A Scalability Study of Web-Native Information Visualization

Donald W. Johnson* T.J. Jankun-Kelly*

Mississippi State University

Abstract

Several web-native information visualization methods (SVG, HTML5's Canvas, native HTML) are studied to contrast their performances at different data scales. Using Java implementations of parallel coordinates and squarified treemaps for comparison, we explore the design space of these web-based technologies in order to determine what design trade-offs are required.

Keywords: information visualization, web-based visualization, parallel coordinates, treemap, Java, scalable vector graphics (SVG), HTML Canvas, asynchronous Javascript (AJAX)

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation, H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based Services, I.3.8 [Computer Graphics]: Application—Visualization

1 Introduction

Web-based visualization technologies enable users from many geographic locations to access, explore, play with, and understand data world-wide. Such interaction has a long history in scientific [3,4,7,12,17,25,26] and information visualization [5,18,19,21] and has seen a recent resurgence in the larger context of social visualization [1,14,22,23]. These visualizations are enabled by web-embedded applications, usually small applets written in programming languages such as Java or Flash. While interpreters for such languages are common on web-platforms, it is worth exploring alternative means of visualization content delivery for devices without such capabilities (e.g., mobile devices) or for situations where dynamic composition of web-services ("mash-ups") are desirable. In this work, we explore different approaches to web-native information visualizations in order to map out the possible design space.

Web-native technologies include vector graphics (SVG [13]), 2-dimensional raster graphics (HTML5's Canvas [15]), and 3-dimensional modeling (VRML and its successor X3D [24]). Interaction is provided by judicious use of Javascript. These formats are "web-native" in that they are document-like descriptions of the display rather than compiled code, and thus integrate naturally into the mark-up structure of HTML. Contemporary browsers support different aspects of these technologies, and it is anticipated that such support will improve over time. As our interest is 2D information visualization, we focus on SVG and Canvas as our experimental native architectures; Java is used as our control architecture. These three technologies were used to implement both parallel coordinates [16] and squarified treemaps [8,20]. A native HTML version was also tested for treemaps. These seven implementations form the basis of our scalability study.

The purpose of this work is to enumerate the strengths and weaknesses of web-native information visualization technologies by using scalability studies. In addition, we seek to determine how the utilization of these technologies differs from interaction with traditional graphics programming environments. This study provides guidance for the design of web-native visualizations and serves to illuminate areas where further research into alternative approaches may be desirable.

2 Background

Web-based visualizations range from fully server-based to fully client-based ("thin-client") [25], though thin-client approaches dominate [3–5,7,12,14,17–19,21–23,25,26]. Whereas thin-client applets can be made interactive, traditional pure web interfaces have always suffered from slow responsiveness due to the necessity of round-trip communication to update results. This state of affairs recently changed as pure web interfaces gained the ability to interactively update web-pages via asynchronous server communication initiated by Javascript events. Applications such as Google Maps and Google Mail demonstrate the effectiveness of this "AJAX"-based approach.[1] In addition, the aforementioned web-native display languages (SVG [13], HTML5's Canvas [15], and X3D [24]) provide XML or Javascript dialects which can be easily integrated into HTML. Already, rich thin-client visualizations have been provided for geographical visual analytics using SVG [11]. However, the data used in this setting was small in scale (300 edge and 200 node graphs [11]). Thus, a question remains outstanding: How scalable are these web-native technologies for visualization? To understand this scalability, we must first understand the methods' design.

This study utilizes SVG, HTML5's Canvas, and some native HTML to create visualizations. Scalable Vector Graphics (SVG) [13] is an XML dialect used to describe vector graphics wherein images are composed of drawing elements rather than raster pixel colors. In this context, a picture consist of a list of paths, basic shapes, and text—each with its own appropriate graphical attributes. These are specified by XML elements such as `<path>` and XML attributes such as `fill`. It is then up to the web browser or SVG client to rasterize the vector image description. Because of this vector format, SVG visualizations can be very concise—only the elements making up the depiction need be specified, rather than the entire pixel plane. However, if the number of visual elements exceeds the equivalent raster image size, then the vector representation becomes memory intensive. This behavior will factor into the performance of our SVG parallel coordinates implementation.

The HTML5 specification is a candidate successor for current HTML/XHTML implementations, and is designed specifically for rich web-based applications [15]. Many of HTML5's additions add user interface or interaction elements. For example, this study makes use of the HTML5 `<canvas>` element to describe immediate mode 2D raster graphics. Given a region of screen space, elements are rendered to the canvas by using Javascript interpreted by the browser. Since this rendering is procedurally generated, only the final image must be stored in memory once the Javascript code is executed; the canvas raster image is cached and only re-generated on request. This contrasts to SVG, where the entire document structure (the vector commands) must be stored indefinitely to generate

*Department of Computer Science and Engineering, James Worth Bagley College of Engineering, Mississippi State University, Mississippi State, MS, 39762. Email: dwj12@msstate.edu, tjk@acm.org

[1] Such technology had been possible previously, but only since 2005 has wide support for it been available in the majority of web-browsers [2]

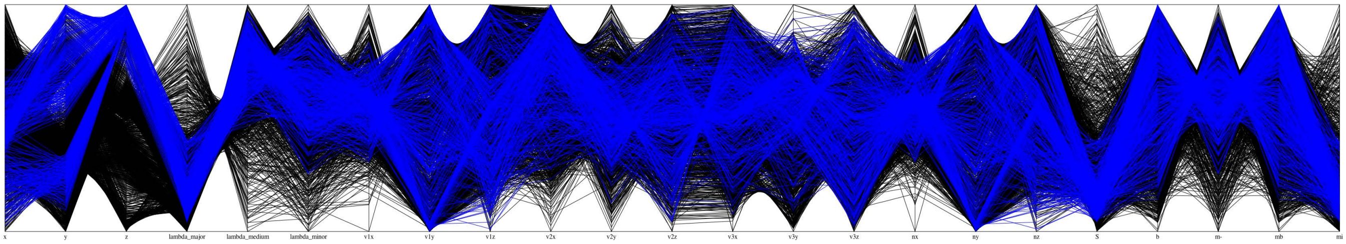


Figure 2: Canavs parallel coordinates implementation (Full disclination dataset)

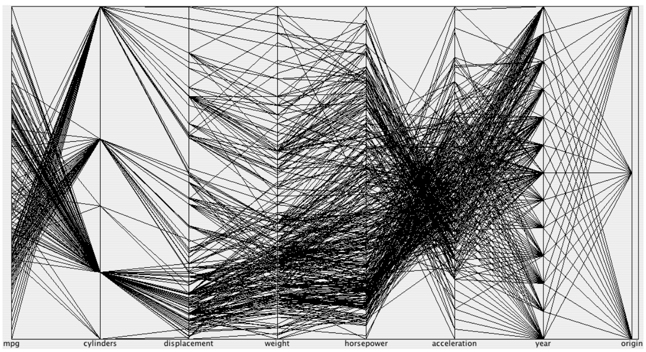


Figure 1: Java parallel coordinates implementation (Cars dataset)

the graphics. However, with HTML5's canvas element, the text of the code must still be loaded or dynamically generated before execution, which increases page transmission or load time.

HTML, through style sheets, has its own limited graphical subset consisting of styled box rendering [6]. Thus, HTML can be used to directly depict images represented only by boxes and text. Each non-inline element in HTML is considered to be a rectangle for layout purposes. The size, color, and other visual characteristics of this box are controlled by style attributes specified by the HTML source or by a linked style file. Since treemaps are formed of nested boxes, and because HTML elements can easily be rendered as boxes, we used HTML `<div>` elements as a web-native rendering method, to study its applicability.

3 Experimental Design

The performance of each of the web-native architectures was measured in regards to two different types of visualizations: parallel coordinates and squarified treemaps. The implementation of each visualization was kept as similar as possible so that differences in display or response time would be due to the performance differences of the respective architectures. Each method was tested on Mac OS X 10.5.2 and Windows Vista using the latest versions of the Firefox (2.0.0.12) and Safari (3.0.4) web-browsers. Internet Explorer was not extensively used as it lacked support for HTML5 and native SVG. Tests were performed on a 2.4 GHz Intel Core Duo chip MacBook Pro with 4GB of 667 MHz DDR2 SDRAM, and an NVidia GeForce 8600M GT card with 256 MB of video memory.

For each visualization, a web page was loaded and then selection was done randomly by hand; previous random-based selection did not sufficiently cover different areas of selection. The web page was forced to redisplay in order to generate layout (creation of the SVG/Canvas structures), display (actual rendering), and selection times. For the parallel coordinates tests, the time required to load the data file was also recorded. For each visualization system, these performance values were sampled 100 times except in instances where initial poor performance times made acquisition of 100 samples unfeasible. In some instances, the web browsers crashed repeatedly, completely preventing data acquisition; however, all successful tests times were recorded and averaged.

3.1 Parallel Coordinates Implementation

In this study, all parallel coordinates implementations load datasets in two structures: One for axes, and the other for data lines. The axes data structure encodes the display location of each axis and the minimum/maximum values on that axis; an incidental effect of this choice is a built-in capacity for efficient reordering of the displayed axes. The data lines consult the relevant data axis to determine their intersection along the axes, repeating this process for each axis and data line. Brushing of data using a mouse-controlled "highlight" line is used for selection; axes can also be repositioned.

The Java applet (Figure 1) does not record separate line layout and display times, since the lines are laid out during the display process. The HTML5 Canvas implementation (Figure 2) utilized a similar code structure, with the following changes: Javascript functions, called when the web page finished loading, pulled data from the server using asynchronous XMLHTTPRequests, constructed the display data structures as the data loaded, then handled interaction. Since the `<canvas>` element does not support text, separate HTML elements were used for axis labels. Their position was updated by Javascript whenever the axes were reordered.

The SVG implementation differs from the Canvas implementation in two ways. First, the SVG document contains actual rendering positions after the data is parsed; these positions are updated if an axis is moved. Secondly, the display is grouped into three sets of drawing primitives: Axes, selected lines, and non-selected lines. Initially, a path was used for each data line (either selected or not); however, because of performance issues, paths are created such that each path is not larger than 100,000 characters. Separate path groups are used for selected and non-selected data lines.

3.2 Squarified Treemap Implementation

The treemap implementations each parse a file containing a flattened file hierarchy. Each line of the input files consists of a file path and an associated size. This information was used to extract the tree hierarchy, stored as Java or Javascript objects. Given a root node, the bounds of the node's children and descendants are then determined recursively, using the squarified treemap layout.

The treemap implementations differed primarily in how display was handled. In all cases, only elements without children were rendered. In the Java and Canvas implementation, a list of elements to be displayed was accumulated as the file hierarchy was traversed. The elements of the list were then rendered with the appropriate rectangle drawing commands. For SVG, a path string for each rectangle was accumulated during the hierarchy transversal and then stored as a single element in the document (Figure 3). Finally, the pure HTML treemap (Figure 4) uses a corresponding `<div>` element for each leaf, absolutely positioned within the display region and surrounded by a one pixel black border.

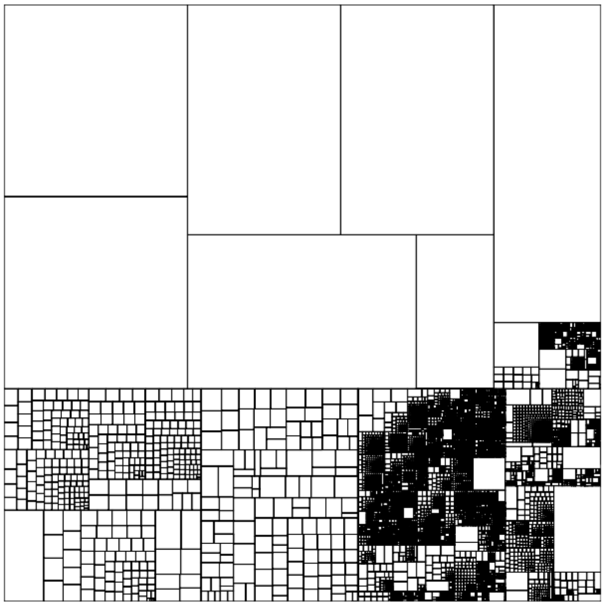


Figure 3: SVG treemap implementation (Medium dataset)

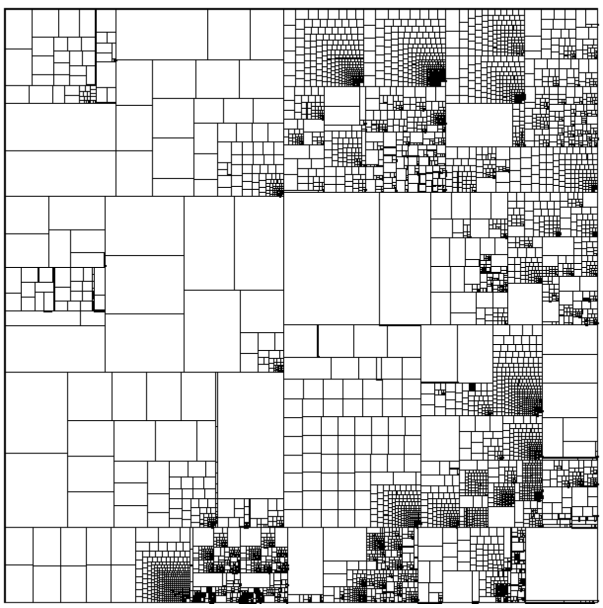


Figure 4: Pure HTML treemap implementation (Small dataset)

3.3 Datasets

To test the scalability of the various web-native visualizations, we chose data sets of ever increasing size. Each data set was larger than its predecessor (in terms of entries) by roughly an order of magnitude. For parallel coordinates, the data used was the standard cars dataset [9, 10] and two datasets from molecular simulations (Table 1). To ensure a fair comparison, tests were run using subsets limited to eight data fields, the limit of the cars data set. For the treemaps, three file hierarchies were used: A directory containing programming projects for the Small dataset, a user's home directory for the Medium dataset, and a complete hard drive for the Large dataset (Table 2). Each hierarchy was then flattened such that the data file stored only the path and cumulative file size for each entry.

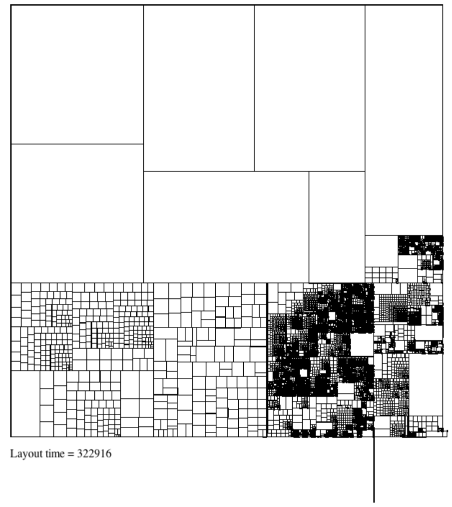


Figure 5: Invalid pure HTML rendering (Medium dataset).

Table 1: Parallel Coordinate Datasets

Dataset	Size	Lines	Columns
Cars	32 kB	406	8
Disclination	604 kB	2664	24
Molecule	39.6 MB	140455	32

Table 2: Treemap Datasets

Dataset	Size	Entries	Max Depth
Small	592 kB	9272	11
Medium	4.3 MB	55997	21
Large	80.3 MB	791931	22

4 Results and Discussion

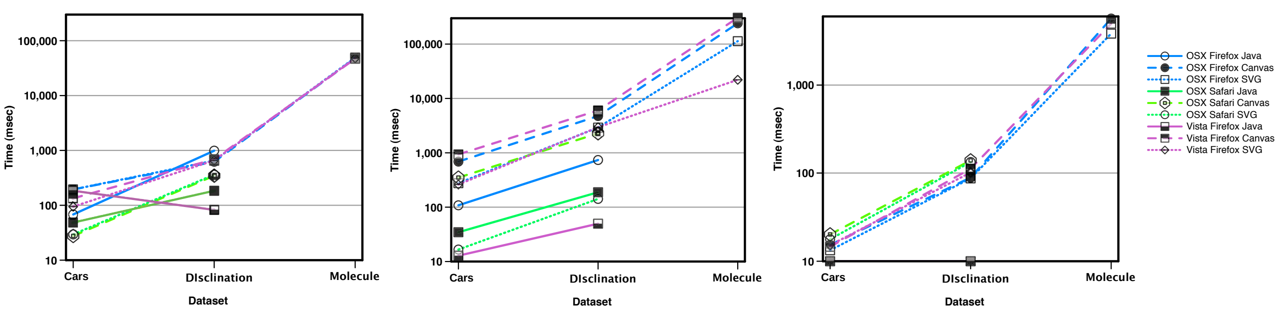
Visually, the implementations have no distinct differences. All were capable of displaying the data in a manner consistent with their Java analogs, though the pure HTML renderer produced poor results on the larger datasets (Figure 5). The primary difference is in performance. Tables 3 and 4 record the averaged timings for the parallel coordinates and treemap tests respectively, with Figures 6–7 summarizing the trends. Uncertainty bars are not shown as they are too small to be visible at the given scale. All times are given in milliseconds. If for some reason timing was not possible, the entry is "—"; reasons for such behavior are discussed further below.

Unsurprisingly, Java generally performed better in all measured categories where the Java virtual machine did not run out of memory—Java is more mature, byte-compiled, and has access to hardware acceleration. While the amount of memory can be increased, our tests were against baseline installations of Java where control of the JVM may not be easily performed or familiar to users. SVG performed next best, with non-selection performance within an order of magnitude of the Java performance for smaller data sets and better than Canvas until very large dataset size. However, as the complexity of the SVG document increased, such as with the treemap datasets, Canvas performed better in terms of display. Selection was similar for Canvas vs. SVG. In terms of interactivity, if we use 100ms as an upper threshold for "interactive" performance, then selection was interactive or near interactive for most of the tested systems except for the large Molecule dataset; web-native

Table 3: Parallel Coordinate Performance (msec)

Dataset	OS	Browser	Code	Load	Layout	Display	Selection
Cars	OS X	Firefox	Java	68	—[a]	109	0.49
	OS X	Firefox	SVG	149	29	195	13
	OS X	Firefox	Canvas	152	21	609	15
	OS X	Safari	Java	49	—[a]	35	0.34
	OS X	Safari	SVG	14	16	17	18
	OS X	Safari	Canvas	13	14	349	20
	Vista	Firefox	Java	184	—[a]	13	0.48
	Vista	Firefox	SVG	21	9	263	15
	Vista	Firefox	Canvas	106	29	934	15
Disclination (8 cols.)	OS X	Firefox	Java	995	—[a]	741	3
	OS X	Firefox	SVG	516	140	653	87
	OS X	Firefox	Canvas	497	125	4770	89
	OS X	Safari	Java	188	—[a]	35	2.57
	OS X	Safari	SVG	233	127	142	134
	OS X	Safari	Canvas	235	109	2274	139
	Vista	Firefox	Java	83	—[a]	50	1.8
	Vista	Firefox	SVG	546	117	2973	102
	Vista	Firefox	Canvas	565	124	6011	112
Molecule (8 cols.)	OS X	Firefox	Java	—[c]	—	—	—
	OS X	Firefox	SVG	36648	11758	48959	3831
	OS X	Firefox	Canvas	36601	11546	241046	5659
	OS X	Safari	Java	—[c]	—	—	—
	OS X	Safari	SVG	—[c]	—	—	—
	OS X	Safari	Canvas	—[c]	—	—	—
	Vista	Firefox	Java	—[c]	—	—	—
	Vista	Firefox	SVG	35376	12694	22026	—[b]
	Vista	Firefox	Canvas	35070	11679	306375	4964

[a]Layout concurrent with loading. [b]Crashed on line selection. [c]Out of memory.



(a) Layout Time (b) Display Time (c) Selection Time

Figure 6: Timing charts for the Parallel Coordinates Renderers (see Table 3). All times in msecs; charts use different log-scales.

rendering was interactive only for the smaller datasets.

Our SVG implementation uses paths, a sequence of pen movement operations: One set of paths for selected lines and another for the non-selected data. Most SVG implementations will draw each element of this path immediately, which causes significant refreshes of the display, leading to unacceptable performance. To resolve this flaw, immediate rendering is disabled with the SVG `suspendRedraw` Javascript command; after the entire path is loaded, a redraw is forced and drawing is re-enabled via `forceRedraw` and `unsuspendRedraw` respectively [13, §5.17]. Even with this optimization, issues occured. Paths must be stored in the web browser's memory, usually in an intermediate form, and this appears to severely degrade performance when the memory requirements exceed a certain threshold. Browser SVG implementations also possess a maximum path size which forces us to add additional paths if a this limit is exceeded (about 100,000 characters per path). Performance degradation is still apparent in both the Molecule parallel coordinates and the Large treemap tests: SVG times either grew by two orders of magnitude (a superlinear increase from previous timings) or failed to display due to lack of memory or excessive processing. The path structure also hurts selection performance as the treemap tests demonstrate: The nonslected paths have to be traversed in order to remove the new selection and add it to the selected paths.

Canvas uses immediate mode rendering, with graphical elements drawn as Javascript commands are issued. The rendering is doubled buffered; partial rendering is only noticed if the browser must interrupt the script to perform other tasks. This implementation is memory efficient, as only the initial data structures and the rasterized pixels must be stored in memory. Thus, Canvas' performance was able to display our largest data set where SVG was not. However, this rendering was nowhere near interactive rates, varying between

Table 4: Treemap Performance (msec)

Dataset	OS	Browser	Code	Layout	Display	Selection
Small	OS X	Firefox	Java	26	313	0
	OS X	Firefox	SVG	818	62	3
	OS X	Firefox	Canvas	482	453	0.6
	OS X	Firefox	HTML	3182	—[a]	8
	OS X	Safari	Java	60	66	0
	OS X	Safari	SVG	1217	0.14	21
	OS X	Safari	Canvas	890	146	0.3
	OS X	Safari	HTML	—[b]	—	—
	Vista	Firefox	Java	28	8	0
	Vista	Firefox	SVG	1480	92	5
	Vista	Firefox	Canvas	572	393	0.5
	Vista	Firefox	HTML	3164	—[a]	9
Medium	OS X	Firefox	Java	104	1693	0
	OS X	Firefox	SVG	5303	347	2
	OS X	Firefox	Canvas	3287	2464	0.6
	OS X	Firefox	HTML	282015	—[a]	114
	OS X	Safari	Java	102	354	0
	OS X	Safari	SVG	8239	0.05	121
	OS X	Safari	Canvas	—[c]	—	—
	OS X	Safari	HTML	—[b]	—	—
	Vista	Firefox	Java	112	40	0.1
	Vista	Firefox	SVG	29759	689	8
	Vista	Firefox	Canvas	3918	2118	0.3
	Vista	Firefox	HTML	25587	—[a]	27
Large	OS X	Firefox	Java	—[c]	—	—
	OS X	Firefox	SVG	—[c]	—	—
	OS X	Firefox	Canvas	116168	74286	8.13
	OS X	Firefox	HTML	—[d]	—	—
	OS X	Safari	Java	—[c]	—	—
	OS X	Safari	SVG	—[b]	—	—
	OS X	Safari	Canvas	—[c]	—	—
	OS X	Safari	HTML	—[b]	—	—
	Vista	Firefox	Java	—[c]	—	—
	Vista	Firefox	SVG	—[d]	—	—
	Vista	Firefox	Canvas	98614	25821	0.4
	Vista	Firefox	HTML	—[d]	—	—

[a]Display/layout concurrent. [b]No output. [c]Out of memory. [d]Failed to terminate.

half a minute for the Large file hierarchy to half an hour for the full Molecule dataset. As previously mentioned, selection times were on par of faster than the SVG implementations; we believe this is due to the lack of path updating in the Canvas case.

The pure HTML treemap implementation was disappointing, performing significantly worse than all implementations excluding Safari's SVG renderer in all cases where it functioned. While HTML does render boxes natively, the structure of a typical document does not posses the complexity we need for the treemap. The highly nested boxes are not rendered correctly; extraneous lines or other visual anomalies were present across browsers and operating systems (Figure 5). In addition, like SVG, each `<div>` element creates memory overhead in the parsed document tree, reducing performance. For the largest dataset, no result was displayed even after a few hours.

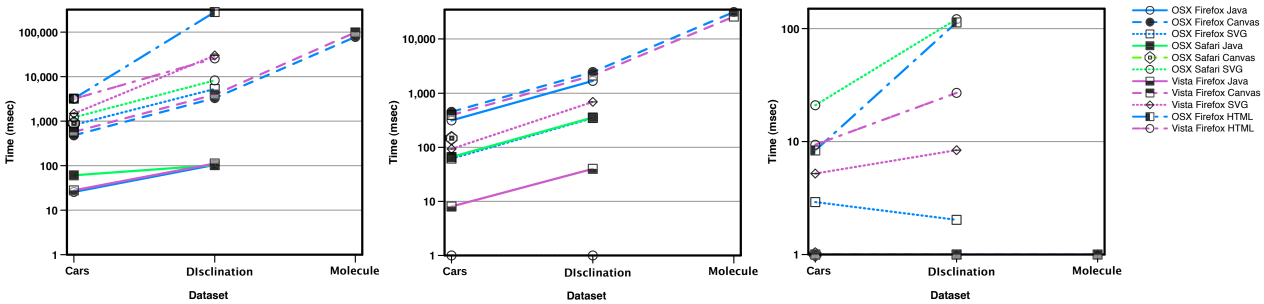
While there were some differences between operating systems, within an OS, the performance trends were similar.

5 Conclusions

Web-native display technology has the potential to expand the impact of visualization to areas where program applets in Java and Flash are not available. However, the limitations of these technologies must be fully understood in order to appreciate their benefits. We have contributed a scalability study of SVG, HTML5's Canvas, and pure HTML display methods for parallel coordinates and treemaps in order to explore this design space.

While Java had the best performance, SVG had adequate interactivity for medium or smaller datasets. When path sizes can be minimized, SVG's vectorized format appears to have the (slight) advantage over Canvas' immediate mode approach for the two visualization tested. We would expect even higher performance if redundant elements in a path (such as data lines between the same points) were removed. With care, SVG can handle very large visualizations as well. However, neither SVG nor Canvas are ideal if large data visualization must be combined with interactive performance. We do not recommend the use of pure HTML rendering as it does not appear to be optimized for highly nested documents.

There are several areas that could be examined more thoroughly. As mentioned, modern Java implementations are hardware accelerated. To level the playing field, acceleration could be disabled for Java or accelerated version of Canvas/SVG sought. Such a study would get more at the fundamental differences of the display meth-



(a) Layout Time (b) Display Time (c) Selection Time

Figure 7: Timing charts for the Treemap Renderers (see Table 4). All times in msecs; charts use different log-scales.

ods, not the effects of hardware. In addition, other traditional visualization methods such as graphs and scatterplots have yet to be explored; these visualization use vastly different visual primitives from the two methods presented here and could provide different insights. Finally, further investigations into optimizations could be performed. We purposefully tried to keep our implementations similar across the Java and web-native architectures to test the architectures instead of our coding ability. However, finely tuned implementations would provide more accurate guidance for the ultimate performance of these web-based techniques.

Acknowledgments

This work was partially funded by the NASA Science Mission Directorate, Earth System Division, Applied Science Program under a contract to Mississippi State University through NASA's Stennis Space Center.

References

[1] Swivel. http://www.swivel.com/ (Accessed December 2007).

[2] Wikipedia: AJAX. http://en.wikipedia.org/wiki/AJAX (Accessed December 2007).

[3] C. S. Ang, D. C. Martin, and M. D. Doyle. Integrated control of distributed volume visualization through the world-wide-web. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of the 5th IEEE Conference on Visualization (Vis'94)*, pages 13–20. ACM Press, 1994.

[4] C. Bajaj and S. Cutchin. Web based collaborative visualization of distributed and parallel simulation. In J. Ahrens, A. Chalmers, and H.-W. Shen, editors, *Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium (PVG'99)*, pages 47–54, Los Alamitos, CA, October 25–26 1999. IEEE Computer Society Press.

[5] M. Bender, R. Klein, A. Disch, and A. Ebert. A functional framework for web-based information visualization systems. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):8–23, January-March 2000.

[6] B. Bos, H. W. Lie, C. Lilley, and I. Jacobs. Cascading style sheets (CSS) level 2. Technical report, World Wide Web Consortium, 1998.

[7] K. Brodlie, S. Lovegrove, and J. Wood. Harnessing the web for scientific visualization. *Computer Graphics*, 34(1), February 2000.

[8] M. Bruls, K. Huizing, and J. J. van Wijk. Squarified treemaps. In *Proceedings of the 2000 Joint Eurographics/IEEE TVCG Symposium on Visualization (VisSym '00)*, pages 33–42, 2000.

[9] L. H. Cox. Cars dataset. http://lib.stat.cmu.edu/datasets/cars.data (Accessed December 2007).

[10] D. Donoho and E. Ramos. PRIMDATA: Data sets for use with PRIM-H. Unpublished, 1982.

[11] S. G. Eick, M. A. Eick, J. Fugitt, B. Horst, M. Khailo, and R. A. Lankenau. Thin client vsualization. In W. Ribarsky and J. Dill, editors, *IEEE Symposium on Visual Analytics Science and Technology 2007*, pages 51–58, 2007.

[12] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In D. S. Ebert, M. Gross, and B. Hamann, editors, *Proceedings of the 10th IEEE Conference on Visualization (Vis '99)*, pages 139–519, Los Alamitos, CA, October 24–29 1999. IEEE Computer Society Press.

[13] J. Ferraiolo, J. Fujisawa, and D. Jackson. Scalable vector graphics (SVG) 1.1 specification. Technical report, World-Wide Web Consortium, 2002.

[14] J. Heer, F. B. Viégas, and M. Wattenberg. Voyagers and voyeurs: Supporting asynchronous collaborative information visualization. In *Proceedings of the 2007 SIGCHI conference on Human factors in computing systems (CHI '07)*, pages 1029–1038, 2007.

[15] I. Hickson. HTML 5 (Working Draft—12 December 2007). Technical report, Web Hypertext Application Technology Working Group, 2007. http://www.whatwg.org/specs/web-apps/current-work/.

[16] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *IEEE Visualization*, pages 361–378, 1990.

[17] T. J. Jankun-Kelly, O. Kreylos, J. M. Shalf, K.-L. Ma, B. Hamann, K. I. Joy, and E. W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, 23(2):40–50, 2003.

[18] A. Mockus, S. Hibino, and T. Graves. A web-based approach to interactive visualization in context. In *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, pages 181–188, New York, NY, USA, 2000. ACM Press.

[19] R. M. Rohrer and E. Swing. Web-based information visualization. *IEEE Computer Graphics and Applications*, 17(4):52–59, July/August 1997.

[20] B. Shneiderman. Tree visualization with treemaps: a 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, Jan. 1992.

[21] V. Turau. A framework for automatic generation of web-based data entry applications based on xml. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1121–1126, New York, NY, USA, 2002. ACM Press.

[22] F. B. Viégas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon. ManyEyes: A site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[23] M. Wattenberg. Baby names, visualization, and social data analysis. In *Proceedings 2005 IEEE Symposium on Information Visualization (InfoVis '05)*, pages 1–6, 2005.

[24] Web3D Consortium, Inc. Information Technology—Computer Graphics and image processing–Extensible 3D (X3D) encodings—Part 1: Extensible Markup Language (XML) encoding. Technical Report ISO/IEC 19776-1:2005, ISO/IEC, 2005.

[25] J. Wood, K. Brodlie, and H. Wright. Visualization over the world wide web and its application to environmental data. In R. Yagel and G. M. Nielson, editors, *Proceedings of the 7th IEEE Conference on Visualization (Vis '96)*, pages 81–86, 470. ACM Press, 1996.

[26] J. Wood, H. Wright, and K. Brodie. Collaborative visualization. In *Proceedings of the 8th IEEE Conference on Visualization (Vis '97)*, pages 253–259, 549, 1997.