Kinetic Volume-Based Persistence for 1D Terrains

Tim Ophelders[1], Willem Sonke[*2], Bettina Speckmann[†2], and Kevin Verbeek[‡2]

1 **Department of Computational Mathematics, Science and Engineering, Michigan State University, USA**
ophelder@egr.msu.edu

2 **Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands**
[w.m.sonke|b.speckmann|k.a.b.verbeek]@tue.nl

1 Introduction

The evolution of the channel network of *braided* rivers is an important topic in geomorphology. The merges and splits of braided river channels evolve over time under the influence of water pressure and sediment transport. Kleinhans *et al.* [6] recently presented algorithms to extract a (static) network of significant channels from the elevation data of a river bed; this method is already used in geomorphological studies [5, 8]. The challenge is to identify *significant* channels from the river bed, since measurement errors and small variations in the terrain generally cause a multitude of possible channels. Two channels can be considered similar if the volume of terrain between the channels is small, modelling the fact that a small volume of sediment can easily be eroded by water flow, merging the two channels into one. A network of significant channels consists of channels which are sufficiently dissimilar.

To analyze the evolution of significant channels over time we could kinetically maintain the network of significant channels computed by the method of Kleinhans *et al.* [6]. However, these networks are not stable over time and also prohibitively expensive to compute. We hence propose a simplified model which uses a *volume-simplified* terrain. Specifically, we *prune* topological features of the terrain (minima and maxima) that can be eliminated by removing only a small amount of volume. The idea is that the remaining topological features separate significant channels.

Pruning of the terrain based on the volume of the removed features resembles terrain simplification based on (height) persistence. The notion of topological persistence was introduced by Edelsbrunner *et al.* [4]. Persistence can be defined via measures other than the vertical distance between points. Carr *et al.* [3] describe a method to simplify contour trees (which capture the topological structure of a terrain) using so-called *local geometric measures*, such as (in 2D terrains) the line length of the contour, the area enclosed by the contour, or the volume of the enclosed region. This last type of persistence is exactly the one we use in this paper and we therefore refer to our simplified terrain as *volume-persistent*.

Our goal is to maintain a volume-persistent terrain over time, which is closely related to maintaining its topological structure over time, as represented, for example, by its contour tree or its split tree. Agarwal *et al.* [1] show how to maintain a 2D contour tree kinetically. They also argue that they can maintain height persistence over time. However, maintaining volume-persistence is much more challenging, because we have to detect the events that occur when a pruned part of the terrain attains a certain threshold volume. The complexity

[*] Supported by the Netherlands Organisation for Scientific Research (NWO); 639.023.208.
[†] Partially supported by the Netherlands Organisation for Scientific Research (NWO); 639.023.208.
[‡] Supported by the Netherlands Organisation for Scientific Research (NWO); 639.021.541.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

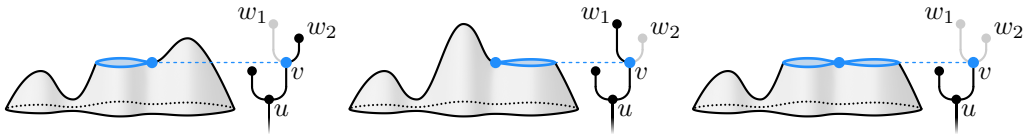


Figure 1 Pruning a terrain at the height of a saddle point v (split tree shown next to the terrain).

of the associated certificates can be high, since the volumes can be determined by linearly many vertices. Furthermore, the volume above a particular contour in the terrain is not continuous: when two contours merge the associated volumes are summed, whereas the associated height is simply the maximum of the two heights. Hence the volume-based pruning can be "stuck" at a critical point (see Fig. 1: pruning just below the saddle point requires a much higher volume-threshold than pruning exactly at the saddle point). Since these issues already manifest themselves for 1D terrains, we restrict ourselves to this setting.

Preliminaries. Let T be a 1D terrain with n vertices, of which each vertex v has a fixed x-coordinate x_v and height h_v that changes linearly over time according to a known *flight plan*: $h_v(t) = a_v\, t + b_v$. Between vertices, the height is interpolated linearly; $h(x,t)$ denotes the height of T at x on time t. At time t, the *superlevel set* of T at height h is the set of points in T with $h(x,t) \geq h$. A superlevel set may consist of several connected components. For $h = -\infty$, the superlevel set spans the entire terrain. If we continuously increase h, at certain moments topological changes happen to the superlevel set. The *split tree* S of T represents those changes: a component splitting is represented by an internal vertex, and a component disappearing is represented by a leaf [2] (see Fig. 2a). We consider S to be rooted at the vertex at $h = -\infty$, so every edge is directed upwards. Let $e = (u, v)$ be a split tree edge. Then the *parent edge* $p(e)$ of e is the incoming edge of u, and the *child edges* of e are the outgoing edges of v. The *subtree* rooted at e is the subtree rooted at v, plus e itself.

To prune the terrain we cut off a single split component at a particular height h (see Fig. 2b). We can equivalently view this operation as pruning the split tree: we identify some point on an edge of the split tree, and remove the entire subtree above it. If we prune at the height of a minimum v, then we need to specify which of the outgoing edges of v we want to prune. We define $\mathcal{A}(e,h)$ as the area of terrain that is cut off if we prune edge e at height h. Let $A \in \mathbb{R}$ be some (fixed) positive area threshold. We define the *area-persistent* terrain

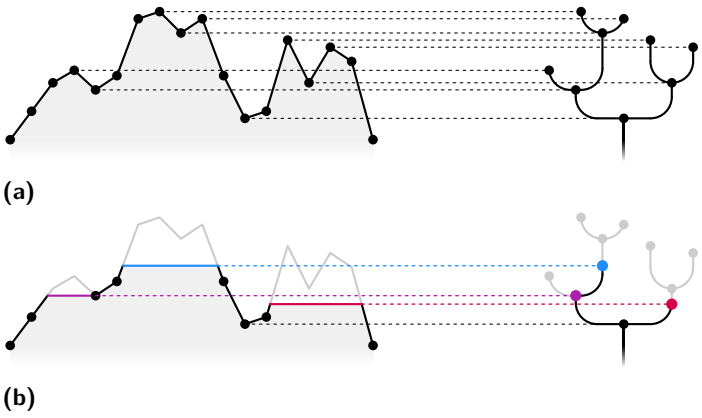


Figure 2 Area-simplifying a 1D terrain: **(a)** before, and **(b)** after.

as the terrain left after pruning all edges e in S at height h, whenever $\mathcal{A}(e,h) \leq A$. In the resulting terrain, all pieces that are cut off have area at most A.

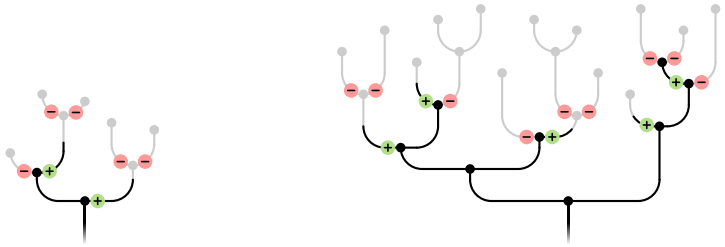
Results. We describe a KDS that maintains the pruned split tree of an area-persistent terrain under linear vertex motion. That is, for each edge $e = (u, v)$, we maintain if $\mathcal{A}(e, h_u)$ is smaller or larger than A. This KDS is compact, responsive, local, and efficient.

2 A KDS for maintaining an area-persistent 1D terrain

We adapt the KDS for 2D contour trees by Agarwal *et al.* [1] to 1D split trees. This KDS uses four types of events: shift, birth, death, and interchange events, which can all be handled by changing the split tree. We maintain the split tree S as a *link–cut* tree [7]. Next, we add *area certificates* to detect so-called *area events* when a pruning boundary moves from one split tree edge to the other.

We define $\mathcal{A}_e := \mathcal{A}(e, h_u)$, so $\mathcal{A}_e$ is the area that is removed when we prune away the subtree rooted at e from S, and we write $\mathcal{A}_e(t)$ to denote $\mathcal{A}_e$ at time t. If $\mathcal{A}_e(t) > A$, we say that e is *significant* at time t, otherwise we say that e is *insignificant*. We call a certificate $\mathcal{A}_e > A$ ($\mathcal{A}_e < A$) an *upper (lower) area certificate* for edge e (see Fig. 3a).

If an edge e is insignificant, then all edges in the subtree rooted at e are insignificant; similarly, if e is significant, then all ancestor edges of e are significant. Therefore we do not need to store area certificates for all edges in S: we store a lower area certificate for e only if $p(e)$ is significant, and an upper area certificate for e only if all children of e are insignificant (see Fig. 3b, only stored area certificates are shown). On each root-to-leaf path through S, at most two area certificates are stored: one upper and one lower certificate.



(a) **(b)**

Figure 3 Two area-persistent split trees with area certificates: **(a)** the terrain from Fig. 2b, and **(b)** a more complicated terrain. (+ and − denote upper and lower area certificates, respectively.)

Computing areas. We use an additional data structure that is based on the flight plans of the vertices and that needs to be updated only when a flight plan changes. Using this data structure, given two arbitrary terrain vertices v and w and some time t, we want to be able to compute $\int_{x_v}^{x_w} h(x,t)\,\mathrm{d}x$ in $O(\log n)$ time. If $e = (p, q)$ is a terrain edge, then

$$\begin{aligned}\int_{x_p}^{x_q} h(x,t)\,\mathrm{d}x &= \tfrac{1}{2}(x_q - x_p)(a_p t + b_p + a_q t + b_q) \\ &= \underbrace{\tfrac{1}{2}(x_q - x_p)(a_p + a_q)}_{=:\,a_e}\, t + \underbrace{\tfrac{1}{2}(x_q - x_p)(b_p + b_q)}_{=:\,b_e}.\end{aligned}$$

Then for arbitrary terrain vertices v and w,

$$\int_{x_v}^{x_w} h(x,t)\,\mathrm{d}x = \sum_{e=(p,q)} \int_{x_p}^{x_q} h(x,t)\,\mathrm{d}x = \Big(\sum_e a_e\Big)t + \sum_e b_e,$$

where the summations range over all terrain edges between v and w. To evaluate $\sum_e a_e$ and $\sum_e b_e$ efficiently, we compute a_e and b_e for each terrain edge e and store them in a balanced binary tree R, augmented with sums of subtrees, using $O(n)$ preprocessing time. We then support $O(\log n)$ time queries for the sum of a_e or b_e values for all edges e between two query vertices v and w, so we can compute $\int_{x_v}^{x_w} h(x,t)$ in $O(\log n)$ time for any v, w and t.

Detecting area events. We need to answer the following question: Given a split tree edge $e = (u, v)$ at time t_0, what is the next time t_{event} at which $\mathcal{A}_e(t_{\text{event}}) = A$? We assume without loss of generality that $e = (u, v)$ is a right-going edge in S. Let $x_{\text{ray}}(t)$ be the x-coordinate of the first point on the terrain to the right of u where $h(x,t) = h_u(t)$. That is, $x_{\text{ray}}(t)$ is the point where a horizontal ray from u to the right stabs the terrain at time t.

We first assume that we know the edge $e_{\text{event}} = (p, q)$ that $x_{\text{ray}}(t_{\text{event}})$ lies on, that is, the first edge intersected by the ray at the time that the area certificate fails (see Fig. 4a). In this case, we can compute the exact failure time t_{event} as follows. $\mathcal{A}_e(t)$ is the sum of the areas induced by the terrain edges between u and p, which are fully above the ray, and the area of the triangle between the ray and (p, q) (see Fig. 4b). This implies that $\mathcal{A}_e(t_{\text{event}}) = A$ is a quadratic equation with a closed-form solution, and can therefore be solved exactly. In other words, given e_{event}, we can compute t_{event} in $O(\log n)$ time.

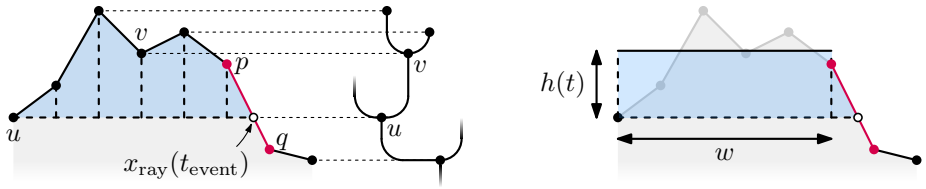


Figure 4 Computing the area $\mathcal{A}_e(t_{\text{event}})$ (shaded in blue).

Finding the stabbed chain. In the following we consider monotone *chains* of the terrain. We call the chain that contains $x_{\text{ray}}(t_0)$ the *stabbed chain* of edge $e = (u, v)$ and describe an algorithm StabbedChain that finds its lower endpoint m (see Fig. 5).

▶ **Lemma 1.** *Let $e = (u, v)$ be a right-going edge in S, and let (m, m') be the last left-going edge on the path π from the root to u. Then m is the lower endpoint of the stabbed chain of e.*

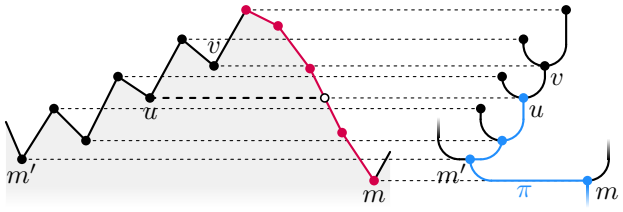


Figure 5 Finding the lower endpoint m of the stabbed chain (red) by traversing the split tree.

Since path π may have linear length we cannot traverse it to search for m. Instead we query the link–cut tree for a binary search tree containing all vertices on π (this is an Expose query). In this tree we find the rightmost vertex m' whose incoming edge is left-going in S, and return the parent m of m'. For this we augment the link–cut tree: we store for each edge whether it is left- or right-going. This results in a $O(\log n)$ running time for Stabbed-Chain.

Finding e_{event}. At current time t_0 we first use Stabbed-Chain to find vertex m. The predecessor of m in the split tree is the maximum M of the stabbed chain. We binary search between M and m for the edge that $x_{\text{ray}}(t_0)$ lies on and then find the exact value for $x_{\text{ray}}(t_0)$. Consider the point s on the terrain at x-coordinate $x_{\text{ray}}(t_0)$. We distinguish three cases based on how s moves over time. In the simplest case, s moves at the same speed as u and hence x_{ray} is constant, so we simply return the edge $x_{\text{ray}}(t_0)$ lies on. Otherwise, s may be moving upwards or downwards relative to u. If s moves upwards, x_{ray} is strictly increasing over time; if s moves downwards, x_{ray} is strictly decreasing. In the following we assume that x_{ray} is strictly increasing (see Fig. 6); the other case is symmetric.

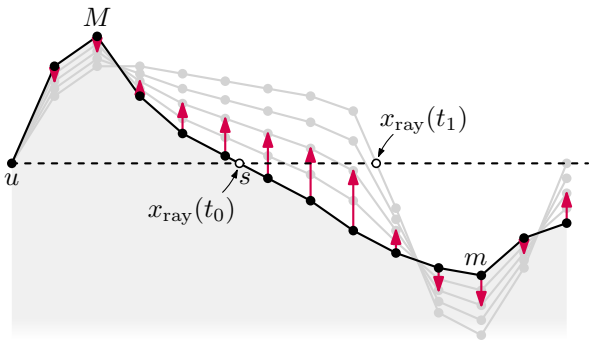


Figure 6 If s moves upwards, then x_{ray} is strictly increasing. Terrain at t_0 in black; terrain at later times in gray. The red arrows indicate how the vertices move relative to u.

We then find the time t_1 at which the next shift, birth or death event occurs involving vertices between u and m. (This can be done efficiently using a 1D range tree storing all such events, with their x-coordinate as the key and their failure time as the value.) Since therefore no events occur between t_0 and t_1, $x_{\text{ray}}(t_1)$ lies in the same chain as $x_{\text{ray}}(t_0)$. Therefore we can compute $x_{\text{ray}}(t_1)$ in the same way we computed $x_{\text{ray}}(t_0)$. Given some $x \in [x_{\text{ray}}(t_0), x_{\text{ray}}(t_1)]$, let $t_{\text{ray}}(x)$ be the time at which the point at x hits the ray. Being the inverse of x_{ray}, t_{ray} is also strictly increasing.

▶ **Lemma 2.** *The function $\mathcal{A}'_e(x) := \mathcal{A}_e(t_{ray}(x))$ is unimodal within $[x_{ray}(t_0), x_{ray}(t_1)]$: there exists an x_{fixed} such that $\mathcal{A}'_e(x)$ is strictly descending for $x \in [x_{ray}(t_0), x_{fixed})$ and strictly increasing for $x \in (x_{fixed}, x_{ray}(t_1)]$.*

Proof. We study the derivative $\mathrm{d}\mathcal{A}'_e/\mathrm{d}x$ by considering the following extension of $\mathcal{A}'_e(x)$:

$$\mathcal{A}'_e(x,t) = \int_u^x \big(h(x',t) - h_u(t)\big)\,\mathrm{d}x'.$$

$\mathcal{A}'_e(x,t)$ represents the area cut off by a horizontal ray to the right starting from vertex u, until x-coordinate x, so $\mathcal{A}'_e(x) = \mathcal{A}'_e(x, t_{\text{ray}}(x))$. We set $t = t_{\text{ray}}(x)$ and compute

$$\frac{\mathrm{d}\mathcal{A}'_e}{\mathrm{d}x} = \frac{\partial \mathcal{A}'_e}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}x} + \frac{\partial \mathcal{A}'_e}{\partial x} = \frac{\partial \mathcal{A}'_e}{\partial t}\frac{\mathrm{d}t}{\mathrm{d}x}.$$

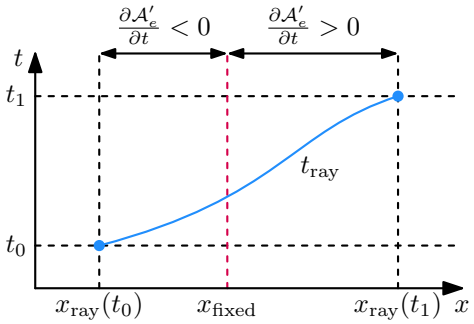


Figure 7 Sketch of the function t_{ray} drawn in the (x, t)-plane.

The last equality follows because $\partial \mathcal{A}'_e/\partial x = 0$, as for $x = x_{\text{ray}}(t)$ by definition $h(x, t) = h_u(t)$. Because t_{ray} is increasing, $\mathrm{d}t/\mathrm{d}x > 0$, so $\mathrm{d}\mathcal{A}'_e/\mathrm{d}x$ has the same sign as $\partial \mathcal{A}'_e/\partial t$.

Let x_{fixed} be the x-coordinate within $[x_{\text{ray}}(t_0), x_{\text{ray}}(t_1)]$ such that $\mathcal{A}'_e(x_{\text{fixed}}, t)$ is constant in t, if such a coordinate exists. The average terrain height on $[x_u, x_{\text{fixed}}]$ is hence constant, and all terrain points in $[x_{\text{fixed}}, x_{\text{ray}}(t_1)]$ are moving upwards relative to u. Therefore, for $x \in (x_{\text{fixed}}, x_{\text{ray}}(t_1)]$, the average terrain height on $[x_u, x]$ is growing, so $\partial \mathcal{A}'_e/\partial t > 0$ (see Fig. 7). Similarly, within $[x_{\text{ray}}(t_0), x_{\text{fixed}})$, $\partial \mathcal{A}'_e/\partial t < 0$. If no x_{fixed} exists, then $\partial \mathcal{A}'_e/\mathrm{d}t > 0$ or $\partial \mathcal{A}_e/\mathrm{d}t < 0$ throughout $[x_{\text{ray}}(t_0), x_{\text{ray}}(t_1)]$. Hence, $\mathcal{A}'_e(x)$ is unimodal. ◀

First we find x_{fixed} by binary searching within the range $[x_{\text{ray}}(t_0), x_{\text{ray}}(t_1)]$, using tree R. Then, we compute $\mathcal{A}'_e(x_{\text{ray}}(t_0))$ and $\mathcal{A}'_e(x_{\text{ray}}(t_1))$ to determine whether e_{event} lies within $[x_{\text{ray}}(t_0), x_{\text{fixed}})$ or $(x_{\text{fixed}}, x_{\text{ray}}(t_1)]$, and do a binary search (again using R) on that range to find e_{event}. The binary search takes $O(\log n)$ steps, each taking constant time, so it takes $O(\log n)$ time to compute $\mathcal{A}'_e(x)$.

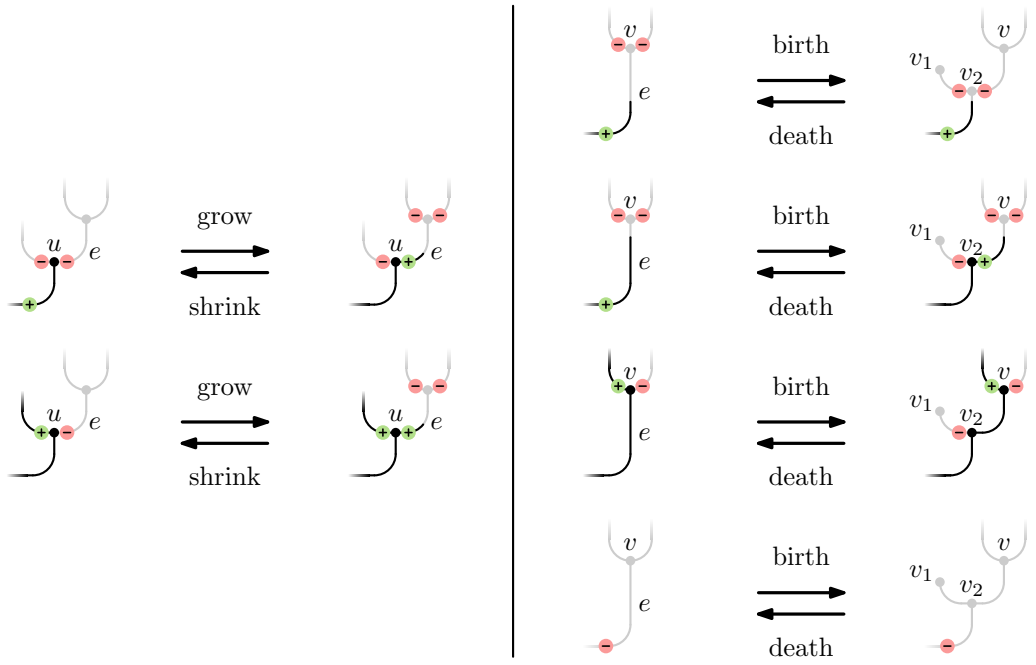


Figure 8 Handling area events *(left)* and birth / death events *(right)*.

Event handling. There are two complementary types of area events: firstly, $\mathcal{A}_e < A$ may fail (a *grow event*), and secondly, $\mathcal{A}_e > A$ may fail (a *shrink event*). Both of them can be handled by inserting and removing area certificates (see the left of Fig. 8). Shift, birth, death, and interchange events are handled like in the KDS from Agarwal *et al.*, but some additional actions are required to ensure that the area certificates are updated (see the right of Fig. 8).

Analysis. The KDS is still compact, responsive, local, and efficient. In particular, each vertex is involved in a constant number of area certificates. Indeed, a vertex is involved in the area certificate of a split tree edge $e = (u, v)$ if it lies between u and m (see Fig. 5). A chain is stabbed by only one upper area certificate, since any rays stabbing the same chain need to originate from vertices on the same root-to-leaf path in the split tree (by Lemma 1); as we store upper area certificates for an edge only if all its children are insignificant, this path can contain only one upper area certificate. Therefore, each vertex in the interior of the chain is involved in at most one upper area certificate and by similar reasoning, at most one lower area certificate. A minimum is part of two chains and hence involved in at most two upper and two lower area certificates. This implies that the locality is $O(1)$.

References

1 Pankaj Agarwal, Thomal Mølhave, Morten Revsbæk, Issam Safa, Yusu Wang, and Jungwoo Yang. Maintaining contour trees of dynamic terrains. In *Proc. 31st International Symposium on Computational Geometry (SoCG)*, pages 796–811, 2015.

2 Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 918–926, 2000.

3 Hamish Carr, Jack Snoeyink, and Michiel van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proc. 15th IEEE Visualization Conference (VIS)*, pages 497–504, 2004.

4 Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete Computational Geometry*, 28:511–533, 2002.

5 Matthew Hiatt, Willem Sonke, Elisabeth Addink, Wout van Dijk, Marc van Kreveld, Tim Ophelders, Kevin Verbeek, Joyce Vlaming, Bettina Speckmann, and Maarten Kleinhans. Geometry and topology of estuary and braided river channel networks extracted from topographic data. Abstract EP32A-08 presented at 2018 Fall Meeting, AGU, Washington, D.C., 10-14 Dec.

6 Maarten Kleinhans, Marc van Kreveld, Tim Ophelders, Willem Sonke, Bettina Speckmann, and Kevin Verbeek. Computing representative networks for braided rivers. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 48:1–48:16, 2017.

7 Daniel Sleator and Robert Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

8 Wout van Dijk, Jasper Leuven, Jana Cox, Jelmer Cleveringa, Marcel Taal, Matthew Hiatt, Willem Sonke, Kevin Verbeek, Bettina Speckmann, and Maarten Kleinhans. The effects of dredging and disposal activity on the resilience of estuary morphodynamics. Abstract EP23C-2305 presented at 2018 Fall Meeting, AGU, Washington, D.C., 10-14 Dec.