Pacific Graphics 2024
R. Chen, T. Ritschel, and E. Whiting
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 7

FastFlow: GPU Acceleration of Flow and Depression Routing for Landscape Simulation

Aryamaan Jain[1], Bernhard Kerbl[2], James Gain[3], Brandon Finley[4], Guillaume Cordonnier[1]

[1]Inria, Université Côte d'Azur, France, [2]TU Wien, Austria, [3]University of Cape Town, South Africa, [4]University of Lausanne, Switzerland

Abstract

Terrain analysis plays an important role in computer graphics, hydrology and geomorphology. In particular, analyzing the path of material flow over a terrain with consideration of local depressions is a precursor to many further tasks in erosion, river formation, and plant ecosystem simulation. For example, fluvial erosion simulation used in terrain modeling computes water discharge to repeatedly locate erosion channels for soil removal and transport. Despite its significance, traditional methods face performance constraints, limiting their broader applicability.

In this paper, we propose a novel GPU flow routing algorithm that computes the water discharge in $\mathcal{O}(\log n)$ iterations for a terrain with n vertices (assuming n processors). We also provide a depression routing algorithm to route the water out of local minima formed by depressions in the terrain, which converges in $\mathcal{O}(\log^2 n)$ iterations. Our implementation of these algorithms leads to a $5\times$ speedup for flow routing and $34\times$ to $52\times$ speedup for depression routing compared to previous work on a 1024^2 terrain, enabling interactive control of terrain simulation.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Shape modeling; Massively parallel algorithms;***

1. Introduction

Natural landscapes have been a source of fascination for generations of artists and scientists. As a consequence, the effective digital representation of terrain and surface features is crucial in several domains. In computer graphics, realistic landscapes provide a backdrop for films and games. In earth and environmental sciences, it supports investigations of the natural forces that shape our planet. In geographic information systems (GIS), it serves as the backbone of digital twin technologies that, in turn, support various forms of decision-making based on geospatial analysis.

In such applications, the inclusion of material (such as water [CBC*16] or sediment [YBG*19]) flow is ubiquitous and its impact cannot be ignored. Water affects surface appearance directly through above-ground accumulation in rivers and lakes, and indirectly through below-ground infiltration and the consequent supply of water to plants through osmosis. Over geological timescales, fluvial erosion driven by water flow sculpts mountains and valleys. Furthermore, water transforming through evaporation and condensation between the land surface and atmosphere drives weather and climate. Consequently, modeling the flow of material is a critical task in many applications involving terrain (Figure 1). (We will refer, without loss of generality, to the specific instance of water flow in the remainder of the paper, as our techniques extend trivially to other materials.)

Instead of treating water explicitly (for instance by solving the

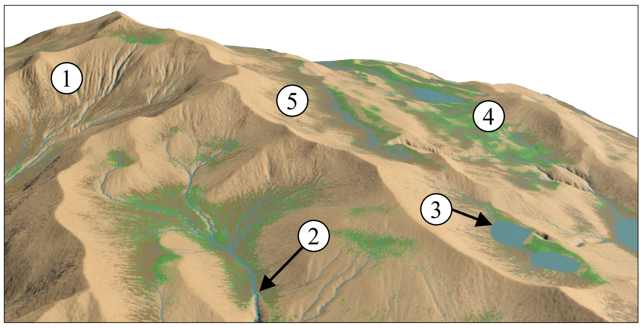


Figure 1: *Our GPU flow and depression routing algorithms can be applied to accelerate multiple aspects of landscape simulation, including 1) fluvial erosion, 2) rivers, 3) lakes, 4) ecosystems, and 5) sediment deposition.*

Shallow Water Equations [Ben07]), researchers in computer graphics [CBC*16], hydrology [GM97] and geomorphology [BW13] have proposed methods for computing water discharge (the volumetric flow rate of water). This is based on the observation that discharge is the upstream integral of precipitation. Typically, a single simulated blanket of rain is applied and runoff is then progressively accumulated from higher elevations downwards to minima along the bounding edges of the terrain. This is complicated due to the presence of local minima (or depressions) in the terrain interior, which trap the flow. This necessitates a global computation of routing through chains of depressions toward the outflow boundary.

For clarity, we define *flow routing* as the computation of the discharge – or any other material flux – over the terrain, and *depression routing* as the computation of the water flow path out of depressions.

While previous work has addressed both flow and depression routing, with optimal solutions for the CPU [CBB19], and a separate focus on distributed computing [Bar16], existing solutions for the GPU [Bar19, SPF*23] are inefficient for flow routing and typically do not consider depression routing at all.

In this paper, we present a GPU algorithm to solve *both* flow and depression routing within the same framework. In the former case, we express flow routing as an accumulation (or scan) across a tree. We use a rake-compress algorithm [SAF05], which combines pointer jumping and tree pruning, to perform this accumulation efficiently. In the latter case, we cast depression routing as an instance of searching for a minimum spanning tree and are thus able to adapt Boruvka's algorithm [Bor26, VHPN09]. Our algorithms for flow and depression routing over a terrain with n nodes complete, respectively, in $\mathcal{O}(\log n)$ and $\mathcal{O}(\log^2 n)$ iterations, assuming n processors. Furthermore, we provide optimized implementations of our algorithms in PyTorch and TensorFlow with custom CUDA kernels, which are well-suited for integration into existing terrain modeling pipelines.

We demonstrate its practicality through three example applications: river generation, terrain erosion, and ecosystem simulation. These use cases illustrate that with minimal modification, our algorithms are applicable to terrain modeling, geospatial analysis, and simulation models in computer graphics, geomorphology, and ecology. In particular, we show how to adapt our GPU algorithm to an implicit time-stepping scheme for erosion simulation using the Stream Power Law. This reduces the number of required time steps and significantly enhances interactivity. We also present a new strategy to account for sediment deposition.

Finally, we benchmark our solution against CPU and distributed computing variants, as well as previous GPU solutions. Our GPU implementation for flow routing provides a $5\times$ speed up on a 1024×1024 resolution terrain over competing GPU implementations, while depression routing gains $34\times$ to $52\times$ speedup compared to parallel CPU approaches, depending on the variant of our algorithm. This improvement in performance enables applications in natural phenomena including river and lake modeling, terrain erosion, and sediment deposition, to cross the threshold and achieve interactive response times, especially when flow and depression routing need to be recomputed over many iterations.

To summarize, our contributions are: 1) an efficient parallel algorithm for flow and depression routing, 2) an accompanying implementation on the GPU, and 3) demonstrations of its suitability for typical application areas.

2. Related Work

While the task of computing water flow over a terrain is a mainstay of hydrology and geomorphology, it is also directly applied in computer graphics to the modeling of natural phenomena, in particular the generation of terrains and placement of vegetation.

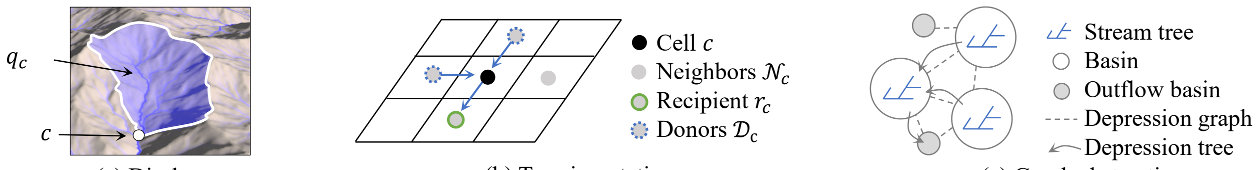
Terrain Simulation. Methods for generating terrain for use in computer graphics can be broadly categorised [GGP*19] into procedural (using algorithmic rules to mimic emergent properties), example-based (learning structure from existing terrain data), and simulation (mathematically emulating natural processes). Within terrain simulation, fluvial erosion is recognized as a primary force in shaping the topography of mountains. Early erosion methods in computer graphics [MKM89] simulated water dynamics directly using shallow water equations [Ben07] or smoothed particle hydrodynamics [KBKv09]. Unfortunately, water dynamics are very short term and need to be applied many thousands of times to capture long-term erosion processes.

Instead, large-scale terrain erosion models [CBC*16] take their inspiration from geomorphology [BW13] and directly compute total water *discharge* by accumulating precipitation from high (mountain ridges) to low elevations (the sea). This flow accumulation is communicated across large stretches of the terrain and is thus less local and less amenable to parallelisation [VBHS11] than direct water dynamics. However, this is more than offset by the sheer number of iterations required for a dynamics solution to reach steady-state. Schott *et al.* [SPF*23] provide an approximate variant of discharge-based Stream Power erosion that propagates discharge by a few cells on each time step. While this strategy is trivially parallelizable, it requires a stable river network and thus precludes outside terrain forces such as time-dependent tectonics or sediment deposition. Furthermore, the underlying explicit time-stepping scheme requires many timesteps, while our method is amenable to an implicit scheme that overcomes this constraint.

Ecosystem Simulation. In ecosystem simulation, soil moisture is one of the viability criteria for plant growth. Here, flow maps based on monthly precipitation patterns [GLCC17] are used to supply moisture inputs for plant growth. This has a particularly strong effect in riparian areas. The most commonly used per-cell moisture proxy, the Topographic Wetness Index [RKKL21], is based on a combination of slope and catchment area, with the latter derived directly from flow routing. Unfortunately, ecosystem simulations suffer from long run times on the order of hours and one of the roadblocks is flow calculation, particularly if the simulation is run at a weekly or daily time-step granularity.

Flow Routing. The problem of calculating water discharge over a terrain is well-studied and has given rise to many competing algorithms. Some methods are adapted to regular grid [GM97, OM84], and others to triangulated irregular network (TIN) [Ban07]. In the case of grids, the connections between cells can be either 4- or 8-way, depending on whether or not the diagonals are included. While our method is equally well-suited to TINs and 4- and 8-connectivity grids, for implementation purposes we currently focus on optimizing for 4-connectivity grids.

A further distinction lies in how these techniques direct flow to neighboring destination cells from a given source. The most common choice is Single Flow Direction (SFD), in which water always



(a) Discharge (b) Terrain notations (c) Graph abstractions

Figure 2: *(a) The discharge q_c corresponds to the integral of the precipitation p upstream of a cell c (Image courtesy of [CBC*16]). (b) Notation for the connectivity between terrain cells. (c) Two levels of graph abstraction. The stream trees cover the terrain grid and abstract the connections from each node to its recipient. At a higher level, the depression graph connects basins, which are regions of the terrain sharing a common local minimum. This depression graph is integral to the depression routing algorithm, which removes local minima: we update the recipients guided by the depression tree rooted in an outflow basin, defined as the minimum spanning tree of the depression graph.*

flows toward one of the lower neighbors of a given cell. The mechanism for selecting among lower cells matters in landscape simulation [TGSC24]. A stochastic selection leads to more natural-looking results than, for instance, always selecting the neighbor with the lowest elevation. In contrast, Multiple Flow Directions (MFD) splits water across multiple destinations in function of the difference in elevation [QBCP91, Coa20]. Our method handles all SFD variants and is agnostic to the choice of the single destination. However, achieving $\mathcal{O}(\log(n))$ complexity is enabled by a tree data structure formed by SFD connections and this is not trivially extendable to the Directed Acyclic Graph (DAG) required by MDF.

There have been previous efforts to accelerate this problem by exploiting parallelism, either on CPU [Bar17] or GPU [Bar19, SPF*23]. These approaches parallelize the propagation of flow among independent flow paths but do not accelerate propagation within the paths themselves, leading to $\mathcal{O}(l)$ iterations, where $l \approx \sqrt{n}$ is the length of the longest channel and n the number of cells. In contrast, our approach requires $\mathcal{O}(\log(n))$ iterations.

Depression Routing. Crucially, previous GPU algorithms do not account for depressions or local minima in the topography, which are prevalent in terrains that have not been subject to depression removal [WQZ19]. In nature, rather than breaking the flow path, such depressions progressively fill and eventually overflow. Depression routing, therefore, aims to find paths out of all depressions, such that water can flow from any point on the terrain to a boundary.

Barnes *et al.* [BLM14] proposed the use of a priority queue in progressively flooding from the boundary to the interior, filling depressions in the process, leading to a $\mathcal{O}(n\log(n))$ algorithm. Subsequent work reduced dependence on the priority queue data structure [ZSF16, WZF18] and investigated CPU parallelization [DLM11, Bar16, ZLFS17, BCW20]. Ultimately, the priority queue was dispensed with altogether by exploiting the observation that depression routing is an instance of a Minimum Spanning Tree search, which can be found in linear time on a planar graph [CBB19]. We build on this idea and adapt it to the GPU.

3. Overview

Our stated aim is to efficiently calculate the direction and volume of water flow over an input terrain with due consideration of local depressions. We choose to represent the terrain as a 2D regular grid of elevations z, often referred to as a Digital Elevation Model (DEM). This is an implementation decision and adaptation of our algorithms to other heightfield structures, such as Triangular Irregular Networks [Ban07], is straightforward. Note, however, that z is a unique mapping $h(x,y) = z$ over the x,y plane and so we do not support true 3D terrains with caves and overhangs [GGP*19].

There are three layers of terrain input to our algorithms. The first is the elevation grid. We refer to each element of the grid as a *terrain cell c*, with n being the total number of cells. We prefer to use the term "cell" as opposed to the more familiar "vertex" or "node" in this context because we want to make clear that this represents a rectangular area on the terrain surrounding the sample. The neighbors of c, denoted by $\mathcal{N}_c$, are the 4 cells directly adjacent to c in the x and y directions (Figure 2 (b)). The second input layer is the amount of water (or any loose material) initially deposited on the terrain and subject to flow. This can be interpreted as rainfall or precipitation and is denoted as p_c. The final layer is a user-defined boolean mask $b_c \mapsto \{0,1\}$ indicating the presence (1) or absence (0) of outflow cells. These serve as boundary conditions and act to trap water flow. Typically, outflow cells would line the edges of the grid, but there are cases, such as groundwater sinks, estuaries, and sea shores where they would be placed within the domain.

We output two maps: one for the per-cell discharge, and another for per-cell recipients corrected by depression routing to yield uninterrupted water paths.

Given these preliminaries, we now provide an overview of the flow and depression routing algorithms.

3.1. Flow routing

Rain falling on a landscape flows downstream and progressively accumulates into streams and rivers. The volume of water flowing through a river cross-section per unit of time is called the discharge. Equivalently, the discharge is the upstream integral of the precipitation map. *Flow routing* involves the computation of this discharge for each cell of the terrain grid.

The direction of the water flow is discretized on the basis of a per-cell *recipient*, r_c, which is chosen among the neighboring cells of the terrain and obeys the condition $z_c > z_{r_c}$. This allows several mechanisms for the choice of neighbour. While the different choices have their place and are compatible with our algorithm, we prefer a random selection, with a probability based on the elevation difference between the source and potential recipient [TGSC24].

Let us consider the cells of a terrain as nodes in a graph, and interpret the connection between a cell and its recipient as a directed edge. The strict monotonically decreasing elevation of recipients guarantees that the graph does not contain any cycles. Since each cell has a single recipient (equivalently, a single edge leaves each node), this graph represents a set of trees (an algorithmic *forest*).

We use the term *stream tree* for each connected subgraph, since it represents a set of water channels. This allows flow routing to be recast as an accumulation (or *scan*). For a given stream tree, the scan proceeds from its leaves (ridges of the terrain) to a single root (an outflow cell). We propose employing *rake and compress* [SAF05] algorithms to perform this task in parallel (Section 4).

We handle the user-defined outflow mask by excluding recipient edges for its cells so that they become root nodes in the stream forest. However, other cells may also lack recipients if none of their neighbours meet the $z_c > z_{r_c}$ condition, i.e., local minima and flat areas. In reality, instead of interrupting flow, these areas would fill and overflow. This necessitates a solution for depression routing.

3.2. Depression routing

We are now in a position to define what a *depression* means in our context. As a preliminary, we use the term *basin* for the set of terrain cells that share the same stream tree, effectively all belonging to the same catchment area and channeling water to a single root cell. If this root cell is an outflow, then the basin is classified as an outflow basin, otherwise, it constitutes a depression (Figure 2 (c)).

Depression routing is the task of finding reasonable water paths out of all depression basins. Unfortunately, this problem is non-local and therefore not amenable to trivial parallelization. For instance, an obvious first thought would be to connect a depression to an adjacent basin at the point of the lowest neighboring elevation, but this may give rise to a chain of connections that form a cycle, and therefore is unable to route the water toward outflow cells.

Two families of solutions have been proposed to tackle this issue. One option is to progressively flood the terrain from the outflow cells towards the interior, keeping track of cells to be processed through a priority queue [BLM14]. While this enables a simple and efficient single-threaded solution, it is not suited to parallelization on the GPU, because it imposes a sequential order, with each processed cell potentially changing the subsequent state of the queue. The other option is to build a *depression graph* by representing depressions as nodes and their adjacency relationships as edges. There are two subtleties. First, all outflow basins are grouped into a single node marked as the start. Second, edges are weighted according to the lowest altitude along their shared boundary. The path out of all depressions is then given by the Minimum Spanning Tree (MST) of the depression graph, which connects all depressions to the collection of outflow basins.

In Section 5, we detail our GPU implementation of depression routing and propose two strategies for correcting the flow path: either creating a jump connection between a local minimum in the current depression and a cell in the next basin or preserving the continuity by inverting flow for a subset of cells in the depression.

4. Flow routing

As a reminder, *flow routing* involves computing the *discharge* q_c for a cell c resulting from the accumulation of precipitation p over a stream tree. This tree is implicitly defined by the relationship between a cell c and its downstream recipient r_c. We can also travel upstream by defining a *donor* of c as a node d, such that $r_d = c$. We write the set of up to 4 donors of c as $\mathcal{D}_c$. In practice, we store the donors in an $n \times 4$ matrix, with an accompanying list of the number of donors per cell. This matrix is derived by parsing the cells in parallel. Each cell increases the donor count of its recipient r with atomic operations and adds its position to the donors of r.

Water flows downstream from donor cells to recipients, and therefore the *flow routing* problem consists of accumulating all of the upstream precipitation:

$$q_c = p_c + \sum_{d \in \mathcal{D}(c)} q_d. \tag{1}$$

Note that this is a recursive definition so that the discharge at a cell is the sum of the precipitation falling on the cell and the discharge from all donor cells.

Algorithm 1: Flow Routing: downstream accumulation

```
Input  : Terrain cells $\mathcal{T}$ with initial discharge
         (precipitation) $q = p$ and their donors
Output: Updated accumulated discharge
1  for $i \leftarrow 1$ to $\log_2(|\mathcal{T}|)$ do
2      foreach cell $c \in \mathcal{T}$ in parallel do
3          foreach donor $d$ of cell $c$ do
4              if $d$ is a leaf then
5                  $q_c \leftarrow q_c + q_d$
6                  remove edge $d \rightarrow c$
7              end
8              else if $d$ has a single donor then
9                  $q_c \leftarrow q_c + q_d$
10                 donor of $c \leftarrow$ donor of $d$
11             end
12         end
13         if all donors are removed then
14             tag $c$ as leaf
15         end
16     end
17 end
```

With the exception of some cells marked by the outflow mask, stream trees cover all of the terrain, which makes the design of an efficient parallel solution to Equation 1 non-trivial.

Fortunately, this problem can be cast as an instance of Parallel Tree Accumulation, which aims to compute the cumulative sum of quantities in tree nodes, from leaf to root. This can be achieved with straightforward pointer jumping, but this may result in a $\mathcal{O}(n)$ span due to the required atomic operations. An alternative is to use an Euler tour [SAF05] along the tree edges combined with pointer jumping, which reduces the span to $\mathcal{O}(\log n)$, since the number of edges is less than the number of nodes. However, this method typically requires $2\log_2 n$ iterations, as each edge is traversed twice.

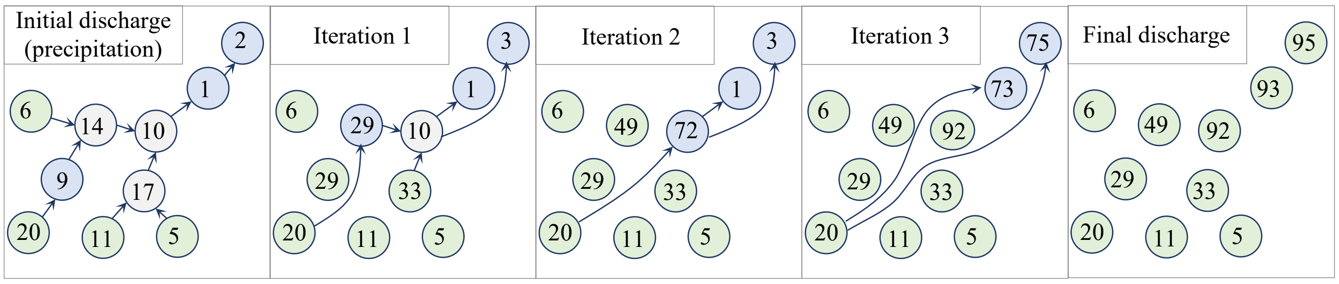


Figure 3: *A few iterations of a parallel rake-compress solution to flow routing. On each iteration, leaf nodes (in green) are pruned and recipients of cells with a single donor (in blue) are rerouted.*

A still more efficient approach is rake-compress, which combines pointer jumping with leaf pruning to achieve a span of $\mathcal{O}(\log n)$. This method typically needs only $\log_2 n$ iterations, making it faster in practice than the Euler tour. Sevilgen *et al.* [SAF05] provide a starting point in the form of a general rake-compress solution to graph accumulation problems for distributed architectures. This relies on the observation that leaves constitute half the nodes in a balanced binary tree. So, if we iteratively transfer accumulated discharge from a leaf to its recipient and then prune the leaf and do this in parallel across all leaves, we only need to iterate $\log_2 n$ times where n is the number of nodes in the terrain.

In general, river trees might be neither binary nor balanced. In the worst case, where a river tree is composed entirely of single donors, the iterations degrade to $\mathcal{O}(n)$. To compensate, we use pointer jumping by re-routing the edges of cells whose donor itself has a single donor (parent). Effectively, the parent (donor) is skipped and an edge is formed directly from the grandparent (donor of the donor). Thereby, worst-case performance is restored to $\log_2(n)$ parallel iterations.

As shown in Algorithm 1, with an accompanying example in Figure 3, on each iteration and for each terrain cell c, we parse each donor d in parallel, with up to 4 donors in our implementation. If d is a leaf, we add its discharge to c and remove it from the donor matrix. Alternatively, if d has only a single donor, we again add its discharge to c but in this case reconfigure the donor matrix so that c instead points at the donor of d. Distinguishing between these cases requires a single lookup in the donor count list, *e.g.*, leaves have a donor count of zero. Straightforward extensions of Algorithm 1 can be used for other flow accumulation functions, including those involving maxima or weighted sums of upstream values.

Note that some care must be taken in the implementation of Algorithm 1, which exhibits potential read-after-write (RAW) hazards when updating the discharge and recipients. An immediate solution is to write the updated q and r to temporary arrays and copy them back at the end of each iteration. Unfortunately, this forces the unnecessary transfer on each iteration of a large number of cells that are unmodified. Instead, we propose a *per-cell ping pong scheme*. We first allocate sufficient space to hold the state of two full terrains, T_A and T_B. We then introduce an additional *source* buffer with one 8-bit integer per cell, initialized to 0. The sign of the integer at cell c in the *source* indicates whether data should be read from T_A and written to T_B, or vice-versa. When cells are updated, the sign is flipped. To avoid RAW hazards within the same iteration, the remaining bytes of each *source* entry store the iteration at which c was last updated: a thread will check the *source* to negate the read sign if the stored iteration is the current one. At the end of each iteration, the latest state of each cell is either in T_A or T_B, as indicated by *source*. Hence, a final merge step is required before returning the result as a single tensor. Incorporating this optimization leads to a $14\times$ speed up on a 1024×1024 terrain.

5. Depression routing

To match the expected natural behaviour, water needs to be routed out of local depressions. Our strategy is to create new donor-recipient connections and reroute others to ensure an uninterrupted, minimum-cost path from any node to one of the designated outflows. In this context, we define the minimum cost to be the smallest increase in altitude needed to move water out of a depression.

Broadly speaking, this involves connecting depression basins with outflow basins through saddle points and then re-orienting a path of edges within each depression to flow uphill and link to the corresponding outlet. The challenge lies in preventing the formation of cycles as this will lead to highly unnatural flow behavior. Once all stream trees have been corrected, we only need to run flow routing once to calculate discharge volumes.

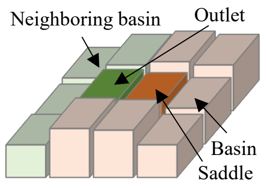
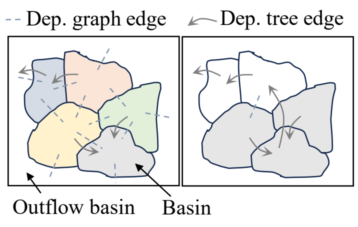


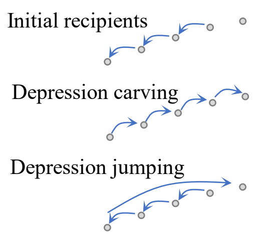
Figure 4: *A saddle between basins: With respect to a source basin (in light red) and its neighbor (light green), the saddle and outlet together form the lowest elevation bridge.*

Depression routing consists of building a depression graph, in which the depression and outflow basins represent nodes, and edges are formed by the adjacent cells between different basins. It is likely that there will be multiple potential candidates between any two basin nodes formed by cells paired along their common border. We choose the one with the lowest maximum altitude and call this the *saddle* (see Figure 4). The saddle altitude is the weight assigned to the new edge in the depression graph. From a physical perspective, choosing the path with the lowest altitude ensures that the potential energy cost of water routed out of the basin is kept to a minimum.

Following Cordonnier *et al.* [CBB19], the minimum cost route out of depressions can now be cast as a minimum spanning tree problem. Among the several MST algorithms (*e.g.*, Prim's, Kruskal's), Boruvka's [Bor26] is perhaps the most amenable to parallelization [VHPN09]. Boruvka's Algorithm constructs a minimum spanning tree by iteratively identifying and merging the minimum-weight edges of each component in parallel until only one component remains. We note that the optimal theoretical complexity for parallel MST algorithms is $\mathcal{O}(\log n)$ [CHL01], but the specific algorithmic improvements needed to achieve this are not portable to the GPU, as they involve complex branching and scheduling operations. Therefore, we build our algorithm upon the more usual $\mathcal{O}(\log^2 n)$ variant.



(a) Iterations of depression routing



(b) Re-routing recipients

Figure 5: *(a) Two iterations of depression routing. The grey basin on the left does not have an outgoing edge, likely because it would have created a loop with a basin of a higher id. In the next iteration, basins are recolored based on their new local minima, and only one edge remains. (b) From the initial configuration of the recipients (top), we apply either depression carving, which reverses the path of water until it reaches the outlet, or depression jumping, which directly connects the local minimum with the outlet.*

Boruvka's algorithm consists of greedily forming sub-trees by treating each as a single collapsed node. It begins by treating all nodes in the initial graph as separate sub-trees. Then, each of these is grown by incorporating the incident edge of minimal weight (and its attached sub-tree). The two sub-trees are joined, their shared edge is collapsed, and the algorithm repeats until a single tree results. The parallel variant of Boruvka's algorithm [VHPN09] applies edge selection in parallel and replaces the edge collapse with a structure that maintains the connected components of the tree under construction. This requires at most $\log_2(V)$ iterations, where V is the number of vertices in the graph (depressions in our case) (Figure 5 (a)). Each of these iterations parses the structure for connected components, adding an extra $\log_2(V)$ iterations at worst, and resulting in an overall complexity of $\mathcal{O}(\log^2(V))$.

While we could build an explicit depression graph, compute a minimum spanning tree, and parse the tree to update recipients, such a construction would be costly as it involves the detection of pairs of adjacent depressions. Furthermore, while Boruvka's algorithm outputs an undirected tree, we require directed edges to orient water towards outflow basins. Instead, we introduce a hybrid structure where terrain data is used in conjunction with the depression

Algorithm 2: Propagation of basin identifiers

```
Input  : Terrain cells $\mathcal{T}$, non-outflow local minima $\mathcal{L}$
Output: Per-cell basin identifiers $bid$
1 for $i \leftarrow 0$ to $|\mathcal{L}| - 1$ in parallel do
2     $bid$ of $\mathcal{L}[i] \leftarrow i$
3 end
4 for $i \leftarrow 1$ to $\log_2(|\mathcal{T}|)$ do
5     foreach cell $c \in \mathcal{T}$ in parallel do
6         $bid$ of $c \leftarrow bid$ of recipient of $c$
7         recipient of $c \leftarrow$ recipient of recipient of $c$
8     end
9 end
```

graph to progressively detect the path of water and update recipients. The algorithm proceeds through the stages of basin identification, saddle edge selection, and donor-recipient re-routing.

Basin identification. Our algorithm begins by segmenting out each stream tree and associating with its cells a unique basin identifier (hereafter referred to as a *basin-id*, or *bid* in the pseudo-code of Algorithm 2). To this end, we assign to each non-outflow local minima a unique identifier – leaving a common outflow identifier for all outflow basins – and use pointer jumping to copy this identifier in parallel to all upstream nodes of its stream tree (Algorithm 2).

Saddle edge selection. We use the per-cell basin annotation to isolate cells lying along the border with neighboring basins. To do so, we tag as a border cell, any cell in the current basin with at least one adjacent cell belonging to another basin. Ultimately, one of these border cells is chosen as a saddle edge in the depression graph. We do not need to construct these edges explicitly as Boruvka's algorithm only requires the edge with the lowest weight.

For each border cell, this weight is set as the lowest elevation that will permit an overflow from it into an adjacent basin. Specifically, this is the maximum of the border cell's altitude and the lowest altitude among its neighboring cells. Then we choose the border cell with minimum weight per basin.

Although we could compute the minimum altitude using a variant of rake-compress in $\mathcal{O}(\log n)$, we found it more efficient to instead compute it in a single step with atomic operations. We also store the *saddle*, which is the source cell in the current basin with minimum weight, and the *outlet*, which is its lowest neighbouring cell across the basin border (see Figure 4).

Close-coupled loops can occur when two basins have saddle edges pointing at each other. We resolve this by simply deleting the edge with the higher saddle basin-id. We also apply a similar strategy to choose among saddles of identical elevation, which could lead to larger cycles. This is prevented by selecting saddles in lexicographic order (elevation concatenated with the basin-id of the outlet). This algorithm is summarized in Algorithm 3.

Re-routing recipients. Ordinarily, the next step in Boruvka's algorithm would be to collapse the newly incorporated edges. Due to

Algorithm 3: Finding saddles and outlets

```
Input  : Terrain cells $\mathcal{T}$ with elevation $z$, set of depression basins $\mathcal{D}$
Output: Per-depression saddle and outlet

// Compute border cells and store the border altitude in $z'$
1  foreach cell $c \in \mathcal{T}$ in parallel do
2      if $bid_c \neq bid_n$ of any 4-neighbour $n$ of $c$ ($n \in \mathcal{N}_c$) then
3          tag $c$ as a border cell
4          $znb \leftarrow \min(z_n$ for $n \in \mathcal{N}_c$ such that $bid_c \neq bid_n)$
5          $z'_c \leftarrow \max(z_c, znb)$
6      end
7  end
// Saddles and outlets (with atomic lexicographic argmin)
8  foreach basin $d \in \mathcal{D}$ in parallel do
9      $saddle$ of $d \leftarrow \text{argmin}((z'_c, bid_n)$ for each border cell $c$
         and $n \in \mathcal{N}_c$ such that $bid_c = d$ and $bid_n \neq d)$
10     $outlet$ of $d \leftarrow \text{argmin}(z_n$ for each neighbor $n$ of the
         $saddle$ such that $bid_n \neq d)$
11 end
// Remove cycles
12 foreach basin $d \in \mathcal{D}$ in parallel do
13     $d' \leftarrow$ basin of outlet of $d$
14     if bid of outlet of $d'$ = bid of saddle of $d$ then
15         if bid of outlet of $d <$ bid of saddle of $d$ then
16             Delete the saddle and outlet of $d$
17         end
18     end
19 end
```

our two-layer structure, with an implicit depression graph and explicit flow routing, this needs to be interpreted differently. Instead, we merge stream trees by adding and re-routing recipient links, using one of two strategies (Figure 5 (b)).

The first strategy, *depression jumping*, involves adding a recipient link directly from the depression's local minima to the newly established outlet cell. This is straightforward and efficient but leads to a jump discontinuity in the flow path, which, depending on the application, may be undesirable.

The second strategy, *depression carving* [Rie98], mimics the course of water channeling down from the saddle. We first compute the flow path that connects a saddle and outlet to the associated local minimum and then reverse the direction of all recipient links. In this way, we allow water to flow upstream from the local minimum to the outlet. The pseudo-code for these two strategies is presented in Algorithm 4.

A single iteration of basin identification, saddle edge selection, and re-routing of recipients ends up being insufficient. It will not necessarily connect all depressions to the set of outflow basins. However, on any given iteration this process removes at least half of the remaining depressions, meaning that a total of $\log(L)$ iterations are required, where L is the number of local minima (Algorithm 5).

Algorithm 4: Re-routing flow paths

```
Input  : Terrain cells $\mathcal{T}$, set of depression basins $\mathcal{D}$
Output: Updated recipients

1  if Depression jumping variant then
2      foreach depression $d \in \mathcal{D}$ in parallel do
3          $l \leftarrow$ local minimum of $d$
4          recipient of $l \leftarrow$ outlet of $d$
5      end
6  else
       // Depression Carving Variant
7      tag $\leftarrow False$ for all cells
8      foreach depression $d \in \mathcal{D}$ in parallel do
9          tag[saddle of $d$] $\leftarrow True$
10     end
       // recipient is local to next two loops
11     for $i \leftarrow 1$ to $\log_2(|\mathcal{T}|)$ do
12         foreach cell $c \in \mathcal{T}$ in parallel do
13             if $tag[c]$ then
14                 tag[recipient of $c$] $\leftarrow True$
15                 recipient of $c \leftarrow$ recipient of recipient of $c$
16             end
17         end
18     end
19     foreach cell $c \in \mathcal{T}$ in parallel do
20         if tag[recipient of $c$] and $c$ is not a local minimum
             then
21             recipient of recipient of $c \leftarrow$ c
22         end
23     end
24     foreach depression $d \in \mathcal{D}$ in parallel do
25         recipient of saddle of $d \leftarrow$ outlet of $d$
26     end
27 end
```

Algorithm 5: Depression Routing

```
Input  : Terrain nodes $\mathcal{T}$ with depressions; non-outflow
         local minima $\mathcal{L}$
Output: Updated terrain graph with rerouted depressions

1  for $i \leftarrow 1$ to $\log_2(|\mathcal{L}|)$ do
2      Construct basins $\mathcal{D}$ and annotate the cells (Algorithm 2)
3      Find min edges, saddles and outlets (Algorithm 3)
4      Reroute recipients (Algorithm 4)
5  end
```

Each iteration, in turn, requires $\log_2(n)$ sub-iterations, where n is the number of cells, yielding an $\mathcal{O}(\log(L)\log(n))$ algorithm.

Implementation. As with flow routing, we prevent read-after-write hazards while minimizing the extent of copying. For basin identification, we store the basin-id only for the local minima and not for all cells, and use the updated recipients at the end of Algorithm 2 as a per-cell pointer to the downstream local minima, which we interrogate to obtain the basin-id. This change relaxes the need for any copy during basin identification, as read-after-write no longer prevents the correct convergence of the algorithm. Furthermore, we observe that this operation is required only for the first iteration. In subsequent iterations, we need only update the pointers to reflect the new connections out of the local minima, yielding an overall complexity of $\mathcal{O}(\log(n) + \log^2(L))$.

Another RAW hazard can occur in the depression carving variant for re-routing flow paths. To prevent this, we use a localized ping-pong strategy (similar to the one presented in Section 4), which is simplified by using the *tag* variable as a condition for copying.

6. Application to landscape simulation

Here we show how we apply flow and depression routing to accelerate landscape simulation, with use cases in river and lake modeling, terrain erosion, and ecosystem simulation. In particular, our GPU algorithms together provide the discharge necessary for erosion simulation. Furthermore, they enable a novel GPU-based solution for implicit time-stepping of erosion using the Stream Power Law, which we combine with a simulation of sediment deposition.

Algorithm 6: Extracting a depression-free water surface

```
Input  : Terrain cells $\mathcal{T}$ with elevation $z$, small slope $\epsilon$
Output: Elevation of the water surface $w$
1 $w \leftarrow$ copy of $z$
2 for $i \leftarrow 1$ to $\log_2(|\mathcal{T}|)$ do
3     foreach cell $c \in \mathcal{T}$ in parallel do
4         $w[c] \leftarrow \max(z[c], w[\text{recipient of } c] + 2^{i-1}\epsilon)$
5         recipient of $c \leftarrow$ recipient of recipient of $c$
6     end
7 end
```

River and lake modeling. To identify the surface of rivers and lakes we begin by filling depressions in the terrain. This is sometimes referred to as *pit removal* and has value as an algorithm in its own right. Our input is the terrain elevation grid, and a set of recipients identified during depression routing to provide the flow path with minimum energy cost. We call the *water surface* the elevations obtained after filling the identified depressions. This surface should obey two conditions: it should be as close as possible to the terrain, and at the same time be monotonically non-decreasing along all stream trees, which we intuitively expect since water flows downhill. To achieve this, in Algorithm 6 we introduce a variant of the basin identification algorithm. We follow the water path of each stream tree upwards from destination to sources, retaining as the water surface the maximum elevation previously reached. For completeness, we optionally allow a small slope ϵ on the water surface as this is sometimes required in hydrology applications. Note that this variant is only possible with the *depression carving* strategy, as it requires a continuous path. Next, cells with water discharge above a specified threshold can be tagged as belonging to bodies of water, as illustrated in Figure 9 where the water surface and its discharge are highlighted.

Ecosystem simulation. The water routes derived through river and lake modeling have direct application to ecosystem simulation. Water, along with sunlight and warmth, is a key requirement for plant growth, with different species having varied adaptation to water availability. Modeling water discharge is therefore a crucial element in ecosystem simulations. To demonstrate applicability, Figure 6 shows the outcome of an ecosystem simulation for a biome in the Pyrenees mountains [PGG*24] at the 50 year mark, with water accumulation provided by our flow and depression routing. For clarity, pioneer species with strong drought tolerance have been removed and what remains are species, such as Sessile Oak and European Beech, that require the greater moisture found in riparian areas. Most ecosystem simulations use a month as the timestep granularity, due in part to the overhead of calculating water flow. The acceleration provided here creates an opportunity for shorter, weekly, or even daily, timesteps with non-uniform rainfall across the landscape [PMG*22] and hence greater simulation accuracy.

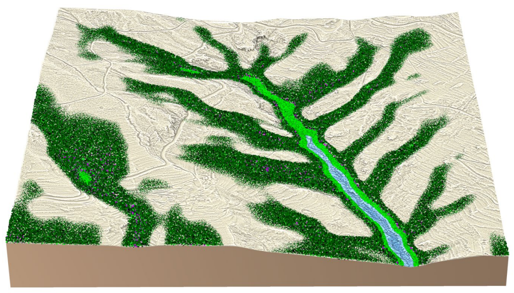


Figure 6: *Ecosystem simulation with water provided by flow routing. Certain species (colored in shades of green, black, and purple) favor river banks due to higher levels of available water.*

Terrain erosion. Flow routing is a critical component of terrain erosion, as it provides the mechanism by which material is worn away and transported, ultimately over geological time shaping the mountains themselves. Furthermore, even the ordering of recipient cells plays a role in the implicit integration of the stream power law erosion equation. This is expressed as [CBC*16]:

$$\frac{\partial z}{\partial t} = u - kQ^m \frac{\partial z}{\partial x}, \tag{2}$$

where the first term u is the tectonically induced uplift (or growth rate of the mountain), and the second term is the erosion, dependent on the discharge Q, erosion coefficients k and m, and the slope $\partial z / \partial x$. The discharge provided by flow routing is scale-independent, so to compensate we scale the discharge Q by accumulating $\Delta x^2 p$, where Δx is the cell size and p is the precipitation. The uplift is usually considered time-independent and therefore applied as a pre-process. This leaves an implicit solution to the second erosive part of Equation 2 as:

$$z_t[c] = z_{t-\Delta t}[c] - K(z_t[c] - z_t[\text{recipient of } c]), \tag{3}$$

where Δt is the time step and $K = kQ^m \Delta t / \Delta x$. Eq. 3 can be further simplified to:

$$z_t[c] = \alpha z_t[\text{recipient of } c] + \beta, \tag{4}$$

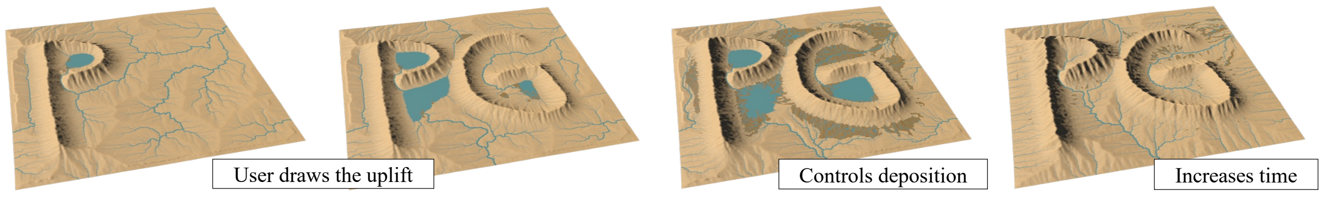


Figure 7: *Different stages in an interactive authoring session. The user paints uplift to provide an initial mountain structure, then changes the deposition parameters (sediment in dark brown and lakes and rivers in blue), and, finally, increases the overall simulation duration.*

where $\alpha = K/(1+K)$ and $\beta = z_{t-\Delta t}[c](1+K)$.

As with water surface extraction, we can compute z_t using a simple modification of basin identification (see Algorithm 7). The outcome of applying our accelerated method of fluvial erosion is illustrated in Figure 8. Note that Equation 2 is a special case of the Stream Power Law with the slope exponent set to one. Further research is required to adapt our solution to the general case, for instance with a global Newton-Raphson's algorithm.

Algorithm 7: Implicit fluvial erosion

```
Input : Terrain cells $\mathcal{T}$ with elevation $z$, per cell
        coefficients $\alpha$ and $\beta$
Output: Eroded terrain $z'$
1 $z' \leftarrow$ copy of $z$
2 for $i \leftarrow 1$ to $\log_2(|\mathcal{T}|)$ do
3     foreach cell $c \in \mathcal{T}$ in parallel do
4         $z'[c] \leftarrow z'[c] + \alpha[c] z'$[recipient of $c$]
5         $\alpha[c] \leftarrow \alpha[c]\alpha$[recipient of $c$]
6         recipient of $c \leftarrow$ recipient of recipient of $c$
7     end
8 end
```

Fluvial erosion is not the only erosive process responsible for shaping terrain. *Hillslope processes* model the gradual accumulation of solid material at the base of mountains and hills [BS97] and is usually expressed as a diffusion equation. We follow Tzathas *et al.* [TGSC24] and approximate it by including additional terms in the Stream Power Equation, changing kQ^m to $kQ^m + k_t + k_h A^{-h}$, where k_t and $h = 0.6$ are hillslope erosion parameters, A is the drainage area (obtained via flow routing with precipitation set uniformly to $p = \Delta x^2$). The parameter k_t accounts for slope-dependent effects (landslides, debris-flow), regrouped in computer graphics under the catch-all term *thermal erosion* [MKM89].

Sediment deposition. Finally, flow routing can more generally propagate any type of transportable material. A case in point is sediment deposition, which acts as a complement to hydraulic erosion. Incorporating sediment deposition requires an additional term in the Stream Power Law [YBG*19]:

$$\frac{\partial z}{\partial t} = -kQ^m \frac{\partial z}{\partial x} + k_d \frac{Q_s}{Q}, \tag{5}$$

where k_d is the sediment deposition coefficient and Q_s is sediment flux, obtained by accumulating the negative elevation balance $-\Delta x^2 \frac{\partial z}{\partial t}$ downstream with flow routing. While Yuan *et al.* [YBG*19] suggest a fully implicit solution that requires iterating between Equation 5 and the accumulation of Q_s, we propose a semi-implicit variant in which Q_s is computed from the elevation balance of the previous time-step. While our solution introduces a time dependency into mass conservation, it does not require iterating, which leads to a faster algorithm with no visual difference.

7. Results

In this section, we show the results of our method applied to the use cases detailed in section 6, and then demonstrate this performance against competitor methods.

7.1. Implementation

We prototyped our algorithms in Python with PyTorch and optimized our implementation with custom CUDA kernels. In particular, we seek to strike a balance by executing high-level functions in PyTorch or TensorFlow and handling compute-intensive kernels with CUDA. The code for this paper is available at: https://gitlab.inria.fr/landscapes/fastflow. For ease of use, this includes our standalone unoptimized PyTorch implementation and a TensorFlow port, as well as our CUDA optimizations. The repository also includes examples of how to use our code to edit terrain with *Houdini* [Sid23], as demonstrated in the accompanying video.

In terms of optimization, we reduce memory allocations and copying and prevent CPU-GPU communication in the control flow. We found that allowing the GPU to run for the maximum number of iterations (*e.g.*, $\log_2 n$ for flow routing) was more efficient than checking for early convergence. The only exception is in depression routing, where we halt the algorithm when no depressions remain.

7.2. Interactive landscape simulation

In this section, we showcase the application of our methods to interactive landscape simulation. In doing so, we compare against a recent approximate GPU solution for flow routing [SPF*23] and evaluate the impact of our new approach to sediment deposition.

As a starting point, Figure 1 shows a real digitized terrain, subsequently altered by fluvial erosion and sediment deposition, and overlaid with flow-dependent vegetation, lakes, and rivers, in under 1.5 seconds.

Interactive landscape authoring. To demonstrate the general applicability of our algorithms we provide extracts from an interactive landscape authoring session in Figure 7 and the accompanying video. Here, user-guided simulation is applied to a 512×512

terrain, which, thanks to GPU acceleration combined with implicit time-stepping, requires only 10 iterations and 0.1s to capture 700,000 years of geomorphological evolution. For comparison, a CPU implementation [CBC*16] requires 2.6s. The user first defines the primary mountains by progressively painting on an uplift map, and can then freely change simulation parameters (here the deposition constant). Finally, the user advances the age of the mountain by increasing the number of iterations to 100, producing the result on the far right in 0.7 seconds. Note that lakes disappear over time as a consequence of filling by deposition and uplift, combined with erosion, which gradually removes the obstructions between them.

Our method scales to higher-resolution landscapes, as illustrated in Figure 8 for a 2048×2048 terrain with 300,000 simulated years (200 iterations) generated in under 10 seconds.

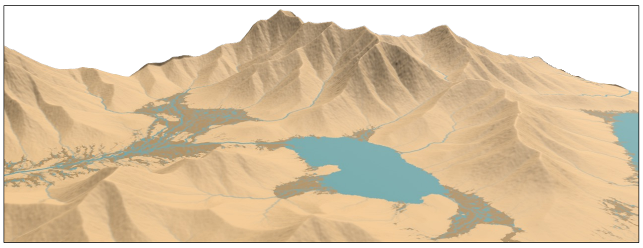


Figure 8: *A* 300,000 *year-old mountain, generated on a* 2048×2048 *grid in under* 10 *seconds, incorporating fluvial erosion, sediment deposition, and river and lake formation.*

Rivers and lakes In Figure 9, we demonstrate how our algorithms can be used on complex large-scale terrains to demarcate lakes and rivers that are registered correctly with existing erosion lines and depressions. On the left in Figure 9, a mountainous region in the Alps transitions into an urbanized flat glacial valley. On the right, a tidal estuary around Mont Saint-Michel, France, leads into the sea. They contain 383,918 and 8,445,644 basins, respectively. In both cases, the elevation data was obtained from the IGN RGE ALTI Digital Elevation 1m dataset [IGN22].

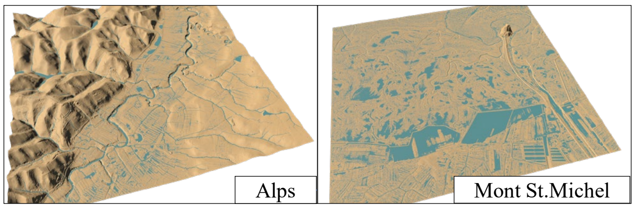


Figure 9: *River and lake identification on an alpine terrain dominated by a flat glacial valley (left), and a complex estuary (right).*

Sediment deposition. Figure 10 highlights the impact of sediment deposition. This increases the diversity of landscape patterns. In particular, sediment deposition, where present, dampens erosion, and leads to steeper slopes. Finally, we show in Figure 11 the visual impact of our semi-implicit deposition strategy. Our results are similar to a fully implicit variant [YBG*19]. We note, however, that our gain in performance comes at the cost of some material loss.

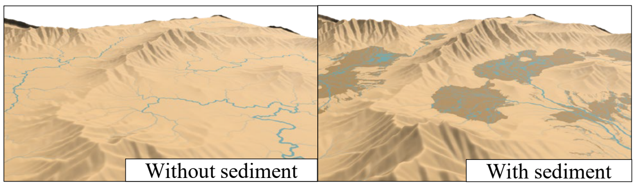


Figure 10: *Sediment deposition acts as a shield against erosion leading to more pronounced slopes.*

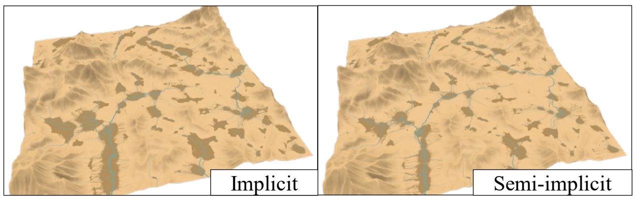


Figure 11: *The visual results of implicit and semi-implicit deposition are on par (with* 100 *iterations at* $\Delta t = 2000$, $\Delta x = 16$, *and a resolution of* 2048^2*).*

Comparative evaluation. An alternative strategy for landscape simulation on the GPU [SPF*23, JBC24] is to use explicit time-stepping for erosion combined with approximate flow routing, where the discharge is propagated by a single cell on each simulation step. This method requires many more iterations due to the stability conditions of the explicit scheme but does eventually converge to a solution qualitatively similar to ours. Crucially, there are several situations where this strategy cannot be applied. One case is the erosion of an existing terrain containing depressions. Another failure case (see Figure 12) occurs when noise is added to uplift to increase the diversity. The approximate solution relies on the assumption that the flow path weakly varies over time, which allows the approximation of the discharge to progressively improve. Adding noise invalidates this assumption and leads to a significant underestimation of drainage and, consequently, an underestimation of erosion. This is visible in Figure 12, where the maximal elevation depends on the amount of noise. In contrast, our combination of implicit-time-stepping and exact-flow does not suffer from this shortcoming.

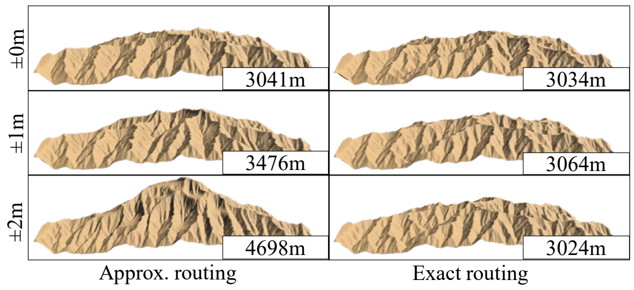


Figure 12: *Approximate flow routing [SPF*23] (left) versus ours (right), subject to noisy uplift with varying amplitude (top to bottom:* ±0, 1, *and* 2m *per timestep). Compared to ours, the elevation with approximate routing depends on the magnitude of that noise.*

Another consideration is the acceleration afforded by the larger timesteps of our implicit solution. We show in Figure 13 that a 512×512 terrain with $\Delta x = 32$m obtained with an explicit scheme ($\Delta t = 1,000$ years) is visually similar to the result of our implicit solution ($\Delta t = 20,000$ years). Our implicit scheme allows an increase of the timestep by a factor of 20, which reduces, by the same factor, the iterations needed to achieve the same total geological timespan. With such timesteps, generating a 10 million year-old landscape requires 7.2s and .5s, for the explicit and our implicit scheme, respectively. Note that larger implicit timesteps are unconditionally stable, and only diverge slightly from the explicit solution. Note that explicit and implicit schemes in general do not yield identical solutions as they accumulate discretization error differently.

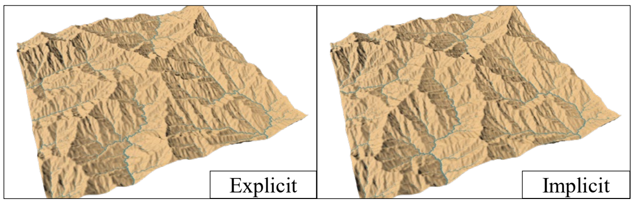


Figure 13: *Our implicit solution with $\Delta t = 20,000$ (right) provides a result similar to the explicit scheme with $\Delta t = 1,000$ (left). The latter is at its upper stability limit.*

We further motivate the need for depression routing by recreating the *escarpment* scenario from Cordonnier *et al.* [CBB19]. This begins with a flat terrain in which the boundary edges are lowered by 100m. Figure 14 shows the results without uplift after 250 iterations of 5,000 years, both with and without depression routing. Depression routing prevents water loss, and therefore leads to a much deeper carving of the valleys.

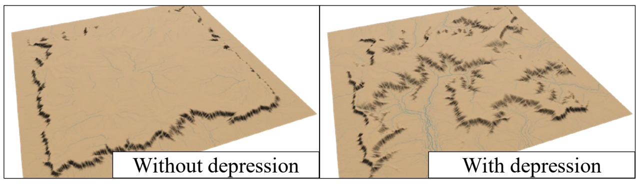


Figure 14: *Erosion carving on an escarpment, without (left) and with (right) depression routing.*

Limitations. For efficiency, our flow routing relies on parallel operations on a tree data structure, which in turn requires a single recipient per cell (Single Flow Direction, or SFD). As a consequence, we cannot handle multiple recipients (Multiple Flow Directions, or MFD), unlike some other schemes [Bar19, SPF*23]. Figure 15 illustrates the differences in appearance between rivers formed by SFD and MFD, using a CPU variant of Barnes approach [Bar19]. From a visual standpoint, MFD is more blurred and SFD more sharply defined. From an application standpoint, SFD is favoured for terrain erosion and river modeling, while MFD is more used in ecosystem simulation. Furthermore, Algorithm 7 for implicit fluvial erosion assumes a linear correlation with slope, meaning it cannot be applied study non-linear behavior in geomorphology.

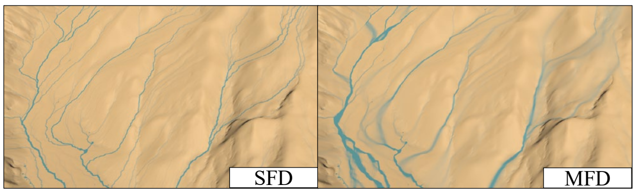


Figure 15: *The visual differences between our Single Flow Direction (SFD) algorithm and Multiple Flow Directions (MFD), which we are unable to handle.*

7.3. Performance

In this section, we compare our approach against other parallel GPU and CPU implementations of flow and depression routing.

Our performance tests were executed on a computer equipped with an Nvidia RTX A6000 GPU with 48GB of memory and used 20 cores of an Intel Xeon Gold CPU clocked at 2.10GHz with 128GB RAM.

For our test dataset, we incorporated a mix of real and synthetic terrains. We began by extracting a set of 10 real terrains from the IGN RGE ALTI Digital Elevation 1m dataset [IGN22] chosen on the basis of topographic variety. These 10 terrains were sampled at cell resolutions of 512×512, 1024×1024, 2048×2048 and 4196×4196 to support scaling experiments.

We also included 5 synthetic terrains at the same resolution levels, and each with a controlled proportion of depression coverage (at *synth*-1%, *synth*-5% and *synth*-15% levels). These were generated by erosion simulation (Section 6), followed by layering different amplitudes of uniformly distributed noise. Note that it is common practice to add this type of noise during erosion simulation to mimic natural stochastic processes.

Finally, we included a uniformly flat terrain as an extreme test case, since in this instance all cells are local minima and hence coded as depressions.

7.3.1. Spatial scaling

We show in Figure 16 the relationship between performance and terrain size, for resolutions ranging (logarithmically) from 64×64 to 8192×8192. We observe that parallelism is limited by the maximum simultaneous threads in our GPU model at around 8192×8192, where performance follows a near-linear trend.

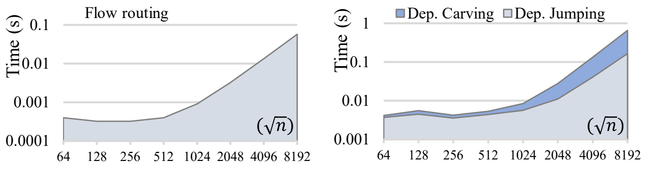


Figure 16: *Scaling experiments: (1) flow routing, (2) depression jumping, and (3) depression carving.*

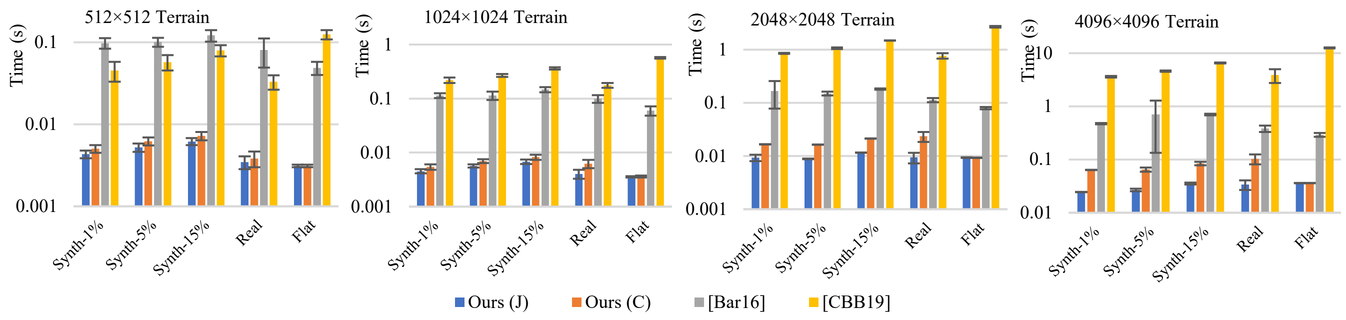


Figure 17: *Performance of depression jumping (Ours (J)) and carving (Ours (C)) against a parallel CPU implementation [Bar16] and a single-core CPU implementation [CBB19].*

7.3.2. Flow routing

Topographic variation does not have a significant impact on the performance of flow routing, so we benchmark flow routing solely on real terrains. We compare our algorithm against:

- Barnes [Bar19], who provides a GPU algorithm for flow routing based on a pre-ordering of the computation graph.
- Schott *et al.* [SPF*23], who use a GPU algorithm similar to ours, but without pointer jumping, which results in as many iterations as the node length of the longest river in the terrain ($\sim \sqrt{n}$).

Note that neither of these algorithms provides a solution for depression routing on the GPU. The results are illustrated in Figure 18, with an average speed-up of 10×, 5×, 3× and 3× against [Bar19] for each target resolution, respectively.

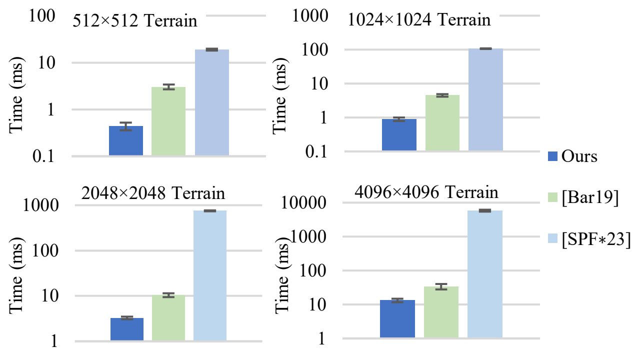


Figure 18: *Performance of our flow routing against [Bar19] and [SPF*23].*

7.3.3. Depression routing

To test the performance of depression routing we include all terrain categories (synthetic, real, and flat) from our benchmarking suite since the distribution of local minima can have a significant impact in this case. We compare two variants of our algorithm and previous work, as follows:

- **Ours (J)** Our algorithm with depression jumping.
- **Ours (C)** Our algorithm with depression carving.
- Barnes [Bar16], who proposed an algorithm optimized for CPU-based parallel or distributed computing.
- Cordonnier *et al.* [CBB19], who provided a $\mathcal{O}(n)$ solution for flow routing on the CPU with a single-core implementation.

Note that [Bar16] does not solve the depression routing problem as posed here since their goal is instead to fill depressions. Extending this to a more general solution that enables flow routing for discharge computation is far from trivial.

The comparative performance of these depression routing algorithms is shown in Figure 17. In particular, we note that our algorithms outperform previous work by one order of magnitude for terrains up to a resolution of 2048×2048. However, this gap narrows somewhat for the largest terrains, which indicates non-favorable scaling at the upper end. Also worth noting is that, with the exception of flat terrains, the comparative advantage of using depression jumping increases for larger terrains.

8. Conclusion

Algorithms for evaluating water flow over terrains are a staple of geo-analysis, with applications in computer graphics and beyond. Consequently, any improvement in their run-time performance is worth serious consideration. In this paper, we provide improved algorithms for solving both flow and depression routing problems, with, respectively, $\mathcal{O}(\log n)$ and $\mathcal{O}(\log^2 n)$ complexity for a terrain with n nodes. This is an improvement on previous methods, which usually require as many iterations as the length of the longest river ($\sim \sqrt{n}$). Most importantly, we are the first to propose a GPU solution for both flow and depression routing.

In comparative terms, our GPU implementation for depression routing outperforms an optimized parallel CPU algorithm by 34× to 52× on a 1024^2 resolution terrain, depending on the strategy for recipient correction. We also improve on previous GPU methods for flow routing [Bar19, SPF*23] by a factor of 5×.

In terms of raw performance, our GPU implementation executes in under 55ms on terrains up to 4096^2 sample resolution. This opens up new opportunities and research avenues for the future use of flow and depression routing in an interactive context. Further work on our algorithm is also required to reach more general applications, for instance by allowing for multiple recipients (Multiple Flow Directions).

Acknowledgements

This project was sponsored by the Agence Nationale de la Recherche project Invterra ANR-22-CE33-0012-01 and research and software donations from Adobe Inc. BK acknowledges funds from *WWTF* project *ICT22-055 (IVILPC)*.

References

[Ban07] BANNINGER D.: Technical Note: Water flow routing on irregular meshes. *Hydrology and Earth System Sciences 11*, 4 (2007), 1243–1247. 2, 3

[Bar16] BARNES R.: Parallel priority-flood depression filling for trillion cell digital elevation models on desktops or clusters. *Computers & Geosciences 96* (2016), 56–68. 2, 3, 12

[Bar17] BARNES R.: Parallel non-divergent flow accumulation for trillion cell digital elevation models on desktops or clusters. *Environmental modelling & software 92* (2017), 202–212. 3

[Bar19] BARNES R.: Accelerating a fluvial incision and landscape evolution model with parallelism. *Geomorphology 330* (2019), 28–39. 2, 3, 11, 12

[BCW20] BARNES R., CALLAGHAN K. L., WICKERT A. D.: Computing water flow through complex landscapes–part 2: Finding hierarchies in depressions and morphological segmentations. *Earth Surface Dynamics 8*, 2 (2020), 431–445. 3

[Ben07] BENES B.: Real-time erosion using shallow water simulation. In *VRIPHYS* (2007), pp. 43–50. 1, 2

[BLM14] BARNES R., LEHMAN C., MULLA D.: Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models. *Comp. and Geosciences 62* (2014), 117–127. 3, 4

[Bor26] BORUVKA O.: O jistém problému minimálním. *Pràce Moravské přirodovědecké společnosti 3*, 3 (1926), 37–58. 2, 6

[BS97] BRAUN J., SAMBRIDGE M.: Modelling landscape evolution on geological time scales: a new method based on irregular spatial discretization. *Basin Research 9*, 1 (1997), 27–52. 9

[BW13] BRAUN J., WILLETT S. D.: A very efficient o(n), implicit and parallel method to solve the stream power equation governing fluvial incision and landscape evolution. *Geomorphology 180-181* (2013), 170–179. 1, 2

[CBB19] CORDONNIER G., BOVY B., BRAUN J.: A versatile, linear complexity algorithm for flow routing in topographies with depressions. *Earth Surface Dynamics 7*, 2 (2019), 549–562. 2, 3, 6, 11, 12

[CBC*16] CORDONNIER G., BRAUN J., CANI M.-P., BENES B., GALIN E., PEYTAVIE A., GUÉRIN E.: Large scale terrain generation from tectonic uplift and fluvial erosion. In *CGF* (2016), vol. 35, Wiley Online Library, pp. 165–175. 1, 2, 3, 8, 10

[CHL01] CHONG K. W., HAN Y., LAM T. W.: Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of the ACM (JACM) 48*, 2 (2001), 297–323. 6

[Coa20] COATLÉVEN J.: Some multiple flow direction algorithms for overland flow on general meshes. *ESAIM: Mathematical Modelling and Numerical Analysis 54*, 6 (2020), 1917–1949. 3

[DLM11] DO H.-T., LIMET S., MELIN E.: Parallel computing flow accumulation in large digital elevation models. *Procedia Computer Science 4* (2011), 2277–2286. 3

[GGP*19] GALIN E., GUÉRIN E., PEYTAVIE A., CORDONNIER G., CANI M.-P., BENES B., GAIN J.: A review of digital terrain modeling. *Computer Graphics Forum 38*, 2 (2019), 553–577. 2, 3

[GLCC17] GAIN J., LONG H., CORDONNIER G., CANI M.-P.: Eco-Brush: Interactive Control of Visually Consistent Large-Scale Ecosystems. *Computer Graphics Forum 36*, 2 (May 2017), 63–73. 2

[GM97] GARBRECHT J., MARTZ L. W.: The assignment of drainage direction over flat surfaces in raster digital elevation models. *Journal of Hydrology 193*, 1 (1997), 204–213. 1, 2

[IGN22] IGN: Rge alti 1m [data set]. IGN, 2022. Accessed 2022-07-01. URL: `https://geoservices.ign.fr/rgealti`. 10, 11

[JBC24] JAIN A., BENES B., CORDONNIER G.: Efficient debris-flow simulation for steep terrain erosion. *ACM TOG 43*, 4 (July 2024). 10

[KBKv09] KRIŠTOF P., BENES B., KŘIVÁNEK J., ŠŤAVA O.: Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum (Proceedings of Eurographics 2009) 28*, 2 (2009), 219–228. 2

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. *ACM Siggraph Computer Graphics 23*, 3 (1989), 41–50. 2, 9

[OM84] O'CALLAGHAN J., MARK D.: The extraction of drainage networks from digital elevation data. *CVGIP 28* (1984), 323–344. 2

[PGG*24] PEYTAVIE A., GAIN J., GUÉRIN E., ARGUDO O., GALIN E.: Deadwood: Including disturbance and decay in the depiction of digital nature. *ACM Trans. Graph. 43*, 2 (feb 2024). 8

[PMG*22] PAŁUBICKI W., MAKOWSKI M., GAJDA W., HÄDRICH T., MICHELS D. L., PIRK S.: Ecoclimates: climate-response modeling of vegetation. *ACM Trans. Graph. 41*, 4 (2022). 8

[QBCP91] QUINN P., BEVEN K., CHEVALLIER P., PLANCHON O.: The prediction of hillslope flowpaths for distributed hydrological modeling using digital terrain models. *Hydrological Processes 5* (1991), 59–80. 3

[Rie98] RIEGER W.: A phenomenon-based approach to upslope contributing area and depressions in DEMs. *Hydrological Processes 12*, 6 (1998), 857–872. 7

[RKKL21] RIIHIMÄKI H., KEMPPINEN J., KOPECKÝ M., LUOTO M.: Topographic wetness index as a proxy for soil moisture: The importance of flow-routing algorithm and grid resolution. *Water Resources Research 57*, 10 (2021). 2

[SAF05] SEVILGEN F. E., ALURU S., FUTAMURA N.: Parallel algorithms for tree accumulations. *Journal of Parallel and Distributed Computing 65*, 1 (2005), 85–93. 2, 4, 5

[Sid23] SIDEFX: Houdini, 2023. URL: `www.sidefx.com`. 9

[SPF*23] SCHOTT H., PARIS A., FOURNIER L., GUÉRIN E., GALIN E.: Large-scale terrain authoring through interactive erosion simulation. *ACM Transactions on Graphics 42* (2023). 2, 3, 9, 10, 11, 12

[TGSC24] TZATHAS P., GAILLETON B., STEER P., CORDONNIER G.: Physically-based Analytical Erosion for fast Terrain Generation. *Computer Graphics Forum* (2024). 3, 4, 9

[VBHS11] VANEK J., BENES B., HEROUT A., STAVA O.: Large-scale physics-based terrain editing using adaptive tiles on the GPU. *IEEE Computer Graphics and Applications 31*, 6 (2011), 35–44. 2

[VHPN09] VINEET V., HARISH P., PATIDAR S., NARAYANAN P.: Fast minimum spanning tree for large graphs on the gpu. In *Proceedings of High Performance Graphics 2009* (2009), pp. 167–171. 2, 6

[WQZ19] WANG Y.-J., QIN C.-Z., ZHU A.-X.: Review on algorithms of dealing with depressions in grid dem. *Annals of GIS 25*, 2 (2019), 83–97. 3

[WZF18] WEI H., ZHOU G., FU S.: Efficient priority-flood depression filling in raster digital elevation models. *International Journal of Digital Earth 0*, 0 (2018), 1–13. 3

[YBG*19] YUAN X. P., BRAUN J., GUERIT L., ROUBY D., CORDONNIER G.: A new efficient method to solve the stream power law model taking into account sediment deposition. *Journal of Geophysical Research: Earth Surface 124*, 6 (2019), 1346–1365. 1, 9, 10

[ZLFS17] ZHOU G., LIU X., FU S., SUN Z.: Parallel identification and filling of depressions in raster digital elevation models. *International Journal of GIS 31*, 6 (2017), 1061–1078. 3

[ZSF16] ZHOU G., SUN Z., FU S.: An efficient variant of the Priority-Flood algorithm for filling depressions in raster digital elevation models. *Computers and Geosciences 90* (2016), 87–96. 3