Efficient Proximity Queries on Simplified Height Maps

YINZHAO YAN, The Hong Kong University of Science and Technology, Hong Kong
RAYMOND CHI-WING WONG, The Hong Kong University of Science and Technology, Hong Kong

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry. The height map is one fundamental 3D surface representation with many advantages over others such as the point cloud and Triangular-Irregular Network (*TIN*). In this paper, we study the shortest path query on a height map. Since performing proximity queries using the shortest path on a height map is costly, we propose a simplification algorithm on the height map to accelerate it. We also propose a shortest path query algorithm and algorithms for answering proximity queries on the original/simplified height map. Our experiments show that our simplification algorithm is up to 21 times and 5 times (resp. 412 times and 7 times) better than the best-known adapted point cloud (resp. *TIN*) simplification algorithm in terms of the simplification time and output size (the size of the simplified surface), respectively. Performing proximity queries on our simplified height map is up to 5 times and 1,340 times quicker than on the simplified point cloud and the simplified *TIN* with an error at most 10%, respectively.

CCS Concepts: • **Information systems → Proximity search**.

Additional Key Words and Phrases: spatial database; proximity queries; height maps; simplification

ACM Reference Format:
Yinzhao Yan and Raymond Chi-Wing Wong. 2025. Efficient Proximity Queries on Simplified Height Maps. *Proc. ACM Manag. Data* 3, 4 (SIGMOD), Article 244 (September 2025), 26 pages. https://doi.org/10.1145/3749162

1 Introduction

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry [65, 72]. Academic researchers studied different types of proximity queries [31, 32, 51, 59, 65, 68, 69, 72], including *shortest path queries* [28, 44, 45, 49, 50, 54, 55, 64–67, 70–72, 74], *k-Nearest Neighbor* (*kNN*) *queries* [31, 32, 59, 65, 68] and *range queries* [51, 61]. In industry, Google Earth [9] and Metaverse [16] employ shortest paths passing on 3D surfaces (e.g., Earth and virtual reality) for user navigation.

Height map, point cloud and *TIN*: There are different representations of a 3D surface, including *height map*, *point cloud* [72] and *Triangular-Irregular Network* (*TIN*) [65, 68, 69]. Figure 1 (a) shows a 3D surface in a 20km × 20km region in Gates of the Arctic [57] national park, USA. Figure 1 (b) shows the height map representation of this surface. Consider a 2D plane with 9 × 9 grid cells in this region. Each cell has *2D coordinate values* representing *2D coordinate values* of its center point, and a *grayscale pixel color* representing its *elevation value* (e.g., calculated using a simple linear interpolation using the pixel color), meaning the height projected from this center point on the 3D surface. If this value is larger, this pixel's color is brighter. Besides, each cell has 8 neighbors, shown as blue points in Figure 1 (b). All these cells form a height map. Figure 1 (c) shows this height map in bird's eye view. Figure 1 (d) shows the point cloud representation of this surface. Each cell in the

Authors' Contact Information: Yinzhao Yan, The Hong Kong University of Science and Technology, Hong Kong, yyanas@cse.ust.hk; Raymond Chi-Wing Wong, The Hong Kong University of Science and Technology, Hong Kong, raywong@cse.ust.hk.

ACM 2836-6573/2025/9-ART244
https://doi.org/10.1145/3749162

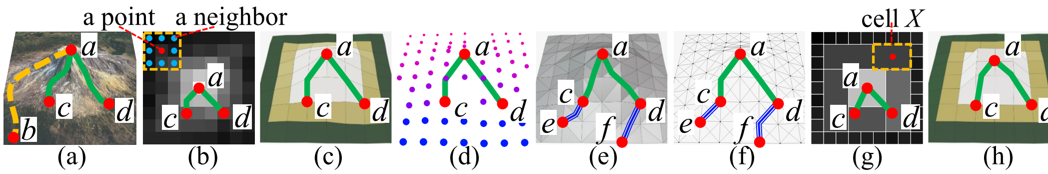

Fig. 1. (a) A 3D surface, (b) a height map, (c) a height map in bird's eye view, (d) a point cloud, (e) a *TIN*, (f) a height map graph, (g) a simplified height map and (h) a simplified height map in bird's eye view

height map could be *one-to-one* mapped to a 3D point, where the x- and y- coordinate values of this point are the 2D coordinate values of the center point of this cell, and the z-coordinate of this point is the elevation value of this cell [26, 47, 63, 77]. This is the best-known exact height map to point cloud conversion algorithm that runs in $O(n)$ time, where n is the number of cells in the height map. The best-known approximate conversion algorithm uses machine learning approach, e.g., uniform random sampling [38, 48] for acceleration. But, the converted point cloud is an approximated representation of the height map without a bound guarantee, since it *randomly* selects *some* (not all) cells for mapping. This runs in $O(n_r)$ time, where n_r is the number of randomly selected cells. Figure 1 (e) shows the *TIN* representation of this surface. A *TIN* has a set of contiguous triangulated *faces*, where each face has three *edges* connecting at three *vertices*. In practice, the *TIN* is converted from the point cloud [72] via *triangulation* [35, 62, 72] where all vertices of faces are the points in the point cloud. This runs in $O(n)$ time. If the triangulation is applied on the approximated point cloud, this runs in $O(n_r)$ time.

1.1 Advantages of Height Map

Height maps offer several advantages over point clouds and *TIN*s.

(1) Compared with point cloud datasets, there are more height map datasets available (e.g., there are 50M height map datasets but only 20M point cloud datasets in an open data 3D surface dataset platform called OpenDEM [17]), with four reasons.

(i) *Longer history of the height map.* The height map and point cloud were introduced in 1884 [2] and 1960 [13], respectively. So, more height map datasets are available due to the earlier adoption.

(ii) *Lower cost of obtaining a height map dataset.* The height map dataset could be obtained from either *optical* images of cost USD \$25 [21] captured by an *optical* satellite or *radar* images of cost USD \$3,300 [8] captured by a *radar* satellite. But, the point cloud dataset could be obtained only from *radar* images captured by a *radar* satellite (if no conversion operation from the height map to the point cloud is involved). The image cost difference is due to the satellite launching cost difference: USD \$0.4 billion [14] for an optical satellite and USD \$1.5 billion [1] for a radar satellite.

(iii) *More region coverage of the height map datasets.* Since optical and radar satellites cover 100% [3] and 80% [20] of Earth's land area, respectively, height map datasets cover more regions compared with point cloud datasets. For example, high-latitude regions (e.g., Gates of the Arctic national park in the northern part of Alaska, USA) are regions covered by height map datasets but not point cloud datasets [22]. We perform a snowfall evacuation case study there, using the only available height map datasets for evacuation.

(iv) *Additional conversion time from height map datasets to point cloud datasets.* In our experiment, converting height map datasets to point cloud datasets [26, 47, 63, 77] for radar satellites' uncovered region takes 21 years[1]. It can be fast (e.g., 42s for a region of 1km^2) for a small area and we need it once. But, in our evacuation case study, we capture the height map (in only 4s for a region of 1 km^2 [53]) *after* snowfall due to avalanches (i.e., the height map is updated), and the weather changes suddenly (that complicates rescue efforts) in 1s [4]. We aim to avoid the conversion to minimize sudden weather changes and save more lives.

(2) Compared with *TIN* datasets (i.e., usually converted from point cloud datasets), height map datasets are *easier to access* since satellites can capture them directly. So, more height map datasets are available, and the 4 reasons above also apply to *TIN*s. Height maps also *use less hard disk space*, since they store cell information, while *TIN*s store vertex, edge and face information.

1.2 Our Focus

1.2.1 ***Height map shortest path query.*** In this paper, we study the shortest path query on the height map. There are two issues.

(1) *Finding the shortest path passing on the height map.* There is no existing study finding the shortest path *directly* on a height map. Most (if not all) algorithms [47, 56, 63, 77] adapt shortest path algorithms on the point cloud [72] or *TIN* [28, 44–46, 50, 66, 67, 71, 74] by converting the height map to point cloud and *TIN*, and then perform the shortest path query on the converted 3D surfaces. We propose a *height map graph* in Figure 1 (f). For each cell in the height map, we construct a corresponding 3D vertex in the graph. For each pair of neighboring cells, we create an edge between their corresponding vertices with a weight equal to the Euclidean distance between them. Based on this graph, we could find the shortest path by Dijkstra's algorithm [33]. Our experiments show that computing the shortest path passing on a height map with 0.5M cells needs 3s, but computing the shortest path passing on a point cloud (see Figure 1 (d)) converted from this height map [72] needs 3.4s due to data conversion. Besides, computing the shortest surface path passing on a *TIN* (see Figure 1 (e)) converted from this height map [28, 66, 74] needs 280s $\approx$ 4.6 min, since the height map's structure is simpler.

(2) *Improving in other proximity queries.* In our experiments, using shortest paths to answer *kNN* or range query for 10k query objects on a height map with 50k cells need 4,400s $\approx$ 1.2 hours, i.e., very long. Thus, we propose a *simplification* process on the height map.

1.2.2 ***Height map simplification.*** In this paper, we also study how to simplify the height map. If we merge *nearby* cells with similar elevation values (with *redundant* information) into one cell, then the number of cells is reduced and Dijkstra's algorithm on this simplified height map is faster. Figure 1 (g) shows a simplified height map of the same surface, where cell X is merged from 6 cells (whose elevation value is the average of these 6 cells). Figure 1 (h) shows this simplified height map in bird's eye view. Consider a pair of points a and c. There is a relative error called the *distance error ratio* of the distance calculated by a studied algorithm compared with the ground-truth or optimal distance, i.e., the *approximate* shortest distance between a and c on the simplified height map in Figure 1 (g) compared with the (*exact*) shortest distance between a and c on the original height map in Figure 1 (b).

Given an error parameter $\epsilon \in [0, 1]$, we study how to simplify the height map so that the distance error ratio for each pair of points on the original height map is at most ϵ. There are two challenges.

(1) *Simplifying the height map with a small size efficiently.* There is no existing study focusing on simplifying a height map. The only closely related work are the simplification algorithms on the point cloud [24, 72] or *TIN* [32, 40, 45, 49]. We adapt them by converting the height

[1]Since the total Earth's land area is 149M km^2 [7], the total areas covered by optical satellites but not radar satellites are 30M km^2 (i.e., 149M km^2 $\times$ (100% - 80%)). In our experiment, converting a height map dataset in a region of 1 km^2 (with 3m $\times$ 3m resolution) to a point cloud dataset takes 42s. Thus, the conversion time is 42s/km^2$\times$ 30M km^2= 1.26 $\times 10^9$s $\approx$ 21 years.

map to point cloud and *TIN*, and then performing the original simplification algorithms on the converted 3D surfaces. But, the size of the simplified point cloud and *TIN* are large since they lack optimization techniques, resulting in large shortest path query time on the simplified 3D surfaces. The simplification time of the point cloud and *TIN* simplification algorithms are large since they lack pruning techniques and *TIN* simplification algorithms involve expensive *TIN re-triangulation* [35, 62, 72].

(2) *Defining the neighborhoods of cells in the simplified height map.* In the original height map (Figure 1 (b)), it is clear to understand the neighborhoods of each cell. But, in the simplified height map (Figure 1 (g)), since each *merged* cell can be adjacent to many different cells, we need to define clearly neighborhoods of each (merged/non-merged) cell for the shortest path query.

1.3 Contribution and Organization

We summarize our contributions as follows.

(1) We are the first to study the shortest path query directly on the height map. We also adopt a height map simplification process so that the distance error ratio for each pair of points on the original height map is at most ϵ. We show that this process is *NP-hard*.

(2) We propose an ϵ-approximate height map simplification algorithm called *Height Map Simplification Algorithm* (*HM-Simplify*). It can significantly reduce the number of cells of the simplified height map, i.e., reduce the output size (the size of the simplified height map), to further reduce the shortest path query time on the simplified height map using a novel cell merging technique (by considering cell information of height maps) for optimization. It can also efficiently reduce the simplification time using the novel cell merging technique and an efficient checking technique during simplification (by considering neighbor information of height maps) for pruning. We also propose a shortest path query algorithm called *Height Map Shortest Path Query Algorithm* (*HM-SP*) on the original/simplified height map. It can efficiently reduce the shortest path query time on the simplified height map using an efficient implicit edge insertion technique (by considering neighbor information of height maps and the single-source-all-destination feature of Dijkstra's algorithm) for pruning. We also design algorithms for answering *kNN* and range queries on the original/simplified height map. It can also efficiently reduce the proximity query time on the original/simplified height map using an efficient parallel computation technique (by considering the single-source-all-destination feature of Dijkstra's algorithm) for pruning.

(3) We give theoretical analysis on (i) algorithm *HM-Simplify*'s simplification time, the number of cells in the simplified height map, output size, simplification memory and error guarantee, and (ii) algorithm *HM-SP* and proximity query algorithms' query time, memory and error guarantee.

(4) Algorithm *HM-Simplify* outperforms the best-known adapted point cloud [24, 72] and *TIN* [42, 45] simplification algorithm concerning the simplification time and output size. Performing proximity queries on the simplified height map is much quicker than the best-known algorithms [28, 66, 72, 74] on the simplified point cloud and the simplified *TIN*. Our experiments show that given a height map with 50k cells, the simplification time and output size are 250s $\approx$ 4.6 min and 0.07MB for algorithm *HM-Simplify*, but are 5,250s $\approx$ 1.5 hours and 0.35MB for the best-known adapted point cloud simplification algorithm [24, 72], and 103,000s $\approx$ 1.2 days and 0.5MB for the best-known adapted *TIN* simplification algorithm [42, 45]. The proximity query time of 10k objects is 50s on the simplified height map, 250s $\approx$ 4.2 min on the simplified point cloud and 67,000s $\approx$ 18.6 hours on the simplified *TIN*.

The remainder of the paper is organized as follows. Section 2 gives the problem definition. Section 3 covers the related work. Section 4 presents our algorithms. Section 5 discusses the experimental results and Section 6 concludes the paper.

2 Problem Definition

2.1 Notation and Definitions

2.1.1 ***Height map.*** Consider a height map $H = (C, N(\cdot))$ on a 2D plane containing a set of *cells* C with size n, and a *neighbor cells* (*hash*) table [30] $N(\cdot)$. In H, each cell $c \in C$ has *2D coordinate values* (representing *2D coordinate values* of its center point) and a *grayscale pixel color* (representing its *elevation value*), denoted as $c.x$, $c.y$ and $c.z$, respectively. Given cell $c \in C$, $N(c)$ returns c's *neighbor cells* in $O(1)$ time, and it is initialized to be c's nearest 8 surrounding cells on H. Figure 2 (a) shows a height map with 9 cells. For point p on cell c, 6 orange and 2 red points form $N(c)$.

We define G to be the *height map graph* of H. For each cell $c \in C$, we create a vertex v_c in G whose x-, y- and z-coordinate values are defined to $c.x$, $c.y$ and $c.z$, respectively. For each cell $c \in C$ and each cell $c' \in N(c)$, we create an edge between vertex v_c and vertex $v_{c'}$ in G (corresponding to c and c') with a weight equal to the Euclidean distance between v_c and $v_{c'}$, and c and c' are said to be *adjacent*. The graphs in Figures 2 (a) and (b) are G on the 2D plane and in a 3D space. Given a pair of points s and t on H, let $\Pi(s, t|H)$ be the (*exact*) *shortest path* between them passing on (G of) H. Let $|\cdot|$ be a path's distance (e.g., $|\Pi(s, t|H)|$ means $\Pi(s, t|H)$'s distance). Figures 2 (a) and (b) show $\Pi(s, t|H)$ in green line.

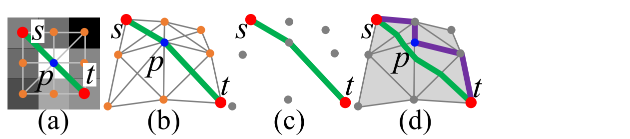

Fig. 2. (a) A height map, (b) a height map graph, (c) a point cloud and (d) a *TIN*

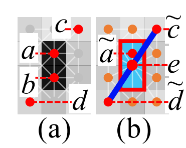

Fig. 3. Cell merging

2.1.2 ***Simplified height map.*** Given H, we can obtain a simplified height map $\widetilde{H} = (\widetilde{C}, \widetilde{N}(\cdot))$ by merging some adjacent cells (deleting these cells and adding a new *larger* cell covering these cells) in H. $\widetilde{C}$ and $\widetilde{N}(\cdot)$ are initialized as C and $N(\cdot)$, and are updated during simplification. Figures 3 (a) and (b) show H and $\widetilde{H}$, where the blue cell in $\widetilde{H}$ is merged from 2 cells in H.

A cell in H that is deleted from (resp. remaining in) $\widetilde{H}$ is referred as a *deleted* (resp. *remaining*) cell. A cell in $\widetilde{H}$ that covers some adjacent deleted and/or previously added cells is referred as an *added* cell. These adjacent deleted cells *belong* to the added cell. A *property of a deleted cell* is that each deleted cell only belongs to one added cell. In Figures 3 (a) and (b), we merge cells a and b to cell e, 10 orange and red points (around e) form all cells in $\widetilde{N}(e)$, $\{a, b\}$ are deleted cells, all other cells in C except $\{a, b\}$ are remaining cells, e is an added cell, and $\{a, b\}$ belong to e. The coordinate and elevation values of the added cell are weighted averages of those of the adjacent deleted cells (if these adjacent deleted cells contain a previously added cell c, the weight is the number of cells in H belonging to c; otherwise, the weight is 1). In Figures 3 (b), we use the coordinate and elevation values of a, b with weight equal to 1 to calculate those values for e. If we keep merging e with other cells, the weight of e is 2, since the number of cells in H belonging to e is 2. We denote a set of remaining cells and added cells as C_{rema} and C_{add}, so $\widetilde{C} = C_{rema} \cup C_{add}$. A set of deleted cells is denoted as $C - C_{rema}$. Given a cell $c \in H$, we define $\widetilde{c}$ be the *estimated cell* of c (on $\widetilde{H}$). In Figures 3 (a) and (b), we have a and $\widetilde{a}$.

Similar to G, let $\widetilde{G}$ be the simplified height map graph of $\widetilde{H}$. We use $\widetilde{C}$ and $\widetilde{N}(\cdot)$ in the definition of G to obtain $\widetilde{G}$'s vertices and edges. The graphs in Figures 3 (a) and (b) are G and $\widetilde{G}$, respectively. Given a pair of points $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{H}$, let $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ be the *approximate shortest path* between them passing on ($\widetilde{G}$ of) $\widetilde{H}$. Figure 3 (b) shows $\Pi(\widetilde{c}, \widetilde{d}|\widetilde{H})$ in blue line. A notation table can be found in our technical report [73].

2.2 Problem

We introduce the concept of *ϵ-approximate simplified height map* in Definition 1 to describe that $\widetilde{H}$ guarantees that for each pair of points on H, their distance error ratio is at most ϵ.

Definition 1 (ϵ-Approximate Simplified Height Map Definition). *Given H, $\widetilde{H}$ and ϵ, $\widetilde{H}$ is said to be an ϵ-approximate simplified height map of H (or $\widetilde{H}$ is said to be an ϵ-approximation of H) if and only if for each pair of points s and t on H,*

$$(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|. \tag{1}$$

We have the following problem, which is *NP-hard*.

Problem 1 (Height Map Simplification Problem). *Given H and ϵ, we want to find an ϵ-approximate simplified height map $\widetilde{H}$ of H with the minimum number of cells.*

Theorem 2.1. *The height map simplification problem is NP-hard.*

Proof Sketch. We transform Minimum T-Spanner Problem [27] (*NP-complete*) to our problem in polynomial time for proving. The detailed proof appears in our technical report [73]. □

3 Related Work

3.1 Point Cloud and *TIN*

Let P be a point cloud converted from H by cell mapping [26, 47, 63, 77], and T be a *TIN* converted from P by point triangulation [35, 62, 72]. Given a pair of points s and t on P, let $\Pi(s,t|P)$ be the *shortest path* between them passing on (*point cloud graph* [72] of) P. The height map graph and point cloud graph are the same. Given a pair of vertices s and t on T, let $\Pi(s,t|T)$ and $\Pi_N(s,t|T)$ be the *shortest surface path* [45] (passing on faces) of T and *shortest network path* [45] (passing on edges) of T between them. Their distances are called the *shortest surface* and *network distance*, respectively. Let θ be the smallest interior angle of a triangle of T. Figure 2 (c) shows a P with $\Pi(s,t|P)$ in green line, and Figure 2 (d) shows a T with $\Pi(s,t|T)$ in green line and $\Pi_N(s,t|T)$ in purple line.

Given a pair of points s and t on H, since the height map graph is the same as the point cloud graph, we know $|\Pi(s,t|H)| = |\Pi(s,t|P)|$. According to Lemma 4.3 of study [72], we know $|\Pi(s,t|H)| \leq \alpha \cdot |\Pi(s,t|T)|$, where $\alpha = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$, and $|\Pi(s,t|H)| \leq |\Pi_N(s,t|T)|$. But, $|\Pi(s,t|H)|$ can be larger or smaller than $|\Pi(s,t|T)|$. In Figures 1 (e) and (f) (see blue lines), $|\Pi(c,e|T)| > |\Pi(c,e|H)|$, but $|\Pi(d,f|T)| < |\Pi(d,f|H)|$.

3.2 Height Map Shortest Path Query Algorithms

There is no existing study finding the shortest path *directly* on a height map. Existing studies [47, 56, 63, 77] adapt shortest path algorithms on the point cloud [72] or *TIN* [28, 44–46, 50, 66, 67, 71, 74] by converting the height map to a point cloud or a *TIN*, and then computing the shortest path passing on the converted 3D surfaces (by defining their 3D surfaces first, e.g., point clouds and *TIN*s, and find paths under their 3D surfaces).

3.2.1 ***Point cloud shortest path query algorithm.*** The best-known exact point cloud shortest path query algorithm called *Point Cloud Shortest Path Query Algorithm* (*PC-SP*) [72] uses Dijkstra's algorithm on the point cloud graph for querying in $O(n \log n)$ time.

3.2.2 ***TIN shortest surface path query algorithms.*** (1) *Exact algorithms*: Two studies use continuous Dijkstra's [50] and checking window [67] algorithms for querying both in $O(n^2 \log n)$ time. The best-known exact *TIN* shortest surface path query algorithm called *TIN Exact Shortest*

Surface Path Query Algorithm (*TIN-ESSP*) [28, 66, 74] uses a line to connect the source and destination on a 2D *TIN* unfolded by the 3D *TIN*, for querying in $O(n^2)$ time.

(2) *Approximate algorithms*: All algorithms [44, 46, 71] use discrete Steiner points to construct a graph and use Dijkstra's algorithm for querying. The best-known $(1+\epsilon)$-approximate *TIN* shortest surface path query algorithm called *TIN Approximate Shortest Surface Path Query Algorithm* (*TIN-ASSP*) [44, 71] runs in $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$ time, where l_{max}/l_{min} are the longest/shortest edge's length of the *TIN*, respectively.

3.2.3 ***TIN shortest network path query algorithm***. Network paths are surface paths restricted to *TIN*'s edge without traversing the faces, resulting in an approximate path. The best-known approximate *TIN* shortest network path query algorithm called *TIN Shortest Network Path Query Algorithm* (*TIN-SNP*) [45] uses Dijkstra's algorithm on *TIN*'s edge for querying in $O(n\log n)$ time.

Adaptations: (1) Given a *Height Map*, we adapt these four algorithms to be algorithms *PC-SP-Adapt(HM)* [72], *TIN-ESSP-Adapt(HM)* [28, 66, 74], *TIN-ASSP-Adapt(HM)* [44, 71] and *TIN-SNP-Adapt(HM)* [45], by converting the height map to a point cloud or a *TIN*, and then computing the shortest path passing on the point cloud or *TIN*. (2) Given a height map without data conversion, algorithm *TIN-ESSP* cannot be directly adapted to the height map since no face can be unfolded in a height map. But, algorithms *PC-SP*, *TIN-ASSP* and *TIN-SNP* can be directly adapted to the height map (using a height map graph), and they become algorithm *HM-SP* (since they are Dijkstra's algorithms).

Drawback: All algorithms are very slow. Our experiments show that for a height map with 50k cells, answering *kNN* queries for all 10k objects needs 4,400s $\approx$ 1.2 hours, 380,000s $\approx$ 4.3 days, 70,000s $\approx$ 19.4 hours and 33,000s $\approx$ 9.2 hours for algorithms *PC-SP-Adapt(HM)*, *TIN-ESSP-Adapt(HM)*, *TIN-ASSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)*, respectively.

3.3 Height Map Simplification Algorithms

There is no existing study focusing on simplifying a height map. The only closely related work are simplification algorithms on the point cloud [24, 72] or *TIN* [32, 40, 45, 49]. They iteratively remove a point in a point cloud, or remove a vertex v in a *TIN* and use *triangulation* [35, 62, 72] to form new faces among the previous adjacent vertices of v. *Point Cloud Simplification Algorithm* (*PC-Simplify*) [24, 72] is the best-known point cloud simplification algorithm. *TIN shortest Network distance Simplification Algorithm* (*TIN-NSimplify*) [45] is the most efficient *TIN* simplification algorithm. By using shortest *surface* distances, we obtain the best-known *TIN* simplification algorithm *TIN shortest Surface distance Simplification Algorithm* (*TIN-SSimplify*) [42, 45]. We adapt them by converting the height map to point cloud and *TIN*, and then performing them for simplification on the converted 3D surfaces.

3.3.1 ***Algorithm PC-Simplify***. Study [24] finds a *TIN* with a minimum number of vertices without *TIN* triangulation, it indeed is a point cloud simplification algorithm. But, each point cloud simplification iteration checks whether the z-coordinate value difference of each point on the original and simplified point cloud is at most ϵ. We adapt it by retaining its simplification process and constructing a point cloud graph [72], so when it removes a point p, we remove p's adjacent edges in the graph, and check for *each* pair of points (we simplify to "each pair of previous *adjacent* points of p"), whether the relative error of the shortest distance between them on the simplified and original point cloud is at most ϵ. Its simplification time is $O(n^2\log n)$ and output size is $O(n)$.

3.3.2 ***Algorithm TIN-NSimplify***. Each *TIN* simplification iteration checks for "each pair of vertices" on the original *TIN*, whether the relative error of the shortest network distance between them on the simplified and original *TIN* is at most ϵ. We simplify to "each pair of previous *adjacent* vertices of the removed vertex v". Its simplification time is $O(n^2 \log n)$ and output size is $O(n)$.

3.3.3 ***Algorithm TIN-SSimplify.*** Similarly, we simplify to "*arbitrary* pair of points[2] on faces including previous *adjacent* vertices of v". We further simplify it by placing Steiner points on these faces (using any-to-any points *TIN* shortest surface path query technique [42]), and check related to "each pair of Steiner points". Its simplification time is $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$ and output size is $O(n)$.

Adaptations: (1) Given a *Height Map*, we adapt these three algorithms to be algorithms *PC-Simplify-Adapt(HM)* [24, 72], *TIN-NSimplify-Adapt(HM)* [45] and *TIN-SSimplify-Adapt(HM)* [42, 45], by converting the height map to a point cloud or a *TIN*, and then applying the corresponding algorithms for point cloud or *TIN* simplification. (2) Given a height map without data conversion, algorithm *PC-Simplify* can be directly adapted to the height map (using a height map graph), and it performs the same as algorithm *PC-Simplify* on point cloud (which has a large simplification time since it lacks pruning techniques). But, algorithms *TIN-NSimplify* and *TIN-SSimplify* cannot, since no vertices can be deleted and no new faces can be created in a height map.

Drawbacks: (1) *Large output size*: All algorithms lack optimization techniques, so their simplified point cloud and *TIN* have a large size, resulting in large shortest path query time on the simplified 3D surfaces. This is because they only *remove* points or vertices without *adding* new ones, so the simplified 3D surfaces differs a lot from the original one, limiting further simplification. (2) *Large simplification time*: All algorithms lack pruning techniques, resulting in a large simplification time. This is because they remove only one point or vertex per iteration, yielding many distance checking iterations. They cannot remove many points or vertices at once; otherwise the simplified point cloud or *TIN* changes a lot. In addition, algorithms *TIN-NSimplify-Adapt(HM)* and *TIN-SSimplify-Adapt(HM)* involve expensive *TIN re-triangulation*, so their simplification time is even larger. Our experiments show that for a height map with 50k cells, the simplification time of algorithms *PC-Simplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)*, *TIN-SSimplify-Adapt(HM)* and *HM-Simplify* (ours) are 5,250s $\approx$ 1.5 hours, 7,100s $\approx$ 2 hours, 103,000s $\approx$ 1.2 days and 250s $\approx$ 4.6 min, respectively. The *kNN* query time of 10k objects on the simplified point cloud, *TIN*, or height map are 250s $\approx$ 4.2 min, 16,800s $\approx$ 4.7 hours, 67,000s $\approx$ 18.6 hours and 50s, respectively.

Solution: If they add new points or vertices by following our novel cell merging technique, their output size is similar to ours. Then, they can remove more than one point or vertex per iteration, and the simplification time of algorithm *PC-Simplify-Adapt(HM)* is similar to ours, but the other two remain high due to expensive *re-triangulation*. Thus, all algorithms perform worse than ours in terms of output size and simplification time.

4 Methodology

4.1 Overview

4.1.1 ***Two phases.*** There are two phases for our framework.

(1) **Simplification phase using algorithm *HM-Simplify***: In Figure 4 and Figures 5 (a) - (f), given H, we generate $\widetilde{H}$ by iteratively cell merging whenever $\widetilde{H}$ is an ϵ-approximation of H (H is deleted).

(2) **Shortest path query phase using algorithm *HM-SP***: In Figure 4 and Figure 5 (g), given $\widetilde{H}$, a pair of points s and t on H, we first calculate s and t's estimated points $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{H}$, and then use Dijkstra's algorithm [33] on $\widetilde{H}$ to compute $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$.

[2] Given a pair of vertices far away from v, the shortest surface path between them may pass on faces including previous adjacent vertices of v but not on the previous adjacent vertices of v. So, only checking the shortest surface distance related to the latter is not sufficient.

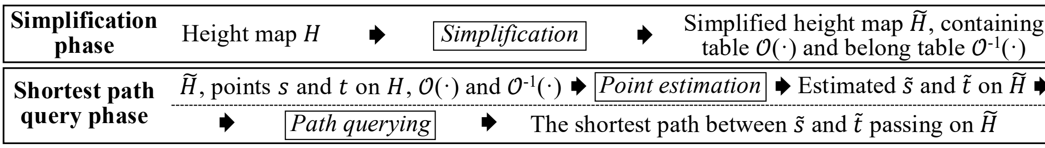

Fig. 4. Overview of algorithm *HM-Simplify* and *HM-SP*

4.1.2 ***Two components***. There are two *hash tables* involved.

(1) **The containing table $O(\cdot)$**: Given an added cell c in $\widetilde{H}$, $O(c)$ returns the set of deleted cells $\{p_1, p_2, \ldots\}$ in H belonging to c in $O(1)$ time. In Figure 5 (b), we merge $\{a, b\}$ to cell c, the deleted cells $\{a, b\}$ belongs to the added cell c, so $O(c) = \{a, b\}$.

(2) **The belonging table $O^{-1}(\cdot)$**: Given a deleted cell c in H, $O^{-1}(c)$ returns the added cell c' in $\widetilde{H}$ such that c belongs to c' in $O(1)$ time. In Figure 5 (b), $O^{-1}(a) = c$.

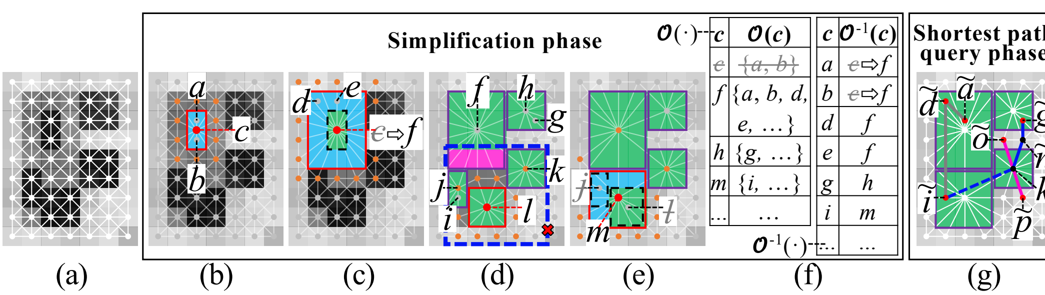

Fig. 5. Details of algorithm *HM-Simplify* and *HM-SP*

4.2 Key Idea

4.2.1 ***Significant output size reducing for algorithm HM-Simplify***. It can significantly reduce the number of cells in $\widetilde{H}$ (output size), to further reduce the shortest path query time of algorithm *HM-SP* on $\widetilde{H}$ using *a novel cell merging technique* with two merging types for optimization. Since we delete some cells and create a newly added cell during merging, $\widetilde{H}$ remains similar to H, enabling further merging. The following two types discuss which cells to delete and the position of the newly added cell.

(1) **Merge two cells**: We first choose two adjacent remaining and non-boundary (not on H's boundary) cells with the smallest elevation value variance. In Figure 5 (b), we merge $\{a, b\}$ to form c, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and go to the next type. If not, we terminate the algorithm.

(2) **Merge added cell with neighbor cells**: Given an added cell c from the previous merge, we merge c with its neighbor cells, i.e., expand c's non-boundary neighbor cells into left, right, top and/or bottom directions to reduce the number of cells in $\widetilde{H}$. Expanding left (resp. right) covers neighbors cells with x-coordinate value smaller (resp. larger) than c, and expanding top (resp. bottom) covers neighbors cells with y-coordinate value larger (resp. smaller) than c, and the origin is set at left-bottom side of H. Let Direction, i.e., $Dir = \{(L, R, T, B), (L, R, T, \cdot), (L, R, \cdot, B), (L, \cdot, T, B), (\cdot, R, T, B), (L, R, \cdot, \cdot), (L, \cdot, T, \cdot), (L, \cdot, \cdot, B), (\cdot, R, T, \cdot), (\cdot, R, \cdot, B), (\cdot, \cdot, T, B), (L, \cdot, \cdot, \cdot), (\cdot, R, \cdot, \cdot), (\cdot, \cdot, T, \cdot), (\cdot, \cdot, \cdot, B)\}$ be the expanded directions, where L, R, T, B means that we expand c into *left*, *right*, *top* and *bottom* directions, and $\cdot$ means no expansion in that direction. For example, $Dir[1] = (L, R, T, \cdot)$

means that we cover c's neighbors cells with x-coordinate value smaller and larger than c, and y-coordinate value larger than c.

In Figure 5 (c), we merge c with $\{d, e, \dots\}$ to form f, i.e., expand c into (L, R, T, B) directions, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and repeat. If not, we go back to the *two cells merging* type by selecting two new cells. In Figure 5 (d), we merge l with $\{j, k, \dots\}$ to form a potential newly added cell with a blue frame, i.e., expand l into (L, R, T, B) directions. For l's neighbor cell k, we cover it as a whole to reduce the number of cells in $\widetilde{H}$. But, four pink deleted cells will belong to both f and the potential newly added cell. This violates the property of the deleted cell in Section 2.1.2, since we do not want the potential newly added cell to overlap with any added cell f. So, we expand l into the direction of other elements in *Dir*. In Figure 5 (e), we merge l with $\{j, \dots\}$ to form m, i.e., expand l into $(L, \cdot, T, \cdot)$ directions, and obtain $\widetilde{H}$. If $\widetilde{H}$ is an ϵ-approximation of H, we confirm this merging and repeat. If not, we go back to the *two cells merging* type. Similarly, we cannot expand four green added cells to cover a in Figure 6.

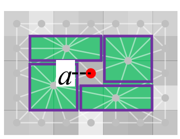

Fig. 6. No further merging

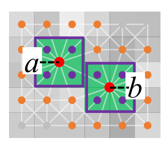

Fig. 7. Distance checking

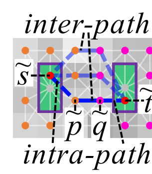

Fig. 8. Intra- and inter-paths

4.2.2 ***Efficient simplification for algorithm HM-Simplify.***

There are three reasons why it has a small simplification time.

(1) **Efficient height map shortest path query**: We use efficient algorithm *HM-SP* to check whether $\widetilde{H}$ is an ϵ-approximation of H.

(2) **Efficient simplification iteration reducing**: Due to the *novel cell merging technique*, $\widetilde{H}$ is similar to H, and we can merge more cells in one iteration to reduce iteration numbers for pruning.

(3) **Efficient ϵ-approximate simplified height map checking**: Checking whether $\widetilde{H}$ is an ϵ-approximation of H involves checking distances related *all* points on H. This naive method is time-consuming. Instead, *our efficient checking technique* only checks distances related to newly added cells' *neighbor* for pruning.

4.3 Simplification Phase

We illustrate the simplification phase using algorithm *HM-Simplify* in Algorithm 1 (showing two merging types), which uses Algorithm 2 (clearly updating the neighborhoods of each cell) twice.

4.3.1 ***Detail and example for Algorithm 1.***

In each simplification iteration, let $\widehat{C} = \{p_1, p_2, \dots\}$ be a set of adjacent cells to be merged, and c_{add} be an added cell merged from cells in $\widehat{C}$. Let *FindTwoCell* $(\widetilde{H})$ be a function that returns two adjacent remaining and non-boundary cells in $\widetilde{H}$ with the smallest elevation values variance. Let *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ be a function that returns a set of cells in $\widetilde{H}$ including c_{add} and its expanded non-boundary neighbor cells (as a whole) in $Dir[i]$ directions without violating the property of deleted cells. Both functions return *NULL* if such cells do not exist. The following shows Algorithm 1 with an example.

(1) *Merge two cells* (lines 2-5): In Figures 5 (a) and (b), $\widehat{C} = FindTwoCell(\widetilde{H}) = \{a, b\}$, and we can merge cells in $\widehat{C}$ to obtain $c_{add} = c$. Suppose that *Update* is *True*, we obtain $\widetilde{H}$.

Algorithm 1 *HM-Simplify* (H)

Input: $H = (C, N(\cdot) = \emptyset)$
Output: $\widetilde{H}$, $O(\cdot)$ and $O^{-1}(\cdot)$
1: initialize $N(\cdot)$ using C, $C_{rema} \leftarrow C$, $C_{add} \leftarrow \emptyset$, $\widetilde{N}(\cdot) \leftarrow N(\cdot)$, $O(\cdot) \leftarrow \emptyset$, $O^{-1}(\cdot) \leftarrow \emptyset$
2: $\widehat{C} \leftarrow$ *FindTwoCell* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)))$
3: **while** $\widehat{C}$ is *NON-NULL* **do**
4:     merge cells in $\widehat{C}$ to form cell c_{add}
5:     **if** *Update* $(C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$ **then**
6:         $i \leftarrow 0$
7:         **while** $i < 15$ **do**
8:             $\widehat{C} \leftarrow$ *FindAddedCellNeig* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)), c_{add}, i)$
9:             **if** $\widehat{C}$ is *NON-NULL* **then**
10:                 merge cells in $\widehat{C}$ to form cell c_{add}
11:                 **if** *Update* $(C_{rema}, C_{add}, \widetilde{N}, \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$ **then**
12:                     $i \leftarrow 0$
13:                 **else**
14:                     $i \leftarrow i + 1$
15:     $\widehat{C} \leftarrow$ *FindTwoCell* $(\widetilde{H} = (C_{rema} \cup c_{add}, \widetilde{N}(\cdot)))$
16: **return** $\widetilde{H} = (C_{rema} \cup C_{add}, \widetilde{N}(\cdot))$, $O(\cdot)$ and $O^{-1}(\cdot)$

(2) *Merge added cell with neighbor cells* (lines 6-14). In Figure 5 (c), $c_{add} = c, i = 0 < 15$, $\widehat{C} =$ *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i) = \{c, d, e, \dots\}$, i.e., we can expand c into $Dir[0] = (L, R, T, B)$ directions to form $c_{add} = f$. Suppose that *Update* is *True*, we obtain $\widetilde{H}$, and set $i = 0$. For later iterations, suppose that *Update* is always *False*, we always increase i by 1. When $i = 15$, we exit this loop. The following is the iteration.

(3) *Merge two or added cell with neighbor cells* (lines 2-15): In Figure 5 (d), we obtain h, j, k, l and $\widetilde{H}$. Then, we further process l.

(4) *Merge added cell with neighbor cells* (lines 6-14): In Figure 5 (d), $c_{add} = l, i = 0 < 15$, we want to use *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i)$ to expand l into $Dir[0] = (L, R, T, B)$ directions (to include $\{j, k, \dots\}$). We get the potential newly added cell with a blue frame. But, four pink deleted cells will belong to both f and the newly added cell, violating the property of the deleted cell, so such cells do not exist and $\widehat{C}$ is *NULL*. We repeat it until $\widehat{C}$ is *NON-NULL*. In Figure 5 (e), $c_{add} = l, i = 6 < 15$, $\widehat{C} =$ *FindAddedCellNeig* $(\widetilde{H}, c_{add}, i) = \{l, j, \dots\}$, i.e., we can expand l into $Dir[6] = (L, \cdot, T, \cdot)$ directions to form $c_{add} = m$. Suppose that *Update* is *True*, we obtain $\widetilde{H}$.

4.3.2 *Detail and example for Algorithm 2*.

The following shows Algorithm 2 with an example. Figure 5 (b) and (c) illustrate steps 1–4 and 5–8, respectively. Figures 5 (d) and (e) are similar.

(1) *Update $O'(\cdot)$ and $O^{-1\prime}(\cdot)$* (lines 2-8): $\widehat{C} = \{a, b\}$ and $c_{add} = c$, since all cells in $\widehat{C}$ are in C_{rema}, we have $O'(c) = \{a, b\}$, $O^{-1\prime}(a) = c$ and $O^{-1\prime}(b) = c$.

(2) *Update neighbor cells* (lines 9-12): We update c and cells represented in orange points as neighbors of each other.

(3) *Update $\widetilde{H}'$* (lines 13-18): $\{a, b\}$ are deleted from C'_{rema} and c is added into C'_{add}, so $C'_{add} = \{c\}$.

(4) *Check ϵ-approximation* (lines 19-21): Suppose that $\widetilde{H}'$ is an ϵ-approximation of H, we have $C_{rema} = C \setminus \{a, b\}$, $C_{add} = \{c\}$, updated $\widetilde{N}(\cdot)$, $\widetilde{H}$, $O(\cdot)$ and $O^{-1}(\cdot)$. The following is the iteration.

(5) *Update $O'(\cdot)$ and $O^{-1\prime}(\cdot)$* (lines 2-8): $C_{add} = \{c\}$, $\widehat{C} = \{f, d, e, \dots\}$ and $c_{add} = f$, since for cells in $\widehat{C}$, c is in C_{add} and other cells are in C_{rema}, we have $O'(f) = \{a, b, d, e, \dots\}$, $O^{-1\prime}(a) = f$, $O^{-1\prime}(b) = f$, $O^{-1\prime}(d) = f$, $O^{-1\prime}(e) = f$, …, and delete $O'(c)$.

Algorithm 2 *Update* $(C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot), O^{-1}(\cdot))$

Input: $C_{rema}, C_{add}, \widetilde{N}(\cdot), \widehat{C}, c_{add}, O(\cdot)$ and $O^{-1}(\cdot)$
Output: updated $C_{rema}, C_{add}, \widetilde{N}(\cdot), O(\cdot), O^{-1}(\cdot)$, and whether the updated height map is an ϵ-approximation of H

1: $C'_{rema} \leftarrow C_{rema}, C'_{add} \leftarrow C_{add}, \widetilde{N}'(\cdot) \leftarrow \widetilde{N}(\cdot), \widetilde{N}'(c_{add}) \leftarrow \emptyset, O'(\cdot) \leftarrow O(\cdot), O^{-1\prime}(\cdot) \leftarrow O^{-1}(\cdot)$
2: **for** each $c \in \widehat{C}$ **do**
3:     **if** $c \in C_{rema}$ **then**
4:         $O'(c_{add}) \leftarrow O'(c_{add}) \cup \{p\}, O^{-1\prime}(c) \leftarrow \{c_{add}\}$
5:     **else if** $c \in C_{add}$ **then**
6:         **for** each $c' \in O'(c)$ **do**
7:             $O'(c_{add}) \leftarrow O'(c_{add}) \cup \{c'\}, O^{-1\prime}(c') \leftarrow \{c_{add}\}$
8:         $O'(\cdot) \leftarrow O'(\cdot) - \{O'(c)\}$
9: **for** each $c \in \widehat{C}$ **do**
10:     **for** each $c' \in N(c)$ such that $c' \notin \widehat{C}$ **do**
11:         $\widetilde{N}'(c_{add}) \leftarrow \widetilde{N}'(c_{add}) \cup \{c'\}, \widetilde{N}'(c') \leftarrow \widetilde{N}'(c') - c \cup \{c_{add}\}$
12: clear $\widetilde{N}'(c)$ for each $c \in \widehat{C}$
13: **for** each $c \in \widehat{C}$ **do**
14:     **if** $c \in C_{rema}$ **then**
15:         $C'_{rema} \leftarrow C'_{rema} - \{p\}$
16:     **else if** $c \in C_{add}$ **then**
17:         $C'_{add} \leftarrow C'_{add} - \{p\}$
18: $C'_{add} \leftarrow C'_{add} \cup \{c_{add}\}$
19: **if** $\widetilde{H}' = (C'_{rema} \cup C'_{add}, \widetilde{N}'(\cdot))$ is an ϵ-approximation of H **then**
20:     $C_{rema} \leftarrow C'_{rema}, C_{add} \leftarrow C'_{add}, \widetilde{N}(\cdot) \leftarrow \widetilde{N}'(\cdot), O(\cdot) \leftarrow O'(\cdot), O^{-1}(\cdot) \leftarrow O^{-1\prime}(\cdot)$
21:     **return** *True*
22: **return** *False*

(6) *Update neighbor cells* (lines 9-12): We update f and cells represented in orange points as neighbors of each other.

(7) *Update* $\widetilde{H}'$ (lines 13-18): $\{d, e, \dots\}$ are deleted from C'_{rema}, c is deleted from C'_{add}, and f is added into C'_{add}, so $C'_{add} = \{f\}$.

(8) *Check* ϵ*-approximation* (lines 19-21): Suppose that $\widetilde{H}'$ is an ϵ-approximaion of H, we have $C_{rema} = C \setminus \{a, b, d, e, \dots\}$, $C_{add} = \{f\}$, updated $\widetilde{N}(\cdot)$, $\widetilde{H}$, $O(\cdot)$ and $O^{-1}(\cdot)$.

4.4 Efficient ϵ-Approximate Simplified Height Map Checking

We illustrate our efficient checking technique.

4.4.1 ***Notation***. Given an added cell $c_{add} \in C_{add}$, we define a set of *adjacent added cells* of c_{add}, denoted by $A(c_{add})$, to be a set of added cells in C_{add} which contain c_{add} and are adjacent to each other. In Figure 7, $A(c_{add} = a) = \{a, b\}$.

4.4.2 ***Detail and example***. In Definition 1, we simplify the checking of Inequality 1 from "each pair of points s and t on H" to points on the following each type of cells related to c_{add}'s *neighbor*.

(1) **Remaining to Remaining cells (*R2R*)**: We simplify to "each pair of points s and t of remaining cells that are neighbor cells of each added cell in $A(c_{add})$". Figure 7 shows these points in orange.

(2) **Remaining to Deleted cells (*R2D*)**: We simplify to "each point s of remaining cell that is a neighbor cell of each cell in $A(c_{add})$, and each point t of deleted cell that belongs to each added cell

in $A(c_{add})$". Figure 7 shows these points in orange (corresponding to s) and purple (corresponding to t).

(3) **Deleted to Deleted cells (*D2D*)**: We simplify to "each pair of points s and t of deleted cells that belong to each added cell in $A(c_{add})$". Figure 7 shows these points in purple.

4.5 Shortest Path Query Phase

We illustrate the shortest path query phase using algorithm *HM-SP* on the simplified height map graph $\widetilde{G}$. Intuitively, we use Dijkstra's algorithm between source s and destination t on $\widetilde{G}$. But, if s is a point on a deleted cell (i.e., s does not exist in $\widetilde{G}$), a naive algorithm uses Dijkstra's algorithm multiple times with all points on neighbor cells of $O^{-1}(s)$ as sources. But, we propose an efficient algorithm using *an efficient implicit edge insertion technique* to use Dijkstra's algorithm only once for pruning.

4.5.1 ***Notation***. Given a point m (on a deleted cell) and a point n (on a remaining/added cell) where $n \in \widetilde{N}(O^{-1}(m))$, we call the path between them passing on $\widetilde{H}$ as the *intra-path*, and denote it by $\Pi_1(\widetilde{m}, \widetilde{n}|\widetilde{H}) = \langle \widetilde{m}, \widetilde{n} \rangle$. Given a pair of point $\widetilde{p}$ (on a remaining/added cell) and point $\widetilde{q}$ (on a remaining/added cell), we call the path between them passing on $\widetilde{H}$ as the *inter-path*, and denote it by $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$. In Figures 5 (g) and 8, $\Pi_1(\widetilde{g}, \widetilde{n}|\widetilde{H})$ and $\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})$ in blue dashed lines are two intra-paths, $\Pi_2(\widetilde{n}, k|\widetilde{H})$ and $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$ in blue solid lines are two inter-paths.

4.5.2 ***Detail and example***. There are two steps.

(1) **Point estimation**: Given a pair of points s and t (on cells) of H, we estimate $\widetilde{s}$ using s, such that $\widetilde{s}.x = s.x$, $\widetilde{s}.y = s.y$, $\widetilde{s}.z = O^{-1}(s).z$ (if s is on a deleted cell), or $\widetilde{s} = s$ (if s is on a remaining cell). In Figure 5 (b), since a is a deleted cell, $O^{-1}(a) = c$, we have $\widetilde{a}.x = a.x$, $\widetilde{a}.y = a.y$ and $\widetilde{a}.z = c.z$. We estimate $\widetilde{t}$ similarly.

(2) **Path querying**: There are three cases depending on whether s and t are on deleted or remaining cells.

(i) *Both cells deleted*: Firstly, there are two special cases that we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H}) = \langle \widetilde{s}, \widetilde{t} \rangle$. One is that s and t are on cells belong to the different added cells u and v, where u and v are neighbor. The other one is that s and t are on cells belong to the same added cell. In Figure 5 (g), $\Pi(\widetilde{d}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{d}, \widetilde{i} \rangle$ (i.e., the first case) and $\Pi(\widetilde{a}, \widetilde{d}|\widetilde{H}) = \langle \widetilde{a}, \widetilde{d} \rangle$ (i.e., the second case). Secondly, for common case, we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$ by concatenating the intra-path $\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})$, the inter-path $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})$, and the intra-path $\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{H})$, such that $|\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})| = \min_{\forall \widetilde{p} \in \widetilde{N}(O^{-1}(s)), \widetilde{q} \in \widetilde{N}(O^{-1}(t))} |\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{H})| + |\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{H})| + |\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{H})|$. In Figure 8, orange and pink points denote possible points on cells $\widetilde{N}(O^{-1}(s))$ and $\widetilde{N}(O^{-1}(t))$, $\widetilde{p}$ and $\widetilde{q}$ are points resulting in the minimum distance among these points, respectively. A naive algorithm uses Dijkstra's algorithm on $\widetilde{H}$ with each point on cell in $\widetilde{N}(O^{-1}(s))$ as a source to compute inter-paths. But, our efficient algorithm uses Dijkstra's algorithm only once. If the number of cells in $\widetilde{N}(O^{-1}(s))$ is less than that of in $\widetilde{N}(O^{-1}(t))$, we implicitly insert intra-paths between $\widetilde{s}$ and each point on cell in $\widetilde{N}(O^{-1}(s))$ as edges in $\widetilde{G}$ (we remove them after this calculation). Then, we use Dijkstra's algorithm on $\widetilde{G}$ with $\widetilde{s}$ as a source, and terminate after visiting all points on cells in $\widetilde{N}(O^{-1}(t))$, to compute the intra-path connecting to $\widetilde{s}$ and the inter-path. We append them with the intra-path connecting to $\widetilde{t}$ and obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$. If the number of cells in $\widetilde{N}(O^{-1}(s))$ is larger than that of in $\widetilde{N}(O^{-1}(t))$, we swap s and t. In Figure 5 (g), $\Pi(\widetilde{g}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{g}, \widetilde{n}, k, \widetilde{i} \rangle$.

(ii) *One cell deleted and one cell remaining*: If s is on cell in C_{rema}, the inter-path connecting to s does not exist, we use Dijkstra's algorithm on $\widetilde{G}$ with s as a source, and terminate after visiting all points on cells in $\widetilde{N}(O^{-1}(t))$. We append them with the intra-path connecting to $\widetilde{t}$ and obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{H})$. If $t \in C_{rema}$, we swap s and t. In Figure 5 (g), $\Pi(\widetilde{n}, \widetilde{i}|\widetilde{H}) = \langle \widetilde{n}, k, \widetilde{i} \rangle$.

(iii) *Both cells remaining*: Both inter-paths do not exist, we use Dijkstra's algorithm on $\widetilde{G}$ between s and t to obtain $\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})$. In Figure 5 (g), $\Pi(\widetilde{o},\widetilde{p}|\widetilde{H}) = \langle \widetilde{o}, k, \widetilde{p} \rangle$.

4.6 Proximity Query Algorithms

Given H and $\widetilde{H}$, a query point i (on cell), a set of n' interested points on cells on H or $\widetilde{H}$, two parameters k (k value in *kNN* query) and r (range value in range query), we can answer *kNN* and range queries using algorithm *HM-SP*. A naive algorithm uses it for n' times between i and all interested points, and then performs a linear scan on the paths to compute *kNN* and range query results.

But, we propose an efficient algorithm using *an efficient parallel computation technique* to use it (i.e., Dijkstra's algorithm) only once for pruning. (1) For algorithm *HM-SP* on H, we use Dijkstra's algorithm once with i as a source and all interested points as destinations, and then directly return *kNN* and range query results without any linear scan. Since these paths are already sorted in order during the execution of Dijkstra's algorithm. (2) For algorithm *HM-SP* on $\widetilde{H}$, we also use Dijkstra's algorithm once. Except for two special cases in Section 4.5.2 case (2-i) that directly return the path $\Pi(\widetilde{i},\widetilde{j}|\widetilde{H}) = \langle \widetilde{i},\widetilde{j} \rangle$, where j is the interested point (of an interested cell), there are two cases. We define S to be a set of points, such that for each j, we store j in S if j is on a remaining cell, or we store points on cells in $\widetilde{N}(O^{-1}(j))$ into S if j is on a deleted cell. The two cases are: (i) If i is on a deleted cell, we change "s" to "i", "terminate after Dijkstra's algorithm visits all points on cells in $\widetilde{N}(O^{-1}(t))$" to "terminate after Dijkstra's algorithm visits all points in S" and "append them with the intra-path connecting to $\widetilde{t}$" to "append them with the intra-path connecting to each $\widetilde{j}$ if j is on a deleted cell" in Section 4.5.2 case (2-i). (ii) If i is on a remaining cell, we apply the same three changes in Section 4.5.2 case (2-ii). Finally, we perform a linear scan on the paths to compute *kNN* and range query results.

4.7 Add-on Data Structure

Given a $(1+\epsilon)$-approximate simplified graph of a complete graph, study [52] constructs a $(1+\epsilon')$-approximate data structure on the simplified graph, to return $(1+\epsilon')(1+\epsilon)$-approximate paths in $O(1)$ time. We can use this data structure (i.e., a graph $G_{\widetilde{H}}$) in the simplified height map graph of $\widetilde{H}$ in algorithm *HM-Simplify*, and use *HM-SP* on $G_{\widetilde{H}}$ for querying in $O(1)$ time. We denote our adapted algorithms (after using $G_{\widetilde{H}}$) to be algorithms *HM-Simplify Data Structure* (*HM-Simplify-DS*) and *HM-SP Data Structure* (*HM-SP-DS*).

4.8 Theoretical Analysis

4.8.1 *Algorithms HM-Simplify and HM-SP.*
We analyze them in Theorems 4.1 and 4.2.

THEOREM 4.1. *The simplification time, number of cells in $\widetilde{H}$, output size and simplification memory of algorithm HM-Simplify are $O(n\lambda \log n)$, $O(\frac{n}{\mu})$, $O(\frac{n}{\mu})$ and $O(n)$ respectively, where $\lambda \in [\sqrt[3]{n}, \frac{n}{2}]$ and $\mu \in [2, \log n]$ are constants depending on H and ϵ, and $\lambda \in [10, 290]$ and $\mu \in [5, 88]$ in our experiments. Given H, it returns $\widetilde{H}$ such that $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|$ for each pair of points s and t on H.*

PROOF SKETCH. The simplification time is due to the usage of Dijkstra's algorithm in $O(n \log n)$ time for $O(1)$ cells in *R2R*, *R2D* and *D2D* checking, with total λ cell merging iterations. The number of cells in $\widetilde{H}$ and output size are due to the total n cells on H and μ deleted cells belonging to each added cell on average. The simplification memory is due to the original size $O(n)$ of H. The error guarantee of $\widetilde{H}$ is due to the *R2R*, *R2D* and *D2D* checking. The detailed proof appears in our technical report [73]. □

Theorem 4.2. *The shortest path query time and memory of algorithm HM-SP are $O(n \log n)$ and $O(n)$ on H, and are $O(\frac{n}{\mu} \log \frac{n}{\mu})$ and $O(\frac{n}{\mu})$ on $\widetilde{H}$, respectively. It returns the exact shortest path passing on H, and returns an approximate shortest path passing on $\widetilde{H}$ such that $(1-\epsilon)|\Pi(s,t|H)| \leq |\Pi(\widetilde{s},\widetilde{t}|\widetilde{H})| \leq (1+\epsilon)|\Pi(s,t|H)|$.*

Proof. Since there are $O(n)$ and $O(\frac{n}{\mu})$ cells in H and $\widetilde{H}$, algorithm *HM-SP* (a Dijkstra's algorithm) returns exact results on both, and $\widetilde{H}$ is an ϵ-approximation of H, we finish the proof. □

4.8.2 ***Proximity query algorithms.*** Given a query point i, let p_f and p'_f be the furthest point to i computed using the ground-truth or optimal distance and a studied algorithm (computed by algorithm *HM-SP* on H and $\widetilde{H}$), respectively. Let the error ratio of *kNN* or range query be $(\frac{|\Pi(i,p'_f|Z)|}{|\Pi(i,p_f|Z)|} - 1)$, where $Z \in \{H, P, T\}$ is the 3D surface (height map, point cloud or *TIN*) used for calculating the ground-truth or optimal distance ($Z = H$ in our case). We analysis *kNN* or range query using algorithm *HM-SP* in Theorem 4.3.

Theorem 4.3. *The kNN or range query time and memory of using algorithm HM-SP are $O(n \log n)$ and $O(n)$ on H and $O(\frac{n}{\mu} \log \frac{n}{\mu})$ and $O(\frac{n}{\mu})$ on $\widetilde{H}$, respectively. It returns the exact result on H and has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{H}$ for kNN or range query.*

Proof Sketch. The query time and memory are due to usage of algorithm *HM-SP* once. The error arises from its error. □

4.8.3 ***Algorithms HM-Simplify-DS and HM-SP-DS.*** We analyze them in Theorems 4.4 and 4.5.

Theorem 4.4. *The simplification time, number of edges in $G_{\widetilde{H}}$, output size and simpflication memory of algorithm HM-Simplify-DS are $O(n\lambda \log n + \frac{n^2}{\mu^2} \log^2 \frac{n}{\mu})$, $O(\frac{n}{\mu} \log \frac{n}{\mu})$, $O(\frac{n}{\mu} \log \frac{n}{\mu})$ and $O(\frac{n}{\mu} \log \frac{n}{\mu})$, respectively. Given a height map H, it returns $G_{\widetilde{H}}$ such that $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$ for each pair of points s and t on H, where $\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})$ is the approximate shortest path between $\widetilde{s}$ and $\widetilde{t}$ passing on $G_{\widetilde{H}}$.*

Theorem 4.5. *The shortest path query time and memory, kNN or range query time and memory of algorithm HM-SP-DS are $O(1)$, $O(\frac{n}{\mu} \log \frac{n}{\mu})$, $O(n')$ and $O(\frac{n'n}{\mu} \log \frac{n}{\mu})$, respectively. It returns an approximate shortest path passing on $G_{\widetilde{H}}$ such that $|\Pi(\widetilde{s},\widetilde{t}|G_{\widetilde{H}})| \leq (1+\epsilon')(1+\epsilon)|\Pi(s,t|H)|$, and has an error ratio $\epsilon' \cdot \epsilon + \epsilon' + \epsilon$ for kNN or range query.*

Proof Sketch. The detailed proofs of Theorems 4.4 and 4.5 appear in our technical report [73]. □

5 Empirical Studies

5.1 Experimental Setup

We performed experiments using a Linux machine with 2.2 GHz CPU and 512GB memory. Algorithms were implemented in C++. The experiment setup follows studies [44, 45, 64, 65, 71, 72, 74].

5.1.1 ***Datasets.*** (1) *Height map* datasets: We conducted experiments using 34 (= 5 + 5 + 24) real height map datasets listed in Table 1, where the subscript h indicates a height map. (i) *5 Original datasets*: GF_h [12, 72], LM_h [15, 72] and RM_h [19, 72] are originally represented as height maps obtained from Google Earth [9]. They are used in study [72]. BH_h [6, 64, 65, 72] and EP_h [6, 64, 65, 72] are originally represented as points clouds, we created height maps with cell's 2D coordinate and elevation values equal to the z-coordinate values of these points. They are used in studies [64, 65, 72].

These five datasets have a 20km × 20km region with a 28m × 28m resolution [45, 65, 71, 72]. (ii) *5 Small-version datasets*: They are generated using the same region as the original datasets, with a 633m × 633m resolution, following the dataset generation steps [65, 71, 72]. (iii) *24 Multi-resolution datasets*: They are generated similarly with varying numbers of cells. (2 & 3) *Point cloud* and *TIN* datasets: We convert the height map datasets to 34 point cloud datasets by cell mapping [26, 47, 63, 77], and then to 34 *TIN* datasets by point triangulation [35, 62, 72]. We use p and t as subscripts, respectively.

Table 1. Height map datasets

Name	$\|n\|$
Original dataset	
GunnisonForest (GF_h) [12, 72]	0.5M
LaramieMount (LM_h) [15, 72]	0.5M
RobinsonMount (RM_h) [19, 72]	0.5M
BearHead (BH_h) [6, 64, 65, 72]	0.5M
EaglePeak (EP_h) [6, 64, 65, 72]	0.5M
Small-version dataset	
GF_h*-small*	1k
LM_h*-small*	1k
RM_h*-small*	1k
BH_h*-small*	1k
EP_h*-small*	1k
Multi-resolution dataset	
GF_h multi-resolution	5M, 10M, 15M, 20M, 25M
LM_h multi-resolution	5M, 10M, 15M, 20M, 25M
RM_h multi-resolution	5M, 10M, 15M, 20M, 25M
BH_h multi-resolution	5M, 10M, 15M, 20M, 25M
EP_h multi-resolution	5M, 10M, 15M, 20M, 25M
EP_h*-small* multi-resolution	10k, 20k, 30k, 40k, 50k

5.1.2 ***Algorithms***. (1) To solve our problem on *Height Maps*, we adapted existing algorithms on point clouds or *TIN*s, by converting the given height maps to point clouds [26, 47, 63, 77] or *TIN*s [26, 35, 47, 62, 63, 72, 77] so that the existing algorithms could be performed. Then, we add "*-Adapt(HM)*" in algorithm names. We have 4 simplification algorithms: (i) the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)* [42, 45], (ii) adapted *TIN* shortest network distance simplification algorithm *TIN-NSimplify-Adapt(HM)* [45], (iii) the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)* [24, 72] and (iv) our height map simplification algorithm *HM-Simplify*. We have 5 proximity query algorithms: (i) the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-ESSP-Adapt(HM)* [28, 66, 74], (ii) the best-known adapted approximate *TIN* shortest surface path query algorithm *TIN-ASSP-Adapt(HM)* [44, 71], (iii) the best-known adapted approximate *TIN* shortest network path query algorithm *TIN-SNP-Adapt(HM)* [45], (iv) the best-known adapted exact point cloud shortest path query algorithm *PC-SP-Adapt(HM)* [72] and (v) our exact height map shortest path query algorithm *HM-SP*. The exact algorithms refer to their particular 3D surfaces only. For 4 proximity query algorithms *TIN-ESSP-Adapt(HM)*, *TIN-SNP-Adapt(HM)*, *PC-SP-Adapt(HM)* and *HM-SP*, we use $\epsilon = 0$ (resp. $\epsilon > 0$) to denote that we apply them on the original (resp. simplified) surfaces. Since *TIN-ESSP-Adapt(HM)* with $\epsilon > 0$ already means calculating the exact shortest surface path passing on a simplified *TIN*, there is no need to use *TIN-ASSP-Adapt(HM)* on the simplified *TIN* again, i.e., no need to distinguish $\epsilon = 0$ or $\epsilon > 0$ for it. So, we only consider it with $\epsilon > 0$ on the original height map for simplicity. We compare all algorithms in Tables 2 and 3.

Table 2. Comparison of simplification algorithms

Algorithm	Simplification time		Output size	
TIN-SSimplify-Adapt(HM) [42, 45]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large
TIN-NSimplify-Adapt(HM) [45]	$O(n^2\log n)$	Medium	$O(n)$	Large
PC-Simplify-Adapt(HM) [24, 72]	$O(n^2\log n)$	Medium	$O(n)$	Large
***HM-Simplify* (ours)**	$\boldsymbol{O(n\lambda\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\mu})}$	**Small**

Table 3. Comparison of proximity query algorithms

Algorithm	Shortest path query time		Error
On the original 3D surfaces			
TIN-ESSP-Adapt(HM) [28, 66, 74]	$O(n^2)$	Large	Small
TIN-ASSP-Adapt(HM) [44, 71]	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-SNP-Adapt(HM) [45]	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [72]	$O(n\log n)$	Medium	No error
***HM-SP* (ours)**	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**
On the simplified 3D surfaces			
TIN-ESSP-Adapt(HM) [28, 66, 74]	$O(n^2)$	Large	Small
TIN-SNP-Adapt(HM) [45]	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [72]	$O(n\log n)$	Medium	Small
***HM-SP* (ours)**	$\boldsymbol{O(\frac{n}{\mu}\log\frac{n}{\mu})}$	**Small**	**Small**

(2) To solve the existing problem on *Point Clouds* [72], we adapted algorithms on height maps or *TIN*s, by converting the given point clouds to height maps [25, 60] or *TIN*s [35, 62, 72]. Then, we add "*-Adapt(PC)*" in algorithm names. Similarly, we have 9 algorithms: (i) *TIN-SSimplify-Adapt(PC)* [42, 45], (ii) *TIN-NSimplify-Adapt(PC)* [45], (iii) *PC-Simplify* [24, 72], (iv) *HM-Simplify-Adapt(PC)*, (v) *TIN-ESSP-Adapt(PC)* [28, 66, 74], (vi) *TIN-ASSP-Adapt(PC)* [44, 71], (vii) *TIN-SNP-Adapt(PC)* [45], (viii) *PC-SP* [72] and (ix) *HM-SP-Adapt(PC)*.

(3) To solve the existing problem on *TIN*s [42, 45], we adapted algorithms on height maps or point clouds, by converting the given *TIN*s to height maps [23, 39] or point clouds [72, 75]. Then, we add "*-Adapt(TIN)*" in algorithm names. Similarly, we have 9 algorithms: (i) *TIN-SSimplify* [42, 45], (ii) *TIN-NSimplify* [45], (iii) *PC-Simplify-Adapt(TIN)* [24, 72], (iv) *HM-Simplify-Adapt(TIN)*, (v) *TIN-ESSP* [28, 66, 74], (vi) *TIN-ASSP* [44, 71], (vii) *TIN-SNP* [45], (viii) *PC-SP-Adapt(TIN)* [72] and (ix) *HM-SP-Adapt(TIN)*.

Points (2) and (3) are *additional* adaptations since we want to see the performance of our algorithms for other problems. These adaptations involve data conversion. If no data conversion is involved, (1) we can adapt *HM-Simplify* and *HM-SP* to the point cloud, and the adapted versions have the same performance as them on the height map since the height map graph and the point cloud graph are the same, and (2) there is no reason to adapt *HM-Simplify* and *HM-SP* to the *TIN* since expensive *TIN* re-triangulation is involved in simplification, and the *TIN*'s structure is more complex, which both significantly harm the performance (i.e., adapted versions have the similar performance of *TIN-NSimplify* and *TIN-SNP* on the *TIN*).

5.1.3 ***Proximity Queries.*** We conducted 3 queries. (1) Shortest path query: we used 100 travelers' real hiking GPS data (recording sources and destinations) in 2023 from Google Maps [11] on each real height map, point cloud or *TIN* dataset to perform the query. We report the average, maximum and minimum results. The experimental result figures' points indicate the average results, and vertical bars represent the maximum and minimum values. (2 & 3) *kNN* and range queries: we used travelers' real nearby searching data of 1000 query objects (viewpoints and hotels) in 2023 from Tripadvisor [76] on each dataset to perform the proximity query algorithm in Section 4.6.

5.1.4 ***Factors and Metrics.*** We studied 5 factors: (1) ϵ, (2) n (dataset size, i.e., the number of cells, points or vertices of a height map, point cloud or *TIN*), (3) d (the maximum pairwise distances among query objects), (4) k (k value in *kNN* query) and (5) r (range value in range query). When not varying $d \in$ [4km, 20km], $k \in$ [200, 1000] and $r \in$ [2km, 10km], we fix d at 10km, k at 500 and r at 5km according to studies [32, 59]. For simplification algorithms, we employed 4 metrics: (1) *preprocessing time* (the data conversion time (if any) plus the simplification time, where the former is 10^6 to 10^9 times smaller than the latter), (2) *the number of cells, points or vertices* in the simplified height map, point cloud or *TIN*, (3) *output size* and (4) *preprocessing memory* (simplification memory). For proximity query algorithms, we employed 9 metrics: (1) *query time* (the data conversion time (if any) plus the shortest path query time, where the former is 10^4 to 10^6 times smaller than the latter), (2 & 3) *kNN or range query time* (the data conversion time (if any) plus *kNN* or range query time), (4) *memory* (during the shortest path query algorithm execution), (5 & 6) *kNN or range query memory*, (7) *distance error ratio* (the error ratio of the distance calculated by a studied algorithm compared with the ground-truth or optimal distance), (8 & 9) *kNN or range query error ratio* (see Section 4.8.2).

There are two sets of experiments regarding distance error ratio calculation. We first introduce the following. The relative error of the *TIN*'s exact shortest surface distance [34, 43] and the height map's exact shortest distance [47, 63] compared with the real shortest distance in the real world (measured in an on-site survey on a real 3D surface by human) are 0.0454 and 0.0613 on average, with variance 0.0015 and 0.0026, respectively. Both distances are *approximation* of the real shortest distance without a bound guarantee (similar to well-known road network models [37, 41]), and the latter is computed on the point cloud converted from the given height map (since the point cloud's exact shortest distance is the same as the height map's exact shortest distance in Section 3.1). Then, we introduce the two sets of experiments. (1) We regard the *TIN*'s exact shortest surface distance (computed by *TIN-ESSP* with $\epsilon = 0$) as the *so-called* ground-truth distance when using height maps, point clouds and *TIN*s as input consistently across experiments. When we write ground-truth distance, we mean the *TIN*'s exact shortest surface distance. The first reason is that compared with the real shortest distance, the average error of this distance, i.e., 0.0454, is smaller than that of the height map's exact shortest distance, i.e., 0.0613, although this distance is not always smaller than the height map's exact shortest distance in Section 3.1 (e.g., Euclidean distance is usually smaller than *TIN*'s exact shortest surface distance, but its error is larger). The second reason is that a *TIN* is a more detailed representation of the underlying 3D surface. The third reason is that this distance is widely used in studies [42, 44, 64, 65, 74]. (2) For the rigorous formulation of our problem and proposed algorithms comparison (based on *height map only*), we regard the height map's exact shortest distance (computed by *HM-SP* with $\epsilon = 0$) as the optimal distance under this particular 3D surface. When we write optimal distance, we mean the height map's exact shortest distance.

5.2 Experimental Results

5.2.1 ***Height maps with ground-truth distance.*** We studied proximity queries on height maps using the ground-truth distance for distance error ratio calculation. We compared all algorithms in Tables 2 and 3 on small-version datasets, and compared *HM-Simplify*, and *HM-SP* and *TIN-ASSP-Adapt(HM)* (since other algorithms are very slow) on original datasets.

(1) **Baseline comparisons**:

(i) **Effect of** ϵ: In Figures 9 (a) to (h), we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_h*-small* dataset while fixing n at 1k for baseline comparisons. The preprocessing time of

HM-Simplify is much smaller than three baselines' due to the efficient height map shortest path query and efficient ϵ-approximate simplified height map checking (although the worst case is $O(n^2 \log n)$ in Theorem 4.1, which never happens in the experiment). The number of cells of the simplified height map and output size of *HM-Simplify* are also much smaller than three baselines' due to the novel cell merging technique. The memory of a simplification algorithm is the same as that of the corresponding shortest path query algorithm with $\epsilon = 0$, since we clear the memory after performing one shortest path query during simplification, the preprocessing memory figures are omitted. The shortest path, *kNN* and range query time ($O(\frac{nn'}{\mu} \log \frac{n}{\mu})$ in Theorem 4.3) of *HM-SP* on the simplified height map are also small since its simplified height map has a small output size. The shortest path, *kNN* and range query memory are similar, since we clear the memory after performing one shortest path query during *kNN* or range query, the *kNN* and range query memory figures are omitted. Although increasing ϵ slightly increases the experimental distance error ratio of *HM-SP* on the simplified height map, i.e., close to 0. So, increasing ϵ has no impact on the experimental *kNN* and range query error ratios, their values are 0 (since $|\Pi(i, p'_f|T)| = |\Pi(i, p_f|T)|$ in Section 4.8.2), and their results are omitted. In Figure 9 (h), the relative error of distance returned by *HM-SP* on the simplified height map compared with the ground-truth distance is 0.0340. Compared with the real shortest distance, recall that the relative error of the ground-truth distance is 0.0454, so the relative error of the distance returned by *HM-SP* on the simplified height map is $0.0809 (= (1 + 0.0340) \times (1 + 0.0454) - 1)$. Since the relative error can have negative values, the relative error is also $0.0779 (= 1 - (1 - 0.0340) \times (1 - 0.0454))$. We take $0.0809 = \max(0.0809, 0.0779)$.

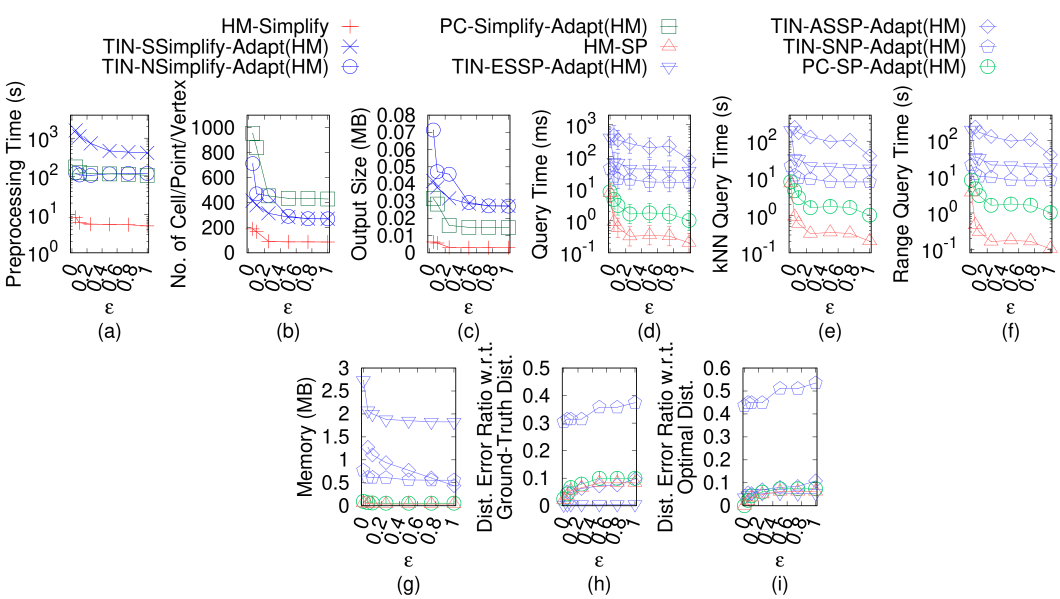

Fig. 9. Effect of ϵ on GF_h-*small* height map dataset

(ii) **Effect of n (scalability test)**: In Figures 10 (a) to (e), we tested 5 values of n in {5M, 10M, 15M, 20M, 25M} on LM_h dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify* (in terms of output size, i.e., 6.8MB) and *HM-SP* on the simplified height map (in terms of range query time, i.e., 310s $\approx$ 5.1 min, and range query memory, i.e., 320MB) are scalable on extremely large height map with 25M cells. Although the theoretical output size of *HM-Simplify* is only μ times smaller than the size of an original height map, it returns a simplified height map with an experimental size of 6.8MB from an original one with size 600MB and 25M cells, and performing range query on them with 500 objects takes 400s ≈ 6.7 min and 35,200s ≈ 9.8 hours, with 320MB and 1.1GB memory, respectively. When n is smaller, i.e., datasets with looser density or fragmentation (since multi-resolution datasets have the same region), algorithms run faster with less memory.

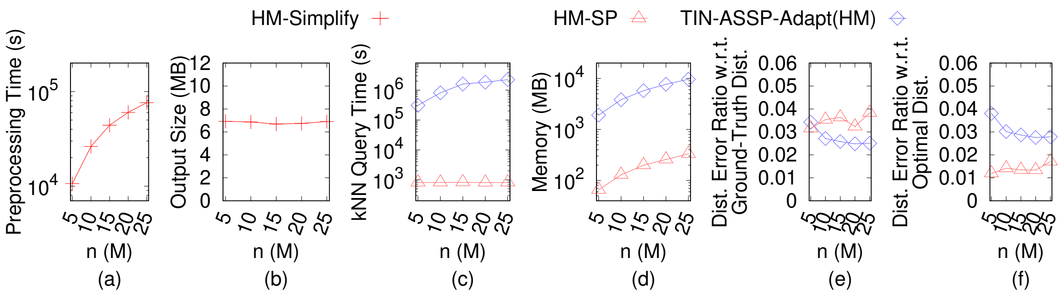

Fig. 10. Effect of n on LM_h height map dataset

(iii) **Effect of** d: In Figure 11, we tested 5 values of d in {4km, 8km, 12km, 16km, 20km} on RM_h dataset while fixing ϵ at 0.25 and n at 0.5M for baseline comparisons. A smaller d reduces shortest path, *kNN* and range query time, since *HM-SP* and our proximity query algorithm use Dijkstra's algorithm once, we can terminate them earlier after visiting all destination objects or query objects. As d increases, there is no upper bound on the increase in shortest path query time (since we terminate Dijkstra's algorithm based solely on d), and also in *kNN* query time (since we append the paths computed by Dijkstra's algorithm and the intra-paths as results, we cannot determine the distance correlations among these paths until we perform a linear scan, i.e., we terminate Dijkstra's algorithm based solely on d). But, there is an upper bound on the increase in range query time (since we can also terminate Dijkstra's algorithm earlier if the searching distance exceeds r).

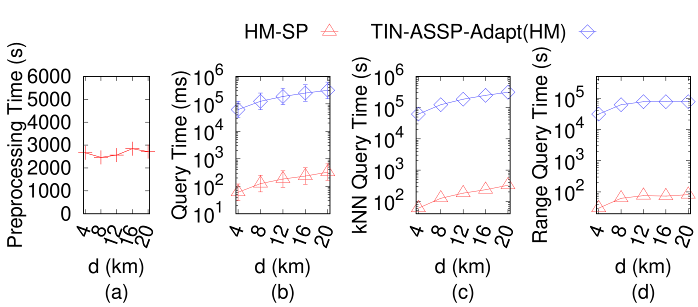

Fig. 11. Effect of d on RM_h height map dataset

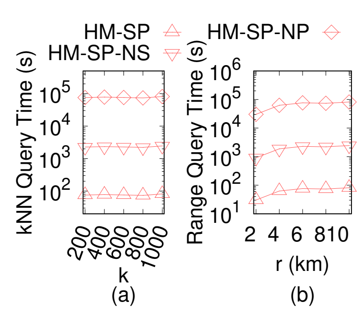

Fig. 12. Ablation study for proximity query algorithms (effect of k and r on RM_h height map dataset)

(2) **Ablation study for proximity query algorithms (effect of** k **and** r**)**: We considered two variations of *HM-SP* (on the simplified height map), i.e., (i) *HM-SP Naive Shortest path query* (*HM-SP-NS*): *HM-SP* using the naive shortest path query algorithm in Section 4.5, but the efficient proximity query algorithm in Section 4.6, and (ii) *HM-SP Naive Proximity query* (*HM-SP-NP*): *HM-SP* using the efficient shortest path query algorithm, but the naive proximity query algorithm. In Figure 12, we tested 5 values of k in {200, 400, 600, 800, 1000} and 5 values of r in {2km, 4km, 6km,

8km, 10km} both on RM_h dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study. On the simplified height map, *HM-SP* outperforms both *HM-SP-NS* and *HM-SP-NP*, due to the efficient querying algorithms. Due to the two reasons in the previous paragraph, k does not affect *kNN* query time, but a smaller r reduces range query time.

(3) **Ablation study for simplification algorithms**: We considered three variations of *HM-Simplify*, i.e., (i) *HM-Simplify Naive Merging* (*HM-Simplify-NM*): *HM-Simplify* using the naive merging technique that only merges two cells in Sections 4.2.1 and 4.2.2 point (2), (ii) *HM-Simplify Naive Checking* (*HM-Simplify-NC*): *HM-Simplify* using the naive checking technique that checks whether Inequality 1 is satisfied for all points in Section 4.2.2 point (3) and (iii) *HM-Simplify-DS* (with $\epsilon' = 0.25$). Let *HM-SP-NM*, *HM-SP-NC* and *HM-SP-DS* be the corresponding proximity query algorithms on the simplified height map. In Figure 13, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_h*-small* dataset while fixing n at 0.5M for ablation study. *HM-Simplify* performs the best, showing the effectiveness of our merging and checking techniques. Since *HM-Simplify-DS* has a large simplification time but *HM-SP-DS* has a small shortest path query time, they are useful when we prioritize the latter time over the former.

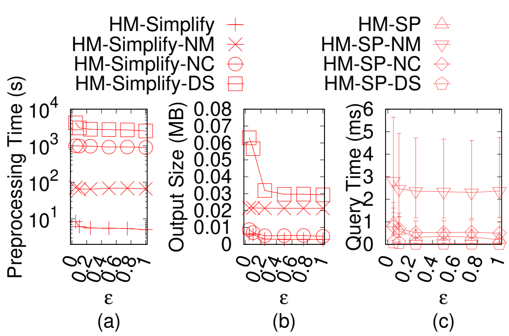

Fig. 13. Ablation study for simplification algorithms on BH_h*-small* height map dataset

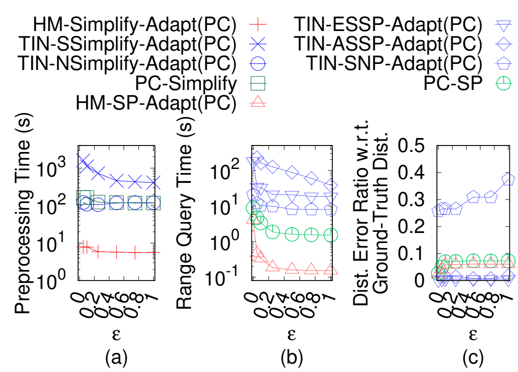

Fig. 14. Effect of ϵ on EP_p*-small* point cloud dataset

5.2.2 ***Point clouds with ground-truth distance***. We studied proximity queries on point clouds using the ground-truth distance for distance error ratio calculation. In Figure 14, we tested 7 values of ϵ in {0, 0.05, 0.1, 0.25, 0.5, 0.75, 1} on EP_p*-small* dataset while fixing n at 1k for baseline comparison. *HM-Simplify-Adapt(PC)* and *HM-SP-Adapt(PC)* on the simplified height map still outperforms other baselines.

5.2.3 ***TINs with ground-truth distance***. We studied proximity queries on *TIN*s using the ground-truth distance for distance error ratio calculation. In Figure 15, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on EP_t*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. *HM-Simplify-Adapt(TIN)* still outperforms other baselines. The distance error ratio of *HM-SP-Adapt(TIN)* on the simplified height map is 0.0401, but the distance error ratio of *TIN-SNP* on the simplified *TIN* is 0.2732.

5.2.4 ***Height maps with optimal distance***. We studied proximity queries on height maps using the optimal distance for distance error ratio calculation. In Figures 9 (i) and 10 (f), the relative error of distance returned by *HM-SP* on the simplified height map compared with the optimal distance is 0.0186. Compared with the real shortest distance, recall that the relative error of the optimal distance is 0.0613, so the relative error of the distance returned by *HM-SP* on the simplified height

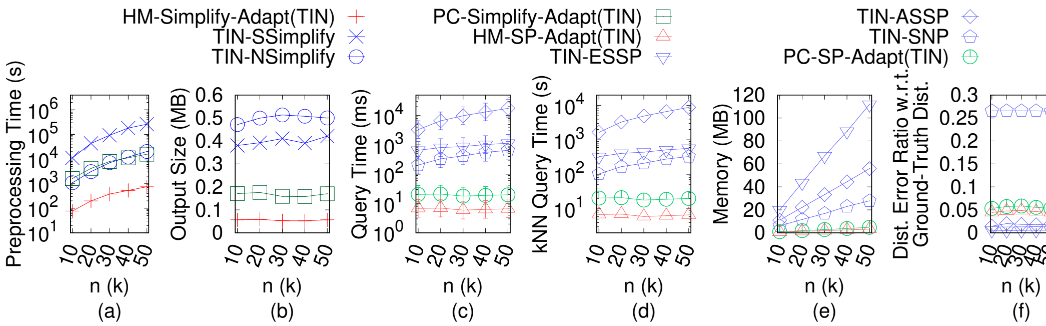

Fig. 15. Effect of n on EP_t*-small TIN* dataset

map is $0.0810 (= (1+0.0186)\times(1+0.0613)-1)$. Since the relative error can have negative values, the relative error is also $0.0788 (= 1-(1-0.0186)\times(1-0.0613))$. We take $0.0810 = \max(0.0810, 0.0788)$.

5.2.5 ***Case study***. We performed a snowfall evacuation case study [5] at Gates of the Arctic [57] to evacuate tourists to nearby hotels. In Figure 1 (a), due to each hotel's capacity constraints, we find shortest paths from viewpoint a to k-nearest hotels b, c, d, where c and d are k-nearest options when $k = 2$. An individual will be buried in snow in 2.4 hours[3], and the evacuation can be finished in 2.2 hours[4]. Thus, we need to compute shortest paths within 12 min (= 2.4 − 2.2 hours). Our experiments show that for a height map with 50k cells, 10k tourist positions and 50 hotels, the simplification time for our algorithm *HM-Simplify*, our adapted algorithm *HM-Simplify-DS*, the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)* and the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)* are 250s ≈ 4.6 min, 125,000s ≈ 1.5 days, 5,250s ≈ 1.5 hours and 103,000s ≈ 1.2 days. Computing 10 nearest hotels for each tourist position on the simplified 3D surfaces of these algorithms takes 50s, 5s, 250s ≈ 4.2 min and 67,000s ≈ 18.6 hours, respectively. Thus, height map simplification is necessary since 4.6 min + 1.6 min ≤ 12 min. Recall that only the height map dataset is available for this region, we capture the height map dataset after snowfall, and we have efficient height map simplification and shortest path query algorithms. So, there is no reason to convert the height map to the point cloud or *TIN*, and perform other slow adapted point cloud or *TIN* algorithms for simplification and shortest path query. In addition, it is known that algorithms with larger (resp. smaller) simplification time but smaller (resp. larger) shortest path query time are better (resp. worse) *if* we need simplification *before* the query issue. But, it is not true in our case, since we capture the height map dataset *after* snowfall, the simplification time is considered after snowfall. So, we design *HM-Simplify* to efficiently *reduce* the simplification time, and significantly *reduce* its output size so that the shortest path query time on the simplified height map is *small*. But, *HM-Simplify-DS* is not suitable due to the large simplification time.

5.2.6 ***Paths visualization***. In Figure 16, we visualize different paths to verify distance relationships in Section 3.1. (1) Given a height map, the paths in Figures 16 (a) (showing the height map) and (b) (showing the same height map in bird's eye view) computed by our algorithm *HM-SP* on

[3] 2.4 hours = $\frac{10\text{centimeters}\times 24\text{hours}}{1\text{meter}}$, since the snowfall rate (i.e., the snow depth over a period [29, 58]) at Gates of the Arctic is 1 meter per 24 hours [5], and when the snow depth exceeds 10 centimeters, it is difficult to walk and easy to bury in the snow [36].

[4] 2.2 hours = $\frac{11.2\text{km}}{5.1\text{km/h}}$, since the average distance between the viewpoints and hotels at Gates of the Arctic is 11.2km [10], and human's average walking speed is 5.1 km/h [18].

the original height map and the path in Figure 16 (c) computed by the best-known adapted point cloud shortest path query algorithm *PC-SP-Adapt(HM)* on the original point cloud are identical (since $|\Pi(s,t|H)| = |\Pi(s,t|P)|$). The paths in Figures 16 (a) and (b) are similar to the green path in Figure 16 (d) computed by the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-ESSP-Adapt(HM)* on the original *TIN* (since $|\Pi(s,t|H)| \leq \alpha \cdot |\Pi(s,t|T)|$), but computing the former path is much quicker. The distance error ratios of the paths in Figures 16 (a) and (b) are smaller than that of the purple (network) path in Figure 16 (d) computed by the best-known approximate *TIN* shortest network path query algorithm *TIN-ESSP* on the original *TIN* (since $|\Pi(s,t|H)| \leq |\Pi_N(s,t|T)|$). The paths in Figures 16 (a) and (b) are similar to the paths in Figures 16 (e), (f), (g) and (h) computed by our algorithm *HM-SP* on the simplified height maps, but computing the latter four paths are quicker due to the simplified height maps. The path in Figures 16 (e) and (f) are similar to the green path in Figure 16 (i) on a simplified point cloud (generated by the best-known adapted point cloud simplification algorithm *PC-Simplify-Adapt(HM)*) and the green (surface) path in Figure 16 (j) on a simplified *TIN* (generated by the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)*). (2 & 3) Given a point cloud or a *TIN*, the path results are the same, since only data conversion is involved in the beginning of the algorithm.

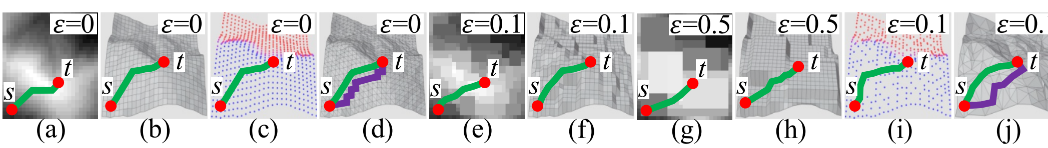

Fig. 16. Paths passing on original/simplified height maps (in bird's view), point clouds and *TIN*s

5.2.7 ***Summary***. On a height map with 50k cells and 10k objects, *HM-Simplify*'s simplification time and output size are 250s ≈ 4.6 min and 0.07MB, which are up to 21 times and 5 times (resp. 412 times and 7 times) better than the best-known adapted point cloud (resp. *TIN*) simplification algorithm *PC-Simplify-Adapt(HM)* (resp. *TIN-SSimplify-Adapt(HM)*). Performing *kNN* query on our simplified height map takes 50s, which is up to 5 times and 1,340 times smaller than on the simplified point cloud and on the simplified *TIN*, respectively. Although the 3D surfaces for GF_h and BH_h datasets are peaky, and for LM_h, RM_h and EP_h datasets are staircase-like, *HM-Simplify* and *HM-SP* outperform other baselines no matter how the 3D surfaces look like.

6 Conclusion

We propose an efficient height map simplification algorithm *HM-Simplify*, that outperforms the best-known adapted algorithm concerning the simplification time and output size. We also propose an efficient shortest path algorithm *HM-SP* on the original/simplified height map, and design algorithms for answering *kNN* and range queries on the original/simplified height map. For future work, we can explore how to simplify an updated height map. That is, given an original height map, we first simplify it. When the original height map is updated, we also update the simplified height map, such that there is no need to perform simplification on the updated height map from scratch for time-saving.

Acknowledgements

We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Yinzhao Yan and Raymond Chi-Wing Wong is supported by GZSTI16EG24.

References

[1] 2017. *1.5 Billion the world's most expensive imaging satellite nisar.* https://syntheticapertureradar.com/nasa-isro-sar-project-nisar/

[2] 2025. *125 Years of topographic mapping.* https://www.esri.com/news/arcnews/fall09articles/125-years.html

[3] 2025. *50 Years of Landsat Science.* https://landsat.gsfc.nasa.gov

[4] 2025. *Avalanche.* https://en.wikipedia.org/wiki/Avalanche

[5] 2025. *Climate and average weather year round in Gates of the Arctic National Park.* https://weatherspark.com/y/150336/Average-Weather-in-Gates-of-the-Arctic-National-Park-Alaska-United-States-Year-Round

[6] 2025. *Data geocomm.* http://data.geocomm.com/

[7] 2025. *The earth.* https://www.nationsonline.org/oneworld/earth.htm

[8] 2025. *Geocento commercial sar satellite imagery resolutions 2022, by cost per scene.* https://www.statista.com/statistics/1293899/geocento-commercial-satellite-sar-imagery-resolution-cost-worldwide/

[9] 2025. *Google Earth.* https://earth.google.com/web

[10] 2025. *Google Map.* https://www.google.com/maps

[11] 2025. *Google Maps Platform.* https://mapsplatform.google.com

[12] 2025. *Gunnison national forest.* https://gunnisoncrestedbutte.com/visit/places-to-go/parks-and-outdoors/gunnison-national-forest/

[13] 2025. *The history of point cloud development.* https://www.linkedin.com/pulse/history-point-cloud-development-bimprove/

[14] 2025. *How satellites work.* https://science.howstuffworks.com/satellite10.htm

[15] 2025. *Laramie mountain.* https://www.britannica.com/place/Laramie-Mountains

[16] 2025. *Metaverse.* https://about.facebook.com/meta

[17] 2025. *Open digital elevation model.* https://www.opendem.info/

[18] 2025. *Preferred walking speed.* https://en.wikipedia.org/wiki/Preferred_walking_speed

[19] 2025. *Robinson mountain.* https://www.mountaineers.org/activities/routes-places/robinson-mountain

[20] 2025. *Shuttle radar topography mission.* https://www.earthdata.nasa.gov/data/instruments/srtm

[21] 2025. *Transparent pricing for commercial Earth observation imagery.* https://skyfi.com/en/pricing

[22] 2025. *Zoom earth.* https://zoom.earth/maps/

[23] Segun M Adedapo and Hamdi A Zurqani. 2024. Evaluating the performance of various interpolation techniques on digital elevation models in highly dense forest vegetation environment. *Ecological Informatics (EI)* 81 (2024).

[24] Pankaj K Agarwal and Pavan K Desikan. 1997. An efficient algorithm for terrain simplification. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 139–147.

[25] Mehul Arora, Louis Wiesmann, Xieyuanli Chen, and Cyrill Stachniss. 2021. Mapping the static parts of dynamic scenes from 3d lidar point clouds exploiting ground segmentation. In *IEEE European Conference on Mobile Robots (ECMR)*. 1–6.

[26] Mehul Arora, Louis Wiesmann, Xieyuanli Chen, and Cyrill Stachniss. 2023. Static map generation from 3d lidar point clouds exploiting ground segmentation. *Robotics and Autonomous Systems (RAS)* 159 (2023), 104287.

[27] Leizhen Cai. 1994. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics (DAM)* 48, 2 (1994), 187–194.

[28] Jindong Chen and Yijie Han. 1990. Shortest paths on a polyhedron. In *Symposium on Computational Geometry (SOCG)*. New York, NY, USA, 360–369.

[29] The Conversation. 2025. *How is snowfall measured? A meteorologist explains how volunteers tally up winter storms.* https://theconversation.com/how-is-snowfall-measured-a-meteorologist-explains-how-volunteers-tally-up-winter-storms-175628

[30] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.

[31] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-nn query processing. In *IEEE International Conference on Data Engineering (ICDE)*. 78–78.

[32] Ke Deng, Xiaofang Zhou, Heng Tao Shen, Qing Liu, Kai Xu, and Xuemin Lin. 2008. A multi-resolution surface distance model for k-nn query processing. *The VLDB Journal (VLDBJ)* 17 (2008), 1101–1119.

[33] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[34] Weili Fang, Weiya Chen, Peter ED Love, Hanbin Luo, Haiming Zhu, and Jiajing Liu. 2024. A status digital twin approach for physically monitoring over-and-under excavation in large tunnels. *Advanced Engineering Informatics (AEI)* 62 (2024), 102648.

[35] Haoan Feng, Yunting Song, and Leila De Floriani. 2024. Critical features tracking on triangulated irregular networks by a scale-space method. In *ACM International Conference on Advances in Geographic Information Systems (GIS)*. 54–66.

[36] Fresh Off The Grid. 2025. *Winter hiking 101: everything you need to know about hiking in snow.* https://www.freshoffthegrid.com/winter-hiking-101-hiking-in-snow/

[37] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A graph-based approach for trajectory similarity computation in spatial networks. In *ACM Conference on Knowledge Discovery & Data Mining (SIGKDD)*. 556–564.

[38] Xian-Feng Han, Huixian Cheng, Hang Jiang, Dehong He, and Guoqiang Xiao. 2024. Pcb-randnet: rethinking random sampling for lidar semantic segmentation in autonomous driving scene. In *IEEE International Conference on Robotics and Automation (ICRA)*. 4435–4441.

[39] Nadine Hobeika, Laurens van Rijssel, Maarit Prusti, Constantijn Dinklo, Denis Giannelli, Balázs Dukai, Arnaud Kok, Rob van Loon, René Nota, and Jantien Stoter. 2024. Automated noise modelling using a triangulated terrain model. *Geo-Spatial Information Science (GSIS)* 27, 6 (2024), 1893–1913.

[40] Hugues Hoppe. 1996. Progressive meshes. In *Conference on Computer Graphics and Interactive Techniques (CGIT)*. 99–108.

[41] Songhua Hu, Mingyang Chen, Yuan Jiang, Wei Sun, and Chenfeng Xiong. 2022. Examining factors associated with bike-and-ride (bnr) activities around metro stations in large-scale dockless bikesharing systems. *Journal of Transport Geography (JTG)* 98 (2022), 103271.

[42] Bo Huang, Victor Junqiu Wei, Raymond Chi-Wing Wong, and Bo Tang. 2023. Ear-oracle: on efficient indexing for distance queries between arbitrary points on terrain surface. In *ACM International Conference on Management of Data (SIGMOD)*, Vol. 1. ACM New York, NY, USA, 1–26.

[43] Md Kamruzzaman, Tanzila Islam, and Smrity Rani Poddar. 2014. Accuracy of handheld gps compairing with total station in land use survey: a case study in RUET campus. *International Journal of Innovation and Applied Studies (IJIAS)* 7, 1 (2014), 343.

[44] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 9. 168–179.

[45] Manohar Kaul, Raymond Chi-Wing Wong, Bin Yang, and Christian S Jensen. 2013. Finding shortest paths on terrains by killing two birds with one stone. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 7. 73–84.

[46] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*, Vol. 30. 527–562.

[47] Jinheng Li, Feng Shuang, Junjie Huang, Tao Wang, Sijia Hu, Junhao Hu, and Hanbo Zheng. 2023. Safe distance monitoring of live equipment based upon instance segmentation and pseudo-LiDAR. *IEEE Transactions on Power Delivery (TPD)* 38, 4 (2023), 2953–2964.

[48] Shujuan Li, Junsheng Zhou, Baorui Ma, Yu-Shen Liu, and Zhizhong Han. 2024. Learning continuous implicit field with local distance indicator for arbitrary-scale point cloud upsampling. In *AAAI Conference on Artificial Intelligence*, Vol. 38. 3181–3189.

[49] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *ACM International Conference on Management of Data (SIGMOD)*. 433–444.

[50] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.

[51] Hoong Kee Ng, Hon Wai Leong, and Ngai Lam Ho. 2004. Efficient algorithm for path-based range query in spatial databases. In *IEEE International Database Engineering and Applications Symposium (IDEAS)*. 334–343.

[52] Eunjin Oh. 2020. Shortest-path queries in geometric networks. In *International Symposium on Algorithms and Computation (ISAAC)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 52–1.

[53] Michael Recla and Michael Schmitt. 2024. The sar2height framework for urban height map reconstruction from single sar intensity images. *Journal of Photogrammetry and Remote Sensing (ISPRS)* 211 (2024), 104–120.

[54] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks. In *IEEE International Conference on Data Engineering (ICDE)*. 652–663.

[55] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path oracles for spatial networks. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 2. 1210–1221.

[56] Olga Schukina, Maksud Abdukarimov, and Albina Valieva. 2024. Creating a 3d terrain model for the territory of the hasti-imom ensemble in tashkent, uzbekistan. In *E3S Web of Conferences*, Vol. 497. EDP Sciences, 02024.

[57] National Park Service. 2025. *Gates of the Arctic.* https://www.nps.gov/gaar/index.htm

[58] National Weather Service. 2025. *Measuring Snow.* https://www.weather.gov/dvn/snowmeasure

[59] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 1. 1020–1031.

[60] Yixin Sun, Yusen Luo, Qian Zhang, Lizhang Xu, Liying Wang, and Pengpeng Zhang. 2022. Estimation of crop height distribution for mature rice based on a moving surface and 3d point cloud elevation. *Agronomy* 12, 4 (2022), 836.

[61] Farhan Tauheed, Laurynas Biveinis, Thomas Heinis, Felix Schurmann, Henry Markram, and Anastasia Ailamaki. 2012. Accelerating range queries for brain simulations. In *IEEE International Conference on Data Engineering (ICDE)*. 941–952.

[62] Tao Wang, Lianbin Deng, Yuhong Li, and Hao Peng. 2021. Progressive tin densification with connection analysis for urban lidar data. *Photogrammetric Engineering & Remote Sensing (PERS)* 87, 3 (2021), 205–213.
[63] Xixun Wang, Yoshiki Mizukami, Makoto Tada, and Fumitoshi Matsuno. 2021. Navigation of a mobile robot in a dynamic environment using a point cloud map. *Artificial Life and Robotics (ALR)* 26 (2021), 10–20.
[64] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance oracle on terrain surface. In *ACM International Conference on Management of Data (SIGMOD)*. New York, NY, USA, 1211–1226.
[65] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2022. Proximity queries on terrain surface. *ACM Transactions on Database Systems (TODS)* (2022).
[66] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, David M Mount, and Hanan Samet. 2024. On efficient shortest path computation on terrain surface: a direction-oriented approach. *IEEE Transactions on Knowledge & Data Engineering (TKDE)* 1 (2024), 1–14.
[67] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Transactions on Graphics (TOG)* 28, 4 (2009), 1–8.
[68] Songhua Xing, Cyrus Shahabi, and Bei Pan. 2009. Continuous monitoring of nearest neighbors on land surface. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 2. 1114–1125.
[69] Da Yan, Zhou Zhao, and Wilfred Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 942–951.
[70] Yinzhao Yan and Raymond Chi-Wing Wong. 2021. Path advisor: a multi-functional campus map tool for shortest path. In *International Conference on Very Large Data Bases (VLDB)*, Vol. 14. 2683–2686.
[71] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Efficient shortest path queries on 3d weighted terrain surfaces for moving objects. In *IEEE International Conference on Mobile Data Management (MDM)*.
[72] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Proximity queries on point clouds using rapid construction path oracle. In *ACM International Conference on Management of Data (SIGMOD)*, Vol. 2. 1–26.
[73] Yinzhao Yan and Raymond Chi-Wing Wong. 2025. Efficient proximity queries on simplified height map (technical report). https://github.com/yanyinzhao/HeightMapPathCode/blob/main/TechnicalReport.pdf
[74] Yinzhao Yan, Raymond Chi-Wing Wong, and Christian S Jensen. 2024. An efficiently updatable path oracle for terrain surfaces. *IEEE Transactions on Knowledge & Data Engineering (TKDE)* 1 (2024), 1–14.
[75] Hongchuan Yu, Jian J Zhang, and Zheng Jiao. 2014. Geodesics on point clouds. *Mathematical Problems in Engineering (MPE)* 2014 (2014).
[76] Yating Zhang, Hongbo Tan, Qi Jiao, Zhihao Lin, Zesen Fan, Dengming Xu, Zheng Xiang, Rob Law, and Tianxiang Zheng. 2024. A Predictive Model Based on TripAdvisor Textual Reviews: Early Destination Recommendations for Travel Planning. *SAGE Open* 14, 2 (2024).
[77] Yifei Zhang and Shiyuan Wang. 2023. A robot navigation system in complex terrain based on statistical features of point clouds. *IEEE Transactions on Intelligent Vehicles (TIV)* (2023).

Received January 2025; revised April 2025; accepted May 2025