Management of multi-resolution data in a mobile spatial information visualization system

Jean-Michel Follin, Alain Bouju, Frédéric Bertrand and Patrice Boursier
Laboratoire Informatique, Image, Interaction
Université de La Rochelle
{jean-michel.follin, ...}@univ-lr.fr

Abstract

We propose a solution for the management of multi-resolution data in a mobile spatial information visualization system. This system has a client server architecture and is characterized by a slow communication link. Our solution is based on use of increments, i.e. the “difference” between two datasets with different levels of detail. It allows reuse of locally available data and reduces the amount of data transferred between client and server.

1. Introduction

Advances in mobile technologies (i.e. communications and devices) have led to new applications in mobile computing, in particular web-based mapping applications. Through Internet, a mobile user can download applications, maps and data. Spatial location of the user must be integrated in the data management ([1]).

The amount of data, which has to be transferred and displayed, can be optimized for the user’s purpose according to theme and scale. The work presented in this paper should be considered as continuation of works from [11], which have focused on importance of the reuse of vector data in a mobile environment characterized by limited data transfer rate. However this previous works has not taken into account the transmission of data at different Levels of Details (LoD, cf. [7]).

Needs of multi-resolution data in an embedded navigation application are obvious: for example, detailed information are required in town centre, and generalized data are sufficient on a highway section.

To fulfil these requirements, we can adopt a multi-scale database where datasets for different predefined scales are precomputed and stored on the server side. During transition between different levels of detail, data transfer between client and server can be reduced by reusing available data on the client side.

This paper proposes a framework allowing multi-resolution navigation in an embedded spatial information visualization system and presents preliminary experimentations.

After a review of previous works in relation with the multi-scale aspect in section 2, we propose an extension of the system to multi-resolution in section 3. It requires LoD data and transmission models in a client-server context. These models aim at managing transitions between different LoD representations of real-world entities ([8]).

2. General architecture of embedded spatial information system

The system is divided in two main parts:

- the client that manages data visualization, user requests and communication with data server,
- the server that manages the data and the access to data sources.

The system uses standard technologies: Global Positioning System (GPS) for the real-time localisation of mobile user, Personal Digital Assistants (PDAs) or pocket computer for the visualization of spatial informations by the client and cellular phone (with communication standards GPRS and UMTS) for the communication between client and server.

Purpose of our system is to minimize the amount of data exchanged between client and server ([11]).

2.1. Basic principles of the system

We have made the following hypothesis for the management and transfer of data:

1. all data are centralized and can only be modified on server,
2. data transfer is performed as an answer to a client request,
3. management is possible on the client side: reuse of locally available data allows to display requested information without connection to the server.

2.2. Data and transfer models

2.2.1. Data model Data organization in the model is based on traditional definition of geographical maps: objects are grouped into layers and the sequence of layers forms a map. An object entity is formed by the quadruple \( o, g, t, v \) with:

- \( o \): unique identifier,
- \( g \): geographical position and description modelled by one among six two dimensional geographical objects: Point, Line or Region for simple (i.e. connected) objects, and MultiPoint, MultiLine and MultiRegion for complex (i.e. not connected) ones,
- \( t \): timestamp value (last modification time),
- \( v \): other informations, values \( v = v_1, v_2, ..., v_n \) accessed through the set of objects attributes \( a = a_1, a_2, ..., a_n \) (for example, the name of a street).

Such a model allows the manipulation of only the useful part of an object during data exchange. A layer is a collection of objects associated with a description of objects attributes (it defines properties shared by all objects). Each layer corresponds to a specific theme (e.g. transportation or buildings) and can be decomposed in various LoD.

A map is a succession of layers grouping objects according to their structure and their semantics. An instance of a map is a concrete set of data entities available at a given time on the server or on the client side. The queries allow data transfer from the server to the client according to certain criteria. The definition of query, noted by \( q \), follows conventional \( \text{OQL} \) (Object Query Language) notations (15):

\[
\text{SELECT } o \text{ FROM } \{l_1, ..., l_n\} \text{ WHERE } C
\]

where \( o \) is an object of the layer \( l_i \) (layer to which belongs the set of selected objects) and \( C \) is the selection condition defined over the objects from the layers in \( L \) such that: \( L = l_1, ..., l_n \) (\( l_i \in L \) with \( 0 \leq i \leq n \)). The result of query \( q \) executed on the instance \( I \) of a map is noted by \( q(I) \).

Because data are not supposed to be created or modified locally, queries are defined with a restriction: the set of selected objects always belongs to only one layer (111). In this way, different objects cannot be combined to create a new object or layer.

The principles of data management and transfer developed in this formal frame have been designed for instances (on the server and client sides) of a same layer.

2.2.2. Transfer model In [1], three cases of client-server transfer are distinguished:

1. all the packet \( (o, g, t, v) \), noted by \( V \), is sent;
2. the set \( T \) of object identifier and modification time (i.e. the packet \( (o, t) \)) is sent;
3. only object identifier \( o \), the set \( O \), is sent.

In order to reduce the volume of exchanged data, three schemas of data transfer between the client and the server have been defined. In simple communication mode, the server sends directly the complete answer \( V \) to a query \( q \) transmitted by the client. In two-step communication schema, the server sends the answer in two steps upon a query \( q \) by the client. First, it sends the set \( T \) of objects from the result \( q(I) \), then, after that the client validates them locally by choosing objects \( O \) that are missing or updated, it sends the missing part \( V \) of the answer. In pre-computed answer mode, the client sends a query \( q \) with the description \( T \) of objects belonging to the answer \( q(I) \) locally computed (i.e. objects available on the client). The server sends the answer \( V \) of objects to create or update with, if necessary, objects \( O \) to delete.

Extension of our system to multi-resolution involves a partial revision of the above defined data and transfer models. Local creation of objects is necessary to reuse parts of the locally available objects: for example, points of the generalized representation of a polyline can be reused for its detailed representation.

3. Extension to multi-resolution of the system

3.1. Our approach

Three solutions are possible for a multi-LoD data management in an embedded context ([7]):

1. **Real Time Generalization** where different object representations for a given resolution (i.e. LoD representations) are computed in real-time,
2. **LoD Approach** where different LoD representations are pre-computed and stored on the server side,
3. **Combined approach**, chosen by [7], that combines these two concepts.
We have chosen a LoD approach combined with the use of increments that allow the real time reconstruction of object representations. According to the data structure, two variants of LoD approaches may be identified ([10]):

1. either "one object = one multi-resolution instance" where "each object has a single representation (i.e. one database instance) including multiple geometries, and all object instances are stored in a single multi-resolution database";
2. either "one object = many single-resolution instances" where "each object has multiple, interconnected representations", one for each resolution where it exists.

We have chosen the second LoD approach which seems more suitable to reduce the amount of transferred data in a mobile environment: only the instance for considered LoD is sent by the server to the client.

We suppose that we have several layers of data at various LoD: in our case, these layers come from only one source by generalization. We consider that all layers are served from a single topologically consistent source.

After the description of a framework for managing multi-LoD data in a client-server context in section 3.2, we examine interest of such a model by presenting preliminary experimentations in section 3.3.


We first formalize the notions of LoD operators (generalization and refinement), matching configuration of LoD data and increment.

Then, we propose an adaptation of existing data and transformation models for the management of geographical objects structured on several LoD layers.

3.2.1. LoD operators and increments

Two types of operators are implied in changes of object representations during navigation of an user across two LoD:

- operators of generalization used during a zoom out (decrease of detail),
- operators of refinement used during a zoom in (increase of detail).

The LoD operators have to take into account the number of changes in participating entities. It corresponds to the matching configuration, i.e. the number of matched LoD representations of same real world entities (when objects are represented at both LoD). Thus operators can be seen as different spatial entity mapping procedures ([2]):

- 1 : 1 spatial entity mapping procedures that map 1 LoD representation of an object to 1 different LoD representation of the same object,
- 1 : n mapping operators that match 1 LoD representation to a composite entity representing the same real-world object,
- n : m matching procedures that can be seen as a 1 : 1 matching case of cluster object.

In the 1 : 1 cases, identification of objects can be based on identifier of object. But in the 1 : n and n : m cases, which are the most frequent cases, matching of different LoD representations is less obvious. It can be based on an identification number included in all objects of both LoD as attribute value ([6]), or on a correspondence between two sets of identifiers. Examples of mapping operators of generalization (1) and refinement (2) are shown in Figure 1.

The generalization and refinement operators allow to perform changes on objects geometries after a request of transition from lod_n to lod_{n-1} or lod_{n+1}. In the case of the 1 : 1 mapping operators, they can be formalized as in the following equations:

\[
\text{generalize}(o_{id}^{lod_n}, \Delta g_{id}^{lod_n,lod_{n+1}}) = o_{id}^{lod_{n+1}}
\]

\[
\text{refine}(o_{id}^{lod_n}, \Delta g_{id}^{lod_{n-1},lod_n}) = o_{id}^{lod_{n-1}}
\]

where \(o_{id}^{lod_n}\) corresponds to an object with identifier \(id\) and a level of detail \(lod_n\), and \(\Delta g_{id}^{lod_{n+1},lod_n}\) is the geometrical difference between \(lod_n\) representation and \(lod_{n+1}\) or \(lod_{n-1}\) representation of the object with identifier \(id\).

In fact, this difference is either related to geometry or attribute, but in this paper only geometrical difference is considered.

The increments can be defined as LoD operators associated to data that correspond to the difference between two consecutive LoD of a vector dataset ([3]).

The increments set allowing transition from lod_n to lod_{n+1} (resp. lod_{n-1}) can be noted by Inc_{n,n+1} (resp. Inc_{n,n-1}). These "differences" are linked to two LoD layers and encoded for each matching configuration. They include:

<table>
<thead>
<tr>
<th>1:1 mapping operator point removal (1) and point insertion (2)</th>
<th>1:n mapping operator fusion (1) and split (2)</th>
<th>n:m mapping operator Selection/ elimination (1) and selection/ addition (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="figure1.png" alt="Diagram" /></td>
<td><img src="figure1.png" alt="Diagram" /></td>
<td><img src="figure1.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Figure 1. Different LoD operators.**
• an identifier for the linked objects that take into account their matching configuration,
• a set of operators and data (geometrical difference).

For example, we can consider the increment composed by the operator \texttt{insertPoints} and the geometrical difference $\Delta g_{\text{lod}_2,\text{lod}_1}$ that contains only coordinates and indexes of points (Fig. 2). This increment allows transition from LoD $\text{lod}_2$ representation of a polyline to its LoD $\text{lod}_1$ representation and its size is expected to be less important than size of the entire object LoD representation.

The increments can be stored on the server side and transmitted in answer to client query. They should allow reducing the amount of data transferred from server to client by reusing, when possible, already locally available data. After its use by the client, the increments set can be deleted from the client side.

By taking into account different LoD of data, we can define a LoD data model in mobile context.

3.2.2. A LoD data model

Several types of data can be distinguished in an embedded application for visualization of multi-LoD data. Figure 3 illustrates real-time navigation across two LoD representations of a same thematic layer. Mobile user accesses to LoD $n$ data at time $t_0$, then zooms out at time $t_1$ by requesting LoD $n+1$ data, and finally makes a new LoD $n$ data request at time $t_2$ by zooming in. Datasets $Dt_0$ and $Dt_1$ (respectively copied from the server at $t_0$ and $t_1$) correspond to instance available on client side and $Dt_2$ is dataset requested at $t_2$.

In $Dt_2$, three sub-sets of data can be identified in function of locally available LoD data.

1. An already available dataset can be reused from the same level of detail (LoD $n$). It corresponds to the intersection between data $Dt_0$ downloaded at $t_0$ and $Dt_2$, i.e. $Dt_0 \cap Dt_2$.

2. A dataset can only be reused from the previous level of detail (LoD $n+1$). It corresponds to the intersection between data $Dt_1$ and the part of data $Dt_2$ that is not covered by data $Dt_0$, i.e. $Dt_1 \cap (Dt_2 \setminus (Dt_0 \cap Dt_2))$.

3. A dataset is unavailable on the client for all LoD layers. It needs to be downloaded from the server.

Objects contained in the second type of dataset must be used as much as possible by the increments to reconstruct a part of data $Dt_2$ required at LoD $n$. These types of LoD data can be used by a client-server transfer model.

![Figure 2. Use of an increment.](image)

![Figure 3. Three types of LoD dataset in mobile context.](image)
We propose a general transfer model of multi-scale data based on communication with a pre-computed answer (cf. 2.2.2). It is divided in three steps (Fig. 4). The notion of working zone $W_z$ is used for reducing the transferred dataset and anticipating the movement of vehicle. It corresponds to “a buffer of data around the visible zone” on the screen ([4]). To simplify our presentation, we consider $W_z$ as a fixed-size rectangular window.

Multi-scale data transfer is performed in order to reuse data already available at a different LoD than required one. Thus, exploitation of LoD data can be made on the one hand as answer to a client request of transition between a LoD $m$ to a LoD $n$ representation (zoom in or out) and on the other hand each time that required data are covered by data available at a different LoD.

The source layer can indifferently be $l$ or $l'$ and is noted by $lm$ on the server side, and $lcm$ on the client side. The destination layer is noted by $ln$ on the server side, and $lcen$ on the client side.

1. The working zone $W_z$ is compared on the client side with the data available at source level $m$ and at queried level $n$: identifiers of objects reusable at the same LoD $n$ layer and at the previous LoD $m$ layer are computed (by partial inclusion of objects belonging to local instance at layers $lcm$ and $lcen$ in $W_z$).

2. A request $q$ is sent to the server including identifiers of $lm$ and $ln$, completed by two sets: $Tn$ for the objects already available at queried level $n$ and $Tm$ for objects exclusively available at source level $m$.

3. An answer is sent by the server for objects missing at the queried level, with:
   - LoD $n$ objects $Vn$ to create or update, and $On$ to delete if necessary (in answer to $Tn$),
   - LoD $m$ objects $Vm$ (to create or update) and $Om$ (to delete) needed for LoD $n$ and the increments set $Inc_{m,n}$ allowing reuse of objects only available in $lcm$ and required for $lcen$ (in answer to $Tm$),
   - and objects $V$ unavailable from both $lcm$ and $lcen$.

   Objects $Vn, On, Vm$ and $Om$ are only transferred in order to update data on the client side. $Inc_{m,n}$ and $V$ are only sent to the client if there are some missing data at requested LoD $n$.

The instructions to send from the server to the client depend on scale change: figure 4 illustrates a zoom out case. Preliminary experimentations have been made to estimate efficiency of this model.

![Figure 4. Three steps of client-server transfer for data at two levels of detail.](image)

### 3.3. Preliminary results

In a client-server context, use of increments is expected to be more efficient (in term of transmission time) than use of entire LoD representations of objects for certain situations. This efficiency can be evaluated by comparing the size of LoD representations of objects and the size of their corresponding increments.

First experimentations are based on two LoD layers representing the roads and streets of the city La Rochelle. To produce LoD 2 layer, the polylines have been manually selected in order to conserve only main lines of network, and simplified with modified Douglas-Peucker algorithm ([9]) to keep their essential shapes. Different LoD representations come from a single and same source. Increments of polylines preserved throughout the LoD correspond to sets of inserted or displaced points. Only the case of 1 : 1 matched polylines is seen.

We propose here a first evaluation of gains in term of sizes with simple encodings of polylines and increments.

#### 3.3.1. Encodings of polylines and increments

We can take into account sizes of Java primitive types to calculate the cases where it is more interesting to use increment rather than polyline:

- **int** is encoded on 4 bytes,
- **double** is encoded on 8 bytes.

Simple encodings of polylines and increments can be seen in Fig. 5.

A polyline is encoded as an int identifier and a vector of $N$ points each encoded by two double $x, y$ for the coordinates.

Increment of a polyline is implemented by an int identifier and a vector of $N'$ points increments. Each point increment is encoded with an int index, a byte which encode the type of operation (shift or insertion) and two double $x, y$ for the point’s coordinates (if it is inserted) or its shift (if it is displaced).

If we consider LoD 1 representation (i.e. the most detailed)
of a polyline, increments points allowing reconstruction of LoD 2 representation are in our case either removed points or displaced points and we have \( N' \leq N \).

### Polyline

<table>
<thead>
<tr>
<th>Polyline</th>
<th>PolylineIncrement</th>
</tr>
</thead>
<tbody>
<tr>
<td>int identifier; vector points;</td>
<td>int identifier; vector points&lt;sup&gt;1&lt;/sup&gt; increments;</td>
</tr>
</tbody>
</table>

\[ 1 \]

\[ 2 \ldots N \]

\[ 0 \ldots N' \]

### Point

<table>
<thead>
<tr>
<th>Point</th>
<th>PointIncrement</th>
</tr>
</thead>
<tbody>
<tr>
<td>double x, y;</td>
<td>int index; double x, y; //coordinates or shift byte operation; //insertion or shift</td>
</tr>
</tbody>
</table>

**Figure 5. Encoding of polylines and increments.**

For a polyline and an increment, identifier uses 4 bytes. However for a polyline each point uses 16 bytes (for its coordinates) while for an increment, each point uses 21 bytes (for its coordinates, its index and its type of operation). So we can evaluate the ratio between \( N' \) and \( N \) where use of increment is really useful with the following equation.

\[
N' \leq \frac{16}{21} \times N
\]

With such a simple encoding, it is more interesting to use only increments than entire polyline when the number of modified (i.e. displaced or removed) points of a polyline is less than 76% of its total number of vertices.

However, increments can only be used in association with objects already present on the client side at different LoD.

We can evaluate interest of using an increment strategy by simulating a multi-LoD data client-server transfer.

#### 3.3.2. Simulation of a multi-LoD data client-server transfer

Simulation of a multi-LoD data client-server transfer aims at comparing two schemas:

- **same LoD strategy**: only available data at the same LoD are used,
- **increment strategy**: available data at different LoD are used in association with increments.

For a given transition request from LoD \( m \) (previous LoD) to LoD \( n \) (required LoD), we will have two possibilities:

- for the first strategy, only LoD \( n \) available data will be considered on the client side, and LoD \( n \) missing data will be transferred from the server,
- whereas for the second, LoD \( m \) useful data (data only available at LoD \( m \) and reusable for LoD \( n \)) will be taken into account for transfer of increments, and only missing data for both LoD \( m \) and LoD \( n \) will be downloaded from the server (Fig. 6).

The missing data can be:

- either omitted objects during generalization process from LoD \( m \) to LoD \( n \),
- either new displayed objects.

**Figure 6. Different types of LoD data in a LoD \( n \) request.**

The simulation of a multi-LoD data client-server transfer using our model is given further in Algorithm 1. It is an useful tool allowing evaluation of different strategies in nearly real conditions.

The dataset is composed of two LoD layers of a digital map representing the transportation network of La Rochelle city. The most detailed layer (LoD 1) contains 1577 polylines and the generalized one (LoD 2) contains 467 polylines.

GPS route is a set of coordinates collected at regular time steps with a car equipped with GPS in La Rochelle city. It is used to simulate the real time navigation of a vehicle. currentBox corresponds to the rectangular box centered on current position of the user (a point of GPS route). It is used to simulate the screen of a mobile computer and to select set of displayed polylines.

Array of LoD is used to associate a current LoD to each point of GPS route. It allows to switch at regular intervals from LoD 1 to LoD 2 and inversely with the method getCurrentLoD. This method allows to simulate LoD change re-
quests of a mobile user. However these LoD changes are computed without taking into account the real behavior of an user (but simply by dividing the number of trajectory’s points by the number of wanted LoD changes).

The conditions for a spatial data transfer (i.e. the creation of a new currentBox) are either a displacement of the user (if current point of the route is no more in currentBox), either a LoD change request (if current LoD is different from previous one).

In these cases, three sets are computed:

- LoD \( n \) missing objects (same LoD strategy),
- missing objects (increment strategy),
- and LoD \( m \) useful increments (increment strategy).

**Input:** A GPS route
- Two LoD of a same layer
- An array of LoD change

**Output:** Statistics

for each point of the GPS route do
  getCurrentLoD
  if current point not in currentBox or change of LoD then
    recreate currentBox
    if currentBox partially covered by data locally available at the same LoD then
      computing of dataset \( D_1 \) with objects unavailable at the current LoD (same LoD strategy)
      computing of dataset \( D_2 \) with objects unavailable at the two LoD (increment strategy)
    else
      computing of dataset \( D_3 \) unavailable at current LoD but available at different LoD (increment strategy)
    end
  end
end

Algorithm 1: Simulation of client-server transfer of data during a mobile user navigation across 2 LoD layers

In our first experimentations, we have computed proportion of reused objects (i.e. the ratio between the number of LoD \( m \) useful objects and the number of LoD \( n \) missing objects) in an increment strategy (Fig.7). We have also computed reduction of transferred points for the increment strategy in relation with the same LoD strategy (Tab. 1). This reduction corresponds to the ratio between number of points belonging to LoD \( m \) useful objects and missing objects, and number of points belonging to LoD \( n \) missing objects.

With a currentBox of 1000 \( \times \) 1000 meters, a number of LoD changes varying from 5 to 30 and three GPS routes of 7.82, 14.1 and 32.7 kilometers, we observe proportions of reused objects from 16 to 24\%. The reduction of transferred vertices vary from 8 to 15\%. Averages are:

- for the reuse of LoD data: 21\%,
- for the reduction of transferred vertices: 10\%

The proportion of reused objects with increment strategy is rather significant. It globally increases relatively to augmentation of LoD changes.

The reduction of the amount of transferred points is significant because we consider a mixed set of objects (missing set) and of increments (LoD \( m \) useful set), and not only increments. Moreover, we do not consider orientation of the increments in our model. In order to reduce the amount of transferred data, different increments might be needed for transition from LoD \( n \) to LoD \( n + 1 \) than for inverse transition.

These first results show interest of using such models of LoD data and transfer in a mobile context with limited resources.

![Figure 7. Reuse of objects in increment strategy.](image)

4. Conclusion

Our solution for managing multi-LoD data in an embedded context can be divided in two main parts:

1. encoding of operations allowing browsing across LoD layers where objects representing the same real-world entities are matched,
2. use of a suitable data transfer schema.
Table 1. Reduction of transferred points.

<table>
<thead>
<tr>
<th>Number of LoD changes</th>
<th>Gps Route 1</th>
<th>Gps Route 2</th>
<th>Gps Route 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>14.11%</td>
<td>9.38%</td>
<td>9.06%</td>
</tr>
<tr>
<td>10</td>
<td>7.85%</td>
<td>10.56%</td>
<td>7.33%</td>
</tr>
<tr>
<td>20</td>
<td>9.30%</td>
<td>10.00%</td>
<td>10.60%</td>
</tr>
<tr>
<td>30</td>
<td>13.70%</td>
<td>11.66%</td>
<td>9.79%</td>
</tr>
</tbody>
</table>

This solution fulfills requirements of multi-resolution navigation in an embedded context by reducing the amount of data transferred between client and server and adapting displayed data to the users needs.

First encouraging experimentation has been made on 1:1 matched data. We will now extend these experimentations to more complex cases, i.e. to 1:n and n:m matched data.

Furthermore, the scope of application of our methodology is limited to LoD data coming from only one source by generalization. We can think about its adaptation to LoD data resulting from different sources: problems of matching between such representations of same real-world entities should be encountered.

Another issue of our work is related to management and visualization of both detailed and generalized data (when an user needs both an overview map and a detailed map of its current position).

Acknowledgment

This research was partially founded by the Communauté d’Agglomération of the city of La Rochelle, France.

References


