

Finding Shortest Path on Land Surface

Lian Liu

Raymond Chi-Wing Wong

The Hong Kong University of Science and Technology
Kowloon, Hong Kong
{lyanliu, raywong}@cse.ust.hk

ABSTRACT

Finding shortest paths is a fundamental operator in spatial databases. Recently, terrain datasets have attracted a lot of attention from both industry and academia. There are some interesting issues to be studied in terrain datasets which cannot be found in a traditional two-dimensional space. In this paper, we study one of the issues called a slope constraint which exists in terrain datasets. In this paper, we propose a problem of finding shortest paths with the slope constraint. Then, we show that this new problem is more general than the traditional problem of finding shortest paths without considering the slope constraint. Since finding shortest paths with the slope constraint is costly, we propose a new framework called surface simplification so that we can compute shortest paths with the slope constraint efficiently. Under this framework, the surface is “simplified” such that the complexity of finding shortest paths on this simplified surface is lower. We conducted experiments to show that the surface simplification is very efficient and effective not only for the new problem with the slope constraint but also the traditional problem without the slope constraint.

Categories and Subject Descriptors

H.2.8 [Database Applications]: Spatial databases and GIS; I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms, languages, and systems

General Terms

Algorithms, Designs, Experimentation, Performance, Measurement

Keywords

shortest path, spatial databases, terrain, land surface, surface simplification, triangular irregular network (TIN) model

1. INTRODUCTION

Recently, terrain datasets have attracted a lot of attention from both industry and academia [27]. In industry, Microsoft and Google

launched tools to display maps with terrain in “Bing Maps for Enterprise” (previously called “Microsoft Virtual Earth”) in Sept 2009 and “Google Earth” in 2005, respectively. In academia, the database community [7, 6, 23, 27] has started to pay attention to studying how to perform some spatial queries over terrain datasets. One example is *surface kNN (skNN) queries*. Given a source point s and a set Q of pre-defined points on the surface, a surface kNN query returns a set Q' containing k objects which are nearest to s . That is, for each point q_i in Q' and each point q_j in $Q - Q'$, $|s, q_i| \leq |s, q_j|$. Note that, in the context of terrain, $|s, q|$ corresponds to the length of the shortest *surface path* between s and q instead of the *Euclidean distance* between s and q . Figure 1 shows a terrain in Bearhead area in Washington State where s and t are two points on the surface. In this figure, p_e is a straight line between s and t whose length corresponds to the Euclidean distance between s and t , and p_s corresponds to the shortest *surface path* between s and t . For clarity, in the following, when we write *paths* (or *shortest paths*), we mean *surface paths* (or *shortest surface paths*).

There are some interesting issues to be studied in terrain datasets which cannot be found in a traditional two-dimensional space. In this paper, we study one of the issues called a *slope* constraint which exists in terrain datasets. Consider Figure 2 where s and t are two points on the surface. Path p_1 corresponds to the shortest path from s to t . Note that p_1 involves a *steep* route from s to the top of the mountain and it is difficult for people to travel via a steep route. However, path p_2 and path p_3 are smoother and do not involve any steep route. We say that these two paths satisfy the *slope* requirement. We define a *slope parameter* $\theta_m \in [0, \pi/2]$ to denote whether a path is steep. Intuitively, if θ_m is set to 0, then the path must be flat without any inclination. If θ_m is $\pi/2$, then the path can have any inclination. Different applications have different values of θ_m . For example, the steepest road in UK (which is in Ffordd Penllech, Harlech) has a slope of 0.331 (in radian) (i.e., 19.0°) and the steepest non-rack railway in Portugal has a slope of 0.13 (in radian) (i.e., 7.69°) [18]. Informally speaking, given a slope parameter θ_m , a path p is said to satisfy the slope requirement if the *slope* of path p is at most θ_m . We will give a formal definition for the slope of a path later in Section 3. We also call the path satisfying the slope requirement the *gentle path*. In Figure 2, path p_2 and path p_3 are gentle but path p_1 is not.

In the following, we study the following problem. Given a source point s and a destination point t on the surface, we want to find the shortest gentle path from s to t on the surface. We call this problem *finding shortest gentle paths (FSGP)*. Note that problem FSGP is more general than the *traditional problem* of finding shortest path on the surface adopted in the literature [1, 7, 6, 23, 27] (which does not consider any slope requirement). This is because, when θ_m is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'11, June 12–16, 2011, Athens, Greece.

Copyright 2011 ACM 978-1-4503-0661-4/11/06 ...\$10.00.

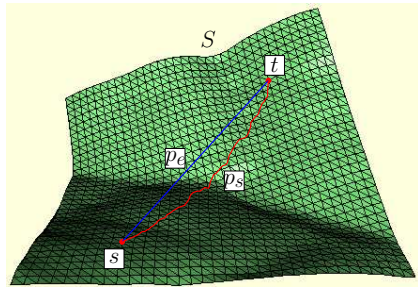


Figure 1: Terrain illustrating a surface path

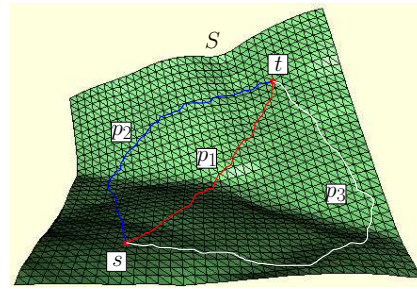


Figure 2: Terrain illustrating gentle paths

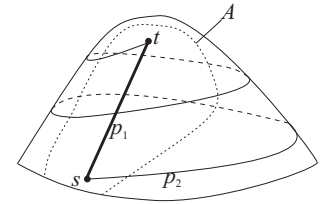


Figure 3: Terrain showing the challenge for problem FSGP

set to $\pi/2$, there is no slope requirement. Then, problem FSGP becomes the traditional problem.

Problem FSGP is more challenging than the traditional problem. Consider Figure 3 where s and t are two points on the surface and A is an area on the surface. Path p_1 is the traditional shortest path from s to t . Path p_2 is the shortest gentle path from s to t . Informally speaking, due to the slope requirement, a gentle path passes through area A multiple times. However, a traditional path passes through A once only. Obviously, the length of the gentle path p_2 is much larger than the length of the traditional path p_1 . Thus, problem FSGP is at least as hard as the traditional problem.

Obviously, one can adapt some existing algorithms in the literature or design new algorithms (e.g., breadth-first search algorithm) for problem FSGP. However, it is expected that the computation costs of these algorithms are *high* as illustrated in Figure 3. Let the algorithm which finds the shortest gentle path from s to t on a surface S be $\mathcal{A}(s, t|S)$. In our experiment on the real terrain dataset of Bearhead area in Washington State, if we set $\theta_m = 0.3$, we found that an algorithm for problem FSGP took about 3 days to find the shortest gentle path of length about 0.5km-1.0km. Even if we do not consider the slope requirement (i.e., $\theta_m = \pi/2$), then the best-known algorithm [1] for the (traditional) problem ran for about 3 days to find the shortest path of length about 7.0km. In many applications, time is a critical factor. One example is some emergency situations (e.g., natural disaster). Quick evacuations in a natural disaster are needed as in the Australian fires in Feb 2009 and the Californian fires in Oct 2007. Besides, in a large-scale natural disaster such as the earthquake in China in May 2008, distributing relief supplies and rescuing are time-critical. Some other applications are military planning and robot path planning [25].

Motivated by the above observation that the computation cost for problem FSGP is high, instead of developing a new algorithm for problem FSGP, in this paper, we propose a new framework called *surface simplification* over a terrain. In the literature [7, 6, 23, 27], the surface of a terrain is usually represented by the Triangular Irregular Network (TIN) model. The TIN model involves a number of non-overlapping triangles which are arranged in a three-dimensional space. Figure 4a shows an example of the terrain in Figure 3 represented by the TIN model. Note that for illustration purpose, Figure 4a involves a *limited* number of triangles. If the number of triangles used in the model is larger, then the surface of the terrain represented by this model becomes smoother. Under the *surface simplification* framework, given the original surface S , we *simplify* or *approximate* S (Figure 4a), and generate a simplified surface \tilde{S} (Figure 4b) such that the number of triangles on the simplified surface \tilde{S} is smaller than that on the original surface S . We run an algorithm \mathcal{A} for problem FSGP to find the shortest gentle path \tilde{p} on this simplified surface \tilde{S} , as shown in Figure 4b. Then, we *map* this path \tilde{p} on \tilde{S} to a path p on S as shown in Figure 4c. We call this step *Path Mapping*.

Surface simplification has its advantage to speed up the computation of finding shortest gentle paths. Intuitively, since it minimizes the number of triangles on the surface, the algorithm for problem FSGP can run in a shorter period of time. Although it can speed up the computation, there are the following two challenges.

Firstly, the mapped path p on S may not satisfy the slope requirement. For example, the mapped path p in Figure 4c does not meet the slope requirement. Consider the triangle f_1 . The path p in f_1 is quite steep. In order to address this issue, in this paper, we study how to construct a mapped path p on S which can satisfy the slope requirement given a gentle path \tilde{p} found on a given simplified surface \tilde{S} .

Secondly, even though the mapped path p on S satisfies the slope requirement, p may be extremely long compared with the shortest gentle path p_o (or the *optimal path*) found on the original surface S . Clearly, surface simplification loses some surface information and thus the length of the mapped (gentle) path on S is larger than that of the optimal path. This motivates us to introduce an additional requirement, the *distance requirement*, in addition to the slope requirement. Let $|p|$ be the length of a path p on S . The distance requirement specifies that, given a *distance error parameter* $\epsilon \geq 0$, the length of the (mapped) path p found should be bounded by ϵ . That is, $|p| \leq (1 + \epsilon)|p_o|$ where p_o is the optimal path on S . In this paper, we study how to simplify the surface such that, given *any* source point s and *any* destination point t on surface S , the mapped (gentle) path from s to t satisfies the distance requirement.

Surface simplification speeds up the computation with the distance guarantee. If we set $\epsilon = 0.1$ and $\theta_m = 0.3$, experiments shows that the total time to find a gentle path p between two randomly chosen vertices under the surface simplification model is 174 seconds. However, the time to find the shortest gentle path p_o on the original surface is 1482 seconds. Thus, the speedup in execution time is about 8.5 times if we sacrifice the length by a factor of at most 10% only. Surface simplification is very useful not only for problem FSGP but also the traditional problem. If we set $\theta_m = \pi/2$, problem FSGP becomes the traditional problem. In this case, if we set $\epsilon = 0.1$, we have the speedup in execution time by 138 times even if we sacrifice the length by a factor of at most 10% only.

Our contributions are summarized as follows. (1) To the best of our knowledge, we are the first to study how to find a shortest path which satisfies both the slope requirement and the distance requirement. (2) To the best of our knowledge, we are the first to propose a novel idea to simplify the surface of a terrain such that the length of the path found is bounded (regardless of whether there is a slope requirement). The surface simplification has its advantages to speed up the computation of finding shortest gentle paths because it minimizes the total number of triangles on the surface. (3) We present a systematic performance study using both real and

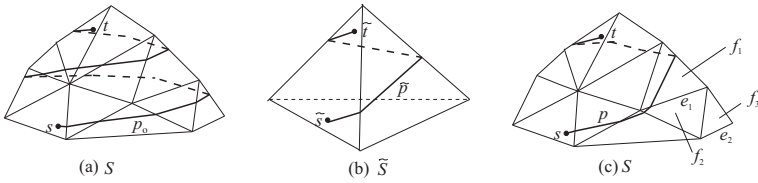


Figure 4: Surface simplification

synthetic datasets to verify the effectiveness and the efficiency of our surface simplification framework.

The rest of the paper is organized as follows. Section 2 reviews the previous work. Section 3 formulates the problem FSGP. Section 4 and Section 5 describe the proposed algorithms. Section 7 evaluates the proposed techniques through extensive experiments with real datasets. Section 8 concludes the paper.

2. RELATED WORK

We classify the related works into two categories, finding shortest paths and surface simplification.

2.1 Finding Shortest Paths

Finding shortest paths has been extensively studied in two-dimensional spatial databases [8, 20]. Dijkstra’s algorithm [8] is the well-known algorithm for finding shortest paths. Some other algorithms are the Bellman-Ford algorithm, the A* search algorithm and the Floyd-Warshall algorithm.

Finding shortest paths is a fundamental operator in a lot of spatial queries like k nearest neighbor searches. Some studies like [2, 17, 21] propose nearest neighbor searches in two-dimensional spaces.

Finding shortest paths on the surface of a terrain has also been studied by [16, 1]. [16] extends the idea of Dijkstra’s algorithm and solves the problem of finding shortest paths on the surface in $O(n^2 \log n)$ time where n is the total number of triangles in the TIN model. [1] discovers a spatial property called *one-angle-one-split* and makes use of this property to design an efficient algorithm which improves the running time to $O(n^2)$ time.

In the context of terrain, finding shortest paths is also involved in other spatial queries [5, 6, 23, 27]. One example is surface kNN (skNN) queries which were first studied by [5, 6]. [5, 6] propose a filter and refinement strategy for this kind of queries. However, as pointed out by [23], the results returned by this strategy [5, 6] are not guaranteed to be correct. Motivated by this observation, [23] proposes an approach based on the Voronoi diagram for the skNN queries and guarantees that the results returned are correct. Another example of spatial queries related to finding shortest paths is the continuous skNN queries which were proposed recently by [27]. [27] studies to find the k nearest neighbors when the objects on the surface are moving.

However, most of the existing works, including the studies which propose to find shortest paths on the surface of a terrain, do not consider the *slope* requirement which appears in a terrain.

Some studies [13, 15, 11, 26] propose *approximate* algorithms for finding shortest paths. However, since they are originally designed for the traditional problem of finding shortest paths and thus do not meet the slope requirement, it is not easy to adapt their proposed algorithms to our problem.

2.2 Surface Simplification

Surface simplification, a fundamental technique in the literature of computer graphics, is widely used in multi-resolution modeling. It was first proposed to accelerate the rendering speed of complex 3D models. In the multi-resolution model, an object can be represented in different levels of details (LOD). Surface simplification

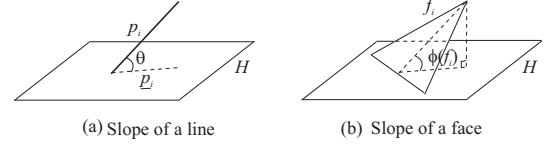


Figure 5: Slope

is a special case of multi-resolution modeling. There are several methods for surface simplification in the literature of graphics. One is *vertex decimation* [22, 24]. Under vertex decimation, a vertex v is selected for removal as shown in Figure 6a. After this vertex is removed, all of its incident edges are removed and a new polygon is formed as shown in Figure 6b. The resulting polygon is *triangulated* as shown in Figure 6c such that the polygon is partitioned into a number of triangles. *Triangulation* is a process which partitions a given polygon into a number of triangles. There are a lot of methods for triangulation. In this paper, we adopt the method proposed by [9]. Other surface simplification methods include *edge contraction* [10], *vertex clustering* [19] and *simplification envelopes* [3].

Note that the focus of this paper is different from the above studies about surface simplification. Our focus is to simplify the surface such that the mapped path on the original surface satisfies both the slope requirement and the distance requirement. However, the focus of the above studies is to simplify the surface of an object such that the simplified surface “looks” similar to the original surface.

3. PROBLEM DEFINITION

3.1 Notation

Each point q in the three-dimensional space has an x-coordinate, a y-coordinate and a z-coordinate, denoted by $q.x$, $q.y$ and $q.z$, respectively. $q.z$ corresponds to the *elevation* of point q .

A *terrain* is the graph of a continuous function that assigns every point on a horizontal plane to an elevation [4]. In the literature [7, 6, 23, 27], the surface of a terrain is usually represented by the *Triangular Irregular Network (TIN) model* consisting of a number of disjoint *triangles*. Each triangle is represented by three corners called *vertices* and three lines connecting these three corners called *edges*. In the literature, a triangle is also referred to as a *face*. In the following, we use term “triangle” and term “face” interchangeably. Note that each vertex is also a point in a three-dimensional space. If an edge is located at the boundary of a terrain, it is *owned* by only one triangle. For example, in Figure 4c, edge e_2 at the boundary is owned by face f_3 only. Otherwise, it is *shared* by two triangles. For example, in Figure 4c, face f_1 and face f_2 share an edge e_1 . Figure 4a shows an example of the terrain in Figure 3 represented by the TIN model. Let S be the surface of the terrain represented by the TIN model. Let H be a (virtual) horizontal plane located at the sea level of the terrain. We define the z-coordinate of each point on plane H to be 0.

Consider Figure 7a showing a terrain with a horizontal plane H . In the following, we follow the convention that the points on the horizontal plane H are denoted by *cross* points and symbolized by *underlined* variables while the points on the surface of a terrain are denoted by *dot* points and symbolized by *non-underlined* variables. For example, the cross point \underline{s} is on the plane H and the dot point s is on surface S . Following the terrain model [4], we assume that any vertical line must intersect with the surface at only one point.

Given a source point s and a destination point t on the surface, a *path* p from s to t is defined to be a sequence of *line segments* on the surface which starts from s and ends at t . Each line segment can

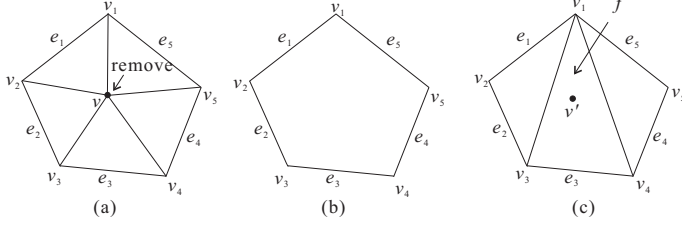


Figure 6: Vertex decimation

also be regarded as a path. For example, in Figure 7a, p' is the path from point s to point t on the surface. Path p' is a sequence of 4 line segments, namely (1) the segment from s to a , (2) the segment from a to b , (3) the segment from b to c and (4) the segment from c to t . The *length* of a line segment p_i from a point a to another point b , denoted by $|p_i|$, is defined to be the Euclidean distance between a and b . The *length* of path p , denoted by $|p|$, is defined to be the sum of the lengths of all line segments of p .

Given a point q on the surface, the *shadow* of q , denoted by \underline{q} , is defined to be a point on the plane H such that $\underline{q}.x = q.x$, $\underline{q}.y = q.y$ and $\underline{q}.z = 0$. Given a path p on the surface, the *shadow* of p , denoted by \underline{p} , is defined to be the shortest path on plane H covering the shadow of any point along p . For example, in Figure 7a, point \underline{s} is the shadow of point s and the line segment from \underline{s} to \underline{a} is the shadow of the line segment from s to a . Given a face f , the *shadow* of f is defined to be the minimal region on plane H covering the shadow of any point on face f .

Consider a line segment p_i on the surface. We define the *slope* of p_i to be the acute angle (in radians) between p_i and the shadow of p_i . Figure 5a shows that the slope of line segment p_i is equal to θ . The *slope* of a path p is defined to be the greatest possible slope of a line segment of a path. The *slope* of a face f_i , denoted by $\phi(f_i)$, is defined to be the angle (in radians) (or formally the dihedral angle) between the plane covering face f_i and the horizontal plane H . Figure 5b shows that the slope of face f_i is equal to $\phi(f_i)$.

3.2 Finding Gentle Path

As described in Section 1, we propose the *slope requirement* to capture the steepness of a path by introducing a slope parameter $\theta_m \in [0, \pi/2]$. For the sake of illustration, in the following, we assume that $\theta_m > 0$. That is, θ_m is not equal to 0. This assumption avoids complicated and uninteresting discussions. If this assumption does not hold, we can set θ_m to be a very small positive value close to 0. With this parameter, we study problem FSGP. Let the algorithm which finds the shortest gentle path from s to t on a surface S for this problem be $\mathcal{A}(s, t|S)$.

A natural question may be raised for problem FSGP: *Does there exist a gentle path from a given source point s to a given destination point t on the surface?* For the sake of clarity, we first assume that there exists a gentle path from any source point to any destination point. In Section 6, we will describe how we relax this assumption.

In this paper, we propose a framework called *surface simplification* which simplifies the original surface S (Figure 4a) to surface \tilde{S} (Figure 4b) such that the number of faces on \tilde{S} is smaller than that on S . Figure 8a shows an overview of the surface simplification. In Section 5, we will study this process.

Figure 8b shows an overview to find the shortest gentle path from a source point s to a destination point t on surface S by using the simplified surface \tilde{S} . We first construct a point \tilde{s} and another point \tilde{t} on the simplified surface \tilde{S} by setting $\tilde{s}(\tilde{t})$ to be a point on \tilde{S} with the same shadow as $s(t)$. For example, in Figure 7a, s and t are the

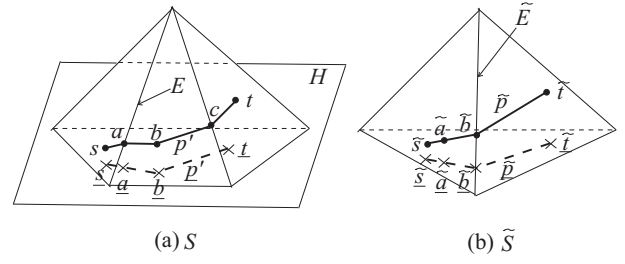


Figure 7: A terrain

source point and the destination point, respectively. We construct \tilde{s} and \tilde{t} on \tilde{S} with the same shadow as s and t , respectively, as shown in Figure 7b. Then, we run algorithm $\mathcal{A}(\tilde{s}, \tilde{t}|\tilde{S})$ to find the shortest gentle path \tilde{p} from \tilde{s} to \tilde{t} on this simplified surface \tilde{S} (Step I). Then, we *map* this path \tilde{p} on \tilde{S} to a path p on S (Step II) such that this mapped path satisfies the slope requirement. In Section 4, we will study how to perform this path mapping.

Since the mapped path may introduce errors, we consider the *distance requirement* in this paper by introducing a distance error parameter $\epsilon \geq 0$.

PROBLEM 1 (SURFACE SIMPLIFICATION). *Given the original surface S of a terrain, generate a new surface \tilde{S} such that (1) the total number of faces on \tilde{S} is minimized and (2) for any source point s and any destination point t on S , the mapped path on the original surface S satisfies both the slope requirement and the distance requirement.*

Note that Step I used in the step of finding shortest gentle paths involves algorithm \mathcal{A} . Our contribution is the proposal of surface simplification over the terrain dataset with the guarantee of both the slope requirement and the distance requirement. Any algorithm \mathcal{A} which finds the shortest gentle path (or the optimal path) (e.g., a breadth-first search algorithm and a best-first search algorithm) can also be used in our framework. In our experiment, we adopt the breadth-first search strategy for algorithm \mathcal{A} . A brief description of this algorithm can be found in Section 7.

For the sake of illustration, we follow the notation convention that all notations used in the simplified surface are symbolized with \sim which appears at the top of the notations (e.g., point \tilde{s} and path \tilde{p} on the simplified surface \tilde{S} in Figure 7b) while all notations used in the original surface are not (e.g., point s and path p' on the original surface S in Figure 7a).

In the following, we first describe Step II in Section 4. Then, in Section 5, we describe how we perform surface simplification.

4. PATH MAPPING

4.1 Algorithm

Let \tilde{p} be the shortest gentle path on the simplified surface \tilde{S} found in Step I (See Figure 8b). In this section, we study Step II (i.e., how to map \tilde{p} on \tilde{S} to a path p on S).

The major idea of path mapping is based on the *common shadow* of the path \tilde{p} found on \tilde{S} and the mapped path p to be found on S . Intuitively, after we find \tilde{p} on \tilde{S} , we generate the shadow of \tilde{p} . Then, we generate a temporary path p' on the original surface S which has the same shadow as \tilde{p} . We call p' the *pseudo path* of \tilde{p} . Since path p' may violate the slope requirement, we perform an additional step called *path adjusting* to adjust path p' to path p (on surface S) such that p satisfies the slope requirement.

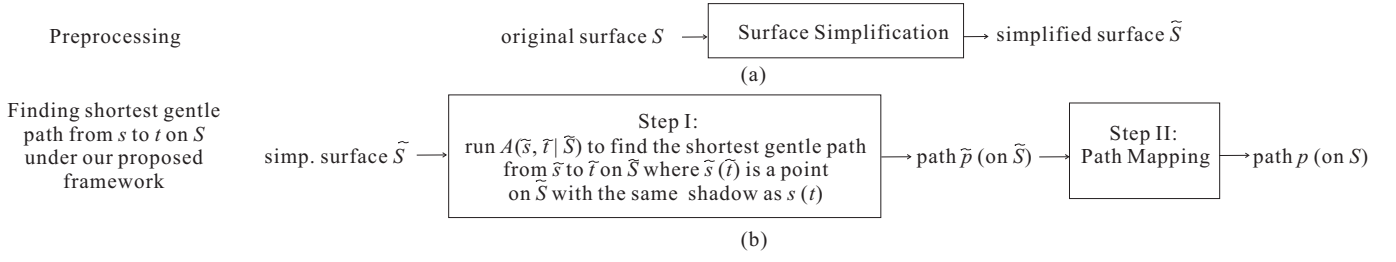


Figure 8: An overview of our proposed framework

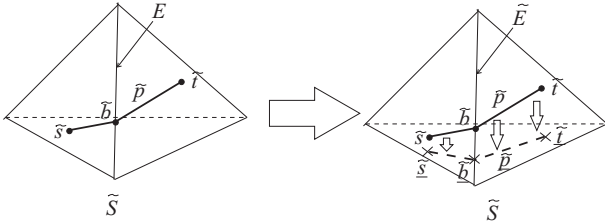


Figure 9: Shadowing

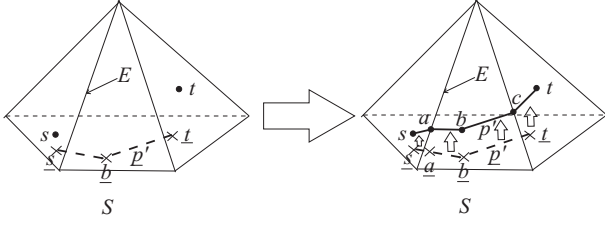


Figure 10: Segmentation

Specifically, we propose a three-step algorithm called algorithm *Path Mapping* to map a path \tilde{p} found on \tilde{S} to a path p on S . We denote the relationship between \tilde{p} and p by a path mapping function PM such that

$$p = PM(\tilde{p}) \quad (1)$$

Step 1 (Shadowing): We find the shadow of \tilde{p} . For example, Figure 9 shows the shortest gentle path \tilde{p} from \tilde{s} to \tilde{t} on \tilde{S} which involves (1) the line segment from \tilde{s} to \tilde{b} and (2) the line segment from \tilde{b} to \tilde{t} . The shadow of \tilde{p} is \underline{p} which involves (1) the line segment from \underline{s} to \underline{b} and (2) the line segment from \underline{b} to \underline{t} . Note that \underline{b} is a point connecting two line segments of path \underline{p} on the surface. The reason why \underline{b} appears in path \underline{p} is that path \tilde{p} passes through edge \tilde{E} via point \tilde{b} . Thus, this leads to the existence of a *shadow point* \underline{b} on the *shadow path* \underline{p} .

Step 2 (Segmentation): We generate a pseudo path p' on the original surface S which has the same shadow as \tilde{p} . For instance, shadow \underline{p} shown in Figure 10 (for S) is the same as shadow \underline{p} shown in Figure 9 (for \tilde{S}). Note that shadow \underline{p} involves (1) the line segment from \underline{s} to \underline{b} and (2) the line segment from \underline{b} to \underline{t} . Path p' shown in Figure 10 is the pseudo path. Although the shadow involves a *single* line segment from \underline{s} to \underline{b} , we have to divide this line into two *segments*, namely (1) the segment from \underline{s} to \underline{a} and (2) the segment from \underline{a} to \underline{b} , due to the existence of edge E . Thus, in Step 2, in addition to creating path p' with the same shadow as \tilde{p} , we have to divide p' into a number of line segments such that each line segment does not pass across any edge on the original surface. After Step 2, we obtain p' which involves k line segments, namely p'_1, p'_2, \dots, p'_k . For example, in Figure 10, we have 4 line segments of p' and thus k is equal to 4.

Step 3 (Path Adjusting): After we obtain a pseudo path p' from Step 2, we perform a step called *path adjusting* and generate path p such that p satisfies the slope requirement. Assume that the final path p contains k continuous (sub-)paths p_1, p_2, \dots, p_k . For each line segment p'_i of p' (obtained in Step 2) with the source point a and the destination point b where $i \in [1, k]$, if it satisfies the slope requirement, we set p_i to be p'_i . Otherwise, we execute algorithm $\mathcal{A}(a, b|S)$ to find the shortest gentle path from a to b on the original surface S and assign it to p_i . The combination of the resulting (sub-)paths p_1, p_2, \dots, p_k is called the *adjusted path* of path p' , which is the path in the output.

Intuitively, each line segment (or path) violating the slope requirement will be adjusted accordingly such that the final adjusted (sub-)path satisfies the slope requirement. Thus, we give the following theorem.

THEOREM 1 (SLOPE REQUIREMENT). *Let \tilde{p} be the shortest gentle path on the simplified surface \tilde{S} . The path p on the original surface S mapped from \tilde{p} by algorithm Path Mapping satisfies the slope requirement.*

4.2 Theoretical Analysis

In the following, we focus on analyzing the distance bound of a path segment instead of the distance bound of the whole path. This is because once the distance bound of a path segment is found, the distance bound of the whole path also holds.

4.2.1 Distance Analysis for a Particular Path Segment

Consider a path \tilde{p} from \tilde{s} to \tilde{t} on surface \tilde{S} returned by $\mathcal{A}(\tilde{s}, \tilde{t}|\tilde{S})$ (in Step 1).

In the previous section, we learned that we can perform *Path Mapping* to map a path \tilde{p} to a pseudo-path p' (by Step 1 and Step 2) and then adjust p' to the final path p (by Step 3). In our algorithm, according to Step 3, we generate k sub-paths in the final path p , namely p_1, p_2, \dots, p_k . In the example shown in Figure 7a, k is equal to 4. Let the correspondence pseudo-path line segments on S be p'_1, p'_2, \dots, p'_k . For example, in Figure 7a, the line from a to b is an example of a line segment p'_i of p' . In order to perform analysis for the distance requirement, we also find the corresponding line segment \tilde{p}_i of \tilde{p} which has the same shadow as line segment p'_i of p' for $i \in [1, k]$. For instance, in Figure 7b, the line from \tilde{a} to \tilde{b} is an example of a line segment \tilde{p}_i of \tilde{p} which has the same shadow as p'_i .

In the following, we want to analyze the ratio of $|p_i|$ to $|\tilde{p}_i|$ (i.e., $|p_i|/|\tilde{p}_i|$) for each $i \in [1, k]$ when we map a *given* gentle line segment \tilde{p}_i to p_i . We call this ratio the *mapping ratio of a particular gentle path \tilde{p}_i* .

Note that p_i is generated from \tilde{p}_i via a pseudo-path line segment p'_i . Thus, in the following, we calculate the mapping ratio with $|p'_i|$.

4.2.2 Distance Analysis for an Arbitrary Path Segment

Note that X is a region on plane H . Let $GP(\tilde{f}, X)$ be a set of all possible gentle paths on face \tilde{f} which shadows are inside X . Note that, from Equation (1), we have $p_i = PM(\tilde{p}_i)$. The greatest possible mapping ratio of an arbitrary gentle path (segment), denoted by $MR(f, \tilde{f})$, is equal to

$$\max_{\substack{\tilde{p}_i \in GP(\tilde{f}, X) \\ \text{where } p_i = PM(\tilde{p}_i)}} MRS(\langle p_i, f \rangle, \langle \tilde{p}_i, \tilde{f} \rangle) \quad (7)$$

The above term $MR(f, \tilde{f})$ can be regarded as the greatest possible mapping ratio of an arbitrary gentle path \tilde{p}_i when we map a gentle path (segment) \tilde{p}_i on a *simplified* face \tilde{f} to a path p_i on an *original* face f . This above term can be in fact expressed in terms of the coordinates of all the vertices of f and \tilde{f} . Details can be found in [14].

In general, the above term can be *generalized* as follows. Consider two different surfaces S and S' each of which represents the same terrain (in the above derivation, S is the original surface and S' is the simplified surface). Given a face f on S and another face f' on S' with an overlapping shadow X , $MR(f, f')$ can be regarded as the greatest possible mapping ratio of an arbitrary gentle path p' when we map a gentle path (segment) p' on face f' to a path p on face f . With this reasoning, in addition to $MR(f, \tilde{f})$, we can also define $MR(\tilde{f}, f)$ similarly (i.e., the greatest possible ratio of an arbitrary gentle path p_i when we map a gentle path (segment) p_i on an *original* face f to a path \tilde{p}_i on a *simplified* face \tilde{f}).

With the above definition of $MR(f, f')$, we have the following lemma.

LEMMA 3 (DIST. BOUND FOR ARBITRARY PATH SEGMENT). *Let S and S' be two different surfaces each of which represents the same terrain. Let f be a face on S and f' be another face f' on S' such that f and f' have an overlapping shadow X . Suppose that p' is the shortest gentle path (segment) on f' . If algorithm Path Mapping maps p' on face f' to a sub-path p on face f , then we have $\frac{|p|}{|p'|} \leq MR(f, f')$.*

5. SURFACE SIMPLIFIER

In this section, we present an algorithm called *Surface Simplifier* to simplify surface S to surface \tilde{S} such that any mapped path p found by algorithm *Path Mapping* satisfies the distance requirement (in addition to the slope requirement).

Let p be the mapped path on surface S (obtained in Step II in Figure 8b). Let p_o be the optimal path (i.e., the shortest gentle path) on surface S . The distance requirement specifies that, given a distance error parameter $\epsilon \geq 0$, we have $|p| \leq (1 + \epsilon)|p_o|$ or $\frac{|p|}{|p_o|} \leq 1 + \epsilon$. We define the *distance error ratio* of path p , denoted by $ER(p)$, to be $\frac{|p|}{|p_o|}$. Note that $ER(p) \geq 1$ since $|p| \geq |p_o|$. If p is the optimal path, then $ER(p) = 1$.

Obviously, if surface \tilde{S} is exactly the same as the original surface S , then any mapped path p found must be optimal and thus $ER(p) = 1$. This is because the mapped path p on S is exactly the same as the path \tilde{p} found on \tilde{S} which is considered as the shortest gentle path on \tilde{S} .

However, if surface \tilde{S} is different from the original surface S , then it is likely that a mapped path p found is not optimal and thus $ER(p) > 1$. Besides, intuitively, if the “difference” between S and \tilde{S} is greater, then it is more likely that $ER(p)$ is greater where p

is a mapped path. In order to satisfy the distance requirement, we want that the “difference” between S and \tilde{S} should not be too large.

The “difference” between S and \tilde{S} is denoted by $\Delta(S, \tilde{S})$. In Section 5.1, we will describe an exact formula for $\Delta(S, \tilde{S})$ such that the following property holds.

PROPERTY 1 (SURFACE BOUND). *Let S be the original surface and \tilde{S} be the simplified surface. If $\Delta(S, \tilde{S}) \leq 1 + \epsilon$, then for any mapped path p , $ER(p) \leq 1 + \epsilon$.*

5.1 Formula for $\Delta(S, \tilde{S})$

Let $\mathcal{CS}(S, \tilde{S})$ be a set of all possible pairs (f, \tilde{f}) where f is a face on S and \tilde{f} is a face on \tilde{S} such that f and \tilde{f} have an overlapping shadow. We define λ and λ' as follows.

$$\lambda = \max_{(f, \tilde{f}) \in \mathcal{CS}(S, \tilde{S})} MR(f, \tilde{f}) \quad (8)$$

$$\lambda' = \max_{(f, \tilde{f}) \in \mathcal{CS}(S, \tilde{S})} MR(\tilde{f}, f) \quad (9)$$

With the notations λ and λ' , we define $\Delta(S, \tilde{S})$ as follows.

DEFINITION 1. $\Delta(S, \tilde{S}) = \lambda \times \lambda'$

With this definition, we have the following lemma for the correctness of Property 1.

LEMMA 4. *Let S be the original surface and \tilde{S} be the simplified surface. If $\Delta(S, \tilde{S}) \leq 1 + \epsilon$, then for any mapped path p , $ER(p) \leq 1 + \epsilon$.*

Proof: Let \tilde{p} be a path found in Step I and p be a path found in Step II. Since \tilde{p} is returned by $\mathcal{A}(\tilde{s}, \tilde{t}, \tilde{S})$, \tilde{p} is the shortest gentle path from \tilde{s} to \tilde{t} on \tilde{S} . We have the following inequality. For any gentle path \tilde{g} from \tilde{s} to \tilde{t} on \tilde{S} ,

$$|\tilde{g}| \geq |\tilde{p}| \quad (10)$$

We adopt the notations used in Section 4.2.1 here. Specifically, in algorithm *Path Mapping*, according to Step 3, we generate k sub-paths in the final path p , namely p_1, p_2, \dots, p_k . Let the correspondence pseudo-path line segments on S be p'_1, p'_2, \dots, p'_k . All of these line segments form a pseudo-path p' . Besides, let \tilde{p}_i be the correspondence line segment of \tilde{p} which has the same shadow as line segment p'_i of p' for $i \in [1, k]$.

Consider a sub-path p_i and a line segment \tilde{p}_i where $i \in [1, k]$. Suppose that p_i is on face f and \tilde{p}_i is on face \tilde{f} . By Lemma 3, we have

$$\frac{|p_i|}{|\tilde{p}_i|} \leq MR(f, \tilde{f})$$

Since each line segment \tilde{p}_i of \tilde{p} has the above inequality with the sub-path p_i of p where $i \in [1, k]$, by some simple derivations, it is easy to verify the following.

$$\frac{|p|}{|p_o|} \leq \max_{(f, \tilde{f}) \in \mathcal{CS}(S, \tilde{S})} MR(f, \tilde{f}) \quad (11)$$

With Equation (8), we have

$$\frac{|p|}{|p_o|} \leq \lambda \quad (12)$$

Let p_o be the optimal path on S (i.e., the shortest gentle path on S). Consider that we apply algorithm *Path Mapping* to map p_o on the original surface S to generate a path \tilde{p}_o on the simplified surface \tilde{S} . Similar to the derivation for Inequality (11), by Lemma 3, we have

$$\frac{|\tilde{p}_o|}{|p_o|} \leq \max_{(f, \tilde{f}) \in \mathcal{CS}(S, \tilde{S})} MR(\tilde{f}, f)$$

Thus, with Equation (9), we have

$$\frac{|\tilde{p}_o|}{|p_o|} \leq \lambda' \quad (13)$$

Note that \tilde{p}_o on \tilde{S} generated by algorithm *Path Mapping* is a gentle path on \tilde{S} (by Theorem 1). Thus, from Inequality (10), we have the following inequality: $|\tilde{p}_o| \geq |\tilde{p}|$. With this inequality, we re-write Inequality (12) as: $\frac{|p|}{|p_o|} \leq \lambda$. With Inequality (13), we re-write this inequality as: $\frac{|p|}{|p_o|} \leq \lambda \times \lambda'$. Since $ER(p) = \frac{|p|}{|p_o|}$, we have

$$ER(p) \leq \lambda \times \lambda' \quad (14)$$

Note that, from the condition of Property 1, we have $\Delta(S, \tilde{S}) \leq 1 + \epsilon$. By Definition 1, we derive that $\lambda \times \lambda' \leq 1 + \epsilon$. Thus, Inequality (14) can be re-written as: $ER(p) \leq 1 + \epsilon$. \square

5.2 Algorithm Simplifier

In this section, we describe an algorithm called *Surface Simplifier* which adopts one of the methods for surface simplification discussed in Section 2.2. Any existing methods about surface simplification reducing the number of faces in the literature of graphics can also be adopted in our algorithm provided that the simplified surface \tilde{S} is generated such that (1) $\Delta(S, \tilde{S}) \leq 1 + \epsilon$ and (2) the shadow of the boundary of S is the same as the shadow of the boundary of \tilde{S} .

In this paper, we use the techniques described in [22, 24] for surface simplification. Specifically, in [22, 24], surface simplification involves a number of iterations. Initially, we create a new surface \tilde{S} which is equal to the original surface S . It is being updated over iterations and finally represents the simplified surface we want. For each iteration, we select a vertex to be removed from \tilde{S} (See Figure 6a) such that \tilde{S} will be updated after vertex decimation (See Figure 6c) and the distance requirement is satisfied for this updated \tilde{S} . We continue the above step until we cannot find any vertex to be removed such that the distance requirement is satisfied.

It is easy to verify the following theorem.

THEOREM 2. *Algorithm Surface Simplifier generates surface \tilde{S} such that $\Delta(S, \tilde{S}) \leq 1 + \epsilon$.*

In our implementation, we design an efficient method for our algorithm *Surface Simplifier* which takes $O(|V| \log |V|)$ time where $|V|$ is the total number of vertices on the original surface. For the sake of space, we give this method in [14].

6. DISCUSSION

In Section 3, we assume that there exists a gentle path from any source point to any destination point. In this section, we relax this assumption. We will introduce a term *reachable* for a point q to describe whether q is reachable or not. In this section, we will show that if both the source point s and the destination point t are reachable, then there exists a gentle path from s to t .

6.1 Reachability

Given two distinct points q and q' on a surface, q is said to be *reachable from q'* if and only if there exists a gentle path from q' to q . q is said to be *unreachable from q'* if q is not reachable from q' .

Given a point q on a surface, q is said to be *reachable* if and only if there exists another point q' on the surface such that q is reachable from q' . q is said to be *unreachable* if q is not reachable.

Given a vertex q , a face f is said to be *adjacent to q* if one of the corners of f is q .

LEMMA 5 (UNREACHABILITY). *A point q on a surface is unreachable if and only if q is a vertex and q is unreachable from any point on all of the faces adjacent to q .*

Proof: Firstly, we want to prove that, if q is a vertex and q is unreachable from any point on all of the faces adjacent to q , then q is unreachable. We prove by contradiction. Suppose that q is reachable. That is, there exists another point q' on the surface such that q is reachable from q' . Consider two cases. *Case 1:* q' is on one of the faces adjacent to q . This leads to a contradiction.

Case 2: q' is not on all of the faces adjacent to q . Since q is reachable from q' , there exists a gentle path from q' to q . Let F be the set of all faces adjacent to q . We know that path p must pass through one of the faces in F , says f . Let q'' be a point along path p on face f . Thus, q is reachable from q'' (which is on one of the faces adjacent to q). This leads to a contradiction.

Secondly, we want to prove that, if q is unreachable, then q is a vertex and q is unreachable from any point on all of the faces adjacent to q . We prove by contradiction. Consider two cases. *Case 1:* q is a vertex and there exists another point q' on one of the faces adjacent to q such that q is reachable from q' . Thus, q is reachable, which leads to a contradiction. *Case 2:* q is not a vertex. That is, q is a non-vertex point on a face f . There exists another point q' on f such that $q'.z = q.z$. Thus, q is reachable from q' . So, q is reachable, which leads to a contradiction. \square

The above lemma suggests that all unreachable points come from vertices on the surface. Thus, we just need to check whether each *vertex* (instead of all possible points on the surface) is unreachable or not. How to check whether a vertex is unreachable or not efficiently will be discussed in Section 6.2.

LEMMA 6 (SOURCE/DESTINATION REACHABILITY). *If a source point s and a destination point t are reachable, then t is reachable from s .*

Proof: From the proof of Lemma 5, it is easy to verify that, if a vertex q on face f is reachable, for any point q' on f , q is reachable from q' . For the sake of space, we omit the proof here. There exists a sequence of adjacent faces, namely f_1, f_2, \dots, f_l , such that s is on face f_1 , t is on face f_l and, for each $i \in [1, l-1]$, face f_i is adjacent to face f_{i+1} . Let e_i be the edge shared by face f_i and face f_{i+1} for $i \in [1, l-1]$. By the first claim in this proof, we know that there exists a non-vertex point t_1 on edge e_1 such that t_1 is reachable from s . Similarly, we know that there exists a non-vertex point t_2 on edge e_2 such that t_2 is reachable from t_1 . In general, there exists a non-vertex point t_i on edge e_i such that t_i is reachable from t_{i-1} for $i \in [2, l-1]$. At the final step, similarly, we know that t is reachable from t_{l-1} (by the first claim in the proof). Since t_1 is reachable from s , t_i is reachable from t_{i-1} where $i \in [2, l-1]$, and t is reachable from t_{l-1} , we conclude that t is reachable from s . \square

6.2 Algorithm

In this general setting, algorithm *Surface Simplifier* will be changed as follows. The basic idea of the change is based on the following two properties. *Property 1:* For each vertex on S which is unreachable, the algorithm still keeps the original vertex v on \tilde{S} . *Property 2:* Whenever the algorithm generates \tilde{S} , it makes sure that it does not introduce any new vertex which is unreachable.

Intuitively, if all vertices are unreachable, then we cannot simplify the surface. If there are only rare unreachable vertices, then we can simplify the surface aggressively.

Since algorithm *Surface Simplifier* maintains Property 1 and Property 2, it is easy to verify that there exists a gentle path on S if and only if there exists a corresponding gentle path on \tilde{S} . Thus, algorithm *Path Mapping* is kept intact in this problem setting.

Efficiency: In algorithm *Surface Simplifier*, we have to check whether a vertex is unreachable or not. The following lemma helps us to perform this checking step efficiently.

Let V be a set of vertices on a surface S . Consider that v is in V . Let $N(v, V)$ be a set of vertices in $V/\{v\}$ each of which shares an edge with v . For example, in Figure 6a, $N(v, V)$ is equal to $\{v_1, v_2, v_3, v_4, v_5\}$.

LEMMA 7 (REACHABILITY). *Consider a vertex v and a surface S . Let V be a set of vertices on S . Let $Y = N(v, V)$. v is reachable if one of the following conditions holds. Condition (1): there exists two vertices in Y , namely v_i and v_j , such that $v_{i,z} \geq v.z$ and $v_{j,z} \leq v.z$, or Condition (2): there exists an edge e with an endpoint equal to v such that the slope of e is at most θ_m .*

Proof: Consider the first condition. Since there exists two vertices v_i and v_j such that $v_{i,z} \geq v.z$ and $v_{j,z} \leq v.z$, we deduce that there exists a face f adjacent to v such that f has two corners/vertices, namely v_a and v_b , where $v_{a,z} \geq v.z$ and $v_{b,z} \leq v.z$. Thus, there exists a point q on face f where $q.z = v.z$ such that v is reachable from q .

Consider the second condition. Since the edge has its slope at most θ_m , there exists a point q on this edge such that v is reachable from q . \square

With the above lemma, if one of the two conditions is satisfied, we are sure that a vertex is *reachable*. Otherwise, it can be either reachable or unreachable. In our implementation, we can remove any vertex which falls in the former case (i.e., it is found to be reachable by this lemma). But, we keep each vertex which falls in the latter case. This implementation does not violate Property 1 and Property 2. We found that only 0.22% of vertices fall in the latter case in our experiment where θ_m is set to 0.3.

The efficiency of checking whether a vertex is reachable can be improved by using this lemma. For each vertex v , we check Condition (1) and Condition (2) in the lemma. If one of the conditions holds, then v is reachable. It is easy to see that performing the checking step with Condition (1) can be done in $O(|Y|)$ time and performing the checking step with Condition (2) can be done in $O(|Y|)$ time.

7. EMPIRICAL STUDY

In this paper, we used two real terrain datasets adopted in previous studies [7, 6, 23, 27]. (1) *Eagle Peak (EP) area in Wyoming State*: This dataset covers an area around 10.7km \times 14km which contains about 3,200,000 faces, and (2) *Bearhead (BH) area in Washington State*: This dataset covers an area around 9.7km \times 13.7km which contains about 2,600,000 faces. Both datasets can be downloaded from <http://data.geocomm.com>. We also created synthetic datasets as follows. Each synthetic dataset contains a terrain bounded by a 10km \times 10km square horizontal plane. On this horizontal plane, we randomly pick four points as the centers of the mountains. Then, the elevation of all points with respect to the center of each mountain is modeled by a 2-dimensional Gaussian distribution denoted by mean m and standard derivation σ . The mean m of each distribution (corresponding to the elevation of the center of the mountain) is randomly generated from 0 to 10000. The standard derivation σ is an input parameter. Since

each point belongs to four mountains with different elevations, we take the greatest elevation among the four distributions as the final elevation of the point. The datasets generated contain 600,000-1,000,000 faces. The default values of θ_m , ϵ , the total number of faces on the surface and σ are 0.3, 0.1, 600,000 and 0.1, respectively.

All programs were written in C/C++ and executed on CentOS linux platform on a 2xQuad Core 3GHz server with 32GB RAM. Our proposed algorithm returns a path denoted by p which satisfies the distance requirement and the slope requirement. We implemented a simple breadth-first algorithm \mathcal{A} for finding shortest gentle paths (FSGP) on a given terrain, whose basic idea is as follows: Note that there may be multiple shortest gentle paths (SGP) in this problem. We proved that one of them must pass through a *simple face sequence*. A *face sequence* is a sequence of faces while a *simple face sequence* is a face sequence which contains no duplicate faces. So, starting from the face that contains source s , we generate all possible simple face sequences in a breadth-first manner until destination t is reached. In order to speed up, those unnecessary face sequences that are impossible to be used for finding SGP are pruned. In each simple face sequence, the length of the SGP can be represented by a *function* of the coordinates of s , t and all the vertices of the triangles in this face sequence. We can also specify a set of *slope constraints* along all faces in the face sequence in a similar way by using the coordinates of these vertices and θ_m . Then, we took the advantage of any existing optimization tool such as MATLAB, ALGLIB, OOL and IOptLib to find the minimum value of this function subject to the constraints. For each unpruned simple face sequence, we obtain the corresponding SGP. Finally, among all SGP's obtained from all unpruned face sequences, we select the shortest one as the final SGP. The detailed description of the algorithm can be found in [14]. Besides, we compared our proposed algorithm with the baseline algorithm that finds the shortest gentle path denoted by p_o on the original surface S .

Note that one may think out an algorithm based on the shortest traditional path on S as follows. Specifically, this algorithm finds the shortest traditional path on S (without considering the slope constraint) and perform a path adjusting step (described in Section 4) such that the final adjusted path satisfies the slope requirement. However, it is possible that the final adjusted path does not satisfy the distance requirement. We denote the final adjusted path by p_c .

All experiments were conducted 100 times by randomly generating 100 queries where each query involves a random source point and a random destination point. We took the average for the results.

For real datasets, we study the performance of algorithms with the following two parameters, namely (1) θ_m and (2) ϵ .

We evaluate our algorithms in terms of five measurements. (1) *Preprocessing time*: Preprocessing time corresponds to the execution time to run algorithm *Surface Simplifier*, (2) *Number of faces*: We measured the number of faces on the original surface and the number of faces (remained) on the simplified surface. (3) *Memory consumption*: Memory consumption corresponds to the storage size to store all faces on a surface (which can be derived from the number of faces measured above). (4) *Path finding time*: Path finding time of our proposed algorithm under surface simplification which finds path p corresponds to the execution time to run Step I and Step II. Path finding time of each of the baseline algorithms corresponds to the execution time of the correspondence whole algorithm. (5) *Path length*: The length of p , p_o and p_c are all measured.

7.1 Real Datasets

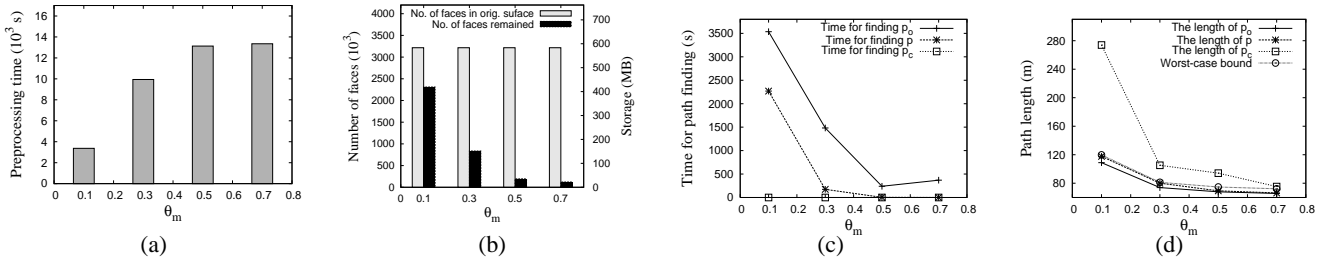


Figure 12: Effect of θ_m (Eagle Peak where $\epsilon = 0.1$)

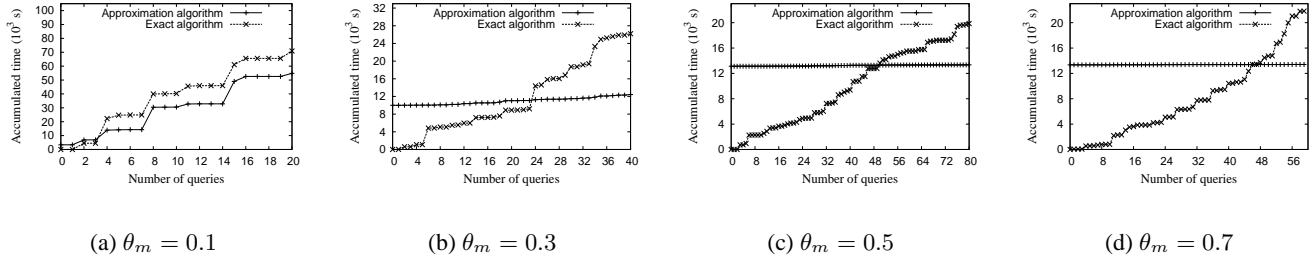


Figure 13: Comparison between the exact algorithm and the approximation algorithm (Eagle Peak where $\epsilon = 0.1$)

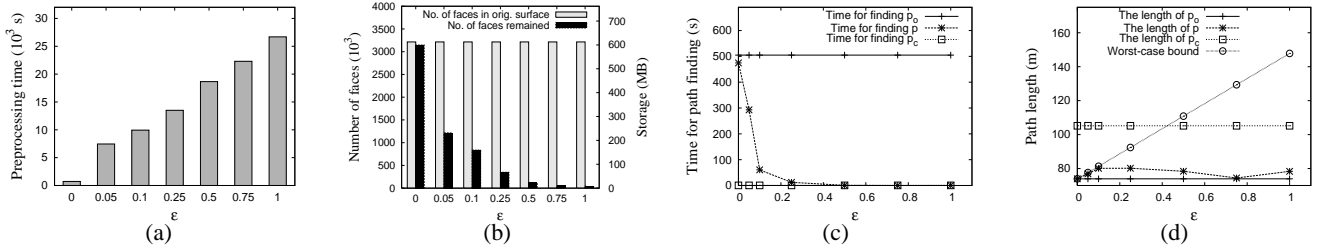


Figure 14: Effect of ϵ (Eagle Peak where $\theta_m = 0.3$)

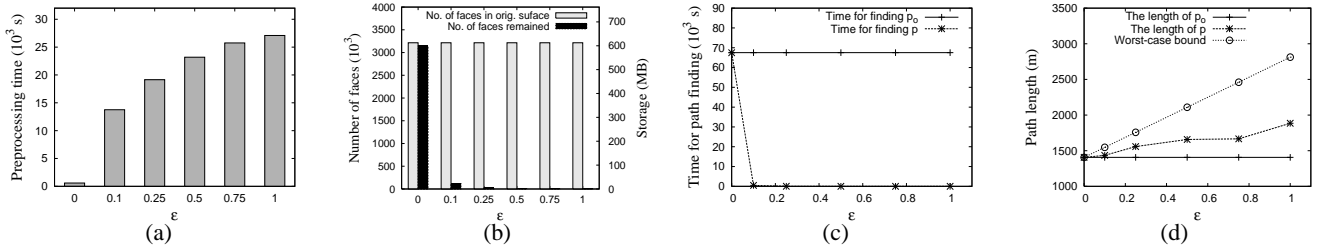


Figure 15: Effect of ϵ (Eagle Peak where $\theta_m = \pi/2$)

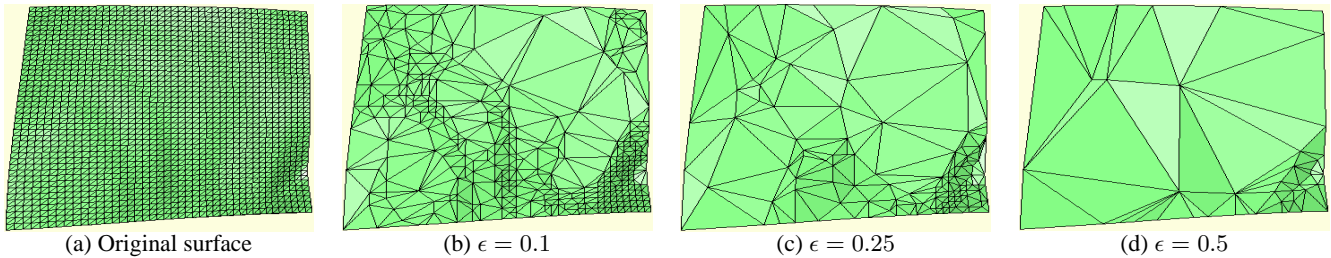


Figure 16: Results of surface simplification (Bearhead where $\theta_m = 0.3$)

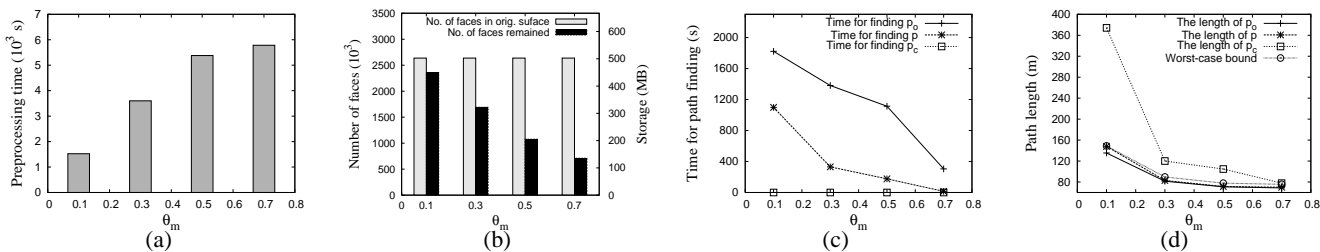


Figure 17: Effect of θ_m (Bearhead where $\epsilon = 0.1$)

We conducted experiments on two real datasets, Eagle Peak and Bearhead. The results for Eagle Peak can be found in Figure 12, Figure 13, Figure 14 and Figure 15.

Effect of θ_m : In Figure 12a, when θ_m increases, the preprocessing time increases. This is because, when θ_m is larger, intuitively, the slope requirement is weaker and thus we can remove more vertices in algorithm *Surface Simplifier*. If there are more vertices removed, then the time to simplify the surface (in the preprocessing step) is larger. In Figure 12b, the number of vertices remained (and thus the memory consumption of the surface) decreases when θ_m increases. In Figure 12c, when θ_m increases, the time for finding p decreases. Since θ_m is larger, the total number of faces remained is smaller. Thus, the time for finding p is shorter. Besides, the time for finding p_o is much longer than the time for finding p . In particular, if $\epsilon = 0.1$ and $\theta_m = 0.3$, on average, our proposed algorithm which finds p takes 174s but the optimal algorithm which find p_o needs 1482s. Thus, the speedup is 8.5 times. If θ_m is set to 0.1, the speedup is 1.6 times. The time for finding p_c is shorter than the time for finding p_o and the time for finding p . Note that path p_c does not have any guarantee on the distance requirement (which will be described later). Besides, when θ_m is set to 0.3, we analyzed that the proportion of path segments (obtained after segmentation) which undergo the process of path adjusting is 26.5%. This suggests that it is not quite frequent to execute path adjusting in Step II. In Figure 12d, we denote an additional curve called ‘‘Worst-case bound’’ to denote the greatest theoretical error bound according to the length of p_o . This value is equal to $(1 + \epsilon)|p_o|$. In this figure, the length of p_o decreases when θ_m increases. Besides, the length of p is smaller than the worst-case bound but the length of p_c is larger than the worst-case bound. In particular, if $\epsilon = 0.1$ and $\theta_m = 0.3$, the ratio of the length of p to the length of p_o is 1.099 but the ratio of the length of p_c to the length of p_o is 2.45.

In our proposed framework, we need to simplify a surface as a preprocessing step. In order to study how this preprocessing step benefits our algorithm for finding a gentle path over the simplified surface, we did the following experiments. In this experiment, we compare two algorithms, namely an *approximation algorithm* and an *exact algorithm*. The *approximation algorithm* corresponds to the algorithm for finding p while the *exact algorithm* corresponds to the algorithm finding p_o . We define the *accumulated time* of an algorithm as follows. Consider a query workload Q containing x queries where x is a non-negative integer. The accumulated time of an approximation algorithm is defined to be the sum of the preprocessing time and the total time of finding all (approximate) gentle paths among all queries in Q by the approximation algorithm. The accumulated time of an exact algorithm is defined to be the total time of finding all (exact) gentle paths among all queries in Q by the exact algorithm. Figure 13a shows that the accumulated times of the two algorithms increase with the number of queries in Q when $\theta_m = 0.1$ and $\epsilon = 0.1$. When Q contains fewer than 3 queries, the accumulated time of the exact algorithm is smaller than that of the approximation algorithm. However, when there are more than 3 queries in Q , the accumulated time of the exact algorithm is greater than that of the approximation algorithm. In other words, the approximation algorithm needs only 3 queries to compensate the preprocessing cost. We conducted other experiments when $\theta_m = 0.3, 0.5$ and 0.7 , which can be found in Figure 13b, Figure 13c and Figure 13d, respectively. We find that 23 queries, 50 queries and 46 queries are needed to compensate the preprocessing cost when $\theta_m = 0.3, 0.5$ and 0.7 , respectively.

Effect of ϵ : Figure 14a shows that the preprocessing time increases with ϵ . This is because the number of faces removed increases with ϵ and thus algorithm *Surface Simplifier* needs more time to simplify

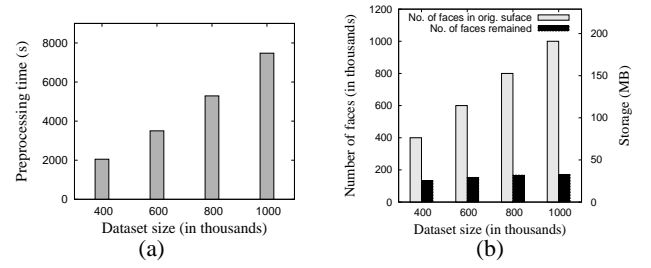


Figure 18: Effect of dataset size (Synthetic datasets)

the surface. In Figure 14b, the number of faces removed increases with ϵ . In Figure 14c, the time to find path p decreases when ϵ increases. This is because the total number of faces remained is smaller if ϵ is larger. For example, when $\epsilon = 0.1$, the time to find path p is 60s but the time to find path p_o is about 505s. The speedup of finding path p (compared with finding path p_o) is about 8.42 times. When $\epsilon = 1.0$, the speedup is nearly to 31,562.5 times. We conclude that when ϵ increases, the speedup increases. In Figure 14d, similarly, the length of p is smaller than the worst-case bound. Note that the worst-case bound increases linearly with ϵ but the length of p does not increase linearly with ϵ . Instead, the difference between the length of p and the worst-case bound becomes larger when ϵ increases. For instance, the distance error is still kept at most 10% when ϵ increases from 0.1 to 1.0.

Effect for Traditional Problem: We studied our proposed framework, surface simplification, for the traditional problem which does not consider any slope constraint (Figure 15). In this experiment, we adopt the implementation of Chen and Han’s algorithm [12] (which is originally designed for the traditional problem) for algorithm \mathcal{A} in our proposed framework. The original implementation of [12] needs to find all the shortest paths from a given source point s to all the other vertices. We modified the implementation such that once the path to a given destination point t is found, the algorithm terminates immediately. The results are also similar to Figure 14 but the preprocessing time and the time for finding p (p_o) are shorter. In particular, our proposed algorithm to find path p takes 489s with 10% distance error guarantee. However, the optimal algorithm takes more than 67,527 seconds to find p_o . The speedup is 138 times. In addition, interestingly, when $\epsilon = 0$, the time for finding p is smaller than the time for finding p_o because *Surface Simplifier* merges some adjacent faces with the same slope which speeds up the computation.

Effect on Surface Simplification: Figure 16 shows the results of our surface simplification on the real dataset, Bearhead, where $\theta_m = 0.3$. For the sake of illustration, we only focus on a portion of this dataset with dimension 420m x 420m, which is shown in Figure 16a. Figures 16b, c and d show the simplified surface when we set ϵ to 0.1, 0.25 and 0.5, respectively. The number of triangles on the simplified surfaces decreases when ϵ increases.

We also did the experiments for Bearhead to study how θ_m and ϵ affect the performance of the algorithms and study how the proposed algorithm works well for the traditional problem. The results are also similar to those obtained from Eagle Peak. For the sake of space, we just show the results for parameter θ_m in Figure 17.

7.2 Synthetic Datasets

We also conducted experiments with synthetic datasets. In addition to the two parameters studied in real datasets (i.e., θ_m and ϵ), *dataset size* is also used to study the performance of the proposed algorithm over synthetic datasets where dataset size corresponds to the total number of faces on the surface.

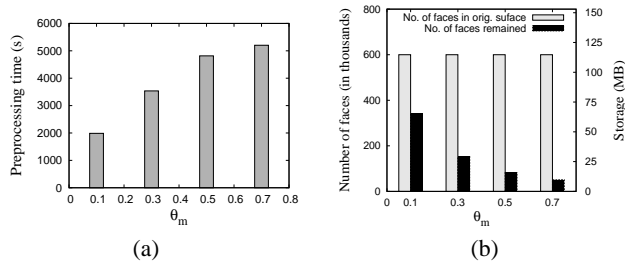


Figure 19: Effect of θ_m (Synthetic datasets)

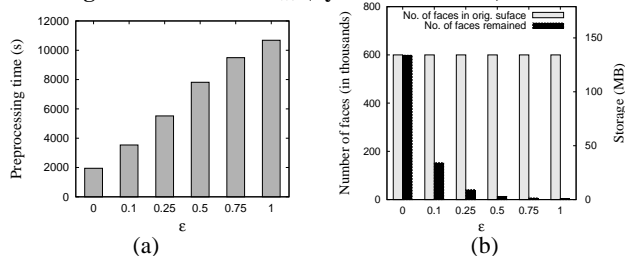


Figure 20: Effect of ϵ (Synthetic datasets)

Effect of Dataset Size: In Figure 18a, the preprocessing time increases with the dataset size. Figure 18b shows that, when the dataset size increases, the number of remaining faces keeps nearly unchanged. This is because when the dataset size is large, the number of vertices on a given fixed surface will be larger. However, since the surface generated by the synthetic data generator is nearly the same when the dataset size is larger, after surface simplification, it is likely that the given surface contains nearly the same number of faces (when the dataset size increases).

Effect of θ_m and ϵ : Figures 19 and 20 shows the results when we vary θ_m and ϵ , respectively. The trends for the effect of θ_m and ϵ on the synthetic datasets are similar to the trends on the real datasets.

Conclusion: Our proposed algorithm runs efficiently if we sacrifice the length of the path a little bit. The larger θ_m (ϵ) is, the faster our algorithm runs. In particular, if $\epsilon = 0.1$ and $\theta_m = 0.3$, our proposed algorithm runs 8.5 times faster than the optimal algorithm which finds the optimal path on S if the path found by our algorithm is at most 10% longer than the optimal path. If we set $\theta_m = \pi/2$, our proposed problem becomes the traditional problem. Our proposed algorithm runs 138 times faster than the optimal algorithm with 10% distance guarantee.

8. CONCLUSION

We study a fundamental operator in spatial databases, finding shortest paths on the surface of a terrain. In this problem, we consider the slope requirement such that the path is not too steep. Since solving this problem is more challenging than solving the traditional problem, we propose a new framework called surface simplification. Under this framework, we can compute shortest gentle paths efficiently. We conducted the experiments to show that our proposed framework is very efficient and effective not only for problem FSGP but also for the traditional problem.

There are a lot of promising research directions. Firstly, it is interesting to consider other popular spatial queries such as k nearest neighbors queries and range queries on the surface with the slope constraint. Secondly, another interesting direction is to study the real time spatial queries such as continuous k nearest neighbors in our problem setting, which have been studied extensively recently.

Acknowledgements: We are grateful to the anonymous reviewers for their constructive comments on this paper. The research is

supported by HKRGC GRF 621309 and Direct Allocation Grant DAG11EG05G.

9. REFERENCES

- [1] J. Chen and Y. Han. Shortest paths on a polyhedron. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 360–369, New York, NY, USA, 1990. ACM.
- [2] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, 2005.
- [3] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *SIGGRAPH*, 1996.
- [4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 3rd ed. edition, 2008.
- [5] K. Deng and X. Zhou. Expansion-based algorithms for finding single pair shortest path on surface. In *W2GIS*, pages 151–166, 2004.
- [6] K. Deng, X. Zhou, H. T. Shen, Q. Liu, K. Xu, and X. Lin. A multi-resolution surface distance model for k-nn query processing. *The VLDB Journal*, 17(5):1101–1119, 2008.
- [7] K. Deng, X. Zhou, H. T. Shen, K. Xu, and X. Lin. Surface k-nn query processing. In *ICDE*, 2006.
- [8] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.
- [9] S. Fortune. A sweepline algorithm for voronoi diagrams. In *Algorithmica* 2(2), 1987.
- [10] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, 1997.
- [11] T. Kanai and H. Suzuki. Approximate shortest path on polyhedral surface based on selective refinement of the discrete graph and its applications. In *GMP*, 2000.
- [12] B. Kaneva and J.O'Rourke. An implementation of chen & han's shortest paths algorithm. In *the 12th Canadian Conference on Computational Geometry*, 2000.
- [13] M. A. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th ACM Symp. on Computational Geometry*, 1997.
- [14] L. Liu and R. C.-W. Wong. Finding shortest gentle path (technical report). In <http://www.cse.ust.hk/~raywong/paper/findingShortestGentlePath-technical.pdf>, 2011.
- [15] C. S. Mata and J. S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In *Proc. 13th ACM Symp. on Computational Geometry*, 1997.
- [16] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.
- [17] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.
- [18] G. W. Records. *Guinness: World Records 2010*. Guinness, 2009.
- [19] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *In Geometric Modeling in Computer Graphics*, pages 455–465, 1993.
- [20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.
- [21] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
- [22] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH*, 1992.
- [23] C. Shahabi, L.-A. Tang, and S. Xing. Indexing land surface for efficient knn query. *PVLDB*, 1(1):1020–1031, 2008.
- [24] M. Soucy and D. Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63:1–14, 1996.
- [25] P. Tompkins, T. Stentz, and W. Whittaker. Mission planning for the sun-synchronous navigation field experiment. In *IEEE International Conference on Robotics and Automation*, 2002.
- [26] K. R. Varadarajan and P. K. Agarwal. Approximating shortest paths on a nonconvex polyhedron. *SIAM J. Comput.*, 30(4):1321–1340, 2000.
- [27] S. Xing, C. Shahabi, and B. Pan. Continuous monitoring of nearest neighbors on land surface. *PVLDB*, 2(1):1114–1125, 2009.