Geographic Indexing Mechanisms of Databases for Real-time Visualisation

Patrik Ottoson
Department of Geodesy and Photogrammetry, KTH
S-100 44, Stockholm, Sweden
patriko@geomatics.kth.se

Abstract. Geographic visualisation and other real-time applications require efficient data access. An indexing mechanism, adapted to the characteristics of geographic data, can be used to improve retrieval of geographic data. This paper describes an indexing mechanism based on a method called Ellipsoidal Quadtree, EQT. EQT adapts the indexing structure of a database to the shape of the earth. EQT fulfils the so-called equal-area criterion, which means that all index cells can be constructed having equal area. Quadtree sub-division makes retrieval faster, because the EQT adjusts to the varying number of objects, which occurs in different areas. The mechanism is tested using a road database containing all national roads in Sweden. The achieved performance is good. The mechanism has a potential to work on parallel-connected computers.

1 Introduction

Geographic indexing mechanisms have been used infrequently in GIS, in systems for computer supported cartography or in other visualisation software. As recent as six-seven years ago, rendering of a few thousands geographic objects required processing times in the order of minutes. Today's computers can render millions of objects per second. Transfer and retrieval of data are the main bottlenecks in real-time visualisation today.

This paper describes how a geographic indexing mechanism based on Ellipsoidal Quadtrees, EQT, (Ottoson and Hauska, 2001) can be used in conjunction with a database management system, DBMS. The main objective using geographic indexing is to obtain fast access to geographic data. Geographic indexing is valuable for any kind of visualisation, e.g. GIS, Internet visualisation, and Virtual Reality applications.

A national database, e.g. the so-called Economical map of Sweden (printed map scale 1:10,000), contains approximately 50 million objects. A global database with similar data may contain billions of objects. The Mapquest database (Mapquest, 2001) is a good example. The database contains a large number of features, like roads, hydrography, zip codes, addresses, etc. It covers nearly the whole earth. Mapquest serves about four million unique users per day (Peterson, 1999). The number of users, at Mapquest and similar web sites, grows for each year and we will probably see new categories of users and applications in the future, e.g. mobile Internet/terminals. These terminals could download information about driving directions, route planning, points of interest, and plain maps. Data will not be stored, but used instantaneously. Today, a search on Mapquest takes about 20-30 seconds and just a simple raster map is shown. Loading complex data like vector geometry, traffic information and other attributes demands faster access, to secure traffic safety, user friendliness, etc. Raster data are good sometimes, but it is nearly impossible to use map-matching algorithms on such data. Vector data are indispensable for navigation systems that use map matching. Such systems can be helped by using geographic indexing mechanisms.

2 Indexing Mechanism

An indexing mechanism improves access time to data (Elmasri and Navathe, 1994). Normally, the mechanism can use data structures (e.g. sorted lists, clustering indices, and binary trees) or mathematical algorithms (e.g. hashing). The indices are used to create so called keys and at least one primary key is used for each table.

2.1 Geographic Indexing Mechanism

In this study, we use the MySQL DBMS, which allows SQL-phrases to be embedded in C programs. MySQL is an open source relational database management system. The features of MySQL make it suitable for

accessing databases on a single computer, over a local area network or on the Internet (MySQL, 2000). We use MySQL on a single SGI Onyx, i.e. both the server and the client are running on the same computer.

In order to test the indexing mechanism on real data, a test database containing all road objects from the Swedish road database (VDB) was built. The database contains geometry and attributes for the state-owned roads. The length of a road object varies between some tenths and thousands of metres. The database has a topology built on nodes and links (road objects). A link connects two nodes and consists of a number of three-dimensional points. The database contains approximately 145,000 links, 130,000 nodes, and 2.8 millions points.

A geographic object differs from an ordinary attribute in a database, because it has a three-dimensional position. A frequent geographic query is finding objects within a region (defined as a polygon or an arbitrary-shaped volume). Finding all points inside a region can be expressed in SQL (Structured Query Language):

```
SELECT node_id FROM point_table WHERE latitude>lat_min AND latitude<lat_max AND
longitude>lon_min AND longitude<lon_max;
```

To answer the query all co-ordinates in the database have to be checked (if the DBMS does not have a sorting mechanism on several tuples). Retrieval of all points within a 25x25 km^2 large area takes 10-20 seconds for the queries shown above. Note that this is only the first step in the retrieval process of entire objects. The objects can be retrieved with the following queries:

```
SELECT object_id FROM node_table WHERE nodeid=node_id;
SELECT data_id FROM object_table WHERE objectid=object_id;
```

If the objects in the database have topology, points (co-ordinates), nodes and objects are separated in different tables. For the example above, this implies that for each point a pointer is needed to each node and object. This solution is memory intensive and the total search time may be some minutes for an area containing about 2-3000 objects. A better solution is based on a natural search order - objects, nodes, and points.

A DBMS normally includes an indexing mechanism, but a separate index database can be constructed to support query and retrieval in a geographic database. Such an index database needs to have a number of index cells. Each index cell should point to objects in the geographic database. In general, geographic indexing works like this (compare figure 1):

1. A user formulates a database query to retrieve a selection of geographic objects inside an area.
2. Determination of which index cells are inside the search area and the associated index pointers.
3. The index pointers are used to obtain pointers to objects in the geographic database.
4. Pointers to nodes and objects points are retrieved.
5. Retrieval of nodes.
6. Retrieval of points (co-ordinates).
7. Geographic data are sent back to the application software, where they may be visualised.

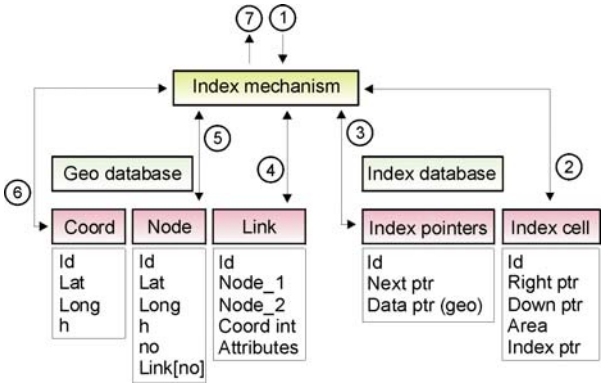

Figure 1. Schema for a geographic indexing mechanism. The figure also shows a simplified definition of the databases, tables, and tuples.

2.2 Global Geographic Indexing

Global geographic indexing is used for global or continental addressing, where the shape of the earth needs to be taken into account. The methods for global indexing, described in the literature, are based either on polyhedral tessellation, adaptive sub-divisions, or map projection techniques.

The polyhedral tessellation is based on sub-dividing a tetrahedron, a hexahedron, an octahedron (Dutton, 1990), (Dutton, 1996), (Otoo and Zhu, 1993), a dodecahedron (Wickman, 1974) or an icosahedron (Fekete, 1990). In adaptive sub-division tessellation of the earth (spherical approximation) is carried out using Voronoi polygons (Lukatela, 1987). One map projection technique is based on the UTM sub-division of the earth (Mark and Lauzon, 1985). The UTM-zones and sub-zones are divided into a regular grid of patches. The patches can be indexed with quadtree technique.

The UTM sub-division is inflexible, because the division has to follow the UTM zones and, consequently, the area of the patches varies. It would be advantageous to have patches of equal area, because then each index cell could hold approximately the same number of objects. This contributes to stabilising the indexing mechanism. A stable mechanism is expected to have approximately the same access time for any object in a database. Ottoson and Hauska (2001) presented a technique that fulfils the equal-area criterion and considers the true shape of the earth, i.e. the ellipsoidal shape. The method is called Ellipsoidal Quadtree, EQT. The method is flexible, because the size of the quadtree base level can be set arbitrarily. The quadtree technique introduced by Klinger and Dyer (1976) and further developed by Samet (1984) can be explained as a tree-like representation of data, where data are decomposed into four new areas for each level.

2.3 Indexing based on Ellipsoidal Quadtrees, EQT

The surface of the earth ellipsoid can be divided into quadrangular facets. These facets are called ellipsoidal quadrangles (analogous to spherical triangles). Four normal vectors span the surface of a facet (figure 2a). The sides of a facet are parallel to the meridians (north/south) and the parallels (east/west). Figure 2b shows how the earth's surface can be divided into facets. The geometric model is based on the geodetic datum WGS84 and the reference ellipsoid GRS80.

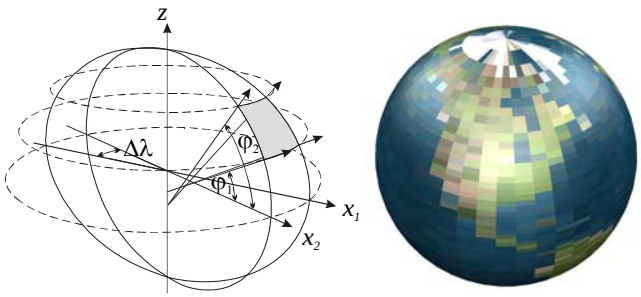

Figure 2. a) Definition of the facets along the earth ellipsoid. **b)** An example of how the earth's surface is divided along the meridians and parallels.

The facets can be used for indexing and addressing geographic data. EQT has a hierarchical structure similar to ordinary quadtrees. The facets can be constructed in such a way that they have equal areas at a certain level in the quadtree. The index mesh created on this level is called the base level.

To fulfil the equal-area criterion either $\Delta\lambda$ or $\Delta\varphi$ has to vary. For most applications, constant $\Delta\lambda$ is preferable. The size of a facet can be chosen arbitrarily, but $\Delta\lambda$ can be set to reach the same dimensions for $\Delta\lambda$ and $\Delta\varphi$ (squared quadrangles), in metres, at the principal latitude. The distribution of data is supposed to be equal in latitude and longitude directions, which means that squared quadrangles are desirable. This helps to stabilise the indexing mechanism.

Computation of the index key requires known and computed parameters: the longitude difference, $\Delta\lambda$, the principal latitude, φ_p, the number of facets with the width of $\Delta\lambda$ for one band, q_λ, the area of each facet, S_f, and the total area of a slice, *S*. The computation of q_λ, *S*, and S_f is described by Ottoson and Hauska (2001). The index key, *k*, is expressed as

$$k = \text{int}\left(\lambda / \Delta\lambda\right) + \text{int}(S / S_f) \cdot q_\lambda \qquad (1)$$

Sub-division of index cells and data is important for geographic databases. Its main purpose is to handle congested index cells. Congestion occurs when the number of objects per index cell is too large. The time for retrieval of object pointers is proportional to the number of pointers for each cell. Sub-division of the base level can be carried out with quadtree decomposition technique. For retrieval of data, only the terminal nodes of the quadtree are considered. The index key, *k*, used for each level in a sub-divided quadtree is expressed as

$$k = \text{int}(\lambda/\Delta\lambda \cdot a) + \text{int}(S/S_f \cdot b) \cdot q_\lambda \quad (2)$$

where $a = 2^{level}$, $b = 4^{level}$, $S = S \cdot a$ and $level = 1,2,...,n$.

3 Results

The results show that a standard relational database management system can be used to store geographic data. It was possible to implement the ellipsoidal quadtree technique, EQT. The performance seems to be good, even if we are using a four years old computer. Using faster processors, internal buses and hard disks would probably enhance performance with a factor 5-10.

3.1 The Quadtree Structure

The principal latitude is set to 62° N, which corresponds to the centre of the database area. In figure 3, the four right-most figures show that the index cells are nearly quadratic at this latitude. The implementation of EQT is based on defining and building the base level first (here level 7). Thereafter, the lower levels in the tree are created.

The size of the index cells, for the base level, was determined automatically. Each index cell was permitted to hold a limited number of objects. We set the limit to 500 objects per index cell. Some index cells were allowed to exceed this amount. Thus, a tolerance level (in percentage) had to be determined as well. The tolerance level was set to 66%. The means that at least 66% of the index cells should hold no more than 500 objects (cf. 3.2). The base level consists of 1128 index cells and the average number of objects per cell is 130. Some cells contain no data and some ten times more.

An ideal quadtree can be created when the number of index cells in the base level is equal to the same power of two (1, 2, 4, 8,…) in each direction. If this is not the case, the lower-level index cells will not be squared (see level 2-4 in figure 3). Consequently, the quadtree decomposition will not be strict. Figure 3 shows projections of the index cells. In the database, the index cells are handled as ellipsoidal curves (see figure 2a). Thus, in order to be correct the straight lines in figure 3 should be curved.

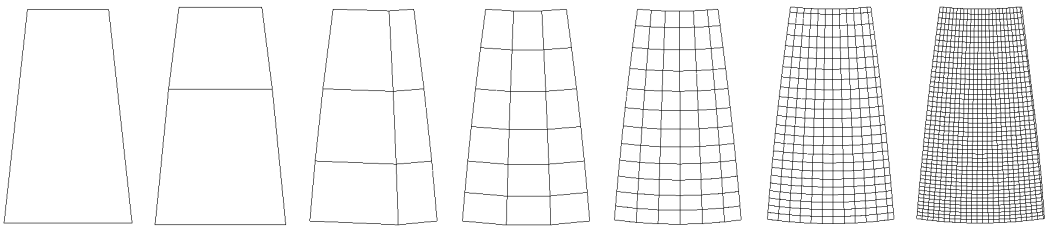

Figure 3. The index cells for seven quadtree levels of the database. The right-most index mesh defines the base level. The number of index cells at each level is 1x1, 1x2, 2x3, 3x6, 6x12, 12x24, and 24x47.

Geographic objects in the database are found by tree-search. The objects are stored in the leaf nodes. The quadtree structure of the index cells in figure 3 is shown in figure 4. The deviation from an ideal quadtree sub-division in figure 3 is illustrated by figure 4, where some nodes in level 2-4 are sub-divided into two nodes instead of four. In real databases, especially for an oblong country like Sweden, this is the typical case. In general, the cells in the north and east will be a bit smaller. Ottoson (2001) described how this discrepancy from a true quadtree can be handled using the programming language Lisp's list handler, called *car* and *cd*r, (McCarthy, 1960).

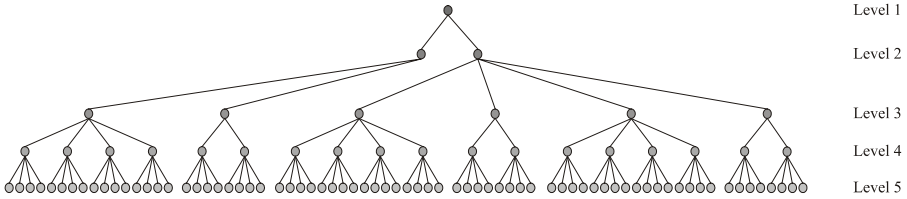

Figure 4. The quadtree structure of the index mesh in figure 3. Only the first five levels are included. The last two levels of figure 3 should be divided in a similar way.

3.2 Sub-division

In the previous section, we described the design of the base level. The limit and tolerance discussed have been set arbitrarily. When not allowing sub-division of the base level, a 95% tolerance may be appropriate. The percentage levels relate to one-sigma and three-sigma levels of a Gaussian distribution curve.

The main idea using quadtrees is to enable sub-division of areas with many objects. Index cells with more than 500 objects were divided into four new cells. In the actual case, it was necessary to make three consecutive sub-divisions. For each new level in the quadtree, the numbers of index cells that were sub-divided were 108, 35, and 6 respectively. Figure 5a presents the index cells for the base level and the sub-divided ones. In figure 5b, cities with more than 50,000 inhabitants are marked. Figures 5a and 5b show that the sub-divided cells are correlated to densely populated areas.

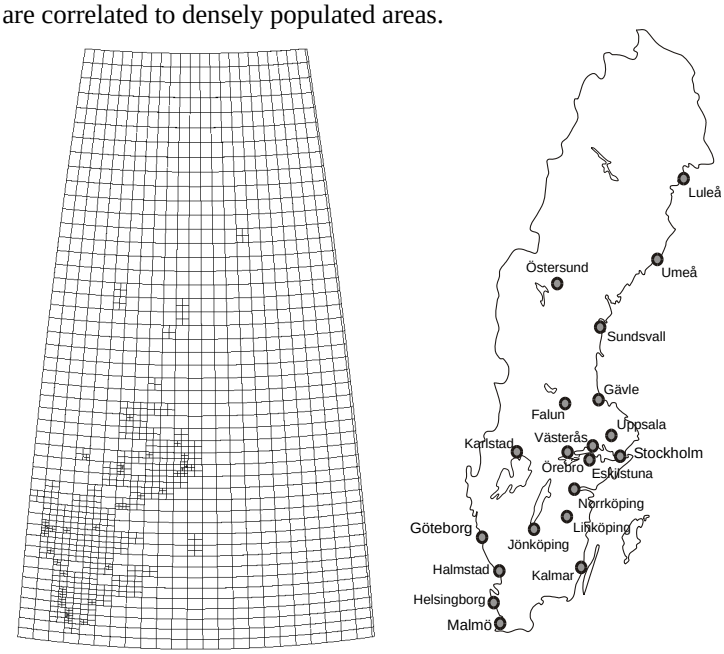

Figure 5. a) Base level and sub-divided index cells. **b)** Larger cities in Sweden. The map has Gauss-Krüger projection, it is therefore not compatible with the index mesh to the left.

The sub-division can be explained using figure 1 as a division of "Area". The area is divided into four equal areas using equation 2. This means that four new index cells are created. The "Down pointer" of the parent cell points to the first child (the southwest cell). The next step is to re-arrange the index pointers. The old index pointers, which refer to the parent cell, are deleted. They are replaced by four series of index pointers. Some "Data pointers" can be found in several series, because some objects extend over more than one index cell.

3.3 Retrieval of Road Objects

Normally, the nodes of a quadtree do not hold any information about geographically adjacent nodes (index cells). There is one exception: all nodes along the same branch have pointers to each other (Ottoson, 2001).

Lets us again consider the query illustrated in figure 1 (find objects within a region). If the queried region can be found inside a single index cell, the tree-search is easy and only performed once. If the queried region affects many index cells, the tree has to be searched several times from the top. In this case, the search can be organised in another way. The identities of geographically adjacent nodes (i.e. in north, south, east, and west) were stored in each node and the search mechanism can directly find the proper nodes, see figure 6. This works also if the index cells are sub-divided.

In figure 6, the search starts by finding the southwest cell. Next the search for adjacent cells is performed by investigating the cells in the north and east directions. The jump from larger to smaller cells goes via the parent cells. The parent cell always points to the southwest child. The search is stopped when the latitude and longitude of an index cell is greater than for the search area. In a program this can simply be implemented as a nested loop. If a node identity is found a second time, it is disregarded.

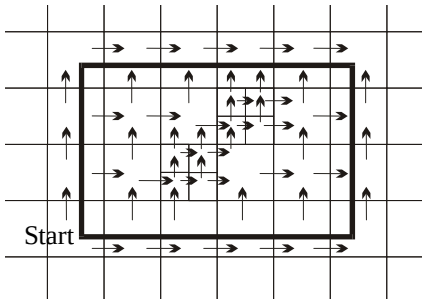

Figure 6. The search order for index cells. The search goes in the north and east directions. Some cells are found twice. Found cells are checked against a list. No cell is retrieved more than once.

A data-retrieving test was carried out to estimate the retrieval times. Four test areas were chosen close to Falun. The test areas are the map sheets: 13F SW, 13F SE, 13F NW, and 13F NE. The geometry for the roads in 13F SE is presented in figure 7b. The areas have the same size (25x25 km^2), but the number of objects varies. The test was carried out for different quadtree levels: one test for the base level and one for the first sub-division level. In the latter case, some index cells are divided because they contain more than 500 objects. Some index cells are not sub-divided. This means that the search areas cover a mixture of larger and smaller index cells (see figure 7a and table 1). Table 2 presents the number of objects per index cell.

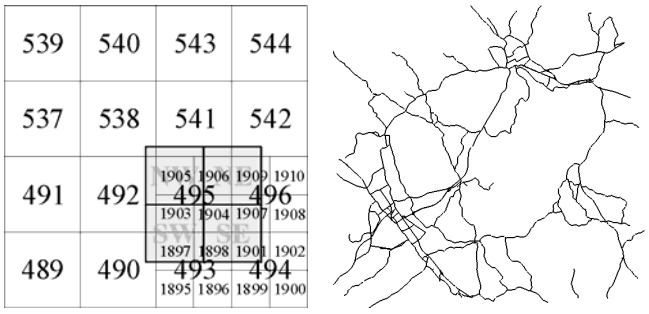

Figure 7. a) A schematic view of the index cells for the base level and the first sub-division level (the numbers show the identities of the index cells). The grey area shows the search areas: 13F SW, 13F SE, 13F NW, and 13F NE. **b)** The image shows the road geometry for the map sheet 13F SE. Only the state-owned roads are shown. The road objects are rendered as complete objects, i.e. no cutting was carried out along the border of the map sheet.

The results show that 300-500 objects can be retrieved in 6-7 seconds and 1000 objects in 9-13 seconds (table 1). The performance depends mainly on the number of objects and points to be retrieved, but also on the number of index cells. When the index cells have been determined, a check has to be carried out to determine if the objects are inside the search area. This has to be done, because the index cells normally contain more data than desired. 13F SE (see figure 7) could be retrieved much faster when sub-dividing the

index cells. Without sub-division, 3143 objects were retrieved; after sub-division 1540 objects were retrieved (see table 2). This means that 2000 respectively 400 objects were omitted by either procedure. Index cell 1898 is actually sub-divided once more (not presented in figure 7a), because it exceeds 500 objects. This increases the performance even more.

Table 1. The index cells for the base level and for the first sub-division level. Performance is measured in seconds. It involves all operations from tree-search to writing the objects to a file. The table also shows the numbers of objects and points for each map sheet.

Map sheet	Index cells (no sub-division)	Index cells (sub-division, first level)	Perf. (no sub)	Perf. (sub)	No. of objects	No. of points
13F SW	490, 492, 493, 495	490, 492, 1897, 1898, 1903, 1904	>7 s.	<6 s.	287	4462
13F SE	493, 494, 495, 496	1898, 1901, 1904, 1907	<13 s.	<9 s.	1127	16026
13F NW	492, 495, 538, 541	492, 541, 1903, 1904, 1905, 1906	>7 s.	>7 s.	455	9679
13F NE	495, 496, 541, 542	541, 542, 1904, 1906, 1907, 1909	<7 s.	<6 s.	289	6009

Table 2. The table shows the numbers of objects per index cell. The index cells between 490 and 542 describe the base level. The index cells 493-496 have been sub-divided.

Index cell	490	492	493	494	495	496	538	541	542	1897	1898	1901	1903	1904	1905	1906	1907
No. of objects	209	198	863	734	902	644	363	357	159	20	590	263	214	292	247	160	395

4 Conclusion and discussion

It can be concluded that building a database using EQT works well. It can also be concluded that EQT can be used to store non-projected data reliably. Implementing the mechanism in software can be tricky. It is advantageous if a DBMS is used for the implementation. Alternatively, it is possible to implement the mechanism using binary files for indices and data. Optimisation of the mechanism using a standard DBMS can be difficult. Integrating the indexing mechanism in a DBMS will probably enhance the performance by a factor of ten.

The limit used to control the number of objects per index cell was determined from experience gained at the National Land Survey. This experience was made using a grid-based index mesh without quadtree decomposition. If the index cells are created with equal areas, some cells will contain more objects than the others. This influences the performance, especially if no sub-division is carried out. The retrieval time depends on the number of objects per index cell. We allowed 34% of the index cells to exceed the limit of 500 objects. The performance can be stabilised using quadtree decomposition. Table 1 shows that sub-division of index cells leads to shorter retrieval times. The sub-division is carried out by dividing the index cells of the base level. The process could be made more efficient, if four adjacent index cells in the base level, which together contain fewer objects than the defined limit, could be amalgamated into one index cell.

In this paper, we only presented retrieval of data using tree-search. Retrieval can be carried out using equation 1 and 2 as well. One problem is that equation 2 is valid only if a total sub-division is performed. This means that all index cells should be sub-divided. In our case, only index cells containing more objects than a defined limit were sub-divided. If the equations could be used, better performance could be expected, because the number of database accesses will be reduced.

In the future, it would be interesting to find out how a large geographic database, based on EQT, could be implemented in a system of parallel-connected computers, e.g. a Beowulf cluster (Beowulf, 2001). We believe that this will significantly enhance the performance.

Acknowledgement

The author is very grateful to Hans Hauska for his engagement in this work and discussions during the preparation of this paper. The work reported in this paper was financially supported by the Swedish National Road Administration and KTH.

References

Beowulf, 2001. http://www.beowulf.com

Dutton G., 1990. Locational Properties of Quaternary Triangular Meshes. In *Proceeding of the 4th International Symposium on Spatial Data Handling*. Zürich, Switzerland, International Geographical Union: 901-910

Dutton G., 1996. Encoding and handling geospatial data with hierarchical triangular meshes. In *Proceeding of SDHS'96*. Technical University of Delft, Netherlands: 15-28

Elmasri R. and Navathe S. B., 1994. *Fundamentals of Database Systems* (second edition). The Benjamin/Cummings Publishing Company, ISBN 0-8053-1753-8

Fekete G., 1990. Rendering and managing spherical data with Sphere Quadtrees. In *Proceedings of IEEE Visualization '90*. San Francisco, CA: 176-186

Klinger A. and Dyer C. R., 1996. Experiments on Picture Representation Using Regular Decomposition. *Computer Graphics and Image Processing* 5: 68-105

Lukatela H., 1987. Hipparchus Geopositioning Model: An Overview. In *Proceedings of Auto-Carto 8*. Baltimore, MD, ASPRS/ACSM: 87-96

Mark D. M. and Lauzon J. P., 1985. Approaches for Quadtree-based Geographic Information Systems at Continental or Global Scales. In *Proceedings of Auto-Carto 7*. Falls Church, VA, ASPRS/ACSM: 355-365

Mapquest, 2001. http://www.mapquest.com

McCarthy J., 1960. Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I. *Communications of the ACM* 3:184-195

MySQL, 2001. http://www.mysql.com

Otoo E. and Zhu H., 1993. Indexing on Spherical Surfaces Using Semi-Quadcodes. In *Proceedings of Advances in Spatial Databases, 3rd International Symposium SSD´93*. Singapore: 510-529

Ottoson P., 2001. Retrieval of Geographic Data using Ellipsoidal Quadtrees. Accepted at *the ScanGIS '2001 Conference*. Ås, Norway

Ottoson P. and Hauska H., 2001. Ellipsoidal Quadtrees for Indexing of Geographic Data. Accepted in *International Journal of GIS*

Peterson M., 1999. Trends in Internet Map Use – A Second Look. In *Proceedings of the ICA 1999 Conference*. Ottawa, Canada: 571-580

Samet H., 1984. The Quadtree and Related Hierarchical Data Structures. *Association for Computing Machinery, Computing Surveys* 16: 187-260

Wickman F. *et al*., 1974. A System of Domains for Global Sampling Problems. *Geografiska Annaler* 56: 201-212