Dynamic Query Visualizations on World Wide Web Clients: A DHTML Solution for Maps and Scattergrams

Evan Golub
egolub@acm.org

Ben Shneiderman
ben@cs.umd.edu

Department of Computer Science &
Human-Computer Interaction Lab, Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

Dynamic queries are gaining popularity as a method for interactive information visualization. Many implementations have been made on personal computers, and there is increasing interest in web-based designs. While Java and Flash strategies have been developed, we believe that a Dynamic HTML implementation could help promote more widespread use. We implemented double-box range sliders with choropleth maps and scattergrams, which are two popular visualizations, using HTML layers and tables. This paper describes our methods for slider control, visual presentation, and displaying/updating results for these visualizations. Visual design issues innovations and performance enhancements were necessary to create viable designs.

Keywords: choropleth maps, dynamic html, dynamic queries, scattergrams, sliders, information visualization

1. Introduction

The thirst for information has grown as Internet applications have become widely accessible. However techniques for exploring large data sets are still dependent on database query methods such as SQL (Structured Query Language) statements and form fill-in techniques for specifying attributes and attribute value ranges. Users type the query or select values and then execute the query against the database. They view results in scrolling lists of textual and numeric output. There are many benefits to this method, especially when known item searches are performed, such as "find the titles and of books whose author is 'Hemingway' and whose publication date is before 1943" or "find employees whose salary is above 100 and whose seniority is between 6 and 9."

However, in many situations users are probing the data to understand patterns, discover outliers, recognize clusters, and find gaps. In these situations the slow pace of query formulation, execution, and review of results (often taking more than a minute) inhibits exploration. In response to these challenges, some designers have developed interfaces, called dynamic queries, to accelerate query processes and provide users with visual presentations of large result sets [2, 6, 9, 13]. The simplified input, rapid updates, and visual display of results produces high user satisfaction and improved performance on benchmark tasks.

The key idea is to tightly couple input widgets such as radio buttons or sliders to a meaningful visual display of results, such as a map, scattergram, bar chart, or other visual display [3, 5, 11, 15]. As users click on radio buttons or move the slider thumbs, the output of results is rapidly updated (in under 100 milliseconds), enabling users to easily spot additions or deletions. For example, a user might move the employee salary slider to show a minimum of 100 and the seniority sliders to show a minimum of 6 and a maximum of 9. As the sliders are moved the

result set changes and can be viewed as a standard scrolling list or as markers on a scattergram that could have axes of salary and seniority [1]. Of course the axes could be age and job title, or other ordinal or nominal variables. The benefits of dynamic querying were demonstrated in two experiments on a chemical database and a real estate database [2, 17] (Figures 1 and 2). Dynamic queries produce faster user performance and high user satisfaction when compared with paper listings, form fill-in, and natural language querying.

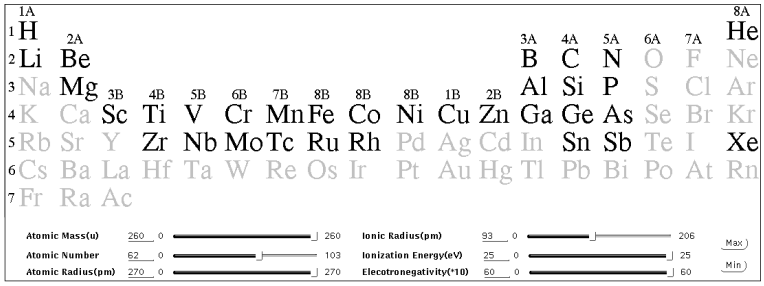

Figure 1. Dynamic Queries on Periodic Table of Elements – maximum values for Atomic Numbers and Ionic Radii have been set.

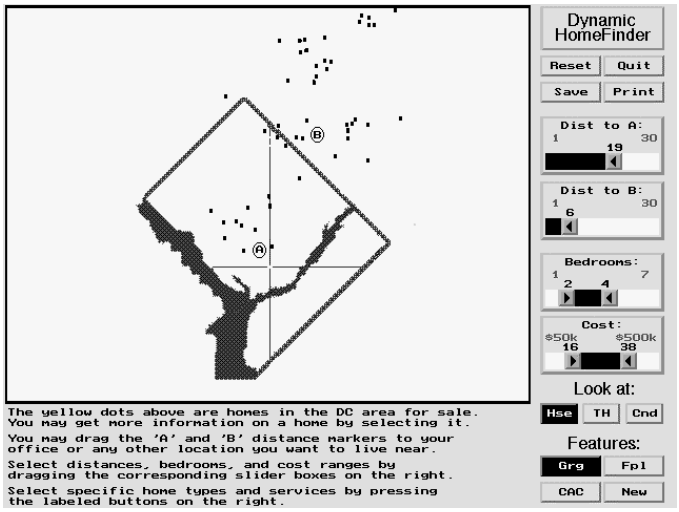

Figure 2. Dynamic Queries on a Real Estate Map looking for homes within 19 miles of Point A, within 6 miles of Point B, with 2-4 bedrooms, costing between $160K and $380K and including a garage.

Ensuring rapid updates from large databases was clearly a problem [10] so special data structures in the high-speed storage were necessary. As user demand grew, commercial versions provided rapid response to databases with as many as one million items [14] (Figure 3). Of course screen size limitations also constrain the practical size of databases that can be explored. For larger databases users typically extract a subset that is appropriate and manageable (tens of thousands of items). A second approach to coping with large databases is to apply an aggregation function as a first step. For example, web log data with tens of millions of entries could be aggregated by hour and URL into tens of thousand of summary items. Then users could select a narrow group of these summary items for further study of details [8, 12, 16].

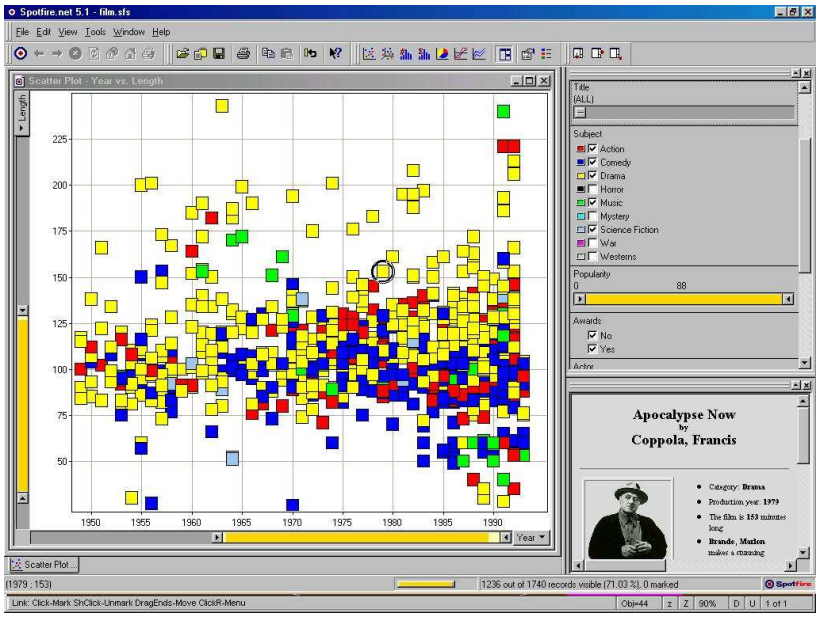

Figure 3. Using Spotfire to visualize a database of films, filtered by category using *Action, Comedy, Drama, Music and Science Fiction*, plotted by year and length of film with the movie "Apocalypse Now" currently selected for detail.

Our Dynamap project for the U. S. Census produced a Visual Basic program with a choropleth U. S. map at the state and county levels, a scattergram, and dynamic queries [5] (see Figures 4 and 5). This technology provides adequate performance when distributed as an executable program on a CD-ROM with data sets.

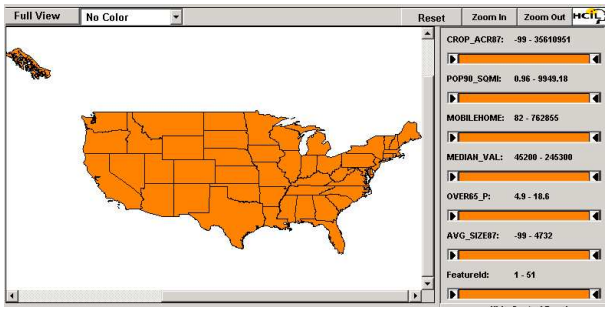

Figure 4. Dynamaps with the sliders set to initial positions, showing all states.

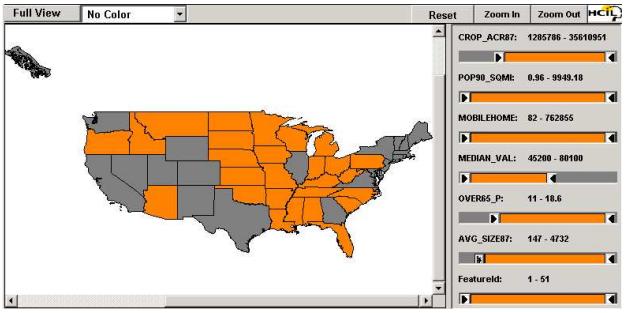

Figure 5. Dynamaps with sliders set with various restrictions, only showing states within those selected bounds (such as Texas).

The enormous popularity of the World Wide Web led some implementers to provide dynamic queries in web-based Java applets [11]. This is a natural evolution and responds to many needs of those who accept Java applets. Unfortunately, some organizations prohibit Java applets because of security concerns and other users have equipment that does not have Java support (or does not yet have free Java support).

As people continue to more regularly look to the World Wide Web as an information resource, enabling non-Java users to perform dynamic queries within web pages will become a useful and powerful ability. This paper addresses two issues involved in providing this ability: displaying results and manipulating sliders.

As with the previously discussed applications, our Web-based version of Dynamaps employs techniques of direct manipulation in conjunction with dynamic queries. In order to minimize response time and eliminate the effect of network congestion once the page is loaded, we assume that the data needed to perform queries on the page will be transmitted with the page. This raw data will be used to affect changes to the displayed image (the details of how are discussed below). Although this does place limits on the number of variables that can be manipulated on a page, there are ways to work within these limits. In practice users could be given a list of all available variables and then choose some subset (in our examples three variables are used per page) and be given a web page that allows for manipulation of those variables. Occasional server requests are acceptable to most users, especially if they can be given rapid query capabilities as they explore. A server-side solution, where each slider change made by the user would be sent back to the Web server, followed by an updated image being sent back to the browser, was considered. However, it was felt that this would cause the rate of the dynamic updating of the displayed results to rely too heavily on the combination of the user's connection to the Internet, the server's connection to the Internet and the server's processing power.

This paper demonstrates how Dynamic Hypertext Markup Language (DHMTL) features can support dynamic queries within web pages. Since different browsers support slightly different web page scripting languages (JavaScript –vs- Jscript) and rules for style sheets (several different levels of CSS support), the code examples and web pages presented were designed to work in Internet Explorer 5.0 or greater. The strategies will work in any browser that supports DHTML [4, 6]. As the DHTML standards propagate, these strategies should work in browsers on both traditional computers (desktops and laptops) as well as devices such as PocketPCs, Palm Pilots and a variety of Internet appliances.

2. Displaying Results: Constructing an Image From Layers

To perform dynamic queries, the image in which the results are displayed needs to show results quickly and meaningfully. Since our goal is to eliminate the effect of network congestion, all image data is transmitted with the web page. Instead of the costly approach of transmitting all possible result images, in our solution we transmit only a small number of images and then locally generate any one of the numerous potential results. We accomplish this by using images with transparent backgrounds and with HTML tables. Both strategies use DHTML layers and the ability to set the visibility of an individual layer. Code Example 1 shows how a layer can be created at a specific location on-screen and have its initial visibility set. By using absolute positioning, all of the layers can lie on top of one another. For this to work, each layer must be given a different **id** value.

```
<div id="MAP.0"
     style="position: absolute; left: 1px; top: 1px; visibility=hidden">
<img src=images/ak.gif border=0>
</div>
```

Code Example 1. Placing an image inside a layer and settings its visibility.

These layers' visibility can then be turned on and off based on selected upper and lower bounds and the raw data that was included with the page. Code Example 2 shows a code segment for how the visibility property can be modified. In this example, we assume a boolean array hide[] has been previously assigned values that reflect whether each layer should currently be shown. This code would be placed in a function and would presumably be called on events such as moving a slider or making a selection of some type.

```
<script>
<!--
…
for (i=0; i<max; i++)
  if (hide[i])
     document.all["MAP."+i].style.visibility
                 ='hidden';
   else
     document.all["MAP."+i].style.visibility
                 ='visible';
…
//-->
</script>
```

Code Example 2. Using code within a loop to update all the layers' visibility programmatically.

Our first example, dynamic queries on a map of the United States of America, has 52 images layered one on top of another. Each of these images would have the same dimensions. In fact, the later images are created as masks from the first image. The first image is the entire country with all states colored in the inactive color. The remaining 51 images are the individual states and the District of Columbia as masks colored in the active color. The remainder of the image area is set to be a transparent background. The net effect is that by dynamically setting each layer's visibility, we can give the appearance of states changing colors. Once this action is tied into double-box sliders on the page and the associated data, users have the ability to perform queries and change the map display as they need. Figure 6 shows the first (base) image and Figures 7 shows one of the 51 mask images.

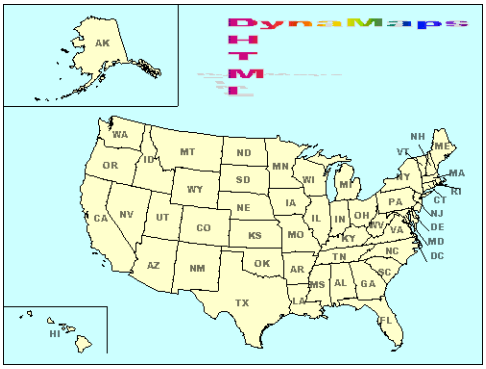

Figure 6. Base image of the United States in the inactive color.

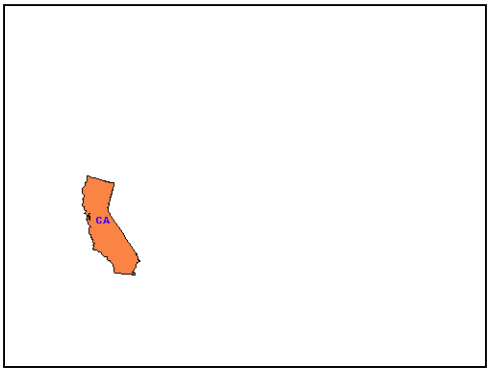

Figure 7. Mask of California in the active color.

Figure 12 shows an example of a map built using this technique being manipulated by sliders.

Our second example, dynamic queries on a scattergram, avoids transmitting images entirely by creating the images for the scattergram with HTML in the client browser. Each layer has a square drawn by a <table> with a single cell with a background color. Unlike the previous example where each layer was positioned directly on top of the other sharing a common upper left-hand coordinate, in this example the upper left-hand coordinate of the layer was positioned according the (X, Y) coordinate for that specific point within the scattergram. Code Example 3 shows a data point being created and positioned.

```
<div id="SCATTER.0" style="position: absolute;
 left:55px; top: 254px; visibility=visible">
  <table border=1 bgcolor=004080>
    <tr height=17><td width=17>
    </td></tr>
  </table>
</div>
```

Code Example 3. A single data point on a scattergram built using an HTML table.

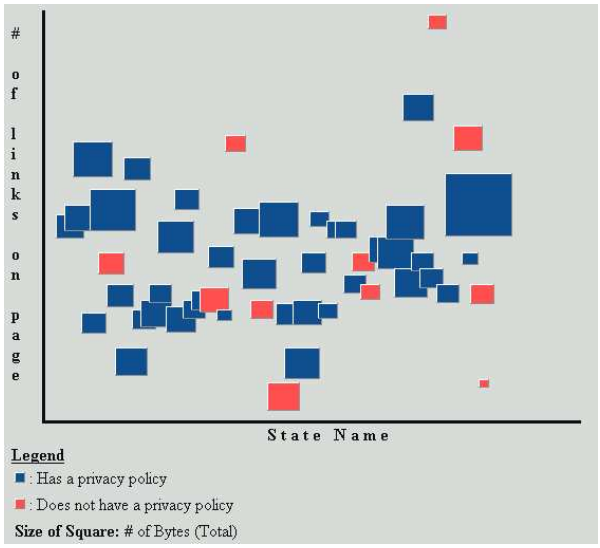

Figure 8. Example scattergram of data

This creates the appearance of the scattergram without requiring any images to be created or downloaded (see Figure 8). Since each point lives within its own layer, the visibility property of the layers creates the desired image as users perform their queries. Web pages with 5000 layers functioned smoothly, but at larger values (e.g.: 10,000-20,000 layers) Internet Explorer warns that a script is causing it to run slowly. Figure 13 shows an example of a scattergram built using this technique being manipulated by sliders.

3. Manipulating Sliders

Dynamic queries are generated with each move of the thumbs on a double-box slider. The two design questions are: how to control sliders and how to visually present them to enhance comprehensibility for novices.

3.1 Slider Control

We pursued two methods for slider control. The first was allowing the left mouse button to control the left thumb and the right mouse button to control the right thumb. By clicking at the point to which users want the thumb moved, the slider is visually updated and an update function is called to change the map or scattergram's appearance. The second method implements the sliding action as it was in the Visual Basic and Java versions of Dynamaps.

Web browsers support some dynamic elements, using HTML and DHTML. One method uses a table with each cell representing each possible stopping point for the thumbs. On events such as onMouseDown and onMouseMove the contents of the table cells are changed using scripting commands. The thumbs start in the two end cells of the table. Users can click and drag the thumbs towards the center.

The second method also uses a table to display the state of the slider, but adds two layers on top of it. Each layer contains one of the thumbs and can be dragged by the user. As the layer is dragged, the table is updated so that the colors within the table represent the new state of the slider.

3.2 Slider Visual Presentation

The sliders are constructed using tables, but there are several possibilities for the visual presentation. The first possibility was to place a GIF image within each cell and use different colors to show the selected range. The cells that held the thumbs had images of arrows to signal to users that these were the things that could be moved. While this technique gave us total control over the appearance of the slider (Figure 9), it's dependency upon images caused it to refresh slowly as users changed positions of the thumbs.

Figure 9. Slider constructed in HTML using images.

A similar visual presentation can be created without GIF images by utilizing the ability to change the background color of the individual cells of the table (Figure 10). Code Example 4 shows how we programmatically build such a table. There are a variety of constants we can set to adjust the appearance of the slider such as the desired table width (TTW) and the number of cells to use (NUMCELLS) which equates to the precision of the slider.

Figure 10. Slider constructed in HTML by coloring cells of table.

Larger thumbs, in the form of a simple black bar, in two layers on top of the table made for a more visually comprehensible design (Figure 11). Code Example 5 shows the code that would be added to create these.

Figure 11. Slider constructed in HTML by coloring cells of a table, with thumbs added on top inside two separate layers using CSS.

```
<script>
<!--
  document.write("<table width="+(Math.floor(TTW/NUMCELLS)*(NUMCELLS+1)) +
                 " cellspacing=0 cellpadding=0 border=4><tr><td>");
  document.write("<table width="+(Math.floor(TTW/NUMCELLS)*(NUMCELLS+1)) +
                 " cellspacing=0 cellpadding=0 border=0>");
//-->
</script>
<tr height=10>
  <script>
  <!--
  for (i=0; i<=NUMCELLS; i++) {
     document.write("<td width="+Math.floor(TTW/NUMCELLS)+" id='TickMark0-"+i+"'
bgColor=F88440>"+
                    "<a onClick='return moveSlider(0,"+i+",0)' "+
                    "onContextMenu='return moveSlider(0,"+i+",1)'>"+
                    "<img src=images/slide.blankslider.gif "+
                    "name='TickMarkI0-"+i+"' height=10 width="+
                    Math.floor(TTW/NUMCELLS)+
                    " border=0></a>"+
                    "</td>");
  }
  //-->
  </script>
</tr></table>
</td></tr></table>
```

Code Example 4. Building a table-based slider programmatically

```
document.write("<div id='left0drag' " +
    "style='position: relative; left: 0px; top: 34px; background-color: black;"+
    " width:1; height=15' ></div>");
document.write("<div id='right0drag' " +
    "style='position: relative; left: 0px; top: 15px; background-color: black;"+
    " width:1'; height=15 ></div>");
```

Code Example 5. Adding thumbs in their own layers, on top of the table

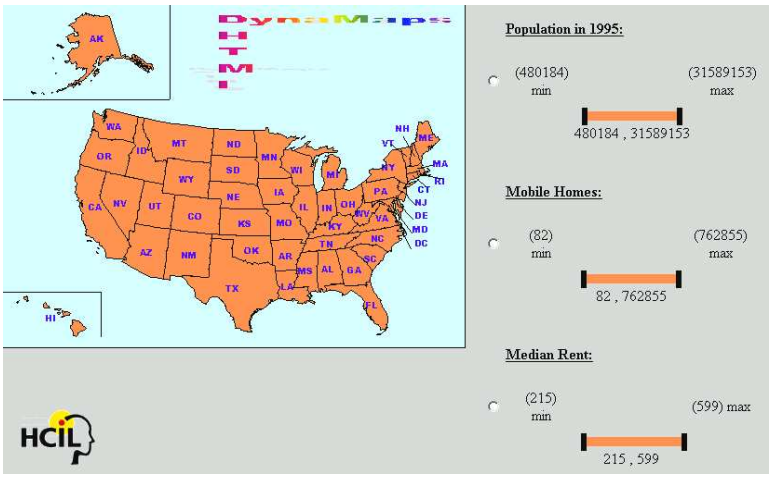

Figure 12a. US Map when first loaded

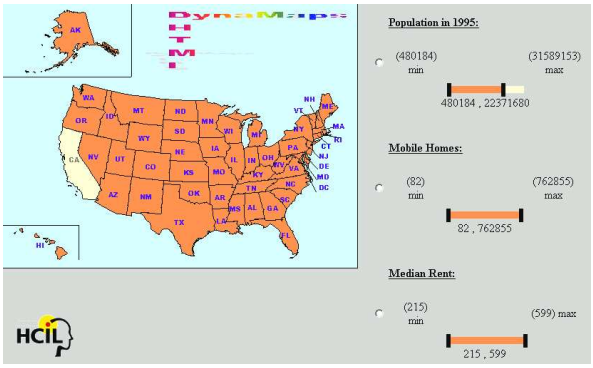

Figure 12b. Left thumb of first slider moved to remove states with very high populations.

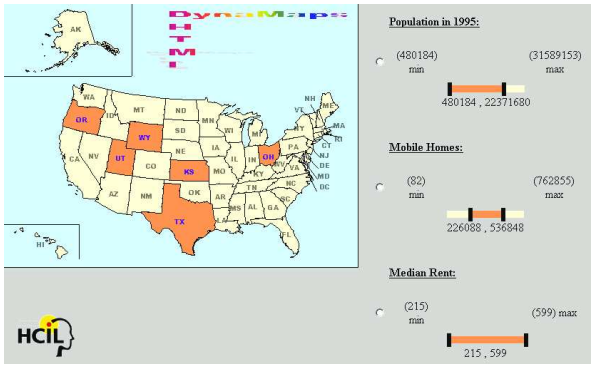

Figure 12c. Both thumbs of second slider moved to remove states with either a high or low number of mobile homes.

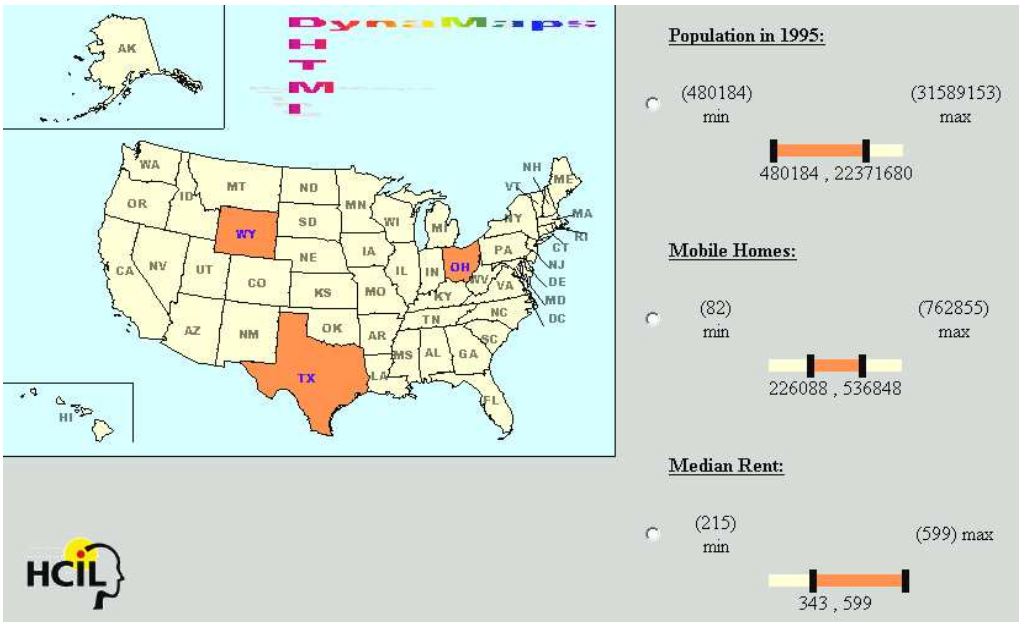

Figure 12d. US Map with all sliders in final positions to show the states that do not have high populations, have a medium number of mobile homes, and do not have low median rent.

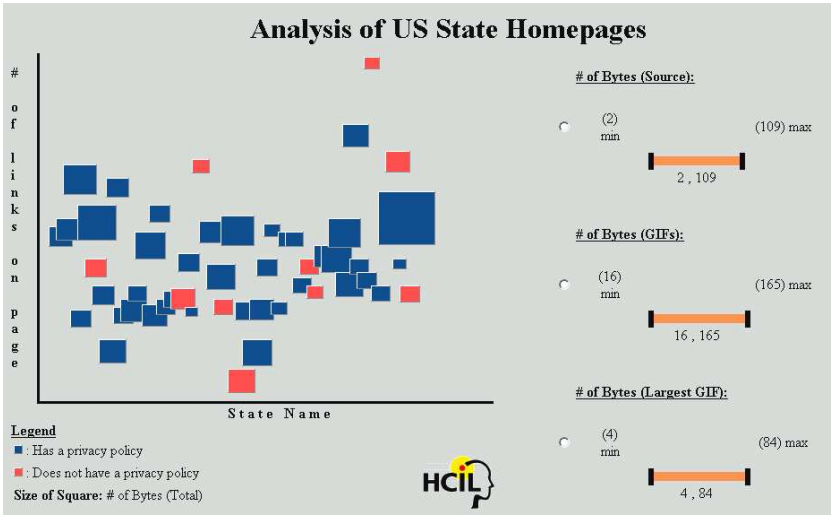

Figure 13a. Scattergram of state web pages when first loaded. Color of box represents privacy policy. Size of box represents total number of bytes for page. Position of box represents the state and the number of links on the page.

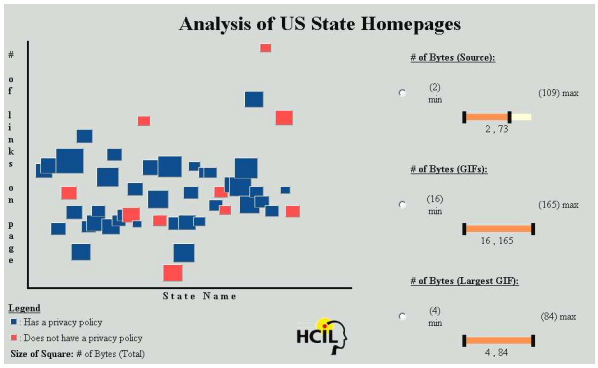

Figure 13b. Left thumb of first slider moved to remove pages with large source code.

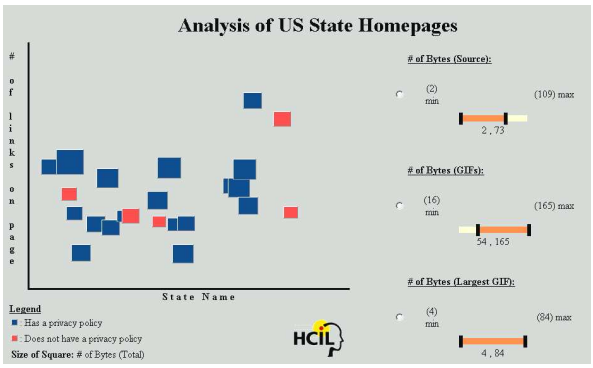

Figure 13c. Both thumbs of second slider moved to remove pages with a small total number of bytes dedicated to GIFs.

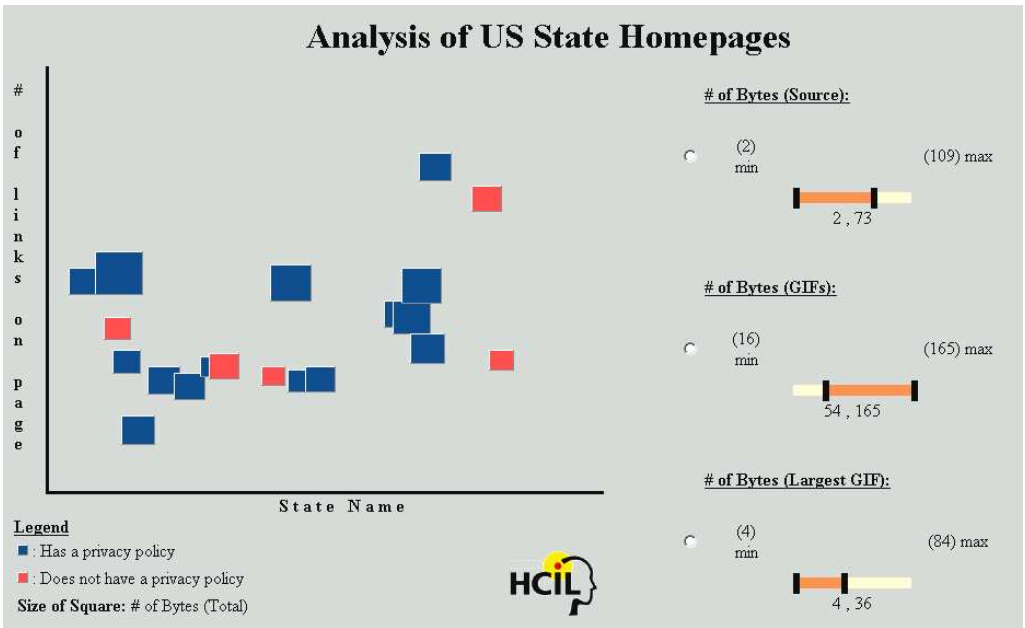

Figure 13d. Scattergram with all sliders in final positions to show the pages that do not have large source code, small number of bytes dedicated to GIFs and not having any very large GIFs.

4. Integration of Components

Since both the image construction techniques and slider construction techniques use DHTML, and the data is transmitted as part of the page, components can easily interact within a web page. Figures 12 and 13 show examples of this. However, there are challenges associated with the use of these technologies within a web page that we had to overcome. The first is browser DHTML standards (or current lack thereof). The second is refresh time for the results of queries. The third is the load time of the page itself.

The first challenge is being resolved with successive browser releases. It can also be worked around in the short-term by either limiting users to a specific browser (e.g.: Internet Explorer 5+) or by creating multiple versions of the DHTML elements. If opting for the multiple-version solution, implementers could either have the web server dispatch the correct one by detecting the browser, or have both sets of codes within the page and use conditional statements and client-side tests to determine the version to execute.

The second challenge is the more lingering and interesting one. Due to the nature of how we construct the result image using multiple layers, there is a time overhead for updates that would grow with the number of layers as well as the number of variables being controlled and the precision of the sliders. In our initial versions, each time a thumb was moved, the values associated with each layer were tested to determine whether (based on the current state of the sliders) they should be hidden or visible. With one variable and 50 layers, this process took a noticeable, but not distracting amount of time. However, with more variables and three sliders on the page, the updates began to be noticeable and interfered with the user experience.

This challenge was overcome with boolean status matrices for the layers and sliders. When a slider's state is altered, each layer is rechecked for visibility based on that slider's variable. Then, when the result image is updated, the matrix is tested and a layer is only displayed if none of it's stored boolean values said that it should be hidden according to the position of one of the sliders. This process was noticeably faster and helped to reduce the distraction while moving sliders. The speed of the machine being used did have an impact on the speed of these updates as well. Updates that were still distracting on a PII-233 were not on a faster Xeon processor.

The next modification addressed updating the sliders themselves. Initially, the entire slider was refreshed each time the range was changed. However, the only region of the slider that needed to be modified was the area between the previous position and new position of the moved thumb. Localizing the refresh to this region produces a dramatic performance improvement.

The data can be sorted and a set of lookup tables used to denote which data corresponds to which layer. This enables a binary search to determine the range of layers whose visibility status would have been altered by the last thumb movement. The status matrices for those layers are updated, and those layers' visibility properties are modified as needed. A linear loop through these layers sets their visibility. If only a small number of layers are affected, there is a significant performance boost. This is most noticeable when users move a thumb slowly. Each time a mouseMove event fires in the previous technique, the entire set of layers is re-evaluated. With this improvement, only the affected subset is re-evaluated. On an example page with 100 layers, the difference is visually noticeable when users drag a thumb at a slow but constant pace. The worst case is that the values associated with most of the layers fall within one move of the thumb. Table 1 shows the number of seconds it took to perform a single sweep through all of the layers on a PII-400 Mhz machine:

Table 1. Seconds to perform a full update

Number of Layers	Seconds (appx)
100	1
200	2
500	4
1000	10
2000	35
5000	200

The vast majority of this time is taken by the updates to the visibility of the layers. For example, of the 35 seconds to update a 2000-layer example, 32 seconds are used to update the screen itself. This time is not in the computations to determine the state of each layer's visibility, but rather on the refresh of the screen (i.e.: if the two lines of code which change the visible property are commented out, 32 seconds are "saved"). Similarly, for the 5000-layer example, 195 of the 200 seconds are taken by Internet Explorer's processing of the screen state changes.

The third performance challenge is the download speed of the page. This is something that will be highly dependent upon the users' Internet connection. However, there are certain factors within the implementer's control. Since the data needs to be transmitted with the page, if there are many variables there will be more data and thus slower download. The connection speed and users' preferences could be taken into account when determining how many variables to allow them to control. For image-based result displays, the quality of the image and number of layers will directly affect download time. While the number of layers is not something that can really be altered for a specific page, the quality of the images used could be a design decision. Also, if there are preliminary pages (such as one on which users are asked to select which variables they want to control) then the images for the layers could get downloaded by that page. This could hide a good portion of the download time. Other options such as a "please wait" page (though less desirable) could also be used to inform users.

5. Conclusions

Enabling Web users to perform dynamic queries is a desirable goal. Simplified input and rapid visual display of results should enable many more people to benefit by exploring consumer product information, demographic data, or business reports. Information visualization with DHTML will contribute to efforts by government agencies to broaden their range of users. Using technologies such as Java or Flash is possible, and we have shown the use of Dynamic HTML is also feasible. As browsers that support CSS become the norm this option will be increasingly appealing because the complexity in the design and maintenance of such pages will decrease. DHTML sliders will work best for tasks with low processing overhead. In the examples of the US map and scattergrams, the responsiveness was much better with a relatively low number of layers (fewer than 250).

DHTML sliders are also useful in other applications. For example, they can be inserted into a Web page with a survey to allow users to move the thumb to correspond to their rating of some question. The thumb's value can be submitted as another piece of data on the form. The

DHTML solution might also prove useful on small handheld devices where the overhead of running a Java program might outweigh the benefits of interaction that the Java applet provides.

In addition to the "sandwich" of layers technique, text tables rather than images could be constructed in such a way that rows appear and disappear based on the users dynamic queries. In general, in situations where it would be desirable to add some ability for users to perform dynamic queries within a Web page, Dynamic HTML techniques are an appealing strategy.

Working prototypes of pages using the three types of sliders to manipulate US maps as well as scattergrams can be found at:

http://www.cs.umd.edu/hcil/census/DHTMLproto

Acknowledgements: We appreciate the support of our research from the US Bureau of the Census.

6. References

[1] Ahlberg, C. and Shneiderman, B., Visual information seeking: Tight coupling of dynamic query filters with starfield displays, *Proc. of ACM CHI94 Conference* (April 1994), 313-317 + color plates.

[2] Ahlberg, Christopher, Williamson, Christopher, and Shneiderman, Ben, Dynamic queries for information exploration: An implementation and evaluation, *Proc. ACM CHI'92: Human Factors in Computing Systems* (1992), 619-626.

[3] Andrienko, G. and Andrienko, N., Interactive maps for visual data exploration, *Intl Journal of Geographical Information Science, 13*, 4 (1999), 355-374.

[4] CSS, http://www.w3.org/Style/CSS

[5] Dang G., North C. and Shneiderman B., Dynamic queries and brushing on choropleth maps, *Proc. International Conference on Information Visualization 2001*, Available from IEEE Press (2001), 757-764.

[6] Dynamic HTML, http://www.webreference.com/dhtml

[7] Fishkin, K. and Stone, M., Enhanced dynamic queries via movable filters, *Proc. ACM CHI'95 Conference on Human Factors and Computing Systems* (1995), 415-420.

[8] Fredrikson, A., North, C., Plaisant, C., Shneiderman, B., Temporal, geographical and categorical aggregations viewed through coordinated displays: A case study with highway incident data, *Proc. 1999 Workshop on New Paradigms in Information Visualization and Manipulation*, ACM New York (November 1999), 26-34.

[9] Goldstein, J., Roth S. F., Using aggregation and dynamic queries for exploring large data sets, *Proc. ACM CHI'94 Conference on Human Factors and Computing Systems*, ACM, New York (April 1994), 23-29.

[10] Jain, V. and Shneiderman, B., Data structures for dynamic queries: An analytical and experimental evaluation, In Catarci, T., Costabile, M., Levialdi, S., and Santucci, G. (Editors), *Proc. Advanced Visual Interfaces Conference '94*, ACM Press, New York (1994), 1-11.

[11] MacEachren, A. M., Hardisty, F., Gahegan, M., Wheeler, M., Dai, X., Guo, D, and Takatsuka, M., Supporting visual integration and analysis of geospatially-referenced data through web-deployable, cross--platform tools, *Proc. National Conference for Digital Government Research*, (2001), 17-24. Available from Digital Government Research Center, http://www.dgrc.org/dgo2001/

[12] Mockus, A., Navigating aggregation spaces, *Proc. IEEE Conference on Information Visualization '98, IEEE*, Los Alamitos, CA (1998), 29-32.
[13] Shneiderman, B., Dynamic queries for visual information seeking, *IEEE Software 11*, 6 (1994), 70-77.
[14] Spotfire, www.spotfire.com, Somerville, MA, 2002.
[15] Takatsuka, M. and Gahegan, M. N., Exploratory geospatial analysis using GeoVISTA *Studio*: From a Desktop to the Web, Proc. *First International Workshop on Web Geographical Information Systems (WGIS2001)* (2001). http://www.geovistastudio.psu.edu/pdf/wgis14_takatsuka_m.pdf
[16] Tang, L. and Shneiderman, B., Dynamic aggregation to support pattern discovery: A case study with web logs, *Proc. Discovery Science 4th International Conference 2001*, Editors (Jantke, K. P. and Shinohara, A.), Springer-Verlag, Berlin, 464-469.
[17] Williamson, Christopher, and Shneiderman, Ben, 1992. The Dynamic HomeFinder: Evaluating dynamic queries in a real-estate information exploration system, *Proc. ACM SIGIR'92 Conference* (June 1992), 338-346.