Object-Indexing: A Solution to Grant Accessibility to a Traditional Raster Map in Location-Based Application to Accomplish a Location-Based Task

Thitivatr PatanasakPinyo[1], Georgi Batinov[2], Kofi Whitney[2], Adel Sulaiman[2] and Les Miller[2]

***Abstract*— In various well-known location-based applications like Google Maps® or Yelp®, a favored type of map that has been implemented is a raster map. A raster map is a map that represented by a bitmap image. Using a raster map yields multiple advantages. It provides faster launching time and non-complicated implementation as well as facile application maintenance. However, the obvious flaw of a raster map is that it has lacked accessing ability, which is an ability that allows a user to access a map component such as a street, an intersection, a water way, etc. This ability is necessary in several professional location-based tasks. We propose a solution, called "Object-Indexing", that allows a traditional raster map to have accessing ability. Both theoretical concept and actual implementation of Object-Indexing are discussed in this article. The implementation was done by developing a location-based application for the address verification task, which is a professional location-based task of the US Census Bureau.**

***Keywords*— location-based application, raster map, accessibility, ESRI Shapefile, human computer interaction, interactive software.**

I. Introduction

A location-based application is an application that its contents are location-related. Thus, a map is a key feature that any location-based application must have provided on its user interface (UI). Good examples of location-based applications are Google Maps®, Yelp®, The Weather Channel®, and Expedia®. A raster map, which is a map that is implemented using a bitmap image, has been the undisputedly first choice for most location-based application developers since it requires less computing power. The cost of loading a raster map on the UI is equal to the cost of loading a bitmap image, which is not significant given the computing power of many devices sold in markets nowadays. Nevertheless, one major drawback of a raster map is that it does not have an ability that allows an application user to directly access map components. A map component is an object/unit on the map that representing important entity such as a street, a building, a railway, a lake, etc. Accessing map components is essential and required in several professional location-based tasks. One good example is the address verification task, which is the task of the US Census Bureau. For address verification task, the task executer (the user of the application) has to be able to access a street on the map to get the information of that street such as a street name, a street type, or a length. This paper proposes a solution to combine both advantages of a raster map and accessing ability together by introducing a new approach, called "Object-Indexing", which can empower a traditional raster map with accessing ability and preserves all map functions.

Thitivatr PatanasakPinyo[1] , [1]Faculty of ICT, Mahidol University, Nakhon Pathom, Thailand, (thitivatr.pat@mahidol.edu).
Georgi Batinov[2], Kofi Whitney[2], Adel Sulaiman[2] and Les Miller[2] ,
[2]Department of Computer Science, Iowa State University, USA
(batinov, kwhitney, aadel, lmiller)@iastate.edu

II. Problem

Two types of maps that were always selected to implement in a location-based application are raster map and vector map. A raster map is implemented by using a bitmap image as a map on the UI [1]. Whereas, a vector map is a map that is implemented by drawing a map based on the data record, called shape record, being read from the Shapefile [2, 3] or other sources that contain shape records. Shapefile [2, 3] is a non-readable file that contains multiple shape records such that each record is corresponding to geospatial vector data (line, point, or polygon). Shapefile is a popular resource for the development of geographic information system (GIS) applications [4-6]. Using a Shapefile to create a vector map in location-based application allows the map to have an indexing ability since every map component must be read and generated from a corresponding record in the Shapefile. However, handling a Shapefile is not easy, especially when a Shapefile contains many records. Chellappan [7] and Chellappan et al. [8] developed a location-based application with a vector map. The scope of the map in their application covered the area of Ames, a small city of Iowa, USA. In the application, the map was drawn by reading each shape record from the Shapefile. One major drawback that was found is the launching time of the application. The application took a significantly large amount of launching time due to the overhead of reading and drawing each map component. Furthermore, when a user operated (pan/zoom) a map, the application had to re-draw the map by reading each shape record again. Therefore, the user had to wait a significant amount of time not only at the launching of the application, but every time an operation caused the map to be re-drawn such as pan operation or zoom operation. This is an unavoidable disadvantage that hurts a location-based application implementing a vector map.

https://doi.org/10.15242/IJCCIE.H0418106

III. Concept And Methodology Of Object-Indexing

The fundamental concept of Object-Indexing is to consolidate a raster map with shape records from the Shapefile. These shape records are keys to grant accessing ability to the raster map. Every shape record would be read to get all information of the shape such as a set of latitude-longitude coordinates. An object corresponding to each shape record then would be created to store the read information. Since the pilot application that we implemented Object-Indexing is an application for the address verification task, every object created from shape record would represented a street, which is one type of map components. A street is the only map component that is relevant to the task because it is an informative item required by application users (address verifying officer). However, in the location-based application for other tasks, objects that representing other map components might be created according to the task's scope, e.g., railway or water way in a transportation-related application.

Fig. 1: Structure of Street Class

We defined the class for street object (Fig. 1). The class consisted of two fields, which were a street name and a set of pairs of latitude and longitude that the street lay on. After we completed reading every shape record from the Shapefile, we got a collection of many street objects. Note that we created only street objects that actually located on the map of the application, i.e., if our application was for address verification task in the city of Ames, Iowa, we created only street objects representing streets in Ames rather than creating street object for every street in USA. Next step was to abstractly combine a raster map, which was just a normal bitmap image representing the map, and the set of street objects. Conceptually, we can look at the final map as a map with two layers. The bottom layer was for the raster map. The top layer was for the set of street objects. This concept was illustrated in Fig. 2.

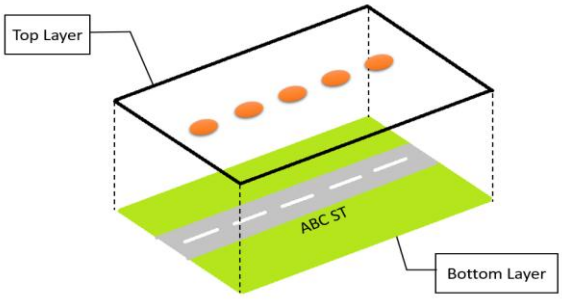

Fig. 1: Concept of Object-Indexing

In Fig. 2, the bottom layer is a bitmap image representing a map with one street, ABC St. The top layer consists of one street object representing ABC ST. Of course, this street object was created by reading shape records of ABC St. from the Shapefile. The street object has a set of pairs of latitude and longitude that ABC St. lies on. Each pair is represented by an orange oval. According to Fig. 2, there are five pairs of latitude and longitude that ABC St. lies on. However, the top layer that we illustrate is only for explanation purpose. We didn't actually draw those ovals on the raster map, i.e., this layer is invisible. Consequently, the question about how to bridge those two layers in the real implementation was raised.

IV. Implementation of Object-Indexing

Before discussing about the technique to bridge two layers together, we would first describe the setup of the implementation of Object-Indexing on a location-based application for address verification task. We deployed the application on Android platform. A touch-screen tablet that we chose to deploy the application was a Google Nexus 7 (2013), which supported location service (GPS) regardless of data connection. The area of the address verification was the neighborhood located next to the main campus of Iowa State University, Ames, Iowa, USA. Since our location-based application was for touch-screen tablet, the incident that allowed a user to access a street is to tap right on the street on the map. Once the user tapped, our approach first calculated the actual latitude and longitude of the tapped point (px, py) relative to the map where px was a horizontal distance from the left edge of the map and py was the vertical distance from the top edge of the map. This calculation can be done using a world file. A world file is a text file that contains five numbers required to transform any coordinate (x, y) on the map to the actual coordinate of (longitude, latitude). We will not talk about the calculation process in this paper. Each raster map must have its corresponding world file. The latitude and longitude that we got from calculation were the key to bridge both raster map layer and (invisible) street objects layer together. To check whether there was any street lay through the tapped point, we created a square (with predefined width), called region of interest, using the longitude and latitude as a center of the square. Thus, the region of interest would cover the tapped point. We then visited every street object. For each street object we visited, we checked whether there was any coordinate (longitude, latitude) that was inside the region of interest. If so, it meant that the street we visited was the street that the user tapped on. We finally displayed the information of the street on the UI, particularly, a street name. Knowing the street name by tapping on it was an example scenario of accessing ability. This approach was also implemented in the study of PatanasakPinyo [9] and PatanasakPinyo et al [10], which was about the adaptive UI of location-based application. The diagram in Fig. 3 illustrates the work flow of Object-Indexing approach when the user tapped on the map. The complexity of Object-Indexing is O(nm) where n is the number of street objects and m is the size

of the largest set of latitude and longitude pairs that belongs to one street object.

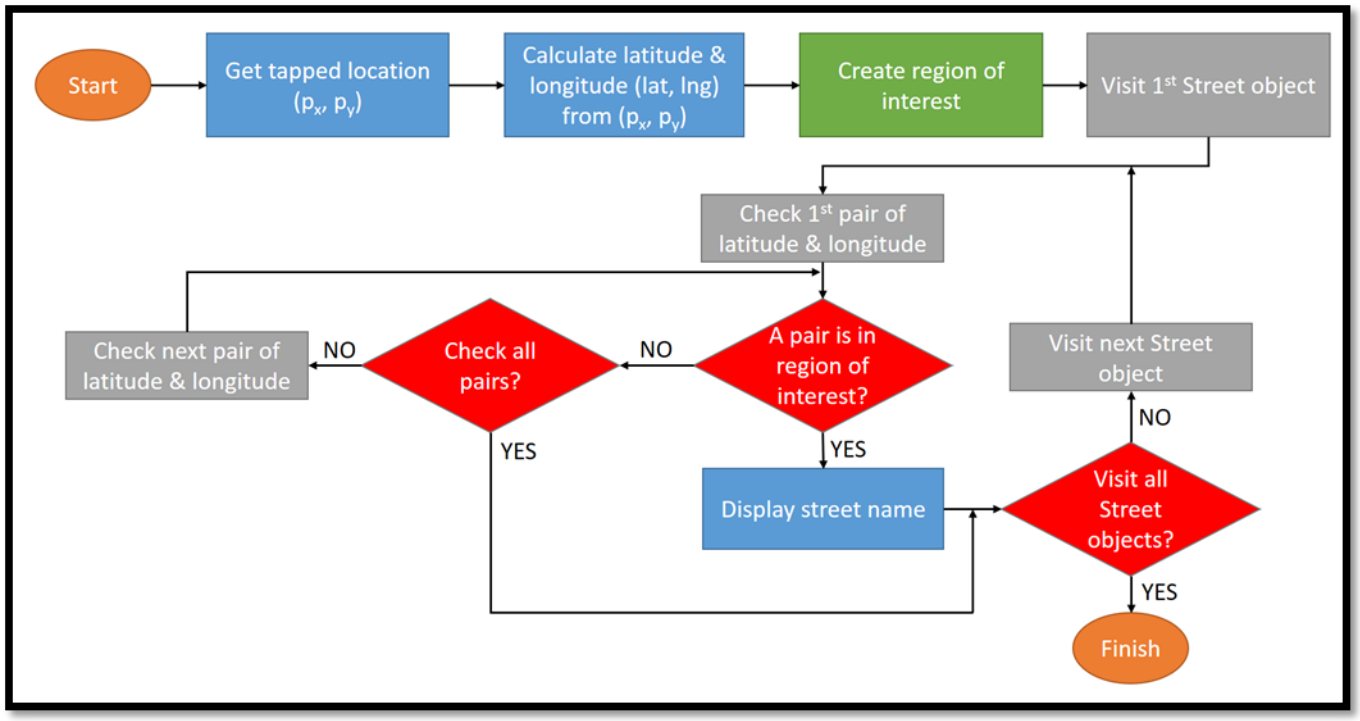

Fig. 3: Work Flow of Object-Indexing

Recall that when the application was started, it read every shape record and created the corresponding street object at that time. Every street object had been active until a user closed the application. With this methodology, the large amount of launching time would take place only once at starting. While the user was using the application, he/she can pan/zoom the map without suffering from the large overhead of map re-drawing, particularly, panning or zooming was just to change the map image. The amount of time for changing the map image was equal to the loading time of a single bitmap image, which was significantly small. Furthermore, the user had an accessing ability via Object-Indexing all the time while he/she was using the application.

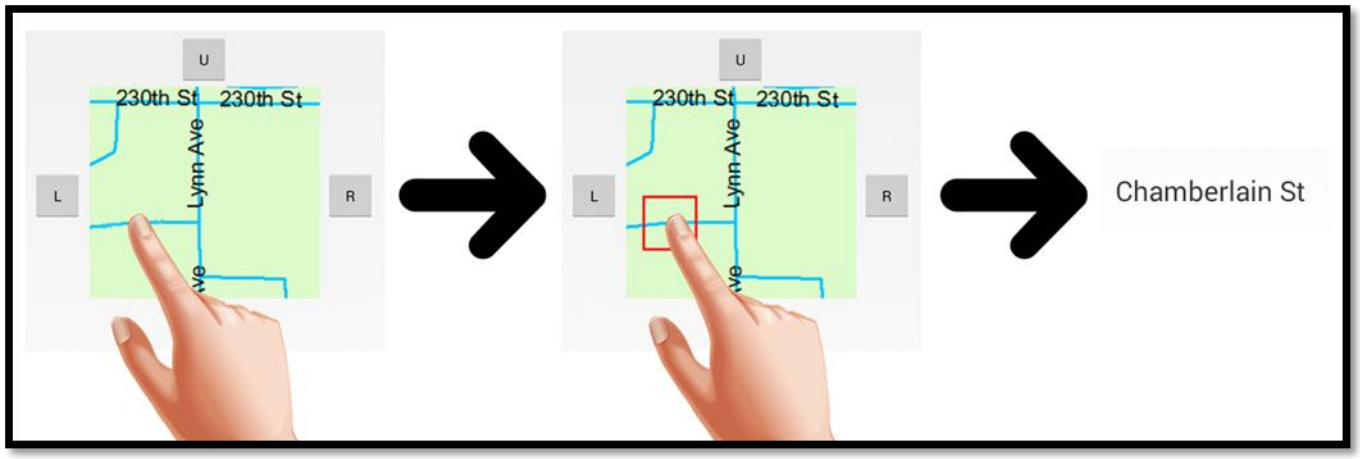

Fig. 2: Street Accessing by Object-Indexing

Fig. 4 illustrates how Object-Indexing allows the user to know the street information, i.e. street name, by tapping right on the street in the map. When the user tapped on an unknown street, Object-Indexing created a region of interest, represented by the red square, then it displayed the street name on the UI. Fig. 5 shows the actual series of screenshots of the application when Object-Indexing was triggered. In Fig. 5, a user tapped on an unknown north-south street at the top of the map. A region of interest was drawn to cover the tapped point. Object-Indexing applied the algorithm in Fig. 3 to detect the street. Finally, Object-Indexing displayed the street name at the bottom-left corner of the UI.

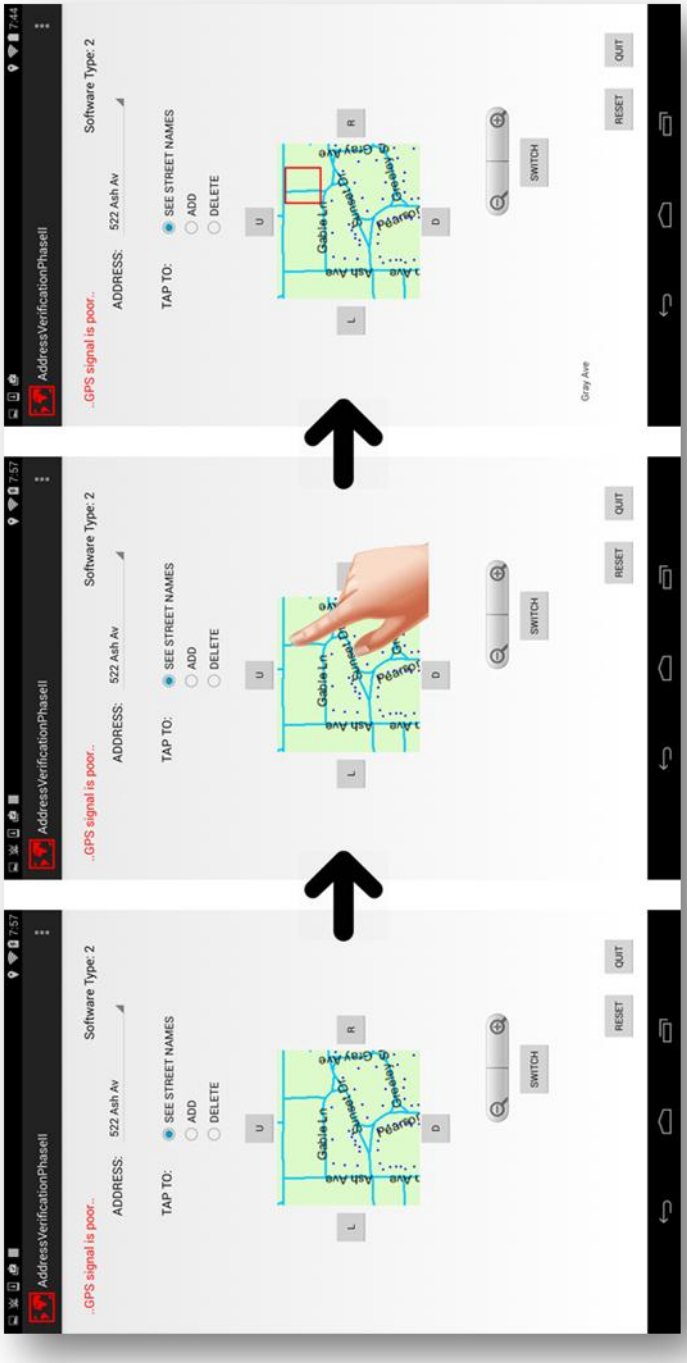

Fig. 5: A user tapped on Gray Ave

V. Discussion And Conclusion

We have developed Object-Indexing, which was the approach to combine the advantages of both raster map and accessing ability together. Object-Indexing equipped a traditional raster map in location-based applications with accessing ability, an ability that allowed an application user to directly access a map component such as street. This ability was essential and required in multiple professional tasks such as the address verification task of US Census Bureau. Although a vector map, which is a map that created by drawing a shape (line, point, or polygon) according to each shape record in the Shapefile, also supported accessing ability, it costed the application a large amount of overhead when launching and every time a map operation was performed. Object-Indexing overcame vector map by reading the Shapefile only once at the starting and creating a map component object corresponding to each shape record being read. It completely removed a waiting time when a map operation (pan or zoom) was done. This approach could be implemented in any location-based

application that takes fast response time and accessing ability as a top priority.

ACKNOWLEDGEMENTS

We would like to express our profound thanks to those who helped, advised, and supported us during the time of design, development, pilot study, and write-up of this paper.

REFERENCES

[1] M.N. DeMers, *GIS modeling in raster*, New York: Wiley, 2002.

[2] E. ESRI, "Shapefile Technical Description, Jul. 1998.," 1998.

[3] B. Stabler, "Read and write ESRI shapefiles.," 2005; http://ftp.auckland.ac.nz/software/CRAN/doc/packages/shapefiles.pdf.

[4] L. Anselin, et al., "GeoDa: an introduction to spatial data analysis.," *Geographical analysis*, vol. 38, no. 1, 2006, pp. 5-22.

[5] T.F. Rangel, et al., "SAM: a comprehensive application for Spatial Analysis in Macroecology," *ECOGRAPHY*, vol. 33, no. 1, 2010, pp. 46-50.

[6] J.C. Seong and J. Choi, "GEODIST: A C++ program for calculating geodesic distances with a shapefile.," *Computers & geosciences*, vol. 33, no. 5, 2007, pp. 705-708.

[7] C. S.K., "An object-oriented approach to maps," Computer Science, Graduate Theses and Dissertations, Iowa State University, http://lib.dr.iastate.edu/etd/12293, 2012.

[8] S.K. Chellappan and L. Miller, "An object oriented approach to dynamic survey unit maps," *Proc. The 31st international conference on computers and their applications (CATA2014)*, International Society for Computers and their Applications, 2014, pp. 273 - 278.

[9] T. PatanasakPinyo, "Flattening methods for adaptive location-based software to user abilities," Graduate Theses and Dissertations, Iowa State University, https://lib.dr.iastate.edu/etd/16191, 2017.

[10] T. PatanasakPinyo, et al., "Methods that flatten the user space for individual differences in location-based surveys on portable devices," *Proc. 1st International Conference on Computers and Their Applications (CATA 2016)*, International Society for Computers and their Applications (ISCA), 2016, pp. 65-70.