Responsive, scalable thinwire visualization: application to large geographic datasets

Chee Yap, Kenneth Been, Zilin Du

Courant Institute, New York University

ABSTRACT

When very large datasets are visualized in a **thinwire setting**, where the bandwidth is both limited and unreliable, current techniques break down. This paper proposes the **responsive visualization system** as a solution. We describe the architecture of such a system, and algorithmic challenges to support it. We introduce a natural and scalable visualization interface called the **zooming telescope**. We apply our techniques to the visualization of the TIGER dataset, a geographic description of the United States. We describe a suite of automatic and semi-automatic tools for processing the TIGER dataset, including the construction of level-of-detail (LOD) hierarchies.

Keywords: Responsive, scalable, thinwire, visualization, geographic, GIS, TIGER

1. INTRODUCTION

Many key issues of current visualization research relate to the largeness of datasets to be visualized. The normal mode for visualizing such data involves the rendering of views that are both partial and dynamic. A key example of this visualization mode is the walkthrough.[1] Techniques that are critical for the success of such visualization include the multiresolution and level-of-detail (LOD) hierarchies,[2,3] data culling methods,[4] out-of-core data structures and algorithms,[5] view-dependent renderings,[6] etc.

In this paper, in addition to the largeness of datasets, we address issues arising in limited bandwidth settings, such as visualizations on the web and via personal digital assistant (PDA) devices. While the above-mentioned techniques remain important, new techniques are needed. For instance, all walkthrough visualization architectures assume some minimum bandwidth, usually the data transfer rate between the secondary memory and the main memory. Although this bandwidth is insufficient for large datasets, it is still fairly substantial *and* reliable. In contrast, the bandwidth we face may be several orders of magnitude smaller, as well as unpredictable. We use the term **thinwire** to cover both these limitations on bandwidth. We want to construct visualization system that are still functional in thinwire connections. Below, we will characterize such systems as **responsive**. There is much related work on web-based computing environments (e.g., in geometric computing[7] and in simulation and steering[8,9]) in which visualization is an important major objective. Our techniques are orthogonal to these concerns, and certainly can be applied in these settings (when there is large size and thinwire issues).

Visualization is a cognitive act whose goal is rather subjective, and certainly ought not to be equated with image transmission. The role of visualization systems is to provide means for a user to quickly explore the dataset and focus on the "relevant data". The data transmission requirement is, roughly speaking, the sum of two quantities: relevant data (R) + exploration overhead (E). We may have no control over (R), but we aim to reduce (E). The quantity (R)+(E) can be substantially less than anything achievable by image compression, even with reasonable lossiness. An idea that goes beyond image transmission is progressive transmission,[10,11] which aims to transmit the important details of an image first. Here "importance" is based on the RMS distortion, but any mathematically defined measure of relevance is surely not the right measure because it depends only on the image, not the user's visualization goal. In contrast, our approach is based on constructing "responsive visualization systems", which provide the user with tools to define his or her notion of relevance. As a simple example, if we first provide a low resolution image (iconic image) of a large image (underlying image), the user can click at various points in the iconic image to direct the system to transmit higher resolution data in the neighborhood of the corresponding points in the underlying image. This fulfills the goal of letting the user define relevance. Such an approach has been explored in our previous work,[12, 13] where the use of multi-foveated images was demonstrated to be particularly useful, and a method for producing such images from uniform raster images using wavelets was described. The mathematical theory of foveation has also been extensively explored.[14, 15] (Note that foveated images, or more generally, space-variant images,[16] are a generalization of multiresolution techniques.) But the major difference between the present work and that on multi-foveated images is that we now address vector datasets, not raster images. Foveation is considerably harder to achieve here. In lieu of foveation, we create the zooming telescope interface in this paper. This device will allow us to recover most of the data reduction advantages of multi-foveated images.

First, it is important to understand "responsiveness" as a property that is *independent* of bandwidth. Of course, if bandwidth temporarily reduces to nothing, we do not expect the system to generate new data, but a responsive system should not freeze under this circumstance, and will continue to provide useful responses to the user. Second, it is important to understand "responsiveness" as the ability to be flexible and to accommodate the user's whims and mind changes. For instance, using the iconic image/underlying image example above, suppose the user first clicks at a point p (so details in the neighborhood of p are requested and may begin to stream in). But the user quickly changes his or her mind (perhaps based on the first data seen), and decides to click another point q. A responsive system must be able to either abort the first request on p or at least prioritize the new request on q. In summary: we will define **responsiveness** as the ability of the system to (1) respond to the user's commands as naturally and quickly as possible, using available data, and (2) adjust its responses to the prevailing bandwidth. In particular, responsive systems must act reasonably under a variety of unreasonable and variable conditions, including user mind changes and the zero bandwidth condition.

The use of LOD hierarchies will be a critical element in achieving responsive behaviors. To see this, suppose every request for (visual) data has two **nominal** parameters: range ρ and scale σ. Range is typically some "rectangular region" of the underlying image, and scale is a numerical parameter that is often a function of the zoom level. We say "nominal" because a responsive system may modify both parameters: when the bandwidth is small, it should be able to reduce the scale* and/or range of a request to optimize other objectives such as time. This can be viewed as an online computational problem with some quality of service (QoS) criterion; see Chang and Yap[17] for a formalization and construction of competitive algorithms.[18, 19]

Our research has the following features:

1. We combine the issues of large datasets with the thinwire requirement. Known techniques for large dataset visualization break down in thinwire settings.

2. Our focus on "responsive visualization", which is the property to respond to user requests intelligently and promptly under unreliable and low bandwidth scenarios, requires special architecture and distinguishes us from other web technology issues (GUI, collaboration, etc).

3. Our zooming telescope is a natural visualization interface that is fully scalable with data size.

4. The emphasis on continuous zooming and panning in the thinwire setting is not only an important visualization feature, but leads to interesting new data structures and algorithmic questions.

5. We address the issues of map simplification and LOD hierarchies for the TIGER dataset. Algorithms and tools are outcomes of this effort.

6. Our transmission of vector data for visualization can form the basis for eventual use of this data for client-side geometric operations and query.

Using the techniques described in this paper, we have implemented a web-based map viewer for the northeastern United States—from Virginia north to Maine. Our client is implemented as a Java applet and the server is implemented in Java, with access to a PostgreSQL relational database system. (This is similar to Pajarola et al,[20, 21] who use an object-oriented relational database.) The size of the database is approximately one gigabyte, including indexes. A demo is available at `http://cs.nyu.edu/visual/home/demos/tigerDemo`.

*Using the convention that "large scale" means fine detail, as in map scales.

In the rest of this paper we describe in more detail the problems we faced (and continue to face) in building the system; describe the architectural, data processing, and user interface solutions we have developed; and conclude with some plans for the direction of the research in the near future.

2. PROBLEM DEFINITION

Our aim, in high-level description, is to build a web-based map viewer, for at least the contiguous mainland United States, with smoothly continuous zooming and panning. This section describes in more detail the various issues we are confronting in order to achieve this aim.

2.1. Display infrastructure

We use the client-server model, with the server holding one or more datasets that a client wishes to visualize via a thinwire. Several clients can access the server at the same time. This thinwire model can be refined as follows: (A) **Ultimate Thinwire Model**: We assume that the server and the client have unlimited computing power. This means that both parties are willing to spend as much preprocessing and high speed computation necessary to minimize the actual data transmitted. This model is justified in situation where we can afford to buy supercomputers at both ends of a connection but have no ability to affect the bandwidth. (B) **Thin Client Model**: This assumes that the client has moderate computing power and memory. The server is willing to absorb most of the computational cost to reduce load on the client as well as the thinwire. (C) **Moderate Server, Thin Client Model**: This is probably the most common realistic situation, and in any case, it is generally applicable.

We found the idealized model (A) useful when faced with difficult design decisions. When the choices are unclear, we often assume this model: it means that our only goal is to reduce bandwidth. Let us illustrate this in our implementation of visually continuous zoom and pan. Visual continuity is critical for a natural exploration of geospatial datasets, so we are committed to paying the necessary overhead. This feature calls for special computational structures: the client must have a dynamic data structure D to build up partial views of the underlying data (at each level of detail). When the user pans the viewer, this amounts to a request for new visual data, but the data may already be in D. For raster images we can search D to see if the data is locally available,[12] but for vector data it is difficult to be sure that the data in D is complete. So we transmit a request to the server saying that data in a given rectangular range is needed. The server will then filter the response to *only* send data that has not been sent before. To achieve this effect, the server must maintain D', a mirror copy of D. Can we justify this duplication of D? Well, under model (A), this overhead is negligible compared to the guaranteed non-redundancy in transmitted data.

2.2. Dataset

Visualization techniques are highly dependent on the nature of the datasets. Our target is vector data such as found in maps and geographical information systems (GIS). More specifically, we address the TIGER (Topologically Integrated Geographic Encoding and Referencing) system data that is freely available from the US Census Bureau.[22] The simplification of vector datasets in 3-D models (e.g., from CAD modelers) has been extensively studied in the recent literature.[23–25] The same problem in GIS is closely related to the generalization problem,[26] but this is much less well-understood. The apparent paradox of a 2-D simplification problem being less tractable than the same 3-D simplification problem comes from the fact that map data has symbolic and semantic content that goes beyond pure geometry. In section 4, we will describe a simple and effective method for automatic generation of LOD hierarchies that takes into account the special nature of the TIGER data.

There are many visualization interfaces for the TIGER dataset. Most commercial products such as ARC/INFO accept TIGER data. There is a free TIGER Map Server (TMS)[27] from the US Census Bureau. Superficially, these interfaces, as well as the many commercial map servers, have capabilities similar to ours. Besides our continuous zoom and pan, a critical difference is that we transmit vector data as opposed to raster data. With vector data, arbitrary zoom-in is meaningful, while only limited zoom-in makes sense for raster data. More importantly, we can construct geometric structures (e.g., for point searching) and perform operations (like map overlay) locally on the transmitted vector data. Our current system is purely a visualization system, but it can form the basis for performing geometric queries in the future.

2.3. Responsiveness

We have characterized "responsiveness" as the ability of a system to respond to user commands as naturally and quickly as possible, and to adjust its responses to the prevailing bandwidth. What does this imply for system design? For implementation purposes, we can consider two (more or less) independent aspects: responsiveness of the data stream, and responsiveness of the visual experience. (Of course, if the data stream is very slow, the visual experience will suffer, but at the implementation level we can provide for responsiveness in the visual experience without any assumptions about the quality of the data stream.)

Concerning the data stream, the first step must be to reduce the amount of data that needs to be sent. Section 4 describes our work in unifying the data and building a LOD hierarchy. But this is not enough. There will still be a lot of data to send, and we have to account for the fact that users make frequent mind changes, and create new requests inspired by only partial responses to old requests. So our server must be able to reorder requests, or even cancel old ones that have not been filled yet. In section 3 we describe how our multi-threaded architecture provides this capability.

To provide for responsiveness of the visual experience, the system should, in response to a user request, at the least, (1) provide *immediate* feedback that the request was received, and (2) give a partial answer to the request using any data that has already been downloaded, without waiting for the complete answer from the server. The system architecture description in section 3 will explain how we achieve this in our system.

Indirectly related to the issue of responsiveness of the visual experience is the question of how to design a user interface that will keep as much relevant information (R) as possible visible as context for the current data exploration, as well as give the user an intuitive and convenient process for recognizing the context and specifying new exploratory requests. In section 5 we describe some new user interface approaches for accomplishing these goals.

3. RUNTIME SYSTEM ARCHITECTURE

A preliminary stab at a multi-threaded architecture for thinwire visualization was made in the work on multi-foveated images.[12, 28] The system architecture described here is an extensive expansion and refinement of that earlier work.

Figure 1 diagrams our runtime system architecture. Arrows indicate the flow direction of data and commands. The client diagram, on the right side of the figure, attempts to show the relationship between our system and the native platform that the client runs on. The GUI Display Queue and the GUI Event Handler Thread are part of the Java Runtime Environment (jre), but the event handler thread executes some code that we control, which is why it is shown outside the Native System box. In the case of the event handler thread, the positioning of the arrows is meant to suggest the flow of data and commands. There are two arrows from the event handler thread to the window: the top one represents actions that result directly from mouse events, and the bottom one represents the flow of data and commands arriving from the Display Manager Thread. The queues are oriented in line with the arrows, showing first-in first-out semantics, and the lists are oriented perpendicular to the arrows, representing some other ordering (sometimes stack, sometimes queue but with the possibility of having items deleted, sometimes more complicated structures).

Several principles have guided the design of the system architecture:

1. Mouse actions by the user should be responded to immediately, even if the response is incomplete.
2. Network bandwidth is limited.
3. The server should be flexible in allowing users to determine which data is downloaded, and in allowing users to change their minds.
4. Computational power on the server and client is large relative to the network bandwidth.
5. On the server, the data may be stored on disk, and the amount of disk space available is unlimited.

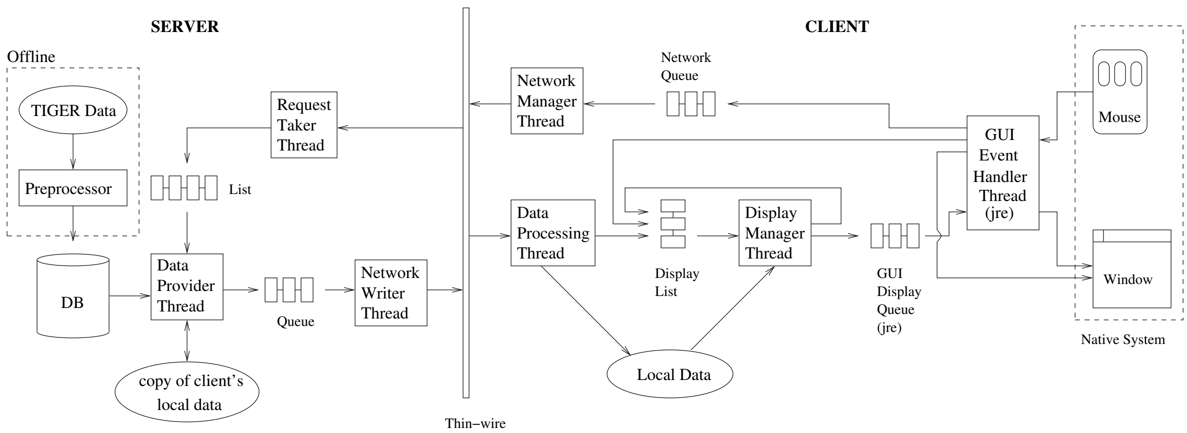

Figure 1: Server and Client Design

6. Storage on the client must be in memory, and the amount of memory available is moderate - i.e., enough to store a cache of "recently" downloaded data, but not enough to store all the data.

First of all, notice that no thread on either the client or the server blocks on more than one I/O device. (The queues and lists are in memory. They do provide for synchronized access for multiple threads, so a thread may block on a queue or list, but the critical sections there are quite small and fast. A queue/list reader may wait for a long time if the queue/list is empty, but the system is designed so that the threads get their tasks from the queues/lists, so if a thread is blocked on reading a queue/list, that thread has nothing else to do anyway.)

To understand this abstraction, it helps to consider the sequence of events resulting from a single mouse action. Suppose the user drags the mouse across the screen, which in our current system moves the underlying map in that direction (i.e., moves the view window in the opposite direction). The native system will pass that event to the event handler thread. The event handler thread then does three things with it, as represented by the three arrows coming out from its top left. First, as represented by the bottom of the three arrows, it instructs the native gui system to pan the current view image, opening up some blank areas on the screen. This gives the user confirmation that the request was recognized, and should happen almost immediately, since no other threads are involved. Next, as represented by the two upper arrows, it places information about the request (for objects in a particular view area) onto the Network Queue and the Display List. These two data paths then complete concurrently.

Probably the path through the Display List will complete first, since it does not go through the thin-wire. The Display Manager thread pulls the request from the Display List, searches the local data for geographic objects in the requested area, and *puts those objects back onto the Display List.* Then the Display Manager will individually pull each object off the Display List and send it to the GUI Display Queue, and from there it will eventually be drawn to the screen.

On the third data path, the Network Manager Thread sends the request across the thinwire to the server, where the Request Taker Thread reads it and places it on the list for the Data Provider Thread. The Data Provider Thread pulls the requested geographic objects out of the database, consults its copy of the client's local data to filter out any objects that the client already has, then passes the remaining objects to the Network Writer Thread, which sends them to the client. Back at the client, the Data Processing Thread reads the objects from the network, stores them in the local data, and puts them on the Display List, from which they will then be passed by the Display Manager Thread to the GUI system for display on the screen.

Much of the complexity of this architecture is geared toward providing responsiveness. Notice that the Event Handler Thread is something of a bottleneck, since it handles both mouse input and window output. Furthermore, this thread is perhaps the most important one for responsiveness, since it is the one that will give the user the initial confirmation that a request has been received. This thread will give preference to handling mouse events over tasks from the GUI Display Queue of the jre, but it won't be interrupted if it is in the middle of handling a task. Therefore, we want all tasks that the event handler thread performs to complete as quickly as possible. This principle, along with the related goal of not having the Event Handler Thread do any unnecessary work, guides the following design choices:

1. *When the user makes a range request (show all objects in a given range), the Event Handler Thread passes this request off to the Display Manager Thread (through the Display List), instead of pulling the objects out of the local data store and displaying them directly.*

   Placing the range request onto the Display List is a very quick operation, essentially atomic (though it does require thread synchronization). On the other hand, pulling a (possibly large) number of objects out of the local data store is a fairly involved operation, and displaying them involves I/O for the window output. So having the Event Handler do that could leave a new mouse event waiting for the completion of many I/O operations before it can be handled—clearly not a good situation for responsiveness.

2. *For sending requests to the server, the Event Handler passes the requests off to the Network Manager Thread (through the Network Queue), instead of just writing the request directly to the network.*

   Clearly our design delays the sending of the request to the server by some small amount. But the more important question is how does the design choice affect the ability of the Event Handler to respond to the user? Adding an element to the Network Queue does require some overhead for thread synchronization (the Network Queue is shared by two threads). Writing an instruction to the network also requires significant overhead, because the communication is by the TCP protocol, which means that some handshaking has to happen between the client and server to verify that the message is sent properly. Since we are assuming the thin-wire is a slow connection, we have assumed that the network writing operation is significantly slower than the local thread synchronization operation. So for the benefit of the Event Handler, it is better to pass the request to the Network Manager thread.

3. *When the Display Manager Thread pulls objects out of the Local Data store, it puts them back onto its input list, instead of drawing them to the window immediately.*

   We can't remove objects from the GUI Display Queue once they are on there, but we can do so with the Display List. If a mouse action comes along that invalidates previous actions, such as a zoom or a jump to a new location, the Event Handler clears out old instructions from the Display List. So, by putting individual objects onto the Display List we reduce the number of objects the Event Handler has to draw unnecessarily.

4. *When the Data Processing Thread receives objects from the server, it passes them to the Display Manager, instead of putting them directly on the GUI Display Queue.*

   The reason is just the same as for the previous choice: allow a new mouse event to clear out objects from the list that are no longer necessary.

4. PREPROCESSOR

For the system to have good performance, the data must be stored in such a way as to facilitate speedy queries. The raw data that we get from the US Census Bureau is not in such a format, so it must be preprocessed. A detailed description of the format of the TIGER[22] data can be found in the technical documentation.[29] Here we give the broad outlines of the preprocessing process, much of which should apply to other sources of data as well.

The preprocessor has two distinct modes of operation. In the first mode, the raw TIGER data is converted to a more convenient and space efficient internal format and stored in the database in (mostly) full detail. (The raw data contains some features that we do not currently use in our system, such as school district boundaries, but the features that we do use are stored in full detail.) The second mode, which is executed multiple times, takes a description of the data at a particular level of detail and creates a more generalized and simplified version for the next level of detail. In the final database, then, each level of detail must be queried independently; it is the runtime system's responsibility to know which level of detail needs to be queried for the current view area.

4.1. Converting the TIGER files

The TIGER data comes packaged in files, where each file contains the data for one county (or similar entity, such as parish or unincorporated area). Not all counties are connected, and the counties that are connected might not be simply connected. For example, New York county (Manhattan) includes the Statue of Liberty, which is surrounded by water in Hudson County, New Jersey. So New York county is not connected, and Hudson County is not simply connected. Within each county, the key geometric object in the TIGER system is the TIGER line: this is really a polygonal line whose non-endpoint vertices are guaranteed to have degree 2. The set of TIGER lines forms a subdivision of the plane and the regions of this subdivision are called TIGER polygons.

The data for each county is complete, so the lines representing borders between counties are duplicated. There is also unnecessary (for our purposes) duplication of data within each county: when two TIGER lines meet at an endpoint, that endpoint is duplicated. If the two lines represent two sections of the same feature, we can combine them into one polygonal line and eliminate the duplication of the endpoint. Polygons can similarly be combined.

In the first mode, the main issues to deal with are the elimination of data duplication and the coordination of cross-county data into one seamless database. The polygon containment hierarchy, which is necessary for display at runtime, must be consistent across counties. All of this must must also take into account that in general no more than one county may fit into main memory at a time, so the protocol has to try to minimize disc accesses. We have developed a protocol to accomplish this, but the details are beyond the scope of this paper.

4.2. Building the LOD hierarchy

In the second mode we build the level of detail (LOD) hierarchy. As mentioned above, the generalization problem for geographic data (maps) is not well understood. Nonetheless, we have a mostly automatic approach that gives adequate results for our purposes. The first step is to partition the counties (or groups at the lower level) into groups for processing. Since we won't be using the full detail, we can fit the data from more than one county into memory. Plus, we want to extend the line and polygon combining step into cross-county potential, to find more opportunities for duplication elimination and simplification. So we are confronted with a problem that has no known solution (other than exhaustive search): partition the counties into a minimum number of groups such that the groups are connected, have a simple shape, and have no more than a given amount of data, which we can specify by the number of points at full detail. Our solution has been to use a simple heuristic that guarantees the groups are nicely shaped and obey the size limit, but does not guarantee connectivity, and then we modify the grouping by hand to get connectivity.

The heuristic is this: repeatedly divide the counties along either the vertical or horizontal axis, using the center point of each county's range. Specifically, pick the axis along which the dataset is "longer", sort the counties along that axis by their range centers, and split the set into two roughly equal sized pieces, where size is number of points at level 0. Recurse until the piece is less than the maximum number of points. When computing the higher levels of detail (higher levels contain less detail), use the groups at the next lower level as the "counties".

We must also choose the resolution, or scale, associated with each level of detail. We use the pixel size, not the inch or centimeter, as the reference unit of length. This is justified by the fact that our target is web visualization, where the applet may display on screens with different resolution. We define **pixel level of detail** (pLOD) to be the number of meters per pixel. In our current dataset we use 6 specially hand-chosen pLOD's, $0, 14, 75, 200, 700$, and 2500 as the breaking points for levels of detail. Relative to this sequence pLOD's, we define 5 "simplification levels", or **S-Level** for short: S-Level 0 corresponds to pLOD's $[0, 14)$, S-Level 1 corresponds to pLOD's $[14, 75)$, and S-Level 5 corresponds to pLOD's $[2500, \infty)$.

The next step is the most difficult to automate: choosing which geographic features to include at each level. After some early experimentation with developing tools that aid in hand selecting features, we decided that for this research project we would sacrifice some map quality and use a mostly automatic approach. The only part done by hand is to specify the classes of features that will be included at each level. For example, level 0 has the full detail, level 1 contains highways and primary and secondary connecting roads but no local streets, level 2 contains only highways and no other roads, etc. These feature class distinctions are specified in the TIGER data, so once these few simple decisions have been made, the preprocessor can run without human intervention. With unlimited resources and time, we would of course prefer to have more flexibility in choosing geographic objects, but the generalization that we get is good enough to demonstrate the feasibility of the responsive, thinwire visualization that is our primary objective.

Once we have determined the groups and the set of feature classes to include, the process for generating level of detail i is similar to the process for level 0. The differences are

1. The data comes from level $i-1$ in the database, instead of from the raw TIGER files.
2. Any polygon that is smaller than twenty square pixels at the resolution associated with this level is dropped, and its area assigned to a neighboring polygon (preferably one in the same feature class).
3. Lines are simplified with the Douglas-Peucker algorithm[30] before being stored in the database. The error bound to use in the algorithm is equivalent to two pixels at the associated resolution.

4.3. Manual generalization

As mentioned above, we are not currently doing any manual tuning of the map generalization process because of lack of time and resources. However, we have developed a tool, pictured in figure 2, for manually choosing features to include in a level of detail. The checkboxes at the right are for selecting/deselecting classes of features. Individual features can be selected with mouse clicks. The user can also simplify the lines, using the Douglas-Peucker[30] algorithm, and filter polygons by area.

5. USER INTERFACE

5.1. Zooming telescope

A natural starting point for designing a user interface for navigating geographic data (maps) is the interface used by MapQuest[31] and the TIGER Map Server (TMS).[27] This is a web page with a map image, of size approximately 400 × 350 pixels for TMS and 350 × 250 for MapQuest, along with links for zooming in and out, re-centering, etc. An improvement over this is the interface used by Chang and Yap[12, 13] for foveated visualization of large images over a thinwire. This interface adds a small iconic image of the entire underlying image, with a black rectangle indicating the location of the current view window, to give users a sense of context. With the large dataset and geographical area that this project works with, this interface quickly became insufficient, because when zooming in to the highest detail the rectangle on the iconic image shrinks to less than one pixel. Therefore, we have generalized the interface to our **zooming telescope**, in which the user can open any number of **telewindows**, and each telewindow shows a closer view of a portion of the previous telewindow (figure 3). Each telewindow can be independently zoomed, panned, resized, and moved around the screen. Clicking on a telewindow causes the next telewindow in the sequence to jump to the clicked location; each telewindow has a "center" button, which re-centers the previous telewindow on the current window's center point. This setup gives the users great control over how they want to visualize the data, and gives plenty of contextual information which makes browsing the geography much more visually satisfying than the standard MapQuest style interface—on top of the advantages of smooth, continuous panning and zooming.

We describe the zooming telescope more formally as follows. The telescope has a formal structure comprising a sequence $W_0, W_1, \ldots, W_h$ ($h \geq 0$) of windows (called **telewindows**). The only formal constraint is that each W_i has a **scale** $\sigma_i \geq 0$ that is higher than that of W_{i-1}. The **length** h of the telescope can be extended by adding a new telewindow, or contracted if $h > 0$. Each W_i displays a **rectangular range** ρ_i (centered at its **position** p_i) of the underlying image at its current scale. The range of W_i in W_{i-1} is indicated by (parts of)

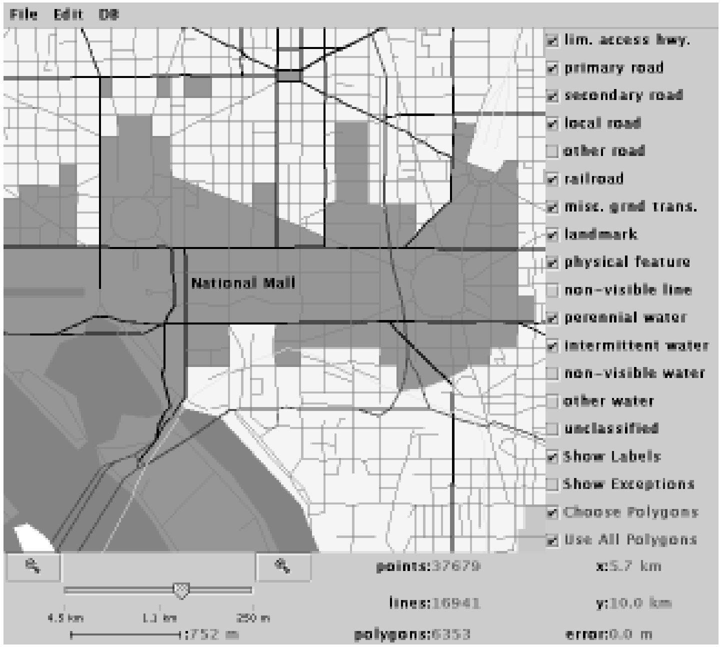

Figure 2. Screenshot of data selection tool

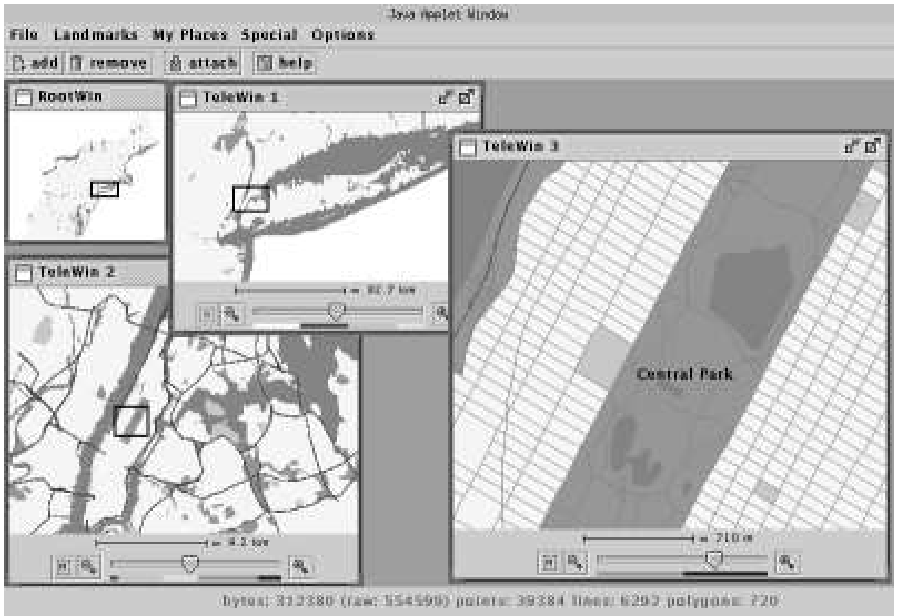

Figure 3. Screenshot of Zooming Telescope interface

a rectangular outline when the range ρ_i intersects the range ρ_{i-1}. We call this outline the **ghost** of W_i. It is possible for W_i to lie outside the range of W_{i-1}. We provide the following operations for each window W_i (with some limitations for $i = 0$):

Pan: This is to a continuous change in p_i, achieved by holding the mouse button while moving the mouse in W_i.

Zoom: This changes σ_i using one of two modes: (a) The continuous mode is achieved by moving a slider along a scale. (b) The discrete, relative mode is achieved by clicking one of two buttons (zoom-in or zoom-out), and σ_i changes by a fixed fraction.

Jump: This corresponds to a discontinuous change in p_i, and is achieved by clicking at any point in W_{i-1}.

Center: This is an "inverse jump" which corresponds to setting p_{i-1} to equal p_i. In effect, it re-positions W_{i-1} so that the ghost of W_i appears at the center of W_{i-1}. This is extremely useful in ensuring that a user is never lost in a large image.

Resize: This corresponds to modifying ρ_i, and uses standard window resizing interface.

The idea of a providing a larger scale image of a portion of another image is standard. It is also reminiscent of concept such as portals and lenses in `pad++`[32, 33] and in other systems. For instance, lenses are used to transform views of a piece of data (a table of numbers might turn into a graph plot of these numbers under a plotting lens). But in contrast to these constructs, our zooming telescope is designed to scale with the size of the underlying image. To illustrate the scalability issue, our original implementation, with $h = 1$, was adequate when the dataset had 5 counties (the 5 boroughs of New York City), but its inadequacy became clear after we expanded our dataset to 20 counties in the NYC metropolitan area. Our current dataset covers 13 states and the District of Columbia, an area that is roughly 624,000 square kilometers (241,000 square miles). To zoom from an overall image of the map down to street level (about 2.5 meters per pixel), the scale changes by a factor of about 1000. This three orders of magnitude change is more than the usual large scale visualization studies. We can comfortably explore this map with a telescope length of $h = 3$.

5.2. Pop-up labels

One of the more difficult problems in automatic map generation is label placement. We have so far avoided this problem altogether by using pop-up labels: when the mouse pauses near a feature (road, park, etc.), a label describing that feature pops up over the map. Currently, we only show one label at a time, so any previous label disappears when the new one pops up. Although our original motivation for showing labels this way was merely to avoid dealing with the label placement problem, we have found that this method has some real advantages. Primarily, not having static labels allows more features to be put onto a screen without making the screen too crowded. In other words, for the same screen size, resolution, and level of detail, we can show a larger geographic area than could be shown on a map with static labels. This gives the user a better sense of context for the area they are viewing. However, the lack of static labels can also be a source of confusion and *lack* of context. Therefore, in the future we hope to incorporate a *limited* number of static labels along with the pop-up labels, to get the best of both worlds.

6. CONCLUSIONS AND CHALLENGES

We have posed responsive visualization as the appropriate goal for thinwire visualization of large datasets. Our current architecture demonstrates some of principles of a responsive system. This is demonstrated in the context of (a) a natural and scalable visualization interface (the zooming telescope), and (b) the visualization of large vector datasets (TIGER). This is a work in progress, but we believe we now understand the basic issues remaining in our path toward an applet-based visualization of the entire TIGER dataset.

Perhaps the most important next research step for responsive visualization is a framework for (i) instrumenting, (ii) measuring, (iii) evaluating and (iv) tuning such systems, in this order. To achieve instrumentation, we must design the system to depend on a set of numerical parameters (for instance, the maximum range in any request for data). Then we need the "hooks" for measuring the performance of the system (the simplest measurement here is the running bandwidth estimates). Next, we need a theory for evaluating this performance. Tuning the system parameters follows naturally when we understand its performance.

Beyond instrumentation, this area also needs (a) a methodology for specifying responsive behaviors, (b) a model for constructing responsive visualization programs, and (c) systems for automatic checking of models against behaviors. The literature on model checking and realtime systems serves as the guide to our research agenda. Model checking addresses non-terminating programs and their behaviors, using canonical examples such as operating systems, pipeline processor architectures, etc. It is seldom noted that a prime example of such behavior in computing today is GUI systems. What we propose is a formalism for specifying and checking GUI systems, specifically those for visualization. Thus, we may specify "when the user pans, some response is expected within X milliseconds". Any implementation (i.e., model) can then be checked against these specifications.

We list several issues that we plan to address in the near future:

- We want to incorporate a limited number of static feature labels (street names, etc.) along with our dynamic pop-up labels.
- We have avoided the use of multi-foveated images[12,13] by the introduction of the zooming telescope. However, it should not be difficult to incorporate a simple form of foveation similar to the 2-level images used in Reeves and Robinson.[34] This could improve the responsiveness of our system.
- We may generalize our zooming telescope interface to allow a rooted tree of telewindows, and pursue some HCI issues within the framework of the interface's potential applications in GIS.
- We are currently only doing visualization, but we can view this visualization interface as a means for performing GIS-type analysis or planning. The fact that our server transmits vector data will support this extension. The vector data can be reconstituted for geometric operations and querying.
- To improve the quality of our maps, we may look for efficient ways to incorporate manual generalization into the process for computing LODs.

REFERENCES

1. J. Helman, "Graphics techniques for walkthrough applications," in *Interactive Walkthrough of Large Geometric Databases, Course Notes 32, SIGGRAPH '95*, pp. B1–B25, 1995.
2. J. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM* **19**(10), pp. 547–554, 1976.
3. P. S. Heckbert and M. Garland, "Multiresolution modeling for fast rendering," in *Graphics Interface '94*, W. David and B. Joe, eds., pp. 43–50, 1994.
4. S. J. Teller and C. H. Séquin, "Visibility preprocessing for interactive walkthroughs," *Computer Graphics* **25**(4), pp. 61–69, 1991. Proceedings of SIGGRAPH'91.
5. Y.-J. Chiang and C. Silva, "I/O optimal isosurface extraction," in *Proc. IEEE Visualization '97 (Vis'97)*, pp. 293–300, 1997.
6. J. El-Sana and A. Varshney, "View-dependent topology simplification," in *Virtual Environments '99*, M. Gervautz, A. Hildebrand, D. Schmalstieg, and E. Wenger, eds., Spring-Verlag, 1999. To Appear.
7. K. F. Aoki and D. T. Lee, "Towards web-based computing," *Int'l. J. Computational Geometry and Applications* **11**(1), 2001.
8. S. Parker, D. Weinstein, and C. Johnson, "The SCIRun computational steering software system," in *Modern Software Tools in Scientific Computing*, A. M. B. E. Arge and H. P. Langtangen, eds., pp. 1–40, Birkhauser Press, 1997.
9. G. Fox and W. Furmanski, "Petaops and exaops: Supercomputing on the web," *IEEE Internet Computing* **1**(1), pp. 38–46, 1997.

10. J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing* **41**, Dec. 1993.
11. A. Said and W. A. Pearlman, "A new fast and efficient image codec on set partitioning in hierarchical trees," *IEEE Trans on Circuits Syst. Video Tech.* **6**, June 1996.
12. E. Chang, C. Yap, and T.-J. Yen, "Realtime visualization of large images over a thinwire," in *IEEE Visualization '97 (Late Breaking Hot Topics)*, pp. 45–48, 1997. See also CD proceedings of conference.
13. E.-C. Chang and C. K. Yap, "A wavelet approach to foveating images," *ACM Symp. on Computational Geometry* **13**, pp. 397–399, 1997.
14. E.-C. Chang, S. Mallat, and C. K. Yap, "Wavelet-based foveation," *J. Applied and Computational Harmonic Analysis* **9**, Oct. 2000.
15. E.-C. Chang, *Foveation Techniques and Scheduling Issues in Thinwire Visualization*. PhD thesis, Department of Computer Science, New York University, May 1998.
16. E. Schwartz, D. Greve, and G. Bonmassar, "Space-variant active vision: Definition, overview and examples," *NeurNet* **8**(7–8), pp. 1297–1308, 1995.
17. E.-C. Chang and C. Yap, "Competitive online scheduling with level of service," *J. of Scheduling* , Oct. 2000. To appear, Proc. 7th Ann. Intl. Computing and Combinatorics Conf. (COCOON), August 20-23, 2001, Guilin, China. To appear, Special Issue of J. of Scheduling on Online Algorithms.
18. D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Comm. of the ACM* **28**(2), pp. 202–208, 1985.
19. S. Albers, "Competitive online algorithms," BRICS Lecture Series LS-96-2, BRICS, Department of Computer Science, University of Aarhus, September 1996.
20. R. Pajarola, T. Ohler, P. Stucki, K. Szabo, and P. Widmayer, "The Alps at your fingertips: Virtual reality and geoinformation system," *Proc. 14th Int. Conf. on Data Engineering, ICDE'98* , 1998.
21. R. Pajarola and P. Widmayer, "Virtual geoexploration: Concepts and design choices," *Int'l. J. Computational Geometry and Applications* **11**(1), 2001.
22. Census 2000 TIGER/Line Files [machine-readable data files], U.S. Census Bureau Washington, DC, 2000, http://www.census.gov.
23. J. Cohen, D. Manocha, A. Varshney, and G. Turk, "Efficient model simplification with global error bounds," in *Fifth MSI-Stony Brook Workshop on Computational Geometry*, pp. 43 – 44, October 1995.
24. P. K. Agarwal and P. K. Desikan, "An approximation algorithm for terrain simplification," in *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pp. 139–147, 1997.
25. M. de Berg, M. van Kreveld, and S. Schirra, "A new approach to subdivision simplification," in *Proc. 12th Internat. Sympos. Comput.-Assist. Cartog.*, pp. 79–88, 1995.
26. J.-C. Muller, "Generalization of spatial databases," in *Geographical Information Systems: Principles and Applications*, D. J. Maguire, M. F. Goodchild, and D. W. Rhind, eds., Longman Scientific & Technical, Essex, England, 1991.
27. "Tiger map server," 1995. Demo from the US Census Bureau. URL http://www.census.gov/ftp/pub/geo/www/tiger/tigermap.html.
28. C. Yap and T.-J. Yen, "Design and instrumentation of a thinwire visualization system," 2000. Manuscript. Available from `ftp://cs.nyu.edu/pub/local/yap/visual/instrument.ps.gz`.
29. U.S. Census Bureau, Washington, DC, *Census 2000 TIGER/Line Files Technical Documentation*, 2000. http://www.census.gov.
30. D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Canadian Cartographer* **10**(2), pp. 112–122, 1973.
31. "Mapquest map server." http://www.mapquest.com.
32. B. Bederson, L. Stead, and J. Hollan, "Advances in the pad++ zoomable graphics widget," in *Proc. USENIX Tcl/Tk'95 Workshop*, 1995.
33. "Pad++ homepage." URL `http://www.cs.unm.edu/pad++/`, 1996. Online guide and papers.
34. T.H.Reeves and J.A.Robinson, "Adaptive foveation of mpeg video," *Proceedings, 4th ACM International Multimedia Conference* , 1996.