STREAMING OF COMPRESSED MULTI-RESOLUTION GEOGRAPHIC VECTOR DATA

Johan Persson
Idevio, Anders Carlssons Gata 7, 417 55 Göteborg, Sweden, johan.persson@idevio.com, Tel: +46 31 7790960

Abstract
There have been compressed, streaming, multi-resolution formats for image data available for several years but there was no such format for geographic vector data until recently. We took on the challenge to create such a database format starting in 2001 and the format is now known as RaveGeo. This paper describes the concept, some of the lessons learned during the design and implementation of RaveGeo and some results from using it.

INTRODUCTION
Applications that use geographic vector data today face a lot of problems. Databases are sometimes huge and therefore cumbersome to deal with. The VMap1 database from NIMA for instance is approximately 50GB in VPF format, which makes it almost unmanageable in normal GIS systems. Applications also have to deal with limited memories, especially in small devices, and limited bandwidth in wireless or Internet solutions. Common makeshift solutions include using databases with lower resolution or just using a part of the database. It is also expensive to prepare data for an application. The standard way today is to prepare a number of more or less separate databases for different scale ranges.

The RaveGeo format is designed to solve or reduce these problems. The goal for the format was to make geographic vector data available efficiently in wide scale ranges. It should also be possible to access the data over wireless networks or the Internet.

There are similar systems to RaveGeo for other types of data. For image information there are at least two commercial systems available, ECW from ER-Mapper and MrSID from LizardTech. There has been quite a lot of research on progressive transmission of triangular meshes, most notably by Hoppe H. (1996). This method is implemented in Microsoft DirectX.

For geographic vector information Bertolotto and Egenhofer (2001) describes a system similar to ours.

THE RAVEGEO SYSTEM
Design goals
The goal was to create a concept that allowed for fast access of geographic vector data both when reading from a local disk and via networks. It should be a system that could be integrated into map software and that would work with available geographic data. The concept should also simplify data preparation and management.
To make it fast over low-bandwidth connections, compression of the data was necessary. This also had a positive effect by simplifying distribution of databases.

In order to increase the perceived speed a progressive approach was desirable. This lead to a multi-resolution database, which in turn required automatic generalization in the preparation stage. An advantage with that solution was the extended usable scale range for the data. We observed that a good generalization should keep the perceived information density fairly constant when zooming. When a small area was viewed in large scale there should be approximately the same amount of information as when zoomed out and a larger area was viewed with a lower resolution.

This technique is most useful for large databases that are really difficult to handle. Therefore a requirement was added for handling databases in the order of 100GB.

To simplify integration into applications a fairly conventional object model was chosen, consisting of the feature types polygons, lines and points. Underneath they are connected with topological links and also links between levels. The topology was required for doing generalizations but also increased the compression by sharing of geometry.

The approach was to create a good enough generalization quality. The goal was not to create “printing quality”. Interaction and dynamic features implemented in software, like tool-tips, can compensate for a quality that is lower than of a printed map that does not have these possibilities. Also a simple approach for keeping consistency during generalizations was chosen; if a generalization produce an inconsistency then avoid doing that generalization. This means that the data at some locations might not be generalized as much as wanted but this has shown not to be a big problem in real cases.

The concept

RaveGeo works conceptually much in the same way as the raster counterparts ECW and MrSID. Source data in standard geographic vector formats is converted in a batch process by a compiler to the compressed multi-resolution format. In runtime there is a server that reads the compiled data and distributes it over the Internet to reader components that unpack the data so that it can be used in applications, see Figure 1.
Streaming of compressed multi-resolution geographic vector data

The Compiler reads the source data and generalizes it into multiple resolutions and compresses it using a lossless algorithm. The Server sends increments of data to a client Reader that puts them together so that the application can use the data like it would use the original data, potentially with a lower resolution.

Seamless data
RaveGeo uses a tiling approach where each tile contains approximately the same amount of data. This means that the tiles have different geographic size; they are large where data is sparse and small where data is dense. Data is clipped at tile borders and reconstructed when needed. There are arguments for and against clipping but for practical use clipping is not possible to completely avoid. Reading and rendering a large polygon like a country when zoomed in to a small part of it is way too expensive if the whole polygon is returned. RaveGeo returns the parts stored in overlapping tiles and connects them if requested.

There is a separate tiling for each resolution level, see Figure 2. Two or more neighbor tiles can be merged into one tile in the lower resolution to keep the stored data volume of the tiles approximately constant. Keeping same tiling through levels would imply a great overhead by having to read very many very small tiles in a zoomed out mode. Typically a tile at the lowest resolution covers a 50 times larger area than a tile at the highest resolution.

![Figure 2: Tiling with tile size controlled by data density.](image)

Multi-resolution
The compiler generalizes the data stepwise according to the following scheme:

\[
\{d_i, A_i\} = g(d_{i-1}) \quad 1 \leq i \leq k
\]

Here \(d_i\) is the dataset at resolution level \(i\). Resolution level 0 is the original data. Each higher level contains data with half the resolution. \(k\) is the level with the lowest resolution, i.e. the highest level value. The function \(g\) transforms a dataset to the next lower resolution and also produces the difference between the two resolutions, here denoted by \(A_i\). This process continues until the lowest required resolution level \(k\) is reached. Then only the lowest resolution and the difference data is stored. Reconstructing a dataset at level \(i\) works in the opposite way using the inverse function \(g^{-1}\):

\[
d_i = g^{-1}(d_{i+1}, A_{i+1}) \quad 0 \leq i < k
\]
This means that the generalization function $g$ must be reversible using the stored difference data $\Delta_i$. This is however not a very hard constraint since all functions can be made reversible given enough extra information, for instance the complete dataset before the function was applied. The challenge is to find functions that minimize the difference data that needs to be stored and that can be applied efficiently, especially in reverse. Minimizing the difference data also means to maximize reuse between levels. A resolution level should build as much as possible on the next higher resolution. This has also influenced the design in that generalization operators that change an object much from one level to another have been avoided.

The biggest challenge, however, was to find algorithms that work locally, i.e. that does not need the complete dataset to work with at once. This would for obvious reasons be impossible for large datasets. Unfortunately this constraint in some cases reduces the generalization quality but the quality has shown to be good enough for most cases.

The generalization function consists of a set of generalization operators described below.

**Selection**
This operator removes objects in lower resolutions that are considered insignificant in that resolution. This means than an object that first appears in level $i$ is stored as a complete object in the difference information for level $i+1$. The selection is based on the attributes of the objects and expressions defining inclusion or exclusion are written in the configuration to the compiler. Note that selection of objects based on size, for instance removal of small lakes in lower resolutions, are considered to be aggregations into void, see below.

**Thinning**
The thinning operator reduces the number of breakpoints in lines. The removal of breakpoints is limited by a maximum allowed error, defined in resolution units. Typically a maximum error of a few resolution units of the level is allowed. Removals that would make the line self intersect or change the topology are prohibited. Thinning is applied locally in each tile. Points at tile borders are locked. This is a compromise between locality and generalization quality but the locked points have shown to be hardly noticeable.

**Aggregation**
The aggregation operator merges neighbor objects in lower resolutions. The effect is that the number of objects is reduced. An object is merged into a neighbor if its size is below a specified size in that resolution or it has identical attributes with the neighbor. A small polygon which has no neighbor along most or its entire border can be removed. This is seen as aggregating to void. Small holes in polygons can in the same way be removed.

Many datasets contain more or less independent sets of polygons that might overlap. An example of this is a dataset that contains woodland and swamp polygons. These often overlap but also often share edges. They should be generalized together but not aggregated. This is controlled in the compiler configuration. It is also possible to prioritize aggregation so that for instance an open area rather aggregates to a cultivated area than a lake.

For polygons that cross a tile border special rules apply. RaveGeo makes an analysis where all neighbor tiles are included before deciding on cross tile polygon aggregation.
Model generalizations

It should be pointed out that all generalization operators described here have the goal to create a description of the world with lower resolution, not to create a cartographic representation suitable for a particular scale. Cartographic generalization (like Harrie, 2001) is in general scale dependent and not suitable for this kind of multi-resolution database formats. Even so there are ideas from that area that might be useful. RaveGeo is a geographic database in contrast to a cartographic database. Since the data is geographic it is possible to use it for calculations and analysis. This puts however higher demand on the visualization software when creating an appealing map presentation. We have noticed that efficient label placement is the most wanted cartographic function.

Resolution

A RaveGeo dataset has a base resolution with which the highest resolution is stored. The resolution is defined as the shortest representable distance. Each next lower resolution level has a doubled distance as the resolution. A resolution unit \( r_i \) for level \( i \) is defined as this distance.

This resolution acts as a reference resolution from which generalization tolerances are defined. For instance the acceptable deviation from the correct line between endpoints is often larger than one resolution unit. The accepted deviation often lies in the range 2 to 6 \( r_i \). To decide the maximum area of polygons that are considered insignificant and can be removed or merged into a neighbor one can look at how cartographers does generalization manually. Lakes have to be quite much more than a dot on a map to qualify for inclusion. This often ends up in setting the area aggregation tolerance to something between 30 and 150 \( r_i^2 \).

In a presentation the requested resolution is normally related to the pixel size. It is useless to request a better resolution than can be visualized. It is often enough to request a resolution that corresponds to 1-4 pixels.

A dataset is often generalized into about 10 levels. This gives the lowest resolution \( r_{10} = r_0 \times 1024 \). An observation is that the data volume in level 10 should be reduced to one millionth of the original data if the perceived information density should be constant since it is proportional to the inverse of the square of the resolution unit. Another observation is that the thinning operator used only can reduce the data linearly to the resolution while the behavior of the aggregation operator is more complex and depends on the distribution of object sizes. An aggregation leads to fewer objects but not necessary to less geometry since only the shared geometry is removed. The selection operator depends completely on the nature of the data and often shows stepwise behavior like when streets are removed and only roads remain. The conclusion is that working together these operators have the chance to reduce the data to a volume proportional to \( 1/r_i^2 \). If for instance selection and aggregation removes half the information, thinning could half the remaining geometry that in total gives a reduction to a quarter.

Streaming

Streaming of data is efficient if data can be sent in increments and the client can start using the data before all is received. Compression is also naturally useful for efficient streaming. For geographic data both a partitioning in space and a partitioning in resolution is
necessary. For RaveGeo the resolution levels represent the partitioning in resolution and the tiling represents the partitioning in space.

As the lowest resolution levels are sent first, so-called progressive presentations are possible. This means that the user first sees a lower resolution version of the map while the higher resolutions are sent. The presentation is updated as higher resolution levels arrive until the requested resolution is received. This is particularly useful for low-bandwidth connections and means that the user in most cases can continue to work while data is transmitted. When the connection is reasonably good the updates are so fast that the progressive effect is hardly recognizable.

RESULTS

Tests have been performed with three different datasets. One uses land-use data from Delaware, a pilot to the US National Map, and consists of a complete coverage of polygons without overlap. The next one comes from the same source but contains road data. It is a complete road network with a mixture of streets and roads. The last dataset is the boundaries part of VMap0 from NIMA. The first two datasets are originally stored in ESRI Shapefile format (ESRI, 1998) and the last one in Vector Product Format (Department of Defense, 1996). Table 1 below summarizes the results.

Table 1: Test on different datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Source Format</th>
<th>Size (KB)</th>
<th>N.Points</th>
<th>RaveGeo Size (KB)</th>
<th>Compr.</th>
<th>N.Points</th>
<th>Size pp</th>
<th>Comp time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LandUse</td>
<td>Shape</td>
<td>37000</td>
<td>2312907</td>
<td>2900</td>
<td>12.8</td>
<td>1156000</td>
<td>1.95</td>
<td>25.8</td>
</tr>
<tr>
<td>Roads</td>
<td>Shape</td>
<td>6800</td>
<td>198778</td>
<td>724</td>
<td>9.39</td>
<td>146000</td>
<td>2.51</td>
<td>1.32</td>
</tr>
<tr>
<td>Boundaries</td>
<td>VPF</td>
<td>54000</td>
<td>4297</td>
<td>12.6</td>
<td>1574000</td>
<td>2.73</td>
<td>58.2</td>
<td></td>
</tr>
</tbody>
</table>

The columns labeled \(N.Points\) contain the total number of stored points (2-dimensional) in each dataset and format, breakpoints and nodes (endpoints). Since RaveGeo shares geometry between objects and only stores a point once, the number is significantly lower for RaveGeo than for the source format. The effect is especially obvious for the land use dataset where almost all edges are shared between 2 polygons. Another mechanism that affects this is that the compiler removes duplicate points in lines.

The column labeled \(Compr\) contains the compression ratio compared with the source data. An average compression rate of 11.6 was achieved for these datasets.

The size used for each stored point in the RaveGeo format is shown in the column \(Size \text{ pp}\). This shows that on average for these datasets 2.4 bytes have been used for storing a compressed point containing an x and a y-value. The value is affected by the redundancy in the dataset but also the resolution that is chosen.

The compilation time (\(Comp \text{ time}\)) is dependent on the complexity of the data but is on average 110 MB/h (source data). The tests were performed at an Intel P4 2.0GHz PC. It seems like the overhead for small datasets affects the results here since we have observed results like 200 MB/h for large datasets like VMap1.

Figure 3 shows the kinds of data stored in the three datasets. The interesting part to note is the overhead required for the multi-resolution functionality. It is biggest in the LandUse
Streaming of compressed multi-resolution geographic vector data

dataset, which is not so surprising since this is the most complex one. It contains complex polygons with many holes and aggregations makes it change a lot between levels.

![Figure 3: Different types of stored data in the test datasets.](image)

In an ideal multi resolution dataset with constant perceived information density in different zoom levels, the data volume stored per level should decrease exponentially. The test dataset that shows the best behavior according to this criterion is Boundaries (Figure 4). From level 2 and up it shows a nice decreasing curve except at level 6. This top comes from the selection operator that has selected away country borders at that level which means that these are created there. The reason why the first levels deviate from the exponential curve is because the source resolution is less than the resolution in these levels. When the dataset was created they aimed at a certain accuracy when digitizing the lines and a certain size for areas to be included. It is this that shows in the curve below.

![Figure 4: Stored data per level in the test datasets.](image)

The curve for the roads dataset shows a very irregular behavior. This is because the generalization is done mainly by the select operator. To make this operator have a smooth
behavior there must be expressions defined in the configuration that eliminates data in each level. Often the source data lacks good enough attributes to make this possible.

CONCLUSIONS

It is possible to create a fast, compressed, streaming, multi-resolution geographic vector format. RaveGeo typically compress databases 10-15 times compared to common data formats and extends the useful scale range to a factor about 1000 (corresponds to 10 levels).

The generalization quality has in general showed to be good enough for most applications. The generalization efficiency, measured as the decrease of data volume with the levels, is very dependent on the data and the compiler configuration. The generalization operators thinning and aggregation can generally remove data automatically in each level while the selection operator depends much on the availability of attributes that describes the data well. The overhead for making the data available in about 10 resolutions is typically 10-20%.

The biggest obstacles and the most effort has been put into two areas that we did not expect to be that difficult when we started. One is dealing with real, imperfect data. In most databases there are geometric problems like zero size polygons or self-intersecting lines. The other area is making everything work with local algorithms that do not require the complete dataset at once to work.

REFERENCES