Cartographic Labeling

Michael C. Lin
University of Maryland
Undergraduate Honors Project
michaell@umd.edu

Date: April 2005

Abstract

Map labeling has long been a manual task done by cartographers. However, as more maps are accessed online in services such as MapQuest and Google Maps, maps need to be dynamically labeled based on interaction with the user. The SAND (Spatial And Non-Spatial Database) browser developed by the University of Maryland is a Geographic Information System that maps data attained from the government funded Tiger/Lines database. The purpose of this project is to research dynamic map labeling which will eventually be integrated into the SAND browser. Using the Tiger/Lines data for the city of Silver Spring, MD the project explores the difficulties in dynamically labeling cartographic data.

Introduction

Cartographic labeling is still done with human intervention in many cases. Because there is an aesthetic quality to how words are arranged around lines, a human eye is still the best judge of how a map should be labeled. The problem is simple to understand but complicated to solve because of the amount of data that must fit in a screen. In addition to the lines of the map, labels must be carefully drawn so that they are associated with their respective streets but do not interfere with the display of other streets. Decisions must be made as to which roads are labeled and which are not when there is a lack of real estate on the screen.

As Freeman[1] says, “Cartographers have refined the art of map making over hundreds of years and have established an extensive body of cartographic skills, conventions, and quality standards.” Developing cartographic text placement algorithms and software is a long term endeavor, requiring expertise in “pattern recognition, image processing, and computational geometry.” In this study we only begin to explore the process.

To demonstrate the text labeling problem, a mapping program was developed in Java to render and label the city of Silver Spring, MD. The program provides basic zooming and panning features to show the dynamic labeling of streets. It also allows the user to highlight or get detailed information on a street by clicking on any segment of the street.

Data Source: Tiger/Lines

The cartographic data provided by Tiger/Lines includes a collection of segments, identified by longitude and latitude. Each segment has a street name, type (Avenue, Road, etc.) and a code that identifies the classification of the road. The classification helps designate the importance of the street the segment belongs to. Below are three segments from the Silver Spring data.

```
Eastern Ave A40 0 0 {line -77.019173 38.97815 -77.02 38.9788}
Harper Road A40 20903 20903 {line -76.9858 39.0292 -76.98727
39.029846}
{Mount Quail} Road A40 0 0 {line -77.012109 39.027303 -77.011902
39.027775}
```

All three segments belong to roads that have the classification A40, meaning they are local, neighborhood, or rural roads.

Data Representation

To deal with all the unconnected segments from the Tiger/Lines database, a PM Quadtree is used. A quadtree is a data structure that extends the principles of a binary tree to two dimensions. By continually splitting a 2-dimensional area into four quadrants and placing shapes into the smallest quadrant containing them, fast and efficient data retrieval can be achieved. The PM Quadtree[2] is a variation of the concept tailored to work with line segments.

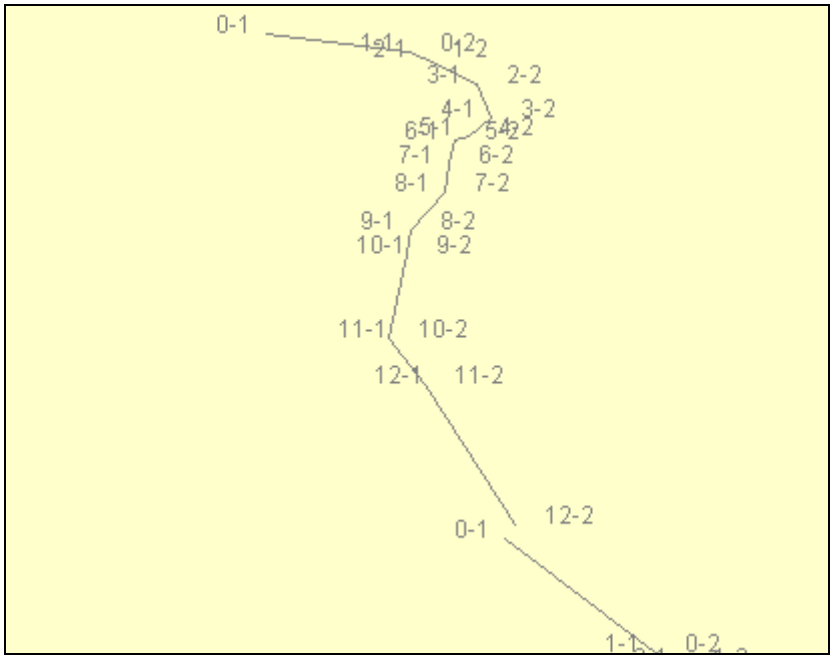

Figure 1: Connecting segments

Segments for each street are provided unsorted and do not always share endpoints. When the user requests a new region to view by either panning or zooming, all segments from the new region are pulled from the quadtree and reassembled into streets. Streets are made of multiple connected paths. Because streets often break in intersections, it is necessary to separate paths, and to label them separately. Different paths are also created when there are sharp angles in a street. As in Figure 2, the two highlighted segments on the right are part of Evergreen St., but the sharp angle between the two creates a break and two paths to be made.

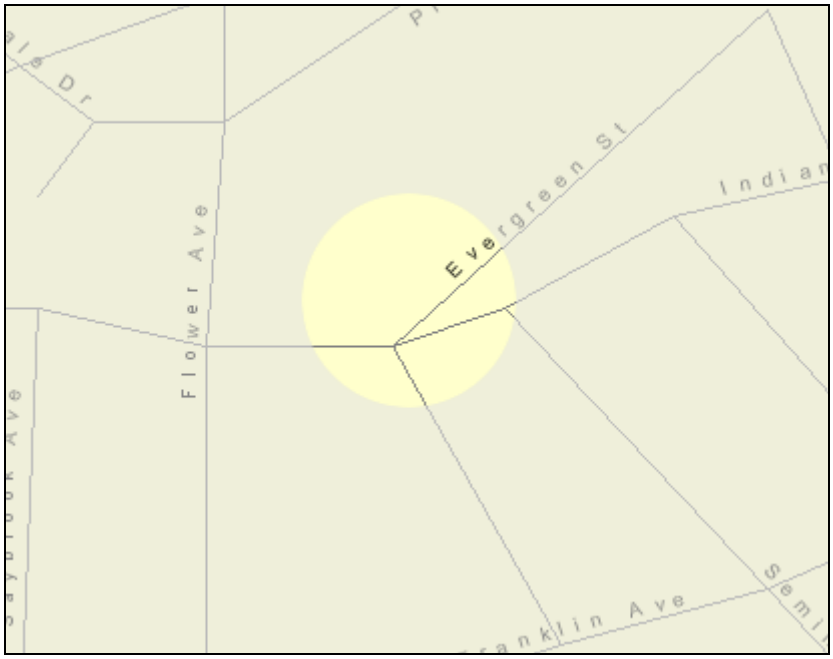

Figure 2: Breaking on sharp angles

For the purpose of this project, streets are identified by a combination of the street name, type and code. Given a larger map there could be a high chance of collision of those values, and streets would have to be identified by location as well. The reconstruction of streets is necessary because the system only renders visible segments, and new streets must be made on the fly to connect those segments.

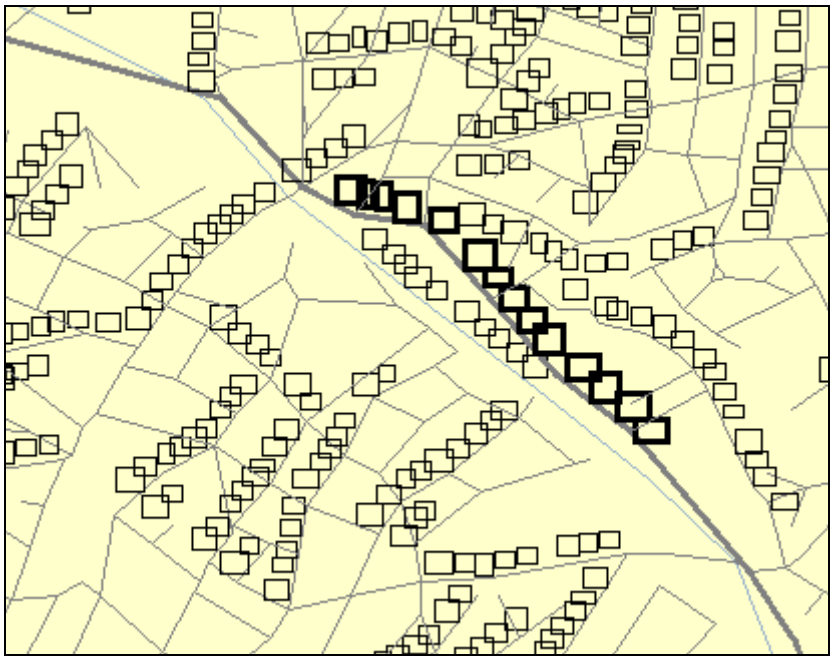

Figure 3: Bounding text for collision detection

A second quadtree is used when rendering text onto the map. A modified MX-CIF Quadtree is used to store the boxes bounding text drawn on the map. This quadtree variation that stores rectangles, allows us to simplify collision detection of map labels. The bounding boxes can be expanded by a fixed amount to create a larger box and more spacing between labels. The quadtree is modified to store intersecting boxes, which we allow when characters belonging to the same word are closely bunched.

Drawing the Map

Once a region has been chosen by the user, and streets constructed from the contained segments, the streets will be rendered. Which streets are rendered depends on the zoom level and the street code. The code provided by Tiger/Lines is used as an indicator for the level of importance for a street. More important lines are drawn with a thicker brush, and features such as railroads and bodies of water are drawn with different colors.

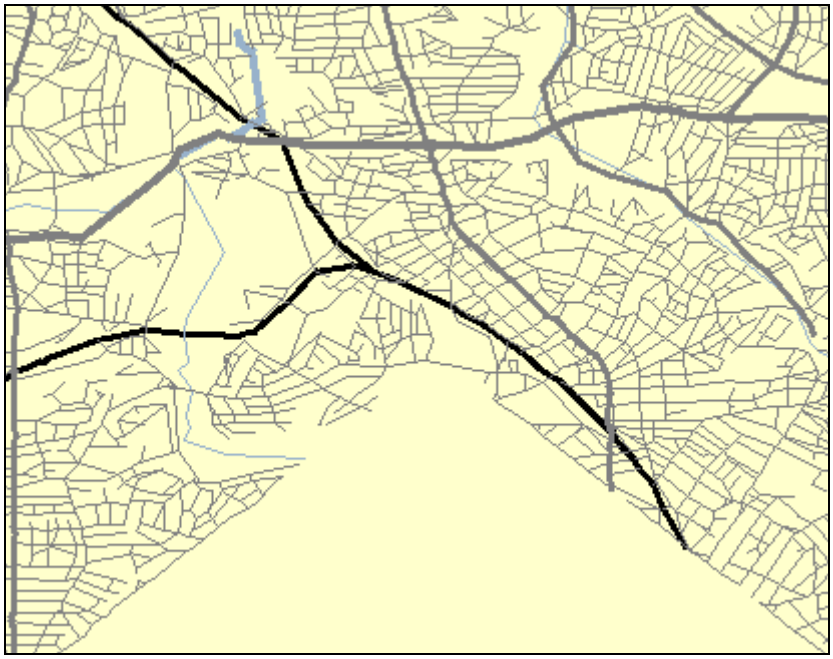

Figure 4: Drawing the map

There are more sophisticated means of determining the importance of a street, but that is a topic that goes beyond the scope of this project.

The Algorithm

The algorithm for text placement is a variation of the one Dr. Herbert Freeman[1] describes in *Automated Cartographic Text Placement*. The one used in this project performs the following steps until text is rendered without collision or the steps are exhausted:

1. Draw the text above the street and centered.
2. Draw the text below the street.
3. Vertically split the text with the street name on top and the street type below.
4. Reposition the text at the beginning of the street and repeat from Step 1. On the next iteration, shift the position by 1/5 of the street length, while the end of the street is not reached.
5. Suppress text.

Below is an example of the first three text placements possible to avoid collision.

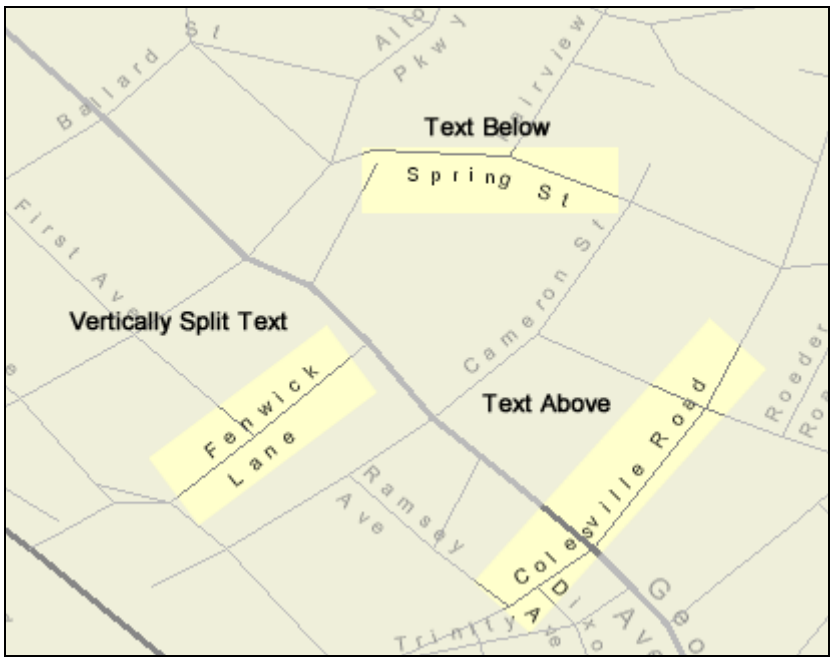

Figure 5: Three different placements

Text Rendering

Rendering text onto the map is the heart of this project. To begin this process we must first be able to render text labels for individual streets. Labeling a street is dependent on the correct construction of streets from the line segments provided. The order and grouping of the segments dictates how the text will flow.

Text labels are drawn in a character by character procedure. The default position to start labeling is the center of the street. The size of the font used depends on the street code. If the street is too small to fit the text, the system will start labeling from the beginning of the street instead of the center. If any character of the text fails to be placed because of overlapping, the labeling fails and the text will be suppressed.

Figure 6: Individual character rendering (illustration)

Text is rendered by character, and so the system will traverse the current path of the street until it finds adequate space to fit the character and spacing from the previous character. The text will be drawn at an angle obtained from the beginning and ending points of the traversal.

Results

The algorithm was tested on a map of Silver Spring, MD. Figures 7-9 show different levels of zoom and areas of the map. Text labeling is easiest at the highest and lowest levels of zoom. At the highest level of zoom there are few streets and enough space to adequately render the text labels for all the streets. At the lowest level of zoom there are few streets designated with a level of importance for rendering. It is the intermediate level of zoom when there are many streets to both draw and label.

The best step in dealing with collision is to move the text from above the street to below. Shifting the text along the street until there is no collision also works well. Previous versions of the project used a font shrinking step, but this technique was ineffective and dropped. It takes just one character collision to designate a street as overlapping with another, and shrinking the font did little to avoid collisions.

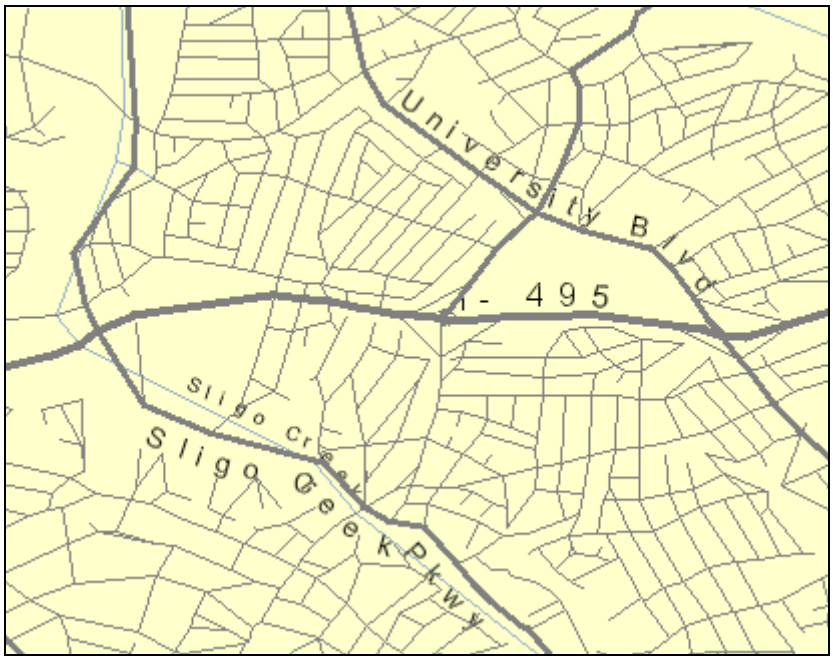

Figure 7: Low zoom level

Figure 7 is an example of a low level of zoom when few streets need to be labeled.

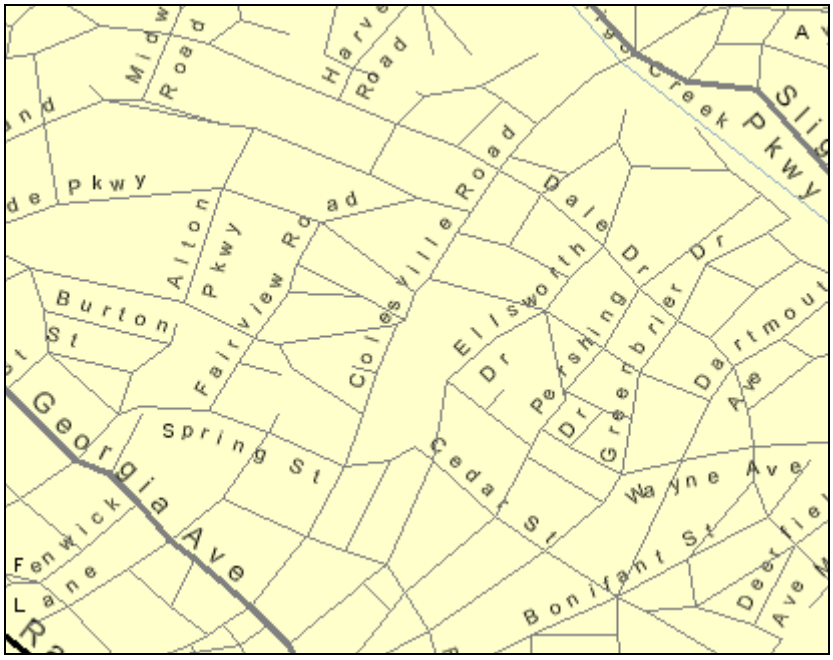

Figure 8: Intermediate zoom level

Figure 8 is an example of an intermediate level of zoom when crowding is the most pervasive. Because of the lack of space some street labels are suppressed.

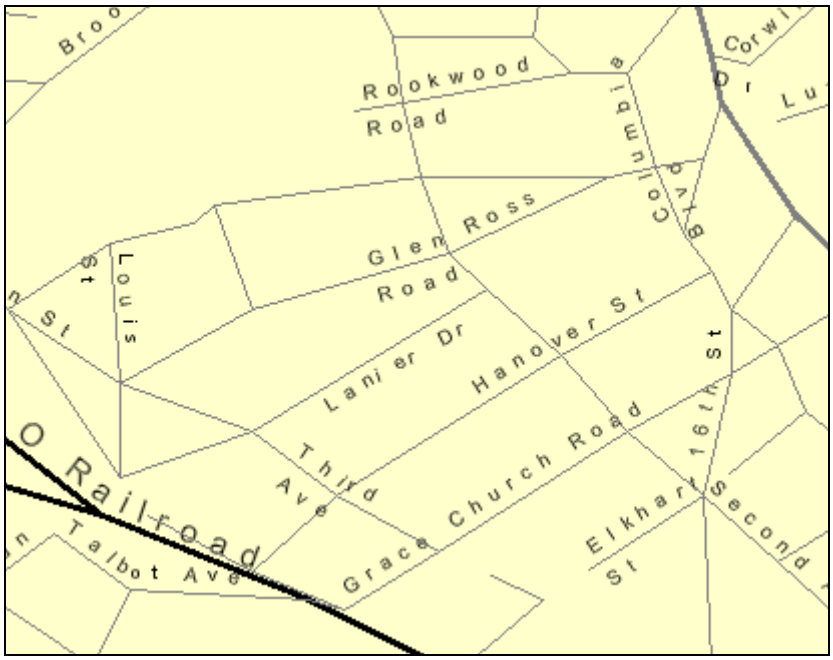

Figure 9: High zoom level

Figure 9 is an example of a high level of zoom, when crowding is not a problem. Few streets labels are suppressed at this level.

Future Enhancements

The next step for this project is to extend the repertoire of tactics for dealing with text label collisions. We could also implement steps Freeman suggests, such as labeling a street with a number that corresponds to a key. Font shrinking may also be implemented again if it complements the other new steps. We must also address limitations of this system. For instance the system currently does not label a street more than once, so it may be hard to find the label for a long street. The next version should label streets multiple times so users can easily follow the streets.

A more sophisticated system should also modify text placements already made, in order to accommodate a new street label. This could be done by re-running the placement algorithm on all labels intersecting the new label being placed. We could also use a pseudo-physics engine and allow labels to “push” each other around until they settle into place.

The final step in this project would be for the map to be labeled in a holistic manner instead of a street by street basis. Because we are dealing with aesthetics, the proper arrangements of multiple labels goes beyond just having non-overlapping text. Labels should also appear pleasing in context to the map lines drawn. Currently the project does not deal with lines overlapping text or text crowding without overlapping.

Conclusion

This project sets the groundwork for integrating text labeling into the SAND browser, which is an important feature for the browser. As with the SAND browser, the quadtree is the essential data structure in efficiently rendering maps and non-overlapping text labels on the fly. With text rendering, collision detection and basic techniques for dealing with collisions, new algorithms for arranging text labels can be easily added.

It is necessary to combine this project with an algorithm for deciding the level of importance of streets in order to present the data users want. To truly mimic the work of cartographers the project also needs to look at the map as a whole and not just as a collection of streets.

Acknowledgements

The author would like to thank his honors project advisor Dr. Samet for his direction and advice. The author would also like to thank Houman Alborzi for his contributions to the formation of the research problem and for our many conversations that guided this project and educated the author about the greater topic of geographical information systems. The Tiger/Lines data was obtained from the United States Department of Transportation, Bureau of Transportation Statistics.

Bibliography

1

Herbert Freeman.
Automated Cartographic Text Placement

2

Hanan Samet.
The Design and Analysis of Spatial Data Structures.
Addison-Wesley Longman Publishing Co., Inc., 1990.