Optimization Tools for HTML, CSS, and JavaScript*

Introduction

In September 2012 at TechCrunch Disrupt, Facebook’s CEO Mark Zuckerberg made the following comment, “The biggest mistake we made as a company was betting too much on HTML5.” [http://www.netmagazine.com/news/facebook-engineer-reveals-html5-flaws-122227] This comment created a lot of noise within the HTML developer community. So much so that a Facebook engineer qualified Zuckerberg’s statement, saying: “The lack of tooling in mobile browsers makes it very difficult to dig down and find out what the real issues are. Hence tooling, or rather, lack-thereof is a key issue.” [http://lists.w3.org/Archives/Public/public-coremob/2012Sep/0021.html]

Writing software for HTML applications is significantly different than writing traditional software because HTML code is commonly generated at runtime rather than pre-installed and run. This fact changes how tools are used to test and optimize the software and hardware performance. This paper will describe some techniques and tools used to help solve some of the issues. For better clarity, this paper will refer to HTML, JavaScript, and CSS in general terms as HTML or HTML5. Also HTML optimization is not referring to improving the ranking in a search list but the overall user experience and responsiveness of the web site or application.

This paper isn’t a pro-and-con paper on the use of HTML and Internet technologies. HTML technology cannot meet the ideal of write-once-run-everywhere as other languages and technologies have touted and debatably failed at. The number of different devices and fragmented implementations make write-once-run-everywhere while providing a great user experience very difficult if not impossible, as Facebook found out. Many believe that HTML, JavaScript, and CSS will become the dominate technologies in client application user interfaces as applications strive to give a seamless experience moving forward. It is way too expensive to write, test, and maintain custom applications for all of these devices. As a consequence, it is essential for software developers to understand how to optimize HTML-associated technologies.

So let’s assume that the Facebook engineer we quoted is correct in attributing the lack of tooling for HTML, JavaScript, and CSS applications as a key contributor in prohibiting Internet technologies from being the tools of choice for application UI. With a long history of optimizing application in-depth understanding of the hardware there isn’t a company better suited to solve this problem than Intel. This paper will provide a simple step approach for developers to dig down, optimize their HTML, and help solve these HTML, JavaScript, CSS problems.

Understand the workload and the browser components

HTML and its associated technologies are typically open—everyone knows you can view the HTML source code of most everything on the Internet. Developers learn by example. This paper will present a simple example of how to isolate and optimize your HTML.

The first step in optimizing HTML is to understand the content of your HTML and the subcomponents of the browser/runtime that are used to present your content. I specifically did not use the word “render” because content rendering is the work done to display the content on the screen.

So let’s get started.
The four files in our example (html-topt.zip) are:

- html-topt.html
- html-topt.css
- html-topt.js
- jquery.js

The browser components are those in Google’s open source Chrome™ browser although all of the concepts discussed can be applied to most browsers.

The page we will optimize is shown in Figure 1. The top left side of the page is a canvas image drawn in real time by using one of two JavaScript functions: setTimeout or requestAnimationFrame. The requestAnimationFrame function will be discussed later. First let’s look at setTimeout. The setTimeout function has been used since the beginning of JavaScript to update user interfaces (UI). When HTML and JavaScript started to be used as an option for native applications, it was obvious there was a need for more interaction and responsiveness from the UI.

Developers decided to opt for the quick and easy solution of shortening the timeout. This works great from the UI perspective, but it is terrible for battery life. This may not seem important to some developers since the application is very responsive and it is a simple fix. But what developers may not realize is that users are uninstalling their applications because users notice that since installing the application, the time between having to charge their battery was cut in half. Using Intel® tools, like the Intel® Power Gadget (http://software.intel.com/en-us/articles/intel-power-gadget-20), developers can easily see what is going on. Before we look at the results, let’s look at the code.

The html-topt HTML and CSS files declare the Canvas location, but all the work is actually done by the JavaScript engine (in the case of Chrome® V8). The HTML/CSS code is shown below:

**html-topt.html**
```html
<canvas class="swoosh-canvas" id="swoosh" style="border: 1px solid #000000;">
   HTML5 Canvas not supported
</canvas>
```

**Html-topt.css**
```css
.swoosh-canvas {
   position: absolute;
   top: 50px;
   left: 30px;
   margin-top: 5px;
   margin-left: 5px;
   width: 400px;
   height: 380px;
}
```
More interesting is the JavaScript. The Canvas animation is done by a simple function called draw.

```javascript
/* draw */
function draw() {
    var time = new Date().getTime() * 0.002;
    var x = Math.sin(time) * 96 + 128;
    var y = Math.cos(time * 0.9) * 96 + 100;
    var colors = ['red', 'orange', 'yellow', 'green', 'blue'];
    context.fillStyle = 'rgb(128,128,128)';
    context.fillRect(0, 0, 0, 380);
    for (var i = 0; i < 25; i++) {
        if (i % 5 == 0)
            y = y + 30;
        context.fillStyle = colors[i % 5];
        context.beginPath();
        context.arc(x + ((i % 5) * 40), y, 10, 0, Math.PI * 2, true);
        context.closePath();
        context.fill();
    }
}
```

This function uses simple math and canvas fillStyle, fillRect, and arc functions to draw the image. The more interesting part of the code is how the draw function is called. I used a function DoAnimation, which provides two options on how to “render” the image to the screen. The first is setTimeout. The second is requestAnimationFrame. For now, let’s focus in on setTimeout and the rate at which it expires as described earlier.

```javascript
/* DoAnimation */
function DoAnimation() {
    if (VisibleState == true) {
        if (UseSetTimeout == true) {
            WinSetTimeout(DoAnimation);
        } else {
            WinRequestAnimationFrame(DoAnimation);
        }
    } else {
        winAnimationFrame(DoAnimation);
    }
    draw();
}
```

The rate at which the timeout occurs is set by the user in the canvas-options form updated by the LengthRestriction function and put into the global variable TimeOutTime.
function LengthRestriction(input, min, max) {
    var uInput = input.value;

    if ( uInput.length >= min && uInput.length <= max ) {
        TimeOutTime = uInput;
        context.fillStyle = 'rgb(128,128,128)';
        context.fillRect(0, 0, 500, 380);
        DoPage();
        return true;
    } else {
        input.focus();
        return false;
    }
}

Before we dive into the power impact of altering the timeout, let’s look at how components interact within the browser. We know that anything that uses class tags to identify an object interacts with the DOM. So we would expect to see some DOM activity. Second, we should expect to see the JavaScript engine high in the list since the canvas functions are using simple math and drawing functions. Also since we are drawing graphics, we expect to see one of two possibilities. If the canvas routing is being rendered to the screen using hardware acceleration, we should see some Microsoft DirectX* functions being called since we are running Windows*. If the graphics are software rendered, we should see a software graphics engine being used at some point.

**Use the browser tools for high-level profiling**

Most of the popular browsers provide developer tool capabilities built into the browser to do profiling. Use them. They are very helpful at getting a high-level view of what is going on. Granted, this view is high level; so to solve the problems that Facebook and others have run into, we need to dig deeper. Let’s go back to our example and see what the browser tools themselves can tell us. We can use the profilers in Chrome, Firefox*, or Internet Explorer*. In this case we will use Chrome’s profiler.
There are no surprises yet. As expected, the majority of the work being done by the CPU when the Canvas animation is being drawn comes from the DoAnimation and the draw functions. Also, when you look closer at the code, you see the jpg’s fading in and out which is being done by the jquery JavaScript as defined in the CSS. We can ignore those for now since we are going to focus on the Canvas animation.

**Use Intel Tools to take a closer look at the system**

Now that we know more about what is going on from the browser profiler, we should look closer at what is going on with the hardware. Using the Intel® VTune™ Amplifier (http://software.intel.com/en-us/intel-vtune-amplifier-xe/) we can actually see how the browser and these functions interact with the operating system APIs. Just how much work is being done in native code compared to the JavaScript engine?

The following is a screen capture from the VTune analyzer showing the output of a system-level event trace.

The top functions of concern for the browser are the DirectX calls being made in OpenAdapter and dxgms1.sys used for hardware rendering. SKIA functions are also being called for software graphics rendering. Since there are no symbols for the driver functions, the OpenAdapter is the closest function to the other functions and is being attributed to the OpenAdapter function. A function of interest and one for potential optimization is the decode_mcu_slow, which is a Huffman decode used to help decode the jpg files that are fading in and out. As you turn the functionality of the html-top site on and off, you see these functions contributing more, less, or not at all to the overall workload. The VTune analyzer will actually let you drill down to the source code of Chrome or other browsers if you have built the browser with symbols. In this example below you can see the number of CPU unhalted clock ticks and instructions retired inside Chrome’s decode_mcu_slow function. You can even drill down to the machine code if you like.
Okay now that we have introduced you to the tools to be used for optimization, let’s look at how to use them to measure the use of setTimeout and compare it to requestAnimationFrame, functions provided by standard JavaScript.

Use all of the tools to help write your code
The html-topt.html page has been intentionally written to identify the differences between using the setTimeout and using setAnimationFrame. The top left corner of the html-topt.html page has a Canvas animation with a user selectable control to direct how the image is updated. Using Intel Power Gadget we are able to determine the amount of power being used for both the setTimeout and requestAnimationFrame methods.

The first test was to run the setTimeout at values of 100, 1,000, and 10,000 (notice inside the JavaScript these values are divided by 60). On an Ultrabook™ system with a 2.5 GHz Intel® Core™ i7 processor, the power used when the value was set to 100 was over 3.2 times the power used when set to 10,000. But there is a significant negative visual impact on the user interface. When you compare the setTimeout value of 1000 to the requestAnimationFrame (raf) API, the user interface is negligible but the power difference is significant at 1.8 times the reduction on package power. As this data shows it does make a difference on how you write applications in HTML. The power difference in Watts is clear in the graph below.
The problem that Mark Zuckerberg and the mobile Facebook engineers had was the ability to see what was going on and optimize for not only performance, but also power. HTML and its associated technologies are evolving into very powerful tools that are coming closer to spanning the bridge of software across the “compute continuum.” As you can see we are barely scratching the surface of what we can do with HTML as a technology. HTML and web technologies are a significant part of how we use computers from desktops to mobile devices. Intel is dedicated to helping make HTML, CSS, JavaScript, and the associated Internet technologies and tools even better.

References

About the Author
Richard Winterton graduated from Brigham Young University in 1986 with a BS degree in Electrical Engineering and a minor in mathematics. Rich worked for Lockheed Martin from 1986-1994. While at Lockheed Martin, he designed two network ASICs and authored two network communication protocol standards. Richard was a member of a team that wrote an operating system for the U.S. Navy's A-12 Avenger, as well as "BIOS" software for the main avionic computer for the F-16 Fighting Falcon. In 1994 Richard joined Intel where he is an engineer responsible for software optimization for Intel's latest and next generation products. For the last eight years Rich has focused on Internet standards, optimizations, tools, and technologies. When Rich isn’t working he enjoys running.
Notices
INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel, the Intel logo, Core, Ultrabook, and VTune are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others

Copyright © 2013 Intel Corporation. All rights reserved.
Optimization Notice


Performance Notice
For more complete information about performance and benchmark results, visit www.intel.com/benchmarks