Retrieving tGAP data with a stateless client for visualization

Usable and well scaled maps for consumers

Work Package 4

Martijn Meijers

Delft, December 2008

RGI Report No. 3

Retrieving tGPA data with a stateless client for visualization

Usable and well scaled maps for consumers

Work Package 4

Martijn Meijers

Delft, December 2008

RGI 233 Report No 3

Summary

Today we see a huge increase of the use of geo-information in mobile devices. All current solutions are based on static copies that are stored on the mobile device. This makes dynamically adapting the map to new information and to the changing circumstances of the user impossible. With the availability of high bandwidth wireless connections (such as UMTS) better, more dynamic, solutions are possible: The server generates a proper, up-to-date map of the region of interest at the right level of detail for display and adjusted to the needs of the user. For a mass market (consumers of mobile maps) the human factors aspect is very important. The currently available mobile maps solutions still have insufficient user-interfaces. Extremely important is the issue of context as the user gets 'lost' very easily on the small mobile displays when zooming and panning. Based on a selection of use cases (navigation, tourist support, etc.), User-Centered Design techniques will be applied to develop small prototypes / simulations and the interaction and the quality of the maps in these prototypes / simulations will be evaluated.

The project runs from 2006 to 2008 and is organized in the following Work Packages:

- WP 1 'Preparation' (month 4-9, 2006);
- WP 2 'Prototype development' (month 10, 2006-6, 2007);
- WP 3 'Evaluation of first prototypes' (month 7-12, 2007);
- WP 4 'Improved prototypes' (month 1-6, 2008);
- WP 5 'Evaluation of improved prototypes' (month 7-12, 2008).

This report belongs to WP 4 and consists of the following chapters:

1. Introduction
2. Mapping a bounding box to an importance value
3. Polygonize
4. Retrieval of tGAP
5. Experiment
6. Discussion of the results

Project leader	:	Section GIS Technology OTB Research Institute for Housing, Urban and Mobility Studies Delft University of Technology
Address	:	Jaffalaan 9 2628 BX Delft
Contact	:	ir. B.M. Meijers
E-mail	:	b.m.meijers@tudelft.nl
Website	:	http://www.rgi-otb.nl/pages/uwms2

Contents

Retrieving tGAP data with a stateless client for visualization

Martijn Meijers

December 24, 2008

1 Introduction

Currently the topological Generalized Area Partitioning (tGAP) structures are implemented in an object-relational geo-DBMS (Oracle DBMS and PostgreSQL with PostGIS extension). The current tGAP implementation is based on topological structured data, stored in a *node edge face* (NEF) data structure. With this data structure, each edge is directed (it has a start and an end node), and the pointers to the faces left and right of each edge are also stored. The tGAP structures (the face tree and the edge forest) store the result of an aggregation process. Each area object is represented by a topological face and all faces have a number assigned representing their relative importance (importance_{low}). This importance value is used to decide when to aggregate a face – the value is based on the size of each object and on the feature classification each object has. The least important face is aggregated with the most compatible neighbour (compatibility is based on the length of the boundary between two faces being merged and the feature classification of both faces). The sum of the two importance values will be assigned to the two old faces as their importance high value (importance_{high}). The newly created face will have this sum value as the importance_{low} value. This process continues, until only one face is left: Result of this process is that each face will have an importance range assigned, i.e. $[\text{importance}_{low}, \text{importance}_{high}\rangle$. Also the edges will have an importance range assigned. For more details on this aggregation process, see Van Oosterom (2005).

In this report, technical details will be provided for transfer of the data from the topological data structures at a server to a stateless client. This means that the client requests data after each user action, does not maintain state of what part of the data is retrieved already and therefore is relatively simple to implement. Another option can be to use a stateful approach, in which the client remembers what it has asked previously (and uses this knowledge to query for information not asked before), but this means that more work and extra data structures at the client side are needed to implement the client.

The remainder of this technical report is structured as follows. First, as we do not want to expose the characteristics of the data structures at the server too much to the client – as it only has 'knowledge' about displaying topological structured data, but not about the importance values stored with all the objects – we describe a way to map a viewport bounding box to an importance value (section 2). This way, a client only has to report the current viewport extent to the server and can be sure to receive the right amount of data for a specific level of detail as the server translates the extent of the viewport to a suitable importance value to query the data structures. Second, in section 3 we describe an algorithm to reconstruct the face geometry called

POLYGONIZE (from the information sent, being faces and edges). Third in section 4, different options are given to query the face and edge information in the data store. One approach is described in depth, as this is the best option in terms of minimalistic data transport. We end the report with some discussion of the obtained result (section 6).

2 Mapping a bounding box to an importance value

As described in the introduction, all faces in the tGAP structure have an importance range associated. To visualize data retrieved from the tGAP structures, the client sends the extent of the current viewport to the server and expects a certain amount of face information in response. This amount of information must not contain too many objects (otherwise the user will suffer from information overload). Therefore, a fixed number of objects is set and will be used for retrieving data in such a way, that the amount of objects, i.e. faces, to be retrieved remains relatively constant (independent from what level of detail is retrieved).

The number of faces wanted on a screen is set to what will be shown when the user is looking at the complete extent of the dataset: the extent of the dataset is then equal to the extent of the viewport (the ratio between the two extents is equal to one). The question to be answered now, is which importance value satisfies this given number of objects. For this a lookup table is created: As the aggregation process each time aggregates two faces, and the two old faces get the sum of the two old faces as their importance high values, one can find how many objects there are at a certain importance value, by grouping the face table per importance high value that is present, counting how many faces are having such an importance high value and relating this to the total number of faces that are present in the original dataset (the original number of faces minus the cumulative sum of all merges until a given importance value gives the number of faces for the complete extent at a certain importance value).

Now, if a user zooms in, a new query window, which is smaller than the previous window, is sent to the server. Assume that the user zooms in to a quarter of the original dataset. The ratio of the full extent of the dataset and the query window extent is $1/\frac{1}{4} \equiv 4$. Subsequently, this number is multiplied by the fixed number, and this gives a new number of objects for the lookup table which leads to a new importance value. With this new importance value the data structures can be queried. Note that a lower importance value will lead to more objects for the full extent of the dataset, but because the user zooms in, a smaller spatial extent is used and the average of the number of objects retrieved remains constant (as was our initial goal).

3 Polygonize

To visualize the faces, the face geometry needs to be reconstructed and therefore all edges are needed that are related to a face. The process of reconstruction, called POLYGONIZE, is dependent on the configuration of the edges. Based on the edges, rings are formed that together describe the geometry of the face.

POLYGONIZE works as follows: In case a face is related to only one edge, one ring is formed based on this edge. This is the simplest case possible. When more edges are related to a face, the process of forming rings is more extensive. First, a node to edge lookup table is created, containing start and end nodes as keys which are pointing to the edges. Second, an edge list is used in which all edges that are belonging to the face are stored. An edge is taken from the list, and based on the face that is reconstructed and the side of the edge on which this face lies

(left or right), a next edge is taken from the list via the lookup table (using the start or end node identifier of the edge). This process continues until the edge list is empty.

In case more than one ring can be formed out of the edges, islands are present in a face. After finding all rings, islands rings always have a smaller area than the (largest) outer boundary ring. Thus when looking at the areas of the rings a decision can be made, which of the formed rings must be the outer one. A special case when reconstructing islands is the case of an island touching at the outer boundary ring of a face. Based on the information from the node to edge lookup table, more edge continuations are possible, i.e. the same node in the lookup table points to more edges. This ambiguity can be solved, by using the geometry of the edges and sorting the edges by angle in which they arrive at the node.

4 Retrieval of tGAP data

Faces and edges are stored in two separate database tables and, apart from the attribute values for the NEF data structure, are extended with an importance range. Four different ways of querying the two tables, based on an importance value and a spatial extent, are described in this section. This is followed by an in-depth description of one the four approaches, as this approach minimizes the number of faces and edges to be retrieved.

4.1 Retrieval methods

For all retrieval methods, it is necessary to first find an importance value to query the tGAP structures (by mapping the given extent to an importance value, as described in section 2) and retrieve a 'slice' of data that then can be processed by POLYGONIZE so visualization of the geometry of the faces can take place. In the following, four different approaches are described:

Option I With one query, we select faces and edges. With this query the database first retrieves the faces based on the overlap of their bounding box with the query window extent. Second, the edges are retrieved from the datastore with an administrative join based on left and right face information. Pro: Complete information, ready for visualizing. Con: Too much information is retrieved (outside the area, but also double edges are retrieved – per edge a version for the left and for the right face is retrieved), probably performance penalty for join (although DBMSs are supposed to perform joins well).

Option II With one query we select the faces, based on the overlap of their bounding boxes with the query window. With a separate query we retrieve all distinct edges based on the identifiers and the aggregated bounding box of all the faces from the first query. Pro: Double edge retrieval is prevented while distinct edges are retrieved, Con: too much information is sent (also edges outside the viewport are retrieved).

Option III We use two separate queries: Faces are selected based on the overlap of bounding box with query window. Subsequently, we select the edges based on the overlap of their adjacency box (abox) with the query window. An abox is a bounding box around the join of the two adjacent faces their bounding boxes. Pro: Complete information is retrieved for POLYGONIZE (even too much, also edges outside the query window, for faces that are not in the viewport, might be retrieved. However, each edge is only retrieved once). Con: To the edge table in the datastore the abox needs to be added plus an extra index for fast searching.

Option IV We use two separate queries: We select faces on overlap of bounding box with query window, then we select the edges based on their own bounding box. Pro: lean (narrowest selection possible on bounding box), Con: missing edges that are currently outside the viewport, but which belong to a face that is visible inside the viewport. These missing edges can be added dynamically, as all the retrieved edges intersect with the edges of the query window and thus can be clipped.

The last option described is theoretically the leanest in terms of the number of topological primitives (faces and edges) to be retrieved. However, this option is not straight forward, due to the missing edges that lay outside the query window. Therefore we developed a method to be able to reconstruct the geometry (that is to be visualized), which is described in the next subsection.

4.2 Clipping of edges and modifications to Polygonize

Clipping of edges works as follows: Only these edges are retrieved that overlap with the current viewport (as described in the last option of the previous subsection). The information retrieved will be incomplete, i.e. implicit edges at the rim of the query window need to be found. The relationship that the retrieved edges have with the rim of the query window is used to find this missing information: After retrieval of the edge data, the edges are processed based on their bounding box: If an edge is completely inside, it is just added to a list, ready for POLYGONIZE. Otherwise an edge likely intersects with the rim of the query window that consists of 4 edges, although an edge still might be completely outside as well having no interaction with the query window at all. Those edges will be processed with an extended Liang-Barsky algorithm for line clipping (see e.g. Hearn and Baker, 2003, chapter 6).

This extended clipping algorithm works as follows: All edges that possibly intersect the query window, are processed segmentwise from start node towards end node. During this process information is gathered on which parts of the geometry of the edge are inside the query window. Those segments always are already adjacent in the original edge. For each part of the geometry that lies inside the query window a completely new edge is created, with information for the start and end node and the faces that are adjacent to the edge part, plus the geometric part that is inside the query window. New start and end nodes are recorded if this is necessary: an identifier is created based on the coordinate where the edge is clipped. These clipped coordinates are stored in the clipped nodes list, together with the four corners of the query window, which are also added. This clipped nodes list is sorted on the angle the nodes have with respect to the viewport centre. The newly created edges are used and added to the list of edges on the screen and the information of the original edge is abandoned.

Based on the list of edges and the clipped nodes list POLYGONIZE can do its work, albeit a modified version needs to be used: A modified version of POLYGONIZE processes all edges related to one face, retrieved from the list created by the clipping process together with the sorted, clipped nodes list. Rings are formed (similar to what is described in section 3). A difference with the 'normal' operation (when all information per face is available) is that when the reconstruction process ends up at a node where no further edge connection is found (via the lookup table of the start and end nodes of the edges), the sorted clipped node list is consulted for finding what the next node is on the rim of the query window (in the correct direction). From this clipped node the reconstruction is then continued, if an edge incident with this rim node is found, the reconstruction continues at this edge. If there is no edge found, because the

node is one of the corners of the query window, a next node from the sorted list is retrieved, until a node with an incident edge is found.

5 Experiment

We conducted an experiment to see how the options we described would perform in practice. In this experiment we used the Python[1] scripting language to connect to a PostGIS[2] database that has the tGAP tables stored. We retrieved tGAP data for 8 different areas. Figures 8 and 9 (p. 11 and 13 respectively) show the faces after retrieval. The orange rectangle is the query window that is used to query the data structures (the viewport).

The queries we used were based on three of the four options described in section 4.1 (option I, II and IV were used). Figures 1, 2 and 3 show the queries used (see pp. 6–8). The queries of the 8 areas were executed after each other, simulating a user zooming in at the center of the dataset. This zoom-in simulation was repeated 10 times for each option tested. The overall best performing set of queries (the ones with the lowest time for all queries together) were used for further analysis.

As can be seen in table 1, we obtained measurements with respect to (a) the number of edges retrieved and (b) the number of coordinates to be transferred. Figure 4 and 5 provide a normalized version of the values, all against option IV (when clipping is enabled). It can be seen that option I is most expensive (as most boundaries are transferred twice) and on average 2.5 times more data is transferred compared to option 4. E.g. if we take a look at the first area, 64,159 are transferred more when comparing option I with option IV. In our experiment geometry is encoded in the Well-Known Binary WKB format, thus this means around 1,000 kB extra data transfer (as each coordinate takes up 16 bytes in this standardized format). Note that Figure 8 shows clearly that (unneeded) information is retrieved that is outside of the viewport to be visualized. Contrary to what is shown in Figure 9, option IV also retrieves edges that are partly outside the viewport, but these are clipped and only the result of this clipping process is shown.

We also obtained timing measurements (see table 2). Query times vary with the area retrieved. Figure 6 provides better insights in the measurements (normalized against option IV). As can be seen in this figure, ption IV is not necessarily always the fastest. However, overall performance is best (it has the lowest sum). This also holds when taking also the execution of Polygonize into account (Figure 7). Observe, that the time Polygonize needs, goes up linearly with the amount of coordinates to be processed. This explains why there is no difference between the options when data is retrieved in case of area 1: The amount of coordinates to be processed by Polygonize are exactly the same as the viewport extent is equal to the dataset extent.

6 Discussion of the results

We presented different methods for retrieval of data from the tGAP structures. In addition, we explored a way of clipping edges. We showed that the clipping approach out-performed the other approaches tested. This is mainly due to the advantages for data transport (the number of objects to be retrieved is smaller) and that this method is rather cheap to perform in real-time (as not a huge number of edges have to be clipped).

[1] www.python.org
[2] www.postgis.org

We conclude this report by saying that if a stateful architecture is considered in which the client knows more about the structure (thus also has an understanding of the importance values used at the server side for instance) the requirements for data retrieval are likely different. Although this is the case, parts of this report may be helpful, while developing such an architecture.

References

D. D. Hearn and M. P. Baker. *Computer Graphics with OpenGL*. Pearson Prentice Hall, 2003. ISBN 0-13-120238-3.

P. Van Oosterom. Scaleless topological data structures suitable for progressive transfer: the gap-face tree and gap-edge forest. In *Proceedings Auto-carto 2005*, Las Vegas, Nevada, March 2005. Cartography and Geographic Information Society (CaGIS).

Appendix – Queries, Measurements and Visualizations

Integrated Face and Edge Query:

```
SELECT
    f.face_id,
    f.feature_class,
    el.edge_id,
    el.start_node_id,
    el.end_node_id,
    el.left_face_id,
    el.right_face_id,
    el.imp_low,
    el.imp_high,
    AsBinary(el.geometry) as geometry
FROM
    :dataset:_tgap_face f
JOIN
    :dataset:_tgap_edge_lod el
ON
    el.imp_low <= :importance: and el.imp_high > :importance:
AND
    (f.face_id = el.left_face_id OR f.face_id = el.right_face_id)
WHERE
    f.mbr_geometry && GeomFromEWKT(':query_window:')
AND
    f.imp_low <= :importance: and f.imp_high > :importance:
```

Figure 1: Integrated Face and Edge query – Option I

Area	Option I	II	IV
1	1750	921	921
2	2832	1530	1268
3	4339	2388	1882
4	4032	2228	1736
5	3564	1992	1507
6	2832	1602	1182
7	900	570	291
8	339	241	80
Σ	20588	11472	8867

(a) Number of edges

Area	Option I	II	IV
1	130381	66222	66222
2	110201	60009	44831
3	64645	36111	25459
4	32017	17864	12537
5	20568	11691	7913
6	12610	7219	4884
7	3992	2514	1055
8	1398	975	285
Σ	375812	202605	163186

(b) Number of coordinates

Table 1: Volume of data retrieved

Area	Option I	II	IV
1	628	338	333
2	604	364	280
3	505	1115	293
4	403	549	836
5	516	398	752
6	211	372	275
7	67	61	153
8	26	26	13
Σ	2960	3223	2935

(a) Time (milliseconds) needed for querying and data retrieval from data store

Area	Option I	II	IV
1	1937	1890	1814
2	1367	1332	1051
3	864	832	639
4	521	485	373
5	358	350	262
6	243	240	188
7	78	75	46
8	28	30	15
Σ	5396	5234	4388

(b) Time (milliseconds) needed for transformation from topological model to simple feature geometry (POLYGONIZE)

Table 2: Time needed for retrieving and processing request

Face query:

```
SELECT
    f.face_id,
    f.feature_class,
    AsBinary(f.mbr_geometry)
FROM
    :dataset:_tgap_face f
WHERE
    f.mbr_geometry && GeomFromEWKT(':query_window:')
AND
    f.imp_low <= :importance: and f.imp_high > :importance:
```

Edge query:

```
SELECT
    el.edge_id,
    el.start_node_id,
    el.end_node_id,
    el.left_face_id,
    el.right_face_id,
    el.imp_low,
    el.imp_high,
    AsBinary(el.geometry) as geometry
FROM
    :dataset:_tgap_edge_lod el
WHERE
        el.imp_low <= :importance:
    AND
        el.imp_high > :importance:
    AND
        (el.right_face_id IN (:face_ids:) OR el.left_face_id IN (:face_ids:))
    AND
        el.geometry && GeomFromEWKT(':aggregated_face_window:')
```

Figure 2: Face and Edge query – Option II

Face query:

```
SELECT
    f.face_id,
    f.feature_class,
    AsBinary(f.mbr_geometry)
FROM
    :dataset:_tgap_face f
WHERE
    f.mbr_geometry && GeomFromEWKT(':query_window:')
AND
    f.imp_low <= :importance: AND f.imp_low > :importance:
```

Edge query:

```
SELECT
    el.edge_id,
    el.start_node_id,
    el.end_node_id,
    el.left_face_id,
    el.right_face_i,
    el.imp_low,
    el.imp_high,
    AsBinary(el.geometry) as geometry
FROM
    :dataset:_tgap_edge_lod el
WHERE
        el.imp_low <= :importance:
    AND
        el.imp_high > :importance:
    AND
        el.geometry && GeomFromEWKT(':query_window:')
```

Figure 3: Face and Edge query – Option IV

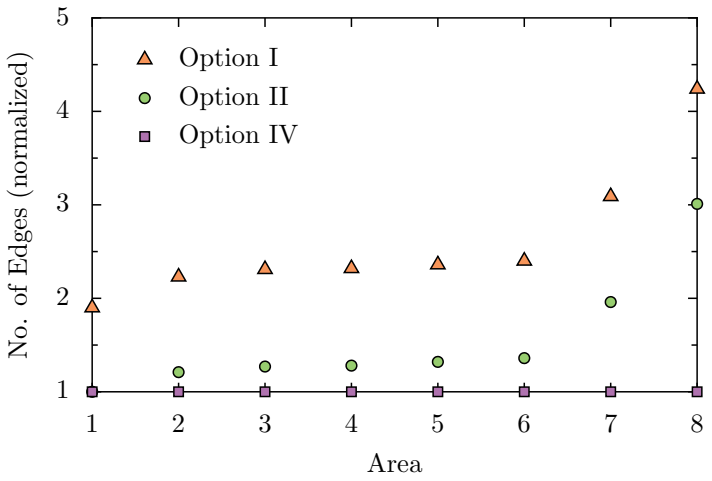


Figure 4: Number of edges retrieved normalized against option IV

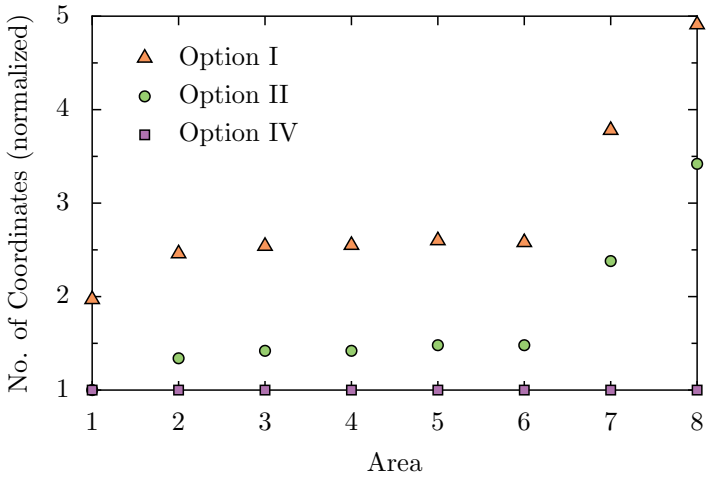


Figure 5: Number of coordinates retrieved normalized against option IV

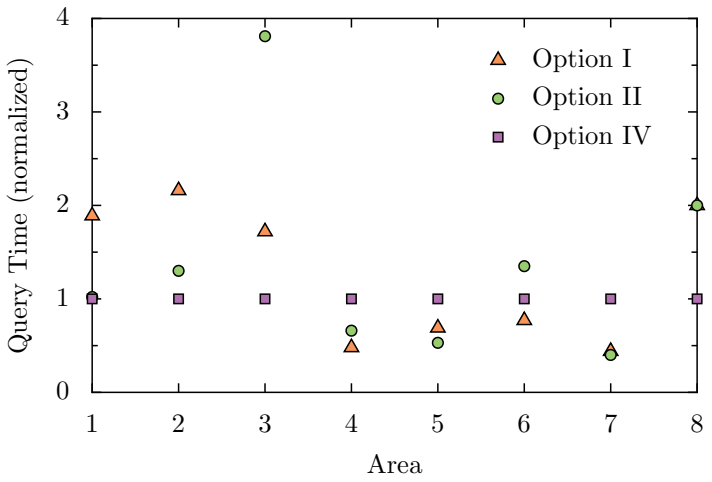


Figure 6: Query time normalized against option IV

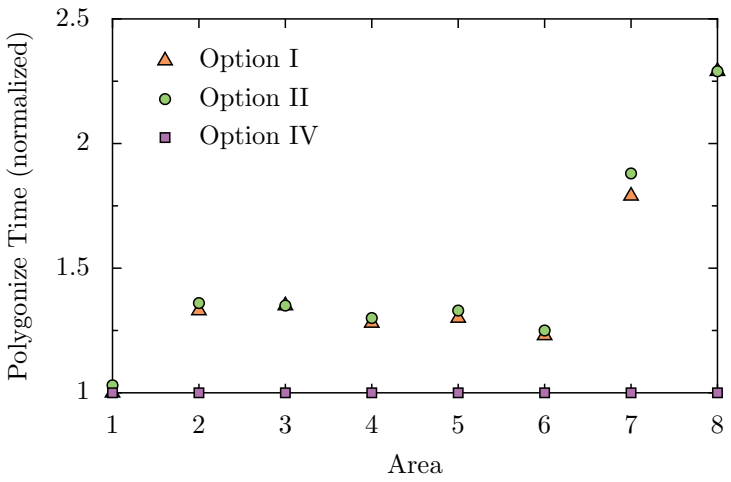


Figure 7: Time for executing POLYGONIZE normalized against option IV

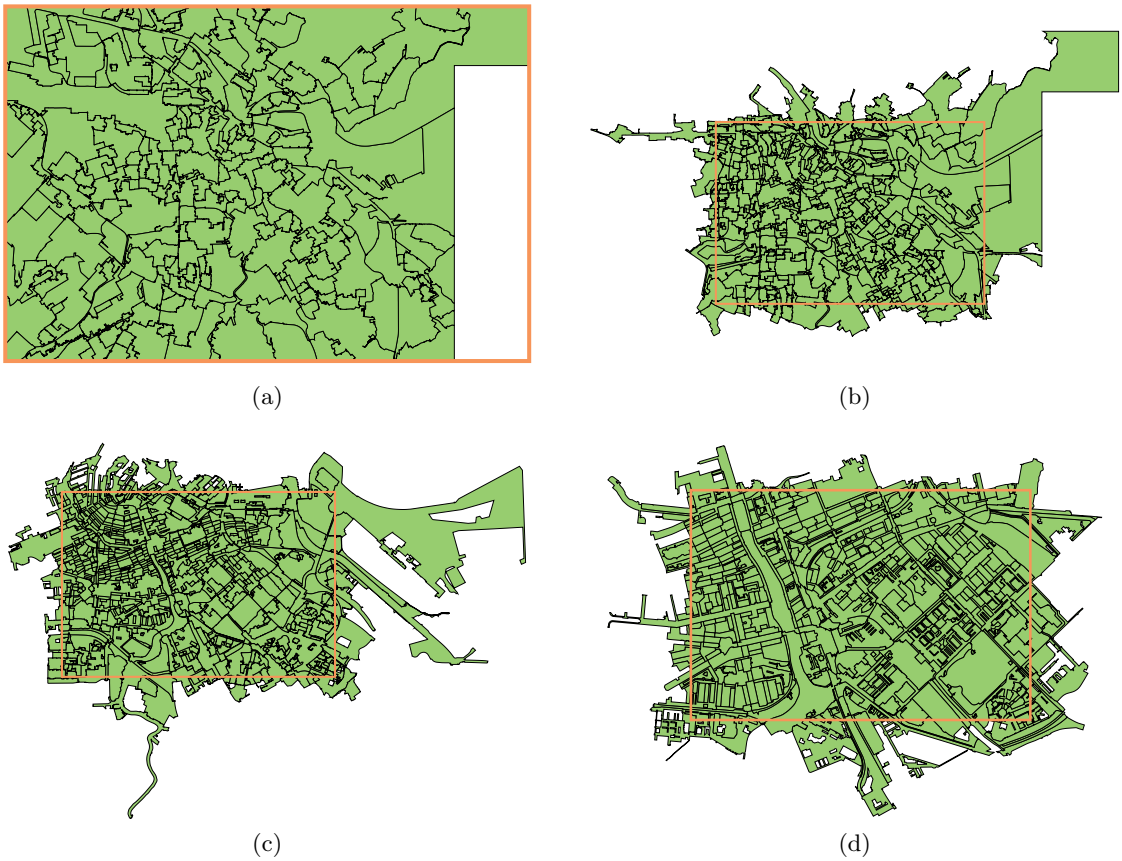
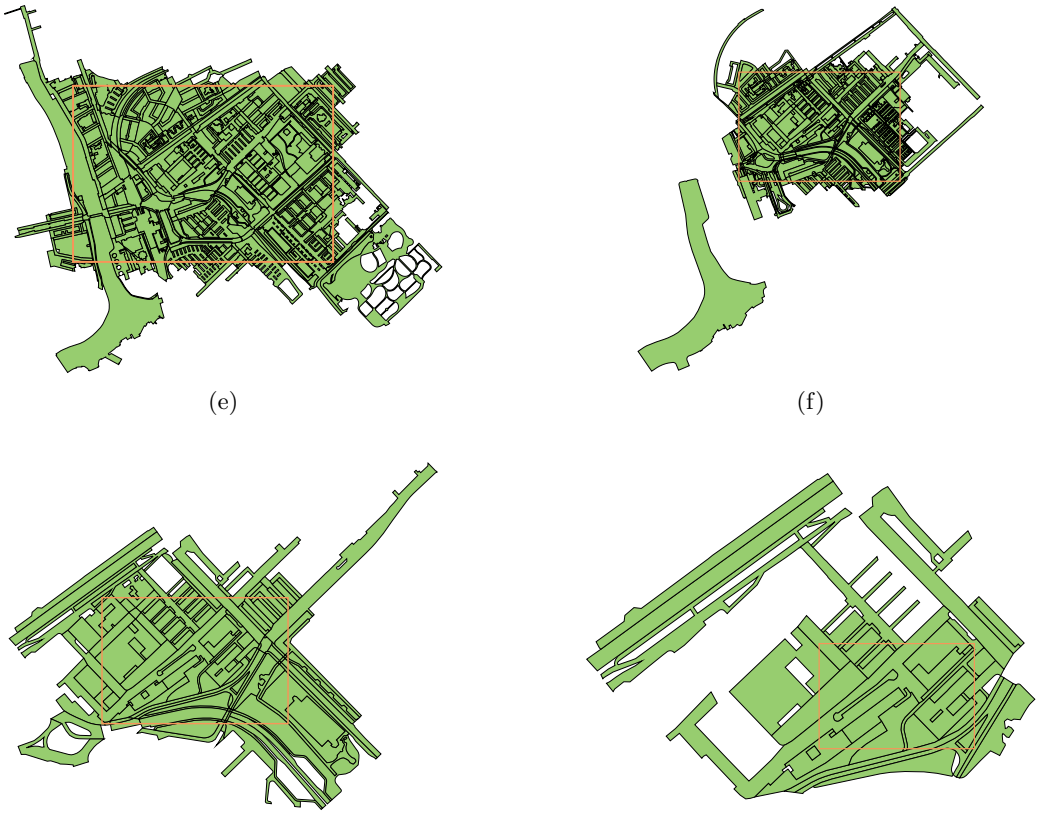


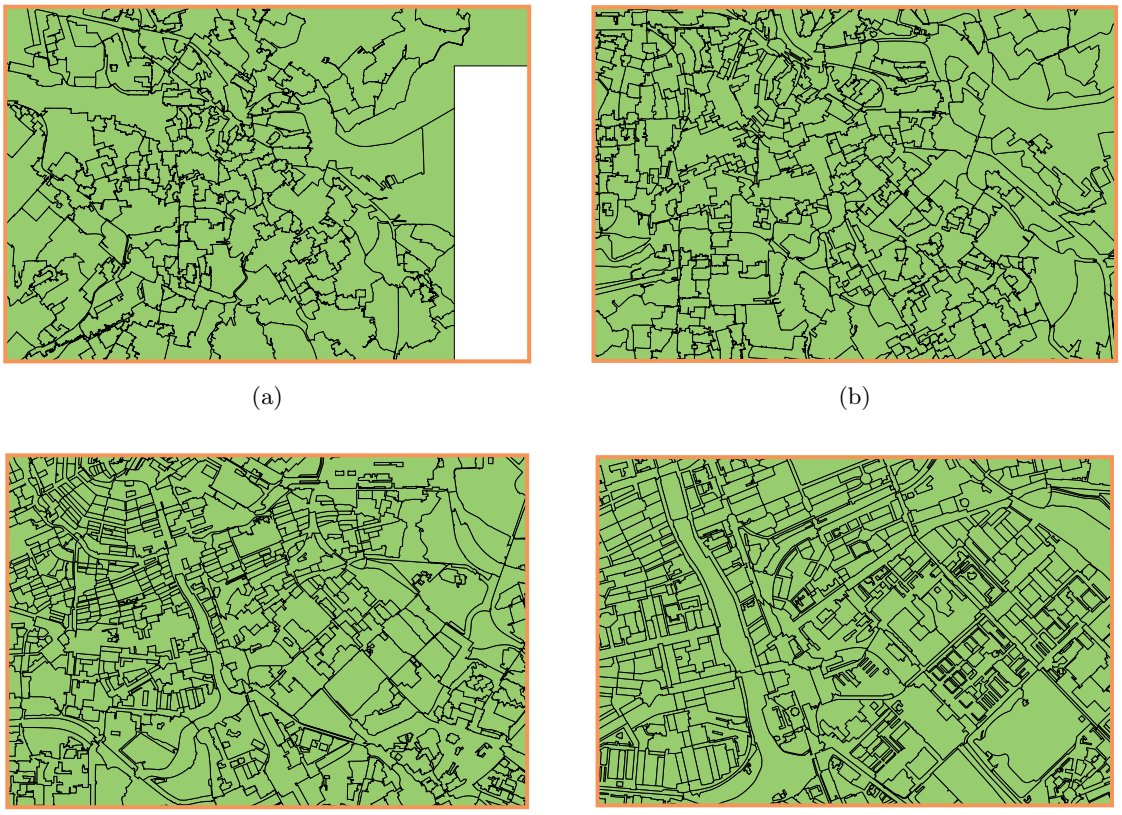
Figure 8: Visualization of Areas. Option I or II used for data retrieval



(e) (f)

(g) (h)

Figure 8: Visualization of Areas. Option I or II used for data retrieval (cont.)



(a) (b)

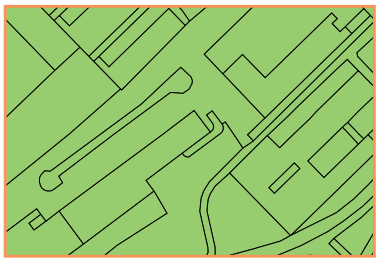
(c) (d)

Figure 9: Visualization of Areas. Option IV used for data retrieval

(e)

(f)

(g)



(h)

Figure 9: Visualization of Areas. Option IV used for data retrieval (cont.)

Corresponding address

Delft University of Technology
OTB Research Institute for Housing,
Urban and Mobility Studies
Jaffalaan 9, 2628 BX Delft
PO Box 5030, 2600 GA Delft
The Netherlands
tel. +31 (0)15 278 30 05
fax +31 (0)15 278 44 22
e-mail: mailbox@otb.tudelft.nl
www.otb.tudelft.nl