12-2010

Mobile Map Browsers: Anticipated User Interaction for Data Pre-fetching

Benjamin T. Weber

MOBILE MAP BROWSERS:
ANTICIPATED USER INTERACTION FOR DATA PRE-FETCHING

By

Benjamin T. Weber

B.S. University of Maine, 2008

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

December, 2010

MOBILE MAP BROWSERS:
ANTICIPATED USER INTERACTION FOR DATA PRE-FETCHING

By Benjamin T. Weber

Thesis Advisor: Dr. Micheal F. Worboys

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Spatial Information Science and Engineering)
December, 2010

When browsing a graphical display of geospatial data on mobile devices, users typically change the displayed maps by panning, zooming in and out, or rotating the device. Limited storage space on mobile devices and slow wireless communications, however, impede the performance of these operations. To overcome the bottleneck that all map data to be displayed on the mobile device need to be downloaded on demand, this thesis investigates how anticipated user interactions affect intelligent pre-fetching so that an on-demand download session is extended incrementally. User interaction is defined as a set of map operations that each have corresponding effects on the spatial dataset required to generate the display. By anticipating user interaction based on past behavior and intuition on when waiting for data is acceptable, it is possible to device a set of strategies to better prepare the device with data for future use.

Users that engage with interactive map displays for a variety of tasks, whether it be navigation, information browsing, or data collection, experience a dynamic display to accomplish their goal. With vehicular navigation, the display might update itself as a result of a GPS data stream reflecting movement through space. This movement is not random, especially as is the case of moving vehicles and, therefore, this thesis suggests that mobile map data could be pre-fetched in order to improve usability. Pre-fetching memory-demanding spatial data can benefit usability in several ways, but in particular it can (1) reduce latency when downloading data over wireless connections and (2) better prepare a device for situations where wireless internet connectivity is weak or intermittent.

This thesis investigates mobile map caching and devises an algorithm for pre-fetching data on behalf of the application user. Two primary models are compared: isotropic (direction-independent) and anisotropic (direction-dependent) pre-fetching. A pre-fetching simulation is parameterized with many trajectories that vary in complexity (a metric of direction change within the trajectory) and it is shown that, although anisotropic pre-fetching typically results in a better pre-fetching accuracy, it is not ideal for all scenarios. This thesis suggests a combination of models to accommodate the significant variation in moving object trajectories. In addition, other methods for pre-fetching spatial data are proposed for future research.

ACKNOWLEDGEMENTS

The task of drafting, writing, and polishing this thesis to its current state has been a challenging yet an enjoyable and rewarding experience. Nonetheless, the entire process would not have been as successful without the continuing support of my family, friends, department colleagues, and most important, my faculty mentors.

I would like to give special thanks to my advisory committee, all of whom were always more than happy to offer their honest yet positive feedback and were genuinely committed to helping me accomplish my goal. I was fortunate enough to work for Mark Jadkowski at Global Relief Technologies beginning in 2008, whose experience in commercial GIS influenced several ideas presented in this thesis. I thank Max Egenhofer for his support and knowledge of mobile GIS. Nicholas Giudice's expertise in experimental design was also valuable in helping me simulate and test my hypothesis. I would also like to thank Michael Worboys for his support and role as thesis advisor, whose positive feedback kept me confident and focused on completing this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Mobile map browsers offer users opportunities to explore spatial data visually and interactively from their current physical perspective on cell phones, GPS units, or tablet computers. In mobile space, "here" prevails for the display of maps (Buttenfield, 2002) as users would typically ask for location information that originates from their current whereabouts, because that information relates most closely to their current task. There are certainly exceptions to this ego-centric paradigm, particularly when a user's task shifts to another geographic area.

<u>*Scenario 1*</u>

Emergency response field personnel are assessing flood damages in northern Maine. Using before and after satellite imagery, they deduce the surrounding areas with the greatest magnitude of damage. Meanwhile, data collected in the field are viewed on maps by operations management.

The scenario above exemplifies how spatial data viewed on a device, whether a mobile phone or a personal computer, may or may not be relevant to the user's current physical location. While field workers were concerned with accessing maps relevant to their current whereabouts, others were interested in maps of a remote area. Most often, however, mobile map users need pertinent to their immediate surroundings, areas they see from a vantage point, or places they are trying to reach from their current location.

Compared to their desktop counterparts, mobile devices are limited in memory, screen space, processing power, and network performance, and this impedes the interaction with mobile maps in the same responsive way often experienced on desktop computers. The lack of local availability of all spatial datasets that a user may need in the field is particularly constraining. One impediment is the sheer size of spatial data sets. For a 1x1 meter resolution coverage of the entire state of Maine (86,000 km^2), approximately 25GB of local storage would be needed for a set of tiled JPEG images that are 256x256 pixels and average about 20KB per tile. This number is just for the state of Maine at that particular resolution. Lower (and higher) resolutions would require even more storage space. As of 2010 the maximum data capacity for Apple's top-line iPhone is 32GB, clearly indicating that entire world map datasets cannot simply be loaded onto mobile devices for reliable data availability. The mere addition of more flash memory will not remedy this bottleneck, as each thematic variation—road map, soil map, weather map—will press further demands on the size availability. Therefore, only a subset of available spatial data is typically stored on a mobile device at any one time, and data outside of the available range need to be added on demand. The dynamic state of maps on a device is dependent of what portion of the globe is visible on the screen, typically controlled by the user or on-board sensors. For example, one can request that the map display an area to the east of what is currently visible, requiring an update of the display and data used to render it. One might also zoom in on the map, requiring the map browser to obtain data at a higher resolution for a more detailed map view. These user operations and how they relate to the data currently available on the mobile device, are described in Chapter 3.

Users expect responsiveness from visual interfaces, and this is a vital component of usability (Nielson, 1994). When data are unavailable for a portion of a requested map—for instance when a cellular data connection is lost—the resulting gaps in the map render the software less informative and, in some instances, completely unusable. The design and implementation of a responsive mobile map is challenging, primarily because it is impossible to know with one hundred percent confidence exactly where the device will be located in the future or know what the user's future task will be, making it difficult to ensure that the correct data exist on the device. A viable solution to this problem is to implement intelligent spatial data caching on behalf of the mapping application. *Caching* is a procedure that stores data temporarily in a memory buffer in order to speed up retrieval time (Alan et al. 1982). Reading data from a cache is much faster than remote retrieval, such as over the Web. Since data processed or rendered by a mobile map is location dependent, location is a reasonable basis for how the mobile map should manage what to store in the cache. Algorithms for traditional caching strategies do not consider spatial attributes of data but rather focus on temporal filtering of the data. Mobile map caching, however, is concerned with both temporal and spatial characteristics of the cached data and, therefore, requires tailored algorithms for managing the data.

1.1 Mobile Maps Scenario

Visualization of spatial data on a mobile device has relevance to a wide variety of applications:

Scenario 2

Field personnel are gathering data on a hurricane-stricken region of a country. Data collectors, who are assessing local damages from the disaster, use maps on their mobile devices for navigation and for geo-referencing a particular scene. As personnel navigate through flooded neighborhoods they may experience intermittent cellular connectivity due to the damaged infrastructure, leaving them without the necessary map coverage. Instead of downloading new data only on demand, alternative strategies are needed to intelligently pre-fetch map data for the emergency response field crew.

Scenario 3

On his iPhone, Alex queries his mobile map for directions to the local ski resort. The map draws a route and directs him towards the mountain. As Alex approaches the mountain he loses cellular connectivity, however, and the lack of on-demand map access over the air yields a blank map display.

Both data collection for emergency response efforts and navigation can rely heavily on visual, location-based software. In either scenario the cause of the incorrect map display is the loss of cellular connectivity, leaving the mobile map unusable. In order to correct this unwanted behavior this thesis investigates a number of ways to anticipate user interaction and movement through space in order to pre-fetch a dataset so as to prepare for unexpected loss of connectivity.

1.2 Mobile GIS Architectures

In a distributed system multiple information systems work together over a network in order to complete a task (Worboys and Duckham, 2004). In a client-server architecture, a client make requests for information from a remote server over a network. Conventional wired environments have the luxury of high-speed, reliable data access, whereas mobile environments suffer from intermittent connectivity and relatively slower transmission speeds (Jing *et al.,* 1999).

Important characteristics of networks are: network speed, reliability, throughput, and frequency of connectedness, and different computing applications lie within the spectrum of these characteristics (Figure 1.1).

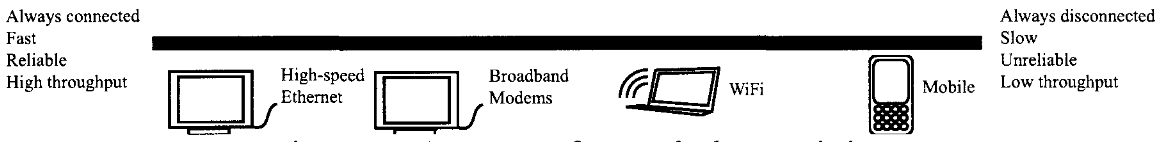


Figure 1.1. Spectrum of network characteristics.

Mobile environments are of particular interest because of their distinct behavior regarding connectivity. Relatively slow and unreliable connections distinguish mobile architectures from conventional desktop ones.

In a mobile environment with large data sets, the repository where data are stored and the location where the user interacts with the data are typically distributed spatially. The data are on a server (or sets of servers), while users initiate operations to display or query the data from their mobile devices. This distribution calls for a special attention to the architecture of such a system, with a particular focus on providing the fastest response times.

1.2.1 Mobile Data Storage

In order to circumvent the requirement of accessing data via wireless communication channels, one might implement a solution to preload a mobile device with all the spatial data necessary to carry out a user's tasks in the field. This setting normally allows for display and processing of the spatial data, but also offers the fastest response times possible for rendering, since the map browser need not wait for data from a remote server. Processing the data can be a bottleneck for mobiles equipped with relatively slow processing speeds, especially if the data is a vector format that requires geospatial overlay operations like buffering. Faster response times for such instances might be achieved by executing demanding operations on a faster CPU such as a remote server. This would, however, introduce the requirement of a data connection.

One major disadvantage of preloading all maps onto a mobile device is the lack of a dynamic dataset for users. Maps change at a varying temporal granularity and although some datasets might remain fairly static (e.g., a roadmap), other overlay datasets are more inclined to change at a high frequency (e.g., a traffic density overlay map). In addition to the lack of a dynamic map dataset, the sheer size of geospatial data puts a limitation on the volume of data that can be preloaded. Consequently, preloading requires knowledge of a user's location and task beforehand so as to load only pertinent data onto the device.

1.2.2 Data On-Demand

As an alternative to preloading all maps to a mobile device, one might implement a solution that relies on Internet connectivity to dynamically download maps onto the mobile device on-demand. With on-demand data access the user sees the most up-to-date datasets available. Each time the user requests a change in the map display, the map browser initiates a request for data from a remote server. The issues with preloading, namely storage requirements and prior knowledge of user location, are no longer relevant. Instead, the server sends only the required data to satisfy a request from the device.

Although on-demand data access is more flexible in terms of what maps are accessible to the user, several disadvantages stem from this approach. First, the assumption of continuous connectivity is not a realistic one. Although cellular data coverage is continually improving, global cellular coverage is not likely in the near future. *Dead-zones* continue to impact task performance of mobile device users through intermittent connectivity and unusable software. Satellite communications such as Broadband Global Area Network (BGAN) allow virtually global data access. Data rates from Internet Service Providers (ISPs) such as Inmarsat, however, are quite expensive and can cost around $8.37 USD per megabyte for the end user (Inmarsat BGAN Factsheet, 2009). Relatively limited bandwidth and slow throughput involved with cellular or satellite communicates also pose an issue to on-demand data access. Response times for map requests involves some latency which increases the overall response time of the map since the renderer must wait for the data prior to displaying it.

1.2.3 Hybrid Approach

In order to more closely enable the concept of "anything, anytime, anywhere" (Reichenbacher, 2004), a mobile map requires dynamic mobile data access via a client-server architecture (Jing *et al.*, 1999). Conventional wired environments have the luxury of high-speed, reliable data access, whereas mobile environments suffer from intermittent connectivity and relatively slower transmission speeds (Jing *et al.*, 1999); therefore, transmitting less data over unreliable networks seems ideal, but enough geographic coverage and detail are necessary to support the user's tasks. This thesis investigates how one can augment on-demand data access by anticipating user interaction and future locations to exploit available data connections for preloading map data to the device. Such an approach requires the use of pre-fetching strategies to predict the maps a user might need in the near future (Chapter 4).

The previous two approaches for exposing geospatial data on a mobile device did not address the data types used during transmission. While vector data would support on mobile devices not only the portrayal of spatial data, but also analytical operations, the display of raster images at the resolution of the screen typically suffices for purely visual tasks. Therefore, intelligent methods to transmit vector data, compressed or incrementally, (Bertolotto and Egenhofer, 2001; Buttenfield, 2002; Yang *et al.*, 2007) are complementary to the approach discussed in this thesis.

This thesis assumes client-server architecture for transmitting raster map datasets. Raster data in the form of common image formats such as JPEG or PNG are ideal for their simplicity and scalability. Tiled maps are a common web-based GIS data source in which pre-rendered raster tiles are used to render the map. Typically tiles are pre-rendered on the server, speeding up the request/response/display process. Tiles are simply replicated from the server's cache to the client's cache, at which point the map browser renders the data to the screen. At any particular time a mobile map browser can utilize a relatively small subset of tile images from the tile dataset repository available online and so the approach described herein refers to a tiled map implementation for its conceptual simplicity and programmatic efficiency (Chrisman, 1990). Map tiles are stored in the device's cache, or temporary memory buffer, in order to reduce latency for requests from the map browser.

1.3 Caching

Application responsiveness results from a number of factors, including the latency involved with retrieving data. In a distributed system, caching is meant to eliminate the need for multiple requests for data. Caching is ineffective and unprofitable unless it greatly improves performance of the application (Fielding *et at.*, 1999).

Cache memory, or caching, attempts to reduce the amount of time necessary to retrieve data for the requestor by storing frequently used data items to a local memory buffer (e.g., a local file). Computer processors use caching in program execution by storing frequently used program instructions in the CPU's cache. Doing so reduces the amount of time the CPU spends outputting data for those instructions (Handy, 1998).

Cache memory exists at a variety of levels. The fastest, but most expensive cache, is CPU cache, which stores data or program instructions for the CPU. Cache can also refer to any data stored in a computer's main memory (i.e., random access memory, or RAM). A computer's local file system can serve as a cache as well. When spatially distributed data repositories are accessed over the web, such as in the case when a web browser retrieves an HTML file from a remote server, it is much faster to read the file from disk than to re-request it over the Internet. Thus, resources that are frequently used by an application are better off copied and stored on the local machine, provided that there enough storage space to do so.

1.3.1 Database Management Systems Cache

Databases are commonly used to store massive amounts of data. Database management systems (DBMSs) create, maintain, and optimize a database for its users. A DBMS often consists of a cache layer, storing frequently-used portions of a database in main memory. The cache layer is a cost savings mechanism because it allows the DBMS to read from in-memory data as opposed to the files on the local disk.

1.3.2 Web-Page Cache

Caching web pages is a performance enhancing component common in web browsers. When a browser first requests a file over HTTP, it stores the file in either main memory or to a local file. The browser retrieves the cached version of a web page when it is re-requested. Navigating the web via a browser typically consists of a series of interactions, such as entering a URL, clicking links, or pressing the 'Back' button. Browsers can build a history of previously visited URLs, eliminating the need to re-enter them. Caching a formerly viewed page reduces the amount of wait if one were to request that page again using the 'Back' button common to most browsers.

1.3.3 Caching Algorithms

Computers have only a finite amount of storage space for caching data and, for this reason, software applications must use some mechanism to manage the cache when the allocated cache space reaches its maximum capacity. A cache replacement algorithm is a way for a program to optimize the cache. By following a set of cache replacement instructions, the algorithm ensures that space is available for new items at any time.

The least recently used (LRU) policy is a commonly used replacement algorithm. A cache is usually implemented as a collection of items with a specified maximum size. Once the cache exceeds the maximum number of items or data size yet the application needs to save a new item to the cache, the LRU algorithm first discards the least recently used item(s). Other cache algorithms include most recently used (MRU), psuedo-LRU, 2-way set associative, and least-frequently used (LFU) (Nagaraj, 2004).

1.4 Mobile Map Cache

One objective of this thesis is to investigate strategies for managing a mobile map cache, which includes both the removal of unwanted data and pre-fetching of anticipated data. A mobile map cache differs from other caches in that a spatial component now exists with the data. Location supplements time as the criterion for managing a cache. As a mobile device moves through space and time or a mobile map user interacts with the display, items are added and removed from the mobile map's cache. The addition and removal of data in the cache should follow not only the temporal locality but also the spatial locality of the data. For example, an LRU algorithm could effectively manage a spatial cache if the mobile device were to move in a straight-line path through space. If the device returns to a location previously visited, however, the LRU algorithm fails to optimize the cache with the most likely needed data. In this case the algorithm discards data that are near the current location of the device (Figure 1.2).

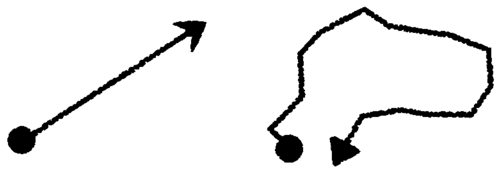


Figure 1.2. Straight line path versus revisiting path.

Managing a spatial cache involves keeping a history of location data coupled with time. Historical data allows the application to predict the future locations of the user or device. A prediction algorithm could, for instance, extrapolate a time series of coordinates sensed by the device, producing a set of possible future locations. Tracking user interactions with the map, such as panning, zooming and display rotation, also contributes a rich history dataset needed for prediction.

1.4.1 Cache Representation

This thesis focuses on information management for mapping in the mobile environment. In a setup where the client application and its main data source are likely to be distributed spatially, it becomes important to ensure the availability of resources to be consumed by the application. As with most client-server applications, the mobile client will access data items from the server and cache them in some local memory buffer to ensure the current and future availability of that item. For instance, a tiled map dataset allows a client to retrieve several tile images (e.g., JPEG, PNG, GIF), stitching them together to create a seamless, raster map display. In this instance the cache is managed as a set of tile images and the dynamic map display dictates the retrieval and removal of items in the cache.

On mobile devices one has the option to cache data items in several ways. Depending on the requirements of the software application one is designing for, a cache implementation might utilize a main memory object cache (e.g., a *list* of map tile objects), flash memory to write cached items to a file, or even a local database on the device to optimize the storage and retrieval of a large number of cached items. While flash or database storage can permit the storage of a substantial amount of data (e.g., flash storage cards > 16 GB), a mapping application may not always make use of this storage space for bulk downloading of data. Not only is data access for mobile devices hindered by intermittent connectivity but also by a variety of data access charges as well. Depending on the ISP, some devices might have an unlimited or very large data download limit whereas others are charged by the megabyte. Even in the case of satellite-based global coverage solutions (BGAN), high data transfer rates highlight the importance of minimizing data transfer in order to maintain reasonable data access costs.

In this thesis the cache management scheme focuses on a main memory cache for tiled images on mobile devices. Chapter 6 explains a prototype implementation that defines a memory cache of size *n* for temporary storage of map image tiles which could be extended to use other larger local memory allocations as well.

1.5 Research Questions and Hypothesis

The caching strategies presented in this thesis attempt to test strategies for reducing the latency issue that arises when one interacts with a mobile map display. The cache management implementation employs a pre-fetching and replacement policy to effectively utilize valuable wireless connectivity periods to retrieve data on behalf of the user.

In order to support this tile cache management, this thesis looks to answer a few questions:

Question 1: In what ways might one pre-fetch maps in anticipation of user interaction and device location?

Pre-fetching maps on behalf of the user is a critical component of mobile map cache management. Pre-fetching strategies make use of past device or map location(s) to estimate the near future locations of the device and thus identifying a pre-fetching dataset relevant to that location. Other strategies can also complement location prediction to further refine pre-fetching success.

Question 2: How much does it cost to pre-fetch maps (i.e., in terms of data size, time, memory space on a device, bandwidth, dollars)?

Question 3: What is the relationship between the amount of data a mobile map pre-fetches and the overall success rate of the pre-fetching.

Question 4: Is there some point where pre-fetching reaches a maximum effectiveness for a given strategy and allowable storage space?

Retrieving a data item over a wireless network is costly, especially with regards to time and storage space. Intuitively, pre-fetching maps should increase the reliability of having that map when needed in the future. However, pre-fetching maps can also become counter-productive, such as in the case of pre-fetching an item that is never used by the application. The prototype implementation measures the accuracy of both *isotropic* and *anisotropic* pre-fetching of map tiles, the latter taking into account the movement history of the user. Movement of a mobile user through space is assumed to be non-random (i.e., future locations are partially dependent on the current and previous locations). For example, when considering a car on a road network, the future location of the car is constrained by its current location, maximum speed, and the road network itself. Using this rationale the hypothesis for the tile cache management is as follows:

Hypothesis:

For non-random movement through space, the overall accuracy for a map tile pre-fetching algorithm is significantly greater when one incorporates past user locations to prioritize pre-fetching in the direction of user movement.

1.6 Results

Several simulations measure the tile pre-fetching accuracy for hundreds of real-world trajectories captured by tracking school buses and construction trucks in the Athens, Greece metropolitan area. The experimental testbed also models a tiled map accessed over the Internet. Using caching strategies proposed in Chapter 4, each simulation varies the volume of data allowed for both isotropic and anisotropic pre-fetching. When pre-fetching prioritizes tiles in the direction of movement the pre-fetching accuracy is on average 3.53% greater than when pre-fetching for an assumed stationary device. In addition, although pre-fetching more data should yield greater reliability, the complexity of the trajectory (i.e., the tendency of the trajectory to change directions) effects the pre-fetching accuracy and shows that a direction-dependent model is not always ideal. The simulations suggest that, for the selected strategies and algorithm used, pre-fetching should consider the interaction history of the map and pre-fetching data volume when deciding which pre-fetching policy(s) to employ.

1.7 Intended Audience

The intended audience of this thesis includes any researcher, developer, or other party interested in mobile data caching and may be of particular interest to any mobile GIS software developer. An exploration of spatial caching and data pre-fetching augments current commercial mobile mapping applications by increasing data availability in unreliable wireless environments and reducing the latency involved with over-the-air data transmission.

1.8 Organization of Thesis

This thesis is organized in the following manner:

Chapter 2 highlights previous research in areas pertinent to this thesis and how these works influence the ideas presented in this thesis. Chapter 3 explores mobile map interaction specific to a tiled map scenario, since experimental pre-caching strategies utilize this type of web data source. User interaction involves a dynamic relationship between the viewable map space on a mobile screen and the data required to generate a display for this space, an understanding that helps to formulate pre-fetching heuristics, discussed in Chapter 4. The heuristics propose a number of ways in which map data might be pre-fetched to exploit an available data connection. A critical part of managing a cache in any software application is cache replacement policy. Chapter 5 proposes a number of ways in which tiles in a cache are discarded so as to make room for new tiled data. The proposed pre-fetching heuristics and replacement policies are tested in an experimental prototype defined in Chapter 6. The chapter also details simulation results and analysis. Chapter 7 concludes the thesis with a summary of this thesis as well as proposed research for the future.

CHAPTER 2

MOBILE MAP FOUNDATIONS

Designing effective mobile solutions has been a topic of study for many years, especially since designers of such solutions must overcome many limitations inherently present in mobile environments. Although science is continually advancing and technology yields faster, more capable hardware and software solutions every day, many of the initial challenges with mobile computing still persist today and the solution lies only in the design of a system that can adapt to its adverse conditions. Section 2.1 discusses some of the ways in which caching can improve the limitations faced by mobile devices in unfavorable conditions. Section 2.1.2 discusses previous work with spatial data caching on mobile devices. The remainder of the chapter discusses visual interface design principles and interaction with map displays, which are later applied to a design for managing mobile map data. Also discussed are ways of tracking movement and interaction history for assisting with data management.

2.1 Mobile Challenges

This thesis focuses on data management for map browsers in mobile settings, both connected and disconnected. The fact that mobile computing has many fundamental differences relative to a static, desktop environment makes it a popular area of research. Satyanarayanan (1996) points out that mobile computing (1) is and and will continue to be resource-poor relative to desktop environments, (2) is hazardous based on its portability and security issues, (3) shows extreme variability in connectivity performance and reliability, and (4) draws from a strictly limited amount of power. The severity of mobile computing limitations is undoubtedly diminishing with the expansion of cellular network coverage and ever-improving hardware on mobile devices. One is hard-pressed to envision a day in the very near future, however, where every location on Earth has an equally reliable, high-speed data connection to the Internet and a mobile device with a virtually unlimited amount of storage space. Until this day arrives engineers must consider the challenges of mobile computing when designing a mobile software solution.

2.1.1 Mobile Caching

The number of mobile units communicating with server-side databases over wireless channels is growing significantly. To effectively handle the massive amounts of data being shared between client and server endpoints, a popular solution is to cache retrieved data locally to the device for repeated consumption. Not only does caching data locally preserve some of the finite amount of bandwidth available over wireless channels but it also prepares a device for unexpected interruptions with wireless connectivity and minimizes wireless communication costs.

Mummert *et al.* (1995) discuss how *hoarding* data to a mobile database in anticipation of intermittent connectivity can improve the overall performance and usability of mobile software. Lack of connectivity for a mobile client causes an increase in cache misses (i.e., requested data not locally available), which impedes the progress of a user's task. Users are also unlikely to tolerate poor software performance due to a loss of connectivity and their research models user patience and proposes that caching data to a mobile unit should be based on whether or not the time it takes to do so will cross the user patience threshold (i.e., the amount of time a user is willing to wait for an item).

When data are cached to a mobile database, the duration of its persistence is highly dependent on the type of data. Generally, one would assume that base maps (e.g., maps with political boundaries, rivers, buildings) are fairly static and change at a much less temporal granularity than news or weather. Some spatial data do change more frequently, however, such as traffic reports that can supplement static base maps. Barbara and Imielinski (1995) describe a cache invalidation framework for checking whether or not cached data should be discarded or updated. A challenge arises when considering invalidation for mobile device databases since mobile devices are often offline and only the server knows whether or not a cached item should be updated. In addition a server lacks knowledge of whether or not devices are connected or even powered on. Mobile devices can periodically communicate reports of their cache to servers to test which items are up-to-date.

2.1.2 Spatial Caching

This thesis is particularly interested in caching maps on mobile devices, inherently requiring a variation of methods used for caching non-spatial data. A *least recently used* (LRU) cache replacement algorithm sorts candidate items to discard based on the timestamp at which that item was last consumed from the cache. Algorithms such as these only compare the temporal locality of items in a cache to choose candidates for removal. With spatial data, however, cache invalidation should emphasize spatial locality of cached items as well (Dunham and Kumar, 1998). As a mobile device moves through space or a user changes the display on a map, data are retrieved from the cache or from a remote resource and then written to the cache. The least recently used data should not necessarily have the highest priority for removal since it may, in fact, have higher spatial similarity to the current location of the device than other cached items. An alternative approach might be to prioritize items in a cache for candidate removal using a *Furthest Away Replacement* (FAR) policy (Ren and Dunham, 2000).

2.2 Geographic Data Transmission

To more closely enable the concept of "anything, anytime, anywhere" (Reichenbacher, 2004), a mobile map requires dynamic mobile data access via a client-server architecture. Conventional wired environments have the luxury of high-speed, reliable data access, whereas mobile environments suffer from intermittent connectivity and relatively slow transmission speeds (Jing *et al.*, 1999); therefore, transmitting less data over unreliable networks seems ideal, but enough geographic coverage and detail are necessary to support the user's tasks. While vector data would support not only the portrayal of spatial data on mobile devices, but also analytical operations, the display of raster images at the resolution of the screen typically suffices for purely visual tasks. Therefore, intelligent methods to transmit vector data compressed or incrementally (Bertollo and Egenhofer, 2001; Buttenfield, 2002; Yang *et al.*, 2007) are a complementary approach to the ideas presented in this thesis.

In this thesis, raster tiles accessed from remote databases over wireless networks are the data domain for mobile map browsers. A tiled representation of large geographic spaces (Chrisman, 1990), which partitions the map area as discrete raster images, has the advantage of managing smaller pieces of data and smaller portions of the map at any one time.

2.3 Visual Information Browsing

Mobile map browsers are interfaces for browsing graphical spatial data on small-screen, mobile devices and, thus, require some degree of visual interface design. Mobile maps are graphics-driven in that they consume either raster or vector data and render them as an image on screen. Interaction with mobile map browsers is often with the graphics where the user directly manipulates the display.

Much work has focused on map interaction and information displays. Developing interactive mapping systems involves determining what control is needed over the map, the degree of such control, and how this control is implemented (Harrower and Sheesley, 2005). Mobile map browsers must be especially careful with these decisions since screen space is limited and methods for interaction have implications on the information display (e.g., less display space due to on-screen graphical buttons).

One important element of interactive graphical displays is how the information is displayed dynamically. For mobile map browsers a dynamic display should avoid information overloading by controlling the amount of information being displayed by relating scale and speed (Igarashi and Hinckley, 2000). Speed-dependent zooming suggests a direct relationship between speed and scale that determines the display of the graphical content (**scale = constant / speed**). Space-scale trajectories were studied to visualize the relationship between panning and zooming two-dimensional spaces (Furnas and Bederson, 1995). Optimizing these trajectories yields a dynamic display that is both "smooth" and "efficient" which is accomplished when a continuous transformation of the graphical display contains no sudden jumps of the image being displayed, improving user cognition (van Wijk and Nuij, 2003).

2.4 Trajectories and Object Tracking

Databases to store information on moving objects have been extensively studied in the past. Whether the object is a moving point (e.g., car, mobile phone, train) or a moving region (e.g., storm, disease), storage of movement history in a database allows for querying the location history at temporal points and intervals (Guting *et al.*, 2009).

Object tracking has gained much popularity due to the plethora of data that are obtainable from moving objects. Relatively cheap sensors like GPS and cellular network antennas allow for the acquisition and storage of huge amounts of moving object data on a daily basis (Chakka *et al.*, 2003).

Trajectories are typically represented by a data model which samples continuous movement of an object through space and time (Macedo *et al.*, 2008). Data models can be as simple as a series of space-time points or represent more complex, dynamically shaped spatial regions with holes (Forlizzi *et al.*, 2000).

2.4.1 Location Prediction

Perhaps the most beneficial uses of storing an object's location history is the ability to predict its near future location. Various mathematical models attempt to predict an object's future location and are of varying complexity. Based on a temporal sequencing of object positioning, location prediction is mostly effective for near-future location estimation only (Jeung *et al.*, 2008). Location prediction is based on an object's last movements and so the level of uncertainty with estimation grows as we move further into the future. Although some models can estimate future locations to some degree, real-world movement is often complex and is not easy to model with mathematical formulas, especially since real-world phenomena (e.g., road construction or traffic volume) has much influence on an object's movement through space and time (Jeung *et al.,* 2008).

The database community has proposed extensions to SQL which would enable one to query both past, present, and future locations of an moving object. Future temporal logic (FTL) uses temporal operators such as *Until*, *Nexttime*, and *Eventually* to perform spatio-temporal queries on objects with dynamic locations (Sistla and Wolfson, 1998). However, it is unclear just how accurate these prediction models are given the variety of movement scenarios.

Categories of location prediction models vary in both complexity and accuracy. Vector-based prediction uses linear and non-linear motion functions for a time-dependent estimation.

Linear models assume that an object's movement is linear and, thus, uses a linear equation to solve the prediction (Equation 2.1) (Saltenis *et al.*, 2000). The object's predicted location, $\boldsymbol{p}$, is dependent on its current location, $\boldsymbol{p_0}$, velocity vector, $\boldsymbol{v}$, and the amount of time into the future.

$$\boldsymbol{p} = \boldsymbol{p_0} + \boldsymbol{v}(\boldsymbol{t} - \boldsymbol{t_0}) \tag{2.1}$$

Non-linear models such as a quadratic functions are able to model the acceleration of a moving object as well as it's velocity. Even so, vector models do not capture curved movement which is often observed with moving objects. To remedy the limitations presented with linear and non-linear motion functions some have proposed using a recursive motion function (RMF), which can predict future locations using an objects "tendency" of movement such as linear, quadratic, or circular motion (Tao *et al.*, 2004). In addition to vector-based movement, location estimation is also possible through the use of pattern recognition. By discretizing space as cells, some have used Markov chain models to compute the likelihood of an object moving from one cell to any neighboring cells based on previous movements (Ishikawa *et al.*, 2004).

Optimizing a prediction model or implementing the most complex and accurate model is outside the scope of this thesis. Instead, later chapters discuss the use of location prediction and will use an estimation model for merely demonstrating its usefulness in mobile map cache management. Let it also be noted that the use of complex, computationally expensive location prediction models coupled with extensive tracking databases are better suited for implementation on servers equipped with the proper resources.

2.5 Summary

This chapter addresses some of the components and issues necessary to address for building a cache management framework for mobile map browsers. Mobile computing faces many challenges such as limited memory and network bandwidth and caching, when implemented correctly, offers a workable solution. The upcoming chapters propose a design for caching strategies based on how individuals interact with a mobile map and design principles for visual information interfaces. Historical logging of user interaction and device location also proves beneficial for managing a mobile map cache.

CHAPTER 3

MOBILE MAP INTERACTION

Building a management framework for mobile spatial data relies on an understanding of the relationship between the user and the mobile map browser. Maps have the ability to give the user a stronger awareness of the environment, typically by graphically displaying spatial data on a static medium (e.g., paper maps) or a dynamic medium (e.g., an electronic screen). On an electronic screen, a *scene selection* is the display of spatial data at any one point in time, established through the relationship between two rectangles: (1) the geographic data representation (map space) and (2) the visible portion of the data (viewport) (Jackson 1990). This chapter discusses the data used to generate a map display on a mobile device.

3.1 Reference Spaces

The process of rendering graphics needs to account for multiple spaces. A mobile map is a representation of the real world that is digitally preserved and used to generate an image

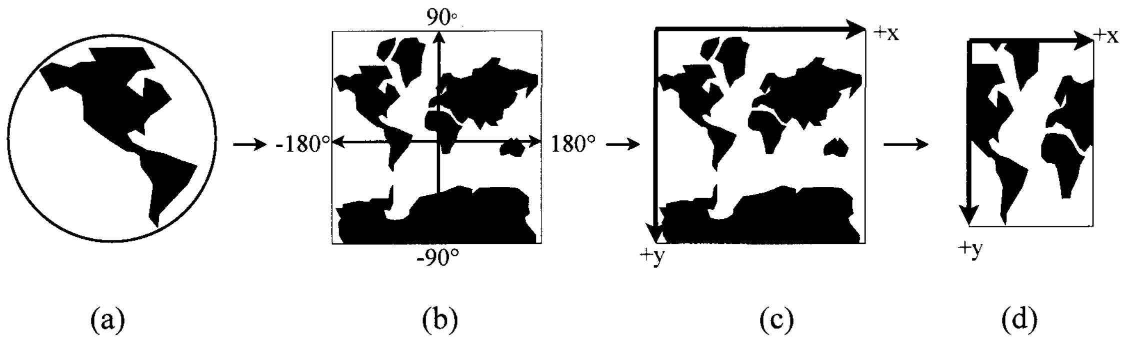


Figure 3.1: Reference spaces: (a) geographic space, (b) map space, (b) pixel space, and (d) screen space.

to display on a mobile device screen. This process occurs over several reference spaces: *geographic space*, *map space*, *pixel space*, and *screen space* (Figure 3.1).

3.1.1 Geographic Space

Geographic space (Figure 3.1a) refers to the three-dimensional space of the Earth that, in order to be portrayed on a mobile device, is often presented to the end user as a two-dimensional map. Representation as a digital map first requires projecting points from the Earth's surface (specified as spherical coordinates longitude, λ, and latitude, φ) onto the two-dimensional Euclidean plane (Bolstad 2005).

3.1.2 Map Space

Map space (M-space) (Figure 3.1b) is the planar representation of geographic space, defined by a projection (e.g., Mercator, Plate Carree, etc.) and often measured in geographic units of longitude (λ) and latitude (φ) (Snyder 1987). In addition to projection, map space holds several properties, such as the map space origin (O_M) and its geographic extent in the *x*- and *y*-directions (Δx_M, Δy_M). A coordinate in the map space is referred to as (x_M, y_M).

3.1.3 Pixel Space

A client-server architecture using pre-rendered raster images is an effective method for distributing mobile map data (Section 1.2). A core data requirement for this architecture is the translation of the map space into pixel space (raster images). Images are measured in pixels and use a reference system originating at (0,0) in their upper left corner with positive x to the right and positive y down (Figure 3.1c). Pixel space (P-space) is a finite two-dimensional array of pixels having an extent specified as (Δx_P, Δy_P), while (x_P, y_P) references a specific pixel in the space.

3.1.4 Screen Space

The final representation of geographic space is drawn to a mobile device screen. Currently, mobile screens are rectangular, two-dimensional arrays with a coordinate system similar to images. A screen's origin (O_S) exists in the upper left corner with positive x values to the right and positive y values down (Figure 3.1d). Screen space (S-space) has a width and height denoted w_S and h_S, respectively, while (x_S, y_S) references a specific coordinate in the space. Screen space is used to generate graphics.

3.2 Tiled Map

An image map has a particular resolution defining the amount of map space captured per pixel. To visualize large geographic areas on small screens, the resolution of the image will be coarse in order to fit a larger area into a small number of pixels. Conversely, detailed displays require a higher-resolution image in order to fit a small geographic area into the allotted screen space. Multi-resolution maps address this need of multiple levels of detail because the map space is portrayed as several images of varying resolution. Doing so permits small geographic features (e.g., roads, rivers, and cities) to be abstracted from view at low levels of detail, only displaying pertinent items at that level. Higher-resolution images show more detail, often including geographic features that were abstracted at a lower level. Single resolution maps are of limited use for interactive map interfaces as they provide no means to increase or decrease visible details.

High-resolution image maps with large geographic coverage inherently become too large to be handled by a single computer. A 10-meter resolution map of the world, for example, would require an image the width of about forty million pixels. A popular solution is to fragment the image map into smaller, equal-sized images called tiles. A *tiled map* is a multi-resolution representation of the map space implemented as several levels of detail known as *zoom levels*. Zoom levels are typically specified as an integer value (Z) within a predefined range (i.e., $Z_{min},<= Z <= Z_{max}$). Therefore, any reference to a pixel in pixel space (x_P, y_P) is relative to a specific zoom level. At any particular zoom level the map is a composition of non-overlapping tiles (Figure 3.2).

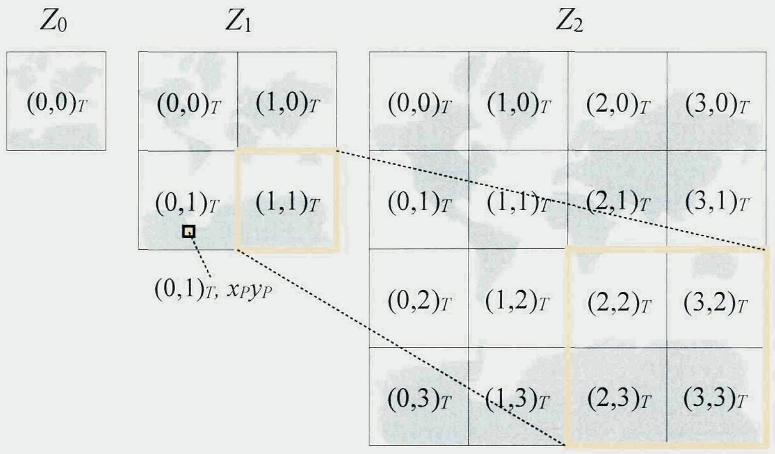


Figure 3.2: Tile coordinates.

Since a tiled map is a repository of pre-rendered map images, tiles need indexing to allow reference to them. Tiled map coordinates consist of three levels of specificity: (1) the zoom level, (2) the tile coordinates, and (3) the pixel coordinates (Figure 3.2). Zoom levels refer to an image map of a particular resolution, tile coordinates $(x, y)_T$ refer to a particular tile within a zoom level, $(0,0)_T$ being the upper-right tile of a tile layer, while pixel coordinates $(x, y)_T\ (x_P, y_P)$ refer to a particular pixel within $(x,y)_T$ (e.g., $(x, y)_T(0, 0)$ is the first pixel in $(x,y)_T$). Any pixel in pixel space (x_P, y_P) translates into a tile coordinate and pixel offset (Section 3.4).

3.3 Viewport

A viewport is the rectangular portion of the map space drawn to the mobile screen, thus displaying the viewable area of the map at any one point in time. Viewport properties (Table 3.1) are measured in map space and pixel space units and are translatable between coordinate systems (Section 3.4).

Property	***Description***
c_M, d_M	Viewport center in map space
c_P, d_P	Viewport center in pixel space
$\boldsymbol{\theta}_V$	Viewport rotation angle
Z_V	Viewport current zoom level
MBR_M	Viewport's minimum bounding rectangle (MBR) in map space
MBR_P	Viewport's MBR in pixel space (relative to current zoom level)
w_M, h_M	Viewport width and height in map space
w_P, h_P	Viewport width and height in pixel space

Table 3.1: Viewport properties.

The viewport's zoom level corresponds to the current tile layer from which the map browser displays data. The chosen zoom level is *closest* in resolution to the resolution of the screen. This selection comes from a function that compares the screen resolution, that is, the ratio of the viewport's map space extent to the screen extent (Equation 3.1) to each zoom level resolution, that is, the ratio of the map space extent to the width of the pixel space (Equation 3.2), returning the zoom level whose resolution is most similar to the viewport's.

$$Resolution_S = w_M / w_S \tag{3.1}$$

$$Resolution_P = \Delta x_M / \Delta x_P \tag{3.2}$$

The function ***getZoomLevel*** returns the zoom level as an integer value given the current viewport (Function 3.3).

```
function getZoomLevel : integer                                   (3.3)
        diff[]
        for $i$ = $Z_{min}$ to $Z_{max}$
                diff[$i$] ← | $Resolution_S$ - $Resolution_P$ |
        end for
return index of MIN(diff)
```

3.4 Coordinate Transformations

Because a map browser uses data modeled from the real world and generates graphics on a electronic screen, the map browser operates in multiple spaces and must translate between coordinate systems. Coordinate transformations, which map the same space from one reference system to another (Bolstad 2005), are dependent on the map projection and tiling scheme (metadata about the tiled map). Defined transformations, for instance, allow one to translate between raster image coordinates and geographic coordinates (Equation 3.4, 3.5).

$$M(x_P, y_P) \rightarrow x_M, y_M \tag{3.4}$$

$$P(x_M, y_M) \rightarrow x_P, y_P \tag{3.5}$$

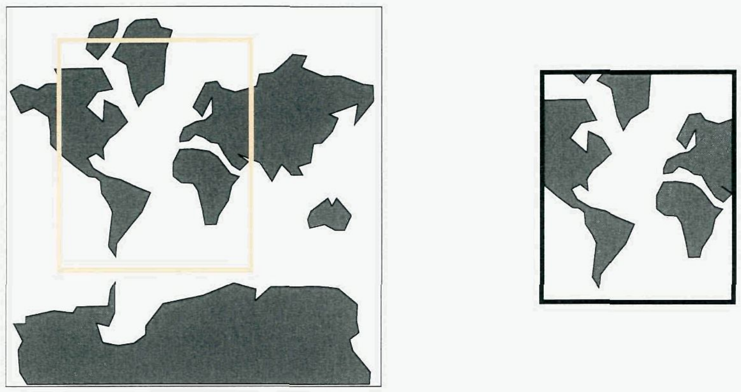

(a) **(b)**

Figure 3.3: Map space clipping: (a) Viewport in map space and (b) corresponding screen graphics.

A raster representation of the map space belongs to a single zoom level Z and has an additional conversion from pixel space to tile space (Equation 3.6, 3.7).

$$T(x_P, y_P) \rightarrow (x,y)_T, (x_P, y_P) \quad (3.6)$$

$$P((x,y)_T, (x_P, y_P)) \rightarrow x_P, y_P \quad (3.7)$$

In order to display a map image, the graphics renderer translates points in the pixel space to the screen space (Equation 3.8, 3.9) (Figure 3.2).

$$S(x_P, y_P) \rightarrow x_S, y_S \quad (3.8)$$

$$P(x_S, y_S) \rightarrow x_P, y_P \quad (3.9)$$

Many viewport properties are obtained through implicit derivation using coordinate transformations. The screen size (Δx_S, Δy_S) ultimately determines the extent of the viewport, obtained thorough accurate tracking of the viewport's center and rotation angle.

3.5 Viewport and Tile Relationship

At any particular time, the viewport covers some portion of the map space and pixel space, given by the properties of V (Table 3.1). To generate a map image the map browser renders a portion of the raster map and draws this image to the screen. Since the raster map is broken up into tiles at each zoom level, the map renderer requires at least one tile to generate a screen image. The set of tiles includes any tile at a given zoom level that intersects the viewport's MBR, a set which itself is specified as an MBR of tiles (Figure 3.4). Formally put, the tile MBR is the tile coordinate bounds of a specific zoom level which satisfies intersection constraints with the viewport (Equation 3.10), where Δx_T and Δy_T are the width and height of a tile in pixel space, respectively).

$$\begin{aligned} MBR_T &= \{x_{minT}, y_{minT}, x_{maxT}, y_{maxT}\} \\ x_{minT} &= \text{FLOOR}(x_{minV} / \Delta x_T) \\ y_{minT} &= \text{FLOOR}(y_{minV} / \Delta y_T) \\ x_{maxT} &= \text{FLOOR}(x_{maxV} / \Delta x_T) \\ y_{maxT} &= \text{FLOOR}(y_{maxV} / \Delta y_T) \end{aligned} \tag{3.10}$$

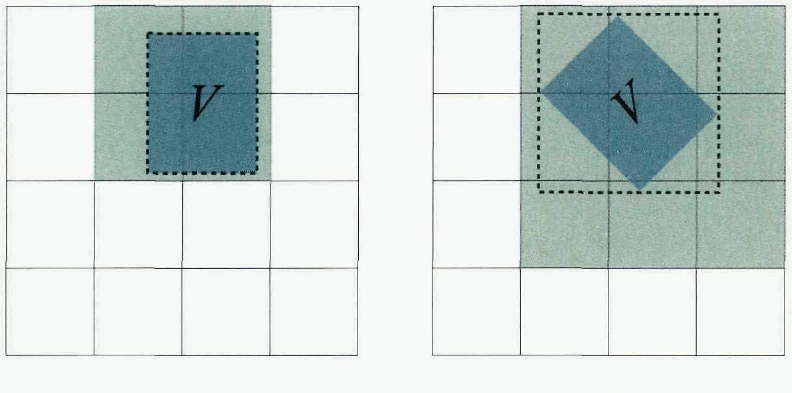


Figure 3.4: MBR_P (dotted) is dependent on the rotation of V and yields MBR_T (shaded). (a) $\theta_V = 0°$ and (b) $\theta_V = 45°$.

3.6 Operations

A user primarily interacts with the screen of a mobile map to initiate a change in the display. These interactions (e.g., panning, rotating, zooming) result in a change of the viewport's location, size, orientation, or extent and impacts the data required to generate the display. If one considers the viewport as a rectangle existent in map space, then user interactions are transformations of the rectangle in the Euclidean plane of map space. The viewport transformation maps each point within the rectangle to a new point in the same space. Panning, rotating, and zooming are user interactions corresponding to the Euclidean transformations of translation, rotation, and scaling, respectively (Worboys and Duckham 2004) (Equation 3.11-3.13).

Translation (3.11)

$$(x,y) \rightarrow (x + a, y + b)$$

Rotation (3.12)

$$(x,y) \rightarrow (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$$

Scaling (3.13)

$$(x,y) \rightarrow (ax, by)$$

A tiled map supports two categories of operations: (1) transformations of the viewport in which the viewport's size and zoom level remain constant, and (2) transformations of the viewport in which the viewport's size and potentially the zoom level will change. Each transformation has a *before* and an *after* state, symbolized by V and V', respectively. Transformations may occur instantaneously or over some duration, each with differences on how the operation appears on the screen and which tiles are required to complete the operation. The transformation appears to either "jump" to the second state, for example, or is animated on the screen for a smooth transition from V to V'.

3.6.1 Operations in the Tile Plane

A raster map display on a mobile unit is dependent on both direct and indirect interactions from the user. A *direct interaction* responds to a user's request to reposition the map on the screen (e.g., a button to pan north) whereas an *indirect interaction* responds to a sensor's reading (e.g., when a GPS receiver detects a change in position) and updates the map browser accordingly. Two operations take place at a single zoom level on a tiled map: panning/moving (Section 3.6.1.1) and rotation (Section 3.6.1.2).

3.6.1.1 Panning/Moving the Map

Geographic space is vast relative to a mobile screen and, consequently, is often difficult to capture within a single display space (Freundschuh and Egenhofer 1997). Mobile maps should support a dynamic display by allowing the user to move the viewport relative to the map space. Panning (or viewport movement due to physical user movement) is a translation of V in the plane. The map interface allows the user to initiate this change by some means of interaction, such as by dragging the map with one's finger or selecting from a set of map controls (e.g., direction buttons). Depending on the implementation, the user pans the map in a discrete set of directions (N, S, E, W) or in any direction, as in the case of dragging the display on the screen (You *et al.* 2007).

A pan operation is a translation of the viewport (Equation 3.11), specified as a change in the x- and y-directions (i.e., $\mathbf{pan}(\Delta x, \Delta y)$). A possible way to invoke a pan operations is for the user to employ a *touch-and-drag* action on the mobile screen. Since the user does this within the screen bounds, a possible constraint on the parameters of **pan** is ($0 <= \Delta x <= w_S$, $0 <= \Delta y <= h_S$). For the purpose of analyzing the affected tiles of a pan operation, assume that this panning constraint is applied.

Prior to the pan operation, the viewport has certain leeway in each cardinal direction and until a pan exceeds this free space, MBR_T remains constant. Free space in each direction is denoted x_{free}^{+}, x_{free}^{-}, y_{free}^{+}, and y_{free}^{-} (Figure 3.4).

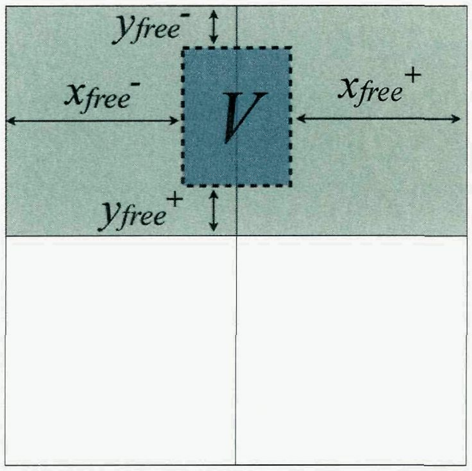


Figure 3.5: Viewport free space.

Panning the viewport may result in an incrementing or decrementing tile bounds where the new values of MBR_T are specified in terms of the free space, the panning distance (i.e., Δx, Δy), and the width and height of a tile. If the panning distance exceeds

$$\Delta x < x_{free}- \Rightarrow x_{minT}' = x_{minT} - 1 - \text{FLOOR}(|\Delta x - x_{free}-| \,/\, \Delta x_T) \quad (3.14a)$$

$$\Delta y < y_{free}- \Rightarrow y_{minT}' = y_{minT} - 1 - \text{FLOOR}(|\Delta y - y_{free}-| \,/\, \Delta y_T) \quad (3.14b)$$

$$\Delta x > xf_{ree}+ \Rightarrow x_{maxT}' = x_{maxT} + 1 + \text{FLOOR}(|\Delta x - x_{free}+| \,/\, \Delta x_T) \quad (3.14c)$$

$$\Delta y > y_{free}+ \Rightarrow x_{maxT}' = y_{maxT} + 1 + \text{FLOOR}(|\Delta y - y_{free}+| \,/\, \Delta y_T) \quad (3.14d)$$

the outer bounds of MBR_T then the new tile bounds increments or decrements once plus however many tiles the viewport crosses (Equation 3.14).

3.6.1.2 Rotating the Map

Rotation is a change in the orientation of the viewport with respect to the map space. *Adoptive orientation* is a term referred to a GIS that senses an orientation and rotates the map in line with the user's frame of reference (Frank 2003). A location sensor (e.g., GPS receiver) in combination with an orientation sensor (e.g., a digital compass) depicts the user's environment in line with his or her frame of reference (Egenhofer and Kuhn 1998). Rotation should, therefore, be considered a common interaction with mobile maps. Rotation, like panning, may also be a direct or indirect interaction, such as when a user requests a map rotation or the rotation is a result of a sensed change in device orientation.

A rotate operation is a rotation of the viewport (Equation 3.12) executed as **rotate ($\Delta\boldsymbol{\theta}$)**, $\Delta\boldsymbol{\theta}$ being the change in viewport orientation. Figure 3.3 shows that even when the size of the viewport remains consistent, the size of MBR_V fluctuates with θ_V. The maximum area covered occurs when $\theta_V = 45°$, 135°, 225°, or 315°, which follows from deriving the function for the area of MBR_V and determining the critical points. Let the dimensions of the viewport be in terms of the width (w_P), the height (h_P), and the angle of rotation (θ) (Figure 3.6). The area of the minimum bounding rectangle is the product of the MBR's width and height. Deriving the function and solving the equation at zero reveals a local maximum, minimum, or inflection point (Equation 3.15). The second derivative test shows that the maximum area of the bounding box occurs when θ is 45 degrees, the same situation also applying to $\theta_V = 135°$, 225°, and 315°.

$$\begin{aligned}
&A(\theta) = [h_P{*}cos(\theta) + w_P{*}sin(\theta)] * [h_P{*}sin(\theta) + w_P{*}cos(\theta)] \qquad (3.15)\\
&A'(\theta) = (h_P^2+w_P^2)(\, cos^2(\theta) - sin^2(\theta)\,)\\
&0 = (h_P^2+w_P^2)(\, cos^2(\theta) - sin^2(\theta)\,)\\
&cos^2(\theta) = sin^2(\theta)\\
&\theta = 45°,\ 0° <= \theta <= 90°\\
&A''(\theta) = (h_P^2+w_P^2)(-2cos(\theta)sin(\theta) - 2cos(\theta)sin(\theta)\,)\\
&A''(45°) < 0 \Rightarrow \text{local maximum at } 45°
\end{aligned}$$

To analyze the effects of a rotation on the underlying map tiles, the new location and extent of the viewport relative to the pixel space must be calculated. All new points for the viewport after the rotation come from the rotation equation (Table 3.2). First, the coordinates are mapped to a local coordinate system where the viewport center acts as the origin (i.e. $\{c_P, d_P\} \rightarrow \{0, 0\}$) and each point is specified relative to this origin. Next, coordinates are transformed to their rotated version $\{(x,y) \rightarrow (x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)\}$. After the rotation the local coordinates are converted back into global coordinates relative to pixel or map space.

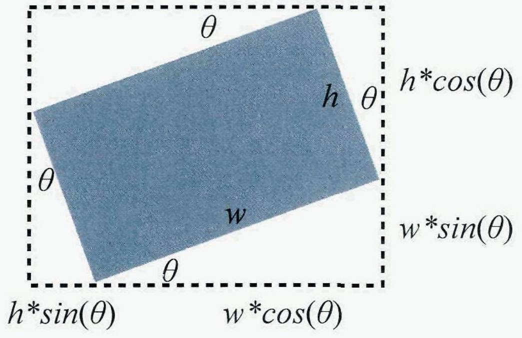


Figure 3.6: Viewport MBR dimensions.

Regardless of the viewport's orientation, MBR_V is always derivable given the properties of V (e.g., c_P, d_P, w_P, h_P, θ_V). As with panning, MBR_V determines the minimum bounding tiles for that space, MBR_T (Equation 3.10). A rotation operation changes MBR_V and, consequently, the tiles intersecting the viewport's bounds. Prior to rotation, the viewport has certain leeway in each cardinal direction and until MBR_V exceeds this free space (Figure 3.4), MBR_T remains constant.

MBR_T' are new tile bounds resulting from **rotate(Δθ)** and contain the same tiles as prior to the rotation plus any new tiles that the viewport overlaps due to ΔMBR_V (Equation 3.16). Changes in the viewport bounds from rotation are denoted Δx_{minV}, Δy_{minV}, Δx_{maxV}, and Δy_{maxV}.

$$\Delta x_{minV} < x_{free-} \Rightarrow x_{minT}' = x_{minT} - 1 - \text{FLOOR}(|\Delta\, x_{minV} - x_{free-}| \,/\, \Delta x_T) \tag{3.16a}$$

$$\Delta y_{minV} < y_{free-} \Rightarrow y_{minT}' = y_{minT} - 1 - \text{FLOOR}(|\Delta\, y_{minV} - y_{free-}| \,/\, \Delta y_T) \tag{3.16b}$$

$$\Delta x_{maxV} > x_{free+} \Rightarrow x_{maxT}' = x_{maxT} + 1 + \text{FLOOR}(|\Delta\, x_{maxV} - x_{free+}| \,/\, \Delta x_T) \tag{3.16c}$$

$$\Delta y_{maxV} > y_{free+} \Rightarrow y_{maxT}' = y_{maxT} + 1 + \text{FLOOR}(|\Delta\, x_{maxV} - y_{free+}| \,/\, \Delta y_T) \tag{3.16d}$$

Rotation involves a transformation of viewport coordinates and is dependent on the rotation angle, θ. A rotation of V implies an expansion or contraction of MBR_V and, consequently, can result in a corresponding expansion of MBR_T. Rotation, like panning, occurs within a single zoom level whereas requesting more or less detail through zooming often requires multiple zoom levels (Section 3.6.2).

3.6.2 Multi-Scale Operations

Maps often support visualizing data at multiple levels of detail because geographic spaces are too large and geographic features are too many in number to view on a single display (Harrower and Sheesley 2005). *Zooming* is an interactive feature that, in addition to panning, allows users to browse and seek information on a map display. Zooming refers to a scaling geometric transformation of the viewport (Equation 3.13) and although it is an instance of scaling, zooming is often described as two distinct operations (i.e., *zoom in* for more detail, *zoom out* for less detail) (Figure 3.7). A mobile map supports zooming to increase or decrease the level of detail in order to adjust the granularity of spatial information.

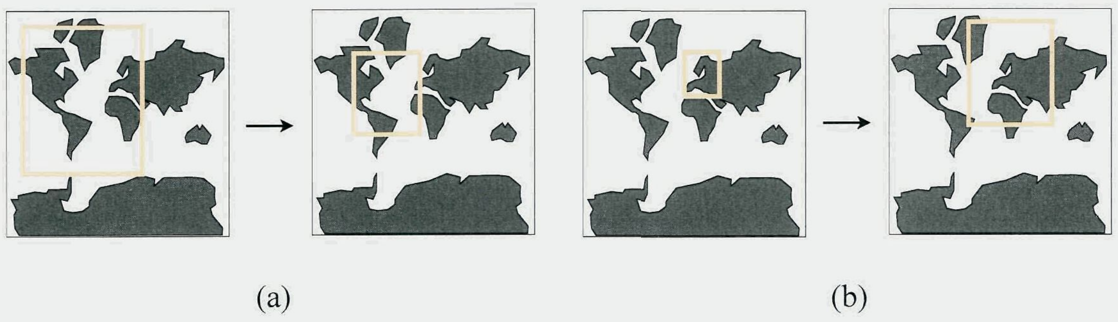


Figure 3.7: Zoom operations. *V* represented as highlighted rectangle: (a) zoom in and (b) zoom out.

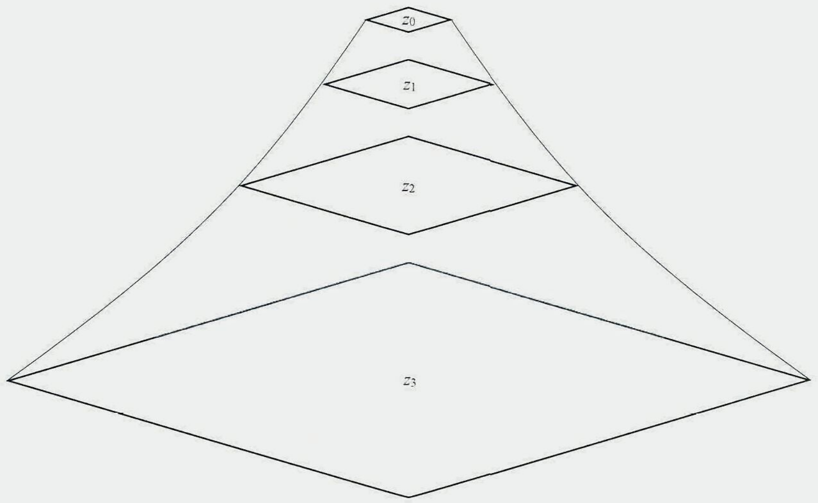


Figure 3.8: A tiled map represents map space at several discrete intervals ($z \in Z$). Continuous tile space includes representations in between these ($z \in R$).

3.6.2.1 Zooming the Map

A tiled map is composed of several discrete zoom levels, each being a distinct pixel space of a particular width and height (Δx_P, Δy_P). At any time the current zoom level, Z_V, is the tile layer from which data is retrieved and displayed on the screen although it is possible to utilize tile images from a different level by scaling them. Z_V is determined by **getZoom** (Function 3.3).

This thesis distinguishes between two categories of zooming: (1) *discrete* and (2) *continuous* zooming. A graphical user interface permits users to browse the map by changing the map scale either in discrete intervals (e.g., users *jump* to a new scale) or on a continuous basis (e.g., a pinch gesture to zoom in gradually). A tiled map typically zooms to a specific set of zoom levels (i.e., $z \in \mathrm{Z}$) versus zooming in between levels (i.e., $z \in \mathrm{R}$). In either case, the map browser references data from a single tile layer corresponding to a particular zoom level.

From the perspective of pixel space to screen space mapping ($P \rightarrow S$), discrete and continuous zooming are different. Discrete zooming permits the viewport to resize itself to only a specific set of resolutions, each of which has the same resolution as the corresponding zoom level (i.e. $Resolution_S = Resolution_P$). Each pixel from the pixel space can map to one and only one screen coordinate, making this mapping an injective function (Figure 3.9a), whereas continuous zooming allows for one-to-many or many-to-one pixel mappings (Figure 3.9b,c).

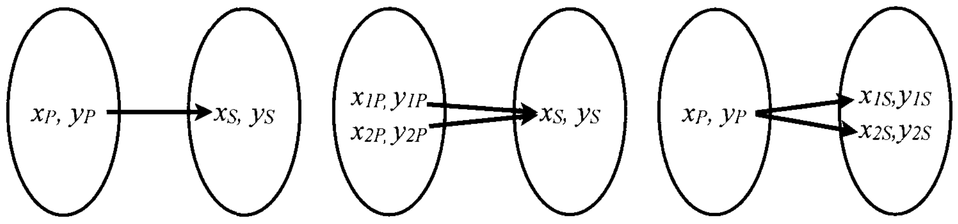


Figure 3.9: Pixel mapping: (a) one-to-one (discrete zooming), (b) one-to-many (continuous zooming), and (c) many-to-one (continuous zooming).

3.6.2.2 Discrete Zoom

This thesis defines two operations for a user under the realm of discrete zooming. The user has the option to either zoom in or zoom out by one zoom level. In either instance, T_{MBR}' may need to be recalculated after the operation, since tiles now come from a new tile layer. The new tile bounds, however, are still spatially dependent on the previous bounds. Each successive zoom level splits a particular tile into four tiles, so that tile coordinates from one level to the next are related. Given tile bounds $MBR_T\ \{x_{minT}, y_{minT}, x_{maxT}, y_{maxT}\}$ at level Z, the tile bounds at level $Z + 1$ for the same area are $MBR_T'\ \{2{*}x_{minT}, 2{*}y_{minT}, 2{*}x_{maxT} + 1, 2{*}y_{maxT} + 1\}$. Similarly, given tile bounds $MBR_T\ \{x_{minT}, y_{minT}, x_{maxT}, y_{maxT}\}$ at level Z, the tile bounds at level Z - 1 for the same area are $MBR_T'\ \{x_{minT} / 2, y_{minT} / 2, x_{maxT} / 2, y_{maxT} / 2\}$. These new bounds, however, still represent the same map space at each zoom level. A **zoomIn()** often requires fewer tiles within these new bounds, while a **zoomOut()** often requires more tiles outside these bounds since the viewport scales up or down relative to the map space.

3.6.2.3 Continuous Zoom

Continuous zooming is an interaction that allows a user to increase or decrease the extent of the viewport in much smaller intervals. Instead of choosing between a set of resolutions, a user can technically zoom to any resolution by setting the size of the viewport explicitly. This thesis defines one operation for continuous zooming, **zoom(Δd)**, where d refers to the diagonal of the viewport in pixel space. A value $\Delta d < 0$ implies a zoom out, whereas a value $\Delta d > 0$ implies a zoom in operation. Given the new diagonal value, d', the scaling coefficient for the zoom is the ratio of the two diagonals (i.e., $s = d' / d$). As with rotation, coordinates for the viewport are transformed to a local reference system and s is applied to the scaling formula (Equation 3.13). Coordinates are then transformed back into their original reference system and the map browser determines if any new tiles are required.

A zoom operation has two states, V and V', and if the two states are within the same zoom level (i.e., **getZoom(V)** = **getZoom(V')**) then the minimum bounding level requires new tiles only if $d < 0$. If a scaling operation is always performed on the center of the viewport (i.e., $c_P, d_P = c_P', d_P'$), then the transformation is isotropic, that is, the stretching or shrinking of the viewport is equal in all directions, implying that there is equal growth (Δp) in each of the minimum bounds of the viewport. Prior to zooming, the viewport has certain leeway in each cardinal direction and until one of these thresholds is crossed by MBR_V, MBR_T remains constant. Free space in each direction is denoted x_{free}^+, x_{free}^-, y_{free}^+, y_{free}^- (Figure 3.4). The new tile bounds are:

$\Delta p < x_{free}\text{-} \Rightarrow x_{minT}' = x_{minT} - 1 - \text{FLOOR}(|\Delta p - x_{free}\text{-}| \,/\, \Delta x_T)$ (3.17a)

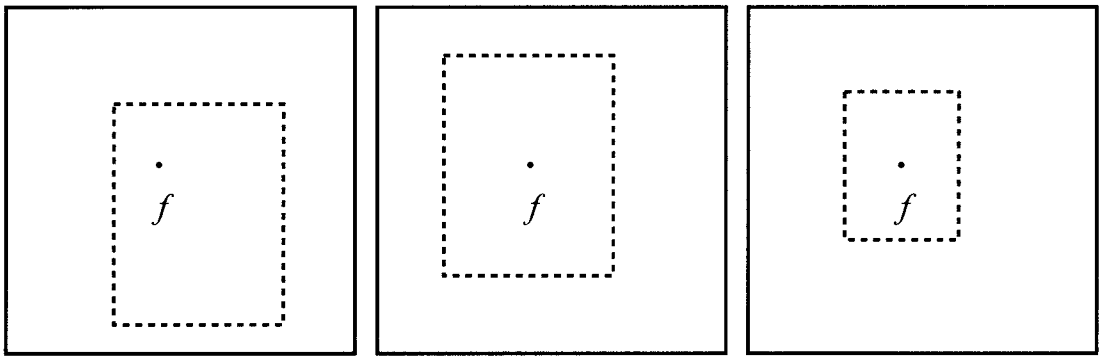
$\Delta p < y_{free}\text{-} \Rightarrow y_{minT}' = y_{minT} - 1 - \text{FLOOR}(|\Delta p - y_{free}\text{-}| \,/\, \Delta y_T)$ (3.17b)

$\Delta p > x_{free}+ \Rightarrow x_{maxT}' = x_{maxT} + 1 + \text{FLOOR}(|\Delta p - x_{free}+| \,/\, \Delta x_T)$ (3.17c)

$\Delta p > y_{free}+ \Rightarrow x_{maxT}' = x_{maxT} + 1 + \text{FLOOR}(|\Delta p - x_{free}+| \,/\, \Delta y_T)$ (3.17d)

3.6.2.4 Focusing the Zoom

A map browser may allow the user to focus the zoom operation on a point, which essentially re-centers the viewport as the zoom operation takes place. In this way the user is not strictly confined to zooming in on center. This operation is essentially a combined pan-and-zoom operation, where the viewport is initially panned to the new center point and then scaled according to the zoom parameter (Figure 3.10).



(a) (b) (c)

Figure 3.10: Pan and zoom: (a) viewport and focal point *f*, (b) viewport pans to re-center over *f*, and (c) viewport zooms in on *f*.

The operation on the interface should not appear to be two distinct operations but rather a smooth animation from one state to the next. The data required to render this operation smoothly for the map browser, however, is covered when thought of in a twostep process. MBR_T' are the new tile bounds resulting from a pan operation to f. MBR_T'' is the new tile bounds resulting from the zoom operation while V is centered over f. Any new tiles required to complete the operation are found in the new bounds. Allowing a zoom operation to focus on a point adds the extra parameter, f, to the zoom operations, which now are: **zoomIn(*f*)**, **zoomOut(*f*)**, and **zoom(Δ*d*, *f*)**.

3.7 Summary

This chapter identifies several spaces inherent in a tiled map implementation for mobile devices. The map space is a rectangular projected representation of the real world globe. The map space is split into several images of different resolutions as a computer representation. Each level of detail, or zoom level, is a virtual image of the map space which is cut into square images called tiles for easier storage, transport, and management. This representation forms a tiled map, also known as an image pyramid, which is a popular method for publishing base maps to clients since the client only needs a subset of tiles to render a map display. The viewport is the region of the tile space drawn to the screen. Such an operation requires mapping pixel locations from the tiles to those of the screen. A dynamic display of the map is accomplished via user- or sensor-invoked interactions such as panning, rotation, and zooming, each resulting in a geometric transformation of the viewport. To render a display the mobile map browser requests a minimum bounds of tiles that changes due to these interactions. User operations typically allow incremental changes in the tile bounds (e.g., panning the display north to the next tile). Any new data that the map browser needs, therefore, has a high spatial dependence on the data used prior to the operation. Conceptually, this idea is consistent with Tobler's *first law of geography*: “everything is related to everything else, but near things are more related than distant things” (Tobler 1970). Interactions with a mobile map browser have the highest impact on *nearby* data.

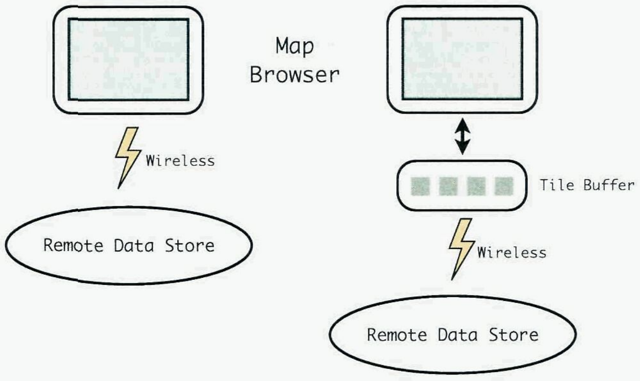
Operations occur individually or in combination. Concurrent operations have a slightly different effect on the tile dataset, since the combined effect is in essence a union of the two distinct operations. Mobile map operations play a role in cache management of the data (Chapter 5). Whether considering individual or simultaneous operations, a history of such operations effect how tiles in the cache are populated and removed.

CHAPTER 4

POPULATING THE TILE BUFFER

User interaction with a mobile map often results in the need to acquire new data accessible from a remote data store. Downloading the data from a remote repository is sometimes a necessary action yet has several drawbacks, such as unreliable connections and relatively low bandwidth, yielding a slower response time. To combat the issues involved with wireless communication a popular solution in mobile computing is caching data locally on the device for repeated consumption. The map browser uses tiled images as the smallest unit of data and in order to ensure proper data availability, tiles are cached to a tile buffer, that is, a data structure for temporarily storing tiles on the mobile device.

A key advantage of the tile buffer is the addition of an intermediate communication layer between the mobile map browser and the remote data store where tiles are accessed (Figure 4.1). While the lack of a buffer enforces the map browser to establish expensive connections (i.e., time intensive and data charges) to the network for each user action, the presence of one permits the browser to first request tiles from the local store and only make a remote request if a tile is not available in the buffer, reducing the amount of client-server communication. The implementation of a tile buffer, coupled with a set of intelligent heuristics on how to populate and manage tiles in the buffer, yields a map browser with substantially higher usability. Improved usability is accomplished via an attempt to decrease the amount of time a user waits for the map to refresh its display in order to avoid testing the threshold of user patience (Mummert *et al.*, 1995).



(a) (b)

Figure 4.1: Tile buffer: (a) map-server communication and (b) map-server communication via the buffer.

4.1 Cost

Although one wishes to minimize the latency involved with refreshing the map display, time is only one of many costs involved with retrieving data over the air via a mobile device. In fact, one can measure data retrieval cost using several metrics. Mobile devices power themselves from a battery consisting of an extremely limited energy source and communication over cellular networks can require a large amount of battery resource (battery cost) relative to local computations. For this reason it is desirable to refrain from too much over-the-air downloading of data to conserve battery power, especially with spatial data, which are large relative to simpler data formats. Depending on the service provider, data communication over wireless mediums can be expensive with regard to money (dollar cost). While some providers offer prepaid, unlimited data plans for communication over their cellular towers, others charge a user for data on a per-megabyte scale. Therefore, one should design a mobile map browser with this in mind and possibly allow the user to configure the allowable amount of downloaded data. Cost might also be measured in data volume (storage cost) since data retrieved from a remote resource is stored to local memory (e.g., RAM or flash). Once the designated storage space becomes full an offloading mechanism must remove data in preparation for new downloads. All costs mentioned are important. Latency of display refreshing due to temporal costs, however, is very crucial for usability. When one must constantly wait for data retrieval on an interactive display such as a mobile map browser, frustration levels rise and usability is compromised. This chapter discusses some ways of populating a device with maps so as to anticipate user interaction to reduce the latency of a map refresh.

4.2 Tile Buffer

This chapter defines the tile buffer and begins the discussion on how it may utilize tiled data more effectively. The tile buffer is implemented as a set of tiles, meaning that the buffer contains unique tiles and, therefore, changes only when adding a tile not currently present in the buffer. This implementation differs from other collection data structures that allow multiple instances of an item (e.g., a *multiset* or *bag* data structure).

Relevant operations on the tile buffer include those which enable caching of tiles to the data structure. Since a tile buffer is a collection object (i.e., a set), there must be an initialization or reset function to establish an empty buffer. The buffer herein is referred to as **B** (a set of tiles) and initialization of the buffer is an instantiation of an empty tile set defined as the function **B.init()** (i.e., $\mathbf{B} \leftarrow \{\}$). The remainder of this chapter is concerned with populating the buffer with tiles and so makes use of a function to add tiles to the buffer. The function **B.add(T)** adds the tile, **T**, to **B** (i.e., $\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T}$).

4.3 Populating Heuristics

A naive approach to populating a tile buffer includes only those tiles explicitly requested by the map browser, meaning each tile in the buffer is or has been visible to the user through the viewport. While this approach is the simplest, it creates a serious impediment to performance of the map browser. Consumption of tiles in the buffer only happens when the viewport intersects a previously used tile, meaning that the fastest map refresh occurs only when the user goes *back* to a earlier location. Each operation resulting in a new area on the map requires another network connection and additional tile requests from the remote server. If the map browser is following a route for navigating the user along a road network, each movement to a new location requires additional data and, thus, delays the map rendering process.

To solve the issue of forcing the user to wait for a map refresh, the buffer can employ better heuristics by anticipating future operations invoked by the user based on the intuition of where a user expects the least amount of delay. Doing so exploits a network connection by retrieving more data than currently required. Establishing a new connection to a server for tile requests is assumed to cost something, mostly with regard to the time required to do so. In addition, wireless networks are characterized by intermittent connectivity and it is impossible to fully determine whether or not a connection will be available in the near future. The unreliability factor of wireless network connections creates an uncertainty to the availability of map data in the future. The strategies for populating the tile buffer described in this chapter anticipate upcoming operations on the map in order to take advantage of any available connection when available by retrieving a larger data set from the server.

Each assumption has implications on the **add()** function defined on the tile buffer. The map's viewable area is described using MBR_P or MBR_T, which describes a set of adjacent tiles, each of which are added to the buffer by the **B.add(T)** method. The governing strategies define a function ***f*(T)** which, given **T**, will derive a set of tiles which are also added to the buffer as a result of **add** (Equation 4.1).

$$\mathbf{B} \leftarrow \mathbf{B} + \mathbf{T} \cup \boldsymbol{f}(\mathbf{T}) \tag{4.1}$$

4.3.1 Coarser Tiles

According to Shneiderman's Visual Information-Seeking Mantra (1996), overview information is of highest priority for a visual interface. Overview information in regards to a tiled map refer to tiles at a lower level of detail than a given tile, which herein are referred to as *coarser* tiles. The first assumption of the buffer populating heuristics is that users expect to have a less detailed map than what is currently available and, thus, the buffer should always keep some set of coarser tiles in memory. The strategy attempts to minimize the amount of wait time of a map refresh for zoom out operations by maintaining coarser tiles in the buffer. When a tile is added to the buffer, a coarsening function, ***c*(T)**, derives a relatively small set of coarser tiles which are also added to the buffer (Equation 4.2).

Assumption 1 (A1) : *Always populate the buffer with lower detail tiles.*

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup c(\mathbf{T}) \tag{4.2}$$

The coarsening finds tiles of a coarser resolution than that of T. Each zoom level coarser than that of **T** has exactly one tile that contains the map extent of T (Figure 4.2). By retrieving coarser detailed tiles, the buffer increases the geographic extent of the cached tile data. Maintaining coarser tiles is especially useful if high detail maps are not available for a specific location because raster images are scalable and may be used for a different zoom level than originally intended. If tile **T** is not available, for example, the map browser may instead find a tile from a lower zoom level, scaling the image to match the resolution of **T**. Scaling an image up or down, however, does not increase or decrease the amount of information and instead decreases image quality and so should only be used as a temporary solution while the correct tiles are cached into the buffer.

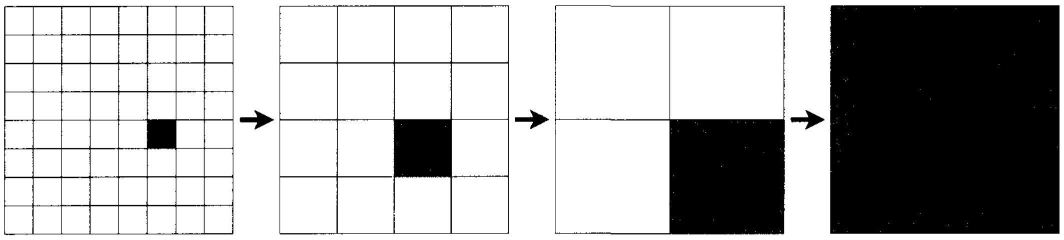


Figure 4.2: Tile containment. Tile T at level z and tiles containing T at lower zoom levels.

The function $c(\mathbf{T})$ might use one of several strategies for retrieving tiles, each of which require a different number of additional tiles to add to the buffer. For instance, the function could retrieve all tiles at coarser zoom levels that contain **T**, such as in Figure 4.2, requiring $(z - z_{min})$ additional tiles. Alternatively, $c(\mathbf{T})$ could also just retrieve one tile at a coarser level that contains the extent of all tiles visible by the map. Regardless of which method is used, the implementation should attempt to minimize the number of tiles to retrieve while still fulfilling the need for less detailed, overview data.

4.3.2 Stationary Map Rotation

In addition to retrieving coarser tiles, strategies for filling the buffer also incorporate data that are *near* the map's current location. Map rotation should be considered an instantaneous operation since it is fair to assume that a user might expect little to no delay when simply rotating the map while keeping its center stationary. Also, rotation only requires the addition of a relatively small amount of tiles so the buffer should, therefore, support rotation by retrieving the neighboring tiles of **T**. The second assumption for anticipating future operations and the corresponding tile set stems from this intuition that users expect minimal delay as the map rotates, meaning the data required for rotation should exist in the buffer. When the buffer adds a tile the function ***r*(T)** derives the tiles around **T** that are included for rotation (Equation 4.3).

Assumption 2 (A2) : *Always populate buffer to allow for stationary map rotation.*

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup \boldsymbol{r}(\mathbf{T}) \qquad (4.3)$$

The rotational tile set obtained by ***r*(T)** is based on one of several neighborhood techniques. For example, one could use the MBR of the circular footprint formed from rotating *V* about its center. The MBR is a square with a length equal to the diagonal of the viewport (Figure 4.3) and is used to derive any addition tiles that could intersect *V* when rotated. The rotational MBR creates some overhead, however, since the area in the

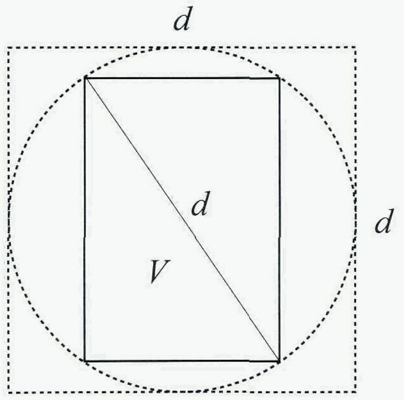


Figure 4.3: Rotational MBR of V (MBR_{rV}).

corners of the MBR are not actually part of the rotational area but does guarantee that all tiles are available for any stationary map rotation.

Alternatively, ***r*(T)** might be based on the situation that occurs when considering only one tile and its immediate neighbors. If *V* resides completely within one tile, the only tiles that can possibly intersect *V*'s rotational footprint are the four adjacent neighbors to that tile. The function ***r*(T)** retrieves the four adjacent neighbors of T and, if multiple tiles are covered by the viewport, the resulting tile set is derived by a matrix addition (Figure 4.4).

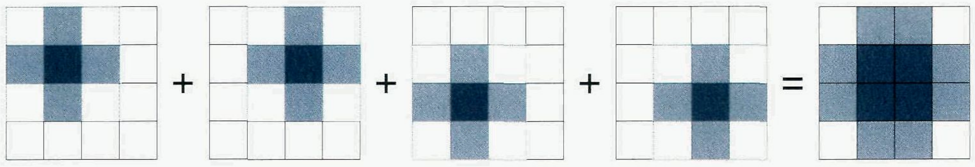


Figure 4.4: Tile neighborhood addition. T (dark) and rotational tiles (light).

Tiles that are pushed into the buffer in preparation of map rotation should also comply with the coarser tile policy **A1**; for each tile added for rotation the buffer also finds the coarser set of tiles. In other words, an extension of **A2** states:

Assumption 2a (A2a) : *Always populate the buffer with coarser tiles derived from* **A2.**

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup c(r(\mathbf{T})) \tag{4.4}$$

The intersection of the tile sets derived from ***c*(T)** and ***c*(*r*(T))** will often be non-empty since coarser tiles cover a larger extent. Therefore, the addition of **A2a** is considered relatively non-expensive but necessary to maintain a set of tiles which provide an overview of the current map.

4.3.3 Details on Demand

The Visual Information-Seeking Mantra suggests that higher detailed information be retrieved only on-demand when requested by the user. Intuitively, users are more willing to accept the fact that retrieving a higher detailed data set requires a wait. From a performance perspective, on-demand access for high detail data is necessary since the number of tiles composing a region of the map space grows exponentially as one increases the zoom level. On-demand access to tiles at a higher zoom level is independent of the anticipation heuristics but still triggers previously defined heuristics (Equation 4.5).

Assumption 3 (A3) : *Add detailed tiles to buffer on-demand only.*

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup c(\mathbf{T}) \cup r(\mathbf{T}) \cup c(r(\mathbf{T})) \tag{4.5}$$

The cost of retrieving additional tiles for **A3** is minimal since the buffer already contains coarser tiles from immediately before the *zoom in* operation.

4.3.4 Nearby Movement

A commonly used feature of mobile mapping software is navigating the user through space, such as along a road network. Navigation implies that the user is moving through space and, hence, will require a dynamic map display. In addition to a navigation scenario that updates the map display through sensed changes in position, browsing the map (e.g., panning) also yields a dynamic map display. These two examples both seek new data that are close in pixel space to the map's current location. In order to anticipate the local movement of the device/user or map display, the next buffer populating strategy assumes that the user will require neighboring tiles of the current map view. A function ***n*****(d,T)** derives nearby tiles which are less than or equal to ***d*** tile units from **T** (Equation 4.6).

Assumption 4 (A4) : *Anticipate nearby movement and pre-fetch tiles for a limited range of the current map view in discrete tile units.*

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup \boldsymbol{n}(\boldsymbol{d},\mathbf{T}) \tag{4.6}$$

Tile distance refers to the number of units between two tiles and is defined using a distance metric. Since tiles are indexed by their *x*- and *y*-coordinates, the indexes determine the distance between two tiles. A common distance metric in two-dimensional space is based on the relation between two items in Euclidean space and uses the Pythagorean theorem (i.e., $\mathrm{d} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, $\mathrm{d} \in \mathbb{R}$) as a quantitative measure of distance. A simpler but equally effective metric for tiles is Manhattan distance (or Taxicab Geometry), where the distance is the sum of the absolute difference between tile coordinates (i.e., $d = |x_1 - x_2| + |y_1 - y_2|$, $d \in \mathbb{Z}$). Using this approach, distances are described as integers and tiles of equal distance form rings around the center tile.

Both **A2** and **A4** are concerned with tiles residing in the immediate neighborhood of **T**. In fact, the two assumptions both consider an isotropic tile range from **T** and will result in some overlap. If **A2** only considers the four adjacent neighbors of **T**, while **A4** considers all tile neighbors up to some distance, d, then the buffer heuristics can discard **A2** so long as d > 0. The strategy populates the buffer in groups of neighboring tiles of equal distance from **T**. In addition to adding the neighboring tiles, the buffer also looks to previous heuristics for each added tile. A revision of **A2a** yields an extension of **A4** (Equation 4.7).

Assumption 4a (A4a) : *Always populate the buffer with nearby tiles up to some distance, d.*

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup c(n(d, \mathbf{T})) \tag{4.7}$$

4.3.5 Direction-Dependent Movement

So far the buffer heuristics discuss pre-fetching nearby map tiles in an isotropic fashion, that is, retrieving an equal number of neighboring tiles in all directions. In addition to anticipating the next move by populating the tile buffer with coarser and nearby tiles, one could also exploit direction of travel for bulk loading tiles into the buffer. Direction of travel refers to the predominant direction of movement in the user's environment and to derive this property, the buffer must record a location history of the user that together form a trajectory which the buffer extrapolates to anticipate a future location. A tiled map trajectory is the movement of the user or viewport through space and is represented as a sequence of location and time entries, each having the form $\{ x_i, y_i, t_i \}$.

Processing on location history data determines whether or not the user is moving and, if so, the direction and speed of the movement. Using the device's location history, the buffer manager uses one of two strategies for bulk loading tiles. If the device has the same position (or close to the same position) over the past **n** units of time, then one assumes the device is stationary, in which case any bulk-loading of tiles acts according to **A4**. A non-stationary mobile device, however, has a trajectory described by its past positions. Processing this trajectory derives a motion vector, $\boldsymbol{v}$, describing both the direction and velocity of the movement. If the buffer manager determines that the device is in motion, a direction-dependent bulk-downloading strategy overrides the stationary version (**A5**).

Assumption 5 (A5) : *When device is moving, populate the buffer with nearby tiles up to some distance, **d**, in the direction given by* $\boldsymbol{\alpha}$.

$$\mathbf{B} \leftarrow \mathbf{B} \cup \mathbf{T} \cup \boldsymbol{n}(\boldsymbol{\alpha}, \boldsymbol{d}, \mathbf{T}) \tag{4.7}$$

Direction-dependent bulk-loading uses the predominant direction of movement of the device to put emphasis on downloading tiles in a specific direction, $\boldsymbol{\alpha}$, where $0 \leq \boldsymbol{\alpha} \leq 2\pi$. The magnitude of the motion vector, $\boldsymbol{d}$, is dependent on the velocity of the motion, where an increase in velocity corresponds to an increase in $\boldsymbol{d}$. Anticipating tile usage based on direction demonstrates an anisotropic approach to pre-fetching nearby tiles. Neighborhood tiles are compared to each other through distances from some center and these distances are weighted based on how astray a tile is from the motion vector. Those tiles opposite the direction will be weighted lower (i.e., larger distance) than those in the direction of movement (i.e., smaller distance) (Figure 4.5).

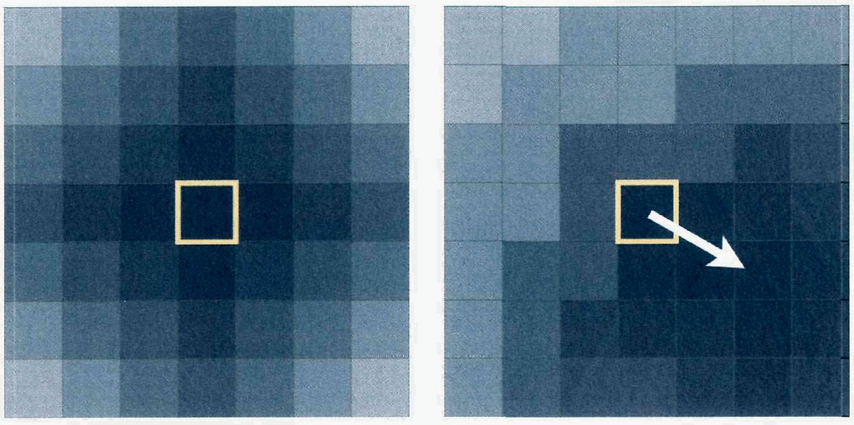

(a) (b)

Figure 4.5: Weighting of tiles. Darker tiles are weighted higher. (a) Isotropic (stationary device) and (b) anisotropic (moving device).

4.4 Server-Side Computation

Mobile devices draw from a finite energy source and will continue to limit how long one can use a device in the field without recharging the battery. In order to maximize battery life on a device one should avoid any intensive processing when possible and instead offload this processing to a server. Mobiles, assuming now the presence of wireless Internet connectivity, can communicate to servers via web services to assist in the future state of a device.

The anticipatory user interaction strategies aim to select some number of tiles to pre-load onto the device given several parameters such as previous interaction or device movement and current map level of detail. For simple location prediction using linear motion models, computation is relatively light, whereas for more complex trajectory extrapolation or might require the assistance from a server. In addition, distributed computing can offer further filtering for prioritizing anticipated locations through the use of additional GIS data. A web service, for example, might determine whether it even is necessary to preload data based on a separate prediction on the future status of cellular signal strength (Figure 4.6a). Also, one could further refine a trajectory prediction based on road networks, where a straight line path is sometimes highly unlikely (Figure 4.6b).

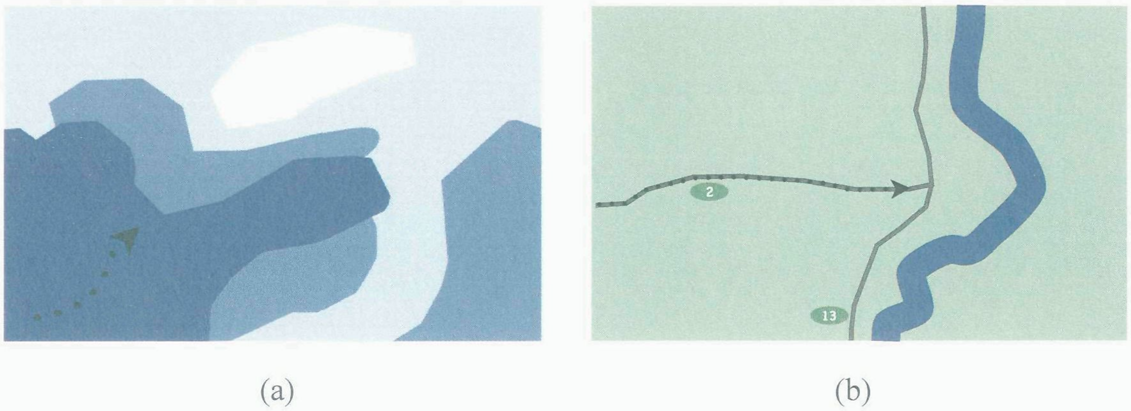


Figure 4.6: Server GIS data repositories: (a) Cellular coverage data plus trajectory and (b) road network plus trajectory.

4.5 Buffer Data Structure

The tile buffer, being a set of unique tile items, is implemented as a data structure with a logical organization of the tiles based on the buffer populating heuristics. One way to envision the buffer data structure is through the use of a linked list of tile objects, where tiles in the buffer are grouped according to how they are inserted into the buffer. For instance, all tiles that are *currently* used by the map browser might be adjacent in the list, followed by **A1** tiles for the map overview, followed by **A2** nearby tiles, and so on

(Figure 4.7). Categorizing tiles in this manner will help to prioritize how tiles are removed from the buffer (Chapter 5).

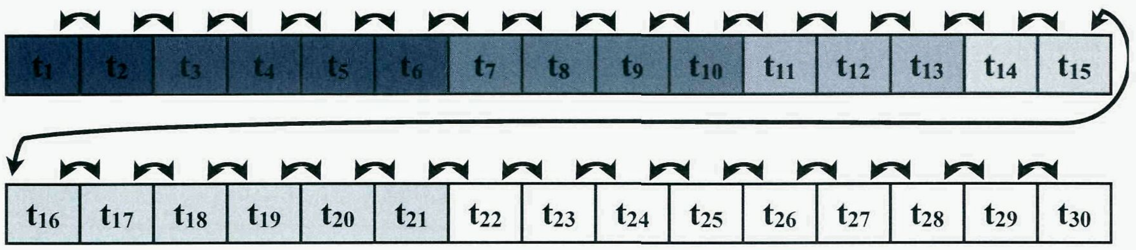


Figure 4.7: Buffer as a list of tiles, categorized by *next* move heuristics.

4.6 Summary

This chapter details a specification for implementing an anticipatory approach to user interaction on mobile map browsers. The specification describes several ways to populate a tile buffer (local storage allocation for map tiles) based on graphical user interface principles, intuition on user patience, and interaction history of the user. When the map browser has satisfied on-demand data requirements and acknowledges the presence of a data connection, the opportunity to gather additional map data from a remote tile repository should be considered. Doing so will speed up future requests for data by improving cache hit rates and also better prepare the device for a lost or slower data connection. Chapter 6 describes a prototype for testing some of the heuristics for pre-downloading map tiles to assess the advantages and disadvantages over on-demand map tile access.

CHAPTER 5

CACHE MANAGEMENT

All software applications, especially mobile ones, are memory-limited and must, therefore, employ some functionality for ensuring sufficient digital storage space for data. Map browsers use memory-demanding raster and vector data for generating the graphics for geo-visualization and regardless of whether maps are stored on a mobile device in main-memory or on disk (i.e., flash memory), at some point it becomes necessary to offload maps to ensure space for incoming data. Although it is possible to require the user to choose data to offload, this task is assumed to be non-trivial since it may involve selecting from a group of map layers, levels of detail, and geographic area and should, therefore, be abstracted from the user and implemented as an automated process. This chapter offers several tile offloading mechanisms and the rationale behind each method. Offloading, or cache replacement, revolves around assessing temporal and spatial relationships between cached items and may utilize one or both of these relations.

5.1 Cache Invalidation

Quality caching of map tiles involves proper anticipation of user interaction for pre-loading data but also proper removal of the data. Since map browsers consume spatial data over time, location is an additional parameter one may use for selecting candidate tiles for deletion. Even so, temporal attributes of the data are important and should still be factored into cache management. A challenge to effective tile data management is cache invalidation, or determining when a cached item is expired. Information changes over time but also at various temporal granularities. While weather and traffic information are characterized by frequent change, housing developments arise much less frequently, street names hardly change, and spatial attributes of geographic features like rivers and mountains vary over a significantly larger time scale. In short, map layers are diverse in the temporal granularity at which they change but nonetheless provides a metric by which to make decisions on when to offload maps on mobile devices. For purposes of managing a tile cache temporally, one might also tag each tile with a "time to live" property and when this duration is exceeded the tile is discarded from the cache.

Although invalidating a tile based on a temporal threshold permits one to define a single comparison for a cached item, a couple issues exist with the approach. For one, the comparison does not take into account the tile's frequency of use. If a tile is relatively old but frequently used and retrieving an updated version is costly (e.g., no data connection exists), discarding the tile is not the best solution. Managing cached tiles should consider the cost of retrieving an updated version of a particular tile prior to committing to deletion.

5.2 Offloading Algorithms

Cache replacement policies compare several cached items at a time in order to prioritize them for deletion. By doing so, a replacement algorithm aims to select data items to delete that are least likely required for future use. With location data it is possible to guide replacement policies using multiple metrics of space in addition to time.

5.2.1 Least Recently Used

One of the many common temporal cache replacement algorithms is the *least recently used* (LRU) policy which deletes cache entries by comparing each item's last usage timestamp and ranks them from least recent to most recent. In spite of the effectiveness of an LRU approach, one can easily derive a scenario in which an LRU approach fails for tiled maps (Figure 5.1). Consider a moving object which is traveling along a path which will eventually return to its initial location. Given the history of data consumed by the map, an LRU approach selects discarded tiles (dark shaded) that would intuitively be saved for future use based on the current location and movement of the object.

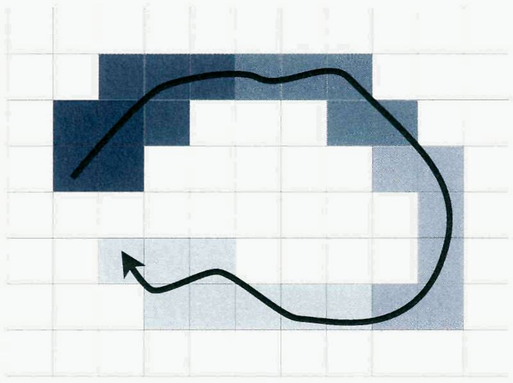


Figure 5.1: LRU ranking (darker shades indicate least recently used).

5.2.2 Furthest Away Replacement

Cache replacement algorithms for spatial data should integrate location-based cache replacement when reasonable to do so. Given a tile buffer storing *n* tiles, a furthest away replacement (FAR) will rank the set of tiles by distance from some point, ***p***. Those tiles that are furthest away from ***p*** are removed from the buffer (Figure 5.2). Alternatively, the replacement algorithm calculates tile distances between the current tile and all other tiles in the buffer. The distance between two tiles, ***d***, comes from an equation that provides some distance metric for tiles, assuming the two tiles are at equal zoom levels. Given that this is true (i.e., $Z_{T1}=Z_{T2}$), the following applies:

$$\boldsymbol{d}(T_1, T_2) = |x_{T1} - x_{T2}| + |y_{T1} - y_{T2}| \tag{5.1}$$

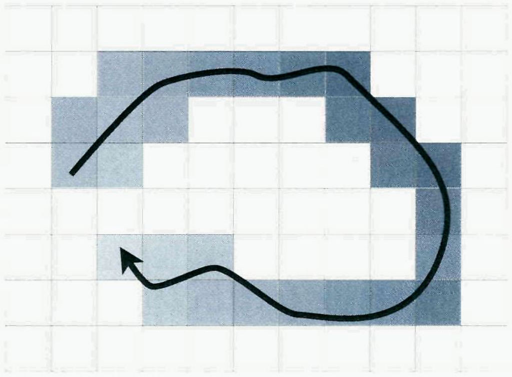


Figure 5.2: FAR discards furthest away tiles
(darker shades indicate greater distance).

5.2.3 Directional FAR

Similar to pre-fetching local tiles based on user or device movement (Chapter 4), cache replacement policies exploit trajectory data to further refine location-based cache management. Chapter 2 discusses previous approaches to represent a moving object via trajectories. In this thesis and for testing a prototype implementation a vector-based, linear motion function is used to describe a mobile device's current state of movement. A device's movement *state* is specified as a location, speed, and direction (Sistla *et al.*, 1997) and is derived from the previous ***n*** device locations.

A caching policy using FAR calculates a distance for each tile in the buffer corresponding to the distance between that tile and the center tile (i.e., the tile containing the device or map's current location and zoom level). Buffered tiles are then ranked by distance, the farthest being removed. In order to make use of direction, each tile's distance is weighted (***w*d***) based on the angle of difference between the two tiles and the general direction of movement (Figure 5.3). Those tiles in the direction of the device's motion vector receive the lowest weight, while those opposite the motion vector receive the largest. Weighting the distances according to direction encourage the buffer manager to keep tiles in the direction of movement and discard those opposite this direction. Of course, a stationary device would have little effect on the distance values as the weights would be the same regardless of direction.

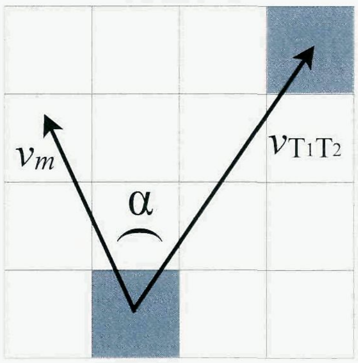


Figure 5.3: Angular difference (α) between motion vector (v_m) and the directional relationship between a tile and the center tile (v_{T1T2}).

5.2.4 Prioritization by Pre-fetching Strategies

Chapter 4 discusses the tile buffer and suggests that the buffer will not only store neighborhood tiles for a particular level of detail but also *overview* data, or tiles at a coarser zoom level. At any time tiles might be cached for several zoom levels, something the buffer must consider when executing a cache replacement algorithm.

A question that arises for multi-resolution tiled maps is how one establishes a distance metric for tiles at different zoom levels, since one cannot simply use the discrete tile distance for two tiles at equivalent zooms (Equation 5.1). For example, consider tiles T_1 and T_2, where T_1 is at a coarser zoom level than T_2 and the distance between the tiles' coordinates is large relative to if the tiles are at the same zoom level. T_1 covers T_2 topologically, however, and offers a less detailed, overview version of T_2 and should thus be preserved in the buffer. A distance metric for tiles at two different zoom levels is required for comparing tile distances at varying zoom levels. One solution is to translate T_2's tile coordinates to the zoom level of T_1 by recursively assigning T_2's parent's tile coordinates to T_2 until both T_1 and T_2 are at the same level. Once translated T_1 and T_2' use the same distance measure as for tiles of equivalent zoom levels. To account for the difference in zoom levels one could also add the zoom level difference to the distance metric (Equation 5.3).

$$\textit{given}\ (Z_{T1} < Z_{T2} \wedge Z_{T1} = Z_{T2'}),\ \boldsymbol{d}(T_1, T_2) = (Z_{T1} - Z_{T2}) + \boldsymbol{d}(T_1, T_2') \qquad \textbf{(5.2)}$$

5.3 Summary

Offloading map tiles is a balancing functionality to populating and pre-populating the memory buffer. As the buffer fills and exceeds some threshold, the buffer manager prioritizes tiles from the buffer for deletion and does so on either a temporal, spatial, or combined basis. Concepts from the previous two chapters are implemented in a prototype tile caching application in order to assess anticipatory caching strategies to on-demand strategies.

CHAPTER 6

PROTOTYPE IMPLEMENTATION

The previous two chapters introduced a number of strategies for both pre-fetching tiles and removing tiles from a memory buffer on a mobile device. With the objective of gaining some insight on the success of these strategies, this chapter introduces a prototype implementation which simulates vehicle movement through a road network. Two strategies, an *isotropic* and *anisotropic* pre-fetching policy, are implemented in a .NET C# application for comparison on a set of trajectory data gathered from real-world vehicles. Isotropic pre-fetching anticipates movement equally in all directions and generally would be useful for a completely random movement pattern through space. Realistically this is seldom the case as movement through space within relatively short temporal intervals is dependent on the past and current locations of a moving object and thus is not random, especially when constrained by a road network, for example. Anisotropic pre-fetching anticipates user movement in a particular direction to retrieve tiles at the current zoom level of the map. As a user moves through space and time, one can derive a motion vector which is then used to prioritize nearby tiles for pre-fetching, given that a data connection is available at the time.

To further augment pre-fetching, one can also pre-fetch tiles at a coarser level of detail than the current map display, which has some advantages. First, since tiles are scalable images, a single tile at the next coarser level can replace the need for retrieving four tiles at the current level. Hence, one actually increases the geographic coverage while decreasing the data volume. However, the resulting tradeoff of doing so is a decrease in the overall visual quality of the map as coarser level tiles can become "pixelated" when used for a resolution higher than originally intended. This chapter describes a prototype to simulate tile pre-fetching at a single zoom level in order to simplify the metrics needed to analyze pre-fetching performance.

6.1 Experimental Design

The prototype aims to simulate as realistic of a scenario as possible, one in which a mobile map cache is dynamically updated as a user moves through the environment or navigates the display via pan and zoom map operations. In order to do so the prototype implementation models (1) user movement through space, (2) a web-based tiled dataset, and (3) tile pre-fetching and removal policies. The success of the pre-fetching algorithm is measured by pre-fetching accuracy, derived from recording both the correct and incorrect classification of tiles for pre-fetching.

6.1.1 Trajectory Data Set

In order to support a realistic approach to measuring the pre-fetching strategies, the caching algorithm makes use of data representing a mobile object whose position changes with time. The R-tree Portal (www.rtreeportal.org) hosts a plethora of software, data generators, and real-world datasets supporting ongoing research for moving-object databases. Generate_Spatio_Temporal_Data (GSTD), for example, is a publicly available software library for generating point or rectangular moving object data and is available for download over the web. Although a trajectory generator enables one to simulate a large number of trajectories very quickly, which is useful for statistical purposes, it remains difficult to build a dataset that closely mimics the complexity of movement of a real-world object. Some generators, such as GSTD, create spatio-temporal datasets according to a statistical distribution. For example, one can specify both the spatial and temporal components to follow a normal distribution. That is, Δx, Δy, and Δt each have a mean and variance that dictates what their values tend to be. A trajectory following a normal distribution will thus progress in one general direction, where increasing the variance can result in a more irregular trajectory. In addition, the change in *x*- and *y*-values for a single timestamp is not dependent on previous sample points for the generator, which is not like the real world. A road that curves continues to curve and a long, straight road does not vary direction for some distance. Therefore, in order to capture the properties of trajectories most inherent in the real world, the map pre-fetching simulation consumes data captured from real road networks.

(a) (b)

Figure 6.1. Trajectory data: (a) school buses and (b) construction trucks.

Frentzos *et al.* (2005) offer two datasets online for metropolitan Athens, one featuring tracking of school buses picking up and dropping off students while the other represents construction trucks moving to various sites within the region (Figure 6.1). Overall, the trajectories offer a variety of trajectory complexity, from relatively straight highway driving to city block driving with frequent directional change, providing the necessary balance for an analysis of trajectory direction dynamics and pre-fetching performance.

The raw data made available on R-tree Portal does bring up a couple issues with regard to the pre-fetching simulation. First, there is a lack of consistency between timestamps of sample points and, consequently, distances between them. Since the implementation prototype uses sampled trajectory points, it must estimate pre-fetching effectiveness not only at the sampled points but also in between them. Assuming straight-line movement from t_0 to t_1, all tiles intersecting the swath of the viewport at and in between these two sample points are tagged as "required" by the map renderer. The pre-fetching algorithms only consider a maximum number of tiles to pre-fetch, both at a maximum distance and of a maximum quantity at any one time. To ensure consistency between the domain of tiles for pre-fetching along a trajectory, the data was resampled so that sample points were evenly spaced with regard to both space and time. As a result, the normalized trajectories represented vehicles moving at a constant speed of about 35 mph. Although this does not represent real world traffic perfectly, it does offer an advantage. The prototype compares two pre-fetching strategies that differ in how they deal with direction. By normalizing properties such as speed along a trajectory to ensure consistency, the direction property of trajectories is the only variant, making the upcoming analysis more statistically sound. The trajectories were also trimmed so that each trajectory contains 40 sample points.

6.1.2 Web Tile Dataset

Tile images from a tiled map dataset can vary depending on the provider, map projection, data compression, image size, among other variables. The pre-fetching prototype simulates the use of a particular data set of a defined set of attributes. OpenStreetMap is a community GIS project for creating and editing a worldwide map and the project's main website (http://openstreetmap.org) features a tiled map browser that pulls pre-rendered raster images from a database.

Our prototype simulates the use of OpenStreetMap tiles for a dynamic tile cache and does so by estimating some attributes of the data source. Tiles themselves are 256x256 pixel PNG images in a Web-Mercator projection, which is consistent with many other web tile data sources (e.g., Google, Bing). Although these attributes remain consistent, the data size of tiles varies depending on how the original image was rendered. To incorporate a varying tile data size the prototype assumes a normal distribution of sizes in the entire dataset. A mean of $\boldsymbol{\mu}$ = 14,146 bytes and standard deviation of $\boldsymbol{\sigma}$ = 4,065 bytes were calculated from a sample of 963 OpenStreetMap tiles. As the prototype simulates user movement, tiles are created with a size according to a normal distribution of these values, which will determine both the cost for retrieving a tile wirelessly and the impact each tile has on the amount of available cache space.

6.1.3 Wireless Coverage

In addition to tile size, the data transmission rate of a cellular connection also determines the cost for retrieving a map tile wirelessly (i.e., *retrieval_time* ≈ *data_size* / *transmission_rate*). Although data rates fluctuate depending on carrier, cellular standards, network user load, signal strength, etc., the International Mobile Union (IMU) called for the 3G standard to provide a minimum speed of 348 kbit/s for a moving vehicle (ITU, 2005). The prototype implementation assumes a consistent transmission rate of 348 kbit/s for downloading tiles when connected.

In reality, one might experience intermittent connectivity as they travel by vehicle along a road network, since one hundred percent wireless coverage is not yet available among any wireless carrier. The pre-fetching simulations do not model intermittent connectivity in an environment, since the objective is to measure pre-fetching accuracy of two direction-based models. This thesis argues, however, that a pre-fetching strategy which yields a higher accuracy and thus higher reliability of having the necessary data in the future implies that it would also prove beneficial in an environment where a wireless connection is intermittent. This rationale assumes that a connection is indeed available part of the time, however. Given a mobile device equipped with GPS, it is still possible to record a location history when in a cellular dead-zone. When a connection does become available one exploits it by pre-fetching data in anticipation that the connection may not be available soon after. One might test this by varying the level of connectivity and measure the confidence in having the necessary data during periods where connectivity is lost. Pre-fetching clearly would become less effective when connectivity becomes more sparse as there is little opportunity for pre-fetching at all.

6.1.4 Pre-fetching and Replacement Policy

Cache management for the prototype implementation includes both pre-fetching and removal of tiles in a limited size memory buffer. Another control variable for the simulation is the size of the memory buffer. As previously mentioned, the tile cache can include a main memory cache, file cache, database cache, or combination of these. The experiment will consider a main memory tile bitmap cache only of a maximum size of 64 tiles or 896 KB. As tiles are fetched and pre-fetched the tile buffer fills and once either threshold is crossed a cleanup policy is invoked to rid the cache of unwanted tiles.

Chapter 5 discusses some strategies for removing tiles from a memory buffer, one through temporal locality (LRU) and the other through spatial locality (FAR). Since the prototype will be pre-fetching tiles in anticipation of movement, the removal algorithm should refrain from discarding pre-fetched tiles as this would be counter-productive. Instead, whenever the buffer reaches either the maximum tile threshold or the maximum data size threshold, it ranks cached tiles according to last access time and discards the least recently used tiles (i.e., LRU).

As the simulation moves from point to point along a trajectory, the pre-fetching algorithm anticipates future needed tiles based on this movement and queues them up for retrieval. First, the algorithm identifies the a set of tiles to prioritize based on movement, defined as any tile within a maximum distance of the current center tile and of the same zoom level. Each tile is then assigned a weight based on its distance from the center tile (***d***), its angle to the center tile relative to the direction of movement (**α**), and the speed movement (***s***). Tiles in line with the direction of movement should receive the highest weight, while a greater speed corresponds to heavier weighting of tiles further out in the direction of movement. To calculate these values the pre-fetching algorithm uses a variation of the standard normal distribution function that captures these three variables (Eq. 6.1) (Figure 6.2).

$$\frac{1}{d}e^{-\frac{\alpha^2}{1/\sqrt{s}}} \tag{6.1}$$

The tile weighting function determines a weight for each tile based on a curve such as that shown in Figure 6.2a. Values along the *x*-axis correspond to **α** and range from -π to π. The speed of the moving object is equivalent to the function's standard deviation and so as speed increases, the bell curve becomes narrower and tiles are weighted higher in the direction of movement. The distance of the weighted tile to the current center tile affects the overall weight as well, where tiles closer to the center will receive a slightly higher weight. In essence, tile weighting is meant to account for both distance and direction of movement for anisotropic pre-fetching only, whereas isotropic pre-fetching only accounts for distance. Tile weighting could certainly be implemented differently, such as through the use of a trigonometric function or a multivariate normal distribution (Hansen, 2008). Such functions would likely produce similar results as they are also feature bell-shaped curves. The prototype uses a tile weighting function since, like document retrieval in search engines, items of interest are determined by their relevance, where relevance in this instance is determined in large part by locality and direction.

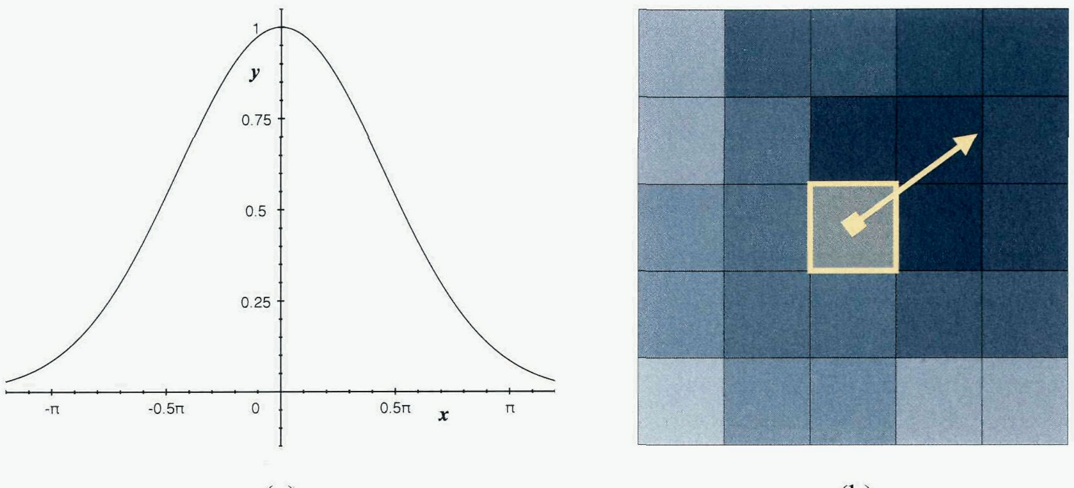


Figure 6.2. Tile weighting function (a) and sample results (b). Darker shades imply a larger weight.

6.1.5 Pre-fetching Metrics

Pre-fetching map tiles is an information retrieval process similar to a querying a web search engine. Google and Yahoo! search engines have specialized algorithms to rank documents so as to return the most relevant search results to its user. Similarly, tile pre-fetching aims to target a set of tiles (albeit a much smaller list than a web search) and prioritizes them based on future relevance to the map. One can measure tile pre-fetching in terms of precision and recall since the prototype is essentially performing a binary classification on a group of tiles, marking them as either one to pre-fetch (i.e., positive classification) or one not to pre-fetch (i.e., negative classification). ***Precision*** refers to the ratio of tiles successfully pre-fetched and later found relevant (i.e., the map consumed this tile from the cache and thus is a true positive) to the total number of pre-fetched tiles. ***Recall*** is the ratio of successfully pre-fetched tiles to the total number tiles requested by

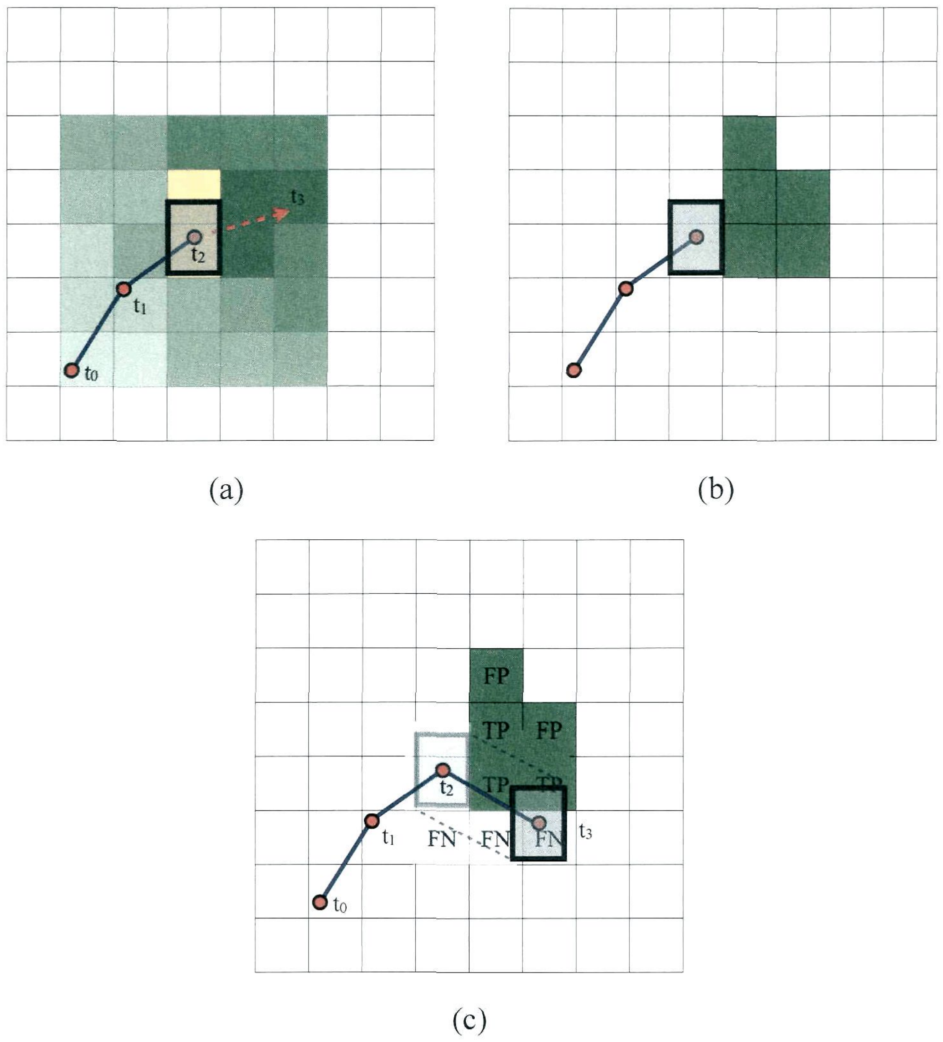


Figure 6.3. Tile pre-fetching: (a) pre-fetching prioritization of tiles of $d <= 2$ based on Eq. 6.2, (b) top 5 tiles are pre-fetched, (c) simulation records correct and incorrect classifications.

the map. At each sample point along a trajectory the cache labels any pre-fetched tiles from the previous movement according to a classification context or confusion matrix

		Actual Classification	
		+	-
Predicted Classification	+	tp (true positive)	fp (false positive)
	-	fn (false negative)	tn (true negative)

Table 6.1. Classification Matrix.

(Table 6.1). All tiles within a distance of 10 tiles from any point on the trajectory are initially labeled as true negatives so as to only consider a small number of tiles (depending on the zoom level the tile space can be composed of millions of tiles). Tiles are labeled as false positives if they are pre-fetched and later discarded from the cache without ever being within the viewport.

Performance measures for predictive processes can be carried out using measures of *precision* and *recall* (Olson and Delen, 2008). A *receiver operating characteristics* (ROC) curve, for example, helps to visualize the relationship between *true positives* and *false positives*. The ROC curve plots the *true positive rate* (Eq. 6.3) against the *false positive rate* (Eq. 6.5). The true positive rate is also known as *sensitivity* while the false positive rate is one minus the *true negative rate* (Eq. 6.4) (Olson and Delen, 2008). *Accuracy* is another measure of success in information retrieval that can measure how well a pre-fetching algorithm identifies or excludes future needed data (Eq. 6.6).

$$\textit{True Positive Rate (sensitivity)} = \frac{TP}{TP + FN} \tag{6.3}$$

$$\textit{True Negative Rate (specificity)} = \frac{TN}{TN + FP} \tag{6.4}$$

$$\textit{False Positive Rate} = 1 - \frac{TN}{TN + FP} \tag{6.5}$$

$$\textit{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{6.6}$$

A trajectory is a sampled representation of the viewport's movement through the map space. Not only will the simulation need to track which tiles intersect the viewport's bounds at each timestamp along the trajectory, but also any tile that might intersect this moving window between timestamps. To determine the entire set of tiles that intersect the swath we simply model the moving viewport as a polygon (Figure 6.2c) and test each tile using a point-in-polygon algorithm. If any of the four points of a particular tile are within the bounds of the polygon, the tile is said to be "required" by the map for rendering. This approach assumes that (1) the viewport moves in a straight line between two trajectory sample points and (2) both the tile width and height are smaller than the width and height of the viewport, which is necessary to ensure an accurate point-in-polygon assessment.

6.2 Simulation Results

In total the tile pre-fetching prototype simulates 251 distinct trajectories (a product of trajectory normalization from the larger dataset), each of which consists of 40 sample points. Simulations are run for both *isotropic* and *anisotropic* models while varying the data volume allowed to pre-fetch. That is, for each trajectory, the prototype simulates a maximum tile pre-fetch of 0-10 for each model, yielding a total of 5,522 simulations (2*11*251). Trajectories vary in *complexity*, herein defined as the average absolute change in heading that the trajectory experiences along its duration. Trajectory complexity values range from 9° to 70° with a mean of 32° and standard deviation of 14°.

As previously stated, the simulation considers a transmission rate of 348 kbps and average tile size of around 14 KB. Trajectory sample points are taken every 30 seconds, yielding a limit on how much data could potentially be pre-fetched per sample and is actually considerably more than 10 tiles. Realistically, however, transmission speeds vary and the time it takes to establish a connection and write data to disk or a database adds to the total time it takes to retrieve one tile. A 10 tile maximum is an appropriate limit as it does lead to some insight on how data volume affects pre-fetching accuracies in conjunction with trajectory complexity.

The prototype implementation is a .NET Windows Console application and all simulations were output to CSV text files. In order to efficiently query and format subsets of the results all data were inserted into three tables of a Microsoft SQL Server database (Table 6.2).

Trajectory					
trajectory_id	*num_points*	*num_turns*	*vehicle_type*	*polyline*	*avg_heading_change*
TrajectoryPoint					
trajectory_pt_id	*trajectory_id*	*latitude*	*longitude*	*speed*	*heading*
Simulation					
simulation_id	*max_prefetch*	*max_distance*	*zoom*	*coverage*	*isotropic*
trajectory_id	*tp*	*tn*	*fp*	*fn*	*tp_rate*
fp_rate	*fn_rate*	*accuracy*	*total_hits*	*total_misses*	*hit_ratio*

Table 6.2. Trajectory database table schema.

6.3 Analysis of Results

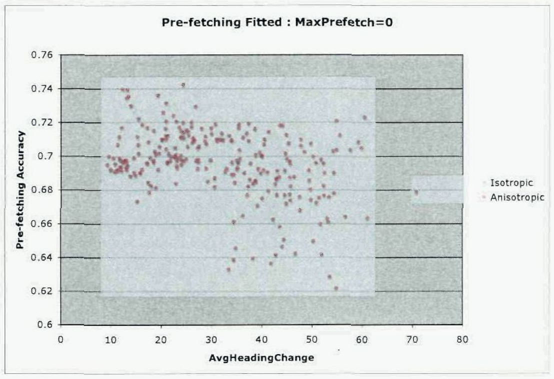
The initial presumption of this thesis was that anisotropic pre-fetching would significantly outperform isotropic pre-fetching in a situation where movement is not totally random, a requirement that is captured through use of real vehicle movement data within a city road network. To test this hypothesis, each group of pre-fetching accuracy values are found to be significantly different or not using a student's t-test. For each data volume level (0-10), each trajectory has an accuracy value for both anisotropic and isotropic pre-fetching. At each level, we apply a t-test to the two groups of anisotropic and isotropic values while also visualizing the results to assess their differences. Table 6.3

displays each data volume level (max_prefetch, **mp**) along with the p-value obtained using a t-test. The test is two-tailed since we are first trying to gauge whether or not the two models yield significantly different results. Included in the table are the isotropic and anisotropic mean accuracies, μ_I and μ_A , respectively, and the differences between the two means.

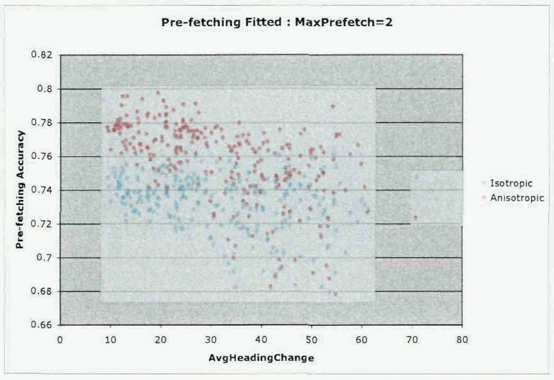
mp	**μI (accuracy)**	**μA (accuracy)**	**μA - μI**	**p <**
0	70.38%	70.38%	0%	n/a
1	72.24%	73.60%	1.36%	9.41e-57
2	74.06%	76.51%	2.45%	6.94e-70
3	74.79%	78.63%	3.84%	4.8e-80
4	75.42%	80.11%	4.69%	1.29e-66
5	75.90%	81.23%	5.33%	1.77e-61
6	76.21%	81.72%	5.51%	9.95e-56
7	76.24%	81.36%	5.12%	2.77e-45
8	75.21%	80.09%	4.88%	2.44e-34
9	74.18%	77.88%	3.7%	1.21e-23
10	72.95%	74.89%	1.94%	7.7e-09

Table 6.3. Pre-fetching average accuracy and p-values.

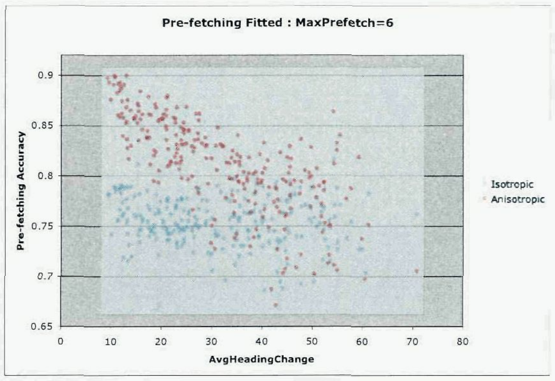
According to the p-values obtained and shown in Table 6.3, in all instances except for when **mp** is zero, the groups of anisotropic and isotropic pre-fetching accuracy values are significantly different. Looking at the mean values and differences of means for each, it appears that the anisotropic algorithm always yields a greater mean pre-fetching accuracy. However, spatial movement patterns in general are complex (Dodge et al., 2008), let alone the endless possibilities of vehicle movement, and so it is not ideal to use one of the above pre-fetching methods exclusively. Instead, in order to capture a wider range of scenarios, it is possible to derive a rule of thumb for when to use one pre-fetching model over the other based on the complexity of movement. Each trajectory in the dataset can be classified by complexity, earlier defined as the average absolute heading change between trajectory sample points. Plotting trajectory complexity versus pre-fetching accuracy for both models shows that anisotropic pre-fetching clearly outperforms that of isotropic pre-fetching for trajectories with a relatively small average heading change (Figure 6.4b,c,d). However, for certain ranges of trajectory complexity, it is less clear which model is best. In Figure 6.4d, where the maximum pre-fetch range is 10 tiles, it appears as though the isotropic model actually proves better on average than the anisotropic model for the more complex trajectories (e.g., **AvgHeadingChange** > 40 degrees).



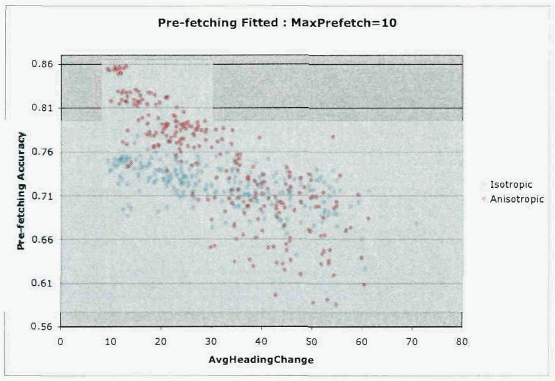
(a)



(b)



(c)



(d)

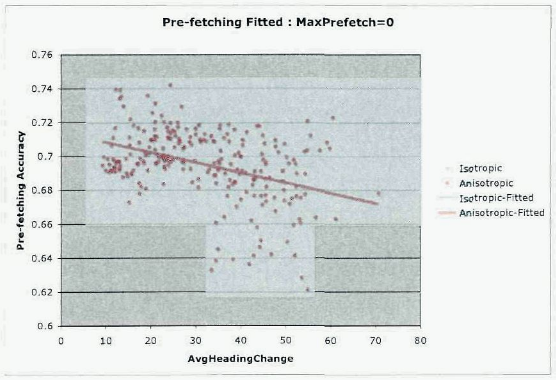
Figure 6.4. Trajectory complexity versus pre-fetching accuracy at (a) max_prefetch = 0, (b) max_prefetch = 2, (c) max_prefetch = 6, and (d) max_prefetch = 10.

To investigate the effect of trajectory complexity on isotropic and anisotropic pre-fetching and where one model might outperform the other, a trend line for the datasets of each model at each data volume were calculated using linear regression. By intersecting the two trend lines, one can get a rough idea for when to use one model over the other. Each linear regression model yields a trend line slope (**a**), y-intercept (**b**), p-value explaining the significance of the model parameters, and $\mathbf{R^2}$ showing how well the model can predict pre-fetching accuracy based on the trajectory (Table 6.4). The average heading change for which the two trend lines intersect is shown as **x'**.

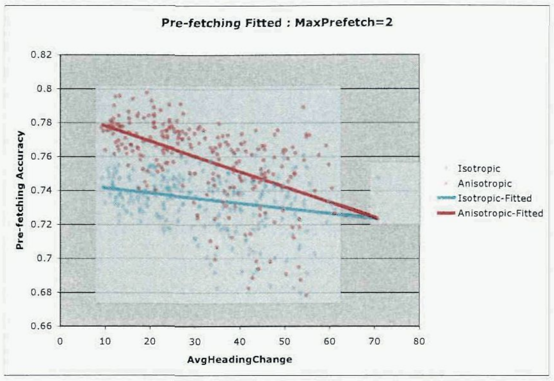
mp	a_I	b_I	a_A	b_A	x'	$p_I <$	$p_A <$	R^2_I	R^2_A
0	0.7144	-0.0006	0.7144	-0.0006	n/a	0.0000	0.0000	0.1525	0.1525
1	0.7260	-0.0004	0.7508	-0.0007	82.67	0.0000	0.0000	0.0815	0.2222
2	0.7445	-0.0003	0.7875	-0.0009	71.67	0.0000	0.0000	0.082	0.2937
3	0.7477	-0.0002	0.8175	-0.0012	69.80	0.0064	0.0000	0.0256	0.3586
4	0.7483	0.0000	0.8435	-0.0015	63.47	0.8380	0.0000	-0.0038	0.3969
5	0.7553	-0.0001	0.8704	-0.0020	60.58	0.4569	0.0000	-0.0017	0.4654
6	0.7632	-0.0002	0.8916	-0.0025	55.83	0.0271	0.0000	0.0155	0.5114
7	0.7704	-0.0005	0.9073	-0.0031	52.65	0.0000	0.0000	0.0821	0.5794
8	0.7610	-0.0005	0.9126	-0.0037	47.37	0.0000	0.0000	0.0828	0.6305
9	0.7621	-0.0009	0.8983	-0.0040	43.94	0.0000	0.0000	0.2114	0.6603
10	0.7544	-0.0010	0.8723	-0.0041	38.03	0.0000	0.0000	0.3448	0.685

Table 6.4. Linear regression results.

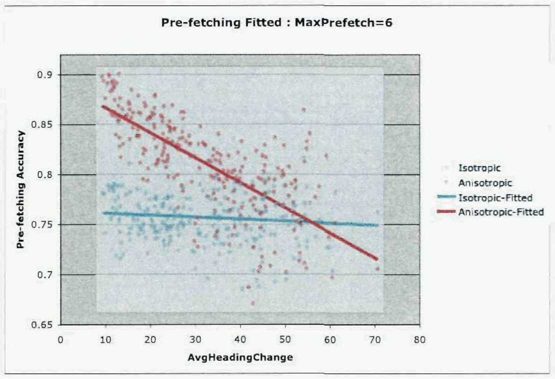
Small p_values suggest that one is confident that the estimated values of the regression model are correct. While the majority of the models do yield very small p_values, there are a couple exceptions, particularly for the isotropic pre-fetching results when **mp=4** and **mp=5**. The anisotropic models also produce higher $\mathbf{R^2}$ values, meaning that these models are a better fit for the sample data. Even so, this chapter aims at obtaining a general rule that may not be flawless in every situation, which is surely the case with the many scenarios of moving objects. Nonetheless, one can now perform a simple analysis on the intersection values of the trend lines. As the data volume for pre-fetching increases, we see that the intersection of the trend lines for isotropic and anisotropic pre-fetching decreases (Figure 6.5). In fact, plotting **max_prefetch** against **x'** along with a trend line shows a fairly consistent decrease in the intersection values (Figure 6.6). This result may be what one would expect, since pre-fetching more data in



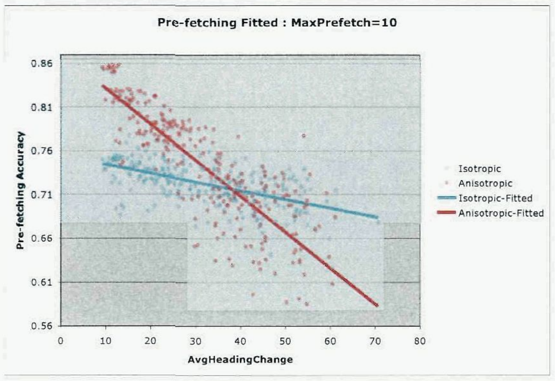
(a)



(b)



(c)



(d)

Figure 6.5. Model trend lines at (a) max_prefetch = 0, (b) max_prefetch = 2, (c) max_prefetch = 6, and (d) max_prefetch = 10.

the direction of movement will typically only prove beneficial if the movement is reasonably straight. If, on the other hand, a moving object is consistently changing direction, it is better to anticipate movement equally in all directions. Using the resulting trend line, one could employ a dynamic pre-fetching algorithm based on the history of movement, specifically the average absolute heading change of the movement. Depending on the average heading change of the object's movement over the past *n* sample points and the amount of data being retrieved, either a direction-dependent or direction-independent method is used. Doing so probably would not yield accuracies greater than the highest accuracies resulting from the simulations, but should result in a lesser decline in pre-fetching accuracy as trajectory complexity increases.

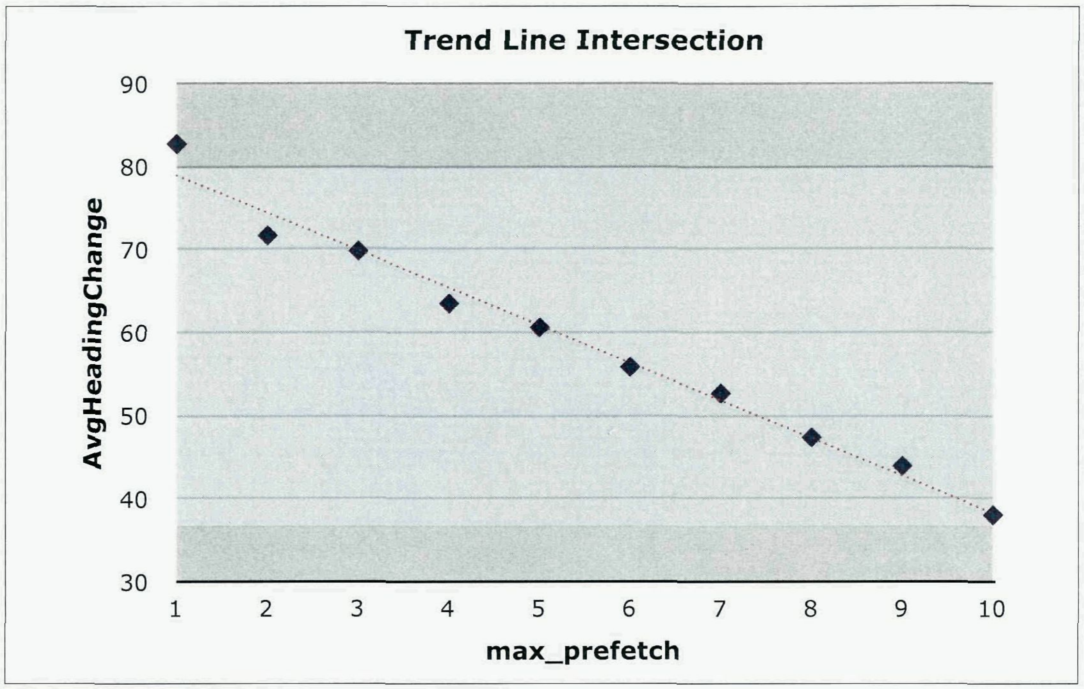


Figure 6.6. Trend Line Intersections.

The results gathered from the simulations are highly related to the experimental setup, where several fixed parameters might otherwise have a varying degree of effect on the results. Pre-fetching algorithm, caching policy, tile size (width, height, #bytes), assumed data transmission speed, trajectory normalization, zoom level, and other variables were set so that the simulations could model a real world example as much as possible while still leading to practical results. A mobile mapping application that implements data pre-fetching should, therefore, consider these parameters into its workflow and adjust its pre-fetching policy accordingly.

Also, accuracy is one of many possible measures to assess pre-fetching success. It should be noted that even when no pre-fetching takes place accuracy values remains positive, due in part to the nature of the pre-fetching classification of tiles. Looking at Eq. 6.6, it is clear why this is so. **TP** (# of tiles pre-fetched then consumed) and **FP** (# of tiles pre-fetched but not needed) are zero. However, **TN** (# of tiles not pre-fetched and needed) and **FN** (# of tiles not pre-fetched yet needed) are greater than zero. In fact, **FN** will be larger given that no pre-fetching takes place. Thus, the overall accuracy value will be, in theory, less whenever no pre-fetching takes place. From the results, it is clear that accuracy values are greater when **mp** is greater than zero.

6.4 Discussion of Hypothesis

In this chapter simulated tile pre-fetching strategies tested the hypothesis that for non-random movement through space one can augment on-demand spatial data downloads by pre-fetching data in the direction of movement, which would yield higher accuracies than a strategy that does not include previous user location(s). Results showed that this is true for situations when the frequency and magnitude of directional change in the movement is relatively low. However, as complexity increases in trajectories (but not necessarily randomness since movement is constrained by a road network) an isotropic strategy sometimes returns better accuracies. The main conclusion here is that no one pre-fetching model will prevail in all scenarios with regard to movement and that implementing a pre-fetching component to a mobile mapping application should take into account this variability.

6.5 Summary

This chapter describes a prototype implementation that models a real world scenario of vehicle movement through a road network and assesses the performance of two direction-based pre-fetching algorithms while varying the pre-fetching data volume. Trajectory data were retrieved from the R-tree Portal website and normalized for consistency between sample points and trajectory length. Linear regression allowed for the derivation of trend lines which led to an understanding of how one can tailor a pre-fetching algorithm to account for a wider range of vehicle movement scenarios. While one might intuitively think that pre-fetching in the direction of movement will always yield a greater reliability of obtaining future needed data, the performance is to a degree reliant on the underlying complexity of the movement. It was found that increasing the pre-fetching data volume, which would be useful in situations where communication to remote data servers is intermittent, is effective only for movement patterns that are relatively straight and, hence, predictable. However, in a scenario where movement is "more random" (e.g., city block driving where a moving object changes direction frequently), a direction-independent pre-fetching algorithm is preferable. These findings suggest that this thesis's original hypothesis does not hold true for all movement scenarios and that to effectively implement a pre-fetching component for commercial mobile maps, one should consider multiple approaches to cover the range of movement possibilities. Doing so can prove beneficial to usability of mobile maps by decreasing latency involved with remote retrieval of spatial data and better prepare a device for disconnected environments.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This chapter provides a summary of content for this thesis on "Mobile Map Browsers: Anticipated User Interaction for Data Pre-fetching." A prototype implementation for assessing the performance of map pre-fetching for mobile devices was developed through an investigation of the core issues and conventions of mobile map browsers (Section 7.1). This section also concludes with the results and conclusions found from the prototype implementation and pre-fetching simulations. The findings of this thesis are ripe with potential for expanding on and can lead to other interesting research projects, which this chapter addresses in Section 7.2.

7.1 Summary

The goal of this thesis is to investigate methods to utilize past user interactions with a mobile map for pre-fetching data in anticipation of future interaction. Mobile computing is challenging with respect to storage space, computing power, battery life, and intermittent wireless communication. Because of storage limitations it is not feasible to preload a high-resolution, global coverage map on a mobile device and, for a truly dynamic map, it is necessary to retrieve map data from a remote data repository in an *on-demand* fashion. This convention, however, causes delays when downloading map imagery over slow transmission rates and even renders the map unusable whenever a data connection is unavailable. In addition, wireless transfer of spatial data is costly (e.g., storage, battery, money). Given the sensor-rich nature of mobile devices, however, it is possible to record the location history of a device or simply the interaction history for the mobile map user. This thesis hypothesizes that since movement through space (e.g,. a vehicle moving along a road network) is realistically non-random, it is possible to derive a direction-dependent pre-fetching algorithm whose accuracy is significantly greater than a direction-independent pre-fetching algorithm.

Chapter 2 summarizes relevant literature, concepts, and issues for mobile computing, mobile and spatial caching, visual information browsers, trajectories and moving object databases, and location prediction. These topics provide a foundation for implementing a software prototype to model a real-world scenario where pre-fetching is beneficial.

Chapter 3 explores the fundamental interactions one typically invokes on a mobile map browser. These interactions include direction manipulation of the map through pinch or swipe gestures and refreshing of the map display as a result of sensor-derived changes of a user's position, orientation, or velocity. Nevertheless, all interactions come down to a three geometric transformations of the viewport: *translation*, *rotation*, and *scaling*. Each viewport transformation affects a small set of tiles within the immediate neighborhood of the map's current center and if one knows the set of candidate tiles needed by the map in the immediate future, it is possible to derive some rules for map pre-fetching. Populating the cache, or tile buffer, with data based on these pre-fetching strategies is the focus of Chapter 4. Whenever a mobile map browser completes the retrieval of on-demand data and realizes that a data connection is available, the opportunity to pre-fetch data on behalf of the user can improve the cache hit ratio and better prepare the device in the event of a dropped data connection.

In addition to populating a tile buffer, cache management policies include rules for removing data from the memory buffer when necessary, such as when the cache size exceeds a maximum number of bytes. Chapter 5 discusses both temporal and spatial cache removal policies and the implications of each on a tile buffer. Whereas traditional caching policies deal with the temporal locality of items in the cache and discard the *least recently* or *least frequently* used items, other implementations such as Furthest Away Replacement (FAR) compare items of a cache spatially and discard the items furthest away from a given location. Mobile map browsers consume location-dependent data and could therefore use either a temporal, spatial, or both cache removal policies.

Chapter 6 describes a software implementation that models moving objects along a road network. The prototype simulates a tile cache management policy using either an isotropic or anisotropic pre-fetching model. Simulations pre-fetch and discard tiles from a virtual cache while recording statistics on the binary classification of tiles (*pre-fetched* and *not pre-fetched*) in order to support a metric for pre-fetching success. Using pre-fetching accuracy (Eq. 6.5) as the metric for comparison, this thesis shows that results for isotropic and anisotropic pre-fetching are significantly different and that for relatively straight movement anisotropic pre-fetching yields better results. However, this is not always the case, as sometimes isotropic pre-fetching is preferable for more *complex* movement, which is defined in this thesis as the average absolute heading change for a trajectory. At each pre-fetching level (i.e., data volume allowed for pre-fetching), there is a point of trajectory complexity at which the two pre-fetching models produce relatively equal pre-fetching accuracies. These thresholds were found by intersecting the trend lines of both pre-fetching models which were calculated using linear regression. Threshold values were also found to be a function of the pre-fetching data volume, leading to a general rule of thumb when pre-fetching tiled data for a mobile map browser. Depending on the amount of data one is interested in pre-fetching, there exists a movement complexity threshold. Movement complexity below this value should employ direction-dependent pre-fetching while complexity above the value should use direction-independent pre-fetching.

7.2 Future Work

There are several ways to extend the work presented in this thesis on anticipatory interaction for data pre-fetching. The Chapter 6 prototype assumes a scenario in which the mobile device is essentially unknowledgeable of the local environment and instead simply exploits past user interactions for predicting what data to pre-fetch for future use. The simulations use a history of GPS coordinates to determine a movement direction and pre-fetch data weighted in that direction. If one were to expand on this approach, there a few ways in which they might do so.

- In addition to direction, a pre-fetching algorithm could utilize the speed of movement to target specific levels of detail for pre-fetching. Similar to "Speed-dependent Automatic Zooming" (Igarashi and Hinckley, 2000), one might have *Speed-dependent Automatic Pre-fetching* which follows the same concept. A device moving at high speeds (e.g., 70 mph on a highway) should not display (or pre-fetch) 1-meter resolution imagery since the flow of information is far too fast to be of any use. Speed could also have an effect on how tiles are weighted in the pre-fetching process, where a increase in speed means tiles are weighted heavier in the direction of movement (Eq. 6.2). The trajectories described in Chapter 6 were normalized for constant speed. In doing so post-simulation analysis could focus on direction as the only spatial / temporal variable.

- Chapter 4 discusses pre-fetching tiles at coarser levels of detail so as to increase geographic coverage of the cache while decreasing the amount of data. This certainly can prove beneficial but a metric is needed for assessing the cache performance when consuming tiles of a coarser or finer level of detail for a given resolution. For example, it would not be correct to pre-fetch the root tile (i.e., a single tile spanning the entire map space) and consider this as a cache hit when a tile does not exist at a higher zoom level. Doing so would result in a cache with a 100% hit ratio when this is in fact misleading.
- Two major advantages of pre-fetching tiles for a mobile map browser are a reduced latency when refreshing the map and a more robust cache for situations such as when a data connection is unavailable. Although not modeled in the prototype of Chapter 6, one can design simulations to model interconnectivity and measure any relationship between data volume, level of interconnectedness, and pre-fetching accuracy (or other caching metric). Intuition suggests that for sparser connectivity areas increasing the data volume results in a more reliable cache.
- Simulations in Chapter 6 only consider the interaction history of a user to determine which tiles to pre-fetch. A more robust solution might be one which incorporates server-side GIS resources to further refine data for pre-fetching.

Consider the moving vehicle scenario from the simulations. Given that it is known the mobile device is traveling along a road network, one could use a web service with GIS processing capabilities to (1) select all possible locations a device could be within *x* time units on the road network and (2) determine which map data are relevant to these locations for pre-fetching. Similarly, if data for cellular coverage were available, a web service could determine the necessity of pre-fetching data given the proximity of the user to any cellular dead zones. If determined that the device is nearing a dead zone then pre-fetching for that region might be helpful. These and other types of GIS data relevant to moving objects and wireless communication are interesting research topics for how one might augment mobile map cache management through data pre-fetching.

- Even with the assistance of GIS data repositories, pre-fetching map data is a result of a software agent anticipating where the user will request a map view for next. However, interactive mapping software often allows users to query for people, places, or directions. Location prediction from user input, despite not being implemented in many commercial or open source packages, is another relatively straightforward way of preparing a mobile device with maps.

BIBLIOGRAPHY

About Mobile Technology and IMT-2000. International Telecommunication Union. 2005. http://www.itu.int/osg/spu/imt-2000/technology.html#Cellular%20Standards %20for%20the%20Third%20Generation. *Accessed* April 19, 2010.

Barbara, Daniel and Tomasz Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments. *The VLDB Journal* 4.4 (1995): 567-602.

Bertolotto, Michela and Max Egenhofer. Progressive Vector Transmission. *7th ACM Symposium on Advances in Geographic Information Systems* (Kansas City, MO). New York, NY: ACM, 1999. 152-157.

Buttenfield, Barbara. Transmitting Vector Geospatial Data Across the Internet. *Lecture Notes in Computer Science* 2478 (2002): 51-64.

Chakka, V. P., Adam Everspaugh and Jignesh Patel. Indexing Large Trajectory Data Sets with SETI. In *Proceedings of the Conference on Innovative Data Systems Research, CIDR* (Asilomar, CA). 2003.

Chrisman, Nicholas. Deficiencies of Sheets and Tiles: Building Sheetless Databases. *International Journal of Geographical Information Systems* 4.2 (1990): 157-167.

Dodge, Somayeh, Robert Weibel and Anna-Katharina Lautenschutz (2008). Towards a taxonomy of movement patterns. *Information Visualization,* 7.3,4 (2008): 240-252.

Dunham, Margaret and Vijay Kumar. Location Dependent Data and its Management in Mobile Databases. Washington, DC: IEEE Computer Society, 1998. 414.

Egenhofer, Max and Werner Kuhn. Beyond Desktop GIS. *GIS PlaNET*: Lisbon, Portugal http://www.spatial.maine.edu/~max/BeyondDesktopGIS.pdf (1998).

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. RFC2616: Hypertext Transfer Protocol--HTTP/1.1. RFC Editor United States. 1999.

Forlizzi, Luca, Ralf Guting, Enrico Nardelli and Markus Schneider. A Data Model and Data Structures for Moving Objects Databases. *ACM Sigmod Record* 29.2 (2000): 319-330.

Furnas, George and Benjamin Bederson. Space-Scale Diagrams: Understanding Multiscale Interfaces. *Proceedings of the SIGCHI Conference on Human Factors* *in Computing Systems* (Denver, CO). New York, NY: ACM Press/Addison-Wesley Publishing Co., 1995. 234-241.

Guting, Ralf, Thomas Behr and Jianqiu Xu. Efficient k-Nearest Neighbor Search on Moving Object Trajectories. *The VLDB Journal* 1.19 (2010).

Handy, Jim. *The Cache Memory Book.* San Diego, CA: Academic Press, Inc., 1998.

Hansen, Nikolaus. The CMA Evolution Strategy: A Tutorial. 2007.

Harrower, Mark and Benjamin Sheesley. Designing Better Map Interfaces: A Framework for Panning and Zooming. *Transactions in GIS* 9.2 (2005): 77-89.

Igarashi, T. and K. Hinckley. Speed-Dependent Automatic Zooming for Browsing Large Documents. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (San Diego, CA). New York, NY: ACM, 2000. 139-148.

Inmarsate Pricing. http://www.thepowerofbgan.com/documents/inm_bgan_fact_prepaid.pdf. *Accessed* August 23, 2010.

Ishikawa, Yoshiharu, Yuichi Tsukamoto and Hiroyuki Kitagawa. Extracting Mobility Statistics from Indexed Spatio-Temporal Datasets. *Spatio-Temporal Database Management.* 2004.

Jackson, Jeffrey. *Visualization of Metaphors for Interaction with Geographic Information Systems.* M.S. Thesis, Department of Spatial Information Science and Engineering, University of Maine, 1990.

Jing, Jin, Abdelsalam Helal and Ahmed Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys* 31.2 (1999): 117-157.

Larrabeiti, David, Ricardo Romeral, Manuel Uruena, Arturo Azcorra and Pablo Serrano. Charging for Web Content Pre-fetching in 3G Networks. *Lecture Notes in Computer Science* 3266 (2004): 358-367.

Laube, Patrick, Todd Dennis, Pip Forer and Mike Walker. Movement beyond the Snapshot--Dynamic Analysis of Geospatial Lifelines. *Computers, Environment and Urban Systems* 31.5. (2007): 481-501.

Lee, Dik Lun, Jianliang Xu, Baihua Zheng and Wang-Chien Lee. Data Management in Location-Dependent Information Services. *IEEE Pervasive Computing* 1.3 (2002): 65-72.

Macedo, J., C. Vangenot, W. Othman, N. Pelekis, E. Frentzos, E. Kuijpers, I. Ntoutsi, S. Spaccapietra and Y. Theodoridis. Trajectory Data Models. *Mobility, Data Mining and Privacy* (2008): 123-150.

Mummert, Lily Barkovic, Maria Rene Ebling and Mahadev Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. Copper Mountain, CO: ACM, 1995. 143-155.

Nagaraj, S. *Web Caching and its Applications*. Norwell, MA: Kluwer, 2004.

Nielsen, Jakob. Response Times: The Three Important Limits. in: *Usability Engineering*. San Francisco CA: Morgan Kaufmam, 1994.

Olson, David and Dursun Delen. Performance Evaluation for Predictive Modeling. *Advanced Data Mining Techniques* (2008):137-147.

Rabinovich, Michael and Oliver Spatscheck. Web Caching and Replication. *ACM Sigmod Record* 32.4 (2003): 107.

Reichenbacher, T. *Mobile Cartography--Adaptive Visualisation of Geographic Information on Mobile Devices*. Dissertation, Technical University Munich, Germany, 2004.

Ren, Qun and Margaret Dunham. Using Semantic Caching to Manage Location Dependent Data in Mobile Computing. *Proceedings of the 6th annual international conference on Mobile computing and networking*. New York, NY: ACM, 2000. 210-221.

Ren, Qun and Margaret Dunham. Using Clustering for Effective Management of a Semantic Cache in Mobile Computing. New York, NY: ACM, 1999. 94-101.

Ren, Qun, Margaret Dunham and Vijay Kumar. Semantic Caching and Query Processing. *IEEE Transactions on Knowledge and Data Engineering* 15.1 (2003): 192-210.

Saltenis, Simonas, Christian Jensen, Scott Leutenegger and Mario Lopez. Indexing the Positions of Continuously Moving Objects. *SIGMOD Record* 29.2 (2000): 331-342.

Satyanarayanan, Mahadev. Fundamental Challenges in Mobile Computing. *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*. New York, NY: ACM, 1996. 1-7.

Shneiderman, Ben and Catherine Plaisant. *Designing the user interface*. Reading, MA: Addison-Wesley, 1998.

Sistla, A. and Ouri Wolfson. Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering* 7.3 (1995): 471-486.

Sistla, Prasad, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and Querying Moving Objects. Citeseer, 1997: 422-433.

Tao, Yufei, Christos Faloutsos, Dimitris Papadias and Bin Liu. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. New York, NY: ACM, 2004. 611-622.

Theodoridis, Yannis, Jefferson Silva and Mario Nascimento. On the generation of spatiotemporal datasets. *Lecture Notes in Computer Science* 1651 (1999): 147-164.

Tobler, Waldo. A Computer Movie Simulating Urban Growth in the Detroit Region. Economic Geography 46.2 (1970): 234–40.

van Wijk, Jarke and Wim Nuij. Smooth and Efficient Zooming and Panning. *IEEE Symposium on Information Visualization* (InfoVis 2003), 2003.

Wolfson, Ouri, Bo Xu, Sam Chamberlain and Liqin Jiang. Moving objects databases: Issues and solutions. *10th International Conference on Scientific and Statistical Database Management* (Capri, Italy). 1998. 111-122.

Worboys, Michael and Matt Duckham. *GIS: A Computing Perspective*. London: Taylor & Francis, 2004.

Yang, Bisheng, Ross Purves and Robert Weibel. Efficient Transmission of Vector Data Over the Internet. *International Journal of Geographical Information Science* 21.2 (2007): 215-237.

BIOGRAPHY OF THE AUTHOR

Benjamin Weber was born on March 17, 1986 in the small, rural town of Teutopolis, Illinois. He attended and graduated Teutopolis High School in May 2004, finishing in the top 10% of the class academically. Benjamin spent his first year of college at Southern Illinois University at Carbondale in the Department of Geography. In the fall of 2005 he transferred to the University of Maine, where he received a Bachelor of Science in Information Systems Engineering in May of 2008. In the fall of 2008, he enrolled as a Masters of Science student in the Department of Spatial Science and Engineering. During his career as a graduate student, he was fortunate to hold an internship with Global Relief Technologies in Portsmouth, NH as a GIS developer and has accepted a full-time offer from the company starting in September 2010. Benjamin Weber is a candidate for a Master of Science degree in Spatial Information Science and Engineering for December 2010.