JIM SANGWINE

3D Web contenders

Jim Sangwine started his career as a lecturer on the 3D animation degree course at Portsmouth University in England in 2002, but by 2004 he had developed an interest in programming and the Web that drew him away from animation and into full-time development. In 2007 Jim left the UK and joined Competa IT in the Netherlands as a platform-agnostic Web application developer working primarily with PHP, C#, and ActionScript. With the support of his colleagues at Competa and funding from NLNET, Jim is soon to begin work on a JavaScript physics library for use with WebGL.

jim@competa.com

3D INTERNET IS FINALLY MOVING OUT of the realm of film and fantasy. We find ourselves at a critical crossroads in the development of this new dimension to our online world. Although the Web is fundamentally standards-based, the reality for developers is that the main browsers adopt and implement standards inconsistently, to varying degrees, and at varying speeds. The situation is exacerbated by vendors of plug-ins vying for market share, and there are already a number of contenders for the 3D content crown. I have looked at what I believe are the three strongest and have run some benchmarks to compare their performance.

The Web Is a Battleground

The defining aspect of the Web development landscape has almost always been (and will, in all probability, always be) the browser wars. While a desktop application developer might have to target a particular platform based on his or her audience, with the possible option of making ports later on, the conceptual platform for Web application developers, and the target audience for our clients, is not educational establishments running UNIX or gamers running Windows 7, but, rather, "the Internet." Our OS is the browser, and it is usually our job to make sure that our application works for as close to 100% of our potential users as possible. This problem is alleviated to some extent by the existence of standards and common languages that all browsers should support (HTML, CSS, JavaScript), but despite (or perhaps because of) the efforts to standardize, the nature of the browser wars means that developers have to put more work, not less, into accessibility and legacy support each year.

The root of the problem is that the update cycle for Web standards and specifications is often very short. The first publicly available description of HTML was seen online in late 1991 and described only 20 elements, of which 13 still existed in the HTML4 specifications published in December 1997. Since 1991 there have been, including XHTML, seven official published updates to the specifications, and HTML5, including XHTML5, is currently in working draft. The first W3C recommendation for CSS was published in December

1996, CSS level 2 was published as a recommendation less than 18 months later, in May 1998, and the first public draft of CSS level 3 was released in April 2000. CSS level 3 is still in development. The standardization model for JavaScript, ECMAScript, has had three published versions since November 1996, but JavaScript is an interpreted language and each of the four major layout engines that are used in the majority of modern browsers offers wildly differing levels of support for the various specifications (see Figure 1) [1].

ECMAScript version support

	Trident	Gecko	WebKit	Presto
Name of ECMAScript Engine	JScript	Spidermonkey/TraceMonkey	JavaScriptCore/SquirrelFish Extreme	Linear B/Futhark/Carakan
ECMAScript Edition 3	Yes	0.6	Yes	1.0
JavaScript 1.5 extensions	No	0.6	Yes	1.0
JavaScript 1.6 extensions (excluding E4X)	No	1.8	Partial	Partial
JavaScript 1.7 extensions	No	1.8.1	No	Partial
JavaScript 1.8 extensions	No	1.9	Partial	No
JavaScript 1.8.1 extensions	No	1.9.1	No	No
JScript .NET extensions	No	No	No	No
ActionScript extensions	No	No	No	No
E4X	No	1.8	No	No

FIGURE 1: COMPARISON OF LAYOUT ENGINES (ECMASCRIPT)

This enormous and rapid development in the definition of Web content has led to an explosion of browser versions. There are five families of browsers with significant share in the market, according to the statistics maintained by w3schools, in descending order of usage as of February 2010: Firefox (46.2%), Internet Explorer (34.9%), Chrome (12.3%), Safari (3.7%), and Opera (2.2%). If you look into the browser versions these stats include, you will find that we are talking about three versions of Firefox, three versions of Internet Explorer, three versions of Chrome, two versions of Safari, and two versions of Opera, for a staggering thirteen distinct variants currently in use [2].

The Problem of Choice

> No study is possible on the battlefield.
> —Ferdinand Foch

It is in this rapid introduction of new browser versions coupled with an apparent reluctance to upgrade on the part of the user base that we find the real barrier to the adoption and support of new standards by commercial developers. The oldest significantly used version of Firefox (3.0–5.6% usage) was released almost two years ago, but much worse is Internet Explorer 6, which was released in August 2001 and yet still represents 8.9% of our potential users. Eight and a half years is almost half the lifespan of HTML and the browser-based Web as we know it today, yet developers the world over are still spending time, effort, and money on supporting this ancient relic. Worse than that, they are often forced to eschew new techniques to maintain legacy support, at the same time restricting the progress of the Internet and prolonging the lifespan of IE 6.

The Browser-Agnostic Alternative

> We can love an honest rogue, but what is more offensive than a false saint?
> —Jessamyn West

Perhaps the most obvious solution to the problem of finding a way to deliver rich, technologically advanced content to users without hitting problems with browser compatibility was always going to be via the use of plug-ins. November 1994 saw the specification of VRML (originally Virtual Reality Modeling Language), a markup standard for the definition of 3D objects, the description of surface properties (shininess, transparency, color, etc.) and the ability to invoke URLs in response to clicks on scene elements. Browser integration was by way of third-party plug-ins, and, unfortunately, was fairly inconsistent. This fragmented support for the standard across the various browsers was likely a big factor in VRML's limited adoption by developers.

In 1996 the first version of Flash was released. Initially called FutureSplash Animator, this application was bought and rebranded as Macromedia Flash in the same year and acquired and rebranded as Adobe Flash in 2007. ActionScript, the object-oriented scripting language of Flash, is based on the ECMAScript standards and so is syntactically very similar to JavaScript. Basic scripting was introduced in version 2 of Flash with a few simple timeline navigation commands (called Actions) that could be embedded in frames of movies, and version 5 implemented the first iteration of ActionScript. Flash deserves special mention here because it has achieved far greater penetration than any other plug-in or even any one browser family. Adobe claims that as of December 2009, 98.9% of users in what they call "Mature Markets" (US, Canada, UK, Germany, France, Japan, Australia, New Zealand) are running at least Flash Player 9 and 94.7% are running the latest version (Flash Player 10). Their "Emerging Market" figures (China, South Korea, Russia, India, Taiwan) are almost as impressive, claiming 98% for at least version 9 and 92.7% for version 10 [3].

The fact is that the Flash Player plug-in is maintained by one organization, and so content is rendered extremely consistently regardless of which browser is used to view it, which makes it a very appealing alternative to other plug-ins and even to JavaScript and AJAX approaches to rich content authoring. Flash was never intended as a 3D platform (although Adobe had started adding rudimentary support for 3D effects with version 10), but there are now a number of extremely powerful ActionScript 3 libraries available, including, most notably, and considered by many to be the current de facto standard for 3D online, the open source Papervision3D.

Papervision3D has already been used for a large number of projects, including some very high-profile integrations with the ARToolkit—for example, the GE Smart Grid Augmented Reality app [4]—and has proven itself as a workable solution.

There are many who feel that the existence of plug-ins, and especially Flash, is detracting from the efforts of (some) vendors to move towards the ideal of an open Web through the introduction of, and adherence to, comprehensive specifications for the native support of rich content such as video and 3D directly in browsers.

Another frequent criticism of Flash and other plug-ins is that they live independently of the page. They are not subject to the browser back and forward buttons, their state cannot be bookmarked in the browser, they don't obey changes to the text size browser settings, they don't appear in page prints, and they do not expose their content to search engines. It is often argued that these are usability issues, and that the use of discrete applications that operate outside of the page breaks the whole concept of the Web.

In addition, plug-ins can represent a security risk, since they do not fit into the security model of browsers. The popularity, bugginess, and implied access to the Internet of browser plug-ins makes them ideal targets.

The Rewards of Rivalry

> Although personally I am quite content with existing explosives, I feel we must not stand in the path of improvement.
>
> —Sir Winston Churchill, August 30, 1941

It can be (and frequently has been) argued that conflict and competition are great catalysts for progress. Just as a world war can stimulate breakthroughs in science and technology, the browser wars have also acted to push forward development. JavaScript has become faster and more powerful, CSS is supported fairly well by the majority of modern browsers, and measures are being taken to improve security across the board. The frustrations faced by developers in trying to keep up with the hacks on top of hacks that are required to maintain consistent rendering for their users are, in the opinion of some, trivial compared to those that would arise from a world without browser competition.

This recently started to hold true for developments in 3D when both Mozilla and Google began work on 3D solutions. Mozilla partnered with the Khronos Group (the organization behind OpenGL) to produce a component called Canvas 3D that exposes the OpenGL ES 2.0 APIs to JavaScript, and Google is working on their O3D plug-in which accepts JavaScript calls through their own proprietary API and uses either OpenGL or Direct3D for rendering. Both projects were intended to be considered for adoption as the basis of an open Web standard for 3D, but it is the efforts of Mozilla and Khronos that have been taken forward for development into a product dubbed WebGL. The WebGL Working Group's members include Apple, Mozilla, Opera, and Google, although Google is still continuing work on O3D.

There is, understandably, a lot of discussion about the relative benefits (and disadvantages) of O3D and WebGL. O3D is currently the more mature technology, and many are electing to sit on the fence awaiting a stable production release of WebGL.

Google's Gregg Tavares posted a message on the o3d-discuss group, stating that they (Google) "have every interest in seeing both WebGL and O3D succeed," but he also voiced an opinion that JavaScript is just too slow to match the potential of O3D [5]. This is mainly because WebGL relies entirely on JavaScript to manipulate the scene, including transforms, culling, sorting, and animation, with only the actual OpenGL calls hardware-accelerated.

Gregg also mentioned that OpenGL ES 2.0 (the API exposed via WebGL) is not supported by "lots of common hardware."

In the same thread Henry Bridge (also from Google) reiterated that they are working on both projects (the two are on the same team). He also said that it makes sense to "standardize GL for JS" but justified continued development of O3D by saying that the gap in performance makes the two technologies appropriate for different situations.

The Facts

Prompted by the comments from Google regarding the potential performance problems inherent in the WebGL model, I decided to try to quickly test the real-world practicality of the three systems.

I started by doing a direct comparison between the latest raft of JavaScript engines and ActionScript in FlashPlayer 10. I ported five of the routines from Apple's SunSpider [6] JavaScript benchmarking application to ActionScript and ran them on a 2.53GHz Intel Core 2 Duo MacBook Pro with 4GB of DDR3 RAM. I experienced a 10+% variance in test times in Flash Player, so I set each test to run 10 times and calculated the mean in order to stabilize the results. JavaScript tests were much more consistent, so I ran those only five times each. The routines I ported were 3D Cube, 3D Morph, Cordic Math, Partial Sums Math, and Spectral Norm Math (see Figure 2).

The browsers I used were the latest nightly builds of WebKit (Safari), Minefield (Firefox), and Chromium (Chrome), so as to represent the current state of JavaScript engine performance. I did not include Internet Explorer in the tests, partially because native versions for my hardware do not exist and testing in a VM would skew the results, but also because WebGL is currently not supported by this family of browsers, making the comparison, at least for now, irrelevant to my investigation.

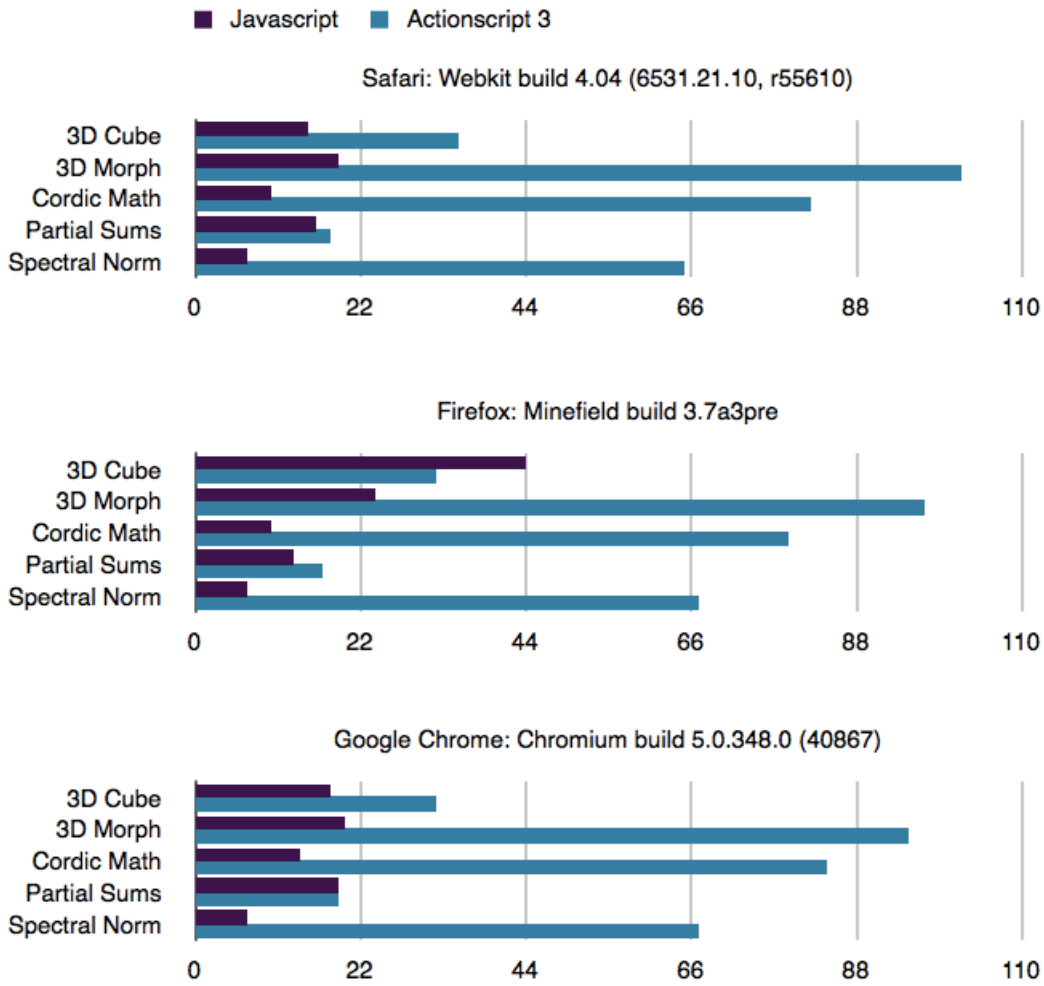

FIGURE 2: THESE CHARTS COMPARE THE RESULTS OF 5 SUNSPIDER BENCHMARK TESTS RUN IN JAVASCRIPT AND ACTIONSCRIPT ON EACH OF THREE PLATFORMS, WITH TIME MEASURED IN MILLISECONDS.

As one would expect, the performance of FlashPlayer was extremely consistent across all browsers in all tests. There was some variance in JavaScript performance, particularly in the case of Firefox, which came out worst, but overall it is very clear that JavaScript is considerably faster than ActionScript

at performing these kinds of calculations. This result was hardly surprising to me, considering the extra overhead involved in running the FlashPlayer. It did, however, suggest that WebGL might indeed be a very strong alternative, at least to what many consider to be the current king of Web 3D solutions, Papervision3D and Flash.

My next test was very crude, and certainly not a comprehensive or even particularly fair comparison, due to the massive differences between the systems, but I think it gives some indication of the usefulness of each, particularly for the kind of undemanding use that the majority of Web developers will put them to. I created three simple projects in ActionScript, WebGL, and O3D, each containing only three elements: a simply shaded sphere, a light, and a camera. I then animated the spheres, rotating them around their Y axis, and experimented with increasing mesh densities to get a very rough feel for the kind of polygon load each technology could handle. The results were pretty interesting.

I only managed to complete all three tests in Minefield; I couldn't get O3D running in WebKit, and neither WebGL nor O3D ran in my version of Chromium. The ActionScript results were again extremely consistent across the three browsers, but WebGL's JavaScript scene processing completed 5 seconds quicker in WebKit than in Minefield, with no noticeable difference in frame rate.

My results in Minefield were as follows:

- ActionScript dropped to 4 frames per second with a sphere of 20,000 triangles and took 2 seconds to load.
- The O3D plug-in couldn't handle a triangle count much greater than 150,000 before exiting with the message "ERROR: The maximum number of elements in a buffer is 1048575." However, a consistent frame rate of 60 fps was maintained with no drop at all up to this limit, and the scene took only 2 seconds to process. It turns out that since some hardware can only handle a maximum of 1048575 vertices per object, the O3D team decided to impose that as a standard limit to ensure the portability of applications. So I ran another test, this time using 10 spheres for a total triangle count of 1,512,500. O3D managed to maintain a frame rate of 15 fps but the scene took 51 seconds to load.
- The real surprise was the WebGL solution, which managed an extremely impressive 15 frames per second with a triangle count of 6,480,000 in a single sphere. The scene took a fairly acceptable 15 seconds to process (and only 10 in WebKit), going a long way towards allaying my fears as to the practicality of handling fairly complex scenes in JavaScript.

Where Do We Go from Here?

In my opinion there are a number of points that need to be considered when choosing among the three solutions I have discussed.

Firstly, there is the question of accessibility. For any technology to succeed on the Web, it must work for the vast majority of users, especially if it is to be considered for serious commercial projects. In this respect, Flash has the obvious advantage as a mature technology that has been around for a very long time in Internet terms and that has achieved (at least according to Adobe) almost total penetration. WebGL, despite still being in beta phase, claims to be supported by most of the major browsers, but the lack of Internet Explorer support will undoubtedly be a deal-breaker in the eyes of many professional developers. Unfortunately, Microsoft has hinted that they have no intention of supporting WebGL in the upcoming Internet Explorer

9. In contrast, O3D claims support for all the major browsers (although my experience did not seem to confirm this), IE included. Also, O3D supports a greater range of hardware than WebGL. However, O3D is just another plug-in, and unless it gets inclusion in the off-the-shelf browser versions like WebGL, it will be an uphill struggle to achieve the kind of penetration Adobe Flash claims.

Another very important consideration is ease of use. The Web is a different environment from the desktop application world, requiring a different set of skills, and Web developers in general are not likely to have experience with 3D animation and physics or shader programming. If 3D is to be adopted for commercial Web projects, it will have to prove itself to be both stable and quick to develop. Agile development has become something of a mantra in the Web application world, and there will likely be strong resistance in commercial projects to any technology that represents a significant cost in training and time. Again, Flash stands out here, thanks to the Papervision3D libraries with which it is incredibly easy to get something up and working with remarkably little code. There is already a vibrant community, and the number of online examples and tutorials is growing. O3D is, out of the box, easier to pick up than WebGL, but still daunting to the uninitiated. However, I see this as a temporary problem, and certainly not a game-winning factor. There are already libraries popping up that encapsulate and automate common tasks in easy-to-implement frameworks, even for the as yet unreleased WebGL. Over time, I believe that the barrier to entry will lower, and for those who wish to, there will still be the option to dive into OpenGL coding.

This leads nicely into the third criterion: power. Here, O3D claims to have the edge over WebGL, and certainly over Flash. As I said earlier, my mesh load experiment was very crude and did nothing to highlight or test specific features of any of the technologies, but to my mind it seems to demonstrate that JavaScript is already more than capable of shifting around some serious numbers at a speed that will allow developers to do much more than is currently possible with Flash and Papervision3D. Perhaps it is not yet possible to produce the next Quake Arena title using only JavaScript and WebGL, but I personally doubt whether that is an appropriate use of the technology right now. I feel that the Web needs to get used to 3D, and that it will be quite some time before developers and users figure out how best to incorporate it into our online world. JavaScript has already increased performance incredibly, and if it becomes popular, WebGL might just be the motivation browser vendors need to invest further in improving their engines and in standardizing their feature support.

Which brings me to my final point: What is better for the Web? Here, I think there can only be one answer: WebGL. While competition can stimulate creativity and progress, I feel that the current chaos of browsers that refuse to die and reinventions of existing technology by plug-in developers are only serving to hold back the progress of the Web. I am almost as passionate about 3D as I am about the Web, and I really want the two to come together. I fear that the plug-in wars that are already beginning will delay the integration of 3D by making it impossible for anything other than Flash to be implemented in applications where reaching a large audience is a critical factor (as it is in the vast majority of projects). Therefore, having a technology that is available to everyone straight from the browser, and whose implementation is controlled by means of industry standards, perhaps even as part of the HTML5 spec, just has to be the best way forward in my view. There might be better options in terms of power or capabilities, but I feel that is a less important factor in the survival of this fledgling relationship.

REFERENCES

[1] Wikipedia, "Comparison of Layout Engines (ECMAScript)": http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(ECMAScript).

[2] W3Schools, "Web Statistics and Trends—Browser Statistics Month by Month": http://www.w3schools.com/browsers/browsers_stats.asp.

[3] Adobe, "Flash Player Version Penetration": http://www.adobe.com/products/player_census/flashplayer/version_penetration.html.

[4] GE, Smart Grid Augmented Reality: http://ge.ecomagination.com/smartgrid/?c_id=Huff#/augmented_reality.

[5] http://groups.google.com/group/o3d-discuss/browse_thread/thread/7bfa31efcc03b5f6.

[6] http://www2.Webkit.org/perf/sunspider-0.9/sunspider.html.