International Journal of Geo-Information

MDPI

Article

An Efficient Visualization Method for Polygonal Data with Dynamic Simplification

Mingguang Wu [1,2,3], Taisheng Chen [4,5,*], Kun Zhang [1,2], Zhimin Jing [1,2], Yangli Han [1,2], Menglin Chen [4,5], Hong Wang [1,2] and Guonian Lv [1,2,3]

1 Key Laboratory of Virtual Geographic Environment of Ministry of Education, Nanjing Normal University, Nanjing 210023, China; wmg@njnu.edu.cn (M.W.); kunznnu@gmail.com (K.Z.); hbxsjzm@163.com (Z.J.); HanYangLi_1@163.com (Y.H.); maswanghong@126.com (H.W.); gnlunjnu@126.com (G.L.)
2 College of Geographic Sciences, Nanjing Normal University, Nanjing 210023, China
3 Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing 210023, China
4 Department of Geographic Information Science, Chuzhou University, Chuzhou 239000, China; ml_chen@163.com
5 Anhui Engineering Laboratory of Geo-Information Smart Sensing and Services, Chuzhou 239000, China
* Correspondence: taisheng.chen@163.com; Tel.: +86-550-3510030

Received: 12 February 2018; Accepted: 21 March 2018; Published: 2 April 2018

Abstract: Polygonal data often require rendering with symbolization and simplification in geovisualization. A common issue in existing methods is that simplification, symbolization and rendering are addressed separately, causing computational and data redundancies that reduce efficiency, especially when handling large complex polygonal data. Here, we present an efficient polygonal data visualization method by organizing the simplification, tessellation and rendering operations into a single mesh generalization process. First, based on the sweep line method, we propose a topology embedded trapezoidal mesh data structure to organize the tessellated polygons. Second, we introduce horizontal and vertical generalization operations to simplify the trapezoidal meshes. Finally, we define a heuristic testing algorithm to efficiently preserve the topological consistency. The method is tested using three OpenStreetMap datasets and compared with the Douglas Peucker algorithm and the Binary Line Generalization tree-based method. The results show that the proposed method improves the rendering efficiency by a factor of six. Efficiency-sensitive mapping applications such as emergency mapping could benefit from this method, which would significantly improve their visualization performances.

Keywords: vector polygon; level-of-detail rendering; cartographic simplification; tessellation; trapezoid

1. Introduction

Cartographic representation is an effective way to model real geographic space. As data acquisition techniques have developed, a large volume of geographic data has been collected, and more is being gathered continuously. These raw data are often symbolized for perception, cognition and communications [1]. Although geographic data are gathered at specific scales, they can be simplified based on zoom level, where each zoom level emphasizes a different level of detail (LoD) [2]. In this paper, rendering symbolization at a certain LoD is called LoD visualization. While LoD visualizations of vector geographic data benefit map legibility, their visualization efficiency is of critical concern in efficiency-sensitive mapping applications such as Human–Computer Interaction mapping, in which maps need to be rapidly responded. There is a critical need for an efficient LoD visualization algorithm not only in distributed mapping architectures but also in desktop mapping environments.

This paper focus on the efficiency of LoD visualization for polygonal data. Polygons are a major geometric type used to model geographic objects such as land cover and administrative boundaries [3,4]. LoD visualization for polygonal data involves three operations: symbolization, simplification and rendering. All three operations are both algorithmically complex [5] and time-consuming to perform; therefore, LoD visualization often suffers from low efficiency, especially when the number of polygons becomes massive. As a compromise, static LoD visualization methods such as the map tiles technique, in which maps are rendered offline for different zoom levels to promote rapid responses, are widely used in current mapping systems. Such static LoD visualization techniques are suitable for applications in which maps are used as background images. In contrast, dynamic LoD visualization methods perform symbolization and simplification on-the-fly, which provides more flexibility for perception and cognition. For example, using dynamic LoD visualization, users can specify the characteristics of symbols used to draw the polygonal data, such as colors or fill textures; users can also customize the geometric details for different zoom levels. However, improving the performance of dynamic LoD visualization is challenging [6].

The available dynamic LoD visualization methods for polygonal data can be classified into three groups: multiresolution meshes, simplification-based methods, and hardware-accelerated methods. In the first group, multiresolution meshes are data structures commonly used for dynamic LoD visualization in the computer graphics field [7]. Various multiresolution meshes have been proposed for geometric modeling and visualization [8]. As a typical example, Hoppe [9] presented a scheme called a "progressive mesh" to organize, transmit and render geometric models in multiple resolutions. By introducing an edge collapse transformation, progressive mesh representations allow efficient, continuous resolutions for arbitrary triangle meshes. Multiresolution mesh techniques, including the progressive mesh methods, are focused on highly detailed geometric models and adapt best to certain data types, such as terrain data [10]. However, because these techniques do not consider cartographic simplifications of polygons, they are limited in their ability to implement polygon LoD rendering [11].

In the second group, the simplification-based methods, many works have focused on polygon LoDs with cartographic simplifications [12,13]. Detailed discussions on polygon and polyline simplifications can be found in Li [14], Galanda [3], Shi et al. [15], Podolskaya et al. [16] and Haunert [17]. An exhaustive investigation of those algorithms is beyond the scope of this paper. Instead, we focus on solutions that use those methods to implement LoD visualization for polygonal data. A straightforward solution is to divide the polygon boundaries into segments and then apply polyline simplification algorithms. Based on the Douglas Peucker (DP) algorithm, Van Oosterom [18] presented a data structure called the Binary Line Generalization tree (BLG-tree), which recursively divides a polyline and organizes the polyline segments into a binary tree. To maintain topological consistency, Van Oosterom [19] proposed the GAP-tree (Generalized Area Partitioning) method, which organizes area partitioning hierarchically. Further, based on the R-tree, a storage structure called the reactive-tree was presented, which was capable of implementing area partitioning using a dynamic LoD. Several improvements on the GAP-tree have been presented, such as the tGAP, which is used to avoid redundant data storage and slivers [17,20], and the smooth tGAP, which is used for smooth zooms and 3D data [21]. In addition, because topological incorrectness (e.g., self-intersections) and shape dissimilarities (e.g., area or perimeter dissimilarities) may occur when simplifying polygons, various topological and shape-preserving techniques have been proposed [22,23], such as using Delaunay triangulation to detect topological errors and using a hierarchical topological data structure to maintain topological consistency between different LoD results [24,25].

Focusing on the LoD of polygonal geometry, the above-mentioned methods are definitely helpful for applications such as progressive data transmission over the Internet [21,26,27]. However, because these methods do not consider rendering operators such as filling, their ability to facilitate LoD visualization of polygonal data is limited. In fact, polygons, no matter how complex they are [28], may be tessellated into drawable primitives, such as triangles or trapezoids (e.g., *triangle-strip* or quad-strip in OpenGL) for cartographic filling [29]. The drawable primitives are then sent to a rendering engine for display [30,31].

Within this process, the tessellation operation is computationally expensive [4], and the rendering performance is also dependent on the efficiency with which drawable primitives can be sent [32]—i.e., the performance is rate sensitive. By applying existing LoD polygonal geometry techniques such as BLG-tree, the polygons can be efficiently simplified. However, the simplified polygons still need to be re-tessellated and re-sent, causing computational and data redundancies that reduce efficiency.

In the third group, hardware-accelerated methods, the speed of both cartographic simplifications and rendering are improved by using hardware-accelerated techniques such as chain-based tessellation [1], graphics processing unit (GPU)-based rendering [31,33,34], and parallel algorithms for simplification [35]. In recent years, hardware-accelerated vector tile rendering has been widely used. However, simplifications, symbolization and rendering are still addressed separately. For example, MapBox uses OpenGL as its native map rendering engine to improve the map-rendering performance. MapBox even provides a WebGL interface to support advanced map rendering in web clients, including polygon symbolization [36]. However, in MapBox, LoDs are achieved by specifying the zoom level of the underlying data rather than dynamically simplifying them. In addition, both cartographic simplifications and rendering are considered in the hybrid vector- and raster-based approach proposed by Mustafa and Krishnan [6], which employs hardware buffers to efficiently create a Voronoi diagram to help simplify polygon boundaries via pixel-color checking. Because Voronoi diagrams and pixel-color checking can be hardware accelerated, this method is efficient for stroking the boundaries of polygons. However, it does not consider polygon filling, which is critical for cartographic representations.

From the above discussion, the existing methods of LoD visualization are focused on geometric models rather than on polygonal data. While LoD techniques for polygonal geometry have been widely addressed, efforts focused on LoD visualization for polygonal data are limited. A common issue is that simplification, symbolization and rendering are addressed separately, yielding computational and data redundancies that reduce efficiency. To address this issue, this paper presents an efficient method to support polygon LoD visualization. The remainder of this paper is organized as follows. Section 2 provides an overview of the proposed method. Section 3 presents a detailed explanation of the method. The approach is then tested in Section 4 and discussed in Section 5. Finally, Section 6 highlights the contributions of this paper and suggests related future works.

2. Method Overview

Polygon LoD visualization involves three major operations: simplification, tessellation and rendering. As discussed in the previous section, this process is computationally expensive and rate sensitive. Notably, for a polygon, different versions of simplified instances are similar. The closer they are in zoom level, the more similar their geometries will be. For example, when a user zooms in on a map by scrolling a mouse wheel, the simplified instances at two consecutive zoom levels may differ only slightly. When a user uses drags a rectangle over an area of interest and then zooms out, the critical boundary points can be preserved to maintain similarity. Furthermore, because polygon instances are mutually similar, drawable primitives, which are the intermediate visualization result, are correspondingly mutually similar. Ignoring similarity, computational redundancy occurs when tessellating similar polygon instances, and data redundancy occurs when rendering similar polygon instances.

Recognizing the similarity, the basic idea of this paper is to use similarity to reduce the computational redundancy and mitigate rate sensitivity. Unlike existing methods, we organize simplification, tessellation and rendering operations into a single mesh generalization process to avoid having to completely re-execute tessellation operations and resend all the drawable primitives. First, based on the sweep line method proposed by Žalik et al. [37], we proposed a topology embedded trapezoidal mesh data structure to organize the tessellated polygons. Second, we use DP algorithms to weight the trapezoidal meshes and then introduce horizontal and vertical generalizations to simplify them. Finally, we define a heuristic testing algorithm to preserve the topological consistency efficiently. Beyond the application of supporting techniques such as the sweep line method and the DP algorithm, the primary contributions of this paper are the trapezoidal mesh data structure to encode tessellated

polygons, the horizontal and vertical generalizations to simplify tessellated polygons, and the heuristic testing algorithm to preserve the topological consistency. These three contributions will be presented in more detail in the following sections.

3. Detailed Method

3.1. Trapezoidal Graph

Three forms of polygon are considered for LoD visualization. While polygons have only one exterior boundary, they may contain zero or more interior boundaries, called simple polygons and polygons with holes, respectively. The regions bounded by these interior boundaries may contain additional sets of interior boundaries; these cases are called polygons with islands. Polygons with cut lines, spikes or punctures are not considered in this paper. The considered types of polygons have three forms of topological relationships: islands may not intersect their surrounding boundaries; all boundaries may not self-intersect; and all boundaries must not mutually intersect. These three types of polygons are illustrated in Figure 1a–c, respectively.

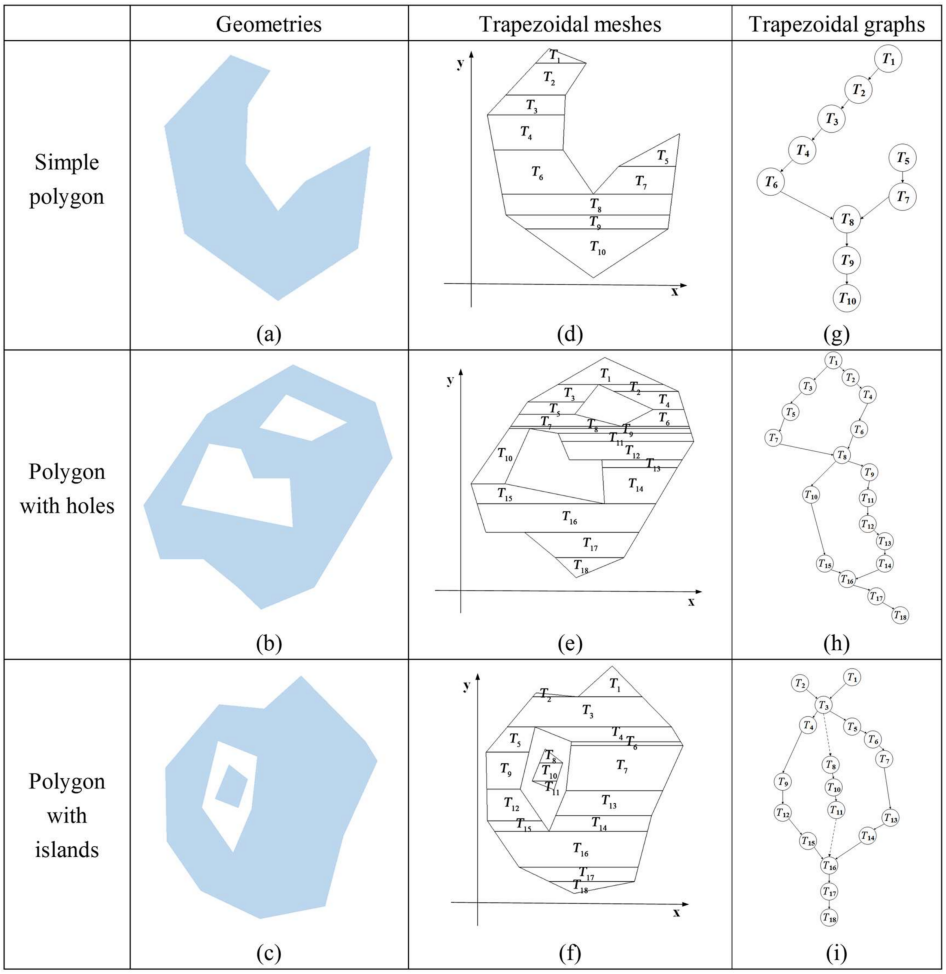

Figure 1. Three types of polygons, their trapezoidations and trapezoidal graphs. (**a**) is a geometry of a simple polygon, (**d**) is the tessellated result of (**a**), (**g**) is the trapezoidal graph of (**a**), (**b**) is a geometry of a polygon with holes, (**e**) is the tessellated result of (**d**), (**h**) is the trapezoidal graph of (**b**), (**c**) is a geometry of a polygon with islands, (**f**) is the tessellated result of (**c**), (**i**) is the trapezoidal graph of a simple polygon of (**d**).

Polygons are tessellated into trapezoidal meshes. For cartographic filling, polygons may need to be tessellated into triangles or trapezoids. To reduce the data rate and increase rendering efficiency, these triangles and trapezoids should be carefully ordered, such as via Hamiltonian triangulations or sequential triangulations in which consecutive triangles share an edge [32]. Trapezoidation is often performed as a first step of triangulation, and with modern hardware, consecutive trapezoids, i.e., those that share an edge, can achieve similar rendering efficiencies to those of sequential triangulations [38]. Trapezoidal meshes are chosen to perform polygon LoD visualization. Here, we simplify the trapezoidal meshes of polygons rather than the polygon boundaries (see Section 5.2 for a more detailed discussion of triangulation and trapezoidation).

Many solutions have been proposed for trapezoidation. One popular solution is the sweep-line algorithm, which can decompose a polygon into trapezoids in any direction [37]. In this paper, horizontal sweep lines are employed to sweep through each vertex of a polygon from top to bottom. All of the sweep lines are parallel, and they decompose a polygon into a series of vertical strips. A sweep line may cross one or more vertexes, decomposing each strip into one or more trapezoids. Generally, a trapezoid consists of four sides (left, right, bottom and top), of which the top and bottom sides are parallel. In practice, a trapezoid may degenerate into a triangle, as shown in T_1 in Figure 1d.

Then, the tessellated polygon is organized as a graph in which a trapezoid is modeled as a node and the relationships between two connected trapezoids are modeled as links; the product is called a trapezoidal graph. Various trapezoidal graphs can be created for different polygon instances [39]. When a node (trapezoid) is connected on one side, either its top or bottom, the node is called a suspended node, as shown by T_1, T_5 and T_{10} in Figure 1d. A convex polygon will be tessellated into a series of trapezoids that are consecutively connected via connecting links. Joined links occur in graphs tessellating a concave polygon, and a division relation occurs when tessellating a polygon with holes. For example, the polygon shown in Figure 1d is a concave instance that includes several concave points at its exterior boundary. In its corresponding trapezoidal graph, shown in Figure 1g, T_6 and T_7 join at T_8. The example shown in Figure 1e is a polygon with two holes. In its corresponding trapezoidal graph, shown in Figure 1h, T_1 will be divided into T_2 and T_3. If a subgraph starts from a suspended node and ends at a join node or starts from a division node and ends at a join node, then its nodes are consecutively chained, creating a trapezoidal chain. Because the relationships between two connected trapezoids are identical in a trapezoidal chain, trapezoidal chains are isomorphic in terms of their links. Therefore, trapezoids are consecutive, and trapezoidal chains are able to facilitate rendering. For example, when using OpenGL as the rendering engine, a trapezoidal chain can be directly translated into a quad-strip for fast rendering. In addition, because self-intersections and intersections are not allowed, triangles' (degenerated trapezoid) nodes exist only at the initial or end node of a trapezoidal chain.

Islands are also tessellated into trapezoidal meshes. The trapezoidal mesh of an island will certainly be disjoined from the other trapezoidal meshes. Nevertheless, the trapezoidal meshes of islands are surrounded by other trapezoidal meshes. Thus, pseudo-divided links and pseudo-joined links are derived from divided links and joined links, respectively, to organize these forms of topological relationships (delineated as dashed arrow lines in Figure 1). The trapezoidal mesh of an island is organized as a subgraph with a pseudo-divided link and a pseudo-joined link. For example, the polygon shown in Figure 1c is a polygon with an island. In its corresponding trapezoidal graph, shown in Figure 1i, T_8 and T_3 are connected via a pseudo-divided link, and T_{16} and T_{11} are connected with a pseudo-joined link.

Trapezoidal graphs encode spatial orders and topological relationships. Because a node corresponds to a trapezoid that specifies a concrete location, nodes are spatially ordered from top to bottom according to the y-coordinates of their top sides and from left to right according to the x-coordinates of their left sides. Specifically, in a trapezoidal graph, a connection link exists for two vertically neighboring trapezoids. A division link also connects two vertically neighboring trapezoids, but the bottom one has at least one sibling. Furthermore, all siblings are mutually separately and

ordered via the increasing x-coordinates of their left sides. Similarly, a joined link connects two vertically neighboring trapezoids, but the top one has siblings. All siblings are ordered by their increasing x-coordinates.

The topmost and bottommost nodes are selected as the starting and ending nodes, respectively. If the topmost sweep line crosses more than one vertex, then several top nodes exist and the leftmost node one is chosen as the starting node. Similarly, the rightmost node is chosen as the ending node when several candidates exist. From the starting node, through the aforementioned three types of links (connection, division and joined, including pseudo-divisions and pseudo-joins), all nodes, including the nodes representing islands, can be tracked downward, upward, and to the left and right. From the starting node, trapezoidal chains can also be generated by tracking the connection links.

3.2. Trapezoidal Mesh Simplifications

Given the cartographic simplification rules, a trapezoidal mesh simplification method is introduced in this section. First, a vertex's weight for preserving the polygon shape is measured. Second, two types of generalization strategies are proposed to generalize the trapezoids. Third, a simplification method is presented that dynamically adapts the trapezoidal meshes according to the zoom level. These operations are discussed in detail below.

One commonly used cartographic simplification rule is to maintain critical points when simplifying polygons. Many methods have been proposed to determine these critical points [14,15], including the widely accepted DP algorithm, which uses a point-to-edge distance to a measure vertex's weight [3]. Here, the DP algorithm is chosen to calculate the vertex weights for polygon LoD rendering (see Section 5.2 for further discussion of the cartographic simplification rules). Similar to the DP-based BLG-tree, all the boundary vertexes are weighted by their point-to-edge distances through binary recursive partitioning. For example, in Figure 2, V_2, V_3, and V_4 are weighted as H_2, H_1 and H_3, respectively.

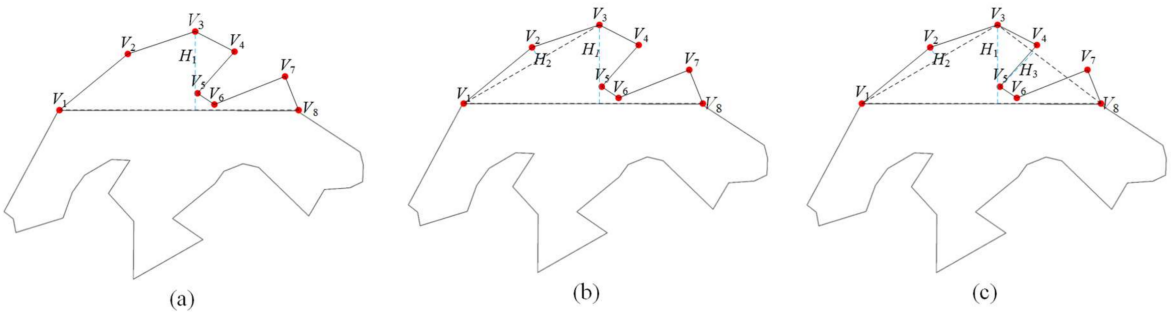

Figure 2. Using the Douglas Peucker (DP) algorithm to weight vertexes [18]. (**a**) V_3 is weighted according to H_1, (**b**) V_2 is weighted according to H_2, (**c**) V_4 is weighted according to H_3.

Trapezoidal generalizations are then introduced to simplify the trapezoidal meshes according to the vertex weights. Given a zoom level, a threshold value of the point-to-edge distance, denoted as ε, can be calculated. When a vertex's weight is less than the threshold value, then that vertex should be deleted; otherwise, it should be kept to preserve the polygon's shape [20]. Considering the diversity of polygonal geometries, various trapezoidal and spatial relationships may exist in concrete trapezoidal meshes. Each node may be singly connected with another node above or below itself (i.e., located at the beginning/ending node of a trapezoidal chain). Such nodes are referred to as one-sided connected nodes. Nodes may also be connected with two nodes, one on either side. These nodes are referred to as two-sided connected nodes. For a connection link, the vertically connected trapezoids should be generalized. For a division or a joined link, horizontally connected trapezoids should be generalized.

In a vertical generalization, removing a vertex may lead to trapezoid merging, degeneration, elimination or adjustment. When tessellating a polygon into trapezoids, one horizontal side of a trapezoid may cross at least one vertex (its weight is denoted as w_1), which is called a one-vertex horizontal side; it also may cross two vertices (whose weights are denoted as w_1 and w_2, respectively, such that $w_1 < w_2$), which is called a two-vertex horizontal side. Specifically, four cases exist for vertical merges, distinguishing whether the vertex which needs to be deleted is shared by two trapezoids and whether the horizontal side is a one-vertex horizontal side or a two-vertex horizontal side: (1) if the vertex is shared by two trapezoids and the horizontal side is a one-vertex side ($w_1 < \varepsilon$), then the two connected trapezoids will be merged into one. Let the symbol $\oplus$ denote the trapezoid merging operator such that $T_i \oplus T_j$ stands for the merged result of trapezoids T_i and T_j. As shown in Figure 3, assume that the weight value of V_{10} is relatively small. To remove V_{10}, T_2 and T_4 should be merged. Then, $T_2 \oplus T_4$ may be further merged with T_1 if V_9 needs to be removed, generating $(T_2 \oplus T_4) \oplus T_1$. (2) If the vertex is shared and the horizontal side is a two-vertex side, then two cases are possible. If the two vertexes that share the horizontal side both need to be deleted ($w_1 < \varepsilon$ and $w_2 < \varepsilon$), as shown in Figure 4b, then the two connected trapezoids will be merged; if the other vertex needs to be kept ($w_1 < \varepsilon$ and $w_2 > \varepsilon$), as shown in Figure 4e, then the two connected trapezoids will be adjusted. (3) If the vertex is not shared and it is a one-weight side ($w_1 < \varepsilon$), as shown in Figure 4a, then the trapezoid will be eliminated; (4) If the vertex is not shared and the side is a two-vertex side, as shown in Figures 4g and 4h, the trapezoid may collapse into a triangle at the vertex with the smaller weight ($w_1 < \varepsilon$ and $w_2 > \varepsilon$), or be removed ($w_1 <$ and $w_2 < \varepsilon$). In total, eight horizontal generalization prototypes are summarized, as shown in Figure 4.

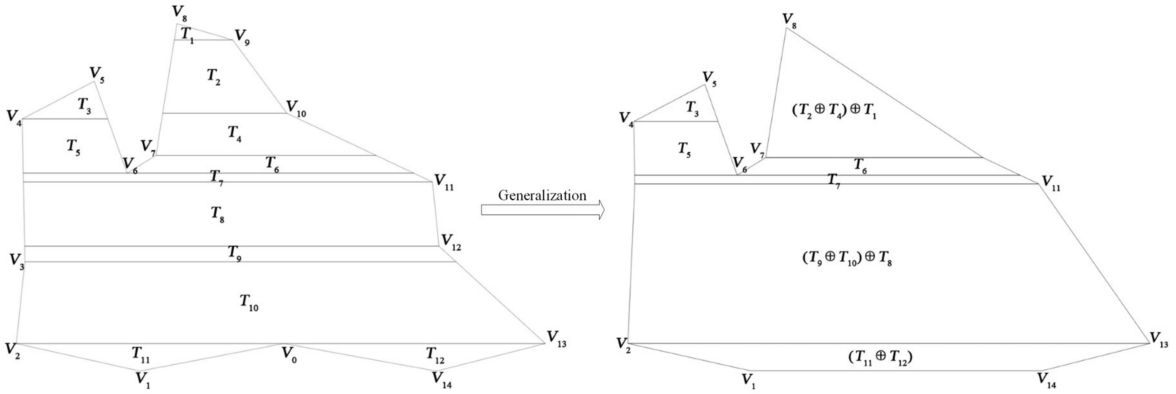

Figure 3. Simplifying a trapezoidal mesh via trapezoidal generalizations.

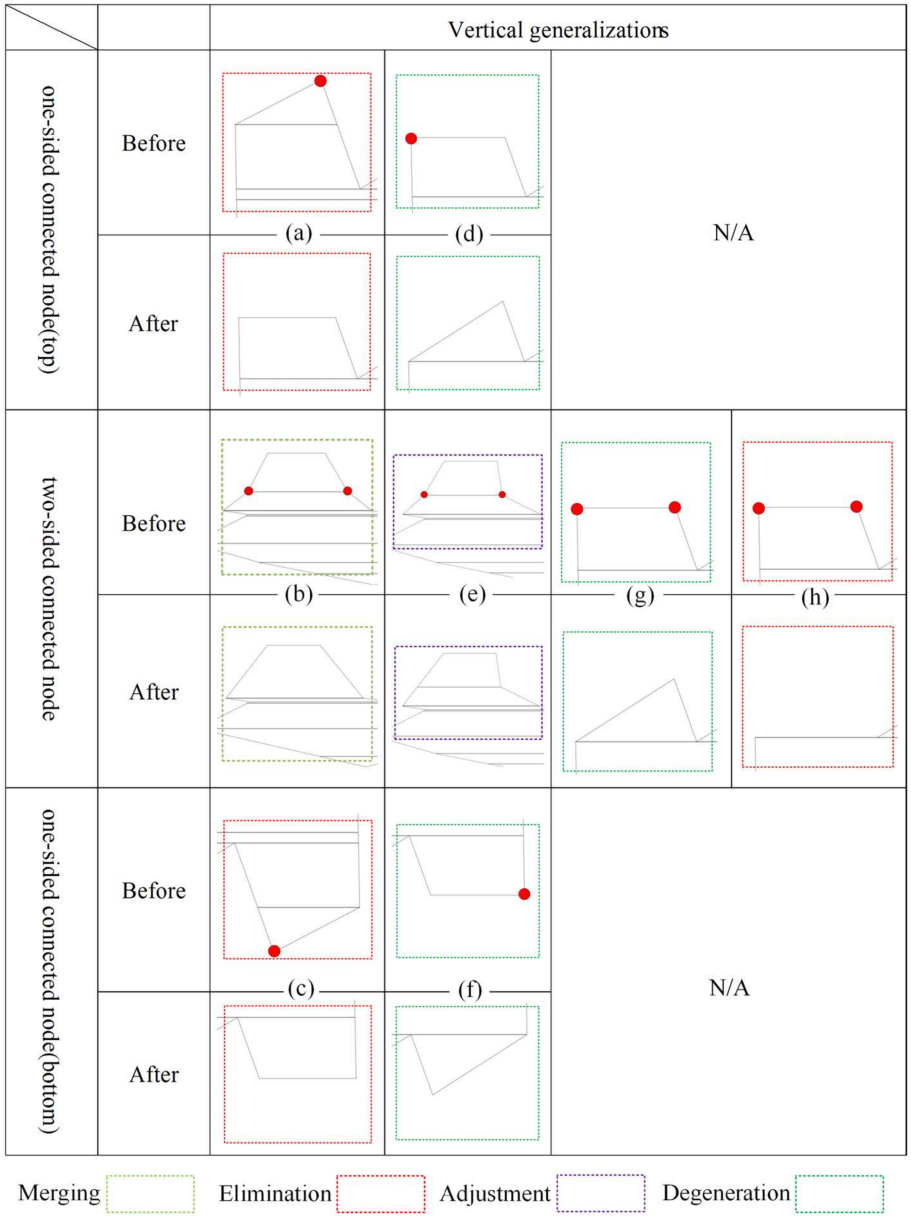

Figure 4. Vertical trapezoidal generalizations, comprising four trapezoidal operators: merging, degeneration, elimination and adjustment. (**a**) trapezoid elimination, (**b**) trapezoid merging, (**c**) is the same with (**a**), but the vertex is at the bottom, (**d**) trapezoid degeneration, (**f**) is the same with (**d**), but the vertex is at the bottom, (**g**) trapezoid degeneration, in which the vertex is a two-weight side and the other vertex needs to be kept, (**h**) trapezoid elimination, in which the vertex is a two-weight side and both need to be deleted, the trapezoid will be removed.

Similarly, trapezoid merging and adjustment also occur with horizontal generalization. Two horizontally neighboring trapezoids can be involved in horizontal merging, which occurs only at division or joining nodes. The two neighboring trapezoids may be merged into a new trapezoid (see Figure 5a) or may be adjusted (see Figure 5c). In total, four merging prototypes are summarized for horizontal generalization, as depicted in Figure 5.

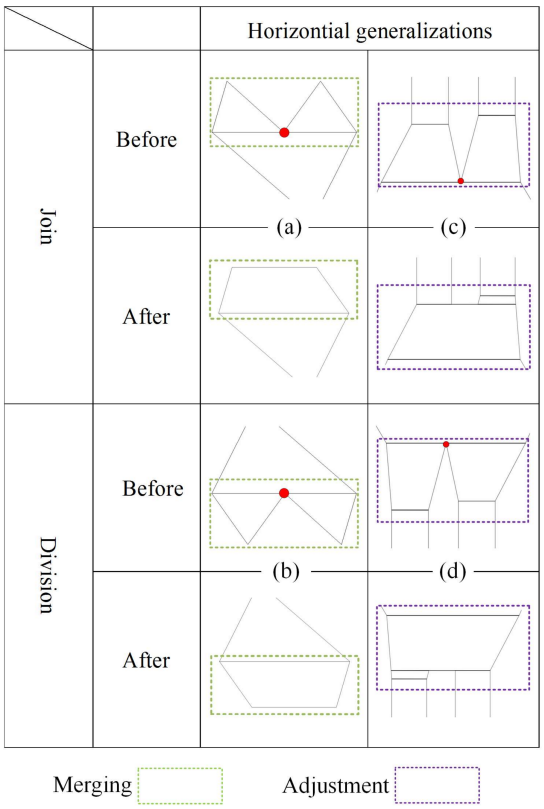

Figure 5. Horizontal trapezoidal generalizations, comprising two trapezoidal operators: merging and adjustment. (**a**) and (**b**) are two cases of merging, the two connected triangles will be merged into a trapezoid, (**c**) and (**d**) are two cases of adjustment, the two connected trapezoids will be adjusted.

Because only two trapezoids experience both horizontal and vertical generalization, their nodes and corresponding links can be updated locally without globally rebuilding the graph (the topological changes will be discussed in next section). For example, Figure 3 shows that merging T_2 and T_4 allows $T_2 \oplus T_4$ to replace T_2 and T_4, and $T_2 \oplus T_4$ will correspondingly inherit the connection relationships of T_2 and T_4. That is, $T_2 \oplus T_4$ are upwardly connected with T_1 and downwardly connected with T_6. Thus, a simplified graph will be generated. The simplified graph represents a simplified trapezoidal mesh and can be translated into consecutive trapezoids for fast rendering.

Based on the generalization prototypes, a trapezoidal mesh simplification method is presented that dynamically chains trapezoidal operators according to their zoom levels. First, a priority stack is introduced. Using binary recursive partitioning, the weighted value of a low level of the partition may be greater than that of a high-level one [18]. In this paper, the original weight values calculated using the DP algorithm are adjusted as follows to guarantee that the vertex with lower weight value has less importance for preserving the polygon shape:

$$H_i = H_i + level_j, \tag{1}$$

where H_i is the weight value for V_i and is calculated at $level_j$ though binary recursive partitioning, and $level_j$ is the parameter for all vertexes that are weighted at $level_j$. Sorting the vertexes according to their weights in descending order generates a priority stack. Then, given a zoom level, a threshold weight value can be calculated. By searching the priority stack, those vertexes whose weights are below the threshold are popped from the priority stack. The related nodes and links of popped vertexes are searched, and the relevant trapezoid is then generalized according to the generalization prototypes. By storing the node and corresponding link statuses before and after each generalization operation, trapezoidal merging, degeneration, elimination and adjustment can be undone. After processing all the popped vertexes, the trapezoidal mesh is dynamically simplified. For a simplified trapezoidal

mesh, moving to a larger zoom level may cause more vertexes to be popped, causing the trapezoidal mesh to be further simplified. Then, a final version of the tessellated polygons can be generated. If the weight of a popped vertex becomes greater than the threshold value, the vertex is pushed back into the priority stack and its status restored. In this manner, a trapezoidal mesh can be dynamically adapted to a zoom level, and the adapted trapezoidal mesh can be translated into consecutive trapezoids for fast rendering.

3.3. Preserving Topological Consistency

As discussed in Section 3.1, three forms of topological relationships are considered and should be kept consistent when performing LoD visualization: for an individual boundary, the boundaries should not self-intersect; no two boundaries, exterior or interior, should intersect; and islands should be preserved without intersections. Using the generalization operators discussed in the previous section, when two neighboring trapezoids are generalized, there is a risk of introducing intersections, which violates topological consistency. We simplify the trapezoidal meshes rather than the polygon boundaries. Using this method, topological errors such as self-intersections of a boundary and any intersections between boundaries, including those of islands, are embodied as intersections between trapezoids. Checking for topological correctness is therefore translated into a problem of testing whether the generalized trapezoids intersect with any other trapezoids.

Clearly, topologic errors can be avoided by exhaustively checking whether a generalized trapezoid intersects with any other trapezoid; however, such checks may be too computationally expensive. Therefore, we propose a heuristic testing algorithm to efficiently check for intersections. First, in a trapezoidal mesh, a trapezoid, T_i cannot intersect with a trapezoid whose projection on the y-axis does not overlap the y-span of T_i. For example, in Figure 6, assuming T_3 and T_8 will be merged, only trapezoids whose y-span intersects the y-span of $T_3 \oplus T_8$ need to be checked.

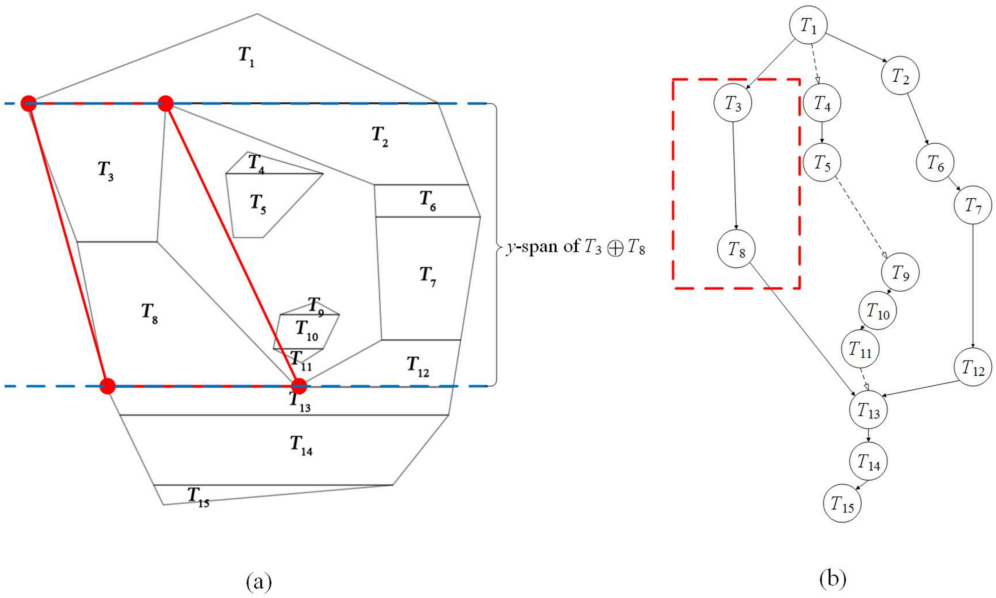

Figure 6. Trapezoid tracking to check for possible intersections. (**a**) is the tessellated result, (**b**) is the trapezoidal graph.

Furthermore, T_i may cover several strips of horizontal trapezoids. In each strip, the trapezoids are ordered by their increasing x coordinates. The trapezoid closest to T_i at its right side is denoted as T_{i+1}. If T_i does not intersect with T_{i+1}, then T_i will not intersect with T_j when $j > i + 1$ and the y-span of T_j is within the y-span of T_{i+1}. For example, in Figure 6, if T_3 does not intersect with T_5, then it will not intersect with T_6. By selecting the closest trapezoids of all the involved strips at both

the right and left sides, a minimal surrounding trapezoid set can be found and denoted as S. S can be identified via trapezoid tracking. For example, in Figure 6, the upwardly tracking node of $T_3 \oplus T_8$ can be identified as the division node T_1. The closest sibling of $T_3 \oplus T_8$ to the right, T_4, can also be identified. For the downwardly tracking node T_4, the node whose y-span is within the y-span of $T_3 \oplus T_8$ belongs to the minimized surrounding trapezoid set. If the y-span of $T_3 \oplus T_8$ is not completely covered, then the downwardly tracking nearest sibling node of $T_3 \oplus T_8$ to the right (if any) is checked until the y-span of $T_3 \oplus T_8$ is either totally covered or all horizontal neighbors to the right side are analyzed. In this manner, S can be generated without performing a brute-force search of the entire trapezoidal mesh. When a trapezoid belonging to S is found, it is checked to determine whether it intersects with $T_3 \oplus T_8$. If the trapezoids intersect, testing ends because an intersection will certainly be introduced. Otherwise, the test continues to the next closest trapezoid. When all items in S have been visited and no intersection has been reported, $T_3 \oplus T_8$ is guaranteed to not intersect with any other trapezoid. In this case, the merging operation will not introduce topological inconsistencies.

More generally, a trapezoid may degenerate, be removed, be merged or be adjusted. In any case, the y-span of a generalized trapezoid can be determined, and the minimized surrounding trapezoid set can also be tracked via graph visiting. Benefiting from the structure of a graph in which both order and topology are embedded, heuristic testing of all generalization cases can be performed without a brute-force search. By rejecting generalization operations that introduce trapezoid intersections, topological consistency can be preserved efficiently without exhaustive checks.

4. Tests and Evaluation

4.1. Test Dataset and Compared Methods

We implemented the algorithm described in Section 3 in C++, using OpenGL 3.0 as the supporting rendering engine. We use the sweep line method proposed by Žalik et al. [37] to tessellate various polygons. While the GAP-tree and its improvements focus on hierarchically organizing an area partitioning, this paper focuses on LoD visualizations of individual polygons. In this sense, GAP-tree and the proposed method are not comparable. Two solutions to support LoD rendering are selected to evaluate the proposed method: in the first compared solution, the polygons are dynamically simplified using BLG-tree (also called the BLG-tree-based method). In the second solution, the polygons are dynamically simplified using the DP algorithm (also called the DP-based method). In both solutions, the simplified polygons are tessellated into trapezoidal meshes and then rendered. Three OpenStreetMap [40] polygon datasets (i.e., including buildings, nature, and land use) were selected as the test datasets. The covered area is located at Shenzhen (113°22′21″ E, 114°57′48″ E; 22°13′30″ N, 22°44′38″ N) in southeast Guangdong Province, China and contains a total of 42,334 polygons. These three datasets are organized as three map layers and are symbolized using solid-color fills. Zoom (in and out) and pan functions were implemented to control the zoom level and view. A volunteer was invited to operate the map to arbitrarily explore these datasets. All the tests were performed at a display resolution of 1600 × 1200 on a PC running Windows 7 with a 2.66 GHz Intel Core 2 Quad CPU, 8 GB of RAM and an ATI Radeon HD 5870 GPU (driver version 10.6).

4.2. Results

The performance efficiencies (in terms of drawing time, including the simplifying, tessellating and rendering times) of 72 consecutive views were recorded both with and without preserving the topological consistency. Five example views and their corresponding trapezoidal meshes are shown in Figure 7. We use the method suggested by Corcoran et al. [23] to preserve topological consistency in both the DP-based and the BLG-tree-based methods. Efficiency comparisons of all three algorithms are shown in Figure 8. For the DP-based method, simplification and trapezoidation were performed for every view. In the BLG-tree-based method, the simplification speed was improved by employing the BLG-tree; therefore, the BLG-tree-based method was significantly more efficient than the DP-based

method. However, trapezoidation was still executed for every zoom level. Because the simplified polygons must be tessellated for every zoom level, the trapezoidal meshes must be sent to the rendering engine for every views when directly applying both the BLG-tree-based method and the DP-based method. In contrast, the proposed method requires the trapezoidal meshes to be generated only for the initial frame; thus, simplification and tessellation can be achieved concurrently though trapezoidal generalization. Using the generalization prototypes, the computational cost of the trapezoidal generalization is far less than when separate simplification and tessellation operations are used. In the proposed method, none of the generalization operators except adjustment introduce any additional vertexes. Therefore, the trapezoidal meshes need to be sent to the rendering engine only once. Only the index of the trapezoidal meshes must be sent for rendering and the trapezoidal meshes are updated only when they are adjusted. Thus, the data rate is dramatically reduced, and, consequently, the rendering efficiency increases. For the above reasons, as illustrated in Figure 8, the proposed method runs significantly faster, achieving speeds approximately six times faster than the other two methods.

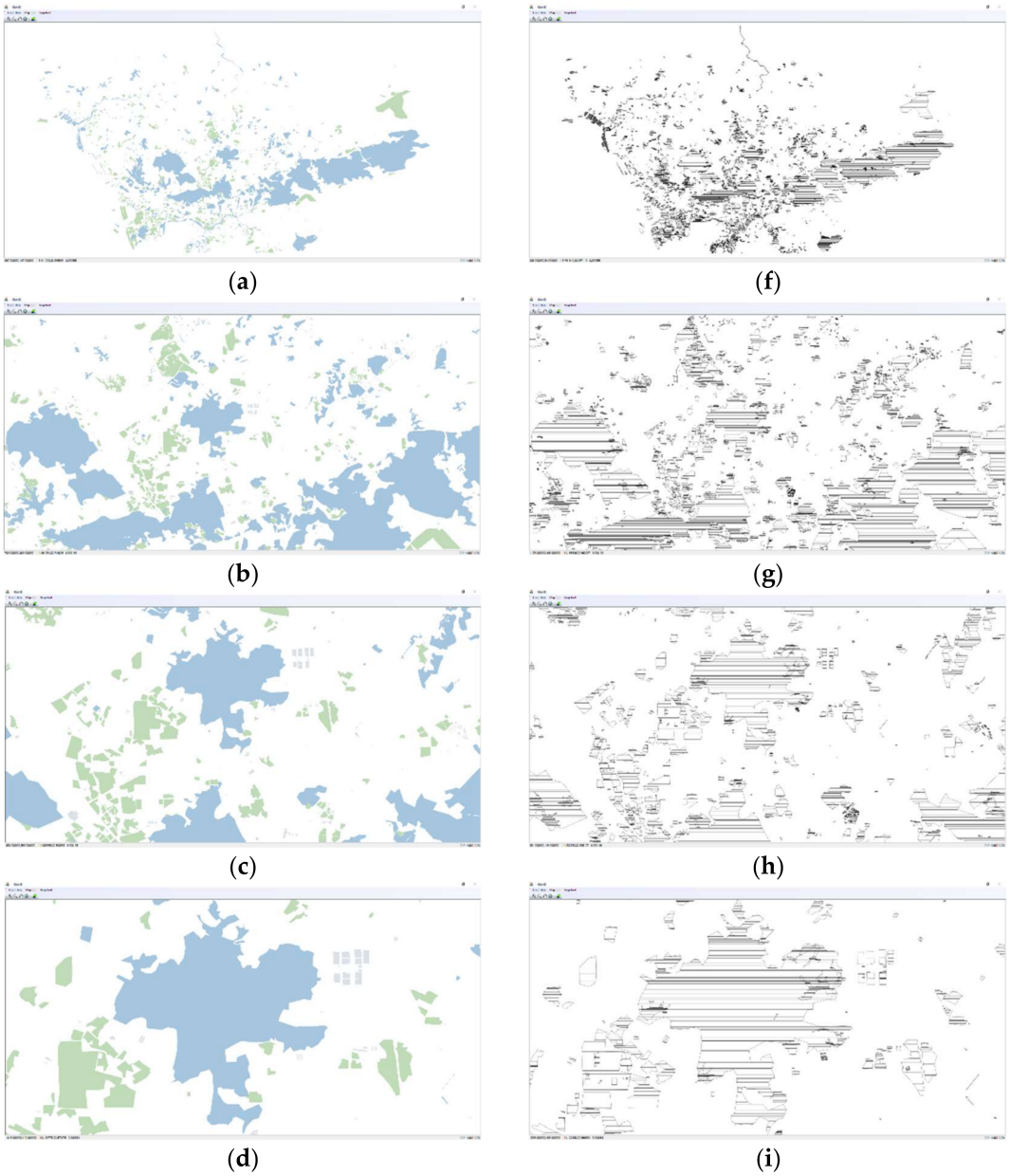

Figure 7. *Cont.*

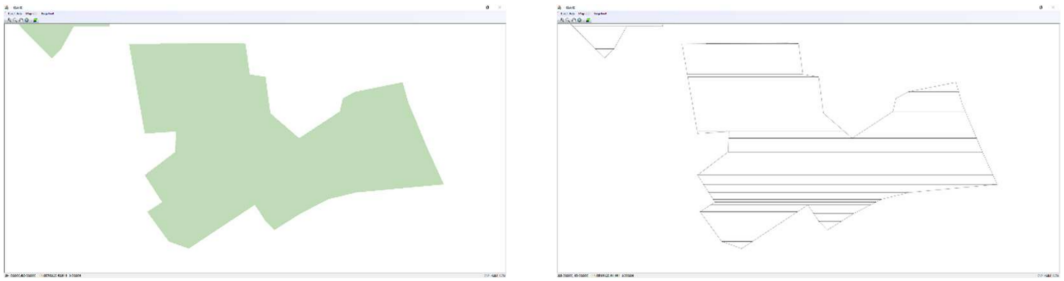

Figure 7. Screenshots of polygon level of detail (LoD) rendering [2]. (**a**–**e**) are the rendered results of the test polygons at five zoom levels (1:250,000, 1:80,000, 1:50,000, 1:29,000, 1:3,000, respectively), while (**f**–**j**) are the corresponding adapted trapezoidal meshes.

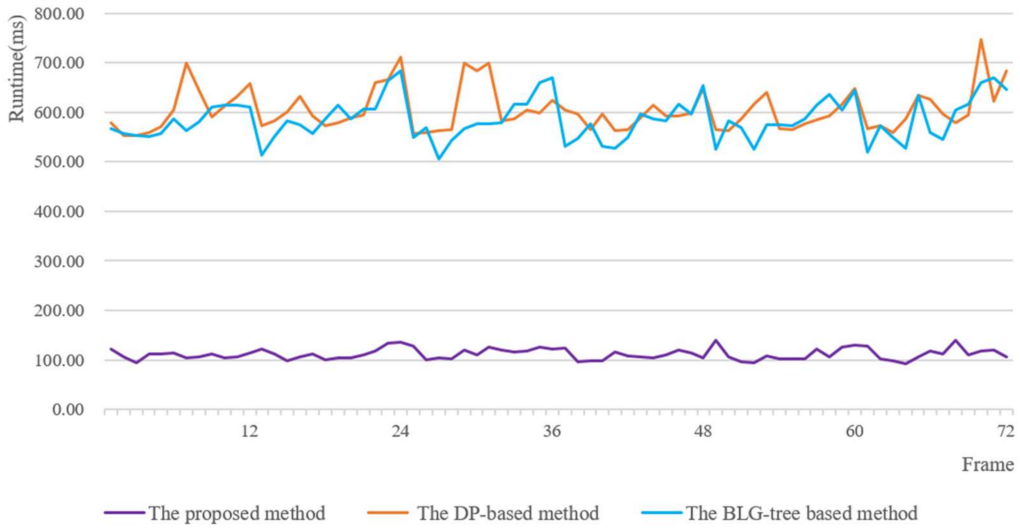

Figure 8. Rendering times for the test polygons using the three compared algorithms.

5. Discussion

5.1. Efficiency

Simplification, tessellation and rendering are the three major phases of polygon LoD visualization. The detailed time costs for simplification, tessellation and rendering steps were analyzed, and the results are shown in Figure 9. Generally, in this test, the average proportions of the time consumed by the simplification, tessellation and rendering stages of the DP-based method are approximately 24%, 9% and 67%, respectively. The BLG-tree-based method speeds up simplification by employing the BLG-tree; thus, the average proportions of its time consumed by the simplification, tessellation and rendering stages are approximately 17%, 8% and 74%, respectively. In the proposed method, the simplification and tessellation stages are achieved concurrently by performing trapezoidal generalizations. By defining the generalization prototypes, the trapezoidal generalizations can be efficiently implemented. As shown in Figure 9, the combined simplification and tessellation step of the proposed method costs significantly less time than do the separate steps in the DP-based and BLG-tree-based methods. Furthermore, because the proposed method also reduces the data rate, the rendering time is significantly less than that of the other two methods. These statistics will certainly vary for different polygons and different software and hardware environments, but they help explain why the proposed method performs significantly better than the other two methods.

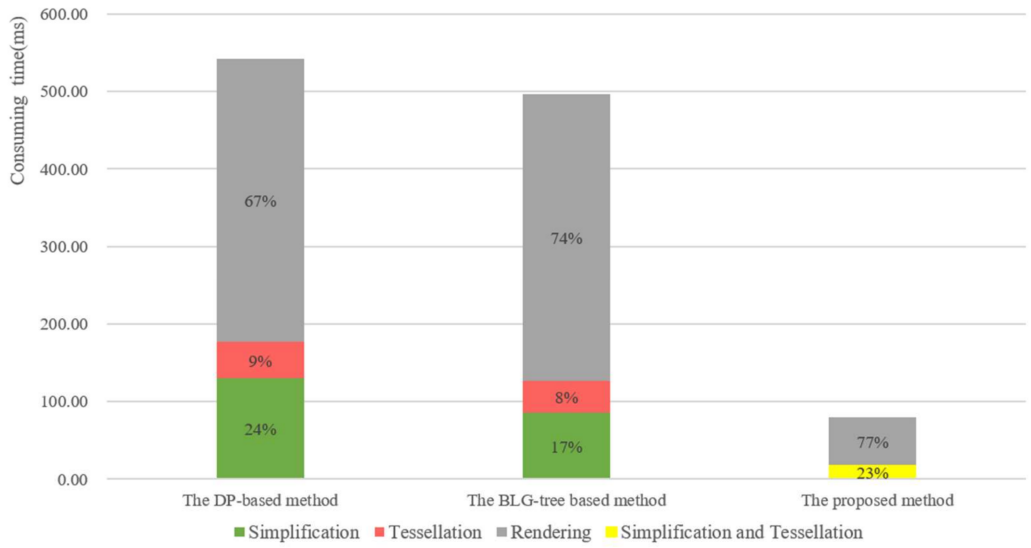

Figure 9. Detailed statistics of the time costs (in milliseconds) of the simplification, tessellation and rendering steps.

In this study, trapezoids are selected as the base unit to simplify and tessellate the polygons. A triangle is also a candidate base unit for polygon simplification, tessellation and rendering. When tessellating polygons into triangle meshes, a single vertex may be linked to many triangles, which can lead to algorithmic complexities when removing vertexes for LoD purposes. In contrast, when tessellating polygons into trapezoidal meshes, a vertex can be connected to only one or two trapezoids, which helps in straightforwardly and efficiently generalizing connected trapezoids for LoD. Certainly, new vertexes could be introduced in trapezoidation that would facilitate simplifying trapezoidal meshes; however, introducing new vertexes would also increase the required memory and the data rate at the rendering stage. In fact, these vertexes are not necessary for rendering. As discussed in Section 3.1, trapezoids in a chain are consecutive: connected trapezoids share an edge, and thus a trapezoidal chain can be translated into a quad-strip. Breaking a quad-strip at the introduced vertexes allows a trapezoidal chain to be translated as a series of triangle-strips without using the introduced vertexes. Then, the data rate can be further reduced, but this method can also result in a relatively large number of drawable primitives and negatively affect the rendering efficiency. The impact of the introduced vertexes is also dependent on the implemented rendering technique, such as whether GPU-accelerated rendering is used, which is an effect that requires further research. In addition, maintaining the graphs needs considerable memory, but it is also needed when tessellating the polygon in the DP-based and BLG-tree-based methods.

The computational complexity of the trapezoidation of simple polygons is $O(n)$, where n is the vertex count of the polygon [37]. Using this method, trapezoidation needs to be conducted only once. Furthermore, only two neighboring trapezoids are involved in the generalization operators discussed in Section 3.2. Tessellated polygons can therefore be simplified in linear time without topological check, and the computational complexity of generalization operations is $O(m)$, where m is the trapezoid count of the tessellated polygon. This approach also offers the possibility of parallelizing the horizontal and vertical generalization operations, which will be addressed in future work.

5.2. Quality

In this paper, vertexes are weighted using the DP algorithm. A variety of methods have been proposed to simply polygons, such as the B-spline snake model [41] and the energy minimization method [3]. Different methods may yield considerably different simplified polygons. Trapezoidal generalizations are based on weighted vertexes. Theoretically, these generalizations are compatible

with different methods of calculating the weights. In future work, we will extend our method to support additional methods of cartographic simplification.

In this study, self-intersections and intersections are checked using the heuristic testing algorithm. Many other issues, such as size conflicts (wherein a simplified polygon may be too small) and proximity conflicts (wherein a simplified polygon may be too close to another simplified polygon), should also be examined in advanced polygon simplifications, or, more generally, in any polygon generalization [3]. Four types of trapezoidal generalization operators, including merging, degeneration, elimination and adjustment, are introduced in this paper; however, more sophisticated operators, such as displacement and exaggeration, can be developed to support other advanced polygon generalizations. For example, as shown in Figure 10a, when a simplified trapezoid mesh is too close to its neighbors after simplification, it should purposely be moved to the opposite side. Additionally, as shown in Figure 10b, if some trapezoids of a simplified trapezoidal mesh are too small, then those trapezoids should be purposefully enlarged while still preserving the connections at shared faces. Those sophisticated operators are also potentially suitable to generate point, and polyline objects because they also need to be tessellated into drawable primitives for symbolization. In this sense, this method can be extended for general maps consisting of point, polyline and polygon objects. Defining these sophisticated operators will require further research.

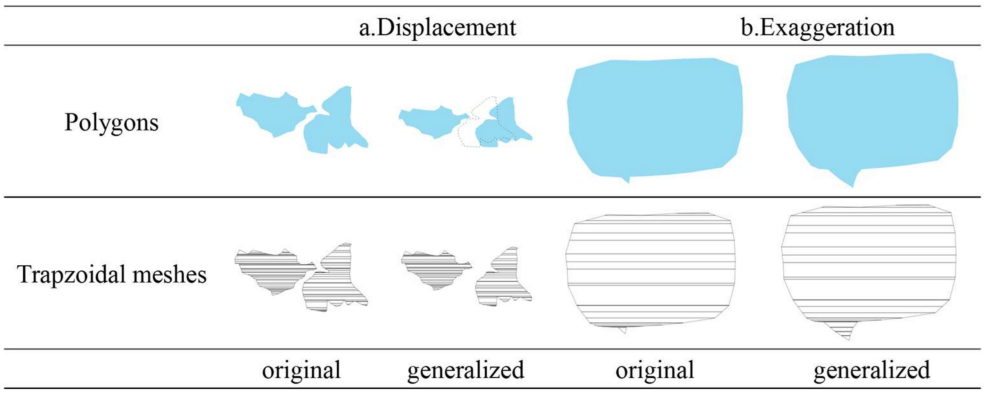

Figure 10. Additional possible sophisticated operators to support polygon displacement and exaggeration.

6. Conclusions

Polygon LoD visualization is functionally necessary in geo visualization but suffers from low efficiency, especially when visualizing complex polygonal data. Simplification, tessellation and rendering are three major successive phases that are treated as isolated processes in the existing methods, yielding computational and data redundancies that reduce efficiency. In this paper, an efficient method is proposed to facilitate polygon LoD visualization. The contributions of this method are summarized as follows.

Unlike the existing methods, we organize simplification, tessellation and rendering operations into a single mesh generalization process that avoids having to completely re-execute tessellation operations and resend all drawable primitives for rendering. Beyond the supporting techniques such as the sweep line method and the DP algorithm, to the best of our knowledge, this paper is the first to propose the trapezoidal mesh data structure to encode the spatial and topological elements of tessellated polygons, the horizontal and vertical generalizations for rapid trapezoidal mesh simplification, and the heuristic testing algorithm to efficiently preserve topological consistency. In our approach, the simplification and tessellation operations are systematically organized into a single process that avoids computational and data redundancies and helps to mitigate the rate sensitivity.

The tests conducted on real world datasets suggest that, compared with the DP-based and BLG-tree-based methods, the proposed method significantly improves the efficiency of polygon

LoD visualization. This method can benefit efficiency-sensitive vector mapping applications such as emergency mapping cases in which polygonal data need to be symbolized and simplified in a timely fashion. We implemented and test this algorithm in a desktop environment; however, it could be implemented in a client/server architecture or an embedded system to provide rapid vector mapping responses. Future work will include the consideration of additional criteria for polygon simplification, such as shape preservation, and additional operators to support more complex cartographic generalizations such as displacement and exaggeration. In addition, while trapezoids are generalized individually in this paper, batch generalization will also be addressed in future work. Extending this method to support polygonal maps and even general maps also requires future work.

Acknowledgments: This work was supported by the National Natural Science Foundation of China (No. 41571433, 41271446, 41571439, 41201485).

Author Contributions: Mingguang Wu wrote the paper; Mingguang Wu, Taisheng Chen and Guonian Lv conceived and designed the experiments; Kun Zhang and Zhimin Jing performed the experiments; Yangli Han and Menglin Chen analyzed the results. Hong Wang conducted the literature search and created the charts.

Conflicts of Interest: There is no conflict of interest.

References

1. Wu, M.; Zhu, A.; Zheng, P.; Cui, L.; Zhang, X. An improved map-symbol model to facilitate sharing of heterogeneous qualitative map symbols. *Cartogr. Geogr. Inf. Sci.* **2017**, *44*, 62–75. [CrossRef]
2. Zeiler, M. *Modeling Our World: The Esri Guide To Geodatabase Design*, 2nd ed.; ESRI Press: California, CA, USA, 1999; ISBN 978-1589482784.
3. Galanda, M. Automated Polygon Generalization in A Multi Agent System. Ph.D. Thesis, University of Zurich, Zurich, Switzerland, 2003.
4. Zhou, M.; Chen, J.; Gong, J. Rendering interior-filled polygonal vector data in a virtual globe. *Int. J. Geogr. Inf. Sci.* **2016**, *30*, 2208–2229. [CrossRef]
5. Wu, M.; Zheng, P.; Lu, G.; Zhu, A. Chain-based polyline tessellation algorithm for cartographic rendering. *Cartogr. Geogr. Inf. Sci.* **2017**, *44*, 491–506. [CrossRef]
6. Mustafa, N.; Krishnan, S.; Varadhan, G.; Venkatasubramanian, S. Dynamic simplification and visualization of large maps. *Int. J. Geogr. Inf. Sci.* **2006**, *20*, 273–302. [CrossRef]
7. Floriani, L.D.; Magillo, P. Multiresolution Meshes Representation: Models and Data Structures. In *Tutorials on Multiresolution in Geometric Modelling*; Iske, E., Quak, E., Floater, M.S., Eds.; Springer: Berlin, Germany, 2002; pp. 193–234. ISBN 978-3-662-04388-2.
8. Azuma, D.I.; Wood, D.N.; Curless, B.; Duchamp, T.; Salesin, D.H.; Stuetzle, W. View-dependent refinement of multiresolution meshes with subdivision connectivity. In Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation And Interaction in Africa, Cape Town, South Africa, 3–5 February 2003; pp. 69–78.
9. Hoppe, H. Progressive meshes. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA, 4–9 August 1996; pp. 99–108.
10. Hoppe, H. Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering. In Proceedings of the Visualization'98, North Carolina, NC, USA, 18–23 October 1998; pp. 35–42.
11. Follin, J.M.; Bouju, A. An incremental strategy for fast transmission of multi-resolution data in a mobile system. In *Map-Based Mobile Services*; Springer: Berlin, Germany, 2008; pp. 57–97. ISBN 978-3-540-26982-3.
12. Putten, J.V.; Oosterom, P.V. New result with Generalized Area Partitionings. In Proceedings of the International Symposium on Spatial Data Handling, SDH'98, Vancouver, Canada, 12–15 July 1998; pp. 485–495.
13. Meijers, M.; Oosterom, P.V. The space-scale cube: An integrated model for 2D polygonal areas and scale. In Proceedings of the 28th Urban Data Management Symposium, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Delft, The Netherlands, 28–30 September 2011; pp. 95–102.
14. Li, Z. An examination of algorithms for the detection of critical points on digital cartographic lines. *Cartogr. J.* **1995**, *32*, 121–125. [CrossRef]

15. Shi, W.; Cheung, C. Performance evaluation of line simplification algorithms for vector generalization. *Cartogr. J.* **2006**, *43*, 27–44. [CrossRef]
16. Podolskaya, E.S.; Anders, K.H.; Haunert, J.H.; Sester, M. Quality Assessment for Polygon Generalization. In *Quality Aspects in Spatial Data Mining*; Taylor & Francis: Oxford, UK, 2009; Volume 211. [CrossRef]
17. Haunert, J.H.; Dilo, A.; Oosterom, P.V. Constrained set-up of the tGAP structure for progressive vector data transfer. *Comput. Geosci.* **2009**, *35*, 2191–2203. [CrossRef]
18. Oosterom, P.V.; Bos, J.V.D. An Object-Oriented Approach to the Design of Geographic Information Systems. *Comput. Graph.* **1989**, *13*, 409–418. [CrossRef]
19. Oosterom, P.V. The GAP-tree, an approach to "On-the-Fly" Map Generalization of an Area Partitioning. In Proceedings of the GISDATA Specialist Meeting on Generalization, Compiègne, France, 15–19 December 1993.
20. Oosterom, P.V. Variable-scale topological data structures suitable for progressive data transfer: The GAP-face tree and GAP-edge forest. *Cartogr. Geogr. Inf. Sci.* **2005**, *32*, 331–346. [CrossRef]
21. Oosterom, P.V.; Meijers, M. Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *Int. J. Geogr. Inf. Sci.* **2014**, *28*, 455–478. [CrossRef]
22. Zhou, S.; Jones, C.B. *Shape-Aware Line Generalisation with Weighted Effective Area*; Springer: Berlin, Germany, 2005; ISBN 978-3-540-22610-9.
23. Corcoran, P.; Mooney, P.; Winstanley, A. Planar and non-planar topologically consistent vector map simplification. *Int. J. Geogr. Inf. Sci.* **2011**, *25*, 1659–1680. [CrossRef]
24. Jones, C.B.; Kdner, D.B.; Luo, L.Q.; Bundy, G.L.; Ware, J.M. Database design for a multi-scale spatial information system. *Int. J. Geogr. Inf. Syst.* **1996**, *10*, 901–920. [CrossRef]
25. Han, Q.; Bertolotto, M. A multi-level data structure for vector maps. In Proceedings of the 12th Annual Acm International Workshop on Geographic Information Systems, Washington, DC, USA, 8–13 August 2004; pp. 214–221.
26. Ai, T.; Li, Z.; Liu, Y. Progressive transmission of vector data based on changes accumulation model. In Proceedings of the Developments in Spatial Data Handling: 11th International Symposium on Spatial Data Handling, Leicester, UK, 23–25 August 2004; pp. 85–96.
27. Yang, B.; Purves, R.; Weibel, R. Efficient transmission of vector data over the internet. *Int. J. Geogr. Inf. Sci.* **2007**, *21*, 215–237. [CrossRef]
28. Papadimitriou, F. The Algorithmic Complexity of Landscapes. *Landsc. Res.* **2012**, *37*, 591–611. [CrossRef]
29. Wood, J.; Kirschenbauer, S.; Döllner, J.; Lopes, L.; Bodum, L. Chapter 14—Using 3D in Visualization. In *Exploring Geovisualization*, 1st ed.; Elsevier: Amsterdam, The Netherlands, 2005; pp. 295–312. ISBN 9780080445311.
30. Zhang, J.; Zhu, Y. A Method Based on Graphic Entity for Visualizing Complex Map Symbols on the Web. *Cartogr. Geogr. Inf. Sci.* **2015**, *42*, 44–53. [CrossRef]
31. Yue, S.; Yang, J.; Chen, M.; Lu, G.; Zhu, A.; Wen, Y. A function-based linear map symbol building and rendering method using Shader language. *Int. J. Geogr. Inf. Sci.* **2016**, *30*, 143–167. [CrossRef]
32. Arkin, E.M.; Held, M.; Mitchell, J.S.B.; Skiena, S.S. Hamiltonian triangulations for fast rendering. *Vis. Comput.* **1996**, *12*, 429–444. [CrossRef]
33. Rueda, A.J.; Miras, J.R.; Feito, F.R. GPU Based Rendering of Curved Polygons Using Simplicial Coverings. *Comput. Graph.* **2008**, *32*, 581–588. [CrossRef]
34. Rougier, N.P. Shader-Based Antialiased Dashed Stroked Polylines. *J. Comput. Graph. Technol.* **2013**, *2*, 105–121.
35. Ma, J.; Xu, S.; Pu, Y.; Chen, G. A real-time parallel implementation of Douglas-Peucker polyline simplification algorithm on shared memory multi-core processor computers. In Proceedings of the International Conference on Computer Application and System Modeling (ICCASM), Taiyuan, China, 22–24 October 2010.
36. MapBox. Available online: https://www.mapbox.com/pricing (accessed on 31 January 2018).
37. Žalik, B.; Jezernik, A.; Žalik, K.R. Polygon trapezoidation by sets of open trapezoids. *Comput. Graph.* **2003**, *27*, 791–800. [CrossRef]
38. Lorenzetto, G.P.; Datta, A.; Thomas, R.C. A fast trapezoidation technique for planar polygons. *Comput. Graph.* **2002**, *26*, 281–289. [CrossRef]
39. Mąka, M. The recurrent algorithm for area discretization using the trapezoidal mesh method. *Zesz. Nauk. Akad. Morska w Szczec.* **2012**, *29*, 134–139.

40. OpenStreetMap. Available online: http://www.openstreetmap.org/ (accessed on 25 May 2018).
41. Guilbert, E.; Saux, E. Cartographic generalisation of lines based on a B-spline snake model. *Int. J. Geogr. Inf. Sci.* **2008**, *22*, 847–870. [CrossRef]