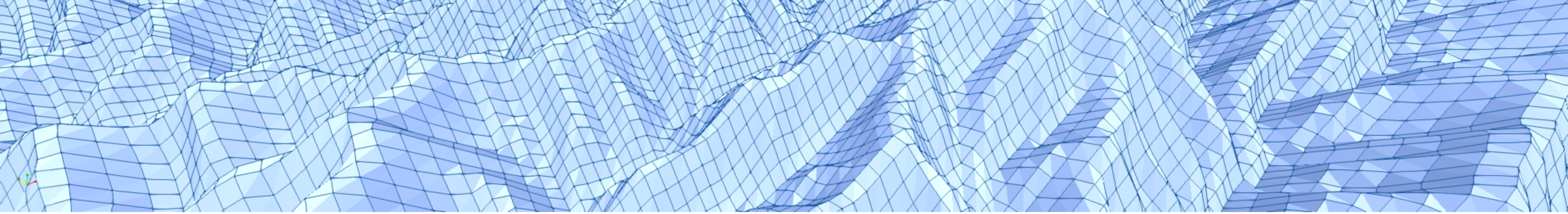


FastScape

1. Computing the receivers

Computing the list of *receivers* consists in finding for each node i the name of the lowest of its eight neighbours, r_i. This is most easily done within a loop but can also be performed using array operations. For ease of comprehension, we will use explicit loops for most of the operations we will perform here. We will improve the efficiency of our python code by using *numba*.

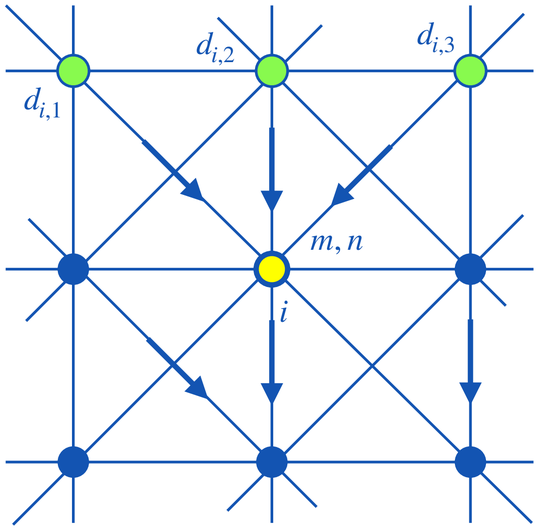


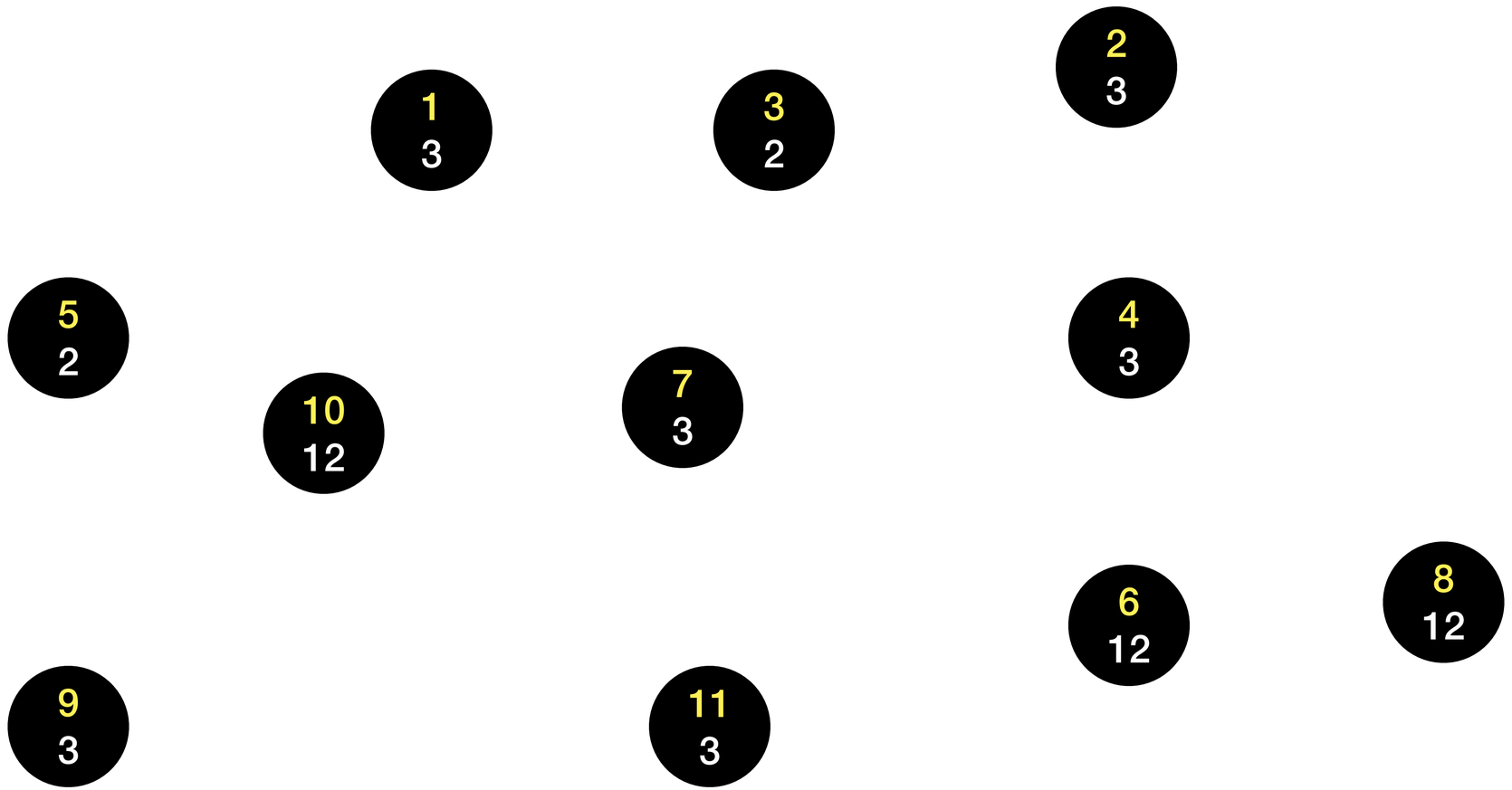


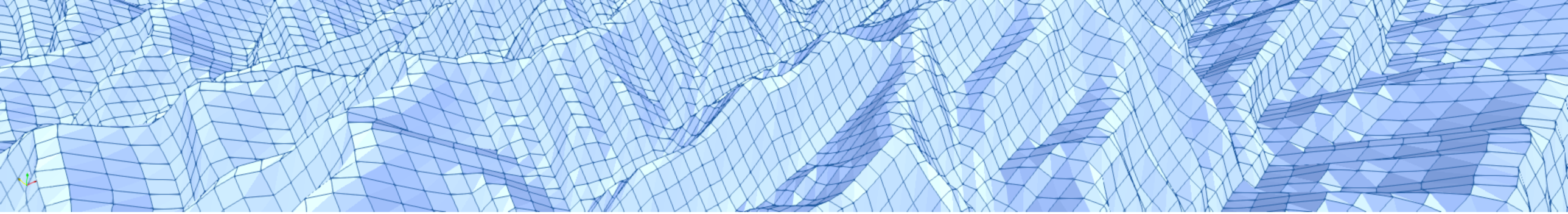
FastScape

2. Computing the donors

The receiver list, r_i, can easily be inverted to find the number of donors n_i of each node and their list, $d_{i,k}$ for $k = 0, \cdots, n_i - 1$




1
3
3
2
2
3
5
2
10
12
7
3
4
3
6
12
8
12
9
3
11
3



FastScape

3. Computing the stack order

The stack order is computed by starting along the boundary (base level nodes) and going through the list of donors in a recursive manner:

1. Initialise stack: $s = 0, n_s = 0$

2. Find next boundary/base level node $i = i_0$

3. Update stack recursively $update\_stack(s, n_s, i, d_{i,j}, n_i)$, with:

$update\_stack(s, n_s, i, d, n)$:

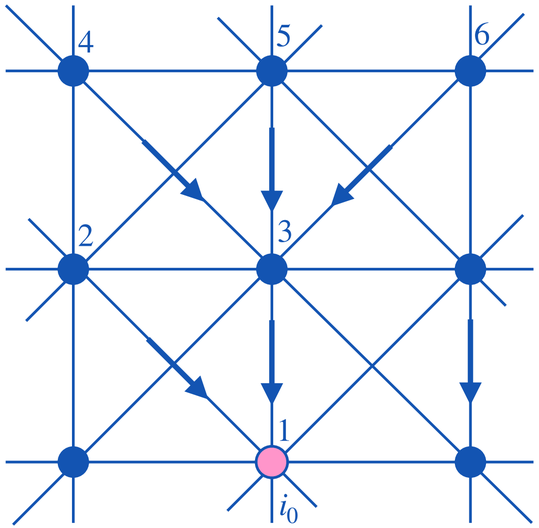
$s(n_s) = i$

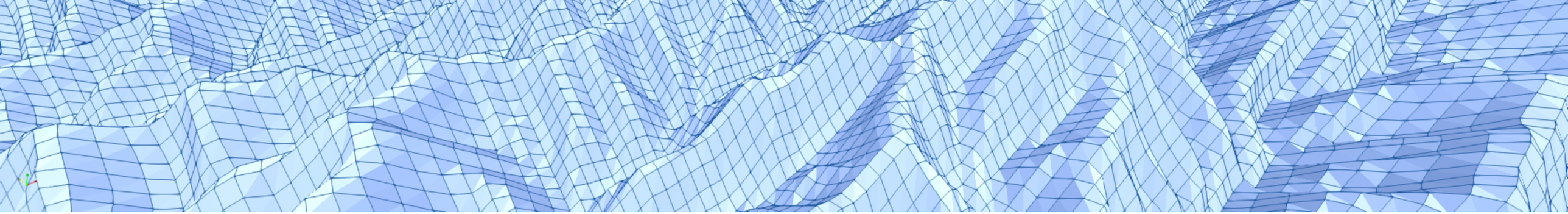
$n_s = n_s + 1$

$for\ j\ in\ n_i$:

$s, n_s = update\_stack(s, n_s, d_{i,j}, d, n)$

Goto 2





FastScape

4. Computing the contributing area

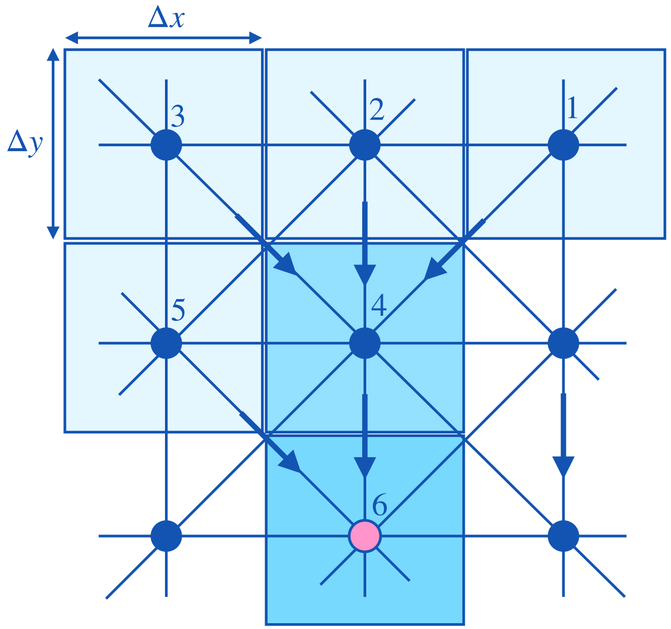
The contributing area for all nodes is obtained by a single summation of the individual node contributing area ($\Delta x \times \Delta y$) over all the nodes in reverse stack order.

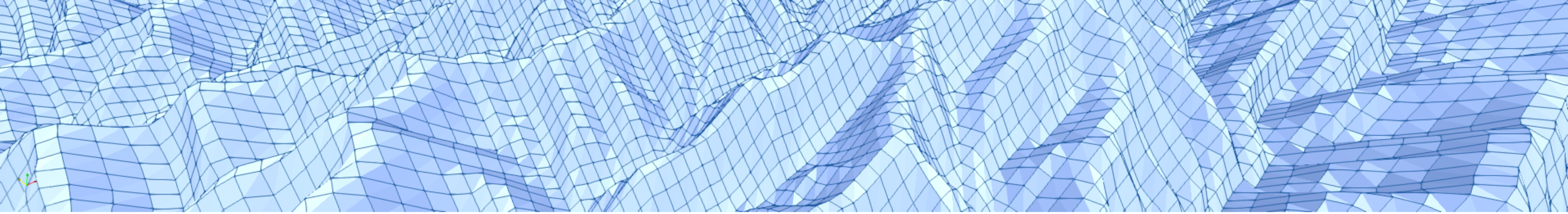
In this operation, all contributing areas are first initialised to $\Delta x \times \Delta y$:

$A_i = \Delta x \times \Delta y$ for all i

Then each nodes adds its current contributing area to that of its receiver, in reverse stack order

$A_{r_i} = A_{r_i} + A_i$ for i in reverse stack order

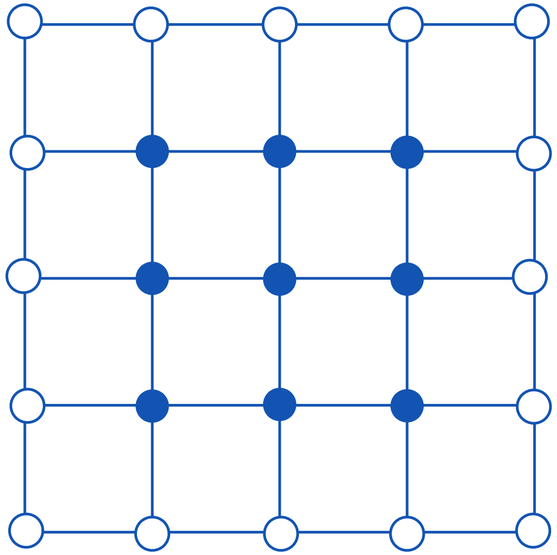


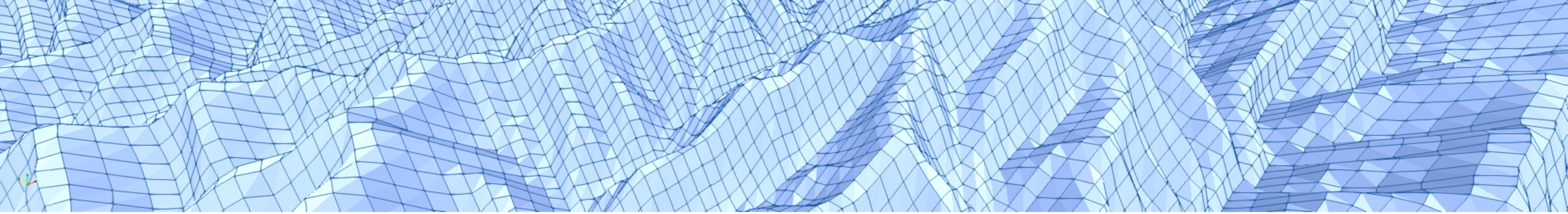


FastScape

5. Add tectonic uplift

Tectonic uplift must only be added to the interior nodes, i.e., those that are not on the boundary.





FastScape

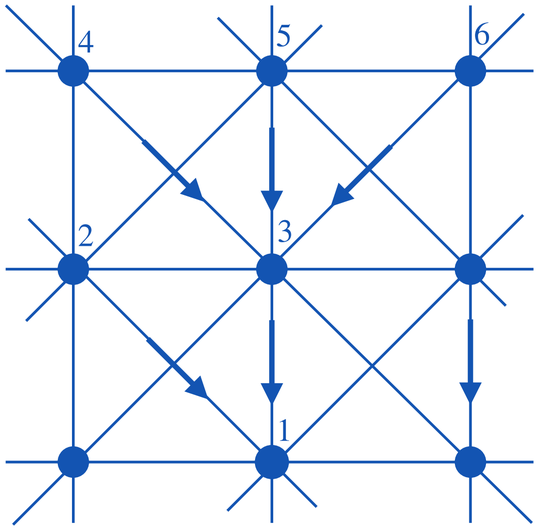
6. Computing the new elevation

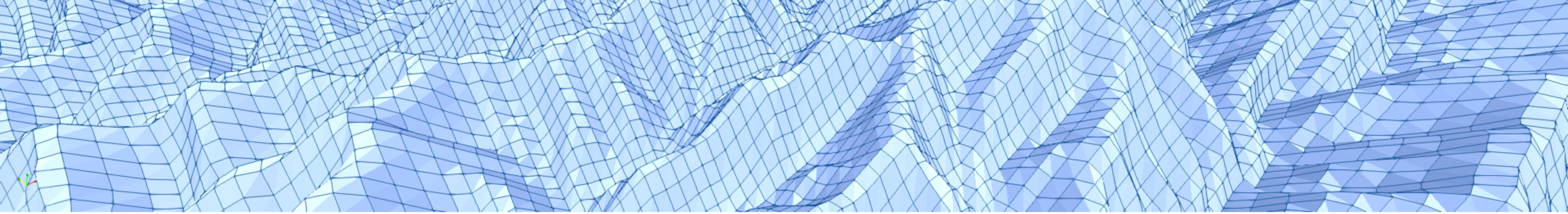
To solve the SPL, we use an implicit backward finite difference scheme:

$$\frac{h_i^{k+1} - h_i^k}{\Delta t} = -KA_i^m \frac{h_i^{k+1} - h_{r_i}^{k+1}}{\Delta l}$$

under the assumption that the slope exponent, n, is 1. Δl is the length of the link between node i and its receiver node, r_i ($= \Delta x$, Δy or $\sqrt{\Delta x^2 + \Delta y^2}$). If we perform this operation in stack order, we ensure that when computing the new height at node i, the receiver node height at time t_{k+1} has already been computed. This implies that there remains only one unknown per equation and its value can be estimated using:

$$h_i^{k+1} = \frac{h_i^k + F h_{r_i}^{k+1}}{1 + F}, \text{ where } F = \frac{KA_i^m \Delta t}{\Delta l}$$





FastScape

Unconditional stability

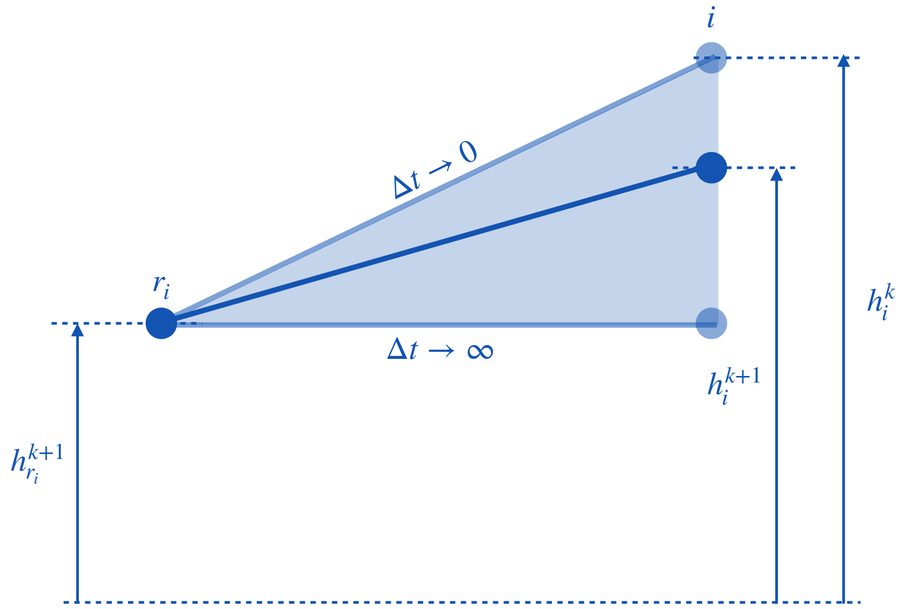
One can easily show that this implicit algorithm is unconditionally convergent (i.e., for all values of the time step, Δt). Indeed, assuming $U_i = 0$, we have:

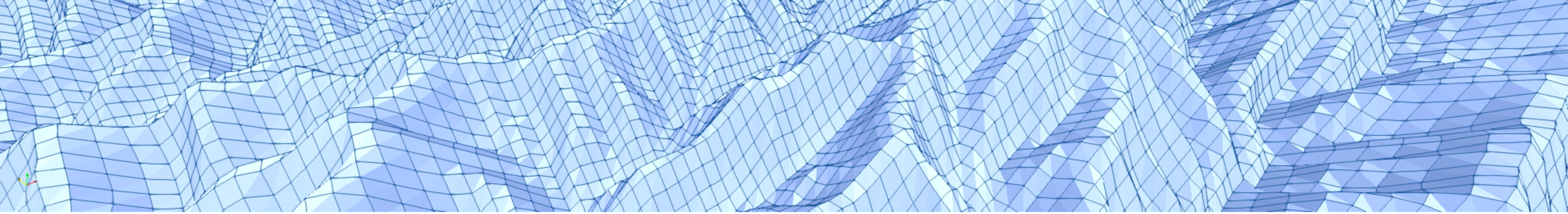
$$\lim_{\Delta t \to 0} h_i^{k+1} = \lim_{F \to 0} h_i^{k+1} = h_i^k$$

and

$$\lim_{\Delta t \to \infty} h_i^{k+1} = \lim_{F \to \infty} h_i^{k+1} = h_r^{k+1}$$

However, Campforts and Govers (2015) have shown that it is less accurate than (explicit) higher order finite difference scheme.





FastScape

Non-linear case: $n \neq 1$

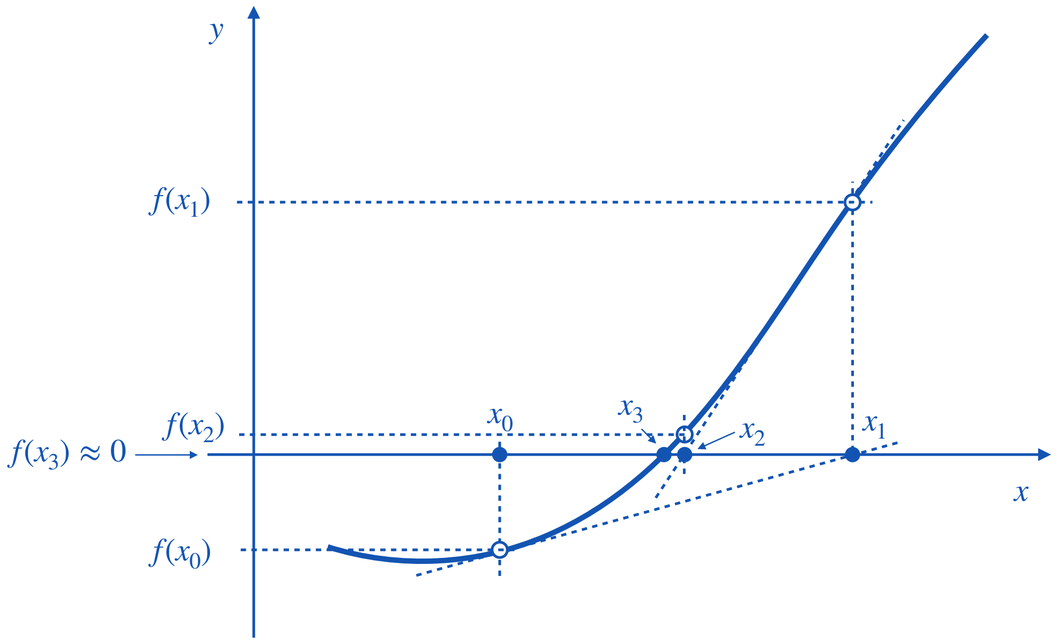
If $n \neq 1$, we must find the root of a non-linear function given by:

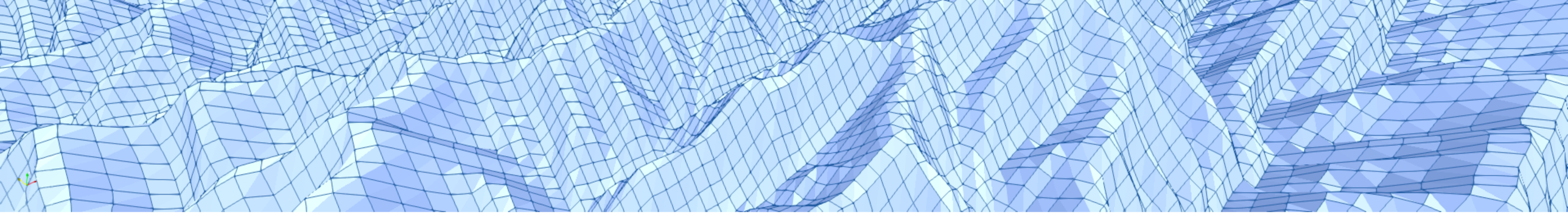
$$f(h_i^{k+1}) = h_i^k - \frac{KA_i^m \Delta t}{\Delta l^n}(h_i^{k+1} - h_{r_i}^{k+1})^n - h_i^{k+1} = 0$$

For this we use an iterative Newton-Raphson scheme:

$$x_{new} = x_{old} - \frac{f(x_{old})}{\partial_x f(x_{old})}$$

which can be shown to be unconditionally and rapidly convergent.





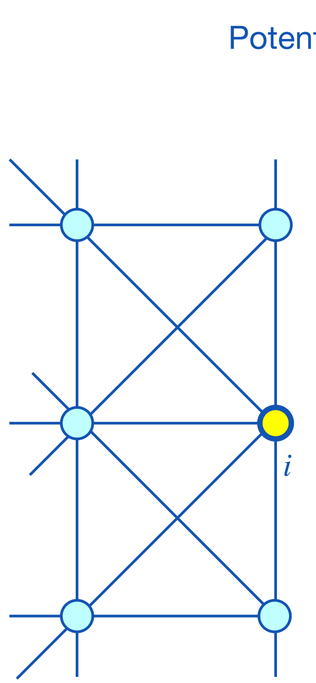
FastScape

Other types of boundary conditions

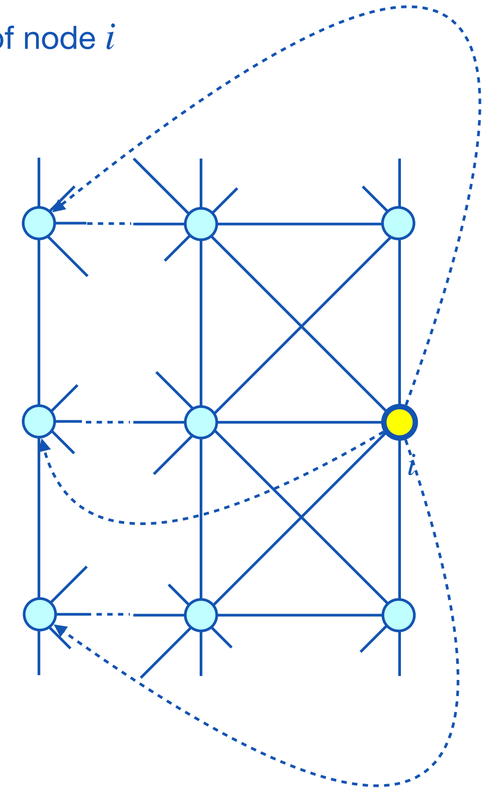
The algorithm can be easily adapted to other types of boundary conditions.

To impose a no-flux, reflective boundary condition, the boundary nodes are allowed to have receivers but only towards the interior of the domain.

To impose a cyclic boundary condition, the boundary nodes are allowed to have receivers in all directions, with the condition that the receiver nodes outside of the domain are replaced by the equivalent nodes on the opposite boundary.



Reflective boundary condition



Cyclic boundary condition