Introduction

The maturation of web browsers has reached the point where it is easy to use native vector graphics without any third party plugins and installs. SVG is supported in all major modern browsers except for IE, which supports VML, a slightly-limited precursor to SVG. Despite maturity of SVG in web browsers there are still gaps in support. Raphaël JavaScript Library (http://raphaeljs.com) is a small JavaScript library that uses SVG as a base. Raphaël equalizes the differences between browsers and emulates the features and benefits of SVG in Internet Explorer with VML. Raphaël enables front-end developers to use vector graphics in a cross-browser compatible way with only a single source to maintain.

Raphaël can be used to create rich interactive graphics, creating vector elements that are native to the browser and the DOM. Native DOM elements are exposed to JavaScript, allowing for them to be used in the same manner as any other HTML element. Raphaël can attach JavaScript handlers, events, animations, and other features native to JavaScript to SVG/VML elements created in Raphaël. Raphaël works in IE6+, FF3+, Safari 3+, and Opera 9.5+.

jQuery (http://www.jquery.com), a popular JavaScript framework used to simplify HTML document traversing, event handling, animating, and Ajax interactions for rapid web development, can be used on top of Raphaël for additional interactivity and management of SVG in the browser.

This paper will demonstrate how Raphaël works, how it can be used in all browsers, why the world needs another JavaScript or SVG library, limitations of both Raphaël and SVG in the browser, comparisons with existing SVG browser solutions, and a demo of the Raphaël API and its capabilities.

Special Acknowledgments

All of the work in this paper and the accompanying presentation could not have been achieved without the hard-work and prescient work of Dmitry Baranovskiy, the author of the Raphaël JavaScript Library. His website is: http://dmitry.baranovskiy.com. He can be reached at his e-mail address: dmitry@baranovskiy.com.

Raphaël Examples

Hello World - Code

This example illustrates how to use the Raphaël library with the jQuery library. jQuery is not required to use Raphaël, but using the DOM manipulation and selector engine (known as Sizzle) saves a lot of effort for very little cost. This example is located online at: http://raphaeljs.com/text-rotation.html.


```
<script type="text/JavaScript" charset="utf-8" src="raphael.js"></script>
<script type="text/JavaScript" charset="utf-8" src="jquery-1.3.2.js"></script>

<script type="text/JavaScript" charset="utf-8">
<!--
window.onload = function () {
    var R = Raphael("holder", 640, 480);

    var hldr = document.getElementById('holder');
    var textNode = hldr.getElementsByTagName('p')[0].childNodes[0];
    var text = textNode.nodeValue;
    var attr = {font: '50px Fontin-Sans, Arial', opacity: 0.5};
    var mouse = null, rot = 0;

    var txt = [
        R.text(320, 240, text).attr(attr).attr({fill: "#0f0"}),
        R.text(320, 240, text).attr(attr).attr({fill: "#f00"}),
        R.text(320, 240, text).attr(attr).attr({fill: "#00f"})
    ];
    textNode.nodeValue = '';

    $(document).mousemove(function (e) {
        if (mouse === null) {
            mouse = e.pageX;
            return;
        }
        rot += e.pageX - mouse;
        txt[0].rotate(rot, true);
        txt[1].rotate(rot / 1.5, true);
        txt[2].rotate(rot / 2, true);
        mouse = e.pageX;
        R.safari();
    });
};
// -->
</script>
```

Hello World - Output

The output of this code would result in the following:
And as you move your mouse, the words would rotate at different intervals relative to each other:
The equivalent SVG to create the basis for this effect would be, not including the rotation based on mouse position:

```
<?xml version="1.0"?>
<svg width="640" height="480">
       <desc>Created with Rapha&#xEB;l</desc>
       <defs/>
       <text x="320" y="257.5" text-anchor="middle" style="font-family: Fontin-
Sans,Arial; font-style: normal; font-variant: normal; font-weight: normal;
font-size: 50px; line-height: normal; font-size-adjust: none; font-stretch:
normal; -x-system-font: none; opacity: 0.5;" font="50px Fontin-Sans, Arial"
stroke="none" fill="#00ff00" opacity="0.5" transform="rotate(577, 320.201,
240)">
              <tspan>Hello World</tspan>
       </text>
       <text x="320" y="257.5" text-anchor="middle" style="font-family: Fontin-
Sans,Arial; font-style: normal; font-variant: normal; font-weight: normal;
font-size: 50px; line-height: normal; font-size-adjust: none; font-stretch:
normal; -x-system-font: none; opacity: 0.5;" font="50px Fontin-Sans, Arial"
stroke="none" fill="#ff0000" opacity="0.5" transform="rotate(384.667, 320.201,
240)">
              <tspan>Hello World</tspan>
       </text>
       <text x="320" y="257.5" text-anchor="middle" style="font-family: Fontin-
Sans,Arial; font-style: normal; font-variant: normal; font-weight: normal;
font-size: 50px; line-height: normal; font-size-adjust: none; font-stretch:
normal; -x-system-font: none; opacity: 0.5;" font="50px Fontin-Sans, Arial"
stroke="none" fill="#0000ff" opacity="0.5" transform="rotate(288.5, 320.201,
240)">
              <tspan>Hello World</tspan>
       </text>
</svg>
```

Additionally, the equivalent VML is created:

```
<?xml:namespace prefix = rvml ns = "urn:schemas-microsoft-com:vml" />
<rvml:group style="POSITION: absolute; WIDTH: 640px; HEIGHT: 480px; TOP: 0px;
LEFT: 0px; rotation: 107" class=rvml coordsize = "640,480">
    <rvml:shape style="WIDTH: 640px; HEIGHT: 480px; TOP: 0px; LEFT: 0px"
class=rvml coordsize = "640,480" filled = "t" fillcolor = "lime" stroked = "f"
path = " m320,238 l321,238 e">
        <rvml:textpath style="FONT: 50px Fontin-Sans, Arial; v-text-align:
center" class=rvml on = "t" string = "Hello World"></rvml:textpath>
        <rvml:path class=rvml textpathok = "t"></rvml:path>
        <rvml:fill class=rvml type = "solid" opacity = ".5"></rvml:fill>
        <rvml:stroke class=rvml opacity = ".5" miterlimit = "8"></rvml:stroke>
```

```
        </rvml:shape>
    </rvml:group>
    <rvml:group style="POSITION: absolute; WIDTH: 640px; HEIGHT: 480px; TOP: 0px;
    LEFT: 0px; rotation: 71.3333333333333" class=rvml coordsize = "640,480">
        <rvml:shape style="WIDTH: 640px; HEIGHT: 480px; TOP: 0px; LEFT: 0px"
    class=rvml coordsize = "640,480" filled = "t" fillcolor = "red" stroked = "f"
    path = " m320,238 l321,238 e">
            <rvml:textpath style="FONT: 50px Fontin-Sans, Arial; v-text-align:
    center" class=rvml on = "t" string = "Hello World"></rvml:textpath>
            <rvml:path class=rvml textpathok = "t"></rvml:path>
            <rvml:fill class=rvml type = "solid" opacity = ".5"></rvml:fill>
            <rvml:stroke class=rvml opacity = ".5" miterlimit = "8"></rvml:stroke>
        </rvml:shape>
    </rvml:group>
    <rvml:group style="POSITION: absolute; WIDTH: 640px; HEIGHT: 480px; TOP: 0px;
    LEFT: 0px; rotation: 53.5" class=rvml coordsize = "640,480">
        <rvml:shape style="WIDTH: 640px; HEIGHT: 480px; TOP: 0px; LEFT: 0px"
    class=rvml coordsize = "640,480" filled = "t" fillcolor = "blue" stroked = "f"
    path = " m320,238 l321,238 e">
            <rvml:textpath style="FONT: 50px Fontin-Sans, Arial; v-text-align:
    center" class=rvml on = "t" string = "Hello World"></rvml:textpath>
            <rvml:path class=rvml textpathok = "t"></rvml:path>
            <rvml:fill class=rvml type = "solid" opacity = ".5"></rvml:fill>
            <rvml:stroke class=rvml opacity = ".5" miterlimit = "8"></rvml:stroke>
        </rvml:shape>
    </rvml:group>
```

Hello World - Analyzing the JavaScript Code

Decomposing this code, we can see how Raphaël transforms this JavaScript code into SVG/VML.

1) Declare the Raphaël Canvas:

```
    var R = Raphael("holder", 640, 480);
```

2) Declare variables and find DOM nodes:

```
    var hldr = document.getElementById('holder');
    var textNode = hldr.getElementsByTagName('p')[0].childNodes[0];
    var text = textNode.nodeValue;
    var attr = {font: '50px Fontin-Sans, Arial', opacity: 0.5};
    var mouse = null, rot = 0;
```

3) Raphaël allows for vector objects to be placed into JSON/JavaScript data structures and modified through loops:

```
    var txt = [
        R.text(320, 240, text).attr(attr).attr({fill: "#0f0"}),
        R.text(320, 240, text).attr(attr).attr({fill: "#f00"}),
```

```
        R.text(320, 240, text).attr(attr).attr({fill: "#00f"})
    ];
    textNode.nodeValue = '';
```

4) Detect the mouse movement and then rotate each text vector node. jQuery is used here in place of awful cross-browser mouse movement detection:

```
    $(document).mousemove(function (e) {
        if (mouse === null) {
            mouse = e.pageX;
            return;
        }
        rot += e.pageX - mouse;
        txt[0].rotate(rot, true);
        txt[1].rotate(rot / 1.5, true);
        txt[2].rotate(rot / 2, true);
        mouse = e.pageX;
        R.safari();
    });
```

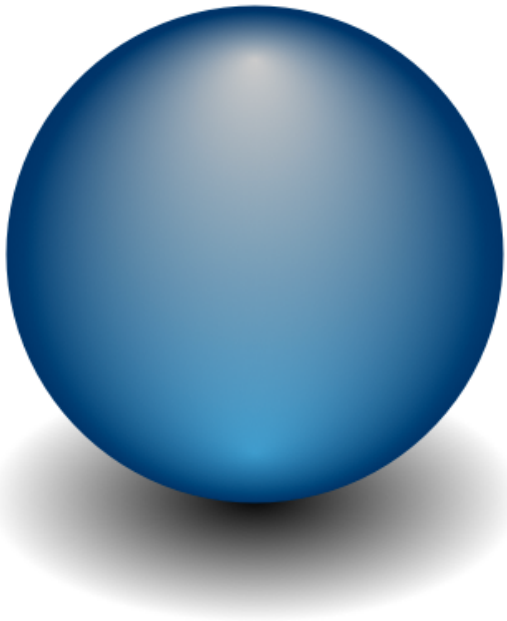
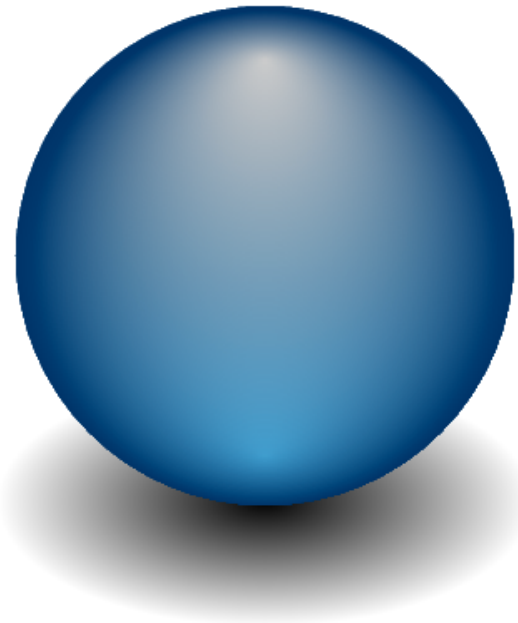
Hello World - Observations

Notice that Raphaël takes data from existing HTML on the page and uses it in generating SVG/VML. This DOM interaction is one of the main benefits in using Raphaël, as it can be used both during initialization and execution. Here, the text node is rotated based on JavaScript's detection of mouse behavior. Additionally, the text node is populated based on existing data. Raphaêl is as much of the DOM as any other element.

Layered Shape - Code

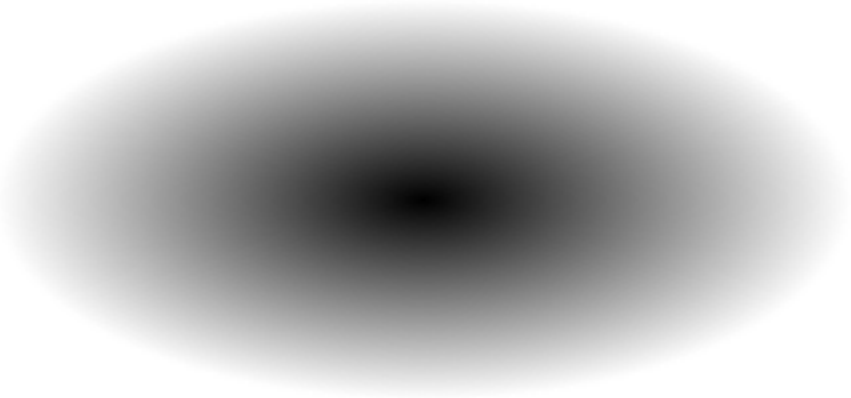
In Raphaël, complicated layered shapes can be easily created in just a few lines of code. And like everything produceable in Raphaël, it is cross-browser without modifications.

```
    var R = paper, x = 310, y = 180, r = 150;
    R.ellipse(x, y + r - 3, r + 10, r / 2)
     .attr({fill: "r#000-#000", stroke: "none", opacity: 0});
    R.circle(x, y, r)
     .attr({fill: "r(.5,.9)#39c-#036", stroke: "none"});
    R.ellipse(x, y, r - 10, r - 3)
     .attr({stroke: "none", fill: "r(.5,.1)#ccc-#ccc", opacity: 0});
```

Firefox 3.5	Internet Explorer 8
	

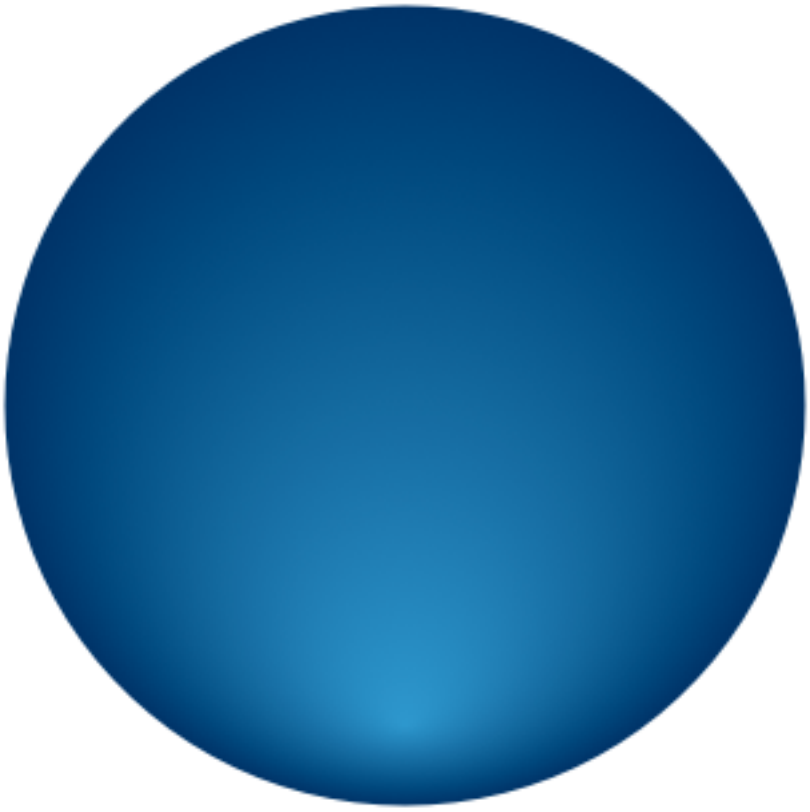
Layered Shape - Decomposition

The shadow:



```
      R.ellipse(x, y + r - 3, r + 10, r / 2)
       .attr({fill: "r#000-#000", stroke: "none", opacity: 0});
```

The sphere:



```
      R.circle(x, y, r)
       .attr({fill: "r(.5,.9)#39c-#036", stroke: "none"});
```

The sphere’s reflection:

```
      R.ellipse(x, y, r - 10, r - 3)
       .attr({stroke: "none", fill: "r(.5,.1)#ccc-#ccc", opacity: 0});
```

Raphaël API

The document, available online at http://raphaeljs.com/reference.html, provides explanations for these native objects, object methods, properties, and attributes.

Native Methods and Objects

Vector Objects	Object Methods	Object Properties	Object Attributes
circle	animate	fill	cx
ellipse	attr	fill-opacity	cy
image	getBBox	font	height
path	hide	font-family	path
rect	insertAfter	font-size	r
text	insertBefore	gradient	rx
	remove	opacity	ry
	rotate	rotation	src
	scale	scale	width
	show	stroke	x
	stop	stroke-dasharray	y
	toBack	stroke-linecap	
	toFront	stroke-linejoin	
	translate	stroke-miterlimit	
		stroke-opacity	
		stroke-width	
		translation	

Extensibility

Raphaël can be extended to allow for plug-ins to use the highest-order functions in both the Raphaël canvas and in Raphaël elements themselves. From the documentation:

Adding your own methods to canvas

You can add your own method to the canvas. For example if you want to draw pie chart, you can create your own pie chart function and ship it as a Raphaël plugin. To do this you need to extend Raphael.fn object. Please note that you can create your own namespaces inside fn object. Methods will be run in context of canvas anyway. You should alter fn object before Raphaël instance was created, otherwise it will take no effect.

```
Raphael.fn.arrow = function (x1, y1, x2, y2, size) {
    return this.path(// some code here);
};
// or create namespace
Raphael.fn.mystuff = {
    arrow: function () {…},
    star: function () {…},
    // etc…
};
var paper = Raphael(10, 10, 630, 480);
// then use it
paper.arrow(10, 10, 30, 30, 5).attr({fill: "#f00"});
paper.mystuff.arrow();
```

```
paper.mystuff.star();
```

Adding your own methods to elements

You can add your own method to elements. This is useful when you want to hack default functionality or want to wrap some common transformation or attributes in one method. In difference to canvas methods, you can redefine element method at any time.

```
Raphael.el.red = function () {
    this.attr({fill: "#f00"});
};
// then use it
paper.circle(100, 100, 20).red();
```

JavaScript Interaction with jQuery

When Raphaël is used in conjunction with jQuery, all Raphaël objects can be transparently passed to jQuery for jQuery objects. These JQuery objects can be used just as any other DOM element that is handled by jQuery, including, but not limited to, animations, styling, timers, event handling, arbitrary data stores, and event and mouse bindings.

While none of these JavaScript methods are specific to jQuery and can all be written in pure JavaScript, the simple fact that jQuery smoothes out many of the differences between different browsers means that you can continue to write cross-browser code in JavaScript with minimal effort necessary to account for one browser over another.

Considerations & Lessons

Performance

Of the few differences between different browsers exposed in the Raphaël API, the largest difference by far is that of performance. SVG runs with speed and perceived smoothness in all browsers that support it. All modern browsers that currently support SVG are also, in fact, putting in a great amount of effort into developing SVG as their implementations correspond to the SVG spec, as well as speed improvements that are found with modern rendering engines which precompute and prefetch objects and interactions with aplomb.

However, Internet Explorer, which only supports VML, renders vector elements in a fraction of the speed. Additionally, Internet Explorer 8 actually increases the delay in interacting with VML elements through JavaScript. Any loops in JavaScript that interact with elements have a slightly noticeable delay, and as the number of elements increase, performance degrades much quicker in Internet Explorer than it does in SVG-capable browsers.

This difference in rendering speed and interaction delays is attributable only to Internet Explorer's implementation of VML. A comparable implementation of SVG in Internet Explorer would be expected to perform on a level consistent with all other modern browsers.

Accessibility

Raphaël, as a JavaScript framework, produces SVG and VML on the client-side. When developers talk about accessibility, they often mean one of two terms. The first is accessibility in terms of search engine optimization (SEO) and marketing (SEM). The second is in terms of usability and reach for those using non-standard forms of browsing.

While Raphaël interprets JavaScript commands into cross-browser vector commands, it is only able to do so after the page is rendered. This means that concerns about Search Engines being able to access the data that is "locked up" in the JavaScript code are the same as concerns about Flash. Google has stated its dedication to unlocking content from Flash[1] and if Raphaël becomes a widely enough used technology, there is no reason to believe that Google will not be able to parse the resulting SVG/VML from interpreted Raphaël JavaScript code.

As for accessibility with screen readers, Raphaël produces markup that is inserted directly into the DOM, which means that the markup produced from interpreting Raphaël's JavaScript commands are no different than any other markup that is directly embedded in the page, such as HTML or raw SVG. Screen readers will be able to take advantage of the fact that Raphaël makes use of the DOM in a way that Flash cannot.

Additionally, if Raphaël is interacting with data that is on the page, such as in a chart or graph, a screen reader will still be able to take advantage of the raw data, in the case of JavaScript being disabled, or the screen reader using the data that is on the page regardless of presentation due to SVG markup.

API Changes

With an unfinished feature set, Raphaël has had to change the API a few times over the course of releasing a version 1.0. A number of methods have been removed or merged into other methods. While this is an unfortunate side-effect of optimizing code reuse, Dmitry explains his rationale on his personal blog:

> *Recently I released Raphaël 1.0 RC1. The most noticeable change for users of the library is differences in the API: removing moveTo, lineTo and friends from the path object. Why did I decide to do this? The biggest drawback of these methods is that they apply immediately. That means that if you draw path consisting of three segments, path*

---

[1] http://searchengineland.com/google-now-crawling-and-indexing-flash-content-14299

element is updated three times – each time with increased number of segments, so total number of segments drawn is 1 + 2 + 3 = 6. The more segments your path has, the longer it takes to draw. In geometric progression.

To avoid this I could introduce some method "draw" that could be called after you define all segments, but this doesn't look like an elegant solution and doesn't suit the library's name. There is a solution that is compact, simple and you could easily work with it. Say, you want to change some point on the path. There is no interface for this apart from SVG path, and I can't think of any elegant and easy to grasp API for this. Manipulating strings is what JavaScript can do very well. So I removed these methods from the library to external plugin, which you could concatenate with the Raphaël if you really rely on them. I suggest you learn SVG path syntax: it is simpler than it looks.

Other big thing is adding support for angle in arc. This is very rare thing, and frankly, I haven't seen it in the wild, but without it SVG path support wasn't complete. To make it happen I rewrote arcTo for VML completely. Now it converts arc to bezier curve, because VML doesn't have support for angles in arc segments. The same method is now used in animation (for SVG & VML), which makes animation more smooth. Conversion of the whole path into bezier curves let me do path bounding box calculations more precisely, especially in VML. In fact, Safari doesn't calculate bounding box for path correctly either.

Caching

JavaScript memoization has been implemented in Raphaël to achieve a drastic speed-up in the reuse of functions, esp. in loops and cycles. Memoization is a method used to increase speed of slow functions by caching the arguments and results. The expense of memoization is a marginal use of additional memory at the benefit of a gain in speed in reused functions.

The limitations of this method of caching are quite evident. Only scalar values and arrays can be used as arguments in memoized JavaScript functions. Objects can not be passed in due to browser security limitations and prohibitions. However, this guarantees idempotency in the function, as the arguments can be dereferenced and stored for cache lookups.

Limitations

Due to the DOM based nature of SVG and VML, the number of elements that can be drawn or interact is limited based on a browser basis. Canvas and Flash, due to their thicker wall between the browser and the canvas, allow for many more elements. Additionally, more elements, in SVG and the DOM, means worse performance, whereas Canvas and Flash do not degrade under such circumstances.

Secondly, Raphaël only produces SVG and VML after page load. The consequences of this are two-fold: (a) there are no SEO benefits for markup that is not on the page at load time, and (b) there is a slight delay for any SVG/VML to appear on the page, since

Raphaël waits until it is run, which must be either after the element in the DOM that will serve as the Raphaël canvas has loaded, or during the document.ready() event.

Alternatives to Raphaël

There are a few existing solutions to the problem of creating interactivity and animated vector graphics. One solution, Adobe Flash, sits closer to the extreme of not allowing the browser to have any control, while SVG is a solution that gives full control to the browser, where some browsers go so far as to display the same code in a slightly-different manner than the next browser. Additionally, using JavaScript on the <canvas> tag, or using JavaScript directly on the DOM, allows for interaction between elements on the page, but elements in a canvas are sand-boxed inside the canvas, whereas SVG elements are native to the DOM.

SVGWeb, an open-source project that is hosted by Google, is a library that converts SVG into a Flash-based canvas on browsers that do not support SVG. The performance of Flash can be noticeably faster, and SVGWeb creates Flash-based canvases that perform similarly to the native SVG implementation. However, SVGWeb suffers from the same constraints as Raphaël with regards to lack of SEO benefits. Additionally, SVGWeb locks the shapes and content behind Flash, which degrades accessibility, and also makes JavaScript interaction not possible. However, JavaScript interactions with elements is not supported, as SVGWeb uses a Flash sandbox in IE (and on other browsers when deliberately set), which prevents the ability to interact with vector elements in JavaScript.

SMIL, a W3C standard for defining timing, animations, layout, and multimedia embeds, is only well-supported on Firefox, Safari, and Opera. Raphaël's goal of being entirely cross-browser precludes the use of SMIL for animations. But SMIL is only one of a handful of browser-native ways of animating vector elements of SVG. Instead, Raphaël relies on JavaScript to produce timers and animations, which also offers better user interface capabilities than SMIL. CSS Animation support also works on Raphaël elements, but is only supported by Safari 4.

Canvas is not a direct competitor to SVG. Canvas is not vector-based but rather bitmap-based. Canvas is more of a complimentary technology to SVG, being used to support graphics that are more prevalent in games and graphs, rather than the vectorized shapes of SVG. However, Canvas is less object-oriented than SVG, due to the drawing nature of Canvas. SVG's elements can be "stored" on the page, better suited to interacting with their data, whereas Canvas paints on to the page, not inherently keeping track of shapes and objects. A recent blog comment on the difference between SVG and Canvas illustrates a common difficulty with Canvas[2]:

> *For instance, suppose one wanted in a generic scatter plot tool to implement a "data selector" which allows the user to highlight rectangular subsets of data points with a*

---

[2] http://neilobremski.wordpress.com/2009/04/30/svg-vs-canvas#comment-89

> *dynamically-sized rectangle dragged over the desired region in the canvas. In this case, it would be necessary to 1) model the points in the scatter plot as some object capturing the "vector graphics" nature of the canvas, 2) and then on each mousemove on which the mouse is down, clear the canvas, redraw all the points, and then have a rectangle drawn from the original depression point of the mouse to its current location. Obviously it would be more convenient if I could simply model the data points and the "selector box" rectangle as separate persistent objects in the DOM, and then simply modify the rectangle's configuration by changing one of its DOM properties.*

Dojo, which has support for many of the same drawing tools provided by Raphaël does not allow for support with other client-side libraries such as jQuery or mootools. Raphaël allows for the ability to drop in another JavaScript library and have it seamlessly work with Raphaël objects.

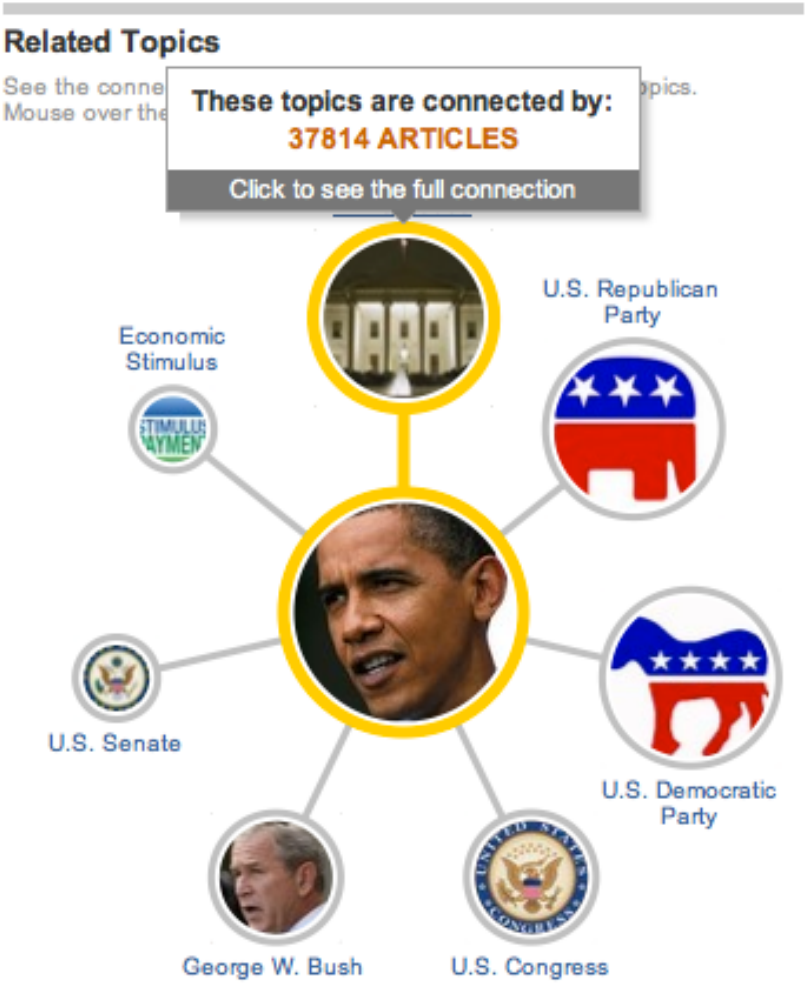
Real World Examples

Raphaël is currently being used in a number of places on the web. Two notable examples:

The Washington Post / EVRI

At the bottom of every article page is an EVRI widget which shows topics extracted from the article and the connections between different topics. See this article for an example: http://www.washingtonpost.com/wp-dyn/content/article/2005/11/29/AR2005112901464.html.

Daylife



Daylife, a news aggregator with articles, photos, and quotes on many major topics in the news, uses Raphaël on all topic pages. See the Barack Obama topic page on Daylife for an example: http://beta.daylife.com/topic/Barack_Obama.

Summary

Similar to jQuery, an open source JavaScript library that simplifies the interaction between HTML and JavaScript, Raphaël simplifies the creation of cross-browser vector graphics and the necessary interaction between the SVG/VML objects and HTML.