Quadstream: Multiscale Polygon Generalization, Persistence and Presentation

Curran Kelleher

Abstract

We present a novel set of algorithms for polygon generalization (i.e. line simplification), persistence, visualization, and interaction. We discuss the design and implementation of an interactive multi-scale web-based system build around these algorithms. Map generalization, multiscale geographic databases, and multiscale data structures for efficient visualization and interaction have all been well studied. We examine how these areas can be practically integrated to create a concise web-based map tool which supports smooth panning, zooming, and polygon selection. Our purpose for this tool is to provide an interactive choropleth map component for a web-based visual analytics system.

1 Related Work

1.1 Polygon Generalization

Map generalization is a well studied problem for which many algorithms exist. Map generalization can have many meanings depending on which generalization operations are performed, such as aggregation, classification, exaggeration, smoothing, omission, symbolization, simplification, collapsing, etc. Our needs are oriented toward building an interactive multi-scale choropleth map. This application demands a generalization algorithm which does not aggregate polygons and does not add or modify points (to avoid redundancy across scales). Therefore we are interested specifically in algorithms which achieve generalization through omission of polygon vertices. Algorithms which perform this operation are termed "point-reduction" or "line

simplification" generalization algorithms [11]. We have the additional requirements that the resulting generalization is multiscale, maps well onto conventional databases, is suitable for progressive data transfer, and maps well onto client side data structures for efficient visualization, navigation, and selection.

McMaster has produced an excellent review of existing line generalization algorithms as well as a classification scheme for them [12]. McMaster defined five categories of line simplification algorithms [13].

The performance of many line simplification algorithms have been compared in the survey by Shi and Cheuhg [15].

Perhaps the mose widely used point-reduction polygon generalization algorithm is the Douglas-Peuker algorithm [6], also known as the Ramer or Fornsen algorithm [11]. Though for many this algorithm is ideal, it has many issues. For example, it often produces self-intersections, cross-intersections, distorted shapes, and jagged edges. Over the years, many researchers have improved the Douglas-Peuker algorithm to correct for some of these problems. For example, Visvalingam and Whyatt developed an algorithm to correct for the heavy shape distortion produced by the Douglas-Peuker algorithm [20]. The asymptotic running time of the original Douglas-Peuker algorithm is $O(n^2)$. Hershberger and Snoeyink [7] showed that using path hulls, the running time of the Douglas-Peuker algorithm can be reduced to $O(n \log_2 n)$.

Most existing line simplification algorithms are not explicitly tied to scale. In other words, they do not guarantee that the density of vertices is below a specified limit. Töpfer and Pillewizer formulated the *principle of selection*, also known as the *radical law* or *law of natural dimension* [17] [8]. This law states that the number of features presented on a map is proportional to the square root of its scale.

The law of natural dimension is very much related to the *natural principle of objective generalization* put forth by Li and Openshaw [10], which states that generalization algorithms should ideally be based on the nature of how objects are perceived at different scales. The human eye cannot perceive distinct features smaller than a certain size, termed the smallest visible size (SVS) by Li and Openshaw, who argue that features smaller than that size can be safely omitted from presentation. Li and Openshaw have formulated a line generalization algorithm based upon this principle, known as the Li-Openshaw Algorithm [9].

The Li-Openshaw algorithm has several modes; vector, raster, and raster-

vector. The raster-vector mode of the Li-Openshaw algorithm involves overlaying a raster grid on polygons and generating a generalization by choosing a single vertex per occupied grid cell to store for presentation, discarding the rest. For each occupied raster cell, the midpoint between the two points where the polyline enters and exits the cell is chosen to represent the cell. Other points could alternatively be chosen, such as the first point inside the cell. Of all existing line generalization algorithm, the Li-Openshaw algorithm most closely resembles our Quadstream algorithm.

1.2 Multiscale Data Structures

Quadtree

R-Tree R-Tree (Guttman 1984)

BLG-Tree BLG Tree [19] ? correct citation ?.

Multi-scale data structures [16].

Multi-scale line tree (Jones and abraham 1987)

1.3 Multiscale Databases for Line Generalization

The problem of designing a multi-scale database for generalized maps has been well studied. Zhou and Jones [21].

Applying reactive data structures in an interactive multi-scale GIS [18]. Reactive-tree (van Oosterom 1991)

Strip tree [3].

Arc tree (Guenther 1988)

1.4 Web-based Systems for Line Generalization

PROGRESSIVE TRANSMISSION OF VECTOR MAP ON THE WEB [1].

Progressive Transmission of Vector Data Based on Changes Accumulation Model [2].

Progressive transmission of vector map data over the world wide web [4].

Block and Harrower created a web service for map generalization with a Flash-based client called *MapShaper* [5]. MapShaper provides a system which allows uploading of shape data, server-side computation of generalization using several common line generalization algorithms, and very responsive multi-resolution rendering on the client side. Their server side implements the Douglas-Peuker, Visvalingam Whyatt, and their own refinement of the

Visvalingam Whyatt algorithm in C++. The multi-resolution drawing in the client is implemented using simple array iteration, only drawing those vertices whose importance value is above a given threshold. This approach requires that all levels of detail must always be transferred to the client.

2 The Quadstream System

In the raster-vector mode of the Li-Openshaw algorithm, the generalization process is done in raster mode and the result is stored and presented in vector mode. The Quadstream algorithm is essentially an extention of this algorithm through scale space using the multiscale raster structure of a quadtree in place of a single-scale raster grid. The quadtree structure also informs vertex partitioning across a database, and thus forms the basis for query generation as well.

2.1 Generalization

(choosing the first vertex in each raster cell)

In fact Li [8](p. 72) suggests that "...a hybrid data structure of vector and raster should be used as part of a strategy for multi-scale spatial representation. The vector structure could be used to hold spatial data in a database since a vector is a feature-primary data structure and is good for organizing data efficiently. The raster structure could be used as a working environment." The Quadstream system does exactly this, using a quadtree as the multiscale raster working environment for the vector polygon vertices stored.

Li-Openshaw algorithm, as pointed out by Weibel (1996), "by virtue of its raster structure, implicitly (but not explicitly) avoids self-overlaps." So too does the Quadstream algorithm.

3 Our System

Our system can be conceptually divided into the following components:

- Input: The original shape data is read from disk.
- Topological aggregation (TODO check term): duplicate vertices are aggregated together, retaining polygon membership information.

- Critical vertex detection: critical (gap-closing) vertices are detected.
- The Quadstream Algorithm: The original shape data is transformed into many small vertex collections amenable to storage, retrieval, and progressive refinement of a map generalization.
- Persistence: Bounding boxes, critical vertices, and the quadstream results are stored in a database or file system.
- Retrieval: A query from the client to the server requests those vertices which create a suitable generalization for a given view (zoom level and location).
- Reconstruction: The returned vertices are incrementally stored in multi-scale data structures suitable for efficient selection and visualization, namely an R-Tree of BLG trees.
- Visualization: The vertices needed to visualize the polygons visible in the current view at a sufficient level of detail are efficiently extracted from the client-side data structures and rendered.
- Selection: The user makes a selection of polygons using a rectangle or lassoo selection tool, which efficiently derives which polygons are selected by querying the client-side data structure.

In practice, some of these steps can be combined. The input, topological aggregation (TODO correct term?) and critical vertex detection steps can all occur at once. The Quadstream and persistence steps can occur at the same time. Reconstruction, retrieval, and visualization all can occur in a progressive data transfer operation. Selection and visualization happen at the same time when brushing.

3.1 Input

When reading a set of geometries, the original vertices are transformed into *qualified vertices* and stored in a hash table backed set. A qualified vertex is comprised of an (x, y) coordinate pair, and a list of one or more polygon memberships represented by a (p, v) pair where p is the polygon id and v is the vertex id. When a duplicate vertex is detected, the polygon membership

of the new verted is appended to the membership list of the existing qualified vertex.

Critical vertices are those qualified vertices whose degree (count of adjacent vertices) is 3 or more. This measure detects those vertices which close gaps between two or more polygons. These are also detected while the data is being read and stored for later processing.

3.2 The Quadstream Algorithm

The *Quadstream Algorithm* is our novel line generalization algorithm, so termed due to its close relationship to the quadtree [14], and it's application in streaming (progressively transferring) maps from server to client. The algorithm is equivalent to a depth-limited depth-first search through a quadtree of bucket capacity 1 containing all unique vertices. However, a quadtree data structure need not be maintained. A set of occupued buckets can be used instead. To accomplish this, we use a list of sets of integers, one set per quadtree level, the integers being occupied bucket identifiers.

Because the Quadstream algorithm generates generalizations containing at most one vertex per bin, and the upper bound of the number of bins for a given scale is proportional to the square root of the scale (TODO check this),Töpfer's law of natural dimension as well as Li and Openshaw's natural principle of objective generalization are both implicitly guaranteed. Also, as is the case with the Li-Openshaw algorithm, the Quadstream algorithm implicitly avoids topological errors (TODO find that good quotation from 'first forty years' article).

In the following pseudocode, C (Critical) represents the set of critical vertices, P (Pool) represents the set of vertices which are not critical vertices, and $lmax$ (level maximum) represents the pseudoquadtree level at which the algorithm terminates. The variable O (Occupied spots) represents the afformentioned list of sets of occupied bucket identifiers, or "spots". The variable LV (level vertices) is a global variable which contains the qualified vertices for the current level. The variable l (level) is a global variable which represents the current level.

```
QUADSTREAM()
 1  l = 0
 2  LV = empty list
 3  O[0..lmax] = empty set
 4  for each vertex v ∈ C
 5       STREAM(v)
 6  OUTPUTLEVEL()
 7  while l < lmax
 8       for each vertex v ∈ P
 9            if SPOT(v, l) ∉ O[l]
10                 STREAM(v)
11       l = l + 1
12       OUTPUTLEVEL()
```

```
STREAM(v)
1  insert v into list LV
2  for i = l to lmax
3       add SPOT(v, i) to set O[i]
```

```
OUTPUTLEVEL(v)
1  P = P \ LV
2  if l − lo ≥ 0
3       persist LV into (l − lo) depth grid (TODO clarify this)
4       clear LV
```

4 Future Work

The Quadstream and Li-Openshaw algorithms, though they are scale-based, allow neighboring vertices which fall into adjacent raster cells to be arbitrarily close together, thus their line segment may be smaller than the smallest visible object. According to the natural principle, we seek to disallow this condition. A vertex inclusion filter based on length, similar to the length-based simplification strategy discussed in [8](p.96), would alleviate this problem. In the future we plan to integrate such a filter into the Quadstream algorithm.

References

[1] Ai Bo A, Ai Tinghua B, and Tang Xinming C. Progressive transmission of vector map on the web.

[2] Tinghua Ai, Zhilin Li, and Yaolin Liu. Progressive transmission of vector data based on changes accumulation model.

[3] D.H. Ballard. Strip trees: A hierarchical representation for curves. 1981.

[4] M. Bertolotto and M.J. Egenhofer. Progressive transmission of vector map data over the world wide web. *GeoInformatica*, 5(4):345–373, 2001.

[5] M. Bloch and M. Harrower. MapShaper. org: A map generalization web service. In *Proc. of Autocarto*, pages 26–28. Citeseer, 2006.

[6] 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. *The Canadian Cartographer*, 1973.

[7] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proc. 5th Intl. Symp. on Spatial Data Handling*, pages 134–143, 1992.

[8] Z. Li. *Algorithmic foundation of multi-scale spatial representation*. CRC, 2006.

[9] Z. Li and S. Openshaw. Algorithms for automated line generalization based on a natural principle of objective generalization. *International Journal of Geographical Information Science*, 6(5):373–389, 1992.

[10] Z. Li and S. Openshaw. A natural principle for the objective generalization of digital maps. *Cartography and Geographic Information Science*, 20(1):19–29, 1993.

[11] Zhilin Li. Digital map generalization at the age of enlightenment: a review of the first forty years. *The Cartographic Journal*, 2007.

[12] R.B. McMaster. Automated line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 24(2):74–111, 1987.

[13] R.B. McMaster. The geometric properties of numerical generalization. *Geographical Analysis*, 19(4):330–346, 1987.

[14] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[15] Wenzhong Shi and Chuikwan Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 2006.

[16] Sabine Timpf and Andrew U. Frank. A multi-scale data structure for cartographic objects. In *17th International Cartographic Conference ICC'95, at*, pages 1381–1386, 1995.

[17] F. Topfer and W. Pillewizer. The principles of selection. *Cartographic Journal, The*, 3(1):10–16, 1966.

[18] P. van Oosterom and V. Schenkelaars. Applying reactive data structures in an interactive multi-scale GIS. *METHODS FOR THE GENERALIZATION OF GEO. DATABASES*, page 37.

[19] P. van Oosterom and J. van den Bos. An object-oriented approach to the design of geographic information systems. In *Proceedings of the first symposium on Design and implementation of large spatial databases*, pages 257–269. Springer-Verlag London, UK, 1989.

[20] M Visvalingam and J D Whyatt. Line generalization by repeated elimination of points. *The Cartographic Journal*, 1993.

[21] Sheng Zhou and Christopher B. Jones. Multi-scale spatial database and map generalisation. In *4th ICA Workshop on Progress in Automated Map Generalizatio*, 2001.