Efficient Proximity Queries on Simplified Height Maps

Yinzhao Yan
The Hong Kong University of Science and Technology
yyanas@cse.ust.hk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
raywong@cse.ust.hk

ABSTRACT

Advancements in computer graphics technology lead to increased use of height maps, which offer advantages over point clouds and *Triangular Irregular Networks*, i.e., *TIN*s (representing terrain surfaces) in proximity queries, including the *shortest path query*, the *k-Nearest Neighbor* (*kNN*) *query*, and the *range query*. Since all existing shortest path query algorithms on height maps, point clouds, and *TIN*s are inefficient, and no algorithm can simplify height maps, we propose an efficient ϵ-approximate algorithm for simplifying a height map and answering the shortest path query on the simplified height map. We also propose ϵ-approximate algorithms for answering *kNN* and range queries on the simplified height map. Our experiments show that our algorithm is up to 412 times, 7 times, and 1,340 times better than the best-known *TIN* simplification algorithm (no point cloud simplification algorithm exists) in terms of the simplification time, output size, and shortest path query time, respectively. Performing the other two proximity queries on our simplified height map are both up to 153 times faster than the best-known algorithms on a point cloud, and 1,340 times faster than the best-known algorithms on a simplified *TIN*.

ACM Reference Format:
Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Efficient Proximity Queries on Simplified Height Maps. In *Proceedings of 2025 International Conference on Management of Data (SIGMOD '25)*. ACM, New York, NY, USA, 34 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Answering proximity queries, including the *shortest path query*, the *k-Nearest Neighbor* (*kNN*) *query* [42], and the *range query* [35], on a 3D surface has gained significant attention in both industry and academia [52]. These queries involve finding the shortest path between a source and a destination, or finding all the shortest paths from a query object to its k nearest objects or to all objects within a specified range. In industry, Google Earth [3] and Metaverse [8] employ shortest paths passing on 3D surfaces (such as Earth or virtual reality) to enhance user navigation. In academia, shortest path queries on 3D models have emerged as a prominent research area in the field of databases [17, 44–46, 49–52]. Various representations of the 3D surface are considered, such as a height map, a point cloud, and a *Triangular Irregular Networks*, i.e., *TIN* (representing a terrain surface). Proximity queries on a point cloud and a *TIN* have been studied in works [45, 48, 49, 52], yet it is a new area concerning height maps. For example, in bushfire fighting or earthquake rescue [26, 43], we can use satellites to capture height maps of the affected regions, and perform proximity queries for rescuing. We can also use satellites to obtain the elevation of the same regions and generate point clouds, and use the captured height maps or generated point clouds to obtain *TIN*s [52].

Height map, point cloud, and *TIN*: (1) A *height map* contains a set of *pixels* storing elevation values. Figure 1 (a) shows a satellite model of Mount Rainier [38] (a USA's national park) in an area of 20km × 20km, and Figure 1 (b) shows the corresponding height map with 63 pixels, where a lighter pixel color indicating a higher elevation value. Figure 1 (c) shows a *conceptual graph* of a height map, whose *vertices* correspond to the pixels of the height map, and *edges* are established between each vertex and its 8 neighboring vertices in the 2D plane (this graph is stored in the memory and is used for proximity queries). (2) A *point cloud* contains a set of 3D *points* in space. Figure 1 (d) is a point cloud of Mount Rainier with 63 points. The conceptual graph of a point cloud is the same as that of a height map in Figure 1 (c). (3) A *TIN* contains a set of triangles known as *faces*. Each face contains three *edges* connecting at three *vertices*. The gray surfaces in Figures 1 (e) and (f) are *TIN*s of Mount Rainier, which consist of vertices, edges, and faces.

Our focus lies on: (1) paths passing on (a conceptual graph of) a height map in Figure 1 (c), (2) paths passing on (a conceptual graph of) a point cloud in also Figure 1 (c), (3) *surface paths* [28] passing on (the faces of) a *TIN* in Figure 1 (e), and (4) *network paths* [28] passing on (the edges of a) a *TIN* in Figure 1 (f).

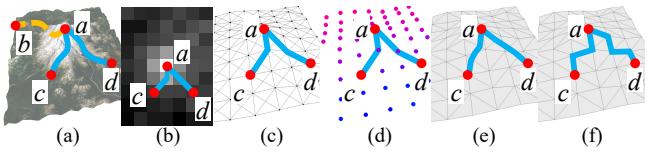

Figure 1: (a) A satellite model, (b) paths passing on a height map, (c) a conceptual graph of a height map or a point cloud, (d) paths passing on a point cloud, (e) surface and (f) network paths passing on a *TIN*

1.1 Motivation

1.1.1 ***Advantages of height map.*** Height maps have several advantages compared with point clouds, *TIN*s, and graphs.

(1) *Longer history and wider usage of height maps compared with point clouds.* Height maps were first utilized in 1884 [1], while point clouds were introduced in 1960 [6]. Consequently, more height map datasets are available compared to point cloud datasets, leading to broader application of the former. For example, height maps are

SIGMOD '25, June 1–6, 2025, New York, USA

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

commonly used to depict Earth's topography and mountainous regions [11], unlike point clouds.

(2) *Lower memory consumption of height maps compared with TINs and graphs.* We store (i) the pixel information for a height map, but store (ii) the vertex, edge, and face information for a *TIN*, and (iii) the vertex and edge information for a graph. Our experiments show that storing a height map with 25M pixels requires 75 MB, while storing a *TIN* and a graph generated from the same height map needs 1.7 GB and 980MB, respectively. The memory consumption of point clouds is similar to height maps.

(3) *More direct access to height maps compared with TINs and graphs.* We can use a satellite to capture the height map of a region in an area of 1km^2 in 10s [36]. But, to obtain a *TIN* or a graph, researchers usually need to convert a height map to a *TIN* or a graph [21, 33, 53]. Our experiments show that transforming a height map with 25M pixels to a *TIN* or a graph both takes 290s ≈ 4.8 min. We can use a similar process and time to acquire point clouds as we do to obtain height maps.

(4) *Faster query time of shortest paths passing on height maps compared with shortest surface paths passing on TINs.* Calculating the shortest path passing on a height map is faster than calculating the shortest surface path passing on a *TIN* generated from this height map, since a height map has a simpler structure than a *TIN*. Our experiments show that calculating the shortest path passing on a height map containing 0.5M pixels needs 3s, but calculating the shortest surface path passing on a *TIN* generated from the same height map needs 280s ≈ 4.6 min.

(5) *Smaller distance error of shortest paths passing on height maps compared with shortest network paths passing on TINs.* In Figures 1 (b) and (e), the shortest path passing on a height map exhibits similarities with the shortest surface path passing on a *TIN* (for the former path, each pixel connects with 8 neighbor pixels). But, in Figures 1 (e) and (f), the shortest surface and network paths passing on a *TIN* differ significantly (for the latter path, each vertex only connects with 6 neighbor vertices). Our experiments show that the length of the shortest path passing on a height map (resp. the shortest network path passing on a *TIN*) is 1.06 (resp. 1.45) times larger than that of the shortest surface path passing on a *TIN*.

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

While answering the shortest path query on a height map is fast, we can further increase query speed by simplifying the height map (the time taken to simplify the height map is called the *simplification time*, the space complexity of the simplified height map is called the *output size*, and the time taken to return the shortest path on the simplified height map is called the *shortest path query time*). Simplifying 3D models is widely used in computer games for faster rendering and in 3D network-based applications for faster processing [14, 18]. We formulate *height map simplification problem*, which describes that given a height map and an *error parameter* $\epsilon \geq 0$, we aim to find an ϵ-approximate simplified height map with a minimum number of pixels, such that the shortest distances between any pair of pixels on the simplified height map is an ϵ-approximate of the distances on the original height map.

1.1.3 ***Example.***

We conduct a case study on an evacuation simulation in Mount Rainier because of snowfall [39]. In Figure 1 (a), we aim to find shortest paths from the position a of some tourists on the mountain to k-nearest hotels b to d, due to each hotel's limited capacity. In Figures 1 (b) to (f), when $k = 2$, c and d are k-nearest hotels. Our experiments show that given a height map with 50k pixels with 10k different tourist positions and 50 hotels, we can simplify it in 250s ≈ 4.6 min, and calculate the shortest paths from each tourist position to its k nearest hotels in 50s on the simplified height map. But, it needs 7,590s ≈ 2.1 hours on the original height map, 7,630s ≈ 2.1 hours on a point cloud (constructed based on the original height map), and 380,000s ≈ 4.3 days on a *TIN* (constructed based on the original height map). Since 250s + 50s = 300s < 7,590s, it is useful to use the simplification process.

1.2 Challenges

1.2.1 ***Inefficiency of the shortest path query on height maps.***

Existing algorithms [23, 37] for conducting proximity queries on a height map are slow, since they (1) first construct a point cloud or a *TIN* using the given height map in $O(n)$ time, where n is the number of pixels in the height map, and (2) then calculate the shortest path passing on this point cloud or *TIN*. The best-known *exact* algorithm [52] for calculating the shortest path passing on a point cloud runs in $O(n \log n)$ time. The best-known *exact* [13, 46] and *approximate* [27, 51] algorithm for calculating the shortest surface path passing on a *TIN* run in $O(n^2)$ and $O((n + n') \log(n + n'))$ time, respectively, where n' is the number of additional points introduced for bound guarantee. The best-known *approximate* [28] algorithm for calculating the shortest network path passing on a *TIN* runs in $O(n \log n)$ time. Our experiments show (1) algorithm [52] needs 7,630s ≈ 2.1 hours, (2) algorithm [13, 46] needs 380,000s ≈ 4.3 days, (3) algorithm [27, 51] needs 70,000s ≈ 19.4 hours, and (4) algorithm [28] needs 33,000s ≈ 9.2 hours to perform the *kNN* query for 10k objects on a height map with 50k pixels.

1.2.2 ***Non-existence of height map simplification algorithm.***

No existing algorithm can simplify a height map. Although algorithms [18, 24, 28, 31] can simplify a *TIN*, they cannot be used for simplifying a height map directly, since they iteratively remove a vertex v in the *TIN* and use *triangulation* [33] to fill the hole formed by the adjacent vertices of v, where triangulation does not apply to a height map. We can adapt these algorithms for height map simplification by constructing a *TIN* using the height map, and utilizing these algorithms on this *TIN*. But, the output sizes of their simplified *TIN*s are large since they do not consider any optimization techniques, resulting in large shortest path query times on their simplified *TIN*s. In addition, their simplification times are large due to the lengthy shortest path query time on a *TIN* and the large number of distance calculations required during simplification. Our experiments show that after constructing a *TIN* from the given height map, the best-known *TIN* simplification algorithm [25, 28] requires 103,000s ≈ 1.2 days to simplify a *TIN* with 50k vertices, and the *kNN* query time of 10k objects on the simplified *TIN* is 67,000s ≈ 18.6 hours. No existing algorithm can simplify a point cloud.

1.3 Contribution and Organization

We summarize our major contributions as follows.

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

(2) We propose the first ϵ-approximate height map simplification algorithm *Memory saving height map Simplification and simplified height map shortest path Query* (*Mem-SimQue*), that first generates a simplified height map, and then answers the shortest path query on it. It achieves state-of-the-art performance in terms of (i) the output size and shortest path query time by the memory saving technique, since it can significantly reduce the number of pixels in the simplified height map to save memory and to improve the efficiency of shortest path queries on the simplified height, and (ii) the simplification time by a newly proposed height map shortest path query algorithm *Efficient height map shortest path Query* (*Eff-Que*) and the pruning of unnecessary checks. We also design efficient algorithms for answering *kNN* and range queries on the simplified height map.

(3) We provide theoretical analysis on (i) the simplification time, output size, shortest path query time, and error bound of algorithm *Mem-SimQue*, (ii) the shortest path query time, memory usage, and error bound of algorithm *Eff-Que*, (iii) the *kNN* query time, range query time, and error bound for other proximity queries, (iv) the distance relationships among the shortest paths passing on a height map, a point cloud, and a *TIN*.

(4) Algorithm *Mem-SimQue* outperforms the best-known *TIN* simplification algorithm [25, 28] in terms of the simplification time, output size, and shortest path query time. The proximity query time on the simplified height map also performs much better than the best-known algorithm on a point cloud [52] and a simplified *TIN* [28, 45]. Our experiments show that given a height map with 50k pixels, the simplification time and output size are 250s $\approx$ 4.6 min and 0.07MB for algorithm *Mem-SimQue*, but are 103,000s $\approx$ 1.2 days and 0.5MB for the best-known *TIN* simplification algorithm [25, 28] (by constructing a *TIN* using the given height map). The *kNN* and range query time of 10k objects on the simplified height map are both 50s for *Mem-SimQue*, but are 7,630s $\approx$ 2.1 hours for the best-known algorithm on a point cloud [52], and 67,000s $\approx$ 18.6 hours for the best-known algorithm on a simplified *TIN* [28, 45].

The remainder of the paper is organized as follows. Section 2 provides the problem definition. Section 3 covers the related work. Section 4 presents our algorithms. Section 5 presents 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 $M = (P, N(\bullet))$ contains a set of *pixels* denoted as P, and a *neighbour pixels* table denoted as $N(\bullet)$ (i.e., a *hash table* [16]). For each pixel $p \in P$, $N(p)$ returns the *neighbour pixels* of p in $O(1)$ time, and $N(p)$ is initialized to be the closest top, bottom, left, right, top-left, top-right, bottom-left, and bottom-right pixels of p on M. Let n be the number of pixels of M. Each pixel $p \in P$ has two coordinate values (represent its x- and y-coordinate values) and a grayscale color integer value (represents its *elevation value* with range $[0, 255]$), denoted as $p.x$, $p.y$, and $p.z$, respectively. We use a point position at the center of a pixel to denote this pixel. In Figure 2 (a), 9 pixels represent a height map M, 6 orange points and 2 red points denote pixels in $N(p)$.

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

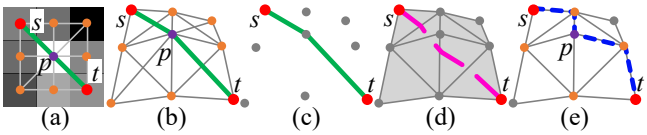

Figure 2: (a) $\Pi(s, t|M)$ passing on a height map M, (b) $\Pi(s, t|M)$ passing on a conceptual graph of M, (c) $\Pi(s, t|C)$ passing on a point cloud C, (d) $\Pi(s, t|T)$ passing on a *TIN* T, and (e) $\Pi_N(s, t|T)$ passing on a conceptual graph of T

2.1.2 ***Point cloud and TIN.*** Let C be a point cloud that contains a set of points in $G.V$, and let the conceptual graph of C also be G. Given a pair of points s and t on C, let $\Pi(s, t|C)$ be the *shortest path* between s and t passing on (the conceptual graph G of) C. Let T be a *TIN* triangulated [21, 33, 53] by the vertices in $G.V$. Similar to G, let G' be a conceptual graph of T. Let $G'.V$ and $G'.E$ be the set of vertices and edges of G', where $G'.V$ consists of the set of vertices of T and $G'.E$ consists of the set of edges of T, respectively. Given a pair of points s and t in $G'.V$, let $\Pi(s, t|T)$ be the *shortest surface path* between s and t passing on T and $\Pi_N(s, t|T)$ be the *shortest network path* between s and t passing on (the conceptual graph G' of) T. The distance of the shortest surface (resp. network) path is called the shortest surface (resp. network) distance. Let θ be the minimum interior angle of a triangle in T. Figures 2 (c), (d), and (e) show a point cloud constructed by M, a *TIN* constructed by M, and a conceptual graph of this *TIN*. The green, pink, and blue lines in these three figures show $\Pi(s, t|C)$, $\Pi(s, t|T)$, and $\Pi_N(s, t|T)$, respectively.

2.1.3 ***Simplified height map.*** Given a height map M, we can obtain a simplified height map $\widetilde{M} = (\widetilde{P}, \widetilde{N}(\bullet))$ by merging some adjacent pixels (deleting these pixels and adding a larger pixel that cover these pixels for replacement) in M, where $\widetilde{P}$ and $\widetilde{N}(\bullet)$ have the similar meaning as of P and $N(\bullet)$. Figures 3 (a) and (b & c) are original and simplified height maps. Let a *deleted* (resp. *remaining*) pixel be a pixel in M that is deleted from (resp. remaining in) $\widetilde{M}$. Let an *added* pixel be a pixel in $\widetilde{M}$ that covers some adjacent deleted and (or) previously added pixels, where these adjacent pixels *belong* to the added pixel. In Figure 3 (b), we merge $\{a, b, c, d\}$ to be e, 12 orange points denote pixels in $\widetilde{N}(e)$, $\{a, b, c, d\}$ are deleted pixels, all other pixels in P except $\{a, b, c, d\}$ are remaining pixels, e is an added pixel, and $\{a, b, c, d\}$ belong to e. The coordinate and elevation values of the added pixel are weighted average values of that of the adjacent pixels (if these adjacent pixels contain a previously added

pixel, the weight is the number of pixels in M that belongs to this previously added pixel; otherwise, the weight is 1). In Figures 3 (c), we merge $\{e, f, g, \dots\}$ to be h, the weight of e is 4 when calculating the coordinate and elevation values of h, since the number of pixels in M that belongs to e is 4. In Figures 3 (d) - (f), we keep simplifying the height map. We denote a set of remaining pixels and added pixels as P_{rema} and P_{add}, so $\widetilde{P} = P_{rema} \cup P_{add}$. A set of deleted pixels is denoted as $P - P_{rema}$.

Similar to G, let $\widetilde{G}$ be a simplified conceptual graph of $\widetilde{M}$. Let $\widetilde{G}.V$ and $\widetilde{G}.E$ be the set of vertices and edges of $\widetilde{G}$, respectively. We just need to use $\widetilde{P}$ and $\widetilde{N}(\bullet)$ to substitute P and $N(\bullet)$ 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 conceptual graphs of the corresponding height maps. Given a pair of pixels s and t on M (i.e., a pair of vertices s and t in $G.V$), let $\Pi(s, t|\widetilde{M})$ be the *shortest path* between s and t passing on (the simplified conceptual graph $\widetilde{G}$ of) $\widetilde{M}$. Figure 3 (g) shows an example of $\Pi(i, k|\widetilde{M})$ in blue line. A notation table can be found in the appendix of Table 3.

2.2 Problem

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

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

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

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

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

Proof Sketch. We can transform Minimum T-Spanner Problem [12] (*NP-complete* problem) to the Height Map Simplification Problem in polynomial time, to prove it is *NP-hard*. The detailed proof appears in the appendix. □

3 RELATED WORK

3.1 Height Map Shortest Path Query Algorithms

Existing algorithms [23, 37] for conducting proximity queries on a height map are slow. Given a height map, they first use it to construct a point cloud or a *TIN* in $O(n)$ time, and then calculate the shortest path passing on this point cloud or *TIN*. No algorithm can directly answer proximity queries on a height map. We introduce algorithms to calculate (1) the *shortest path* passing on a *point cloud* [52], (2) the *shortest surface path* [13, 27, 30, 34, 46, 47, 51] passing on a *TIN*, and (3) the *shortest network path* [28] passing on a *TIN*.

3.1.1 ***Shortest path query algorithm on a point cloud.*** The best-known exact point cloud shortest path query algorithm *Conceptual graph point cloud shortest path Query* (*Con-Que*) [52] uses Dijkstra's algorithm on the conceptual graph of the point cloud to calculate the path in $O(n \log n)$ time.

3.1.2 ***Shortest surface path query algorithms on a TIN.*** (1) *Exact algorithms*: Algorithm [34] and algorithm [47] use continuous Dijkstra's algorithm and checking window algorithm to calculate the path in both $O(n^2 \log n)$ time, respectively. The best-known exact *TIN* shortest surface path query algorithm *Unfold TIN shortest path Query* (*Unf-Que*) [13, 46] unfolds the 3D *TIN* into a 2D *TIN*, and then uses a line segment to connect the source and destination on this 2D *TIN* to calculate the path in $O(n^2)$ time. *Unf-Que* has two variants with the same time complexity: an initial version [13] and an extended version [46] with a better experimental running time.

(2) *Approximate algorithms*: All algorithms [27, 30, 51] place discrete Steiner points on edges of a *TIN*, and construct a graph using these Steiner points and the original vertices to calculate the path. The best-known $(1+\epsilon)$-approximate *TIN* shortest surface path query algorithm *efficient Steiner point TIN shortest path Query* (*Ste-Que*) [27, 51] runs in $O(\frac{l_{max}\boldsymbol{n}}{\epsilon l_{min}\sqrt{1-\cos\theta}} \log(\frac{l_{max}\boldsymbol{n}}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$, where l_{max} and l_{min} are the length of the longest and shortest edge of the *TIN*, respectively. Algorithm [27] and algorithm [51] run on an unweighted and weighted *TIN*, respectively. They are the same if we set the weight of each face on *TIN* in algorithm [51] to be 1, so we regard them as one algorithm.

3.1.3 ***Shortest network path query algorithm on a TIN.*** The shortest network path does not traverse the faces of a *TIN*, so it is an approximate path. The best-known approximate *TIN* shortest network path query algorithm *Dijkstra TIN shortest path Query*, i.e., algorithm *Dij-Que* [28] runs in $O(n \log n)$ time.

Adaptions: (1) Given a *height Map*, we adapt algorithm (i) *Con-Que* [52], (ii) *Unf-Que* [13, 46], (iii) *Ste-Que* [27, 51], and (iv) *Dij-Que* [28] to be algorithm (i) *Con-Que-AdpM*, (ii) *Unf-Que-AdpM*, (iii) *Ste-Que-AdpM*, and (iv) *Dij-Que-AdpM*, by first constructing a point cloud or *TIN* using the height map, and then computing the corresponding shortest path passing on the point cloud or *TIN*. (2) Given a height map, algorithm *Unf-Que* cannot be directly adapted on the height map since no face can be unfolded in a height map. But, algorithm *Con-Que*, *Ste-Que*, and *Dij-Que* can be directly adapted on the height map (and they become algorithm *Eff-Que*) by forcing the path to pass on the conceptual graph of the height map instead of constructing a point cloud or *TIN* first.

Drawback: All existing algorithms are very slow, although we pre-process the height map and store the generated point cloud or *TIN* in the memory and clear the given height map from the memory. Our experiments show that algorithm *Con-Que-AdpM* first needs to convert a height map with 50k pixels to a point cloud in 0.3s, and then perform the *kNN* query for all 10k objects on this point cloud in 7,630s ≈ 2.1 hours. In the same setting, algorithm *Unf-Que-AdpM*, *Ste-Que-AdpM* and *Dij-Que-AdpM* first need to convert the height map to a *TIN* in 0.42s, and then perform the query on this *TIN* in 380,000s ≈ 4.3 days, 70,000s ≈ 19.4 hours, and 33,000s ≈ 9.2 hours, respectively.

3.2 Height Map Simplification Algorithms

No existing algorithm can directly simplify a height map or a point cloud. Algorithms [18, 24, 28, 31] can simplify a *TIN* by iteratively removing a vertex v in a *TIN* and using *triangulation* [33] to

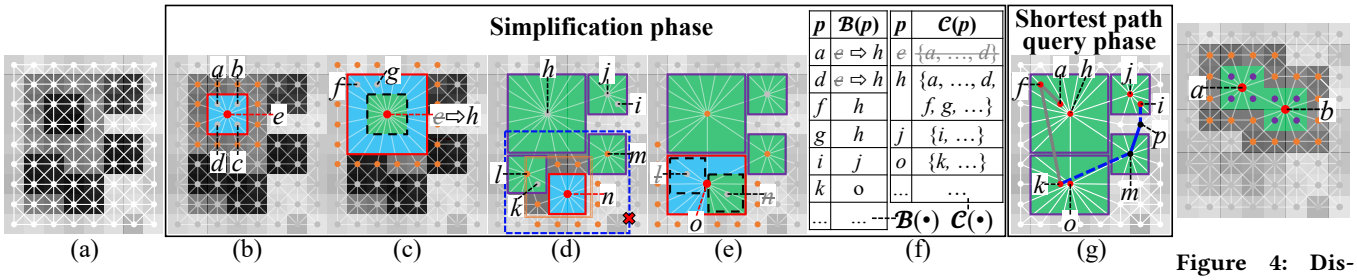

Figure 3: Algorithm *Mem-SimQue*

Figure 4: Distance checking

form new faces among the adjacent vertices of v. But, they cannot be used for simplifying a height map directly, since no vertices can be deleted nor no new faces can be created in a height map. Among these algorithms, the best algorithm *shortest Network distance TIN Simplification and simplified TIN shortest path Query* (*Net-SimQue*) [28] can simplify a *TIN* with an error bound guarantee (i.e., the *shortest network distances* between any pairs of vertices on the simplified *TIN* is an ϵ-approximate of the distances on the original *TIN*) and answer the shortest path query on the simplified *TIN*.

Adaptions: (1) Given a point cloud, we adapt our algorithm *Mem-SimQue* to be algorithm *Memory saving simulated point cloud Simplification and simplified point cloud shortest path Query* (*Mes-SimQue*) [52] by simulating our algorithm on the conceptual graph of the point cloud to obtain a simplified point cloud and query shortest paths on the simplified point cloud. (2) Given a *TIN*, we adapt algorithm *Net-SimQue* to be the best-known *TIN* simplification algorithm *shortest Surface distance TIN Simplification and simplified TIN shortest path Query* (*Sur-SimQue*) [25, 28] by using the *shortest surface distance* as the distance metric. (3) Given a *Height map*, we adapt algorithm (i) *Mes-SimQue* [52], (ii) *Net-SimQue* [28], and (iii) *Sur-SimQue* [25, 28] to be algorithm (i) *Mes-SimQue-AdpM*, (ii) *Net-SimQue-AdpM*, and (iii) *Sur-SimQue-AdpM*, by first constructing a point cloud or *TIN* using the height map, and then applying the corresponding algorithms for point cloud and *TIN* simplification and shortest path query.

3.2.1 ***Algorithm Mes-SimQue-AdpM.*** It is similar to our algorithm *Mem-SimQue* since the height map and point cloud have the same conceptual graph.

3.2.2 ***Algorithm Net-SimQue-AdpM.*** During each *TIN* simplification iteration, it checks whether the shortest network distances between pairs of *adjacent* vertices (instead of all pairs of vertices) of the removed vertex v on the simplified *TIN* is an ϵ-approximate of the distances on the original *TIN*. Its simplification time, output size, and shortest path query time are $O(n^2 \log n)$, $O(n)$, and $O(n \log n)$, respectively.

3.2.3 ***Algorithm Sur-SimQue-AdpM.*** During each iteration, it places Steiner points on the faces *adjacent* to v (using the technique used in work [25] for answering the arbitrary points-to-arbitrary points shortest surface path query on a *TIN*), and checks whether the shortest surface distances between any pairs of these Steiner points (instead of all pairs of vertices) on the simplified *TIN* is an ϵ-approximate of the distances on the original *TIN*. Its simplification time, output size, and shortest path query time are $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}} \log \frac{1}{\epsilon})$, $O(n)$, and $O(n^2)$, respectively.

Drawbacks: (1) *Large output size and shortest path query time*: Both algorithm *Net-SimQue-AdpM* and *Sur-SimQue-AdpM* do not use any optimization techniques during *TIN* simplification, so the output size of the simplified *TIN*s are large, and the shortest path query time on their simplified *TIN*s are large. But, algorithm *Mem-SimQue* uses several optimization techniques to reduce the output size of the simplified height map. (2) *Large error*: Since algorithm *Net-SimQue-AdpM* uses the shortest network path passing on a *TIN*, its simplified *TIN* has a large error. But, algorithm *Mem-SimQue* uses the shortest path passing on a height map with a smaller error. (3) *Large simplification time*: Since algorithm *Sur-SimQue-AdpM* uses the shortest surface path passing on a *TIN* and involves numerous distance calculations due to a large amount of Steiner points, its simplification time is large. But, algorithm *Mem-SimQue* uses the shortest path passing on a height map (which is computationally efficient) and involves fewer distance calculations due to the pruning out of unnecessary checks. Our experiments show that for a height map with 50k pixels, the simplification time algorithm *Net-SimQue-AdpM*, *Sur-SimQue-AdpM*, and *Mem-SimQue* are 25,800s ≈ 7.2 hours, 103,000s ≈ 1.2 days, and 250s ≈ 4.6 min, respectively. The *kNN* query time of 10k objects on the simplified *TIN* or height map generated by them are 16,800s ≈ 4.7 hours, 67,000s ≈ 18.6 hours, and 50s, respectively.

4 METHODOLOGY

4.1 Overview of Algorithm *Mem-SimQue*

4.1.1 ***Two components.*** We give two components of algorithm *Mem-SimQue*, (1) *belonging table* and (2) *containing table*.

(1) **The belonging table $\mathcal{B}(\bullet)$**: It is a *hash table*. Given a deleted pixel p in M, $\mathcal{B}(p)$ returns the added pixel p' in $\widetilde{M}$ such that p belongs to p' in $O(1)$ time. In Figure 3 (b), we merge $\{a, b, c, d\}$ to be e, the deleted pixel a belongs to the added pixel e, so $\mathcal{B}(a) = e$.

(2) **The containing table $\mathcal{C}(\bullet)$**: It is a *hash table*. Given an added pixel p in $\widetilde{M}$, $\mathcal{C}(p)$ returns the set of deleted pixels $\{p_1, p_2, \dots\}$ in M belong to p in $O(1)$ time. In Figure 3 (b), the deleted pixels $\{a, b, c, d\}$ belongs to the added pixel e, so $\mathcal{C}(e) = \{a, b, c, d\}$.

4.1.2 ***Two phases.*** There are two phases of algorithm *Mem-SimQue*, (1) *simplification phase* and (2) *shortest path query phase*.

(1) **Simplification phase**: In Figures 3 (a) - (f), given a height map $M = (P, N(\bullet))$, we iteratively merge some adjacent pixels to get a simplified height map $\widetilde{M} = (P_{rema} \cup P_{add}, \widetilde{N}(\bullet))$, store added and deleted pixels information in $\mathcal{B}(\bullet)$ and $C(\bullet)$, until Property 1 does not satisfy.

(2) **Shortest path query phase**: In Figure 3 (g), given $\widetilde{M}$, a pair of pixels s and t on M, $\mathcal{B}(\bullet)$, and $C(\bullet)$, we use Dijkstra's algorithm [19] to calculate the shortest path passing on the conceptual graph of $\widetilde{M}$ between s and t.

4.2 Key Idea of Algorithm *Mem-SimQue*

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

(1) **Merge four pixels**: We start by choosing four adjacent non-merged pixels in a square such that the variances of their elevation values are the smallest. In Figure 3 (b), we merge $\{a, b, c, d\}$ to be e, and get $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and process to the next merging type. If not, we terminate the algorithm.

(2) **Merge added pixels**: Given an added pixel p obtained from the previous merging type, we then merge p with some of its neighbour pixels, i.e., we enlarge p by expanding its one neighbour pixel in left / right / top / bottom direction (or combinations of several directions) to reduce the number of pixels in $\widetilde{M}$. Let *Direction*, i.e., $Dir = \{(L, R, T, B), (L, R, T, \bullet), (L, R, \bullet, B), (L, \bullet, T, B), (\bullet, R, T, B), (L, R, \bullet, \bullet), (L, \bullet, T, \bullet), (L, \bullet, \bullet, B), (\bullet, R, T, \bullet), (\bullet, R, \bullet, B), (\bullet, \bullet, T, B), (L, \bullet, \bullet, \bullet), (\bullet, R, \bullet, \bullet), (\bullet, \bullet, T, \bullet), (\bullet, \bullet, \bullet, B)\}$ be the directions that we expand, where L, R, T, B means that we expand p to cover its one neighbour pixels in the direction of *left*, *right*, *top*, *bottom*, and $\bullet$ means that we do not expand in that direction. In Figure 3 (c), we merge e with $\{f, g, \dots\}$ to be h, i.e., we enlarge e by expanding into all four directions, which corresponds to (L, R, T, B), and get $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and repeat this process. If not, we go back to the *four pixels merging* type by selecting four new pixels. In Figure 3 (d), we merge n with $\{l, m, \dots\}$ to be an added pixel shown in the blue frame, i.e., we still enlarge n by expanding into all four directions. We need to cover n's neighbour pixel as a whole in the selected direction to further reduce the number of pixels in $\widetilde{M}$. If this partially covers a pixel h (that is not n's neighbour pixel), we cannot merge them, so we enlarge n by expanding into directions using the next element in Dir until it does. In Figure 3 (e), we merge n with $\{l, \dots\}$ to be o, i.e., we enlarge n by expanding into the top and left directions, which corresponds to $(L, \bullet, T, \bullet)$, and get $\widetilde{M}$. If Property 1 satisfies, we confirm this merging and repeat this process. If not, we go back to the *four pixels merging* type.

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

(1) **Efficient height map shortest path query by algorithm *Eff-Que***: When checking whether Property 1 is satisfied, we use algorithm *Eff-Que* (a Dijkstra's algorithm) to directly calculate the shortest path passing on the conceptual graph of M and $\widetilde{M}$. But, calculating the shortest surface path passing on a *TIN* is slow. Although we calculate the path passing on the conceptual graph of M and $\widetilde{M}$, algorithm *Mem-SimQue* is very different from simplifying a graph, since we utilize the unique technique of height map, i.e., pixel merging, for significant memory saving, which is missing in graph simplification.

(2) **Efficient Property 1 checking**: Checking Property 1 involves checking whether Inequality 1 is satisfied for *all* pixels on M, this naive method is time-consuming. Instead, we can check distances only related to the *neighbour* pixels of the newly added pixels in each iteration. In Figure 4, suppose that the green pixel b is the newly added pixel, we need to check whether the distances between pairs of pixels denoted as orange and purple points satisfy Inequality 1.

4.2.3 ***Efficient shortest path query.*** We use an example to explain why algorithm *Mem-SimQue* has a small shortest path query time. In Figure 3 (g), given a pair of pixels i and k, we use Dijkstra's algorithm to calculate the shortest path passing on the conceptual graph of $\widetilde{M}$ between them. Since they are deleted pixels, their x- and y-coordinate values on $\widetilde{M}$ are the same as the values on M, and we approximate their elevation values on $\widetilde{M}$ using the elevation value of $\mathcal{B}(i)$ and $\mathcal{B}(k)$, i.e., j and o on M. If they are not deleted pixels, their such values on $\widetilde{M}$ are the same as those on M.

4.3 Simplification Phase

Algorithms 1 and 2 show the process in detail, Algorithm 2 is a sub-algorithm used for 2 times in Algorithm 1.

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

Algorithm 1 *Mem-SimQue* (M)

Input: $M = (P, N(\bullet))$
Output: $\widetilde{M}$, $\mathcal{B}(\bullet)$, and $C(\bullet)$
1: $P_{rema} \leftarrow P$, $P_{add} \leftarrow \emptyset$, $\widetilde{N}(\bullet) \leftarrow N(\bullet)$, $\mathcal{B}(\bullet) \leftarrow \emptyset$, $C(\bullet) \leftarrow \emptyset$
2: **while** we can merge four adjacent non-merged pixels in a square in $\widetilde{M} = (P_{rema} \cup p_{add}, \widetilde{N}(\bullet))$ **do**
3: $\widehat{P} \leftarrow$ four pixels with the smallest elevation values variance, $p_{add} \leftarrow \emptyset$
4: **if** *UpdateCheck* $(P_{rema}, P_{add}, \widetilde{N}(\bullet), \widehat{P}, p_{add}, \mathcal{B}(\bullet), C(\bullet))$ is *True* **then**
5: **while** we can enlarge p_{add} by expanding its one neighbor pixel in left / right / top / bottom direction (or combinations of several directions) in $\widetilde{M} = (P_{rema} \cup P_{add}, \widetilde{N}(\bullet))$ that covers p_{add}'s neighbour pixels as a whole and does not partially cover any other pixels that are not p_{add}'s neighbour pixels **do**
6: **for** i-th element in Dir **do**
7: $\widehat{P} \leftarrow \{p_{add}\} \cup$ pixels after expanding one neighbour pixel in the directions using the i-th element that the newly added pixel covers p_{add}'s neighbour pixels as a whole and does not partially cover any other pixels that are not p_{add}'s neighbour pixels
8: **if** *UpdateCheck* $(P_{rema}, P_{add}, \widetilde{N}, \widehat{P}, p_{add}, \mathcal{B}(\bullet), C(\bullet))$ is *Ture* **then**
9: **break**
10: **return** $\widetilde{M} = (P_{rema} \cup P_{add}, \widetilde{N}(\bullet))$, $\mathcal{B}(\bullet)$, and $C(\bullet)$

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

(2) *Merge added pixels* (lines 5-9). In Figure 3 (c), $p_{add} = e$, we can merge $\widehat{P} = \{e\} \cup \{f, g, \dots\} = \{e, f, g, \dots\}$, i.e., we can enlarge e by

expanding it to cover its one neighbour pixel in the directions of 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 get $p_{add} = h$ 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 pixels iteration* (lines 2-4): In Figure 3 (d), we use a similar process to get new pixels j, l, m and $\widetilde{M}$. Suppose that we cannot merge added pixels for j, l, m (lines 5-9), we go back to line 2 after each merging. Then, we use a similar process to get a new pixel n and $\widetilde{M}$.

(4) *Merge added pixels iteration* (lines 5-9): In Figure 3 (d), $p_{add} = n$, if we merge $\widehat{P} = \{n\} \cup \{l, m, \dots\} = \{l, m, n, \dots\}$, i.e., if we enlarge n by expanding it to cover its one neighbour pixel in the directions of the 1-st element (L, R, T, B) in *Dir* (lines 5-6), since the newly added pixel with blue frame partially covers h that is not n's neighbour pixel, we cannot merge them. So, we enlarge n by expanding into directions using the next element in *Dir* until it does (line 6). In Figure 3 (e), $p_{add} = n$, we can merge $\widehat{P} = \{n\} \cup \{l, \dots\} = \{n, l, \dots\}$, i.e., we can enlarge n by expanding into the top and left directions of the 7-th element $(L, \bullet, T, \bullet)$ in *Dir* (lines 5-7), suppose that the result of *UpdateCheck* is *True* (lines 8-9), we get $p_{add} = o$ and $\widetilde{M}$.

4.3.2 ***Detail and example for Algorithm 2.*** The following shows Algorithm 2 with an example. We use Figures 3 (b) and (c) for illustration, since Figures 3 (d) and (e) are similar.

Algorithm 2 *UpdateCheck* (P_{rema}, P_{add}, $\widetilde{N}(\bullet)$, $\widehat{P}$, p_{add}, $\mathcal{B}(\bullet)$, $C(\bullet)$)

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

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

(2) *Update neighbour pixels* (lines 10-13): In Figures 3 (b) and (c), for e and h, we update their neighbour pixels to be the pixels represented in orange points; for these neighbor pixels, we also update e and h to be their neighbor pixels.

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

(4) *Check Property 1* (lines 20-22): In Figures 3 (b) and (c), suppose that $\widetilde{M}'$ satisfy Property 1, we have $\widetilde{M}$. In Figure 3 (f), we have the updated $\mathcal{B}(\bullet)$ and $C(\bullet)$.

4.4 Shortest Path Query Phase

4.4.1 ***Concept.*** (1) **Estimated pixel**: Given a deleted pixel $p \in P - P_{rema}$, let $\widetilde{p}$ be an *estimated pixel* of p, such that $\widetilde{p}.x = p.x$, $\widetilde{p}.y = p.y$, $\widetilde{p}.z = \mathcal{B}(p).z$.

(2) **Intra-path and inter path**: Given a deleted pixel $p \in P - P_{rema}$ and a pixel $p' \in \widetilde{N}(\mathcal{B}(p))$ that is a neighbour pixel of the added pixel that p belongs to, let $\Pi_1(p, p'|\widetilde{M}) = (\widetilde{p}, \widetilde{p'})$ be the *intra-path* between p and p' passing on $\widetilde{M}$. Given a pair of pixels p and q in $\widetilde{P}$, let $\Pi_2(p, q|\widetilde{M})$ be the *inter-path* between p and q passing on ($\widetilde{G}$ of) $\widetilde{M}$ calculated using Dijkstra's algorithm on $\widetilde{G}$. In Figure 3 (g), $\Pi_1(i, p|\widetilde{M})$ in blue dashed line is an intra-path and $\Pi_2(p, m|\widetilde{M})$ in blue solid line is an inter-path.

4.4.2 ***Detail and example.*** Given a pair of s and t on M, there are three cases.

(1) **Both pixels deleted** (both s and t are in $P - P_{rema}$): There are three more cases.

(i) *Different and non-adjacent belonging pixel* (s and t belong to the different added pixels u and v, u and v are not neighbour, i.e., $\mathcal{B}(s) = u$, $\mathcal{B}(t) = v$, $v \notin \widetilde{N}(u)$ nor $u \notin \widetilde{N}(v)$): We return $\Pi(s, t|\widetilde{M})$ by concatenating the intra-path $\Pi_1(s, p|\widetilde{M})$, the inter-path $\Pi_2(p, q|\widetilde{M})$, and the intra-path $\Pi_1(q, t|\widetilde{M})$, such that $|\Pi(s, t|\widetilde{M})| = \min_{\forall p \in \widetilde{N}(\mathcal{B}(s)), q \in \widetilde{N}(\mathcal{B}(t))} |\Pi_1(s, p|\widetilde{M})| + |\Pi_2(p, q|\widetilde{M})| + |\Pi_1(q, t|\widetilde{M})|$. A naive algorithm uses Dijkstra's algorithm on the conceptual graph $\widetilde{G}$ of $\widetilde{M}$ with each pixel in $\widetilde{N}(\mathcal{B}(s))$ as a source to calculate inter-paths. But, we propose an efficient algorithm by using Dijkstra's algorithm only once. If the number of pixels in $\widetilde{N}(\mathcal{B}(s))$ is less than that of $\widetilde{N}(\mathcal{B}(t))$, we temporarily insert intra-paths between $\widetilde{s}$ and each pixel in $\widetilde{N}(\mathcal{B}(s))$ as edges in $\widetilde{G}$ (we remove these edges after we finish this round of calculation), and then we use Dijkstra's algorithm on $\widetilde{G}$ with $\widetilde{s}$ as a source, and terminate when it has visited all pixels in $\widetilde{N}(\mathcal{B}(t))$, to calculate the intra-path connecting to $\widetilde{s}$ and the inter-path. We append them with the intra-path connecting to $\widetilde{t}$ and get $\Pi(s, t|\widetilde{M})$. If the number of pixels in $\widetilde{N}(\mathcal{B}(s))$ is larger than that of $\widetilde{N}(\mathcal{B}(t))$, we swap s and t. In Figure 3 (g), $\Pi(j, k|\widetilde{M}) = (\widetilde{j}, p, m, o, \widetilde{k})$.

(ii) *Different and adjacent belonging pixel* (s and t belong to the different added pixels u and v, u and v are neighbour, i.e., $\mathcal{B}(s) = u$, $\mathcal{B}(t) = v$, $v \in \widetilde{N}(u)$, and $u \in \widetilde{N}(v)$): We return $\Pi(s, t|\widetilde{M}) = (\widetilde{s}, \widetilde{t})$. In Figure 3 (g), $\Pi(f, k|\widetilde{M}) = (\widetilde{f}, \widetilde{k})$.

(iii) *Same belonging pixel* (s and t belong to the same added pixel, i.e., $\mathcal{B}(s) = \mathcal{B}(t)$): We also return $\Pi(s, t|\widetilde{M}) = (\widetilde{s}, \widetilde{t})$. In Figure 3 (g), $\Pi(a, f|\widetilde{M}) = (\widetilde{a}, \widetilde{f})$.

(2) **One pixel deleted and one pixel remaining** (either s or t is in $P - P_{rema}$): If $s \in P_{rema}$, the inter-path connecting to s does not exist, we use Dijkstra's algorithm on $\widetilde{G}$ with s as a source, and terminate when it has visited all $q \in \widetilde{N}(\mathcal{B}(t))$. We append them with the intra-path connecting to $\widetilde{t}$ and get $\Pi(s, t|\widetilde{M})$. If $t \in P_{rema}$, we swap s and t.

(3) **Both pixels remaining** (both s and t are in P_{rema}): Both inter-paths do not exist, we just use Dijkstra's algorithm on $\widetilde{G}$ with s as a source and t as a destination to get $\Pi(s, t|\widetilde{M})$.

4.5 Efficient Property 1 Checking

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

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

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

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

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

4.6 Proximity Query Algorithms

Given M and $\widetilde{M}$, a set of n' interested pixels on M, a query pixel $i \in P$, a user parameter k, and a range value r, we can answer other proximity queries, i.e., *kNN* and range queries, using algorithm *Eff-Que* and the shortest path query phase of algorithm *Mem-SimQue*. A naive algorithm uses them for n' times between q and all interested pixels, and then performs a linear scan on the paths to return *kNN* and range query results.

But, we propose an efficient algorithm. The basic idea is to use both algorithms only *once* for time-saving, since they are single-source-all-destination algorithms (Dijkstra's algorithms). (1) For algorithm *Eff-Que*, we use Dijkstra's algorithm once with i as a source and all interested pixels as destinations, and then directly return *kNN* and range query results without any linear scan, since these paths are already sorted in order during the running of Dijkstra's algorithm. (2) For the shortest path query phase of algorithm *Mem-SimQue*, we also use Dijkstra's algorithm once. There are two changes in Section 4.4.2. (i) We change s to i and always use i or $\widetilde{i}$ as a source (depending on whether i is a deleted pixel). (ii) We change "terminate when Dijkstra's algorithm has visited all pixels in $\widetilde{N}(\mathcal{B}(t))$" to "terminate when Dijkstra's algorithm has visited all pixels in S", where S is a set of pixels, such that for each interested pixel i', we store i' in S if i' is a remaining pixel, or we store pixels in $\widetilde{N}(\mathcal{B}(i'))$ into S if i' is a deleted pixel. Then, we perform a linear scan on the paths to return *kNN* and range query results.

4.7 Theoretical Analysis

4.7.1 ***Algorithm Eff-Que and Mem-SimQue.*** We analysis them in Theorems 4.1 and 4.2.

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

Proof. Since algorithm *Eff-Que* is Dijkstra's algorithm and there are $O(n)$ pixels, we obtain the shortest path query time and memory usage. Since Dijkstra's algorithm returns the exact result, algorithm *Eff-Que* returns the exact shortest path passing on M. □

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

Proof Sketch. The simplification time $O(n\sqrt[3]{n}\log n) = O(n\log n \cdot \sqrt[3]{n})$ is due to the usage of Dijkstra's algorithm in $O(n\log n)$ time for $O(1)$ pixels in *R2R*, *R2D*, and *D2D* distance checking in each pixel merging iteration, and there are total $O(\sqrt[3]{n})$ pixel merging iterations. The output size $O(n \div \log n) = O(\frac{n}{\log n})$ is due to the $O(\log n)$ deleted pixels belonging to each added pixel, and there are total n pixels on M. The shortest path query time $O(\frac{n}{\log n}\log\frac{n}{\log n})$ is due to the one usage of Dijkstra's algorithm on $\widetilde{M}$ with $O(\frac{n}{\log n})$ pixels. The error bound of $\widetilde{M}$ is due to the *R2R*, *R2D*, and *D2D* pixels checking when simplifying M. If the distances on $\widetilde{M}$ between any pair of pixels near p_{add} do not change a lot, then the distances on $\widetilde{M}$ between any pair of pixels far away from P_{add} cannot change a lot. The detailed proof appears in the appendix. □

4.7.2 ***The shortest path passing on a height map or a point cloud, and the shortest surface or network path passing on a TIN.*** We show the distance relationship of $\Pi(s, t|M)$ with $\Pi(s, t|C)$, $\Pi_N(s, t|T)$, and $\Pi(s, t|T)$ in Lemma 4.3.

Lemma 4.3. *Given a pair of objects s and t on M, we have (1) $|\Pi(s,t|M)| = |\Pi(s,t|C)|$, (2) $|\Pi(s,t|M)| \le |\Pi_N(s,t|T)|$ and (3) $|\Pi(s,t|M)| \le k \cdot |\Pi(s,t|T)|$, where $k = \max\{\frac{2}{\sin\theta}, \frac{1}{\sin\theta\cos\theta}\}$.*

Proof. (1) Since $\Pi(s, t|M)$ and $\Pi(s, t|C)$ passes on the same conceptual graph, $|\Pi(s, t|M)| = |\Pi(s, t|C)|$. (2) In Figure 2 (a), the green pixel q on M can connect with eight blue neighbours, while in Figure 2 (b), the black vertex q on T can only connect with six blue neighbours. So, $|\Pi(s, t|M)| \le |\Pi_N(s, t|T)|$. (3) Using the left-hand side equation in Lemma 2 of work [28], we know $|\Pi_E(s, t|T)| \le$

$k \cdot |\Pi(s,t|T)|$, where $\Pi_E(s,t|T)$ is the shortest path passing on the edges (of the faces that $\Pi(s,t|T)$ passes) of T. Since $\Pi_N(s,t|T)$ considers all the vertices on T, $|\Pi_N(s,t|T)| \leq |\Pi_E(s,t|T)|$. We combine them to complete the proof. □

4.7.3 ***Proximity query algorithms.*** We provide analysis on proximity queries (i.e., *kNN* and range queries) algorithms using algorithm *Eff-Que* and *Mem-SimQue*. Given a query pixel i, let p_f and p'_f be the furthest returned pixel to i calculated by algorithm *Eff-Que* and *Mem-SimQue*, respectively. We define the error rate of *kNN* and range queries to be $\frac{|\Pi(i,p'_f|M)|}{|\Pi(i,p_f|M)|}$. We give the query time and error rate of *kNN* and range queries using algorithm *Eff-Que* and *Mem-SimQue* in Theorem 4.4.

Theorem 4.4. *The query time of both kNN and range queries by using algorithm (1) Eff-Que is $O(n \log n)$ and (2) Mem-SimQue is $O(\frac{n}{\log n} \log \frac{n}{\log n})$, respectively. Algorithm (1) Eff-Que returns the exact result and (2) Mem-SimQue has an error rate $\frac{1+\epsilon}{1-\epsilon}$ for both kNN and range queries, respectively.*

Proof Sketch. The query time is due to the usage of algorithm *Eff-Que* and the shortest path query phase of algorithm *Mem-SimQue* once. The error rate is due to their definition and the error of algorithm *Mem-SimQue*. □

5 EMPIRICAL STUDIES

5.1 Experimental Setup

We conducted our experiments using a Linux machine with 2.2 GHz CPU and 512GB memory on height maps, point clouds, and *TIN*s. All algorithms were implemented in C++. The following experiment setup follows the work [27, 28, 44, 45, 51, 52].

5.1.1 ***Datasets.*** (1) *Height map* datasets: We conducted experiments on 34 real height map datasets in Table 1, where the subscript m means a height map. (i) BH_m and EP_m are originally represented as points clouds with 8km × 6km covered region, we created height maps with pixel's x-, y-coordinate values, and elevation values equal to the x-, y-, and z-coordinate values of these points, respectively. GF_m, LM_m, and RM_m are originally represented as height maps with 8km × 6km covered region and are obtained from Google Earth [3]. These five datasets have a resolution of 10m × 10m [28, 45, 51, 52]. (ii) Small-version datasets are generated using same region of the original datasets with a (lower) resolution of 70m × 70m with the dataset generation procedure in work [45, 51, 52]. (iii) Multi-resolution datasets with different numbers of pixels are generated similarly. (2 & 3) *Point cloud* and *TIN* datasets: Based on the 34 height map datasets, we use the vertices of the conceptual graph of these height map datasets to generate another 34 point cloud datasets, and then triangulate [21, 33, 53] them to generate another 34 *TIN* datasets. We use c and t as the 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 using the vertices of the conceptual graph of the height map to get a point cloud and triangulating [21, 33, 53] these vertices to get a *TIN*. We have four height map simplification algorithms, i.e., (i) *Sur-SimQue-AdpM* [25, 28], (ii) *Net-SimQue-AdpM* [28], (iii) *Mes-SimQue-AdpM* [52], and (iv) our algorithm *Mem-SimQue*. We have five shortest path query algorithms, i.e., (v) *Unf-Que-AdpM* [13, 46], (vi) *Ste-Que-AdpM* [27, 51], (vii) *Dij-Que-AdpM* [28], (viii) *Con-Que-AdpM* [52], and (ix) our algorithm *Eff-Que*. We compare these algorithms in Table 2.

(2) To solve the existing problem on *point Clouds* [52], we adapted algorithms on height maps or *TIN*s, by using the vertices of the conceptual graph of the point cloud to get a height map and triangulating [21, 33, 53] these vertices to get a *TIN*. Similarly, we have nine algorithms, i.e., (i) *Sur-SimQue-AdpC* [25, 28], (ii) *Net-SimQue-AdpC* [28], (iii) *Mes-SimQue* [52], (iv) *Mem-SimQue-AdpC*, (v) *Unf-Que-AdpC* [13, 46], (vi) *Ste-Que-AdpC* [27, 51], (vii) *Dij-Que-AdpC* [28], (viii) *Con-Que* [52], and (ix) *Eff-Que-AdpC*.

(3) To solve the existing problem on *TIN*s [25, 28], we adapted algorithms on height maps or point clouds, by using the vertices of the *TIN* to obtain a height map or a *TIN*. Similarly, we have nine algorithms, i.e., (i) *Sur-SimQue* [25, 28], (ii) *Net-SimQue* [28], (iii) *Mes-SimQue-AdpT* [52], (iv) *Mem-SimQue-AdpT*, (v) *Unf-Que* [13, 46], (vi) *Ste-Que* [27, 51], (vii) *Dij-Que* [28], (viii) *Con-Que-AdpT* [52], and (ix) *Eff-Que-AdpT*.

5.1.3 ***Query Generation.*** We conducted three proximity queries, i.e., (1) shortest path query, (2) all objects *kNN* query, and (3) all objects range query. (1) For the shortest path query, we issued 100 query instances where for each instance, we randomly chose two pixels on the height map (or two points on the point cloud, or two vertices on the *TIN*), one as a source and the other as a destination. The average, minimum, and maximum results were reported. In the experimental result figures, the vertical bar and the points mean the minimum, maximum, and average results. (2 & 3) For all objects *kNN* query and range query, we randomly chose 1000 of pixels on the height map (or points on the point cloud, or vertices on the *TIN*) as objects, and we perform the proximity query algorithm in Section 4.6 for all the objects as query objects.

5.1.4 ***Factors and Measurements.*** We studied four factors, i.e., (1) ϵ (i.e., the error parameter), (2) n (i.e., the dataset size, meaning the number of pixels in a height map, points in a point cloud, or vertices in a *TIN*), (3) k (i.e., the parameter in the *kNN* query), and (4) r (i.e., the parameter in the range query). When not varying k and r, we fix k as the number of chosen objects, and r as the distance of the longest path. In addition, we used nine measurements to evaluate the algorithm performance, namely (1) *simplification time*, (2) *memory consumption* (i.e., the space consumption when running the algorithm), (3) *output size*, (4) *query time* (i.e., the shortest path query time), (5 & 6) *kNN or range query time* (i.e., all objects *kNN* or range query time), (7) *distance error* (i.e., the error of the distance returned by the algorithm compared with the exact distance), (8 & 9) *kNN or range query error* (i.e., the error rate of the *kNN* or range query defined in Section 4.7.3).

5.2 Experimental Results

5.2.1 ***Experimental results for height maps.*** We study proximity queries on height maps. The distance of the path calculated by *Eff-Que* is used for distance error calculation since the path is the

Table 1: Height map datasets

Name	$\|N\|$
Original dataset	
BearHead (BH_m) [2, 44, 45]	0.5M
EaglePeak (EP_m) [2, 44, 45]	0.5M
GunnisonForest (GF_m) [5]	0.5M
LaramieMount (LM_m) [7]	0.5M
RobinsonMount (RM_m) [10]	0.5M
Small-version dataset	
BH_m*-small*	10k
EP_m*-small*	10k
GF_m*-small*	10k
LM_m*-small*	10k
RM_m*-small*	10k
Multi-resolution dataset	
BH_m multi-resolution	1M, 1.5M, 2M, 2.5M
EP_m multi-resolution	1M, 1.5M, 2M, 2.5M
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
EP_m*-small* multi-resolution	20k, 30k, 40k, 50k

Table 2: Comparison of algorithms

Algorithm	Simplification time		Output size		Shortest path query time		Error
Simplification algorithm							
Sur-SimQue-AdpM [25, 28]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large	$O(n^2)$	Large	Small
Net-SimQue-AdpM [28]	$O(n^2\log n)$	Medium	$O(n)$	Large	$O(n\log n)$	Medium	Medium
Mes-SimQue-AdpM [52]	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
***Mem-SimQue* (ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	**Small**
Shortest path query algorithm							
Unf-Que-AdpM [13, 46]	-	N/A	-	N/A	$O(n^2)$	Large	Small
Ste-Que-AdpM [27, 51]	-	N/A	-	N/A	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
Dij-Que-AdpM [28]	-	N/A	-	N/A	$O(n\log n)$	Medium	Medium
Con-Que-AdpM [52]	-	N/A	-	N/A	$O(n\log n)$	Medium	No error
***Eff-Que* (ours)**	-	N/A	-	N/A	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**

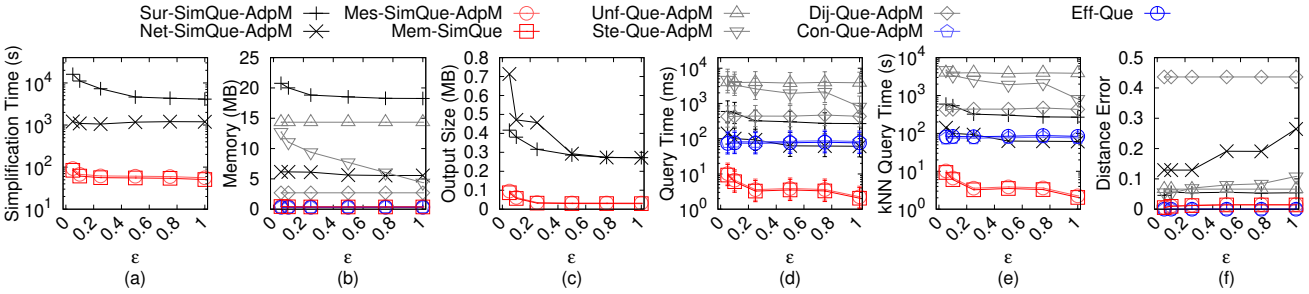

Figure 5: Baseline comparisons (effect of ϵ on BH_m*-small* height map dataset)

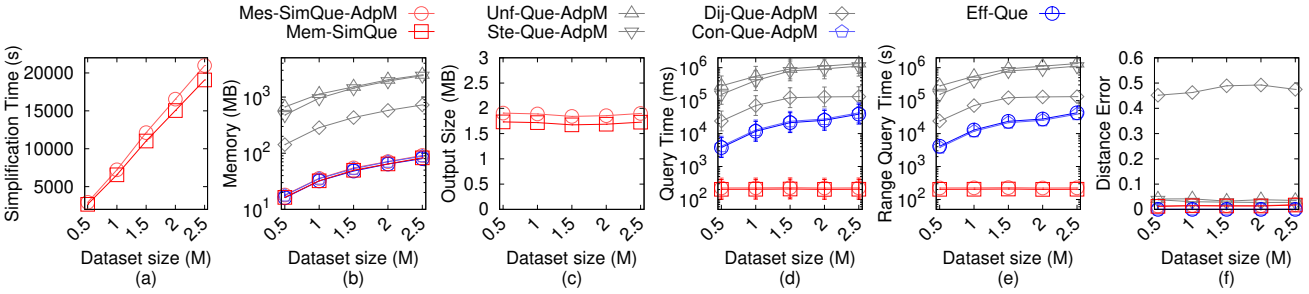

Figure 6: Baseline comparisons (effect of n on EP_m height map dataset)

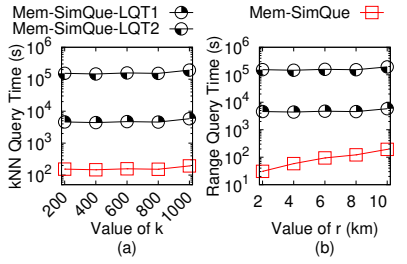

Figure 7: Ablation study for proximity query algorithms (effect of k and r on GF_m height map dataset)

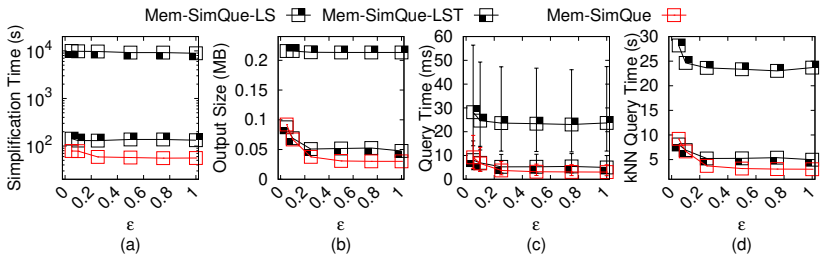

Figure 8: Ablation study for simplification algorithms on LM_m*-small* height map dataset

exact shortest path passing on the height map. Since *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* have excessive simplification time on original datasets, we compared all algorithms in Table 2 on

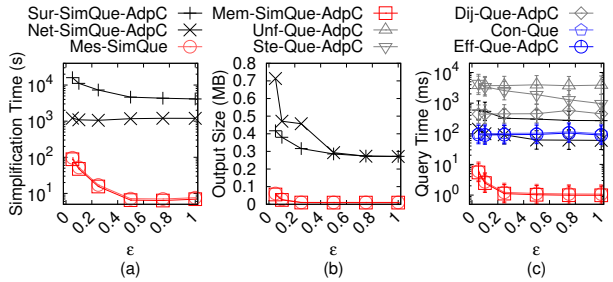

Figure 9: Baseline comparisons (effect of ϵ on RM_c-*small* point cloud dataset)

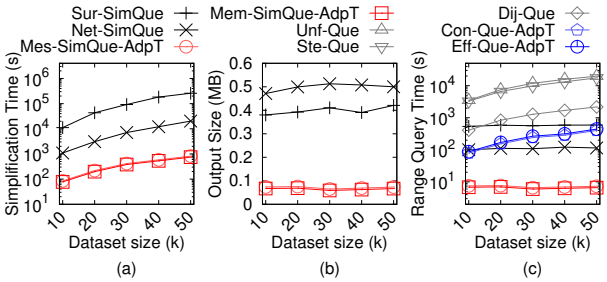

Figure 10: Baseline comparisons (effect of n on EP_t-*small TIN* dataset)

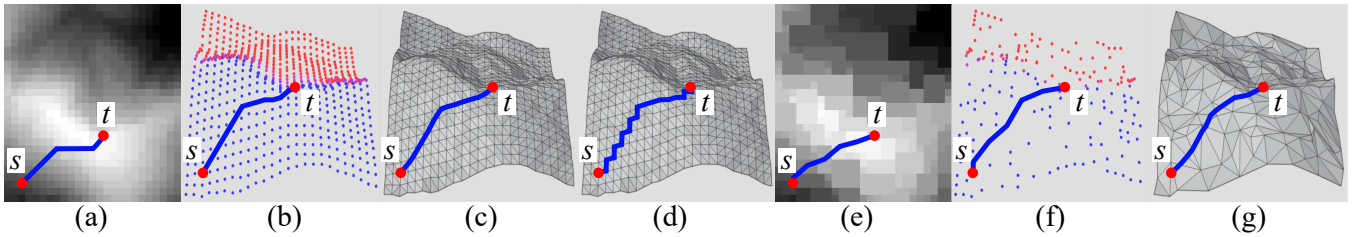

Figure 11: The shortest path passing on a (a) height map and (b) point cloud, the shortest (c) surface and (d) network path passing on a *TIN*, the shortest path passing on a simplified (e) height map and (f) point cloud, and (g) the shortest surface path passing on a simplified *TIN*

small-version datasets, and compared all algorithms except *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* on original datasets. Since *Eff-Que* uses Dijkstra's algorithm on the conceptual graph of a height map, it is the same as the shortest path algorithm on a general graph (constructed by the given height map), there is no need to compare them.

(1) **Baseline comparisons**: We study the effect of ϵ and n.

(i) **Effect of ϵ**: In Figure 5, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_m-*small* dataset by setting n to be 10k for baseline comparisons. The simplification time of *Mem-SimQue* is much smaller than that of *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* due to the efficient height map shortest path query and efficient Property 1 checking. The output size of *Mem-SimQue* is also much smaller than that of *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* due to the novel memory saving technique. The shortest path query time and the *kNN* query time of *Mem-SimQue* are also small since its simplified height map has a small output size. The distance error of *Mem-SimQue* is small (close to 0). The *kNN* and range query error are all equal to 0 (due to the small distance error), so their results are omitted.

(ii) **Effect of n (scalability test)**: In Figure 6, we tested 5 values of n from {0.5M, 1M, 1.5M, 2M, 2.5M} on EP_m dataset by setting ϵ to be 0.25 for baseline comparisons. *Mem-SimQue* performs better than all the remaining algorithms. The simplification time of *Mem-SimQue* is 19,000s ≈ 5.2 hours for a height map with 2.5M pixels, which shows its scalability. *Mes-SimQue-AdpM* performs similarly as *Mem-SimQue*, since they have the same simplification process, and the height map and point cloud have the same conceptual graph. *Eff-Que* performs better than *Unf-Que-AdpM* and *Ste-Que-AdpM* in terms of the range query time since it calculates the shortest path passing on a height map.

(2) **Ablation study for proximity query algorithms (effect of k and r)**: We consider two variations of *Mem-SimQue*, i.e., (i) *Mem-SimQue Large Query Time* (*Mem-SimQue-LQT1*): *Mem-SimQue* using the naive algorithm in the shortest path query phase in Section 4.4, but the efficient proximity query algorithm in Section 4.6 (i.e., we use each pixel in $\widetilde{N}(\mathcal{B}(i))$ as a source in each Dijkstra's algorithm and terminate when each algorithm has visited all pixels in S, where i and S have the same meaning in Section 4.6), and (ii) *Mem-SimQue-LQT2*: *Mem-SimQue* using the efficient algorithm in the shortest path query phase, but the naive proximity query algorithm. In Figures 7 (a) and (b), we tested 5 values of k from {200, 400, 600, 800, 1000} and 5 values of r from {2km, 4km, 6km, 8km, 10km} both on GF_m dataset by setting ϵ to be 0.25 and n to be 0.5M for ablation study. We show the results for the *kNN* and range query time since k and r only affect these two terms. *Mem-SimQue* performs better than both *Mem-SimQue-LQT1* and *Mem-SimQue-LQT2*, since we use the efficient algorithm for querying. Varying k does not affect the *kNN* query time of *Mem-SimQue*, since we append the paths calculated by Dijkstra's algorithm and the intra-paths as the path results, and we do not know the distance relationships among these paths before we perform a linear scan on them. But, a smaller r value can reduce the range query time of *Mem-SimQue*, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r. *Mem-SimQue-LQT1* are *Mem-SimQue-LQT2* not affected by k and r since they need to calculate all the paths, and then perform a linear scan on these paths.

(3) **Ablation study for simplification algorithms**: We consider two more variations of *Mem-SimQue*, i.e., (i) *Mem-SimQue*

Large output Size (*Mem-SimQue-LS*): *Mem-SimQue* using the naive merging technique that only merges four pixels in Section 4.2, and (ii) *Mem-SimQue Large Simplification Time* (*Mem-SimQue-LST*): *Mem-SimQue* using the naive checking technique that checks whether Inequality 1 is satisfied for all pixels in Section 4.2. In Figure 8, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on LM_m*-small* dataset by setting n to be 0.5M for ablation study. *Mem-SimQue* still performs better than these two variations, showing the effectiveness of our merging and checking techniques.

5.2.2 ***Experimental results for point clouds.*** We study proximity queries on point clouds to justify why our proximity queries on height maps are useful. The distance of the path calculated by *Con-Que* is used for distance error calculation since the path is the exact shortest path passing on the point cloud. For baseline comparisons, we study the effect of ϵ as follows, and study the effect of n in the appendix.

Effect of ϵ: In Figure 9, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on RM_c*-small* dataset by setting n to be 10k for baseline comparison. *Mem-SimQue-AdpC* still performs better than other baselines.

5.2.3 ***Experimental results for TINs.*** We study proximity queries on *TIN*s to justify why our proximity queries on height maps are useful. The distance of the path calculated by *Unf-Que* is used for distance error calculation since the path is the exact shortest surface path passing on the *TIN*. For baseline comparisons, we study the effect of n as follows, and study the effect of ϵ in the appendix.

Effect of n: In Figure 10, we tested 5 values of n from {10k, 20k, 30k, 40k, 50k} on EP_t*-small* dataset by setting and ϵ to be 0.1 for baseline comparisons. Although a *TIN* is given as input, *Mem-SimQue-AdpT* performs better than *Sur-SimQue* and *Net-SimQue* in terms of the simplification time, output size, and shortest path query time. The range query time of *Eff-Que-AdpT* is 100 times smaller than that of *Unf-Que* (although *Eff-Que-AdpT* needs to construct a height map from the given *TIN*, and there are no other additional steps for *Unf-Que*). The distance error of *Eff-Que-AdpT* is 0.06, but the distance error of *Dij-Que* is 0.45.

5.2.4 ***Case study.*** We conducted a case study on an evacuation simulation in Mount Rainier [38] due to frequent heavy snowfall [39]. The winter storms in January 2024 left 89 dead across the USA [32] and there exists the risk of asphyxiation if buried in snow [29]. In the event of snowfall, tourists on the mountain are promptly evacuated to nearby hotels for safety. The time of a human being buried in the snow is 2.4 hours[1], and the evacuation (i.e., the time of human's walking from the viewpoints to hotels) can be finished in 2.2 hours[2]. Thus, the calculation of the shortest paths is anticipated to be completed within 12 min (= 2.4 − 2.2 hours). Our experiments show that for a height map with 50k pixels with 10k different tourist positions and 50 hotels, the simplification time for (1) *Mem-SimQue* is 250s ≈ 4.6 min and (2) the best-known *TIN* simplification algorithm *Sur-SimQue-AdpM* is 103,000s ≈ 1.2 days. Under the same setting, the query time of calculating 10 nearest hotels for each tourist position is (1) 50s for *Mem-SimQue*, (2) 67,000s ≈ 18.6 hours for *Sur-SimQue-AdpM*, (3) 7,630s ≈ 2.1 hours for the best-known point cloud shortest path query algorithm *Con-Que-AdpM*, and (4) 380,000s ≈ 4.3 days for the best-known approximate *TIN* shortest surface path query algorithm *Ste-Que-AdpM*. Thus, *Mem-SimQue* is the most efficient for evacuation as 4.6 min + 50s ≤ 12 min. *Mem-SimQue* also supports real-time responses, capable of simplifying a height map in 5s and answering *kNN* and range queries in both 0.04s on a height map with 1k pixels and 200 objects.

5.2.5 ***Summary.*** Figures 11 (a) and (b) show the shortest paths passing on a height map and point cloud, Figures 11 (c) and (d) show the shortest surface and network paths passing on a *TIN*, Figures 11 (e) and (f) show the shortest paths passing on a simplified height map and point cloud, and Figure 11 (g) shows the shortest surface path passing on a simplified *TIN* of Mount Rainier in an area of 20km × 20km. The path in Figures 11 (a) and (b) are the same. The path in Figures 11 (a) and (c) are similar, but calculating the former path is much faster than the latter path, since the query region of the former path is smaller than the latter path. The path in Figure 11 (a) has a smaller error than the path in Figure 11 (d).

In terms of the simplification time and output size, and shortest path query time, *Mem-SimQue* is up to 412 times, 7 times, and 1,340 times better than the best-known *TIN* simplification algorithm *Sur-SimQue-AdpM*, respectively. Performing *kNN* and range queries on our simplified height map are both up to 153 times faster than *Con-SimQue-AdpM* on a point cloud, and 1,340 times faster than *Sur-SimQue-AdpM* on its simplified *TIN*. On a height map with 50k pixels and 10k objects, the simplification time, output size, and *kNN* query time are 250s ≈ 4.6 min, 0.07MB, and 50s for *Mem-SimQue*, but are 103,000s ≈ 1.2 days, 0.5MB, and 67,000s ≈ 18.6 hours for *Sur-SimQue-AdpM*, respectively.

6 CONCLUSION

We propose an efficient ϵ-approximate height map simplification algorithm *Mem-SimQue*, which has a good performance (in terms of the simplification time, output size, and shortest path query time) compared with the best-known algorithm. We also propose ϵ-approximate algorithms for answering other proximity queries, i.e., *kNN* and range queries on our simplified height map. For future work, we can propose new pruning techniques to further reduce the simplification time and output size of *Mem-SimQue*.

REFERENCES

[1] 2024. *125 Years of Topographic Mapping.* https://www.esri.com/news/arcnews/fall09articles/125-years.html
[2] 2024. *Data Geocomm.* http://data.geocomm.com/
[3] 2024. *Google Earth.* https://earth.google.com/web
[4] 2024. *Google Map.* https://www.google.com/maps
[5] 2024. *Gunnison National Forest.* https://gunnisoncrestedbutte.com/visit/places-to-go/parks-and-outdoors/gunnison-national-forest/
[6] 2024. *The History of Point Cloud Development.* https://www.linkedin.com/pulse/history-point-cloud-development-bimprove/
[7] 2024. *Laramie Mountain.* https://www.britannica.com/place/Laramie-Mountains
[8] 2024. *Metaverse.* https://about.facebook.com/meta
[9] 2024. *Preferred walking speed.* https://en.wikipedia.org/wiki/Preferred_walking_speed

[1] 2.4 hours = $\frac{10\text{centimeters}\times 24\text{hours}}{1\text{meter}}$, since the maximum snowfall rate (which is defined to be the maximum amount of snow accumulates in depth during a given time [15, 41]) in Mount Rainier is 1 meter per 24 hours [40], and it becomes difficult to walk, easy to lose the trail and get buried in the snow if the snow is deeper than 10 centimeters [22].

[2] 2.2 hours = $\frac{11.2\text{km}}{5.1\text{km/h}}$, since the average distance between the viewpoints and hotels in Mount Rainier National Park is 11.2km [4], and the average human walking speed is 5.1 km/h [9].

[10] 2024. *Robinson Mountain.* https://www.mountaineers.org/activities/routes-places/robinson-mountain
[11] 2024. *Visable Earth.* https://visibleearth.nasa.gov/images/73934/topography
[12] Leizhen Cai. 1994. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics* 48, 2 (1994), 187–194.
[13] Jindong Chen and Yijie Han. 1990. Shortest Paths on a Polyhedron. In *SOCG*. New York, NY, USA, 360–369.
[14] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. 1998. A comparison of mesh simplification algorithms. *Computers & Graphics* 22, 1 (1998), 37–54.
[15] The Conversation. 2024. *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
[16] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms.* MIT press.
[17] Ke Deng, Heng Tao Shen, Kai Xu, and Xuemin Lin. 2006. Surface k-NN query processing. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 78–78.
[18] 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* 17 (2008), 1101–1119.
[19] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.
[20] Hristo N Djidjev and Christian Sommer. 2011. Approximate distance queries for weighted polyhedral surfaces. In *Proceedings of the European Symposium on Algorithms*. 579–590.
[21] Michael Garland and Paul S Heckbert. 1995. Fast polygonal approximation of terrains and height fields. (1995).
[22] Fresh Off The Grid. 2024. *Winter Hiking 101: Everything you need to know about hiking in snow.* https://www.freshoffthegrid.com/winter-hiking-101-hiking-in-snow/
[23] 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 *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 1383–1388.
[24] Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 99–108.
[25] 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. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
[26] Shruti Kanga and Suraj Kumar Singh. 2017. Forest Fire Simulation Modeling using Remote Sensing & GIS. *International Journal of Advanced Research in Computer Science* 8, 5 (2017).
[27] Manohar Kaul, Raymond Chi-Wing Wong, and Christian S Jensen. 2015. New lower and upper bounds for shortest distance queries on terrains. *Proceedings of the VLDB Endowment* 9, 3 (2015), 168–179.
[28] 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. *Proceedings of the VLDB Endowment* 7, 1 (2013), 73–84.
[29] Russell LaDuca. 2020. *What would happen to me if I was buried under snow?* https://qr.ae/prt6zQ
[30] Mark Lanthier, Anil Maheshwari, and J-R Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* 30, 4 (2001), 527–562.
[31] Lian Liu and Raymond Chi-Wing Wong. 2011. Finding shortest path on land surface. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 433–444.
[32] Katie Hawkinson Louise Boyle, Kelly Rissman. 2024. *Winter storms leave 89 dead across US as chill settles over Great Lakes and Northeast.* https://www.independent.co.uk/climate-change/news/winter-storm-warning-weather-forecast-snow-b2480627.html
[33] de Berg Mark, Cheong Otfried, van Kreveld Marc, and Overmars Mark. 2008. *Computational geometry algorithms and applications.* Spinger.
[34] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. 1987. The discrete geodesic problem. *SIAM J. Comput.* 16, 4 (1987), 647–668.
[35] Hoong Kee Ng, Hon Wai Leong, and Ngai Lam Ho. 2004. Efficient algorithm for path-based range query in spatial databases. In *Proceedings. International Database Engineering and Applications Symposium, 2004. IDEAS'04.* IEEE, 334–343.
[36] Janet E Nichol, Ahmed Shaker, and Man-Sing Wong. 2006. Application of high-resolution stereo satellite images to detailed landslide hazard assessment. *Geomorphology* 76, 1-2 (2006), 68–75.
[37] 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.
[38] National Park Service. 2024. *Mount Rainier.* https://www.nps.gov/mora/index.htm
[39] National Park Service. 2024. *Mount Rainier Annual Snowfall Totals.* https://www.nps.gov/mora/planyourvisit/annual-snowfall-totals.htm
[40] National Park Service. 2024. *Mount Rainier Frequently Asked Questionss.* https://www.nps.gov/mora/faqs.htm
[41] National Weather Service. 2024. *Measuring Snow.* https://www.weather.gov/dvn/snowmeasure
[42] Cyrus Shahabi, Lu-An Tang, and Songhua Xing. 2008. Indexing land surface for efficient knn query. *Proceedings of the VLDB Endowment* 1, 1 (2008), 1020–1031.
[43] Jiaojiao Tian, Allan A Nielsen, and Peter Reinartz. 2015. Building damage assessment after the earthquake in Haiti using two post-event satellite stereo imagery and DSMs. *International Journal of Image and Data Fusion* 6, 2 (2015), 155–169.
[44] Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and David M. Mount. 2017. Distance oracle on terrain surface. In *SIGMOD/PODS'17*. New York, NY, USA, 1211–1226.
[45] 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).
[46] 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.
[47] 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.
[48] Songhua Xing, Cyrus Shahabi, and Bei Pan. 2009. Continuous monitoring of nearest neighbors on land surface. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1114–1125.
[49] Da Yan, Zhou Zhao, and Wilfred Ng. 2012. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 942–951.
[50] Yinzhao Yan and Raymond Chi-Wing Wong. 2021. Path Advisor: a multi-functional campus map tool for shortest path. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2683–2686.
[51] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Efficient shortest path queries on 3d weighted terrain surfaces for moving objects. In *25th IEEE International Conference on Mobile Data Management (MDM)*. IEEE.
[52] Yinzhao Yan and Raymond Chi-Wing Wong. 2024. Proximity queries on point clouds using rapid construction path oracle. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 1, 1–26.
[53] Xianwei Zheng, Hanjiang Xiong, Jianya Gong, and Linwei Yue. 2016. A virtual globe-based multi-resolution tin surface modeling and visualizetion method. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41 (2016).

A SUMMARY OF ALL NOTATION

Table 3 shows a summary of all notation.

B COMPARISON OF ALL ALGORITHMS

Table 4 shows a comparison of all algorithms.

C EMPIRICAL STUDIES

C.1 Experimental Results for Height Maps

C.1.1 ***Baseline comparisons.*** We study the effect of ϵ and n for proximity queries on height maps for baseline comparisons. We compared algorithm *Sur-SimQue-AdpM*, *Net-SimQue-AdpM*, *Mes-SimQue-AdpM*, *Mem-SimQue*, *Unf-Que-AdpM*, *Ste-Que-AdpM*, *Dij-Que-AdpM*, *Con-Que-AdpM*, and *Eff-Que* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 12, Figure 13, Figure 15, Figure 16, and Figure 17, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_m*-small*, EP_m*-small*, GF_m*-small*, LM_m*-small*, and RM_m*-small* dataset by setting n to be 10k for baseline comparisons. In Figure 18, Figure 20, Figure 22, Figure 24, and Figure 26, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_m, EP_m, GF_m, LM_m, and RM_m dataset by setting n to be 0.5M for baseline comparisons. The simplification time of *Mem-SimQue* is much smaller than that of *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* due to the efficient height map shortest path query and efficient Property 1 checking.

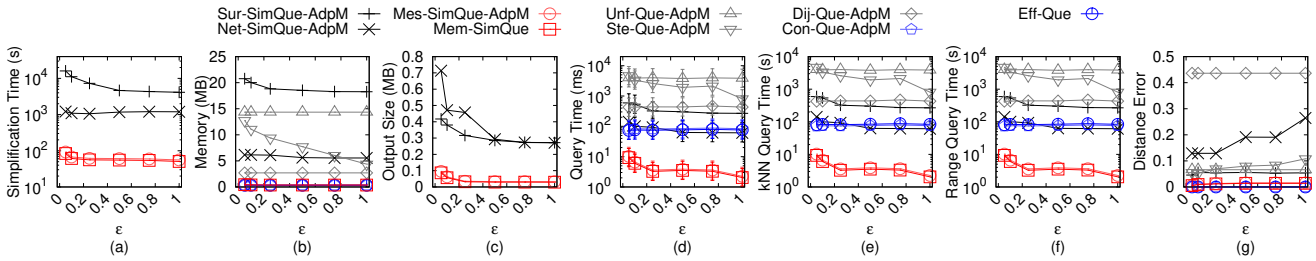

Figure 12: Baseline comparisons (effect of ϵ on BH_m-*small* height map dataset)

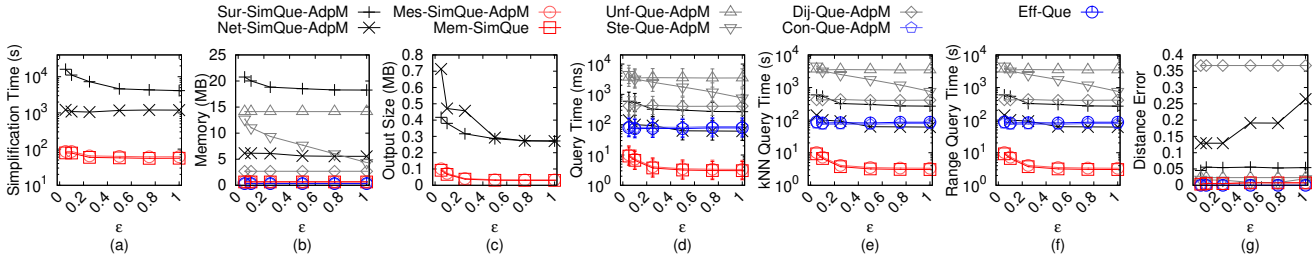

Figure 13: Baseline comparisons (effect of ϵ on EP_m-*small* height map dataset)

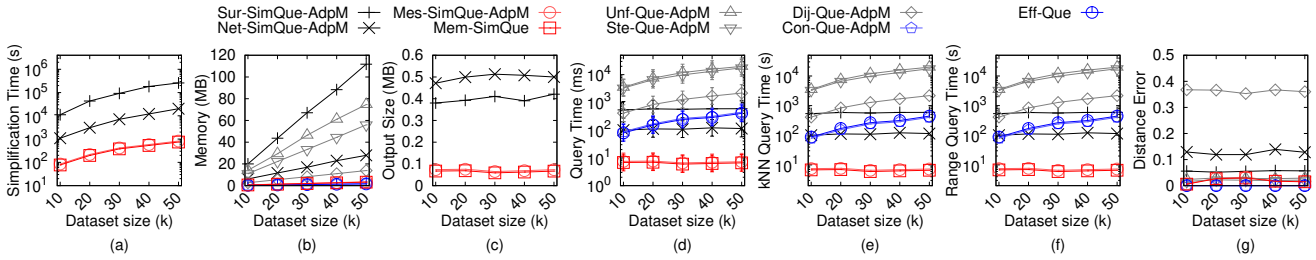

Figure 14: Baseline comparisons (effect of n on EP_m-*small* height map dataset)

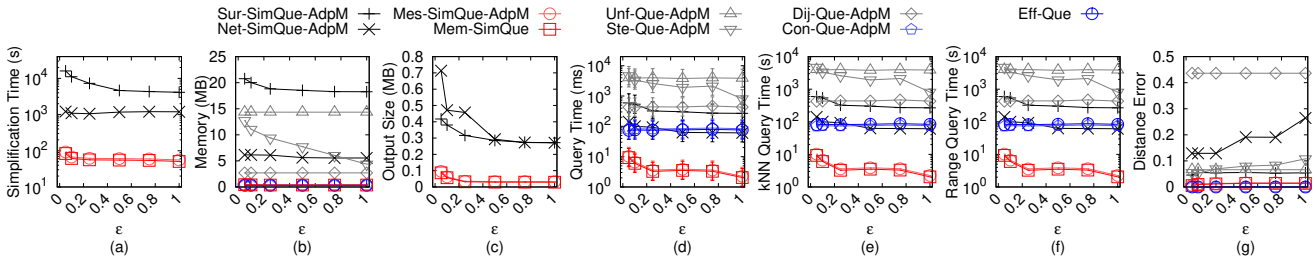

Figure 15: Baseline comparisons (effect of ϵ on GF_m-*small* height map dataset)

The output size of *Mem-SimQue* is also much smaller than that of *Sur-SimQue-AdpM* and *Net-SimQue-AdpM* due to the novel memory saving technique. The shortest path query time and the *kNN* query time of *Mem-SimQue* are also small since its simplified height map has a small output size. The distance error of *Mem-SimQue* is small (close to 0).

Effect of n (scalability test): In Figure 14, we tested 5 values of n from {50, 100, 150, 200, 250} on EP_m-*small* dataset by setting ϵ to be 0.1 for baseline comparisons. In Figure 19, Figure 21, Figure 23, Figure 25, and Figure 27, we tested 5 values of n from {500, 1000, 1500, 2000, 2500} on BH_m, EP_m, GF_m, LM_m, and RM_m dataset by setting ϵ to be 0.25 for baseline comparisons. *Mem-SimQue* performs better than all the remaining algorithms. The simplification time of *Mem-SimQue* is salable for a height map with 2.5M pixels. *Mes-SimQue-AdpM* performs similarly as *Mem-SimQue*, since they have the same simplification process, and the height map and point cloud have the same conceptual graph. *Eff-Que* performs better than *Unf-Que-AdpM* and *Ste-Que-AdpM* in terms of the range query time since it calculates the shortest path passing on a height map.

C.1.2 ***Ablation study for proximity query algorithms.*** **Effect of k and r**: In Figure 28, Figure 30, Figure 32, Figure 34, and

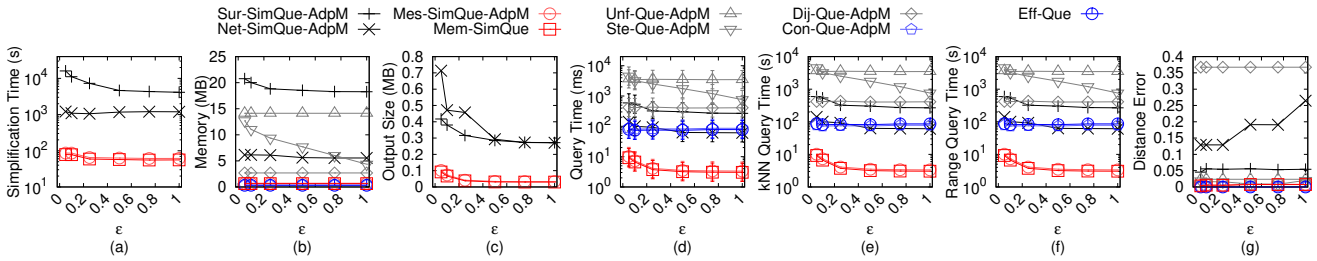

Figure 16: Baseline comparisons (effect of ϵ on LM_m-*small* height map dataset)

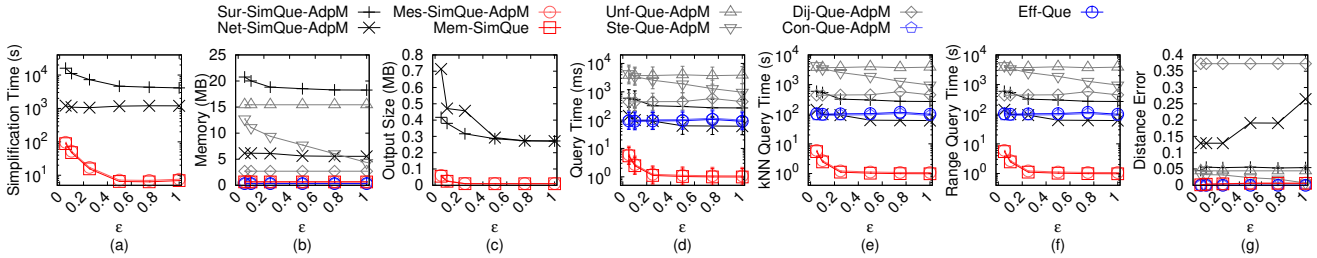

Figure 17: Baseline comparisons (effect of ϵ on RM_m-*small* height map dataset)

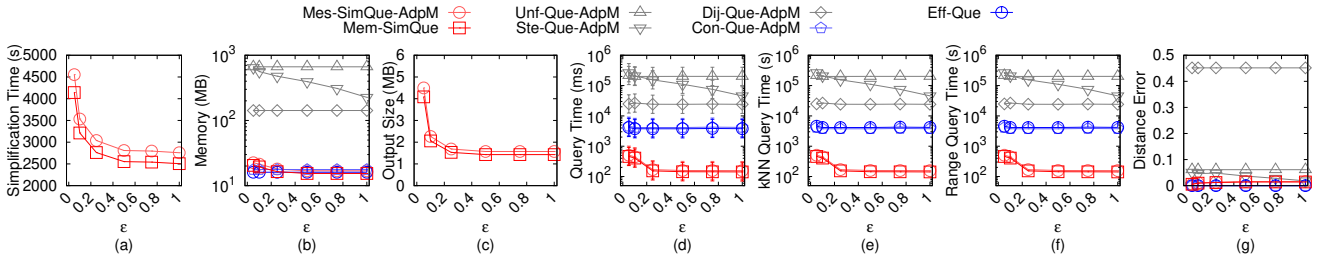

Figure 18: Baseline comparisons (effect of ϵ on BH_m height map dataset)

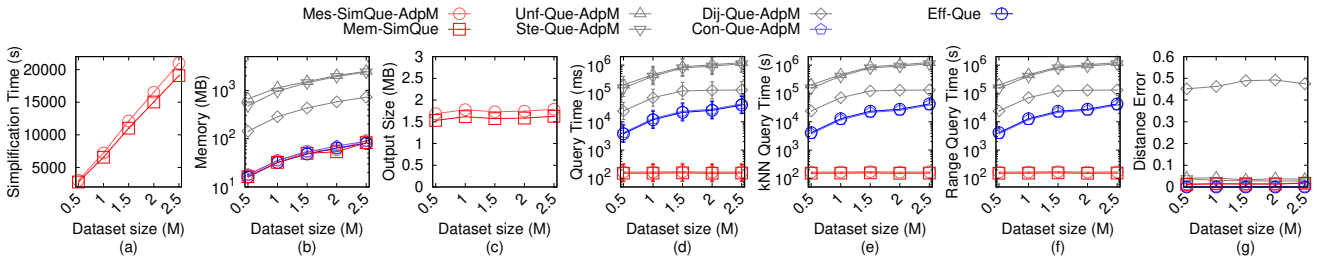

Figure 19: Baseline comparisons (effect of n on BH_m height map dataset)

Figure 36, we tested 5 values of k from {200, 400, 600, 800, 1000} on BH_m, EP_m, GF_m, LM_m, and RM_m dataset by setting ϵ to be 0.25 and n to be 0.5M for ablation study for proximity query algorithms (among algorithm *Mem-SimQue*, *Mem-SimQue-LQT1*, and *Mem-SimQue-LQT2*). In Figure 29, Figure 31, Figure 33, Figure 35, and Figure 37, we tested 5 values of r from {2km, 4km, 6km, 8km, 10km} on BH_m, EP_m, GF_m, LM_m, and RM_m dataset by setting ϵ to be 0.25 and n to be 0.5M for ablation study for proximity query algorithms (among algorithm *Mem-SimQue*, *Mem-SimQue-LQT1*, and *Mem-SimQue-LQT2*). *Mem-SimQue* performs better than both *Mem-SimQue-LQT1* and *Mem-SimQue-LQT2*, since we use the efficient algorithm for querying. Varying k does not affect the *kNN* query time of *Mem-SimQue*, since we append the paths calculated by Dijkstra's algorithm and the intra-paths as the path results, and we do not know the distance relationships among these paths before we perform a linear scan on them. But, a smaller r value can reduce the range query time of *Mem-SimQue*, since we can terminate Dijkstra's algorithm earlier when the searching distance is larger than r. *Mem-SimQue-LQT1* are *Mem-SimQue-LQT2* not affected by k and r since they need to calculate all the paths, and then perform a linear scan on these paths.

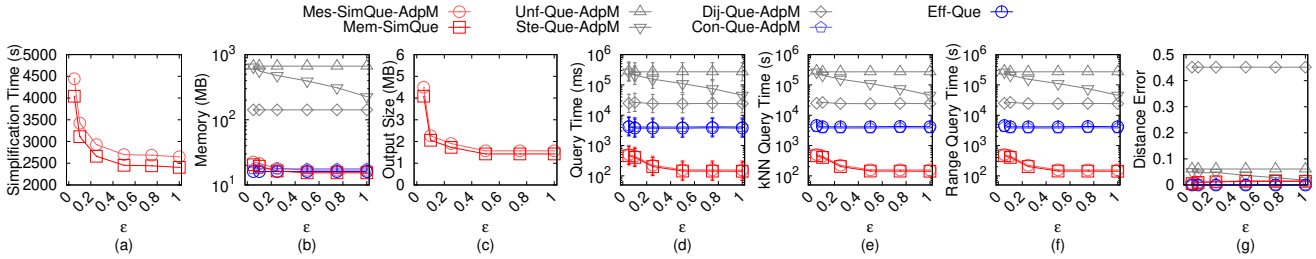

Figure 20: Baseline comparisons (effect of ϵ on EP_m height map dataset)

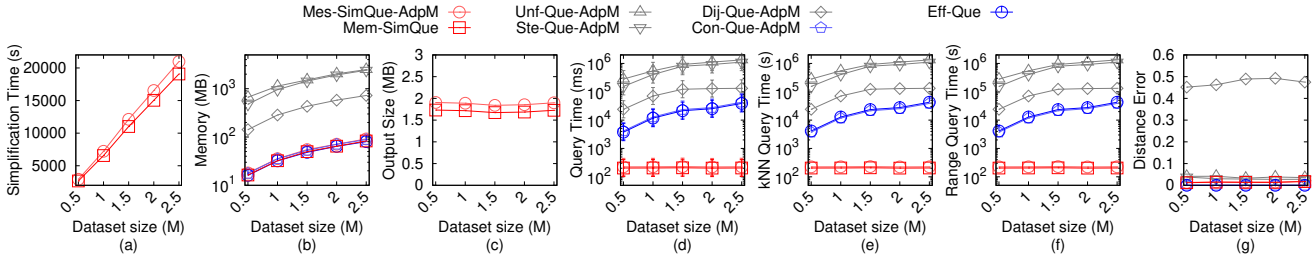

Figure 21: Baseline comparisons (effect of n on EP_m height map dataset)

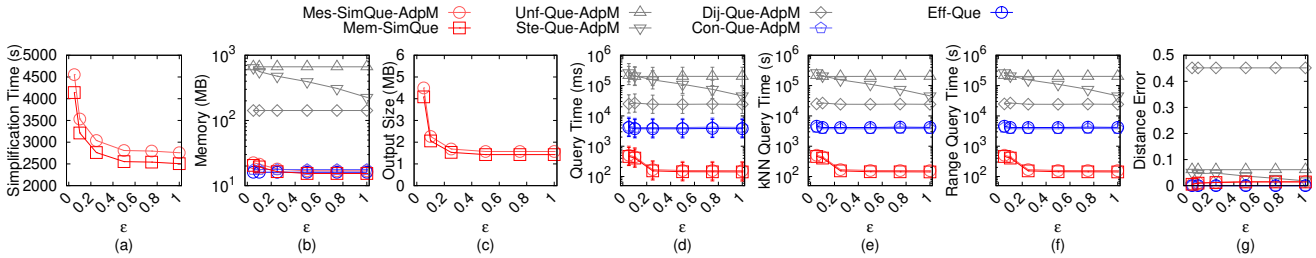

Figure 22: Baseline comparisons (effect of ϵ on GF_m height map dataset)

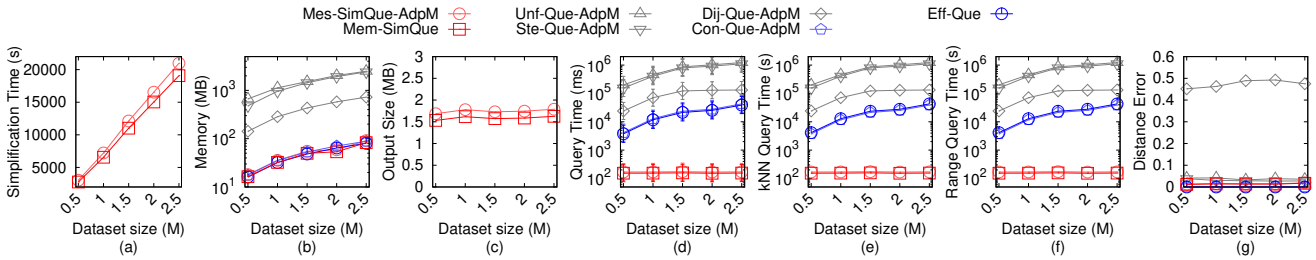

Figure 23: Baseline comparisons (effect of n on GF_m height map dataset)

C.1.3 ***Ablation study for simplification algorithms.*** In Figure 38, Figure 39, Figure 40, Figure 41, and Figure 42, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_m*-small*, EP_m*-small*, GF_m*-small*, LM_m*-small*, and RM_m*-small* dataset by setting n to be 0.5M for ablation study for simplificaiton algorithms (among algorithm *Mem-SimQue*, *Mem-SimQue-LS*, and *Mem-SimQue-LST*). *Mem-SimQue* still performs better than these two variations, showing the effectiveness of our merging and checking techniques.

C.2 Experimental Results for Point Clouds

We study the effect of ϵ and n for proximity queries on point clouds for baseline comparisons to justify why our proximity queries on height maps are useful. We compared algorithm *Sur-SimQue-AdpC*, *Net-SimQue-AdpC*, *Mes-SimQue*, *Mem-SimQue-AdpC*, *Unf-Que-AdpC*, *Ste-Que-AdpC*, *Dij-Que-AdpC*, *Con-Que*, and *Eff-Que-AdpC* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 43, Figure 44, Figure 46, Figure 47, and Figure 48, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75,

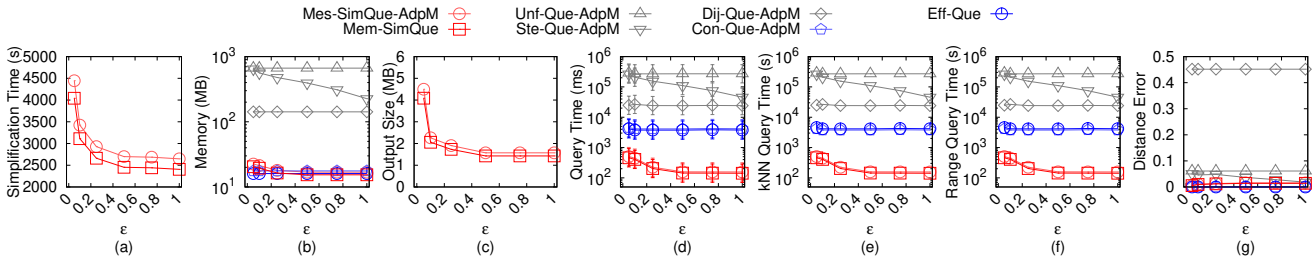

Figure 24: Baseline comparisons (effect of ϵ on LM_m height map dataset)

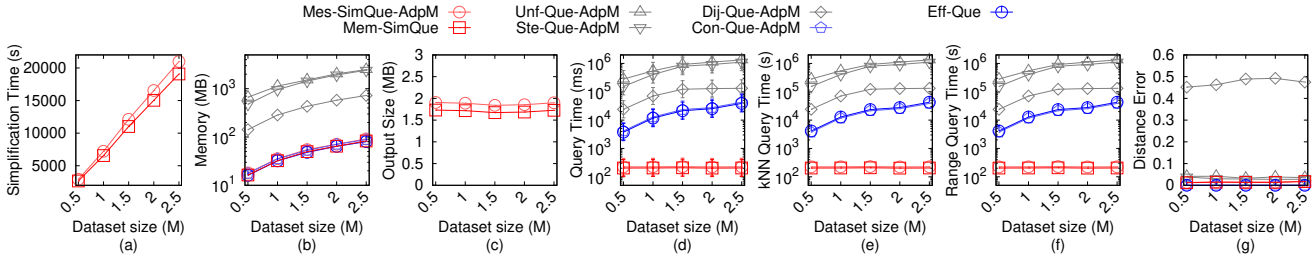

Figure 25: Baseline comparisons (effect of n on LM_m height map dataset)

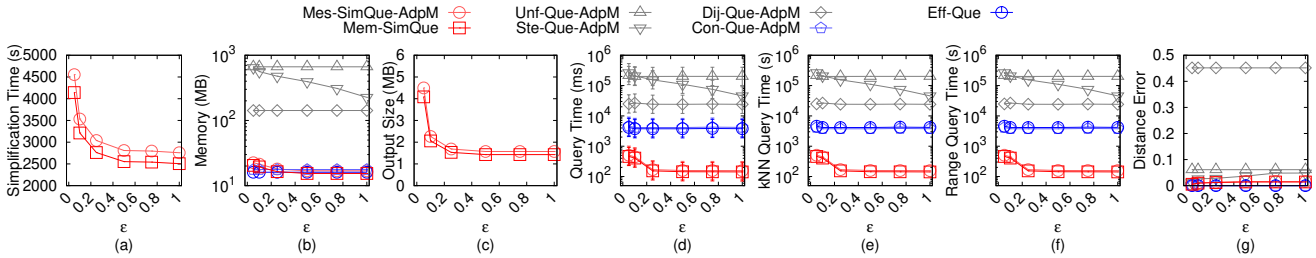

Figure 26: Baseline comparisons (effect of ϵ on RM_m height map dataset)

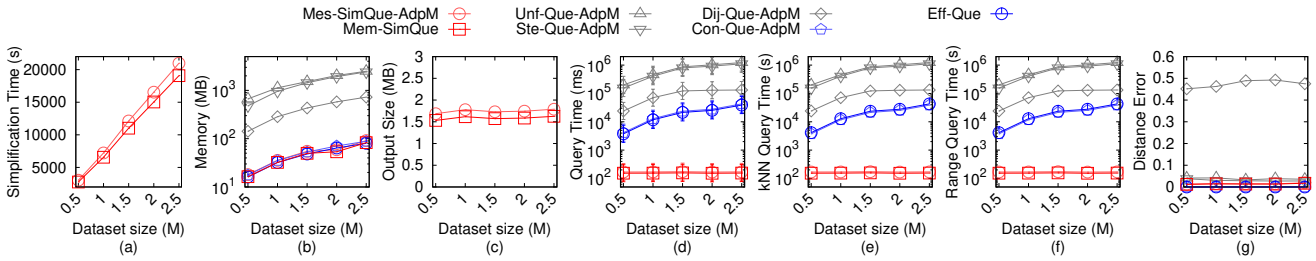

Figure 27: Baseline comparisons (effect of n on RM_m height map dataset)

1} on BH_c-*small*, EP_c-*small*, GF_c-*small*, LM_c-*small*, and RM_c-*small* dataset by setting n to be 10k for baseline comparisons. In Figure 49, Figure 51, Figure 53, Figure 55, and Figure 57, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_c, EP_c, GF_c, LM_c, and RM_c dataset by setting n to be 0.5M for baseline comparisons. *Mem-SimQue-AdpC* still performs better than other baselines.

Effect of n (scalability test): In Figure 45, we tested 5 values of n from {50, 100, 150, 200, 250} on EP_c-*small* dataset by setting ϵ to be 0.1 for baseline comparisons. In Figure 50, Figure 52, Figure 54, Figure 56, and Figure 58, we tested 5 values of n from {500, 1000, 1500, 2000, 2500} on BH_c, EP_c, GF_c, LM_c, and RM_c dataset by setting ϵ to be 0.25 for baseline comparisons. *Mem-SimQue-AdpC* performs better than all the remaining algorithms. The simplification time of *Mem-SimQue-AdpC* is salable for a height map with 2.5M pixels. *Mes-SimQue* performs similarly as *Mem-SimQue-AdpC*, since they have the same simplification process, and the height map and point cloud have the same conceptual graph.

C.3 Experimental Results for *TIN*s

We study the effect of ϵ and n for proximity queries on *TIN*s for baseline comparisons to justify why our proximity queries on height maps are useful. We compared algorithm *Sur-SimQue*, *Net-SimQue*,

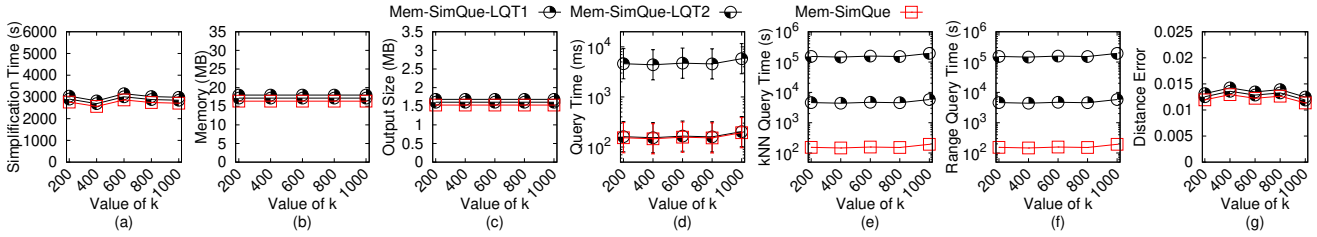

Figure 28: Ablation study for proximity query algorithms (effect of k on BH_m height map dataset)

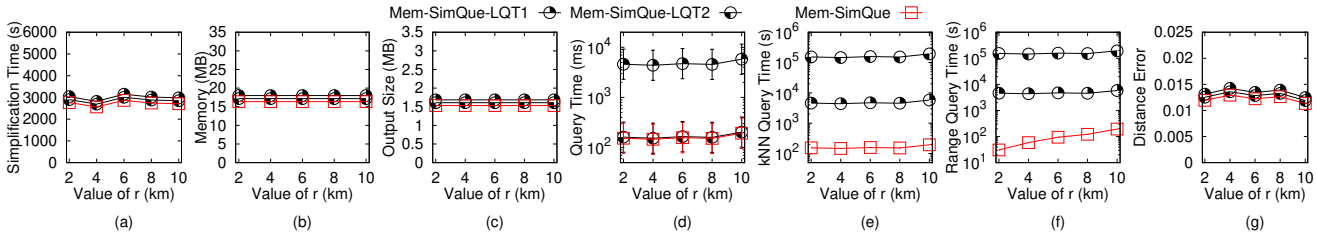

Figure 29: Ablation study for proximity query algorithms (effect of r on BH_m height map dataset)

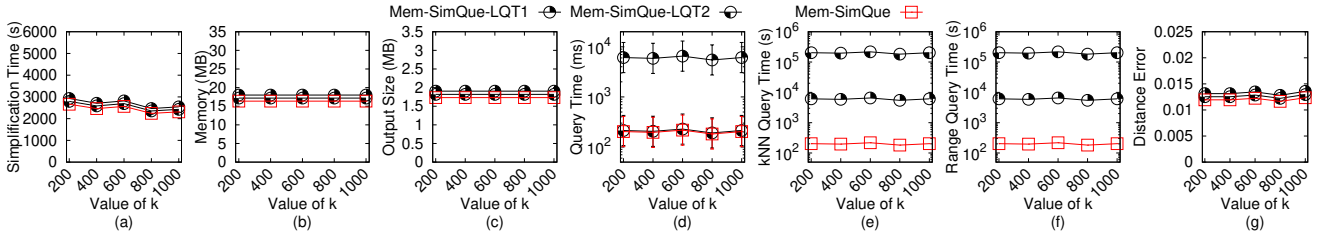

Figure 30: Ablation study for proximity query algorithms (effect of k on EP_m height map dataset)

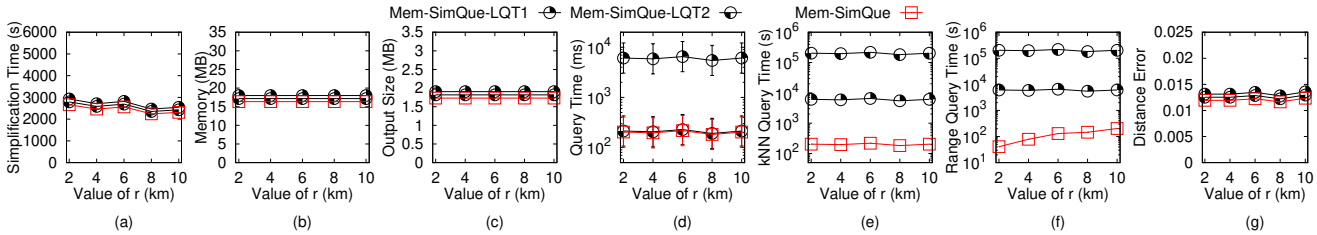

Figure 31: Ablation study for proximity query algorithms (effect of r on EP_m height map dataset)

Mes-SimQue-AdpT, *Mem-SimQue-AdpT*, *Unf-Que*, *Ste-Que*, *Dij-Que*, *Con-Que-AdpT*, and *Eff-Que-AdpT* on small-version datasets, and compared all algorithms except the first two algorithm on original datasets.

Effect of ϵ: In Figure 59, Figure 60, Figure 62, Figure 63, and Figure 64, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_t*-small*, EP_t*-small*, GF_t*-small*, LM_t*-small*, and RM_t*-small* dataset by setting n to be 10k for baseline comparisons. In Figure 65, Figure 67, Figure 69, Figure 71, and Figure 73, we tested 6 values of ϵ from {0.05, 0.1, 0.25, 0.5, 0.75, 1} on BH_t, EP_t, GF_t, LM_t, and RM_t dataset by setting n to be 0.5M for baseline comparisons. Although a *TIN* is given as input, *Mem-SimQue-AdpT* performs better than *Sur-SimQue* and *Net-SimQue* in terms of the simplification time, output size, and shortest path query time. The range query time of *Eff-Que-AdpT* is 100 times smaller than that of *Unf-Que* (although *Eff-Que-AdpT* needs to construct a height map from the given *TIN*, and there are no other additional steps for *Unf-Que*). The distance error of *Eff-Que-AdpT* is 0.06, but the distance error of *Dij-Que* is 0.45.

Effect of n (scalability test): In Figure 61, we tested 5 values of n from {50, 100, 150, 200, 250} on EP_t*-small* dataset by setting ϵ to be 0.1 for baseline comparisons. In Figure 66, Figure 68, Figure 70, Figure 72, and Figure 74, we tested 5 values of n from {500, 1000, 1500, 2000, 2500} on BH_t, EP_t, GF_t, LM_t, and RM_t dataset by setting ϵ to be 0.25 for baseline comparisons. *Mem-SimQue-AdpT* still performs better than other baselines.

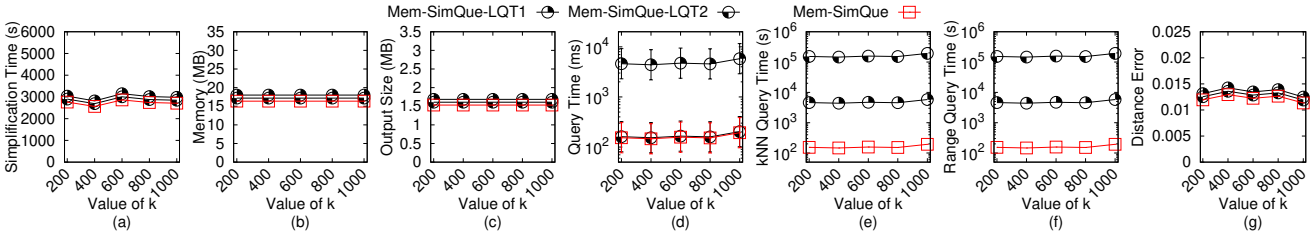

Figure 32: Ablation study for proximity query algorithms (effect of k on GF_m height map dataset)

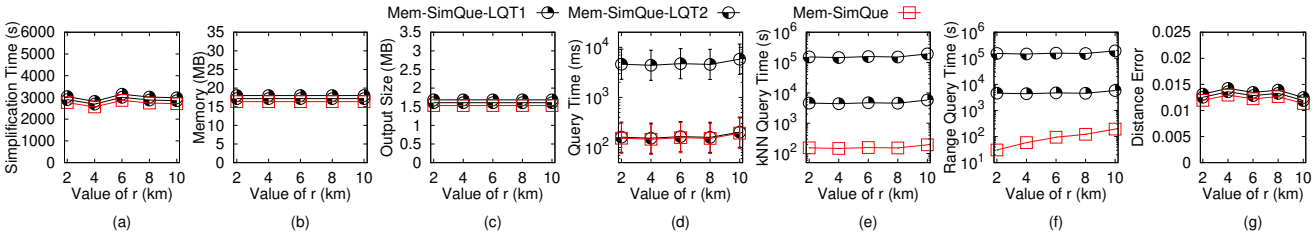

Figure 33: Ablation study for proximity query algorithms (effect of r on GF_m height map dataset)

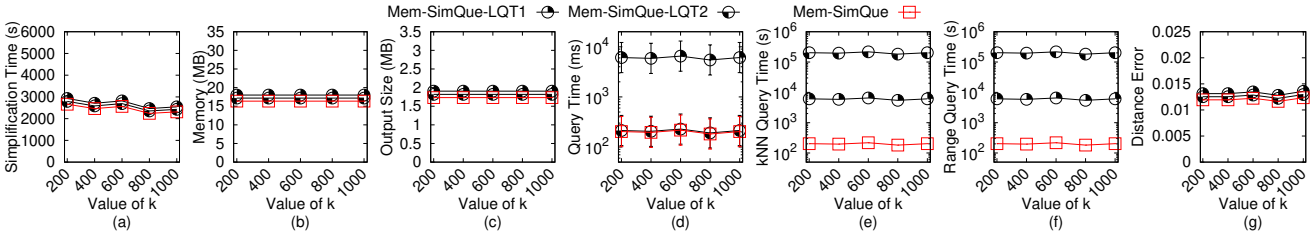

Figure 34: Ablation study for proximity query algorithms (effect of k on LM_m height map dataset)

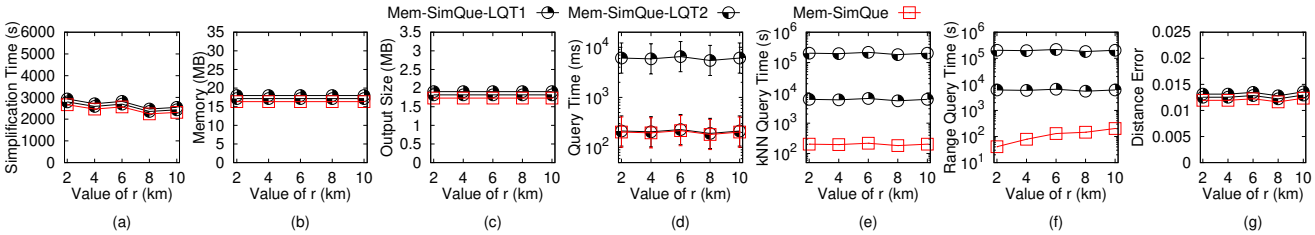

Figure 35: Ablation study for proximity query algorithms (effect of r on LM_m height map dataset)

D PROOF

Proof of Theorem 2.1. The proof is by transforming Minimum T-Spanner Problem [12] in Problem 2, which is an *NP-complete* problem, to the Height Map Simplification Problem.

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

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

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

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

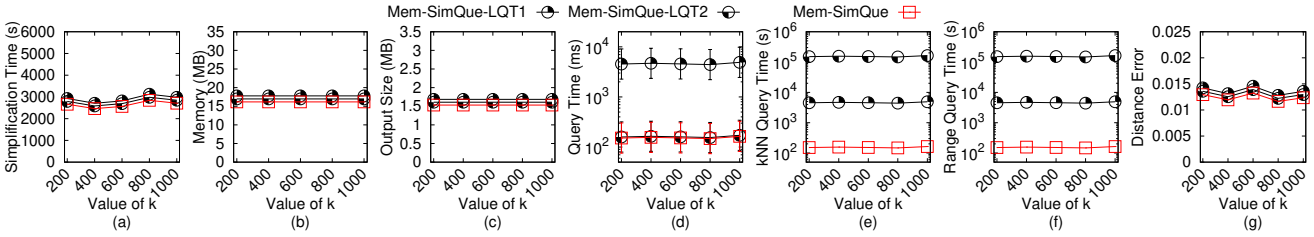

Figure 36: Ablation study for proximity query algorithms (effect of k on RM_m height map dataset)

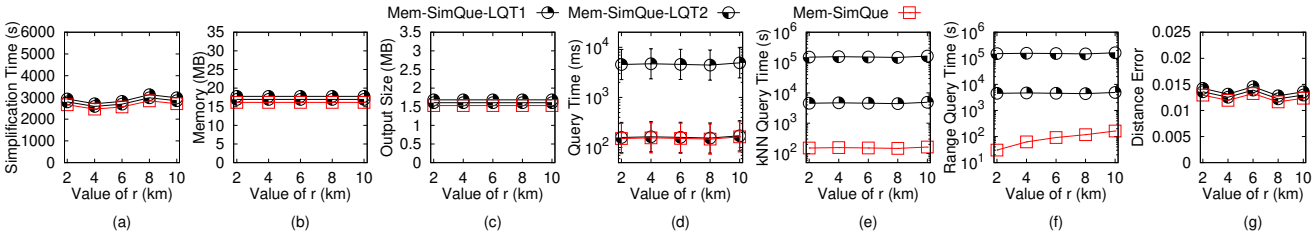

Figure 37: Ablation study for proximity query algorithms (effect of r on RM_m height map dataset)

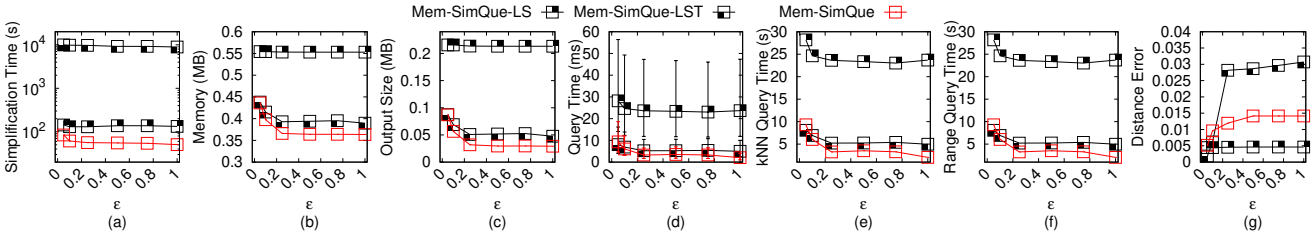

Figure 38: Ablation study for simplification algorithms on BH_m-*small* height map dataset

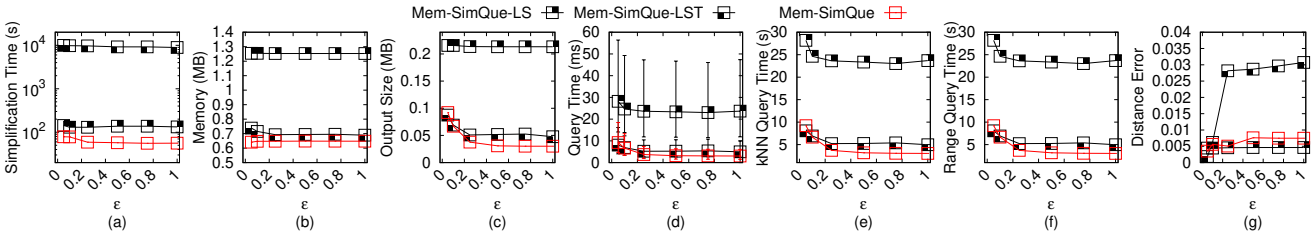

Figure 39: Ablation study for simplification algorithms on EP_m-*small* height map dataset

vertices in $\widetilde{G}$ (i.e., all possible added pixels in $\widetilde{M}$). Figure 75 (a) shows a height map, Figure 75 (b) shows a complete conceptual graph of the height map in a 2D plane. In Figure 75 (b), (1) each orange point represents 1 vertex with the same x-, y-, and z-coordinate of the corresponding vertex in the original conceptual graph G, and (2) each green point represents 256 vertices with the same x- and y-coordinate values of the possible new vertex, but with 256 different z-coordinate values in [0, 255] (because in a height map, a pixel with different grayscale color can represent at most 256 different elevation value), (3) the middle green point with an orange outline represents (3a) 1 vertex with the same x-, y-, and z-coordinate of the corresponding vertex in G, and (3b) 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 any pairs of vertices in $G_C.V$, and these edges form $G_C.E$. Figure 75 (c) shows G_C with 4 + 256 = 260 vertices in 3D space. In this figure, there should have total 256 green points, but only 5 of them are shown for the sake of illustration. In addition, there should have an edge between each pair of points, we omit some of them for the sake of illustration. Clearly, G and $\widetilde{G}$ are both sub-graphs of G_C. Given a pair of vertices s and t in $G_C.V$, and the original conceptual graph G, let $\Pi(s, t|G_C)$ be the shortest path between s and t passing on G_C, and we set $\Pi(s, t|G_C) = \Pi(s, t|G)$. We can simply regard $\Pi(s, t|G_C)$ as a function, such that given s, t, and G, it can return a

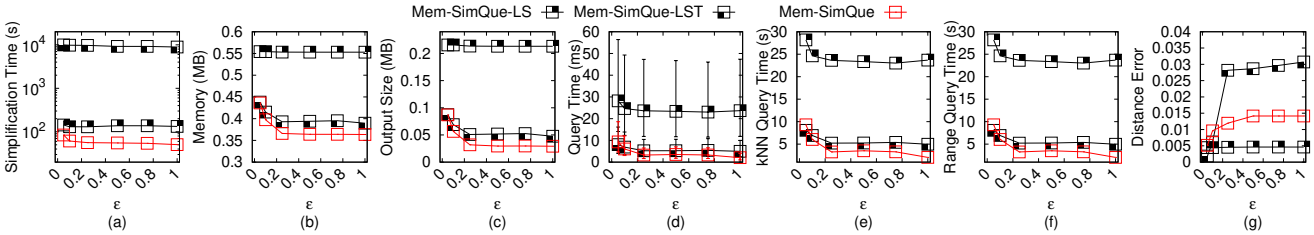

Figure 40: Ablation study for simplification algorithms on GF_m-*small* height map dataset

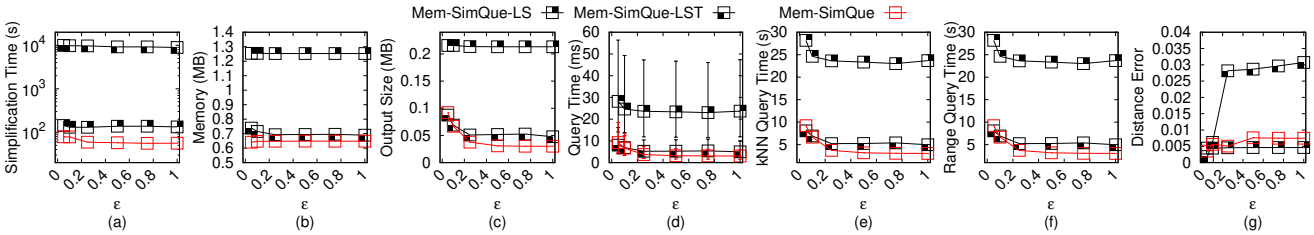

Figure 41: Ablation study for simplification algorithms on LM_m-*small* height map dataset

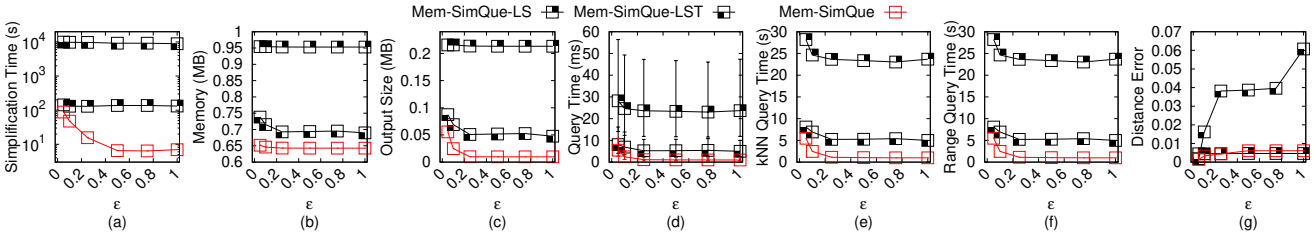

Figure 42: Ablation study for simplification algorithms on RM_m-*small* height map dataset

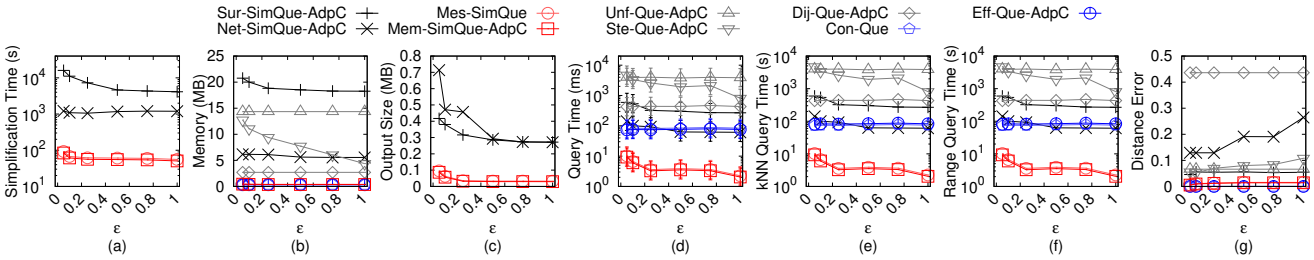

Figure 43: Baseline comparisons (effect of ϵ on BH_c-*small* point cloud dataset)

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

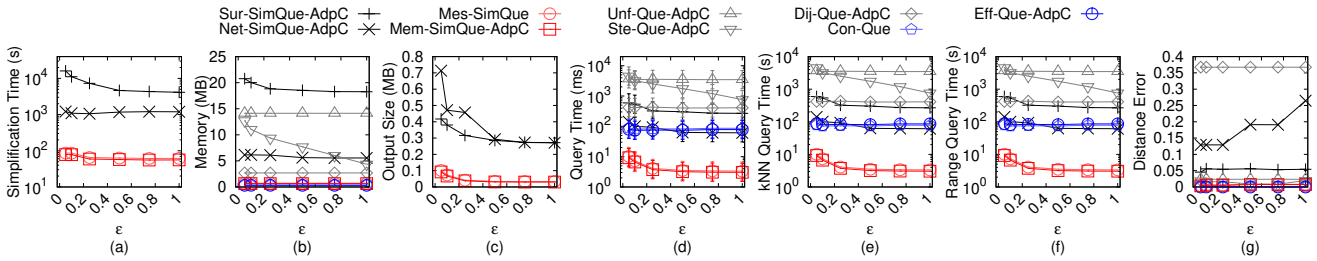

Figure 44: Baseline comparisons (effect of ϵ on EP_c-*small* point cloud dataset)

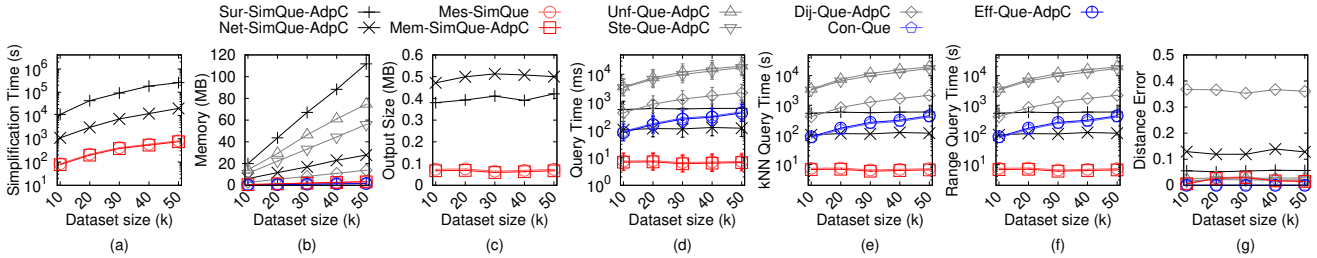

Figure 45: Baseline comparisons (effect of n on EP_c-*small* point cloud dataset)

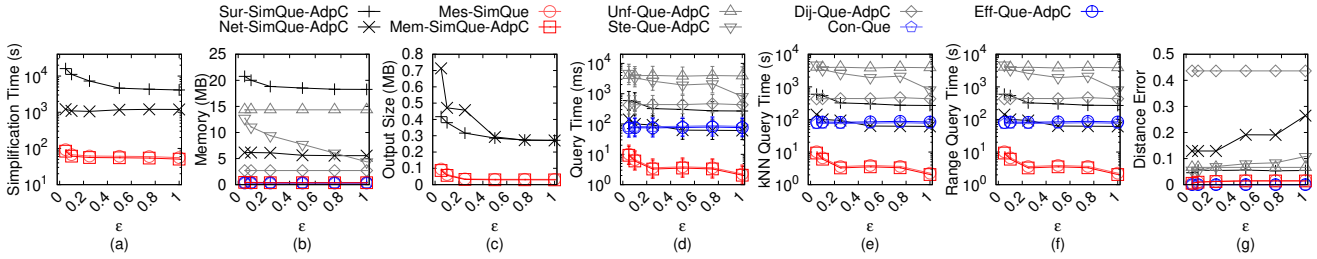

Figure 46: Baseline comparisons (effect of ϵ on GF_c-*small* point cloud dataset)

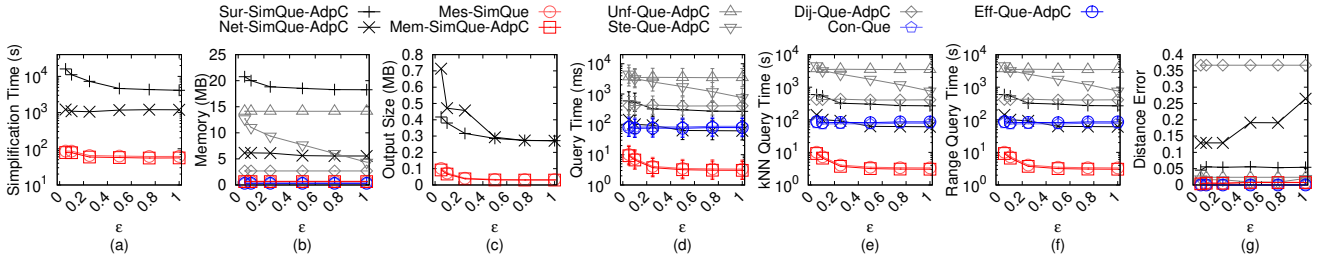

Figure 47: Baseline comparisons (effect of ϵ on LM_c-*small* point cloud dataset)

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

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

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

- Firstly, we show that $(1-\epsilon)|\Pi(s_1,t_1|M)| \leq |\Pi(s_1,t_1|\widetilde{M})|$. 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

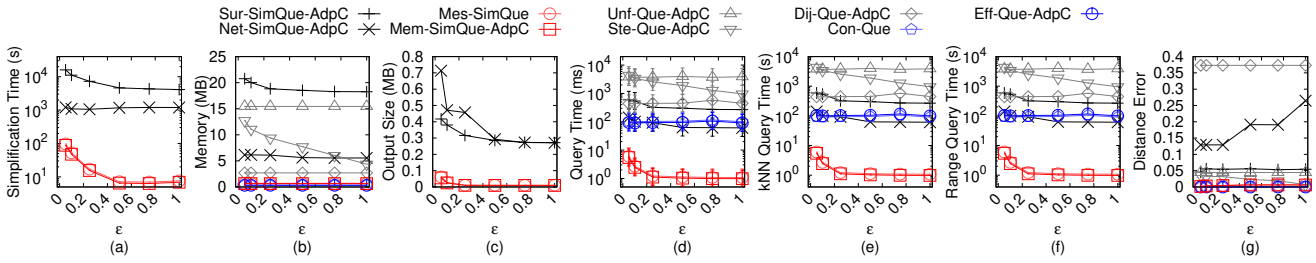

Figure 48: Baseline comparisons (effect of ϵ on RM_c-*small* point cloud dataset)

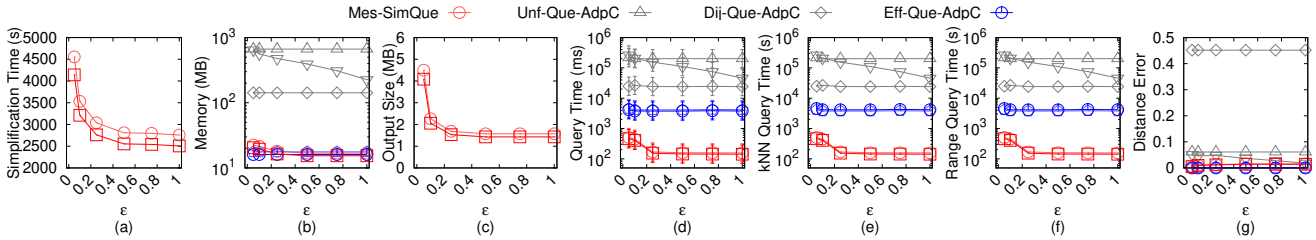

Figure 49: Baseline comparisons (effect of ϵ on BH_c point cloud dataset)

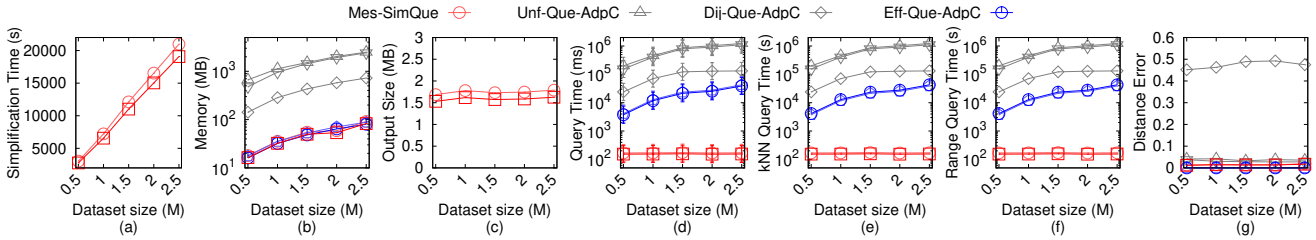

Figure 50: Baseline comparisons (effect of n on BH_c point cloud dataset)

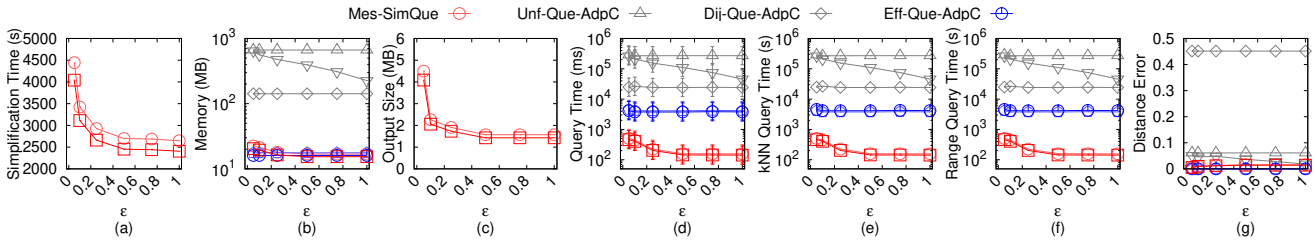

Figure 51: Baseline comparisons (effect of ϵ on EP_c point cloud dataset)

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

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

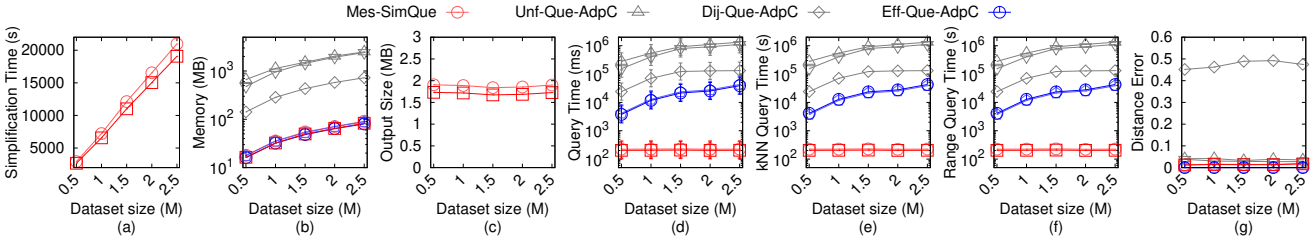

Figure 52: Baseline comparisons (effect of n on EP_c point cloud dataset)

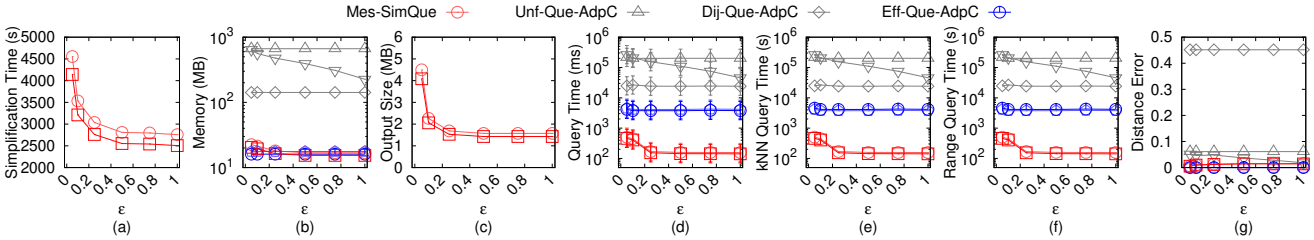

Figure 53: Baseline comparisons (effect of ϵ on GF_c point cloud dataset)

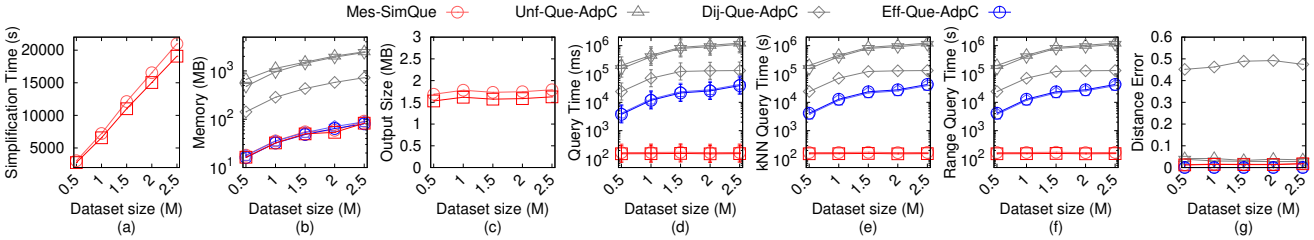

Figure 54: Baseline comparisons (effect of n on GF_c point cloud dataset)

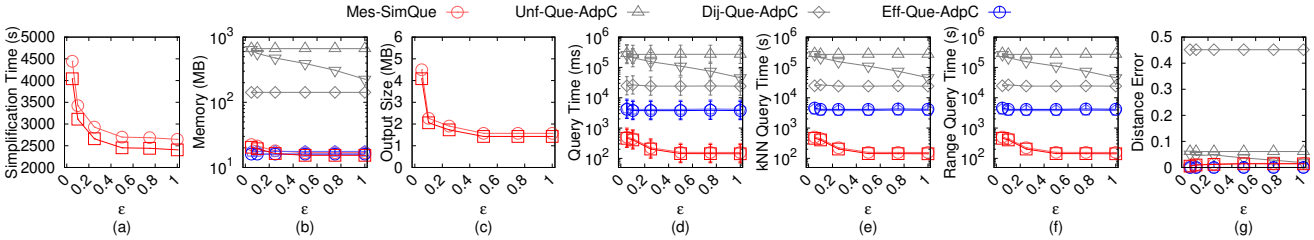

Figure 55: Baseline comparisons (effect of ϵ on LM_c point cloud dataset)

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 any pairs of pixels s_1 and t_1 both in P_{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 P_{add} be the added pixel 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 pixel between $\Pi(s_1, t_1|\widetilde{M})$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1, t_1|\widetilde{M})| = |\Pi(s_1, \overline{p}|\widetilde{M})| + |\Pi(\overline{p}, \overline{q}|\widetilde{M})| + |\Pi(\overline{q}, t_1|\widetilde{M})|$. Since $\overline{p}$ and $\overline{q}$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $(1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| \leq |\Pi(\overline{p}, \overline{q}|\widetilde{M})|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}$ and $\overline{q}$ are also in P_{rema}, we have $(1-\epsilon)|\Pi(s_1, \overline{p}|M)| \leq |\Pi(s_1, \overline{p}|\widetilde{M})|$ 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 $|\Pi(s_1, t_1|\widetilde{M})| = |\Pi(s_1, \overline{p}|\widetilde{M})| + |\Pi(\overline{p}, \overline{q}|\widetilde{M})| + |\Pi(\overline{q}, t_1|\widetilde{M})| \geq (1-\epsilon)|\Pi(s_1, \overline{p}|M)| + (1-\epsilon)|\Pi(\overline{p}, \overline{q}|M)| + (1-\epsilon)|\Pi(\overline{q}, t_1|M)| \geq (1-\epsilon)|\Pi(s_1, t_1|M)|$.

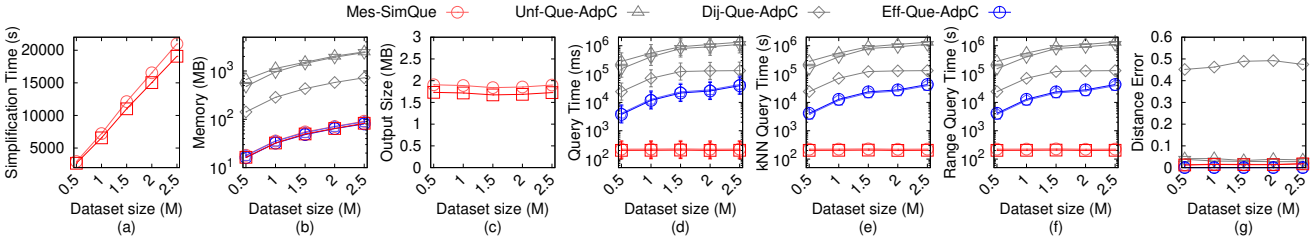

Figure 56: Baseline comparisons (effect of n on LM_c point cloud dataset)

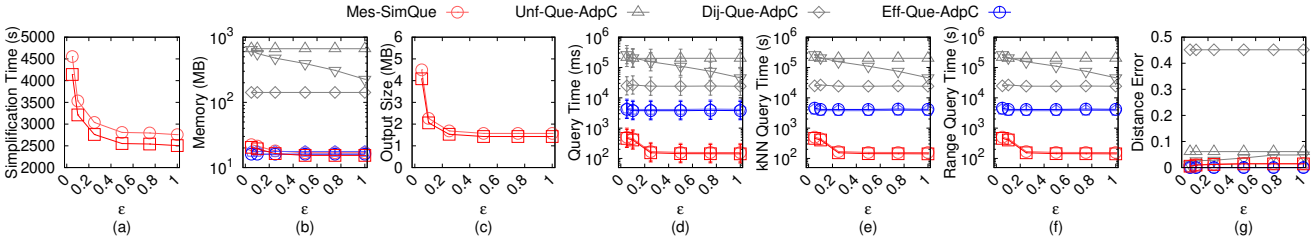

Figure 57: Baseline comparisons (effect of ϵ on RM_c point cloud dataset)

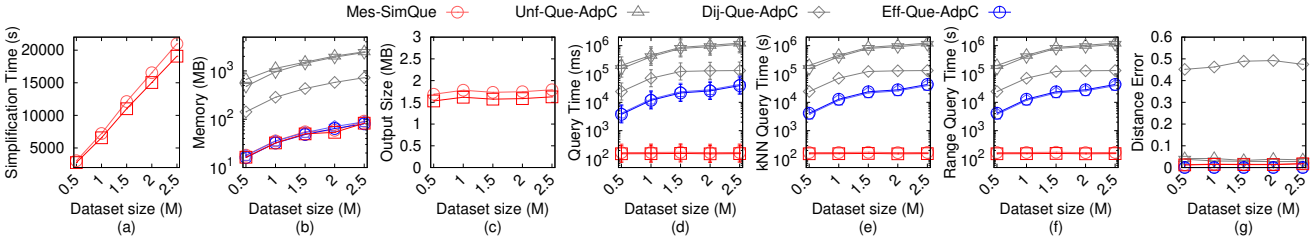

Figure 58: Baseline comparisons (effect of n on RM_c point cloud dataset)

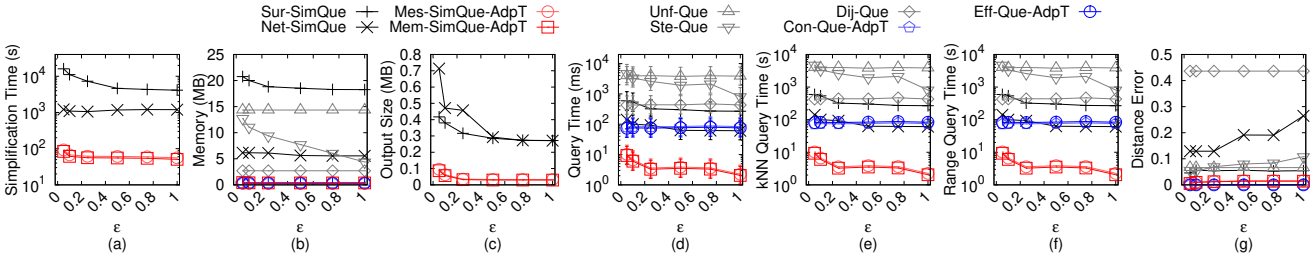

Figure 59: Baseline comparisons (effect of ϵ on BH_t *-small TIN* dataset)

- 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 pixel between $\Pi(s_1, t_1|M)$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_1, t_1|M)| = |\Pi(s_1, \overline{p}'|M)| + |\Pi(\overline{p}', \overline{q}'|M)| + |\Pi(\overline{q}', t_1|M)|$. Since $\overline{p}'$ and $\overline{q}'$ are in P_{rema}, and they are remaining neighbour pixels of linked added pixels of P_{add}, we have $|\Pi(\overline{p}', \overline{q}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{p}', \overline{q}'|M)|$ due to the *R2R* distance checking. Since s_1 and t_1 are in P_{rema}, and $\overline{p}'$ and $\overline{q}'$ are also in P_{rema}, we have $|\Pi(s_1, \overline{p}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, \overline{p}'|M)|$ 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 any pairs of pixels s_1 and t_1 both in P_{rema}, $(1-\epsilon)|\Pi(s_1, t_1|M)| \leq |\Pi(s_1, t_1|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_1, t_1|M)|$. □

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

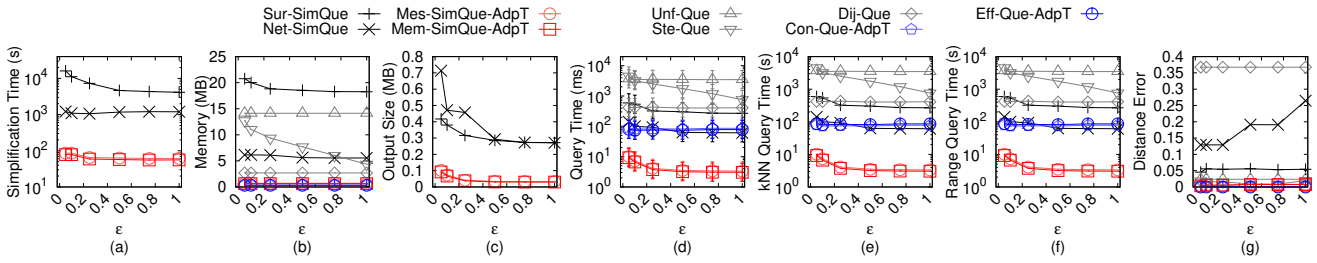

Figure 60: Baseline comparisons (effect of ϵ on EP_t-*small TIN* dataset)

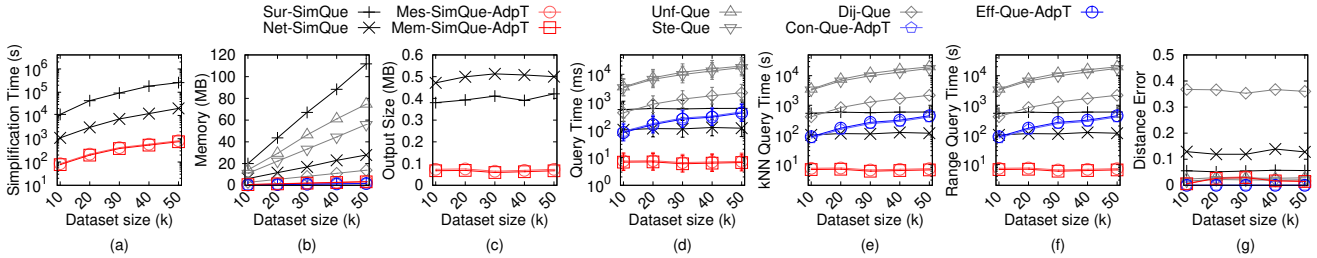

Figure 61: Baseline comparisons (effect of n on EP_t-*small TIN* dataset)

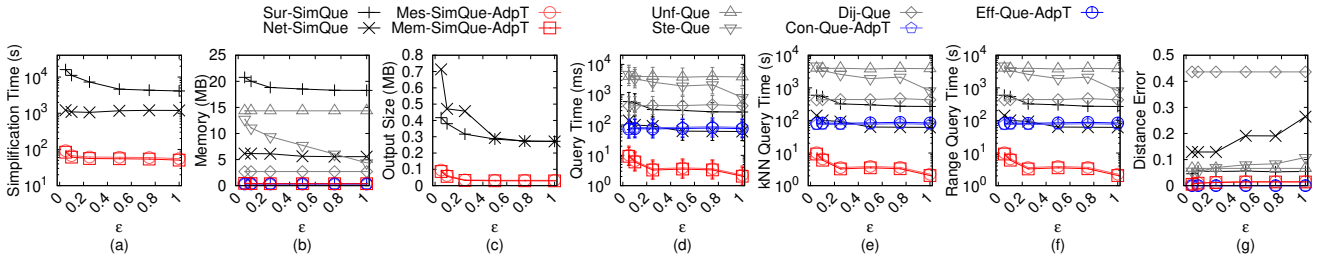

Figure 62: Baseline comparisons (effect of ϵ on GF_t-*small TIN* dataset)

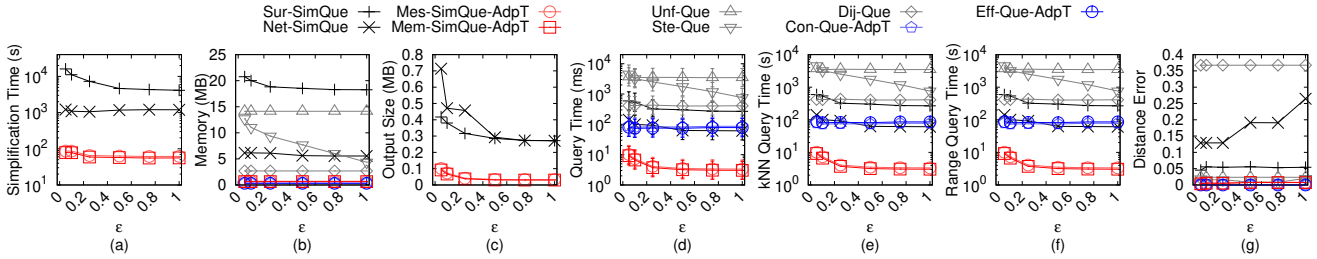

Figure 63: Baseline comparisons (effect of ϵ on LM_t-*small TIN* dataset)

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

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

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

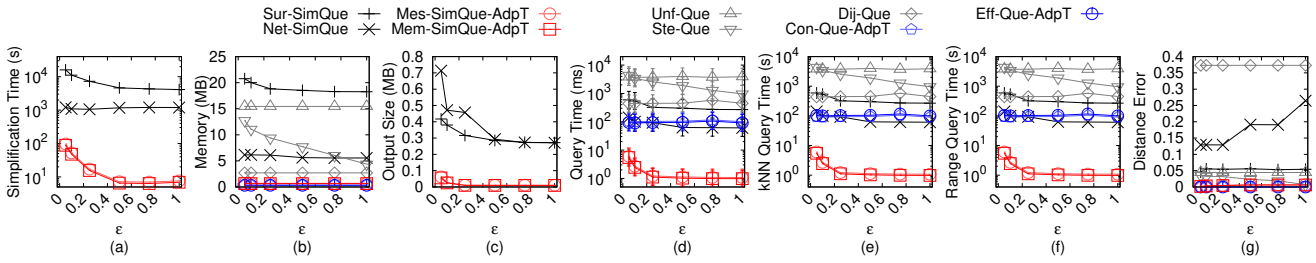

Figure 64: Baseline comparisons (effect of ϵ on *RM_t-small TIN* dataset)

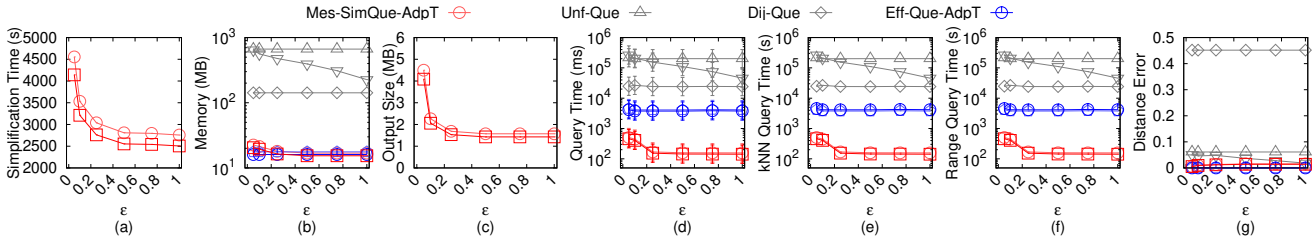

Figure 65: Baseline comparisons (effect of ϵ on *BH_t TIN* dataset)

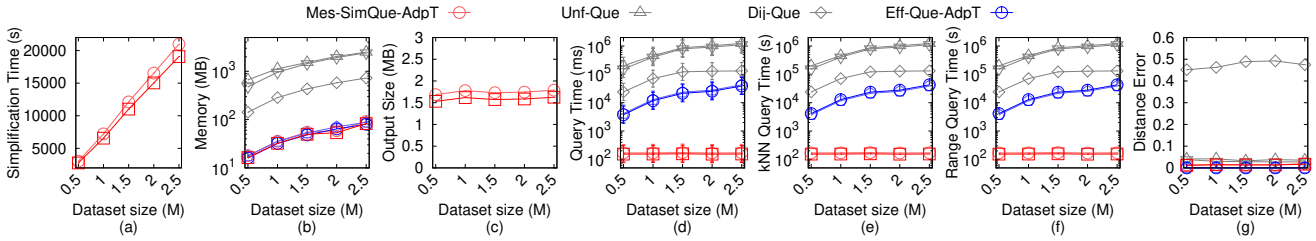

Figure 66: Baseline comparisons (effect of n on *BH_t TIN* dataset)

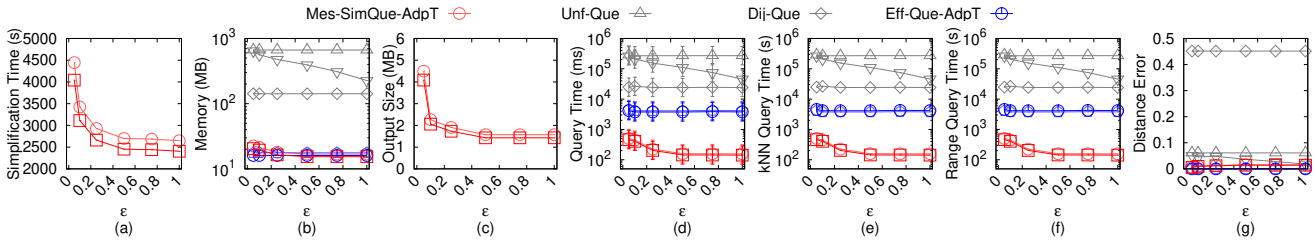

Figure 67: Baseline comparisons (effect of ϵ on *EP_t TIN* dataset)

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 pixel between $\Pi(s_2, t_2|M)$ and the remaining neighbour pixels of linked added pixels of P_{add}. We have $|\Pi(s_2, t_2|M)| = |\Pi(s_2, \overline{m}'|M)| + |\Pi(\overline{m}', t_2|M)|$. Since $\overline{m}'$ is in P_{rema}, which is a remaining neighbour pixel of linked added pixels of P_{add}, and t_2 is in $P - P_{rema}$, we have $|\Pi(\overline{m}, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{m}, t_2|M)|$ due to the *R2D* distance checking. Since s_2 and $\overline{m}'$ are both in P_{rema}, we have $|\Pi(s_2, \overline{m}'|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, \overline{m}'|M)|$ 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 any pairs of pixels s_2 in P_{rema} and t_2 in $P - P_{rema}$, we have $(1-\epsilon)|\Pi(s_2, t_2|M)| \leq |\Pi(s_2, t_2|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_2, t_2|M)|$. We show that for the $(i+1)$-th simplification iteration, the inequality holds. Let P_{add} be the added pixel in this iteration. Since the difference of $\widetilde{M}$ after the i-th simplification iteration and the $(i+1)$-th simplification iteration is due to the changes of P_{add}, we just need to show that the inequality holds when t_2 is any one of the deleted pixels belong to linked added pixels P_{add}. The proof is exactly the same as in the base case.

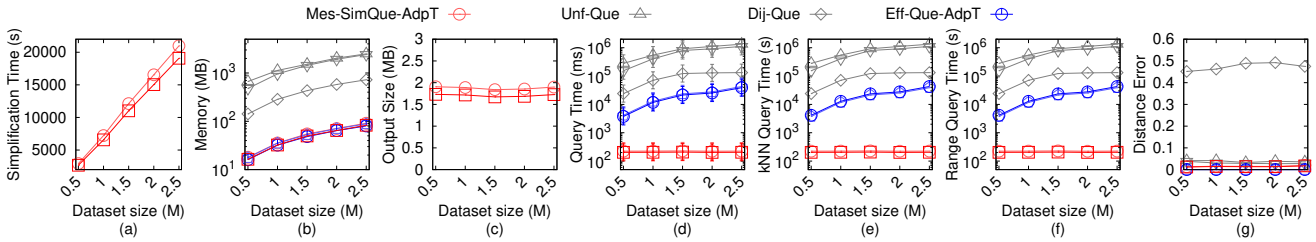

Figure 68: Baseline comparisons (effect of n on EP_t *TIN* dataset)

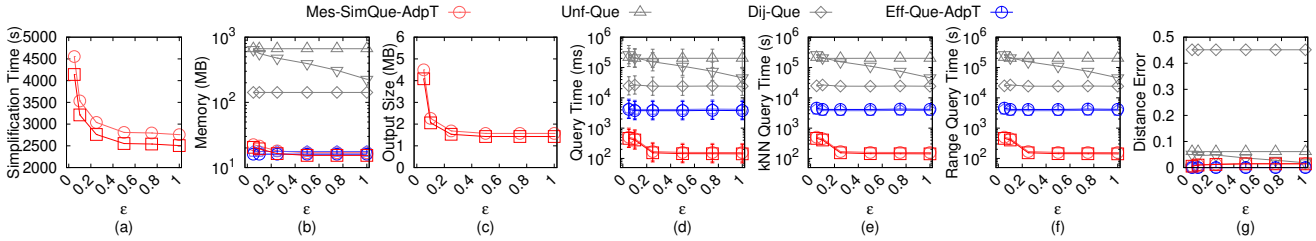

Figure 69: Baseline comparisons (effect of ϵ on GF_t *TIN* dataset)

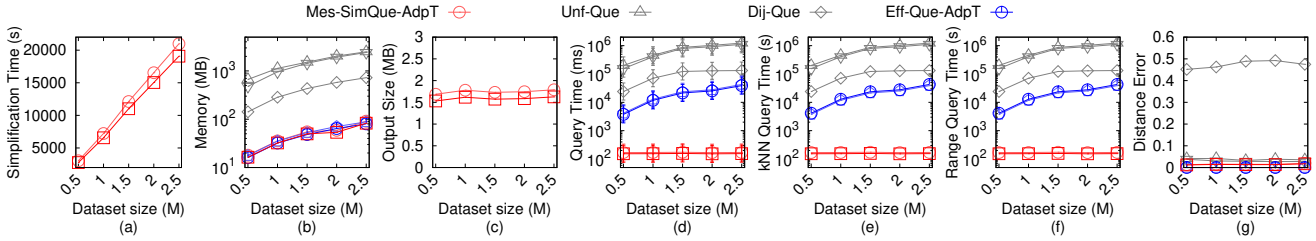

Figure 70: Baseline comparisons (effect of n on GF_t *TIN* dataset)

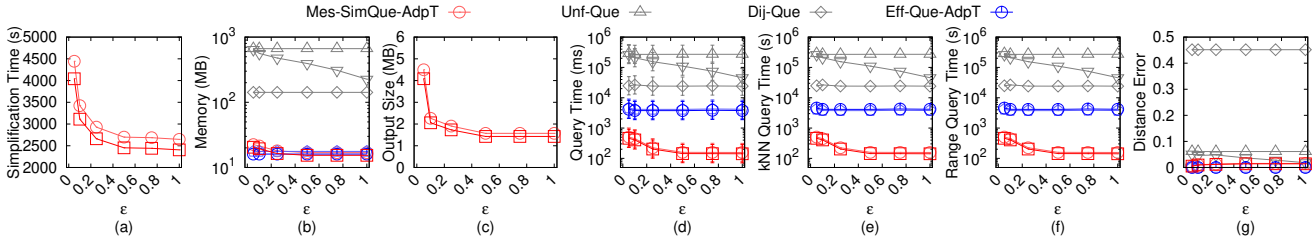

Figure 71: Baseline comparisons (effect of ϵ on LM_t *TIN* dataset)

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

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

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

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

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

For the hypothesis case, assume that after the i-th simplification iteration, for any pairs of pixels s_3 and t_3 both in $P - P_{rema}$, we have

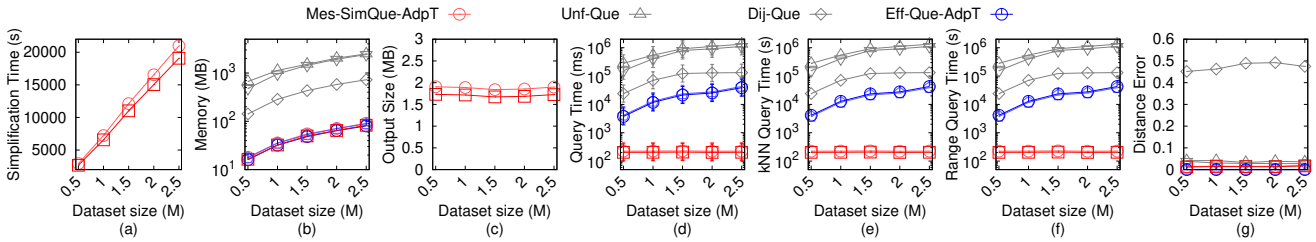

Figure 72: Baseline comparisons (effect of n on LM_t *TIN* dataset)

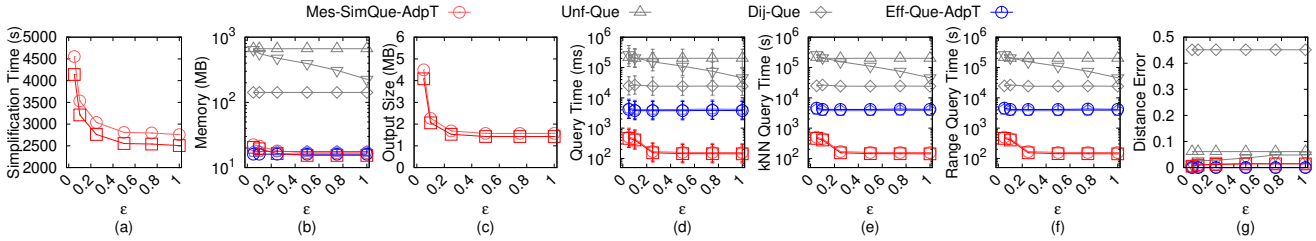

Figure 73: Baseline comparisons (effect of ϵ on RM_t *TIN* dataset)

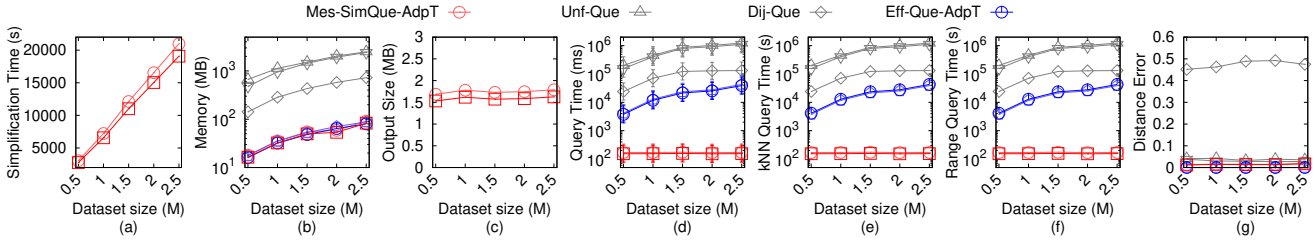

Figure 74: Baseline comparisons (effect of n on RM_t *TIN* dataset)

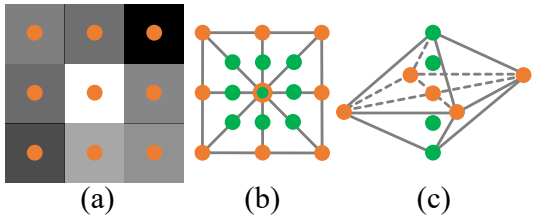

Figure 75: (a) A height map, (b) a complete conceptual graph of the height map in a 2D plane, and (c) a complete conceptual graph in a 3D space

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

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

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

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

Table 3: Summary of all notation

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

$|\Pi_1(\overline{q}',t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(\overline{q}',t_3|M)|$ 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 any pairs of pixels s_3 in P_{rema} and t_3 in $P-P_{rema}$, when $\Pi(s,t|\widetilde{M})$ passes on pixels in P_{rema}, $(1-\epsilon)|\Pi(s_3,t_3|M)| \leq |\Pi(s_3,t_3|\widetilde{M})| \leq (1+\epsilon)|\Pi(s_3,t_3|M)|$.

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

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

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

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

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

Proof of Lemma 4.4. Firstly, we prove the query time of both the *kNN* and range query algorithm.

- For algorithm *Eff-Que*, given a query pixel q, we just need to perform one Dijkstra's algorithm on M.
- For algorithm *Mem-SimQue*, given a query pixel q, if q is a remaining pixel, we just need to perform one Dijkstra's algorithm on $\widetilde{M}$; if q is a deleted pixel, we just need to perform $\widetilde{N}(\mathcal{B}(q))$

Table 4: Comparison of algorithms

Algorithm	Simplification time		Output size		Shortest path query time		*kNN* and range query time		Error
Simplification algorithm									
Sur-SimQue-AdpM [25, 28]	$O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$	Large	$O(n)$	Large	$O(n^2)$	Large	$O(n^2)$	Large	Small
Net-SimQue-AdpM [28]	$O(n^2\log n)$	Medium	$O(n)$	Large	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
Mes-SimQue-AdpM [52]	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
Mem-SimQue-LQT1	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	$O(\frac{n^2}{\log n}\log\frac{n}{\log n})$	Medium	Small
Mem-SimQue-LQT2	$O(n\sqrt[3]{n}\log n)$	Small	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{nn'}{\log n}\log\frac{n}{\log n})$	Medium	Small
Mem-SimQue-LS	$O(n\sqrt[3]{n}\log n)$	Small	$O(n)$	Large	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Small
Mem-SimQue-LST	$O(n^2\sqrt[3]{n}\log n)$	Large	$O(\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	$O(\frac{n}{\log n}\log\frac{n}{\log n})$	Small	Small
***Mem-SimQue* (ours)**	$\boldsymbol{O(n\sqrt[3]{n}\log n)}$	**Small**	$\boldsymbol{O(\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	$\boldsymbol{O(\frac{n}{\log n}\log\frac{n}{\log n})}$	**Small**	**Small**
Shortest path query algorithm									
Unf-Que-AdpM [13, 46]	-	N/A	-	N/A	$O(n^2)$	Large	$O(n^2)$	Large	Small
Ste-Que-AdpM [27, 51]	-	N/A	-	N/A	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	$O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$	Large	Small
Dij-Que-AdpM [28]	-	N/A	-	N/A	$O(n\log n)$	Medium	$O(n\log n)$	Medium	Medium
Con-Que-AdpM [52]	-	N/A	-	N/A	$O(n\log n)$	Medium	$O(n\log n)$	Medium	No error
***Eff-Que* (ours)**	-	N/A	-	N/A	$\boldsymbol{O(n\log n)}$	**Medium**	$\boldsymbol{O(n\log n)}$	**Medium**	**No error**

Dijkstra's algorithm on $\widetilde{M}$. Since $\widetilde{N}(\mathcal{B}(q))$ 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 *Eff-Que* and *Mem-SimQue*), respectively, the query time of both the *kNN* and range query by using algorithm *Eff-Que* is $O(n\log n)$ and *Mem-SimQue* is $O(\frac{n}{\log n}\log\frac{n}{\log n})$.

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

- For algorithm *Eff-Que*, it returns the exact shortest path passing on M, so it also returns the exact result for the *kNN* and range query.
- For algorithm *Mem-SimQue*, 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 calculated by algorithm *Eff-Que* as X, where X contains either (1a) k nearest objects to query object q, or (1b) objects within a range of distance r from q. Similarly, we denote the set of objects returned using the shortest distance on $\widetilde{M}$ calculated by algorithm *Mem-SimQue* as X', where X' contains either (2a) k nearest objects to query object q, or (2b) objects within a range of distance r from q. In Figure 1 (a), suppose that the exact k nearest objects ($k=2$) of a is c, d, i.e., $X=\{c,d\}$. Suppose that our *kNN* query algorithm finds the k nearest objects ($k=2$) of a is b, c, i.e., $X'=\{b,c\}$. Recall that let p_f (resp. p'_f) be the object in X (resp. X') that is furthest from q based on the shortest distance on M, i.e., $|\Pi(q,p_f|M)| \le \max_{\forall p\in X}|\Pi(q,p|M)|$ (resp. $|\Pi(q,p'_f|M)| \le \max_{\forall p'\in X'}|\Pi(q,p'|M)|$). We further let w_f (resp. w'_f) be the object in X (resp. X') that is furthest from q based on the shortest distance on $\widetilde{M}$ returned by algorithm *Mem-SimQue*, i.e., $|\Pi(q,w_f|\widetilde{M})| \le \max_{\forall w\in X}|\Pi(q,w|\widetilde{M})|$ (resp. $|\Pi(q,w'_f|\widetilde{M})| \le \max_{\forall w'\in X'}|\Pi(q,w'|\widetilde{M})|$). Recall the error rate of *kNN* and range queries is $\alpha=\frac{|\Pi(q,p'_f|M)|}{|\Pi(q,p_f|M)|}$. According to Theorem 4.1, we have $|\Pi(q,p'_f|\widetilde{M})| \ge (1-\epsilon)|\Pi(q,p'_f|M)|$. Thus, we have $\alpha \le \frac{|\Pi(q,p'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(q,p_f|M)|}$. By the definition of p_f and w_f, we have $|\Pi(q,p_f|M)| \ge |\Pi(q,w_f|M)|$. Thus, we have $\alpha \le \frac{|\Pi(q,p'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(q,w_f|M)|}$. By the definition of p'_f and w'_f, we have $|\Pi(q,p'_f|\widetilde{M})| \le |\Pi(q,w'_f|\widetilde{M})|$. Thus, we have $\alpha \le \frac{|\Pi(q,w'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(q,w_f|M)|}$. According to Theorem 4.1, we have $|\Pi(q,w_f|\widetilde{M})| \le (1+\epsilon)|\Pi(q,w_f|M)|$. Then, we have $\alpha \le \frac{(1+\epsilon)|\Pi(q,w'_f|\widetilde{M})|}{(1-\epsilon)|\Pi(q,w_f|\widetilde{M})|}$. By our *kNN* and range query algorithm, we have $|\Pi(q,w'_f|\widetilde{M})| \le |\Pi(q,w_f|\widetilde{M})|$. Thus, we have $\alpha \le \frac{1+\epsilon}{1-\epsilon}$. So, algorithm *Mem-SimQue* has an error rate $\frac{1+\epsilon}{1-\epsilon}$ for the *kNN* and range query.

□

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

Proof. Firstly, we prove the simplification time. It first needs to construct the *TIN* using the height map in $O(n)$ time. Then, in each vertex removal iteration, it places $O(\frac{1}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ Steiner points [20, 25] on each face adjacent to the deleted vertex, and use algorithm *Unf-Que* [13, 46] in $O(n^2)$ time to check the distances between these Steiner points on the original *TIN* and the simplified *TIN*, so this step needs $O(\frac{n^2}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$. In general, the total simplification time is $O(n+\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon}) = (\frac{n^3}{\sin\theta\sqrt{\epsilon}}\log\frac{1}{\epsilon})$.

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

Thirdly, we prove the shortest path query time. Since there are $O(n)$ vertices on $\widetilde{T}$, we use algorithm *Unf-Que* [13, 46] in $O(n^2)$ time for the shortest path query. Thus, the shortest path query time is $O(n^2)$.

Fourthly, we prove the *kNN* and range query time. Since we just need to use algorithm *Unf-Que* [13, 46] (i.e., a single-source-all-destination algorithm) once for both the *kNN* and range query, the *kNN* and range query time is $O(n^2)$.

Finally, we prove that for any pairs of vertices s and t on T, algorithm *Sur-SimQue-AdpM* 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 any 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 work [25], given any pair of points p and q (on the faces that are adjacent to the deleted vertex) on T, if $(1-\epsilon)|\Pi(u,v|T)| \leq |\Pi(u,v|\widetilde{T})| \leq (1+\epsilon)|\Pi(u,v|T)|$, then $(1-\epsilon)|\Pi(p,q|T)| \leq |\Pi(p,q|\widetilde{T})| \leq (1+\epsilon)|\Pi(p,q|T)|$. Following the similar proof in Theorem 4.2, we know that for any 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 any pairs of vertices s and t on T, algorithm *Sur-SimQue-AdpM* has $(1-\epsilon)|\Pi(s,t|T)| \leq |\Pi(s,t|\widetilde{T})| \leq (1+\epsilon)|\Pi(s,t|T)|$. □

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

Proof. Firstly, we prove the simplification time. It first needs to construct the *TIN* using the height map in $O(n)$ time. Then, in each vertex removal iteration, it uses algorithm *Dij-Que* [28] in $O(n \log n)$ time to check the distances between any pair of vertices that are neighbours of the deleted vertex on the original *TIN* and the simplified *TIN*. Since there are only $O(1)$ vertices that are neighbours of the deleted vertex, this step needs $O(n \log n)$ time. Since there are total $O(n)$ vertex removal iterations, the simplification time for simplifying a *TIN* is $O(n^2 \log n)$. In general, the total simplification time is $O(n + n^2 \log n) = O(n^2 \log n)$.

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

Thirdly, we prove the shortest path query time. Since there are $O(n)$ vertices on $\widetilde{T}$, we use algorithm *Dij-Que* [28] in $O(n \log n)$ time for the shortest path query. Thus, the shortest path query time is $O(n \log n)$.

Fourthly, we prove the *kNN* and range query time. Since we just need to use algorithm *Dij-Que* [28] (i.e., a single-source-all-destination algorithm) once for both the *kNN* and range query, the *kNN* and range query time is $O(n \log n)$.

Finally, we prove that for any pairs of vertices s and t on T, algorithm *Net-SimQue-AdpM* 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 any 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 any pairs of vertices s and t on T, algorithm *Net-SimQue-AdpM* 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 the work [28]. □

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

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

The output size, shortest path query time, *kNN* and range query time, and error bound of algorithm *Mes-SimQue-AdpM* are the same as algorithm *Mem-SimQue*. □

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

Proof. Firstly, we prove the shortest path query time. Since it needs to use Dijkstra's algorithm with each pixel in $\widetilde{N}(\mathcal{B}(s))$ or $\widetilde{N}(\mathcal{B}(t))$ as a source to calculate inter-path, and the size of $\widetilde{N}(\mathcal{B}(s))$ or $\widetilde{N}(\mathcal{B}(t))$ is $O(n)$, so its shortest path query time is $O(n)$ times the shortest path query time of algorithm *Mem-SimQue*. 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 *Mem-SimQue-LQT1* once for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{n^2}{\log n}\log\frac{n}{\log n})$.

The simplification time, output size, and error bound of algorithm *Mem-SimQue-LQT1* are the same as algorithm *Mem-SimQue*. □

Theorem D.8. *The simplification time, output size, shortest path query time, and kNN and range query time of algorithm Mem-SimQue-LQT2 are $O(n\sqrt[3]{n}\log n)$, $O(\frac{n}{\log n})$, $O(\frac{n}{\log n}\log\frac{n}{\log n})$, and $O(\frac{nn'}{\log n}\log\frac{n}{\log n})$, respectively. Given a height map M, it returns a 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 *Mem-SimQue* n' times for both the *kNN* and range query, the *kNN* and range query time is $O(\frac{nn'}{\log n}\log\frac{n}{\log n})$.

The simplification time, output size, shortest path query time, and error bound of algorithm *Mem-SimQue-LQT2* are the same as algorithm *Mem-SimQue*. □

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

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

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

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

The simplification time and error bound of algorithm *Mem-SimQue-LS* are the same as algorithm *Mem-SimQue*. □

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

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

The output size, shortest path query time, *kNN* and range query time, and error bound of algorithm *Mem-SimQue-LST* are the same as algorithm *Mem-SimQue*. □

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

Proof. Firstly, we prove the shortest path query time. The proof of the shortest path query time of algorithm *Unf-Que* is in [13, 46]. But since algorithm *Unf-Que-AdpM* first needs to construct the *TIN* using the height map, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time is $O(n+n^2)=O(n^2)$.

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

Thirdly, we prove the memory consumption. The proof of the memory consumption of algorithm *Unf-Que* is in [13, 46], which is similar to algorithm *Unf-Que-AdpM*. Thus, the memory consumption is $O(n^2)$.

Finally, we prove the error bound. Compared with $\Pi(s,t|T)$, the proof that it returns the exact shortest path passing on a *TIN* is in [13, 46]. Since the *TIN* is constructed by the height map, so algorithm *Unf-Que-AdpM* returns the exact shortest surface path passing on a *TIN* (that is constructed by the height map). Compared with $\Pi(s,t|M)$, since we regard $\Pi(s,t|M)$ as the exact shortest path passing on the height map, algorithm *Unf-Que-AdpM* returns the approximate shortest path passing on a height map. □

Theorem D.12. *The shortest path query time, kNN and range query time, and memory consumption of algorithm Ste-Que-AdpM 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 $|\Pi_{Ste\text{-}Que\text{-}AdpM}(s,t|T)| \le (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T, where $\Pi_{Ste\text{-}Que\text{-}AdpM}(s,t|T)$ is the shortest surface path of algorithm Ste-Que-AdpM passing on a TIN T (that is constructed by the height map) between s and t. Compared with $\Pi(s,t|C)$, it returns the approximate shortest path passing on a height map.*

Proof. Firstly, we prove the shortest path query time. The proof of the shortest path query time of algorithm *Ste-Que* is in [27]. Note that in Section 4.2 of [27], the shortest path query time of algorithm *Ste-Que-AdpM* 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 [27], we obtain that its error bound ϵ is equal to $\frac{1}{K-1}$. Thus, we can derive that the shortest path query time of algorithm *Ste-Que-AdpM* 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 *Ste-Que-AdpM* 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 *Ste-Que-AdpM* first needs to construct a *TIN* using the point cloud, it needs an additional $O(n)$ time for this step. Thus, the shortest path query time of algorithm *Ste-Que-AdpM* 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 [51], it omits the constant term in the shortest path query time. After adding back these terms, the shortest path query time is the same.

Secondly, we prove the *kNN* and range query time. Since it is a single-source-all-destination algorithm, we use it once for

both the *kNN* and range query. So, the *kNN* and range query is $O(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}\log(\frac{l_{max}n}{\epsilon l_{min}\sqrt{1-\cos\theta}}))$.

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

Finally, we prove the error bound. Compared with $\Pi(s,t|T)$, the proof of the error bound of algorithm *Ste-Que-AdpM* is in [27, 51]. Since the *TIN* is constructed by the point cloud, so algorithm *Ste-Que-AdpM* always has $|\Pi_{Ste\text{-}Que\text{-}AdpM}(s,t|T)| \leq (1+\epsilon)|\Pi(s,t|T)|$ for any pairs of vertices s and t on T. Compared with $\Pi(s,t|C)$, since we regard $\Pi(s,t|C)$ as the exact shortest path passing on the point cloud, algorithm *Ste-Que-AdpM* returns the approximate shortest path passing on a point cloud. □

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

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

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

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

Finally, we prove the error bound. Recall that $\Pi_N(s,t|T)$ is the shortest network path passing on T (that is constructed by the height map) between s and t, so actually $\Pi_N(s,t|T)$ is the same as $\Pi_{Dij\text{-}Que\text{-}AdpM}(s,t|T)$. 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 $|\Pi_E(s,t|T)| \leq k\cdot|\Pi(s,t|T)|$ (according to left hand side equation in Lemma 2 of [28]) 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 $|\Pi_{Dij\text{-}Que\text{-}AdpM}(s,t|T)| \leq k\cdot|\Pi(s,t|T)|$ for any pairs of vertices s and t on T. Compared with $\Pi(s,t|C)$, since we regard $\Pi(s,t|C)$ as the exact shortest path passing on the height map, algorithm *Dij-Que-AdpM* returns the approximate shortest path passing on a height map. □

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

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

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

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

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