Terrain-Toolkit: A Multi-Functional Tool for Terrain Data

Qi Wang* Manohar Kaul† Cheng Long, Raymond Chi-Wing Wong‡
*Zhejiang University †Aarhus University ‡The Hong Kong University of Science and Technology
*steveqiwang@zju.edu.cn †mkaul@cs.au.dk ‡{clong, raywong}@cse.ust.hk

ABSTRACT

Terrain data is becoming increasingly popular both in industry and in academia. Many tools have been developed for visualizing terrain data. However, we find that (1) they usually accept *very few data formats* of terrain data only; (2) they do not support *terrain simplification* well which, as will be shown, is used heavily for query processing in spatial databases; and (3) they do not provide the *surface distance* operator which is fundamental for many applications based on terrain data. Motivated by this, we developed a tool called *Terrain-Toolkit* for terrain data which accepts a comprehensive set of data formats, supports terrain simplification and provides the surface distance operator.

1. INTRODUCTION

Terrain which usually refers to the land surface (simply surface[1]) is increasingly popular in industry. For example, Google (specifically, Google Maps) and Microsoft (specifically, Bing Maps for Enterprise) maintain a huge database of terrain data and use them for better user experience of exploring the maps. Terrain data also attracts extensive attention in academia. For example, researchers have devoted considerable efforts to kNN queries, range queries and shortest path queries (called *shortest surface path* queries in the context of terrain data) based on terrain data [3, 6, 7, 9, 10].

The most intuitive way for using and analyzing terrain data is to visualize it. In fact, many tools have been developed for visualizing terrain data. Some examples include MeshMan, CityGML, Terraserver, Google Earth, and Spaceye3D (A comprehensive survey could be found at http://vterrain.org/). Different tools have their different major purposes. For example, Google Earth puts its focus usually on viewing the terrain data freely in many ways (multi-layers, rotation, zoom-in and zoom-out) while some others put their focuses on the visual effect, rendering process and so on.

However, we observe that these existing visualization tools have several insufficiencies, especially from the perspective of computing with terrain data. We show them one by one as follows.

[1]In the following, we use "terrain" and "surface" interchangeably.

Proceedings of the VLDB Endowment, Vol. 7, No. 13

First, one existing visualization tool alone usually does not cover a rich enough set of models (and file formats) for the terrain data. These file formats can be mainly classified as *Raster* (e.g. DEM, GTiff and SDTS) and *Non-Raster* (e.g. XYZ, OFF, PLY) formats. Raster formats require points with elevation values to be positioned on a uniform-grid, while non-raster formats are free to position their points anywhere. We classify collections of 2D/3D polygons or triangles (i.e. mesh models) as non-raster files too. To the best of our knowledge, there does not exist a single tool that supports all the previously mentioned data formats and allows for conversions between the formats.

Second, to the best of our knowledge, no existing visualization tool supports *surface simplification* [5] which refers the process of simplifying a given surface to a *simpler* one. Surface simplification is used as a core component in many computations based on terrain data [5,6], and the reason is that computations based on terrain data are usually rather expensive (for example, the fastest algorithm for computing the *shortest surface path* between two points has its time complexity of $O(n^2)$ where n is the number of vertices involved in the model representing the surface) and the surface simplification process helps to reduce the complexity of the surface considerably such that the computations based on the simpler surface become acceptably efficient.

Third, as far as we know, no existing visualization tool supports the *surface distance* operator which computes the length of the shortest surface path between two given points. Note that surface distance is usually not equal to the Euclidean distance, and in fact, Euclidean distance corresponds to a lower bound of the surface distance. Based on the surface distance, many queries have been defined on terrain data, e.g., kNN queries, range queries and shortest surface path queries [3, 6, 7, 9, 10].

Contributions. Motivated by the aforementioned insufficiencies of the existing visualization tools for terrain data, in this paper, we develop a new tool called *Terrain-Toolkit* which accepts a rich set of models and file formats for terrain data, provides the surface simplification functionality and supports the surface distance operator.

In the remainder of this paper, we first introduce our design of Terrain-Toolkit in Section 2. Afterwards, we describe the demonstration setup in Section 3 and conclude with some future directions in Section 4.

2. DESIGN

In this part, we give some background knowledge on terrain in Section 2.1, present the architecture of Terrain-Toolkit in Section 2.2, and then introduce three major components of Terrain-Toolkit, namely *model/format conversion*, *surface simplification* and *shortest surface path computation* in Sections 2.3, 2.4 and 2.5, respectively.

2.1 Background Knowledge

A *point* in three-dimensional space is represented by an (x, y, z) co-ordinate where z represents the elevation of the point.

A *Digital Elevation Model* (DEM) is a 3D geometric representation of the terrain. A DEM can either be represented as a *raster* where 3D points are positioned on a uniform-grid structure or as a *Triangular Irregular Network* (TIN). A TIN is a vector-based representation of the terrain as a collection of non-overlapping triangles (also referred to as *faces*). Each triangle/face is comprised of three corner points called *vertices* that are connected to each other by line segments called *edges*. Two faces are *adjacent* to each other if they have share a common edge. Figure 1 illustrates a simple TIN model with a few faces, where face F consists of vertices a,b and c. For larger and more realistic TIN models, refer to Figure 3.

TIN representations are usually *triangulated* from raster grids. *Delaunay* triangulation [2] is one of the most popular methods of triangulation employed. In comparison to a raster grid, a TIN model consumes more storage space but provides a superior ability to describe the underlying surface at different levels of resolution.

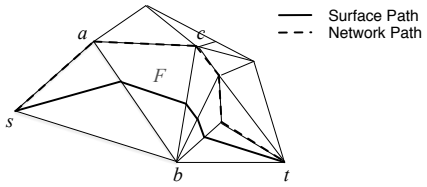

Figure 1: Surface Path (bold lines) and Network Path along the edges (dashed line).

The vertices and edges in a TIN already form a graph G, where every vertex is associated with a 3D coordinate and the weight on an edge is the 3D Euclidean distance between the two endpoint vertices.

For an arbitrary pair of vertices s and t on a TIN, a *surface path* is a path that traces along the surface of the TIN, while a *network path* is a sequence of edges in the graph representation G of the TIN. Figure 1 shows the two paths. Note that a surface path can *cut-across* any face in the TIN, while a network path must be confined to only the edges present in the TIN. The shortest surface path is the surface path with the least length and the shortest network path is the network path with least length.

2.2 Architecture of Terrain-Toolkit

The architecture of Terrain-Toolkit is shown in Figure 2. Terrain-Toolkit accepts terrain data, stored either in a *mesh file* or in a *point file*, as input. A mesh file stores terrain data by using a terrain model (such as TIN) and a point file stores a set of raw points which are usually sampled from the surface of terrain. With the *Model/Format Conversion* utility, terrain data stored in a file format can be converted to terrain data stored in another file format.

The input of terrain data stored in a mesh file can be directly processed via the *Preprocessing* utility to construct an *Internal Terrain Model* which maintains the terrain data in our own way for better manipulation (e.g., visualization, simplification, and shortest surface path computation) while the input of terrain data stored in a point file has to be *triangulated* first via the *Triangulation* utility and then could be processed via the *Preprocessing* utility to construct an *Internal Terrain Model*.

The *Internal Terrain Model* based on terrain data consists of two parts. The first part is a *triangulation model* (TIN) which maintains the vertices, edges and faces of the terrain and the second part is a *graph* which is another structure for capturing the vertices and edges of the terrain and is used for the purpose of computing the shortest surface path.

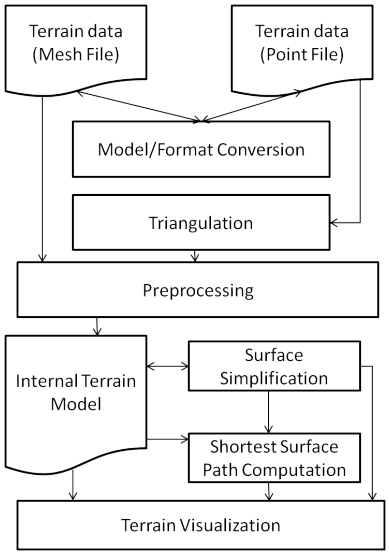

Figure 2: Architecture of Terrain-Toolkit

The *Internal Terrain Model* can be visualized via the *Terrain Visualization* utility and simplified via the *Surface Simplification* utility (we used OpenGL for our visualization purpose), whose results can either be visualized or used for further computation. Given two points on the surface of a terrain represented by either the *Internal Terrain Model* or the result of the *Surface Simplification* utility which also corresponds to an *Internal Terrain Model*, the shortest surface path can be computed via the *Shortest Surface Path Computation* utility.

In the following, we explain three key utilities of Terrain-Toolkit in detail.

2.3 Model/Format Conversion

The surface file formats can broadly be classified as *Raster* and *Non-Raster* formats. In the context of 3D surfaces, a raster consists of points laid out in a uniform grid-like fashion and each point has a height (z-value) associated with it. In contrast, in a non-raster format, the position of each point are not constrained to be in a uniform grid. Next, we briefly describe each file format.

- **USGSDEM**: is a raster format with elevation values per grid cell. We adopt the widely-used *United States Geological Survey (USGS)* DEM format in this paper.
- **GeoTiff (GTiff)**: uses a *TIFF* file format that stores raster graphics and embeds geo-spatial information as metadata.
- **Spatial Data Transfer Standard (SDTS)**: is a raster format and is primarily intended to be used for distribution and archiving of spatial data in the form of rasters that are generated in adherence to an open standard.
- **XYZ**: is a non-raster format and is a list of raw (x, y, z) coordinates generated from measurements of the actual land surface.
- **Geomview Object File Format (OFF)**: is a non-raster format and stores a description of 2D/3D surfaces comprising of polygons.
- **Polygon File Format (PLY)**: is a non-raster format and stores 2D/3D surfaces and is a widely used and well-known file format that is similar to the OFF file format.

Terrain-Toolkit allows convenient conversions among various file formats.

2.4 Surface Simplification

Surface simplification is the process of reducing the number of faces used in the surface while trying to keep the overall topology preserved as much as possible. Reducing the number of faces in a surface model can greatly speedup the visualization of the model and also computations based on the model, e.g., shortest surface path computations. Many surface simplification techniques have proposed in the research literature (for a detailed survey, please read [4]). We adopt the surface simplification method proposed in [5] since it guarantees that the *shortest surface distance* based on the simplified surface is within a bounded distance from the shortest surface distance based on the original surface.

The method [5] is based on an iterative approach of removing vertices from G. In each iteration, a chosen vertex $\bar{v}$ together with all edges connected to it are *removed*. After the removal, the set of all remaining vertices adjacent to this $\bar{v}$ forms a *polygonal hollow*. This polygonal hollow is re-triangulated to complete the original triangulation [2]. On completion of re-triangulation, two properties are tested against to ensure that the removal of vertex $\bar{v}$ does not break our distance guarantee on the shortest network distance. The properties are as following.

- **Intra-distance property:** The estimated network distance between any two adjacent vertices in the neighborhood of $\bar{v}$ should satisfy our distance guarantee.
- **Inter-distance property:** The estimated network distance between any adjacent vertex of $\bar{v}$ and a previously removed neighbor of $\bar{v}$ should satisfy our distance guarantee.

If removing a vertex and re-triangulating its polygonal neighborhood satisfies *both* the *intra-distance property* and the *inter-distance property*, then the vertex is removed and the simplified surface is ready for the next iteration of vertex removal. Otherwise, vertex $\bar{v}$ along with all edges originally connected to it are reinstated. For more detailed information about this surface simplification algorithm, please refer to [5].

Terrain-Toolkit allows a user to input a real-valued error parameter, whose value has the range $[1, 2]$. The toolkit provides the user with statistics about the number of vertices, faces and edges in the surface prior to and after simplification. In addition, the toolkit also provides the user a *side-by-side* visual comparison of the surface *before* the simplification process and *after*. Figures 3(a) and 3(b) show the before and after views, respectively.

2.5 Shortest Surface Path Computation

The computation of the shortest surface path between two points s and t on 3D surfaces is a fundamental operation in applications such as robotics, motion planning, geographic information systems (GIS), and is an important topic in fields like computational geometry and computer graphics.

The state-of-the-art *exact* shortest surface path computation algorithm was proposed by Chen and Han (CH) [1]. This algorithm has a $O(n^2)$ time complexity, where n is the number of vertices in the polyhedral surface. The CH algorithm uses a technique called *planar unfolding*. This time-consuming method rotates the faces of the surface, so that all faces end up on the same level plane. Similar to the computation of *visibility graphs*, the CH algorithm computes *shadows/projections* of the source vertex s on each edge in the triangulation and stores the results in a *sequence tree* that encodes information about the shortest paths from the source s to points on the edges. More recently, an improved CH algorithm was proposed by Xin et al. [8] which has the same theoretical time complexity but outperforms the original CH algorithm by orders of magnitude because it prunes a majority of unnecessary nodes from the sequence tree. We adopt this improved CH algorithm in our system.

Terrain-Toolkit allows a user to *pick* vertices s and t from the displayed surface model by clicking on the vertices. On completion of the shortest surface path computation, the path is displayed to the user on the visualized surface model along with additional statistics like the total path length and the number of vertices along the path. The user also has the ability to click on faces that the path cuts across to view additional information about the faces (e.g., face IDs and vertices on the faces). Figures 3(c) and (d) show the shortest surface paths computed on the simplified surface, while Figures 3(e) and (f) show the shortest surface path computed on the original surface.

3. DEMONSTRATION

In this section, we demonstrate our Terrain-Toolkit in Section 3.1 and describe a use-case that highlights the main features of Terrain-Toolkit in Section 3.2.

3.1 UI Demonstration

Terrain-Toolkit provides an easy-to-use and intuitive user interface by allowing the user to interact with the system at various stages. For example, the user is allowed to better visualize a 3D surface by rotating, scaling and translating the surface. In addition, the user can choose a smaller *rectangular region* on a large terrain for simplification and also shortest path computation. When doing Surface Simplification, the user is also allowed to fine-tune the error parameter (β) to control the desired resolution of the simplified surface. In addition, to further improve the user experience, we provide the user with clear instructions about how to use our system. Besides, meaningful statistics are displayed after completion of procedures such as Surface Simplification and Shortest Surface Path Computation. We shot a video for demonstrating the UI of Terrain-Toolkit which could be found at *http://www.cse.ust.hk/~raywong/paper/Terrain-Toolkit.mp4*.

In the following, we illustrate a use-case of Terrain-Toolkit.

3.2 Use Case: Emergency Response

The use-case outlined in this part highlights the flexibility and ease-of-use of our prototype in a real-life emergency response situation where computing the shortest surface path is critical. Consider the following scenario.

The *Search and Rescue* team in Montana receives a distress call from a hiker scaling the *Bearhead mountain (BH)* (located in the "Glacier National Park, Montana, U.S.A"), who accidentally fell into a deep crevice and needs immediate medical attention.

Sylvester, the team-leader of this rescue mission, instantly opens up the 3D surface model of BH in Terrain-Toolkit with the intent of finding the shortest and fastest path to reach the distressed hiker.

However, Sylvester realizes that the model has a very high resolution (i.e., with a large number of faces) and path computations is slow. Hence, he decides to load a previously simplified surface with much fewer faces to speed up this computation.

In the past, he had simplified the BH surface by loading the surface model, choosing an error parameter (β) in the pop-up dialog box and then clicking *Simplify Terrain* button (with the black-blue spinning wheel icon). Since he chose larger values of β, he got simplified models with fewer faces. He was able to compare the *before* and *after* simplification terrains visually and then he chose to either revert or save the simplified terrain based on his satisfaction. Figures 3(a) and 3(b) show this comparative view.

Equipped with this knowledge, he browses his file system for a simplified surface with the desired number of faces, loads it and marks the last known location of the hiker on the terrain by double-clicking and choosing one of the vertices on the terrain. He then

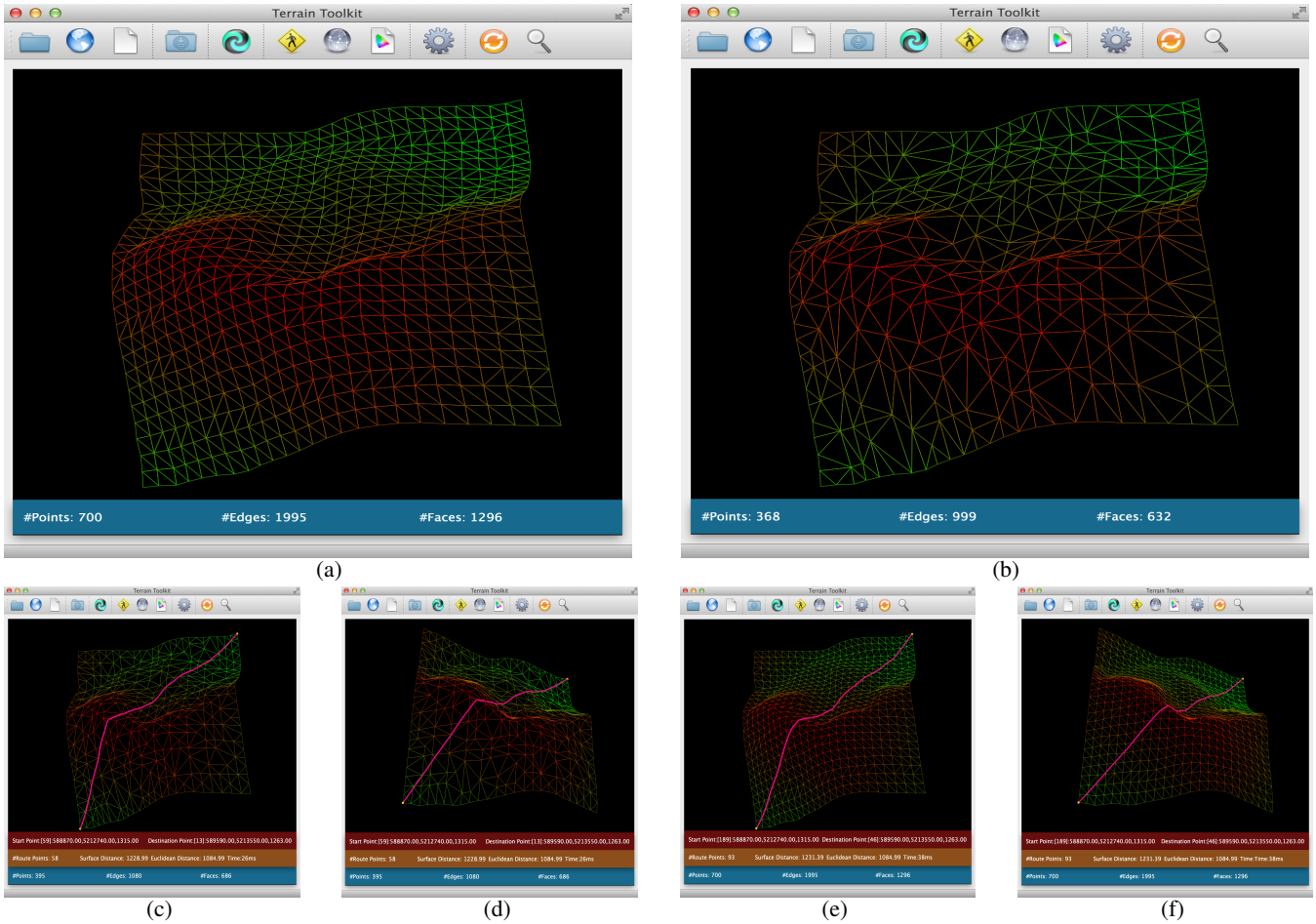

Figure 3: Effect of Surface Simplification and Shortest Surface Path Computation

visually locates the closest peak to this marked point, where the rescue helicopter can land safely, from where the rescue team and medics must navigate the terrain on foot to get to the hiker as soon as possible.

Sylvester marks the nearest peak by picking another vertex and clicks the *Find Shortest Surface Path* button (with the walking man icon). He receives an immediate response within 1-2 seconds, with the shortest path showing on the visualization as a bright yellow path. Furthermore, he is given with the total distance of the path and he can click on the faces that which the path cuts across to extract further information. Figures 3(c) and 3(d) show the shortest surface path on the simplified surface, viewed from two different angles.

Sylvester immediately dispatches his helicopter rescue team. The rescue team on the helicopter decides to utilize the journey time to load a higher resolution model of BH and recompute the shortest path from their landing location to the hiker's last known location. This computation takes longer (nearly 30-40 seconds), but provides them with a more accurate path and distance to locate, stabilize and rescue the hiker in distress. Figures 3(e) and 3(f) show the shortest surface path on the original surface, viewed from two different angles.

4. CONCLUSION

Motivated by the fact that there are very few toolkits available for terrain/surface manipulation, we developed Terrain-Toolkit. We demonstrate the techniques used in Terrain-Toolkit to efficiently simplify surfaces [5] and compute shortest surface paths [8] on them. There are several interesting features that can be added to our demonstration proposal. First, the user can choose multiple source points to reach a destination on the terrain. Second, various other simplification algorithms can be implemented and visualized using Terrain-Toolkit.

Acknowledgements: The research is supported by grant FS-GRF14EG34.

5. REFERENCES

[1] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 360–369, New York, NY, USA, 1990. ACM.

[2] B. N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, (6):793–800, 1934.

[3] 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. *VLDB J.*, 17(5), 2008.

[4] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms, 1995.

[5] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen. Finding shortest paths on terrains by killing two birds with one stone. *PVLDB*, 7(1):73–84, 2013.

[6] L. Liu and R. C.-W. Wong. Finding shortest path on land surface. In *SIGMOD Conference*, pages 433–444, 2011.

[7] C. Shahabi, L. A. Tang, and S. Xing. Indexing land surface for efficient knn query. *PVLDB*, 1(1):1020–1031, 2008.

[8] S.-Q. Xin and G.-J. Wang. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Trans. Graph.*, 28:104:1–104:8, September 2009.

[9] S. Xing, C. Shahabi, and B. Pan. Continuous monitoring of nearest neighbors on land surface. In *VLDB*, 2009.

[10] D. Yan, Z. Zhao, and W. Ng. Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces. In *CIKM*, 2012.