Efficient Proximity Queries on Simplified Height Maps

Anonymous
Anonymous
Anonymous

Anonymous
Anonymous
Anonymous

ABSTRACT

The extensive use of geo-spatial positioning with height maps highlights their advantages over other 3D surface representations (e.g., point clouds and *Triangular Irregular Networks*, i.e., *TIN*s (representing terrain surfaces)). In terms of proximity queries, i.e., *shortest path queries*, *k-Nearest Neighbor* (*kNN*) *queries* and *range queries*, all existing shortest path query algorithms on height maps, point clouds and *TIN*s are inefficient. Additionally, there is no existing study focusing on simplifying height maps. To address this, we propose an efficient ϵ-approximate algorithm that can simplify a height map and answer shortest path queries on the simplified height map. We also propose ϵ-approximate algorithms for answering *kNN* and range queries on the simplified height map. Our experiments show that our algorithm is up to 412 times, 7 times and 1,340 times better than the best-known *TIN* simplification algorithm concerning the simplification time, output size and shortest path query time, respectively. Performing *kNN* and range queries on our simplified height map are both up to 153 times and 1,340 times quicker than the best-known algorithms on a point cloud and a simplified *TIN* with an error at most 12%, respectively.

ACM Reference Format:
Anonymous and Anonymous. 2025. Efficient Proximity Queries on Simplified Height Maps. In *Proceedings of 2026 International Conference on Management of Data (SIGMOD '26)*. ACM, New York, NY, USA, 34 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Shortest path queries on a 3D surface have gained significant attention in both industry and academia [53]. In industry, Google Earth [4] and Metaverse [10] employ shortest paths passing on 3D surfaces (such as Earth and virtual reality) to enhance user navigation. In academia, this topic is a key research area in databases [18, 45–47, 50–54]. Various 3D surface representations, including height maps, point clouds and *Triangular Irregular Networks*, i.e., *TIN*s (representing terrain surfaces), are considered. Shortest path queries on a point cloud and a *TIN* have been studied in the studies in [46, 49, 50, 53], yet it is a new area concerning height maps. For example, in bushfire fighting or earthquake rescue [27, 44], we can use satellites to obtain the elevation of a 1km^2 affected region and generate a height map in total 10s [37], and perform shortest path queries for rescuing. We can also use the elevation of the same region and generate a point cloud with the same time, and use the corresponding height map or point cloud to obtain a *TIN* [22, 34, 53, 55] in 50s (shown by our experiments).

Proximity queries: In addition to the shortest path query, a fundamental type of *proximity query*, other proximity queries include *k-Nearest Neighbor* (*kNN*) *queries* [43] and *range queries* [36]. They are used for finding all shortest paths from a query object to its k nearest objects or to all objects within a specified range. For example, in bushfire fighting, we need to compute shortest paths from the center of the bushfires to k nearest fire stations, or to all fire stations within a certain radius, for rescuing.

Height map, point cloud, *TIN* and simplified height map: (1) A *height map* consists of a set of *pixels* storing elevation values. Figure 1 (a) shows a satellite model of a USA's ski hill, called Mount Rainier [39], in 20km × 20km region, and Figure 1 (b) shows the corresponding height map consisting of 63 pixels, where a lighter pixel color indicating a higher elevation value. Figure 1 (c) shows a height map's *conceptual graph* (stored in the memory and used for proximity queries), where the *vertices* represent the pixels of the height map, and the *edges* connect each vertex with its 8 neighbouring vertices in the x-y plane. (2) A *point cloud* consists of a set of 3D *points*. Figure 1 (d) is a point cloud of Mount Rainier with 63 points. A point cloud's conceptual graph is the same as that of a height map's one in Figure 1 (c). (3) A *TIN* consists of a set of triangulated *faces*. Each face has three *edges* that connect at three *vertices*. The gray surfaces in Figure 1 (e) is a *TIN* of Mount Rainier with 63 vertices, 158 edges and 96 faces. (4) A *simplified height map* has fewer pixels than the original one. Figure 1 (f) is a simplified height map of Figure 1 (b) with 9 pixels. Regarding different paths, Figure 1 (b) shows paths passing on a height map, Figure 1 (c) shows paths passing on a height map or point cloud('s conceptual graph), Figure 1 (d) shows paths passing on a point cloud, Figure 1 (e) shows *surface paths* [29] (with green line) passing on a *TIN*('s faces) and *network paths* [29] (with purple line) passing on a *TIN*('s edges), and Figure 1 (f) shows paths passing on a simplified height map.

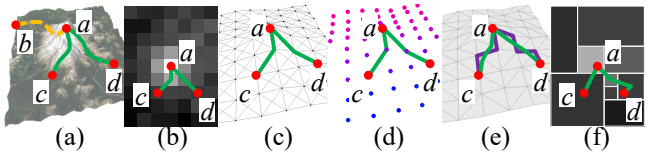

Figure 1: Paths passing on (a) a satellite mode, (b) a height map, (c) a height map's and point cloud's conceptual graph, (d) a point cloud, (e) a *TIN* and (f) a simplified height map

1.1 Motivation

1.1.1 ***Height maps' advantages.*** Height maps have several advantages compared with point clouds, *TIN*s and graphs.

SIGMOD '26, May 31 – June 5, 2026, Bengaluru India

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

(1) *Longer history and wider usage of height maps compared with point clouds.* Height maps and point clouds were introduced in 1884 [1] and 1960 [7], respectively. So, more height map datasets are available compared to point cloud datasets, leading to broader application of the former. In OpenDEM [11], 50 million height map datasets and 20 million point cloud datasets are available.

(2) *Easier visualization and path navigation of height maps compared with point clouds, TINs and graphs.* Visualizing a 2D image-based height map is easy using simple applications, even on mobile devices. But, visualizing a 3D point cloud, *TIN*, or graph requires specialized software such as Blender [2] and MeshLab [9]. For rescue teams, skiers, or mountain travelers, using a 2D height map for navigation after finding the shortest path (incorporating 3D elevation) is more convenient, since we move in the direction parallel to the ground (consider the navigation features in Google Map [5]). Navigating with a 3D point cloud, *TIN*, or graph is more complex since we are not interested in moving in the direction perpendicular to the ground.

(3) *Lower hard disk consumption of height maps compared with TINs and graphs.* In hard disks, we store (i) the pixels of a height map, but store (ii) the vertices, edges and faces of a *TIN*, and (iii) the vertices and edges of a graph. Our experiments show that storing a height map with 25M pixels needs 75MB, while storing a *TIN* and a graph generated from the same height map needs 1.7GB and 980MB, respectively. The hard disk consumption of point clouds is similar to height maps.

(4) *Smaller query time of shortest paths passing on height maps compared with shortest surface paths passing on TINs.* Computing the shortest path passing on a height map is quicker than computing the shortest surface path passing on a *TIN* generated from this height map, since a height map has a simpler structure than a *TIN*. Our experiments show that computing the former path on a height map containing 0.5M pixels needs 3s, while computing the latter path on the corresponding *TIN* needs 280s ≈ 4.6 min.

(5) *Smaller distance error of shortest paths passing on height maps compared with shortest network paths passing on TINs.* In Figures 1 (b) and (e), the shortest path passing on a height map exhibits similarities with the shortest surface path passing on a *TIN*. But, in Figure 1 (e), the shortest surface and network paths passing on a *TIN* differ significantly. Our experiments show that compared with the shortest surface path passing on a *TIN*, the length of the shortest path passing on a height map and the shortest network path passing on a *TIN* are 1.06 times and 1.45 times larger, respectively.

1.1.2 ***Simplify a height map.***

While computing shortest paths on a height map is fast, we can further increase query speed by simplifying the height map (the time required to simplify the height map is termed the *simplification time*, the storage complexity of the simplified height map is termed the *output size*, and the time required to find the shortest path on the simplified height map is termed the *shortest path query time*). We formulate *height map simplification problem*, which describes that given a height map and an *error parameter* $\epsilon \geq 0$, we find an ϵ-approximate simplified height map with a minimum number of pixels, such that the shortest distances between any pair of pixels on the simplified height map are ϵ-approximations of the distances on the original height map.

1.1.3 ***Case study.***

We conducted a case study on a snowfall evacuation at Mount Rainier [40]. In Figure 1 (a), due to each hotel's capacity constraints, we aim to identify shortest paths from the position a of some skiers to k-nearest hotels b, c, d, where c and d are k-nearest options when $k = 2$. Our experiments show that given a height map with 50k pixels, 10k possible skier positions and 50 hotels, we can simplify it in 250s ≈ 4.6 min, and compute shortest paths between each skier position and its k-nearest hotels in 50s on the simplified height map. But, it needs 7,590s ≈ 2.1 hours on the original height map, 7,630s ≈ 2.1 hours on a point cloud (constructed based on the original height map), and 380,000s ≈ 4.3 days on a *TIN* (constructed based on the original height map). Since 250s + 50s = 300s < 7,590s, the simplification process is useful.

1.2 Challenges

1.2.1 ***Inefficient of height map shortest path query algorithm.***

Existing studies [24, 38] for conducting proximity queries on a height map are slow. They first use $O(n)$ time to construct a point cloud or *TIN* using the given height map (n is the number of height map pixels), and then compute the shortest path passing on the point cloud or *TIN*. The best-known *exact* point cloud shortest path query algorithm operates in $O(n \log n)$ time [53]. The best-known *exact* and *approximate TIN* shortest surface path query algorithms operate in $O(n^2)$ time [15, 47, 54] and $O((n+n') \log(n+n'))$ time [28, 52] (n' is the number of supplementary points added for error guarantee), respectively. The best-known *approximate TIN* shortest network path query algorithm operates in $O(n \log n)$ time [29]. We adapt algorithms [15, 28, 29, 47, 52–54] on a height map by constructing a point cloud or *TIN* using the height map, and then computing the shortest path passing on the point cloud or *TIN*. Our experiments show for a height map with 50k pixels and 10k objects, answering *kNN* queries takes (1) 7,630s ≈ 2.1 hours for adapted algorithm [53], (2) 380,000s ≈ 4.3 days for adapted algorithm [15, 47, 54], (3) 70,000s ≈ 19.4 hours for adapted algorithm [28, 52], and (4) 33,000s ≈ 9.2 hours for adapted algorithm [29].

1.2.2 ***No height map simplification algorithm.***

There is no existing study focusing on simplifying a height map. Although algorithms [19, 25, 29, 32] can simplify a *TIN*, they cannot be used for simplifying a height map directly, since they iteratively remove a vertex v in the *TIN* and use *triangulation* [34] to fill the hole formed by the adjacent vertices of v, which does not apply to a height map. We can adapt these algorithms for height map simplification by constructing a *TIN* using the height map, and utilizing these algorithms on this *TIN*. But, their simplified *TIN*s' output size is large since they do not consider any optimization techniques, resulting in large shortest path query times on their simplified *TIN*s. In addition, their simplification times are large due to the lengthy shortest path query time on a *TIN* and the large number of distance calculations required during simplification. Our experiments show that after constructing a *TIN* from the provided height map, the best-known *TIN* simplification algorithm [26, 29] requires 103,000s ≈ 1.2 days to simplify a *TIN* with 50k vertices, and the *kNN* query time of 10k objects on the simplified *TIN* is 67,000s ≈ 18.6 hours. There is no existing study focusing on simplifying a point cloud.

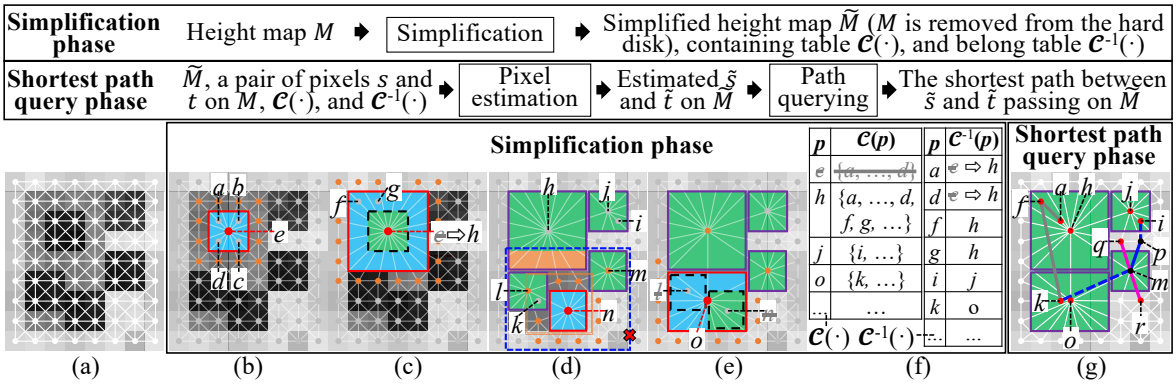

Figure 2: Algorithm *HM-MemSimQ*

1.3 Contribution and Organization

Our major contributions are as follows.

(1) We show the height map simplification problem is *NP-hard*, indicating that no efficient algorithm available can solve it *exactly*.

(2) We propose the first ϵ-approximate height map simplification algorithm *Height Map Memory saving Simplification and simplified height map shortest path Query* (*HM-MemSimQ*). In the text description in Figure 2, given a height map M, it first generates a simplified height map $\widetilde{M}$ during the *simplification phase* (and removes M from the hard disk). Then, given a pair of pixels s and t, it answers the shortest path between estimated $\tilde{s}$ and $\tilde{t}$ (calculated based on s, t and $\widetilde{M}$, since M is removed) passing on $\widetilde{M}$ during the *shortest path query phase* (where the containing and belong tables are two assistant components for computation). It has leading performance concerning (i) the output size and shortest path query time by the memory saving technique, since it can considerably reduce the number of pixels in $\widetilde{M}$ to save memory and to enhance the efficiency of shortest path queries on $\widetilde{M}$, and (ii) the simplification time by a newly proposed height map shortest path query algorithm *Height Map Efficient shortest path Query* (*HM-EffQ*) and the pruning of unnecessary checks. We also design efficient algorithms for answering *kNN* and range queries on $\widetilde{M}$.

(3) We offer theoretical analysis on (i) the simplification time, output size, shortest path query time and error guarantee of algorithm *HM-MemSimQ*, (ii) the shortest path query time, memory usage and error guarantee of algorithm *HM-EffQ*, and (iii) the *kNN* and range query time and error guarantee for *kNN* and range queries.

(4) Algorithm *HM-MemSimQ* outperforms the best-known adapted *TIN* simplification algorithm [26, 29] concerning the simplification time, output size and shortest path query time. The proximity query time on the simplified height map also performs much better than the best-known adapted algorithm on a point cloud [53] and a simplified *TIN* [29, 46]. Our experiments show that given a height map with 50k pixels, the simplification time and output size are 250s ≈ 4.6 min and 0.07MB for algorithm *HM-MemSimQ*, but are 103,000s ≈ 1.2 days and 0.5MB for the best-known adapted *TIN* simplification algorithm [26, 29]. The *kNN* and range query time of 10k objects on the simplified height map are both 50s for *HM-MemSimQ*, but are 7,630s ≈ 2.1 hours for the best-known adapted algorithm on a point cloud [53], and 67,000s ≈ 18.6 hours for the best-known adapted algorithm on a simplified *TIN* [29, 46].

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

2 PRELIMINARY

2.1 Notation and Definitions

2.1.1 ***Height map.*** Consider a height map $M = (P, N(\cdot))$ contains a set of *pixels* denoted as P, and a *neighbour pixels* table denoted as $N(\cdot)$ (i.e., a *hash table* [17]). Each pixel $p \in P$ has two 2D coordinate values (represent its x- and y-coordinate values) and a grayscale color integer value (represents its *elevation value* with range $[0, 255]$), denoted as $p.x$, $p.y$ and $p.z$, respectively. We use a point position at the center of a pixel to denote this pixel. For each pixel $p \in P$, $N(p)$ returns the *neighbour pixels* of p in $O(1)$ time, and $N(p)$ is initialized to be the nearest upper, lower, leftward, rightward, upper-left, upper-right, lower-left and lower-right pixels of p on M. We need to store P in hard disks and do not need to store $N(\cdot)$ since we can use P to derive it. Let n be the number of pixels of M. In Figure 3 (a), 9 pixels represent a height map M, 6 orange points and 2 red points denote pixels in $N(p)$.

Let G be M's conceptual graph, and let $G.V$ and $G.E$ be G's vertices and edges, respectively. Each vertex $v \in G.V$ has the x-, y- and z-coordinate values equal to $p.x$, $p.y$ and $p.z$ of each pixel $p \in P$, respectively. $G.E$ contains a set of edges between each $p \in P$ and $p' \in N(q)$, where each edge has a weight equal to the Euclidean distance between its two vertices. The graphs in Figures 3 (a) and (b) are 2D and 3D conceptual graphs of M. Given a pair of pixels s and t on M, let $\Pi(s, t|M)$ be the *shortest path* between them passing on M('s conceptual graph G). Let $|\cdot|$ be a path's length (e.g., $\Pi(s, t|M)$'s length is $|\Pi(s, t|M)|$). Figures 3 (a) and (b) show $\Pi(s, t|M)$ in green line.

2.1.2 ***Point cloud and TIN.*** Let C be a point cloud containing the set of points in $G.V$, and let C's conceptual graph also be G. Given a pair of points s and t on C, let $\Pi(s, t|C)$ be the *shortest path* between them passing on C('s conceptual graph G). Let T be a *TIN*

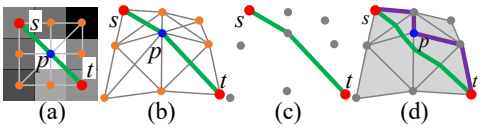

Figure 3: (a) $\Pi(s,t|M)$ passing on a height map M, (b) $\Pi(s,t|M)$ passing on M's conceptual graph, (c) $\Pi(s,t|C)$ passing on a point cloud C, and (d) $\Pi(s,t|T)$ (with green line) and $\Pi_N(s,t|T)$ (with purple line) passing on a *TIN* T

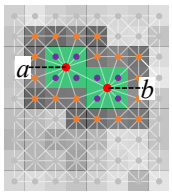

Figure 4: Distance checking

triangulated [22, 34, 55] by the vertices in $G.V$. 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* and *shortest network path* between them passing on T, whose distances are called the shortest surface and network distance, respectively. Let θ be the smallest interior angle of a triangle of T. Figure 3 (c) shows a point cloud constructed by M with $\Pi(s,t|C)$ in green line, and Figure 3 (d) shows a *TIN* constructed by M with $\Pi(s,t|T)$ in green line and $\Pi_N(s,t|T)$ in purple line.

2.1.3 ***Simplified height map.*** Given a height map M, we can obtain a simplified height map $\widetilde{M} = (\widetilde{P}, \widetilde{N}(\cdot))$ by merging some adjacent pixels (deleting these pixels and adding a new pixel that cover these pixels for replacement) in M, where $\widetilde{P}$ and $\widetilde{N}(\cdot)$ are initialized as P and $N(\cdot)$, and are updated during simplification. We need to store $\widetilde{P}$ and $\widetilde{N}(\cdot)$ in hard disks. We can perform the merge if $\widetilde{M}$ is an ϵ-approximate simplified height map of M after merging. Figures 2 (a) and (b & c) are original and simplified height maps.

Let a *deleted* (resp. *remaining*) pixel be a pixel in M that is deleted from (resp. remaining in) $\widetilde{M}$. Let an *added* pixel be a pixel in $\widetilde{M}$ that covers some adjacent deleted and/or previously added pixels, where these adjacent deleted pixels *belong* to the added pixel. A *property of a deleted pixel* is that each deleted pixel only belongs to one added pixel. In Figure 2 (b), we merge $\{a,b,c,d\}$ to be e, 12 orange points denote pixels in $\widetilde{N}(e)$, $\{a,b,c,d\}$ are deleted pixels, all other pixels in P except $\{a,b,c,d\}$ are remaining pixels, e is an added pixel, and $\{a,b,c,d\}$ belong to e. The coordinate and elevation values of the added pixel are weighted average values of those of the adjacent deleted pixels (if these adjacent deleted pixels contain a previously added pixel, the weight is the number of pixels in M belonging to this previously added pixel; otherwise, the weight is 1). In Figures 2 (c), we merge $\{e,f,g,\dots\}$ to be h, the weight of e is 4 when computing the coordinate and elevation values of h, since the number of pixels in M belonging to e is 4. We denote a set of remaining pixels and added pixels as P_{rema} and P_{add}, so $\widetilde{P} = P_{rema} \cup P_{add}$. A set of deleted pixels is denoted as $P - P_{rema}$. Given a pixel $p \in M$, let $\widetilde{p} \in \widetilde{M}$ be an *estimated pixel* of p, such that $\widetilde{p}.x = p.x$, $\widetilde{p}.y = p.y$, $\widetilde{p}.z$ is the elevation value of the added pixel such that p belongs to (if p is a deleted pixel), or $\widetilde{p} = p$ (if p is a remaining pixel). In Figure 2 (a), since a is a deleted pixel, a belongs to e, we have $\widetilde{a}.x = a.x$, $\widetilde{a}.y = a.y$ and $\widetilde{a}.z = e.z$.

Similar to G, let $\widetilde{G}$ be $\widetilde{M}$'s simplified conceptual graph, and let $\widetilde{G}.V$ and $\widetilde{G}.E$ be $\widetilde{G}$'s vertices and edges, respectively. We just need to use $\widetilde{P}$ and $\widetilde{N}(\cdot)$ to substitute P and $N(\cdot)$ in the definition of $G.V$ and $G.E$, to obtain $\widetilde{G}.V$ and $\widetilde{G}.E$. The graphs in Figures 2 (a) and (b) are original and simplified conceptual graphs of the corresponding height maps. Given a pair of pixels $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{M}$, let $\Pi(\widetilde{s},\widetilde{t}|\widetilde{M})$ be the *shortest path* between them passing on $\widetilde{M}$('s simplified conceptual graph $\widetilde{G}$). Figure 2 (g) shows $\Pi(\widetilde{i},\widetilde{k}|\widetilde{M})$ in blue line. A notation table can be found in the appendix of Table 3.

2.2 Problem

Property 1 describes ϵ-approximate simplified height map property, Problem 1 describes the height map simplification problem, and Theorem 2.1 shows the *NP-hardness* of the problem.

Property 1 (ϵ-Approximate Simplified Height Map Property). *Given M, $\widetilde{M}$ and ϵ, $\widetilde{M}$ is an ϵ-approximate simplified height map of M if and only if for all pairs of pixels s and t on M,*

$$(1-\epsilon)|\Pi(s,t|M)| \le |\Pi(\widetilde{s},\widetilde{t}|\widetilde{M})| \le (1+\epsilon)|\Pi(s,t|M)|. \quad (1)$$

Problem 1 (Height Map Simplification Problem). *Given M, a non-negative integer i and ϵ, can we find an ϵ-approximate simplified height map $\widetilde{M}$ of M with at most i pixels?*

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

Proof Sketch. We can transform Minimum T-Spanner Problem [14] (*NP-complete* problem) to the Height Map Simplification Problem in polynomial time, to prove it is *NP-hard*. The detailed proof appears in the appendix. □

3 RELATED WORK

3.1 Height Map Shortest Path Query Algorithms

Existing studies [24, 38] for conducting proximity queries on a height map are slow, since given a height map, they first use 2D coordinate and elevation values of pixels in the height map to construct a point cloud, or triangulate the points of the point cloud to construct a *TIN* in $O(n)$ time, and then compute the shortest path passing on the point cloud or *TIN*. There is no existing study focusing on directly addressing proximity queries on a height map. We introduce (1) *point cloud shortest path* query algorithm [53], (2) *TIN shortest surface path* query algorithm [15, 28, 31, 35, 47, 48, 52, 54], and (3) *TIN shortest network path* query algorithm [29].

3.1.1 ***Point cloud shortest path query algorithm.*** The best-known exact point cloud shortest path query algorithm *Point Cloud Conceptual graph shortest path Query* (*PC-ConQ*) [53] uses Dijkstra's algorithm on the point cloud's conceptual graph to compute the path in $O(n \log n)$ time.

3.1.2 ***TIN shortest surface path query algorithms.*** (1) *Exact algorithms*: Two studies use continuous Dijkstra's algorithm [35] and checking window algorithm [48] to compute the path both in $O(n^2 \log n)$ time. The best-known exact *TIN* shortest surface path query algorithm *TIN Unfold shortest path Query* (*TIN-UnfQ*) [15, 47, 54] uses a line segment to connect the source and destination on a 2D *TIN* (unfolded by the 3D *TIN*) to compute the path in $O(n^2)$ time. It has two variants with the same time complexity: an initial version [15] and an extended version [47] with a better experimental running time. Study [54] uses algorithm *TIN-UnfQ* as an on-the-fly algorithm to build an index.

(2) *Approximate algorithms*: All algorithms [28, 31, 52] construct a graph with discrete Steiner points (placed on a *TIN*'s edges) and the *TIN*'s vertices to compute the path. The best-known $(1+\epsilon)$-approximate *TIN* shortest surface path query algorithm *TIN efficient Steiner point shortest path Query* (*TIN-SteQ*) [28, 52] operates in $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$, where l_{max} and l_{min} are the longest and shortest edge's length of the *TIN*, respectively. Algorithm [28] and algorithm [52] run on an unweighted and weighted *TIN*, respectively. They are the same if the weight of each face in the *TIN* is 1 in algorithm [52].

3.1.3 ***TIN shortest network path query algorithm.*** The shortest network path cannot traverse the faces of a *TIN*, resulting in an approximate path. The best-known approximate *TIN* shortest network path query algorithm *TIN Dijkstra shortest path Query*, i.e., algorithm *TIN-DijQ* [29] operates in $O(n\log n)$ time.

Adaptions: (1) Given a *Height Map*, we adapt (i) the best-known exact point cloud shortest path query algorithm *PC-ConQ* [53], (ii) the best-known exact *TIN* shortest surface path query algorithm *TIN-UnfQ* [15, 47, 54], (iii) the best-known approximate *TIN* shortest surface path query algorithm *TIN-SteQ* [28, 52], and (iv) the best-known approximate *TIN* shortest network path query algorithm *TIN-DijQ* [29] to be algorithm (i) *PC-ConQ-Adapt(HM)*, (ii) *TIN-UnfQ-Adapt(HM)*, (iii) *TIN-SteQ-Adapt(HM)*, and (iv) *TIN-DijQ-Adapt(HM)*, by first constructing a point cloud or *TIN* using the height map, and then computing the shortest path passing on the point cloud or *TIN*. (2) Given a height map, if we do not construct a point cloud or *TIN* first, algorithm *TIN-UnfQ* cannot be directly adapted on the height map since no face can be unfolded in a height map. But, algorithm *PC-ConQ*, *TIN-SteQ* and *TIN-DijQ* can be directly adapted on the height map (by not applying them on the point cloud or *TIN*, but on the height map's conceptual graph), and they become algorithm *HM-EffQ* (since they are Dijkstra's algorithms).

Distance relationships: Given a pair of pixels s and t on a height map, algorithm *PC-ConQ-Adapt(HM)*, *TIN-UnfQ-Adapt(HM)* and *TIN-DijQ-Adapt(HM)* return $\Pi(s,t|C)$, $\Pi(s,t|T)$ and $\Pi_N(s,t|T)$, respectively. Since the height map and the point cloud have the same conceptual graph, we know $|\Pi(s,t|M)| = |\Pi(s,t|C)|$. According to Lemma 4.3 of study [53], we know $|\Pi(s,t|M)| \leq \alpha \cdot |\Pi(s,t|T)|$, where $\alpha = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$, and $|\Pi(s,t|M)| \leq |\Pi_N(s,t|T)|$.

Drawback: All existing algorithms are very slow, even if we pre-generate a point cloud or *TIN* using the given height map. Our experiments show that algorithm *PC-ConQ-Adapt(HM)* first needs to generate a point cloud from a height map with 50k pixels in 0.3s, and then answer *kNN* queries for all 10k objects on this point cloud in 7,630s $\approx$ 2.1 hours. In the same setting, algorithm *TIN-UnfQ-Adapt(HM)*, *TIN-SteQ-Adapt(HM)* and *TIN-DijQ-Adapt(HM)* first need to generate a *TIN* from the height map in 0.42s (= 0.3s + 0.12s, since we need 0.3s to obtain a set of vertices of the *TIN* and 0.12s to triangulate them to obtain the *TIN*), and then answer queries on this *TIN* in 380,000s $\approx$ 4.3 days, 70,000s $\approx$ 19.4 hours and 33,000s $\approx$ 9.2 hours, respectively.

3.2 Height Map Simplification Algorithms

There is no existing study focusing on simplifying a height map or a point cloud. Algorithms [19, 25, 29, 32] can simplify a *TIN* by iteratively removing a vertex v in a *TIN* and using *triangulation* [34] to form new faces among the adjacent vertices of v. But, they cannot be used to simplify a height map directly, since no vertices can be deleted nor no new faces can be created in a height map. Among these algorithms, algorithm *TIN shortest Network distance Simplification and simplified TIN shortest path Query* (*TIN-NetSimQ*) [29] is the most efficient one and can simplify a *TIN* with an error guarantee (i.e., the *shortest network distances* between all pairs of vertices on the simplified *TIN* are ϵ-approximations of the distances on the original *TIN*). We use the *shortest surface distance* as the distance metric in algorithm *TIN-NetSimQ* to obtain the best-known *TIN* simplification algorithm *TIN shortest Surface distance Simplification and simplified TIN shortest path Query* (*TIN-SurSimQ*) [26, 29].

Adaptions: Given a *Height Map*, we adapt (1) *TIN* shortest network distance simplification algorithm *TIN-NetSimQ* [29] and (2) *TIN* shortest surface distance simplification algorithm *TIN-SurSimQ* [26, 29] to be algorithm (1) *TIN-NetSimQ-Adapt(HM)* and (2) *TIN-SurSimQ-Adapt(HM)*, by first constructing a *TIN* using the height map, and then applying the corresponding algorithms for *TIN* simplification and shortest path query.

3.2.1 ***Algorithm TIN-NetSimQ-Adapt(HM).*** After obtaining a *TIN* from the height map, each *TIN* simplification iteration checks whether the shortest network distances between pairs of *adjacent* vertices of the removed vertex v on the simplified *TIN* are ϵ-approximations of the distances on the original *TIN*. Its simplification time, output size and shortest path query time are $O(n^2\log n)$, $O(n)$ and $O(n\log n)$, respectively.

3.2.2 ***Algorithm TIN-SurSimQ-Adapt(HM).*** After obtaining a *TIN* from the height map, each iteration places Steiner points on faces *adjacent* to v (using the technique in study [26] for any points-to-any points *TIN* shortest surface path query), and checks whether the shortest surface distances between all pairs of Steiner points on the simplified *TIN* are ϵ-approximations of the distances on the original *TIN*. Its simplification time, output size and shortest path query time are $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$, $O(n)$ and $O(n^2)$, respectively.

Drawbacks: (1) *Large output size and shortest path query time*: Both algorithm *TIN-NetSimQ-Adapt(HM)* and *TIN-SurSimQ-Adapt(HM)* do not have optimization techniques during *TIN* simplification, so the output size of the simplified *TIN*s are large, and the shortest path query time on their simplified *TIN*s are large. But, algorithm *HM-MemSimQ* uses several optimization techniques to reduce the output size. (2) *Large simplification time*: Since algorithm *TIN-SurSimQ-Adapt(HM)* uses the shortest surface path passing on a *TIN* and involves numerous distance calculations due to a large amount of Steiner points, its simplification time is large. But, algorithm *HM-MemSimQ* uses the shortest path passing on a height map (i.e., computationally efficient) and involves fewer distance calculations due to the pruning out of unnecessary checks. Our experiments show that for a height map with 50k pixels, the simplification time of algorithm *TIN-NetSimQ-Adapt(HM)*, *TIN-SurSimQ-Adapt(HM)* and *HM-MemSimQ* are 25,800s $\approx$ 7.2 hours, 103,000s $\approx$ 1.2 days and 250s $\approx$ 4.6 min, respectively. The *kNN* query time of 10k objects on the simplified *TIN*s or height map generated by them are 16,800s $\approx$ 4.7 hours, 67,000s $\approx$ 18.6 hours and 50s, respectively.

4 METHODOLOGY

4.1 Overview of Algorithm *HM-MemSimQ*

4.1.1 ***Two components.*** There are two components.

(1) **The containing table $C(\cdot)$**: It is a *hash table*. Given an added pixel p in $\widetilde{M}$, $C(p)$ returns the set of deleted pixels $\{p_1, p_2, \dots\}$ in M belong to p in $O(1)$ time. In Figure 2 (b), we merge $\{a, b, c, d\}$ to be e, the deleted pixels $\{a, b, c, d\}$ belongs to the added pixel e, so $C(e) = \{a, b, c, d\}$.

(2) **The belonging table $C^{-1}(\cdot)$**: It is a *hash table*. Given a deleted pixel p in M, $C^{-1}(p)$ returns the added pixel p' in $\widetilde{M}$ such that p belongs to p' in $O(1)$ time. In Figure 2 (b), the deleted pixel a belongs to the added pixel e, so $C^{-1}(a) = e$.

4.1.2 ***Two phases.*** There are two phases.

(1) **Simplification phase**: In Figures 2 (a) - (f), given a height map $M = (P, N(\cdot))$, we iteratively merge some adjacent pixels to obtain a simplified height map $\widetilde{M} = (P_{rema} \cup P_{add}, \widetilde{N}(\cdot))$, store added and deleted pixels information in $C(\cdot)$ and $C^{-1}(\cdot)$, until Property 1 does not satisfy. Note that M is removed from the hard disk.

(2) **Shortest path query phase**: In Figure 2 (g), given $\widetilde{M}$, a pair of pixels s and t on M, $C(\cdot)$ and $C^{-1}(\cdot)$, we first calculate s and t's estimated pixels $\widetilde{s}$ and $\widetilde{s}$ on $\widetilde{M}$, and then use Dijkstra's algorithm [20] to compute $\Pi(\widetilde{s}, \widetilde{t} | \widetilde{M})$.

4.2 Key Idea of Algorithm *HM-MemSimQ*

4.2.1 ***Novel memory saving technique.*** Algorithm *HM-MemSimQ* is specifically designed to save memory (i.e., reduce the output size and shortest path query time) by considerably reducing the number of pixels in $\widetilde{M}$ using a novel pixel merging technique with two merging types:

(1) **Merge four pixels**: We start by choosing four adjacent non-merged pixels in a square such that the variances of their elevation values are the smallest. In Figure 2 (b), we merge $\{a, b, c, d\}$ to be e, and obtain $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and process to the next merging type. If not, we terminate the algorithm.

(2) **Merge added pixels**: Given an added pixel p from the previous merge, we merge p with its neighbour pixels, i.e., we enlarge p by expanding its neighbour pixels in left, right, top and/or bottom directions to reduce the number of pixels in $\widetilde{M}$, where expanding left (resp. right) covers neighbours pixels with x-coordinate value smaller (resp. larger) than p, and expanding top (resp. bottom) covers neighbours pixels with y-coordinate value smaller (resp. larger) than p. 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 directions that we expand, where L, R, T, B means that we expand p to cover its neighbour pixels in the direction of *left*, *right*, *top*, *bottom*, and $\cdot$ means that we do not expand in that direction. For example, $(L, R, T, \cdot)$ means we cover p's neighbours pixels with x-coordinate value larger and smaller than p, and y-coordinate value smaller than p.

In Figure 2 (c), we merge e with $\{f, g, \dots\}$ to be h, i.e., we enlarge e by expanding into all four directions, i.e, (L, R, T, B), and obtain $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and repeat this process. If not, we go back to the *four pixels merging* type by selecting four new pixels. In Figure 2 (d), we merge n with $\{l, m, \dots\}$ to be an added pixel shown in the blue frame, i.e., we still enlarge n by expanding into all four directions. For one of n's neighbour pixels, i.e., m, we cover it as a whole to reduce the number of pixels in $\widetilde{M}$. But, this results in four orange deleted pixels belonging to both h and the newly added pixel, violating the property of the deleted pixel in Section 2.1.3 that each deleted pixel only belongs to one added pixel. So, we enlarge n by expanding into directions using other elements in Dir until we successfully enlarge n. In Figure 2 (e), we merge n with $\{l, \dots\}$ to be o, i.e., we enlarge n by expanding into the top and left directions, i.e., $(L, \cdot, T, \cdot)$, and obtain $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and repeat this process. If not, we go back to the *four pixels merging* type.

4.2.2 ***Efficient simplification.*** There are two reasons why algorithm *HM-MemSimQ* has a small simplification time.

(1) **Efficient height map shortest path query**: When checking whether Property 1 is satisfied, we use algorithm *HM-EffQ* to compute the shortest path passing on M and $\widetilde{M}$. However, it is slow to compute the shortest surface path passing on a *TIN*.

(2) **Efficient Property 1 checking**: Checking Property 1 involves checking whether Inequality 1 is satisfied for *all* pixels on M, this naive method is time-consuming. Instead, we can check distances only related to the *neighbour* pixels of the newly added pixels in each iteration. In Figure 4, suppose that the green pixel b is the newly added pixel, we need to check whether the distances between pairs of pixels denoted as orange and purple points satisfy Inequality 1. Intuitively, if the distances on $\widetilde{M}$ between any pair of pixels near b do not change a lot, then the distances on $\widetilde{M}$ between any pair of pixels far away from b cannot change a lot.

4.2.3 ***Efficient shortest path query.*** We illustrate why algorithm *HM-MemSimQ* has a small shortest path query time. In Figure 2 (g), given a pair of pixels i and k, we first estimate $\widetilde{i}$ and $\widetilde{k}$. Since they are deleted pixels, their x- and y-coordinate values on $\widetilde{M}$ are the same as i and k, and their elevation values is the elevation value of $C^{-1}(i) = j$ and $C^{-1}(k) = o$. If they are remaining pixels, $\widetilde{i} = i$ and $\widetilde{k} = k$. Then, we use Dijkstra's algorithm to compute the shortest path between $\widetilde{i}$ and $\widetilde{k}$ passing on $\widetilde{M}$'s conceptual graph.

4.3 Simplification Phase

In the text description of Figure 2, given M, we simplify it to obtain $\widetilde{M}$, $C(\cdot)$ and $C^{-1}(\cdot)$. Algorithms 1 and 2 show the simplification process in detail, Algorithm 2 is a sub-algorithm used for 2 times in Algorithm 1.

4.3.1 ***Detail and example for Algorithm 1.*** The following shows Algorithm 1 with an example. In each simplification iteration, let $\widehat{P} = \{p_1, p_2, \dots\}$ be a set of adjacent pixels that we need to merge, and p_{add} be an added pixel formed by merging each pixel $\widehat{P}$.

(1) *Merge four pixels* (lines 2-4): Given M in Figure 2 (a), in Figure 2 (b), we can merge $\widehat{P} = \{a, b, c, d\}$ (lines 2-3), suppose that the result of *UpdateCheck* is *True* (line 4), we obtain $p_{add} = e$ and $\widetilde{M}$.

(2) *Merge added pixels* (lines 5-9). In Figure 2 (c), $p_{add} = e$, we can merge $\widehat{P} = \{e\} \cup \{f, g, \dots\} = \{e, f, g, \dots\}$, i.e., we can enlarge e by expanding it to cover its neighbour pixels in the directions using the 1-st element (L, R, T, B) in Dir (lines 5-7), suppose that the result of *UpdateCheck* is *True* (lines 8-9), we obtain $p_{add} = h$ and

Algorithm 1 *HM-MemSimQ* (M)

Input: $M = (P, N(\cdot) = \emptyset)$
Output: $\widetilde{M}$, $C(\cdot)$ and $C^{-1}(\cdot)$
1: initialize $N(\cdot)$ using P, $P_{rema} \leftarrow P$, $P_{add} \leftarrow \emptyset$, $\widetilde{N}(\cdot) \leftarrow N(\cdot)$, $C(\cdot) \leftarrow \emptyset$, $C^{-1}(\cdot) \leftarrow \emptyset$
2: **while** in $\widetilde{M} = (P_{rema} \cup p_{add}, \widetilde{N}(\cdot))$, we can merge any four adjacent non-merged pixels in a square **do**
3: $\widehat{P} \leftarrow$ four pixels with the smallest elevation values variance, $p_{add} \leftarrow \emptyset$
4: **if** *UpdateCheck* $(P_{rema}, P_{add}, \widetilde{N}(\cdot), \widehat{P}, p_{add}, C(\cdot), C^{-1}(\cdot))$ is *True* **then**
5: **while** in $\widetilde{M} = (P_{rema} \cup p_{add}, \widetilde{N}(\cdot))$, we can enlarge p_{add} by expanding its neighbour pixels (as a whole) in the direction using any element in *Dir*, without violating the property of deleted pixels (each deleted pixel only belongs to one added pixel) **do**
6: **for** i-th element in *Dir* **do**
7: $\widehat{P} \leftarrow \{p_{add}\} \cup$ pixels after expanding neighbour pixels (as a whole) in the directions using the i-th element, without violating the property of deleted pixels
8: **if** *UpdateCheck* $(P_{rema}, P_{add}, \widetilde{N}, \widehat{P}, p_{add}, C(\cdot), C^{-1}(\cdot))$ is *Ture* **then**
9: **break**
10: **return** $\widetilde{M} = (P_{rema} \cup P_{add}, \widetilde{N}(\cdot))$, $C(\cdot)$ and $C^{-1}(\cdot)$

$\widetilde{M}$. Suppose that when we enlarge h by expanding into directions using other elements in *Dir* (lines 5-6), the result of *UpdateCheck* is always *False* (lines 8-9), we exit this loop.

(3) *Merge four pixels iteration* (lines 2-4): In Figure 2 (d), we use a similar process to obtain new pixels j, l, m and $\widetilde{M}$. Suppose that we cannot merge added pixels for j, l, m (lines 5-9), we go back to line 2 after each merging. Then, we use a similar process to obtain a new pixel n and $\widetilde{M}$.

(4) *Merge added pixels iteration* (lines 5-9): In Figure 2 (d), $p_{add} = n$, we merge $\widehat{P} = \{n\} \cup \{l, m, \dots\} = \{l, m, n, \dots\}$, i.e., we enlarge n by expanding it to cover its neighbour pixels in the directions using the 1-st element (L, R, T, B) in *Dir* (lines 5-6). For m, we cover it as a whole. We get the newly added pixel with blue frame. But, there exist four orange deleted pixels belong to both h and the newly added pixel, this violates the property of the deleted pixel, and we cannot merge them. So, we enlarge n by expanding into directions using other elements in *Dir* until we successfully enlarge n (line 6). In Figure 2 (e), $p_{add} = n$, we can merge $\widehat{P} = \{n\} \cup \{l, \dots\} = \{n, l, \dots\}$, i.e., we can enlarge n by expanding into the top and left directions using the 7-th element $(L, \cdot, T, \cdot)$ in *Dir* (lines 5-7), suppose that the result of *UpdateCheck* is *True* (lines 8-9), we obtain $p_{add} = o$ and $\widetilde{M}$.

4.3.2 ***Detail and example for Algorithm 2***. The following shows Algorithm 2 with an example. We use Figures 2 (b) and (c) for illustration, since Figures 2 (d) and (e) are similar.

(1) *Update* $C'(\cdot)$ *and* $C^{-1\prime}(\cdot)$ (lines 3-9): In Figure 2 (b), $p_{add} = e$ and $\widehat{P} = \{a, b, c, d\}$, since all pixels in $\widehat{P}$ are in P_{rema}, we have $C'(e) = \{a, b, c, d\}$, $C^{-1\prime}(a) = e, \dots, C^{-1\prime}(d) = e$. In Figure 2 (c), $p_{add} = e$, $\widehat{P} = \{e, f, g, \dots\}$ and $P_{add} = \{e\}$, since for pixels in $\widehat{P}$, e is in P_{add} and other pixels are in P_{rema}, we have $C'(h) = \{a, \dots, d, f, g, \dots\}$, $C^{-1\prime}(a) = h, \dots, C^{-1\prime}(d) = h, C^{-1\prime}(f) = h, C^{-1\prime}(g) = h, \dots$, and remove $C'(e)$.

(2) *Update neighbour pixels* (lines 10-13): In Figures 2 (b) and (c), for e and h, we update their neighbour pixels to be the pixels represented in orange points; for these neighbour pixels, we also update e and h to be their neighbour pixels.

(3) *Update* $\widetilde{M}'$ (lines 14-19): In Figure 2 (b), $\{a, b, c, d\}$ are deleted from P'_{rema} and e is added into P'_{add}, so $P'_{add} = \{e\}$. In Figure 2 (d),

Algorithm 2 *UpdateCheck* $(P_{rema}, P_{add}, \widetilde{N}(\cdot), \widehat{P}, p_{add}, C(\cdot), C^{-1}(\cdot))$

Input: $P_{rema}, P_{add}, \widetilde{N}(\cdot), \widehat{P}, p_{add}, C(\cdot)$ and $C^{-1}(\cdot)$
Output: updated $P_{rema}, P_{add}, \widetilde{N}(\cdot), C(\cdot), C^{-1}(\cdot)$, and whether the updated height map satisfy Property 1
1: $P'_{rema} \leftarrow P_{rema}, P'_{add} \leftarrow P_{add}, \widetilde{N}'(\cdot) \leftarrow \widetilde{N}(\cdot), \widetilde{N}'(P_{add}) \leftarrow \emptyset, C'(\cdot) \leftarrow C(\cdot), C^{-1\prime}(\cdot) \leftarrow C^{-1}(\cdot)$
2: merge pixels in $\widehat{P}$ to be p_{add}
3: **for** each $p \in \widehat{P}$ **do**
4: **if** $p \in P_{rema}$ **then**
5: $C'(p_{add}) \leftarrow C'(p_{add}) \cup \{p\}$, $C^{-1\prime}(p) \leftarrow \{p_{add}\}$
6: **else if** $p \in P_{add}$ **then**
7: **for** each $p' \in C'(p)$ **do**
8: $C'(p_{add}) \leftarrow C'(p_{add}) \cup \{p'\}$, $C^{-1\prime}(p') \leftarrow \{p_{add}\}$
9: $C'(\cdot) \leftarrow C'(\cdot) - \{C'(p)\}$
10: **for** each $p \in \widehat{P}$ **do**
11: **for** each $p' \in N(p)$ such that $p' \notin \widehat{P}$ **do**
12: $\widetilde{N}'(p_{add}) \leftarrow \widetilde{N}'(p_{add}) \cup \{p'\}$, $\widetilde{N}'(p') \leftarrow \widetilde{N}'(p') - p \cup \{p_{add}\}$
13: clear $\widetilde{N}'(p)$ for each $p \in \widehat{P}$
14: **for** each $p \in \widehat{P}$ **do**
15: **if** $p \in P_{rema}$ **then**
16: $P'_{rema} \leftarrow P'_{rema} - \{p\}$
17: **else if** $p \in P_{add}$ **then**
18: $P'_{add} \leftarrow P'_{add} - \{p\}$
19: $P'_{add} \leftarrow P'_{add} \cup \{p_{add}\}$
20: **if** $\widetilde{M}' = (P'_{rema} \cup P'_{add}, \widetilde{N}'(\cdot))$ satisfy Property 1 **then**
21: $P_{rema} \leftarrow P'_{rema}, P_{add} \leftarrow P'_{add}, \widetilde{N}(\cdot) \leftarrow \widetilde{N}'(\cdot), C(\cdot) \leftarrow C'(\cdot), C^{-1}(\cdot) \leftarrow C^{-1\prime}(\cdot)$
22: **return** *True*
23: **return** *False*

$\{f, g, \dots\}$ are deleted from P'_{rema}, e is deleted from P'_{add}, and h is added into P'_{add}, so $P'_{add} = \{h\}$.

(4) *Check Property 1* (lines 20-22): In Figures 2 (b) and (c), suppose that $\widetilde{M}'$ satisfy Property 1, we have $\widetilde{M}$. In Figure 2 (f), we have the updated $C(\cdot)$ and $C^{-1}(\cdot)$.

4.4 Shortest Path Query Phase

In the text description of Figure 2, given $\widetilde{M}$, s, t, $C(\cdot)$ and $C^{-1}(\cdot)$, we first calculate $\widetilde{s}$ and $\widetilde{t}$ by pixel estimation step, then calculate $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$ by path querying step. We give one notation first.

4.4.1 ***Notation***. **Intra- and inter-paths**: Given a deleted pixel $p \in P - P_{rema}$ and a pixel $p' \in \widetilde{N}(C^{-1}(p))$ that is a neighbour pixel of the added pixel that p belongs to, let $\Pi_1(\widetilde{p}, \widetilde{p'}|\widetilde{M}) = (\widetilde{p}, \widetilde{p'})$ be the *intra-path* between them passing on $\widetilde{M}$. Given a pair of pixels $\widetilde{p}$ and $\widetilde{q}$ in $\widetilde{P}$, let $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{M})$ be the *inter-path* between them passing on $\widetilde{M}$. In Figure 2 (g), $\Pi_1(\widetilde{i}, \widetilde{p}|\widetilde{M})$ in blue dashed line is an intra-path and $\Pi_2(\widetilde{p}, \widetilde{m}|\widetilde{M})$ in blue solid line is an inter-path.

4.4.2 ***Detail and example***. We then discuss the two steps.

(1) **Pixel estimation**: Since M is removed from the hard disk, we estimate $\widetilde{s}$ using s, such that $\widetilde{s}.x = s.x$, $\widetilde{s}.y = s.y$, $\widetilde{s}.z = C^{-1}(s).z$ (if p is a deleted pixel), or $\widetilde{s} = s$ (if s is a remaining pixel). We estimate $\widetilde{t}$ similarly.

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

(i) *Both pixels deleted*: Firstly, there are two special cases that we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M}) = (\widetilde{s}, \widetilde{t})$. One is that s and t belong to the different added pixels u and v, where u and v are neighbour, the other one is that s and t belong to the same added pixel. In Figure 2 (g), $\Pi(\widetilde{f}, \widetilde{k}|\widetilde{M}) = (\widetilde{f}, \widetilde{k})$ (i.e., the first case) and $\Pi(\widetilde{a}, \widetilde{f}|\widetilde{M}) =$

$(\widetilde{a}, \widetilde{f})$ (i.e., the second case). Secondly, for common case, we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$ by concatenating the intra-path $\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{M})$, the inter-path $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{M})$, and the intra-path $\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{M})$, such that $|\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})| = \min_{\forall \widetilde{p} \in \widetilde{N}(C^{-1}(s)), \widetilde{q} \in \widetilde{N}(C^{-1}(t))} |\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{M})| + |\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{M})| + |\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{M})|$. A naive algorithm uses Dijkstra's algorithm on $\widetilde{M}$ with each pixel in $\widetilde{N}(C^{-1}(s))$ as a source to compute inter-paths. But, we propose an efficient algorithm by using Dijkstra's algorithm only once. If the number of pixels in $\widetilde{N}(C^{-1}(s))$ is less than that of in $\widetilde{N}(C^{-1}(t))$, we temporarily insert intra-paths between $\widetilde{s}$ and each pixel in $\widetilde{N}(C^{-1}(s))$ as edges in $\widetilde{G}$ (we remove them after this calculation), and then we use Dijkstra's algorithm on $\widetilde{G}$ with $\widetilde{s}$ as a source, and terminate after visits all pixels in $\widetilde{N}(C^{-1}(t))$, 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 $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$. If the number of pixels in $\widetilde{N}(C^{-1}(s))$ is larger than that of in $\widetilde{N}(C^{-1}(t))$, we swap s and t. In Figure 2 (g), $\Pi(\widetilde{i}, \widetilde{k}|\widetilde{M}) = (\widetilde{i}, p, m, \widetilde{k})$.

(ii) *One pixel deleted and one pixel remaining*: If $s \in P_{rema}$, 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 visits all $q \in \widetilde{N}(C^{-1}(t))$. We append them with the intra-path connecting to $\widetilde{t}$ and obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$. If $t \in P_{rema}$, we swap s and t. In Figure 2 (g), $\Pi(\widetilde{p}, \widetilde{k}|\widetilde{M}) = (p, m, \widetilde{k})$.

(iii) *Both pixels remaining*: Both inter-paths do not exist, we use Dijkstra's algorithm on $\widetilde{G}$ between s and t to obtain $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$. In Figure 2 (g), $\Pi(\widetilde{q}, \widetilde{r}|\widetilde{M}) = (q, m, r)$.

4.5 Efficient Property 1 Checking

Checking Property 1 involves many unnecessary distance checks. Before we introduce our efficient checking, we give one notation.

4.5.1 ***Notation.*** **Linked added pixels**: Given an added pixel $p_{add} \in P_{add}$, let $L(p_{add})$ be a set of *linked added pixels* of p_{add}, i.e., a set of added pixels in P_{add} contain p_{add} and are linked to each other. In Figure 4, suppose that $p_{add} = a$, then $L(a) = \{a, b\}$.

4.5.2 ***Detail and example.*** We then discuss our efficient checking. In Property 1, we change "all pairs of pixels s and t on M" (involving more pixels) to the following three types of pixels related to the *neighbour* pixels of the added pixel p_{add} (involving fewer pixels), and then use Inequality 1 for each type to efficiently check whether Property 1 is satisfied.

(1) **Remaining pixels to Remaining pixels (*R2R*)**: We change to "all pairs of remaining pixels s and t in P_{rema} that are neighbour pixels of each added pixel in $L(p_{add})$". Figure 4 shows these pixels as orange points.

(2) **Remaining pixels to Deleted pixels (*R2D*)**: We change to "any remaining pixel s in P_{rema} that is a neighbour pixel of each pixel in $L(p_{add})$, and any deleted pixel t in $P - P_{rema}$ that belongs to each added pixel in $L(p_{add})$". Figure 4 shows these pixels as orange points (correspond to s) and purple points (correspond to t).

(3) **Deleted pixels to Deleted pixels (*D2D*)**: We change to "all pairs of deleted pixels s and t in $P - P_{rema}$ that belong to each added pixel in $L(p_{add})$". Figure 4 shows these pixels as purple points.

4.6 Proximity Query Algorithms

Given M and $\widetilde{M}$, a query pixel $i \in P$, a set of n' interested pixels on M or $\widetilde{M}$, two parameters k and r, we can answer *kNN* and range queries using algorithm *HM-EffQ* and the shortest path query phase of algorithm *HM-MemSimQ*. A naive algorithm uses them for n' times between i and all interested pixels, and then performs a linear scan on the paths to compute *kNN* and range query results.

But, we have an efficient algorithm. The basic idea is to use both algorithms only *once* for time-saving, since they are single-source-all-destination (Dijkstra's) algorithms. (1) For algorithm *HM-EffQ*, we use Dijkstra's algorithm once with i as a source and all interested pixels 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 the shortest path query phase of algorithm *HM-MemSimQ*, we also use Dijkstra's algorithm once. Except for two special cases in Section 4.4.2 cases (2-i) that directly return the path $\Pi(\widetilde{i}, \widetilde{i'}|\widetilde{M}) = (\widetilde{i}, \widetilde{i'})$, where i' is the interested pixel, there are two cases. (i) If i is a deleted pixel, we change "s" to "i", "terminate after Dijkstra's algorithm visits all pixels in $\widetilde{N}(C^{-1}(t))$" to "terminate after Dijkstra's algorithm visits all pixels in S, where S is a set of pixels, such that for each interested pixel i', we store i' in S if i' is a remaining pixel, or we store pixels in $\widetilde{N}(C^{-1}(i'))$ into S if i' is a deleted pixel", and "append them with the intra-path connecting to $\widetilde{t}$" to "append them with the intra-path connecting to each $\widetilde{i'}$ if i' is a deleted pixel" in Section 4.4.2 case (2-i). (ii) If i is a remaining pixel, we apply the same three changes in Section 4.4.2 case (2-ii). Finally, we perform a linear scan on the paths to compute *kNN* and range query results.

4.7 Theoretical Analysis

4.7.1 ***Algorithm HM-EffQ and HM-MemSimQ.*** We analysis them in Theorems 4.1 and 4.2.

Theorem 4.1. *The shortest path query time and memory usage of algorithm HM-EffQ are $O(n \log n)$ and $O(n)$, respectively. It returns the exact shortest path passing on M.*

Proof. Since there are $O(n)$ pixels and algorithm *HM-EffQ* is Dijkstra's algorithm which returns the exact result, we know its shortest path query time, memory usage and error guarantee. □

Theorem 4.2. *The simplification time, output size and shortest path query time of algorithm HM-MemSimQ are $O(n\sqrt[3]{n} \log n)$, $O(\frac{n}{\log n})$ and $O(\frac{n}{\log n} \log \frac{n}{\log n})$, respectively. Given a height map M, it returns a simplified height map $\widetilde{M}$ such that $(1-\epsilon)|\Pi(s,t|M)| \leq |\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})| \leq (1+\epsilon)|\Pi(s,t|M)|$ for all pairs of pixels s and t on M.*

Proof Sketch. The simplification time $O(n\sqrt[3]{n} \log n) = O(n \log n \cdot \sqrt[3]{n})$ is due to the usage of Dijkstra's algorithm in $O(n \log n)$ time for $O(1)$ pixels in *R2R*, *R2D* and *D2D* checking in each pixel merging iteration, and there are total $O(\sqrt[3]{n})$ pixel merging iterations. The output size $O(n \div \log n) = O(\frac{n}{\log n})$ is due to the $O(\log n)$ deleted pixels belonging to each added pixel, and there are total n pixels on M. The shortest path query time $O(\frac{n}{\log n} \log \frac{n}{\log n})$ is due to the usage of Dijkstra's algorithm once on $\widetilde{M}$ with $O(\frac{n}{\log n})$ pixels. The error guarantee of $\widetilde{M}$ is due to the

R2R, *R2D* and *D2D* checking. The detailed proof appears in the appendix. □

4.7.2 ***Proximity query algorithms.*** We show query time and error guarantee of *kNN* and range queries using algorithm *HM-EffQ* and *HM-MemSimQ* in Theorem 4.3. Given a query pixel i, let p_f and p'_f be the furthest pixel to i computed by algorithm *HM-EffQ* and *HM-MemSimQ*, respectively. Let the error rate of *kNN* and range queries be $(\frac{|\Pi(i,p'_f|M)|}{|\Pi(i,p_f|M)|} - 1)$.

Theorem 4.3. *The query time of both kNN and range queries by using algorithm (1) HM-EffQ is $O(n \log n)$ and (2) HM-MemSimQ is $O(\frac{n}{\log n} \log \frac{n}{\log n})$, respectively. Algorithm (1) HM-EffQ returns the exact result and (2) HM-MemSimQ has an error rate $\frac{2\epsilon}{1-\epsilon}$ for both kNN and range queries, respectively.*

Proof Sketch. The query time is due to the usage of algorithm *HM-EffQ* and the shortest path query phase of algorithm *HM-MemSimQ* once. The error rate arises from their definition and the error of algorithm *HM-MemSimQ*. □

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 [28, 29, 45, 46, 52–54].

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 m indicates a height map. (i) *5 Original datasets*: GF_m, LM_m and RM_m are originally represented as height maps with 8km × 6km region, whose elevations are obtained from Google Earth [4]. BH_m and EP_m are originally represented as points clouds with 8km × 6km region, we created height maps with pixel's 2D coordinate and elevation values equal to 3D coordinate values of these points. These five datasets have a 10m × 10m resolution [29, 46, 52, 53]. (ii) *5 Small-version datasets*: They are generated using the same region as the original datasets, with a reduced 70m × 70m resolution, following the dataset generation steps in studies [46, 52, 53]. (iii) *24 Multi-resolution datasets*: They are generated similarly with varying numbers of pixels. (2 & 3) *Point cloud* and *TIN* datasets: We utilize 2D coordinate and elevation values of pixels in the height map datasets to generate 34 point cloud datasets, and then triangulate [22, 34, 55] them to generate 34 *TIN* datasets. We use c and t as subscripts, respectively.

5.1.2 ***Algorithms.*** Since there is no existing study focusing on simplifying point clouds, we adapt our algorithm *HM-MemSimQ* to be algorithm *Point Cloud Memory saving simulated Simplification and simplified point cloud shortest path Query* (*PC-MesSimQ*) [53] by simulating our algorithm on the point cloud's conceptual graph to obtain a simplified point cloud and answer shortest paths on the simplified point cloud. Then, we discuss three types of algorithms.

(1) To solve our problem on *Height Maps*, we adapted existing algorithms on point clouds or *TIN*s, by (i) using 2D coordinate and elevation values of pixels in the height map to obtain a point cloud with points of the corresponding 3D coordinate values, and (ii) triangulating [22, 34, 55] these points to obtain a *TIN*. Their algorithm names are appended by "*-Adapt(HM)*". We have four height map simplification algorithms: (i) *TIN-SurSimQ-Adapt(HM)* [26, 29], (ii) *TIN-NetSimQ-Adapt(HM)* [29], (iii) *PC-MesSimQ-Adapt(HM)* [53] and (iv) our algorithm *HM-MemSimQ*. We have five shortest path query algorithms: (v) *TIN-UnfQ-Adapt(HM)* [15, 47, 54], (vi) *TIN-SteQ-Adapt(HM)* [28, 52], (vii) *TIN-DijQ-Adapt(HM)* [29], (viii) *PC-ConQ-Adapt(HM)* [53] and (ix) our algorithm *HM-EffQ*. We compare them in Table 2.

(2) To solve the existing problem on *Point Clouds* [53], we adapted algorithms on height maps or *TIN*s, by (i) using 3D coordinate values of points in the point cloud to obtain a height map with pixels of the corresponding 2D coordinate and elevation values, and (ii) triangulating [22, 34, 55] the points in the point cloud to obtain a *TIN*. Their algorithm names are appended by "*-Adapt(PC)*". Similarly, we have nine algorithms: (i) *TIN-SurSimQ-Adapt(PC)* [26, 29], (ii) *TIN-NetSimQ-Adapt(PC)* [29], (iii) *PC-MesSimQ* [53], (iv) *HM-MemSimQ-Adapt(PC)*, (v) *TIN-UnfQ-Adapt(PC)* [15, 47, 54], (vi) *TIN-SteQ-Adapt(PC)* [28, 52], (vii) *TIN-DijQ-Adapt(PC)* [29], (viii) *PC-ConQ* [53] and (ix) *HM-EffQ-Adapt(PC)*.

(3) To solve the existing problem on *TIN*s [26, 29], we adapted algorithms on height maps or point clouds, by (i) using 3D coordinate values of vertices in the *TIN* to obtain a height map with pixels of the corresponding 2D coordinate and elevation values, and (ii) using the vertices in the *TIN* to obtain a point cloud. Their algorithm names are appended by "*-Adapt(TIN)*". Similarly, we have nine algorithms: (i) *TIN-SurSimQ* [26, 29], (ii) *TIN-NetSimQ* [29], (iii) *PC-MesSimQ-Adapt(TIN)* [53], (iv) *HM-MemSimQ-Adapt(TIN)*, (v) *TIN-UnfQ* [15, 47, 54], (vi) *TIN-SteQ* [28, 52], (vii) *TIN-DijQ* [29], (viii) *PC-ConQ-Adapt(TIN)* [53] and (ix) *HM-EffQ-Adapt(TIN)*.

5.1.3 ***Proximity Queries.*** We conducted three queries. (1) Shortest path query: we generate 100 query instances by randomly selecting two pixels on the height map (or two points on the point cloud, or two vertices on the *TIN*) as source and destination. We report the average, maximum and minimum results. The experimental result figures' vertical bars represent the maximum and minimum values, and points indicate the average results. (2 & 3) All *kNN* and range queries: we randomly select 1000 pixels on the height map (or points on the point cloud, or vertices on the *TIN*) as objects, and use all of them as query objects to perform the proximity query algorithm in Section 4.6.

5.1.4 ***Factors and Metrics.*** We studied four factors: (1) ϵ (the error parameter), (2) n (the dataset size, meaning the number of pixels of a height map, points of a point cloud, or vertices of a *TIN*), (3) k (the parameter in the *kNN* query), and (4) r (the parameter in the range query). When not varying $k \in [200, 1000]$ and $r \in [2\text{km}, 10\text{km}]$, we fix k at 500 and r at 5km according to studies [19, 43]. We employ nine metrics: (1) *simplification time*, (2) *memory consumption* (the storage complexity during algorithm execution), (3) *output size*, (4) *query time* (the shortest path query time), (5 & 6) *kNN or range query time* (all *kNN* or range query time), (7) *distance error* (the distance error of the algorithm compared with the exact algorithm), (8 & 9) *kNN or range query error* (the error rate of the *kNN* or range query described in Section 4.7.2).

Table 1: Height map datasets

Name	$\|n\|$
Original dataset	
GunnisonForest (GF_m) [6]	0.5M
LaramieMount (LM_m) [8]	0.5M
RobinsonMount (RM_m) [13]	0.5M
BearHead (BH_m) [3, 45, 46]	0.5M
EaglePeak (EP_m) [3, 45, 46]	0.5M
Small-version dataset	
GF_m*-small*	10k
LM_m*-small*	10k
RM_m*-small*	10k
BH_m*-small*	10k
EP_m*-small*	10k
Multi-resolution dataset	
GF_m multi-resolution	1M, 1.5M, 2M, 2.5M
LM_m multi-resolution	1M, 1.5M, 2M, 2.5M
RM_m multi-resolution	1M, 1.5M, 2M, 2.5M
BH_m multi-resolution	1M, 1.5M, 2M, 2.5M
EP_m multi-resolution	1M, 1.5M, 2M, 2.5M
EP_m*-small* multi-resolution	20k, 30k, 40k, 50k

Table 2: Comparison of algorithms

Algorithm	Simplification time		Output size		Shortest path query time		Error
Simplification algorithm							
TIN-SurSimQ-Adapt(HM) [26, 29]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large	$O(n^2)$	Large	Small
TIN-NetSimQ-Adapt(HM) [29]	$O(n^2\log n)$	Medium	$O(n)$	Large	$O(n\log n)$	Medium	Medium
PC-MesSimQ-Adapt(HM) [53]	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
***HM-MemSimQ* (ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	**Small**
Shortest path query algorithm							
TIN-UnfQ-Adapt(HM) [15, 47, 54]	-	N/A	-	N/A	$O(n^2)$	Large	Small
TIN-SteQ-Adapt(HM) [28, 52]	-	N/A	-	N/A	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-DijQ-Adapt(HM) [29]	-	N/A	-	N/A	$O(n\log n)$	Medium	Medium
PC-ConQ-Adapt(HM) [53]	-	N/A	-	N/A	$O(n\log n)$	Medium	No error
***HM-EffQ* (ours)**	-	N/A	-	N/A	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**

5.2 Experimental Results

Due to the page limit, we provided some selected metrics performance figures. We provided full sets of metrics performance figures in the appendix.

5.2.1 ***Height maps.*** We studied proximity queries on height maps. *HM-EffQ* returns the height map's exact shortest path and its computed path's distance is used for distance error calculation. We compared all algorithms in Table 2 on small-version datasets, and compared all algorithms except *TIN-SurSimQ-Adapt(HM)* and *TIN-NetSimQ-Adapt(HM)* on original datasets (since they have excessive simplification time on original datasets).

(1) **Baseline comparisons**:

(i) **Effect of ϵ**: In Figure 5, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_m*-small* dataset while fixing n at 10k for baseline comparisons. The simplification time of *HM-MemSimQ* is much smaller than that of *TIN-SurSimQ-Adapt(HM)* and *TIN-NetSimQ-Adapt(HM)* due to the efficient height map shortest path query and efficient Property 1 checking. The output size of *HM-MemSimQ* is also much smaller than these two algorithms' due to the novel memory saving technique. The shortest path query time and the *kNN* query time (i.e., $O(n'\cdot\frac{n}{\log n}\log\frac{n}{\log n}) = O(\frac{nn'}{\log n}\log\frac{n}{\log n})$ in Theorem 4.3, since we perform the *kNN* query for n' query objects) of *HM-MemSimQ* are also small since its simplified height map has a small output size. The distance error of *HM-MemSimQ* is close to 0. So, the experimental *kNN* and range query error rates are 0 (since $|\Pi(i,p'_f|M)| = |\Pi(i,p_f|M)|$ in Section 4.7.2, although the theoretical error rates are $\frac{2\epsilon}{1-\epsilon}$ in Theorem 4.3), and their results are omitted.

(ii) **Effect of n (scalability test)**: In Figure 6, we tested 5 values of n in {0.5M, 1M, 1.5M, 2M, 2.5M} on LM_m dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-MemSimQ* outperforms all baselines. The simplification time of *HM-MemSimQ* is 19,000s ≈ 5.2 hours for a height map with 2.5M pixels, which shows its scalability. *PC-MesSimQ-Adapt(HM)* performs similarly as *HM-MemSimQ*, since they have the same simplification process, and the height map and point cloud have the same conceptual graph.

(2) **Ablation study for proximity query algorithms (effect of k and r)**: We considered two variations of *HM-MemSimQ*, i.e., (i) *HM-MemSimQ Large Query Time* (*HM-MemSimQ-LQT1*): *HM-MemSimQ* using the naive algorithm in the shortest path query phase in Section 4.4, but the efficient proximity query algorithm in Section 4.6 (i.e., we use each pixel in $\widetilde{N}(C^{-1}(i))$ as a source in each Dijkstra's algorithm and terminate after each algorithm visits all pixels in S, where i and S have the same meaning in Section 4.6), and (ii) *HM-MemSimQ-LQT2*: *HM-MemSimQ* using the efficient algorithm in the shortest path query phase, but the naive proximity query algorithm. In Figures 7 (a) and (b), 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_m dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study. *HM-MemSimQ* outperforms both *HM-MemSimQ-LQT1* and *HM-MemSimQ-LQT2*, since we use the efficient algorithm for querying. k does not affect the *kNN* query time of *HM-MemSimQ*, since we append the paths computed by Dijkstra's algorithm and the intra-paths as the path results, and we do not know the distance correlations among these paths before we perform a linear scan on them. But, a smaller r reduces their range query time, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r.

(3) **Ablation study for simplification algorithms**: We considered two more variations of *HM-MemSimQ*, i.e., (i) *HM-MemSimQ Large output Size* (*HM-MemSimQ-LS*): *HM-MemSimQ* using the naive merging technique that only merges four pixels in Section 4.2, and (ii) *HM-MemSimQ Large Simplification Time* (*HM-MemSimQ-LST*): *HM-MemSimQ* using the naive checking technique that checks whether Inequality 1 is satisfied for all pixels in Section 4.2. In Figure 8, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_m*-small* dataset while fixing n at 0.5M for ablation study. *HM-MemSimQ* still outperforms these two variations, showing the effectiveness of our merging and checking techniques.

5.2.2 ***Point clouds.*** We studied proximity queries on point clouds. *PC-ConQ* returns the point cloud's exact shortest path and its computed path's distance is used for distance error calculation.

Effect of ϵ: In Figure 9, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on EP_c*-small* dataset while fixing n at 10k for baseline comparison. *HM-MemSimQ-Adapt(PC)* still outperforms other baselines. We studied the effect of n in the appendix.

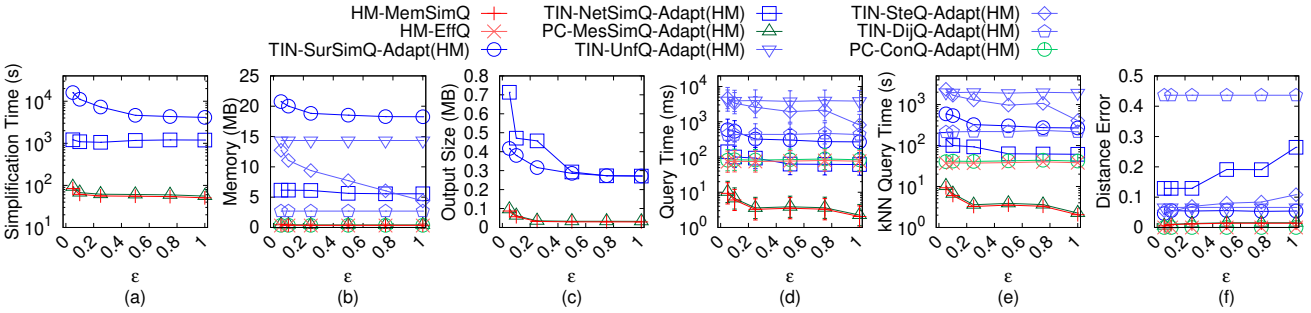

Figure 5: Baseline comparisons (effect of ϵ on GF_m-*small* height map dataset)

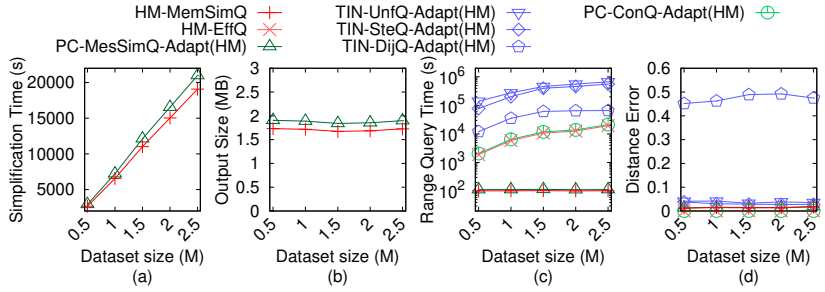

Figure 6: Baseline comparisons (effect of n on LM_m height map dataset)

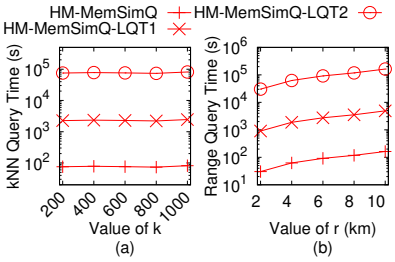

Figure 7: Ablation study for proximity query algorithms (effect of k and r on RM_m height map dataset)

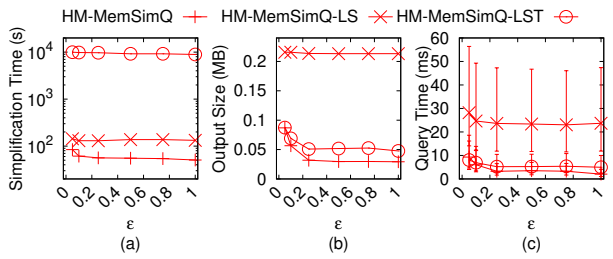

Figure 8: Ablation study for simplification algorithms on BH_m-*small* height map dataset

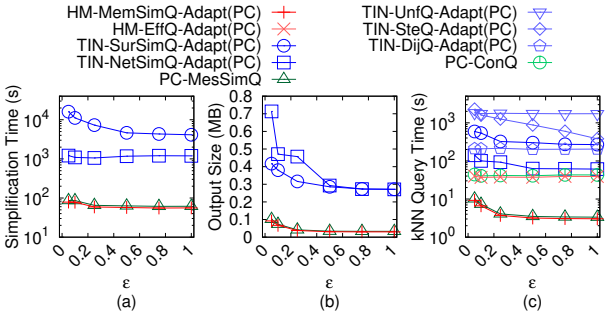

Figure 9: Baseline comparisons (effect of ϵ on EP_c-*small* point cloud dataset)

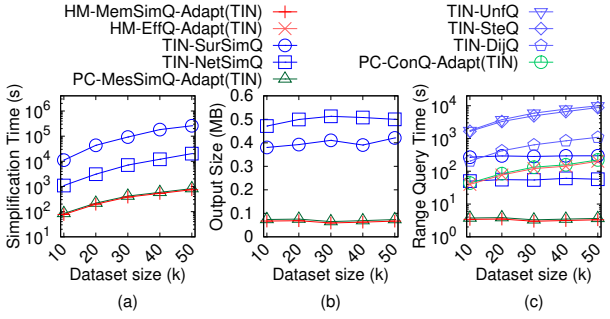

Figure 10: Baseline comparisons (effect of n on EP_t-*small TIN* dataset)

5.2.3 ***TINs.*** We studied proximity queries on *TIN*s. *TIN-UnfQ* returns the *TIN*'s exact shortest surface path and its computed path's distance is used for distance error calculation.

Effect of n: In Figure 10, 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. Despite giving a *TIN* as input, *HM-MemSimQ-Adapt(TIN)* outperforms *TIN-SurSimQ* and *TIN-NetSimQ* concerning the simplification time, output size and shortest path query time. The range query time of *HM-EffQ-Adapt(TIN)* is 100 times smaller than that of *TIN-UnfQ* (although *HM-EffQ-Adapt(TIN)* requires constructing a height map using the given *TIN*, and *TIN-UnfQ* does not involve any additional steps). The distance error of *HM-EffQ-Adapt(TIN)* is 0.06, but the distance error of *TIN-DijQ* is 0.45. We studied the effect of ϵ in the appendix.

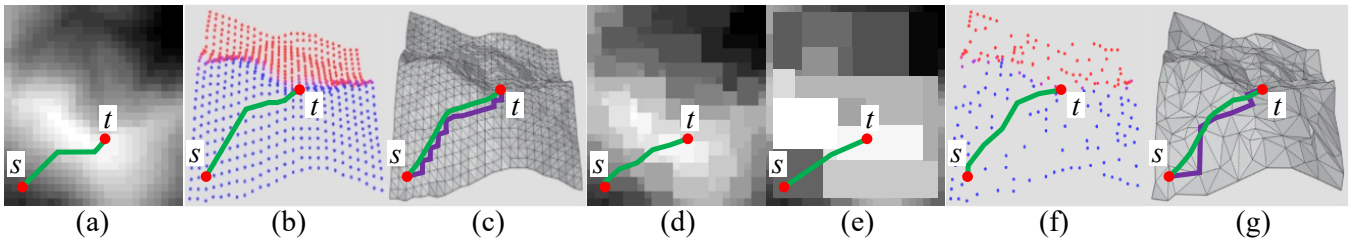

Figure 11: The shortest path passing on (a) a height map and (b) a point cloud, (c) the shortest surface (resp. network) path passing on a *TIN* with green (resp. purple) line, the shortest path passing on (d) a simplified height map ($\epsilon = 0.1$), (e) a simplified height map ($\epsilon = 0.5$) and (f) a simplified point cloud ($\epsilon = 0.1$), and (g) the shortest surface (resp. network) path passing on a simplified *TIN* ($\epsilon = 0.1$) with green (resp. purple) line

5.2.4 ***Case study on the usefulness of the height map simplification.*** In Figure 1, we performed an evacuation case study at Mount Rainier [39] in response to routinely heavy snowfall [40]. The winter storms in January 2024 left 89 dead across the USA [33] and there exists the risk of asphyxiation if snow-covered [30]. During snowfall, skiers on the mountain are promptly evacuated to nearby hotels for safety. An individual will be buried in snow in 2.4 hours[1], and the evacuation (i.e., walking from the viewpoints to hotels) can be finished in 2.2 hours[2]. 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 pixels, 10k possible skier positions and 50 hotels, the simplification time for (1) *HM-MemSimQ* is 250s ≈ 4.6 min and (2) the best-known adapted *TIN* simplification algorithm *TIN-SurSimQ-Adapt(HM)* is 103,000s ≈ 1.2 days. Under the same setting, the query time of computing 10 nearest hotels for each skier position is (1) 50s for *HM-MemSimQ*, (2) 67,000s ≈ 18.6 hours for *TIN-SurSimQ-Adapt(HM)*, (3) 7,630s ≈ 2.1 hours for the best-known adapted point cloud shortest path query algorithm *PC-ConQ-Adapt(HM)*, and (4) 380,000s ≈ 4.3 days for the best-known adapted approximate *TIN* shortest surface path query algorithm *TIN-SteQ-Adapt(HM)*. Thus, *HM-MemSimQ* is the most efficient for evacuation since 4.6 min + 50s ≤ 12 min.

5.2.5 ***Case study on paths visualization.*** In Figure 11, we performed a case study for path visualization to verify distance relationships in Section 3.1.

(1) Given a height map, the path in Figure 11 (a) computed by our algorithm *HM-EffQ* and the path in Figure 11 (b) computed by the best-known adapted point cloud shortest path query algorithm *PC-ConQ-Adapt(HM)* are identical (since $|\Pi(s,t|M)| = |\Pi(s,t|C)|$). The path in Figure 11 (a) is similar to the green path in Figure 11 (c) computed by the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-UnfQ-Adapt(HM)* (since $|\Pi(s,t|M)| \leq \alpha \cdot |\Pi(s,t|T)|$), but computing the former path is much quicker due to its smaller query region compared to the latter. The path in Figure 11 (a) is similar to the paths in Figures 11 (d) and (e) computed by our algorithm *HM-MemSimQ*, but computing the latter two paths are quicker due to the simplified height maps. The path in Figure 11 (d) is the same as the path in Figure 11 (f) computed by the best-known adapted point cloud simplification algorithm *PC-MesSimQ-Adapt(HM)*, and similar to the green path in Figure 11 (g) computed by the best-known adapted *TIN* simplification algorithm *TIN-SurSimQ-Adapt(HM)*.

(2) Given a point cloud, the path in Figure 11 (a) computed by our adapted algorithm *HM-EffQ-Adapt(PC)* is the same as the path in Figure 11 (b) computed by the best-known point cloud shortest path query algorithm *PC-ConQ*.

(3) Given a *TIN*, the path in Figure 11 (a) computed by our adapted algorithm *HM-EffQ-Adapt(TIN)* is similar to the green path in Figure 11 (c) computed by the best-known exact *TIN* shortest surface path query algorithm *TIN-UnfQ*. The distance error of the path in Figure 11 (a) is smaller than that of the purple path in Figure 11 (d) computed by the best-known approximate *TIN* shortest network path query algorithm *TIN-UnfQ* (since $|\Pi(s,t|M)| \leq |\Pi_N(s,t|T)|$).

5.2.6 ***Summary.*** Concerning the simplification time and output size and shortest path query time, *HM-MemSimQ* is up to 412 times, 7 times and 1,340 times better than the best-known adapted *TIN* simplification algorithm *TIN-SurSimQ-Adapt(HM)*, respectively. Performing *kNN* and range queries on our simplified height map are both up to 153 times quicker than the best-known adapted point cloud shortest path query algorithm*PC-ConQ-Adapt(HM)* on a point cloud, and 1,340 times quicker than *TIN-SurSimQ-Adapt(HM)* on its simplified *TIN*. On a height map with 50k pixels and 10k objects, the simplification time, output size and *kNN* query time are 250s ≈ 4.6 min, 0.07MB and 50s for *HM-MemSimQ*, but are 103,000s ≈ 1.2 days, 0.5MB and 67,000s ≈ 18.6 hours for *TIN-SurSimQ-Adapt(HM)*, respectively.

6 CONCLUSION

We propose an efficient ϵ-approximate height map simplification algorithm *HM-MemSimQ*, that outperforms the best-known algorithm concerning the simplification time, output size and shortest path query time. We also propose ϵ-approximate algorithms for answering *kNN* and range queries on our simplified height map. For future work, we can propose new pruning techniques to further reduce the simplification time and output size of *HM-MemSimQ*.

[1] 2.4 hours = $\frac{10\text{centimeters} \times 24\text{hours}}{1\text{meter}}$, since the maximum snowfall rate (defined as the maximum accumulation of snow depth over a specified time [16, 42]) at Mount Rainier is 1 meter per 24 hours [41], and when the snow depth exceeds 10 centimeters, it is difficult to walk and easy to bury in the snow [23].

[2] 2.2 hours = $\frac{11.2\text{km}}{5.1\text{km/h}}$, since the average distance between the viewpoints and hotels at Mount Rainier is 11.2km [5], and human's average walking speed is 5.1 km/h [12].

REFERENCES

[1] 2025. *125 Years of Topographic Mapping.* https://www.esri.com/news/arcnews/fall09articles/125-years.html
[2] 2025. *Blender.* https://www.blender.org
[3] 2025. *Data Geocomm.* http://data.geocomm.com/
[4] 2025. *Google Earth.* https://earth.google.com/web
[5] 2025. *Google Map.* https://www.google.com/maps
[6] 2025. *Gunnison National Forest.* https://gunnisoncrestedbutte.com/visit/places-to-go/parks-and-outdoors/gunnison-national-forest/
[7] 2025. *The History of Point Cloud Development.* https://www.linkedin.com/pulse/history-point-cloud-development-bimprove/
[8] 2025. *Laramie Mountain.* https://www.britannica.com/place/Laramie-Mountains
[9] 2025. *MeshLab.* https://www.meshlab.net
[10] 2025. *Metaverse.* https://about.facebook.com/meta
[11] 2025. *Open Digital Elevation Model (OpenDEM).* https://www.opendem.info/
[12] 2025. *Preferred walking speed.* https://en.wikipedia.org/wiki/Preferred_walking_speed
[13] 2025. *Robinson Mountain.* https://www.mountaineers.org/activities/routes-places/robinson-mountain
[14] Leizhen Cai. 1994. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics* 48, 2 (1994), 187–194.
[15] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *Proceedings of the Symposium on Computational Geometry (SOCG)*. New York, NY, USA, 360–369.
[16] 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
[17] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.
[18] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-NN query processing. In *International Conference on Data Engineering (ICDE)*. IEEE, 78–78.
[19] 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.
[20] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[21] Hristo N Djidjev and Christian Sommer. 2011. Approximate distance queries for weighted polyhedral surfaces. In *Proceedings of the European Symposium on Algorithms*. 579–590.
[22] Michael Garland and Paul S Heckbert. 1995. Fast polygonal approximation of terrains and height fields. (1995).
[23] 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/
[24] Yuhang He, Long Chen, Jianda Chen, and Ming Li. 2015. A novel way to organize 3D LiDAR point cloud as 2D depth map height map and surface normal map. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 1383–1388.
[25] Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*. 99–108.
[26] 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.
[27] Shruti Kanga and Suraj Kumar Singh. 2017. Forest Fire Simulation Modeling using Remote Sensing & GIS. *International Journal of Advanced Research in Computer Science* 8, 5 (2017).
[28] 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.
[29] 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.
[30] Russell LaDuca. 2020. *What would happen to me if I was buried under snow?* https://qr.ae/prt6zQ
[31] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*, Vol. 30. 527–562.
[32] 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.
[33] Katie Hawkinson Louise Boyle, Kelly Rissman. 2024. *Winter storms leave 89 dead across US as chill settles over Great Lakes and Northeast.* https://www.independent.co.uk/climate-change/news/winter-storm-warning-weather-forecast-snow-b2480627.html
[34] De Berg Mark, Cheong Otfried, Van Kreveld Marc, and Overmars Mark. 2008. *Computational geometry algorithms and applications.*
[35] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.
[36] 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.
[37] Janet E Nichol, Ahmed Shaker, and Man-Sing Wong. 2006. Application of high-resolution stereo satellite images to detailed landslide hazard assessment. *Geomorphology* 76, 1-2 (2006), 68–75.
[38] Youssef Saab and Michael VanPutte. 1999. Shortest path planning on topographical maps. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 29, 1 (1999), 139–150.
[39] National Park Service. 2025. *Mount Rainier.* https://www.nps.gov/mora/index.htm
[40] National Park Service. 2025. *Mount Rainier Annual Snowfall Totals.* https://www.nps.gov/mora/planyourvisit/annual-snowfall-totals.htm
[41] National Park Service. 2025. *Mount Rainier Frequently Asked Questions.* https://www.nps.gov/mora/faqs.htm
[42] National Weather Service. 2025. *Measuring Snow.* https://www.weather.gov/dvn/snowmeasure
[43] 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.
[44] Jiaojiao Tian, Allan A Nielsen, and Peter Reinartz. 2015. Building damage assessment after the earthquake in Haiti using two post-event satellite stereo imagery and DSMs. *International Journal of Image and Data Fusion* 6, 2 (2015), 155–169.
[45] 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.
[46] 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).
[47] 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.
[48] Shi-Qing Xin and Guo-Jin Wang. 2009. Improving Chen and Han's algorithm on the discrete geodesic problem. *ACM Transactions on Graphics* 28, 4 (2009), 1–8.
[49] 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.
[50] 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.
[51] 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.
[52] 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)*.
[53] 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.
[54] 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.
[55] Xianwei Zheng, Hanjiang Xiong, Jianya Gong, and Linwei Yue. 2016. A virtual globe-based multi-resolution tin surface modeling and visualizetion method. In *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, Vol. 41.

A SUMMARY OF ALL NOTATION

Table 3 shows a summary of all notation.

B COMPARISON OF ALL ALGORITHMS

Table 4 shows a comparison of all algorithms.

C EMPIRICAL STUDIES

C.1 Experimental Results for Height Maps

C.1.1 ***Baseline comparisons.*** We studied the effect of ϵ and n for proximity queries on height maps for baseline comparisons.

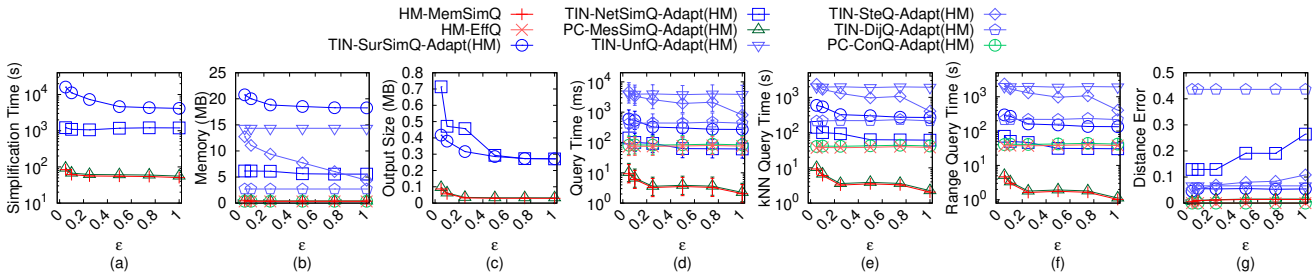

Figure 12: Baseline comparisons (effect of ϵ on GF_m-*small* height map dataset)

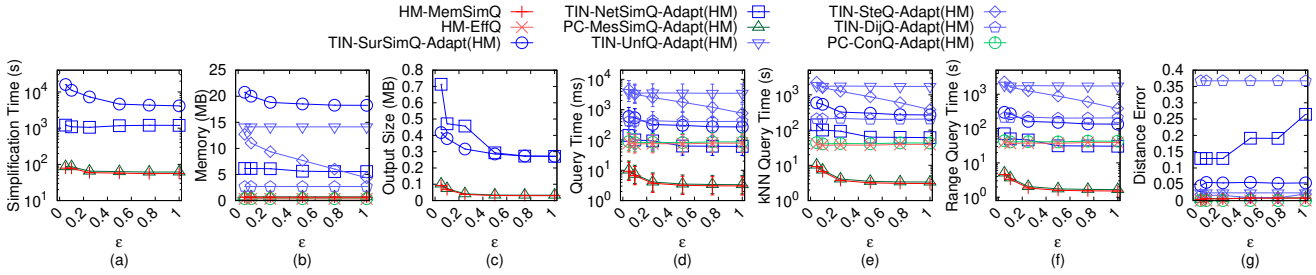

Figure 13: Baseline comparisons (effect of ϵ on LM_m-*small* height map dataset)

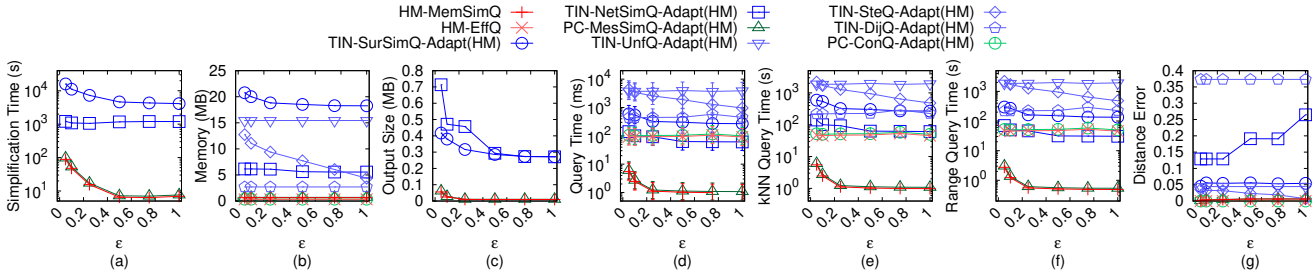

Figure 14: Baseline comparisons (effect of ϵ on RM_m-*small* height map dataset)

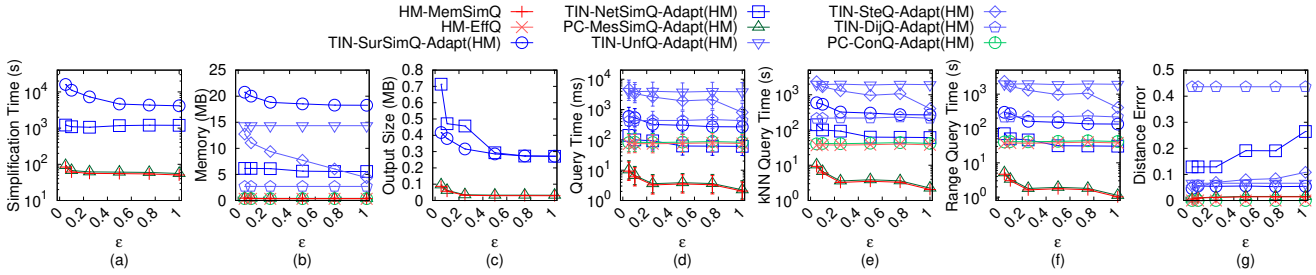

Figure 15: Baseline comparisons (effect of ϵ on BH_m-*small* height map dataset)

We compared algorithm *TIN-SurSimQ-Adapt(HM)*, *TIN-NetSimQ-Adapt(HM)*, *PC-MesSimQ-Adapt(HM)*, *HM-MemSimQ*, *TIN-UnfQ-Adapt(HM)*, *TIN-SteQ-Adapt(HM)*, *TIN-DijQ-Adapt(HM)*, *PC-ConQ-Adapt(HM)* and *HM-EffQ* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 12, Figure 13, Figure 14, Figure 15 and Figure 16, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_m-*small*, LM_m-*small*, RM_m-*small*, BH_m-*small* and EP_m-*small* dataset while fixing n at 10k for baseline comparisons. In Figure 18, Figure 20, Figure 22, Figure 24 and Figure 26, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_m, LM_m, RM_m, BH_m and EP_m dataset while fixing n at 0.5M for baseline comparisons. The simplification time of *HM-MemSimQ* is much smaller than that of *TIN-SurSimQ-Adapt(HM)* and *TIN-NetSimQ-Adapt(HM)* due to the efficient height map shortest path query and efficient Property 1 checking. The output size of *HM-MemSimQ* is also much smaller than that of *TIN-SurSimQ-Adapt(HM)* and *TIN-NetSimQ-Adapt(HM)* due to the novel memory saving technique. The shortest path query

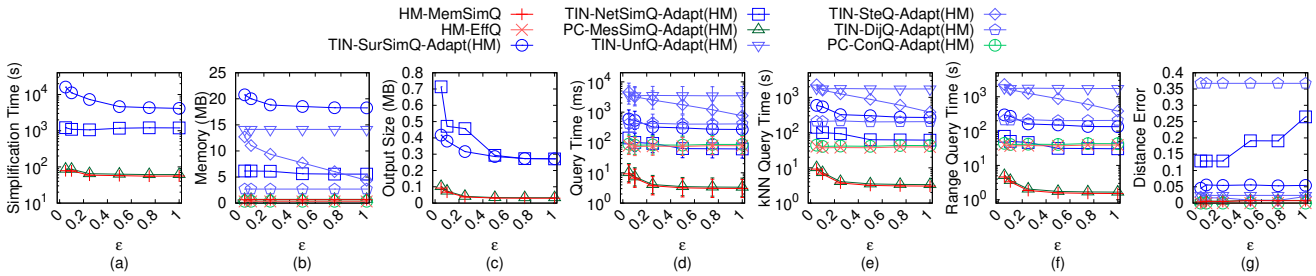

Figure 16: Baseline comparisons (effect of ϵ on EP_m-*small* height map dataset)

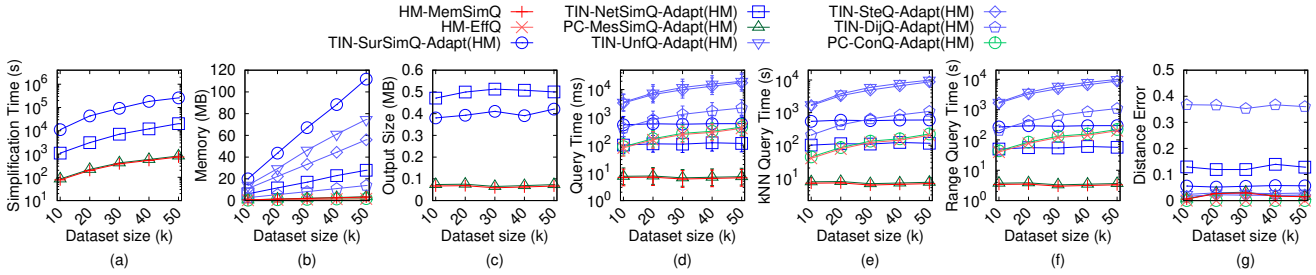

Figure 17: Baseline comparisons (effect of n on EP_m-*small* height map dataset)

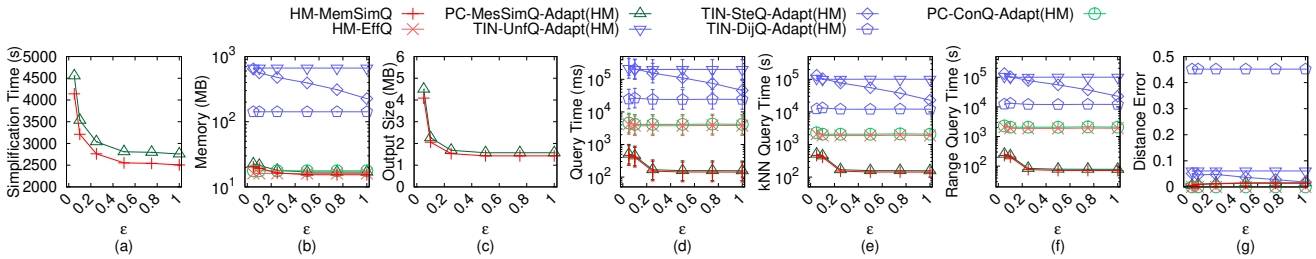

Figure 18: Baseline comparisons (effect of ϵ on GF_m height map dataset)

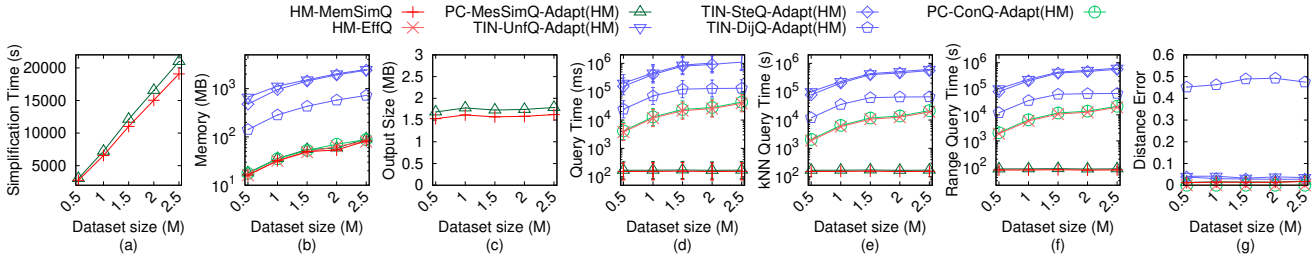

Figure 19: Baseline comparisons (effect of n on GF_m height map dataset)

time and the *kNN* query time of *HM-MemSimQ* are also small since its simplified height map has a small output size. The distance error of *HM-MemSimQ* is small (close to 0).

Effect of n (scalability test): In Figure 17, we tested 5 values of n in {50, 100, 150, 200, 250} on EP_m-*small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 19, Figure 21, Figure 23, Figure 25 and Figure 27 we tested 5 values of n in {500, 1000, 1500, 2000, 2500} on GF_m, LM_m, RM_m, BH_m and EP_m dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-MemSimQ* outperforms all the remaining algorithms. The simplification time of *HM-MemSimQ* is salable for a height map with 2.5M pixels. *PC-MesSimQ-Adapt(HM)* performs similarly as *HM-MemSimQ*, since they have the same simplification process, and the height map and point cloud have the same conceptual graph. *HM-EffQ* outperforms *TIN-UnfQ-Adapt(HM)* and *TIN-SteQ-Adapt(HM)* concerning the range query time since it computes the shortest path passing on a height map.

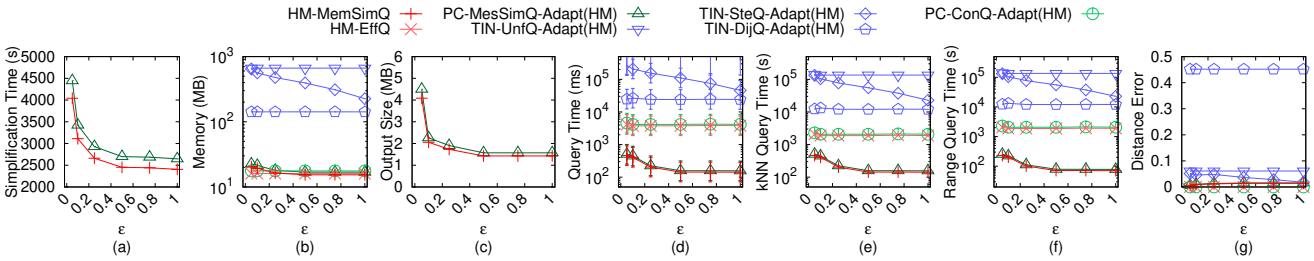

Figure 20: Baseline comparisons (effect of ϵ on LM_m height map dataset)

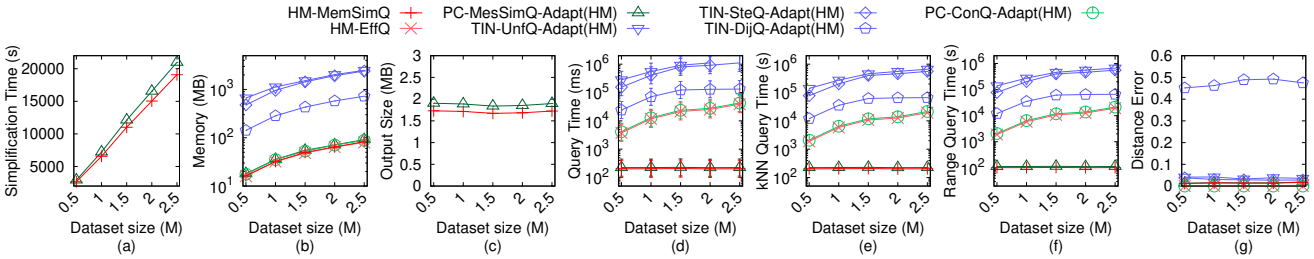

Figure 21: Baseline comparisons (effect of n on LM_m height map dataset)

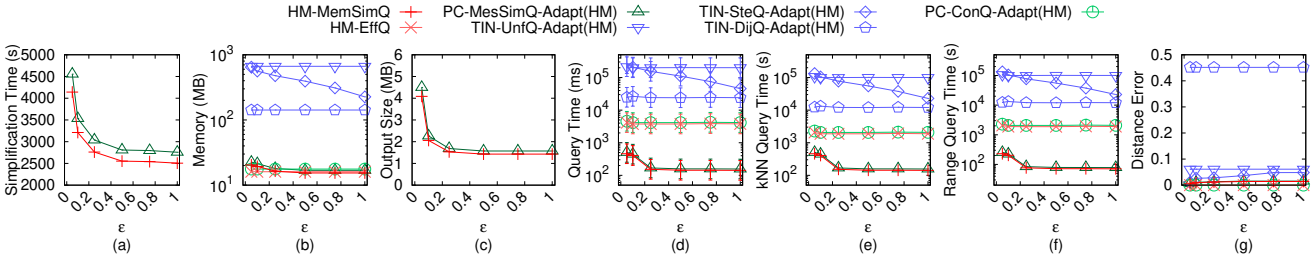

Figure 22: Baseline comparisons (effect of ϵ on RM_m height map dataset)

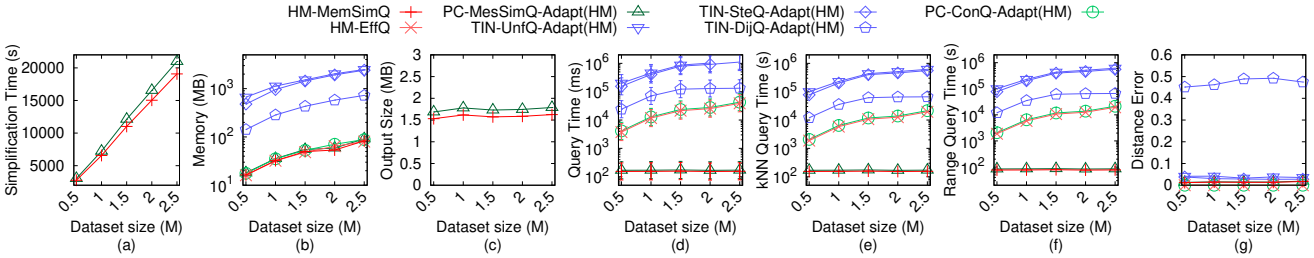

Figure 23: Baseline comparisons (effect of n on RM_m height map dataset)

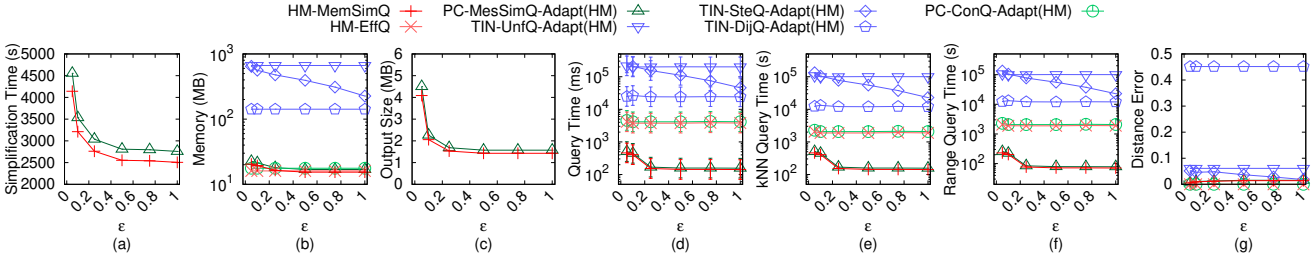

Figure 24: Baseline comparisons (effect of ϵ on BH_m height map dataset)

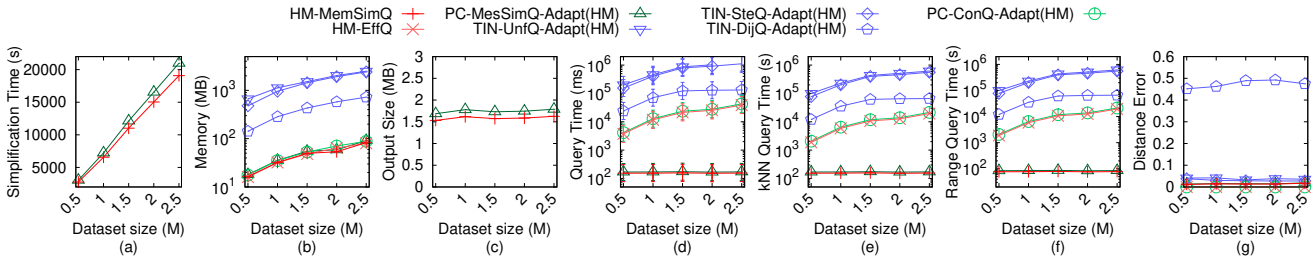

Figure 25: Baseline comparisons (effect of n on BH_m height map dataset)

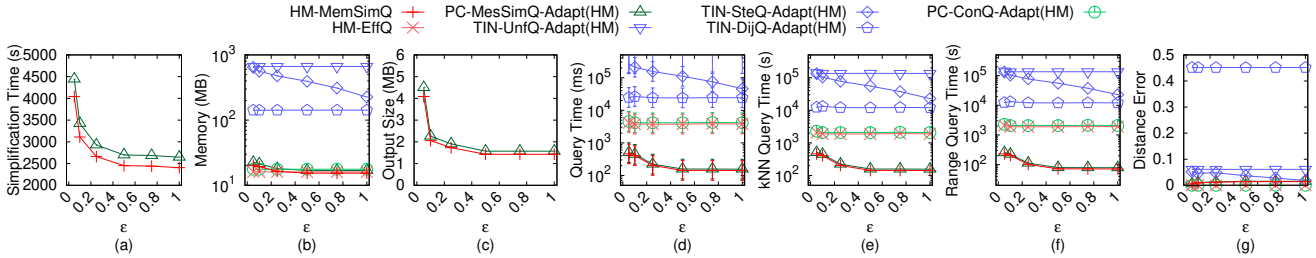

Figure 26: Baseline comparisons (effect of ϵ on EP_m height map dataset)

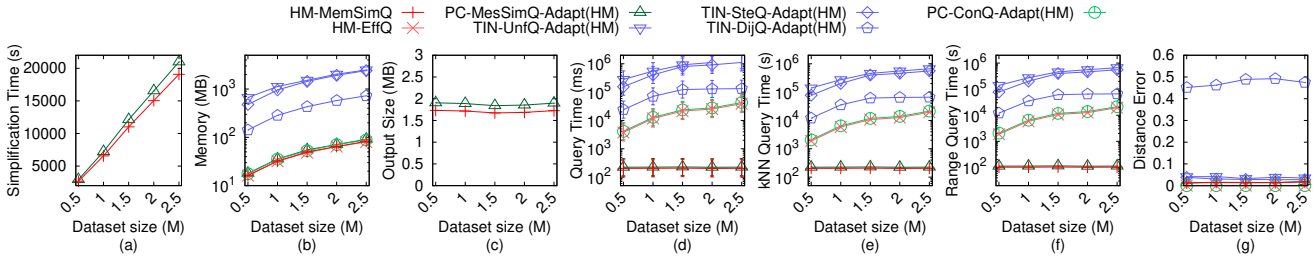

Figure 27: Baseline comparisons (effect of n on EP_m height map dataset)

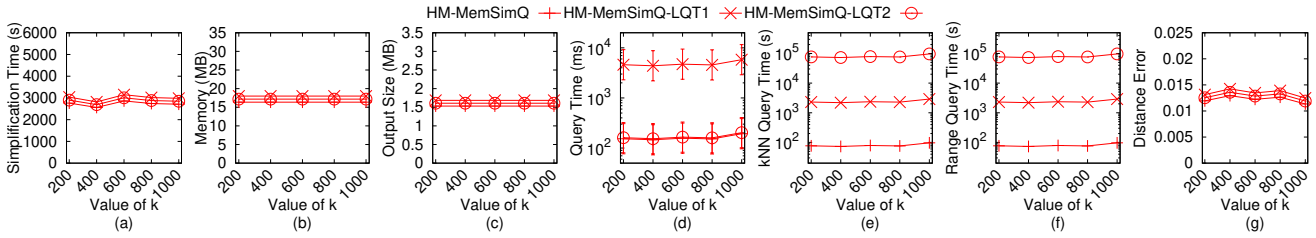

Figure 28: Ablation study for proximity query algorithms (effect of k on GF_m height map dataset)

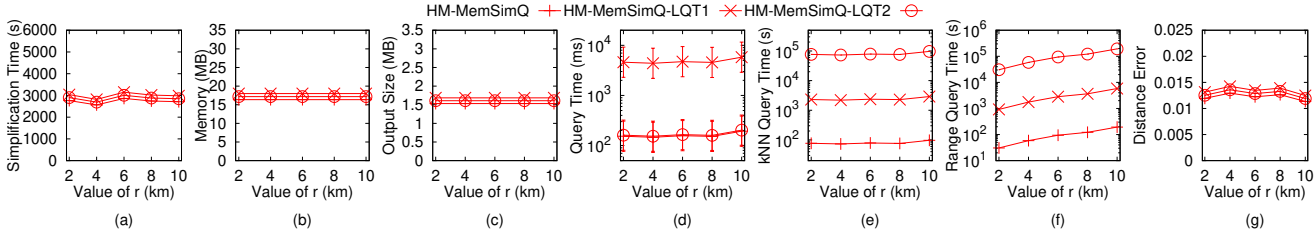

Figure 29: Ablation study for proximity query algorithms (effect of r on GF_m height map dataset)

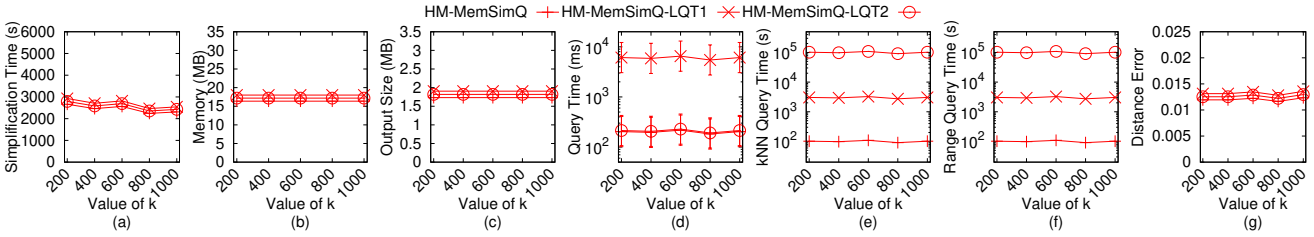

Figure 30: Ablation study for proximity query algorithms (effect of k on LM_m height map dataset)

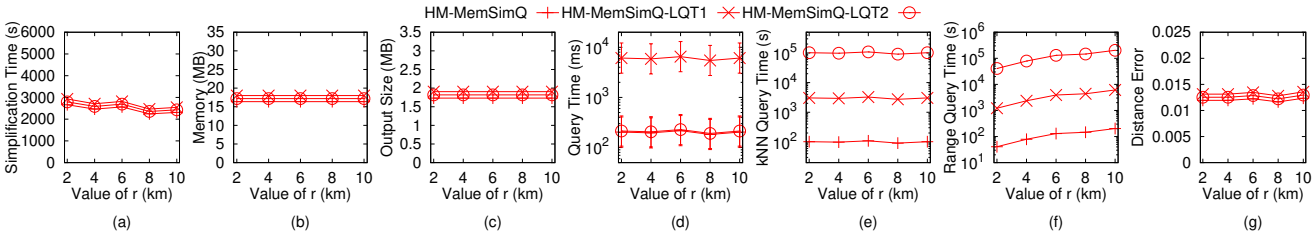

Figure 31: Ablation study for proximity query algorithms (effect of r on LM_m height map dataset)

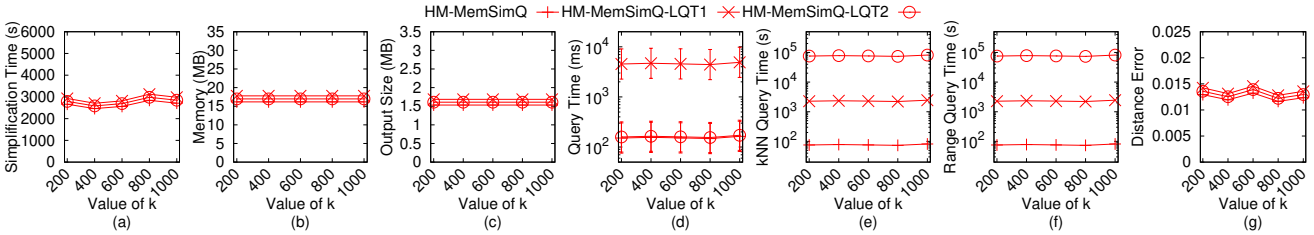

Figure 32: Ablation study for proximity query algorithms (effect of k on RM_m height map dataset)

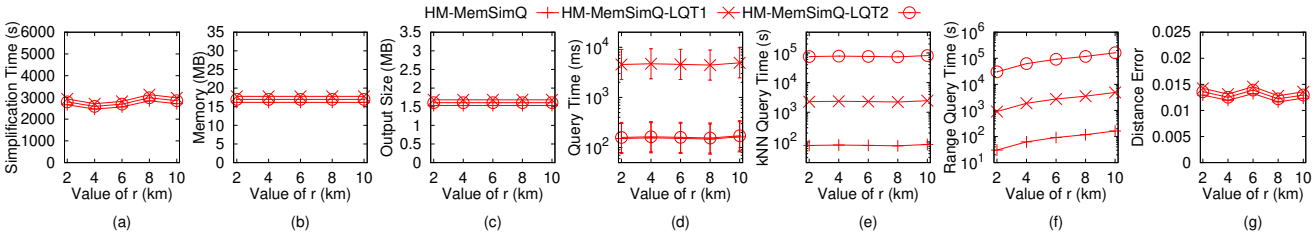

Figure 33: Ablation study for proximity query algorithms (effect of r on RM_m height map dataset)

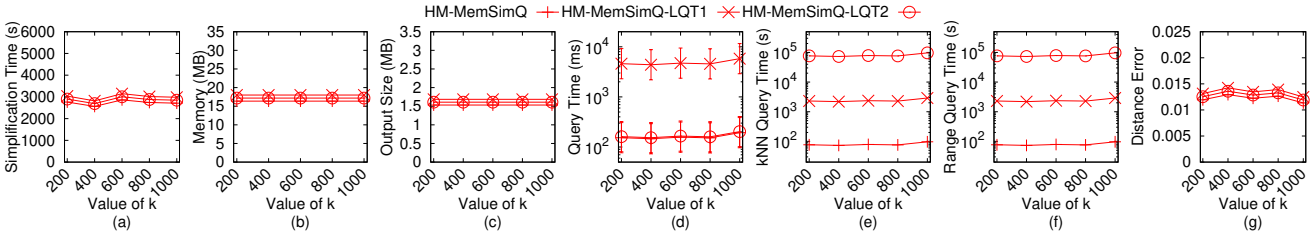

Figure 34: Ablation study for proximity query algorithms (effect of k on BH_m height map dataset)

C.1.2 ***Ablation study for proximity query algorithms.*** **Effect of k and r:** In Figure 28, Figure 30, Figure 32, Figure 34 and Figure 36 we tested 5 values of k in {200, 400, 600, 800, 1000} on GF_m, LM_m, RM_m, BH_m and EP_m dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study for proximity query algorithms (among

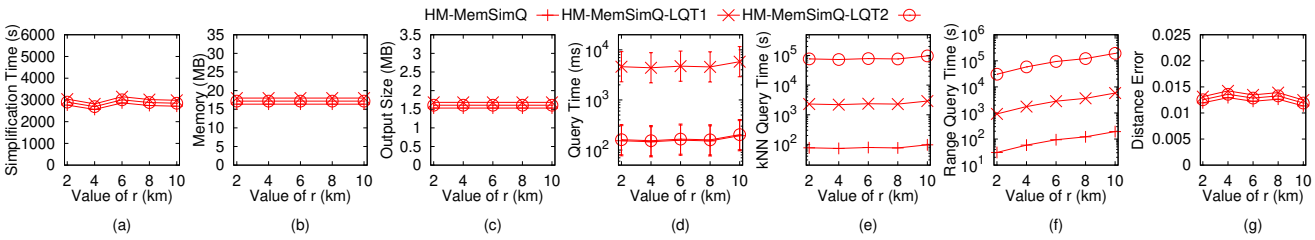

Figure 35: Ablation study for proximity query algorithms (effect of r on BH_m height map dataset)

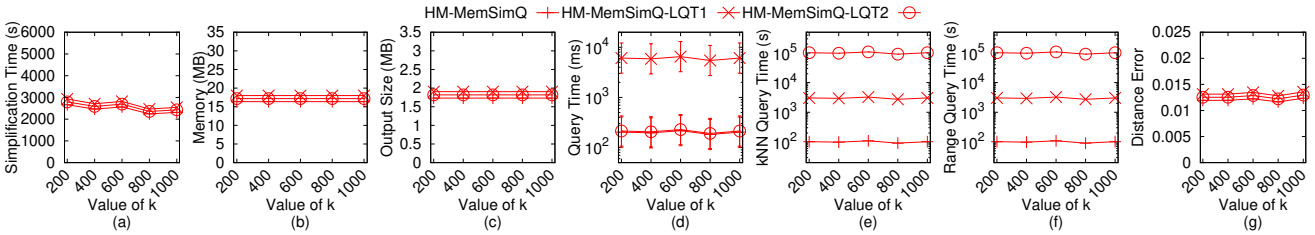

Figure 36: Ablation study for proximity query algorithms (effect of k on EP_m height map dataset)

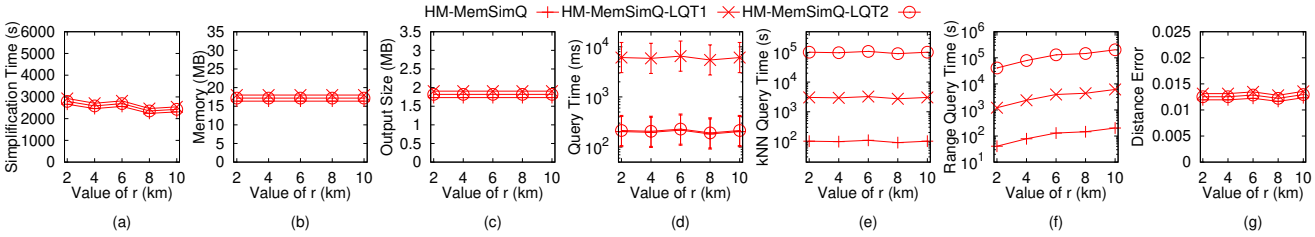

Figure 37: Ablation study for proximity query algorithms (effect of r on EP_m height map dataset)

algorithm *HM-MemSimQ*, *HM-MemSimQ-LQT1* and *HM-MemSimQ-LQT2*). In Figure 29, Figure 31, Figure 33, Figure 35 and Figure 37 we tested 5 values of r in {2km, 4km, 6km, 8km, 10km} on GF_m, LM_m, RM_m, BH_m and EP_m dataset while fixing ϵ at 0.25 and n at 0.5M for ablation study for proximity query algorithms (among algorithm *HM-MemSimQ*, *HM-MemSimQ-LQT1* and *HM-MemSimQ-LQT2*). *HM-MemSimQ* outperforms both *HM-MemSimQ-LQT1* and *HM-MemSimQ-LQT2*, since we use the efficient algorithm for querying. Varying k does not affect the *kNN* query time of *HM-MemSimQ*, since we append the paths computed by Dijkstra's algorithm and the intra-paths as the path results, and we do not know the distance correlations among these paths before we perform a linear scan on them. But, a smaller r value can reduce the range query time of *HM-MemSimQ*, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r. *HM-MemSimQ-LQT1* are *HM-MemSimQ-LQT2* not affected by k and r since they need to compute all the paths, and then perform a linear scan on these paths.

C.1.3 ***Ablation study for simplification algorithms.*** In Figure 38, Figure 39, Figure 40, Figure 41 and Figure 42 we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_m*-small*, LM_m*-small*, RM_m*-small*, BH_m*-small* and EP_m*-small* dataset while fixing n at 0.5M for ablation study for simplification algorithms (among algorithm *HM-MemSimQ*, *HM-MemSimQ-LS* and *HM-MemSimQ-LST*). *HM-MemSimQ* still outperforms these two variations, showing the effectiveness of our merging and checking techniques.

C.2 Experimental Results for Point Clouds

We studied the effect of ϵ and n for proximity queries on point clouds for baseline comparisons to demonstrate the usefulness of our proximity queries on height maps. We compared algorithm *TIN-SurSimQ-Adapt(PC)*, *TIN-NetSimQ-Adapt(PC)*, *PC-MesSimQ*, *HM-MemSimQ-Adapt(PC)*, *TIN-UnfQ-Adapt(PC)*, *TIN-SteQ-Adapt(PC)*, *TIN-DijQ-Adapt(PC)*, *PC-ConQ* and *HM-EffQ-Adapt(PC)* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 43, Figure 44, Figure 45, Figure 46 and Figure 47 we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_c*-small*, LM_c*-small*, RM_c*-small*, BH_c*-small* and EP_c*-small* dataset while fixing n at 10k for baseline comparisons. In Figure 49, Figure 51, Figure 53, Figure 55 and Figure 57 we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_c, LM_c, RM_c, BH_c and EP_c dataset while fixing n at 0.5M for baseline comparisons. *HM-MemSimQ-Adapt(PC)* still outperforms other baselines.

Effect of n (scalability test): In Figure 48, we tested 5 values of n in {50, 100, 150, 200, 250} on EP_c*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 50, Figure 52, Figure 54, Figure 56 and Figure 58 we tested 5 values of n in {500, 1000, 1500,

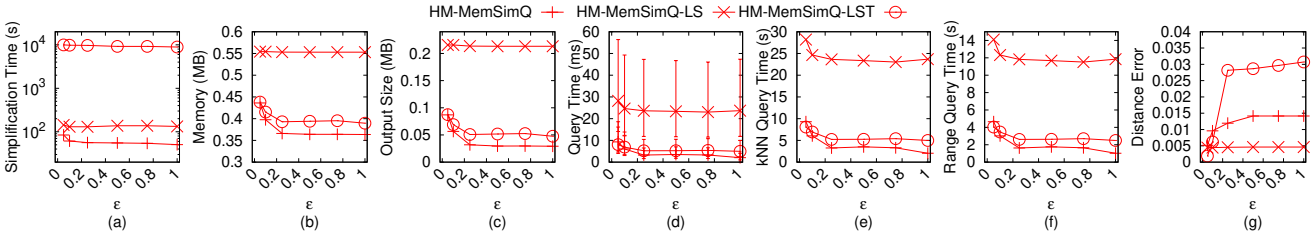

Figure 38: Ablation study for simplification algorithms on *GF_m-small* height map dataset

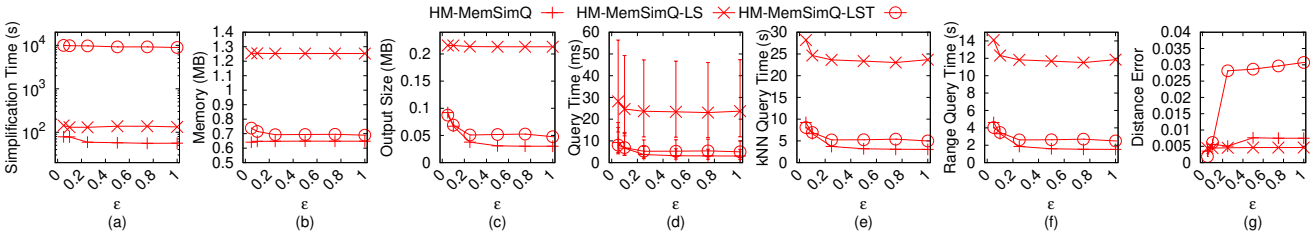

Figure 39: Ablation study for simplification algorithms on *LM_m-small* height map dataset

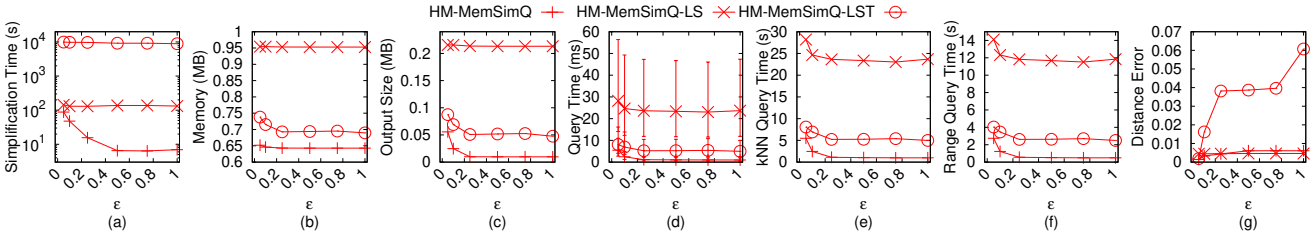

Figure 40: Ablation study for simplification algorithms on *RM_m-small* height map dataset

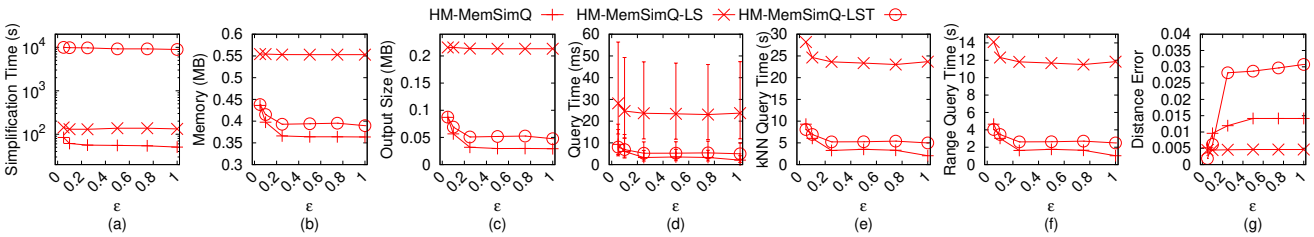

Figure 41: Ablation study for simplification algorithms on *BH_m-small* height map dataset

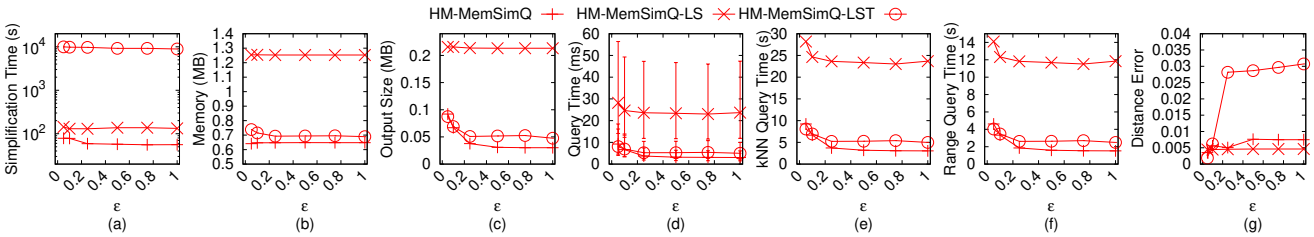

Figure 42: Ablation study for simplification algorithms on *EP_m-small* height map dataset

2000, 2500} on GF_c, LM_c, RM_c, BH_c and EP_c dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-MemSimQ-Adapt(PC)* outperforms all the remaining algorithms. The simplification time of *HM-MemSimQ-Adapt(PC)* is salable for a height map with 2.5M pixels. *PC-MesSimQ* performs similarly as *HM-MemSimQ-Adapt(PC)*,

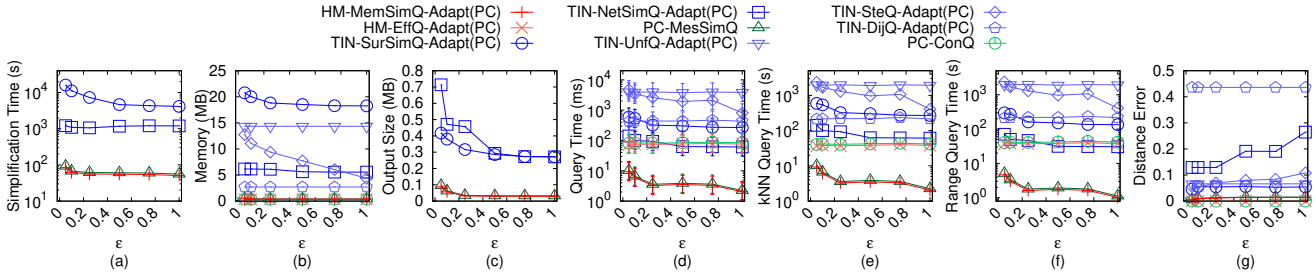

Figure 43: Baseline comparisons (effect of ϵ on GF_c-*small* point cloud dataset)

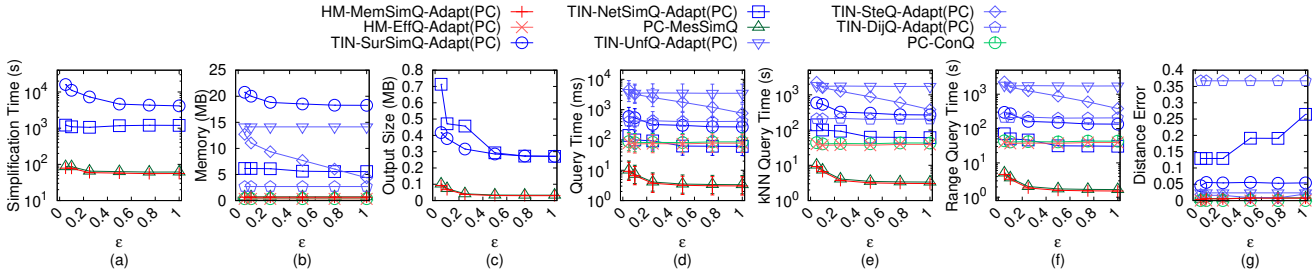

Figure 44: Baseline comparisons (effect of ϵ on LM_c-*small* point cloud dataset)

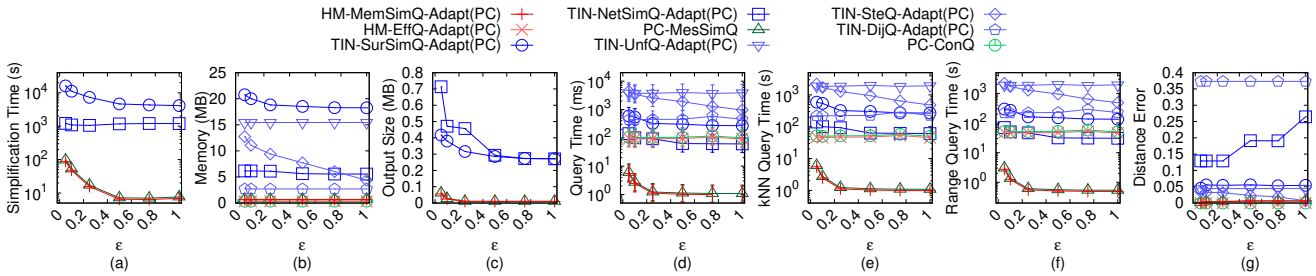

Figure 45: Baseline comparisons (effect of ϵ on RM_c-*small* point cloud dataset)

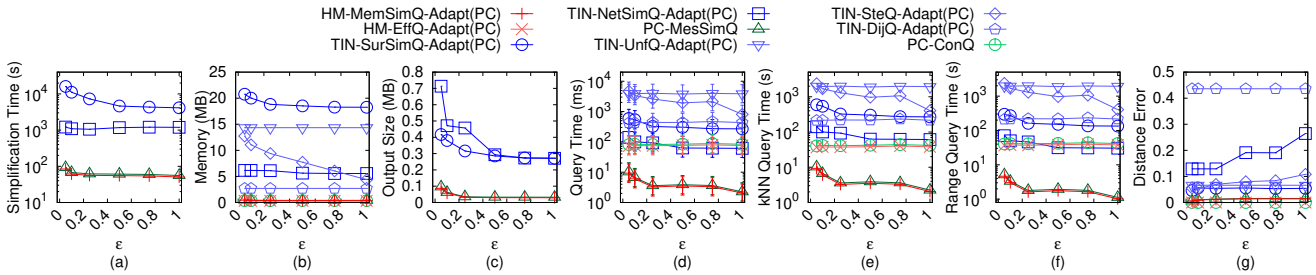

Figure 46: Baseline comparisons (effect of ϵ on BH_c-*small* point cloud dataset)

since they have the same simplification process, and the height map and point cloud have the same conceptual graph.

C.3 Experimental Results for *TIN*s

We studied the effect of ϵ and n for proximity queries on *TIN*s for baseline comparisons to demonstrate the usefulness of our proximity queries on height maps. We compared algorithm *TIN-SurSimQ*, *TIN-NetSimQ*, *PC-MesSimQ-Adapt(TIN)*, *HM-MemSimQ-Adapt(TIN)*, *TIN-UnfQ*, *TIN-SteQ*, *TIN-DijQ*, *PC-ConQ-Adapt(TIN)* and *HM-EffQ-Adapt(TIN)* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 59, Figure 60, Figure 61, Figure 62 and Figure 63 we tested 6 values of ϵ in $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on GF_t-*small*, LM_t-*small*, RM_t-*small*, BH_t-*small* and EP_t-*small* dataset while fixing n at 10k for baseline comparisons. In Figure 65, Figure 67, Figure 69, Figure 71 and Figure 73 we tested 6 values of ϵ in $\{0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ on GF_t, LM_t, RM_t, BH_t and EP_t dataset

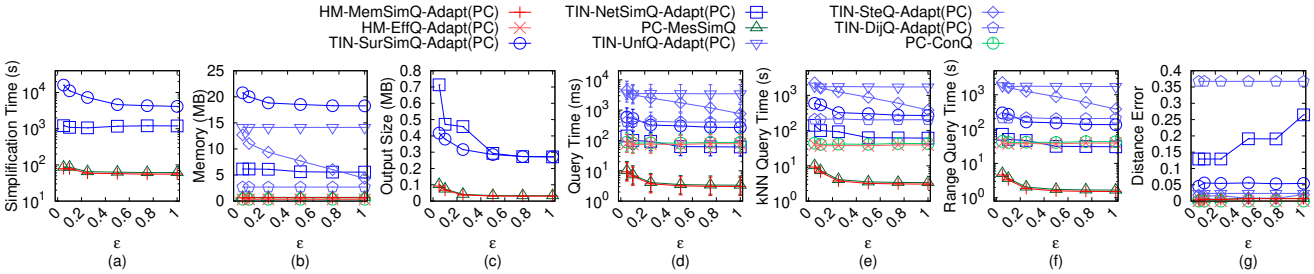

Figure 47: Baseline comparisons (effect of ϵ on EP_c-*small* point cloud dataset)

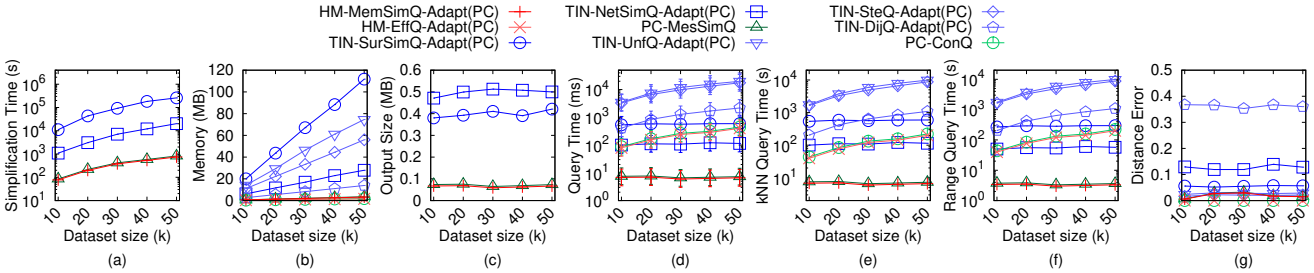

Figure 48: Baseline comparisons (effect of n on EP_c-*small* point cloud dataset)

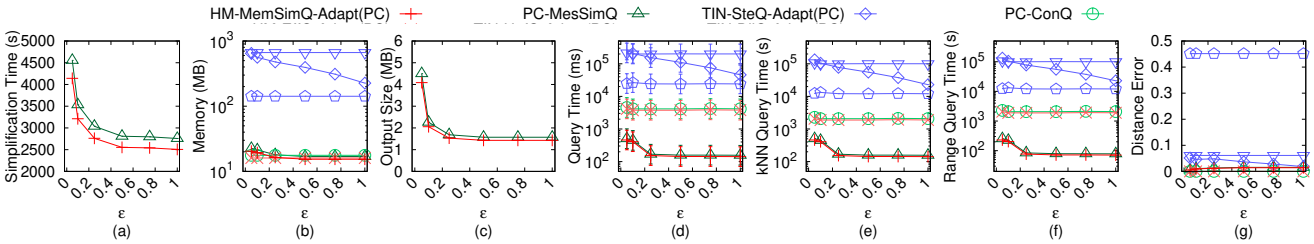

Figure 49: Baseline comparisons (effect of ϵ on GF_c point cloud dataset)

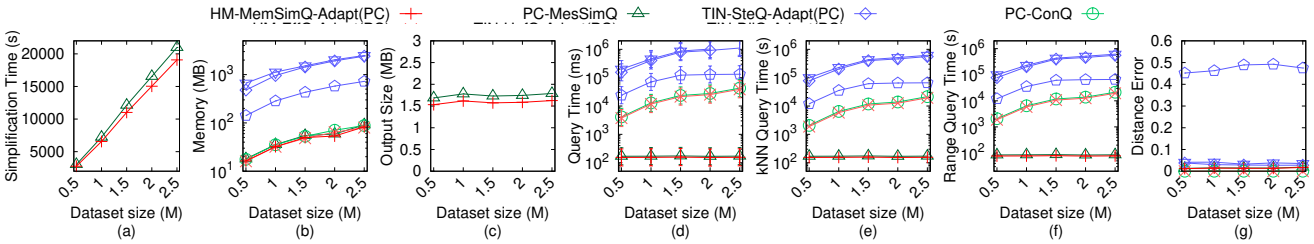

Figure 50: Baseline comparisons (effect of n on GF_c point cloud dataset)

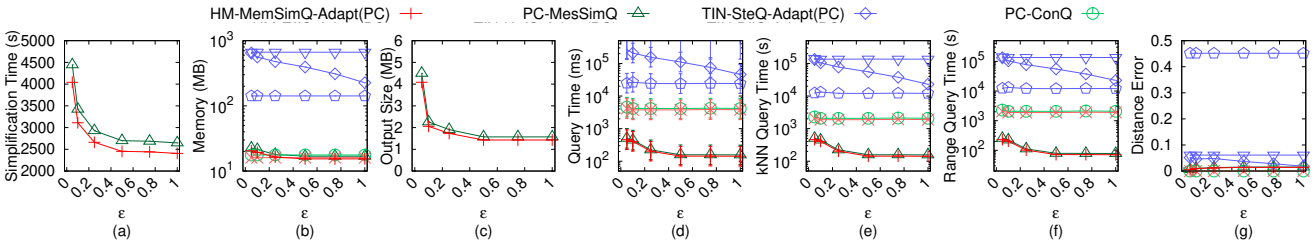

Figure 51: Baseline comparisons (effect of ϵ on LM_c point cloud dataset)

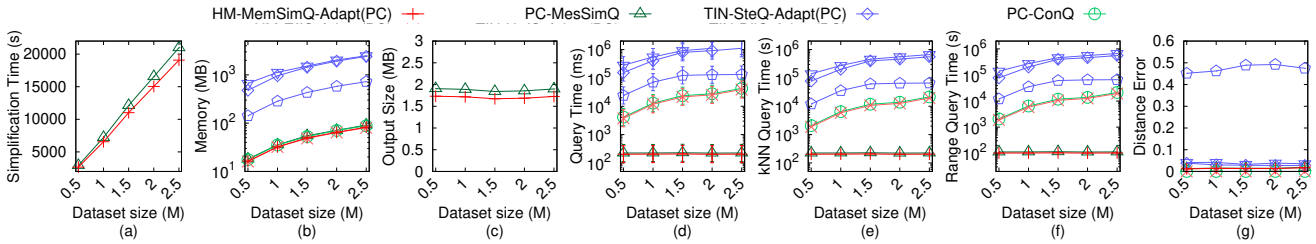

Figure 52: Baseline comparisons (effect of n on LM_c point cloud dataset)

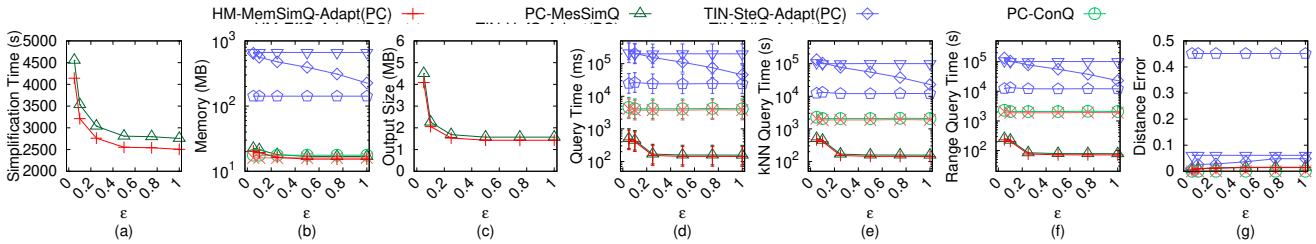

Figure 53: Baseline comparisons (effect of ϵ on RM_c point cloud dataset)

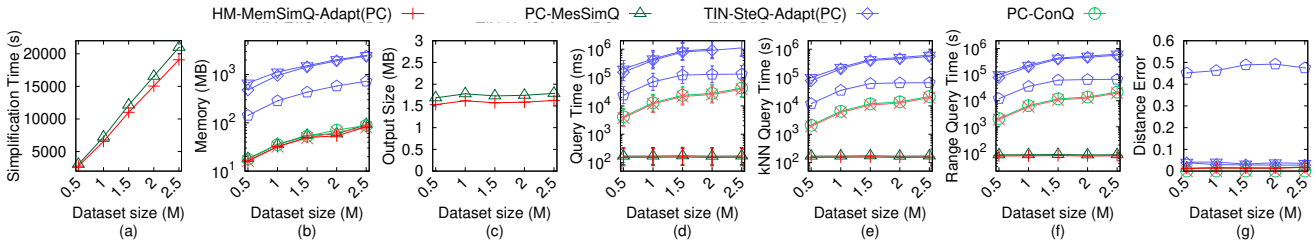

Figure 54: Baseline comparisons (effect of n on RM_c point cloud dataset)

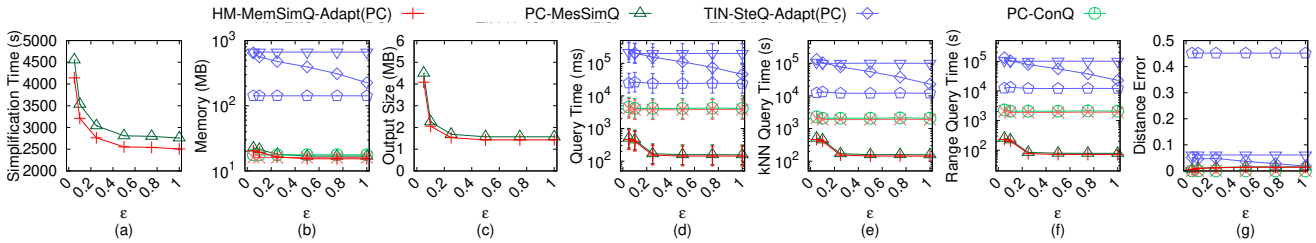

Figure 55: Baseline comparisons (effect of ϵ on BH_c point cloud dataset)

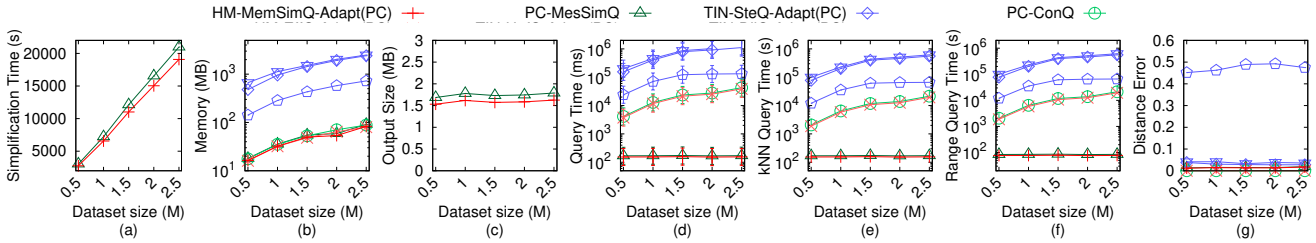

Figure 56: Baseline comparisons (effect of n on BH_c point cloud dataset)

while fixing n at 0.5M for baseline comparisons. Despite giving a *TIN* as input, *HM-MemSimQ-Adapt(TIN)* outperforms *TIN-SurSimQ* and *TIN-NetSimQ* concerning the simplification time, output size

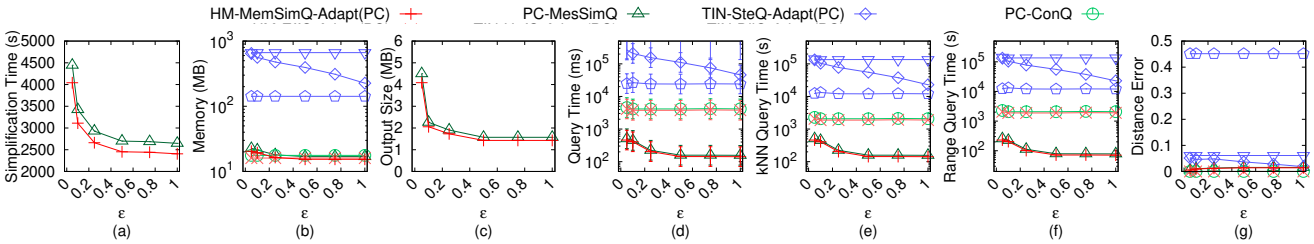

Figure 57: Baseline comparisons (effect of ϵ on EP_c point cloud dataset)

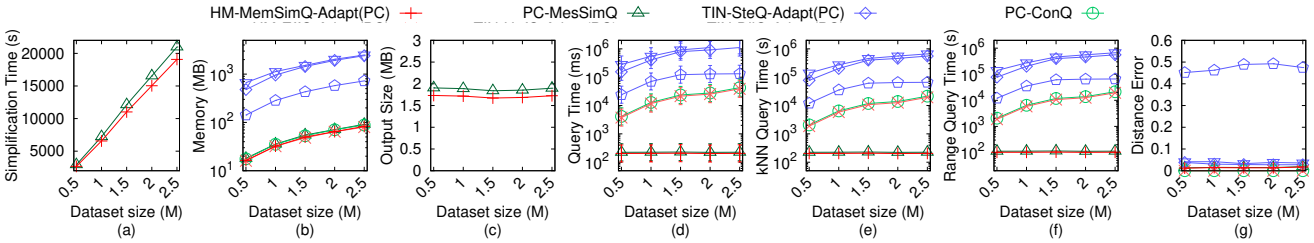

Figure 58: Baseline comparisons (effect of n on EP_c point cloud dataset)

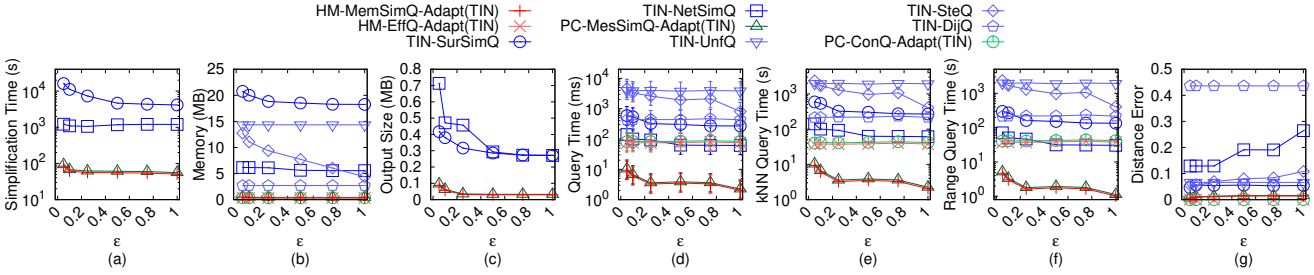

Figure 59: Baseline comparisons (effect of ϵ on GF_t *-small TIN* dataset)

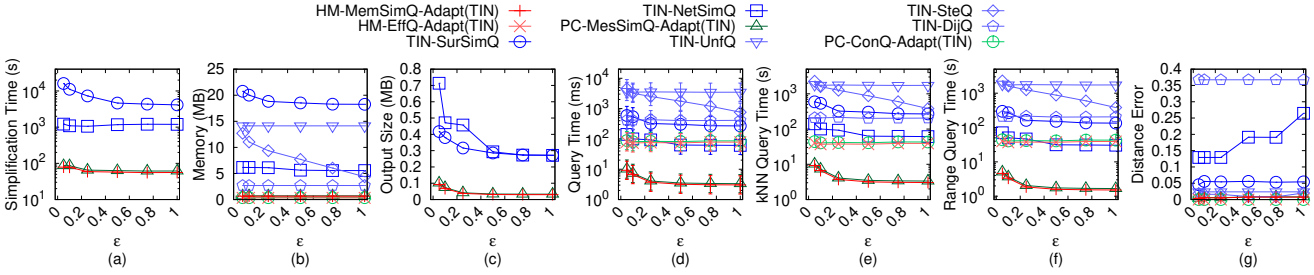

Figure 60: Baseline comparisons (effect of ϵ on LM_t *-small TIN* dataset)

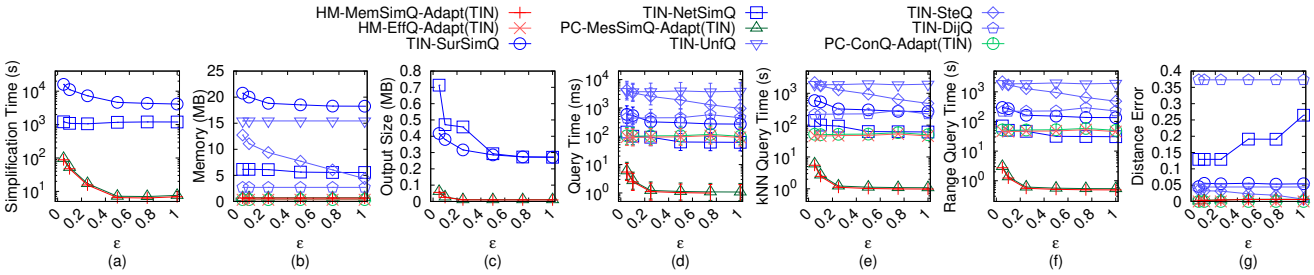

Figure 61: Baseline comparisons (effect of ϵ on RM_t *-small TIN* dataset)

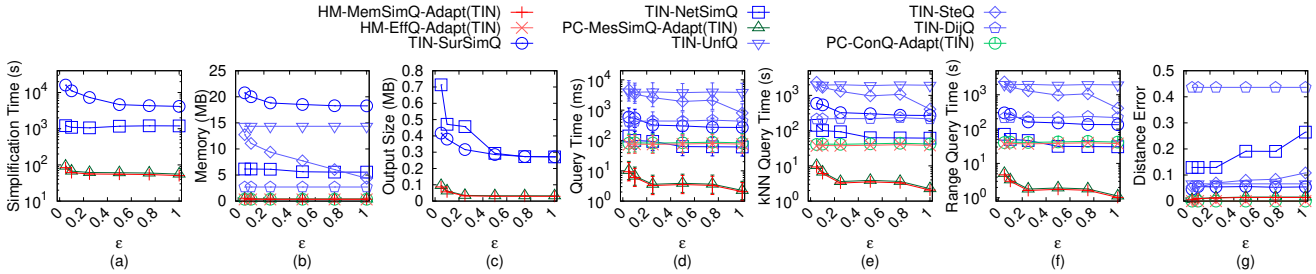

Figure 62: Baseline comparisons (effect of ϵ on BH_t *-small TIN* dataset)

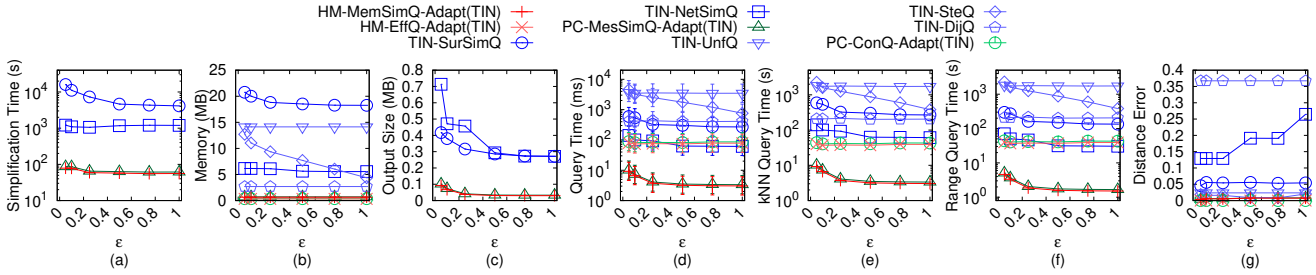

Figure 63: Baseline comparisons (effect of ϵ on EP_t *-small TIN* dataset)

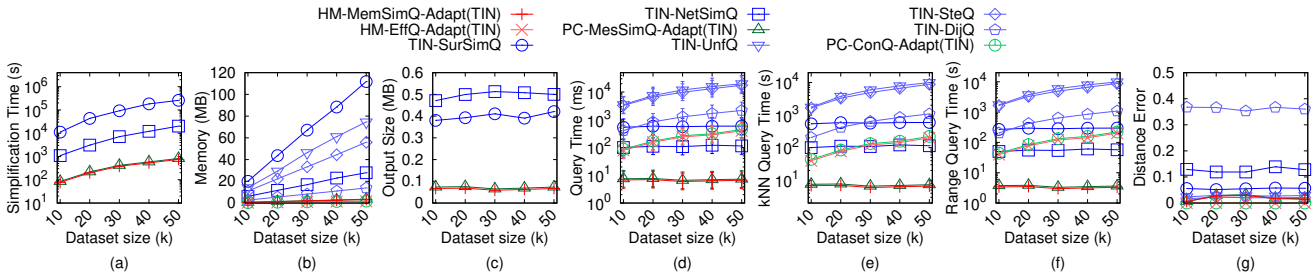

Figure 64: Baseline comparisons (effect of n on EP_t *-small TIN* dataset)

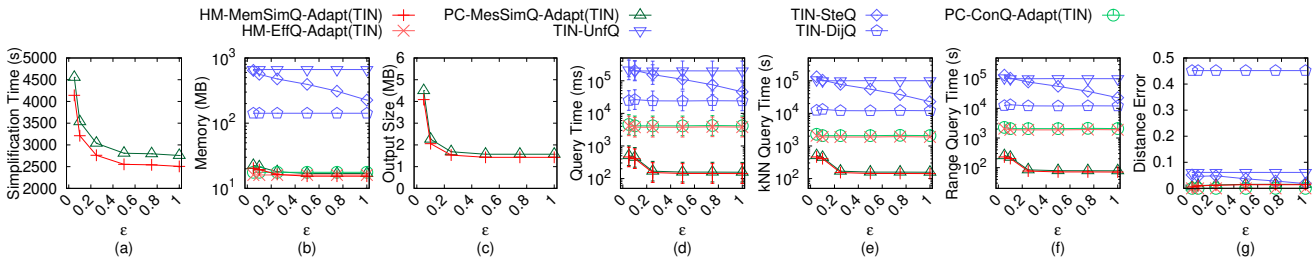

Figure 65: Baseline comparisons (effect of ϵ on GF_t *TIN* dataset)

and shortest path query time. The range query time of *HM-EffQ-Adapt(TIN)* is 100 times smaller than that of *TIN-UnfQ* (although *HM-EffQ-Adapt(TIN)* requires constructing a height map using the given *TIN*, and *TIN-UnfQ* does not involve any additional steps). The distance error of *HM-EffQ-Adapt(TIN)* is 0.06, but the distance error of *TIN-DijQ* is 0.45.

Effect of n (scalability test): In Figure 64, we tested 5 values of n in {50, 100, 150, 200, 250} on EP_t*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 66, Figure 68, Figure 70, Figure 72 and Figure 74 we tested 5 values of n in {500, 1000, 1500, 2000, 2500} on GF_t, LM_t, RM_t, BH_t and EP_t dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-MemSimQ-Adapt(TIN)* still outperforms other baselines.

D PROOF

Proof of Theorem 2.1. The proof is by transforming Minimum T-Spanner Problem [14] in Problem 2, which is an *NP-complete* problem, to the Height Map Simplification Problem.

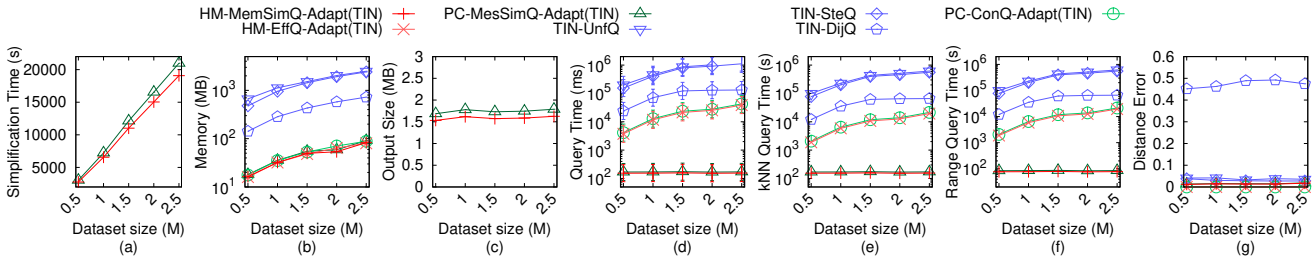

Figure 66: Baseline comparisons (effect of n on GF_t *TIN* dataset)

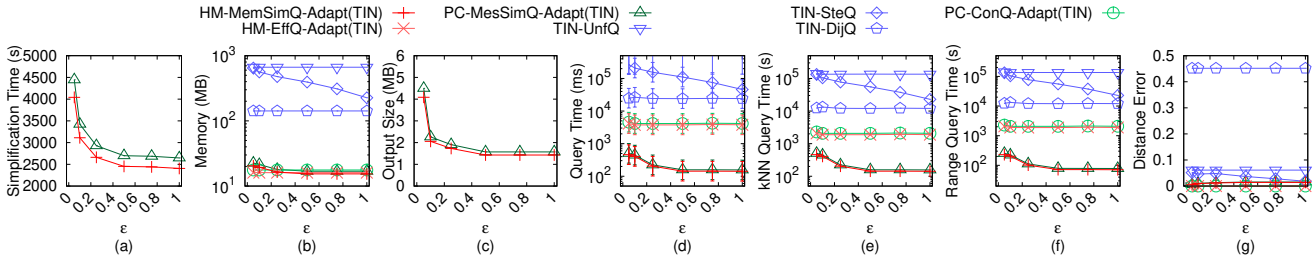

Figure 67: Baseline comparisons (effect of ϵ on LM_t *TIN* dataset)

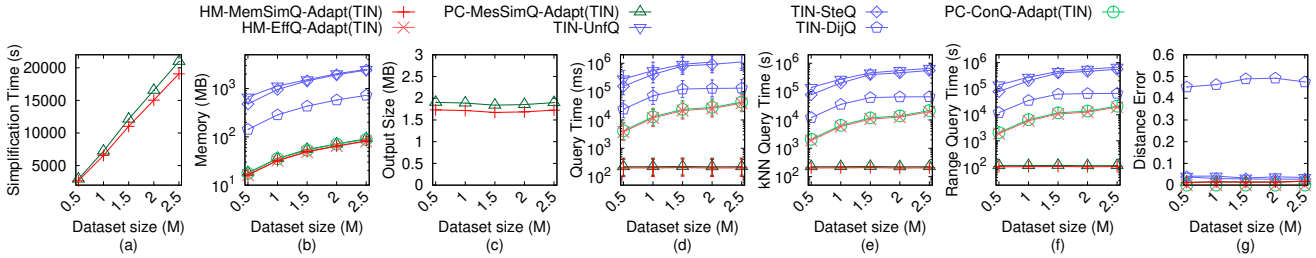

Figure 68: Baseline comparisons (effect of n on LM_t *TIN* dataset)

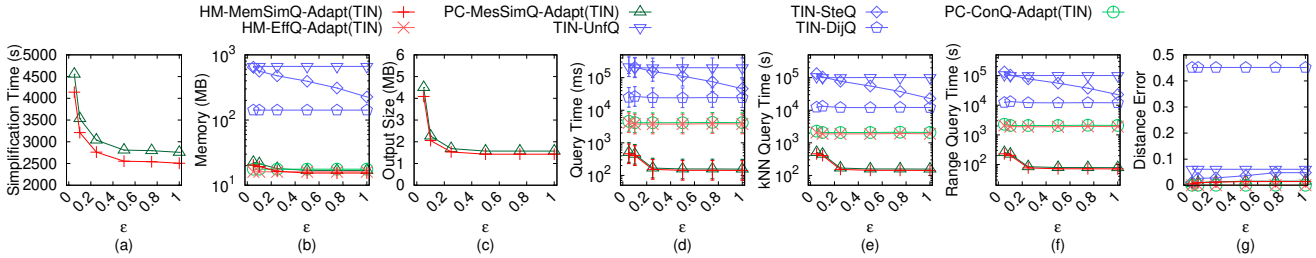

Figure 69: Baseline comparisons (effect of ϵ on RM_t *TIN* dataset)

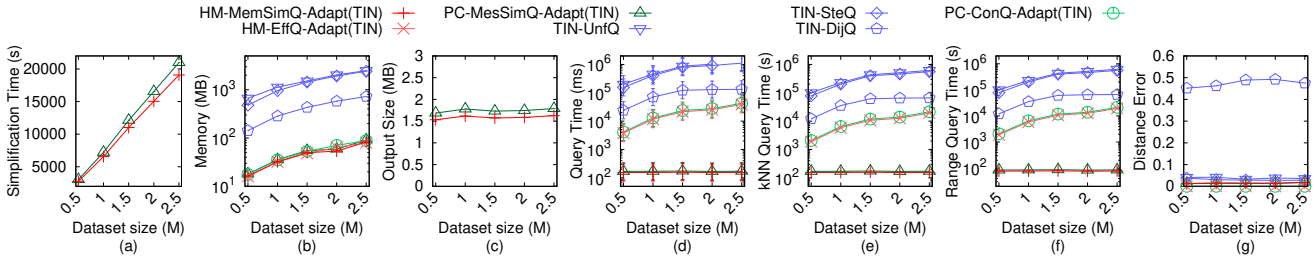

Figure 70: Baseline comparisons (effect of n on RM_t *TIN* dataset)

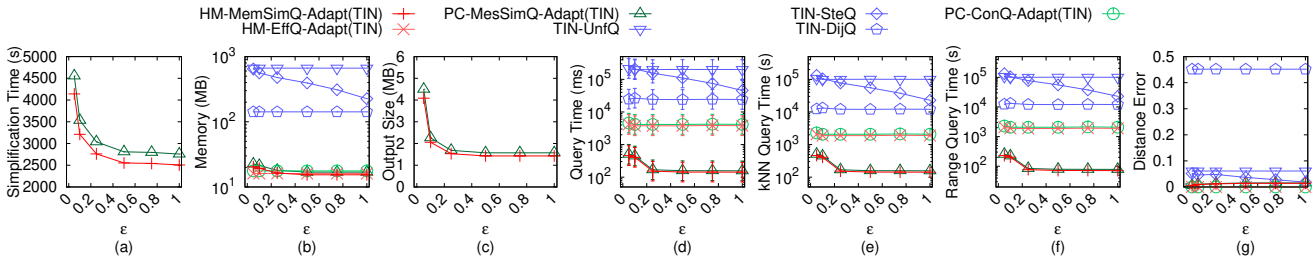

Figure 71: Baseline comparisons (effect of ϵ on BH_t *TIN* dataset)

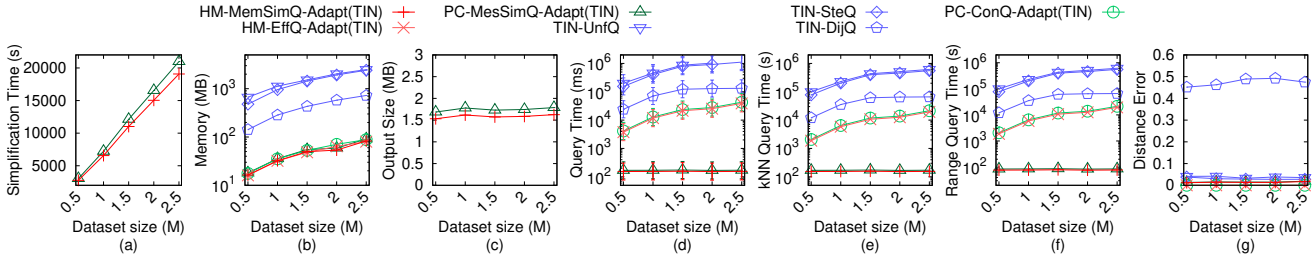

Figure 72: Baseline comparisons (effect of n on BH_t *TIN* dataset)

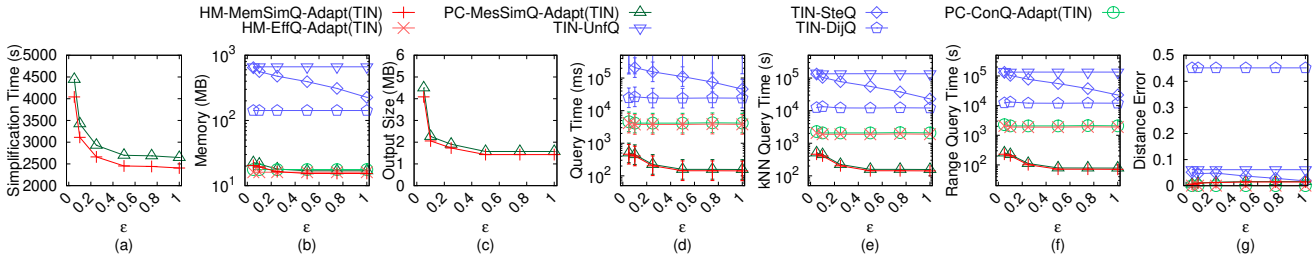

Figure 73: Baseline comparisons (effect of ϵ on EP_t *TIN* dataset)

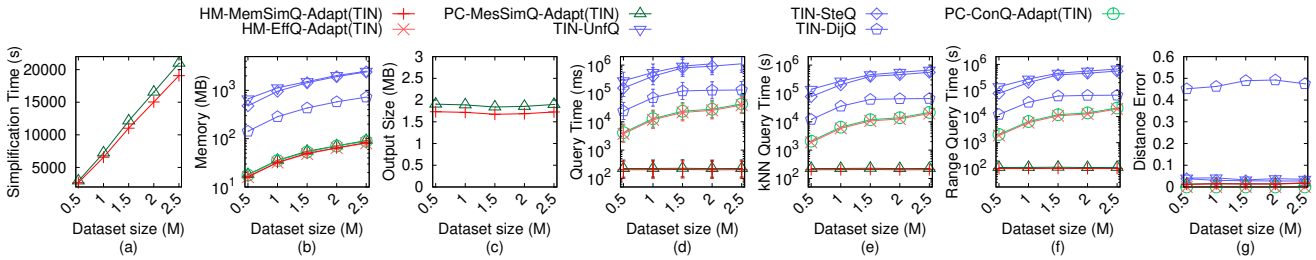

Figure 74: Baseline comparisons (effect of n on EP_t *TIN* dataset)

Problem 2 (Minimum T-Spanner Problem). *Given a graph G_{NPC} with a set of vertices $G_{NPC}.V$ and a set of edges $G_{NPC}.E$, a non-negative integer j and an error parameter t, can we find a sub-graph $\widetilde{G}_{NPC}$ of G_{NPC} with at most j edges, such that for all pairs of vertices s and t in $G_{NPC}.V$, $|\Pi(s,t|\widetilde{G}_{NPC})| \leq (1+\epsilon)|\Pi(s,t|G_{NPC})|$, where $\Pi(s,t|\widetilde{G}_{NPC}$ (resp. $\Pi(s,t|G_{NPC})$) is the shortest path between s and t on $\widetilde{G}_{NPC}$ (resp. G_{NPC})?*

But, in order to do this transformation, we need the Conceptual Graph Simplification Problem in Problem 3. We transfer the Minimum T-Spanner Problem to the Conceptual Graph Simplification Problem, and show that the Height Map Simplification Problem is equivalent to the Conceptual Graph Simplification Problem.

Problem 3 (Conceptual Graph Simplification Problem). *Given a conceptual graph G of M, a non-negative integer i' and an error parameter ϵ, can we find a simplified conceptual graph $\widetilde{G}$ of $\widetilde{M}$, with at most i' edges, such that for all pairs of vertices s and t in $G.V$, $(1-\epsilon)|\Pi(s,t|G)| \leq |\Pi(s,t|\widetilde{G})| \leq (1+\epsilon)|\Pi(s,t|G)|$?*

We then construct a complete conceptual graph G_C, with a set of vertices $G_C.V$ and a set of edges $G_C.E$. In $G_C.V$, it contains all the vertices in G (i.e., the pixel centers of M) and all possible new

Table 3: Summary of all notation

Notation	Meaning
M	The height map with a set of pixels
P	The set of pixels of M
$N(\cdot)$	The neighbour pixels table of M
n	The number of pixels of M
C	The point cloud constructed by M
T	The *TIN* constructed by M
θ	The minimum inner angle of any face in T
G	M's and C's conceptual graph
$G.V/G.E$	The set of vertices and edges of G
$\Pi(s,t\|M)$	The shortest path passing on M between s and t
$\|\Pi(s,t\|M)\|$	$\Pi(s,t\|M)$'s length
$\Pi(s,t\|C)$	The shortest path passing on C between s and t
$\Pi(s,t\|T)$	The shortest surface path passing on T between s and t
$\Pi_N(s,t\|T)$	The shortest network path passing on T between s and t
$\Pi_E(s,t\|T)$	The shortest path passing on the edges of T between s and t where these edges belongs to the faces that $\Pi(s,t\|T)$ passes
$\widetilde{M}$	The simplified height map
$\widetilde{P}$	The set of pixels of $\widetilde{M}$
$\widetilde{N}(\cdot)$	The neighbour pixels table of $\widetilde{M}$
P_{rema}	The set of remaining pixels
P_{dd}	The set of added pixels
$\widetilde{G}$	$\widetilde{M}$'s simplified conceptual graph
$\widetilde{G}.V/\widetilde{G}.E$	The set of vertices and edges of $\widetilde{G}$
$\Pi(s,t\|\widetilde{M})$	The shortest path passing on $\widetilde{M}$ between s and t
ϵ	The error parameter
l_{max}/l_{min}	The longest / shortest edge's length of T
$C(\cdot)$	The containing table
$C^{-1}(\cdot)$	The belonging table
$\widehat{P}$	The set of adjacent pixels that we need to merge in each simplification iteration
p_{add}	The added pixel formed by merging each pixel $\widehat{P}$
$\widetilde{p}$	The estimated pixel of p
$\Pi_1(p,q\|\widetilde{M})$	The intra-path passing on $\widetilde{M}$ between p and q
$\Pi_2(p,q\|\widetilde{M})$	The inter-path passing on $\widetilde{M}$ between p and q
$L(p_{add})$	The set of linked added pixels of p_{add}

vertices in $\widetilde{G}$ (i.e., all possible added pixels in $\widetilde{M}$). Figure 75 (a) shows a height map, Figure 75 (b) shows the height map's complete conceptual graph in a x-y plane. In Figure 75 (b), (1) each orange point represents 1 vertex with the same x-, y- and z-coordinate of the corresponding vertex in the original conceptual graph G, and (2) each green point represents 256 vertices with the same x- and y-coordinate values of the possible new vertex, but with 256 different z-coordinate values in [0, 255] (because in a height map, a pixel with different grayscale color can represent at most 256 different elevation value), (3) the middle green point with an orange outline represents (i) 1 vertex with the same x-, y- and z-coordinate of the corresponding vertex in G, and (ii) 255 vertices with the same x- and y-coordinate values of the possible new vertex, but with 255 different z-coordinate values in [0, 255] except for the z-coordinate value of the corresponding vertex in G. These points form a set of vertices in $G_C.V$. There is an edge connecting all pairs of vertices in $G_C.V$, and these edges form $G_C.E$. Figure 75 (c) shows G_C with 4 + 256 = 260 vertices in 3D space. In this figure, there should have total 256 green points, but only 5 of them are shown for the sake of illustration. In addition, there should have an edge between each pair of points, we omit some of them for the sake of illustration. Clearly, G and $\widetilde{G}$ are both sub-graphs of G_C. Given a pair of vertices s and t in $G_C.V$, and the original conceptual graph G, let $\Pi(s,t|G_C)$ be the shortest path between s and t passing on G_C, and we set $\Pi(s,t|G_C) = \Pi(s,t|G)$. We can simply regard $\Pi(s,t|G_C)$ as a function, such that given s, t, and G, it can return a result. When s or t are on G_C, but not on G nor $\widetilde{G}$, we can simply regard $\Pi(s,t|G_C)$ as *NULL*.

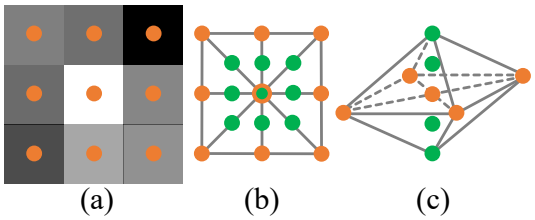

Figure 75: (a) A height map, (b) a height map's complete conceptual graph in a x-y plane, and (c) a complete conceptual graph in a 3D space

The transformation from the Minimum T-Spanner Problem to the Conceptual Graph Simplification Problem is as follows. We transfer G_{NPC} to G_C, transfer checking "can we find a sub-graph $\widetilde{G}_{NPC}$ of G_{NPC} with at most j edges, such that for all pairs of vertices s and t in $G_{NPC}.V$, $|\Pi(s,t|\widetilde{G}_{NPC})| \leq (1+\epsilon)|\Pi(s,t|G_{NPC})|$" to "can we find a simplified conceptual graph $\widetilde{G}$ of G_C, with at most i' edges, such that for all pairs of vertices s and t in $G_C.V$, $(1-\epsilon)|\Pi(s,t|G)| = (1-\epsilon)|\Pi(s,t|G_C)| \leq |\Pi(s,t|\widetilde{G})| \leq (1+\epsilon)|\Pi(s,t|G_C)| = (1+\epsilon)|\Pi(s,t|G)|$". Note that in the Conceptual Graph Simplification Problem, no matter whether the given graph is G or G_C, the given graph G or G_C will not affect the problem transformation, since the transformation is about the checking of the distance requirement, and given s and t, we have defined $\Pi(s,t|G_C) = \Pi(s,t|G)$. The transformation can be finished in polynomial time. Since the height map M and the conceptual graph G are equivalent, and i and i' can be any value, the Height Map Simplification Problem is equivalent to the Conceptual Graph Simplification Problem. Thus, when the Conceptual Graph Simplification Problem is solved, the Height Map Simplification Problem is solved equivalently, and the Minimum T-Spanner Problem is also solved. Since the Minimum T-Spanner Problem is *NP-complete*, the Height Map Simplification Problem is *NP-hard*. □

Lemma D.1. *Given a height map M, algorithm HM-MemSimQ returns a simplified height map $\widetilde{M}$ of M, such that for all pairs of pixels s_1 and t_1 both in P_{rema}, $(1-\epsilon)|\Pi(s_1,t_1|M)| \leq |\Pi(s_1,t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1,t_1|M)|$.*

Table 4: Comparison of algorithms

Algorithm	Simplification time		Output size		Shortest path query time		*kNN* and range query time		Error
Simplification algorithm									
TIN-SurSimQ-Adapt(HM) [26, 29]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-NetSimQ-Adapt(HM) [29]	$O(n^2\log n)$	Medium	$O(n)$	Large	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
PC-MesSimQ-Adapt(HM) [53]	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
HM-MemSimQ-LQT1	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	Small
HM-MemSimQ-LQT2	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{nn'}{\log n}\log\frac{n}{\log n})$	Medium	Small
HM-MemSimQ-LS	$O(n\sqrt[3]{n}\log n)$	Small	$O(n)$	Large	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Small
HM-MemSimQ-LST	$O(n^2\sqrt[3]{n}\log n)$	Large	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
***HM-MemSimQ* (ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	**Small**
Shortest path query algorithm									
TIN-UnfQ-Adapt(HM) [15, 47, 54]	-	N/A	-	N/A	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-SteQ-Adapt(HM) [28, 52]	-	N/A	-	N/A	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-DijQ-Adapt(HM) [29]	-	N/A	-	N/A	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
PC-ConQ-Adapt(HM) [53]	-	N/A	-	N/A	$O(n\log n)$	Medium	$O(n\log n)$	Medium	No error
***HM-EffQ* (ours)**	-	N/A	-	N/A	$\boldsymbol{O(n\log n)}$	**Medium**	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**

Proof. We use mathematical induction to prove it. In algorithm *HM-MemSimQ*, even though it simplifies a height map using two different two simplification techniques, i.e., four adjacent pixels merging and adjacent pixels any direction expanded merging, the logic is the same, and we always perform the same distance checking, i.e., *R2R* distance checking, *R2D* distance checking and *D2D* distance checking. Thus, there is no need to distinguish these two simplification techniques in the following proof, and we regard any one step of the simplification process in these two simplification techniques as one equivalent iteration.

For the base case, we show that after the first simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration.

- Firstly, we show that $(1-\epsilon)|\Pi(s_1,t_1|M)| \le |\Pi(s_1,t_1|\widetilde{M})|$. Along $\Pi(s_1,t_1|\widetilde{M})$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}$ (resp. $\overline{q}$) be the first intersection pixel between $\Pi(s_1,t_1|\widetilde{M})$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1,t_1|\widetilde{M})| = |\Pi(s_1,\overline{p}|\widetilde{M})| + |\Pi(\overline{p},\overline{q}|\widetilde{M})| + |\Pi(\overline{q},t_1|\widetilde{M})|$. Since $\overline{p}$ and $\overline{q}$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $(1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| \le |\Pi(\overline{p},\overline{q}|\widetilde{M})|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}$ and $\overline{q}$ are also in P_{rema}, and there is no difference between $\widetilde{M}$ and M (apart from the changes of P_{add}), we have $(1-\epsilon)|\Pi(s_1,\overline{p}|M)| = (1-\epsilon)|\Pi(s_1,\overline{p}|\widetilde{M})| \le |\Pi(s_1,\overline{p}|\widetilde{M})|$ and $(1-\epsilon)|\Pi(\overline{q},t_1|M)| = (1-\epsilon)|\Pi(\overline{q},t_1|\widetilde{M})| \le |\Pi(\overline{q},t_1|\widetilde{M})|$. Thus, we have $|\Pi(s_1,t_1|\widetilde{M})| = |\Pi(s_1,\overline{p}|\widetilde{M})| + |\Pi(\overline{p},\overline{q}|\widetilde{M})| + |\Pi(\overline{q},t_1|\widetilde{M})| \ge (1-\epsilon)|\Pi(s_1,\overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q},t_1|M)| \ge (1-\epsilon)|\Pi(s_1,t_1|M)|$.
- Secondly, we show that $|\Pi(s_1,t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,t_1|M)|$. Along $\Pi(s_1,t_1|M)$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}'$ (resp. $\overline{q}'$) be the first intersection pixel between $\Pi(s_1,t_1|M)$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1,t_1|M)| = |\Pi(s_1,\overline{p}'|M)| + |\Pi(\overline{p}',\overline{q}'|M)| + |\Pi(\overline{q}',t_1|M)|$. Since $\overline{p}'$ and $\overline{q}'$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $|\Pi(\overline{p}',\overline{q}'|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|M)|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in P_{rema}, and there is no difference between $\widetilde{M}$ and M (apart from the changes of P_{add}), we have $|\Pi(s_1,\overline{p}'|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,\overline{p}'|\widetilde{M})| = (1+\epsilon)|\Pi(s_1,\overline{p}'|M)|$ and $|\Pi(\overline{q}',t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{q}',t_1|\widetilde{M})| = (1+\epsilon)|\Pi(\overline{q}',t_1|M)|$. Thus, we have $(1+\epsilon)|\Pi(s_1,t_1|M)| = (1+\epsilon)|\Pi(s_1,\overline{p}'|M)| + (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|M)| + (1+\epsilon)|\Pi(\overline{q}',t_1|M)| \ge |\Pi(s_1,\overline{p}'|\widetilde{M})| + |\Pi(\overline{p}',\overline{q}'|\widetilde{M})| + |\Pi(\overline{q}',t_1|\widetilde{M})| \ge |\Pi(s_1,t_1|\widetilde{M})|$.

For the hypothesis case, assume that after the i-th simplification iteration, for all pairs of pixels s_1 and t_1 both in P_{rema}, we have $(1-\epsilon)|\Pi(s_1,t_1|M)| \le |\Pi(s_1,t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,t_1|M)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration.

- Firstly, we show that $(1-\epsilon)|\Pi(s_1,t_1|M)| \le |\Pi(s_1,t_1|\widetilde{M})|$. Along $\Pi(s_1,t_1|\widetilde{M})$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}$ (resp. $\overline{q}$) be the first intersection pixel between $\Pi(s_1,t_1|\widetilde{M})$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1,t_1|\widetilde{M})| = |\Pi(s_1,\overline{p}|\widetilde{M})| + |\Pi(\overline{p},\overline{q}|\widetilde{M})| + |\Pi(\overline{q},t_1|\widetilde{M})|$. Since $\overline{p}$ and $\overline{q}$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $(1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| \le |\Pi(\overline{p},\overline{q}|\widetilde{M})|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}$ and $\overline{q}$ are also in P_{rema}, we have $(1-\epsilon)|\Pi(s_1,\overline{p}|M)| \le |\Pi(s_1,\overline{p}|\widetilde{M})|$ and $(1-\epsilon)|\Pi(\overline{q},t_1|M)| \le |\Pi(\overline{q},t_1|\widetilde{M})|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $|\Pi(s_1,t_1|\widetilde{M})| = |\Pi(s_1,\overline{p}|\widetilde{M})| + |\Pi(\overline{p},\overline{q}|\widetilde{M})| + |\Pi(\overline{q},t_1|\widetilde{M})| \ge (1-\epsilon)|\Pi(s_1,\overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q},t_1|M)| \ge (1-\epsilon)|\Pi(s_1,t_1|M)|$.
- Secondly, we show that $|\Pi(s_1,t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,t_1|M)|$. Along $\Pi(s_1,t_1|M)$ from s_1 to t_1 (resp. from t_1 to s_1), let $\overline{p}'$ (resp. $\overline{q}'$) be the first intersection pixel between $\Pi(s_1,t_1|M)$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1,t_1|M)| = |\Pi(s_1,\overline{p}'|M)| + |\Pi(\overline{p}',\overline{q}'|M)| + |\Pi(\overline{q}',t_1|M)|$. Since $\overline{p}'$ and $\overline{q}'$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $|\Pi(\overline{p}',\overline{q}'|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|M)|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in P_{rema}, we have $|\Pi(s_1,\overline{p}'|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,\overline{p}'|M)|$ and $|\Pi(\overline{q}',t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{q}',t_1|M)|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $(1+\epsilon)|\Pi(s_1,t_1|M)| = (1+\epsilon)|\Pi(s_1,\overline{p}'|M)| + (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|M)| + (1+\epsilon)|\Pi(\overline{q}',t_1|M)| \ge |\Pi(s_1,\overline{p}'|\widetilde{M})| + |\Pi(\overline{p}',\overline{q}'|\widetilde{M})| + |\Pi(\overline{q}',t_1|\widetilde{M})| \ge |\Pi(s_1,t_1|\widetilde{M})|$.

Thus, we have proved that for all pairs of pixels s_1 and t_1 both in P_{rema}, $(1-\epsilon)|\Pi(s_1,t_1|M)| \le |\Pi(s_1,t_1|\widetilde{M})| \le (1+\epsilon)|\Pi(s_1,t_1|M)|$. □

Lemma D.2. *Given a height map M, algorithm HM-MemSimQ returns a simplified height map $\widetilde{M}$ of M, such that for all pairs of pixels s_2 in P_{rema} and t_2 in $P-P_{rema}$, $(1-\epsilon)|\Pi(s_2,t_2|M)| \le |\Pi(s_2,t_2|\widetilde{M})| \le (1+\epsilon)|\Pi(s_2,t_2|M)|$.*

Proof. We use mathematical induction to prove it. Similar to the proof of Lemma D.1, there is no need to distinguish two simplification techniques, and we regard any one step of the simplification process in the two simplification techniques as one equivalent iteration.

For the base case, we show that after the first simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration. Since this is the first iteration, there are no other deleted pixels except the pixels belonging to P_{add}, we just need to show that the inequality holds when t_2 is any one of the deleted pixels belong to linked added pixels of P_{add}.

- Firstly, we show that $(1-\epsilon)|\Pi(s_2,t_2|M)| \le |\Pi(s_2,t_2|\widetilde{M})|$. Along $\Pi(s_2,t_2|\widetilde{M})$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}$ be the first intersection pixel between $\Pi(s_2,t_2|\widetilde{M})$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_2,t_2|\widetilde{M})| = |\Pi(s_2,\overline{m}|\widetilde{M})| + |\Pi(\overline{m},t_2|\widetilde{M})|$. Since $\overline{m}$ is in P_{rema}, which is a remaining neighbour pixel of linked added pixels of P_{add}, and t_2 is in $P-P_{rema}$, we have $(1-\epsilon)|\Pi(\overline{m},t_2|M)| \le |\Pi(\overline{m},t_2|\widetilde{M})|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}$ are both in P_{rema}, we have $(1-\epsilon)|\Pi(s_2,\overline{m}|M)| \le |\Pi(s_2,\overline{m}|\widetilde{M})|$ from Lemma D.1. Thus, we have $|\Pi(s_2,t_2|\widetilde{M})| = |\Pi(s_2,\overline{m}|\widetilde{M})| + |\Pi(\overline{m},t_2|\widetilde{M})| \ge (1-\epsilon)|\Pi(s_2,\overline{m}|M)| + (1-\epsilon)|\Pi(\overline{m},t_2|M)| \ge (1-\epsilon)|\Pi(s_2,t_2|M)|$.
- Secondly, we show that $|\Pi(s_2,t_2|\widetilde{M})| \le (1+\epsilon)|\Pi(s_2,t_2|M)|$. Along $\Pi(s_2,t_2|M)$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}'$ be the first intersection pixel between $\Pi(s_2,t_2|M)$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_2,t_2|M)| = |\Pi(s_2,\overline{m}'|M)| + |\Pi(\overline{m}',t_2|M)|$. Since $\overline{m}'$ is in P_{rema}, which is a remaining neighbour pixel of linked added pixels of P_{add}, and t_2 is in $P-P_{rema}$, we have $|\Pi(\overline{m},t_2|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{m},t_2|M)|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}'$ are both in P_{rema}, we have $|\Pi(s_2,\overline{m}'|\widetilde{M})| \le (1+\epsilon)|\Pi(s_2,\overline{m}'|M)|$ from Lemma D.1. Thus, we have $(1+\epsilon)|\Pi(s_2,t_2|M)| = (1+\epsilon)|\Pi(s_2,\overline{m}'|M)| + (1+\epsilon)|\Pi(\overline{m}',t_2|M)| \ge |\Pi(s_2,\overline{m}'|\widetilde{M})| + |\Pi(\overline{m}',t_2|\widetilde{M})| \ge |\Pi(s_2,t_2|\widetilde{M})|$.

For the hypothesis case, assume that after the i-th simplification iteration, for all pairs of pixels s_2 in P_{rema} and t_2 in $P-P_{rema}$, we have $(1-\epsilon)|\Pi(s_2,t_2|M)| \le |\Pi(s_2,t_2|\widetilde{M})| \le (1+\epsilon)|\Pi(s_2,t_2|M)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration. Since the difference of $\widetilde{M}$ after the i-th simplification iteration and the $(i+1)$-th simplification iteration is due to the changes of P_{add}, we just need to show that the inequality holds when t_2 is any one of the deleted pixels belong to linked added pixels P_{add}. The proof is exactly the same as in the base case.

Thus, we have proved that for all pairs of pixels s_2 in P_{rema} and t_2 in $P-P_{rema}$, $(1-\epsilon)|\Pi(s_2,t_2|M)| \le |\Pi(s_2,t_2|\widetilde{M})| \le (1+\epsilon)|\Pi(s_2,t_2|M)|$. □

Lemma D.3. *Given a height map M, algorithm HM-MemSimQ returns a simplified height map $\widetilde{M}$ of M, such that for all pairs of pixels s_3 and t_3 in $P-P_{rema}$, $(1-\epsilon)|\Pi(s_3,t_3|M)| \le |\Pi(s_3,t_3|\widetilde{M})| \le (1+\epsilon)|\Pi(s_3,t_3|M)|$.*

Proof. Similar to the proof of Lemma D.1, there is no need to distinguish two simplification techniques, and we regard any one step of the simplification process in the two simplification techniques as one equivalent iteration. There are two sub-cases. (a) $\Pi(s,t|\widetilde{M})$ does not pass on pixels in P_{rema}. (b) $\Pi(s,t|\widetilde{M})$ passes on pixels in P_{rema}.

(1) We prove the first sub-case, i.e., $\Pi(s,t|\widetilde{M})$ does not pass on pixels in P_{rema}. We use mathematical induction to prove it.

For the base case, we show that after the first simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration. Since this is the first iteration, there are no other deleted pixels except the pixels belonging to P_{add}, we just need to show that the inequality holds when s_3 and t_3 are any one of the deleted pixels belong to P_{add}. Due to the *D2D* distance checking, we have $(1-\epsilon)|\Pi(s_3,t_3|M)| \le |\Pi(s_3,t_3|\widetilde{M})| \le (1+\epsilon)|\Pi(s_3,t_3|M)|$.

For the hypothesis case, assume that after the i-th simplification iteration, for all pairs of pixels s_3 and t_3 both in $P-P_{rema}$, we have $(1-\epsilon)|\Pi(s_3,t_3|M)| \le |\Pi(s_3,t_3|\widetilde{M})| \le (1+\epsilon)|\Pi(s_3,t_3|M)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration. Since the difference of $\widetilde{M}$ after the i-th simplification iteration and the $(i+1)$-th simplification iteration is due to the changes of P_{add}, we just need to show that the inequality holds when t_3 is any one of the deleted pixels belong to linked added pixels of P_{add}. The proof is exactly the same as in the base case.

Thus, we have proved that for all pairs of pixels s_3 in P_{rema} and t_3 in $P-P_{rema}$, when $\Pi(s,t|\widetilde{M})$ does not pass on pixels in P_{rema}, $(1-\epsilon)|\Pi(s_3,t_3|M)| \le |\Pi(s_3,t_3|\widetilde{M})| \le (1+\epsilon)|\Pi(s_3,t_3|M)|$.

(2) We prove the second sub-case, i.e., $\Pi(s,t|\widetilde{M})$ passes on pixels in P_{rema}. We use the Lemma D.1 and Lemma D.2 to prove it.

- Firstly, we show that $(1-\epsilon)|\Pi(s_3,t_3|M)| \le |\Pi(s_3,t_3|\widetilde{M})|$. Along $\Pi(s_3,t_3|\widetilde{M})$ from $\widetilde{s_3}$ to $\widetilde{t_3}$ (resp. from $\widetilde{t_3}$ to $\widetilde{s_3}$), let $\overline{p}$ (resp. $\overline{q}$) be the first intersection pixel between $\Pi(s_3,t_3|\widetilde{M})$ and the remaining neighbour pixels of linked added pixels of $C^{-1}(s_3)$ (resp. $C^{-1}(t_3)$). We have $|\Pi(s_3,t_3|\widetilde{M})| = |\Pi_1(s_3,\overline{p}|\widetilde{M})| + |\Pi_2(\overline{p},\overline{q}|\widetilde{M})| + |\Pi_1(\overline{q},t_3|\widetilde{M})|$. Since $\overline{p}$ and $\overline{q}$ are in P_{rema}, we have $(1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| \le |\Pi_2(\overline{p},\overline{q}|\widetilde{M})|$ by Lemma D.1. Since s_3 and t_3 are in $P-P_{rema}$, and $\overline{p}$ and $\overline{q}$ are in P_{rema}, we have $(1-\epsilon)|\Pi(s_3,\overline{p}|M)| \le |\Pi_1(s_3,\overline{p}|\widetilde{M})|$ and $(1-\epsilon)|\Pi(\overline{q},t_3|M)| \le |\Pi_1(\overline{q},t_3|\widetilde{M})|$ by Lemma D.2. Thus, we have $|\Pi(s_3,t_3|\widetilde{M})| = |\Pi_1(s_3,\overline{p}|\widetilde{M})| + |\Pi_2(\overline{p},\overline{q}|\widetilde{M})| + |\Pi_1(\overline{q},t_3|\widetilde{M})| \ge (1-\epsilon)|\Pi(s_3,\overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p},\overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q},t_3|M)| \ge (1-\epsilon)|\Pi(s_3,t_3|M)|$.
- Secondly, we show that $|\Pi(s_3,t_3|\widetilde{M})| \le (1+\epsilon)|\Pi(s_3,t_3|M)|$. Along $\Pi(s_3,t_3|M)$ from s_3 to t_3 (resp. from t_3 to s_3), let $\overline{p}'$ (resp. $\overline{q}'$) be the first intersection pixel between $\Pi(s_3,t_3|M)$ and the remaining neighbour pixels of linked added pixels of $C^{-1}(s_3)$ (resp. $C^{-1}(t_3)$). We have $|\Pi(s_3,t_3|M)| = |\Pi(s_3,\overline{p}'|M)| + |\Pi(\overline{p}',\overline{q}'|M)| + |\Pi(\overline{q}',t_3|M)|$. Since $\overline{p}'$ and $\overline{q}'$ are in P_{rema}, we have $|\Pi_2(\overline{p}',\overline{q}'|\widetilde{M})| \le (1+\epsilon)|\Pi(\overline{p}',\overline{q}'|M)|$ by Lemma D.1. Since s_3 and t_3 are in $P-P_{rema}$, and $\overline{p}'$ and $\overline{q}'$ are in

P_{rema}, we have $|\Pi_1(s_3, \overline{p}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, \overline{p}'|M)|$ and $|\Pi_1(\overline{q}', t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{q}', t_3|M)|$ by Lemma D.2. Thus, we have $(1+\epsilon)|\Pi(s_3, t_3|M)| = (1+\epsilon)|\Pi(s_3, \overline{p}'|M)| + (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)| + (1+\epsilon)|\Pi(\overline{q}', t_3|M)| \geq |\Pi_1(s_3, \overline{p}'|\widetilde{M})| + |\Pi_2(\overline{p}', \overline{q}'|\widetilde{M})| + |\Pi_1(\overline{q}', t_3|\widetilde{M})| \geq |\Pi(s_3, t_3|\widetilde{M})|$.

Thus, we have proved that for all pairs of pixels s_3 in P_{rema} and t_3 in $P - P_{rema}$, when $\Pi(s, t|\widetilde{M})$ passes on pixels in P_{rema}, $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$.

In general, we have proved that for all pairs of pixels s_3 in P_{rema} and t_3 in $P - P_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$. □

Proof of Theorem 4.2. Firstly, we prove the simplification time. In each simplification iteration of the *R2R*, *R2D* and *D2D* distance checking, since we only check the pixels related to the neighbour pixels of linked added pixels of an added pixel, there are $O(1)$ such pixels. Since we use Dijkstra's algorithm in $O(n \log n)$ time for distance calculation, the distance checking needs $O(1)$ time. In both of the four adjacent pixels merging and the adjacent pixels any direction expanded merging, we always expand by one pixel in four directions. That is, we keep removing $2^2, 3^2, \ldots, i^2$ until we have deleted all n points. Let i be the total number of iterations we need to perform, and we have $2^2 + 3^2 + \cdots + i^2 = n$, which is equivalent to $\frac{i(i+1)(2i+1)}{6} - 1 = n$. We solve i and obtain $i = O(\sqrt[3]{n})$. In general, we need $O(\sqrt[3]{n})$ iterations, where each iteration need $O(n \log n)$ for distance checking. Thus, the simplification time is $O(n\sqrt[3]{n} \log n)$.

Secondly, we prove the output size. Our experiments show that each added pixel can dominate $O(\log n)$ deleted pixels on average. Since there are total n pixels on M, we obtain that there are $O(\frac{n}{\log n})$ pixels on $\widetilde{M}$.

Thirdly, we prove the shortest path query time. Since there are $O(\frac{n}{\log n})$ pixels on $\widetilde{M}$, and we use Dijkstra's algorithm on $\widetilde{M}$ for once, the shortest path query time is $O(\frac{n}{\log n} \log \frac{n}{\log n})$.

Finally, we prove that $\widetilde{M}$ is an ϵ-approximate simplified height map of M. We need to show that for all pairs of pixels s and t on M, $(1-\epsilon)|\Pi(s, t|M)| \leq |\Pi(s, t|\widetilde{M})| \leq (1+\epsilon)|\Pi(s, t|M)|$. There are three cases. (1) For the *both pixels remaining case*, from Lemma D.1, we know that for all pairs of pixels s_1 and t_1 both in P_{rema}, $(1-\epsilon)|\Pi(s_1, t_1|M)| \leq |\Pi(s_1, t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, t_1|M)|$. (2) For the *one pixel deleted and one pixel remaining case*, from Lemma D.2, we know that for all pairs of pixels s_2 in P_{rema} and t_2 in $P - P_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$. (3) For the *both pixels deleted case*, there are two more sub-cases: (i) $\Pi(s, t|\widetilde{M})$ does not pass on pixels in P_{rema}, which contains a special case of *different and non-adjacent belonging pixel in both pixels deleted case* (i.e., $\Pi(s, t|\widetilde{M})$ only passes on added pixel and other linked added pixels of it), *different and adjacent belonging pixel in both pixels deleted case* and *same belonging pixel in both pixels deleted case*. (ii) $\Pi(s, t|\widetilde{M})$ passes on pixels in P_{rema}, which contains *different and non-adjacent belonging pixel in both pixels deleted case*. From Lemma D.3, we know that for all pairs of pixels s_3 and t_3 both in $P - P_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$ for both these two sub-cases. In general, we have considered all three cases for s and t, and we obtain that $\widetilde{M}$ is an ϵ-approximate simplified height map of M. □

Proof of Lemma 4.3. Firstly, we prove the query time of both the *kNN* and range query algorithm.

- For algorithm *HM-EffQ*, given a query pixel q, we just need to perform one Dijkstra's algorithm on M.
- For algorithm *HM-MemSimQ*, given a query pixel q, if q is a remaining pixel, we just need to perform one Dijkstra's algorithm on $\widetilde{M}$; if q is a deleted pixel, we just need to perform $\widetilde{N}(C^{-1}(q))$ Dijkstra's algorithm on $\widetilde{M}$. Since $\widetilde{N}(C^{-1}(q))$ is a constant, it can be omitted in the big-O notation.

Since performing one Dijkstra's algorithm on M and $\widetilde{M}$ are $O(n \log n)$ and $O(\frac{n}{\log n} \log \frac{n}{\log n})$ (i.e., the shortest path query time for algorithm *HM-EffQ* and *HM-MemSimQ*), respectively, the query time of both the *kNN* and range query by using algorithm *HM-EffQ* is $O(n \log n)$ and *HM-MemSimQ* is $O(\frac{n}{\log n} \log \frac{n}{\log n})$.

Secondly, we prove the error rate of both the *kNN* and range query algorithm.

- For algorithm *HM-EffQ*, it returns the exact shortest path passing on M, so it also returns the exact result for the *kNN* and range query.
- For algorithm *HM-MemSimQ*, we give some notation first. For the *kNN* query and the range query, both of which return a set of objects, we can simplify the notation by denoting the set of objects returned using the shortest distance on M computed by algorithm *HM-EffQ* as X, where X contains either (1a) k nearest objects to query object i, or (1b) objects within a range of distance r in q. Similarly, we denote the set of objects returned using the shortest distance on $\widetilde{M}$ computed by algorithm *HM-MemSimQ* as X', where X' contains either (2a) k nearest objects to query object i, or (2b) objects within a range of distance r to i. In Figure 1 (a), suppose that the exact k nearest objects ($k = 2$) of a is c, d, i.e., $X = \{c, d\}$. Suppose that our *kNN* query algorithm finds the k nearest objects ($k = 2$) of a is b, c, i.e., $X' = \{b, c\}$. Recall that let p_f (resp. p'_f) be the object in X (resp. X') that is furthest from i based on the shortest distance on M, i.e., $|\Pi(i, p_f|M)| \leq \max_{\forall p \in X} |\Pi(i, p|M)|$ (resp. $|\Pi(i, p'_f|M)| \leq \max_{\forall p' \in X'} |\Pi(i, p'|M)|$). We further let q_f (resp. q'_f) be the object in X (resp. X') that is furthest from i based on the shortest distance on $\widetilde{M}$ returned by algorithm *HM-MemSimQ*, i.e., $|\Pi(i, q_f|\widetilde{M})| \leq \max_{\forall q \in X} |\Pi(i, q|\widetilde{M})|$ (resp. $|\Pi(i, q'_f|\widetilde{M})| \leq \max_{\forall q' \in X'} |\Pi(i, q'|\widetilde{M})|$). Recall the error rate of *kNN* and range queries is $\beta = \frac{|\Pi(i, p'_f|M)|}{|\Pi(i, p_f|M)|} - 1$. According to Theorem 4.1, we have $|\Pi(i, p'_f|\widetilde{M})| \geq (1-\epsilon)|\Pi(i, p'_f|M)|$. Thus, we have $\beta \leq \frac{|\Pi(i, p'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(i, p_f|M)|} - 1$. By the definition of p_f and q_f, we have $|\Pi(i, p_f|M)| \geq |\Pi(i, q_f|M)|$. Thus, we have $\beta \leq \frac{|\Pi(i, p'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(i, q_f|M)|} - 1$. By the definition of p'_f and q'_f, we have $|\Pi(i, p'_f|\widetilde{M})| \leq |\Pi(i, q'_f|\widetilde{M})|$. Thus, we have $\beta \leq \frac{|\Pi(i, q'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(i, q_f|M)|} - 1$. According to Theorem 4.1, we

have $|\Pi(i, q_f|\widetilde{M})| \leq (1+\epsilon)|\Pi(i, q_f|M)|$. Then, we have $\beta \leq \frac{(1+\epsilon)|\Pi(i,q'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(i,q_f|\widetilde{M})|} - 1$. By our *kNN* and range query algorithm, we have $|\Pi(i, q'_f|\widetilde{M})| \leq |\Pi(i, q_f|\widetilde{M})|$. Thus, we have $\beta \leq \frac{1+\epsilon}{1-\epsilon} - 1 = \frac{2\epsilon}{1-\epsilon}$. So, algorithm *HM-MemSimQ* has an error rate $\frac{2\epsilon}{1-\epsilon}$ for the *kNN* and range query.

□

Theorem D.4. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm TIN-SurSimQ-Adapt(HM) are* $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$, $O(n)$, $O(n^2)$ *and* $O(n^2)$, *respectively. Given a height map M, it first constructs a TIN T using M, and then returns a simplified TIN* $\widetilde{T}$ *of T such that* $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ *for all pairs of vertices s and t on T, where* $\Pi(s,t|\widetilde{T})$ *is the shortest surface path between s and t passing on* $\widetilde{T}$.

Proof. Firstly, we prove the simplification time. It first needs to construct the *TIN* using the height map in $O(n)$ time. Then, in each vertex removal iteration, it places $O(\frac{1}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ Steiner points [21, 26] on each face adjacent to the deleted vertex, and use algorithm *TIN-UnfQ* [15, 47, 54] in $O(n^2)$ time to check the distances between these Steiner points on the original *TIN* and the simplified *TIN*, so this step needs $O(\frac{n^2}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$. In general, the total simplification time is $O(n + \frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon}) = (\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$.

Secondly, we prove the output size. Although this algorithm could simplify a *TIN*, our experimental results show the simplified *TIN* still has $O(n)$ vertices. Thus, the output size is $O(n)$.

Thirdly, we prove the shortest path query time. Since there are $O(n)$ vertices on $\widetilde{T}$, we use algorithm *TIN-UnfQ* [15, 47, 54] in $O(n^2)$ time for the shortest path query. Thus, the shortest path query time is $O(n^2)$.

Fourthly, we prove the *kNN* and range query time. Since we just need to use algorithm *TIN-UnfQ* [15, 47, 54] (i.e., a single-source-all-destination algorithm) once for both the *kNN* and range query, the *kNN* and range query time is $O(n^2)$.

Finally, we prove that for all pairs of vertices s and t on T, algorithm *TIN-SurSimQ-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$. In each vertex removal iteration, it performs a check between all pairs of Steiner points u and v (on the faces that are adjacent to the deleted vertex) on T whether $(1-\epsilon)|\Pi(u,v|T)| \leq |\Pi(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi(u,v|T)|$. According to study [26], given any pair of points p and q (on the faces that are adjacent to the deleted vertex) on T, if $(1-\epsilon)|\Pi(u,v|T)| \leq |\Pi(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi(u,v|T)|$, then $(1-\epsilon)|\Pi(p,q|T)| \leq |\Pi(p,q|\widetilde{T})| \leq (1+\epsilon)|\Pi(p,q|T)|$. Following the similar proof in Theorem 4.2, we know that for all pairs of points s' and t' on any faces of T, we have $(1-\epsilon)|\Pi(s',t'|T)| \leq |\Pi(s',t'|\widetilde{T})| \leq (1+\epsilon)|\Pi(s',t'|T)|$. This is because if the distances between any pair of points on the faces near the deleted vertex do not change a lot, then the distances between any pair of points on the faces far away from the deleted vertex cannot change a lot. Since s and t can be any vertices of T, and s' and t' can be any points on any faces of T, we obtain that for all pairs of vertices s and t on T, algorithm *TIN-SurSimQ-Adapt(HM)* has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$. □

Theorem D.5. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm TIN-NetSimQ-Adapt(HM) are* $O(n^2\log n)$, $O(n)$, $O(n\log n)$ *and* $O(n\log n)$, *respectively. Given a height map M, it first constructs a TIN T using M, and then returns a simplified TIN* $\widetilde{T}$ *of T such that* $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$ *for all pairs of vertices s and t on T, where* $\Pi_N(s,t|\widetilde{T})$ *is the shortest network path between s and t passing on* $\widetilde{T}$.

Proof. Firstly, we prove the simplification time. It first needs to construct the *TIN* using the height map in $O(n)$ time. Then, in each vertex removal iteration, it uses algorithm *TIN-DijQ* [29] in $O(n\log n)$ time to check the distances between any pair of vertices that are neighbours of the deleted vertex on the original *TIN* and the simplified *TIN*. Since there are only $O(1)$ vertices that are neighbours of the deleted vertex, this step needs $O(n\log n)$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(n^2\log n)$. In general, the total simplification time is $O(n + n^2\log n) = O(n^2\log n)$.

Secondly, we prove the output size. Although this algorithm could simplify a *TIN*, our experimental results show the simplified *TIN* still has $O(n)$ vertices. Thus, the output size is $O(n)$.

Thirdly, we prove the shortest path query time. Since there are $O(n)$ vertices on $\widetilde{T}$, we use algorithm *TIN-DijQ* [29] in $O(n\log n)$ time for the shortest path query. Thus, the shortest path query time is $O(n\log n)$.

Fourthly, we prove the *kNN* and range query time. Since we just need to use algorithm *TIN-DijQ* [29] (i.e., a single-source-all-destination algorithm) once for both the *kNN* and range query, the *kNN* and range query time is $O(n\log n)$.

Finally, we prove that for all pairs of vertices s and t on T, algorithm *TIN-NetSimQ-Adapt(HM)* has $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$. In each vertex removal iteration, it performs a check between all pairs of vertices u and v (adjacent to the deleted vertex) on T whether $(1-\epsilon)|\Pi_N(u,v|T)| \leq |\Pi_N(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(u,v|T)|$. If the distances between any pair of vertices adjacent to the deleted vertex do not change a lot, then the distances between any pair of vertices far away from the deleted vertex cannot change a lot. So we obtain that for all pairs of vertices s and t on T, algorithm *TIN-NetSimQ-Adapt(HM)* has $(1-\epsilon)|\Pi_N(s,t|T)| \leq |\Pi_N(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi_N(s,t|T)|$. The detailed proof can be found in study [29]. □

Theorem D.6. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm PC-MesSimQ-Adapt(HM) are* $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\log n})$, $O(\frac{n}{\log n}\log\frac{n}{\log n})$ *and* $O(\frac{n}{\log n}\log\frac{n}{\log n})$, *respectively. Given a height map M, it first constructs a point cloud C using M, and then returns a simplified point cloud* $\widetilde{C}$ *of C such that* $(1-\epsilon)|\Pi(s,t|C)| \leq |\Pi(s,t|\widetilde{C})| \leq (1+\epsilon)|\Pi(s,t|C)|$ *for all pairs of points s and t on C, where* $\Pi(s,t|\widetilde{C})$ *is the shortest surface path between s and t passing on* $\widetilde{C}$.

Proof. We prove the simplification time. It first needs to construct the point cloud using the height map in $O(n)$ time. Then, since the point cloud and the height map has the same conceptual graph, its simplification process is the same as algorithm *HM-MemSimQ*. Thus, the simplification time is $O(n + n\sqrt[3]{n}\log n) = O(n\sqrt[3]{n}\log n)$.

The output size, shortest path query time, *kNN* and range query time and error guarantee of algorithm *PC-MesSimQ-Adapt(HM)* are the same as algorithm *HM-MemSimQ*. □

Theorem D.7. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm HM-MemSimQ-LQT1 are* $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\log n})$, $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$ *and* $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$, *respectively. Given a height map M, it returns an* ϵ*-approximate simplified height map* $\widetilde{M}$ *of M.*

Proof. Firstly, we prove the shortest path query time. Since it needs to use Dijkstra's algorithm with each pixel in $\widetilde{N}(C^{-1}(s))$ or $\widetilde{N}(C^{-1}(t))$ as a source to compute inter-path, and the size of $\widetilde{N}(C^{-1}(s))$ or $\widetilde{N}(C^{-1}(t))$ is $O(n)$, so its shortest path query time is $O(n)$ times the shortest path query time of algorithm *HM-MemSimQ*. Thus, the shortest path query time is $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$.

Secondly, we prove the *kNN* and range query time. Since we just need to use the shortest path query phase of algorithm *HM-MemSimQ-LQT1* once for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$.

The simplification time, output size and error guarantee of algorithm *HM-MemSimQ-LQT1* are the same as algorithm *HM-MemSimQ*. □

Theorem D.8. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm HM-MemSimQ-LQT2 are* $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\log n})$, $O(\frac{n}{\log n}\log\frac{n}{\log n})$ *and* $O(\frac{nn'}{\log n}\log\frac{n}{\log n})$, *respectively. Given a height map M, it returns an* ϵ*-approximate simplified height map* $\widetilde{M}$ *of M.*

Proof. We prove the *kNN* and range query time. Since we need to use the shortest path query phase of algorithm *HM-MemSimQ* n' times for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{nn'}{\log n}\log\frac{n}{\log n})$.

The simplification time, output size, shortest path query time and error guarantee of algorithm *HM-MemSimQ-LQT2* are the same as algorithm *HM-MemSimQ*. □

Theorem D.9. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm HM-MemSimQ-LS are* $O(n\sqrt[3]{n}\log n)$, $O(n)$, $O(n\log n)$ *and* $O(n\log n)$, *respectively. Given a height map M, it returns an* ϵ*-approximate simplified height map* $\widetilde{M}$ *of M.*

Proof. Firstly, we prove the output size. Since it uses the naive merging technique that only merges four pixels in Section 4.2, although it could simplify a height map, our experimental results show the simplified height map still has $O(n)$ pixels. Thus, the output size is $O(n)$.

Secondly, we prove the shortest path query time. Since there are $O(n)$ pixels on $\widetilde{M}$, and we use Dijkstra's algorithm on $\widetilde{M}$ for once, the shortest path query time is $O(n\log n)$.

Thirdly, we prove the *kNN* and range path query time. Since we just need to use Dijkstra's algorithm once for both the *kNN* and range query, the *kNN* and range query time is $O(n\log n)$.

The simplification time and error guarantee of algorithm *HM-MemSimQ-LS* are the same as algorithm *HM-MemSimQ*. □

Theorem D.10. *The simplification time, output size, shortest path query time and kNN and range query time of algorithm HM-MemSimQ-LST are* $O(n^3\sqrt[3]{n}\log n)$, $O(\frac{n}{\log n})$, $O(\frac{n}{\log n}\log\frac{n}{\log n})$ *and* $O(\frac{n}{\log n}\log\frac{n}{\log n})$, *respectively. Given a height map M, it returns an* ϵ*-approximate simplified height map* $\widetilde{M}$ *of M.*

Proof. We prove the simplification time. Since it use the naive checking technique that checks whether Inequality 1 is satisfied for all pixels in Section 4.2, in each pixel merging iteration, it needs to check the distance between all pairs of pixels on $\widetilde{M}$ and M, i.e., run Dijkstra's algorithm in $O(n\log n)$ time for $O(n)$ pixels, which needs $O(n^2\log n)$ time. According to Theorem 4.2, there are total $O(\sqrt[3]{N})$ pixel merging iterations. So the total simplification time is $O(n^2\sqrt[3]{N}\log n)$.

The output size, shortest path query time, *kNN* and range query time and error guarantee of algorithm *HM-MemSimQ-LST* are the same as algorithm *HM-MemSimQ*. □

Theorem D.11. *The shortest path query time, kNN and range query time and memory consumption of algorithm TIN-UnfQ-Adapt(HM) are* $O(n^2)$, $O(n^2)$ *and* $O(n^2)$, *respectively. Compared with* $\Pi(s,t|T)$, *it returns the exact shortest surface path passing on a TIN (that is constructed by the height map). Compared with* $\Pi(s,t|M)$, *it returns the approximate shortest path passing on a height map.*

Proof. Firstly, we prove the shortest path query time. The proof of the shortest path query time of algorithm *TIN-UnfQ* is in [15, 47, 54]. But since algorithm *TIN-UnfQ-Adapt(HM)* first needs to construct the *TIN* using the height map, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n^2) = O(n^2)$.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n^2)$.

Thirdly, we prove the memory consumption. The proof of the memory consumption of algorithm *TIN-UnfQ* is in [15, 47, 54], which is similar to algorithm *TIN-UnfQ-Adapt(HM)*. Thus, the memory consumption is $O(n^2)$.

Finally, we prove the error guarantee. Compared with $\Pi(s,t|T)$, the proof that it returns the exact shortest path passing on a *TIN* is in [15, 47, 54]. Since the *TIN* is constructed by the height map, so algorithm *TIN-UnfQ-Adapt(HM)* returns the exact shortest surface path passing on a *TIN* (that is constructed by the height map). Compared with $\Pi(s,t|M)$, since we regard $\Pi(s,t|M)$ as the exact shortest path passing on the height map, algorithm *TIN-UnfQ-Adapt(HM)* returns the approximate shortest path passing on a height map. □

Theorem D.12. *The shortest path query time, kNN and range query time and memory consumption of algorithm TIN-SteQ-Adapt(HM) are* $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$,

$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$ *and* $O(n)$*, respectively. Compared with* $\Pi(s,t|T)$*, it always has* $|\Pi_{TIN\text{-}SteQ\text{-}Adapt(HM)}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ *for all pairs of vertices* s *and* t *on* T*, where* $\Pi_{TIN\text{-}SteQ\text{-}Adapt(HM)}(s,t|T)$ *is the shortest surface path of algorithm TIN-SteQ-Adapt(HM) passing on a TIN* T *(that is constructed by the height map) between* s *and* t*. Compared with* $\Pi(s,t|C)$*, it returns the approximate shortest path passing on a height map.*

Proof. Firstly, we prove the shortest path query time. The proof of the shortest path query time of algorithm *TIN-SteQ* is in [28]. Note that in Section 4.2 of [28], the shortest path query time of algorithm *TIN-SteQ-Adapt(HM)* is $O((n+n')(\log(n+n')+(\frac{l_{max}K}{l_{min}\sqrt{1-\cos\theta}})^2))$, where $n' = O(\frac{l_{max}K}{l_{min}\sqrt{1-\cos\theta}}n)$ and K is a parameter which is a positive number at least 1. By Theorem 1 of [28], we obtain that its error guarantee ϵ is equal to $\frac{1}{K-1}$. Thus, we can derive that the shortest path query time of algorithm *TIN-SteQ-Adapt(HM)* is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}})+\frac{l_{max}^2}{(\epsilon l_{min}\sqrt{1-\cos\theta})^2})$. Since for n, the first term is larger than the second term, so we obtain the shortest path query time of algorithm *TIN-SteQ-Adapt(HM)* is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$. But since algorithm *TIN-SteQ-Adapt(HM)* first needs to construct a *TIN* using the point cloud, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time of algorithm *TIN-SteQ-Adapt(HM)* is $O(n+\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}})) = O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$. In [52], it omits the constant term in the shortest path query time. After adding back these terms, the shortest path query time is the same.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$.

Thirdly, we prove the memory consumption. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory consumption is $O(n)$.

Finally, we prove the error guarantee. Compared with $\Pi(s,t|T)$, the proof of the error guarantee of algorithm *TIN-SteQ-Adapt(HM)* is in [28, 52]. Since the *TIN* is constructed by the point cloud, so algorithm *TIN-SteQ-Adapt(HM)* always has $|\Pi_{TIN\text{-}SteQ\text{-}Adapt(HM)}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for all pairs of vertices s and t on T. Compared with $\Pi(s,t|C)$, since we regard $\Pi(s,t|C)$ as the exact shortest path passing on the point cloud, algorithm *TIN-SteQ-Adapt(HM)* returns the approximate shortest path passing on a point cloud. □

Theorem D.13. *The shortest path query time, kNN and range query time and memory consumption of algorithm TIN-DijQ-Adapt(HM) are* $O(n\log n)$*,* $O(n\log n)$ *and* $O(n)$*, respectively. Compared with* $\Pi(s,t|T)$*, it always has* $|\Pi_{TIN\text{-}DijQ\text{-}Adapt(HM)}(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ *for all pairs of vertices* s *and* t *on* T*, where* $\Pi_{TIN\text{-}DijQ\text{-}Adapt(HM)}(s,t|T)$ *is the shortest network path of algorithm TIN-DijQ-Adapt(HM) passing on a TIN* T *(that is constructed by the height map) between* s *and* t*,* $\alpha = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$*. Compared with* $\Pi(s,t|C)$*, it returns the approximate shortest path passing on a height map.*

Proof. Firstly, we prove the shortest path query time. Since algorithm *TIN-DijQ* only computes the shortest network path passing on T (that is constructed by the height map), it is a Dijkstra's algorithm and there are total n vertices, the shortest path query time is $O(n\log n)$. But since algorithm *TIN-DijQ-Adapt(HM)* first needs to construct a *TIN* using the height map, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n\log n) = O(n\log n)$.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n\log n)$.

Thirdly, we prove the memory consumption. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory consumption is $O(n)$.

Finally, we prove the error guarantee. Recall that $\Pi_N(s,t|T)$ is the shortest network path passing on T (that is constructed by the height map) between s and t, so actually $\Pi_N(s,t|T)$ is the same as $\Pi_{TIN\text{-}DijQ\text{-}Adapt(HM)}(s,t|T)$. Recall that $\Pi_E(s,t|T)$ is the shortest path passing on the edges of T (where these edges belong to the faces that $\Pi(s,t|T)$ passes) between s and t. Compared with $\Pi(s,t|T)$, we know $|\Pi_E(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ (according to left hand side equation in Lemma 2 of [29]) and $|\Pi_N(s,t|T)| \leq |\Pi_E(s,t|T)|$ (since $\Pi_N(s,t|T)$ considers all the edges on T), so we have $|\Pi_{TIN\text{-}DijQ\text{-}Adapt(HM)}(s,t|T)| \leq \alpha\cdot|\Pi(s,t|T)|$ for all pairs of vertices s and t on T. Compared with $\Pi(s,t|C)$, since we regard $\Pi(s,t|C)$ as the exact shortest path passing on the height map, algorithm *TIN-DijQ-Adapt(HM)* returns the approximate shortest path passing on a height map. □

Theorem D.14. *The shortest path query time, kNN and range query time and memory consumption of algorithm PC-ConQ-Adapt(HM) are* $O(n\log n)$*,* $O(n\log n)$ *and* $O(n)$*, respectively. It returns the exact shortest path passing on a height map and a point cloud (that is constructed by the height map).*

Proof. Firstly, we prove the shortest path query time. Since algorithm *PC-ConQ* computes the shortest path passing on C (that is constructed by the height map), it is a Dijkstra's algorithm and there are total n points, the shortest path query time is $O(n\log n)$. But since algorithm *PC-ConQ-Adapt(HM)* first needs to construct a point cloud using the height map, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n\log n) = O(n\log n)$.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for both the *kNN* and range query. So, the *kNN* and range query is $O(n\log n)$.

Thirdly, we prove the memory consumption. Since it is a Dijkstra's algorithm and there are total n points on the point cloud, the memory consumption is $O(n)$.

Finally, we prove the error guarantee. Since the height map and point cloud have the same conceptual graph, its error guarantee is the same as algorithm *HM-EffQ*. □