2D Information Visualization
SVG and HTML5

Version of 20 May 2011

Christian Kantner
Albert Leimüller
Evgenia Popova
Michael Steurer

Abstract

There are many different solutions and technologies to create a web-based 2D information visualization. The most common are Java Applets, JavaScript and Ajax, Flash, SVG and since 2008 HTML5 with the Canvas element. In this paper a brief overview of these technologies and short introduction by example in SVG and HTML5 Canvas architectures is conducted. A few libraries which enable the use of high quality 2D information visualizations like Protovis, RGrapgh, Google Chart Tool and the Jit is demonstrated in this work and finally a performance benchmarking of the introduced technologies which allows to judge upon technologies' potential.

Contents

Chapter 1

Introduction

1.1 HTML

1.1.1 Idea

HTML, in short HyperText Markup Language, is a markup language that was invented in 1989 by Tim Berners-Lee. The basic idea behind HTML is to connect related content by means of hyperlinks. Hyperlinks enable jumping from one content to another even if the latter is located far away. This web of information is created by the world wide web. The information itself is then shared by HTML. Raggett [1998]

Another basic idea behind the web is to reduce information replication, which is another reason why links are so important. Tim Berners-Lee could hardly foresee what potential his idea would have in the future. One important aspect of his work and its wide appreciation was the fact that he invented HTML based on SGML. SGML (Standard Generalized Markup Language) is an ISO standardized language that uses markup to describe a text by adding tags to point out what kind of content it is.

In 1995 the second version of HTML was published, HTML 2.0. HTML 3.2 was released in early 1997 and in the end of 1997 HTML 4.0 came up in three different versions (strict, transitional and frame-set). A further release of HTML 4 was already in the making when the W3C decided to stop any further development on HTML 4 and created another version of HTML, XHTML. XHTML 1.0 was released in 2000 and brought some new features and a closer implementation to XML. In 2008 HTML5 was finally published as a first draft. HTML5 is a major step and brought many new tags and possibilities like the canvas element and the svg tag. Even video is supported natively. Lubbers [2010]

1.1.2 Markup languages

HTML is one of many markup languages. It uses a list of given tags to divide the content of a web page in different types of elements like a header, paragraph or a hyperlink. Basically an opening tag for a paragraph looks like `<p>` and a closing tag of a paragraph like `</p>`.

Below is a simple markup language example. In this case the used language is XML. The tags indicate the type of content. `<book>` is the opening tag and `</book>` the closing tag. In these opening and closing tags is the content which can also include tags and further content. The example shows how a book could be divided in chapter, title and content elements.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <book>
3   <chapter>
4     <title>First Chapter</title>
5       <content>Some text.</content>
6       </chapter>
7   <chapter>
```

```
8       <title>Second Chapter</title>
9       <content>Even more text.</content>
10      </chapter>
11  </book>
```

1.1.3 JavaScript

JavaScript is an object-oriented scripting language that was published in 1995. It is mostly used in web browsers where the JavaScript code is inserted directly into the HTML code. It enables interactivity that would not be possible with the use of simple HTML which only generates a static web page. JavaScript is client-sided and calculates independently from the server-side. So combining JavaScript with HTML and PHP the calculation can be splitted up to the client-side and the server-side which increases the loading of a web page. Flanagan [2006]
JavaScript is used in a wide variety of fields. Very simple examples include the validation of forms without reloading the web page. This would not be possible in a form simply written in PHP and HTML.
Below is a simple example of how JavaScript can directly be added into the header of an HTML page using script tags.

```
1  <script type="text/javascript">
2    document.write('Hello World!');
3  </script>
```

Next to the JavaScript combination with HTML another technique became popular, AJAX. AJAX is an acronym for Asynchronous JavaScript and XML. The basic idea behind AJAX is to use already existing tools and combine their strengths. It also takes advantage of the described above technique and uses a asynchronous process combining client-side and server-side calculation. These shared resources make it possible for client browser to calculate while another calculation is done on the server and the result can be represented asynchronously in the web browser. The tools used next to JavaScript and PHP are HTML and CSS for presentation, DOM and the HTTPRequest. The HTTPRequest is used to further enhance the power of JavaScript so that it can communicate directly with the server without reloading of a page.

1.2 Java Applet, Adobe Flash

Another technology allowing the user to interact with a web page is the Java Applet.[1] It was first introduced in 1995 by Java Sun. Java Applets are programs written in Java or a programming language that compiles to Java bytecode like Jython, JRuby and Eiffel. The Java code is embedded in the HTML code. If a user opens a web page including a Java Applet, the Java code is downloaded and run in the browsers virtual machine. It is also possible to run a Java Applet independently from a web browser in an Applet viewer. Java Applets run faster than JavaScript and are executable in many different operating systems. But a disadvantage of Java Applets is that it works only if Java is installed. Figure 1.1 shows an animated implementation of a graph using a Java Applet.[2]

Flash was developed by Adobe and is a multimedia platform to add all kinds of multimedia to web pages. It enables the use of videos, animations and interactivity. Flash software is written in a programming language called ActionScript. Flash is featured on many web pages but in order to consume this content a Flash Player has to be installed on the user's PC. The Flash Player is free but the software to develop Flash is not. Florio [2008]

[1] http://java.sun.com/applets/

[2] http://java.sun.com/applets/jdk/1.4/demo/applets/GraphLayout/example1.html

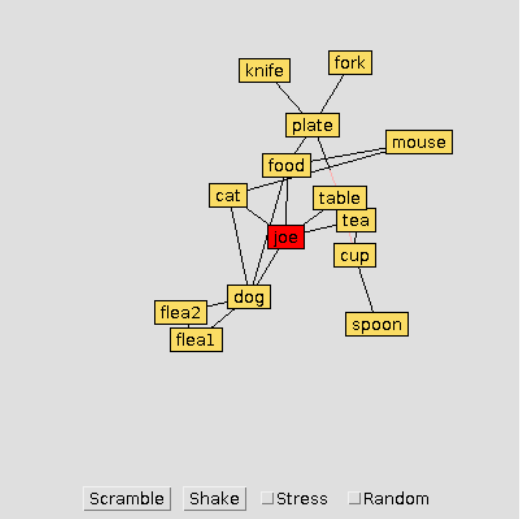

Figure 1.1: Java Applet

1.3 JSON

JSON or JavaScript Object Notation is a data transfer format that is used in many different programming languages but mostly in JavaScript. It is similar to XML but there are some differences. XML uses tags to separate data, JSON on the contrary uses key and value pairs. One of the main advantages of such data transfer formats is that they are easy to read for for humans. The parsing by a computer is also very simple. JSON is language independent but bases it's implementation on JavaScript. The two basic key structures of JSON are that of a collection of name and value pairs and that of an ordered list like an array. Based on the fact that these two structures are universal data structures almost all programming languages can handle JSON. Zakas [2010]

JSON consists of a few different forms. There are JSON objects, arrays, values, strings and numbers. The basic JSON object starts with an opening curly brace and ends with a closing curly brace. A JSON object can be empty or consist of members. Members consist of one or more pairs. A pair is a string, a ":" and a value. A value can be a string, a number, an object, an array, true, false or null. That is basically the concept behind JSON. A very simple example of a JSON object: `{"key" : "value"}`.

1.4 Overview 2D Graphics

1.4.1 SVG

SVG is a graphic format and stands for scalable vector graphics. It is a W3C standard and can be used it together with other standards of the W3C. Therefore the direct embedding of SVG graphics in a HTML page using the HTML5 svg tag is possible. SVG uses vector images which have a number of advantages over bitmap graphics. The scaling works well even if you zoom in very closely and the file size compared to that of a bitmap graphic is very good. Furthermore SVG renders colors more accurate on different machines than bitmap graphics. A few other interesting points are that it is text based, written in XML and can even be animated. Pearlman [2003]

1.4.2 HTML5 Canvas Element

The Canvas element is a new feature in HTML5. It is a rectangular field that enables you to manipulate with it in a wide range of possibilities. Using JavaScript you can draw, add simple shapes, manipulate them and even create animations. HTML5 Canvas does not use vector graphics which means that zooming in and out like in SVG is not possible and the elements on the canvas are not part of the pages DOM. But it performs fast and the API provides a wide variety of features to work with. Lubbers [2010]

Chapter 2

Scalable Vector Graphics

2.1 About SVG

Scalable Vector Graphics abbreviated as SVG is a graphics format based on XML grammar. It has benn developed by the World Wide Web Consortium (W3C) since the late 90's. In SVG a markup language is used to describe two-dimensional graphical objects. W3C [2011c,d]

In the context of SVG "scalable" means that it is possible to uniformly scale an SVG graphic to any desired screen resolution. "Vector" means that a series of geometric shapes is used to describe a picture or scene. In SVG it's also possible to mix standard bitmap graphics and vector graphics thus the term "graphics" should give a hint on this feature, W3C [2011d]. SVG is based on XML which is the abbreviation for *Extensible Markup Language* also developed by the W3C, W3C [2011a]. The SVG specification also permits the use of *Cascading Style Sheets* (CSS) to be used for styling vector graphic elements, W3C [2011d].

2.2 History of SVG

In 1998 Adobe Systems and Sun Microsystems proposed a new vector graphics standard called *Precision Graphics Markup Language* (PGML). This standard defines an XML based language for representing vector graphics. It was submitted to the W3C but never got adopted as a recommendation, Wikipedia [2011a]. At the same time Autodesk, Hewlett-Packard, Macromedia, Microsoft and Visio also proposed and submitted their own standard for vector graphics called "Vector Markup Language". VML is also an XML based language for defining vector elements. It was also rejected by the W3C, Wikipedia [2011c]. Instead of accepting one of those suggested standards the W3C decided to create their own standard, SVG. The first recommendation SVG 1.0 entered the final stage in September 2001. In 2003 an update to the standard was released. In that updated the previous standard was modularized in order to allow implementers only to support a subset of features. Those subsets were called "Tiny", "Basic" and "Full". In 2008 SVG 1.2 reached the recommendation state. In 2010 the second edition of SVG 1.1 was released. It did not contain any new features but instead many clarifications. Currently SVG 2.0 is in development, Wikipedia [2011b].

2.3 SVG and HTML5

The HTML5 standard permits the use of the `<svg>` - element which belongs to the SVG namespace. According to the specification of HTML5 the `<svg>` element belongs to the family of tags containing embedded content, W3C [2011b]. All tags of this family import resources into the document, Bruce Lawson [2011]. The HTML5 specification does not contain any processing rules regarding to the way of handling SVG content. How SVG elements are inserted into the DOM tree and how they should be dealt with is part of the SVG specification. With other words the entire semantic of the `<svg>` element is defined in the SVG specification and

not in the HTML5 specification W3C [2011b].

2.4 Embedding SVG in HTML5 documents

There are several tags available for embedding SVG graphics in HTML documents. In previous versions of HTML there were 4 tags which provided this function. The <embed>, <object>, <img> and <iframe> tag. Each one has its advantages and disadvantages. The <embed> requires a plug-in to be installed and can not be used if the target document needs to be XHTML conform. The use of the <object> tag does not require a plug-in and also does not break XHTML conformity but SVG graphics will not be displayed if the plug-in for the <embed> tag is already installed (at least in Internet Explorer). The best way to embed SVG graphics in HTML4 documents is to use the <img> or <iframe> tag. This works in browsers which support SVG, W3C [2011e]. In HTML5 SVG can be embedded by in-lining the SVG XML code inside the HTML document. This can be done with the help of the previously mentioned new <svg> tag. Documents which contain an <svg> - element must be considered valid even the browser does not support SVG graphics. Sub elements of <svg> must be declared to be part of the SVG namespace identified by "http://www.w3.org/2000/svg".

The next code snippet shows how an HTML5 document with in-lined SVG may look:

```
<!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>inline SVG in HTML5</title>
  </head>
  <body>
    <svg id="svgelem"
         height="220"
         width="220"
         xmlns="http://www.w3.org/2000/svg">
         <!-- ... svg content goes here -->
    </svg>
  </body>
</html>
```

The <svg> element contains 4 attributes to tell the browser how to deal with the defined <svg> element. The *id* attribute assigns a document unique id to the <svg> element. The *width* and *height* attribute defines the dimensions of the SVG viewport and the *xmlns* attribute defines the namespace to use as default for all child elements of the <svg> element.

2.5 SVG Elements and Functions

In SVG geometric primitives and basic shapes are used to describe a picture. In this section the basic elements will be explained and a practical examples of drawing them on the SVG viewport will be demonstrated. All examples in this section are taken from Watt [2011].

Primitives and basic shapes. One of the most basic elements is a simple line that connects a start-point with an end-point. The following code sample shows how to draw a line in SVG:

```
<line x1="50" y1="20" x2="300" y2="300"
      style="stroke:black;"/>
```

The <line> - element contains attributes that define the start point (x1,y1), the endpoint (x2,y2) and a style attribute to define the line style. Here only the color for the line is defined using the *stroke* keyword. Notice the x-axis is from left to right and the y-axis from top to bottom so the root point (0,0) is in the upper left corner.
Drawing a rectangle can be accomplished using 4 <line> elements but there is a simple <rect> element which ease this job.

```
<rect x="10" y="10" width="50" height="25"
      style="stroke:black; fill:none;"/>
```

In this example the <rect> element is used, the upper left corner of the rectangle is defined with (x,y), the dimensions of the rectangle are set using the "width" and "height" attributes. The "style" attribute defines the line color using the keyword "stroke" and with keyword "fill" the rectangle is defined to be empty, the only the surrounding line is displayed.
Drawing a triangle can be accomplished in several ways. The naïve way would be to use 3 <line> elements. But there is also a <polyline> element which can be used.

```
<polyline points="50,50 100,100 0,100 50,50"
          style="stroke:black; fill:none;"/>
```

The "points" attribute contains a series of coordinate tuples which represent the points that should be connected. The line that is drawn starts at the first point and is connected with the next point in the series and so on. Again the "style" attribute defines line color and prevents to fill the resulting surface with the default color. A second way to draw a triangles using the <polygon> element.

```
<polygon points="50,50 100,100 0,100"
         style="stroke:black; fill:none;"/>
```

The difference between these two approaches is that the <polyline> element requires 4 points, the <polygon> element only three because the shape is automatically closed. In the series of coordinates the last point is automatically connected with the start point again.

There are several other elements which are available to draw basic shapes for example <circle>, <ellipse> or <path> element.

2.6 Styling, Scripting and Animations with SVG using CSS and JavaScript

One way of styling shapes is the "style" attribute that has been used in previous examples. Another way of styling elements is to use cascading style sheet (CSS) definitions that can be defined in the header of the HTML document or in an external CSS file and referenced using a <link> element. The following code snippet shows an example of using CSS to style an element Watt [2011]:

```
<style type="text/css">
  blackTriangle { stroke: black; fill: none;}
</style>
<polygon id=" blackTriangle " points="50,50 100,100 0,100"/>
```

Here an "id" was assigned to the <polygon> element. Using id's makes it possible to use JavaScript to obtain SVG elements using the document.getElementById() - function. Further it is possible to change certain style attributes using JavaScript. It is also possible to register event listeners directly on an SVG element, shown in the next code sample, Watt [2011]:

```
<polygon points="50,50 100,100 0,100"
         style="stroke: black;"
```

```
3          onclick="alert('you clicked me');"/>
```

Running this sample code and clicking on the triangle will pop up a message box. Other listeners like mouse over, mouse out and others can be registered as well. In combination with the setTimeout JavaScript function also animations can be implemented. Watt [2011]

For information visualization in the web browser there exists various frameworks based on SVG and JavaScript. In this document a glimpse on two of them is given. The Google Chart Tools framework which is an open source framework developed and maintained by Google and the Protovis framework which is developed and maintained by the Stanford Visualization Group. Other frameworks like Highcharts JS or Raphaël are also based on SVG and JavaScript but are not discussed in this paper.

2.7 The Google Chart Tools

The Google Chart Tools provides an API for visualizing data on web sites. It uses the SVG tag provided by HTML5 and VML for older Internet Explorer versions. It works on mobile devices like iPhone and iPad as well as on devices based on Android without any plug-ins. The look and feel of all charts can be customized using CSS. Different types of data sources can be used to obtain data that should be visualized.

Simple example of a Line Chart created with Google Chart Tools. The following code demonstrates how to use the Google Chart Tools to display a simple line chart (see Figure 2.1):

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5      <!-- include Google APi -->
6      <script type="text/javascript"
7              src="https://www.google.com/jsapi"></script>
8      <script type="text/javascript">
9
10       // load visualization module
11       google.load("visualization", "1", {packages:["corechart"]});
12
13       // register event listener, catches page load event
14       google.setOnLoadCallback(drawChart);
15
16       // defines event listener
17       function drawChart() {
18         // creates basic data source
19         var data = new google.visualization.DataTable();
20
21         data.addColumn('string', 'Year');    // define column of table
22         data.addColumn('number', 'Revenue'); //
23
24         data.addRows(5); // define count of data rows
25
26         // add data points to data table
27         // paramaters: row, cell, value
28         data.setValue(0, 0, '2007');
29         data.setValue(0, 1, 1000);
30         data.setValue(1, 0, '2008');
31         data.setValue(1, 1, 1170);
32         data.setValue(2, 0, '2009');
33         data.setValue(2, 1, 860);
```

```
34          data.setValue(3, 0, '2010');
35          data.setValue(3, 1, 1030);
36          data.setValue(3, 0, '2011');
37          data.setValue(3, 1, 2030);
38
39          // create chart object
40          var chart =
41              new google.visualization.LineChart(document.getElementById('
                chartContainer'));
42
43          var config = new Object(); // create chart configuration
44
45          config.width = 600;  // define dimensions of graph
46          config.height = 340; //
47
48          // define graph title and axis lables
49          config.title = 'Venture revenue chart';
50          config.vAxis = { title: 'in million $'};
51          config.hAxis = { title: 'Year'};
52
53          chart.draw(data, config); // init graph drawing
54        }
55      </script>
56    </head>
57  <body>
58  <!-- define container that will contain graph -->
59  <div id="chartContainer"></div>
60  </body>
61  </html>
```

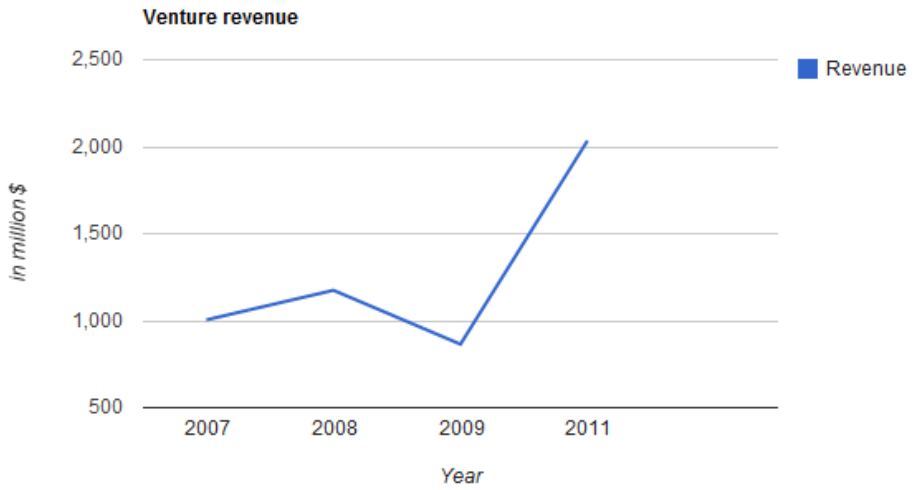

Figure 2.1: Line chart.

The Google Chart Tools API is well defined and clearly structured therefore it is easy to switch from one chart type to another (see Figure 2.2) without much code effort. For example to switch from a line chart to a bar chart just one line of code has to be changed. Replace line 34 in the previous sample with the following:

```
1  var chart = new
2  google.visualization.BarChart(document.getElementById('chartContainer'));
```

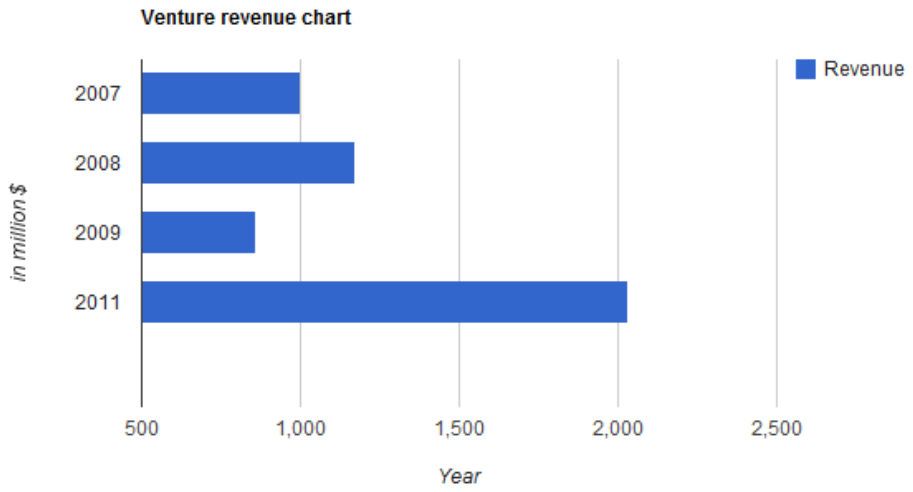

Figure 2.2: Bar chart.

The Google Chart Tools API also provides some hooks to register event listeners. Event listeners are functions that are called when the user interacts with the chart. The previous sample can easily be modified to demonstrate the use of event handlers. At the end of the “drawChart” function the following line has to be appended:

```
1    google.visualization.events.addListener(chart, 'select',
2        function() { alert('you clicked me'); });
```

The ”addListener” function registers a new event listener at the ”chart” which is triggered every time the user clicks on a bar in the chart. The third argument of this function represents the actual event listener which is called. Using event listeners a sophisticated way of exploring rich data sets can be implemented.

Inside the Google Chart Tools there are several different visualization methods available. The basic visualizations are Pier Chart, Line Chart, Area Chart, Treemap, Scatter Chart, Table, Gauge, Bar Cart ad Column Chart. There are also several chart types available which were created by the community. For example Dygraphs, Parallel Coordinates Chart, Area Chart, Candlestick Chart and Sparkline. All of those chart types are public available and can be used free of charge. The API is hosted by Google and can be included with two lines of JavaScript. While the Google Chart Tools uses SVG and VML it works in all major browsers moreover it is an easy to use toolkit for providing data visualizations on web sites.

2.8 SVG in Action: Protovis

Protovis is a framework for information visualization developed by the Stanford Visualization Group. It is based on SVG and JavaScript and provides a wide range of different graph types. It does not require any plug-ins and works in all major modern browsers. Protovis is designed to be a declarative framework. Users who want to create visualizations can do so by describing the graph they prefer with function calls to Protovis Group [2011].

To get a better idea of Protovis a small sample is shown below:

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
```

```
5       <script type="text/javascript" src="protovis-r3.2.js"></script>
6     </head>
7   <body>
8
9   <script>
10
11  // create a new panel which will contain the graph
12  var vis = new pv.Panel()
13      .width(300)
14      .height(250);
15
16  // add vertical axis labels
17  vis.add(pv.Rule)
18      // start at 0 up to 3 in step of .5
19      .data(pv.range(0, 3, .5))
20      .bottom(function(d) { return d * 80 + .5; })
21     .add(pv.Label);
22
23  // add bar's
24  vis.add(pv.Bar)
25      // add data points
26      .data([1, 2, 1.5, 2, 2, 1])
27      .width(40) // width of bar
28      // define function to calculate height of bar
29      .height(function(d) { return d * 80; })
30      // bottom offset of each bar
31      .bottom(0)
32      // location of lower-left point of each bar
33      .left(function() { return this.index * 45 + 25 })
34      // assign some nice color's ...
35      .fillStyle(pv.Colors.category20().by(pv.index));
36
37  // add char label
38  vis.add(pv.Label)
39      .left(120) // 120 px from left border
40      .top(24) // 24 px from top border
41      .textAlign("center") // center text
42      .font("20px sans-serif") // font style
43      // content of text label
44      .text("Revenue in million");
45
46  // render and display the graph
47  vis.render();
48
49  </script>
50
51  </body>
52  </html>
```

The result of this sample is shown in Figure 2.3

Protovis supports a rich set of styling functions and thus can be adapted to many look and feels. Compared to Google Chart Tools it has the advantage to be downloadable. Google's Terms of Service prohibits downloading the Google Chart Tools thus to use it the client needs to be connected to the internet Google [2011].

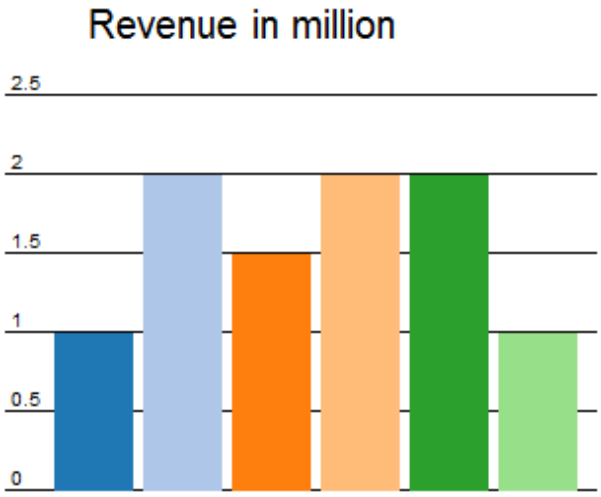

Figure 2.3: Protovis sample result.

Chapter 3

HTML5

3.1 An Introduction to the Standard

By now the HTML5 standard is not more than a draft but in 2012 there is a first candidate recommendation with a complete specification. Although the draft will be released in over a year, browser vendors already add the new HTML5 functionality to their browsers. There are two major organisations working on the standard. First, the *Web Hypertext Application Technology Working Group* (*WHATWG*) which is a group of all major browser vendors and the *World Wide Web Consortium* (*W3C*) as international standardisation group for the World Wide Web.

HTML5 should not be a revolution but an evolution and so it is not necessary to reinvent the wheel but make a better wheel instead. For instance, HTML4 lacks of security in several points and HTML5 tries to bring security already from the starting point. Nevertheless, programmers are responsible to prevent from cross site scripting (XSS) attacks and other code injections.

All new programmer's interfaces should be powerful but yet simple which requires an accurate standardization process that prevents from misinterpretation. Finally, the presentation of webpages written in HTML5 should be independent of the used devices. This implies that users do not need any browser plugins for video or data visualization and can use the native functionality of their browsers. As depicted in Table 3.1 most modern webbrowsers support the new features of HTML5.

3.2 Feature Space

HTML5 introduces several new features and possibilities for programmers and in this section we describe the most important ones.

Browser Name	Basic	Text API	CSS	3D
Chrome 11.0	Yes	Yes	Yes	Partially
Firefox 4.0	Yes	Yes	No	Partially
Internet Explorer 9.0	Yes	Yes	No	No
Opera 11.1	Yes	Yes	No	No
Safari 5.0	Yes	Yes	Yes	No

Table 3.1: Comparion of current browser version and their HTML5 Support. Deveria [2011]

Video tag. The new video tag supports a native display of non-proprietary video-formats in a webpage. An earlier version of the standard draft states Ogg-Theora as video format but due to problems with licensing the actual format of the video tag is not specified any more, Pfeiffer and Parker [2009]. Possible other formats are H.264, MPEG4 and WebM.

Geo location. To support location-based web applications the HTML5 standard introduces a geo-location API that allows programmers to determine the actual geo-spatical location of a client. This can be used to provide additional services to customers based on their geo-spatial location, Vaughan-Nichols [2010]. In contrast to mobile devices with a GPS sensor on-board for home computers only the IP address of the machine can be used to determine a user's region. To refine this location the web browser can send a list with all available WIFI networks to a special service. If the geo-location of one of these WIFI Networks can be identified, the accurate position of the user's computer can be determined, Netzwelt.de [2011].

Offline applications. By now most web applications require a permanent connection to the Internet whereas HTML5 allows programmers to write applications for offline use. The user is required to have an internet connection only during the start of the application but can use the service even if there is no internet connection available any more. Meanwhile, data is either stored in a client-sided SQL database or simple caching mechanisms are used. As soon as the user is online again all data is synchronized with the web server. Examples for this new features are mail-clients or web-based office applications. Vaughan-Nichols [2010]

Canvas tag. One of the most important features of HTML5 is a new element that allows programmers to present graphics, images, and text-elements to their users. All rendering and computation is done on the client and therefore does not consume computational power on the server side, Vaughan-Nichols [2010].

3.3 Canvas HTML5 API

3.3.1 The Canvas element

There are several popular visualization frameworks like Microsoft's Silverlight or Adobe's Flash. Although, these solutions are powerful have an important disadvantage as they require browser plugins which are probably not available on some platforms like mobile devices. In most cases these plugins are proprietary solutions and the performance of the visualization highly depends on the vendor's implementation on a certain platform. The canvas tag in the HTML5 standard aims at providing all visualization features of existing solutions in a native way without any browser extensions or plugins. As mentioned in Section 3.1 the specification of HTML5 and in particular the Canvas tag is still a working draft but most of modern web-browsers support basic features of the canvas element.
Compared to SVG (see Chapter 2) Canvas is a bitmap image because elements are not part of the *Document Object Model* (*DOM*) tree. The objects within a Canvas element are final and so resizing is not seamless but requires a recomputation of the entire element. On the one hand this implies that simple objects like lines or rectangles are not selectable but on the other hand processing is in general faster. Lubbers et al. [2010]

The access to the new HTML5 programming interfaces should be as simple as possible and so programmers can easily create and modify an element by using HTML and JavaScript. The Canvas element is part of the *DOM* tree of an HTML document and is identified by a unique ID.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>Canvas Demo</title>
5 </head>
6
```

```
<body>
  <canvas id="sampleCanvas" style="border: 3px solid"> </canvas>
</body>
```

In order to modify the element with id *sampleCanvas* and draw simple objects, *i.e.* rectangles, onto it programmers have to use JavaScript. In the following code snippet the 2D context is extracted from the Canvas element by using the functions *getElementByID(...)* and *getContext('2d')*. Now, the methods of the context class can be used to actually draw primitive shapes like lines or circles, other images, or even text.
In the code snippet we use the method *rect(...)* to create a rectangle and finally *stroke(...))* to draw it onto the canvas.

```
<script type="text/javascript">
  var canvas = document.getElementById("sampleCanvas");
  var context = canvas.getContext('2d');

  context.rect(250, 50, 100, 100);    // Draw a rectangle
  context.stroke();
</script>
```

HTML5 does not allow to register handlers with particular objects but only with the Canvas element. These object listeners can be registered with the `addEventListener(...)` method where *mousemove* is the event name and *mouseMoveEvent* is the name of the JavaScript function triggered upon the event. An example that traces the mouse movement and draws the trace of the mouse pointer is presented below.

```
<script type="text/javascript">
  var canvas = document.getElementById("sampleCanvas");
  canvas.addEventListener("mousemove", mouseMoveEvent, false);

  function mouseMoveEvent(e) {
    context.lineTo(e.pageX, e.pageY);
    context.stroke();
  }
</script>
```

The full reference with all methods of the canvas element can be found in `http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html` but here is an overview of the most important commands.

- Stroke and fill styles. The stroke style of primitive elements can be modified by using *Cascading Style Sheets*, CSS. This includes the line width, the color, and the shape of a line joining. Further, one can color closed shapes with either solid color or even with a color gradient.

- Complex Shapes. Besides primitive shapes one can create groups of objects, referred to as paths. Within the context of paths, quadratic- and cubic bezier curves, arcs, and lines can be created.

- Images and background. Simple images can be added to the canvas element with the method *canvas.drawImage(...)*.

- Canvas transformations. Objects of a path can be rotated, sheered, and scaled with linear matrix manipulations.

- Text. Text objects can be added to the canvas element with the method *filltext(...)* and parameters for the actual text and the coordinates of the text within the canvas element. There are limited ways of formatting the text by using the methods *font(...)*, *fillStyle(...)*, and *textAlign(...)*.

- Shadow. The Canvas API provides functions to add shadow to the objects by using *shadowColor(...)* that specifies the color, *shadowOffsetX(...)* and *shadowOffsetY(...)* for the orientation of the shadow, and finally *shadowBlur(..)* to blur the shadow.

3.4 Existing Frameworks

The HTML5 canvas element was designed as a simple but powerful API that allows creating simple and complex objects, images, and even text without external libraries or browser extensions. Programmers do not need to reinvent the wheel to visualize the data but can use existing frameworks that abstract functions to create diagrams and well known data representations. In this section the most popular ones will be described and compared upon their functionality.

3.4.1 JavaScript InfoVis Toolkit

This JavaScript framework focuses on information visualization and is written by Nicolas Garcia Belmonte from SenchaLabs. It provides an easy-to-use interface for the most common visualization graphs like area-, pie-, bar-, sunburst-charts, tree maps, hyper trees, and space trees. Programmers can easily create charts by instantiating the required diagram and load JSON data to be represented. SenchaLabs [2011]

```
1  var areaChart = new $jit.AreaChart({injectInto: 'id_of_an_arbitrary_div'});
2  areaChart.loadJSON(jsonData);
```

This basic diagram can be modified and customized with additional parameters. The customization options include color, line style, labels, and tips and simple user interaction.

3.4.2 jQuery Sparklines

Is is a free-of-charge plugin for the JavaScript Framework jQuery to create simple visualization diagrams. It comprises line diagrams, bar charts, box plots, pie charts, and bullet charts. The framework is easy-to-use and diagrams can be created with only a few lines of code. Inc. [2011]

```
1  var data = [10, 8, 5, 7, 4, 4, 1];
2  $('id_of_an_arbitrary_div').sparkline(data, {type: 'bar'});
```

A programmer can specify the layout of the chart, change fill colors and line colors, and automatically update the chart. The created diagrams are not interactive but can only be shown to the users statically.

3.4.3 Flot

A visualization library released under the MIT license based on the jQuery JavaScript library. It only supports line digrams, point diagrams and bar charts. Although the range of diagrams is very limited users can turn on and off data series, zoom into the diagrams, interact with datapoints, and track the curves with crosshair. Due to the simplicity of the solution programmers can plot data with a few lines of JavaScript code. Laursen [2011]

```
1  var data = [[0, 3], [4, 8], [8, 5], [9, 13]];
2  $.plot($("#id_of_an_arbitrary_div"), data);
```

Flot is mainly developed by Ole Laursen but supported by the community.

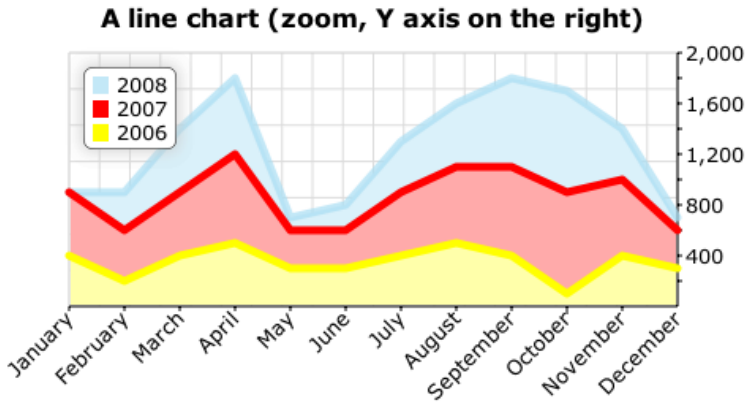

Figure 3.1: A line diagram created by RGraph.

3.4.4 jqPlot

This visualization tool is implemented as a plugin for the jQuery library and supports different types of diagrams like line charts, bar charts, pie- and donut charts, and bubble plots. The programming interface is easy-to-use and diagrams can be created with a few lines of code. Leonello [2011]

```
1  var data = [[3, 7, 9, 1, 4, 6, 8, 2, 5]];
2  var plot1 = $.jqplot('id_of_an_arbitrary_div', data);
```

jqPlot is an open source project by Chris Leonello and enhances pure display of data with trendlines, dragging and dropping of datapoints and log-scales.

3.4.5 RGraph

This is probably the most advanced chart library for the canvas element. It is free of charge for non-commercial use but costs 49 GBP for commercial users. It supports bar charts, donut charts, funnel charts, Gantt charts, odometer charts, pie charts, rose charts, radar charts, and many more. Users can interact with the diagrams and select particular data points or zoom into diagrams. Diagrams are customizable with tooltips, different labels and variable scales (see Figure 3.1). RGraph [2011]

The following example is taken from RGraph's Webpage and shows the simplicity of creating a line diagram with customized colors, lines, and labels.

```
1  var data = [10, 4, 17, 50, 25, 19, 20, 25, 30, 29, 30, 29];
2
3  var line = new RGraph.Line("myLine", data);
4  line.Set('chart.background.barcolor1', 'rgba(255,255,255,1)');
5  line.Set('chart.background.barcolor2', 'rgba(255,255,255,1)');
6  line.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
7  line.Set('chart.colors', ['rgba(255,0,0,1)']);
8  line.Set('chart.linewidth', 2);
9  line.Set('chart.filled', true);
10 line.Set('chart.hmargin', 5);
11 line.Set('chart.labels', ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
12                           'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']);
13 line.Set('chart.gutter', 40);
14 line.Draw();
```

Chapter 4

Conclusion

It this chapter the performance of the two above discussed web-based visualization technologies will be compared.

4.1 Studies on Performance

In order to complete this task two different studies will be considered. One of them has been done at Mississippi State University and has enumerated the strengths and weaknesses of web-native information visualization technologies such as SVG, HTML5, and pure HTML and compared their performance to the web-embedded Java application, Johnson and Jankun-Kelly [2008]. Another study has been performed by a volunteer from Australia with the aim to benchmark performance of different technologies for animation. Although this study cannot be called scientific, it is still important for this comparison since smooth-moving, nice animated information visualization becomes more and more interesting and performance of web-native and web-embedded technologies in aspect animation is an essential factor in the technology choice. Moreover, this is an independent study with available source code to download. The animation study has compared performance of SVG, HTML5, pure HTML and Java, Adams [2010]. It is also important to mention that the study of Mississippi University has been implemented on Mac OS X 10.5.2 and Windows Vista using Firefox 2.0.0.12 and Safari 3.0.4, whereas the volunteer study has been carried out on Mac OS X and Windows XP using browsers Safari 4.0.3, Firefox 3.6, Chrome 4.0.249.89 and Internet Explorer 8.

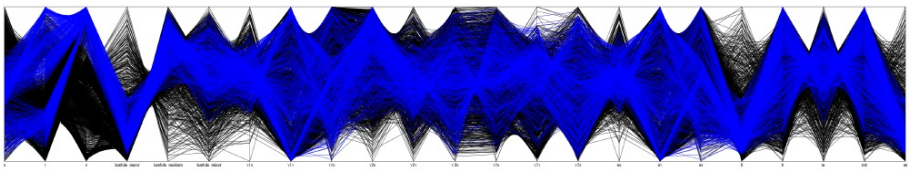

Figure 4.1: Canvas parallel coordinates implementation

Study of Mississippi University has performed two tests, where each technology loaded a dataset, layout and display the data and then react upon the random manual selections of the user.

The performance times for each of these activities was recorded whenever possible for both parallel coordinates (see Figure 4.1) and squarified treemap information visualization (see Figure 4.2) All the implementations have been tested on the datasets of small, medium and large sizes (see Figure 4.3).

The volunteer study implemented a particle engine animation which can be translatable among all four technologies as they roughly use the same animation techniques, calculations or timers. Instead of using different datasets the number of particles has been varied from 250 up to 4000. To demonstrate the layout performance the framerate was considered as comparison parameters (see Figure 4.4).

Both studies kept their implementations similar across all the architectures in order to test the technologies and not the coding ability of the programmers.

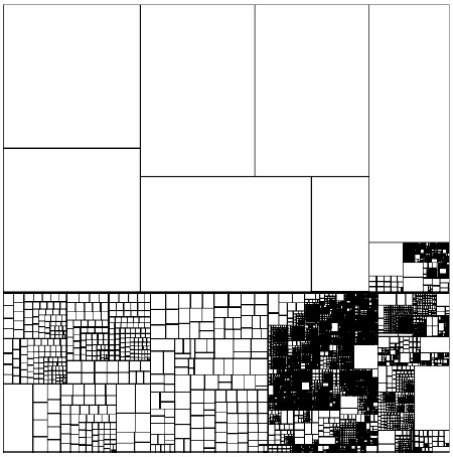

Figure 4.2: SVG treemap implementation (Medium dataset)

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

Figure 4.3: Datasets for information visualization tests

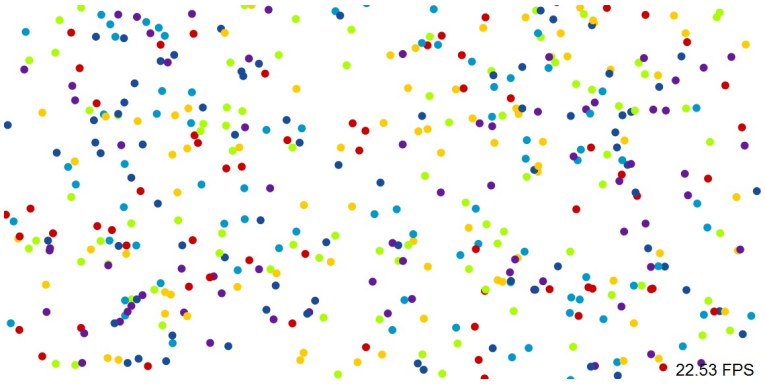

Figure 4.4: Particle animation benchmarking

4.2 Results

Observing all of the results, most attention will be paid to SVG and HTML5 performance. In both studies the different architectures did not differ in the actual visualization but only in the performance. Unsurprisingly, Java and Flash as mature technologies with access to hardware acceleration have generally performed better even though they are not web-native. In the study of Mississippi University SVG performed generally next best after Java and was better than HTML5 Canvas until very large dataset size (see Figure 4.5). Nevertheless, with increasing complexity of the SVG document, as it is in case of treemap datasets, HTML5 Canvas performed better (see Figure 4.6).

Talking more specifically SVG and HTML5 Canvas had similar performance in terms of load, whereas SVG

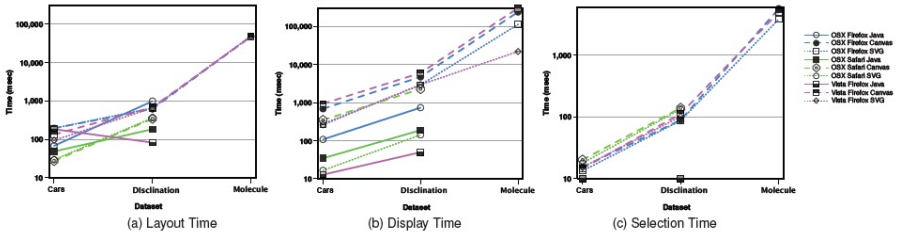

Figure 4.5: Timing charts for the Parallel Coordinates Renderers. All times in msecs; charts use different log-scales

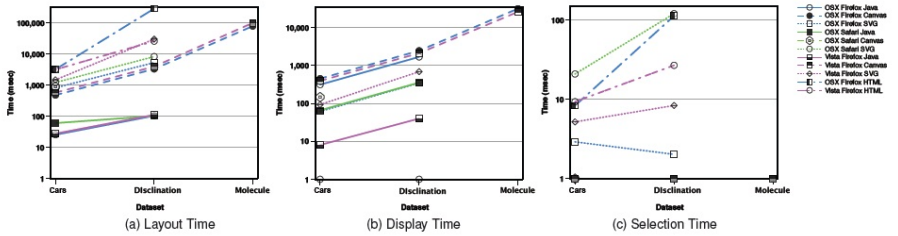

Figure 4.6: Timing charts for the Treemap Renderers. All times in msecs charts use different log-scales

performed much better on display and HTML5 Canvas was mostly ahead on layout. Selection performance has been also equal. In terms of interactivity, if 100ms is an upper threshold for "interactive" performance, web-native rendering was interactive only for the smaller datasets. Pure HTML has demonstrated the worst performance through all the tests, [Johnson and Jankun-Kelly, 2008]. In contrast, the volunteer study shows that HTML5 canvas is the next best after Flash, pure HTML follows it and SVG performs worst in particle animation benchmarking (see Figure 4.7).

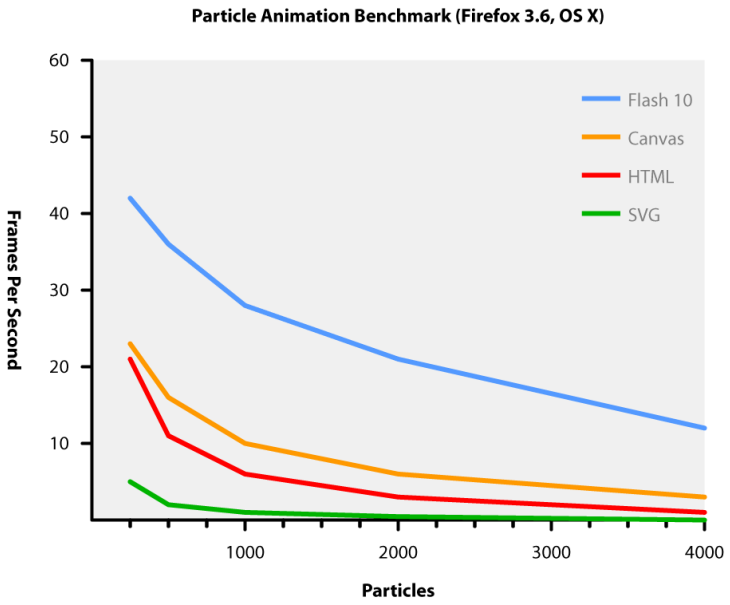

Figure 4.7: Animation benchmarking results

The poor performance of SVG can be explained by the fact that it stores the entire document structure (the vector commands) in order to generate the graphics. For the simple animations in the experiment this approach is inefficient.

4.3 Conclusions

Web-native display technology has the potential to expand the impact of visualization to the areas where program applets in Java and Flash are not available. Nevertheless, the limitations of these technologies must be fully understood. While web-embedded technologies have the best performance, SVG has an adequate interactivity for medium or smaller datasets. However, neither SVG nor Canvas are perfect if large data visualization are combined with interactive performance.

References

Adams, Cameron [2010]. *HTML5 versus Flash: Animation Benchmarking*. http://www.themaninblue.com/writing/perspective/2010/03/22/.

Bruce Lawson, Remy Sharp [2011]. *Introducing HTML5*.

Deveria, Alexis [2011]. *When Can I Use*. http://caniuse.com.

Flanagan, David [2006]. *JavaScript, The Definitive Guide*. First Edition. O'Reilly, 1–9 pages.

Florio, Chris [2008]. *ActionScript 3.0 for Adobe Flash CS4 Professional Classroom in a Book*. First Edition. Adobe Press, 9–10 pages.

Google [2011]. *Google Chart Tools - Frequently Asked Questions*. http://code.google.com/apis/chart/interactive/faq.html#localdownload.

Group, Stanford Visualization [2011]. *Protovis - a graphical approach to visualization*. http://vis.stanford.edu/protovis/.

Inc., Splunk [2011]. *jQuery Sparklines*. http://omnipotent.net/jquery.sparkline/.

Johnson, Donald W. and T.J. Jankun-Kelly [2008]. *A Scalability Study of Web-Native Information Visualization*. http://portal.acm.org/citation.cfm?id=1375743.

Laursen, Ole [2011]. *Attractive Javascript plotting for jQuery*. http://code.google.com/p/flot/.

Leonello, Chris [2011]. *Pure JavaScript Plotting*. http://www.jqplot.com/index.php.

Lubbers, Peter [2010]. *Pro HTML5, Powerful APIs for Richer Internet Application Development*. First Edition. Apress, 1–2,25–26 pages.

Lubbers, Peter, Brian Albers, Ric Smith, and Frank Salim [2010]. *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*. 1st Edition. Apress, Berkely, CA, USA. 1430227907, 9781430227908.

Netzwelt.de [2011]. *Location Based Services: Wenn der eigene Standort mit dem Netz verknÃ¼pft wird*. http://www.netzwelt.de/news/84819_2-location-based-services-eigene-standort-netz-verknuepft.html.

Pearlman, Ellen [2003]. *SVG for Web Developers*. First Edition. Prentice Hall, 1–9 pages.

Pfeiffer, Silvia and Conrad Parker [2009]. *Accessibility for the HTML5 video element*. In *Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*, pages 98–100. W4A '09, ACM, New York, NY, USA. 978-1-60558-561-1. doi:http://doi.acm.org/10.1145/1535654.1535679. http://doi.acm.org/10.1145/1535654.1535679.

Raggett, Dave [1998]. *Raggett on HTML4*. Second Edition. Addison-Wesley, 17–19 pages.

RGraph [2011]. *RGraph: HTML5 canvas graph library*. http://www.rgraph.net/.

SenchaLabs [2011]. *JavaScript InfoVis Toolkit*. http://thejit.org/.

Vaughan-Nichols, S.J. [2010]. *Will HTML 5 Restandardize the Web? Computer*, 43(4), pages 13 –15. ISSN 0018-9162. doi:10.1109/MC.2010.119.

W3C [2011a]. *Extensible Markup Language (XML)*. http://www.w3.org/XML/.

W3C [2011b]. *HTML5 - A vocabulary and associated APIs for HTML and XHTML*. http://dev.w3.org/html5/spec/Overview.html.

W3C [2011c]. *SCALABLE VECTOR GRAPHICS (SVG)*. http://www.w3.org/Graphics/SVG/.

W3C [2011d]. *SVG 1.1 (Second Edition)*. http://www.w3.org/TR/SVG/concepts.html.

W3C [2011e]. *SVG In HTML Pages*. http://www.w3schools.com/svg/svg_inhtml.asp.

Watt, Andrew H. [2011]. *Designing SVG Web Graphics*.

Wikipedia [2011a]. *Precision Graphics Markup Language*. http://en.wikipedia.org/wiki/Precision_Graphics_Markup_Language.

Wikipedia [2011b]. *Scalable Vector Graphics*. http://en.wikipedia.org/wiki/Scalable_Vector_Graphics.

Wikipedia [2011c]. *Vector Markup Language*. http://en.wikipedia.org/wiki/Vector_Markup_Language.

Zakas, Nicholas C. [2010]. *High Performance JavaScript*. First Edition. O'Reilly, 137–140 pages.