One Click Focusing: An SQL-based Fast Loop Road Extraction Method for Mobile Map Services

Daisuke Yamamoto
Nagoya Institute of Technology
Gokiso-cho, Showa-ku,
Nagoya, Aichi, Japan
daisuke@nitech.ac.jp

Hiroki Itoh
Nagoya Institute of Technology
Gokiso-cho, Showa-ku,
Nagoya, Aichi, Japan
hito@moss.elcom.nitech.ac.jp

Naohisa Takahashi
Nagoya Institute of Technology
Gokiso-cho, Showa-ku,
Nagoya, Aichi, Japan
naohisa@nitech.ac.jp

***Abstract*—This paper proposes a method for the fast loop roads extraction for mobile Web map services and applications. Since the existing loop road extraction method has drawbacks such as those related to the processing speed and interactivity, it has been difficult to apply the method to real-time applications such as mobile Web map services directly. Therefore, this paper proposes a fast extraction method that involves acquiring information on all loop roads with high efficiency in advance and storing the information in a database and querying those with SQL statements. The proposed method is 51.0 times faster than the previous method, and for expanded loop road extraction, it is 16.4-25.3 times faster than the previous method. Further, when used to build a loop road database, the proposed method, which involves the use of a tabulation method, is 3.86 times faster than the conventional method. We have developed the Web API function in order to acquire loop roads easily from other Web services. For the application of the proposed method, we have developed the One Click Focusing function that can modify the size, position, and scale of the focus automatically in fisheye-view maps.**

***Keywords*-fisheye views, focus+glue+context,Web map service**

I. INTRODUCTION

Advanced Web map services, such as Google Maps, have become available in recent years. Smart phones equipped with GPS sensors, such as iPhones, have also become increasingly popular, which means that anyone can access mobile Web map services. Web map services allow users to determine their current position, but they also enable the sharing of content mapped with positions in many applications, such as location-based SNS (Social Network Services) [1] and pedestrian navigation systems [2].

Many applications are based on existing Web map services, such as location mapping systems based on points (latitude and longitude) and navigation systems based on lines (road networks), but few services are based on polygons, such as areas and city blocks.

However, it is important that mobile web map services include surfaces, such as areas and city blocks, as well as points and lines. For example, let us consider a situation where a user wants to find certain areas using mobile maps. A user must adjust the scale and position of the map to visualize the whole area in the display, after scrolling through the map. However, this operation is slightly too complex in a mobile environment. Therefore, there is a user need for target area adjustments that use simple one-click or one-touch operations on mobile maps. Some map databases are based on polygon data for areas and structures, such as the National Land Numerical Information download service [3] and U.S. Census Bureau Tiger/Line [4], so we aimed to develop a method for automatically adjusting maps using these polygons. In general, humans create polygons, but polygon data is not available for all areas and structures. Therefore, we must automatically extract these polygons from road databases.

Our study was focused on a loop road. A loop road is the smallest road network, which surrounds a target point on a road network map, as shown in Figure 1. The area surrounded by a loop road is a city block. Many areas consist of some city blocks, including shopping centers, universities and parks, so a city block is an important unit. Thus, if we can handle city blocks easily on Web map services, more area-based Web map applications will be made available.

The purpose of this study is to develop a fundamental system to enable Web map services that can handle city blocks easily. We propose a fast extraction method for loop roads and expanded loop roads. Moreover, we developed a “ One Click Focusing ” function on this method.

The following requirements had to be satisfied to implement the proposed system.

- Requirement 1 The processing speed must be fast and stable, in order to allow a serviceable response with Web map services.
- Requirement 2 Loop roads must be calculated from existing road networks without human costs.
- Requirement 3 A Web API must be defined in order to implement the proposed system easily from other Web services.
- Requirement 4 Applications based on the proposed system must be available.

Thus, the proposed system possesses the following characteristics, in order to satisfy the requirements.

 ISBN: 978-1-61208-178-6

Characteristic 1 Loop roads extraction can be rapid and stable by querying to a loop road database with SQL statements. Since SQL is one of the fastest technologies that involve searching massive data, we believe that our approach is the best solution for extracting loop roads. (Corresponding to Requirement 1)

Characteristic 2 The loop road database can be built automatically from existing road databases efficiently. (Corresponding to Requirement 2)

Characteristic 3 The Web API can communicate with clients and other Web servers. (Corresponding to Requirement 3)

Characteristic 4 We developed a One Click Focusing function for fisheye view maps as an application of the proposed system. (Corresponding to Requirement 4)

Our method will provide a fundamental technological contribution to innovative and intelligent Web map services.

This paper has eight sections. Section II summarizes existing loop road extraction methods. Section III describes a fast method for extraction. Section IV presents the structure of the proposed system. Section V contains the experimental results. Section VI describes an application based on the proposed system. Section VII contains related work. Section VIII concludes this paper.

II. Loop Road Extraction Method

We previously proposed a loop road algorithm [5] for extracting loop roads and expanded loop roads. This section summarizes existing methods and provides definitions of a loop road and an expanded loop road. These methods can extract loop roads and expanded loop roads with high accuracy, but the processing speeds of these methods are too slow for Web map service applications. Therefore, we propose fast extraction methods in the next section.

A loop road is the smallest loop road network, which surrounds a target position, as shown in Figure 1. A city block is the area surrounded by a loop road. A level 1 expanded loop road is the loop road network that surrounds a loop road. A level N expanded loop road is the road network that surrounds a level N-1 expanded loop road.

A loop road is important for the following reasons. Some areas, including parks, universities, and shopping centers, generally consist of city blocks, so we can treat a city block as a semantic unit that divides an area. Thus, a loop road treats road networks as areas. Some parks and universities contain several city blocks, so we need to address neighboring city blocks as well as single city blocks. We can process neighboring city blocks simultaneously by extracting expanded loop roads.

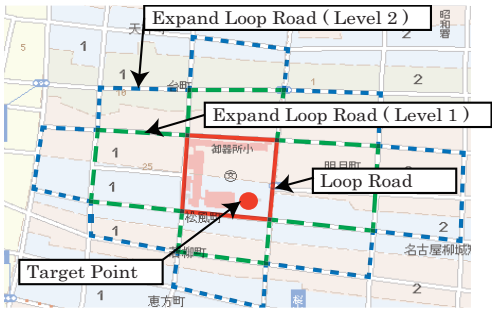

Figure 1. Sample of a loop road and expanded loop roads. The area surrounded by a loop road is a city block.

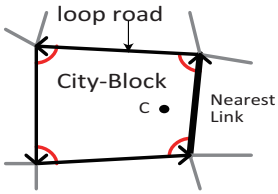

Figure 2. Example of a loop road.

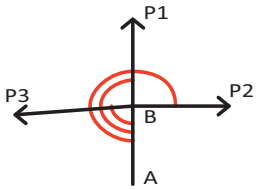

Figure 3. Selection of a loop road.

A. Method for Loop Road Extraction

We propose a loop road algorithm that extracts a loop road surrounding point C. We initially remove orphan links that do not form a loop road, such as straight roads.

First, we find the nearest link from point C then we follow the link in a counterclockwise direction. When the link connects to the tail of the first link, the path connecting these links is a loop road. However, when the link comes to a dead end, we follow a neighboring link using a depth-first search algorithm, as shown in Figure 2.

Algorithm 1 finds the loop road algorithm in the counterclockwise direction. Here, *start.tail* and *start.head* are the front and tail nodes, respectively, of the nearest link from point C. *setVisitedLink(node1, node2)* is a function that sets a *visited flag* on the link that connects node1 and node2. *node.UnvisitedLeftChild()* is a function that returns the unvisited and leftmost links connected to the *node*. Figure 3 shows that the function returns node P3 when node B is the current node and no links (P1, P2, P3) are not as visited flags. Likewise, *node.UnvisitedRightChild()* is a function that returns the unvisited and rightmost links connected to the current node.

B. Method for Extracting Expanded Loop Roads

The expanded loop road algorithm expands to city blocks surrounding a loop road in a radial direction, block-by-block.

The expanded loop road algorithm proceeds as follows. For each link in the level N expanded loop road, apply the loop road algorithm in the clockwise direction, as shown in Figure 4. This finds the level N+1 expanded loop road and expands to city blocks in a radial direction. However, we

 ISBN: 978-1-61208-178-6

Algorithm 1 $LoopRoad$ algorithm (counterclockwise)

Require: $Link : Start$
1: $stack := \mathbf{new} Stack()$
2: $setVisitedLink(Start.tail, Start.head)$
3: $stack.push(Start.head)$
4: **while not** $stack.empty()$ **do**
5: $node := stack.top()$
6: **if** $node = Start.tail$ **then**
7: **return** $stack$
8: **else**
9: $child := node.UnvisitedLeftChild()$
10: **if** $child = null$ **then**
11: $stack.pop()$
12: **else**
13: $setVisitedLink(node, child)$
14: $stack.push(child)$
15: **end if**
16: **end if**
17: **end while**

must apply the loop road algorithm in a counterclockwise direction for some links when expanding to city blocks in a complex road network. Thus, we must apply the loop road algorithm in both clockwise and counterclockwise directions for each link. The algorithm is provided below.

Step 1 Initialize List A.
Step 2 Store links for an initial loop road (Figure 4-a) in List A.
Step 3 For each link stored in List A, apply the loop road algorithm in both clockwise and counterclockwise directions, as shown in Figure 4-b.
Step 4 Remove links stored in List A from links generated in Step3.
Step 5 Apply the loop road algorithm in the counterclockwise direction using the links generated in Step4, as shown in Figure 4-c. This provides an expanded loop road.
Step 6 Repeat from Step 2 if the city blocks need to be further expanded, as shown in Figure 4-d and 4-e.

List A is the list structure storing the loop road. Finally, we find the radius and center point of the Focus connected to the loop road at each level.

III. Fast Extraction Method

This section describes fast extraction methods for loop roads and expanded loop roads.

The basic principle of the proposed method is as follows. First, we build a loop road database to store all loop roads, which are calculated in advance to improve efficiency. The loop road database contains a loop road table with the road links of loop roads and a neighboring table with relationships between roads and loop roads. Thus we can rapidly find any loop road by querying the loop road database without searching road networks, because searching huge networks would be too computationally intensive. We can also find expanded loop roads by combining neighboring loop roads, which also eliminates the need to search road networks.

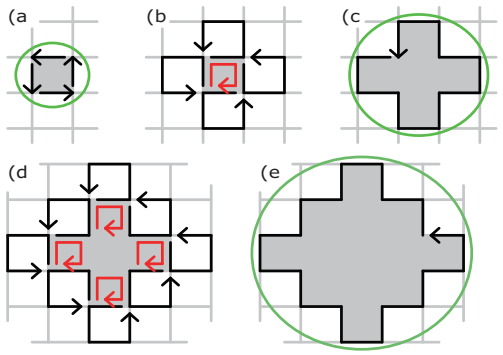

Figure 4. Expanded loop road algorithm. Gray lines, black arrows, red arrows, and the green circle indicate roads, links, previous links, and the Focus area, respectively.

A. Definitions of databases

First, we provide details of the road database and the loop road database.

The road database contained a road table with road links between intersections for navigation. The columns of the road table contained road link IDs, coordinates of road start points and end points, and other road link IDs connected to this link. The road database was converted from the “Navigation Road Map 2007” published by Yahoo Japan.

A loop road database includes the LoopRoadTbl table and the NeighborTbl table. The LoopRoadTbl table contains road links for loop roads. The columns of the LoopRoadTbl table contain loop road ID(id), lists of road link IDs and coordinates connected with a loop road (roadList, coordList), and a rectangular region containing a loop road (north, south, west, east), as shown in Table I. The roadList column is a text field containing a list of road link IDs in Comma Separated Value (CSV) format. For example, the CSV format of a loop road with road IDs of 100, 101, and 102 is ”100,101,102”. The coordList column is also a text field containing a list of coordinates for the loop road.

The NeighborTbl table contains relationships between a road and loop roads containing this road. Figure 5 shows that a road connects with two loop roads: a left loop road and a right loop road. The columns of the NeighborTbl table contain road link ID (id), left-hand side loop road ID (leftId), and right-hand side loop road ID (rightId), as shown in Table II. We can find loop road IDs connected with any roads by searching the neighboring table.

B. Loop Road Database Construction

Here we describe an automatic method for building a loop road database from an existing road database to satisfy requirement 2 in Section I.

 ISBN: 978-1-61208-178-6

Table I
DEFINITION OF THE LOOP ROAD TABLE: LOOPROADTBL

column	type	description
id	integer	ID of the loop road
north	real	northernmost latitude of loop road
south	real	southernmost latitude of loop road
west	real	westernmost longitude of loop road
east	real	easternmost longitude of loop road
roadList	text	list of road IDs
coordList	text	list of coordinates

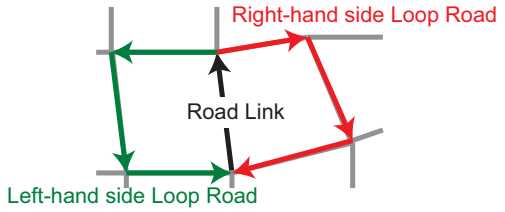

Figure 5. Road connected to two neighboring loop roads (left-hand side and right-hand side).

A very high number of road links exist in Japan, so it is difficult to generate a loop road database in a short period of time. We needed to generate a loop road database as quickly as possible, so we tested the following three methods.

1) Grid Division Method: First, we discuss a simple method of dividing the map into a grid made of vertical and horizontal lines at a constant interval before applying the loop road algorithm described in Section II to all vertices of the grid.

The algorithm for this method is as follows. First, we divide a map into a grid with a distance interval n. The following steps are then applied to each vertex P of this grid.

Step 1 Apply the loop road algorithm described in Section II to point P.
Step 2 Check whether the same loop road is stored in the loop road database.
Step 3 Insert the loop road to the loop road database if the loop road does not exist in the database.

This method can extract loop roads containing any vertices in the grid, but this method cannot extract loop roads that contain no vertices in the grid. A loop road with multiple vertices in the grid must be calculated redundantly with the same number of vertices included in the loop road. We may extract all loop roads if the distance n is sufficiently small, such as 1 meter, but the extraction time is too great since there are a lot of vertices. In contrast, we can extract loop roads in a short time if distance n is sufficiently high, such as 10 km, but we might not extract many loop roads because small loop roads might include no vertices. The density of roads varies in different areas, which makes it difficult to determine the best distance n for solving this dilemma.

2) Road Link Method: We next discuss the road link method, which extracts loop roads by searching from each road instead of vertices on a grid. This method applies clockwise and counterclockwise loop road algorithms based on Algorithm 1 to all road links and inserts these loop roads into the loop road database.

Table II
DEFINITION OF THE NEIGHBORING TABLE: NEIGHBORTBL

column	type	description
id	integer	ID of the road
leftId	integer	left-hand side loop road ID
rightId	integer	right-hand side loop road ID

The algorithm is as follows. We apply the following steps to each road link L.

Step 1 Apply the clockwise loop road algorithm based on Algorithm 1, to a road link L.
Step 2 Apply counterclockwise loop road algorithm to a road link L.
Step 3 Check whether the same loop road is already stored in the loop road database, for each loop road detected in Step 1 and Step 2.
Step 4 Insert each loop road to the loop road database if the same loop road is not present in the database.

In contrasts to the grid division method, this method can extract all loop roads. A problem with this method is the need to calculate n times for one loop road when a loop road has n road links. Ideally, a loop road must be calculated only once, so the extraction time of a loop road with this method is n times longer compared with the ideal method. For example, when $n = 4$ in the grid road network, the road link method takes four times longer than the ideal method. It is desirable to solve this duplication problem.

3) Road Link Method without Duplication: Finally, we propose a road link method without duplication. This method can extract all loop roads without calculating duplicate loop roads by using the NeighborTbl table.

The algorithm is as follows. We apply the following steps to each road link K, where a link ID of link K is k. The detailed algorithm is shown in Algorithm 2.

Step 1 Select the row for a column, where the link ID is k from the NeighborTbl table. Go to Step 6, if the leftId of this row is not null.
Step 2 Find a loop road R by applying the counterclockwise loop road algorithm based on Algorithm 1, where the loop road ID of the loop road R is r.
Step 3 Go to Step 6 if Step 2 cannot find a loop road.
Step 4 Insert loop road R into the LoopRoadTbl table.
Step 5 For each road K' of the loop road R, update the leftId column in the row of the linkId column in the NeighborTbl table for k' as follows.

```
UPDATE NeighborTbl SET leftId=r WHERE id=k'
```

Step 6 Likewise, apply Step 1 to Step 5 using the clockwise loop road algorithm and rightId.

This method is fast, because this method can build the

Algorithm 2 Road Link Method without Duplications (counterclockwise). $LoopRoad(link)$ function returns the loop road started from $link$ based on Algorithm 1.

Require: $Link : k$

```
1: row :=select * from NeighborTbl where id = k
2: if row.leftId = null then
3:    loop := LoopRoad(row.ID)
4:    if loop ≠ null then
5:       for all link ∈ loop do
6:          update NeighborTbl set leftId = loop.ID
            where id = link.ID
7:       end for
8:    end if
9: end if
```

LoopRoadTbl table without duplications by referring to the NeighborTbl table. The number of loop road calculations required this method can be four times smaller than that required by the road link method, when a loop road has 4 road links. This method requires extra disk space for the NeighborTbl table, but the NeighborTbl table is also required for speeding up expanded loop road extraction. For these reasons, we adopted this method.

C. SQL-based Loop Road Extraction

We propose a rapid method for finding the loop road for a target point P by referring to the previously constructed loop road database.

The algorithm of this method is as follows, where the latitude of target point P is $P.lat$, and the longitude is $P.long$. The detailed algorithm is shown in Algorithm 3.

Step 1 Find the candidate loop roads by querying the LoopRoadTbl table using the following SQL query.

```
SELECT * FROM LoopRoadTbl where
  north<P.lat and south>P.lat and
  west <P.long and east>P.long
```

Step 2 Apply a point-in-polygon[6] algorithm to the point P and each loop road candidate to determine the loop road including the point P.

Step 3 The result is a candidate loop road including the point P. If no candidate loop road includes the point P, the result is null.

The advantage of this method is that we can rapidly acquire loop roads using a simple SQL query and without searching any road networks.

D. SQL-based Expanded Loop Road Extraction

Next, we propose a method for finding an expanded loop road for a target point P. The algorithm for this method is as follows, where Set A and Set B are null. The detailed algorithm is shown in Algorithm 4.

Step 1 Find a loop road that surrounds the point P by using the loop road extraction method described in Section III.C.

Algorithm 3 $FastLoopRoad$ Algorithm.

Require: $Position : P$

```
1: rows :=select * from LoopRoadTbl where north <
   P.lat and south > p.lat and west < P.long and
   east > P.long
2: for all row ∈ rows do
3:    if PointInPolygon(P, row) then
4:       return loop
5:    end if
6: end for
```

Algorithm 4 $FastExpandedLoopRoad$ Algorithm. N means a level of the expanded loop road. $LinksOf(A)$ function returns road links included in loop roads of A.

Require: $Position : P, Integer : N$

```
1: A :=new Set()
2: B :=new Set()
3: loop := FastLoopRoad(P)
4: if loop ≠ null then
5:    A.add(loop.ID)
6:    for i = 1 → N do
7:       B.addAll(A)
8:       rows :=select leftId, rightId from
         NeighborTbl where id in (select id from
         NeighborTbl where leftId in A or rightId in
         A)
9:       A.clear()
10:      for all row ∈ rows do
11:         A.add(row.leftId)
12:         A.add(row.rightId)
13:      end for
14:      A.removeAll(B)
15:   end for
16:   return LoopRoad(LinksOf(A) − LinksOf(B))
17: end if
```

Step 2 Add the loop road ID to Set A if the loop road is not null.

Step 3 Repeat the following Step 3.1 to Step 3.4 N times.

Step 3.1 Add Set A values to Set B.

Step 3.2 Find loop roads with road links in loop roads included in Set A by querying the NeighborTbl table with the following query.

```
SELECT leftId, rightId FROM neighborTbl
  WHERE id IN (SELECT id FROM neighborTbl
    WHERE leftId IN ( Set A values ) OR
    rightId IN ( Set A values ))
```

Step 3.3 Update Set A with the values of leftId and rightId based on the SQL results of Step 3.2.

Step 3.4 Remove Set B values from Set A.

Step 4 Remove links in loop roads of Set B from links in loop roads of Set A. Apply LoopRoad method described in Algorithm 1 to the links in order to

 ISBN: 978-1-61208-178-6

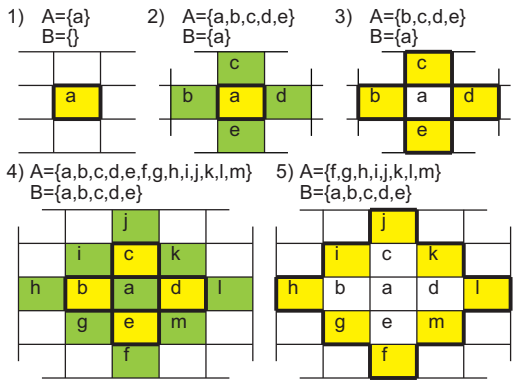

Figure 6. Example of Level 2 expanded loop road algorithm. 1) a is a loop road extracted by FastRoadLink function, 2) Add values of A to B. And, $a, b, c, d, e \in A$ are the loop roads extracted by the SQL query of line 8 of Algorithm 4, which share the edges of a. 3) Remove values of B from A. 4) Add values of A to B. A are the loop roads extracted by the SQL query of line 8, which share edges of b, c, d, e. 5) Remove values of B from A. Remove links in loop roads of B from links in loop roads of A.

connect the links.

The result of Step 5 is the target expanded loop road (Level N).

The advantage of this method is that it is almost completed by the SQL query in Step 3.1. The SQL query is only conducted N times for level N and the size of the NeighborTbl table is smaller than the LoopRoadTbl table, and the road database. Thus, we expect that the processing speed of this method will be lower than other methods.

IV. System Architecture

In order to satisfy Requirement 3 described in Section I, we had to develop a function to communicate with other Web services. Thus, we examined the following two methods. One is a library approach used by Web engineers to install a whole system to a Web server, including the programs and the large road loop road databases. The other method is a Web API approach used by Web services to communicate on-demand with our system using a Web API. The library approach cannot satisfy Requirement 3, because this approach uses large volumes of disk spaces and requires engineers to install programs on Web servers. Therefore, we adopted a Web API approach that more easily allows Web engineers to use the proposed methods.

A. System Architecture

We adopted a server-client architecture, as shown in Figure 7. This system includes an application server and database servers. A database server includes the previously constructed road databases and the loop road database. We adopted MySQL 5 as the database management system. We adopted Tomcat 6 and Java 1.6 in the application server. The method of communication between the server and clients was based on a Web API described in the next section. The

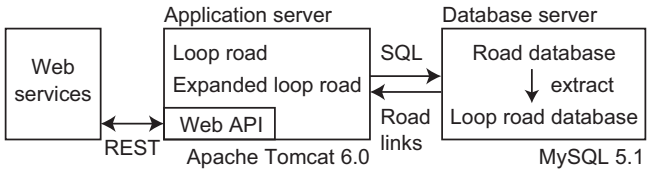

Figure 7. System architecture. We adopted a server-client architecture with Web API functionality.

adoption of Web API allows clients and other Web servers to find loop roads and expanded loop roads on demand.

B. Web API

Web API provides methods such as REST (Representational State Transfer) and SOAP (Simple Object Access Protocol). We adopted the REST method, because this method is easiest for Web services. This method submits parameters by adding them to a request URL and returns results in XML format. A sample URL request is as follows.

```
http://server/api?mode=loop&lat=X&long=Y&level=N
```

In this request, $server$ is the URL of proposed system, while the $mode$ attribute indicates whether the request wants to acquire a loop road or an expanded loop road. The system returns a loop road if the $mode$ attribute is “loop”. The system returns an expanded loop road if the $mode$ attribute is “expand”. We can set $lat/long$ using latitude and longitude attributes. We can set the appropriate level of an expanded loop road using the $level$ attribute. The response contains the road links of the loop road. An example of a response is as follows.

```
<looproad>
  <header>
    <circle lat="" long="" radius=""/>
    <rect top="" bottom="" left="" right=""/>
  </header>
  <roads>
    <road id="" lat1="" lng1="" lat2="" lng2=""/>
    ...
  </roads>
</looproad>
```

The $header$ element contains a summary of the response, which includes a $circle$ element and a $rectangle$ element. The $circle$ element contains a center coordinate and the radius of a circle including the loop road. The $rectangle$ element shows a rectangle containing the loop road. The $roads$ element includes $road$ elements with road links for the loop road, where the $lat1$ and $long1$ attributes denote the starting point of the road link, and $lat2$ and $long2$ indicate the end points. Web services can access loop road information by reading this XML format.

V. Experimental Result

We tested the applicability of the proposed system from the perspective of Web services. Thus, we compared the processing time of the existing method with that of the proposed method.

 ISBN: 978-1-61208-178-6

The experimental environment consisted of a database server and an application server, as described in Section IV. The size of the road database was about 22 GB, so we could not load all roads in the memory. Instead, we loaded road links from the road database on demand. The specification of the database server was as follows: Intel Core i7 2.9 GHz, 12.0 GB memory. The application server contained the programs for the proposed system. The specification of the application server was as follows: Intel Core i7 3.0 GHz, 8.0 GB memory.

A. Loop Road Database Construction

First, we investigated the time required for constructing the loop road database. The loop road database can be constructed automatically in advance from road databases, but the time required to construct the loop road database must be short. We must rebuild the loop road database after the original road databases are updated, so it is better to construct the loop road database as quickly as possible to reduce the lead time.

Therefore, we compared the following three methods. The target area was a central area in a major Japanese city (Nagoya city). The area has 21820 road links.

- method 1 We used the grid division method described in Section III.B.1 and divided the map into 50 m, 100 m, or 200 m intervals.
- method 2 Road link method, as described in Section III.B.2.
- method 3 Road link method without duplication, as described in Section III.B.3.

We investigated the construction time, the number of calculations required, and the number of loop roads with each method. The number of calculations referred to the total number of function calls by the loop road algorithm. The number of loop roads means the actual number of loop roads without duplications, because the same loop road may be counted many times.

Table III shows the results. First, we compared method 1 with method 2. Method 2 was 0.68-9.8 times faster than method 1. In particular, the total time required for method 1 with 50 m intervals was significant longer than other method, including method 1 with 100 m and 200 m. Method 1 with 50 m intervals detected almost as many loop roads as method 2, whereas method 1 with 50 m and 100 m intervals failed to detect many loop roads. This result suggests that method 1 has a problem with the trade-off between calculation speed and the number of loop roads, which contrasts with methods 2 and 3.

Next, we compared method 2 with method 3. Method 3 was 3.86 times faster than method 2. The loop roads calculation number with method 3 (8737) was 4.99 times higher than method 2 (43640), which suggests that method 3 is effective in avoiding duplications. Since the number of extracted loop roads with method 3 was almost identical to method 2, the extraction rate was almost the same. Method 3 has the disadvantage that it requires extra disk space in the NeighborTbl table, but we consider this problem to be trivial because the NeighborTbl table is also useful for accelerating the expanded loop extraction method. However, method 3 has the advantage that it constructs a stable loop road database rapidly and robustly.

Table III
TIME REQUIRED TO CONSTRUCT LOOP ROAD DATABASES USING EACH METHOD. METHOD 3 IS PROPOSED METHOD.

	total time	calculation number	loop road
method 1 (50m)	17291 s	26649	7518
method 1 (100m)	4518 s	6745	4402
method 1 (200m)	1206 s	1828	1407
method 2	1769 s	43640	8082
method 3	459 s	8737	8058

The target area was only one of more than 4800 areas in Japan, so the time required to construct a database for all areas in Japan may be more than 4000 times greater. Therefore, the advantages of the proposed method become more significant when applied to all areas in Japan.

B. Loop Road Extraction

Next, we investigated the loop road extraction time. The extraction time for a loop road included: 1) the time required for accessing databases; and 2) the time required for calculating road networks. Thus, we measures the time required for each process. The target area was the center area of a major city in Japan (Nagoya city). We measures the extraction time for loop roads at 1000 random time points. We compared the following two methods.

- method 1 Existing method. The system applied the algorithm described in Section II.A, after loading road links within $1km^2$ of the target point.
- method 2 Proposed method described in Section III.C. This system can detect loop roads much more quickly by using the loop road database, rather than the road database.

Table IV shows results. The total time required for method 2 was 45.4 times faster than that with method 1. The reason for this was as follows. Both methods had to access the database once, but the time required for accessing the database with method 2 was 51.0 times faster than with method 1. Method 1 had to acquire road links from the database in a large range (1 km^2), because the range of the target loop road was previously unknown, whereas method 2 had the advantage of acquiring candidate loop roads directly from database by issuing a simple SQL query. Method 1 had to apply a loop road extraction algorithm after acquiring the SQL results, whereas method 2 had the advantage of not applying this algorithm. The average road network calculation time with method 2 was only about 1 ms.

 ISBN: 978-1-61208-178-6

The standard variance with method 2 was much smaller than that with method 1. The reason for this was as follows. The road network calculation time with method 1 varied with the size and complexity of the loop road, whereas this was constant with method 2 which acquired any result by issuing a simple SQL query to the database without any complex processing requirements.

The precision of method 2 (98.0%) was higher than that of method 1 (92.3%). Method 1 searched a loop road starting from only the nearest link of target point, whereas method 2 could search a loop road starting from any of the links. The main reason why the precision of method 2 was not 100 percent was that some areas lacked loop roads, such as the coast. The precision of method 2 was low if the loop road database could not be constructed with high accuracy, which suggests that loop road extraction and construction of the loop road database had high accuracy with the proposed method. Method 2 was more stable and quicker at extracting loop roads than method 1, which suggest that the proposed method is more suitable for Web services.

C. Expanded Loop Road Extraction

Finally, we investigated the extraction time for expanded loop roads. The extraction time for an expanded loop road includes: 1) the time required to access the database; and 2) the time required to calculate data. Thus, we measured the time required for each process. The target area was the center area of a major city in Japan (Nagoya city). We measured the time required to calculate the loop road extraction at 100 random points, for each Level of the expanded loop road. We compared the following two methods.

- method 1 Existing method. The system applied the algorithm described in Section II.B to road links, after loading road links within 2 km^2 of the target point.
- method 2 Proposed method described in Section III.D. This system used the loop road database to find loop roads more quickly.

Table V shows the results. The total processing time with method 2 was 16.4-25.3 times faster than with method 1. The reason for this was as follows. The detection time with method 1 was higher, because method 1 had to load a very large range of road links from the road database, whereas method 2 had the advantage that it only had to make as many SQL queries as there were loop road levels. Method 1 had to apply the loop road extraction algorithm to road networks for the total number of city blocks, whereas method 1 only had to apply this algorithm once.

The standard variance with method 2 (9.1-39.3) is much smaller than that with method 1 (2096-8321). The reason for this was as follows. The road network calculation time with method 1 varied with the size and complexity of the loop road, while it was constant with method 2 which acquired any result by issuing an SQL query to the database N times, for the level N of the expanded loop road. Thus, method 2 was more stable at detecting loop roads compared with method 1.

These results suggest that the proposed system is better for Web map services than existing systems.

VI. Application

This section describes the application of the loop road extraction method to satisfy Requirement 4 in Section I.

So, we have proposed and developed the Focus+Glue+Context type fisheye view maps Emma [7][8][9]. Emma is the first Fisheye view map system for Web map services. Figure 8 shows that Emma has a Focus to show large-scale maps of target areas, a Context to show a small-scale map, and a Glue that shows the routes connecting Focus with Context. Unlike existing fisheye views [10][11][12][13][14], the Focus and Context have no distortion, because Glue contains all the distortion. Only the Glue must be drawn dynamically, so this system is rapid and suitable for Web map services. Emma users can observe detailed maps of target areas and the geographical relationships between the targets. Users can also adjust the size, scale, and position of the Focus according to target area, by mouse-dragging the edge of the Focus.

The Context and Glue is too small when the Focus is too large when using a small display, so it is important to adjust the size, position, and scale of the Focus to make the Focus as small as possible and fitting the Focus in the target area. Users cannot use a mouse with mobile maps, such as smart phones, which makes it is difficult to adjust the Focus manually. Therefore, we must adjust the Focus automatically based on the target area.

Thus, we propose a “One Click Focusing” function to automatically adjust the size, position, and scale of the Focus, based on the target city blocks. We adopted a server-client model using Web API. The algorithm of this function is as follows.

- Step 1 A user clicks any point on the map in the client.
- Step 2 The client submits the clicked point to the server using a Web API function.
- Step 3 The server calculates a loop road for the clicked point and responds to the client in XML format.
- Step 4 The client finds the center position P and the radius R of a circle that contains the loop road based on the XML response.
- Step 5 The client adjusts the Focus, based on the position P and radius R.

In Step 5, the center of the Focus, F_P, and the scale of the Focus, F_S, are calculated using the following functions where the radius of the Focus is F_R, which is limited by the display size.

$$F_P = p \tag{1}$$

$$F_S = k \cdot F_R / r \tag{2}$$

Table IV
MEAN LOOP ROAD EXTRACTION TIME AND PRECISION FOR EACH METHOD. DATABASE TIME MEANS THE TIME OF ACCESSING DATABASES. CALCULATION TIME MEANS THE TIME OF CALCULATING ROAD NETWORKS. UNIT IS MILLISECOND. METHOD 2 IS PROPOSED METHOD.

	database time		calculation time		total			precision
	mean	std.	mean	std.	mean	std.	factor	
method 1	931.1 ms	258.8	12.2 ms	15.6	943.3 ms	259.9	51.0×	92.3%
method 2	18.5 ms	6.2	0 ms	0	18.5 ms	6.2	1.0×	98.0%

Table V
EXPANDED LOOP ROAD DETECTION TIME USING EACH METHOD, FOR EACH LEVEL OF THE EXPANDED LOOP ROAD. DATABASE TIME MEANS THE TIME REQUIRED TO ACCESS THE DATABASES. CALCULATION TIME MEANS THE TIME REQUIRED TO CALCULATE THE ROAD NETWORKS. UNIT IS MILLISECOND. METHOD 2 IS PROPOSED METHOD.

		database time		calculation time		total		
		mean	std.	mean	std.	mean	std.	factor
level 1	method 1	1212.3 ms	428.9	414.1 ms	2098.6	1626.4 ms	2096.4	14.8×
	method 2	106.6 ms	8.2	3.3 ms	3.1	109.9 ms	9.12	1.0×
level 2	method 1	1174.3 ms	416.7	1540.0 ms	3532.0	2714.7 ms	3684.5	17.2×
	method 2	149.8 ms	8.4	8.2 ms	9.7	158.1 ms	14.8	1.0×
level 3	method 1	1233.9 ms	401.1	2085.2 ms	3977.3	3319.1 ms	3989.4	16.4×
	method 2	190.2 ms	31.62	12.2 ms	20.2	202.4 ms	36.5	1.0×
level 4	method 1	1181.9 ms	369.3	5329.0 ms	8296.7	6510.9 ms	8321.3	25.3×
	method 2	238.0 ms	32.7	19.3 ms	19.5	257.3 ms	39.3	1.0×

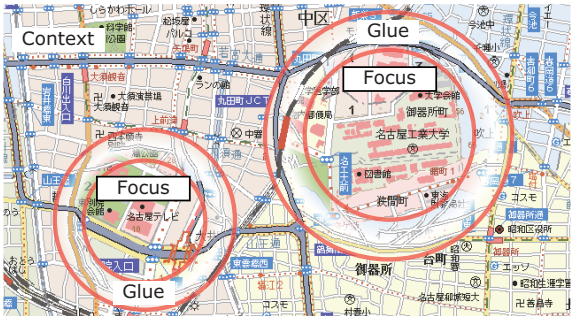

Figure 8. "Focus+Glue+Context" fisheye view maps in Emma. Users can adjust the size, position, and scale of the Focus by mouse-dragging.

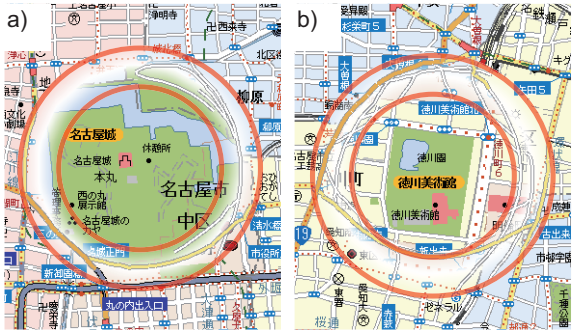

Figure 9. "One Clicking Focusing" function can automatically adjust the size, position, and size of the Focus, based on loop roads. a) The route to the target area is interrupted, because part of a loop road is not shown in the Focus. b) The route to the target area is connected to the target area, because loop roads are shown in the Focus.

k is a constant determined by the display resolution. If users want to expand the Focus, the Focus can be expanded step-by-step by acquiring expanded loop roads, instead of a loop road.

The advantage of the proposed method is that the Focus can be part of target area and also boundary roads of the target area, because the size of the Focus is determined by the loop roads. Therefore, the proposed system can find the routes for the target area of the Focus with certainty, as shown in Figure 9. This suggests that we can develop advanced Web map services by applying the loop road algorithm to existing Web map services.

VII. RELATED WORK

One of the most advanced Web map services is the Open Street Map [15]. Open Street Map enables users to easily edit maps using Web browsers like a Wiki. Users can modify and add roads in the Open Street Map, but users cannot edit and control the map based on city blocks or areas, as described in our current study.

Many studies have detected roads by analyzing paper maps or satellite images. For example, satellite images have been used in many studies in the academic fields of geo-science and pattern recognition, particularly in methods for detecting roads by recognizing road edges [16][17], pattern matching [18][19], and methods based on local coidentity of roads [20]. These methods are effective when a user wants to generate new maps for an area that has no road maps. Our method is effective for areas found in road map databases, which contrasts with these image-based methods.

In some field of academic study, such as graph theory [21] and mobile distributed network theory [22], it is popular to detect cycle graphs. In particular, when a network is down due to a loop problem where a data packet goes through the same routes many times if the system dynamically builds the network, such as occurs with cycle graphs in mobile distributed networks. Thus, there are many approaches for

 ISBN: 978-1-61208-178-6

generating directed acyclic graphs (DAG) without cycle graphs, which contrasts with our approach. These studies are relevant to our proposed method, but the definition and purpose of the loop road are different from cycle graphs. For example, a cycle graph may not be the smallest network surrounding any point, which contrasts with a loop road.

VIII. Conclusion

This study proposes rapid methods for extracting loop roads and expanded loop roads by constructing a loop road database from an existing road database, for use in Web map services. The proposed method can find a loop road 51.0 times faster than the previous method, and an expanded loop road 16.4-25.3 times faster than previous method. The precision of the proposed method (98.0%) is higher than that of the previous method (92.3%). The proposed method can effectively build a loop road database by avoiding duplications and it constructs a loop road database 3.86 times faster than conventional method. We developed a Web API, which allows Web engineers to develop Web Map services based on city blocks. We also developed a One Click Focusing function as an example application of loop roads. This function can automatically adjust the size, scale, and position of the Focus, according to the target area and city blocks. This system is particularly effective for mobile maps on smartphones.

This system allows Web map services to handle maps based on surfaces, such as city blocks, which contrasts with existing Web map services. We consider that our study contributes to the interface of mobile Web map services, such as One Click Focusing, but also to novel Web map services based on surfaces, such as city blocks. The feature of the proposed system is calculation using SQL technology mainly, so we believe that the processing speed of the proposed system could be further increased by adopting other database technologies, such as replications, which are popular technologies used in Web services. Thus, our system contributes to GIS and novel Web services.

Acknowledgments In the development of the prototype system, we were supported by Yahoo Japan Corporation. We would like to thank Yahoo Japan Corporation. This work was supported by JSPS KAKENHI 20509003.

References

[1] D. Yamamoto, I. Takumi, and H. Matsuo, “Location-based social network services employing student cards for university,” in *Proceedings of the 2009 International Workshop on Location Based Social Networks*, 2009, pp. 21–24.

[2] M. Arikawa, S. Konomi, and K. Onishi, “Navitime: Supporting pedestrian navigation in the real world,” *IEEE Pervasive Computing*, vol. 6, no. 3, pp. 21–29, 2007.

[3] “National land numerical information downalod service,” http://nlftp.mlit.go.jp/ksj-e/index.html, Nov. 8, 2011.

[4] “Tiger data,” http://www.census.gov/geo/www/tiger/, Nov. 8, 2011.

[5] D. Yamamoto, K. Hukuhara, and N. Takahashi, “A focus control method based on city blocks for the focus+glue+context map,” in *Proceedings of the IEEE 24th Internatinal Conference on Adavanced Information Networking and Applications Workshops*, 2010, pp. 956–961.

[6] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, “A characterization of ten hidden-surface algorithms,” *ACM Computing Surveys*, no. 1, 1974.

[7] D. Yamamoto, S. Ozeki, and N. Takahashi, “Focus+glue+context: An improved fisheye approach for web map services,” in *Proceedings of the ACM SIGSPATIAL GIS 2009*, 2009, pp. 101–110.

[8] N. Takahashi, “An elastic map system with cognitive map-based operations,” *International Perspectives on Maps and the Internet, Michel P. Peterson (Ed.), Lecture Notes in Geoinformation and Cartography*, pp. 73–87, 2008.

[9] “Focus+glue+context map,” http://tk-www.elcom.nitech.ac.jp/demo/fisheye/, Nov. 8, 2011.

[10] M. Sarkar and M. H. Brown, “Graphical fisheye views of graphs,” in *Proceedings of the CHI 92 conference on Human factors in computing systems*, 1992, pp. 83–91.

[11] M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss, “Stretching the rubber sheet: a metaphor for viewing large layouts on small screens,” in *Proceedings of the 6th annual ACM symposium on User interface software and technology*, 1993, pp. 81–91.

[12] L. Harrie, L. T. Sarjakoski, and L. Lehto, “A variable-scale map for small-display cartography,” in *Proceedings of the Symposium on GeoSpatial Theory, Processing, and Applications*, 2002, pp. 8–12.

[13] C. Gutwin and A. Skopik, “Fisheye views are good for large steering tasks,” in *Proceedings of the CHI 2003 conference on Human factors in computing systems*, 2003, pp. 5–10.

[14] C. Gutwin and C. Fedak, “A comparison of fisheye lenses for interactive layout tasks,” in *Proceedings of the Graphics Interface 2004*, 2004, pp. 213–220.

[15] M. Haklay and P. Weber, “Openstreetmap: User-generated street maps,” *IEEE Pervasive*, vol. 7, no. 4, pp. 12–18, 2008.

[16] Y. T. Zhou, V. Venkateswar, and R. Chellapa, “Edge detection and linear feature extraction using a 2-d random field model,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 84–95, 1989.

[17] C. Steger, “An unbiased detector of curvilinear structures,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 113–125, 1998.

[18] R. Bajcsy and M. Tavakoli, “Computer recognition of roads from satellite pictures,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-6, no. 9, pp. 623–637, 1976.

[19] W. Shi and C. Zhu, “The line segment match method for extracting road network from high-resolution satellite images,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 2, pp. 511–514, 2002.

[20] J. Hu, A. Razdan, J. C. Femiani, M. Cui, and P. Wonka, “Road network extraction and intersection detection from aerial images by tracking road footprints,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 12, pp. 4144–4157, 2007.

[21] N. Biggs, *Algebraic Graph Theory 2nd Edition*. Cambridge Mathematical Library, 1994.

[22] V. D. Park and M. S. Corson, “A highly adaptive distributed routing algorithm for mobile wireless networks,” in *Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 1997, pp. 1405–1413.

 ISBN: 978-1-61208-178-6