Tests of graphics rendering in browsers

Tests of Graphic 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 most commonly used software, important communication tool to access our data-driven, increasingly visual Internet-based ecosystem. 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 produce graphics '. For creating visual images browsers use two modes: Retained Mode and Immediate Mode. In order to understand differences in these modes, especially in differences in speed, we created nine different versions for animation of Lunar Eclipse on 20.03.2015 and tested them 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 Retained Mode (images inserted in html text or animated with CSS3) is at least twice quicker than Immediate Mode (Javascript animation on Canvas) in all tested browsers.

Categories and Subject Descriptors: • **Information systems~Browsers** • **Computing methodologies~Computer graphics** • **Computing methodologies~Graphics file formats**

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

Additional Key Words and Phrases: HTML5, CSS3, JavaScript

1 INTRODUCTION

The first program what (nearly) everyone open when sitting behind a computer, starting a tablet or internet-enabled phone is browser. Browsers have become the most common communication/interaction tool for all Internet-connected Mankind - more than half of whole Mankind. Currently (spring 2017) there available (for PC-s) more than 100 different browsers [Web Developers Notes 2017] with different features – built-in e-mail, add-blocking, torrent downloading and streaming etc. About quarter of these are already discontinued (but mostly still available), but every year appear some new ones, e.g. in 2016 : 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 [Wikipedia 2016. Usage share of web browsers] or for some users they may be very important, thus web developers 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 documents more attractive. Current browsers present several technologies for creating animations, but little is known about their efficiency. In the following are analyzed browser's graphic possibilities and then created series of tests to compare speed of different possibilities for creating browser animations.

2 BROWSER AS A TRANSLATOR

Browser has become the major mechanism for presenting digital content on screen. Only very specialized programs (Photoshop, Excel) or big games still use their own windowing systems but even those have already many browser-based analogues. Browser is the first program what we usually open and browsers are 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/syntax of input channels into format/syntax required by processor, graphics processor and other output devices. But its task is essentially more complex. Browser accepts input from several

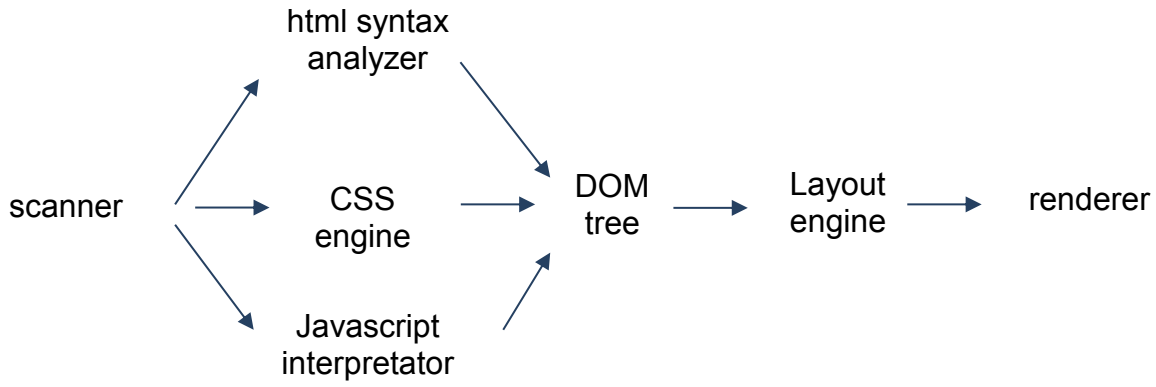
channels – Html/Css/JavaScript/SVG text with links to other sources, images, video, sound tags, keyboard and mouse or/and touch input; input may be provided from several files – the Html-document itself and data from linked external sources, e.g. external CSS (Cascading Style Sheets) and JavaScript files, images, sound/video files. With these different types of inputs browsers compile complex output for computer/mobile screen, create voice and play sound and video.

A browser-based interactive media is essentially a frame-based video. Processor calculates next frame content and accompanying sounds, from these bits is then rendered text/images on screen and played sounds/music. For best results developers want maximal color depth (bpp – Bits Per Pixel, determines color quality) and framerate (fps – Frames Per Second), which determines animation smoothness and game responsiveness. But both these features increase processor load, memory use and decrease speed, thus may make animation jumpy or sound disrupted.

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

SCANNER → SYNTAX ANALYZER → PARSER → ABSTRACT TREE → CODE GENERATION

There is no 'browser theory' and established structure (yet) for browsers, which are developed by different organizations and use different subsystems. The functional scheme of browser is more complex, e.g. can be presented as



But the 'inner workings' even of major browser engines are for web developers 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 much; 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 (for Windows and Mac), Crash and Error Reporting etc. [Chris Hoffman 2014], thus the codebase is essentially larger.

All major browsers use their own, independently developed html/CSS/JavaScript parsers, layout and rendering engines.

Table. 1. 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 creates many questions. Performance of some components may be rather different [Jen Looper 2015], how this influences the overall performance, in which order are applied CSS rules, if determining DOM tree element attributes is better from HTML text or from JavaScript, levelling browsers build-in defaults (e.g. different built-in margins) – these and many other practical issues present problems for web developers and are mostly unexplained in browsers documentation.

Usually page opening speed is connected with network speed. While this is the most important factor, page designer possibilities to change network configuration and/or servers are minimal.

However, there is another important factor behind page 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, using most effective technologies to influence images display (e.g. transparency) etc. is of utmost importance for achieving fluid page presentation.

2.1 Browser graphic modes

After arrival of HTML5 browsers use two graphic modes for processing and rendering images: Retained Mode and Immediate Mode.

Retained mode was historically the first and is used e.g. for images presented by html `<img>` tag. In retained mode this is for browser not a direct command 'start drawing', but a command 'show this image in page', but where on page (sometimes also if/when) are left to decide for browser.

Suppose browser scans in html-text and CSS style sheet lines

```
<img id=moon src=images/moon.png>
      #moon{
            position:fixed;
            left:50%;
            top:50%;
            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, rendered on screen. Besides html `<img>` this is the display mode also for CSS background

images, thus the most often used mode of graphics display in browser, but this has to use rather complicated data structures (the display list) and layout algorithms.
The immediate mode was introduced with HTML5 new element *canvas*. Canvas is an area on screen (page's window) where JavaScript commands draw directly. For drawing on canvas can be used several interpreters for graphic commands, e.g. CanvasRenderingContext2D (2D graphics) allows to draw 2D graphical objects – lines, rectangles, circles, text. The major difference is that here programmer has itself to decide where to place the object and when to draw it. Since all digital screens are redrawn regularly (e.g. PC srveen – in 60 times/sec, i.e with frequency 60 HZ), this is a direct way to create animations. For instance, the following Javascript function sets the moon to rotate around Earth (without using the browser's display list and layout algorithms):

```
function animate(){
      ctx.clearRect(0,0,winW,winH); //winW,winH – window width, height
      earth.draw();
      moonAngle += moonOrbitSpeed;
      moonAngle = moonAngle % (Math.PI);
      moonX = earth.x - moonOrbitR*Math.cos(moonAngle)-moonR;
      moonY = earth.y +  moonOrbitR*Math.sin(moonAngle)-moonR; //hakkab tõusma
      moon.style.left = String(moonX)+"px";
      moon.style.top = String(moonY)+"px";
      requestAnimationFrame(animate); } //restart the function
```

To compare analogue with translators - the retained mode can be compared with compilers (everything is first computed and then used repeatedly), immediate mode – with interpreters (what is computed is at once also executed).

In recent years have been significantly extended possibilities of CSS, the latest version CSS3 allows to create rotating Earth using only CSS3 (i.e. in the retained mode). The main drawback (until now) is, that CSS does not have 'real' variables – variables, which could get values from DOM 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 on canvas; this old image format contains series of frames for storing short animations, is restricted to 8 bpp (bits per pixel) color resolution (i.e. image can have max 256 colors) and animation speed cannot be controlled by browser, it has to be 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 webm-video – 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 to some figure;
- procedural texture – merging texture images changing their opacity [Professor Cloud 2013]; 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 a game developing course presented by the first author we decided to create series of 'real-world' animation tests.

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 - users leave the site, e.g. 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;
- A 1 second delay in page response can result in a 7% reduction in conversions;
- If an e-commerce site is making $100,000 per day, a 1 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 working with minimal resources for understanding inner workings, debugging, but at the same 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 (commercial) web developers.

Web is a visual media and the most important factor influencing page's speed (besides network) is the page's graphics and animations.

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 may be different, e.g. for quite a long time Microsoft browsers (IE6..IE9) did not follow W3C standards, but introduced their own functionality and syntax.

In order to satisfy all the time growing needs of web developers, all major browsers are adding/proposing new features to their browsers and its input languages in frantic temp. All major browsers – Google Chrome, Microsoft Internet Explorer (IE), Mozilla's Firefox (FF), Opera - publish several major updates per year, minor updates appear in every 6-8 weeks and most of them are already 'evergreen' - they automatically update themselves.

In order to leverage this soup were introduced many Javascript libraries under different names – pre-processors, frameworks, shims, polyfills, which introduced missing functionality or syntax with JavaScript functions. Best known among them is the jQuery, which was introduced to make Microsoft browsers similar to standards-obeying ones, but by now contains lot of other functionality - animation, physics, fade ins and fade outs (a 'visual sugar') etc.

But all browsers are all the time improving themselves and their update rate is frantic. All major browsers – Google Chrome, Microsoft Internet Explorer (IE), Mozilla's Firefox (FF), Opera - publish several major updates per year, minor updates appear in every 6-8 weeks, thus all these JavaScript frameworks are quickly becoming obsolete; besides, they introduce a nontransparent layer of Javascript code)often minified), which makes debugging very difficult. Therefore in the following tests of these libraries was used only jQuery (in one test).

3 TESTS

All tests implemented the same animation - Moon Eclipse (actually happened during the course on 20.03.2015), using different formats/technologies for its elements. The scene contains two animated objects: Sun with animated lava texture and rotating Earth and two objects with different transparency/opacity – Moon's shadow on Earth surface and Earth's air (the white circle around Earth). All test animations eclipse1.htm..eclipse9.htm were HTML5-documents looking similar on screen, but since implemented using different technologies/formats in order to test their performance, - i.e. time required to show fixed-length animation.

Fig. 1. The test application (captured from PC screen) and its animated objects: Sun (upper left corner with animated lava texture), Earth (lower right corner, rotating), Moon (small grey circle close to earth), Moon shadow (larger half-transparent grey circle on Earth surface), Earth atmosphere (light half-transparent halo surrounding Earth).

3.1 Test files

In the following table are described test files T1..T9; they all are available (with summary of results) at http://deepthought.ttu.ee/users/jaak/slideshow/.

Table. 2. Test files and their structure

Test files	Animation description
T1: eclipse1.htm	The whole screen is covered with canvas with 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 image (32 frames) with transparent background, Moon, Moon shadow, Earth atmosphere – images, transparency is 'built-in' to images with Photoshop (is not adjusted in the HTML-document); placement of images is defined with CSS using position attribute value 'fixed'.
T2: eclipse2.htm	The whole screen is covered with canvas with cosmos as the CSS-defined background image; Earth is animated with JavaScript (texture is constantly moved left-to-right 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: Sun is animated .gif image (32 frames) with transparent background, transparency of Moon shadow and Earth atmosphere images is 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 image, 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 and Earth atmosphere images 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 image), 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

In 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.

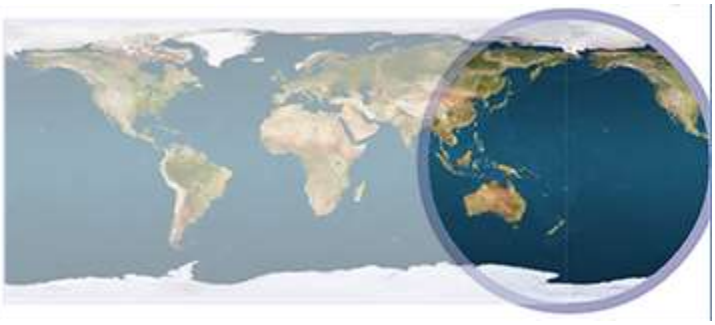


Fig. 2. Creating rotating Earth with dragging Earth layout behind a clipping cirle.

In tests were also changed ways to create half-transparent images (moon's shadow and Earth air), the issue what we wanted to consider was transparency/opacity adjustment in browser (i.e. browser has to calculate alpha mask) versus pre-set transparency in .png image.

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 in a separate small DIV and stored using the HTML5 local storage feature. For memory performance there is not yet general standards; in Chrome is

available a proprietary method `performance.memory` [Colt McAnlis 2013], but this can be used only if Chrome is started with specific switch and 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]:

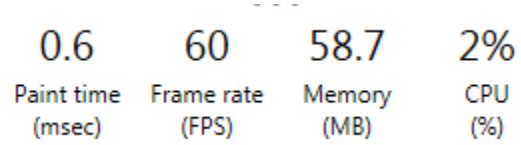


Fig. 3. The window of the Microsoft UI-responsiveness tool

The values produced by this tool varied 4-10% and therefore 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).

Thus the main measured parameters were time-based – the framerate FPS (Frames Per Second) and total time needed to execute a constant-size animation (5 rotations of Earth). A JavaScript script measured several parameters of test (FPS, time for one rotation, number of frames rendered) and at the end of animation showed the main result – total time for the animation (5 rotations of Earth) in milliseconds on screen (it was also stored using the HTML5 feature localstorage). 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 gain better understanding of these were results 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 this test Ti with this browser and the second – percentage of this time of the 'etalon' time, i.e. calculated with formula `time(Ti)*100/time(T1)`, where `time(T1)` is the time reported for test T1 by Firefox.

Table. 3. 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	-	20010ms 29% no video clipping
T7	20010ms 29%	19999ms 29%	CSS clipping does not work	-	20006ms 29%
T8	20010ms 29%	19992ms 29%	CSS clipping 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'); here 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 Chrome as the etalon.

Table. 4. 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%	-	19129ms 29% no clipping
T8	19993ms 30%	-	19835ms 30% no clipping
T9	68086ms 100%	-	71027ms 106%

5 CONCLUSIONS

The performed tests allow to draw several conclusions concerning the speed of different browsers and their graphic modes:

- there are no essential differences in speed between major browsers, but Microsoft browsers do not (yet) implement CSS3 (only CSS2)

– HTML5 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 2D graphics commands, which has to build its own name table etc. Using again 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);

- changing opacity of bitmaps with JavaScript does not make application slower and allows better to control of result (tests T2, T4)

- CSS3 animations and CSS3 clipping (with circle) are quick, but quite difficult to scale (changing size of frame-based CSS animation is very error prone) and did not work in Microsoft browsers

- 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 (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. The jQuery library was introduced to make Microsoft browsers IE6..IE10 to understand standards, but currently Microsoft has also started to follow them, so jQuery is (mostly) not needed. But jQuery introduces rather difficult to understand cryptic syntax (what has to be learned) and is and changing custom semantics, e.g. 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 all 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 - we actually had to change only 8 lines to convert the script into 'clean' JavaScript, where jQuery was not used.

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
WebKit 2017. WebCore Rendering I – The Basics. Retrieved June 4 2017 from https://webkit.org/blog/114/webcore-rendering-i-the-basics
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/
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/
Paul Lewis, Paul Irish 2014. High Performance Animations. Retrieved June 3, 2017 from https://www.html5rocks.com/en/tutorials/speed/high-performance-animations/
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