Tests of Graphics Rendering in Browsers

JAAK HENNO Tallinn University of Technology
HANNU JAAKKOLA, Tampere University of Technology, Pori Department
JUKKA MÄKELÄ, University of Lapland, Rovaniemi, Finland

Web browsers have become one of the most commonly used software and are important communication tool to access our data-driven, increasingly visual Internet. Browser graphics speed is essential for many commercial web applications – e-commerce sites, web portals, content management systems (CMS's), therefore web developers should well understand their possibilities. Browsers can be seen as multi-input (HTML-text, images, CSS, Scripts) multi-output (code for processor, graphics card, sound system) translators, but little is known about their 'internal life', especially how they render graphics. Browsers interpreting HTML5 documents have two graphic modes: Retained Mode (images defined in HTML text and animated with CSS) and Immediate Mode (images created on canvas and animated with JavaScript). In order to understand differences of these modes in animation rendering speed were created nine different versions of an animation of Lunar Eclipse which were tested in six major PC and Android mobile phone browsers. Results indicate, that there are no significant differences in major browsers except that IE and Edge (still) lag behind in implementing novel graphics/video formats and that in all tested browsers Retained Mode is at least two times quicker than Immediate Mode.

Categories and Subject Descriptors: **I.3.6 [Computer graphics]:** Methodology and Techniques; **I.3.4 [Computer graphics]** Graphics Utilities

General Terms: Browser, Visualization, Layout Engine, Rendering Engine, Retained mode, Immediate Mode

Additional Key Words and Phrases: HTML5, CSS3, JavaScript, SVG, rendering

1 INTRODUCTION

The first program what (usually) everyone opens when starting a computer, tablet or internet-enabled phone is a browser. Browsers have become the most common communication/interaction tool for all Internet-connected world. Currently (spring 2017) there are available (for PC-s) more than 100 different browsers [Web Developers Notes 2017] with different features – built-in e-mail, add-blocking, torrent handling, streaming etc. About quarter of these are already discontinued (still available), but every year appear some new ones, e.g. only in 2016 appeared Brave (automatically blocks ads and trackers, handles torrents) [Brave 2016], Microsoft Edge (only for Windows 10), [Microsoft Edge 2016], Vivaldi [Vivaldi] and several others.

Although in general statistics usage of some of them may seem negligible, in some areas of the World and for some users they may be very important [Wikipedia 2016. Usage share of web browsers], therefore developers of commercial web applications should try to comply with most of them.

Web is first of all visual media and animation, movement is the major way to make WWW attractive. Modern browsers enable several technologies for creating animations, but little is known about their efficiency, especially speed and for a commercial website even one second delay in page opening could cost millions in lost sales over a year. In the following are analyzed browser's graphic possibilities and created series of tests to compare the speed of different formats for creating browser animations.

Author's addresses: Jaak Henno, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia; email: jaak.henno@ttu.ee; Hannu Jaakkola, Tampere University of Technology, Pori Department, email: hannu.jaakkola@tut.fi; Jukka Mäkelä, University of Lapland, 96101 Rovaniemi Finland, email: jukka.makela@ulapland.fi.

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Belgrade, Serbia, 11-13.09.2017. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

2 BROWSER AS TRANSLATOR

Browser has become the major mechanism for presenting digital content on screen, i.e. a graphical User interface (UI)' HTML/CSS is currently the most advanced and flexible UI framework. Browsers are often already considered as the basis of a whole Operating System (OS) [Prakash 2016].

Browser works like a programming language's translator – it transforms information presented in format of input channels into format required by processor, graphics processor and other output devices, but its task is far more complex. Browser accepts input from several channels in several different formats – HTML/XML, CSS (Cascading Style Sheets), JavaScript, SVG (Scalable Vector Graphics), sound tags, keyboard and mouse or/and touch input; input may be provided from several files – the HTML-document itself and data from linked external sources, e.g. external CSS and JavaScript files, images, sound/video files. From these different types of inputs browser compiles complex output for computer/mobile screen containing text, images, video and sound.

Fig. 1. Broser as compiler with multiple input and output channels.

A browser output is essentially a frame-based interactive video. Computer/tablet/mobile redraw the screen with constant frequency (currently usually 60 Hz), thus browser should constantly calculate from inputs (HTML, CSS, JavaScript, inputs received from user and network) next frame content - text/images on screen and accompanying audio. And browser should also manage device (PC, mobile) resources - developers want to use maximal color depth (bpp – Bits Per Pixel, determines color quality) and framerate (fps – Frames Per Second).

For classical (programming languages) translators we have a nice 'translator theory' and (more or less) established functional structure [Aho Ullman 1972]:

SCANNER → SYNTAX ANALYZER → ABSTRACT TREE GENERATION → CODE GENERATION

There does not yet exist comparable 'browser theory' and established functional structure for browsers. The complex functional scheme of browsers could be presented as in the scheme in Figure 2.

Browsers are developed by different organizations and use different subsystems and the 'inner workings' even of major browsers are mostly mystery. For open source browsers something is presented in [WebKit 2017], [Tali Garsiel, Paul Irish 2011], [Spyros Doulgeridis 2017] but in rather general terms. Even the term 'open source' has become nearly meaningless, knowledge that 16,231,043 lines of C++ code of the open source Chromium browser [Chromium (Google Chrome) 2017] are available does not help web developers; the Google's Chrome browser uses the Chromium codebase, but adds lot of its own, proprietary code (licensed codecs for proprietary media formats AAC, H.264, MP3, Google Update, Crash and Error Reporting etc.) [Chris Hoffman 2014], thus the codebase is essentially larger. The situation with other browsers is similar, e.g. codebase of the major open-source browser Firefox is freely downloadable [Firefox 54.0.1 source code], but what could a web developer do with > 1 GB, ca 85000000 lines of c/c* code? For a Web developer this is useless, web

developers are not c/c* experts, they are interested what this code can/will do and suggesting to search answers from code is similar to trying to guess human's behavior investigating his/her genome sequence.

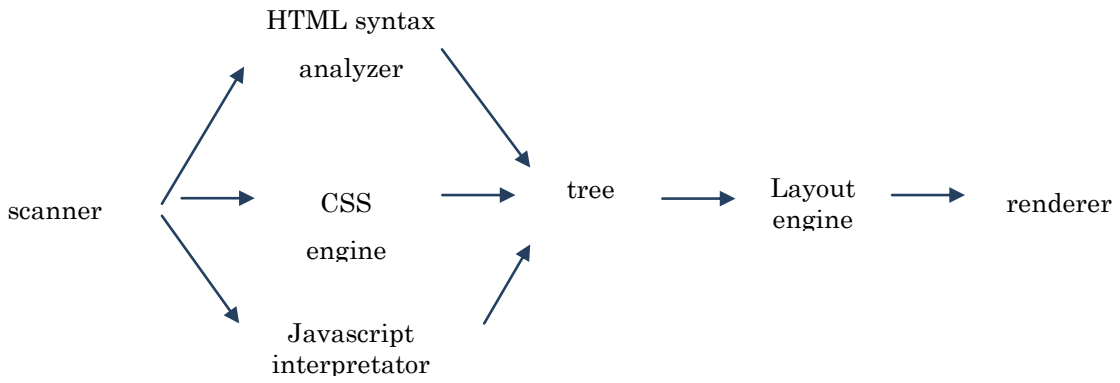

Fig. 2. Possible functional scheme of a browser

All browsers have at least three major subsystems: XML parser (with refinements for HTML/XHTML, SVG, CSS), the JavaScript interpreter and the Layout/Rendering Engine [Grosskurth Godfrey 2005]. But all major browsers use their own, independently developed HTML/CSS/JavaScript parsers, layout and rendering engines.

Table I. Layout and JavaScript engines in some major browsers

BROWSER	Layout Engine	JavaScript interpreter
FIREFOX	Gecko	SpiderMonkey
CHROME	Blink (developed from WebKit)	V8
INTERNET EXPLORER	Trident	Chakra
MICROSOFT EDGE	EdgeHTML	Chakra
OPERA	WebKit	V8
SAFARI	WebKit	JavaScriptCore

This variety creates many questions. Performance of some components may be rather different [Jen Looper 2015]. How this influences browser's overall performance, in what order are applied CSS rules, weather determining attributes of elements of HTML document is better from HTML text or from JavaScript, levelling browsers build-in defaults (e.g. different built-in margins) – these and many other practical important problems for web developers are mostly unexplained in documentation.

The main factor affecting webpage opening speed is network speed. While this is the most important factor, page designer possibilities to change network configuration and/or servers are limited.

But there is also another important factor affecting page's opening speed – its graphics. HTTP Archive shows that in average images/animations/video make up 64% of a Web page's total size [Shull 2015]. Suitable organization of page graphical content, using best formats for animations, most effective technologies for images and video attributes (e.g. transparency) is of utmost importance for achieving fluid page presentation, essential for commercial web applications and games.

2.1 Graphic modes of browsers

With HTML5 browsers use two graphic modes for processing and rendering images: the Retained Mode and the Immediate Mode.

Retained mode was historically the first and is used e.g. with images presented by HTML `<img>` tag. This is for browser not a direct command 'start drawing', but a command 'show this image in page'. Where on page, over/under of which other page elements (sometimes also if/when) remain to for browser to decide.

Suppose browser finds in HTML-text and CSS stylesheet lines:

```
<img id=sun src=images/sun.gif>
      #sun{
            position:fixed;
            left:0;
            top:0;
            z-index:1
      }
```

The image is added to data structure (display list) for the all objects which should be displayed on page and layout engine decides, where and in which order (i.e. what appears on top of what) should be displayed on screen. Besides HTML `<img>` tag this is the display mode also for CSS background images, thus the most often used mode for displaying graphics in browser; for displaying are used rather complicated data structures (the display list) and layout algorithms.

The immediate mode was introduced in HTML5 with new element `canvas`. Canvas is an area on screen (html5 document window) where JavaScript commands draw directly; browser does not change anything, but sends the whole canvas as it is drawn by JavaScript to screen. For drawing on canvas can be used several interpreters of graphic commands, e.g. CanvasRenderingContext2D (2D graphics) allows to draw 2D graphical objects – lines, rectangles, circles, text, but there are also 3D contexts interpreting WebGL 3D commands. The major difference is that here programmer has himself to decide where to place the object and when to draw it. Since all digital screens are constantly redrawn, this enables animations.

To continue analogy with translators - the retained mode can be compared with compilers (everything is first computed and then used repeatedly, in every screen redraw), the immediate mode – with interpreters (what is computed is also used/executed at once, on next screen redraw).

The Cascaded Stylesheets language (CSS) appeared as a simple tool to format text in HTML documents. In next versions CSS2 and CSS3 its functionality has been significantly extended to cover also manipulating images (image placement, rotation, transparency, pixel noise, animation) and sound. Thus e.g. the latest version CSS3 allows to create rotating Earth and Moon rotating around the Earth using only CSS3, i.e. in the retained mode of browser. The main drawback (until now) is that CSS does not have 'real' variables – variables, which could get values from DOM (Domain Object Model – the data structure keeping all page elements) attributes. The preprocessor variables and CSS custom properties are a closed system accessible only from CSS and were in our tests not used.

2.2 Browser's graphic possibilities

HTML5 allows to implement animations using several formats and technologies:

- showing animated GIF images either as a part of HTML-document (i.e. with HTML-code) or drawing with JavaScript; this old image format contains series of frames for storing short animations but is restricted to 8 bpp color resolution (i.e. image can have max 256 colors) and animation speed cannot be controlled by browser, it is pre-set when creating the GIF animation file; but animation (image) is easy to scale (make smaller, making it bigger destroys quality); this is similar to showing video in WebM format – browser does not have control, but shows sequence of already rendered frames;
- animation on HTML5 canvas with JavaScript (i.e. using immediate mode): showing/moving images, possibly clipping them;

- procedural texture – merging texture images changing their opacity [Cloud Procedural texture]; here this was used to produce Sun's lava texture from only one image;
- CSS3 can animate object's position, scale, rotation and opacity [Paul Lewis, Paul Irish 2014]; it also allows to create frame-based sprite sheet animation without using JavaScript;
- SVG; SVG elements can be manipulated like HTML elements using transform functions, but many commands and attributes do not work the same way on SVG elements as they do on HTML elements, JavaScript feature detection fails, the local coordinate system of an element works differently for HTML elements and SVG elements, the CSS properties of SVG elements have different names, e.g. instead of background-color should be used fill etc.

For web developers and browser game authors is essential to know, how selection of some presentation format influences the overall performance. Thus on the game developing course presented by the first author in Tallinn University of Technology we decided to create series of 'real-world' animation tests to compare various technologies/formats both in Retained Mode and in Immediate Mode.

Graphics rendering in browsers has been tested also earlier ([Roast 2013], [Vladić et al 2012], [MotionMark 2016]), but these tests had different aims, they did not compare influence of different graphic and animation design formats to animation rendering speed.

2.3 Concerns of Web Developers

One of the most important issues for web page designers is page opening speed. Research shows, that users form an opinion about web page visual appeal already in 50 milliseconds [Lingaard et all 2006].

For commercial web applications - web store fronts, product advertisements etc. this may be crucial. Google research shows, that 53% of mobile users abandon sites that take over 3 seconds to load [Doubleclick 2016]. Another recent study [Kissmetrics 2011] found that:

- 47% of consumers expect a web page to load in 2 seconds or less;
- 40% of people abandon a website that takes more than 3 seconds to load;
- one second delay in page response may result in a 7% reduction in conversions;
- if an e-commerce site is making $100,000 per day, one second page delay could potentially cost you $2.5 million in lost sales every year.

Especially intense is this problem in mobile, web and cloud programming. Here a programmer is has minimal possibilities for understanding inner workings of code and debugging, but the code should not only work, but should be memory-efficient and work quickly.

It has been claimed that Internet user's attention span is all the time diminishing. Although recently this claim become dubious [Maybin 2017], understanding factors which influence page opening speed is of utmost importance for web developers.

2.4 Artificial aids: Shims, polyfills, jQuery

Development of web languages – HTML(5), CSS, JavaScript should be based on standards established by World Wide Web Consortium (W3C) [W3C 2017.Standards]. But browsers and their subsystems – HTML and CSS parsers, JavaScript engine etc. are developed by different organizations, thus features what and how they implement are often different, e.g. for quite a long time Microsoft browsers (IE6..IE9) did not follow W3C standards, but tried to introduce their own functionality and syntax.

In order to make browsers to behave (more or less) the same way so that content on users screen would appear similar in whatever browser is used were introduced many JavaScript libraries under different names – pre-processors, frameworks, shims, polyfills etc. They introduced for browsers which originally did not have some functionality or syntax the missing functionality with JavaScript. Best known is the jQuery, which was introduced to make Microsoft browsers IE6..IE9 similar to standards-obeying ones, but by now contains lot of additional features - animation, physics, fade-ins and fade-outs (the 'visual sugar') etc.

With rapid updates of browsers JavaScript frameworks intended to enrich their functionality are quickly becoming obsolete. Besides, these 'functionality enrichment' frameworks introduce a nontransparent layer of JavaScript code (often minified, i.e. made difficult for humans to read), which makes JavaScript debugging very difficult. The original reason for jQuery has vanished (Microsoft browsers IE10 and Edge are already more or less following standards), but jQuery is still sometimes considered almost as a standard, therefore in one test jQuery was also used. The jQuery library was introduced to make Microsoft browsers IE6..IE9 to understand standards, but currently Microsoft has also started to follow them, so jQuery is (mostly) not needed. jQuery introduces rather cryptic, difficult to understand syntax (what has to be learned) and is changing custom semantics. For instance, a cryptic jQuery command to get canvas object:

var $canvas = $('#canvas');

returns array (#canvas suggests, that there are several canvas objects having the same id?!); equivalent to this, but more understandable plain JavaScript command

```
var $canvas = document.getElementById('canvas');
```

returns 'flat' variable, thus uses of these variables also require different syntax.

Use of jQuery also increased memory requirements (as measured in IE 11). It was relatively easy to remove all dependencies of jQuery - only 8 lines had to be changed to convert the test script into 'clean' JavaScript without use of jQuery.

3 TESTS

All tests implemented the same animation - Moon Eclipse (actually happened during the course on 20.03.2015), using different formats and technologies. The scene contains two animated objects: Sun with animated lava texture and rotating Earth. Since handling transparency has been for browsers problematic for a long time (see and compare e.g. discussions [CSS Transparency Settings for All Browsers 2007], [Cross Browser Opacity 2014]), in tests were used also three objects with different transparency/opacity – Moon's shadow on Earth surface, Earth's air - the white half-transparent halo around Earth – and Earth night side – gradient overlay changing from transparent (opacity = 1) to grey (opacity = 0.2). All test animations eclipse1.htm..eclipse9.htm were HTML5-documents looking similar on screen, but implemented using different technologies/formats in order to test their performance, i.e. time required to show fixed-length animation.

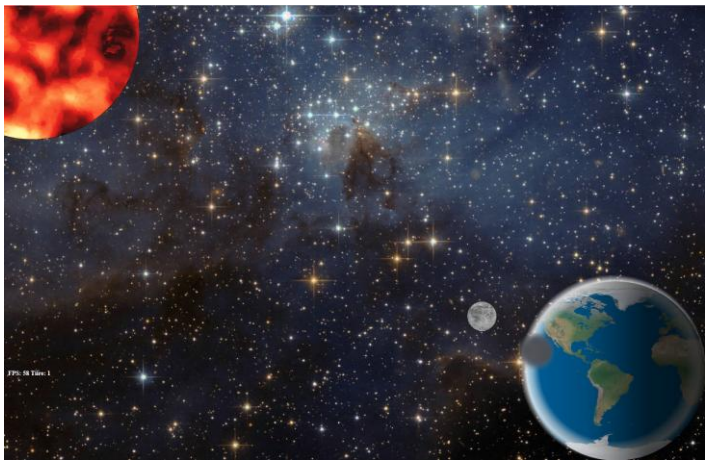

Fig. 3. The test application and its animated objects: Sun (in upper left corner with animated lava texture), Earth (in lower right corner, rotating), Moon (small grey circle close to earth), Moon shadow (half-transparent grey circle on Earth surface), Earth atmosphere (light half-transparent halo surrounding Earth), night (right) side of Earth.

3.1 Test files

In the following table are described test files T1..T9; they all are available online with summary of results [Henno 2016].

Table II. Test files and their structure

Test files	Description
T1: eclipse1.htm	The whole screen is covered with canvas having cosmos as the CSS-defined background image; Earth is animated with JavaScript (texture is constantly moved behind a clipping circle, drawn on canvas by JavaScript); all other objects are images in HTML document placed using CSS attribute z-order over the canvas: Sun is animated GIF (32 frames) with transparent background, Moon, Moon shadow, Earth atmosphere are images, 50% transparency of Moon shadow, Earth atmosphere and day-side of Earth is 'built-in' to images in .png format with Photoshop (is not adjusted in the HTML-document); placement of images is defined with CSS.
T2: eclipse2.htm	Sun and Earth are as in previous, Earth Air and day-night mask are DIV-s, their transparency and clipping are defined with CSS rules
T3: eclipse3.htm	The whole screen is a DIV with cosmos as the CSS-defined background image; Sun is animated GIF, minimal canvas is used only behind Earth, which is animated with JavaScript (texture is constantly moved behind a clipping circle drawn by JavaScript using the 2D-graphics context of canvas); all other objects are images in HTML document placed using CSS attribute z-order over the canvas, 50% transparency of Moon shadow, Earth atmosphere and day-night mask is defined in Photoshop
T4: eclipse4.htm	As previous, but 50% transparency of Moon shadow and Earth atmosphere images is defined with CSS rules
T5: eclipse5.htm	The whole screen is a series of DIV-s (no canvas); the main DIV with cosmos as the CSS-defined background image covers the whole screen, smaller DIV-s for Earth, Moon, Moon shadow and Earth atmosphere images are placed over it using the CSS position, width/height and z-order attributes; Sun is animated with CSS3 rules (the 32 frames of the sprite sheet were obtained from the animated GIF), Earth is also animated with CSS (texture is constantly moved behind a CSS clipping circle); transparency of Moon shadow and Earth atmosphere images is defined in Photoshop
T6: eclipse6.htm	The whole screen is a series of DIV-s (no canvas); the main DIV with cosmos as the CSS-defined background image covers the whole screen, smaller DIV-s for Earth, Moon, Moon shadow and Earth atmosphere images are placed over it using the CSS position, width/height and z-order attributes; Sun is video in WebM format (defined by <source src="images/sun.webm" type='video/webm;codecs="vp8, vorbis"'/>), the video is clipped by CSS clipping circle (works correctly only in Firefox; Chrome has its own proprietary format for alpha transparency in WebM-video); Earth is animated with CSS (texture is constantly moved behind a CSS clipping circle); transparency of Moon shadow and Earth atmosphere images is created in Photoshop
T7: eclipse7.htm	Sun texture is procedurally generated by JavaScript and `jQuery` on minimal canvas using two additional canvases (the idea from [Professor Cloud 2013]); all other elements are as in previous example
T8: eclipse8.htm	As in previous but `jQuery` library (253 kb) was removed
T9: eclipse9.htm	Texture of Sun is procedurally generated (without `jQuery`), Earth is JavaScript animation on a separate canvas, thus together there are 4 canvases

3.2 Animation technologies

All tests used two objects animated with different technologies: Sun (bubbling lava texture) and rotating Earth.

For Sun was used frame-based animation. In tests T1..T6 browser gets pre-rendered frames (from animated GIF, WebM-video or spreadsheet; in tests T7..T9 Sun's texture is procedurally generated (using additional canvases).

Earth rotation was created dragging Earth texture from left to right, but showing only a part of it in clipping circle - screen's 'hole' to see through. In tests T1..T4 the clipping circle was created on canvas with JavaScript, in tests T5..T9 was used similar feature available in CSS3.

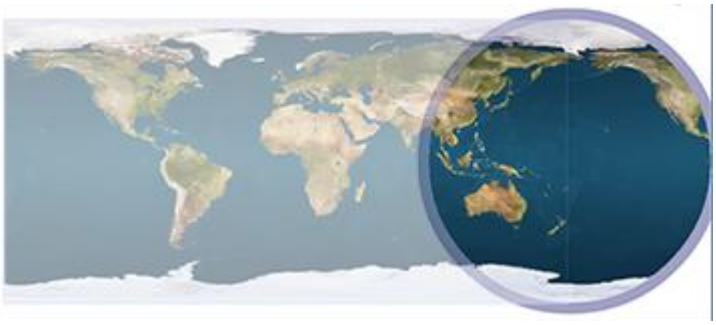

Fig. 4. Creating rotating Earth with dragging Earth layout behind a clipping cirle.

In tests were also changed ways to create half-transparent images - Moon shadow, Earth air and Earth day-night half-transparent overlay. We wanted to compare transparency/opacity adjustment in browser using JavaScript and CSS (i.e. browser has to calculate alpha mask) versus pre-set transparency in .png image. The Earth Air (in immediate mode) was a div with radial gradient on background; the Earth day-night overly was created by linear clipped by a SVG circle.

4 RESULTS

All test files had a small JavaScript script, which measured the time what was used to run 5 rotations of the Earth; the results were shown on screen and stored using the HTML5 local storage feature. For measuring memory performance there are not yet general standards. In Chrome is available a proprietary method `performance.memory` [Colt McAnlis 2013], but in our tests Chrome reported always the same values. IE allows to see some memory statistics with the UI Responsiveness tool [Microsoft. Improving UI Responsiveness]:

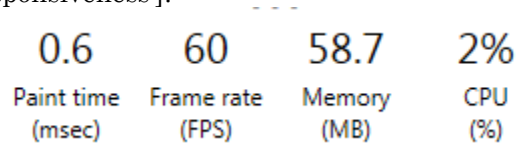

Fig. 5. The window of the Microsoft UI-responsiveness tool

The values produced by this tool varied 4-10% and therefore are not presented here; the only more or less constant change was 3-4% increase in used memory when the jQuery library was used (the test eclipse8.htm).

In order to eliminate computer speed and network latency, all tests were done locally under a local Wamp server.

These tests produced lot of numbers. In order to compare influence of different formats results were normalized, using the `eclipse1.htm` in Firefox as the control case, i.e. in the following table the first number is `time(Ti)` - total time for test `Ti` with this browser and the second – percentage of this time of the 'etalon' time, i.e. calculated with formula `time(Ti)*100/timeFF(T1)`, where `timeFF(T1)` is the time reported for test `T1` by Firefox.

Table III. Results for desktop PC (Windows 7 64 bit)

Test	**Browser**				
	FF 51	***Chrome 55***	***IE 11***	***Edge***	***Opera***
T1	66773ms 100%	68281ms 102%	60008ms 98%	65097ms 98%	65046ms .97%
T2	66744ms 99%	68230ms 102%	59996ms 98%	66007ms 98%	65070ms 97%
T3	66755ms 99%	67874ms 101%	60036ms 99%	66764ms 99%	65065ms 97%
T4	66720ms 99%	68016ms 102%	60032ms 99%	67065ms 100%	65065ms 98%
T5	20011ms 29%	20004ms 29%	CSS clipping with circle does not work	CSS clipping with circle does not work	19974ms 29%
T6	20023ms 29%	19837ms 29%	IE 11 does not play WebM video, CSS clipping does not work	-CSS clipping with circle does not work	20010ms 29% no video clipping
T7	20010ms 29%	19999ms 29%	CSS clipping does not work	CSS clipping with circle does not work -	20006ms 29%
T8	20010ms 29%	19992ms 29%	CSS clipping does not work	CSS clipping with circle does not work -	20004ms 29%
T9	66721ms 99%	67070ms 99%	59626ms 98%	60101ms 99%	64057ms 97%

These tests were performed also with a mobile phone browsers in an android mobile (LG E975a, Android 4.4.2 'KitKat'). In mobile were used the phone's OS built-in browser, Chrome and UC cloud browser (made in China), which currently is a 'rising star' in the landscape of mobile browsers [StatCounter 2017]. In the following table are presented the raw results (test times in milliseconds) and results after normalizing using the Chrome browser as an etalon.

Table IV. Results of tests in mobile browsers

Test	Browser		
	Chrome	*LG OS-browser*	*UC browser*
T1	66972ms 100%	80060ms 119%	78215ms 117%
T2	66882ms 100%	83453ms 125%	98413ms 145%
T3	66973ms 100%	85029ms 124%	69619ms 127%
T4	67308ms 100%	82938ms 124%	85169ms 127%
T5	19894ms 30%	20060ms 30% – no clipping	19109ms 29% no clipping
T6	19835ms 30%	No WebM	19930ms 30% no clipping
T7	19838ms 30%	- no clipping	19129ms 29% no clipping
T8	19993ms 30%	- no clipping	19835ms 30% no clipping
T9	68086ms 100%	- no clipping	71027ms 106%

5 CONCLUSIONS

The performed tests allow to draw several conclusions:

- there are no essential differences in speed between major browsers, but Microsoft browsers do not (yet) implement CSS3 (only CSS2);
- HTML5 immediate mode (canvas+JavaScript) allows to create complicated animations, but reduces animation speed ca three times (tests T5,T6,T7 did not use canvas); this result was a bit surprising, since canvas is commonly considered the main element in all graphics-intense web applications (games, portals etc.) but becomes understandable if one thinks what actually is loaded as the canvas 2D context – this is an interpreter of 2D graphic commands, which has to build its own name table etc.; returning to the analogue with translators: compiled code (i.e. retained mode) is quicker than interpreted (immediate mode);
- using several canvases does not make application slower (test T9) – they all use the same 2D context, i.e. graphics interpreter;
- creating transparent masks with CSS and changing their opacity (also opacity bitmaps – Moon shadow) with JavaScript does not make application slower and allows better control of result (tests T2, T4);
- CSS3 animations and CSS3 clipping (with circle) are quick, but difficult to scale (changing size of frame-based CSS animation is very error prone) and did not work in Microsoft browsers IE10 and Edge;
- video in WebM format with transparent background (test T6) can be currently achieved (using CSS3 clipping) only in Firefox (Chrome has for this a proprietary extension [Sam Dutton 2016]);
- the speed of canvas animation depends essentially on size of animated objects – scaling page down decreased rendering time (we did some separate scaling tests);
- results of tests T7, T8, T9 indicate, that jQuery was officious – big, especially for mobile applications (current version 3.1.1 – 261 kB) and did not have any advantages.

REFERENCES

Web Developers Notes 2017. Browsers List. Retrieved June 3, 2017 from http://www.webdevelopersnotes.com/browsers-list
Brave 2016. Retrieved June 3, 2017 from https://brave.com/
Microsoft Edge. Retrieved June 3, 2017 https://www.microsoft.com/en-us/windows/microsoft-edge
Vivaldi. Retrieved June 3, 2017 from https://vivaldi.com/?lang=en_US
Wikipedia 2016. Usage share of web browsers. Retrieved June 3, 2017
https://en.wikipedia.org/wiki/Usage_share_of_web_browsers

A. Prakash 2016. Browser Based Operating Systems Reviewed. Retrieved June 4 2017 from https://tech-tweak.com/browser-based-operating-systems/

Alfred V. Aho, Jeffrey D. Ullman 1972. Theory of Parsing, Translation and Compiling. Prentice-Hall 1972, 542 pp, ISBN-13: 978-0139145568

Tali Garsiel, Paul Irish 2011. How Browsers Work: Behind the scenes of modern web browsers. https://www.HTML5rocks.com/en/tutorials/internals/howbrowserswork/

Spyros Doulgeridis 2017. How Browsers Work: Behind the Scenes. Retrieved June 4 2017 from https://dzone.com/articles/how-browsers-work-behind

Chromium (Google Chrome) 2017. Retrieved June 4 2017 https://www.openhub.net/p/chrome

Chris Hoffman 2014. What's the Difference Between Chromium and Chrome? Retrieved June 4 2017 from https://www.howtogeek.com/202825/what%E2%80%99s-the-difference-between-chromium-and-chrome/

Firefox 54.01.1 source code. Retrieved June 3, 2017 from https://archive.mozilla.org/pub/firefox/releases/54.0.1/source/

Grosskurth, Godfrey 2005. A Reference Architecture for Web Browsers. ICSM '05 Proceedings of the 21st IEEE International Conference on Software Maintenance, pp 661-664

Jen Looper 2015. A Guide to JavaScript Engines for Idiots. Retrieved June 4 2017 from http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/

Cloud Procedurel texture. Retrieved June 3, 2017 from https://doc.babylonjs.com/extensions/cloudproceduraltexture

Paul Lewis, Paul Irish 2014. High Performance Animations. Retrieved June 3, 2017 from https://www.HTML5rocks.com/en/tutorials/speed/high-performance-animations/

Roast 2013. CarvasMark. HTML5 Canvas Rendering benchmark. Retrieved June 23, 2017 from http://www.kevs3d.co.uk/dev/canvasmark/

Vladić et al 2012. Gojko Vladić; Neda Milić; Željko Zeljković; Darko Avramović. Evaluating Web browser graphics rendering system performance by using dynamically generated SVG. Journal of Graphic Engineering and Design. 2012;3(1):15-22

MotionMark 2016. Jon Lee, Said Abou-Hallawa, Simon Fraser. MotionMark: A New Graphics Benchmark. Retrieved June 23, 2017 from https://webkit.org/blog/6943/motionmark-a-new-graphics-benchmark/

CSS Transparency 2017. Retrieved June 3, 2017 from https://css-tricks.com/css-transparency-settings-for-all-broswers/

Cross Browser Opacity 2014. Retrieved June 3, 2017 from https://css-tricks.com/snippets/css/cross-browser-opacity/

Henno 2016. Eclipse. Retrieved June 3, 2017 from http://deepthought.ttu.ee/users/jaak/slideshow/

Colt McAnlis 2013. Static Memory Javascript with Object Pools. Retrieved June 3, 2017 from https://www.HTML5rocks.com/en/tutorials/speed/static-mem-pools/

Microsoft. Improving UI responsiveness. Retrieved June 3, 2017 from https://msdn.microsoft.com/en-us/library/dn255009(v=vs.85).aspx

StatCounter 2017. Browser Market Share Worldwide. May 2016 to May 2017. Retrieved June 3, 2017 from http://gs.statcounter.com/

Sam Dutton 2016. Alpha transparency in Chrome video. Retrieved June 3, 2017 from https://developers.google.com/web/updates/2013/07/Alpha-transparency-in-Chrome-video