Algorithms for On-the-Fly Generalization of Point Data Using Quadtrees

Pia Bereuter and **Robert Weibel**

ABSTRACT: With a focus on mobile and web mapping, we propose several algorithms for on-the-fly generalization of point data, such as points of interest (POIs) or large point collections. In order to achieve real-time performance we use a quadtree. With their hierarchical structure and progressive levels of detail, indexes of the quadtree family lend themselves as auxiliary data structures to support algorithms for generalization operations, including selection, simplification, aggregation, and displacement of point data. The spatial index can further be used to generate several measures on the relative point position and proximity of points, neighborhood, clusters, local and global density, etc. Using the tiles of the tree structure, such measures can be efficiently generated and updated. These measures can then serve to make educated guesses on the density and proximity of points across a series of map scales, and thus enable to control the operation of the generalization algorithms. An implementation of the proposed algorithms has shown that thanks to the quadtree index, real-time performance can be achieved even for large point sets. Furthermore, the quadtree data structure can be extended into a storage structure, which can be used to store pre-computed generalizations; thus, a desired level of detail can simply be retrieved from storage.

KEYWORDS: mobile and web mapping, on-the-fly generalization, point data generalization, POI, spatial index, auxiliary data structure, quadtree

Introduction

In comparison to map generalization for paper maps, map generalization for mobile mapping and interactive web mapping has a different set of requirements, in particular its need for on-the-fly generalization and adaptation to user interaction and content. Web or mobile mapping applications typically display some thematic foreground data — predominantly in the form of point data such as points of interest (POIs) or large point collections — against the spatial reference provided by some background data (e.g. a topographic map or orthoimagery). The background data is usually assumed to be static in content and can thus be rendered by a pre-generalized tile service (OSM-based map services, Google Maps, Bing Maps) to provide seamless map interaction. The foreground data, on the other hand, will vary in content depending on the user's request. Thus, cartographic generalization of foreground data has to be achieved in real-time, requiring flexible, on-the-fly generalization algorithms (Weibel and Burghardt, 2008). However, so far only few algorithms for on-the-fly generalization of point data have been proposed in the literature. As a quick survey of online mapping applications will easily show, most of

these applications simply render the foreground point data in an ungeneralized way, potentially resulting in massive overlaps and clutter, depending on the density and distribution of the point data. Such applications are usually implemented as mash-ups on the basis of map services such as the ones mentioned above. Of course, since the map service is interactive, the user has the option to zoom in further until spatial conflicts among the foreground data disappear. On the other hand, the overview of the overall spatial distribution will then disappear.

Thus, we believe that on-the-fly generalization of large point data sets is necessary in web and mobile map services. Such generalization functions should be sufficiently fast to achieve real-time performance, and they should be flexible in order to adapt to varying contexts and user requirements.

In this paper, we propose several algorithms that use a quadtree index as an auxiliary data structure. Using this index structure allows achieving real-time performance for large point sets. The quadtree also supports the computation of various measures such as local point density, and allows to implement different generalization operators, including selection, simplification, aggregation, and point feature displacement. Using these different generalization operators informed by measures computed from the quadtree, mobile mapping applications for point data can achieve flexibility and adaptation.

Background and Related Work

Generalization Operators

In classic map generalization different generalization operators are applied to reduce spatial conflicts (McMaster and Shea, 1992). Due to the fact that in most maps geographic space is not distorted, object-directed approaches are mostly used (Bereuter and Weibel, 2010), where map objects not map space are modified. Space-directed generalization operators, where map space is distorted to improve spatial conflicts and improve map legibility (Bereuter and Weibel, 2010), are also possible in on-the-fly point generalization, but are not the focus of this paper. In the case of point features four object-directed generalization operators (Figure 1) are of most interest: *selection, simplification* and *aggregation* reduce the number of features represented on the map, while and *displacement* moves the map features away from each other to remove overlaps and congestion.

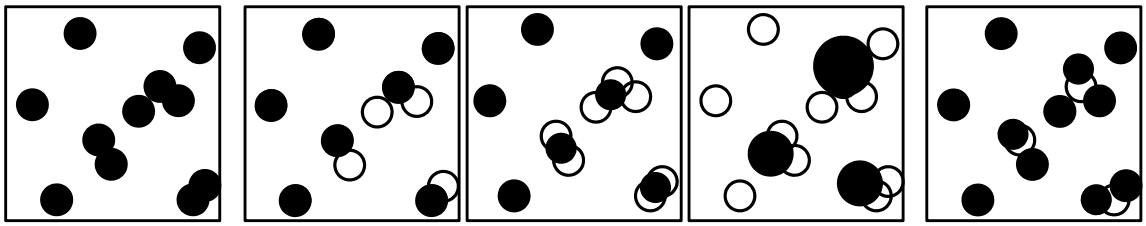


Figure 1: Overview of point generalization operators; filled points denote displayed points, circles denote removed or displaced points.

Each of the above generalization operators has different requirements. However, for all operators the detection of spatial conflicts is of crucial importance. For selection and aggregation, hierarchical ordering plays a key role, while displacement necessitates region queries and neighborhood queries. A hierarchical data structure like the quadtree may help to detect spatial conflicts, as it has inherent spatial hierarchical ordering, and supports region and neighborhood queries.

Note that *generalization operators*, which define a particular generalization process conceptually (e.g. aggregation), are implemented by *generalization algorithms*. For a given operator, several algorithms are usually possible (McMaster and Shea, 1992).

Related Work

On-the-fly generalization for mobile mapping is typically achieved by relying either on fast generalization algorithms or on pre-computation and hierarchical data structures (van Oosterom 2005; van Oosterom and Meijers, 2011; Weibel and Burghardt, 2008). The first approach commonly has to sacrifice cartographic quality to reduce computational complexity and achieve speed, while the second, as a consequence of pre-computation, lacks flexibility.

The first group — fast generalization algorithms — relies on simple but effective algorithms and heuristics. The preference is on rather straightforward generalization operators such as selection and simplification. For instance, feature selection based on the Radical Law (Töpfer and Pillewizer, 1966) and ordered attributes (Lehto and Sarjakoski 2005), or applying local priority criteria (Edwardes, et al. 2005).

The second group represents methods that fully rely on pre-computated generalizations stored in hierarchical data structures. Van Oosterom (2005) and van Oosterom and Meijers (2011) review and propose several data structures for real-time generalization. Initial ideas to use quadtrees and hierarchical drainage basins to support generalization have been proposed by Burghardt et al. (2004) and Edwardes et al. (2005). De Berg et al. describe an algorithm using a k-d-tree.

What is still missing are methods combining the advantages of both approaches, that is, real-time algorithms exploiting hierarchical spatial data structures. Also, no algorithm has been proposed yet that was designed for real-time performance.

Basics of Quadtrees

Useful Properties of Quadtrees

The quadtree is a well known spatial tree data structure and a generic term for a family of tessellations in which every node has four children (Samet, 1989). It is widely used for 2-D spatial indexing in GIS or in other domains, such as in computer graphics for collision detection (Moore and Wilhelms, 1988). Most importantly for our case, it has several properties that make it useful for real-time generalization.

Useful properties for real-time generalization provided by the quadtree are (Table 1): a spatial index that speeds up spatial queries; spatial coverage providing information on existence or absence of geographic features in a specified region and thus enabling estimates on density/distribution; topology of quad neighbors, providing information on local neighborhood; and recursive hierarchical subdivision of map space, thus adapting to point density and progressively building up a spatial hierarchy with implicit scale.

Table 1: Properties of quadtrees useful for real-time point generalization

Property	*Use*	*Generalization operator*
Spatial Index	Spatial queries	Selection, aggregation
Coverage	Provides estimates on densities and distribution	Selection, displacement
Topology	Quad neighborhood	displacement
Hierarchy	Recursive and progressive subdivision	All operators

Basic Operations on Quadtrees

The basic operations on quadtrees include: *insert*, *delete*, *get neighbors* and *query* the quadtree. They form the foundation of the generalization algorithms described below. Note that in the following, we will focus exclusively on quadtrees for point data. For reviews of quadtrees and associated algorithms, see Samet (1984) or Samet (1990).

A quadtree for point data can be created by recursively *inserting* the points in the tree structure until an empty leaf node is found or otherwise subdividing the tree further. The order of insertion does not define the shape of the tree, rather it is highly dependent on the spatial distribution of the inserted point set. For two close points many levels of partioning may be needed. This can be alleviated by allowing more than one object per leaf node, creating a variant of the quadtree, a so-called bucket quadtree. *Deletion* of a leaf node from the tree also happens by recursively removing all the children of the parent node if they hold less than one point altogether. To find all the points within a specific range in the quadtree (*range query*), the intersection of the *quadnodes* (i.e. nodes of the quadtree representing a quad) is checked against the query range, and in the case of intersection with the quadnodes children recursively. If results of a generalization operator are stored in the nodes of the quadtree (see “Storage”) the query range may be restricted to a specified depth inherently representing scale.

Finding neighbor(s) in a quadtree corresponds to finding adjacent leafnodes for each side and corner of a particular quadnode. An in-depth description of neighbor finding methods in quadtrees may be found in Samet (1990). Finding neighbors is important for displacement operations in generalization.

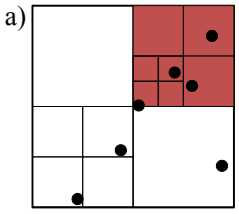
Quadtree-Based Algorithms for Point Data Generalization

General Overview

The basic idea of the quadtree-based generalization approach is to apply generalization operations to quadtree nodes according to the desired level of detail (LOD), which is mapped to the depth of a quadtree.

The generic flow of quadtree-based generalization consists of three steps. In a first step the quadtree is created by inserting the point data and their (optional) attributes. Attributes such as feature category, rank (e.g. relevance ranking generated in an external application), or other measures are assigned to the point data if they are parameters required by the desired generalization algorithm.

Table 2: Generic flow of quadtree-based generalization, applying a simple "random" selection algorithm.

1 Creation	Insert points into the quadtree. (No additional measures or attributes required for random selection algorithm of Step 2.)
2 Generalization algorithm	 **Random selection** Select one point randomly (black dot) from all the children in the current quadnode.
3 Storage	Optional: If the results are used for further zooming, it is recommended to store the results based on previous LODs.

In the second step, a quadtree-based generalization algorithm returns for each quadnode the generalized results at the desired LOD. The LOD is mapped to the tree depth (and the LOD can further be related to map scale). For each visited quadnode at the defined depth the algorithm is either applied in real-time to all child nodes, or it is retrieved from stored results from previous, pre-computed runs (see Section "Storage").

In the third step the results from step 2 are either displayed (display-and-forget), or stored in the nodes for fast retrieval in subsequent iterative generalization (see "Storage").

This process is shown in Table 2, using a very simplistic algorithm (for illustration purposes only). After the creation of the quadtree, the operator visits all quadnodes of a desired LOD and randomly selects one point from its subtree. For this particular algorithm, no attributes need to be assigned to the data points besides their 2-D position.

Quadtree-Based Generalization Algorithms

This section presents different quadtree-based generalization algorithms for the generalization operators – *selection*, *simplification*, *aggregation*, and *displacement*. First, however, we briefly review the role of geometrical measures in support of algorithms.

Measures. Geometrical measures help to inform the operation of generalization algorithms, and parameterize their outcome (Bereuter and Weibel, 2011). *Local measures* are derived from each quadnode, such as the number of points stored in the subtree, maximum depth, size and balancing of the subtree. *Global measures* are based on the complete dataset, such as total number of stored points, density, or spatial distribution. Global measures such as the average number of elements per node at one LOD or the total number of elements can be used for weighting or as a boolean decision factor for the generalization algorithm. For instance, for changing the point size (Fig. 10), or pruning elements from the tree to avoid an over-homogenization of the underlying spatial pattern.

Selection. Chooses a subset of points from the original set of points, based solely on attributes, such as a relevance value per point object (Bereuter and Weibel, 2010).

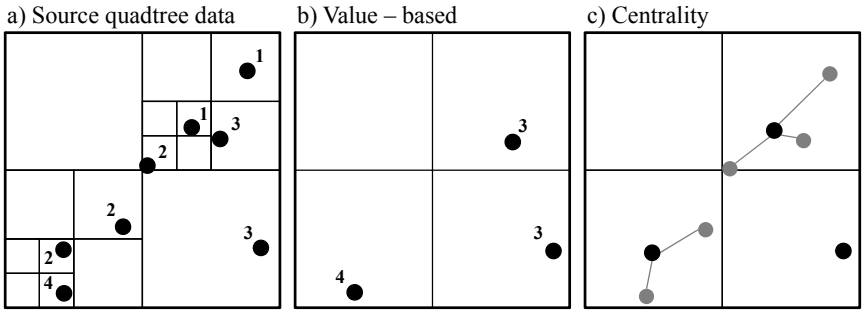


Figure 2: Point reduction algorithms, with results for depth 1: a) quadtree source data; b) value-based selection, retaining elements with highest attribute value per subtree; c) centrality-based simplification, retaining elements closest to the mean center.

Value-based selection and *rank-based selection*. Returns the most significant element per quadnode. 'Most significant' denotes a function of a numeric attribute, such as maximum value of an attribute of points occurring in a subtree. For POI data, such as restaurants, highest relevance may be used (obtained from a relevance ranker). For point collections (i.e. point data sets that encompass large collections of count or observation data), such as animal observation data, the most or least frequently occurring species may be used. If the value of the point is weighted with the leaf node's depth, the algorithm will generate a generalization result that is weighted approximately by the local point density.

Simplification. Like selection, it chooses a subset of the original points. However, it is governed by geometric properties, as in line simplification (Bereuter and Weibel, 2010; McMaster and Shea, 1992).

Centrality-based simplification. Retains the most central point per quadnode. Based on mean center (or centroid, midpoint); thus solely requires the positions of the point features.

Aggregation. Replaces two or more point features with a new placeholder feature (i.e. the point positions change). Reduces the number of points by grouping together semantically similar or spatially close points (Bereuter and Weibel, 2010).

Midpoint aggregation. Generalizes by the mean center point of a quadnode subtree's points (Fig. 3b). Like centrality simplification, it uses solely the positions of the points.

Clustering-based aggregation. Returns a placeholder (e.g. the modal center) for highly clustered and densely populated quadnodes with a high number of children, based on the positions and attributes of the points.

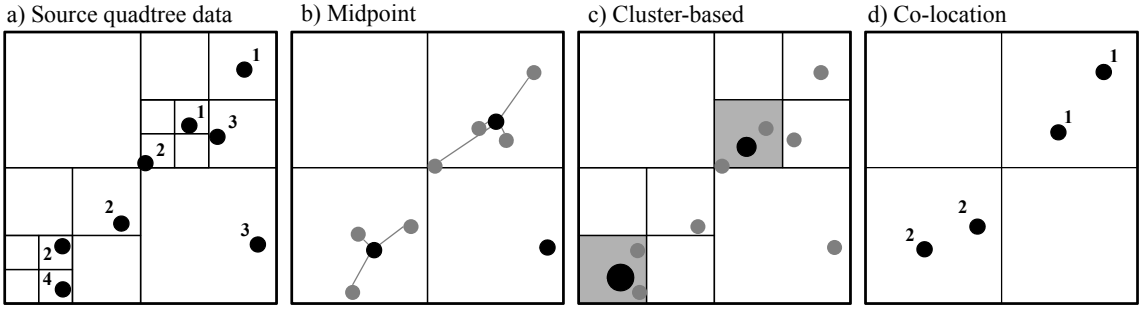


Figure 3: Aggregation algorithms: a) quadtree source data, b) midpoint aggregation; c) cluster-based aggregation; d) co-location filtering for co-occurring same valued points.

Co-location filtering. Generalizes quadnodes based on a co-location rule, such as the co-occurrence of features in a quadnode belonging to the same class, or capturing logical relationships between different point categories. Co-location of one or more point features in space seems to influence the user's decision in location-based services (Reichenbacher and De Sabbata, 2011). An example for co-location is finding co-located parking places and restaurants in a city. For each quadnode's subtree the co-location algorithm checks if it contains restaurants *and* parking spaces, and if so, how often, to weight the co-occurrence of restaurants and parking spaces within that node. The more points a child node represents the higher the probability of co-location and the number of times co-location occurs.

Displacement. Locally reconfigures point symbols to resolve spatial conflicts by moving them apart (Bereuter and Weibel 2010). Displacement does not reduce the number of points represented. It is thus usually applied for the 'finishing touches', as the last in the chain of generalization operators. Its application limited to resolve spatial conflicts in a narrow band of scales, typically from one LOD to the subsequent one. The width of the quadnode's boundary at the maximum depth of the current LOD denotes the smallest *required distance* to resolve spatial conflicts (Fig. 4b).

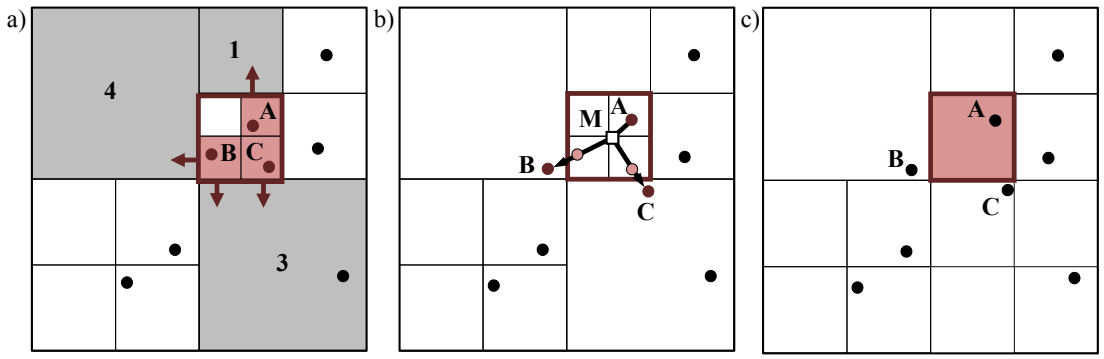


Figure 4: Steps applied by the displacement algorithm for the red quadnode: a) Points A, B, C stored in the child nodes are candidates for displacement. Gray shaded areas denote candidate nodes for displacement, numerals denote their storage capacity. Red arrows indicate possible movement direction for the points contained in child nodes. b) Optimal displacement directions for B and C. A is kept as the most central point. c) B and C are displaced and the child nodes removed.

The proposed algorithm tries to relocate points (that would otherwise be removed by a selection or aggregation operator) of a quadnode, and moves them to neighboring quadnodes. The current LOD defines the deepest possible quadnode (red square in figure 4a) at which one point can be displayed, given cartographic constraints (e.g. point symbol size). Points are moved to the most favorable (i.e. closest and of depth ≤ current LOD) neighbor quadnode with no childnodes (gray shaded quadnodes in Fig. 4a). If there is space for displacement, as in the example process illustrated in Figure 4 for Points B and C, these are displaced, while A as the most central point is kept (Fig. 4b). The child nodes of the red quadnode can be removed as only one point is now remaining inside (Fig 4c).

In order to find candidate neighbors for displacement of a given quadnode (framed red in Fig. 4) the displacement algorithm checks for each neighbor quadnode of the same depth or lower if it already stores points. The algorithm only needs to displace points if the quadnode contains more than one point, and thus is not satisfying its cartographic constraints.

For each child quadnode two or three directions, respectively, exist for the point displacement to the neighboring quadnodes, depending on whether the king's or rook's move neighborhood is used (Fig. 5a, b). For instance in Figure 4, where the rook's move neighborhood is used (Fig. 5b), for the NW child node neighbors to the North and West need to be checked. These are exactly those shared with the sides of the parent quadnode.

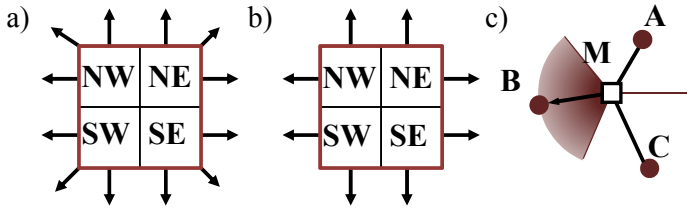


Figure 5: Options for displacement moves for point data: a) three movement options per child quadnode (king's move); b) two movement options per quadnode (rook's move); c) preferred range of movements (shaded area) for point B from the midpoint.

The *direction* in which a point is displaced is preferably directly opposed to the direction of the midpoint formed by all the quadnodes points (Fig. 5c). The preferred range of directions in which a point can be displaced is limited by the bisectors of its immediate neighboring points to the midpoint and by the neighbor nodes having sufficient capacity.

A point can only be displaced to the neighboring quadnode if that node shares the same depth and is empty, or has a lower depth. This is controlled by the quadnode *storage capacity*: how many points can the neighboring quadnode possibly store, without subdividing to a level higher than the maximal allowed depth for the current LOD (see Fig. 4a). For each possible direction (see below) the storage capacity of the neighbor node is collected and the child nodes are sorted in ascending order of the number of possible displacement neighbors and their associated storage capacity. The child node with the least possibilities for displacement is displaced first. After its displacement to its neighbor, the neighbor's storage capacity is reduced by one. If there is no other further possibility of displacement the point is kept. In the case all points can be displaced (Fig. 4a) the point closest to the midpoint is kept (Point A in Fig. 4b). If for more than one point there is no

possibility for displacement, the most central point (or the one with the highest importance value) is kept and the others removed (which is then equivalent to selection/simplification).

If the neighboring quadnode is of a lower depth and already stores one point, this node will be further subdivided to insert the moved point (Point C in Fig. 4c). If after subdivision both points happen to be inside the same childnode, the moved point is removed.

The displacement process is completed if for the quadnode in question (framed red in Fig. 4) only one point remains and the child nodes have been removed (Fig. 4c).

Extension. The above algorithm can be extended by allowing point displacement to be propagated to neighbors of degree 2, if a point cannot be displaced to direct (degree 1) neighbors using the above algorithm. For performance reasons, it is advisable to limit the search to a pre-defined, small degree of neighborhood. Due to lack of space, the extended algorithm will not be further elaborated on.

Storage

For interactive zooming the quadtree lends itself nicely as a storage structure for generalization results (rather than serving merely as a spatial index). Figure 6 shows the use of the quadtree as a storage structure schematically. The generalization algorithm returns the results for each visited node and its subtree (Fig. 6a, algorithm search range). The subtree in this case may be the complete subtree (Fig. 6a) or a depth restricted subtree (Fig. 6c). "Depth restricted" means that the generalization algorithm does not consider the whole subtree, but solely up to a predefined level. Each result returned from a generalization algorithm is stored in the visited quadnode. If, due to zooming, the same LOD is queried a second time, the generalization process simply retrieves the stored result from each quadnode at the queried level of detail.

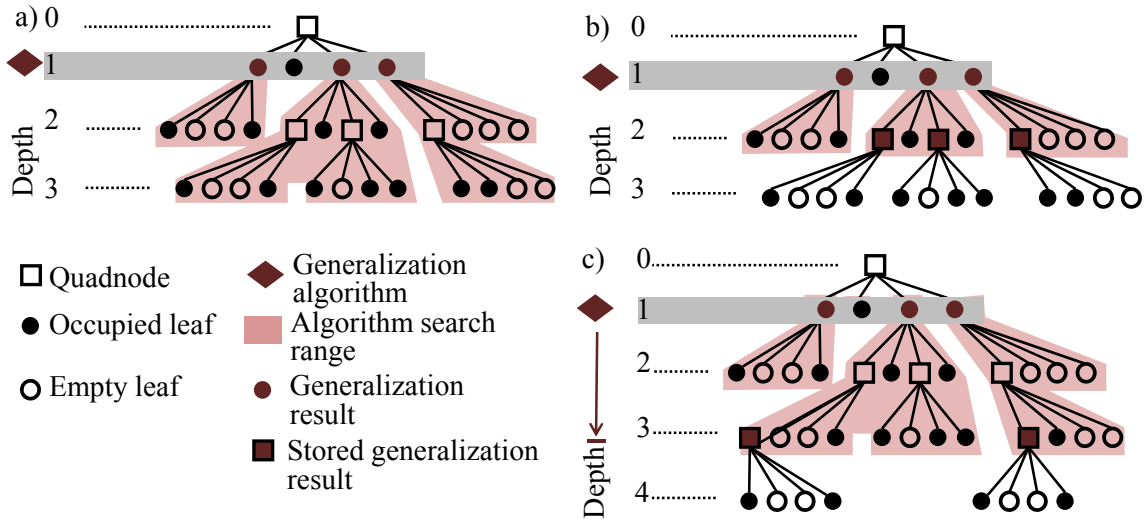


Figure 6: Storage options: a) for each quadnode only the leaf nodes for generalization are investigated; b) for each quadnode the results (black square with red fill) from the previous nodes are considered; c) depth restricted generalization algorithm.

If the generalization ascends in the tree, decreasing the LOD, for each new level only the previous LOD needs to be visited to derive the next LOD, optimizing the algorithm in terms of speed (Fig. 6b). It is furthermore possible, depending on the generalization algorithm, to limit the search range per quadnode and apply a depth restricted algorithm (Fig. 6c). For instance, in the case of the displacement algorithm based on the result of a previously run centrality simplification algorithm, the displacement algorithm moves the points resulting from the centrality algorithm at the next higher LOD.

Storing the results in the quadnodes supports interactive zooming and allows to apply some of the generalization algorithms only once on the whole tree.

Experiments

Prototype and Data

The proposed quadtree-based generalization algorithms have been implemented in a prototype generalization platform using Java and Processing (www.processing.org).

The foreground data used originates from OpenStreetMap OSM POI data for the Canton of Zurich in Switzerland. For the background map a generated soft-toned map style from CloudMade (www.cloudmade.com) was used. The POI dataset features several categories, from which a subset was extracted as a thematic layer for this section.

Results and Discussion

Following the description of different generalization algorithms based on the quadtree, selected results illustrate their application below. The chosen geographic region depicts either the city or region of Zurich, Switzerland, depending on scale. The chosen POI dataset contains all types of eating places for the city of Zurich such as restaurants, pubs or coffee bars. The dataset shows strong clustering in the dense city area and regions with marked differences in point density, thus its selection as a test area and dataset.

How one selects and aggregates elements as representatives for the next scale, to abstract and reduce the depicted information, has a strong influence on how the resulting pattern is perceived by the user. Due to the regular structure of the quadtree, quadtree-based generalization may lead to a certain degree of homogenization of the point distribution. Figure 7b shows an example of strong homogenization, where an (admittedly) cartographically suboptimal algorithm was used that simply aggregates points of a quadnode to the position of the corresponding tile center (note that this algorithm was not described above). However, even for such strong homogenization, the map can be improved, giving the user a better idea of the underlying spatial pattern, by weighting the point symbol size by the number of input points they represent (Fig. 7c). Figure 7 also shows points that fall into Lake Zurich as a result of generalization, which hints at the lack of spatial constraints used in this case.

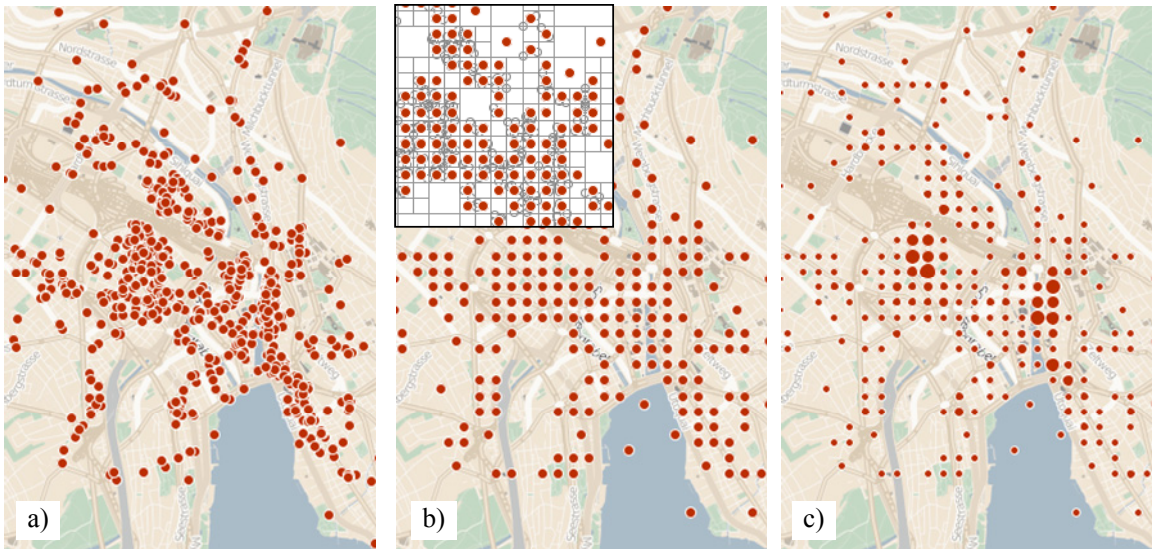
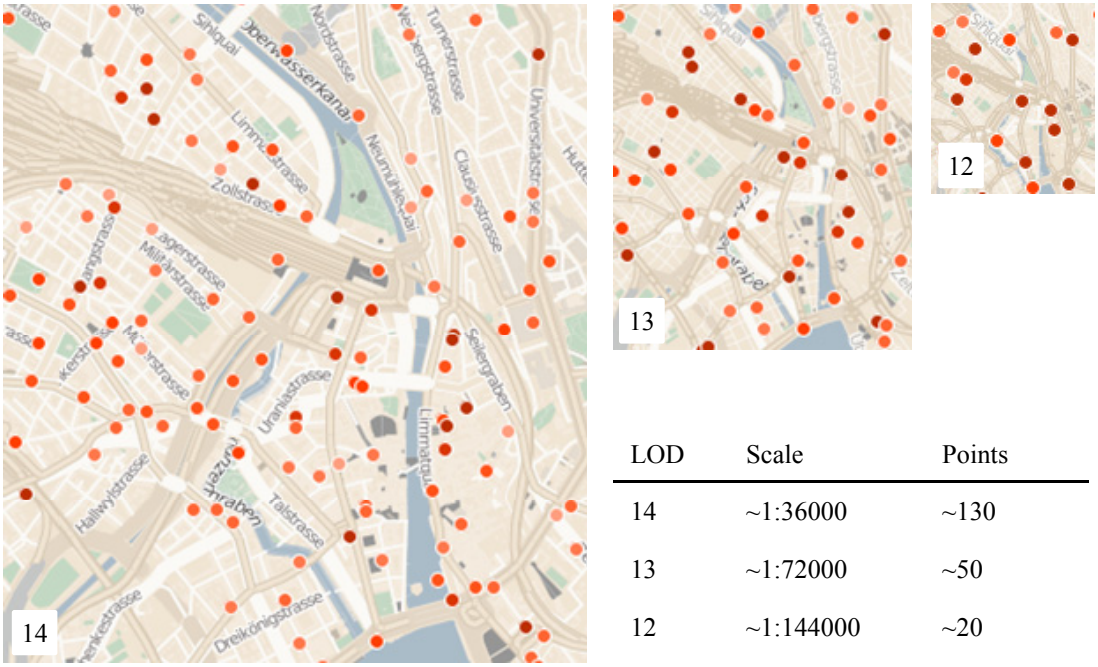


Figure 7: a) Distribution of eating places in the City of Zurich; b) aggregation of points inside a quadnode to the tile center; c) weighting the point symbols by the number of original points they represent.

Selection based on ranking or a selection function based on attributes only retains one representing element per quadnode. In Figure 8 only the highest ranked points per quadnode are kept for a given LOD, representing the highest ranked eating places in Zurich. Due to the strong selection confined to quadtree tiles, most of the spatial conflicts are resolved, though no displacement to resolve remaining overlaps was applied. The reduction of LOD shows nicely how the higher ranked (dark red elements) persist across scales, and how the generalization approximately retains the spatial point configuration.



LOD	Scale	Points
14	~1:36000	~130
13	~1:72000	~50
12	~1:144000	~20

Figure 8: Rank-based selection, progressively reducing the LOD from left to right, retaining the highest ranked points per quadnode (dark red points).

While value-based selection retains values a user is most interested in, simplification by centrality keeps the most central point per node (Fig. 9), that is, the point closest to the mean center of all points falling within the generalized node. Figure 9b illustrates how the most central point is selected. As becomes noticeable, the selection of the most central point becomes more stable when more points are contained in a quadnode, while generating questionable results for quadnodes with only two points.

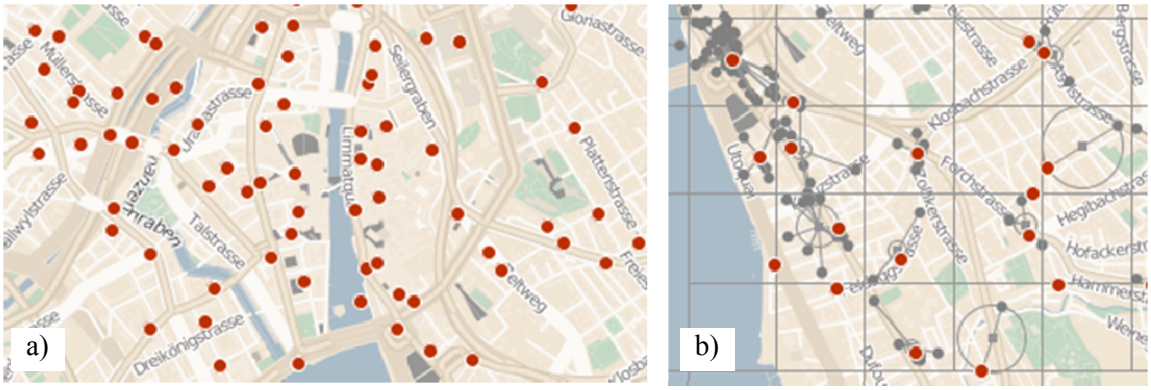


Figure 9: a) centrality-based selection for LOD 14; b) debug view, rectangles depict a node, gray circles removed points, and the square denotes the mean center of the points inside the node's boundary.

Aggregation groups two or more elements and replaces them by a new placeholder point feature, weighted by the number of features aggregated. Figure 10a shows the result of midpoint aggregation, as blue, graduated circles weighted according to the number of aggregated points. For comparison, the resulting points of a ranking-based selection for the same LOD are overlaid.

Co-location filtering is another type of aggregation where spatially co-occurring points are retained and weighted by their frequency. Figure 10b side shows aggregation weighted by the number of co-occurrences of restaurants and parking places per node.

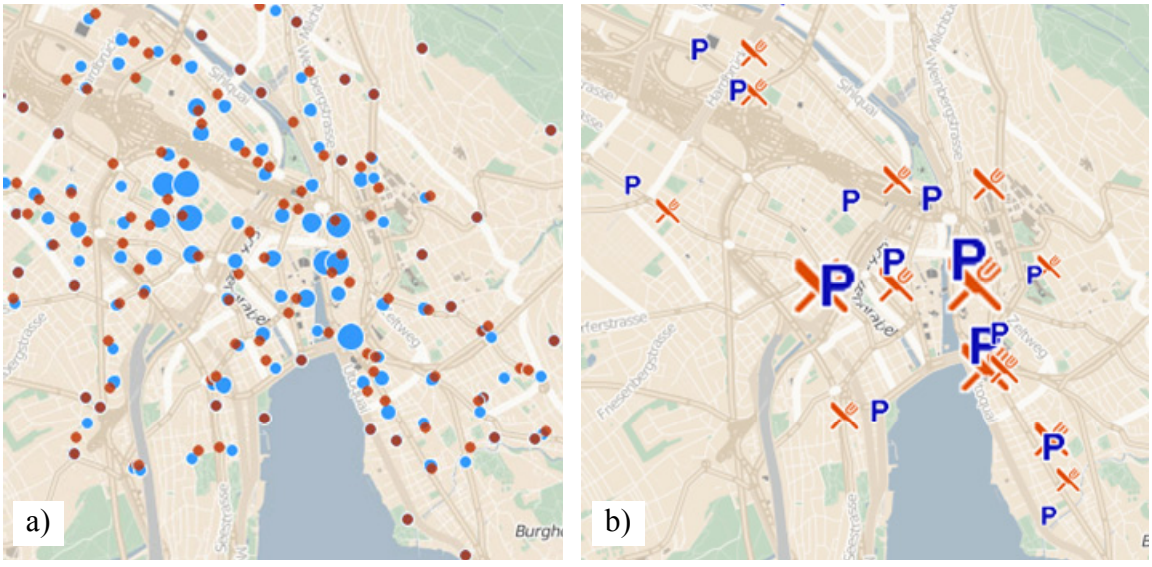


Figure 10: a) overlay of midpoint aggregation (blue, with symbol size weighted by number of aggregated points), and ranking-based selection (red) for LOD 13, overlaps shown in purple; b) aggregation by co-location, with symbol size weighted by number of co-occurrences for restaurant *and* parking.

To further resolve spatial conflicts in a map and retain more elements than for instance in the case of centrality-based simplification a displacement algorithm can be applied (Fig. 11). The displacement algorithm tries to accommodate as many points as possible, keeping only one point per quadnode, denoting the highest possible resolution for the current LOD (given cartographic legibility constraints). The proposed displacement algorithm only displaces points to its closest neighbor if the neighboring quadnode has a storage capacity higher than one for the current LOD. If for a node no displacement is needed, no check is applied for overlaps with points stored in neighboring quadnodes that are adjacent to the node's boundary. Thus, some overlaps are still visible in Figure 11b, which could be resolved by an additional check while applying the displacement algorithm. The displacement result (Fig. 11b) is capable of displaying more points than the corresponding centrality-based simplification (Fig. 11a), since it was possible to remove some overlaps and move points to adjacent quadnodes. Figure 12, on the example of a sequence over three consecutive LODs, shows how the displacement algorithm operates.

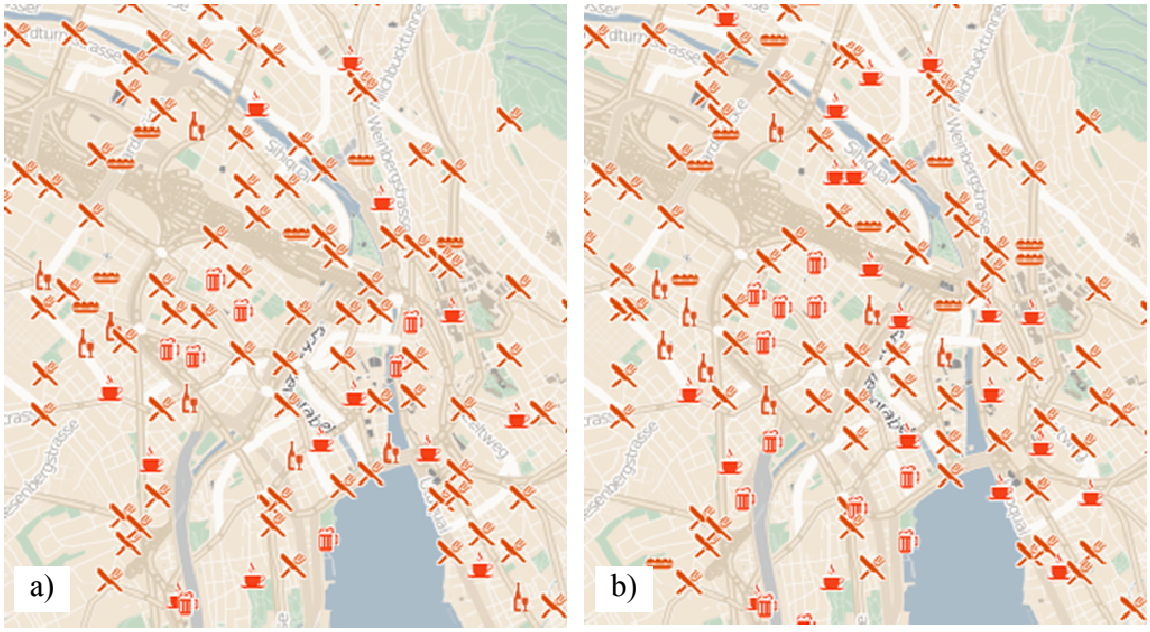


Figure 11: a) result of centrality based simplification, displaying 114 elements; b) displacement applied after selection based on centrality displaying 124 elements.

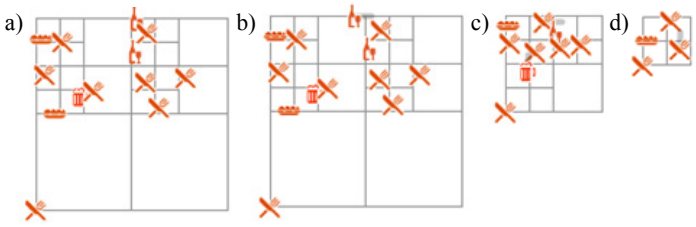


Figure 12: Example of displacement algorithm: a) raw data; b-d) displacement sequence for three LODs.

Finally, Figure 13 illustrates in a series of steps how different generalization operations may be applied and chained together. As a reference the raw, ungeneralized input data is displayed in Figure 13a. In a first step (Fig. 13b) centrality-based simplification is used to relax spatial conflicts, while largely maintaining the spatial distribution of the input data.

Remaining spatial conflicts are then further resolved by the displacement algorithm in (Fig. 13c). In Bereuter and Weibel, 2012, we have introduced 'content zooming', which provides the user with the capability to change the amount and granularity of foreground information presented, while keeping the geometric map scale the same. Content zooming is intended to allow overriding the effects of 'standard' map generalization, focusing on optimized content representation from the perspective of information seeking by a mobile user. In our example, content zooming is shown by overriding standard settings for displacement: Figures 13d and 13e show displacement results for an increased and a decreased LOD, respectively. Zooming in (13e) and out (13d) of the content enables the user to get a better idea of the spatial distribution and allows interactively adapting the amount of map content to his/her needs.

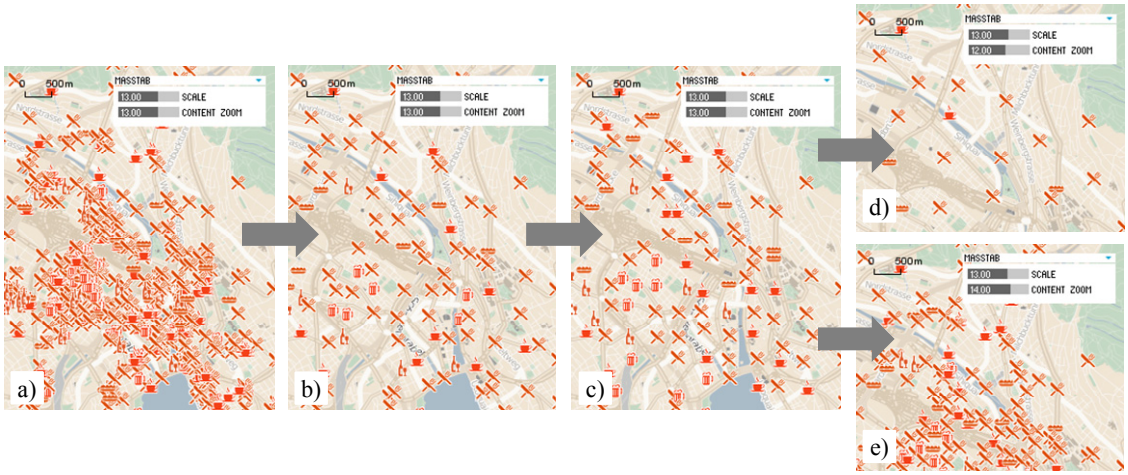


Figure 13: Series of steps of the application of different generalization algorithms (see text).

Conclusions

In a move to combine the advantage of efficient heuristics and hierarchical data structures, we have proposed a series of algorithms for on-the-fly point data generalization based on the quadtree, which has several useful properties that make it interesting for computationally efficient generalization. On-the fly generalization of point data is required to effectively display foreground data in online map services for web and mobile mapping. We have proposed algorithms that implement the major generalization operators that can be applied to point data: selection, simplification, aggregation, and displacement. We have also shown how the quadtree data structure cannot only be used as a spatial index to make the generalization efficient, but also as a storage structure.

In this paper, only few experiments have been shown. They may be sufficient to illustrate the potential of using quadtrees for point data generalization, but more systematic experiments will be required to thoroughly assess the strengths and limitations of the proposed algorithms, and compare them to 'classical' generalization algorithms. Further algorithmic improvements are also possible, such as inclusion of spatial constraints imposed by the background map (e.g. the lake in Fig. 7).

Acknowledgements

The research reported in this paper represents part of the PhD project of the first author. Funding by the Swiss National Science Foundation through project *Generalisation for Portrayal in Web and Wireless Mapping (GenW2+)* (SNF no. 200020-138109) is gratefully acknowledged.

Image sources: Base map © 2012 CloudMade – map data CC BY SA 2012 OpenStreetMap.org. Map icons CC BY SA Nicolas Mollet mapicons.nicolasmollet.com.

References

Bereuter, P. and Weibel, R. (2010) Generalisation of point data for mobile devices: A problem-oriented approach. *Proceedings of the 13th Workshop on Progress in Generalisation and Multiple Representation ICA Commission on Generalisation and Multiple Representation,* pp. 1–8.

Bereuter, P. and Weibel, R. (2011) A Diagnostic Toolbox for Assessing Point Data Generalisation Algorithms. *Proceedings, 25th Intl. Cartographic Conference*, Paris, 3-8 July 2011. CD-ROM, 10 pgs.

Bereuter, P., Weibel, R. and Burghardt, D. (2012) Content zooming and exploration for mobile maps, *Proceedings of the AGILE 2012 International Conference of Geographic Information Science*, pp. 74–80.

de Berg, M., Bose, P., Cheong, O., Morin, P., & de Berg, M. (2004) On simplifying dot maps. *Computational Geometry*. *27*, 1, pp. 43–62.

Burghardt, D., Purves, R. S. and Edwardes, A. J. (2004) Techniques for on the-fly generalisation of thematic point data using hierarchical data structures. *Proceedings of the GIS Research UK 12th Annual Conference*.

Edwardes, A. J., Burghardt, D. and Weibel, R. (2005) Portrayal and Generalisation of Point Maps for Mobile Information Services. *Map-based mobile services: theories, methods and implementations*. 2, pp. 11–28. Berlin, Springer.

Lehto, L. and Sarjakoski, L. T. (2005) Real-time generalization of XML-encoded spatial data for the Web and mobile devices. *International Journal of Geographical Information Science* 19, 8-9, pp. 957–973.

McMaster, R. B. and Shea, K. S. (1992) Generalization in Digital Cartography. Association of American Geographers, Washington D.C.

Moore, M. and Wilhelms, J. (1988) Collision detection and response for computer animation. *ACM SIGGRAPH Computer Graphics*, *22*, 4, pp. 289–298.

Reichenbacher, T., & De Sabbata, S. (2011). Geographic relevance: different notions of geographies and relevancies. *SIGSPATIAL Special*, *3*(2), pp. 67–70.

Samet, H. (1984) The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys* 16, 2, pp. 187-260.

Samet, H. (1990) *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley Publishing Company.

Weibel, R., and Burghardt, D. (2008) Generalization, On-the-Fly. *Encyclopedia of GIS*. pp. 339–344. New York: Springer.

van Oosterom, P. (2005) Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest. *Cartography and Geographic Information Science*, *32*, 4, pp. 331–346.

van Oosterom, P. and Meijers, M. (2011) Towards a true vario-scale structure supporting smooth-zoom. *14th ICA/ISPRSWorkshop on Generalisation and Multiple Representation & the ISPRS Commission II/2 WG on Multiscale Representation of Spatial Data*.

Pia Bereuter, Department of Geography, University of Zurich, Winterthurerstrasse 190. Email <pia.bereuter@geo.uzh.ch>

Robert Weibel, Department of Geography, University of Zurich, Winterthurerstrasse 190. Email <robert.weibel@geo.uzh.ch>