A Parallel Scale-Space Method for Critical Features Tracking on Triangulated Irregular Networks

HAOAN FENG, Computer Science, University of Maryland College Park, College Park, United States
YUNTING SONG, Geographical Sciences, University of Maryland at College Park, College Park, United States
LEILA DE FLORIANI, Geographical Science, University of Maryland College Park, College Park, United States

The scale-space method is a well-established framework that constructs a hierarchical representation of an input signal and facilitates coarse-to-fine reasoning about its features. Considering the terrain elevation function as the input signal, the scale-space method can identify and track significant topographic features across different scales. The persistence of a feature, called its *life span*, indicates its importance and enables the automatic selection of critical features for applications such as cartography, nautical charting, and land-use planning. Traditional scale-space methods rely on gridded Digital Elevation Models (DEMs), which lack the flexibility to adapt to irregular input distributions and varied terrain complexity. In contrast, Triangulated Irregular Networks (TINs) can be directly generated from irregular point clouds and naturally preserve key features. In this work, we introduce a novel scale-space analysis pipeline for TINs, addressing the challenges in extending grid-based scale-space methods to irregular meshes. Building on our prior conference version, we further extend the pipeline in several directions: (i) a scale-aware, seam-free sampling strategy (S3) for high-quality TIN construction, (ii) an improved TIN smoothing method that combines virtual neighbors and angle re-weighting with a quantitative evaluation of geometric and gradient fidelity, and (iii) a fully parallel critical point tracking algorithm that eliminates global sorting and achieves substantial GPU speedups. Comprehensive experiments demonstrate that our TIN-based pipeline achieves superior geometric and topological fidelity, improved efficiency, and stronger resolution robustness than grid-based methods, making it a scalable and accurate framework for multi-scale terrain analysis.

CCS Concepts: • **Information systems** → **Geographic information systems**; • **Computing methodologies** → **Mesh geometry models**; **Shape analysis**; **Concurrent algorithms**.

Additional Key Words and Phrases: Scale-space data analysis, Triangulated Irregular Networks, Topological methods, Cartography

1 Introduction

Scale-space methods have been widely utilized in the fields of computer vision and image processing since the pioneering work of Koenderink [17] and Witkin [36]. The scale-space framework mimics the multi-resolution property of the human visual cortex by representing objects at different scales. The level of detail (LOD) of the input signal decreases as the scale increases. Lindeberg [24] introduced a method to analyze the change of input signal across different scales by tracking the zero-crossings of the differential invariants of the signal. These zero-crossing points, named *critical points* or *features*, describe the topological structure of the input signal.

Authors' Contact Information: Haoan Feng, Computer Science, University of Maryland College Park, College Park, Maryland, United States; e-mail: hfengac@umd.edu; Yunting Song, Geographical Sciences, University of Maryland at College Park, College Park, Maryland, United States; e-mail: ytsong@umd.edu; Leila De Floriani, Geographical Science, University of Maryland College Park, College Park, Maryland, United States; e-mail: deflo@umd.edu.

ACM 2374-0361/2026/4-ART
https://doi.org/10.1145/3812549

Critical points that persist across scales are particularly essential as they are resistant to the change in scale and represent more prominent data points to convey the topological information of the input signal.

Features extracted through scale-space methods are valuable for analysis tasks in remote sensing and Geographical Information Systems (GIS) [1, 18, 20, 23]. For instance, generating nautical charts from sounding datasets heavily relies on the manual selection of soundings by human experts, who face constraints from safety requirements and cartographic criteria. To support human experts, many analysis systems aim to automatically identify important data points or provide useful auxiliary information [9, 22, 33]. Similarly, critical points can also assist in identifying topologically significant locations (e.g., *spot heights*) on a map that potentially satisfies the cartographers' requirements.

When using scale-space methods to extract critical features from terrain, gridded terrain surface models, i.e., gridded Digital Elevation Models (gridded DEMs) are usually used [30]. However, when the input point clouds are distributed irregularly, the process of interpolating them into regular grid cells causes a loss of information. Besides, the fixed cell size of gridded DEMs poses difficulty in capturing the irregularity of the terrain surface with minimal costs. For instance, a coarser grid reduces the storage cost but cannot fully represent the details in mountainous regions, while a finer grid requires a higher storage cost. Therefore, when input points are distributed irregularly, Triangulated Irregular Networks (TINs) are usually used to represent the terrain surface instead. TINs are more efficient in representing topographical information at multiple resolutions for different regions of various complexity. Moreover, when represented as a gridded DEM, the boundary information of the terrain surface may be lost or roughly approximated. Thus, in applications where boundary information is crucial, such as coastal and ocean modeling [3, 6], TINs are preferred as they can better represent line features than gridded DEMs.

In this work, we present a novel pipeline for scale-space tracking of critical points on TINs inspired by the scale-space method defined for gridded DEMs in [30]. Our experiments show that direct analysis of a TIN through the scale-space method can produce more accurate results with limited storage. We address several challenges in performing a scale-space method on TINs due to their irregular structure. Specifically, the contributions of this work are:

- **TIN construction and preprocessing:** a curvature-guided sampling strategy combined with the α-shape method [2] to generate high-quality TINs directly from raw point sets, preserving surface details under storage constraints.
- **Critical feature tracking:** an extension of the grid-based tracking method [30] to TINs, accounting for irregular vertex degrees and transitions involving k-fold saddles.
- **TIN smoothing:** a parallel iterative Gaussian filtering method adapted for TINs, enhanced with virtual neighbors and angle re-weighting to better approximate the heat diffusion process and improve fidelity across scales.
- **Parallel critical point tracking:** a fully parallel GPU implementation of edge-flip detection, vertex timeline construction, and edge transition labeling, eliminating global sorting and exploiting GPU architectures for substantial speedups.
- **Scale-space aware sampling:** a new seam-free, multi-scale curvature-driven sampling (S3) method that reduces patch boundary artifacts and improves geometric, curvature, and topological fidelity compared to patch-based FPS and other baselines.
- **Comprehensive evaluation:** quantitative and qualitative experiments comparing sampling and smoothing methods, and speedup and scalability of the parallel implementation.

The first three contributions were introduced in our earlier conference publication [13], while the latter three (parallel tracking, scale-aware sampling, and comprehensive evaluations) are new to this extended journal version.

The remainder of the paper is organized as follows. Section 2 provides background information and the problem setup, followed by Section 3, which reviews previous work on scale space. In Section 4, we review the discrete grid-based scale-space framework for gridded terrains proposed in [30, 31]. Section 5 presents our pipeline for high-quality scale-space aware TIN generation and scale-space critical point tracking method on TINs, in which two lemmas are formalized to support the parallel adaptation to GPU kernels. In Section 7, experimental results are presented. Finally, in Section 8, we discuss a potential application to bathymetric data and summarize key innovative points and future development of our pipeline.

2 Background

Triangulated Irregular Networks (TINs) are defined based on discrete point sets in the plane, where each data point has an elevation associated with it. A TIN consists of the set of vertices $\mathcal{V}$ and connectivity information C describing how the vertices are connected through the triangle mesh. Each vertex v in a TIN consists of a *Point* $p \in \mathbb{R}^2$ representing the xy-coordinate, and an elevation value. Besides vertices, triangles $\mathcal{T}$ are the primary components of a TIN, from which the connectivity information C can be derived.

Critical features, such as peaks, saddles, and pits, on a TIN can be extracted using Banchoff's theory [4], which defines critical points for polyhedral surfaces and is a pillar of Piece-wise Linear (PL) Morse theory [10]. In this paper, we use the term *critical point* to denote a TIN vertex whose PL Morse type is *non-regular*, i.e., a *maximum*, *minimum*, or *saddle* vertex (including k-fold saddles with $k > 1$). Accordingly, peaks and pits correspond to maxima and minima, respectively. Following [4], each vertex v in a TIN Θ is classified into one of five categories: *regular*, *maximum*, *minimum*, *saddle*, and *k-fold saddle* (with $k > 1$), as shown in Figure 1. The classification is done through a comparison between elevations at v and its adjacent vertices $[v_1, v_2, \ldots, v_m]$ listed clockwise. To this aim, the *vertex signature* is defined as the sequence of Boolean values $[v > v_1, v > v_2, \ldots, v > v_m]$. The head and tail of this sequence are considered neighbors in this list. As annotated in shadow in Figure 1, consecutive Booleans with the same value are grouped into *higher/lower components*, denoting neighboring vertices belonging to them having higher/lower elevations than v does. For instance, if the vertex signature of v is [*true, false, true, false, true*], the number of higher/lower components k_{higher}/k_{lower} equals 2 since the last and the first vertices that are *true* are adjacent and belong to the same component. It helps determine the point type of v. In general, a vertex v is a *maximum/minimum* when $k_{higher} = 0/k_{lower} = 0$. And, a k-fold saddle has $k_{higher} = k_{lower} = k + 1$. When $k = 1$, v is considered of type *simple saddle*.

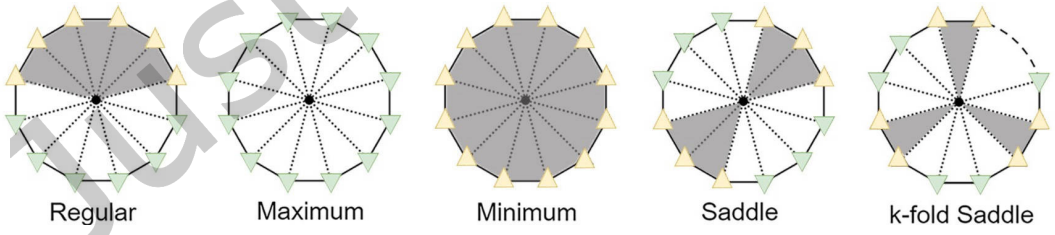


Fig. 1. Critical point types by *vertex signature*. For each neighbor vertex, ▲ indicates its elevation is higher than the center vertex; otherwise, ▼ is drawn. Derived from [10].

The Poincaré-Hopf index theorem [26] specifies that the numbers of maxima, saddles, and minima on a polyhedral surface (and thus on a TIN) follow an invariant relation:

$$N_{max} + N_{min} - N_{saddle} = 2 \tag{1}$$

In a scale space, this invariant property puts a constraint on the number of critical points and their interactions on a polyhedral surface at any scale. It helps to understand how critical points interact and transition, which makes the tracking of critical points across scales possible.

3 Related Work

Scale-space methods were first systematically introduced by Koenderink [17] and Witkin [36], and have since been extensively studied in various fields such as computer vision, image processing, and signal analysis. A Scale space is generally defined as a single-parameter family $f(s)$ of a signal f, where the scale s is the only parameter. Imitating the human visual system, the captured visual signal contains various features that become prominent at different scales. Invariant features are considered more important in the visual recognition process as the scale progresses from fine to coarse. Scale-space methods process the input signal f iteratively using filters that satisfy the heat-diffusion model, such as the Gaussian kernel. As the scale increases, the kernel size σ increases correspondingly. This process progressively eliminates finer details of the signal while retaining a clue from the coarse features to some finer details that have been removed, leaving only the coarse features at each scale layer. By sharing the same idea with the *Deep Structure* reasoning proposed by Lindeberg [24], the topological importance of critical points of the function f can be obtained by tracking their interactions and analyzing their persistence across scales.

On gridded DEMs, critical point identification and cross-scale correspondence matching are well explored. Lin et al. [23] utilized scale space to filter out unnecessary finer details for ridge line detection and merge the ridge line scratches across scales for improved results. Li et al. [22] utilized topological features from bathymetric sounding data to aid in the production of nautical charts. They identified feature points on each scale and used a proximity-based matching approach between consecutive scales, known as *neighbor searching*, to find critical feature points that exist across multiple scales. However, this matching approach can be error-prone, especially when the scale is rough and critical point movement is substantial [30]. To improve the neighbor searching of critical points' correspondence, Reininghaus et al. [28] considered the underlying gradient of the scalar function. This change improves the robustness to noise and the matching rate of the neighbor searching. However, there still exists uncertainty in the matching of maxima and minima at rough scales.

Rocca et al. [30, 31] proposed a more robust method for tracking the transformation of critical points for gridded DEMs. Their method defines a virtually continuous scale space where the movements and interactions of critical points across scales can be explicitly tracked. However, this method works only for gridded datasets. A local regular triangle mesh is generated from the grid by splitting each grid cell into two through a diagonal to identify the critical points. This choice of the diagonal leads to different critical point labeling results as illustrated in Figure 2.

In this paper, we extend the *grid-based method* proposed by Rocca et al. [30] to TINs, which are defined by irregular triangle meshes. A discrete scale space based on a TIN is constructed for critical features tracking from raw point clouds. This extension is not trivial due to the different spatial subdivisions used by TINs and gridded DEMs and to the irregularity of TINs.

4 A discrete scale-space method on regular triangle meshes

In [30], a discrete scale space Φ on a gridded DEM is constructed by iteratively processing the input gridded DEM f with a Gaussian smoothing operator g, i.e., $\Phi = \{f, g(f), g(g(f)), ...\}$. They convert the input gridded DEM into a regular triangle mesh by splitting each grid cell into two triangles through one diagonal, as illustrated in Figure 2. Each element of the scale space is a regular triangle mesh with the same vertices and triangles as the others, except that the vertex elevations are different, representing the input f at different scales. From fine to coarse, the elements in Φ are denoted as layers at different scales s_i for $i \in \mathbb{N}$, i.e., $\Phi = \{s_0, s_1, s_2, ...\}$. The discrete

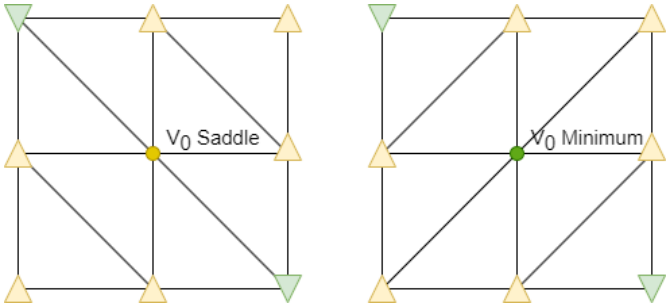


Fig. 2. Preset edge connection of a gridded DEM affects critical point types. Vertices with ▲/▼ markers have elevations higher/lower than v_0. For the type of v_0, the left model implies *Saddle*, while the right model implies *Minimum*.

scale space Φ is defined as a piecewise-linear space based on these layers. Thus, the intermediate scale space between two consecutive layers s_i and s_{i+1} is obtained by linear interpolation. Thereby, the intermediate scale $s_{i+\delta}$ can be calculated as

$$s_{i+\delta} = \delta s_{i+1} + (1-\delta)s_i \tag{2}$$

where $0 < \delta < 1$. From this equation, the elevation of each vertex v is determined in the scale space as $s_t(v)$, where t represents the scale of the input f, also known as *timestamp* in this context.

As detailed in Section 2, the point type of a vertex v_m depends only on its relative elevation relationship to its adjacent vertices in the mesh. Thus, the type of v_m can only change when its relative elevation to one of its neighbors changes. Suppose this neighbor vertex is v_n and this change happens at timestamp t, then these two vertices have the same elevation value at timestamp t, i.e., $s_t(v_m) = s_t(v_n)$. Formally, we have:

$$\operatorname{sgn}\left(s_{t-\epsilon}(v_m) - s_{t-\epsilon}(v_n)\right) = -\operatorname{sgn}\left(s_{t+\epsilon}(v_m) - s_{t+\epsilon}(v_n)\right) \tag{3}$$

where ϵ represents a small change in the timestamp. The values at the extremes of the edge between v_m and v_n swap. We call this phenomenon an *edge flip* at timestamp t, where $t \in [i, i+1)$. Following the piecewise-linear assumption of this discrete scale space, this timestamp $t_{\text{edge flip}}$ is calculated as:

$$t_{\text{edge flip}} = i + \frac{s_i(v_m) - s_i(v_n)}{s_{i+1}(v_n) - s_i(v_n) + s_i(v_m) - s_{i+1}(v_m)} \tag{4}$$

Since the type of a vertex changes only during an edge flip, tracking the critical points in the discrete scale space Φ is equivalent to processing edge-flip events in *chronological order*.

To better illustrate this process, we consider the critical point transition of a 1D signal as an example (see Figure 3). Vertex v_m is a maximum at scale s_i, and it becomes a regular point at scale s_{i+1} as one of its adjacent vertices, v_n, now has a higher elevation value than v_m. Meanwhile, vertex v_n becomes a maximum at scale s_{i+1} since it now has a higher elevation value than all adjacent vertices. These changes happen because edge $\overline{v_m v_n}$ flips at timestamp $i+\delta$ and the relative elevation between v_m and v_n is changed, as formulated in Equation 3. Thus, an edge flip happens at timestamp $i+\delta$, making a maximum move from vertex v_m to v_n.

In the *grid-based method* by [30], the transition of critical points is limited to preset cases because the edge connection is fixed and each internal vertex of the mesh has six neighbors. However, on a TIN, the number of vertex neighbors is not constant, and the existence of k-fold saddles makes the critical point transition much more complex than the preset cases in the grid-based method.

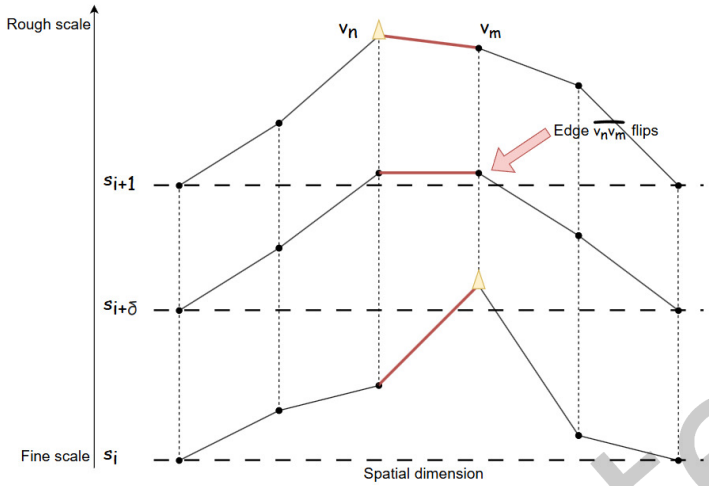


Fig. 3. A 1D illustration of maximum transition at the edge-flip event. The horizontal axis represents the spatial dimension, while the vertical axis represents the scale, progressing from fine (bottom) to rough (top). At scale s_i, vertex v_m is a maximum (▲); as the scale increases to s_{i+1}, the maximum transitions to the adjacent vertex v_n. The red arrow marks the edge-flip timestamp $s_{i+\delta}$, the intermediate scale at which the edge $\overline{v_m v_n}$ becomes flat and the relative elevation between v_m and v_n is reversed.

5 Methodology

We propose a TIN-based pipeline for feature tracking through a scale-space method. The proposed pipeline takes raw point cloud data as input and outputs critical points with a quantitative measure of their importance. Point cloud data are obtained from various sources such as Light Detection and Ranging (LiDAR) scans of landscape terrain [29]. The pipeline is composed of three consecutive stages: (I) input point cloud preprocessing (see Section 5.1), (II) TIN generation and encoding (see Section 5.2), and (III) scale-space analysis on a TIN (see Section 5.3). To further improve the efficiency of the pipeline, we also design and implement a fully parallel algorithm for critical point tracking on a TIN (see Section 6).

5.1 Point cloud preprocessing

In the scale-space analysis, critical points move and interact with each other through the evolution from fine to rough scales. In the discrete setting, the trajectory of a critical point is approximated by a continuous path of TIN edges. Thus, the quality of the TIN directly affects the accuracy of the critical point tracking result. However, terrain point clouds are irregularly distributed and contain outliers, leading to erroneous TIN smoothing results and a waste of computational resources. It is necessary to eliminate outliers and prevent data point overlap. One widely used method to down-sample a point cloud is Furthest Point Sampling (FPS) [11]. FPS starts from an initial seed point and iteratively selects the point that has the maximum distance to the current sampled set. By always choosing the farthest point, the algorithm spreads samples as evenly as possible over the surface and maximizes pairwise coverage. As a result, the selected vertices tend to be approximately uniformly distributed across the terrain. However, such near-uniform density does not adapt to spatially varying topological complexity: complex landscapes with rich local features require more vertices for accurate representation, whereas relatively flat regions can be described with fewer samples.

To address this issue, we propose a patch-based extension of the FPS method (PFPS), which divides the terrain into equal-sized regions (*patches*) and applies the FPS sampling method to sample varying numbers of points from different regions of the terrain surface based on the per-patch estimated curvature (see Algorithm 1). Each patch's curvature is estimated using eigenvalues of the covariance matrix of the point cloud. This value represents the local terrain complexity, and thus, a patch with a small curvature tends to be a flat region. As a result, in Figure 4(b), patches belonging to the flat suburban area in the terrain dataset require fewer points to be represented than mountainous regions do.

Algorithm 1: Patch-based Furthest Point Sampling

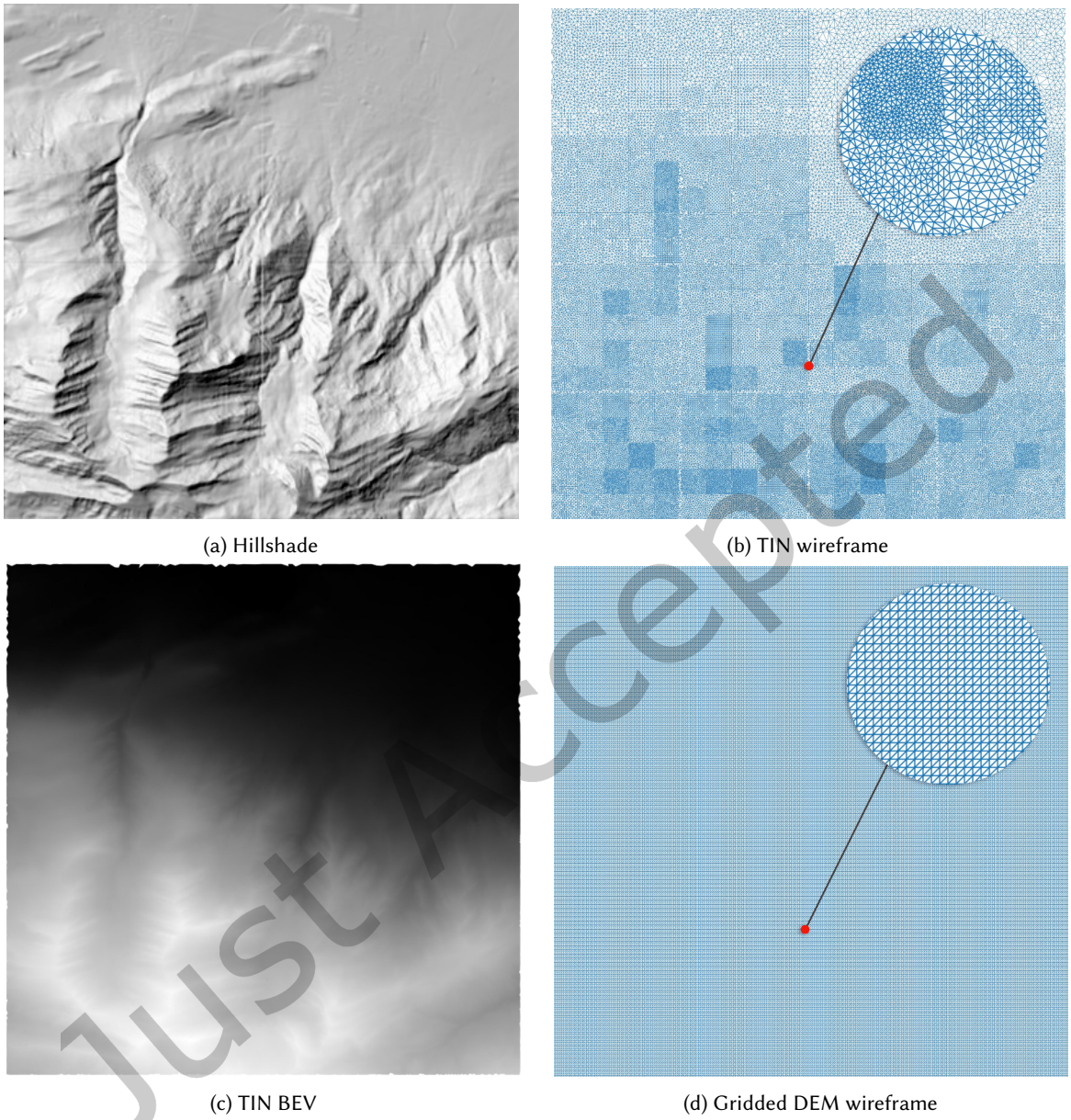
Input: Point cloud pcd, patch size s, percentage of points to keep ρ
Output: Down-sampled point cloud pcd_{down}
$pcd_{down} \leftarrow \emptyset$
// Divide the point cloud into patches over xy-plane
$patches \leftarrow subdivide(pcd, s)$
foreach *patch in patches* **do**
  // Estimate the local curvature via the covariance matrix
  $covmat \leftarrow Covariance\ Matrix\ (patch)$
  $\lambda_1, \lambda_2, \lambda_3 \leftarrow Eigenvalues\ (covmat)$ // $\lambda_1 \geq \lambda_2 \geq \lambda_3$
  $patch_{curv} \leftarrow \lambda_3/(\lambda_1 + \lambda_2 + \lambda_3)$
end
// Normalize the estimated curvatures as sampling weights
$patch_w \leftarrow patch_{curv}/\sum_{patches} patch_{curv}$
foreach *patch in patches* **do**
  $pcd_{patch} \leftarrow FPS\ (patch, \rho \times patch_w)$
  $pcd_{down} \leftarrow pcd_{down} \cup pcd_{patch}$
end
return pcd_{down}

While effective in principle, PFPS has three notable limitations: (i) it remains computationally intensive because local curvature is repeatedly estimated on irregular neighborhoods; (ii) curvature estimates from patches are noisy especially when input points are unevenly distributed; and (iii) independent sampling across patches can introduce visible seams at patch boundaries, leading to artifacts in the generated TIN. Moreover, considering the transition of critical points in the scale space, sampled points need to take the terrain surface roughness at different scales into account to keep accurate tracking of critical point interactions.

To overcome these shortcomings, we extend PFPS to a **Scale-aware and Seam-free Sampling method (S3)** (see Algorithm 2). Instead of estimating terrain complexity from a raw point cloud, this method begins by a high-resolution rasterization, which provides a uniform domain for reliable curvature computation. Laplacian curvature estimation is performed on the raster. Following the scale-space convention, curvature maps at different scales χ_s are then derived from a Gaussian pyramid of the raster. The responses are aggregated into a stable *scale-space curvature map* χ' by exponential moving average (EMA), i.e.,

$$\chi'_{s+1} = \gamma * \chi_{s+1} + (1 - \gamma) * \chi'_s,$$

where γ is a decay factor empirically set to 0.7. We selected this value by evaluating both the quantitative reconstruction accuracy of the generated TINs and their visual quality. Specifically, we observed that smaller values tend to over-concentrate vertices in highly curved regions and create overly sparse sampling in flat areas,



(a) Hillshade (b) TIN wireframe

(c) TIN BEV (d) Gridded DEM wireframe

Fig. 4. PFPS method result of Reichenburg suburban region. (a) is the hillshade of the Reichenburg Suburban region, and (c) is the Bird's Eye View (BEV) of the TIN generated. (b) and (d) are wireframe plots of the TIN and gridded DEM.

while larger values oversmooth curvature variations and reduce adaptivity. The chosen value provides a balanced trade-off between feature sensitivity and spatial coverage. Based on this map, each patch is assigned a target number of samples through a normalized allocation rule (with preset lower bounds), ensuring that both flat and

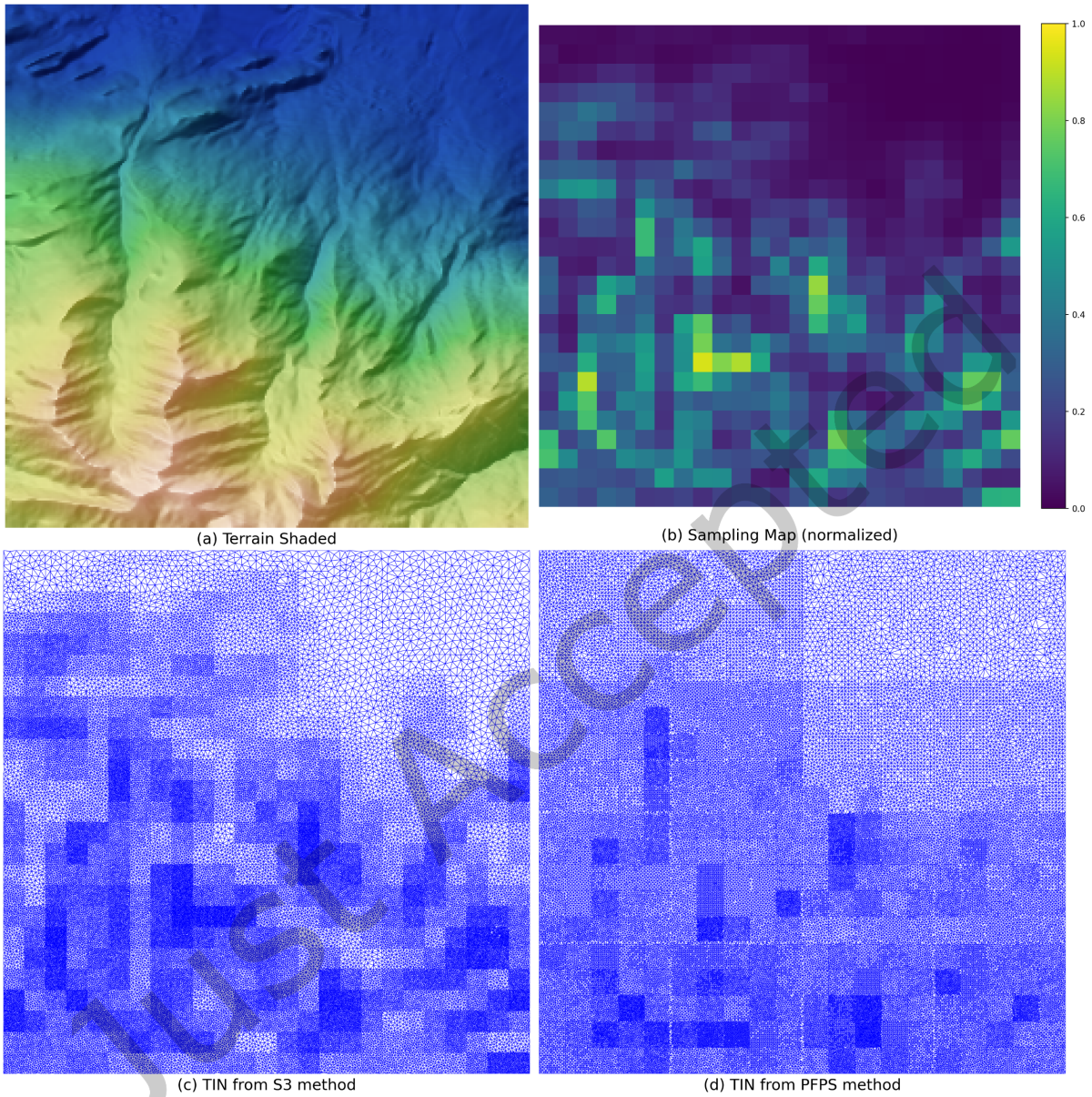


Fig. 5. Comparison of the proposed S3 method (c) with the original PFPS method (d). The S3 method demonstrates improved sampling density according to the sampling map (a) and reduced artifacts on patch boundaries.

rugged regions are represented appropriately. To avoid seams at patch boundaries, instead of sampling the raw point cloud independently, each patch is sampled, given the boundary points already sampled in neighboring patches as constraints. This is achieved by a *constrained FPS* procedure, which iteratively selects the farthest point from the union of already selected points and candidate points in the current patch (*neighbor-aware FPS*

with boundary sharing). As shown in Figure 5, the final output of our proposed S3 method is a point set with a density that reflects terrain complexity at multiple scales, while being free of seams and artifacts.

Algorithm 2: Scale-aware Seam-free Sampling (S3)

Input: Point cloud pcd, raster resolution r, patch size p, percentage of points to keep ρ
Output: Down-sampled point cloud pcd_{down}
$raster \leftarrow Rasterize(pcd, r)$
// Multi-scale curvature estimation
$curvmap \leftarrow EMA(\chi_0)$
for $s \leftarrow 1$ **to** S **do**
  $\chi_s \leftarrow LaplacianCurvature(raster)$
  $curvmap \leftarrow EMA(curvmap, \chi_s)$
  $raster \leftarrow GaussianSmooth(raster)$
end
// Allocate sampling budget per patch
$patches \leftarrow subdivide(raster, p)$
foreach *patch in patches* **do**
  $budget(patch) \leftarrow Normalize(curvmap(patch)) \times \rho$
end
// Neighbor-aware FPS with boundary sharing
$pcd_{down} \leftarrow \emptyset$
foreach *patch in patches* **do**
  $neighbors \leftarrow GetBoundaryPoints(patch, pcd_{down})$
  $pcd_{patch} \leftarrow FPS(patch, budget(patch), share = neighbors)$
  $pcd_{down} \leftarrow pcd_{down} \cup pcd_{patch}$
end
return pcd_{down}

The result is a *scale-aware, seam-free* down-sampled point set that balances geometric fidelity with sampling efficiency. Compared to PFPS, our method yields more accurate curvature preservation, reduced computation time, and visually consistent TINs without boundary artifacts. A quantitative evaluation is reported in Section 7.

Complexity analysis. Let N denote the number of input points and $K = \rho N$ the number of sampled points. Both PFPS and S3 involve several preprocessing operations, including curvature estimation (either from point-cloud patches or from rasterized data) and budget allocation. These steps are linear or nearly linear in N in practice and are not the dominant computational cost. The overall complexity of both methods is primarily governed by the FPS-based sampling stage, which requires maintaining distances between selected and candidate points and has approximately $O(NK)$ cost in the worst case. Therefore, PFPS and S3 have comparable asymptotic complexity. The main difference lies not in asymptotic complexity but in stability and sampling quality: S3 replaces patch-wise point-based curvature estimation with a structured raster-based multi-scale analysis, improving robustness and seam consistency with only modest preprocessing overhead.

5.2 TIN generation and encoding

As introduced in Section 2, a TIN consists of a 2D triangle mesh and a scalar function (elevation) defined on it. The underlying triangle mesh of a TIN is generated through a Delaunay triangulation [14] of the preprocessed point cloud. In our pipeline, the transitions of critical points are tracked along edges of the TIN, as detailed in

Section 5.3.2. Thus, sliver triangles may lead to erroneous results. For example, in Figure 6, the long edges on the boundary of TIN should be removed to avoid critical points accidentally transiting through them.

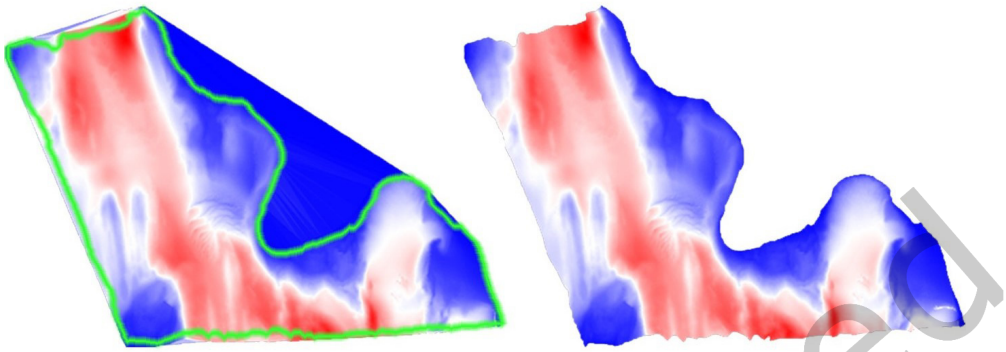


Fig. 6. Application of the α-shape method to remove long boundary triangles. The left figure shows the Delaunay triangulation of the input points. The right figure shows the resulting TIN after applying the α-shape filtering, where elongated border triangles are removed. The green polyline overlaid on the left figure represents the boundary of the filtered TIN (right), indicating which regions are preserved and which are discarded after applying the α-shape method.

To address this issue, we apply the *α-shape* method [2] to the underlying triangle mesh of the TIN, which defines a filter criterion to remove triangles with circumradii larger than a preset threshold α. The α-shape method removes long triangles on the border of the TIN, which also reduces the time and memory costs for the following scale-space analysis. Furthermore, this step is especially beneficial for datasets with irregular boundaries, like the bathymetry dataset shown in Figure 6.

One challenge that hinders the usage of TINs for large-scale terrain analysis is the high memory and time costs of processing TINs. In this work, we use a compact data representation for TINs, the Terrain trees [12], to encode TINs and to perform analysis operations. Terrain trees surpass commonly utilized structures like the Indexed data structure with Adjacencies (IA data structure) regarding memory usage and local neighbor extraction performance [12]. Leveraging Terrain trees, we are able to process with less memory and latency. Moreover, the spatial decomposition used by Terrain trees also allows for parallel processing using OpenMP [7], further enhancing the computation speed and efficiency.

5.3 Virtually continuous scale-space analysis on a TIN

Inspired by virtually continuous scale-space [31], we propose a discrete scale-space method based on TINs, which consists of three parts: the construction of scale space from the generated TIN by iterative TIN smoothing (see Section 5.3.1), the tracking of critical points across scales (see Section 5.3.2), and life span computation of critical points (see Section 5.3.3). Following previous work, critical points are tracked across scales in time/scale order, which allows for a simplified implementation (via *bookkeeping vertex-vertex relationship*) but limits its efficiency and scalability. The entire scale-space analysis is parallelized to take advantage of modern GPU architecture (see Section 6).

5.3.1 Iterative TIN smoothing. As mentioned in Section 4, the scale space of a gridded DEM is generated through a sequence of Gaussian smoothing operations. Gaussian smoothing [15] is well-defined in the image domain and efficiently achieved through convolution operations. The Gaussian smoothing kernel size, denoted as σ, increases as the scale increases. By iteratively processing the smoothed layer with a larger Gaussian kernel, the scale space

is established by stacking these smoothing results. Heuristically, the kernel size σ_i for the i-th layer of the scale space is set as $\sigma_i^2 = 2^i$. Applying smoothing to a gridded DEM is straightforward, with the convolution matrices' size linearly increasing with the kernel size. For a TIN, however, this process requires more adaptation to account for the irregular spatial decomposition inherent in a TIN.

To bridge this gap, we introduce an iterative approximation of the Gaussian smoothing process on a TIN Θ using a small, fixed kernel size, leveraging the *semi-group property* of the Gaussian kernel [27]. This property states that consecutively smoothing operations with two kernels of sizes σ_1 and σ_2 are equivalent to a single smoothing operation with a kernel of size $\sqrt{\sigma_1^2 + \sigma_2^2}$. By presetting a small kernel size $\sigma_{\texttt{small}}$, we limit the local influence region of a single smoothing operation. The small kernel size leads to more iterations and longer computation time. Therefore, we utilize parallel computing on a GPU to accelerate the smoothing process. The smoothing process can be regarded as the weighted average of adjacent vertices. It can be calculated through a matrix-vector multiplication

$$\vec{f}^{(t)} = M_w^{(t)} \vec{f}^{(t-1)} \tag{5}$$

where the vector $\vec{f}^{(t)}$ denotes the scalar function values at the vertices of the TIN after the t-th smoothing step. The matrix $M_w^{(t)}$ represents the transition operator applied at step t. It is a $V \times V$ matrix, where V is the number of vertices in Θ, and the entry (i, j) encodes the influence weight of vertex v_j on vertex v_i (nonzero only if $v_j \in Neighbor(i)$).

To approximate Gaussian smoothing with an increasingly larger effective kernel, we repeatedly apply the same small-kernel operator. Specifically, let M_w denote the weight matrix constructed using the fixed kernel size $\sigma_{\texttt{small}}$. Then the t-step smoothing operator is given by the matrix power $M_w^{(t)} = (M_w)^{\chi}$, where χ depends on the equivalent kernel size. This follows from the semi-group property of the Gaussian kernel, where repeated small-scale smoothing approximates smoothing with a larger kernel.

To fit within GPU memory constraints, M_w is stored in sparse format. With the fixed small kernel size $\sigma_{\texttt{small}}$, the one-step weight matrix M_w is computed as follows:

$$M_w[i, j] = \begin{cases} \frac{1}{Z} \exp\left(-d(i, j)^2 / \left(2\sigma_{\texttt{small}}^2\right)\right) & \text{if } j \in Neighbor(i) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

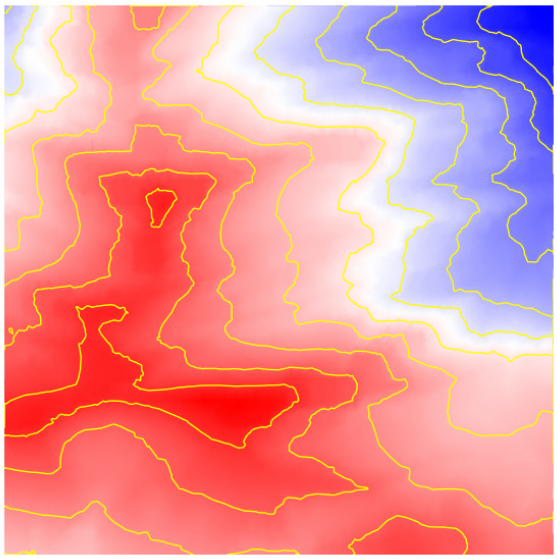
where $d(i, j)$ is the distance between v_i and v_j on the xy-plane, and Z is the normalization factor that ensures the total weight of vertex v_i sums to one. The smoothing process is then executed by performing sparse matrix-vector multiplication on GPUs as:

$$\vec{L}_{t+1} = sparse\_matvec(M_w^{(t)}, \vec{L}_t) \tag{7}$$

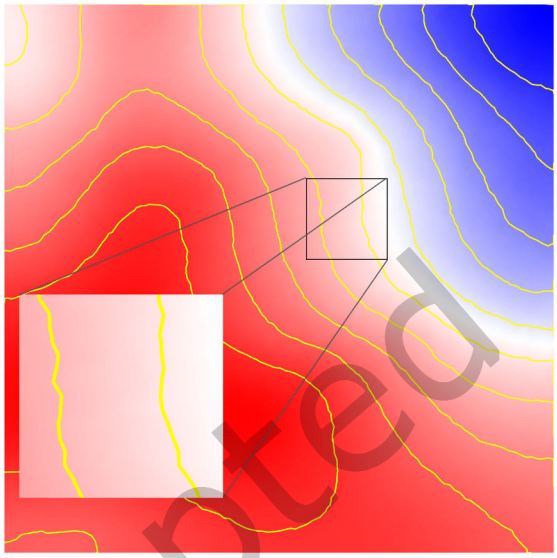
where L_t represents the scalar function value of the t-th layer of the scale space in vector form and $M_w^{(t)}$ is the transition matrix.

Differently from regular meshes, vertices adjacent to the same vertex v in a TIN have different distances from v. The weight of an adjacent vertex diminishes significantly when it is located farther away than others, due to the application of an inverse exponential function. This phenomenon becomes more pronounced when iterations are performed with a smaller kernel size, leading to a build-up of minor errors. Figure 7(a) shows the TIN smoothing result when directly re-weighting adjacent vertices with an equivalent kernel size. It is noteworthy that the TIN smoothing process fails to produce a smooth surface, which results in contour lines appearing disjointed rather than continuous.

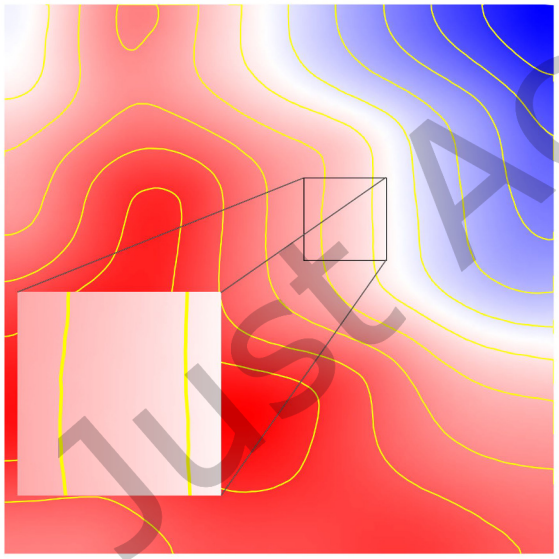
To rectify this, we introduce *virtual neighbors* during the smoothing process. When the distance between a vertex v and its adjacent vertex v_i exceeds a specified threshold τ, a virtual vertex is added to an edge $\overline{vv_i}$. As illustrated in Figure 8, since the distance $d(v, v_1)$ between v and v_1 is greater than τ, a virtual neighbor v_1' is projected onto edge $\overline{vv_1}$ at distance τ to v. The interpolated elevation value at this point, z_1', is calculated as



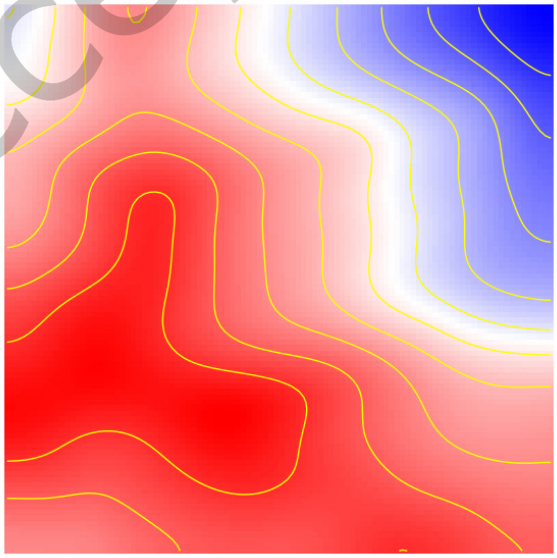
(a) TIN smoothing w.o. virtual neighbors and angle re-weighting



(b) TIN smoothing w.o. angle re-weighting



(c) TIN smoothing



(d) Gridded DEM smoothing

Fig. 7. Comparison of different smoothing results. Figure (a,b,c) progressively applies virtual neighbors and angle re-weighting techniques for the TIN smoothing. Figure (d) shows the reference image smoothing result of gridded DEM.

$\tau z_1 + (1-\tau)z$. The same principle applies to v' as a virtual neighbor to v_1. The weights of virtual neighbors can be calculated in a way similar to Equation 6, the only difference being that the interpolated values of virtual neighbors need an extra update at the end of each smoothing iteration.

Another factor influencing the smoothing result is the spatial distribution of adjacent vertices. If neighboring vertices are unevenly distributed around a central vertex, the Gaussian weights alone may bias the smoothing toward regions with higher point density, leading to distorted results, as illustrated in Figure 7(b). To mitigate this effect, we introduce an *angle-based re-weighting* scheme. For a vertex v_i, let its adjacent vertices be ordered counterclockwise in the xy-plane. For each neighbor v_j, we compute the angle θ_j (measured in degrees) at v_i between the two edges connecting v_i to the neighboring vertices immediately before and after v_j in circular order. This angle represents the portion of the local neighborhood "covered" by v_j. Rather than using θ_j directly, we use its normalized form $\theta_j/360$, which approximates the expected proportional contribution of v_j during smoothing. When neighbors are uniformly distributed around v_i, each θ_j is approximately equal and the normalized angles are close to $1/\deg(v_i)$, leading to nearly uniform angular contributions. If neighbors cluster in a particular direction, their corresponding θ_j values become significantly smaller, reducing their influence, while isolated neighbors covering larger sectors receive larger angular weights. The re-weighted matrix M'_w is therefore computed as:

$$M'_w[i,j] = \frac{1}{Z'}\left(\frac{\theta_j}{360}\right)\exp\left(-\frac{d(i,j)^2}{2\sigma^2_{\texttt{small}}}\right), \tag{8}$$

where Z' is a normalization factor ensuring $\sum_j M'_w[i,j] = 1$. As shown in Figure 7(c), incorporating this angle-based re-weighting alleviates density-induced bias and produces a smoothing result that more closely resembles image-based Gaussian smoothing (Figure 7(d)) and the expected terrain evolution.

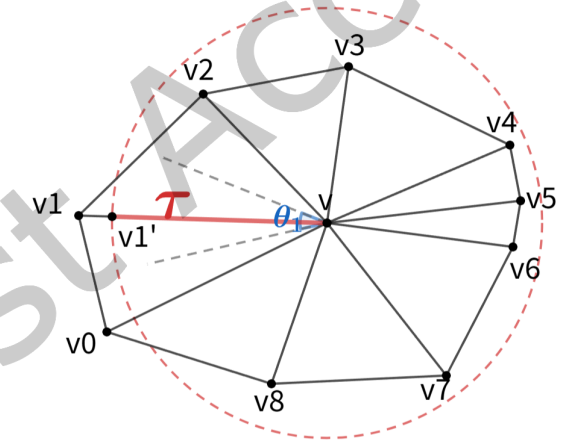


Fig. 8. Virtual neighbors and angle re-weighting. For neighboring vertex v_1, a virtual neighbor v'_1 is projected and θ_1 is its angle re-weighting coefficient.

5.3.2 Critical points tracking. Critical point tracking is achieved by processing the critical point transitions (equivalently, edge-flip events) between two consecutive scales in chronological order. As described in Section 2, the type of a vertex v_i is defined by its vertex signature at the timestamp t, denoted as $\eta_{i,t}$. For an edge $\overline{v_n v_m}$ flipping at the timestamp t, the critical point types at the timestamp before and after edge flipping are determined.

This critical point transition can be written as:

$$(\eta_{n,t-\delta}, \eta_{m,t-\delta}) \rightarrow (\eta_{n,t+\delta}, \eta_{m,t+\delta}) \tag{9}$$

where $\delta > 0$ represents a small change in the timestamp.

In the grid-based method by [30], the types of transitions of critical points are limited on regular triangle meshes with a preset edge connection. There are no k-fold saddles (with $k > 2$) since each vertex of a regular mesh has exactly six adjacent vertices. A 2-fold saddle is treated as two simple saddles sharing the same vertex. Therefore, the transition is simplified to 32 preset cases in a transition table. However, on a TIN, the number of vertex neighbors is not constant and the existence of k-fold saddles (with $k > 2$) makes the critical point transition much more complex. This grid-based method leaves out the complex transitions involving more than two critical points.

In our method, we propose a generalized critical feature tracking approach that works on TINs and handles complex transitions involving k-fold saddles and more than two critical points. Critical point transitions are categorized into three types: *Displacement*, *Appearance*, and *Collapse*. For instance, the transition (*maximum*, *regular*) → (*regular*, *maximum*) denotes a displacement of a maximum through the flipping edge from the first vertex to the other vertex. For the collapse and appearance transition, by the constraint of the Poincaré-Hopf index theorem [26] (see Equation 1), the transition can only happen to critical points in pairs of the form *(maximum-saddle)* or *(minimum-saddle)*, e.g. after a *Collapse* transition on a flipping edge, a *maximum* collapses with a *saddle* resulting in two *regular* points, as illustrated in Figure 9.

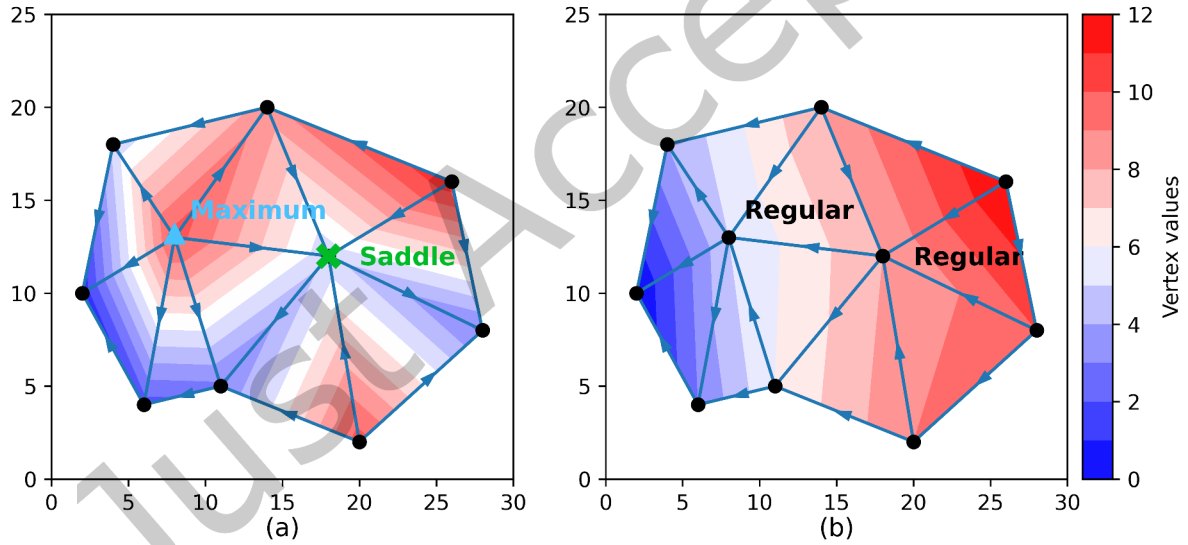


Fig. 9. *Maximum* and *saddle* collapse to two *Regular* points with the scale changes from fine (a) to rough (b). Face colors of triangles are calculated from interpolation on vertex values, and arrows on edges denote the value relationship from higher to lower.

A k-fold saddle ($k > 1$) can be interpreted as the co-location of k simple saddles at the same vertex. Therefore, when an edge flip affects a k-fold saddle, we decompose the event into a sequence of elementary transitions, each involving *one* simple saddle branch at a time. In each elementary step, only one saddle participates in the transition and the transition type of that step is determined by which critical-point pair is involved. For example,

the transition *(2-fold saddle, maximum) → (regular, saddle)* involves three critical points: the collapse of the (maximum-saddle) pair and the displacement of the other saddle. This transition involves two distinct saddles: one is paired with the maximum and *collapses* with it, while the other is *displaced* and survives as the remaining saddle. Thus, the key ambiguity is to decide which saddle collapses with the maximum and which one is displaced.

To resolve this matching, we use the *trace* of each critical point across previous flips. Intuitively, we prefer to match the critical point that is most likely to traverse the current flipping edge. We quantify this by a simple matching score: for each candidate saddle, we look at the most recent edge that it traversed in its trace and compare it with the current flipping edge. If these edges have similar direction in the xy-plane, we assign a higher score, indicating a higher likelihood that this critical point continues through the current edge. We then greedily match the highest-score saddle at the shared vertex to the critical point on the opposite side of the flipping edge (e.g., the maximum), and classify that pair as the collapsing pair; the remaining saddle is treated as the displaced one.

5.3.3 Critical points life span. After recording all the traces of critical points, to quantitatively measure the importance of each critical point, a *life span* is defined as the number of scales in which a critical point persists. Two types of critical points are distinguished:

- **Initial critical point**: a critical point extracted from the original TIN (scale s_0).
- **Newborn critical point**: a critical point that first appears at a coarser scale during the scale-space evolution.

During the evolution of the scale space, newborn critical points may collapse with initial critical points, shortening the life span of the initial critical points.

Therefore, a **life span recovery** process is necessary. A connection can be established between initial and newborn critical points at intermediate scales, using two mate relationships:

- **birth_mate**: newborn critical points always appear in pairs; the two points in a pair are each other's *birth_mate*.
- **death_mate**: when two critical points collapse, they become each other's *death_mate*.

Given an initial critical point p that collapses with a newborn critical point q, if $q.birth\_mate$ is still alive after the collapse timestamp, $q.birth\_mate$ is considered a substitution of p and is tracked until there is another collapse. In this way, the life span of p is extended by the life span of $q.birth\_mate$. This recovery process is applied repetitively and terminates when:

(a) the end of the scale space is reached, or
(b) the critical point (or its substitution) collapses with another initial critical point.

Thus, the life span of an initial critical point p is the timestamp when p collapses with another initial critical point (or its substitution), or the timestamp of the last scale layer if p remains alive throughout.

Within the sequential pipeline, every edge in the TIN must be examined across all scale layers. Let E and V denote the number of edges and vertices of the TIN, respectively, S the number of scale intervals, and F the number of edge-flip events generated during the analysis. For each scale, edge-flip detection requires $O(E)$ work, while the chronological processing of events requires a global sort of all F flips, leading to an overall runtime of $O(ES + F \log F)$. According to Euler's formula [35], for a connected planar graph, $F = O(E)$ and $E = O(V)$, which is the case for a TIN. Thus, the total complexity can be simplified to $O(ES \log E)$. In addition, computing vertex types involves neighborhood queries and bookkeeping operations that cost $O(\deg(v))$ per vertex, which further contributes to runtime overhead in dense regions.

The memory requirement of the sequential method is $O(E + V + F)$, dominated by storage of edge connectivity, per-vertex states, and the list of edge-flip events. The proposed pipeline has linear time and memory costs with respect to the number of vertices in the TIN, though with larger constants due to event storage and per-vertex auxiliary data. While providing efficient processing and low memory footprint, the sequential algorithm exhibits

both higher time complexity and less favorable scaling behavior due to the data structure overhead and the need to process edge-flip events in chronological order.

6 Fully Parallel Critical Point Tracking on TINs

In the sequential implementation, the use of complex data structures and frequent neighborhood queries results in long scale-space runtime. While in the parallel case, we make a trade-off between the space and time complexity. We do not use any data structure; instead, a plain descriptive array of vertices and edges is used. This leads to a much faster memory access at the cost of higher memory usage. However, one can still try to optimize the memory usage by using a GPU-friendly data structure and processing point clouds in chunks. This is left for future work. Here, we describe the parallel implementation and compare the speed-up factor with the sequential implementation.

In this section, we present a fully parallel algorithm for tracking critical points across the scale space of a TIN. Building upon the sequential pipeline described in Section 5, our parallel formulation explicitly exploits the locality of topological changes and the independence of edge-flip events. The section begins with a discussion of the motivation and problem setup (see Section 6.1), followed by the presentation of two key structural lemmas that form the theoretical foundation of our method. Subsequent subsections describe the parallel pipeline and implementation details (see Section 6.2), and complexity analysis (see Section 6.7).

6.1 Motivation, lemmas, and problem setup

The sequential implementation of critical point tracking is computationally expensive. At each scale layer, all edges of the TIN must be examined. Whenever edge-flip events occur, local neighborhoods must be queried to determine vertex type changes, and the resulting events must be globally sorted in chronological order. This design, while correct, scales poorly for large datasets. Moreover, as described in Section 5.2, the choice of data structures introduces a trade-off between memory usage and computational efficiency: compact representations reduce memory footprint but incur repeated local queries, whereas flat structural arrays simplify access at the cost of higher memory consumption.

As detailed in Section 4, the scale space is constructed through iterative smoothing operations, each of which may induce edge-flip events. The number of such events can be large, especially for complex terrains. However, these events are inherently *local: an edge flip affects only its incident vertices and these two vertices' critical point types are determined by their local neighbors*. This locality suggests that edge-flip events can be processed independently and in parallel across the entire scale space.

Formally, let $\mathcal{T} = (V, E)$ denote a TIN with vertex set V and edge set E. Each vertex $v \in V$ has an elevation value $s_t(v)$ at timestamp t, and an *edge flip* occurs on $e = (u, v) \in E$ when $s_t(u) = s_t(v)$ at some intermediate timestamp t between two consecutive scale layers. For each $v \in V$, let $N(v)$ denote its *1-ring neighborhood*, i.e., the set of vertices adjacent to v, ordered counter-clockwise in the xy-plane. As described in Section 2, we define the *vertex signature* of v at timestamp t based on the relative elevation of v with respect to the vertices in $N(v)$. Traversing the circular order of $N(v)$, the number of sign alternations between higher and lower neighbors equals the number of connected components k_{higher} and k_{lower}, which determines the critical point type of v (see Figure 1).

To enable efficient parallelization, we distill two structural lemmas that govern the behavior of critical points:

Lemma 6.1. *Critical point type of a vertex v changes if and only if one of its incident edges flips.*

It is obvious that the relative elevation relationship between two vertices changes only when the edge connecting them flips, which may alter the vertex signature and thus the critical point type.

Lemma 6.2. *When an edge flips, only its two incident vertices change their relative elevation relationship.*

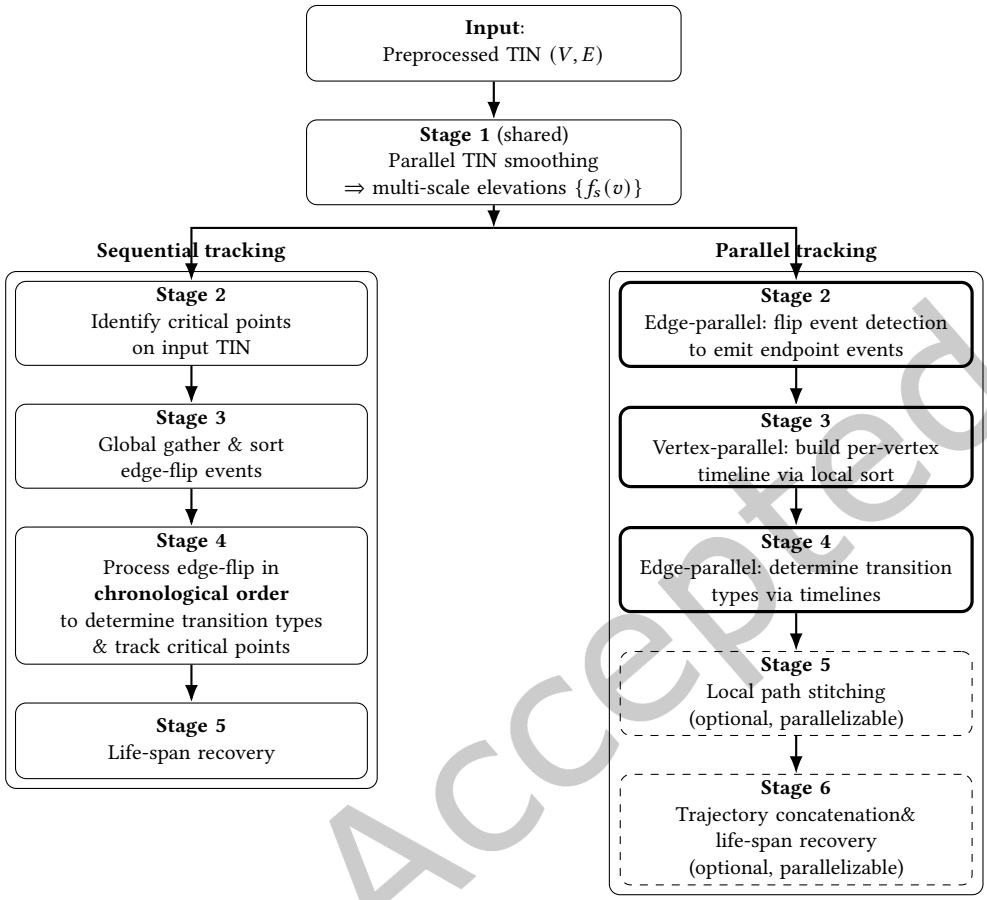


Fig. 10. **Side-by-side comparison of sequential and parallel critical-point tracking pipelines.** Both pipelines start from the same preprocessed TIN and share Stage 1 (parallel multi-scale TIN smoothing). The sequential pipeline performs global gathering and chronological sorting of edge-flip events before determining transitions and recovering life spans. In contrast, the parallel pipeline replaces global sorting with edge-parallel event detection and vertex-parallel local timeline construction (Stages 2–4), followed by optional trajectory reconstruction and life-span recovery (Stages 5–6). **Thick borders** indicate highly parallel stages, while **dashed boxes** denote optional stages.

As in [30], with some tie-breaking strategy, any two intersected edges are not flipped at the same timestamp. As a consequence, the relationships between the two vertices of the flipped edge with their direct neighboring vertices remain unchanged before and after this edge-flipping timestamp. Therefore, for numerical stability, instead of computing the vertex elevation at the exact edge-flip timestamp, the relative elevation between two vertices is tracked and toggled only when their connecting edge flips. In special cases where multiple edges incident to the same vertex flip simultaneously, a consistent order of processing these edges can be enforced to ensure deterministic results, or a small perturbation can be added to the elevation values to break ties as suggested in [30].

These lemmas imply that critical point transitions are strictly local to flipping edges. Consequently, the computation can be parallelized at both the edge and vertex levels: independent flips may be processed concurrently, and vertex-type updates can be performed without exhaustive neighborhood scans at every scale.

6.2 Parallel pipeline overview

Figure 10 gives an overview of the six-stage parallel pipeline. The parallel pipeline for critical point tracking is designed around *embarrassingly parallel kernels* and per-vertex operations, which avoid costly global synchronizations such as a full sort over all edge flip events. The procedure unfolds in six logically connected stages.

In **Stage 1**, multi-scale values are precomputed by applying parallel smoothing, yielding $\{f_s(v)\}$ for every vertex v across all scales s. This parallel TIN smoothing operation is detailed in Section 5.3.1. Based on these multiscale elevation values, **Stage 2** examines each edge within every consecutive scale interval $[s, s+1]$ to detect possible flip events. According to Equation 4, the edge-flip timestamp t^* between these two consecutive scales can be computed via linear interpolation. Therefore, $t^* \in (0, 1)$ represents the normalized timestamp of the flip event between two consecutive scale layers, with $t^* = 0$ corresponding to scale s and $t^* = 1$ to scale $s+1$. For each detected flip event, compact records are emitted for the two endpoints of the edge. According to Lemma 6.2, only these two vertices need to be updated, and their relative elevation relationship is toggled. These per-endpoint event records provide the input for the following vertex-centric processing.

Stage 3 processes all events associated with each vertex independently. Events are locally sorted by their timestamps, and a single scan over the ordered events constructs a piecewise-constant *vertex timeline* that consists of $(t^*, s_i, k_{upper}, k_{lower})$ records, and thus the critical point type. This local approach eliminates the need for a global event ordering, since the relevant updates are confined to the neighborhood of each vertex.

Once vertex timelines are established, **Stage 4** labels each flipping edge with critical point transition types. At the recorded flip time t^*, the critical point types of its two endpoints are queried from their respective timelines. Comparing the before-and-after states allows each flip to be classified as a *displacement*, an *appearance*, a *collapse*, or as a *non-event* if no critical type change occurs. Because this step involves only read operations from precomputed timelines, it is inherently parallel and lock-free.

Two additional stages are optional for parallelization, but further complete this pipeline. In **Stage 5**, local path stitching is performed in parallel at each vertex, where incident flips are matched to form short trajectory segments of critical points. **Stage 6** then concatenates these local segments into complete trajectories, allowing the computation of life span, birth and death times, and life span recovery of each critical point.

This pipeline design ensures that the dominant operations - *edge-based flip detection*, *vertex-level event processing*, and *edge-based transition labeling* - are performed in parallel. Optional post-processing stages extend the framework to reconstruct continuous critical point trajectories across the scale space. Pseudocode for the main stages is provided in Appendix E.

6.3 Stage 2: edge flip detection and event emission

The second stage detects edge-flip events that occur when the relative elevation of two adjacent vertices changes between consecutive scale layers. For each edge $\overline{v_m v_n}$, the elevation values at the two endpoints are linearly interpolated between two consecutive scales s_i and s_{i+1} via Equation 2, and the flip time $t^* \in (0, 1)$ is computed by Equation 4. This computation can be efficiently parallelized by assigning each edge to a GPU thread.

When a flip is detected, event records are emitted for both endpoints v_m and v_n. Each event record (s_i, t^*, v_{other}) encodes the flip time, the scale interval index, and the other vertex involved. The relative elevation relationship between v_m and v_n is toggled to reflect the flip. This design ensures that only local information is updated, adhering to Lemma 6.2.

The implementation uses a two-pass parallel scheme to handle the variable number of events per vertex. In the first pass, atomic counters record the number of events for each vertex, and an exclusive prefix-sum computes offsets for writing events into a compact array. In the second pass, events are written into per-vertex arrays using the computed offsets. This lock-free design ensures deterministic output and efficient use of memory bandwidth.

6.4 Stage 3: vertex timeline

The third stage constructs a temporal sequence of critical point signatures for each vertex independently. All records attached to a vertex are first sorted by their time keys (s_i, t^*, v_{other}) using a lightweight parallel sort suitable for small per-vertex lists. The number of records of each vertex v is bounded $S \times deg(v)$, where S is the number of scale layers and $deg(v)$ is the degree of the vertex.

Initialization is performed at the left endpoint of the first interval by comparing the elevation of the vertex against all neighbors to compute the initial upper- and lower-components (k_{upper}, k_{lower}). A initial *knot* $(0, 0, k_{upper}, k_{lower}, \texttt{type})$ is inserted to the vertex time. Processing then proceeds in a single pass over the sorted events. As illustrated in Figure 11, each event flips one neighbor x's sign, i.e., $\text{sgn}(x) \to -\text{sgn}(x)$. Let $a = \text{prev}(x)$ and $b = \text{next}(x)$ in the ring of v, and take old signs $\text{sgn}(a), \text{sgn}(x), \text{sgn}(b) \in \{+, -\}$. Only the two boundaries (a, x) and (x, b) can change the connected component count. Define ΔC as the change in the number of alternations contributed by these two boundaries:

$$\Delta C = \begin{cases} +2, & \text{if } \text{sgn}(a) = \text{sgn}(b) = \text{sgn}(x) \quad \text{(split)}, \\ -2, & \text{if } \text{sgn}(a) = \text{sgn}(b) \neq \text{sgn}(x) \quad \text{(merge)}, \\ 0, & \text{if } \text{sgn}(a) \neq \text{sgn}(b) \quad \text{(boundaries swap)}. \end{cases}$$

Update k_{upper} and k_{lower} by $\Delta C/2$ accordingly and flip $\text{sgn}(x)$.

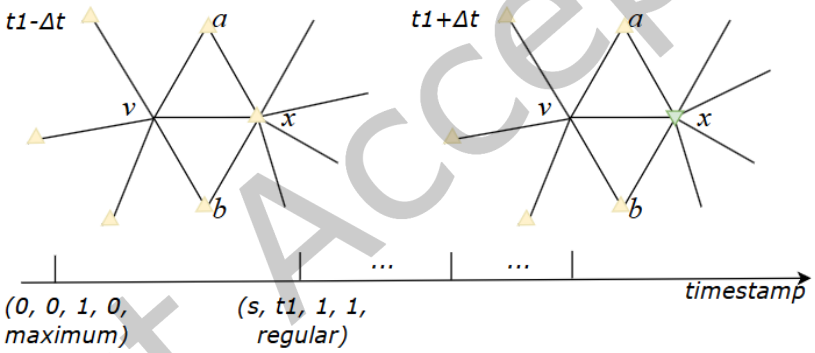


Fig. 11. Vertex timeline construction. Vertex v starts as a *maximum* with $k_{upper} = 1$ and $k_{lower} = 0$. The first edge flip happens at scale s and time t_1 on the edge (v, x). Vertex type of v transitions to a *regular* with $k_{upper} = 1$ and $k_{lower} = 1$, because $\text{sgn}(a) = \text{sgn}(b) = -\text{sgn}(x) = +$.

After processing all events with the same time key, a new *knot* $(s_i, t^*, k_{upper}, k_{lower}, \texttt{type}_{\text{after}})$ is appended to the vertex timeline, recording the signature and critical point type after edge flipping. Boundary vertices are handled consistently by adding a virtual global minimum neighbor (i.e., $\text{sgn}(x) := -$) to the 1-ring as suggested by [30]. This vertex-centric construction is embarrassingly parallel, as each vertex operates on its local event list only.

6.5 Stage 4: edge transition labeling

Once vertex timelines are available, each flipping edge can be labeled independently. For a flip event at time key (s_i, t^*) on edge $e = (v_n, v_m)$, the algorithm queries the critical point types of v_n and v_m immediately before and after the flip by searching their timelines. Similarly, the number of knots on each timeline is bounded by

$S \times deg(v)$, and the relevant knot can be found efficiently via local binary search. Based on this information, an edge-flip event is classified as in Section 5.3.2.

Because all required information is available in read-only form from precomputed timelines, this labeling step is both deterministic and highly parallelizable for each flipping edge. Each flip is processed by an independent thread, requiring no synchronization with other edges. The result is a complete, parallel classification of all topological events across the scale space.

6.6 Stage 5 and 6: local path stitching and global trajectory concatenation

Stages 5 and 6 complete our parallel processing pipeline by reconstructing explicit critical point trajectories across the scale space. These steps are optional because the number of edges for critical point trajectories is significantly smaller than the total number of edges in the TIN, and thus their processing has a negligible impact on overall complexity.

In Stage 5, local path stitching is performed independently at each vertex. For every time key t where multiple incident edges flip simultaneously, the corresponding events are matched in pairs to form short trajectory segments. Following the method in Section 5.3.2, matching follows simple local rules: if exactly two incident flips occur, they are paired directly; if more than two events coincide, pairs are selected based on type compatibility and continuity heuristics (e.g., momentum from the previous direction of movement). In cases of odd multiplicity, one event remains unmatched, representing the start or end of a trajectory. Higher-order saddles are decomposed into multiple simple saddle slots to ensure consistent processing. Each successful match yields a compact segment record linking incoming and outgoing edges at the vertex.

Stage 6 concatenates these local segments into global critical point trajectories. This is achieved by traversing the segment graph in parallel, following edge-in to edge-out links until a birth or death event (appearance or collapse) is reached. Each trajectory records its birth time (first appearance) and death time (final collapse), and the same life span recovery process can be applied to initial critical points as in Section 5.3.3.

6.7 Complexity and scalability

The parallel pipeline achieves nearly linear complexity relative to the input size. Edge-flip detection requires $O(ES)$ work across the scale space, fully parallelizable. Per-vertex sorting and timeline construction cost $O(\sum_v m_v \log m_v)$, where m_v is the number of events at vertex v, but since vertex degrees are small, this step is practically close to linear in the total number of flips K. Both edge transition labeling and optional trajectory reconstruction add linear work in K. Memory usage is $O(V + E + K)$, dominated by storage of TIN structures, per-vertex events, timelines, and edge flip labels. Compared with the $O(ES \log E)$ sequential baseline, the parallel design reduces runtime to $O((ES + K)/P)$ on P threads, trading higher memory consumption for scalable performance and efficient GPU execution on large TINs.

7 Experimental comparison

In this section, we evaluate the effectiveness and efficiency of our pipeline through an experimental comparison with the *grid-based method* [30]. Experiment setup and evaluation metrics are first introduced in Section 7.1. Then, Section 7.2 shows the effectiveness of our scale-space method for TINs compared to gridded DEMs. Additionally, Section 7.3 further evaluates the resolution robustness of our TIN-based method compared to the grid-based method.

To complement the spot height selection benchmark, we add three new evaluations: runtime and memory performance of the parallel pipeline (Section 7.4), quality of the scale-aware seam-free sampling (S3) (Section 7.5.1), and quantitative fidelity of the proposed TIN smoothing method (Section 7.5.2).

7.1 Experiment setup and evaluation metrics

Our experimental setup encompasses a comparative assessment of our extension on TINs versus prior work on gridded DEMs by [30]. The same evaluation task, *spot height selection*, is employed, in which maxima of input terrain data are extracted based on their topological prominence. The terrain datasets consist of the point cloud collected from different regions and the spot heights manually compiled by human experts to represent terrain topography. These datasets are available from the Swiss Federal Office of Topography (swisstopo) and the dense ground point clouds are from their sub-dataset swissSURFACE3D [34], featuring an average of $15 - 20 pts/m^2$ of the terrain topological data, which are directly fed into our pipeline.

Figure 12 is an example of the spot heights in a mountainous region. Most prominent spot heights are given names during the map generation process, denoted as *named peaks*. Other spot heights are noted as *summits* in the rest of the paper. Named peaks and summits together are considered the ground truth for the spot height selection task. To evaluate our method, the maxima identified by our pipeline are compared with the ground truth. As a benchmark, gridded DEMs of different resolutions are generated from the same input point clouds using the PDAL library [8]. A summary of the datasets tested in the experiment can be found in Table 1.

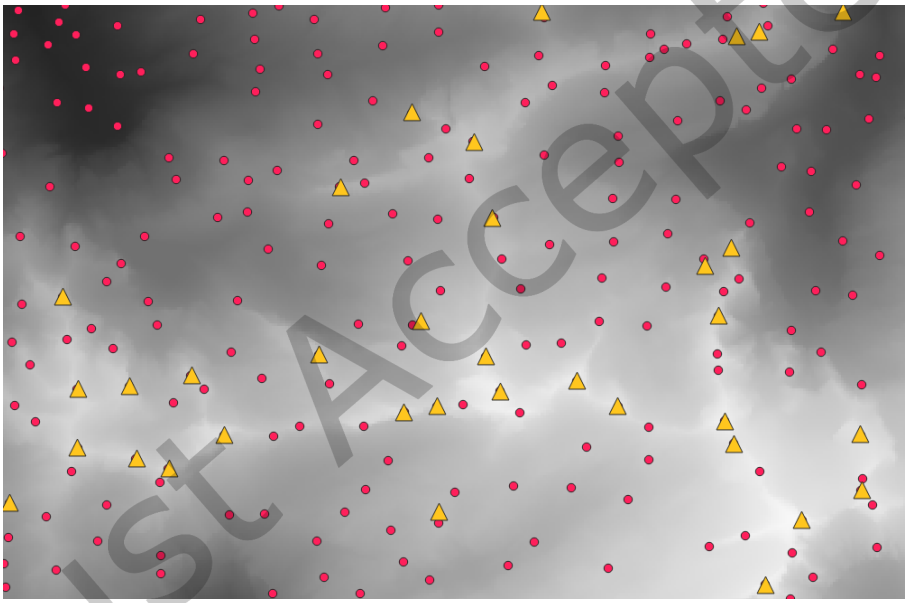


Fig. 12. An example of a spot heights dataset. Red circles denote spot heights and yellow triangles denote named peaks.

To evaluate the performance of our pipeline, the spot height selection task is formulated as a binary classification problem. Maxima are matched with peaks and summits in the ground-truth dataset. The matching distance is preset as 50 meters, which is aligned with the previous work [30]. Let TP denote the number of detected maxima that are correctly matched to a ground-truth spot height (true positives), FP the number of detected maxima that do not match any ground-truth spot height (false positives), and FN the number of ground-truth spot heights left unmatched (false negatives). Precision is then defined as $TP/(TP + FP)$, i.e., the fraction of detected maxima that correspond to a true spot height. Recall is defined as $TP/(TP + FN)$, i.e., the fraction of ground-truth spot heights that are successfully detected. The precision-recall curve (PR curve) and the F_β score are used as evaluation metrics, generated by adjusting the life span filtration threshold for maxima. The F_β score is a weighted harmonic

Table 1. A summary of the datasets used in the experiment. Datasets are classified into two categories, mountainous and suburban, by their terrain types. #points denotes the number of points in the raw point cloud from the swissSURFACE3D [34].

Dataset	Dimension (km)	#summits	#peaks	#points (million)	Type
Reichenburg	4×4	7	9	167	Suburban
Sörenberg	8×8	16	16	1529	Mountainous
Bannelpsee	12×12	22	67	3548	Mountainous

mean of precision and recall rate, as per Equation 10:

$$F_\beta = (1 + \beta^2) \frac{precision \times recall}{\beta^2 \times precision + recall} \tag{10}$$

where β is set to 0.5 to prioritize precision twice as important as the recall rate. In addition, an alternative evaluation metric $dist_{avg}$ is introduced, which denotes the average distance from the critical points to the corresponding ground-truth spot heights. It measures the precision and robustness of the critical point identification.

To evaluate the computation efficiency of our pipeline, we measure the runtime and peak memory usage of the critical point tracking stage. Point cloud preprocessing, TIN generation, and TIN smoothing are excluded from the evaluation as they depend on various factors such as the scale of the input point cloud, the targeted resolution, and the topographical complexity of the terrain. Therefore, these stages are processed offline for each dataset. The smoothed elevation value of each vertex is stored as pre-computed vertex properties. Experiments are conducted on a workstation with an Intel Core i9-12900K CPU, 64GB of RAM, and an NVIDIA GeForce RTX 3080Ti GPU.

7.2 Spot height selection results

For the input dense point cloud data, we generate both gridded DEMs and TINs of a similar number of vertices by tuning the FPS sampling rate. To take advantage of low memory footprint of Terrain Tree data structure, both gridded DEMs and TINs are processed using the *sequential algorithm* described in Section 5. From the results summarized in Table 2, our pipeline's runtime and peak memory usage are close to those of the grid-based method since our extension on TINs does not introduce additional computational overhead. In line with our analysis, the time elapsed for the scale-space critical point tracking stage reveals a linear correlation of the runtime and peak memory usage with the total number of edges in a TIN. The random memory access time for TINs is slightly longer than that for gridded DEMs.

Table 2. Evaluation results of the grid-based and TIN-based methods on the datasets. For both methods, columns represent different metrics: number of edges *#edges*, average critical point matching distance $dist_{avg}$, and best F_β *score*. The time elapsed t_{ss} and peak memory usage Mem_{peak} are recorded for the critical point tracking stage.

Datasets	Grid	#edges		$dist_{avg}$ (m)		F_β *score*		t_{ss} (s)		Mem_{peak} (MBs)	
	cell size (m)	Grid	TIN	Grid	TIN	Grid	TIN	Grid	TIN	Grid	TIN
Reichenburg	16	186,501	170,265	18.20	**4.85**	0.79	**0.92**	39.33	36.52	24.38	22.75
Sörenberg	40	119,201	112,000	30.38	**12.79**	0.85	**0.87**	26.077	27.81	17.35	17.00
Bannelpsee	40	268,801	255,945	32.18	**14.19**	0.77	**0.79**	56.88	63.43	32.65	32.04

The preprocessing stage plays an important role in our pipeline in taking advantage of TINs. Taking the $4 \times 4\ km^2$ suburban region Reichenburg dataset as an example, the number of vertices and edges of the TIN are

57, 065 and 170, 265 respectively, while the gridded DEM has 62, 500 vertices and 186, 501 edges. With the help of our sampling method, the variable-resolution capability of the TIN allows allocating of more vertices in the mountainous area than in the flat one. A small adjustment is applied in the grid-based method to flat edges as a tie-breaking solution [30]. This solution, however, introduces artificial critical points within the flat regions, called *artifacts*, to the model as seen in Figure 13(b). While our method adopts the same tie-breaking method in flat areas, the number of vertices retained within flat regions is significantly reduced by our PFPS procedure, thereby reducing the occurrence of artifacts. Furthermore, the frequency of edge flip events within flat areas decreases due to the reduced number of edges subjected to the smoothing process.

For a more detailed comparison, Figure 13 provides a zoomed-in view of the sub-regions. Within the same dataset, different regions are of the same resolution in the gridded DEM. In the urban region (Figures 13 (a-b)), more artifacts are introduced in the gridded DEM, because of the high-resolution grid and small differences between neighboring vertices. The number of critical points reduces from 226 (in the gridded DEM) to 98 (in the TIN), while the same prominent critical points are recognized. However, in the mountainous sub-region shown in Figures 13 (c-d), the gridded DEM provides much fewer surface details with only 1, 600 vertices, and fewer critical points are identified from the gridded DEM compared to the TIN, which contains 11, 447 vertices in the same region. The gridded DEM, constrained by its fixed grid size, fails to adapt to the varying distribution of topographical features, thereby leading to inefficient utilization of computational resources. On the other hand, the variable-resolution capability of TIN enables more accurate locating and tracking of critical points in scale space.

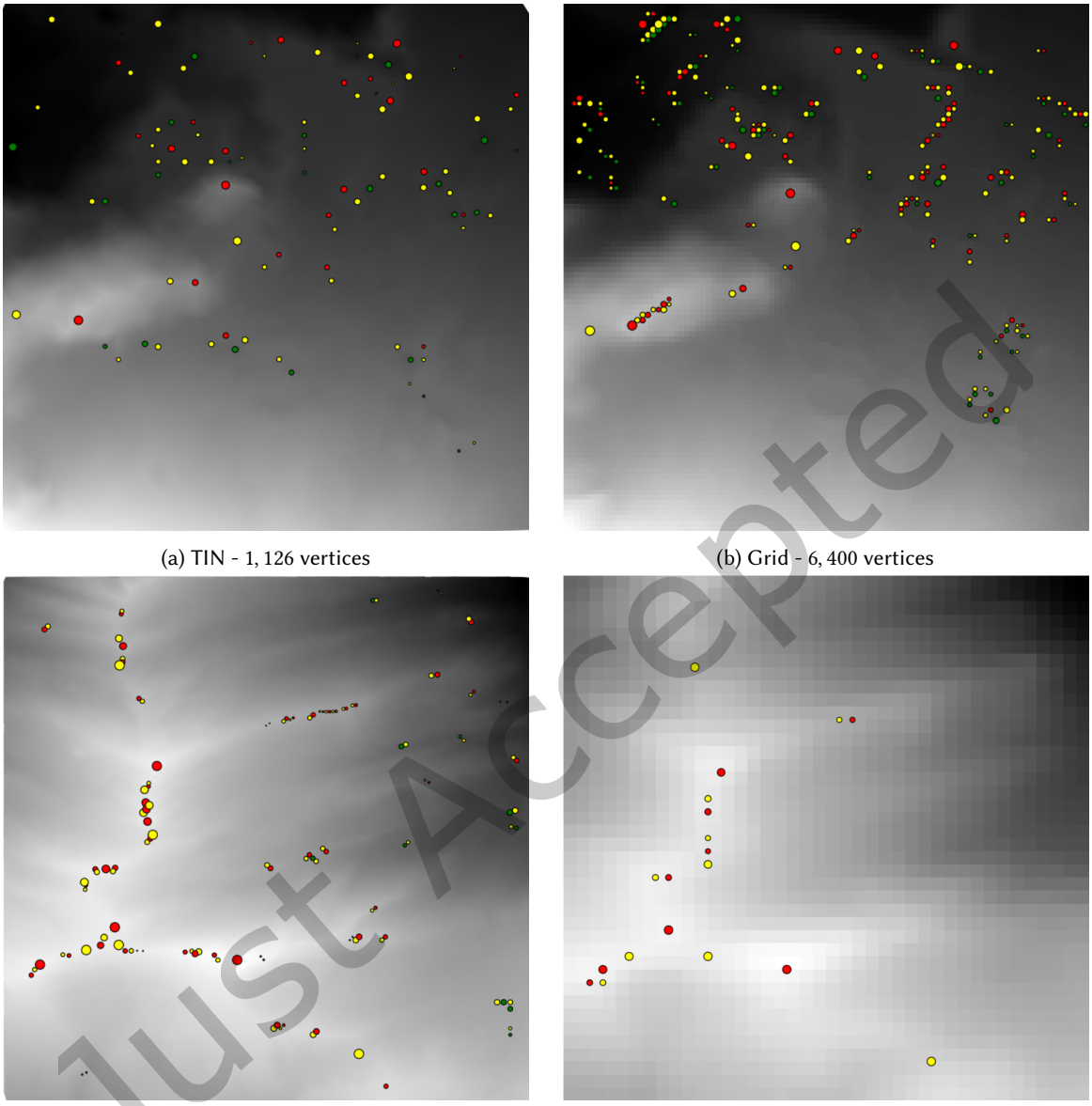
For a quantitative evaluation, the PR curve and F_β score graph of the grid-based and TIN-based methods are drawn, respectively. For example, Figure 14 illustrates the result derived from a mountainous region near Bannelpsee. It is worth noting that the gridded DEM fails to identify some of the critical maxima as the recall rate stops at 0.73. In contrast, the TIN-based method's PR curve covers higher recall rate regions, and the best F_β score is 0.79 compared to 0.77 of the grid-based method. The high resolution of the TIN in regions with more terrain features facilitates the recognition of more critical points in the input data and, thus, more spot heights.

As shown in Table 2, our TIN-based method exhibits a slight improvement in the F_β score in mountainous regions, with increases of 2.35% in Sörenberg and 2.60% in Bannelpsee, and a large improvement in a suburban region, with an increase of 16.5% in Reichenburg, compared to the grid-based method when the numbers of edges are similar in two terrain models. This highlights the advantages of the TIN-based method, as it can accurately identify critical points and minimize the risk of artifacts. Furthermore, leveraging the benefits of the PFPS method and flexible resolution, the average distance $dist_{avg}$ of the TIN-based method is considerably smaller than that of the grid-based method. For instance, in the mountainous region Bannelpsee, $dist_{avg}$ of the TIN-based method is $14.19m$, while that of the grid-based method is $32.18m$. In the suburban region Reichenburg, the $dist_{avg}$ of the TIN-based method is 3.75 times smaller than that of the grid-based method.

7.3 Resolution robustness evaluation

We evaluate the robustness of our TIN-based method compared to the grid-based method under varying resolutions. For clarity, we define resolution as the side length of the grid cell in the case of the gridded DEM. To make a fair comparison, for the input point dataset, we generate a TIN with a similar number of vertices to the gridded DEM. This allows us to compare the resolution robustness under a similar experimental configuration. Without loss of generality, we use the Bannelpsee dataset as an example to show the comparisons between these two terrain models. Experiments on other datasets show similar results and are not listed here.

In Figure 15, the x-axis shows the resolution changing from $40m$ to $320m$ and the y-axis represents the F_β score and $dist_{avg}$. The graphs show that our TIN-based pipeline has superior resolution robustness compared to



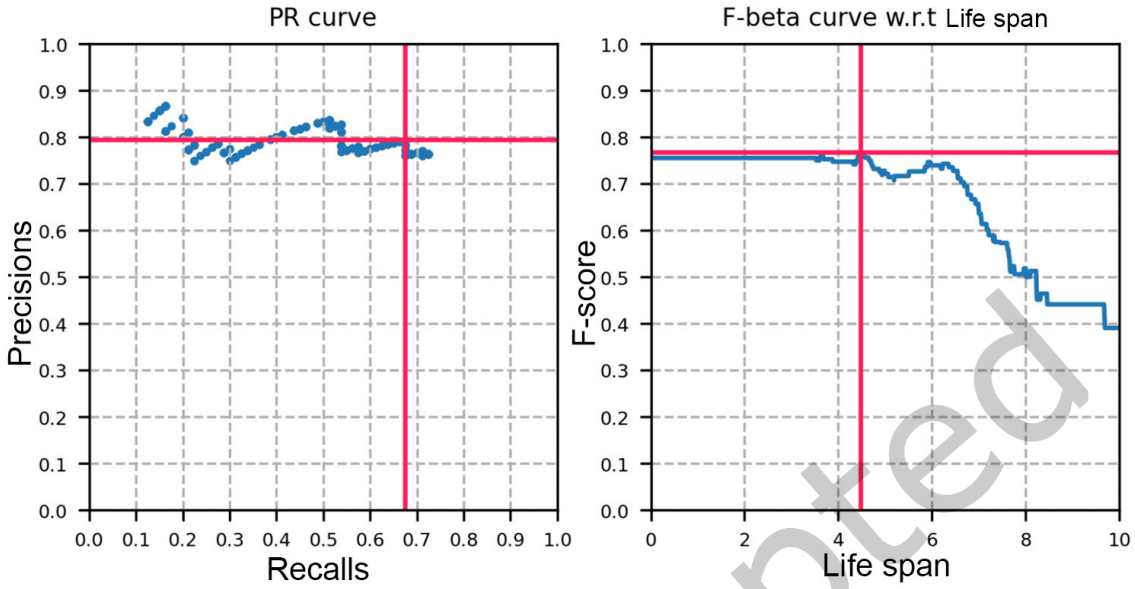
(a) TIN - 1, 126 vertices

(b) Grid - 6, 400 vertices

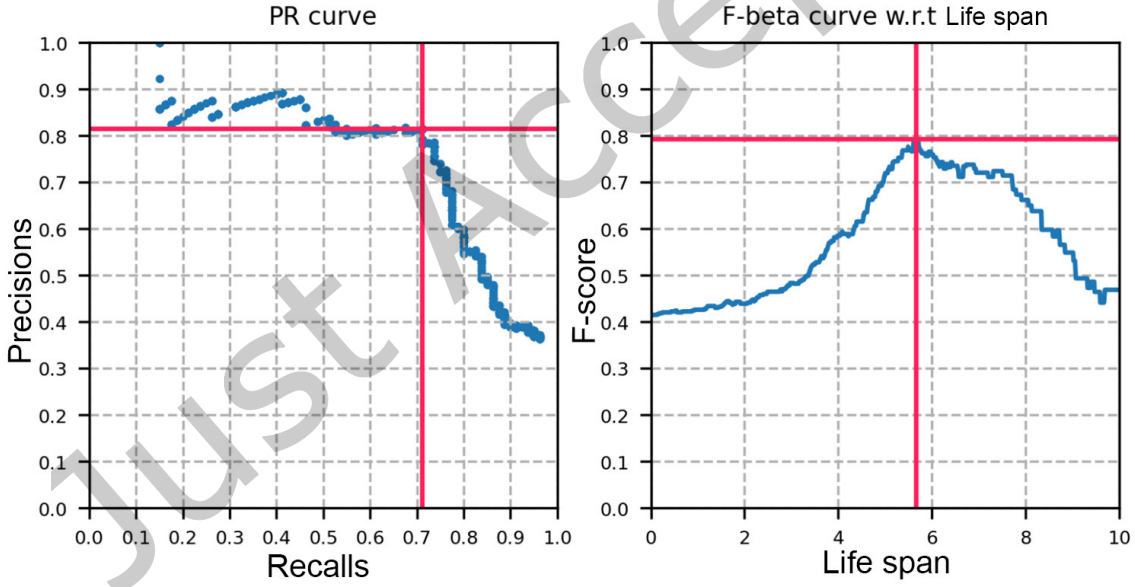
(c) TIN - 11, 447 vertices

(d) Grid - 1, 600 vertices

Fig. 13. Visual comparison of two zoomed-in sub-regions of the Reichenburg dataset. Among them, (a,b) are of an urban region with fewer terrain changes. (c,d) are of a mountainous region with complex terrain features. Maxima, minima, and saddles are marked in red, green, and yellow circles with sizes proportional to the life span of the critical points.



(a) Gridded DEM - best $F_\beta = 0.77$



(b) TIN - best $F_\beta = 0.79$

Fig. 14. PR curve and F-score graph of the gridded DEM and TIN methods on the mountainous regions Bannelpsee.

the grid-based method, especially when there is limited memory and the grid cell size is large. Our curvature-based sampling strategy ensures that regions with sparser topographical information are subjected to greater downsampling. Since critical points do not appear on a flat surface, fewer vertices in the flat areas will not affect the identification of critical points in the TIN. In contrast, gridded DEMs' resolution directly impacts the initial identification of critical points, and consequently, the final critical point tracking results. Besides, $dist_{avg}$ also grows inversely proportional to the resolution diminishing. These observations show that, when a suitable downsampling strategy is applied, TINs have better resolution robustness compared to gridded DEMs. Visualizations of gridded DEMs and TINs at different resolutions are provided in Appendix C for further comparison.

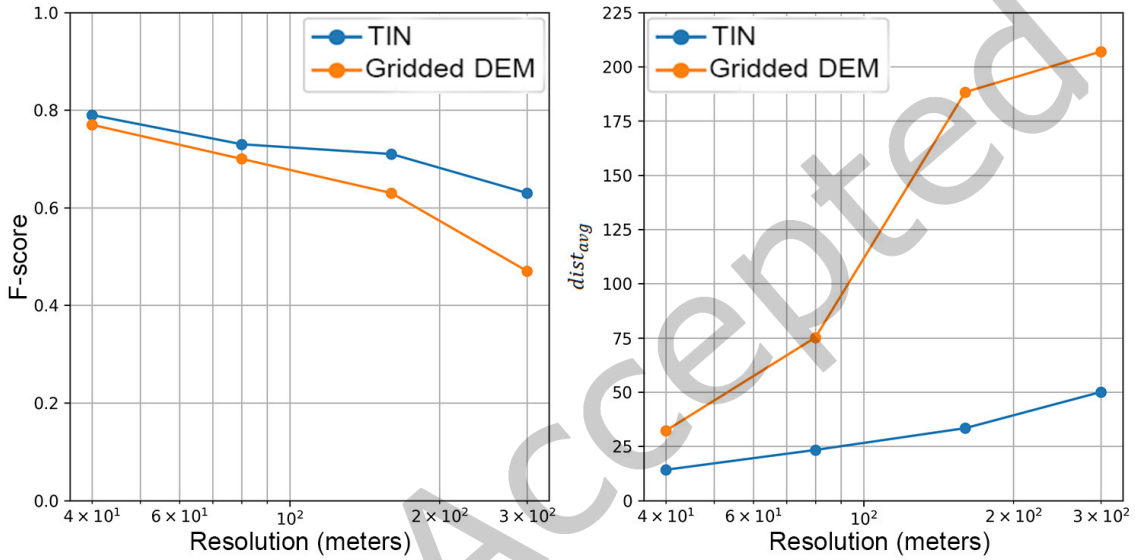


Fig. 15. Resolution robustness comparison between grid-based and TIN-based methods in Bannelpsee. F_β *score* (left) and $dist_{avg}$ (right) are plotted as lines. x-axes are in log scale.

7.4 Parallel pipeline evaluation

To assess the efficiency and scalability of the proposed parallel pipeline, we evaluate Stages 2–4 (edge-flip detection, vertex timeline construction, and edge transition labeling) on smoothed TINs with pre-computed per-vertex elevation values at multiple scales. Our evaluation focuses on these three stages for the following reasons. First, Stage 1 (TIN smoothing) is shared between the sequential and parallel pipelines and is already parallelized on the GPU (see Section 5.3.1), so it does not contribute to the differential comparison. Second, Stages 5 and 6 (path stitching and trajectory concatenation) operate only on the subset of edges that participate in critical point trajectories, which is significantly smaller than the total edge count; consequently, their runtime is negligible relative to Stages 2–4, and parallelizing them yields limited additional speedup. Third, as shown in Figure 16, Stages 2–4 constitute the computational bottleneck and scale almost linearly with $|V|$, so the end-to-end speedup of the full pipeline is dominated by their performance.

Although parallel-friendly spatial data structures for GPUs have been explored in prior work [19, 25, 32], our focus here is on the parallelization of the core algorithm rather than memory optimization. For this reason, the

TIN is represented as a structure of arrays (SoA) on the GPU, including vertex coordinates, scale-dependent elevation values, connectivity information, and auxiliary arrays for fast neighbor lookup (1-ring of each vertex), as well as storage for events and vertex timelines. All preprocessing steps are performed on the CPU using the Terrain Tree data structure, after which the relevant arrays are transferred once to GPU memory prior to Stage 2.

The GPU implementation launches dedicated CUDA kernels for each stage, with a block size empirically set to 256 threads on our test platform. As a baseline, we implement the same algorithmic stages sequentially in Python on the CPU. This baseline follows the same algorithmic design as the GPU version: relative elevation orderings are maintained per vertex, and edge-flip events are processed in strict chronological order. We emphasize that this comparison is *not* intended to benchmark against a production-quality CPU implementation; a highly optimized C++ implementation with specialized data structures (e.g., the Terrain Tree [12]) would be substantially faster on the CPU. Rather, the purpose of this experiment is to validate that the parallel formulation is *lock-free and highly scalable*: Figure 16 shows that runtime scales almost linearly with the number of GPU threads and vertices, confirming the embarrassingly parallel nature of the algorithm as analyzed in Section 6.7.

We measure wall-clock runtime and GPU allocated memory pool after a one-time warmup. Varying the sampling size of the Reichenburg point cloud, runtime comparisons are reported as a function of $|V|$, the number of vertices in the TIN. As shown in Figure 16, the parallel pipeline achieves at least 1000× speedup over the sequential Python baseline on a RTX3080Ti, with memory usage remaining approximately linear to vertex count.

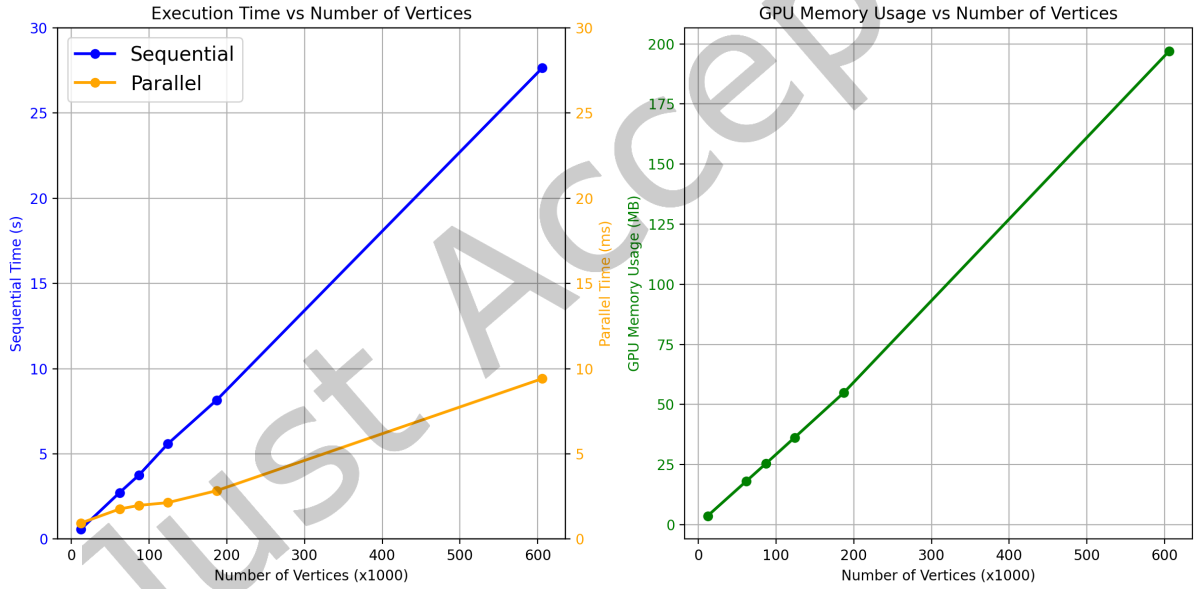


Fig. 16. Scaling performance of the parallel pipeline for Stages 2–4. (Left) Runtime comparison between the sequential CPU baseline (Python) and the parallel GPU implementation as a function of the number of vertices $|V|$. Note the unit of sequential implementation is seconds while the unit of parallel implementation is milliseconds. (Right) Allocated GPU memory pool size for the parallel implementation. The parallel pipeline achieves at least 1000× speedup on a RTX3080Ti, with memory usage remaining linear to vertex count.

The observed results confirm the theoretical complexity analysis. Sequential execution requires $O(ES \log E)$ time, where E is the number of TIN edges and S is the number of scale intervals, due to global sorting of events, whereas the parallel pipeline reduces this cost to approximately $O((ES + K)/P)$, where K is the number of edge

flips and P is the number of GPU threads. In practice, runtime is dominated by Stage 2 (edge-flip detection) and Stage 3 (per-vertex sorting and timeline construction) as $|V|$ increases, consistent with their $O(ES)$ and $O(\sum_v m_v \log m_v)$ complexity. Memory usage remains linear in $V + E + K$, with overhead primarily due to storing per-vertex event buffers and timelines. While the SoA representation incurs additional memory relative to more compact data structures, it enables highly efficient parallel access patterns and predictable scaling, making it well-suited for large terrain datasets. Future work may explore hybrid GPU data structures to further reduce memory consumption without sacrificing performance.

CPU multi-threading for edge flip detection. Among the sequential pipeline stages, only Stage 2 (edge-flip detection) is directly amenable to CPU multi-threading: each edge can be examined independently, with no data dependency between edges. Stages 3 and 4 in the sequential formulation, by contrast, require processing edge-flip events in strict chronological order to correctly determine critical point types and transition labels, precluding straightforward CPU parallelization. Detailed timing results for Stage 2 with 1, 2, and 4 CPU threads versus the GPU are provided in the Appendix. In short, CPU multi-threading yields up to 3.2× speedup with 4 threads but does not scale further due to Python's Global Interpreter Lock (GIL), which prevents true parallel execution of CPU-bound threads beyond what NumPy's internal BLAS routines can release. Even so, the GPU implementation achieves approximately 80–194× speedup over the single-threaded CPU baseline for Stage 2 alone, underscoring the advantage of massively parallel GPU architectures for this workload.

7.5 Additional evaluations

7.5.1 Scale-aware sampling evaluation. We compare the proposed S3 sampling method against various sampling methods: patch-based FPS (PFPS), grid-based sampling, Poisson disk sampling [5], uniform FPS [11], and random sampling, under the same vertex budgets. For our S3 method, the raw point cloud is first rasterized for curvature estimation with a resolution of 8 meters. The multiscale curvature map is constructed using the Laplacian curvature estimator via exponential moving average with a smoothing factor $\gamma = 0.7$. Within each 160m×160m patch, the number of samples is determined by the normalized mean curvature of the patch (and a lower bound of 10 samples per patch is enforced). Without losing generality, we report the sampling result on the Reichenburg dataset, in which the sampling budget is set to 62500 points (the same as in the previous experiments).

Table 3. Quantitative comparison of different point cloud sampling methods on the Reichenburg dataset. Geometric fidelity is evaluated by pointwise absolute error (average, maximum, and 95th percentile), and curvature preservation is measured by curvature map MAE. Topological preservation is assessed through the number of detected maxima, recall against ground-truth spot heights, and matching distances. Best and second-best results in each column are highlighted in **bold** and underline, respectively.

	Pointwise Abs. Error ↓			Curvature MAE ↓	Topological info			
	Avg	Max	P95		#maxima	recall ↑	$dist_{min}$ ↓	$dist_{avg}$ ↓
S3 (ours)	0.69	20.66	2.39	**3.6e-4**	205	**1.00**	2.40	**21.82**
PFPS (ours)	0.71	19.98	2.42	4.2e-4	304	**1.00**	2.80	24.14
grid-based	1.90	27.20	5.93	6.8e-4	121	0.74	8.21	21.86
Poisson disk [5]	**0.60**	**19.87**	**2.09**	7.4e-4	489	**1.00**	2.70	23.45
uniform FPS	0.69	21.58	2.48	7.8e-4	463	**1.00**	2.95	22.14
random	0.86	25.32	3.17	8.0e-4	477	**1.00**	**2.04**	26.34

Table 3 reports three metrics: pointwise absolute error, curvature map MAE, and topological information preservation. Pointwise absolute error (reported by average, maximum, and 95-percentile values), is calculated

by the difference between the original elevation and the barycentric interpolated elevation values of the input point cloud. Curvature map MAE is the mean absolute error between the curvature map estimated from the sampled points and that from the original point cloud at resolution 16 meters via the Moving Least Squares (MLS) method [21]. From the table, S3 consistently achieves strong performance for geometric fidelity while avoiding visible seams in the generated TINs (see Figure 5). For topological information preservation, we report the number of initial critical maxima, spot height recall rate, and the minimum and average matching distance to the ground truth spot heights. S3 achieves the highest recall rate and the smallest matching distances, indicating its effectiveness in preserving topological features. Other methods either miss spot heights (e.g., grid-based sampling) or introduce many insignificant critical points (e.g., Poisson disk sampling). In addition, although uniform FPS and Poisson disk sampling method achieve good geometric fidelity, they are not scalable to process large-scale point clouds and they took 2 magnitudes longer processing time.

7.5.2 TIN smoothing evaluation. Similar to the evaluation of sampling method, we report the comparison of different TIN smoothing methods on the Reichenburg dataset in Table 4. To quantitatively assess the accuracy of TIN smoothing, we employ a high-resolution rasterization of the input point cloud (8 meters per pixel) as ground truth. Four variants are compared: *naive Gaussian smoothing*, Gaussian smoothing with *virtual neighbors*, Gaussian smoothing with *angle re-weighting*, and our *full method* combining both improvements. We report two complementary metrics:

- **Geometric fidelity.** For each scale layer, we compute the absolute elevation error by comparing TIN vertex values with bilinearly interpolated ground-truth elevations. Errors are summarized by mean, 95th percentile (P95), and maximum values.
- **Gradient fidelity.** At an intermediate middle scale (5th layer), we smooth the ground-truth raster, compute gradient magnitudes (via Sobel filtering [16]), bilinear interpolate to TIN vertices, and compare them with vertex gradient magnitudes estimated from the TIN (via aggregated face-gradient of each vertex). The mean absolute error (MAE) of gradient magnitudes is reported.

Table 4. Quantitative evaluation of TIN smoothing quality. Geometric fidelity is reported as absolute elevation error (mean, 95th percentile, and maximum) at all scales. Gradient fidelity is reported as the gradient magnitude MAE at the middle layer (5th). Best and second-best results in each column are highlighted in **bold** and underline, respectively.

Method	Geometric fidelity (Abs. error)			Gradient fidelity
	Avg↓	Max↓	P95↓	MAE↓
Naive Gaussian	5.51	35.89	15.69	4.71e-2
+Virtual neighbors	5.67	20.78	12.16	3.50e-2
+Angle re-weighting	<u>3.17</u>	<u>20.46</u>	<u>8.85</u>	<u>2.35e-2</u>
Full method (ours)	**2.38**	**9.88**	**5.27**	**1.74e-2**

Overall, the results in Table 4 demonstrate that both virtual neighbors and angle re-weighting are essential for accurately approximating Gaussian smoothing on TINs. Aligned with the qualitative analysis with contour lines in Figure 7, our full method yields the lowest absolute elevation errors and halves the gradient MAE compared to the angle-only variant. These improvements translate into smoother contours and more reliable preservation of terrain slopes in the scale space.

7.6 Case study: Bathymetry dataset with irregular boundary

To illustrate the applicability of our pipeline to real-world datasets with irregular boundaries and missing data, we apply it to a bathymetric LiDAR dataset from the National Oceanic and Atmospheric Administration (NOAA)

describing the offshore bottom surface in the vicinity of Swanquarter Bay. Bathymetric terrains are characterized by highly irregular coastlines and large regions without valid measurements, which pose challenges for grid-based DEM representations. In gridded models, missing cells must be interpolated or masked, complicating both smoothing and topological analysis and often introducing artificial artifacts along the boundary.

In contrast, a TIN directly represents only the available sample points and naturally conforms to the irregular coastline without requiring artificial padding or interpolation. This property is particularly advantageous for scale-space analysis, where smoothing and critical point tracking must respect the geometry of the domain. Similar motivations for TIN-based terrain analysis have been discussed in prior work on continuous scale-space representations and critical feature tracking on irregular meshes [30], where TINs were shown to better preserve geometric and topological structure near complex boundaries.

Figure 17 shows the maxima and minima extracted from the Swanquarter Bay dataset with life span values larger than 6. The size of each circle encodes the life span of the corresponding critical point and thus provides a measure of its topological persistence across scales. The detected extrema are well separated and aligned with salient bathymetric structures, demonstrating that the proposed TIN-based pipeline can robustly identify meaningful topological features even in the presence of irregular boundaries and spatially varying sampling density.

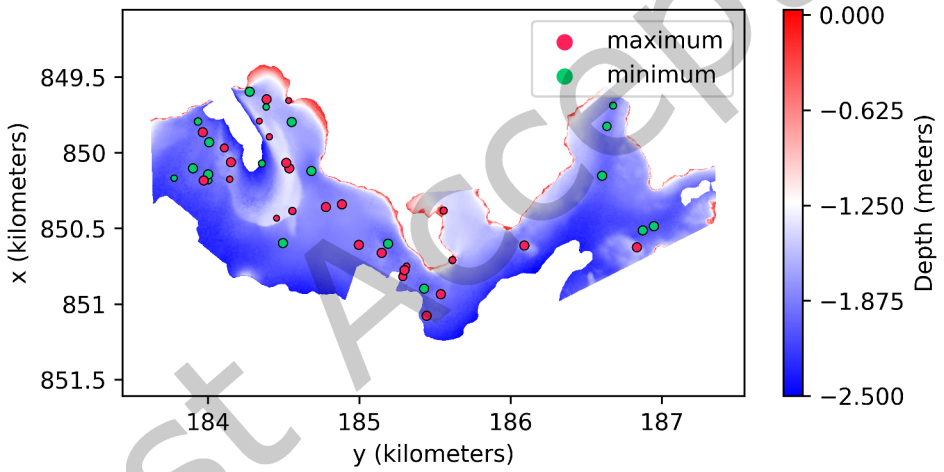


Fig. 17. Maxima and minima from the bathymetry dataset of Swanquarter Bay with life span values larger than 6. Circle size represents the life span value of each critical point and can be used as a metric of importance.

8 Conclusion

We presented a scale-space pipeline on TINs to identify and track critical points, leveraging the variable-resolution representation of TINs for irregularly distributed point data. Compared to gridded DEMs, TINs provide an adaptive discretization that represents terrain directly from the input points, which reduces redundant vertices in flat regions and mitigates spurious critical points and unnecessary events during tracking.

Our pipeline combines three key components. First, we approximate Gaussian smoothing on a TIN using only local neighborhoods, which enables an efficient and scalable construction of the scale space. Second, we improve robustness to irregular vertex spacing and uneven neighbor distribution through lightweight, GPU-friendly

strategies, including virtual neighbors and angle-based re-weighting, leading to better smoothing accuracy and more reliable critical feature extraction. Third, we introduce a curvature-guided down-sampling strategy that allocates samples according to multi-scale surface complexity while avoiding patch boundary artifacts, improving geometric fidelity, and preserving topological structure under aggressive reductions.

To make the overall workflow practical on large datasets, we also designed a fully parallel critical point tracking algorithm on TINs. By organizing the computation into edge-parallel and vertex-parallel kernels, the pipeline avoids costly global sorting and achieves nearly linear parallel cost with predictable scaling in runtime and memory on GPUs.

Although in this paper we focused on applying our pipeline to the selection of spot heights, this pipeline can be applied to other fields where topological features of a scalar field are of interest. As future work, we plan to extend the pipeline to additional application domains, explore alternative TIN smoothing operators and parameterizations that may further improve critical feature stability, and investigate more parallel-friendly data structures to reduce the GPU memory footprint.

Acknowledgments

This work has been partially supported by the US National Science Foundation under grant numbers IIS-1910766 and IIS-2114451. This work was completed while Yunting Song was at the University of Maryland, College Park.

References

[1] Tinghua Ai and Jingzhong Li. 2010. The lifespan model of GIS data representation over scale space. *Geomatics and Information Science of Wuhan University* 35, 7 (2010), 757–762.

[2] Nataraj Akkiraju, Herbert Edelsbrunner, Michael Facello, Ping Fu, EP Mucke, and Carlos Varela. 1995. Alpha shapes: definition and software. In *Proceedings of the 1st international computational geometry software workshop*. 63–66.

[3] Alexandros Avdis, Adam S. Candy, Jon Hill, Stephan C. Kramer, and Matthew D. Piggott. 2018. Efficient unstructured mesh generation for marine renewable energy applications. *Renewable Energy* 116 (2018), 842–856. doi:10.1016/j.renene.2017.09.058

[4] Thomas F. Banchoff. 1970. Critical points and curvature for embedded polyhedral surfaces. *The American Mathematical Monthly* 77, 5 (1970), 475–485.

[5] Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. *SIGGRAPH sketches* 10, 1 (2007), 1.

[6] Adam S. Candy, Alexandros Avdis, Jon Hill, Gerard Gorman, and Matthew D. Piggott. 2014. Integration of Geographic Information System frameworks into domain discretisation and meshing processes for geophysical models. *Geoscientific Model Development Discussions* 7 (2014), 5993–6060.

[7] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. 2001. *Parallel programming in OpenMP*. Morgan kaufmann.

[8] PDAL Contributors. 2022. PDAL Point Data Abstraction Library. doi:10.5281/zenodo.2616780

[9] Noel Dyer, Christos Kastrisios, and Leila De Floriani. 2022. Label-based generalization of bathymetry data for hydrographic sounding selection. *Cartography and Geographic Information Science* 49, 4 (2022), 338–353.

[10] Herbert Edelsbrunner, John Harer, and Afra Zomorodian. 2001. Hierarchical Morse complexes for piecewise linear 2-manifolds. In *Proceedings of the seventeenth annual symposium on Computational geometry*. 70–79.

[11] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. 1997. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing* 6, 9 (1997), 1305–1315.

[12] Riccardo Fellegara, Federico Iuricich, Yunting Song, and Leila De Floriani. 2023. Terrain trees: a framework for representing, analyzing and visualizing triangulated terrains. *GeoInformatica* 27, 3 (2023), 525–564.

[13] Haoan Feng, Yunting Song, and Leila De Floriani. 2024. Critical Features Tracking on Triangulated Irregular Networks by a Scale-Space Method. In *Proceedings of the 32nd ACM International Conference on Advances in Geographic Information Systems* (Atlanta, GA, USA) *(SIGSPATIAL '24)*. Association for Computing Machinery, New York, NY, USA, 54–66. doi:10.1145/3678717.3691218

[14] Steven Fortune. 2017. Voronoi diagrams and Delaunay triangulations. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 705–721.

[15] Rafael C Gonzalez. 2009. *Digital image processing*. Pearson education india.

[16] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. 1988. Design of an image edge detection filter using the Sobel operator. *IEEE Journal of solid-state circuits* 23, 2 (1988), 358–367.

[17] Jan J Koenderink. 1984. The structure of images. *Biological cybernetics* 50, 5 (1984), 363–370.

[18] Yue Kong. 2016. An extraction method for main ridge lines based on gaussian scale space. *Computer Engineering & Science* 38, 06 (2016), 1207.
[19] Marcos Lage, Thomas Lewiner, Hélio Lopes, and Luiz Velho. 2005. CHF: a scalable topological data structure for tetrahedral meshes. In *XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)*. IEEE, 349–356.
[20] Jhe-Syuan Lai and Fuan Tsai. 2019. Improving GIS-based landslide susceptibility assessments with multi-temporal remote sensing and machine learning. *Sensors* 19, 17 (2019), 3717.
[21] Peter Lancaster and Kes Salkauskas. 1981. Surfaces generated by moving least squares methods. *Mathematics of computation* 37, 155 (1981), 141–158.
[22] Mengyuan Li, Anmin Zhang, Dianjun Zhang, Mingwei Di, and Qingju Liu. 2021. Automatic Sounding Generalization Maintaining the Characteristics of Submarine Topography. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), 10278–10286.
[23] Tie Lin, Hanchao Zhang, and Jiao Jin. 2016. The Extraction of Valley Line and Ridge Line Based on Scale Space and Multi-angle Terrain Profile. *Bulletin of Surveying and Mapping* 0, 2, Article 97 (2016), 2 pages. doi:10.13474/j.cnki.11-2246.2016.0059.
[24] Tony Lindeberg. 2013. *Scale-space theory in computer vision*. Vol. 256. Springer Science & Business Media.
[25] Ahmed H Mahmoud, Serban D Porumbescu, and John D Owens. 2021. RXMesh: a GPU mesh data structure. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.
[26] John Willard Milnor. 1963. *Morse theory*. Number 51. Princeton university press.
[27] Delio Mugnolo. 2007. Gaussian estimates for a heat equation on a network. *Networks and Heterogeneous Media* 2, 1 (2007), 55–79. doi:10.3934/nhm.2007.2.55
[28] Jan Reininghaus, Natallia Kotava, David Günther, Jens Kasten, Hans Hagen, and Ingrid Hotz. 2011. A scale space based persistence measure for critical points in 2d scalar fields. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2045–2052.
[29] Stephen E Reutebuch, Hans-Erik Andersen, and Robert J McGaughey. 2005. Light detection and ranging (LIDAR): an emerging tool for multiple resource inventory. *Journal of forestry* 103, 6 (2005), 286–292.
[30] Luigi Rocca, Bernhard Jenny, and Enrico Puppo. 2017. A continuous scale-space method for the automated placement of spot heights on maps. *Computers & Geosciences* 109 (2017), 216–227.
[31] Luigi Rocca and Enrico Puppo. 2013. A virtually continuous representation of the deep structure of scale-space. In *Image Analysis and Processing–ICIAP 2013: 17th International Conference, Naples, Italy, September 9-13, 2013, Proceedings, Part II 17*. Springer, 522–531.
[32] Jarek Rossignac, Alla Safonova, and Andrzej Szymczak. 2003. Edgebreaker on a Corner Table: A simple technique for representing and compressing triangulated surfaces. In *Hierarchical and geometrical methods in scientific visualization*. Springer, 41–50.
[33] Andriani Skopeliti, Leda Stamou, Lysandros Tsoulos, and Shachak Pe'eri. 2020. Generalization of soundings across scales: From DTM to harbour and approach nautical charts. *ISPRS International Journal of Geo-Information* 9, 11 (2020), 693.
[34] Swisstopo. 2023. swissSURFACE3D datasets. https://www.swisstopo.admin.ch/en/geodata/height/surface3d.html. Accessed: 2023-06-14.
[35] Douglas Brent West. 2001. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River.
[36] Andrew P Witkin. 1987. Scale-space filtering. In *Readings in computer vision*. Elsevier, 329–332.

Received 9 December 2025; revised 11 March 2026; accepted 20 April 2026