

Database Support for Spatial Generalisation for WWW and Mobile Applications

Xiaofang Zhou Sham Prasher
The University of Queensland, Australia
{zxf, sham}@itee.uq.edu.au

Masaru Kitsuregawa
The University of Tokyo, Japan
kitsure@tkl.iis.u-toyko.ac.jp

Abstract

The need of using spatial vector data for web-based and mobile applications has been increasing rapidly. Vector spatial data is difficult to generalise (that is, to derive a suitable Level of Detail, or LoD, of the data for a given application). In a client-server environment, excessive details of spatial data cannot always be appreciated on the client side, but could consume a significant amount of resources on both the server and the client side, not to mention the extra cost of data transfer. Spatial data generalisation has been investigated extensively in the area of cartography. Most cartographical generalisation algorithms, however, are post-query operations where all the data, including unneeded data, is retrieved from the database for simplification. In this paper, we incorporate spatial data simplification algorithms into query processing within DBMS. This novel approach targets primarily web-based and mobile spatial applications that stand to benefit from early data reduction. Experiments on real data reveal significant performance improvements for our approach.

1. Introduction

Many web sites now offer spatial information, using maps for navigational purposes as well as a background for displaying search results. These 'location-aware' web sites allow the user to interact with a map, to zoom and pan on the client side, to click a spatial object to request further information, or to form a query with spatial conditions [16]. This trend also becomes evident recently for mobile applications. Except some simplest applications where static maps at one or several scales are used, a spatial database management system (SDBMS) is used on the server side to manage spatial data. A map can be generated on demand using the data retrieved from the SDBMS where the spatial objects are individually stored and the geometry of an object is represented as a sequence of points (known as the *vector* data format, as opposed to the raster format such as JPEG). The vector format has a number of advantages over the raster format, such as high accuracy, compactness, and easiness of object identification and manipulation on the client side.

Spatial data is stored in the database with a certain LoD. Obviously, not all applications require the same

LoD. For many web-based and mobile applications, the LoD required can be very low, in particular when the area of interest is relatively large. The *multi-representation* approach physically stores data at different scales in the database [3]. A higher storage overhead and difficulties caused by updates are among the problems related to this approach; however, a more significant problem preventing this approach from being generally useful is that the LoDs required by different applications, or the same application at different stages, can vary. It is known that LoDs should meet the law of *constant information density* [3]. One way to state the law is that the rendered image on a target display should maintain a constant ratio of the number of pixels for objects over the number of pixels for the background; therefore, information is not too crowded visually. Information density, nonetheless, is dependant on query conditions (e.g., the size of the area of interest, and the spatial and non-spatial conditions for object selection in the query), as well as the resolution of the target display. The required LoD of the resultant spatial data from a query can only be determined when all these parameters are known. The number of possible combinations of spatial and non-spatial query conditions and target display sizes can be prohibitively large for any approach based on pre-materialisation. The approach of *multi-resolution* spatial databases, on the other side, attempts to provide support for deriving proper LoDs on-the-fly [2,6]. With the data at the finest available LoD stored in the database, a multi-resolution database system is capable to dynamically reduce LoDs according to applications. Such on-demand deviation of spatial data can be done by a process known as *cartographical generalisation*, which is a process to "derive, from a data source, a symbolically or digitally encoded cartographic data set ... to reduce in scope, the amount, type, and cartographical portrayal of the mapped or encoded data consistent with the chosen map purpose and intended audience; and to maintain clarity of presentation at the target scale" [9].

Generalisation of spatial data is a non-trivial task. It is a step beyond object selection, which can be readily processed by a DBMS. It involves object simplification and several other operations such as tokenisation (replacing geometrically small but semantically important objects, such as telephone boxes and police stations, by a token) and amalgamation (merging similar objects, such

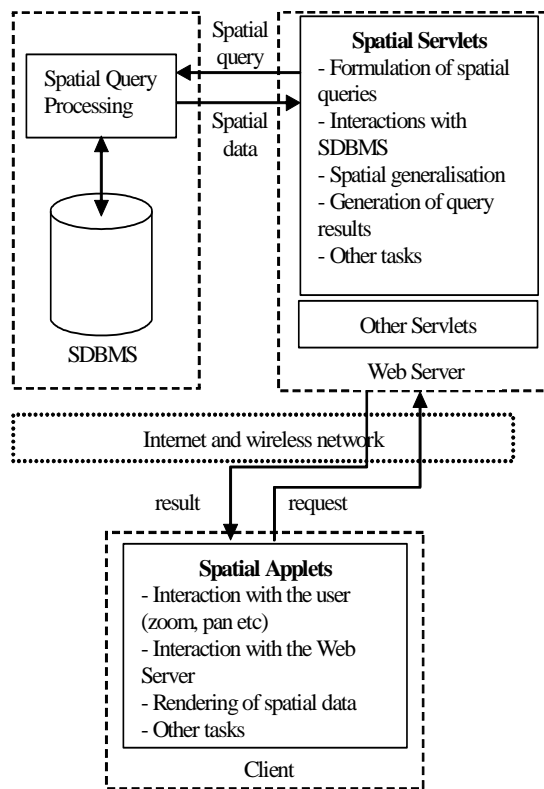


Figure 1. Typical spatial application architecture.

as several neighbouring buildings, into one large object) [9,14]. These operations may lead to an alternation of object geometry, dimensionality (e.g., a building as a polygon is simplified into a point), and sometimes even existence (i.e., an object is not displayed at all).

There are three aspects to be taken into account for a proper generalisation. Firstly, the generalised data should be *correct*. This means that spatial relationships among the objects, which can be topological (e.g., containment and connectivity), directional (e.g., left, right, above and below) or metrical (e.g., relative distance between two objects from any given object), should remain the same or at least similar according to the application requirement before and after generalisation. Generalisation of an object in vector format typically involves a removal of some selected vertices from its geometry. While there are many generalisation algorithms that attempt to make a generalised object bear the maximum geometric similarity to the original object, an important yet very much under investigated goal is to maintain all spatial relationships between an object and all other objects during generalisation. This goal is much harder to achieve than preserving geometric similarity of individual objects. It can only be achieved by treating a spatial data set as a whole.

Secondly, the generalised data should maintain the general *look and feel* of the original data set. This visual

aspect of generalisation not only requires individual objects to maintain their original geometric properties (locations and shapes), but also the overall impression perceived by a human user. The concept of visually significant objects (VSOs) was proposed in [14]. There are three types of VSOs: large objects (including objects of a small area but a large spatial extend, such as rivers and roads), objects in sparse regions (such as a petrol station in a desert region, no matters how small the footprint of the petrol station is), and representative objects in crowded regions (for example, some shops will be selected from a busy commercial area to indicate the characteristics of the area).

Thirdly, the generalisation process must be *efficient* to be useful for interactive applications such as map-based navigation for WWW applications. This performance aspect has been a secondary issue in cartographic generalisation, which concerns, by and large, the production of paper-based maps. Figure 1 is a typical architecture of database-based spatial applications. It is highly desirable to retrieve a minimum amount of data from the database, because data retrieval is one of the most expensive operations in an application, and the benefit of using a smaller set of spatial data will flow to other post-query operations including generalisation operations. Any brute-force approach of retrieving all objects from the database first and then discarding many of them should be avoided whenever possible. These two closely related operations, spatial data retrieval and spatial generalisation, have, unfortunately, not been considered together in the past. There is little work in literature on spatial database support for generalisation; and most work on spatial generalisation does not consider database issues either. In this paper, we present a method to simplify spatial data, as a means of cartographic generalisation, within the DBMS, that reduces simplification time and data retrieval cost. With some new spatial data structures, we demonstrate that one most commonly used spatial generalisation algorithm, the Li-Openshaw algorithm [7], can be achieved using a single SQL query.

The remainder of this paper is as follows. Section 2 gives a brief introduction to the problem of spatial generalisation, with a review of related work in literature. In section 3 we propose to encode vertices in spatial objects such that the simplification process can be processed using simple SQL queries. Experimental results of our database-oriented approach against a currently used simplification algorithm are also presented in this section. We conclude this paper in section 4.

2. Spatial Generalisation

Generalisation has been the subject of intensive research in the area of cartography and computer science

[9]. Besides proposing numerous algorithms for simplifying spatial objects in a way that they bear maximum similarity' with the source objects, this research is also characterised by a strong emphasis on correctness aspects [12]. What makes spatial generalisation so difficult is that there is no unique solution, but numerous constraints have to be taken into account during the process of generalisation. The following four constraints have to hold for spatial data that is to be displayed to humans [12]:

- *Metric constraints*: This class of constraints tries to ensure that all details of spatial data are perceptible for a human observer. Examples of metric constraints are minimal separability, minimal size and minimal width.
- *Topological constraints*: Existing topological relations between source data objects should be preserved by the generalisation process. For example, neighbouring spatial objects should also be adjacent after generalisation. Examples of topological relations include intersection, adjacency, and containment.
- *Semantic constraints*: During the process of generalisation, the semantics of objects may need to be considered. For example, a river flowing parallel to a road should not bear the name of the road after generalisation.
- *Gestalt constraints*: This type of constraint aims at capturing the overall impression of a scenario and is concerned with perceptual criteria such as maintaining the distribution and the arrangement of features.

Noticeably, Gestalt constraints are the most difficult type of constraints to enforce and it is very hard to translate them in terms of operations. Furthermore, they can only be checked after all other types of constraints have been enforced.

As mentioned before, the spatial generalisation process involves object selection, simplification, tokenisation and amalgamation operations. The latter three operations are regarded as post-query processing, after the selection step. By doing so, some objects (and some parts of objects) may need to be retrieved from the database, and then discarded by the follow-on generalisation operations. In order to make spatial generalisation more efficient, it is desirable to find a way to build part of the generalisation process into spatial query processing, such that some spatial objects or some parts of some spatial objects do not need to be retrieved from the database. The problem to let the SDBMS to decide which objects to select is simpler than the problem to let the SDBMS to choose which parts of an object to select. Which objects to select can typically be described as query conditions in SQL. Which parts of objects to select, a problem equivalent to spatial object simplification, is beyond the capability of

most database systems, and is typically done by applying some computational geometry algorithms once the spatial objects are retrieved from the database.

Some preliminary work attempting to incorporate spatial generalisation into query processing can be found in [14]. The concept of VSO proposed in [14], however, still concerns whole objects only (that is, the relative importance of different parts of a complex spatial objects is not considered). VSO does not consider the four correctness criteria we mentioned before. In other words, the VSO approach still follows largely the query-first-generalisation-later paradigm.

The majority of work in post-query generalisation concentrates on line simplification, which generalises spatial objects by simplifying its lines to generate simpler representations while maintaining a good semblance of the original. One of the original is the Douglas-Peucker algorithm [1]. Its effectiveness comes from the selection of points whose vector distance from a baseline, consisting of the first and last points, such that the greatest noticeable change in shape of the whole line is minimised. As the baseline changes new vector displacements are calculated and potentially other points are added to the generalised line, refining it. The process can be set to stop when the generalisation is down to a certain number of points. Other selection criteria are used, such as line length change, angular change, or area displacement. All of these suffer the same drawback from the Douglas-Peucker algorithm when considering image mapping to a screen resolution – topology is not always preserved. Part of the reason for this is that the algorithms only take into account the vector properties of line elements meaning that no consideration is given to how the simplification will affect the look of the data in a different environment. This contributes to the reason that conversions from vector to raster formats error-prone. Although there are a number of refining and smoothing techniques that are designed to handle such after-effects, they are themselves imperfect and still cannot completely prevent the chance for error. So renderings of simplified data can be different from that of the original from a visual perspective.

Another popular algorithm designed by Li and Openshaw introduces a novel idea that can help avoid such problems [7]. It does this by considering the equivalent raster representation of given vector data during computation. The algorithm uses the concept of the *smallest visible object* (SVO) as the finest cartographic detail visible by a human user – in raster format this is a single pixel. A layer of pixels is placed over the vector data. The layer is specified at the resolution the data is being generalised to. Wherever a pixel contains a point the centre of that pixel is returned as a point in the simplified line, representing all former points within that pixel. Its drawback is that not retaining the original points can cause slight deviations, which in the worst case may

produce slight topological errors. Because of this we refer to this method as a near-perfect generalisation (NPG), as opposed to perfect generalisation discussed in [11].

From this overview, it is clear to see that current simplification algorithms focus only on simplifying each individual objects, thus cannot always guarantee the four correctness criteria. These algorithms are used in the post-query generalisation, without any considerations to minimising the amount of data it requires to perform generalisation. There is a need to have an alternative approach, such as the one taken in this paper, to derive an algorithm that is inherently considerate to these correctness factors, and treat data retrieval and data generalisation as integral parts of spatial query processing, rather than completely separate operations.

The algorithms discussed above concern about how a required property is achieved. No performance considerations are given, in particular to query performance when retrieving data from the SDBMS. With very large amounts of spatial data, it is important to perform generalisation over large spatial dataset efficiently. In the context of SDBMS, the maximum benefit of using a smaller amount of data demands that the generalisation is done as part of the database query processing, rather than post-query processing. For the rest part of this paper, we will focus on this aspect

3. Database Support for Simplification

In this section, we study how to retrieve spatial data to achieve NPG using a purposely built data structure. By doing so we show how only the data required by NPG can be fetched from the database. This reduced dataset, then, can be used by any post-query generalisation algorithms for further processing, if necessary.

3.1. Z-values

A spatial index is an auxiliary data structure to support fast search of spatial data [4]. High efficiency of using a spatial index in query processing, in comparison to using the actual spatial data, comes from the fact that an index is typically small in size, hierarchically structured thus most irrelevant data can be pruned effectively, and an index always uses some kind of approximation of spatial objects, which, in many cases, is non-spatial, such as numbers of certain kind. In this subsection, we briefly explain the ideas of a well-known and widely used quadtree indexing. Its application in spatial data generalisation will be discussed later.

The data space D of a spatial data set S is the minimum region containing all the objects in S . Typically, it is represented as a rectangle. D is divided, at its centre, into four smaller rectangles of the same size. The four

quadrants are numbered as 1 to 4 following certain order (e.g., the z-order [10]). The data in S is decomposed according to their location in these quadrants. For point data, a point in S is assigned uniquely into one of the quadrants that encloses the point. For data types with spatial extent, such as lines and polygons, an object can be assigned into more than one quadrant if it overlaps with more than one quadrant. Therefore, D is decomposed into D_1, D_2, D_3 and D_4 , and S is decomposed into S_1, S_2, S_3 and S_4 accordingly. The number of objects inside a quadrant should not exceed a given threshold M . For any quadrant with more than M objects, say D_2 , the quadrant needs to be further decomposed and numbered recursively, into D_{21}, D_{22}, D_{23} and D_{24} (S_2 is also decomposed accordingly, into S_{21}, S_{22}, S_{23} and S_{24}). A rectangle obtained from arbitrary level of decompositions in such a way is called a *Peano cell*. Such decomposition stops when the number of data objects in a Peano cell is no more than M , or a given maximum number of decomposition level is reached. The maximum number of decomposition level is also called the *resolution*.

A Peano cell can be of different sizes, depending on the number of decompositions leading to the cell. Let a Peano cell at level n be D_{z_1, z_2, \dots, z_n} (i.e., obtained from n times decomposition), where $1 \leq z_i \leq 4$. We call $z_1 z_2 \dots z_n$ the z-value of the cell. Note that a z-value can be represented as an integer. Every Peano cell has a unique z-value, and for a given z-value, there is a unique corresponding Peano cell. It is easy to identify the Peano cell of a z-value on the data space D .

A spatial object can be associated with one Peano cell, or several Peano cells if it has spatial extent which crossing multiple Peano cells. Thus, it can be associated with one or several numbers (i.e., the z-values). This mapping forms the base of what known as quadtree-based spatial indexing, which manages a set of numbers (z-values) and maintains a link from a z-value to all objects inside the corresponding Peano cell. The quadtree based spatial indexing has a number of advantages. Firstly, by transforming a two-dimensional object into a set of one-dimensional points (i.e., the z-values), spatial objects can be represented as numbers and therefore can be maintained by a ubiquitous one-dimensional access method such as the B⁺-tree. Secondly, many spatial properties between two Peano cells can be identified by looking at their z-values (i.e., without looking at the actually spatio data, which is usually more expensive to fetch from the database and to compare). For example, the size of a Peano cell, or relative sizes of Peano cells, can be identified from looking at the length of z-values (i.e., the longer is a z-value, the smaller is the cell it represents; and the Peano cells of the same size must have equal lengths for their z-values). In addition, for two z-values z_1 and z_2 , the Peano cell represented by z_2 is nested inside the cell of z_1 if and only if z_1 is a prefix of z_2 .

3.2 Achieving NPG

There are many different ways to store spatial data in a database [5]. Depending on the typical queries a spatial database is designed to support, a spatial object can be stored as an instance of an abstract data type (so points and lines of a polygon may not be individually accessible), or represented by linking line segments and subsequently points. Without limiting the scope of applicability of our method, we assume a general three-tier data structure in a spatial database, namely, objects (i.e., polygons), lines and points. An object is a sequence of line segments; each line segment in turn is a pair of points (the start and end points). A point can be represented by its two coordinates (e.g., longitude and latitude).

Note that the data in a spatial database is always an approximation of the data in the real world to some level of accuracy. Therefore, a point can be represented by the area it occupies, instead of using its coordinates. That is, a point is approximated by the smallest Peano cell enclosing it, and the desired level of accuracy can be achieved by simply increasing the level of decomposition (or the length of the z-values).

In this paper, we propose to use z-values to encode each point in the database. Let an object (a line or a polygon) be represented by a sequence of points $\{p_1, p_2, \dots, p_n\}$. A vertex is represented as $(z, order)$, where z is the z-value for the point, and order records the order of the point in the object (obviously, $\{a, b, c, d\}$ and $\{a, c, b, d\}$ are different for the same points a, b, c and d). As mentioned earlier, NPG can be achieved by placing a layer of pixels over the vector data. A single z-value, which is equivalent to a single SVO in the pixel layer, can then be used to represent any number of points of an object in the same pixel. This leads to the algorithm below, called SQL-Delta, which uses a single SQL statement such that the generalisation is done during the query processing stage. In other words, those vertices which are not part of the simplified objects are not retrieved from the database at all. Note that this approach is not applied to single objects; rather it generalises the entire dataset.

3.2.1 Data Structures

We use z-values to represent each point. That is, a spatial object is represented as a sequence of points, and each point is represented by a z-value. The length of the z-values for all points is the same, to the level of the finest resolution required. Let the length be l , then a z-value of length l can encode a point to the accuracy of 1 cm for an area of $2^l \times 2^l$ cm². For example, for a spatial data set in an area of 2684×2684 km², the z-values need to be integers

of 28 digits to achieve the resolution of 1cm ($2^{28}/10^3/10^2 = 2684$). Note that each z-value digit needs two bits. As a comparison, two double-precision coordinates needs 128 bits.

Using the encoding method above, a simple way to design our database is to simply include the following three attributes:

$zdata(object_id, point_id, z)$

where z is the z-value for the point. For example, a tuple of (1, 2, 123432223322232323432) gives the z-value of the 2nd point of object 1. An object can be reconstructed by ordering the points of that object by *point_id*. In relation to our purpose of spatial data generalisation, it is easy to combine all points in a *data pixel*, which is pixel in an area in D corresponding to one pixel in the target display device, into one point, using the following SQL query (where we assume a data pixel is of the size at level l Peano cell. See the algorithm below for detail).

```
SELECT object_id, substring(z, 1, l), min(point_id)
FROM zdata
GROUP BY object_id, substring(z, 1, l)
```

A where-clause can be used to add other conditions to choose spatial objects. The point index can be that of any point of the object in the data pixel (we use $min(point_id)$ in the query). Note that attribute z is treated as a string type here, which makes the query easy to understand (this is not necessary as we see later). The method to reconstruct the simplified object is identical to that to reconstruct an object without simplification (i.e., to connect to the next point), though here the indexes of the two points to be connected may not necessarily be consecutive any more.

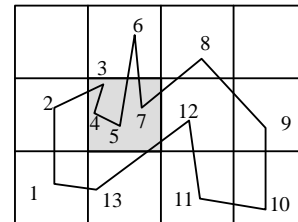


Figure 2. Object line segments.

Such a naïve method to represent all points in the same data pixel by a single point, however, does not consider the case when the line of a spatial object may enter into the same data pixel multiple times (see Figure 2). Doing so will lose information regarding certain line segments. To illustrate, in Figure 2 if point 3 (or 4 or 5) is chosen to represent the shaded pixel then the line segment between 7 and 8 is lost, and 6 will be connected to 8 incorrectly. If 7 is chosen then 3, 4 and 5 are lost, connecting 2 directly to 6 in the point order sequence, which can result in a poor

simplification of the object. This problem can be solved by checking each line segment. For any line segment, if both end points are inside the same data pixel, then an arbitrary point in the data pixel can be taken as the representative point of the line segment, as is done with the Li-Openshaw algorithm. However, if one end point is outside the data pixel, the other end point must be selected. In the shaded pixel of Figure 2, point 4 can be discarded, whereas 3, 5, and 7 must be kept. Once all line segments that satisfy this condition are chosen, simplification has been achieved. While this can be done using the simple table above, it will be a complex join query with substring functions to find which point connects to which, thus the performance is a problem. The condition to check if two z-values (of a line segment) are within the same data pixel using SQL functions is

$$\text{substring}(z1, 1, 15) \lt \text{substring}(z2, 1, 15)$$

here it is assumed the data pixel size of level 15 Peano cell. Notice that an index on attribute z in the table will not be useful for substring functions, thus the query is very inefficient. Because of this performance issue we propose to use the following table design:

$$zdata(object_id, point_id, z, ze, delta)$$

where two additional attributes ze and $delta$ are added. Attribute ze is the z -value for the point z links to in the object, and $delta$ is a derived attribute defined as $|z-ze|$, where both z and ze are represented as decimal numbers. The $delta$ value is important because determines the degree to which both z -values differ. For example given the z -values 122344 and 122322, the $delta$ would be 22. In this example, we can infer that both z -values are very similar and only differ at the last 2 levels of decomposition because the $delta$ is 2-digits long.

3.2.2 SQL-Delta Algorithm

Now we can give the detail of our algorithm that incorporates the Li-Openshaw algorithm into a single SQL query based on the data structures discussed above. In the description below we also explain how a suitable data pixel size can be determined and used. The algorithm consists of the following four steps.

Step 1: *establish the Peano cell size for a pixel.* Given the area of interest in the data space (as a rectangle) and the display size in terms of pixels (also a rectangle), the size of the area in the data corresponding to a pixel space (i.e., the size of data pixel for this query) can be determined. From this, the level of decomposition that leads to a Peano cell whose size is the largest one smaller than a data pixel. Let the level of decomposition be l .

Step 2: *choose line segments that have both endpoints in different data pixels.* If a line segment's two endpoints occur in the same data pixels then its detail is lost within that data pixel. If both endpoints occur in different pixels, then the line segment is kept. Because endpoints are shared between consecutive line segments, we only need to retrieve the first point of line segments. Selected points will therefore be joined to the next encountered point that occurs outside their data pixel. For example in Figure 2, point 2 can be joined directly to point 5. Simplification can then be implemented using the following SQL query:

```
SELECT object_id, point_id, z
FROM zdata
WHERE delta > c
```

where $c = 10^l$. Of course, other application-specific conditions can be used in the where-clause for object selection. The reason for using constant c is that the resolution at which data is to be simplified is equated with the level of decomposition l . Therefore all that need to be done is to check which line segments have $delta$ values that exhibit differences between both endpoints significant enough to be noticed at l . For example if data is to be simplified for a 1024×768 monitor, which is the same as $l=10$, then all $delta$ values with more than or equal to 10 digits (10^{10}) represent the line segments with endpoints in different pixels on the monitor. The major inherent advantage of the $delta$ field is it can be used to derive simplifications for multiple resolutions even though it holds a pre-computed value (thus, can be indexed). This is supported by the fact that the value for l is issued during run-time as part of the query-condition. Unlike the SQL query used earlier, there is no `group_by` clause. This is because the simple condition, " $delta > c$ ", automatically excludes all those line segments whose both end points are in the same data pixel.

Step 3: *object reconstruction.* Once the simplified data has been retrieved all that is left is for the objects to be rendered. Since line segments are already ordered in the database by their respective object IDs, they do not need to be sorted on the client side to be rendered.

In summary, we discussed database-level support for NPG. Those vertices and line segments that will not be part of any generalised objects can be eliminated in the SQL query to retrieve data from the database. Furthermore it is important to avoid costly join or value-manipulation operations, such as $substr(a, v, l)$ during run-time in order to achieve efficient processing. In our approach this is done by storing line segments and using a pre-generated $delta$ value. While other generalisation operations may still need to be applied on the data retrieved from database queries, the queries above return a smaller set of intermediate data,

thus those post-query processes can be more efficient. The benefit of using the techniques introduced in this section is particularly large when a large amount of data will be removed by the generalisation operations. Such scenarios are common for web-based and mobile spatial applications.

3.3 Performance Comparisons

So far we have established a basis and methodology for performing NPG within a database. In this section we illustrate performance of SQL-Delta simplification against that of Li-Openshaw simplification. We used a vector point dataset on Californian topographic network data on transportation, communications and utilities. The dataset contains 61,193 points. For the Li-Openshaw method data was stored as complete polygons/polylines using the Oracle spatial data cartridge. An object identifier is identified with each polygon/polyline and is indexed using a B-Tree. For the SQL-Delta approach we used the table scheme given in the previous section, decomposing the data to 21 levels. An B-Tree index was also defined on the object identifier column as well as on attribute *delta*.

Testing the Li-Openshaw approach involved first retrieving all of the data from the database using an SQL query, then simplifying it. Under SQL-Delta we only needed to issue the query shown in the previous section. Both methods were tested for various screen resolutions (i.e. for different devices). Performance times, and data reduction during simplifications are as follows:

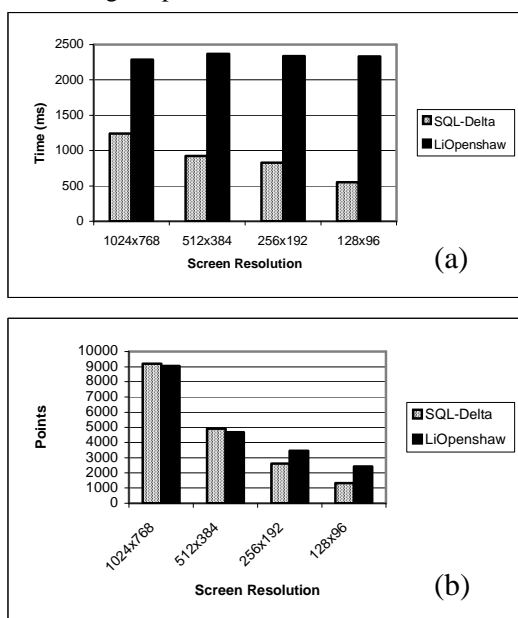


Figure 3 – a) Time to simplify and render to screen the data over different resolutions and b) The number of points produced from simplifications over different resolutions.

Simplification times show drastic differences, the SQL-Delta approach outperforms the Li-Openshaw algorithm over all resolutions. Under the former, time to simplify also decreases as resolution becomes smaller, whereas the Li-Openshaw time remains predictably constant. The reason for this is SQL-Delta needs only retrieve data that is part of the final simplified image, hence speeds up as data reduction increases. The Li-Openshaw however, must always retrieve all of the data before operating on it, increasing data transfer dramatically and thus slowing the simplification procedure. Given that the final simplifications for the given resolutions use less than 16% of the original data this overhead should be avoided.

Figure 3b shows that both methods are generally comparable in terms of data reduction. SQL-Delta retrieves and outputs line segments whereas Li-Openshaw outputs single points. Reductions are comparable because different line segments contain the same points, hence some level of redundancy is allowed. However, the number of line segments shown for SQL-Delta is also how much data is retrieved from the database, whereas Li-Openshaw always retrieves all 60k points. Differences in these figures occur mainly because the raster layer that the Li-Openshaw uses to simplify is not always perfectly aligned with the grid that results from decomposing the data space to code z-values. Note that SQL-Delta results in around the same number of points at 256x192 to that of Li-Openshaw at 128x96. If, for some reason, a more detailed image than the SQL-Delta 128x96 results is desired, the next higher resolution simplification could be used instead. From the results shown, this would still provide a very large time saving.

3.3 Discussions

So far we have discussed database issues for supporting NPG. Those discussions offered low-level technical details. We will give a more general view on database support for spatial data generalisation, of which what we discussed before is only a part.

For a given spatial query, two additional sets of information are needed to derive implicit visual constraints. The first set of information is about the size of data space, how a spatial index is created (such as the origin, orientation and maximum decomposition of Peano cells) Such information can be found in the data dictionary of an SDBMS. The second set of information concerns the rendering device the user is using for the query, mainly in the form of $x \times y$ resolution numbers. With such information, the minimum Peano cell corresponds to a pixel on the rendering device can be calculated. Now it is ready to enhance the query given by using the techniques introduced in this section to filtering out the parts of objects which can be removed without altering the visual

perception once the query results are displayed on the given rendering device.

We have not considered other generalisation operations, such as object selection, tokenisation and amalgamation. Some of these operations can also benefit of using the z-value structure we proposed in this paper. Tokenisation, for example, requires only the location of an object (not its geometry). Thus, the z-value for any vertex of an object to be tokenised is sufficient. Object selection and amalgamation, on the other side, may need indexes at object level (instead of line segment level as used in this paper). We studied the methods of supporting object selection and object amalgamation using z-value indices in [14,15] respectively. However, it is still an open problem on how to adopt a holistic approach to consider all these operations. The use of z-value data structures for other traditionally costly operations such as spatial joins [13] and multi-resolution spatial data mining [8] is also an avenue for future work.

4. Conclusions

In this paper we have presented a new approach to using spatial data within a cartographic and mobile computing context. The approach involved encoding spatial coordinates as z-values. This scheme allows us to further exploit SQL syntax to issue queries that simplify the data in a way identical to the Li-Openshaw algorithm during the query process. This method holds the major advantage of drastically reducing the amount of data that is retrieved from the database, and hence that is subsequently transmitted over a network. This last issue specifically targets mobile and other online situations, such as the WWW, where network bandwidth restrictions are a major concern and response time is critical. Test results have shown this approach only requires from 20% to 50% of the time to simplify than current 'brute-force-retrieval' methods.

ACKNOWLEDGEMENT: The first author's work was partially conducted during a visit to the University of Tokyo supported by a Research Fellowship funded by the Telecommunications Advancement Organisation of Japan.

References

- [1] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", *The Canadian Cartographer*, 10(2), 1972, pp. 112-122.
- [2] G. Dutton, "Digital map generalisation using a hierarchical coordinate system", *Proc Auto Carto 13*, Seattle WA, 1997, pp 267-376.
- [3] A. U. Frank and S. Timpf, "Multiple representations for cartographical objects in a multi-scale tree – an intelligent graphical zoom", *Computers and Graphics*, 18(6), 1994, pp. 823-929.
- [4] V. Gaede and O. Güther, "Multidimensional Access Methods". *ACM Computing Surveys*, 30(2), 1998, pp. 170-231.
- [5] R. H. Güting, "An introduction to spatial database systems", *VLDB J.*, 3(4), 1994, pp. 357-399.
- [6] C. B. Jones, D. B. Kinder, L. Q. Luo, G. L. Bundy and J. M. Ware, "Database design for a multi-scale spatial information system", *International J. of GIS*, 10(8), 1996, pp. 901 – 920.
- [7] Z. Li & S. Openshaw, "Algorithms for automated line generalization based on a natural principle of objective generalization", *International J. of GIS*, 6(5), 1992, pp. 373-389.
- [8] X. Lin, X. Zhou, and C. Liu, "Efficient Computation of a Proximity Matching in Spatial Databases", *Data and Knowledge Engineering*, 33(1), 85-102, 2000
- [9] R. B. McMaster and K. S. Shea, *Generalization in Cartography*. Association of American Geographers, Washington, D. C., 1992.
- [10] J. Orenstein and T. H. Merret (1984). "A class of data structures for associative searching", *Proc. 3rd ACM SIGACT-SIGMOD Symp. on PODS*.
- [11] S. Prasher, "Eerfect line simplification for visualisation in digital cartography". *Proc. 6th IFIP Working Conference on Visual Database Systems*, 2002, pp. 203-218
- [12] R. Weibel. "Generalisation of spatial data", in *CISM Advanced School on Algorithmic Foundations of Geographical Information Systems*, 1996, pp. 346-367.
- [13] J. Xiao, Y. Zhang, X. Jia, "Clustering Non-uniform Sized Spatial Objects to Reduce I/O Cost for Spatial Join Processing", *Computer Journal*, 44(5), pp.384-397, 2001
- [14] X. Zhou, A. Krumm-Heller and V. Gaede, "Generalisation of spatial data for Web presentation", *Proc. of the 2nd Asia Pacific Web Conference*, 1999, pp115 – 122.
- [15] X. Zhou, D. Truffet and J. Han, "Efficient Polygon Amalgamation Methods for Spatial OLAP and Spatial Data Mining", *Proc. of SSD99 (LNCS 1651)*, 1999. pp. 167-187.
- [16] X. Zhou, J. Yates and G. Chen, "Using Visual Spatial Search Interface for WWW Applications", *Information Systems*, vol. 6, no. 2, pages 61-74, 200