Efficient Proximity Queries on Simplified Height Maps

Anonymous
Anonymous
Anonymous

Anonymous
Anonymous
Anonymous

ABSTRACT

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry. Although there are different representations of a 3D surface, the height map representation is one important and fundamental representation with a lot of advantages over others such as the point cloud and <u>T</u>*riangular-*<u>I</u>*rregular* <u>N</u>*etwork* (*TIN*) representations. In this paper, we study the shortest path query on the height map representation. Since finding the shortest path on the height map representation is costly, we propose to perform a simplification process, namely <u>H</u>*eight* <u>M</u>*ap* <u>*Simplify*</u>*ication Algorithm* (*HM-Simplify*), on the height map representation to reduce the size of the height map so that finding the shortest path on the simplified representation could be faster. We also propose a shortest path algorithm, namely <u>H</u>*eight* <u>M</u>*ap* <u>S</u>*hortest* <u>P</u>*ath Algorithm* (*HM-SP*), on the original/simplified height map. We give theoretical analysis on the performance of *HM-Simplify* and *HM-SP*. We also design efficient algorithms for answering *kNN* and range queries on the original/simplified height map. Our experiments show that our simplification algorithm *HM-Simplify* is up to 412 times and 7 times better than the best-known adapted simplification algorithm (which is the *TIN* simplification algorithm) in terms of the simplification time and output size (referring to the size of the simplified surface), respectively. Performing proximity queries on our simplified height map are 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 15%, 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, 37 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Performing proximity queries on a 3D surface has gained significant attention from both academic and industry [51, 58]. Academic researchers studied different types of proximity queries [25, 26, 42, 48, 51, 54, 55, 58] such as *shortest path queries* [22, 35, 36, 38, 41, 44, 45, 50–53, 56–59], *k-*<u>N</u>*earest* <u>N</u>*eighbor* (*kNN*) *queries* [25, 26, 48, 51, 54] and *range queries* [42, 49]. In industry, Google Earth [8] and Metaverse [14] employ shortest paths passing on 3D surfaces (such as Earth and virtual reality) to enhance user navigation.

SIGMOD '26, May 31 – June 5, 2026, Bengaluru India
© 2025 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Height map, point cloud and *TIN*: There are different representations of a 3D surface. The most popular representations are the *height map* representation, the *point cloud* representation [58] and the <u>T</u>*riangular-*<u>I</u>*rregular* <u>N</u>*etwork* (*TIN*) [51, 54, 55] representation. Figure 1 (a) shows a 3D surface in a 20km × 20km region in Gates of the Arctic [46] national park in the northern part of Alaska, USA. Figure 1 (b) shows the height map representation of this surface. Consider a 2D horizontal plane in this region which is partitioned into a 9 × 9 grid. Each (grid) cell is (implicitly) associated with the 2D coordinate values in this 2D plane which is defined to be the 2D coordinate values of its centering point. Each cell is also assigned with an elevation value denoting the height of the point projected from this centering point in the 2D plane on the 3D surface. Note that we could regard each cell as a 3D point (by considering its 2D coordinate elevation values). Besides, each cell has 8 neighbors (including the 4 *adjacent neighbors* and the 4 *diagonal neighbors*, as shown in blue points and yellow points at the top-left corner of Figure 1 (b), respectively). In the height map representation, the elevation value is usually in the range of [0, 255] and thus, could be represented as a gray color scale to be shown as a "pixel" color. If this value is larger, this pixel gray color is brighter. Based on the pixel values, an image with size 9 × 9 could be generated as the height map representation as shown in Figure 1 (b) if we regard each cell in the grid as a pixel in an image. With this height map representation, it is easy to know that this representation denotes the 3D surface as shown in Figure 1 (c) (which is the 3D surface in bird's eye view). Figure 1 (d) shows the point cloud representation of this surface. Each cell in the height map representation could be mapped to a 3D point where the x- and y- coordinate values of this point are the 2D coordinate values of the centering point of this cell, and the z-coordinate of this point is the elevation value of this cell. Figure 1 (e) shows the *TIN* representation of this surface. In the literature, it could also be named as a mesh representation. We could regard that in this representation, the surface is composed of a number of contiguous triangulated *faces*, where each face has three *edges* that connect at three *vertices*, and each face forms a part of the surface as shown in the figure. In practice, the *TIN* representation is converted from the point cloud representation [58] via *triangulation* [40] where all vertices of faces are the points in the point cloud representation.

1.1 Advantages of Height Map Representation

Although there are different representations of a 3D surface, the height map representation is one important and fundamental representation with many advantages over other representations such as the point cloud and *TIN* representations. In a common practice, the *TIN* representation is converted from the point cloud representation. In the following, we focus on the comparison between the height map and point cloud representations. Importantly, there are more height map datasets available in some popular 3D surface

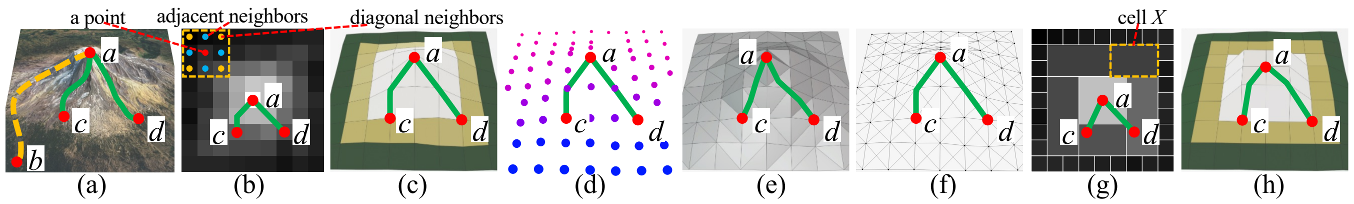


Figure 1: Paths passing on (a) a 3D surface, (b) a height map, (c) a height map in bird's eye view, (d) a point cloud, (e) a *TIN*, (f) a height map graph, (g) a simplified height map and (h) a simplified height map in bird's eye view

dataset platforms. Specifically, in OpenDEM [15], an open data platform similar to OpenStreet Map, there are 50M height map datasets but only 20M point cloud datasets. There are four reasons why there are more height map datasets.

(1) *Longer history of height map representation.* The height map representation was introduced in 1884 [2] but the point cloud representation was introduced in 1960 [11]. Due to the earlier adoption of height map representation, more height map datasets are available.

(2) *Lower cost of obtaining a height map dataset.* Obtaining the height map dataset is much cheaper than obtaining the point cloud dataset. The height map dataset could be obtained from either *optical* images of cost USD \$25 [19] captured by an *optical* satellite or *radar* images of cost USD \$3,300 [7] captured by a *radar* satellite. But, the point cloud dataset could be obtained only from *radar* images captured by a *radar* satellite (if we do not involve any conversion operation from one representation to another representation). The substantial cost difference between optical images and radar images is probably due to the substantial cost difference of launching satellites with the launching costs of an optical satellite and a radar satellite as USD \$0.4 billion [12] and USD \$1.5 billion [1], respectively. The low-cost height map dataset could also explain the popularity of this dataset nowadays.

(3) *More region coverage of the height map datasets.* Since optical and radar satellites cover 100% [3] and 80% [18] of Earth's land area, respectively, height map datasets cover more regions compared with point cloud datasets. For example, high-latitude regions (e.g., the northern part of Alaska, USA) are regions covered by height map datasets but not point cloud datasets [20].

(4) *Large conversion time among different representations.* According to our experiment, converting the height map datasets to the point cloud datasets [32, 43] in only the region not covered by radar satellites takes around 21 years[1], which could not be tolerable.

1.2 Our Focus

1.2.1 ***Height map shortest path query.***

In this paper, we study the shortest path query on the height map representation. Given a source point s and a destination point t on the height map, we want to find the shortest path from s to t. There are two issues.

(1) *No height map shortest path query algorithm.* To the best of our knowledge, there is no existing study finding the shortest path *directly* on the height map representation. Most (if not all) algorithms [32, 43] have to convert the height map representation to another representation (e.g., point cloud and *TIN*) and then perform the shortest path algorithm on the converted representation. In this paper, we propose the concept of "*height map graph*" which is constructed from the height map representation as shown in Figure 1 (f). Roughly speaking, each cell in the height map representation corresponds to a 3D point in a 3D space. For each cell c, we construct a vertex v_c denoting this cell in the graph and, for each (adjacent/diagonal) neighbor of c, says c', create an edge between v_c and $v_{c'}$ where the weight of this edge is set to the Euclidean distance between the coordinate values of the 3D points denoting these 2 cells, namely c and c'. Based on this graph, we could find the shortest path easily by Dijkstra's algorithm [27]. Our experiments show that computing the shortest path directly passing on a height map with 0.5M cells needs 3s, but computing the shortest path passing on a point cloud (green paths in Figure 1 (d)) generated from this height map [58] needs 6s since it involves data conversion. Besides, computing the shortest surface path passing on a *TIN* (green paths in Figure 1 (e)) generated from this height map [22, 52, 59] needs 280s $\approx$ 4.6 min, since the structure of a height map is simpler.

(2) *Further improvement of the proposed height map shortest path query algorithm in other proximity queries.* It is costly to compute the shortest path on the height map representation in some proximity queries (e.g., *kNN* and range queries). Our experiments show that given a height map with 50k cells and 10k possible query objects, answering a *kNN* query for these query objects needs 7,590s $\approx$ 2.1 hours, which is quite long. To overcome this issue, in this paper, we propose to perform a *simplification* process on the height map representation.

1.2.2 ***Height map simplification.***

In this paper, we also study how to simplify the height map representation. It is observed that some "nearby" cells have similar elevation values, which could be considered as "redundant" information. If we could merge these cells into one cell, then the total number of cells will be reduced and thus, Dijkstra's algorithm on this simplified height map with fewer cells will be faster. Figure 1 (g) shows a simplified height map representation of the same surface, where some cells are merged into one single cell. For example, cell X at the top right part of the figure is a cell merged from 6 cells. The elevation value of cell X is the average elevation value of these 6 cells. It is easy to verify that this simplified height map denotes the 3D surface in bird's eye view as shown in Figure 1 (h). Consider point a as a source and

[1]Since the total Earth's land area is 149M km^2 [6], the total areas covered by optical satellites and radar satellites are 149M km^2 (with 100% coverage) and 119M km^2 (with 80% coverage), respectively. In our experiment, converting a height map dataset in a region of 1 km^2 to a point cloud dataset (which runs in a linear time) takes 42s. Based on this, the total conversion time of the region of the *large* uncovered region by the radar satellites (which is 149M km^2 − 119M km^2 = 30M km^2) is 42s/km$^2\times$ 30M km^2= 1.26 $\times 10^9$s which is equivalent to 21 years.

point c as a destination. Note that there is a relative error called the *distance error ratio* of the shortest distance from point a to point c on the simplified height map (Figure 1 (g)) called the *estimated shortest distance* compared with the shortest distance from point a to point c on the original height map (Figure 1 (b)) called the *original shortest distance*. It is obvious that we want a small distance error ratio even though we perform a simplification process.

Given a user error parameter $\epsilon \in [0, 1]$, we study how to simplify the height map representation such that for any source point and any destination point on the original height map representation, its distance error ratio is at most ϵ. There are two challenges.

(1) *How to perform a simplification process on the height map representation.* To the best of our knowledge, there is no existing paper studying the simplification process on the height map representation. The only closely related work for simplification is the simplification algorithms on the *TIN* representation [26, 33, 36, 38]. We could adapt these algorithms originally designed on the *TIN* representation for the height map representation by first converting the height map to the *TIN* and then performing the original simplification algorithm on this *TIN*. It is worth mentioning that *TIN* simplification algorithms are very slow since the *TIN* simplification process involves an expensive operation of *re-triangulation* [40]. In our experiments, the best-known adapted algorithm [34, 36] takes 103,000s ≈ 1.2 days to simplify a height map with 50k cells.

(2) *How to perform a shortest path query on the simplified map.* In the original height map as shown in Figure 1 (b), it is clear to understand the neighborhood of each cell. However, in the simplified height map as shown in Figure 1 (g), since some cells are merged and one "merged" cell could be adjacent to many different cells, it is needed to define clearly the neighborhood of each (merged/non-merged) cell in the simplified height map representation.

1.3 Contribution and Organization

We summarize our contributions as follows.

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

(2) We propose an ϵ-approximate simplification algorithm called *$\underline{H}$eight $\underline{M}$ap $\underline{Simplify}$ication Algorithm* (*HM-Simplify*) to simplify the height map. We also propose a shortest path algorithm called *$\underline{H}$eight $\underline{M}$ap $\underline{S}$hortest $\underline{P}$ath Algorithm* (*HM-SP*) on the original/simplified height map. For any source point s and any destination point t on the original height map, the distance error ratio of the estimated shortest distance compared with the original shortest distance is at most ϵ. Furthermore, we design efficient algorithms for answering *kNN* and range queries on the original/simplified height map.

(3) We give theoretical analysis on the performance of our simplification algorithm *HM-Simplify* concerning the simplification time, output size (referring to the size of the simplified height map) and error guarantee, and our shortest path algorithm *HM-SP* concerning the shortest path query time, memory usage and error guarantee.

(4) Algorithm *HM-Simplify* outperforms the best-known adapted simplification algorithm (which is the *TIN* simplification algorithm) [34, 36] concerning the simplification time and output size. The proximity query time on the simplified height map also performs much better than the best-known adapted algorithm on a point cloud [58] and a simplified *TIN* [36, 51]. Our experiments show that given a height map with 50k cells, the simplification time and output size are 250s ≈ 4.6 min and 0.07MB for algorithm *HM-Simplify*, but are 103,000s ≈ 1.2 days and 0.5MB for the best-known adapted simplification algorithm [34, 36]. The *kNN* and range query time of 10k objects on the simplified height map are both 50s for *HM-SP*, but are 7,630s ≈ 2.1 hours for the best-known adapted algorithm on a point cloud representation [58], and 67,000s ≈ 18.6 hours for the best-known adapted algorithm on a simplified *TIN* representation [36, 51].

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

2 PROBLEM DEFINITION

2.1 Notation and Definitions

2.1.1 ***Height map.*** Consider a height map $M = (C, N(\cdot))$ on a 2D horizontal plane containing a set C of *cells*, and a *neighbor cells* table $N(\cdot)$ (which is a *hash table* [24]). In M, each cell $c \in C$ has two 2D coordinate values (representing its x- and y-coordinate values) and a grayscale color integer value (representing its *elevation value* with range $[0, 255]$), denoted as $c.x$, $c.y$ and $c.z$, respectively. When we refer to the position/coordinate of a cell on the horizontal plane, we use the position/coordinate of the point at the center of this cell. For each cell $c \in C$, $N(c)$ returns the *neighbor cells* of c in $O(1)$ time, and $N(c)$ is initialized to be the nearest upper, lower, leftward, rightward, upper-left, upper-right, lower-left and lower-right cells of c on M (corresponding to all adjacent and diagonal neighbors). Based on M, let n be the number of cells of M. Figure 2 (a) shows a height map M with 9 cells. Consider cell c containing point p. 6 orange points and 2 red points form $N(c)$.

We define the *height map graph* of M, says G, as follows. Let $G.V$ and $G.E$ be G's vertices and edges, respectively. For each cell $c \in C$, we create a vertex v_c whose x-, y- and z-coordinate values are defined to $c.x$, $c.y$ and $c.z$ of c, respectively. All vertices created form $G.V$. For each $c \in C$ and each $c' \in N(c)$, we create an edge between vertex v_c and vertex $v_{c'}$, where v_c and $v_{c'}$ correspond to cells c and c', respectively, and its weight is defined to be the Euclidean distance between these two vertices. All edges created form $G.E$. The graphs in Figures 2 (a) and (b) are the height map graph of M on the 2D horizontal plane and in a 3D space, respectively. Given a pair of points s and t on M, let $\Pi(s, t|M)$ be the *shortest path* between them passing on (G of) M. Let $|\cdot|$ be a path's length (e.g., $\Pi(s, t|M)$'s length is denoted by $|\Pi(s, t|M)|$). Figures 2 (a) and (b) show $\Pi(s, t|M)$ in green line.

2.1.2 ***Simplified height map.*** Given a height map M, we can obtain a simplified height map $\widetilde{M} = (\widetilde{C}, \widetilde{N}(\cdot))$ by merging some adjacent cells (deleting these cells and adding a new "larger" cell covering these cells for replacement) in M, where $\widetilde{C}$ and $\widetilde{N}(\cdot)$ are initialized as C and $N(\cdot)$, and are updated during simplification. Given a user error parameter $\epsilon \in [0, 1]$, we can perform the merging operation if, after merging, all pairs of points on M have distance

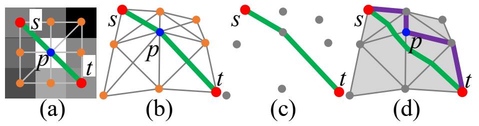


Figure 2: Paths passing on (a) a height map, (b) a height map graph, (c) a point cloud and (d) a *TIN*

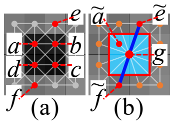


Figure 3: Cell merging

error ratios at most ϵ. Figures 3 (a) and (b) are original and simplified height maps M and $\widetilde{M}$, where the cell denoted by the pale blue color region in $\widetilde{M}$ is the "larger" cell merged from 4 cells in M.

In the following, we refer a cell in M that is deleted from (resp. remaining in) $\widetilde{M}$ during simplification as a *deleted* (resp. *remaining*) cell. We also refer a cell in $\widetilde{M}$ that covers some adjacent deleted and/or previously added cells as an *added* cell. We say that these adjacent deleted cells *belong* to the added cell. A *property of a deleted cell* is that each deleted cell only belongs to one added cell. In Figures 3 (a) and (b), we merge cells a, b, c and d to cell g, 12 orange and red points (around g) form all cells in $\widetilde{N}(g)$, $\{a, b, c, d\}$ are deleted cells, all other cells in C except $\{a, b, c, d\}$ are remaining cells, g is an added cell, and $\{a, b, c, d\}$ belong to g. The coordinate and elevation values of the added cell are weighted average values of those of the adjacent deleted cells (if these adjacent deleted cells contain a previously added cell, the weight is the number of cells in M belonging to this previously added cell; otherwise, the weight is 1). In Figures 3 (b), we use the coordinate and elevation values of a, b, c, d with weight equal to 1 to calculate the corresponding values of g. If we keep merging g with other cells, the weight of g is 4, since the number of cells in M belonging to g is 4. We denote a set of remaining cells and added cells as C_{rema} and C_{add}, so $\widetilde{C} = C_{rema} \cup C_{add}$. A set of deleted cells is denoted as $C - C_{rema}$.

Given a cell $c \in M$, we define the *estimated cell* of c (on $\widetilde{M}$), denoted by $\widetilde{c}$, such that $\widetilde{c}.x = c.x$, $\widetilde{c}.y = c.y$, $\widetilde{c}.z$ is the elevation value of the added cell such that c belongs to (if c is a deleted cell), or $\widetilde{c} = c$ (if c is a remaining cell). In Figure 3 (b), since a is a deleted cell, a belongs to g, we have $\widetilde{a}.x = a.x$, $\widetilde{a}.y = a.y$ and $\widetilde{a}.z = g.z$.

Similar to G, let $\widetilde{G}$ be the simplified height map graph of $\widetilde{M}$, and let $\widetilde{G}.V$ and $\widetilde{G}.E$ be $\widetilde{G}$'s vertices and edges, respectively. We just need to use $\widetilde{C}$ and $\widetilde{N}(\cdot)$ to substitute C 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 3 (a) and (b) are original and simplified height graphs on the 2D horizontal plane. Given a pair of points $\widetilde{s}$ and $\widetilde{t}$ on $\widetilde{M}$, let $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$ be the *estimated shortest path* between them passing on ($\widetilde{G}$ of) $\widetilde{M}$. Figure 3 (b) shows $\Pi(\widetilde{e}, \widetilde{f}|\widetilde{M})$ in blue line. A notation table can be found in the appendix of Table 4.

2.2 Problem

Next, we introduce a concept of the *ϵ-approximate simplified height map* (as shown in Definition 1) to describe that the simplified height map $\widetilde{M}$ guarantees that for each pair of points in the original height map M, their distance error ratio is at most ϵ.

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

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

We have the following problem.

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

The following theorem shows this problem is *NP-hard.*

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

Proof Sketch. We can transform Minimum T-Spanner Problem [21] (*NP-complete* problem) to the Height Map Simplification Problem in polynomial time, to prove that it is *NP-hard*. The detailed proof appears in the appendix. □

3 RELATED WORK

3.1 Point cloud and *TIN*

Let P be a point cloud containing the set of 3D points. The *point cloud graph* (named as the *conceptual graph* in study [58]) of P, says G', is defined as follows. Let $G'.V$ and $G'.E$ be G''s vertices and edges, respectively. All points in P form $G'.V$. For each point $p \in P$ and each neighbor point p' (which has the similar definition in height map), we create an edge between vertex v_p and vertex $v_{p'}$, where v_p and $v_{p'}$ correspond to points p and p', respectively, and its weight is defined to be the Euclidean distance between these two vertices. All edges created form $G'.E$. The height map graph and point cloud graph are the same. Given a pair of points s and t on P, let $\Pi(s, t|P)$ be the *shortest path* between them passing on (G of) P. Let T be a *TIN* triangulated [29, 40, 62] 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* [36] (passing on faces of T) and *shortest network path* [36] (passing on edges of T) between them, whose distances are called the *shortest surface* and *network distance*, respectively. Let θ be the smallest interior angle of a triangle of T. Figure 2 (c) shows a point cloud converted from M with $\Pi(s, t|P)$ in green line, and Figure 2 (d) shows a *TIN* converted from M with $\Pi(s, t|T)$ in green line and $\Pi_N(s, t|T)$ in purple line.

Given a pair of points s and t on a height map, since the height map graph is the same as the point cloud graph, we know $|\Pi(s, t|M)| = |\Pi(s, t|P)|$. According to Lemma 4.3 of study [58], 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 $|\Pi(s, t|M)| \leq |\Pi_N(s, t|T)|$.

3.2 Height Map Shortest Path Algorithms

There is no existing study focusing on directly addressing proximity queries on a height map. Existing studies [32, 43] for conducting proximity queries on a height map are slow. Given a height map, they first convert it to a point cloud using 2D coordinate and elevation values of the height map's cells, or convert the point cloud to a *TIN* by triangulating the point cloud's points in $O(n)$ time, and then compute the shortest path passing on the point cloud or *TIN*. Next, we describe (1) *point cloud shortest path* algorithm [58] (on a point cloud), (2) *TIN shortest surface path* algorithm [22, 35, 37, 41, 52, 53, 57, 59] (on a *TIN*), and (3) *TIN shortest network path* algorithm [36] (on a *TIN*).

3.2.1 ***Point cloud shortest path algorithm.***

The best-known exact point cloud shortest path algorithm called <u>P</u>*oint* <u>C</u>*loud* <u>S</u>*hortest* <u>P</u>*ath Algorithm* (*PC-SP*) [58] uses Dijkstra's algorithm on the point cloud graph to compute the path in $O(n \log n)$ time.

3.2.2 ***TIN shortest surface path algorithms.***

(1) *Exact algorithms*: Two studies use continuous Dijkstra's algorithm [41] and checking window algorithm [53] to compute the exact path both in $O(n^2 \log n)$ time. The best-known exact *TIN* shortest surface path algorithm <u>*TIN*</u> <u>E</u>*xact* <u>S</u>*hortest* <u>S</u>*urface* <u>P</u>*ath* (*TIN-ESSP*) [22, 52, 59] uses a line to connect the source and destination on a 2D *TIN* unfolded by the 3D *TIN*, to compute the path in $O(n^2)$ time.

(2) *Approximate algorithms*: All approximate algorithms [35, 37, 57] construct a graph with discrete Steiner points (placed on a *TIN*'s edges) and *TIN*'s vertices to compute the path. The best-known $(1+\epsilon)$-approximate *TIN* shortest surface path algorithm called <u>*TIN*</u> <u>A</u>*pproximate* <u>S</u>*hortest* <u>S</u>*urface* <u>P</u>*ath Algorithm* (*TIN-ASSP*) [35, 57] runs in $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}} \log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$ time, where l_{max}/l_{min} are the longest/shortest edge's length of the *TIN*, respectively.

3.2.3 ***TIN shortest network path algorithm.***

The shortest network path on *TIN* is the shortest surface path restricted on passing along edges of the *TIN* model without traversing the faces of the *TIN* model, resulting in an approximate path. The best-known approximate *TIN* shortest network path algorithm called <u>*TIN*</u> <u>S</u>*hortest* <u>N</u>*etwork* <u>P</u>*ath Algorithm* (*TIN-SNP*) [36] operates in $O(n \log n)$ time.

Adaptions: Given a <u>H</u>*eight* <u>M</u>*ap*, we adapt these four best-known point cloud or *TIN* shortest path algorithms to be algorithms (1) *PC-SP-Adapt(HM)* [58], (2) *TIN-ESSP-Adapt(HM)* [22, 52, 59], (3) *TIN-ASSP-Adapt(HM)* [35, 57] and (4) *TIN-SNP-Adapt(HM)* [36], by converting the height map to a point cloud or *TIN*, and then computing the shortest path passing on the point cloud or *TIN*.

Drawback: All existing algorithms are very slow, even if we pre-convert the given height map to a point cloud or *TIN*. Our experiments show that algorithm *PC-SP-Adapt(HM)* first needs to convert a height map with 50k cells to a point cloud 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, algorithms *TIN-ESSP-Adapt(HM)*, *TIN-ASSP-Adapt(HM)* and *TIN-SNP-Adapt(HM)* first need to convert the height map to a *TIN* 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.

3.3 Height Map Simplification Algorithms

There is no existing study focusing on simplifying a height map. The only closely related work for simplification is the simplification algorithm on the *TIN* [26, 33, 36, 38]. They iteratively remove a vertex v in a *TIN* and use *triangulation* [40] to form new faces among the adjacent vertices of v. Specifically, <u>*TIN*</u> *shortest* <u>N</u>*etwork distance* <u>*Simplification*</u> *Algorithm* (*TIN-NSimplify*) [36] is the most efficient one such that for all pairs of vertices on the original *TIN*, their *network* distance error ratios (i.e., similar to the distance error ratio defined for original/simplified height map) are at most ϵ. By using the shortest *surface* distances in this algorithm, we obtain the best-known simplification algorithm called <u>*TIN*</u> *shortest* <u>S</u>*urface distance* <u>*Simplification*</u> *Algorithm* (*TIN-SSimplify*) [34, 36].

Adaptions: Given a <u>H</u>*eight* <u>M</u>*ap*, we adapt these two algorithms to be algorithms *TIN-NSimplify-Adapt(HM)* [36] and *TIN-SSimplify-Adapt(HM)* [34, 36], by converting the height map to a *TIN*, and then applying the corresponding algorithms for *TIN* simplification.

3.3.1 ***Algorithm TIN-NSimplify-Adapt(HM).***

After converting the height map to a *TIN*, each *TIN* simplification iteration checks whether the shortest network distance error ratios for all pairs of *adjacent* vertices of the removed vertex v (instead of *all* pairs of vertices) on the original *TIN* are at most ϵ. Its simplification time and output size are $O(n^2 \log n)$ and $O(n)$, respectively.

3.3.2 ***Algorithm TIN-SSimplify-Adapt(HM).***

After converting the height map to a *TIN*, each *TIN* simplification iteration checks whether the shortest surface distance error ratios (i.e., similar to the network distance error ratio) for *arbitrary* pairs of points[2] on the *adjacent* faces of v (instead of *all* pairs of vertices) on the original *TIN* are at most ϵ. We further simplify it by placing Steiner points on the *adjacent* faces of v (using the technique in study [34] for any points-to-any points *TIN* shortest surface path query), and check distances related to all pairs of Steiner points. Its simplification time and output size are $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$ and $O(n)$, respectively.

Drawbacks: (1) *Large simplification time*: They involves an expensive operation of re-triangulation during *TIN* simplification, resulting in a large simplification time. (2) *Large output size*: They do not have optimization techniques during *TIN* simplification, so their simplified *TIN*s have a large size, resulting in large shortest path query time on their simplified *TIN*s. Our experiments show that for a height map with 50k cells, the simplification time of algorithms *TIN-NSimplify-Adapt(HM)*, *TIN-SSimplify-Adapt(HM)* and *HM-Simplify* 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

4.1.1 ***Two phases.***

There are two phases for our framework.

(1) **Simplification phase using algorithm *HM-Simplify***: In Figure 4, given a height map M, we generate a simplified height map $\widetilde{M}$ (M is deleted). Specifically, in Figures 5 (a) - (f), given $M = (C, N(\cdot))$, we iteratively merge some adjacent cells to obtain $\widetilde{M} = (C_{rema} \cup C_{add}, \widetilde{N}(\cdot))$, whenever $\widetilde{M}$ is an ϵ-approximate simplified height map of M.

(2) **Shortest path query phase using algorithm *HM-SP***: In Figure 4 and Figure 5 (g), given $\widetilde{M}$, a pair of points s and t on (cells c_s and c_t of) M, we first calculate s and t's estimated points $\widetilde{s}$ and $\widetilde{t}$ on (cells $\widetilde{c_s}$ and $\widetilde{c_t}$ of) $\widetilde{M}$, and then use Dijkstra's algorithm [27] on $\widetilde{M}$ to compute $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})$.

4.1.2 ***Two components.***

The above two phases involve two tables, they are used for storing added and deleted cell information. We introduce them as follows.

[2]Given a pair of vertices that are not adjacent to v, the shortest surface path between them may pass on the adjacent faces of v but not on the adjacent vertices of v. So, only checking the shortest surface distance between all pairs of *adjacent* vertices of v is not sufficient.

Simplification phase	Height map M ➡ $\boxed{\textit{Simplification}}$ ➡ Simplified height map $\widetilde{M}$ (M is deleted), containing table $O(\cdot)$ and belong table $O^{-1}(\cdot)$
Shortest path query phase	$\widetilde{M}$, a pair of points s and t on M, $O(\cdot)$, and $O^{-1}(\cdot)$ ➡ $\boxed{\textit{Point estimation}}$ ➡ Estimated $\tilde{s}$ and $\tilde{t}$ on $\widetilde{M}$ ➡ $\boxed{\textit{Path querying}}$ ➡ The shortest path between $\tilde{s}$ and $\tilde{t}$ passing on $\widetilde{M}$

Figure 4: Overview of algorithm *HM-Simplify* and *HM-SP*

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

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

4.2 Key Idea

4.2.1 ***Novel memory saving technique for algorithm HM-Simplify.*** It is designed to save memory (i.e., reduce the output size, to further reduce the shortest path query time of algorithm *HM-SP* on $\widetilde{M}$) by considerably reducing the number of cells in $\widetilde{M}$ using a novel cell merging technique with two merging types:

(1) **Merge four cells**: We start by choosing four adjacent non-merged and non-boundary cells in a square such that the variances of their elevation values are the smallest, where a non-boundary cell means it does not lie on the boundary of M. In Figure 5 (b), we merge $\{a, b, c, d\}$ to cell e, and obtain $\widetilde{M}$. If $\widetilde{M}$ is an ϵ-approximate simplified height map of M, we confirm this merging and process to the next merging type. If not, we terminate the algorithm.

(2) **Merge added cells**: Given an added cell c from the previous merge, we merge c with its neighbor cells, i.e., we enlarge c by expanding its non-boundary neighbor cells in left, right, top and/or bottom directions to reduce the number of cells in $\widetilde{M}$, where expanding left (resp. right) covers neighbors cells with x-coordinate value smaller (resp. larger) than c, and expanding top (resp. bottom) covers neighbors cells with y-coordinate value smaller (resp. larger) than c. Let <u>Dir</u>ection, 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 c to cover its neighbor cells 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 that we cover c's neighbors cells with x-coordinate value larger and smaller than c, and y-coordinate value smaller than c.

In Figure 5 (c), we merge e with $\{f, g, \dots\}$ to form cell h, i.e., we enlarge e by expanding into all four directions, i.e, (L, R, T, B), and obtain $\widetilde{M}$. If $\widetilde{M}$ is an ϵ-approximate simplified height map of M, we confirm this merging and repeat this process. If not, we go back to the *four cells merging* type by selecting four new cells. In Figure 5 (d), we merge n with $\{l, m, \dots\}$ to form an added cell shown in the dark blue (dashed line) frame, i.e., we still enlarge n by expanding into all four directions. For one of n's neighbor cells, i.e., m, we cover it as a whole to reduce the number of cells in $\widetilde{M}$. But, this results in four pink deleted cells belonging to both h and the newly added cell, violating the property of the deleted cell in Section 2.1.2 that each deleted cell only belongs to one added cell. Roughly speaking, we do not want that the newly merged cell (e.g., the potential added cell shown in the dark blue (dashed line) frame) overlaps with any merged cell (e.g., cell h). So, we enlarge n by expanding into directions using other elements in Dir until we successfully enlarge n. In Figure 5 (e), we merge n with $\{l, \dots\}$ to form cell 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 $\widetilde{M}$ is an ϵ-approximate simplified height map of M, we confirm this merging and repeat this process. If not, we go back to the *four cells merging* type.

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

(1) **Efficient height map shortest path query**: When checking whether $\widetilde{M}$ is an ϵ-approximate simplified height map of M, we use efficient algorithm *HM-SP*.

(2) **Efficient ϵ-approximate simplified height map checking**: Checking whether $\widetilde{M}$ is an ϵ-approximate simplified height map of M involves checking whether Inequality 1 is satisfied for *all* points on M, this naive method is time-consuming. Instead, we can check distances only related to the *neighbor* cells of the newly added cells in each iteration.

4.3 Simplification Phase

We illustrate the simplification phase using algorithm *HM-Simplify* in Algorithms 1 and 2. Algorithm 2 is a sub-algorithm used for 2 times in Algorithm 1.

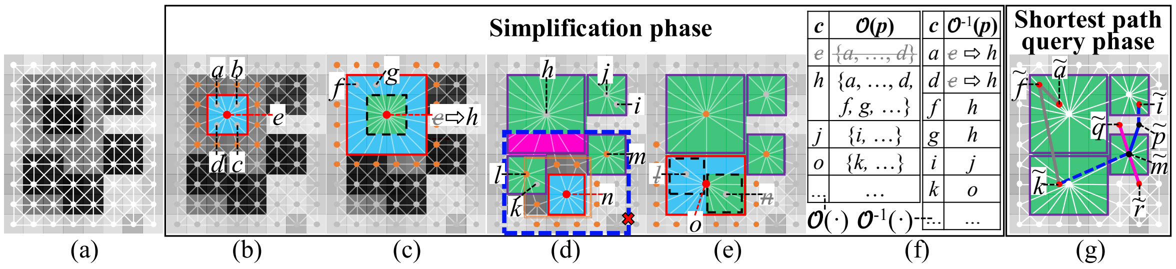
4.3.1 ***Detail and example for Algorithm 1.*** The following shows Algorithm 1 with an example. In each simplification iteration, let $\widehat{C} = \{p_1, p_2, \dots\}$ be a set of adjacent cells that we need to merge, and c_{add} be an added cell formed by merging cells in $\widehat{C}$.

(1) *Merge four cells* (lines 2-4): In Figures 5 (a) and (b), we can merge cells in $\widehat{C} = \{a, b, c, d\}$ (lines 2-3), suppose that the result of *UpdateCheck* is *True* (line 4), we obtain $c_{add} = e$ and $\widetilde{M}$.

(2) *Merge added cells* (lines 5-9). In Figure 5 (c), $c_{add} = e$, we can merge cells in $\widehat{C} = \{e\} \cup \{f, g, \dots\} = \{e, f, g, \dots\}$, i.e., we can enlarge e by expanding it to cover its non-boundary neighbor cells 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 $c_{add} = h$ and $\widetilde{M}$. 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 cells iteration* (lines 2-4): In Figure 5 (d), we use a similar process to obtain new cells j, l, m and $\widetilde{M}$. Suppose that we cannot merge added cells for j, l, m (lines 5-9), we go back to line 2. Then, we use a similar process to obtain a new cell n and $\widetilde{M}$.

(4) *Merge added cells iteration* (lines 5-9): In Figure 5 (d), $c_{add} = n$, we merge cells in $\widehat{C} = \{n\} \cup \{l, m, \dots\} = \{l, m, n, \dots\}$, i.e., we



c	$\mathcal{O}(p)$
~~e~~	~~$\{a, \ldots, d\}$~~
h	$\{a, \ldots, d, f, g, \ldots\}$
j	$\{i, \ldots\}$
o	$\{k, \ldots\}$
…	…

c	$\mathcal{O}^{-1}(p)$
a	~~e~~ ⇨ h
d	~~e~~ ⇨ h
f	h
g	h
i	j
k	o
…	…

Figure 5: Details of algorithm *HM-Simplify* and *HM-SP*

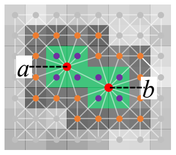


Figure 6: Distance checking

Algorithm 1 *HM-Simplify* (M)

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

enlarge n by expanding it to cover its neighbor neighbor cells in the directions using the 1-st element (L, R, T, B) in *Dir* (lines 5-6). We get the newly added cell with a blue frame. But, there exist four pink deleted cells belonging to both h and the newly added cell, this violates the property of the deleted cell, 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 5 (e), $c_{add} = n$, we can merge cells in $\widehat{C} = \{n\} \cup \{l, \ldots\} = \{n, l, \ldots\}$, 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 $c_{add} = o$ and $\widetilde{M}$.

4.3.2 ***Detail and example for Algorithm 2***.

The following shows Algorithm 2 with an example. We use Figures 5 (b) and (c) for illustration, since Figures 5 (d) and (e) are similar.

(1) *Update* $O'(\cdot)$ *and* $O^{-1\prime}(\cdot)$ (lines 3-9): In Figure 5 (b), $c_{add} = e$ and $\widehat{C} = \{a, b, c, d\}$, since all cells in $\widehat{C}$ are in C_{rema}, we have $O'(e) = \{a, b, c, d\}$, $O^{-1\prime}(a) = e, \ldots, O^{-1\prime}(d) = e$. In Figure 5 (c), $c_{add} = h$, $\widehat{C} = \{e, f, g, \ldots\}$ and $C_{add} = \{e\}$ (in lines 3-9, we have not updated C_{add} yet), since for cells in $\widehat{C}$, e is in C_{add} and other cells are in C_{rema}, we have $O'(h) = \{a, \ldots, d, f, g, \ldots\}$, $O^{-1\prime}(a) = h, \ldots, O^{-1\prime}(d) = h$, $O^{-1\prime}(f) = h$, $O^{-1\prime}(g) = h, \ldots$, and remove $O'(e)$.

(2) *Update neighbor cells* (lines 10-13): In Figures 5 (b) and (c), for e and h, we update their neighbor cells to be the cells represented in orange points; for these neighbor cells, we also update e and h to be their neighbor cells.

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

```
Input: $C_{rema}$, $C_{add}$, $\widetilde{N}(\cdot)$, $\widehat{C}$, $c_{add}$, $O(\cdot)$ and $O^{-1}(\cdot)$
Output: updated $C_{rema}$, $C_{add}$, $\widetilde{N}(\cdot)$, $O(\cdot)$, $O^{-1}(\cdot)$, and whether the updated height map is an $\epsilon$-approximate simplified height map of $M$
1: $C'_{rema} \leftarrow C_{rema}$, $C'_{add} \leftarrow C_{add}$, $\widetilde{N}'(\cdot) \leftarrow \widetilde{N}(\cdot)$, $\widetilde{N}'(c_{add}) \leftarrow \emptyset$, $O'(\cdot) \leftarrow O(\cdot)$, $O^{-1\prime}(\cdot) \leftarrow O^{-1}(\cdot)$
2: merge cells in $\widehat{C}$ to form cell $c_{add}$
3: for each $c \in \widehat{C}$ do
4:     if $c \in C_{rema}$ then
5:         $O'(c_{add}) \leftarrow O'(c_{add}) \cup \{p\}$, $O^{-1\prime}(c) \leftarrow \{c_{add}\}$
6:     else if $c \in C_{add}$ then
7:         for each $c' \in O'(c)$ do
8:             $O'(c_{add}) \leftarrow O'(c_{add}) \cup \{c'\}$, $O^{-1\prime}(c') \leftarrow \{c_{add}\}$
9:         $O'(\cdot) \leftarrow O'(\cdot) - \{O'(c)\}$
10: for each $c \in \widehat{C}$ do
11:     for each $c' \in N(c)$ such that $c' \notin \widehat{C}$ do
12:         $\widetilde{N}'(c_{add}) \leftarrow \widetilde{N}'(c_{add}) \cup \{c'\}$, $\widetilde{N}'(c') \leftarrow \widetilde{N}'(c') - c \cup \{c_{add}\}$
13: clear $\widetilde{N}'(c)$ for each $c \in \widehat{C}$
14: for each $c \in \widehat{C}$ do
15:     if $c \in C_{rema}$ then
16:         $C'_{rema} \leftarrow C'_{rema} - \{p\}$
17:     else if $c \in C_{add}$ then
18:         $C'_{add} \leftarrow C'_{add} - \{p\}$
19: $C'_{add} \leftarrow C'_{add} \cup \{c_{add}\}$
20: if $\widetilde{M}' = (C'_{rema} \cup C'_{add}, \widetilde{N}'(\cdot))$ is an $\epsilon$-approximate simplified height map of $M$ then
21:     $C_{rema} \leftarrow C'_{rema}$, $C_{add} \leftarrow C'_{add}$, $\widetilde{N}(\cdot) \leftarrow \widetilde{N}'(\cdot)$, $O(\cdot) \leftarrow O'(\cdot)$, $O^{-1}(\cdot) \leftarrow O^{-1\prime}(\cdot)$
22:     return True
23: return False
```

(3) *Update* $\widetilde{M}'$ (lines 14-19): In Figure 5 (b), $\{a, b, c, d\}$ are deleted from C'_{rema} and e is added into C'_{add}, so $C'_{add} = \{e\}$. In Figure 5 (d), $\{f, g, \ldots\}$ are deleted from C'_{rema}, e is deleted from C'_{add}, and h is added into C'_{add}, so $C'_{add} = \{h\}$.

(4) *Check ϵ-approximate simplified height map* (lines 20-22): In Figures 5 (b) and (c), suppose that $\widetilde{M}'$ is an ϵ-approximate simplified height map of M, we have $\widetilde{M}$. In Figure 5 (f), we have the updated $O(\cdot)$ and $O^{-1}(\cdot)$.

4.4 Efficient ϵ-Approximate Simplified Height Map Checking

Checking whether $\widetilde{M}$ is an ϵ-approximate simplified height map of M involves many unnecessary distance checks. Before we introduce our efficient checking, we give one notation.

4.4.1 ***Notation***.

Linked added cells: Given two cells in C_{add}, they are said to be *linked* if there is an edge between two vertices (corresponding to these two cells) in the simplified height map graph. Given an added cell $c_{add} \in C_{add}$, we define a set of *linked added cells* of c_{add}, denoted by $L(c_{add})$, to be a set of added cells in C_{add} which contain c_{add} and are linked to each other. In Figure 6, suppose that $c_{add} = a$, then $L(a) = \{a, b\}$.

4.4.2 ***Detail and example.*** We then discuss our efficient checking. In Definition 1, we change "all pairs of points s and t on M" (involving more points) to points on the following three types of cells related to the *neighbor* cells of the added cell c_{add} (involving fewer points), and then use Inequality 1 for each type to efficiently check whether $\widetilde{M}$ is an ϵ-approximate simplified height map of M.

(1) **Remaining to Remaining cells (*R2R*)**: We change to "all pairs of points s and t on remaining cells in C_{rema} that are neighbor cells of each added cell in $L(c_{add})$". Figure 6 shows these points in orange.

(2) **Remaining to Deleted cells (*R2D*)**: We change to "any point s on remaining cell in C_{rema} that is a neighbor cell of each cell in $L(c_{add})$, and any point t on deleted cell in $C - C_{rema}$ that belongs to each added cell in $L(c_{add})$". Figure 6 shows these points in orange (corresponding to s) and purple (corresponding to t).

(3) **Deleted to Deleted cells (*D2D*)**: We change to "all pairs of points s and t on deleted cells in $C - C_{rema}$ that belong to each added cell in $L(c_{add})$". Figure 6 shows these points in purple.

4.5 Shortest Path Query Phase

We illustrate the shortest path query phase using algorithm *HM-SP* on the simplified height map. We give one notation first.

4.5.1 ***Notation.*** **Intra- and inter-paths**: We define two types of paths used in our algorithm *HM-SP*, namely *intra-paths* and *inter-paths*. An intra-path is a path from a point (on a deleted cell) to a point (on a remaining/added cell) and an inter-path is a path from a point (on a remaining/added cell) to a point (on a remaining/added cell). Specifically, given a point m on a deleted cell c_m and a point n on a cell $c_n \in \widetilde{N}(O^{-1}(c_m))$ that is a neighbor cell of the added cell that c_m belongs to, we call the path between them passing on M as the *intra-path*, and denote it by $\Pi_1(\widetilde{m}, \widetilde{n}|\widetilde{M})$ (whose path is expressed as $\langle \widetilde{m}, \widetilde{n} \rangle$). Given a pair of points $\widetilde{p}$ and $\widetilde{q}$ on cells $\widetilde{c_p}$ and $\widetilde{c_q}$ in $\widetilde{C}$, we call the path between them passing on $\widetilde{M}$ as the *inter-path*, and denote it by $\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{M})$. In Figure 5 (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.

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

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

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

(i) *Both cells deleted*: Firstly, there are two special cases that we return $\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M}) = \langle \widetilde{s}, \widetilde{t} \rangle$. One is that c_s and c_t belong to the different added cells c_u and c_v, where c_u and c_v are neighbor. The other one is that c_s and c_t belong to the same added cell. In Figure 5 (g), $\Pi(\widetilde{f}, \widetilde{k}|\widetilde{M}) = \langle \widetilde{f}, \widetilde{k} \rangle$ (i.e., the first case) and $\Pi(\widetilde{a}, \widetilde{f}|\widetilde{M}) = \langle \widetilde{a}, \widetilde{f} \rangle$ (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 $|\Pi(\widetilde{s}, \widetilde{t}|\widetilde{M})| = \min_{\forall \widetilde{c_p} \in \widetilde{N}(O^{-1}(c_s)), \widetilde{c_q} \in \widetilde{N}(O^{-1}(c_t))} |\Pi_1(\widetilde{s}, \widetilde{p}|\widetilde{M})| + |\Pi_2(\widetilde{p}, \widetilde{q}|\widetilde{M})| + |\Pi_1(\widetilde{q}, \widetilde{t}|\widetilde{M})|$, where $\widetilde{p}$ and $\widetilde{q}$ are a pair of points on cells $\widetilde{c_p}$ and $\widetilde{c_q}$. A naive algorithm uses Dijkstra's algorithm on $\widetilde{M}$ with each point on cell in $\widetilde{N}(O^{-1}(c_s))$ as a source to compute inter-paths. But, we propose an efficient algorithm by using Dijkstra's algorithm only once. If the number of cells in $\widetilde{N}(O^{-1}(c_s))$ is less than that of in $\widetilde{N}(O^{-1}(c_t))$, we temporarily insert intra-paths between $\widetilde{c_s}$ and each cell in $\widetilde{N}(O^{-1}(c_s))$ 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 visiting all points on cells in $\widetilde{N}(O^{-1}(c_t))$, 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 cells in $\widetilde{N}(O^{-1}(c_s))$ is larger than that of in $\widetilde{N}(O^{-1}(c_t))$, we swap s and t. In Figure 5 (g), $\Pi(\widetilde{i}, \widetilde{k}|\widetilde{M}) = \langle \widetilde{i}, \widetilde{p}, \widetilde{m}, \widetilde{k} \rangle$.

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

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

4.6 Proximity Query Algorithms

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

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

Table 1: Height map datasets

Name	$\|n\|$
Original dataset	
GunnisonForest (*GF_m*) [10]	0.5M
LaramieMount (*LM_m*) [13]	0.5M
RobinsonMount (*RM_m*) [17]	0.5M
BearHead (*BH_m*) [5, 50, 51]	0.5M
EaglePeak (*EP_m*) [5, 50, 51]	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 simplification algorithms

Algorithm	Simplification time		Output size	
TIN-SSimplify-Adapt(HM) [34, 36]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$	Large	$O(n)$	Large
TIN-NSimplify-Adapt(HM) [36]	$O(n^2 \log n)$	Medium	$O(n)$	Large
HM-Simplify **(ours)**	$\boldsymbol{O(n \sqrt[3]{n} \log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**

Table 3: Comparison of proximity query algorithms

Algorithm	Shortest path query time		Error
TIN-ESSP-Bas-Adapt(HM) [22, 52, 59]	$O(n^2)$	Large	Small
TIN-ESSP-Adv-Adapt(HM) [22, 52, 59]	$O(n^2)$	Large	Small
TIN-ASSP-Adapt(HM) [35, 57]	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}} \log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
TIN-SNP-Bas-Adapt(HM) [36]	$O(n \log n)$	Medium	Medium
TIN-SNP-Adv-Adapt(HM) [36]	$O(n \log n)$	Medium	Medium
PC-SP-Adapt(HM) [58]	$O(n \log n)$	Medium	No error
HM-SP-Bas **(ours)**	$\boldsymbol{O(n \log n)}$	**Medium**	**No error**
HM-SP-Adv **(ours)**	$\boldsymbol{O(\frac{n}{\log n} \log \frac{n}{\log n})}$	**Small**	**Small**

4.7 Theoretical Analysis

4.7.1 ***Algorithm HM-Simplify and HM-SP.*** We analyze these two algorithms in Theorems 4.1 and 4.2.

Theorem 4.1. *The simplification time and output size algorithm HM-Simplify are $O(n\sqrt[3]{n}\log n)$ and $O(\frac{n}{\log n})$, respectively. Given M, it returns $\widetilde{M}$ 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 points 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 $O(1)$ cells in *R2R*, *R2D* and *D2D* checking in total $O(\sqrt[3]{n})$ cell merging iterations. The output size $O(n \div \log n) = O(\frac{n}{\log n})$ is due to the $O(\log n)$ deleted cells belonging to each added cell, and there are total n cells on M. The error guarantee of $\widetilde{M}$ is due to the *R2R*, *R2D* and *D2D* checking. The detailed proof appears in the appendix. □

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

Proof. Since there are $O(n)$ and $O(\frac{n}{\log n})$ cells in M and $\widetilde{M}$, respectively, and algorithm *HM-SP* is Dijkstra's algorithm which returns the exact result on M and $\widetilde{M}$, and $\widetilde{M}$ is an ϵ-approximate simplified height map of M, we can finish the proof. □

4.7.2 ***Proximity query algorithms.*** We show query time and error guarantee of *kNN* and range queries using algorithm *HM-SP* on M and $\widetilde{M}$ in Theorem 4.3. Given a query point i, let p_f and p'_f be the furthest point to i computed by algorithm *HM-SP* on M and $\widetilde{M}$, respectively. Let the error ratio 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 HM-SP is $O(n\log n)$ on M and is $O(\frac{n}{\log n}\log\frac{n}{\log n})$ on $\widetilde{M}$, respectively. It returns the exact result on M and has an error ratio $\frac{2\epsilon}{1-\epsilon}$ on $\widetilde{M}$ for both kNN and range queries, respectively.*

Proof Sketch. The query time is due to the usage of algorithm *HM-SP* once. The error ratio arises from its definition and the error of algorithm *HM-Simplify*. □

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 [35, 36, 50, 51, 57–59].

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 [8]. *BH_m* and *EP_m* are originally represented as points clouds with 8km × 6km region, we created height maps with cell's 2D coordinate and elevation values equal to the z-coordinate values of these points. These five datasets have a 10m × 10m resolution [36, 51, 57, 58]. (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 [51, 57, 58]. (iii) *24 Multi-resolution datasets*: They are generated similarly with varying numbers of cells. (2 & 3) *Point cloud* and *TIN* datasets: We convert the height map datasets to 34 point cloud datasets using 2D coordinate and elevation values of the height map's cells [32, 43], and then convert the point cloud datasets to 34 *TIN* datasets by triangulating [29, 40, 62] point cloud's points [58]. We use c and t as subscripts, respectively.

5.1.2 ***Algorithms.*** (1) To solve our problem on *Height Maps*, we adapted existing algorithms on point clouds or *TIN*s, by converting the given height maps to point clouds [32, 43] or *TIN*s [32, 43, 58] so that the existing algorithms could be performed. Their algorithm names are appended by "*-Adapt(HM)*". We have 3 height map simplification algorithms: (i) *TIN-SSimplify-Adapt(HM)* [34, 36], (ii) *TIN-NSimplify-Adapt(HM)* [36], and (iii) *HM-Simplify*. For 3 proximity query algorithms *TIN-ESSP-Adapt(HM)* [22, 52, 59], *TIN-SNP-Adapt(HM)* [36] and *HM-SP*, we append "*-Bas/Adv*" (basic/advanced) in their algorithm names to denote we use them on original/simplified *TIN*s or height maps (calculated by 3 simplification algorithms), respectively, since the advanced version involves an additional point estimation step. We have 8 proximity query algorithms: (i) *TIN-ESSP-Bas-Adapt(HM)* [22, 52, 59], (ii) *TIN-ESSP-Adv-Adapt(HM)* [22, 52, 59], (iii) *TIN-ASSP-Adapt(HM)* [35, 57], (iv) *TIN-SNP-Bas-Adapt(HM)* [36], (v) *TIN-SNP-Adv-Adapt(HM)* [36], (vi) *PC-SP-Adapt(HM)* [58], (vii) *HM-SP-Bas* and (viii) *HM-SP-Adv*. We compare them in Tables 2 and 3. Note that *TIN-ASSP-Adapt(HM)* does not have basic/advanced versions, since given a simplified *TIN*, we already used *TIN-ESSP-Adv-Adapt(HM)* to calculate the exact shortest surface path passing on this simplified *TIN*. Besides, *PC-SP-Adapt(HM)* does not have basic/advanced versions, since there is no existing point cloud simplification algorithm.

(2) To solve the existing problem on Point Clouds [58], we adapted algorithms on height maps or *TIN*s, by converting the given point clouds to height maps [31, 60] or *TIN*s [58]. Their algorithm names are appended by "*-Adapt(PC)*". Similarly, we have 11 algorithms: (i) *TIN-SSimplify-Adapt(PC)* [34, 36], (ii) *TIN-NSimplify-Adapt(PC)* [36], (iii) *HM-Simplify-Adapt(PC)*, (iv) *TIN-ESSP-Bas-Adapt(PC)* [22, 52, 59], (v) *TIN-ESSP-Adv-Adapt(PC)* [22, 52, 59], (vi) *TIN-ASSP-Adapt(PC)* [35, 57], (vii) *TIN-SNP-Bas-Adapt(PC)* [36], (viii) *TIN-SNP-Adv-Adapt(PC)* [36], (ix) *PC-SP* [58], (x) *HM-SP-Bas-Adapt(PC)* and (xi) *HM-SP-Adv-Adapt(PC)*.

(3) To solve the existing problem on *TIN*s [34, 36], we adapted algorithms on height maps or point clouds, by converting the given *TIN*s to height maps [39] or point clouds [58, 61]. Their algorithm names are appended by "*-Adapt(TIN)*". Similarly, we have 11 algorithms: (i) *TIN-SSimplify* [34, 36], (ii) *TIN-NSimplify* [36], (iii) *HM-Simplify-Adapt(TIN)*, (iv) *TIN-ESSP-Bas* [22, 52, 59], (v) *TIN-ESSP-Adv* [22, 52, 59], (vi) *TIN-ASSP* [35, 57], (vii) *TIN-SNP-Bas* [36], (viii) *TIN-SNP-Adv* [36], (ix) *PC-SP-Adapt(TIN)* [58], (x) *HM-SP-Bas-Adapt(TIN)* and (xi) *HM-SP-Adv-Adapt(TIN)*.

5.1.3 ***Proximity Queries.***

We conducted 3 queries. (1) Shortest path query: we generated 100 queries by randomly selecting two points on the height map, point cloud or *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) *kNN* and range queries: we randomly selected 1000 points on the height map, point cloud or *TIN* as query objects to perform the proximity query algorithm in Section 4.6.

5.1.4 ***Factors and Metrics.***

We studied 4 factors: (1) ϵ (the error parameter), (2) n (the dataset size, meaning the number of cells of a height map, points of a point cloud, or vertices of a *TIN*), (3) k (k value in *kNN* query) and (4) r (range value in range query). When not varying $k \in [200, 1000]$ and $r \in$ [2km, 10km], we fix k at 500 and r at 5km according to studies [26, 48]. For simplification algorithms, we employ 2 metrics: (1) *preprocessing time* (which is the height map, point cloud and *TIN* data conversion time (if any) plus the simplification time, where the data conversion time is 10^6 to 10^9 times smaller than the simplification time) and (2) *output size*. For proximity query algorithms, we employ 7 metrics: (1) *query time* (the data conversion time (if any) plus the shortest path query time, where the data conversion time is 10^4 to 10^6 times smaller than the shortest path time), (2 & 3) *kNN or range query time* (the data conversion time (if any) plus *kNN* or range query time), (4) *memory consumption* (the storage complexity during algorithm execution), (5) *distance error ratio* (the distance error ratio of the algorithm compared with the exact algorithm), (6 & 7) *kNN or range query error ratio* (see Section 4.7.2).

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-SP-Bas* returns the height map's exact shortest path and its computed path's distance is used for distance error ratio calculation. We compared all algorithms in Tables 2 and 3 on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(HM)* and *TIN-NSimplify-Adapt(HM)* on original datasets (since they have excessive simplification time on original datasets), and except *TIN-ESSP-Adv-Adapt(HM)* and *TIN-SNP-Adv-Adapt(HM)* (since they depend on the previous two algorithms).

(1) **Baseline comparisons**:

(i) **Effect of ϵ**: In Figures 7 and 8, 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-Simplify* is much smaller than that of *TIN-SSimplify-Adapt(HM)* and *TIN-NSimplify-Adapt(HM)* due to the efficient height map shortest path and efficient ϵ-approximate simplified height map checking. The output size of *HM-Simplify* 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 ($O(n' \cdot \frac{n}{\log n} \log \frac{n}{\log n}) = O(\frac{nn'}{\log n} \log \frac{n}{\log n})$ in Theorem 4.3) of *HM-SP-Adv* are also small since its simplified height map has a small output size. The distance error ratio of *HM-SP-Adv* is close to 0. So, the experimental *kNN* and range query error ratios are 0 (since $|\Pi(i, p'_f|M)| = |\Pi(i, p_f|M)|$ in Section 4.7.2, although the theoretical error ratios are $\frac{2\epsilon}{1-\epsilon}$ in Theorem 4.3), and their results are omitted.

(ii) **Effect of n (scalability test)**: In Figures 9 and 10, 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-Simplify* outperforms all baselines. The simplification time of *HM-Simplify* is 19,000s $\approx$ 5.2 hours for a height map with 2.5M cells, which shows its scalability.

(2) **Ablation study for proximity query algorithms (effect of k and r)**: We considered two variations of *HM-SP-Adv*, i.e., (i) *HM-SP-Adv Large Query Time* (*HM-SP-Adv-LQT1*): *HM-SP-Adv* using the naive algorithm in the shortest path query phase in Section 4.5, but the efficient proximity query algorithm in Section 4.6, and (ii) *HM-SP-Adv-LQT2*: *HM-SP-Adv* using the efficient algorithm in the shortest path query phase, but the naive proximity query algorithm. In Figure 11, 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-SP-Adv* outperforms both *HM-SP-Adv-LQT1* and *HM-SP-Adv-LQT2*, since we use the efficient algorithm for querying. k does not affect the *kNN* query time of *HM-SP-Adv*, 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

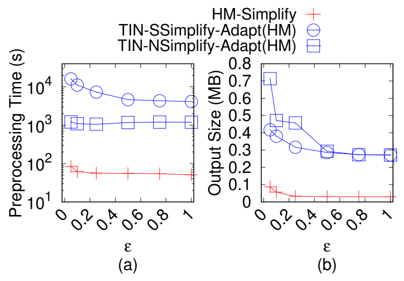


Figure 7: Effect of ϵ on *GF$_m$-small* height map dataset for simplification algorithms

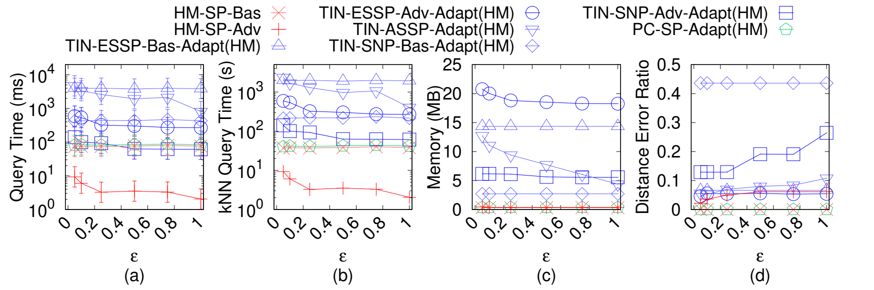


Figure 8: Effect of ϵ on *GF$_m$-small* height map dataset for proximity query algorithms

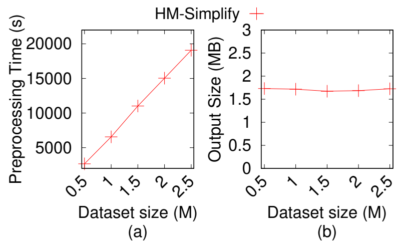


Figure 9: Effect of n on *LM$_m$* height map dataset for simplification algorithms

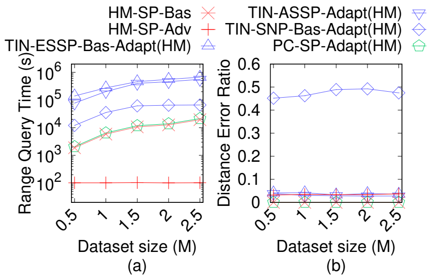


Figure 10: Effect of n on *LM$_m$* height map dataset for proximity query algorithms

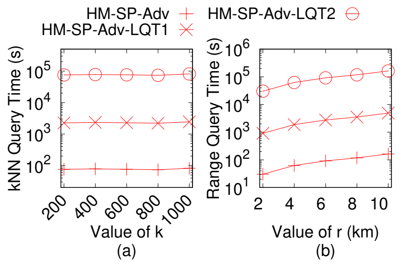


Figure 11: Ablation study for proximity query algorithms (effect of k and r on *RM$_m$* height map dataset)

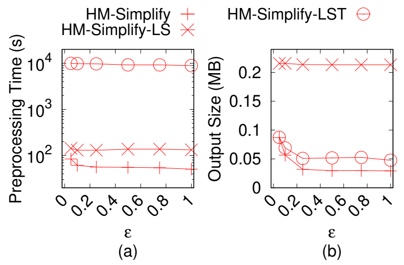


Figure 12: Ablation study for simplification algorithms on *BH$_m$-small* height map dataset

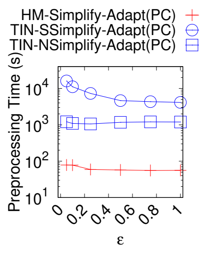


Figure 13: Effect of ϵ on *EP$_c$-small* point cloud dataset for simplification algorithms

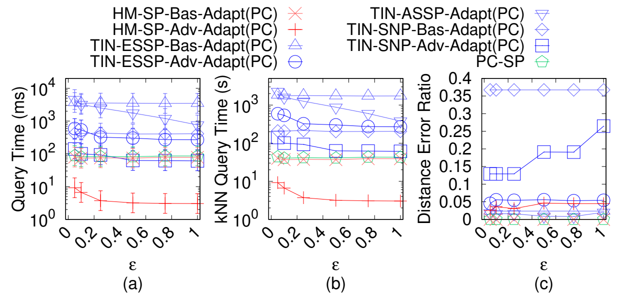


Figure 14: Effect of ϵ on *EP$_c$-small* point cloud dataset for proximity query algorithms

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 variations of *HM-Simplify*, i.e., (i) *HM-Simplify Large output Size* (*HM-Simplify-LS*): *HM-Simplify* using the naive merging technique that only merges four cells in Section 4.2, and (ii) *HM-Simplify Large Simplification Time* (*HM-Simplify-LST*): *HM-Simplify* using the naive checking technique that checks whether Inequality 1 is satisfied for all points in Section 4.2. In Figure 12, 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-Simplify* 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-SP* returns the point cloud's exact shortest path and its computed path's distance is used for distance error ratio calculation.

Effect of ϵ: In Figures 13 and 14, 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-Simplify-Adapt(PC)* and *HM-SP-Adv-Adapt(PC)* still outperforms other baselines. We studied the effect of n in the appendix.

5.2.3 *TINs*.

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

Effect of n: In Figures 15 and 16, we tested 5 values of n in {10k, 20k, 30k, 40k, 50k} on *EP$_t$-small* dataset while fixing ϵ at

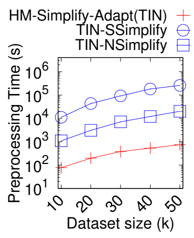


Figure 15: Effect of n on EP_t-small TIN dataset for simplification algorithms

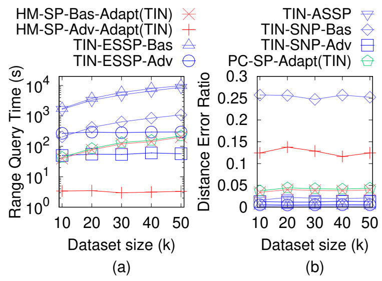


Figure 16: Effect of n on EP_t-small *TIN* dataset for proximity query algorithms

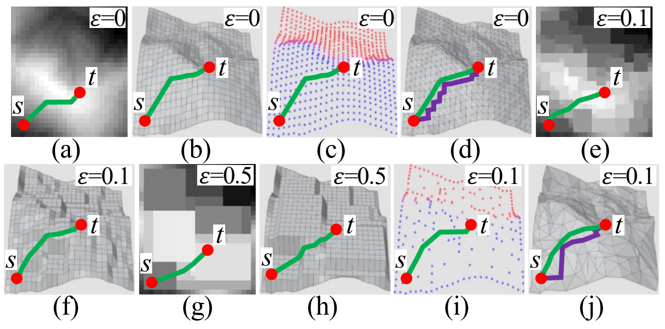


Figure 17: Paths passing on original/simplified height maps (in bird's view), point clouds and *TIN*s

0.1 for baseline comparisons. Despite giving a *TIN* as input, *HM-Simplify-Adapt(TIN)* outperforms *TIN-SSimplify* and *TIN-NSimplify* concerning the simplification time. The distance error ratio of *HM-SP-Adv-Adapt(TIN)* is 0.15, but the distance error ratio of *TIN-SNP* is 0.26. We studied the effect of ϵ in the appendix.

5.2.4 ***Case study***. We performed an evacuation case study at Gates of the Arctic [46] in response to snowfall [4], and tourists on the mountain are promptly evacuated to nearby hotels for safety. In Figure 1 (a), due to each hotel's capacity constraints, we aim to identify shortest paths from the position a of some tourists to k-nearest hotels b, c, d, where c and d are k-nearest options when $k = 2$. An individual will be buried in snow in 2.4 hours[3], and the evacuation (i.e., walking from the viewpoints to hotels) can be finished in 2.2 hours[4]. Thus, we need to compute shortest paths within 12 min (= 2.4 − 2.2 hours). Our experiments show that for a height map with 50k cells, 10k possible tourist positions and 50 hotels, the simplification time for our algorithm and the best-known adapted simplification algorithm *TIN-SSimplify-Adapt(HM)* are 250s ≈ 4.6 min and 103,000s ≈ 1.2 days, and computing 10 nearest hotels for each tourist position on our simplified height map, a point cloud and a simplified *TIN* takes 50s, 67,000s ≈ 18.6 hours and 7,630s ≈ 2.1 hours, respectively. Thus, height map simplification is necessary for evacuation since 4.6 min + 50s ≤ 12 min.

5.2.5 ***Paths visualization***. In Figure 17, we visualize different paths to verify distance relationships in Section 3.1.

(1) Given a height map, the paths in Figures 17 (a) (showing the height map) and (b) (showing the 3D surface of the same height map in the bird's eye view) computed by our algorithm *HM-SP* and the path in Figure 17 (c) computed by the best-known adapted point cloud shortest path query algorithm *PC-SP-Adapt(HM)* are identical (since $|\Pi(s,t|M)| = |\Pi(s,t|P)|$). The paths in Figures 17 (a) and (b) are similar to the green path in Figure 17 (d) computed by the best-known adapted exact *TIN* shortest surface path query algorithm *TIN-ESSP-Adapt(HM)* (since $|\Pi(s,t|M)| \leq \alpha \cdot |\Pi(s,t|T)|$), but computing the former path is much quicker. The paths in Figures 17 (a) and (b) are similar to the paths in Figures 17 (e), (f), (g) and (h) computed by our algorithm *HM-SP-Adv*, but computing the latter four paths are quicker due to the simplified height maps. The paths in Figures 17 (e) and (f) are the same as the path in Figure 17 (i) on a simplified point cloud (generated by adapting *HM-Simplify* on a point cloud), and similar to the green (surface) path in Figure 17 (j) on a simplified *TIN* (generated by the best-known adapted *TIN* simplification algorithm *TIN-SSimplify-Adapt(HM)*).

(2) Given a point cloud, the paths in Figures 17 (a) and (b) computed by our adapted algorithm *HM-SP-Adapt(PC)* are the same as the path in Figure 17 (c) computed by the best-known point cloud shortest path query algorithm *PC-SP*.

(3) Given a *TIN*, the paths in Figures 17 (a) and (b) computed by our adapted algorithm *HM-SP-Adapt(TIN)* are similar to the green (surface) path in Figure 17 (d) computed by the best-known exact *TIN* shortest surface path query algorithm *TIN-ESSP*. The distance error ratios of the paths in Figures 17 (a) and (b) are smaller than that of the purple (network) path in Figure 17 (d) computed by the best-known approximate *TIN* shortest network path query algorithm *TIN-ESSP* (since $|\Pi(s,t|M)| \leq |\Pi_N(s,t|T)|$).

5.2.6 ***Summary***. On a height map with 50k cells and 10k objects, *HM-Simplify*'s simplification time and output size are 250s ≈ 4.6 min and 0.07MB, which are up to 412 times and 7 times better than the best-known adapted simplification algorithm *TIN-SSimplify-Adapt(HM)*, respectively. Performing *kNN* query on our simplified height map takes 50s, which is up to 153 times quicker than the best-known adapted point cloud shortest path query algorithm *PC-SP-Adapt(HM)* on a point cloud, and 1,340 times quicker than the best-known adapted *TIN* shortest path query algorithm *TIN-ESSP-Adapt(HM)* on the simplified *TIN*.

6 CONCLUSION

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

[3] 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 [23, 47]) at Gates of the Arctic is 1 meter per 24 hours [4], and when the snow depth exceeds 10 centimeters, it is difficult to walk and easy to bury in the snow [30].

[4] 2.2 hours = $\frac{11.2\text{km}}{5.1\text{km/h}}$, since the average distance between the viewpoints and hotels at Gates of the Arctic is 11.2km [9], and human's average walking speed is 5.1 km/h [16].

REFERENCES

[1] 2017. *1.5 Billion the world's most expensive imaging satellite NISAR.* https://syntheticapertureradar.com/nasa-isro-sar-project-nisar/
[2] 2025. *125 Years of Topographic Mapping.* https://www.esri.com/news/arcnews/fall09articles/125-years.html
[3] 2025. *50 Years of Landsat Science.* https://landsat.gsfc.nasa.gov
[4] 2025. *Climate and average weather year round in Gates of the Arctic National Park.* https://weatherspark.com/y/150336/Average-Weather-in-Gates-of-the-Arctic-National-Park-Alaska-United-States-Year-Round
[5] 2025. *Data Geocomm.* http://data.geocomm.com/
[6] 2025. *The Earth.* https://www.nationsonline.org/oneworld/earth.htm
[7] 2025. *Geocento commercial SAR satellite imagery resolutions 2022, by cost per scene.* https://www.statista.com/statistics/1293899/geocento-commercial-satellite-sar-imagery-resolution-cost-worldwide/
[8] 2025. *Google Earth.* https://earth.google.com/web
[9] 2025. *Google Map.* https://www.google.com/maps
[10] 2025. *Gunnison National Forest.* https://gunnisoncrestedbutte.com/visit/places-to-go/parks-and-outdoors/gunnison-national-forest/
[11] 2025. *The History of Point Cloud Development.* https://www.linkedin.com/pulse/history-point-cloud-development-bimprove/
[12] 2025. *How satellites work.* https://science.howstuffworks.com/satellite10.htm
[13] 2025. *Laramie Mountain.* https://www.britannica.com/place/Laramie-Mountains
[14] 2025. *Metaverse.* https://about.facebook.com/meta
[15] 2025. *Open Digital Elevation Model (OpenDEM).* https://www.opendem.info/
[16] 2025. *Preferred walking speed.* https://en.wikipedia.org/wiki/Preferred_walking_speed
[17] 2025. *Robinson Mountain.* https://www.mountaineers.org/activities/routes-places/robinson-mountain
[18] 2025. *Shuttle Radar Topography Mission (SRTM).* https://www.earthdata.nasa.gov/data/instruments/srtm
[19] 2025. *Transparent pricing for commercial Earth observation imagery.* https://skyfi.com/en/pricing
[20] 2025. *Zoom Earth.* https://zoom.earth/maps/
[21] Leizhen Cai. 1994. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics* 48, 2 (1994), 187–194.
[22] 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.
[23] 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
[24] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.
[25] 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.
[26] 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.
[27] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[28] Hristo N Djidjev and Christian Sommer. 2011. Approximate distance queries for weighted polyhedral surfaces. In *Proceedings of the European Symposium on Algorithms.* 579–590.
[29] Michael Garland and Paul S Heckbert. 1995. Fast polygonal approximation of terrains and height fields. (1995).
[30] 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/
[31] AJ Harrison, BM Miller, JT Carley, IL Turner, R Clout, and B Coates. 2017. NSW beach photogrammetry: A new online database and toolbox. In *Australasian Coasts & Ports 2017: Working with Nature: Working with Nature.* Engineers Australia, PIANC Australia and Institute of Professional Engineers, 565–571.
[32] 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.
[33] Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques.* 99–108.
[34] 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.
[35] 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.
[36] 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.
[37] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. In *Algorithmica*, Vol. 30. 527–562.
[38] 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.
[39] Niraj Manandhar. 2005. Comparison of TIN and GRID method of contour generation from spot height. *Journal on Geoinformatics, Nepal* (2005), 1–8.
[40] De Berg Mark, Cheong Otfried, Van Kreveld Marc, and Overmars Mark. 2008. *Computational geometry algorithms and applications.*
[41] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.
[42] 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.
[43] 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.
[44] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance oracles for spatial networks. In *International Conference on Data Engineering (ICDE).* IEEE, 652–663.
[45] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path oracles for spatial networks. 2, 1 (2009), 1210–1221.
[46] National Park Service. 2025. *Gates of the Arctic.* https://www.nps.gov/gaar/index.htm
[47] National Weather Service. 2025. *Measuring Snow.* https://www.weather.gov/dvn/snowmeasure
[48] 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.
[49] Farhan Tauheed, Laurynas Biveinis, Thomas Heinis, Felix Schurmann, Henry Markram, and Anastasia Ailamaki. 2012. Accelerating range queries for brain simulations. In *International Conference on Data Engineering (ICDE).* IEEE, 941–952.
[50] 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.
[51] 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).
[52] 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.
[53] 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.
[54] 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.
[55] 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.
[56] 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.
[57] 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).*
[58] 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.
[59] 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.
[60] Tao Yang, Peiqi Li, Huiming Zhang, Jing Li, and Zhi Li. 2018. Monocular vision SLAM-based UAV autonomous landing in emergencies and unknown environments. *Electronics* 7, 5 (2018), 73.
[61] Hongchuan Yu, Jian J Zhang, and Zheng Jiao. 2014. Geodesics on point clouds. *Mathematical Problems in Engineering* 2014 (2014).
[62] 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 4 shows a summary of all notation.

Table 4: Summary of all notation

Notation	Meaning
M	The height map with a set of cells
C	The set of cells of M
$N(\cdot)$	The neighbor cells table of M
n	The number of cells of M
P	The point cloud converted from M
T	The *TIN* converted from M
θ	The minimum inner angle of any face in T
G	The height map graph of M and the point cloud graph of P
$G.V/G.E$	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
$\Pi(s,t\|P)$	The shortest path passing on P 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
$\widetilde{C}$	The set of cells of $\widetilde{M}$
$\widetilde{N}(\cdot)$	The neighbor cells table of $\widetilde{M}$
C_{rema}	The set of remaining cells
C_{dd}	The set of added cells
$\widetilde{G}$	The simplified height graph of $\widetilde{M}$
$\widetilde{G}.V/\widetilde{G}.E$	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
$O(\cdot)$	The containing table
$O^{-1}(\cdot)$	The belonging table
$\widehat{C}$	The set of adjacent cells that we need to merge in each simplification iteration
c_{add}	The added cell formed by merging each cell $\widehat{C}$
$\widetilde{c}$	The estimated cell of c
$\Pi_1(p,q\|\widetilde{M})$	The intra-path passing on $\widetilde{M}$ between c and q
$\Pi_2(p,q\|\widetilde{M})$	The inter-path passing on $\widetilde{M}$ between c and q
$L(c_{add})$	The set of linked added cells of c_{add}

B COMPARISON OF ALL ALGORITHMS

Tables 5 and 6 show comparisons of all simplification and proximity query algorithms. Recall that we have two variations of *HM-SP-Adv* in terms of proximity queries, i.e., *HM-SP-Adv-LQT1* and *HM-SP-Adv-LQT2*. Let *HM-Simplify-LQT1* and *HM-Simplify-LQT2* be the simplification algorithms, so that *HM-SP-Adv-LQT1* and *HM-SP-Adv-LQT2* are applied on the simplified height map of these two simplification algorithms. Recall that we have two variations of *HM-Simplify* in terms of the simplification process, i.e., *HM-Simplify-LS* and *HM-Simplify-LST*. Let *HM-SP-Adv-LQT1* and *HM-SP-Adv-LQT2* be the proximity query algorithms, so that they answer the corresponding proximity query algorithms on the simplified height map of these two simplification algorithms.

C EMPIRICAL STUDIES

C.1 Experimental Results for Height Maps

C.1.1 ***Baseline comparisons.*** We studied the effect of ϵ and n on height maps for baseline comparisons. We compared algorithms *TIN-SSimplify-Adapt(HM)*, *TIN-NSimplify-Adapt(HM)*, *HM-Simplify*, *TIN-ESSP-Bas-Adapt(HM)*, *TIN-ESSP-Adv-Adapt(HM)*, *TIN-ASSP-Adapt(HM)*, *TIN-SNP-Bas-Adapt(HM)*, *TIN-SNP-Adv-Adapt(HM)*, *PC-SP-Adapt(HM)*, *HM-SP-Bas* and *HM-SP-Adv* on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(HM)* and *TIN-NSimplify-Adapt(HM)* on original datasets (since they have excessive simplification time on original datasets), and except *TIN-ESSP-Adv-Adapt(HM)* and *TIN-SNP-Adv-Adapt(HM)* (since they depend on the previous two algorithms).

Effect of ϵ: In Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, Figure 23, Figure 24, Figure 25, Figure 26 and Figure 27, 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 30, Figure 31, Figure 34, Figure 35, Figure 38, Figure 39, Figure 42, Figure 43, 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_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-Simplify* is much smaller than that of *TIN-SSimplify-Adapt(HM)* and *TIN-NSimplify-Adapt(HM)* due to the efficient height map shortest path and efficient ϵ-approximate simplified height map checking. The output size of *HM-Simplify* 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 of *HM-SP-Adv* are also small since its simplified height map has a small output size. The distance error ratio of *HM-SP-Adv* is close to 0. So, the experimental *kNN* and range query error ratios are 0, and their results are omitted.

Effect of n (scalability test): In Figure 28 and Figure 29, 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 32, Figure 33, Figure 36, Figure 37, Figure 40, Figure 41, Figure 44, Figure 45, Figure 48 and Figure 49 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-Simplify* outperforms all the remaining simplification algorithms and *HM-SP-Adv* outperforms all the remaining proximity query algorithms. The simplification time of *HM-Simplify* is 19,000s $\approx$ 5.2 hours for a height map with 2.5M cells, which shows its scalability.

C.1.2 ***Ablation study for proximity query algorithms.*** **Effect of k and r**: In Figure 50, Figure 51, Figure 54, Figure 55, Figure 58, Figure 59, Figure 62, Figure 63, Figure 66 and Figure 67 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. In Figure 52, Figure 53, Figure 56, Figure 57, Figure 60, Figure 61, Figure 64, Figure 65, Figure 68 and Figure 69 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. *HM-SP-Adv* outperforms both *HM-SP-Adv-LQT1* and *HM-SP-Adv-LQT2*, since we use the efficient algorithm for querying. k does not affect the *kNN* query time of *HM-SP-Adv*, since

Table 5: Comparison of all simplification algorithms

Algorithm	Simplification time		Output size	
TIN-SSimplify-Adapt(HM) [34, 36]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large
TIN-NSimplify-Adapt(HM) [36]	$O(n^2\log n)$	Medium	$O(n)$	Large
HM-Simplify-LQT1	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small
HM-Simplify-LQT2	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small
HM-Simplify-LS	$O(n\sqrt[3]{n}\log n)$	Small	$O(n)$	Large
HM-Simplify-LST	$O(n^2\sqrt[3]{n}\log n)$	Large	$O(\frac{n}{\log n})$	Small
***HM-Simplify* (ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**

Table 6: Comparison of all proximity query algorithms

Algorithm	Shortest path query time		*k*NN and range query time		Error
TIN-ESSP-Bac-Adapt(HM) [22, 52, 59]	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-ESSP-Adv-Adapt(HM) [22, 52, 59]	$O(n^2)$	Large	$O(n^2)$	Large	Small
TIN-ASSP-Adapt(HM) [35, 57]	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	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-SNP-Bac-Adapt(HM) [36]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
TIN-SNP-Adv-Adapt(HM) [36]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
PC-SP-Adapt(HM) [58]	$O(n\log n)$	Medium	$O(n\log n)$	Medium	No error
HM-SP-Adv-LQT1	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	Small
HM-SP-Adv-LQT2	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{nn'}{\log n}\log\frac{n}{\log n})$	Medium	Small
HM-SP-Adv-LS	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Small
HM-SP-Adv-LST	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
***HM-SP-Bac* (ours)**	$\boldsymbol{O(n\log n)}$	**Medium**	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**
***HM-SP-Adv* (ours)**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	**Small**

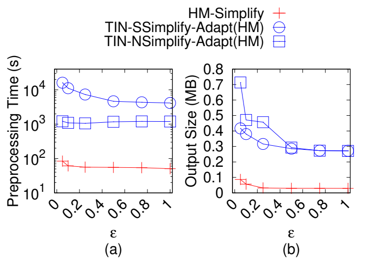


Figure 18: Effect of ϵ on *GF$_m$-small* height map dataset for simplification algorithms

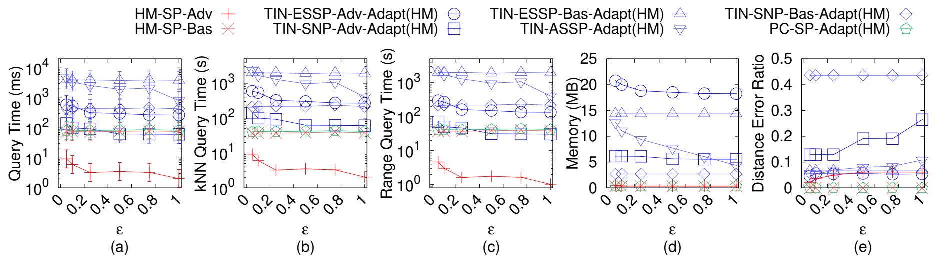


Figure 19: Effect of ϵ on *GF$_m$-small* height map dataset for proximity query algorithms

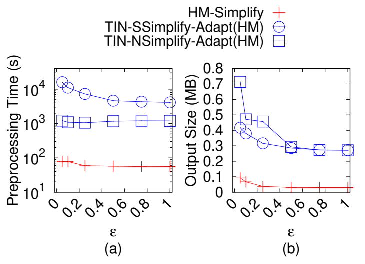


Figure 20: Effect of ϵ on *LM$_m$-small* height map dataset for simplification algorithms

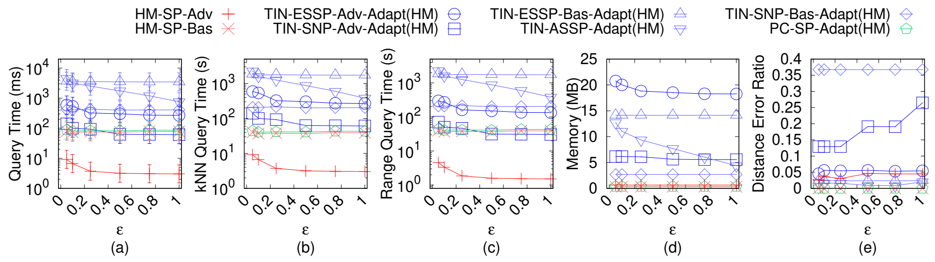


Figure 21: Effect of ϵ on *LM$_m$-small* height map dataset for proximity query algorithms

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.

*C.1.3 **Ablation study for simplification algorithms**.*

In Figure 70, Figure 71, Figure 72, Figure 73, Figure 74, Figure 75, Figure 76, Figure 77, Figure 78 and Figure 79 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. *HM-Simplify* outperforms these two

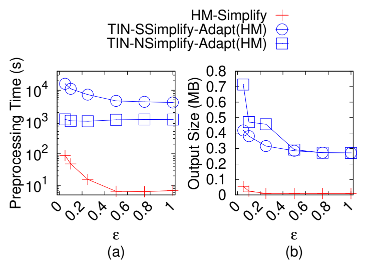


Figure 22: Effect of ϵ on *RM_m-small* height map dataset for simplification algorithms

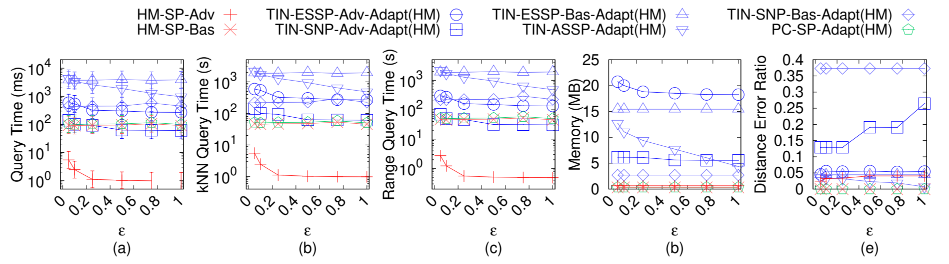


Figure 23: Effect of ϵ on *RM_m-small* height map dataset for proximity query algorithms

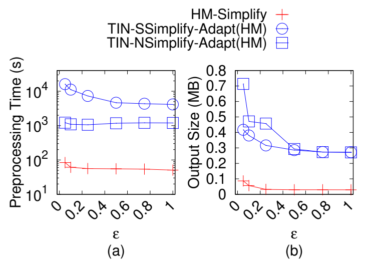


Figure 24: Effect of ϵ on *BH_m-small* height map dataset for simplification algorithms

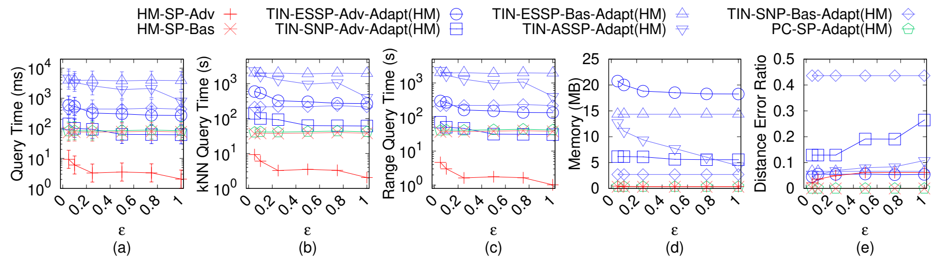


Figure 25: Effect of ϵ on *BH_m-small* height map dataset for proximity query algorithms

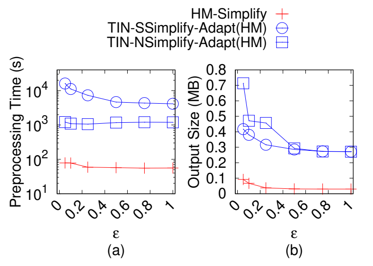


Figure 26: Effect of ϵ on *EP_m-small* height map dataset for simplification algorithms

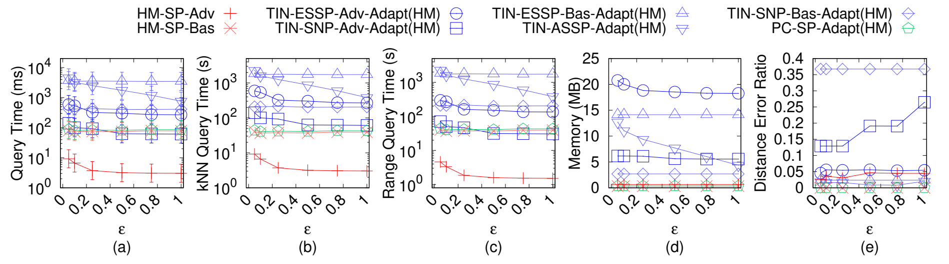


Figure 27: Effect of ϵ on *EP_m-small* height map dataset for proximity query algorithms

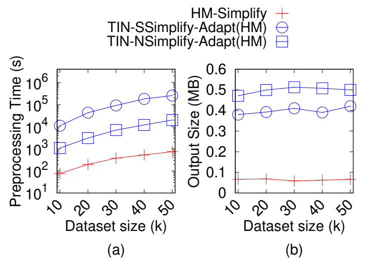


Figure 28: Effect of n on *EP_m-small* height map dataset for simplification algorithms

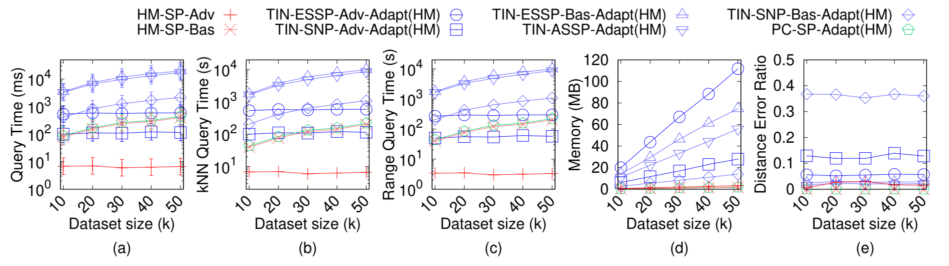


Figure 29: Effect of n on *EP_m-small* height map dataset for proximity query algorithms

variations, showing the effectiveness of our merging and checking techniques.

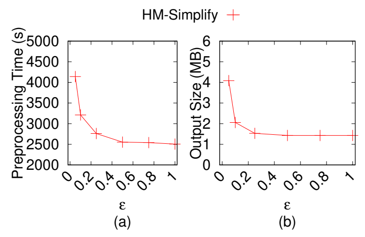


Figure 30: Effect of ϵ on GF_m height map dataset for simplification algorithms

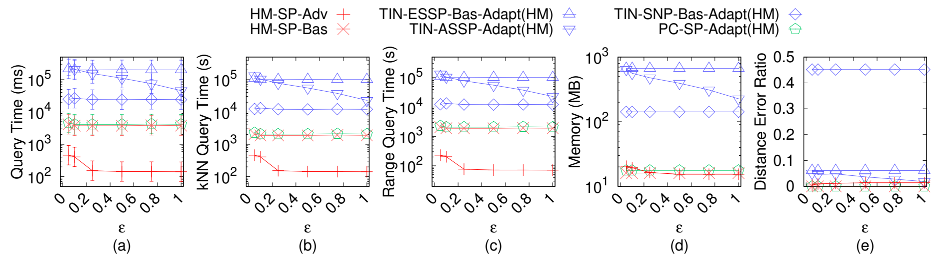


Figure 31: Effect of ϵ on GF_m height map dataset for proximity query algorithms

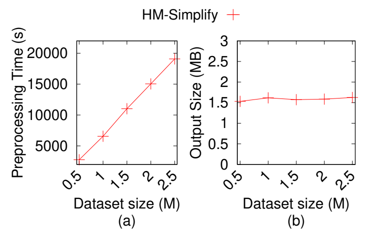


Figure 32: Effect of n on GF_m height map dataset for simplification algorithms

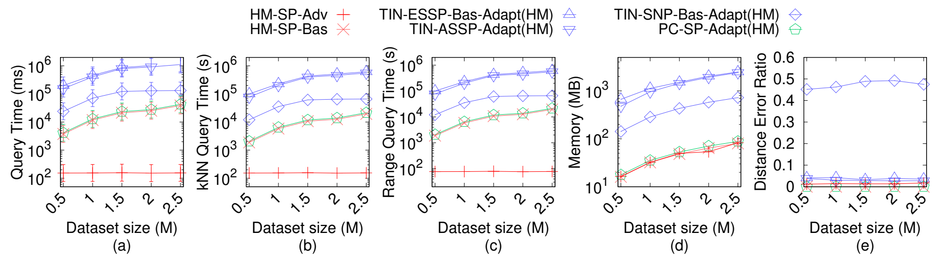


Figure 33: Effect of n on GF_m height map dataset for proximity query algorithms

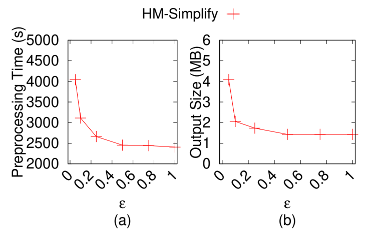


Figure 34: Effect of ϵ on LM_m height map dataset for simplification algorithms

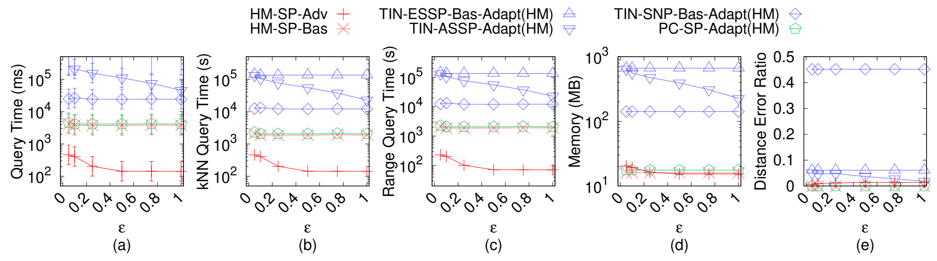


Figure 35: Effect of ϵ on LM_m height map dataset for proximity query algorithms

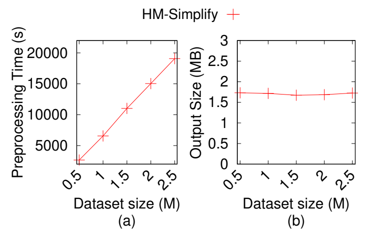


Figure 36: Effect of n on LM_m height map dataset for simplification algorithms

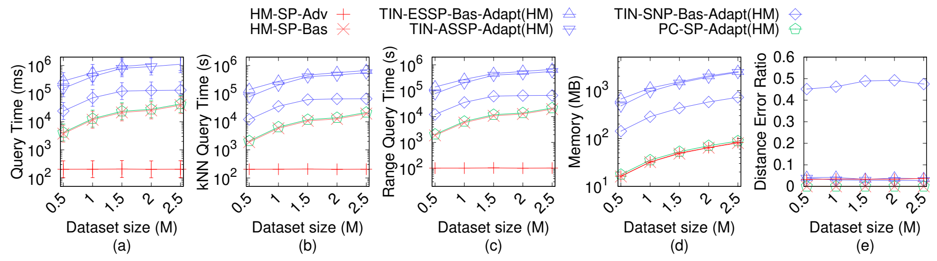


Figure 37: Effect of n on LM_m height map dataset for proximity query algorithms

C.2 Experimental Results for Point Clouds

We studied the effect of ϵ and n on height maps for baseline comparisons. We compared algorithms *TIN-SSimplify-Adapt(PC)*, *TIN-NSimplify-Adapt(PC)*, *HM-Simplify-Adapt(PC)*, *TIN-ESSP-Bas-Adapt(PC)*, *TIN-ESSP-Adv-Adapt(PC)*, *TIN-ASSP-Adapt(PC)*, *TIN-SNP-Bas-Adapt(PC)*, *TIN-SNP-Adv-Adapt(PC)*, *PC-SP*, *HM-SP-Bas-Adapt(PC)* and *HM-SP-Adv-Adapt(PC)* on small-version datasets, and compared all algorithms except *TIN-SSimplify-Adapt(PC)* and

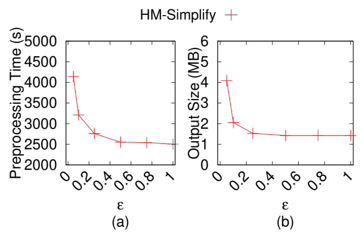


Figure 38: Effect of ϵ on RM_m height map dataset for simplification algorithms

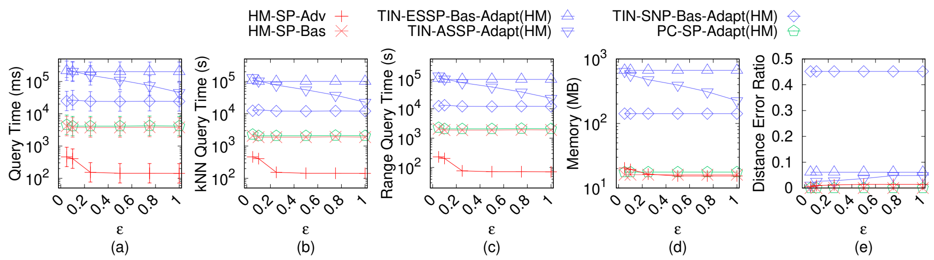


Figure 39: Effect of ϵ on RM_m height map dataset for proximity query algorithms

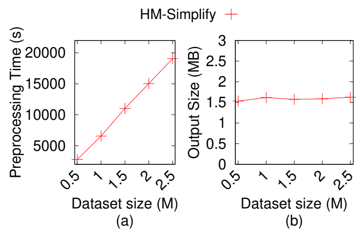


Figure 40: Effect of n on RM_m height map dataset for simplification algorithms

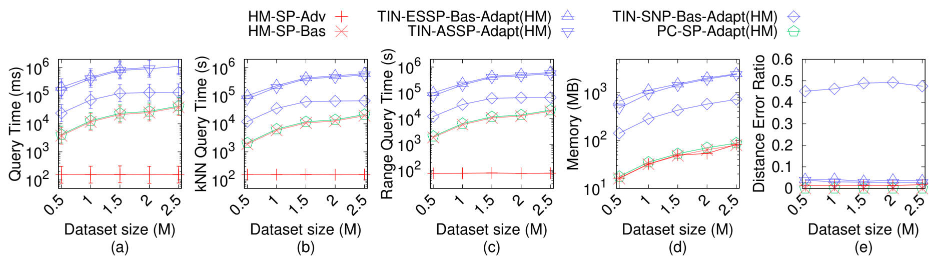


Figure 41: Effect of n on RM_m height map dataset for proximity query algorithms

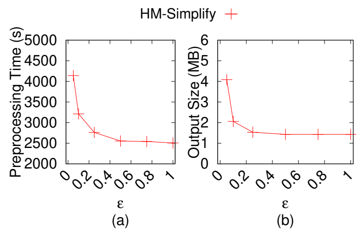


Figure 42: Effect of ϵ on BH_m height map dataset for simplification algorithms

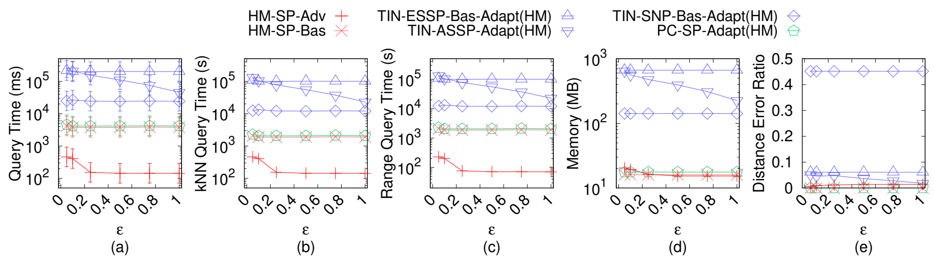


Figure 43: Effect of ϵ on BH_m height map dataset for proximity query algorithms

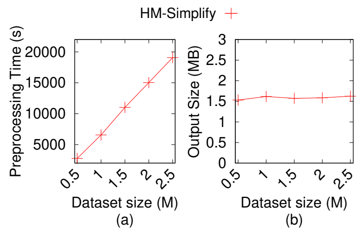


Figure 44: Effect of n on BH_m height map dataset for simplification algorithms

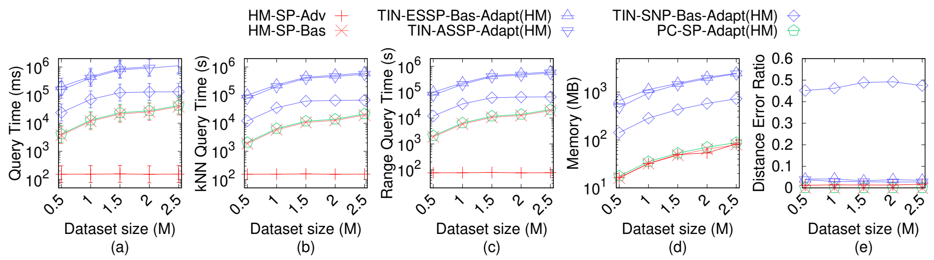


Figure 45: Effect of n on BH_m height map dataset for proximity query algorithms

TIN-NSimplify-Adapt(PC) on original datasets (since they have excessive simplification time on original datasets), and except *TIN-ESSP-Adv-Adapt(PC)* and *TIN-SNP-Adv-Adapt(PC)* (since they depend on the previous two algorithms).

Effect of ϵ: In Figure 80, Figure 81, Figure 82, Figure 83, Figure 84, Figure 85, Figure 86, Figure 87, Figure 88 and Figure 89, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on *GF_p-small*, *LM_p-small*, *RM_p-small*, *BH_p-small* and *EP_p-small* dataset while fixing n at 10k for baseline comparisons. In Figure 92, Figure 93, Figure 96, Figure 97, Figure 100, Figure 101, Figure 104, Figure 105, Figure 108 and Figure 109, we tested 6 values of ϵ in {0.05, 0.1, 0.25, 0.5, 0.75, 1} on GF_p, LM_p, RM_p, BH_p and EP_p dataset while fixing

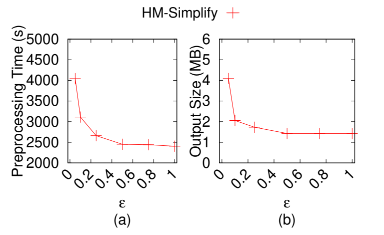


Figure 46: Effect of ϵ on EP_m height map dataset for simplification algorithms

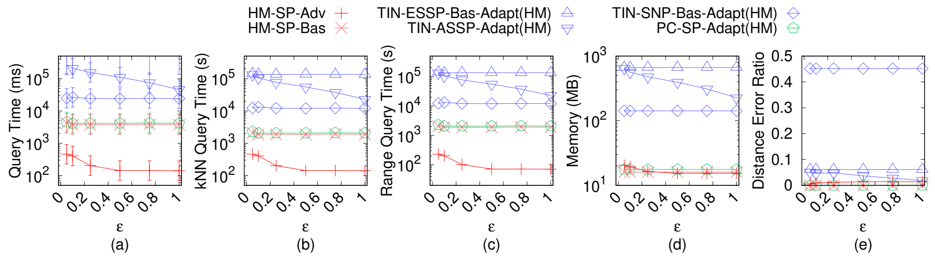


Figure 47: Effect of ϵ on EP_m height map dataset for proximity query algorithms

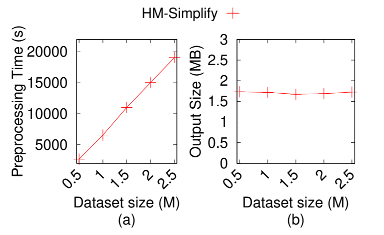


Figure 48: Effect of n on EP_m height map dataset for simplification algorithms

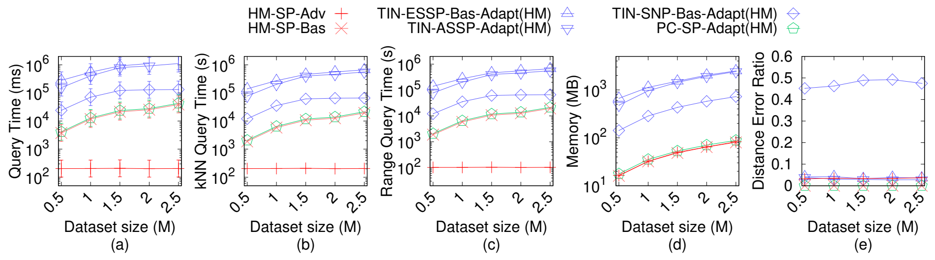


Figure 49: Effect of n on EP_m height map dataset for proximity query algorithms

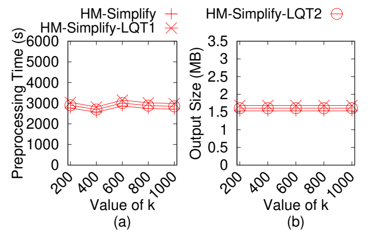


Figure 50: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of k on GF_m height map dataset)

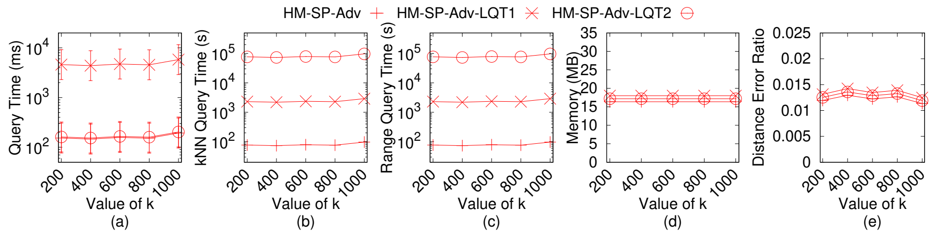


Figure 51: Ablation study for proximity query algorithms (effect of k on GF_m height map dataset)

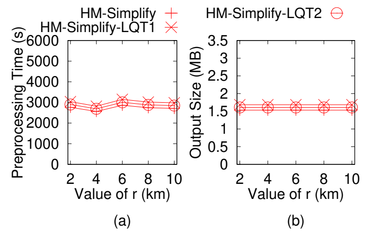


Figure 52: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of r on GF_m height map dataset)

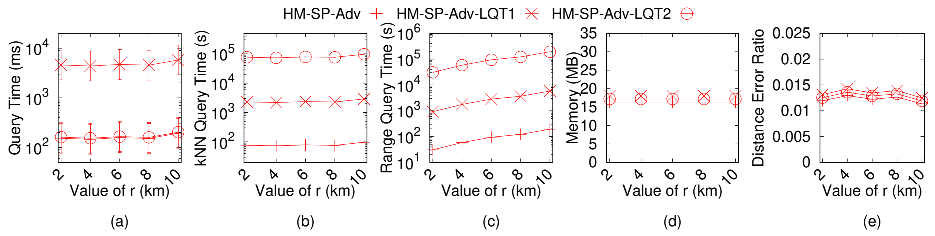


Figure 53: Ablation study for proximity query algorithms (effect of r on GF_m height map dataset)

n at 0.5M for baseline comparisons. The simplification time and output size of *HM-Simplify-Adapt(PC)* are much smaller than that of *TIN-SSimplify-Adapt(PC)* and *TIN-NSimplify-Adapt(PC)*. The proximity queries time of *HM-SP-Adv-Adapt(PC)* are also small since its simplified height map has a small output size.

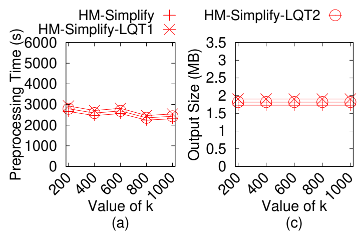


Figure 54: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of k on LM_m height map dataset)

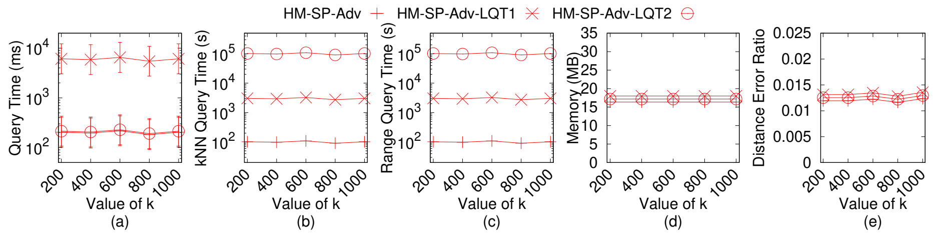


Figure 55: Ablation study for proximity query algorithms (effect of k on LM_m height map dataset)

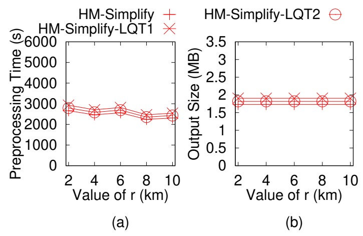


Figure 56: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of r on LM_m height map dataset)

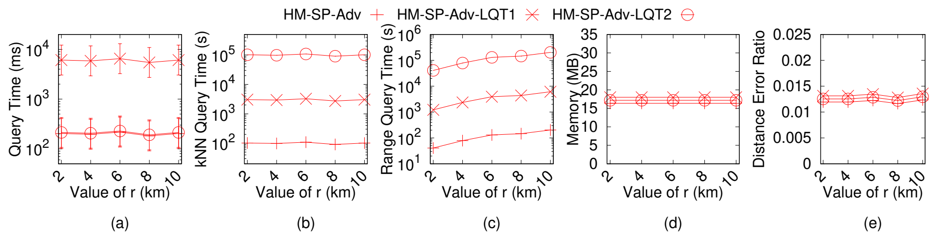


Figure 57: Ablation study for proximity query algorithms (effect of r on LM_m height map dataset)

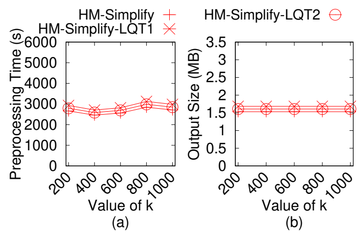


Figure 58: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of k on RM_m height map dataset)

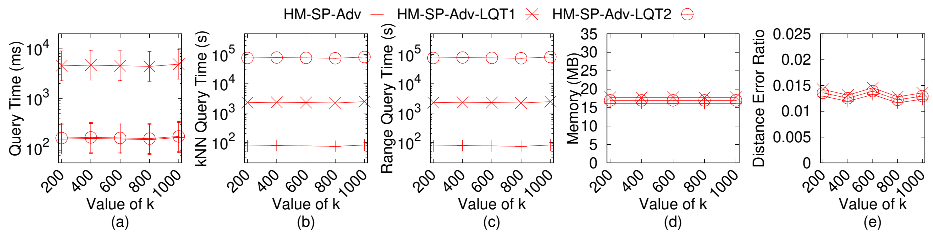


Figure 59: Ablation study for proximity query algorithms (effect of k on RM_m height map dataset)

Effect of n (scalability test): In Figure 90 and Figure 91, we tested 5 values of n in {50, 100, 150, 200, 250} on EP_p*-small* dataset while fixing ϵ at 0.1 for baseline comparisons. In Figure 94, Figure 95, Figure 98, Figure 99, Figure 102, Figure 103, Figure 106, Figure 107, Figure 110 and Figure 111 we tested 5 values of n in {500, 1000, 1500, 2000, 2500} on GF_p, LM_p, RM_p, BH_p and EP_p dataset while fixing ϵ at 0.25 for baseline comparisons. *HM-Simplify-Adapt(PC)* outperforms all the remaining simplification algorithms and *HM-SP-Adv-Adapt(PC)* outperforms all the remaining proximity query algorithms.

C.3 Experimental Results for *TIN*s

We studied the effect of ϵ and n on *TIN*s for baseline comparisons. We compared algorithms *TIN-SSimplify*, *TIN-NSimplify*, *HM-Simplify-Adapt(TIN)*, *TIN-ESSP-Bas*, *TIN-ESSP-Adv*, *TIN-ASSP*, *TIN-SNP-Bas*, *TIN-SNP-Adv*, *PC-SP-Adapt(TIN)*, *HM-SP-Bas-Adapt(TIN)* and *HM-SP-Adv-Adapt(TIN)* on small-version datasets, and compared all algorithms except *TIN-SSimplify* and *TIN-NSimplify* on original datasets (since they have excessive simplification time on original datasets), and except *TIN-ESSP-Ad* and *TIN-SNP-Adv* (since they depend on the previous two algorithms).

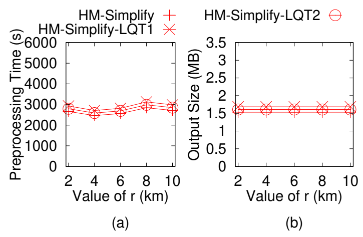


Figure 60: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of r on RM_m height map dataset)

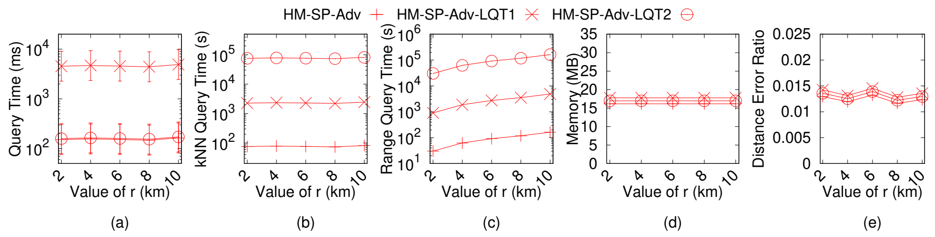


Figure 61: Ablation study for proximity query algorithms (effect of r on RM_m height map dataset)

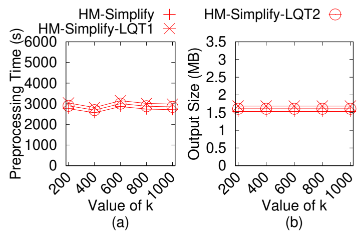


Figure 62: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of k on BH_m height map dataset)

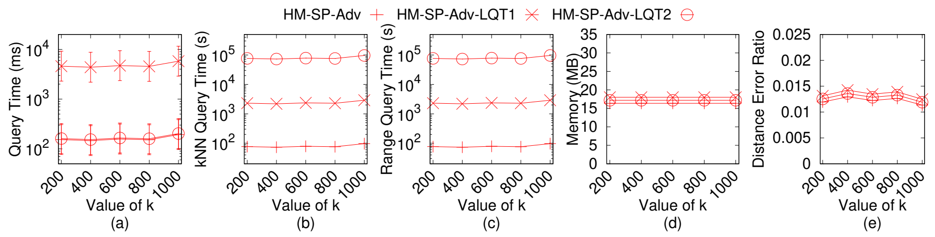


Figure 63: Ablation study for proximity query algorithms (effect of k on BH_m height map dataset)

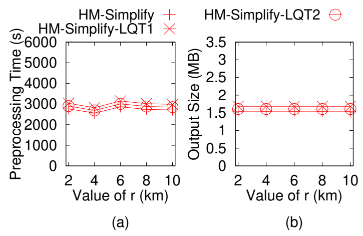


Figure 64: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of r on BH_m height map dataset)

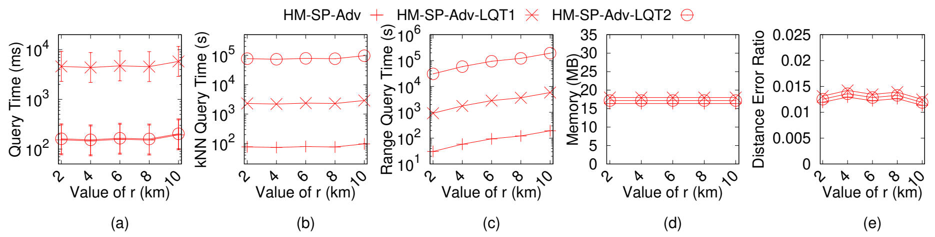


Figure 65: Ablation study for proximity query algorithms (effect of r on BH_m height map dataset)

Effect of ϵ: In Figure 112, Figure 113, Figure 114, Figure 115, Figure 116, Figure 117, Figure 118, Figure 119, Figure 120 and Figure 121, 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 124, Figure 125, Figure 128, Figure 129, Figure 132, Figure 133, Figure 136, Figure 137, Figure 140 and Figure 141, 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 while fixing n at 0.5M for baseline comparisons. The simplification time and output size of *HM-Simplify-Adapt(TIN)* are much smaller than that of *TIN-SSimplify* and *TIN-NSimplify*. The proximity queries time of *HM-SP-Adv-Adapt(TIN)* are also small since its simplified height map has a small output size.

Effect of n (scalability test): In Figure 122 and Figure 123, 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 126, Figure 127, Figure 130, Figure 131, Figure 134, Figure 135, Figure 138, Figure 139, Figure 142 and Figure 143 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-Simplify-Adapt(TIN)* outperforms all the remaining simplification algorithms and *HM-SP-Adv-Adapt(TIN)* outperforms all the remaining proximity query algorithms.

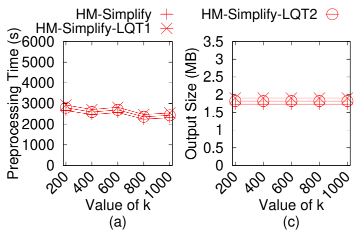


Figure 66: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of k on EP_m height map dataset)

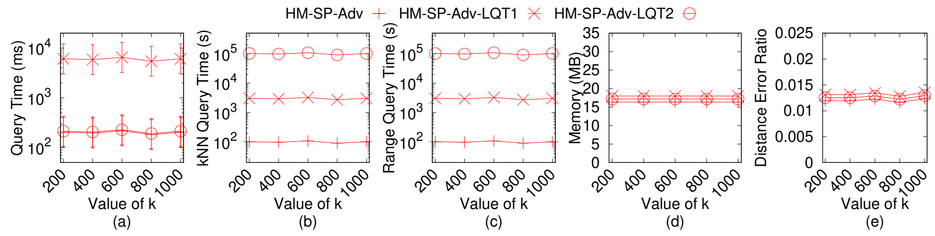


Figure 67: Ablation study for proximity query algorithms (effect of k on EP_m height map dataset)

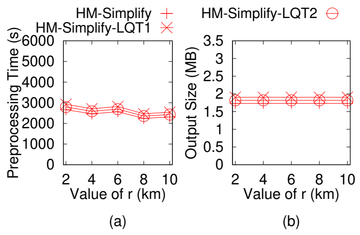


Figure 68: Ablation study for proximity query algorithms with results for corresponding simplification algorithms (effect of r on EP_m height map dataset)

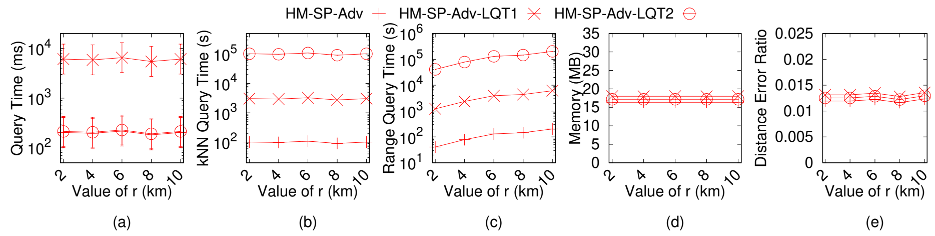


Figure 69: Ablation study for proximity query algorithms (effect of r on EP_m height map dataset)

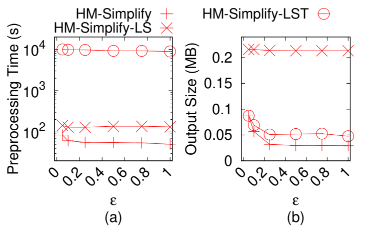


Figure 70: Ablation study for simplification algorithms on GF_m-*small* height map dataset

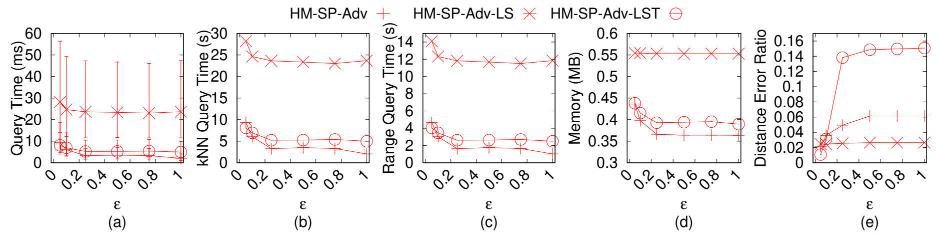


Figure 71: Ablation study for simplification algorithms with results for corresponding proximity query algorithms on GF_m-*small* height map dataset

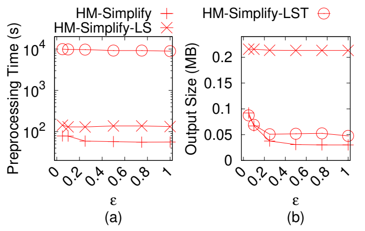


Figure 72: Ablation study for simplification algorithms on LM_m-*small* height map dataset

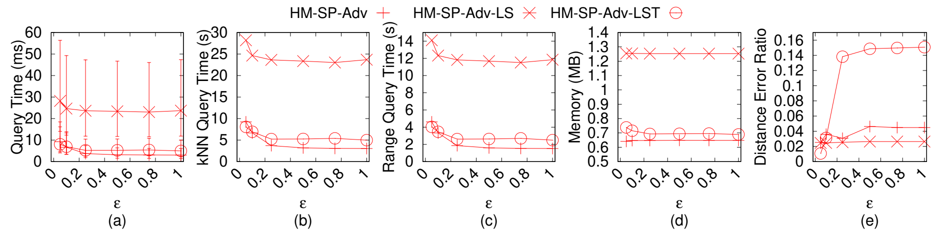


Figure 73: Ablation study for simplification algorithms with results for corresponding proximity query algorithms on LM_m-*small* height map dataset

D PROOF

Proof of Theorem 2.1. Based on the Height Map Simplification Problem in Problem , we first need to find the Height Map Simplification Decision Problem in Problem .

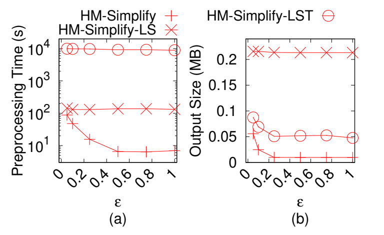


Figure 74: Ablation study for simplification algorithms on RM_m-small height map dataset

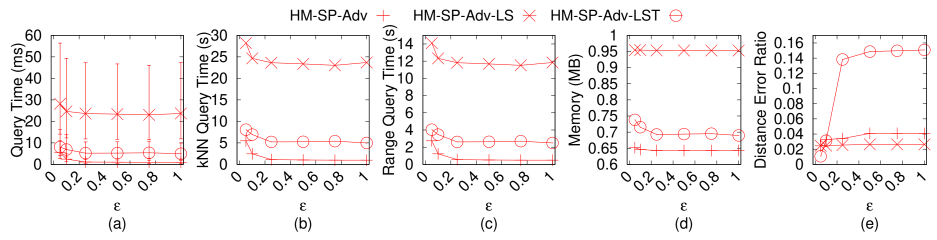


Figure 75: Ablation study for simplification algorithms with results for corresponding proximity query algorithms on RM_m-small height map dataset

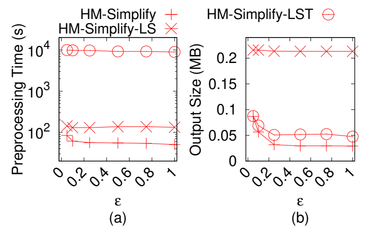


Figure 76: Ablation study for simplification algorithms on BH_m-small height map dataset

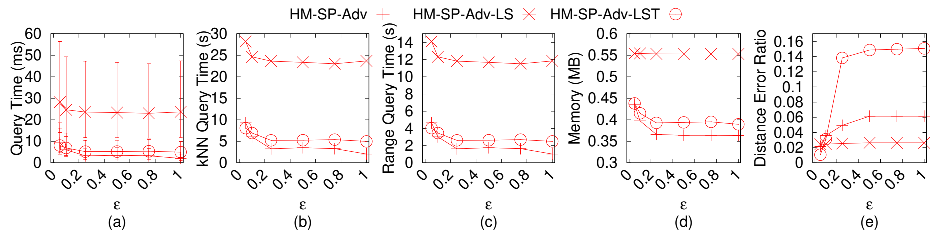


Figure 77: Ablation study for simplification algorithms with results for corresponding proximity query algorithms on BH_m-small height map dataset

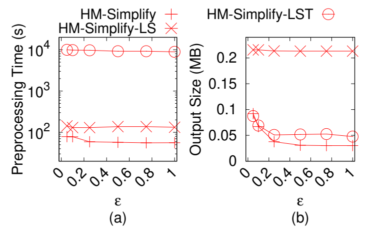


Figure 78: Ablation study for simplification algorithms on EP_m-small height map dataset

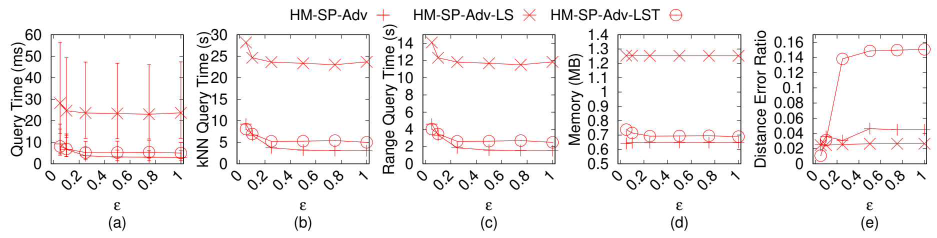


Figure 79: Ablation study for simplification algorithms with results for corresponding proximity query algorithms on EP_m-small height map dataset

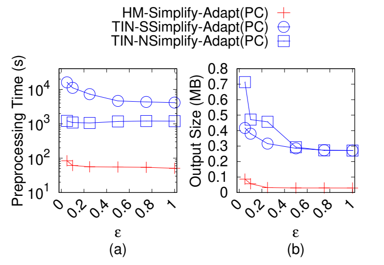


Figure 80: Effect of ϵ on GF_p-small point cloud dataset for simplification algorithms

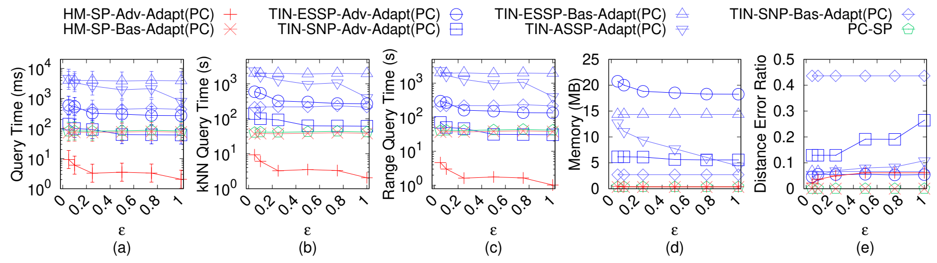


Figure 81: Effect of ϵ on GF_p-small point cloud dataset for proximity query algorithms

Problem 2 (Height Map Simplification Decision Problem). *Given M, a non-negative integer i and ϵ, we want to find an ϵ-approximate simplified height map $\widetilde{M}$ of M with at most i cells.*

Then, the proof is by transforming Minimum T-Spanner Problem [21] in Problem 3, which is an *NP-complete* problem, to the Height Map Simplification Decision Problem.

Problem 3 (Minimum T-Spanner Decision Problem). *Given a graph G_{NPC} with a set of vertices $G_{NPC}.V$ and a set of edges $G_{NPC}.E$,*

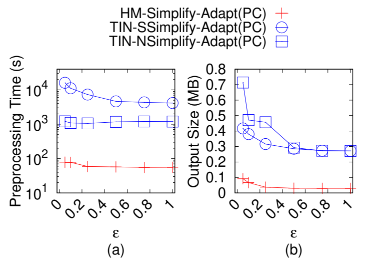


Figure 82: Effect of ϵ on *LM_p-small* point cloud dataset for simplification algorithms

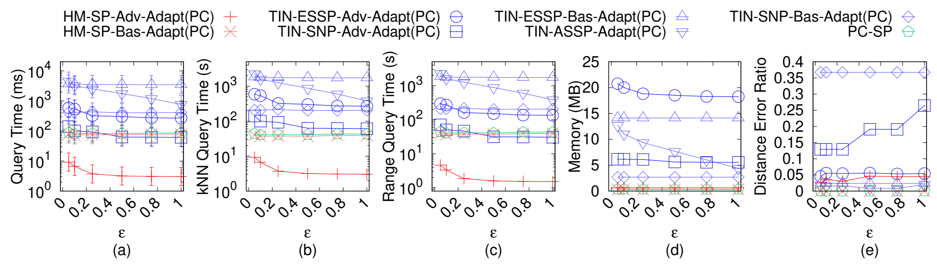


Figure 83: Effect of ϵ on *LM_p-small* point cloud dataset for proximity query algorithms

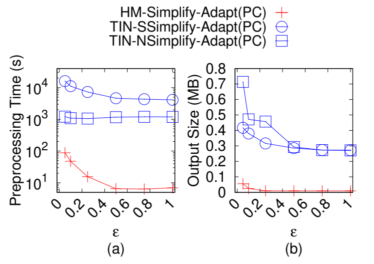


Figure 84: Effect of ϵ on *RM_p-small* point cloud dataset for simplification algorithms

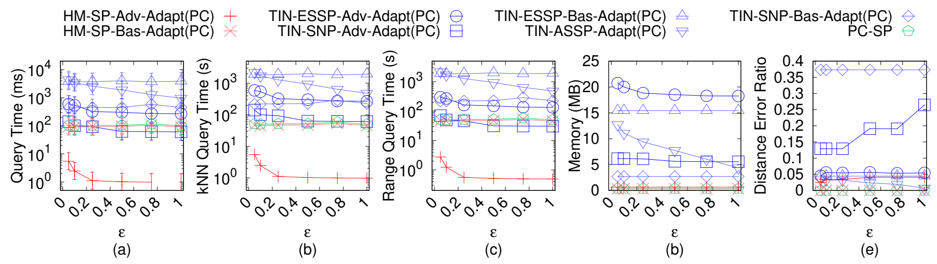


Figure 85: Effect of ϵ on *RM_p-small* point cloud dataset for proximity query algorithms

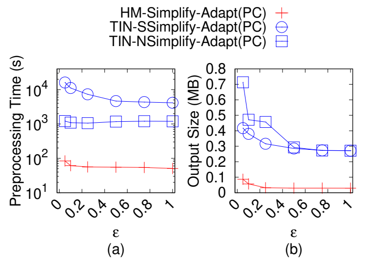


Figure 86: Effect of ϵ on *BH_p-small* point cloud dataset for simplification algorithms

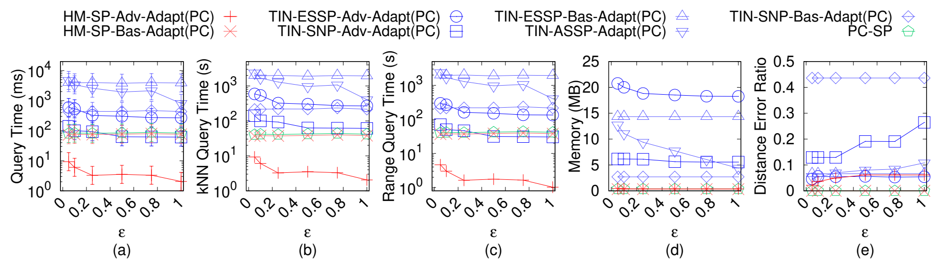


Figure 87: Effect of ϵ on *BH_p-small* point cloud dataset for proximity query algorithms

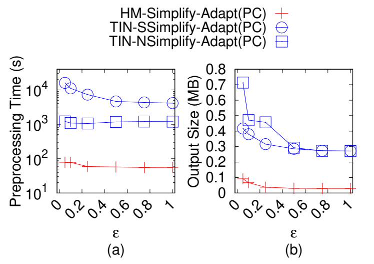


Figure 88: Effect of ϵ on *EP_p-small* point cloud dataset for simplification algorithms

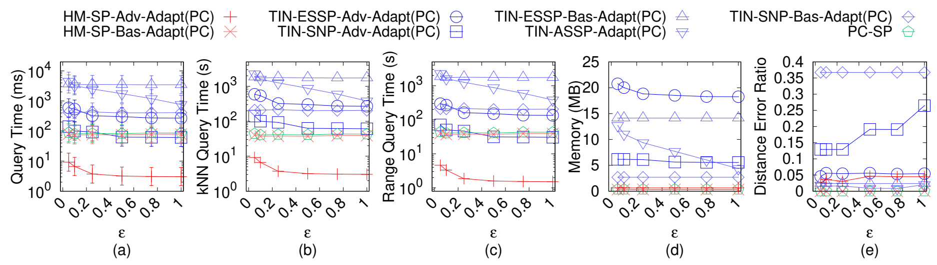


Figure 89: Effect of ϵ on *EP_p-small* point cloud dataset for proximity query algorithms

a non-negative integer j and an error parameter t, we want to 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 $G_{NPC}.V$, $|\Pi(s,t|\widetilde{G}_{NPC})| \leq (1+\epsilon)|\Pi(s,t|G_{NPC})|$,

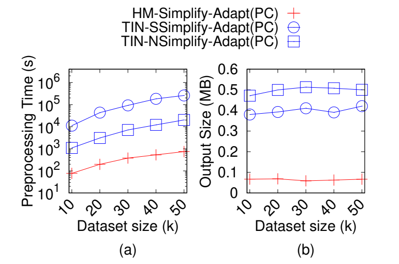


Figure 90: Effect of n on EP_p-small point cloud dataset for simplification algorithms

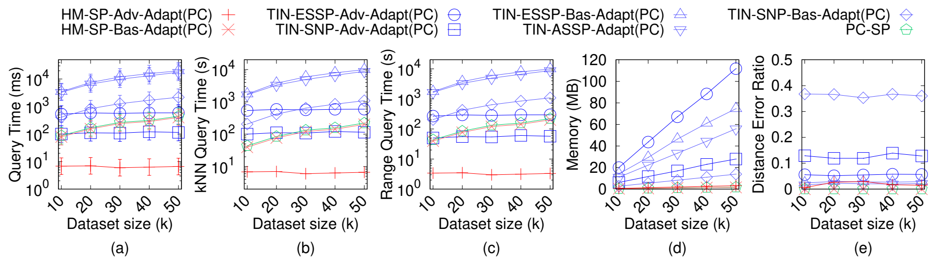


Figure 91: Effect of n on EP_p-small point cloud dataset for proximity query algorithms

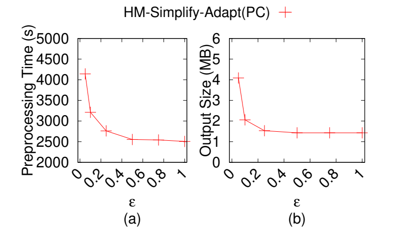


Figure 92: Effect of ϵ on GF_p point cloud dataset for simplification algorithms

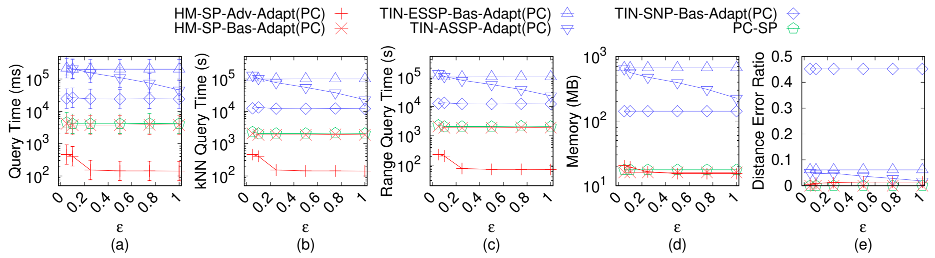


Figure 93: Effect of ϵ on GF_p point cloud dataset for proximity query algorithms

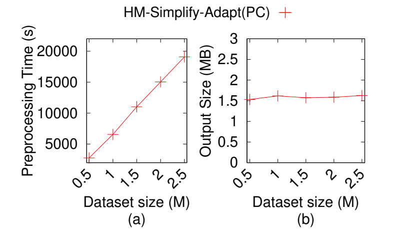


Figure 94: Effect of n on GF_p point cloud dataset for simplification algorithms

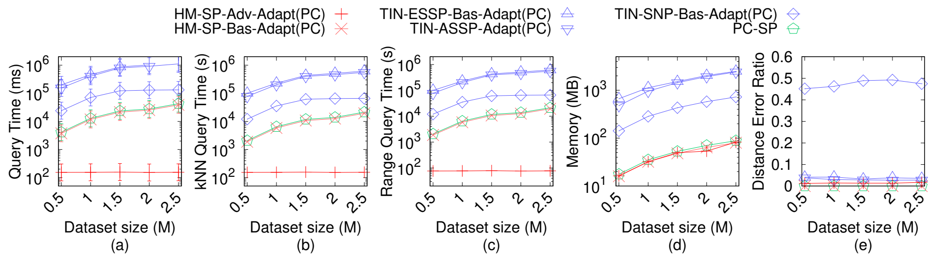


Figure 95: Effect of n on GF_p point cloud dataset for proximity query algorithms

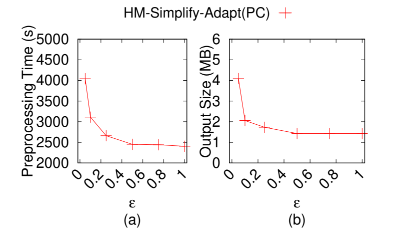


Figure 96: Effect of ϵ on LM_p point cloud dataset for simplification algorithms

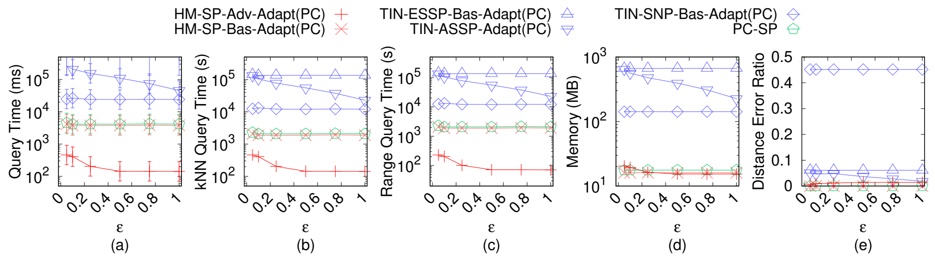


Figure 97: Effect of ϵ on LM_p point cloud dataset for proximity query algorithms

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 Height Map Graph Decision Simplification Problem in Problem 4. We transfer the Minimum T-Spanner Decision Problem to the Height Map Graph Simplification Decision Problem, and show that the Height Map Simplification Decision Problem is equivalent to the Height Map Graph Simplification Decision Problem.

PROBLEM 4 (HEIGHT MAP GRAPH SIMPLIFICATION DECISION PROBLEM). *Given a height map graph G of M, a non-negative integer i'*

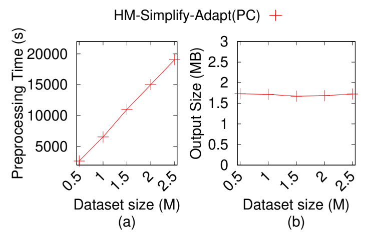


Figure 98: Effect of n on LM_p point cloud dataset for simplification algorithms

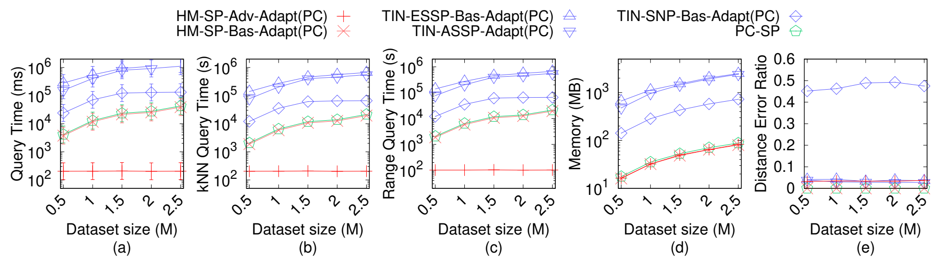


Figure 99: Effect of n on LM_p point cloud dataset for proximity query algorithms

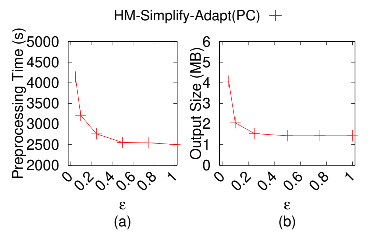


Figure 100: Effect of ϵ on RM_p point cloud dataset for simplification algorithms

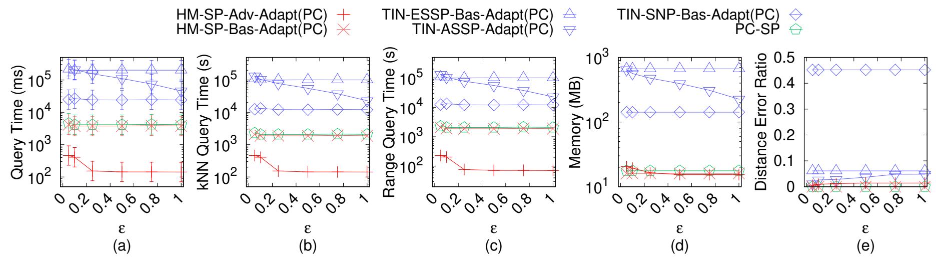


Figure 101: Effect of ϵ on RM_p point cloud dataset for proximity query algorithms

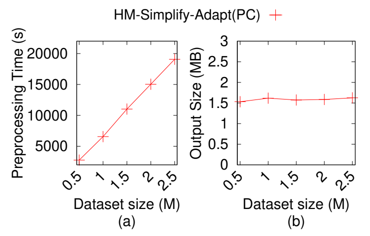


Figure 102: Effect of n on RM_p point cloud dataset for simplification algorithms

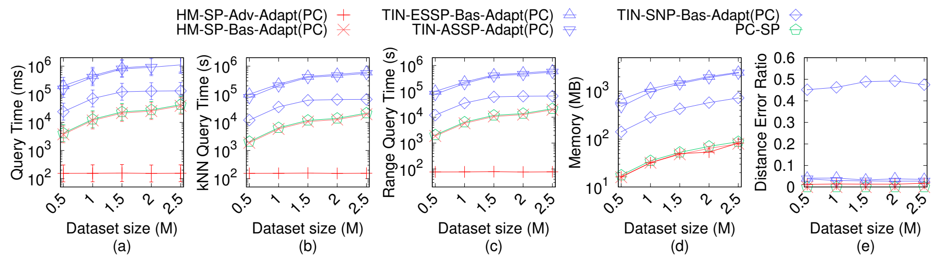


Figure 103: Effect of n on RM_p point cloud dataset for proximity query algorithms

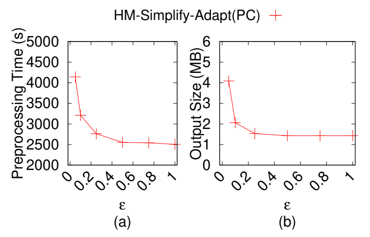


Figure 104: Effect of ϵ on BH_p point cloud dataset for simplification algorithms

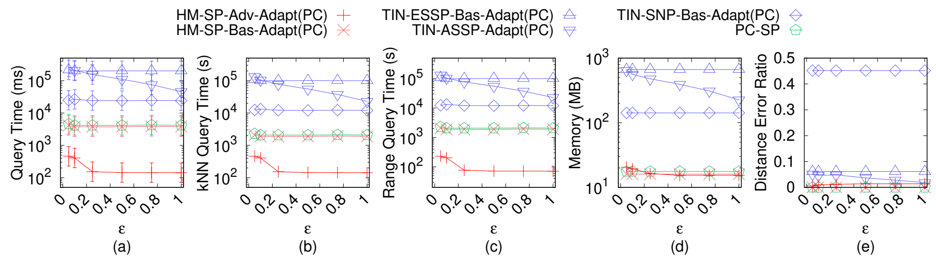


Figure 105: Effect of ϵ on BH_p point cloud dataset for proximity query algorithms

and an error parameter ϵ, we want to find a simplified height map 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 height map 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 cell centers of M) and all possible new vertices in $\widetilde{G}$ (i.e., all possible added cells in $\widetilde{M}$). Figure 144 (a) shows a height map, Figure 144 (b) shows the complete height map graph in a 2D horizontal plane. In Figure 144 (b), (1) each orange point represents 1 vertex with the same x-, y- and z-coordinate values of the corresponding vertex in the original height map graph G, and (2) each green point represents 256 vertices with the same

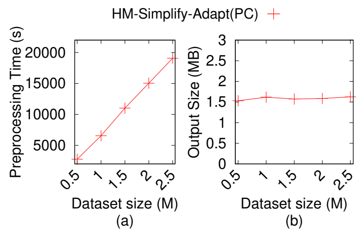


Figure 106: Effect of n on BH_p point cloud dataset for simplification algorithms

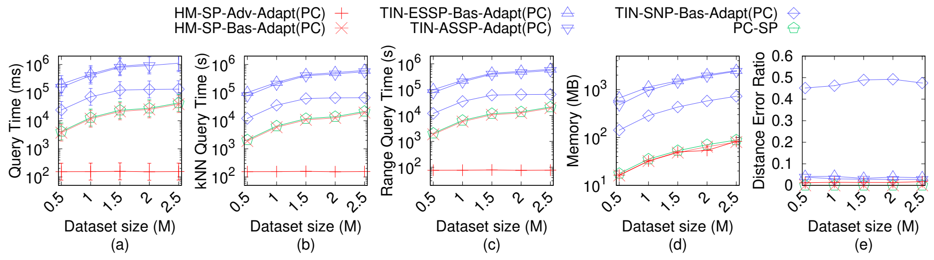


Figure 107: Effect of n on BH_p point cloud dataset for proximity query algorithms

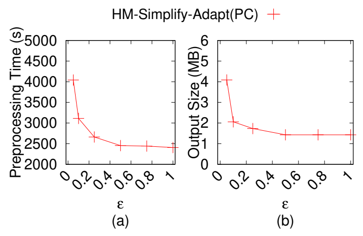


Figure 108: Effect of ϵ on EP_p point cloud dataset for simplification algorithms

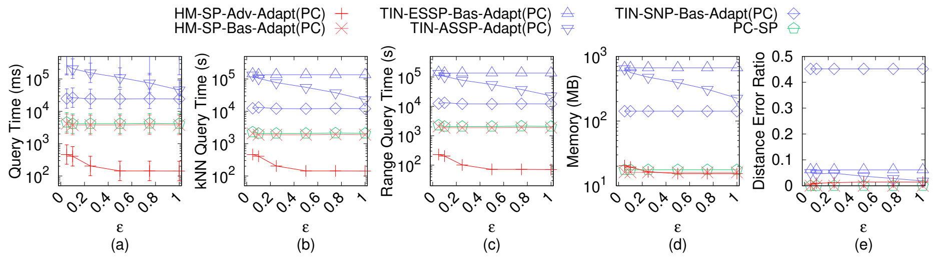


Figure 109: Effect of ϵ on EP_p point cloud dataset for proximity query algorithms

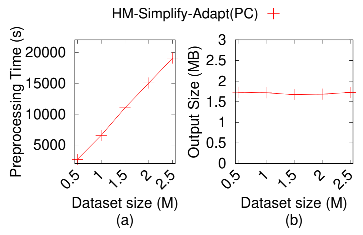


Figure 110: Effect of n on EP_p point cloud dataset for simplification algorithms

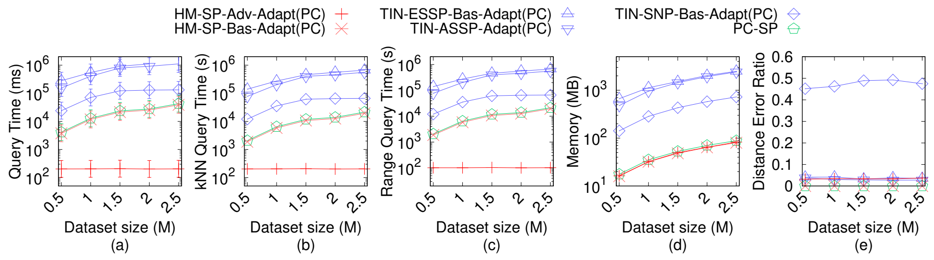


Figure 111: Effect of n on EP_p point cloud dataset for proximity query algorithms

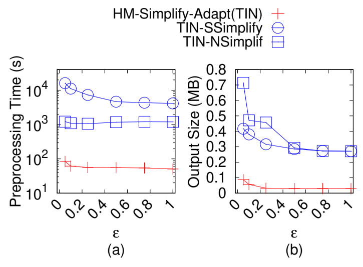


Figure 112: Effect of ϵ on GF_t-*small TIN* dataset for simplification algorithms

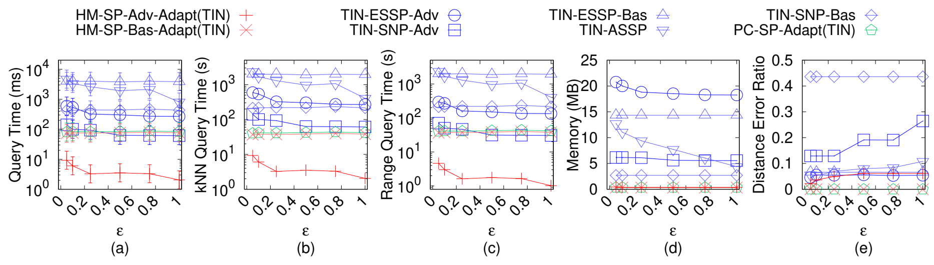


Figure 113: Effect of ϵ on GF_t-*small TIN* dataset for proximity query algorithms

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 cell 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

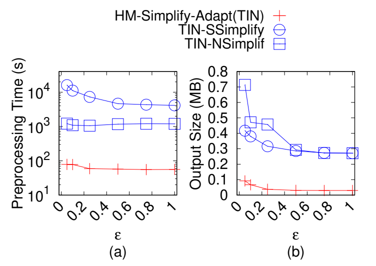


Figure 114: Effect of ϵ on *LM$_t$-small TIN* dataset for simplification algorithms

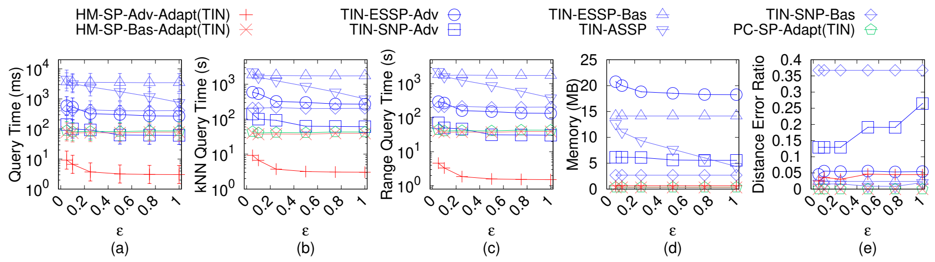


Figure 115: Effect of ϵ on *LM$_t$-small TIN* dataset for proximity query algorithms

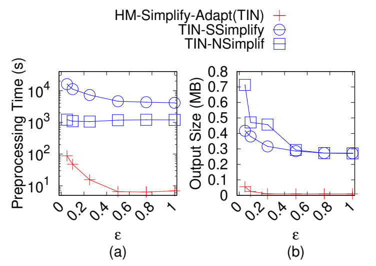


Figure 116: Effect of ϵ on *RM$_t$-small TIN* dataset for simplification algorithms

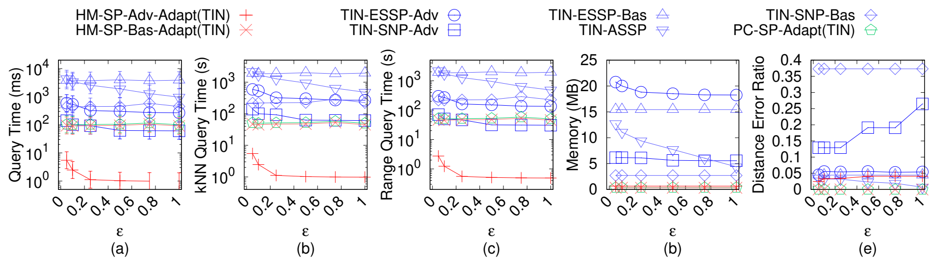


Figure 117: Effect of ϵ on *RM$_t$-small TIN* dataset for proximity query algorithms

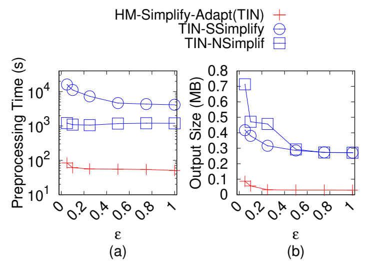


Figure 118: Effect of ϵ on *BH$_t$-small TIN* dataset for simplification algorithms

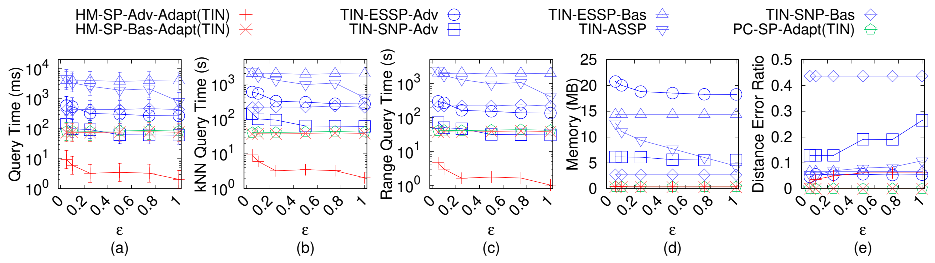


Figure 119: Effect of ϵ on *BH$_t$-small TIN* dataset for proximity query algorithms

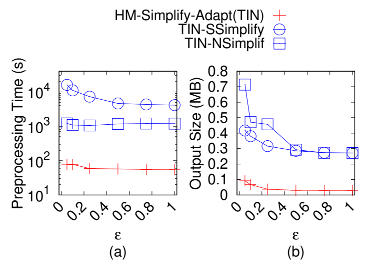


Figure 120: Effect of ϵ on *EP$_t$-small TIN* dataset for simplification algorithms

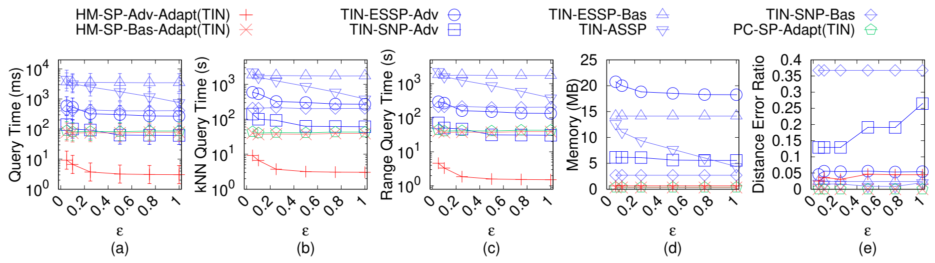


Figure 121: Effect of ϵ on *EP$_t$-small TIN* dataset for proximity query algorithms

pairs of vertices in $G_C.V$, and these edges form $G_C.E$. Figure 144 (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

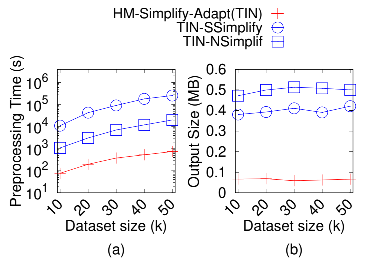


Figure 122: Effect of n on EP_t-small TIN dataset for simplification algorithms

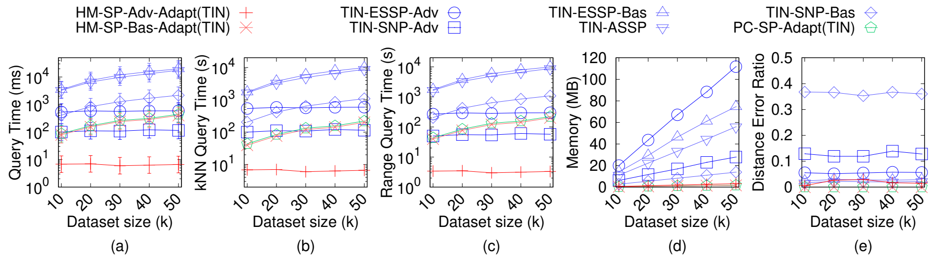


Figure 123: Effect of n on EP_t-small TIN dataset for proximity query algorithms

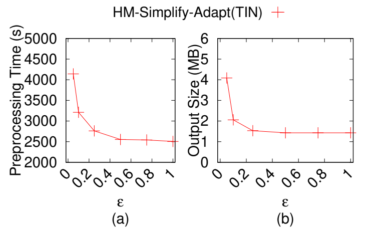


Figure 124: Effect of ϵ on GF_t TIN dataset for simplification algorithms

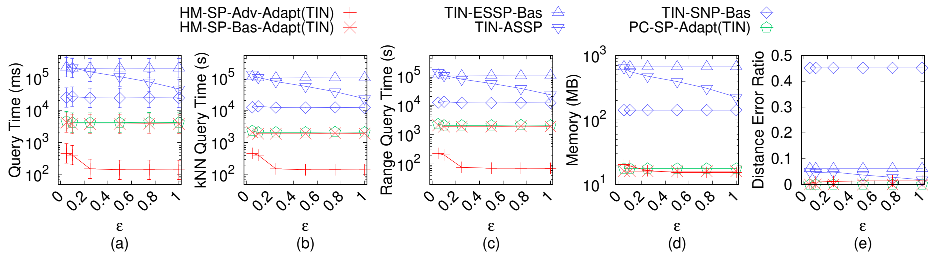


Figure 125: Effect of ϵ on GF_t TIN dataset for proximity query algorithms

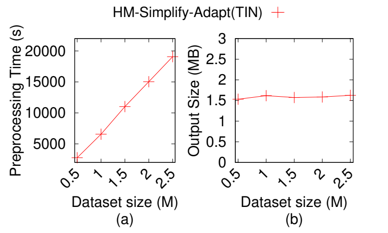


Figure 126: Effect of n on GF_t TIN dataset for simplification algorithms

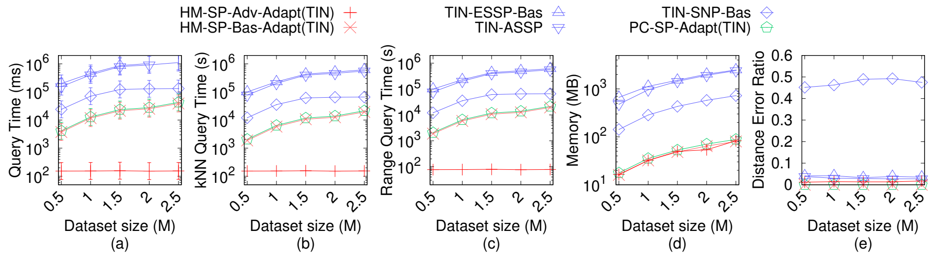


Figure 127: Effect of n on GF_t TIN dataset for proximity query algorithms

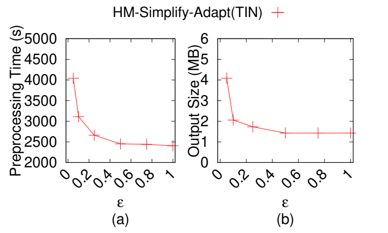


Figure 128: Effect of ϵ on LM_t TIN dataset for simplification algorithms

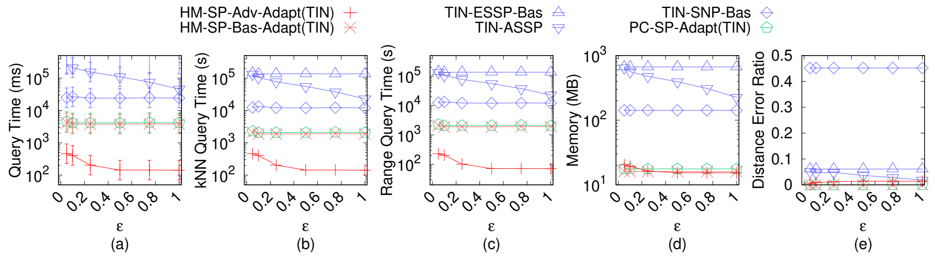


Figure 129: Effect of ϵ on LM_t TIN dataset for proximity query algorithms

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 height map 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*.

The transformation from the Minimum T-Spanner Decision Problem to the Height Map Graph Simplification Decision 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 height map graph $\widetilde{G}$ of G_C, with at most i' edges, such that for all pairs of vertices s and t in $G_C.V$, $(1-\epsilon)|\Pi(s,t|G)| = (1-\epsilon)|\Pi(s,t|G_C)| \leq |\Pi(s,t|\widetilde{G})| \leq$

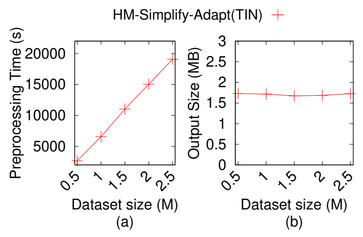


Figure 130: Effect of n on LM_t *TIN* dataset for simplification algorithms

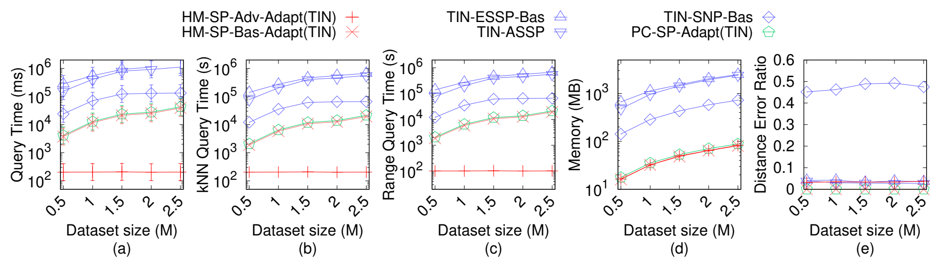


Figure 131: Effect of n on LM_t *TIN* dataset for proximity query algorithms

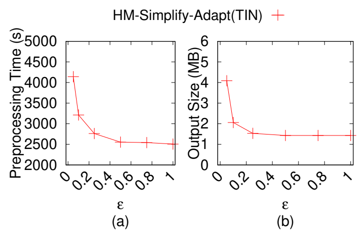


Figure 132: Effect of ϵ on RM_t *TIN* dataset for simplification algorithms

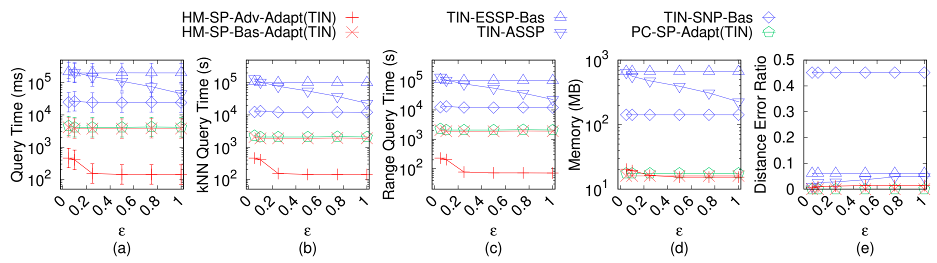


Figure 133: Effect of ϵ on RM_t *TIN* dataset for proximity query algorithms

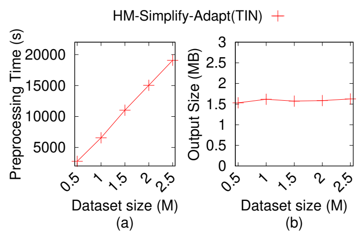


Figure 134: Effect of n on RM_t *TIN* dataset for simplification algorithms

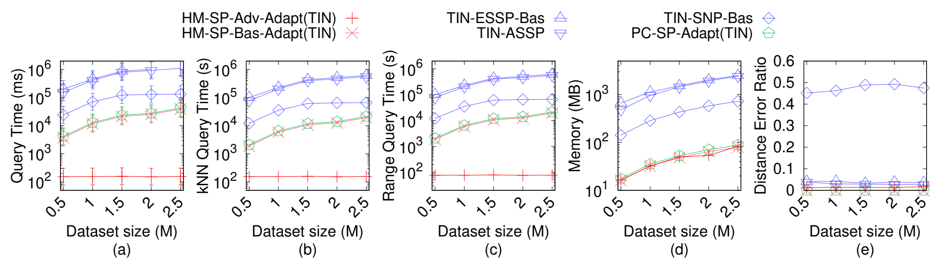


Figure 135: Effect of n on RM_t *TIN* dataset for proximity query algorithms

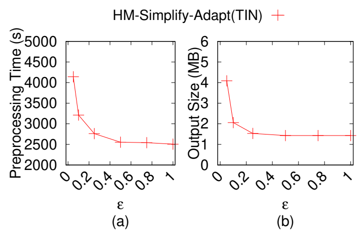


Figure 136: Effect of ϵ on BH_t *TIN* dataset for simplification algorithms

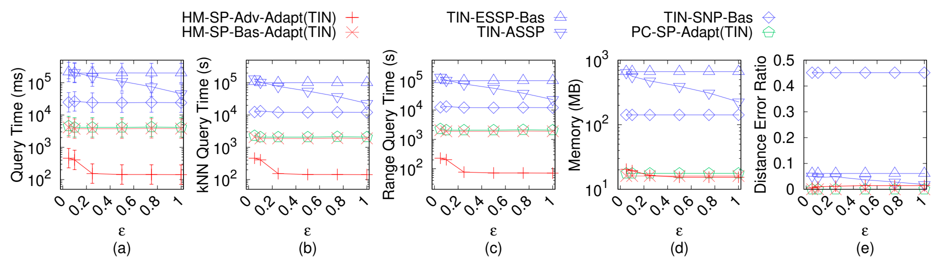


Figure 137: Effect of ϵ on BH_t *TIN* dataset for proximity query algorithms

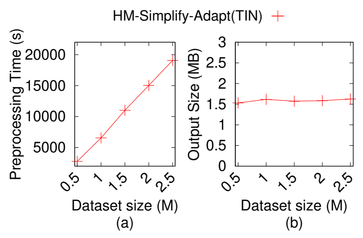


Figure 138: Effect of n on BH_t *TIN* dataset for simplification algorithms

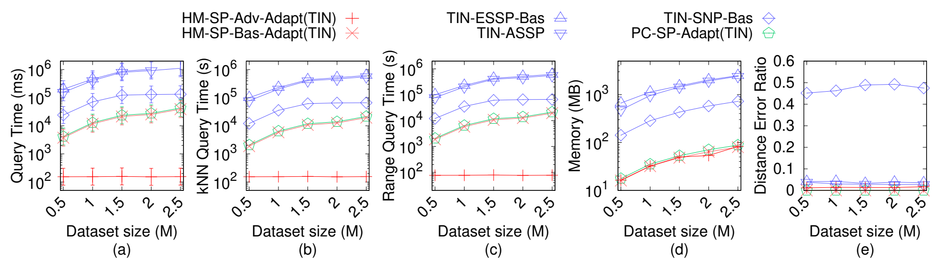


Figure 139: Effect of n on BH_t *TIN* dataset for proximity query algorithms

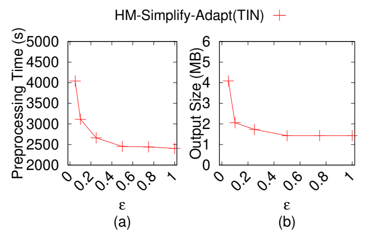


Figure 140: Effect of ϵ on EP_t *TIN* dataset for simplification algorithms

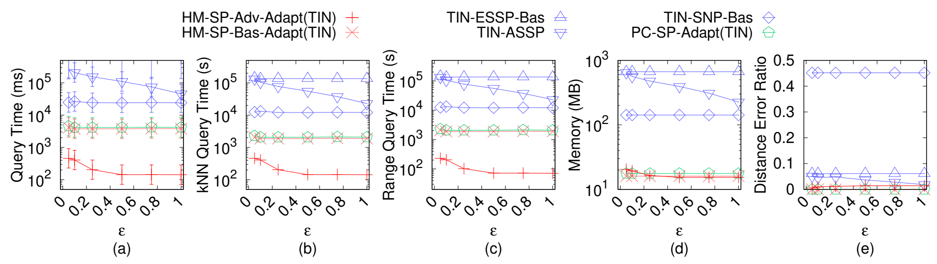


Figure 141: Effect of ϵ on EP_t *TIN* dataset for proximity query algorithms

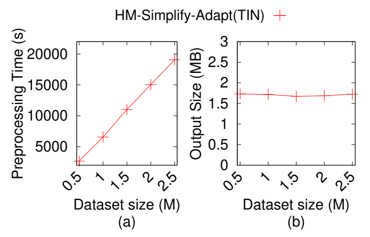


Figure 142: Effect of n on EP_t *TIN* dataset for simplification algorithms

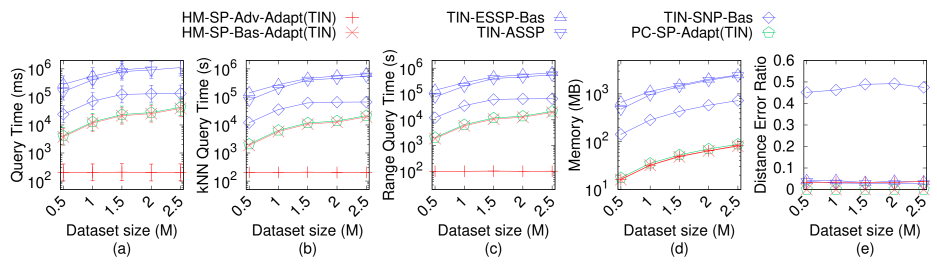


Figure 143: Effect of n on EP_t *TIN* dataset for proximity query algorithms

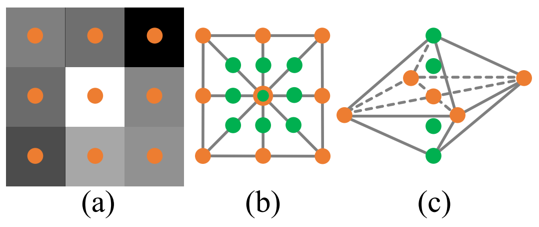


Figure 144: (a) A height map, (b) a complete height map graph in a 2D horizontal plane, and (c) a complete height map graph in a 3D space

$(1+\epsilon)|\Pi(s,t|G_C)| = (1+\epsilon)|\Pi(s,t|G)|$". Note that in the Height Map Graph Simplification Decision 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 height map graph G are equivalent, and i and i' can be any value, the Height Map Simplification Decision Problem is equivalent to the Height Map Graph Simplification Decision Problem. Thus, when the Height Map Graph Simplification Decision Problem is solved, the Height Map Simplification Decision Problem is solved equivalently, and the Minimum T-Spanner Decision Problem is also solved. Since the Minimum T-Spanner Decision Problem is *NP-complete*, the Height Map Simplification Decision Problem is *NP-hard*, and Height Map Simplification Problem is *NP-hard* □

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

Proof. We use mathematical induction to prove it. In algorithm *HM-Simplify*, even though it simplifies a height map using two different two simplification techniques, i.e., four adjacent cells merging and adjacent cells 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 C_{add} be the added cell 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 cell between $\Pi(s_1,t_1|\widetilde{M})$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, and they are remaining neighbor cells of linked added cells of C_{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 C_{rema}, and $\overline{p}$ and $\overline{q}$ are also in C_{rema}, and there is no difference between $\widetilde{M}$ and M (apart from the changes of C_{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 cell between $\Pi(s_1,t_1|M)$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, and they are remaining neighbor cells of linked added cells of C_{add}, we have $|\Pi(\overline{p}', \overline{q}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)|$ due to the *R2R* distance checking. Since s_1 and t_1 are in C_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in C_{rema}, and there is no difference between $\widetilde{M}$ and M (apart from the changes of C_{add}), we have $|\Pi(s_1, \overline{p}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, \overline{p}'|\widetilde{M})| = (1+\epsilon)|\Pi(s_1, \overline{p}'|M)|$ and $|\Pi(\overline{q}', t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{q}', t_1|\widetilde{M})| = (1+\epsilon)|\Pi(\overline{q}', t_1|M)|$. Thus, we have $(1+\epsilon)|\Pi(s_1, t_1|M)| = (1+\epsilon)|\Pi(s_1, \overline{p}'|M)| + (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)| + (1+\epsilon)|\Pi(\overline{q}', t_1|M)| \geq |\Pi(s_1, \overline{p}'|\widetilde{M})| + |\Pi(\overline{p}', \overline{q}'|\widetilde{M})| + |\Pi(\overline{q}', t_1|\widetilde{M})| \geq |\Pi(s_1, t_1|\widetilde{M})|$.

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

- Firstly, we show that $(1-\epsilon)|\Pi(s_1, t_1|M)| \leq |\Pi(s_1, t_1|\widetilde{M})|$. 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 cell between $\Pi(s_1, t_1|\widetilde{M})$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, and they are remaining neighbor cells of linked added cells of C_{add}, we have $(1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| \leq |\Pi(\overline{p}, \overline{q}|\widetilde{M})|$ due to the *R2R* distance checking. Since s_1 and t_1 are in C_{rema}, and $\overline{p}$ and $\overline{q}$ are also in C_{rema}, we have $(1-\epsilon)|\Pi(s_1, \overline{p}|M)| \leq |\Pi(s_1, \overline{p}|\widetilde{M})|$ and $(1-\epsilon)|\Pi(\overline{q}, t_1|M)| \leq |\Pi(\overline{q}, t_1|\widetilde{M})|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $|\Pi(s_1, t_1|\widetilde{M})| = |\Pi(s_1, \overline{p}|\widetilde{M})| + |\Pi(\overline{p}, \overline{q}|\widetilde{M})| + |\Pi(\overline{q}, t_1|\widetilde{M})| \geq (1-\epsilon)|\Pi(s_1, \overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q}, t_1|M)| \geq (1-\epsilon)|\Pi(s_1, t_1|M)|$.
- Secondly, we show that $|\Pi(s_1, t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, t_1|M)|$. 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 cell between $\Pi(s_1, t_1|M)$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, and they are remaining neighbor cells of linked added cells of C_{add}, we have $|\Pi(\overline{p}', \overline{q}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)|$ due to the *R2R* distance checking. Since s_1 and t_1 are in C_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in C_{rema}, we have $|\Pi(s_1, \overline{p}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, \overline{p}'|M)|$ and $|\Pi(\overline{q}', t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{q}', t_1|M)|$ due to the *R2R* distance checking after the i-th simplification iteration. Thus, we have $(1+\epsilon)|\Pi(s_1, t_1|M)| = (1+\epsilon)|\Pi(s_1, \overline{p}'|M)| + (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)| + (1+\epsilon)|\Pi(\overline{q}', t_1|M)| \geq |\Pi(s_1, \overline{p}'|\widetilde{M})| + |\Pi(\overline{p}', \overline{q}'|\widetilde{M})| + |\Pi(\overline{q}', t_1|\widetilde{M})| \geq |\Pi(s_1, t_1|\widetilde{M})|$.

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

Lemma D.2. *Given a height map M, algorithm HM-Simplify returns a simplified height map $\widetilde{M}$ of M, such that for all pairs of cells s_2 in C_{rema} and t_2 in $C - C_{rema}$,* $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$.

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 C_{add} be the added cell in this iteration. Since this is the first iteration, there are no other deleted cells except the cells belonging to C_{add}, we just need to show that the inequality holds when t_2 is any one of the deleted cells belong to linked added cells of C_{add}.

- Firstly, we show that $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})|$. Along $\Pi(s_2, t_2|\widetilde{M})$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}$ be the first intersection cell between $\Pi(s_2, t_2|\widetilde{M})$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, which is a remaining neighbor cell of linked added cells of C_{add}, and t_2 is in $C - C_{rema}$, we have $(1-\epsilon)|\Pi(\overline{m}, t_2|M)| \leq |\Pi(\overline{m}, t_2|\widetilde{M})|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}$ are both in C_{rema}, we have $(1-\epsilon)|\Pi(s_2, \overline{m}|M)| \leq |\Pi(s_2, \overline{m}|\widetilde{M})|$ from Lemma D.1. Thus, we have $|\Pi(s_2, t_2|\widetilde{M})| = |\Pi(s_2, \overline{m}|\widetilde{M})| + |\Pi(\overline{m}, t_2|\widetilde{M})| \geq (1-\epsilon)|\Pi(s_2, \overline{m}|M)| + (1-\epsilon)|\Pi(\overline{m}, t_2|M)| \geq (1-\epsilon)|\Pi(s_2, t_2|M)|$.
- Secondly, we show that $|\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$. Along $\Pi(s_2, t_2|M)$ from $\widetilde{t_2}$ to $\widetilde{s_2}$, let $\overline{m}'$ be the first intersection cell between $\Pi(s_2, t_2|M)$ and the remaining neighbor cells of linked added cells of C_{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 C_{rema}, which is a remaining neighbor cell of linked added cells of C_{add}, and t_2 is in $C - C_{rema}$, we have $|\Pi(\overline{m}, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{m}, t_2|M)|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}'$ are both in C_{rema}, we have $|\Pi(s_2, \overline{m}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, \overline{m}'|M)|$ from Lemma D.1. Thus, we have $(1+\epsilon)|\Pi(s_2, t_2|M)| = (1+\epsilon)|\Pi(s_2, \overline{m}'|M)| + (1+\epsilon)|\Pi(\overline{m}', t_2|M)| \geq |\Pi(s_2, \overline{m}'|\widetilde{M})| + |\Pi(\overline{m}', t_2|\widetilde{M})| \geq |\Pi(s_2, t_2|\widetilde{M})|$.

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

Thus, we have proved that for all pairs of cells s_2 in C_{rema} and t_2 in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$. □

Lemma D.3. *Given a height map M, algorithm HM-Simplify returns a simplified height map $\widetilde{M}$ of M, such that for all pairs of cells s_3 and t_3 in $C - C_{rema}$,* $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$.

Proof. 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 cells in C_{rema}. (b) $\Pi(s, t|\widetilde{M})$ passes on cells in C_{rema}.

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

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

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

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

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

- Firstly, we show that $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})|$. 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 cell between $\Pi(s_3, t_3|\widetilde{M})$ and the remaining neighbor cells of linked added cells of $O^{-1}(s_3)$ (resp. $O^{-1}(t_3)$). 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 C_{rema}, we have $(1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| \leq |\Pi_2(\overline{p}, \overline{q}|\widetilde{M})|$ by Lemma D.1. Since s_3 and t_3 are in $C - C_{rema}$, and $\overline{p}$ and $\overline{q}$ are in C_{rema}, we have $(1-\epsilon)|\Pi(s_3, \overline{p}|M)| \leq |\Pi_1(s_3, \overline{p}|\widetilde{M})|$ and $(1-\epsilon)|\Pi(\overline{q}, t_3|M)| \leq |\Pi_1(\overline{q}, t_3|\widetilde{M})|$ by Lemma D.2. Thus, we have $|\Pi(s_3, t_3|\widetilde{M})| = |\Pi_1(s_3, \overline{p}|\widetilde{M})| + |\Pi_2(\overline{p}, \overline{q}|\widetilde{M})| + |\Pi_1(\overline{q}, t_3|\widetilde{M})| \geq (1-\epsilon)|\Pi(s_3, \overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q}, t_3|M)| \geq (1-\epsilon)|\Pi(s_3, t_3|M)|$.
- Secondly, we show that $|\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$. 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 cell between $\Pi(s_3, t_3|M)$ and the remaining neighbor cells of linked added cells of $O^{-1}(s_3)$ (resp. $O^{-1}(t_3)$). 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 C_{rema}, we have $|\Pi_2(\overline{p}', \overline{q}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)|$ by Lemma D.1. Since s_3 and t_3 are in $C - C_{rema}$, and $\overline{p}'$ and $\overline{q}'$ are in C_{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 cells s_3 in C_{rema} and t_3 in $C - C_{rema}$, when $\Pi(s, t|\widetilde{M})$ passes on cells in C_{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 cells s_3 in C_{rema} and t_3 in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$. ◻

Proof of Theorem 4.1. Firstly, we prove the simplification time. In each simplification iteration of the *R2R*, *R2D* and *D2D* distance checking, since we only check the cells related to the neighbor cells of linked added cells of an added cell, there are $O(1)$ such cells. 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 cells merging and the adjacent cells any direction expanded merging, we always expand by one cell 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 cell can dominate $O(\log n)$ deleted cells on average. Since there are total n cells on M, we obtain that there are $O(\frac{n}{\log n})$ cells on $\widetilde{M}$.

Finally, we prove that $\widetilde{M}$ is an ϵ-approximate simplified height map of M. We need to show that for all pairs of cells 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 cells remaining case*, from Lemma D.1, we know that for all pairs of cells s_1 and t_1 both in C_{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 cell deleted and one cell remaining case*, from Lemma D.2, we know that for all pairs of cells s_2 in C_{rema} and t_2 in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$. (3) For the *both cells deleted case*, there are two more sub-cases: (i) $\Pi(s, t|\widetilde{M})$ does not pass on cells in C_{rema}, which contains a special case of *different and non-adjacent belonging cell in both cells deleted case* (i.e., $\Pi(s, t|\widetilde{M})$ only passes on added cell and other linked added cells of it), *different and adjacent belonging cell in both cells deleted case* and *same belonging cell in both cells deleted case.* (ii) $\Pi(s, t|\widetilde{M})$ passes on cells in C_{rema}, which contains *different and non-adjacent belonging cell in both cells deleted case.* From Lemma D.3, we know that for all pairs of cells s_3 and t_3 both in $C - C_{rema}$, $(1-\epsilon)|\Pi(s_3, t_3|M)| \leq |\Pi(s_3, t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3, t_3|M)|$ 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-SP* on M, given a query cell q, we just need to perform one Dijkstra's algorithm on M.
- For algorithm *HM-SP* on $\widetilde{M}$, given a query cell q, if q is a remaining cell, we just need to perform one Dijkstra's algorithm on $\widetilde{M}$; if q is a deleted cell, we just need to perform $\widetilde{N}(O^{-1}(q))$ Dijkstra's algorithm on $\widetilde{M}$. Since $\widetilde{N}(O^{-1}(q))$ 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-SP* on M and $\widetilde{M}$), respectively, the query time of both the *kNN* and range query by using algorithm *HM-SP* is $O(n \log n)$ on M and is $O(\frac{n}{\log n} \log \frac{n}{\log n})$ on $\widetilde{M}$.

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

- For algorithm *HM-SP* on M, 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-SP* on $\widetilde{M}$, 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-SP* on M 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-SP* on $\widetilde{M}$ 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 c_f (resp. c'_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-SP* on $\widetilde{M}$, 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 ratio of *kNN* and range queries is $\beta = \frac{|\Pi(i,p'_f|M)|}{|\Pi(i,p_f|M)|} - 1$. According to Theorem 4.2, 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 c_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 c'_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 4.2, 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-SP* on $\widetilde{M}$ has an error ratio $\frac{2\epsilon}{1-\epsilon}$ for the *kNN* and range query.

□

Theorem D.4. *The simplification time and output size of algorithm TIN-SSimplify-Adapt(HM) are $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$ and $O(n)$, respectively. Given a height map M, it first convert M to a TIN T, 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 convert the height map to a *TIN* 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 [28, 34] on each face adjacent to the deleted vertex, and use algorithm *TIN-ESSP* [22, 52, 59] 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 $O(n)$.

Finally, we prove that for all pairs of vertices s and t on T, algorithm *TIN-SSimplify-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 [34], given any pair of points c 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 4.1, 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-SSimplify-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 and output size of algorithm TIN-NSimplify-Adapt(HM) are $O(n^2 \log n)$ and $O(n)$, respectively. Given a height map M, it first converts M to a TIN T, 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 convert the height map to a *TIN* in $O(n)$ time. Then, in each vertex removal iteration, it uses algorithm *TIN-SNP* [36] in $O(n \log n)$ time to check the distances between any pair of vertices that are neighbors of the deleted vertex on the original *TIN* and the simplified *TIN*. Since there are only $O(1)$ vertices that are neighbors 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 $O(n)$.

Finally, we prove that for all pairs of vertices s and t on T, algorithm *TIN-NSimplify-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-NSimplify-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 [36]. □

THEOREM D.6. *The simplification time and output size of algorithm HM-Simplify-LQT1 are $O(n\sqrt[3]{n}\log n)$ and $O(\frac{n}{\log n})$, respectively. Given a height map M, it returns an ϵ-approximate simplified height map $\widetilde{M}$ of M.*

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

THEOREM D.7. *The simplification time, output size of algorithm HM-Simplify-LQT2 are $O(n\sqrt[3]{n}\log n)$ and $O(\frac{n}{\log n})$, respectively. Given a height map M, it returns an ϵ-approximate simplified height map $\widetilde{M}$ of M.*

PROOF. The simplification time, output size and error guarantee of algorithm *HM-Simplify-LQT2* are the same as algorithm *HM-Simplify.* □

THEOREM D.8. *The simplification time and output size of algorithm HM-Simplify-LS are $O(n\sqrt[3]{n}\log n)$ and $O(n)$, 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 cells in Section 4.2, although it could simplify a height map, our experimental results show the simplified height map still has $O(n)$ cells. Thus, the output size is $O(n)$.

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

THEOREM D.9. *The simplification time and output size of algorithm HM-Simplify-LST are $O(n^3\sqrt[3]{n}\log n)$ and $O(\frac{n}{\log n})$, 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 uses the naive checking technique that checks whether Inequality 1 is satisfied for all cells in Section 4.2, in each cell merging iteration, it needs to check the distance between all pairs of cells on $\widetilde{M}$ and M, i.e., run Dijkstra's algorithm in $O(n\log n)$ time for $O(n)$ cells, which needs $O(n^2\log n)$ time. According to Theorem 4.1, there are total $O(\sqrt[3]{N})$ cell merging iterations. So the total simplification time is $O(n^2\sqrt[3]{N}\log n)$.

The output size and error guarantee of algorithm *HM-Simplify-LST* are the same as algorithm *HM-Simplify.* □

THEOREM D.10. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-ESSP-Adapt(HM) are $O(n^2)$, $O(n^2)$ and $O(n^2)$ on a TIN T, and are $O(n^2)$, $O(n^2)$ and $O(n^2)$ on a simplified TIN $\widetilde{T}$, respectively. Compared with $\Pi(s,t|T)$, it returns the exact shortest surface path passing on a TIN (that is converted from the height map), and always has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi_{\textit{TIN-ESSP-Adapt(HM)}}(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ for all pairs of vertices s and t on T, where $\Pi_{\textit{TIN-ESSP-Adapt(HM)}}(s,t|\widetilde{T})$ is the estimated shortest surface path of algorithm TIN-ESSP-Adapt(HM) passing on a simplified TIN $\widetilde{T}$ (that is calculated by algorithm TIN-SSimplify-Adapt(HM)) between s and t. 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 on both T and $\widetilde{T}$. The proof of the shortest path query time of algorithm *TIN-ESSP* on a *TIN* with $O(n)$ vertices is in [22, 52, 59]. Since there are both $O(n)$ vertices on T and $\widetilde{T}$, the shortest path query time is $O(n^2)$. But, since algorithm *TIN-ESSP-Adapt(HM)* first needs to convert the height map to a *TIN*, 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 on both T and $\widetilde{T}$. 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 usage on both T and $\widetilde{T}$. The proof of the memory usage of algorithm *TIN-ESSP* is in [22, 52, 59], which is similar to algorithm *TIN-ESSP-Adapt(HM).* Thus, the memory usage 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 [22, 52, 59]. Since the *TIN* is converted from the height map, so algorithm *TIN-ESSP-Adapt(HM)* returns the exact shortest surface path passing on a *TIN* (that is converted from the height map). Since the simplified *TIN* is calculated by algorithm *TIN-SSimplify-Adapt(HM)*, so it has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi_{\textit{TIN-ESSP-Adapt(HM)}}(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$ for all pairs of vertices s and t on T. 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-ESSP-Adapt(HM)* returns the approximate shortest path passing on a height map. □

THEOREM D.11. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-ASSP-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 $|\Pi_{\textit{TIN-ASSP-Adapt(HM)}}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for all pairs of vertices s and t on T, where $\Pi_{\textit{TIN-ASSP-Adapt(HM)}}(s,t|T)$ is the shortest surface path of algorithm TIN-ASSP-Adapt(HM) passing on a TIN T (that is converted from the height map) between s and t. Compared with $\Pi(s,t|P)$, 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-ASSP* is in [35]. Note that in Section 4.2 of [35], the shortest path query time of algorithm *TIN-ASSP-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 [35], 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-ASSP-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-ASSP-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-ASSP-Adapt(HM)* first needs to convert the height map to a *TIN*, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time of algorithm *TIN-ASSP-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 [57], 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 usage. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory usage is $O(n)$.

Finally, we prove the error guarantee. Compared with $\Pi(s,t|T)$, the proof of the error guarantee of algorithm *TIN-ASSP-Adapt(HM)* is in [35, 57]. Since the *TIN* is converted from the point cloud, so algorithm *TIN-ASSP-Adapt(HM)* always has $|\Pi_{\textit{TIN-ASSP-Adapt(HM)}}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for all pairs of vertices s and t on T. Compared with $\Pi(s,t|P)$, since we regard $\Pi(s,t|P)$ as the exact shortest path passing on the point cloud, algorithm *TIN-ASSP-Adapt(HM)* returns the approximate shortest path passing on a point cloud. □

Theorem D.12. *The shortest path query time, kNN and range query time and memory usage of algorithm TIN-SNP-Adapt(HM) are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a TIN T, and are $O(n\log n)$, $O(n\log n)$ and $O(n)$ on a simplified TIN $\widetilde{T}$, respectively. Compared with $\Pi(s,t|T)$, it always has $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)| \leq \alpha \cdot |\Pi(s,t|T)|$ for all pairs of vertices s and t on T, where $\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)$ is the shortest network path of algorithm TIN-SNP-Adapt(HM) passing on a TIN T (that is converted from the height map) between s and t, $\alpha = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$, and returns the approximate shortest path passing on a simplified TIN $\widetilde{T}$ (that is calculated by algorithm TIN-NSimplify-Adapt(HM)). Compared with $\Pi(s,t|P)$, it returns the approximate shortest path passing on a height map.*

Proof. Firstly, we prove the shortest path query time on both T and $\widetilde{T}$. Since algorithm *TIN-SNP* only computes the shortest network path passing on T (that is converted from the height map with total n vertices) and $\widetilde{T}$ (that is calculated by algorithm *TIN-NSimplify-Adapt(HM)* total n vertices), it is a Dijkstra's algorithm, the shortest path query time is $O(n\log n)$. But since algorithm *TIN-SNP-Adapt(HM)* first needs to convert the height map to a *TIN*, 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 on both T and $\widetilde{T}$. 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 usage. Since it is a Dijkstra's algorithm and there are total n vertices on the *TIN*, the memory usage 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 converted from the height map) between s and t, so actually $\Pi_N(s,t|T)$ is the same as $\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)$. 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 [36]) 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 $|\Pi_{\textit{TIN-SNP-Adapt(HM)}}(s,t|T)| \leq \alpha \cdot |\Pi(s,t|T)|$ for all pairs of vertices s and t on T. Since the simplified *TIN* is calculated by algorithm *TIN-NSimplify-Adapt(HM)*, algorithm *TIN-SNP-Adapt(HM)* returns the approximate shortest path passing on a simplified *TIN*. Compared with $\Pi(s,t|P)$, since we regard $\Pi(s,t|P)$ as the exact shortest path passing on the height map, algorithm *TIN-SNP-Adapt(HM)* returns the approximate shortest path passing on a height map. □

Theorem D.13. *The shortest path query time, kNN and range query time and memory usage of algorithm PC-SP-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 converted from the height map).*

Proof. Firstly, we prove the shortest path query time. Since algorithm *PC-SP* computes the shortest path passing on P (that is converted from 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-SP-Adapt(HM)* first needs to convert the height map to a point cloud, 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 usage. Since it is a Dijkstra's algorithm and there are total n points on the point cloud, the memory usage is $O(n)$.

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

Theorem D.14. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-Adv-LQT1 are $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$, $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$ and $O(\frac{n}{\log n})$, respectively. It returns an approximate shortest path passing on 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 cell in $\widetilde{N}(O^{-1}(s))$

or $\widetilde{N}(O^{-1}(t))$ as a source to compute inter-path, and the size of $\widetilde{N}(O^{-1}(s))$ or $\widetilde{N}(O^{-1}(t))$ is $O(n)$, so its shortest path query time is $O(n)$ times the shortest path query time of algorithm *HM-SP-Adv*. 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-SP-Adv-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 memory usage and error guarantee of algorithm *HM-SP-Adv-LQT1* are the same as algorithm *HM-SP-Adv*. □

Theorem D.15. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-Adv-LQT2 are* $O(\frac{n}{\log n} \log \frac{n}{\log n})$ *and* $O(\frac{nn'}{\log n} \log \frac{n}{\log n})$ *and* $O(\frac{n}{\log n})$, *respectively. It returns an approximate shortest path passing on* 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-SP-Adv* 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 shortest path query time, memory usage and error guarantee of algorithm *HM-SP-Adv-LQT2* are the same as algorithm *HM-SP-Adv*. □

Theorem D.16. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-Adv-LS are* $O(n \log n)$ *and* $O(n \log n)$ *and* $O(n)$, *respectively. It returns an approximate shortest path passing on* ϵ*-approximate simplified height map* $\widetilde{M}$ *of* M.

Proof. Firstly, we prove the shortest path query time. Since it is applied on the simplified height map calculated by algorithm *HM-Simplify-LS* with $O(n)$ cells on $\widetilde{M}$, and we use Dijkstra's algorithm on $\widetilde{M}$ for once, the shortest path query time is $O(n \log n)$.

Secondly, 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 $O(n \log n)$.

Thirdly, we prove the memory usage. Since there are $O(n)$ cells on $\widetilde{M}$, the memory usage is $O(n)$.

The error guarantee of algorithm *HM-SP-Adv-LS* is the same as algorithm *HM-SP-Adv*. □

Theorem D.17. *The shortest path query time, kNN and range query time and memory usage of algorithm HM-SP-Adv-LST are* $O(\frac{n}{\log n} \log \frac{n}{\log n})$ *and* $O(\frac{n}{\log n} \log \frac{n}{\log n})$ *and* $O(\frac{n}{\log n})$, *respectively. It returns an approximate shortest path passing on* ϵ*-approximate simplified height map* $\widetilde{M}$ *of* M.

Proof. The shortest path query time, *kNN* and range query time, memory usage and error guarantee of algorithm *HM-SP-Adv-LST* are the same as algorithm *HM-SP-Adv*. □