Efficient Simplification of Large Vector Maps Rendered onto 3D Landscapes

Ling Yang and Liqiang Zhang ■ *State Key Laboratory of Remote Sensing Science*

Jingtao Ma ■ *PTV America*

Zhizhong Kang ■ *China University of Geosciences*

Lixin Zhang ■ *State Key Laboratory of Remote Sensing Science*

Jonathan Li ■ *University of Waterloo*

Real-time rendering of large-scale vector maps over terrain surfaces requires displaying substantial numbers of polylines and polygons. The proposed approach simplifies such maps, permitting more efficient rendering and reducing latency in the display and manipulation of a virtual environment.

Overlaying 2D vector maps onto 3D digital elevation models (DEMs) communicates information about the topography and landscape objects in a convenient, easily understood form. So, quickly rendering vector maps in 3D landscapes is an important goal in spatial-decision VR applications, such as land-use planning with computerized models.

Level-of-detail (LOD) terrain models decrease the complexity of 3D spatial object representation and are often used with high-resolution landscape visualization to achieve high-performance rendering. However, in these models, changing the virtual observer's point of view often requires terrain surface reconstructions, and the overlay process should make the corresponding adjustments in real time. Furthermore, different applications might use different LOD terrain models, and many overlaying methods are developed ad hoc and are hard to adapt to general situations.

Maintaining valid, consistent topology in vector map data is a prerequisite to obtaining correct results for queries and spatial analysis. However, owing partly to the intrinsic complexity of simplifying vector maps while maintaining consistent topology, few studies have tackled this problem in a real map context, working with multiple object types such as lines and polygons while achieving computational efficiency.

Here, we show how we construct LOD vector map models that can rapidly overlay large-scale vector maps on multiresolution terrain models. Our approach doesn't generate perceivable cracks or aliasing artifacts and is independent of LOD DEMs. Performance isn't affected by the DEM dataset; it's related only to the vector maps' complexity. To generate the vector maps, we developed a map simplification algorithm that avoids changes to the original topological connectivity. This algorithm is also applicable to other tasks, such as vector data progressive transmission.

Vector Map Simplification

Our study focuses on polyline simplification because most map features are represented as lines or as polygons made up of lines. (For a look at

 Published by the IEEE Computer Society

Related Work on Map Simplification

Many researchers have investigated map simplification. The Douglas-Peucker algorithm, one of the most popular line simplification methods, can keep important geometric characteristics of original lines.[1] Mahes Visvalingam and J.D. Whyatt's line simplification algorithm removes the vertices forming the smallest "effective area" triangles.[2] Zhiyuan Zhao and Alan Saalfeld proposed a sleeve-fitting algorithm that runs in linear time.[3] Wenzhong Shi and ChuiKwan Cheung evaluated several geometric line simplification methods and found that the Douglas-Peucker algorithm produced the most accurate results.[4]

However, all these methods can introduce topological errors such as intersections between lines and self-intersections. The algorithms must include restrictions for avoiding such errors.

To address this issue, Regina Estkowski and Joseph Mitchell described a heuristic method for simplifying parallel lines without intersections—in cubic time in the worst case.[5] Leonidas Guibas and his colleagues pointed out that the problem of computing a minimum-link approximation of a simple polygon while preventing self-intersections is NP-hard.[6]

Andrea Mantler and Jack Snoeyink presented a Voronoi diagram algorithm that can maintain topological relationships.[7] Their method first divides a complex polyline into a collection of safe sets. It then simplifies each safe set by using a single line segment or a standard polyline simplification algorithm such as the Douglas-Peucker algorithm. This method's time complexity for computing a Voronoi diagram is $O(n\log n)$ and for computing safe sets is $O(n)$.

Nabil Mustafa and his colleagues also implemented a Voronoi-diagram-based algorithm to perform dynamic view-dependent simplification of large geographical maps.[8] By using graphics hardware, they achieved high computational efficiency and avoided intersections among the output lines. However, the frame buffer resolution limits that algorithm's precision, and it can't avoid self-intersections.

References

1. D.H. Douglas and T.K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature," *Canadian Cartographer*, vol. 10, no. 2, 1973, pp. 112–122.
2. M. Visvalingam and J.D. Whyatt, "Line Generalisation by Repeated Elimination of Points," *Cartographic J.*, vol. 30, no. 1, 1993, pp. 46–51.
3. Z. Zhao and A. Saalfeld, "Linear-Time Sleeve-Fitting Polyline Simplification Algorithms," *Proc. Auto-Carto XIII*, pp. 214–223.
4. W.Z. Shi and C.K. Cheung, "Performance Evaluation of Line Simplification Algorithms for Vector Generalization," *Cartographic J.*, vol. 43, no. 1, 2006, pp. 27–44.
5. R. Estkowski and J.S.B. Mitchell, "Simplifying a Polygonal Subdivision While Keeping It Simple," *Proc. 17th ACM Symp. Computational Geometry*, ACM Press, 2001, pp. 40–49.
6. L.J. Guibas et al., "Approximating Polygons and Subdivisions with Minimum Link Paths," *Int'l J. Computational Geometry and Applications*, vol. 3, no. 4, 1993, pp. 383–415.
7. A. Mantler and J. Snoeyink, "Safe Sets for Line Simplification," *Proc. 10th Ann. Workshop Computational Geometry*, Stony Brook Univ., 2000.
8. N. Mustafa et al., "Dynamic Simplification and Visualization of Large Maps," *Int'l J. Geographical Information Science*, vol. 20, no. 3, 2006, pp. 273–320.

other map simplification research, see the related sidebar.) We first split each polyline into monotone pieces. Then, we simplify the resulting subpolylines through the rendering processes of graphics hardware. Finally, we connect the simplified subpolylines to produce the vector map. Besides avoiding intersections and self-intersections, this algorithm can perform better than conventional geometric techniques.

Generating Monotone Subpolylines

A polyline l is monotone if a line l_d exists such that any line l_s perpendicular to l_d has at most one intersection with l (see Figure 1). A monotone polyline doesn't intersect with itself. Its simplified version will still be monotone and won't self-intersect either.

To get monotone pieces, we first connect each pair of successive vertices to form vectors. Then, we define the angle formed by the vectors $\vec{p}$ and $\vec{q}$. If the angle determined by counterclockwise rotation from $\vec{p}$ to $\vec{q}$ is less than 180 degrees, we use that value. If it's greater than 180 degrees, we subtract 360 from it and use that value instead. So, both angles lie in the range (–180°, 180°].

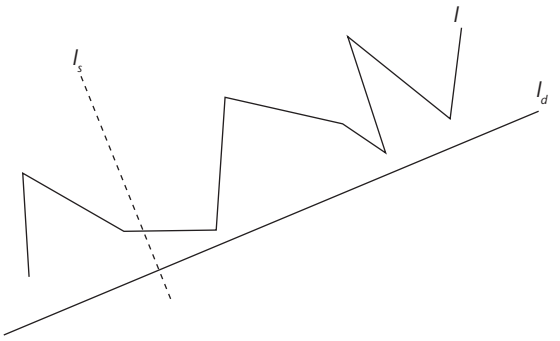
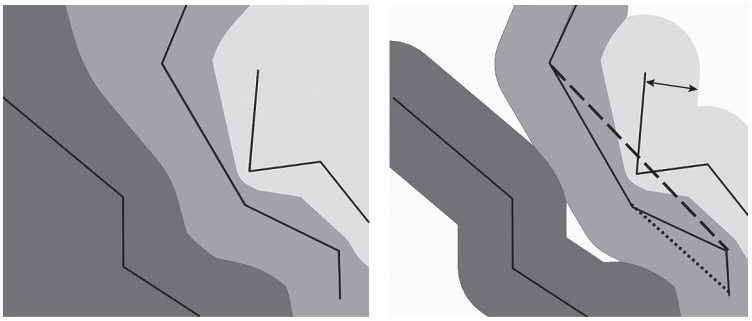


Figure 1. A monotone polyline. Line l_s, perpendicular to line l_d, intersects with the polyline only once, so the polyline is monotone.



(a) (b)

Figure 2. To simplify monotone subpolylines, we use ε-Voronoi diagrams. (a) Voronoi diagrams of three polylines. The dark gray zone is the leftmost polyline's Voronoi cell because this zone's pixels are closer to the leftmost polyline than the others. (b) ε-Voronoi cells. The distance from a polyline to its ε-Voronoi cell's boundary isn't larger than ε. The dashed line shows the noncompliant segment, which is a segment of the middle polyline and beyond this polyline's ε-Voronoi cell. The dotted line shows the compliant segment, which lies completely in the middle polyline's ε-Voronoi cell.

For a polyline $l = (v_0, v_1, ..., v_n)$, we define the *edge angle* θ_i associated with edge $\overline{v_i v_{i+1}}$ by the angle from $\overrightarrow{v_0 v_1}$ to $\overrightarrow{v_i v_{i+1}}$. V. Chandru and colleagues have proved that l is monotone if and only if its edge-angle sequence $\{\theta_0, \theta_1, ..., \theta_{n-1}\}$ satisfies $(\theta_{max} - \theta_{min}) < 180$, where $\theta_{max} = \max(\theta_0, \theta_1, ..., \theta_{n-1})$ and $\theta_{min} = \min(\theta_0, \theta_1, ..., \theta_{n-1})$.[1] So, starting with θ_0, we compute the edge angles in sequence and update θ_{max} and θ_{min} simultaneously. If $(\theta_{max} - \theta_{min}) \geq 180$ degrees at θ_i, we split the polyline at v_i and restart the computation with v_i. In this way, we can generate monotone subpolylines with time complexity $O(n)$.

Simplifying Monotone Subpolylines

Now, we further simplify the subpolylines and ensure that the simplified results don't intersect with each other.

To constrain the simplification, we rely on ε-*Voronoi diagrams*. A Voronoi diagram of a set of geometric objects is a partition of the surrounding space into cells, each of which comprises the points closer to one of the objects than to any others (see Figure 2a). An object's ε-*Voronoi cell* is the portion of its Voronoi cell lying within distance ε of the object (see Figure 2b).

To avoid intersections between polylines, Nabil Mustafa and his colleagues proposed a simplification algorithm that removes noncompliant line segments on the basis of the polyline's ε-Voronoi cell.[2] In their method, a polyline segment connects any two vertices of the polyline, whether adjacent or not. A compliant segment lies completely within the polyline's ε-Voronoi cell (see Figure 2b). So, compliant segments don't intersect other compliant segments. However, they might still cross themselves.

Our approach avoids intersections and self-intersections by computing ε-Voronoi cells for each monotone subpolyline rather than for the polylines themselves. Simplification involves five main steps.

First, we generate a set of shortcut segments for each monotone subpolyline.

Second, we render the ε-Voronoi cell for each monotone subpolyline in the graphics hardware's stencil buffer. Each monotone segment's Voronoi region gets a unique stencil value—essentially, a different color. Most computer graphics cards permit stencil values from 0 to 255. If there are more than 255 monotone subpolylines, we use simple heuristics to color the adjacent ε-Voronoi cells with different values. For each subpolyline, we scan surrounding pixels in the stencil buffer before rendering the ε-Voronoi cell and record these pixels' stencil values. Then, we set this subpolyline's stencil value to be different than that of any of those pixels.

Third, we draw each shortcut segment in the color buffer in a unique color with the stencil test enabled. A segment passes the test only if its stencil value doesn't equal the stencil value of the subpolyline's ε-Voronoi cell. Any segment lying completely in the Voronoi region fails the test. The result is that compliant segments don't appear in the color buffer.

Fourth, we scan the color buffer. If the scan detects a shortcut segment's color, that segment is noncompliant and is removed.

Finally, we connect the remaining compliant segments.

The depth and stencil buffers let us generate the Voronoi regions quickly, without complex geometric computation. But because the color buffer represents polylines as pixels, a noncompliant segment might be hidden by other segments. To solve this problem, we repeatedly clear the color buffer, rerender the segments, and eliminate noncompliant segments until the process uncovers no new noncompliant segments.

Generating fewer shortcut segments can reduce the iterations necessary to complete this process. We employ the Douglas-Peucker algorithm to select an optimal set of shortcut segments.[3] We first connect the original polyline's starting and ending vertices to get a segment. We then calculate the distances from the intermediate vertices to that segment and identify the vertex at the greatest distance. We connect the vertex having the maximum distance with the starting and ending vertices to generate two segments. If the maximum distance is less than ε, we add the segment to the segments set. We repeat this process until it generates no more segments.

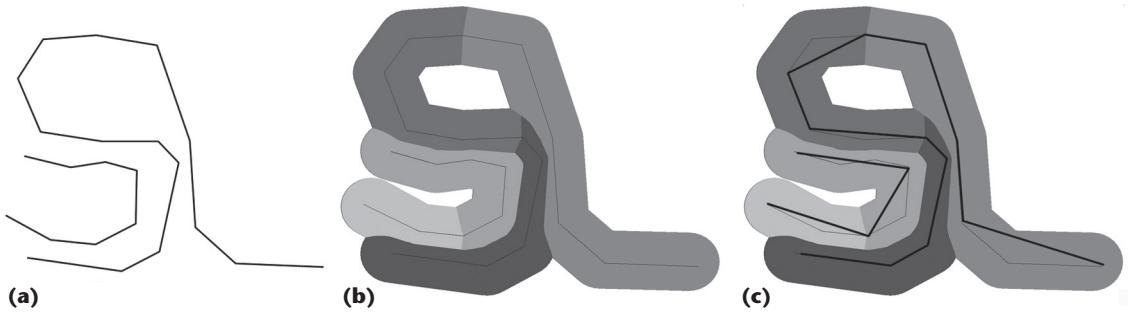


Figure 3. Simplifying polylines. (a) The original polylines. (b) ε-Voronoi diagrams of monotone subpolylines. (c) The simplified results (bold black lines). Each subpolyline's simplified result always lies in its ε-Voronoi cell. So, the simplified results won't cross with each other or cross themselves.

Because a monotone subpolyline's compliant segments lie completely in its ε-Voronoi cell, a subpolyline simplified in this way can't cross other simplified polylines. Because the simplified subpolylines are monotone, they'll never cross themselves. So, this method removes all the intersections and self-intersections. Figure 3 illustrates the main procedure.

Retaining Precision

Vector features drawn at large scales in the frame buffer might be represented by several pixels or only one pixel. To reduce inexactness and occlusions due to rasterization, we subdivide the vector map to guarantee that each line segment can be represented by at least two pixels.

We divide the map into at least n_W subregions in the width direction according to the function $n_W = \lceil W/(d \bullet w) \rceil$, where d is the average distance between two adjacent vertices of a polyline, w is the screen width, and W is the map width. The $\lceil\ \rceil$ function rounds a number up to an integer value, which ensures that different vertices in the map don't cover the same pixel in the frame buffer. Next, we divide the map into n_H subregions in the height direction. Loosely speaking, the simplification time increases as $O(n_W \bullet n_H)$.

Overlay on Multiresolution Terrain Models

We render the simplified vector features according to their LOD. At a relatively coarse LOD, we render the features using the shadow volume approach.[4] At the coarsest level—farthest from the viewpoint—we render the features using substitute technologies.

For a look at other research on rendering vector maps on multiresolution terrain models, see the related sidebar.

Accurate Overlay

The overlaying involves two steps: constructing shadow volumes and rendering the shadows.

A vector feature's shadow volume is a polyhedron

Related Work on Rendering Vector Maps onto Multiresolution Terrain Models

Methods for visualizing vector data over digital elevation models (DEMs) are either geometry based or texture based. To avoid cracks between vector maps and the associated level-of-detail (LOD) terrain models, geometry-based methods must reconstruct the maps dynamically as the viewpoint changes. Some methods rely on terrain-rendering algorithms, such as Zachary Wartell and his colleagues' algorithm for overlaying polylines on LOD terrain models.[1] Other algorithms generate static geometric primitives during preprocessing, but users can't manipulate the vector data at runtime.

Texture-based methods generate 2D textures from the vector data and then project them onto DEMs. However, when the texture resolution is less than the screen resolution, these methods produce blurry edge and, as the viewpoint moves toward terrain surfaces, serious aliasing artifacts. Increasing the texture resolution or using multiple texture maps with different resolutions can resolve the difficulties, but these approaches consume huge amounts of texture memory. Oliver Kersting and Jürgen Döllner used an off-screen buffer to generate textures dynamically,[2] but performance depends on the availability of advanced graphics hardware.

On the other hand, Martin Schneider and Reinhard Klein adopted the shadow volume algorithm,[3] which overcomes the limitations of conventional geometry-based and texture-based methods. It's independent of LOD terrain models and overlays the vector maps on the terrain precisely. However, rendering large, complicated vector maps in real time can be difficult.

References

1. Z. Wartell et al., "Rendering Vector Data over Global, Multiresolution 3D Terrain," *Proc. 2003 Symp. Data Visualization*, ACM Press, 2003, pp. 213–222.
2. O. Kersting and J. Döllner, "Interactive Visualization of Vector Data in GIS," *Proc. 10th ACM Int'l Symp. Advances in GIS*, ACM Press, 2002, pp. 107–112.
3. M. Schneider and R. Klein, "Efficient and Accurate Rendering of Vector Data on Virtual Landscapes," *J. WSCG*, vol. 15, nos. 1–3, 2007, pp. 59–65.

whose lateral faces are perpendicular to the horizontal plane. To construct a polygon's shadow volume, each side is extruded vertically to create its lateral faces. The top and bottom caps of the polyhedron are parallel to the horizontal plane; their heights are the maximum and minimum heights, respectively, of the DEM region covered by the polygon. To construct a polyline's shadow volume, we broaden it to a narrow strip and then construct lateral faces and caps the same way as for polygons.

To render distant polygon features, we use texture mapping; the long distance to the viewpoint alleviates the aliasing problems this technique causes.

To render the resulting polyhedron's shadows, we disable the color buffer and depth buffer write functions and enable the stencil buffer write and stencil test. First, we set the stencil test rule to increase the stencil value when the depth test fails and render the polyhedron's back faces. According to the depth test rule, if a pixel's depth is larger than that already stored in the depth buffer, the depth test fails. Because we measure the depth in terms of the distance to the viewpoint, and the terrain has been rendered in the depth buffer, the parts of polyhedron that are concealed by the terrain fail the depth test. Second, we alter the stencil test to decrease the stencil value when the depth test fails and render the front faces. Finally, we disable the depth test, enable the color buffer override (with the stencil test passing only if the stencil value doesn't equal 0), and render the back faces again with the specific color.

Polyline Overlays

In a large landscape, gaps between the polylines and the underlying terrain surface that are small and far from the viewpoint don't significantly affect the landscape's visual quality. So, we can overlay the far polylines using our coarse models. During preprocessing, we first interpolate new vertices into the simplified polylines and then convert the new and original vertices into 3D vertices using the terrain surface elevations. In this way, we generate 3D polylines.

To avoid *z-buffer fighting* (the generation of visual artifacts by two objects that are rendered close to each other), we enter the artifacts and the polylines into the terrain models' undersurfaces. We raise the 3D polylines' heights by a specified offset during rendering, with the offset increasing linearly with the distance from the polyline to the viewpoint. After rendering, few polylines intersect with the terrain models, and the gaps between the polylines and the terrain surface are inconspicuous.

Polygon Overlays

To render distant polygon features, we use texture mapping; the long distance to the viewpoint alleviates the aliasing problems this technique causes. First, we render vector features into a 2D texture by filling the simplified polygons with specific colors and making the other regions transparent. We then fuse the resulting texture with other textures of the terrain surface and project it onto the surface.

The overlay process should update the texture in real time according to the distance between the polygons and the viewpoint. If the viewpoint moves toward a polygon, we set the polygon's corresponding region in the texture to be transparent by setting its alpha value to 0. Then, we render the polygon on the terrain models using the shadow volume approach. If the viewpoint moves away from a polygon, we just set the corresponding region's texture of terrain models to be opaque.

LOD Selection

The distance between a spatial object and the viewpoint serves as the criterion for the LOD. Because of vector features' irregular shapes and variable lengths (for polylines) or areas (for polygons), computing the distance and constructing continuous LOD models are difficult. Moreover, computing the distance to the viewpoint for all vertices is time-consuming. So, we compute the LOD level value for each monotone subpolyline generated during simplification instead of for each feature.

To accelerate LOD selection, we use a block-indexing algorithm. We compute a map's minimum bounding rectangle and the number of subpolylines n_{subs}. Given a value n_{avg}, we split the rectangle into $\lceil n_{subs}/n_{avg} \rceil$ regular blocks, implying that on average, every block contains n_{avg} subpolylines. For each block, we compute and store the subpolylines that are fully or partly contained in it. Because the vector features are unevenly distributed, some blocks might not contain subpolylines; we remove those blocks.

As the viewpoint changes, we calculate the distance d between it and the center of a block. Given distance thresholds f_1 and f_2, where $f_1 > f_2$, there are four possible LODs:

- invisible (the block is culled and invisible),

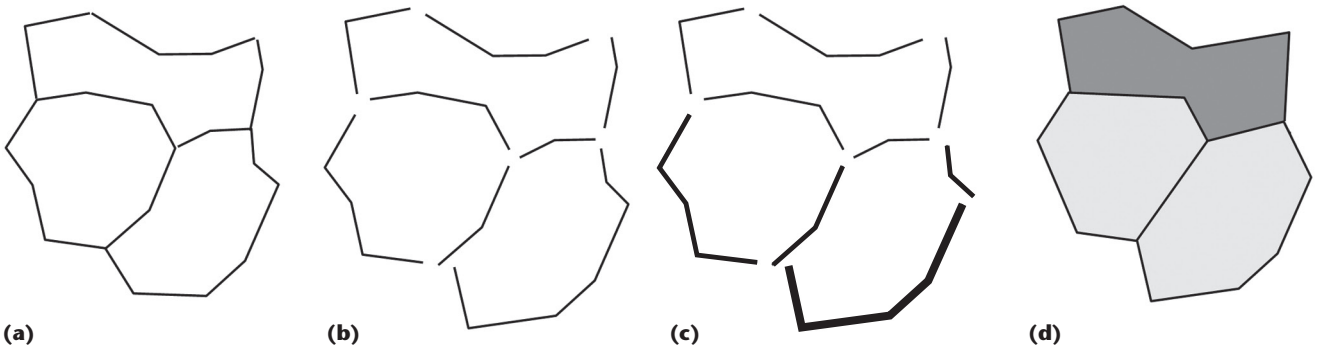


Figure 4. Constructing level-of-detail (LOD) polygon models. (a) The original polygons. (b) Splitting the polygons into arcs. (c) Splitting the arcs into monotone subpolylines. The boldest subpolylines are at the full LOD, the bolder ones are at the medium LOD, and the finest ones are at the coarse LOD. (d) Reconstructing the polygons. We render the light gray polygons using the shadow volume algorithm and the dark gray one using texture mapping.

- coarse ($d > f_1$.),
- medium ($f_2 < d < f_1$), and
- full ($d < f_2$).

If a subpolyline is contained by several blocks with different LOD values, we choose the highest level of all the blocks as the subpolyline's level.

Vector Map Construction

At this point, the polylines are represented by their monotone subpolylines. If a subpolyline's LOD is full, we overlay it on the terrain model using the shadow volume algorithm. If the LOD is medium, we overlay the simplified subpolyline instead, again using the shadow volume algorithm. If the LOD is coarse, we render the simplified subpolyline with the 3D polylines.

Constructing LOD polygon models is a bit more complicated: if one polygon rendered at the full LOD is adjacent to one rendered at the coarse LOD using the simplified polygons, the two polygons' boundaries will no longer match. We avoid this problem by splitting the boundaries into arcs (see Figure 4b) according to their adjacent topological relations. (We use Esri's ArcMap to build topological relations before the simplification.) We then simplify these arcs instead of the original polygon borders by splitting them into monotone subpolylines (see Figure 4c), which we simplify separately in turn. When rendering, we first get each subpolyline's LOD, then decide how to render each polygon.

A polygon has three options regarding LOD:

- One or more of its subpolylines are at the medium or full LOD.
- None of the subpolylines is at the medium or full LOD, and one or more are at the coarse LOD.
- The whole polygon is at the invisible LOD.

In the first case, we construct the polygon's new border by using the simplified or original subpolylines—the original when the subpolyline is at the full LOD and its simplified version otherwise (see Figure 4d). Then, we render the reconstructed polygon using the shadow volume algorithm and set its corresponding region in the texture to be transparent. In the second case, we render the polygon using texture mapping and set its corresponding region to be opaque. In the third case, we simply ignore the polygon.

Results and Discussion

We implemented our algorithms using Visual C++ and OpenGL and successfully tested them on different real-world terrain vector map datasets. For the tests, we used a PC with a 2.4-GHz Intel Core 2 CPU, 2 Gbytes of RAM, and an ATI Radeon HD 4650 graphics card.

Map Simplification

We used four types of vector maps: land use/land cover (LULC), contour, stream, and soil (see Table 1). Figures 5, 6, and 7 illustrate the first three original maps and their simplified versions. Table 2 presents the total simplification time and the

Table 1. Tested vector datasets.

Dataset type	No. of polylines or polygons	No. of vertices	Source
Land use/land cover (LULC) map	1,197	90,145	State of Hawaii, www.state.hi.us/dbedt/gis/lulc.htm
500-ft. contour map	305	93,187	State of Hawaii, http://hawaii.gov/dbedt/gis/cntrs500.htm
Stream map	2,651	114,023	State of Hawaii, http://hawaii.gov/dbedt/gis/huntareas.htm
Soil map	5,548	987,204	State of Hawaii, http://hawaii.gov/dbedt/gis/soils.htm

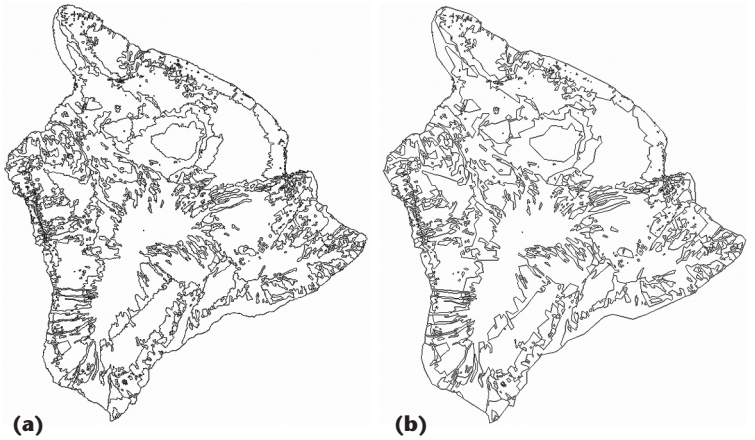


Figure 5. Simplifying a land use/land cover (LULC) map. (a) The original map has 90,145 vertices. (b) The simplified map has 9,530 vertices. ε = 5‰, where ε is the tolerance parameter for the Zhao-Saalfeld[5] method, and 5‰ means ε is set as 5‰ of the original map's width.

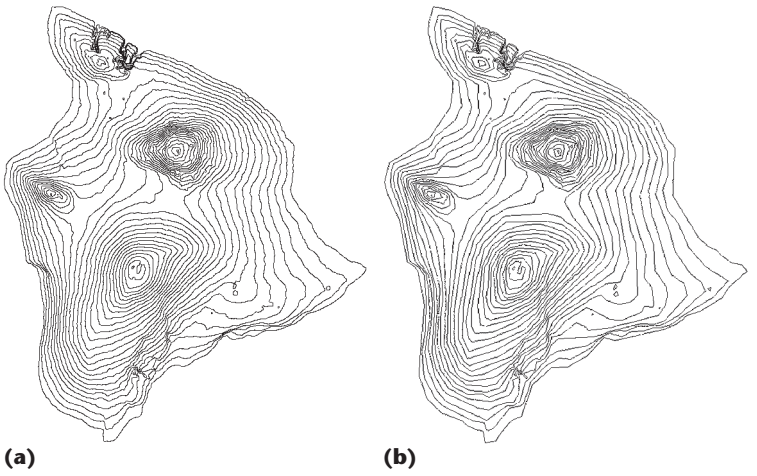


Figure 6. Simplifying a contour map. (a) The original map has 93,187 vertices. (b) The simplified map has 1,831 vertices. ε = 5‰.

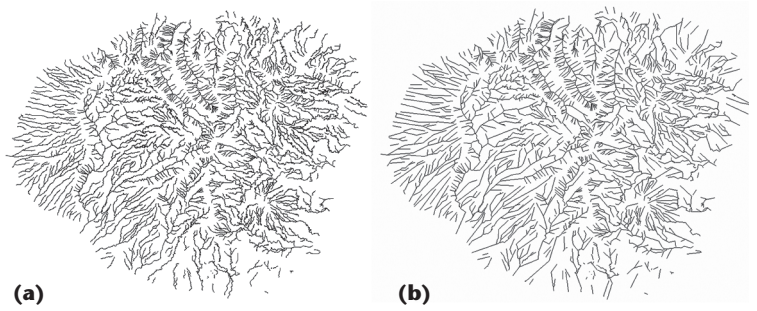


Figure 7. Simplifying a stream map. (a) The original map has 114,023 vertices. (b) The simplified map has 5,720 vertices. ε = 5‰.

number of remaining vertices. Table 3 describes the time spent on each step.

We also compared our method to Mustafa and his colleagues' method. Table 4 shows that these methods require roughly the same time to get the same compression ratios. Our method generates fewer segments and thus spends less time detecting the noncompliant segments. On the other hand, it computes ε-Voronoi cells for the monotone subpolylines instead of for the original polylines. Our approach's main advantage is that it can avoid both intersections and self-intersections, whereas Mustafa and his colleagues' method might introduce self-intersections.

Figure 8 shows the simplifications from both methods; using ArcMap to check the simplified maps' topological relationships, we found no topological errors. With almost the same compression ratios as Mustafa and his colleagues' method, our method can keep the exact topological consistency.

To the best of our knowledge, few researchers have addressed map simplification using graphics hardware. So, we also compared our method with Bisheng Yang and his colleagues' geometry-based method,[6] which can likewise prevent intersections and self-intersections. To reduce computing time, Yang and his colleagues removed small features before simplification. In contrast, our method keeps all the features.

In the experiment, we used Yang and his colleagues' method without removing small features first (see Table 4). When the data volume is small, our method offers no obvious advantages because clearing and reading the frame buffer takes a certain amount of time. However, it's more efficient and faster with large datasets because it doesn't require complex geometric computing processes.

Overlaying Vector Maps on 3D Terrain Models

Using the datasets described in Table 1, we constructed four LOD map models and overlaid them on a Hawaiian terrain model containing $1{,}025^2$ height points. We rendered the contour map (ε = 5‰) and stream map (ε = 0.5‰) on the LOD terrain's surface as visualized with Samuel Atlan and Michael Garland's algorithm.[7] For the LULC map (ε = 5‰) and soil map (ε = 0.5‰), we rendered the terrain with Stefan Röttger's algorithm.[8]

We also visualized these four maps using Martin Schneider and Reinhard Klein's method.[4] Figures 9 and 10 show the results; we don't see aliasing artifacts or gaps. Figure 11 demonstrates that our method achieves higher frame rates than Schneider and Klein's method does, especially when the viewpoint is far from the terrain surface. Furthermore, our overlaying efficiency is almost unaffected by the complexity of the underlying terrain datasets and the terrain LOD algorithms.

Future research will explore issues such as adaptation of an out-of-core algorithm to efficiently

Table 2. The simplification time and remaining vertices in our method.

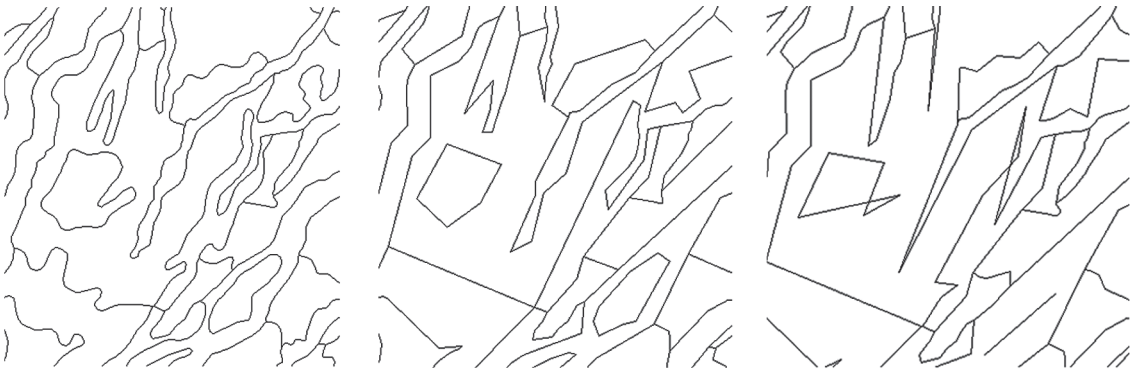
Map	LULC		Contour		Stream		Soil	
Original no. of vertices	90,145		93,187		114,023		987,204	
Distance tolerance ε (‰)	5.0	0.5	5.0	0.5	5.0	0.5	5.0	0.5
Simplification time (sec.)	4.2	3.2	3.5	2.5	4.5	3.0	28.7	21.6
No. of remaining vertices	9,530	23,680	1,831	11,961	5,720	11,749	70,643	165,936
Compression ratio (%)	10.6	26.3	2.0	12.8	5.0	10.3	7.2	16.8

Table 3. The time cost for each step in our method (sec.).

	Map							
	LULC		Contour		Stream		Soil	
Step	ε = 5.0‰	ε = 0.5‰	ε = 5.0‰	ε = 0.5‰	ε = 5.0‰	ε = 0.5‰	ε = 5.0‰	ε = 0.5‰
1. Get monotone subpolylines	0.1	0.1	0.1	0.1	0.1	0.1	1.2	0.8
2. Generate shortcut segments	0.4	0.4	0.6	0.3	0.8	0.5	3.9	3.0
3. Render ε-Voronoi cells	2.1	1.2	1.4	0.9	1.4	0.7	8.1	5.5
4. Remove noncompliant segments	1.5	1.0	1.3	1.1	2.0	1.6	13.4	10.5
5. Get final simplifications	0.2	0.1	0.1	0.1	0.3	0.2	2.1	1.6
Total time	4.2	3.2	3.5	2.5	4.5	3.0	28.7	21.6

Table 4. Comparing three methods for simplifying vector maps.

	Method					
	Ours		Bisheng Yang and his colleagues[6]		Nabil Mustafa and his colleagues[2]	
Map	No. of remaining vertices	Simplification time (sec.)	No. of remaining vertices	Simplification time (sec.)	No. of remaining vertices	Simplification time (sec.)
LULC	9,530	4.2	9,644	2.7	9,294	3.7
Contour	1,831	3.5	1,806	3.4	1,862	2.7
Stream	5,720	4.5	5,693	3.2	5,835	3.2
Soil	70.630	28.7	70,648	45.1	70,461	27.1



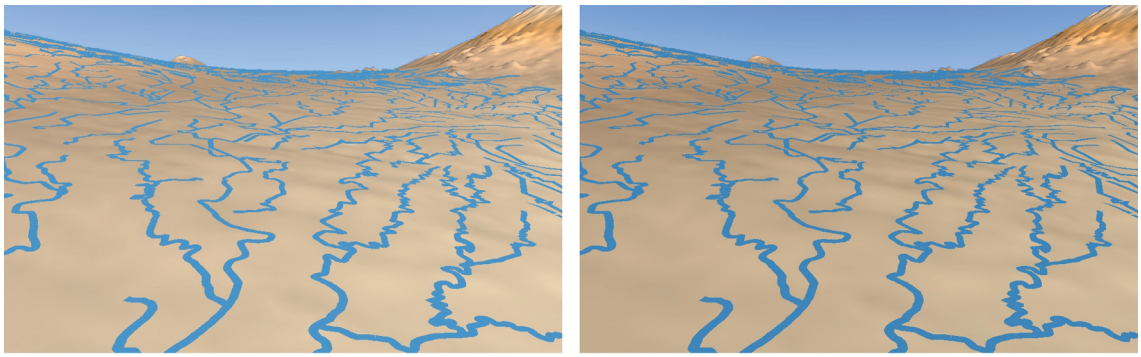
(a) **(b)** **(c)**

Figure 8. Simplifying a soil map. (a) A small region in the original map, which has 987,204 vertices. (b) The same region in our simplified map, which has 70,630 vertices. (c) The same region in the simplified map generated by Nabil Mustafa and his colleagues' method,[2] which has 70,461 vertices. The proposed method keeps the exact topological consistency, with almost the same compression ratios as Mustafa and his colleagues' method.

implement large vector map simplification, and fast transmission of large-scale vector maps and DEM datasets in networked environments.

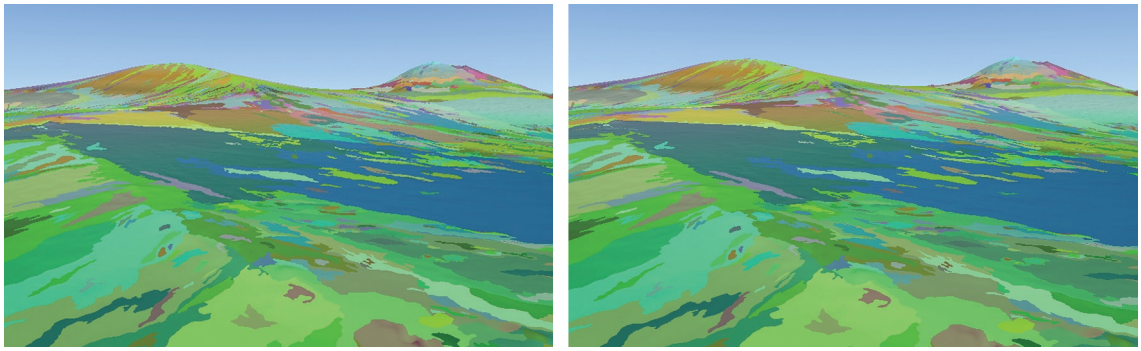
Acknowledgments

This research was supported by the National Natural Science Foundation of China (grants 60736007



(a) (b)

Figure 9. Visualizing a stream map. (a) A view-dependent LOD stream map overlay (ε = 0.5‰, f_1 = 14,000 m, and f_2 = 7,000 m, where f_1 is the distance threshold between the coarse level and medium level and f_2 is the threshold between the medium level and full level). (b) The original contour map overlay implemented by Martin Schneider and Reinhard Klein's method.[4] Our method renders the faraway polylines with fewer details. Compared with Schneider and Klein's method, ours doesn't generate violent changes or perceivable cracks.



(a) (b)

Figure 10. Visualizing a soil map. (a) A view-dependent LOD soil map model overlay (ε = 0.5‰, f_1 = 9,000 m, and f_2 = 4,500 m). (b) The original soil map overlay implemented by Schneider and Klein's method. Our method renders the faraway polygons by texture mapping but doesn't generate aliasing.

and 60972128), Program for New Century Excellent Talents in University (grant NCET-07-0099), 973 Program (grant 2007CB714403), and Open Project Program of the State Key Lab of Computer-Assisted Design and Computer Graphics (grant A0901), Zhejiang University. We thank our editor and reviewers for their kind suggestions.

References

1. V. Chandru, V.T. Rajan, and R. Swaminathan, "Monotone Pieces of Chains," *ORSA J. Computing*, vol. 4, no. 4, 1992, pp. 439–446.
2. N. Mustafa et al., "Dynamic Simplification and Visualization of Large Maps," *Int'l J. Geographical Information Science*, vol. 20, no. 3, 2006, pp. 273–320.
3. D.H. Douglas and T.K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature," *Canadian Cartographer*, vol. 10, no. 2, 1973, pp. 112–122.
4. M. Schneider and R. Klein, "Efficient and Accurate Rendering of Vector Data on Virtual Landscapes," *J. WSCG*, vol. 15, nos. 1–3, 2007, pp. 59–65.
5. Z. Zhao and A. Saalfeld, "Linear-Time Sleeve-Fitting Polyline Simplification Algorithms," *Proc. Auto-Carto XIII*, pp. 214–223.
6. B.S. Yang, R. Purves, and R. Weibel, "Efficient Transmission of Vector Data over the Internet," *Int'l J. Geographical Information Science*, vol. 21, no. 2, 2007, pp. 215–237.
7. S. Atlan and M. Garland, "Interactive Multiresolution Editing and Display of Large Terrains," *Computer Graphics Forum*, vol. 25, no. 2, 2006, pp. 211–224.
8. S. Röttger et al., "Real-Time Generation of Continuous Levels of Detail for Height Fields," *Proc. 6th Int'l Conf. in Central Europe on Computer Graphics and Visualization* (WSCG '98), IEEE Computer Society Press, 1998, pp. 315–322.

Ling Yang *is a master's student in geographic information systems at Beijing Normal University. Her research interests include map generalization and geographic information systems. Yang has a BA in geoinformatics from Beijing Normal University. Contact her at yangling0531@126.com.*

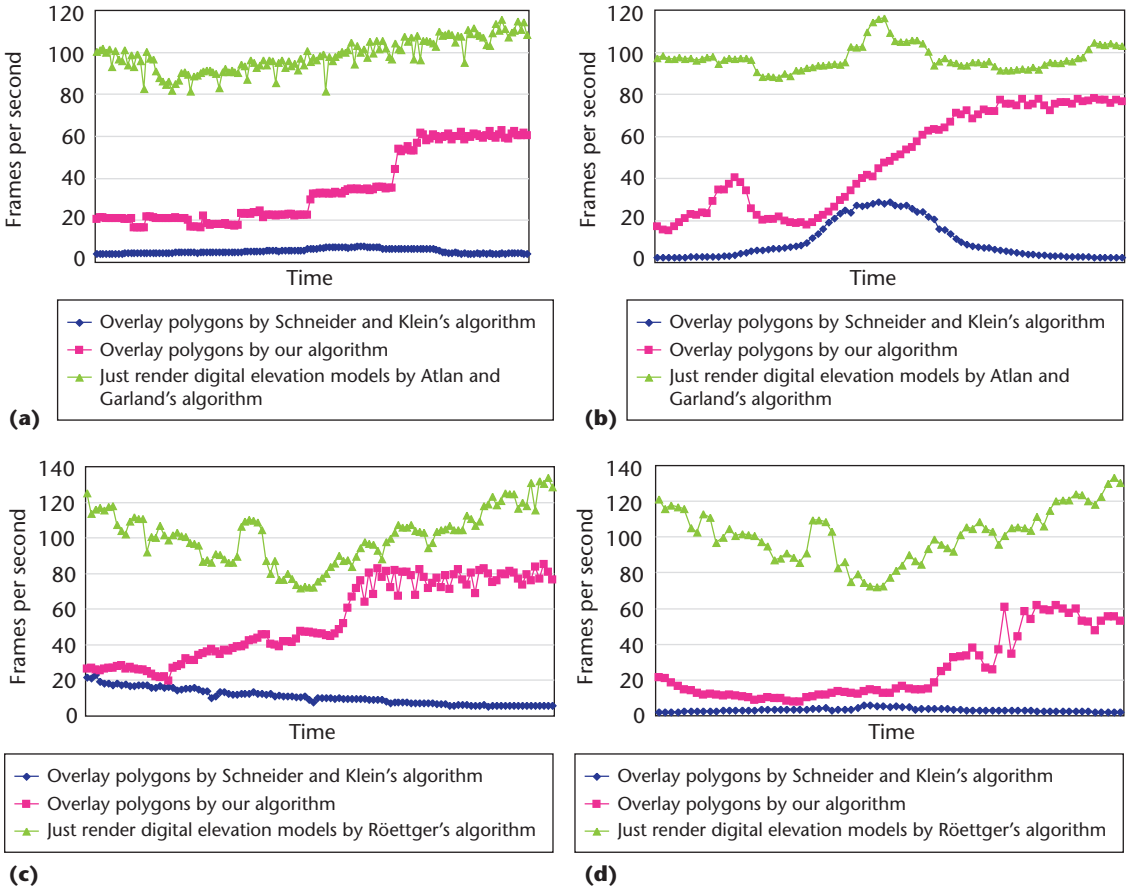


Figure 11. Frame rate curves from rendering the Hawaiian terrain and vector maps. (a) The contour map. (b) The stream map. (c) The LULC map. (d) The soil map. The frame rate fluctuates with the viewpoint moving to the terrain's surface and then to the distance. Our method achieves higher frame rates and is almost unaffected by the terrain LOD algorithms.

Liqiang Zhang *is a professor of image processing and geographic information systems at Beijing Normal University. His research interests include 3D geographic information systems and remote-sensing image processing. Zhang has a PhD in geoinformatics from the Chinese Academy of Science's Institute of Remote Sensing Applications. He's the corresponding author of this article. Contact him at zihaozhang2003@yahoo.com.cn.*

Jingtao Ma *is an intelligent-transportation-systems product manager at PTV America. His research interests include urban transportation and intelligent transportation systems. Ma has a PhD in transportation engineering from University of California, Davis. Contact him at jtma@ptvamerica.com.*

Zhizhong Kang *is an associate professor of remote sensing and lidar image processing at the China University of Geosciences. His research interests include real-time rendering and lidar data processing. Kang has a PhD in digital photogrammetry from Wuhan University. Contact him at dr_zzkang@yahoo.com.cn.*

Lixin Zhang *is a professor of remote-sensing image processing at Beijing Normal University and the vice director of the State Key Laboratory of Remote Sensing Science. His research interests include remote-sensing image processing. Zhang has a PhD in remote sensing from the Chinese Academy of Science's Cold and Arid Regions Environmental and Engineering Research Institute. Contact him at lxzhang@bnu.edu.cn.*

Jonathan Li *is a professor in the University of Waterloo's Department of Geography and Environmental Management. His research ranges from remote sensing to distributed geospatial information services. Li has a PhD from the University of Cape Town. Contact him at junli@uwaterloo.ca.*