ImageJ in the web? - Image processing in the browser using HTML5

Kai Uwe Barthel and Karsten Schulz

HTW Berlin, Wilhelminenhofstraße 75A, 12459 Berlin, Germany

ABSTRACT

Up to now online image processing in the browser could be done in two ways: It could be achieved using Java applets or a Flash component. However both approaches needed a browser plugin, which is not available for all platforms. Now with the new HTML5 canvas element, which will soon be supported by most browsers, image processing tasks can be achieved using JavaScript only. This paper will explain how to use this new technique and will compare the performance with ImageJ. This might be the first step to a JavaScript version of ImageJ.

1. INTRODUCTION

Strictly speaking, this paper is not directly related to the ImageJ programming framework. However, we will show how the HTML5 canvas element can be used to perform image processing directly in the browser without any specific plugins or add-ons. The possibility to access and modify pixels within a browser introduces a new wide field of web applications, which might also be useful for the ImageJ community.

Further author information: (Send correspondence to Kai Uwe Barthel) E-mail: barthel@HTW-Berlin.de

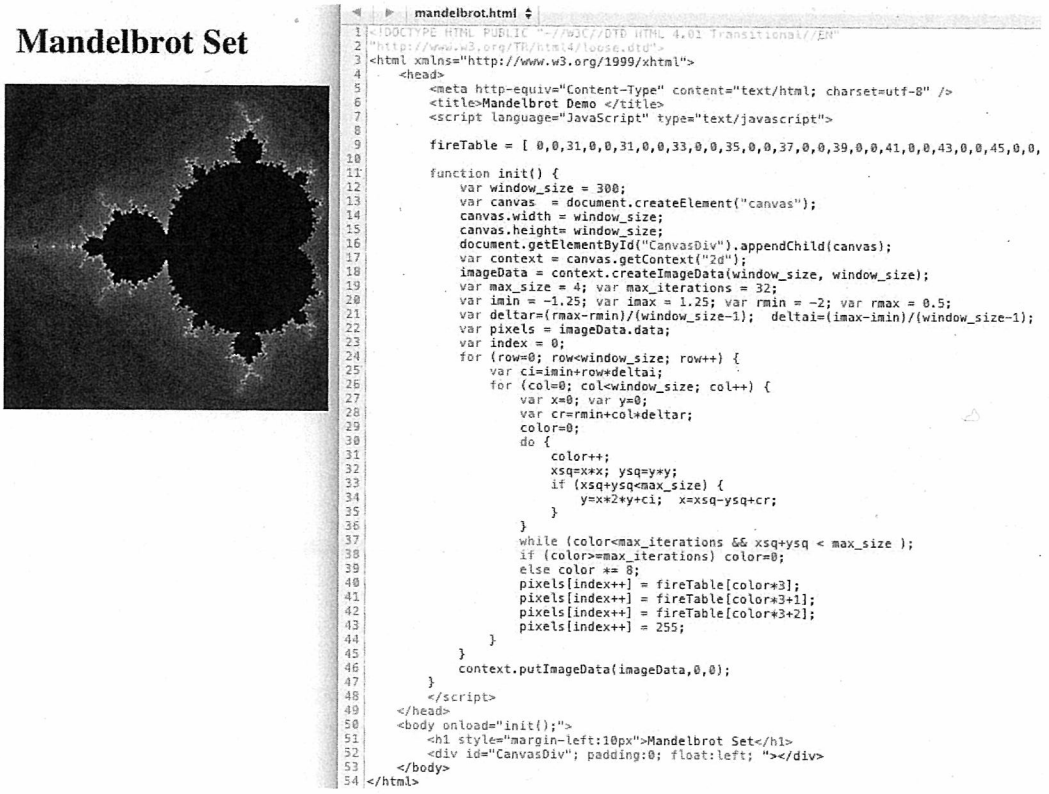

Figure 1. Example of the Mandelbrot set rendered with HTML5

In this paper we will show the basic principle to use a standard web browser for image processing. We will explain in detail a simple example that will lead the way to other more complex applications. For two typical image-processing tasks we will compare the performance with ImageJ's macro and plugin implementations.

2. IMAGE PROCESSING IN THE BROWSER

Up to now image processing in the browser of a client computer could only be achieved in two ways: One way was the usage of Java applets; the other possibility consisted in using ActionScript from Adobe Flash. Both approaches required having specific browser plugins. While these plugins are widely available for standard web browsers on regular computers, there is only little or no support for mobile devices such as the iPhone. In addition, setting up an applet or a Flash component usually requires more knowledge from the developer.

2.1 HTML5

HTML5 is the fifth major revision of the World Wide Web's core language to describe web pages [1]. Although already quite powerful, especially in connection with CSS, the vague subject that is referred to as "Web Applications" hasn't been adequately supported so far. The fifth version addresses this problem and other issues that have come up in the past and presents web programmers with new means to tackle problems in pure "HTML way" without third-party plugins or heavy server-side software. One of these means, which certainly is of special interest for the ImageJ community, is the canvas element. JavaScript functions that are embedded in or included from HTML pages are used to interact with the Document Object Model (DOM) of the page. JavaScript can be characterized as a prototype-based object-oriented scripting language that is dynamic, and weakly typed.

2.2 Canvas Element

The new canvas element describes a resolution dependent bitmap region in a web page, providing web programmers with a set of functions via a context structure to modify and manipulate its contents. The steps to manipulate are always the same:

1. Define a context element with width and height
2. Obtain a drawing context
3. Modify the canvas region with methods provided by the context

The W3C specs say the canvas element is "implemented and widely deployed". Although this is true for the element itself, this doesn't count for the types of contexts which can be accessed, namely, the 2D context and the WebGL context, specified by the Khronos Group [2]. Regarding the 2D context, the newest versions of Safari, Opera, Google Chrome and Firefox definitely support the following groups of drawing functions. (The announced new Microsoft Internet Explorer 9 will also support the canvas element.)

- 2D transformations
- Line and fill styles
- Gradients
- Bezier paths and quadratic curves
- Text rendering
- Image rendering
- Pixel access

The last group, that is the direct pixel access, is what makes it possible to perform client-side image processing in real time in the browser, offering a new platform to include some of ImageJ's functionality in web pages. A more detailed introduction to the canvas element can be found in [3].

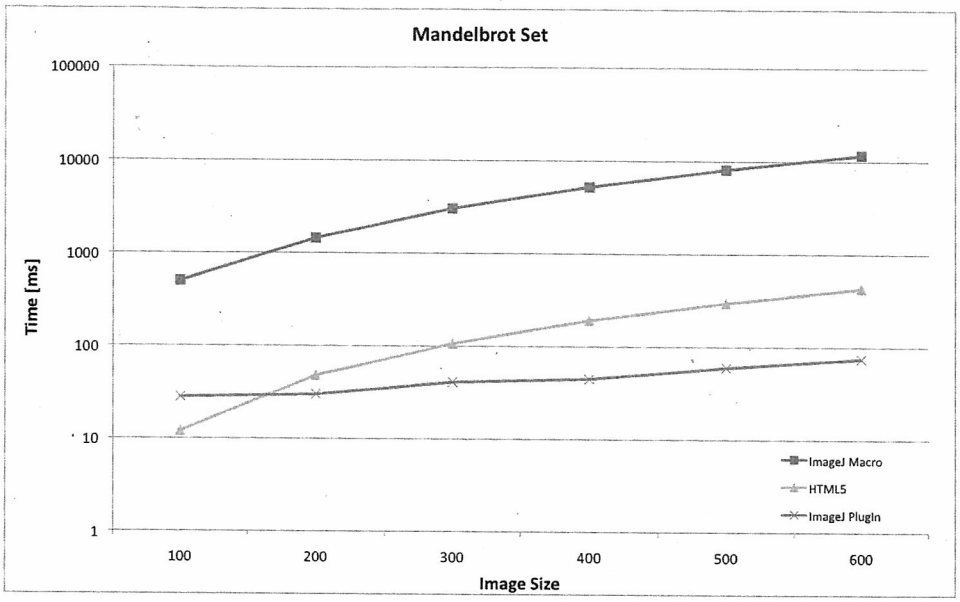

Figure 3. Computation time for the Mandelbrot set

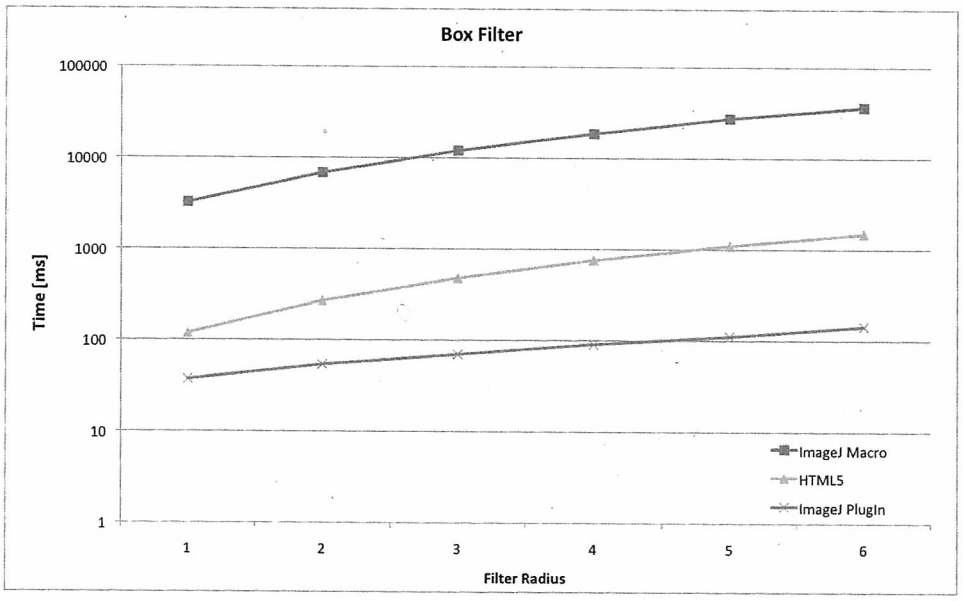

Figure 4. Computation time for the box filter

3. BENCHMARK TESTS

In order to compare the performance of HTML5 image processing with ImageJ we have developed two examples that are more complex than the one from the previous chapter. The first example was adapted from the ImageJ Mandelbrot macro [4]. The second example is a box lowpass-filter which replaces the center pixel with the mean value of the surrounding square of a given radius. For both examples an ImageJ plugin, an ImageJ macro and the corresponding HTML5-code were developed. The actual code was kept as a similar as possible.

Figure 3 shows the computation time needed when varying the image size of the Mandelbrot set. For the box filter experiment (figure 4) an image of 350x336 pixels was used. The filter radius was varied from 1 to 6 pixels.

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2  "http://www.w3.org/TR/html4/loose.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4      <head>
5          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
6          <script language="JavaScript" type="text/javascript">
7
8          function process() {
9
10             var image = document.getElementById("origImage"); // get image element from document tree
11             var width = image.width;
12             var height = image.height;
13
14             var canvas  = document.createElement("CANVAS"); // create new canvas element
15             canvas.width = width;                           // set canvas dimensions to image dimensions
16             canvas.height= height;
17             var context = canvas.getContext("2d");          // get the 2d context of the canvas
18             context.drawImage(image,0,0);                   // draw the image to the context
19
20             var imageData = context.getImageData(0, 0, width, height); // get the image data of the context
21             var pixels = imageData.data;                    // get the pixels
22
23             var length = width*height*4;                    // precompute the length of the pixel array
24             for(var index=0; index<length; ) {
25                 pixels[index] = 255-pixels[index++];        // compute the negative
26                 pixels[index] = 255-pixels[index++];
27                 pixels[index] = 255-pixels[index++];
28                 index++;
29             }
30
31             context.putImageData(imageData,0,0);            // set the new pixels to the context (canvas)
32             document.getElementById("negative").appendChild(canvas); // add canvas to the document tree
33         }
34         </script>
35     </head>
36
37     <body onload="process();">
38         <h1 style="margin-left:10px">Image processing using the canvas element </h1>
39         <img id="origImage" src="mandrill.png" style="border:1px solid black; margin:10px; float:left;"/>
40         <div id="negative" style="border:1px solid black; padding:0; float:left; margin:10px"></div>
41     </body>
42 </html>
```

Figure 2. A simple example calculating the negative of a color image

2.3 A simple example - Computing the negative of a color image

Figure 2 shows the example code to calculate the negative of a color image. We will explain the basic structure by indicating the line numbers. The HTML page has a heading (line 38) and shows an image (line 39). In addition there is a div tag with the id "negative" (line 40) that defines a section that will be used to display the inverted image. When this negative image has been calculated it will be shown next to the original image. After the HTML-page has been loaded (line 37) the JavaScript function process() will be called, where the image processing and rendering will take place.

The first thing that happens in the process() function is to get the image element from the document tree (line 10). The width and the height are read from the image element (lines 11+12). This information is used to create a new "CANVAS"-element with the same size as the image (lines 14 - 16). From this canvas element we get the 2d-context (line 17), which we will use for drawing and pixel access. In line 18 the image is drawn to this context. Next we get the imageData from the context (line 20) in order to be able to access the pixels (line 21).

Before starting the actual image processing, we precompute the length of the pixel array (line 23). If we used the expression width*height*4 in the for-loop, the code would be slower because - due to the nature of JavaScript - the length would always be recalculated. Next we invert the RGB-values of the pixels and leave the alpha value unchanged (lines 25-28). In line 31 the changed pixel values are written back to the context. Finally in line 32 the canvas element (containing the negative image) is added to the document tree (the div section of line 40). Now the inverted image will be displayed in the browser next to the original image. More complex image processing tasks such as convolutions that cannot be computed in-place would require two canvas objects.

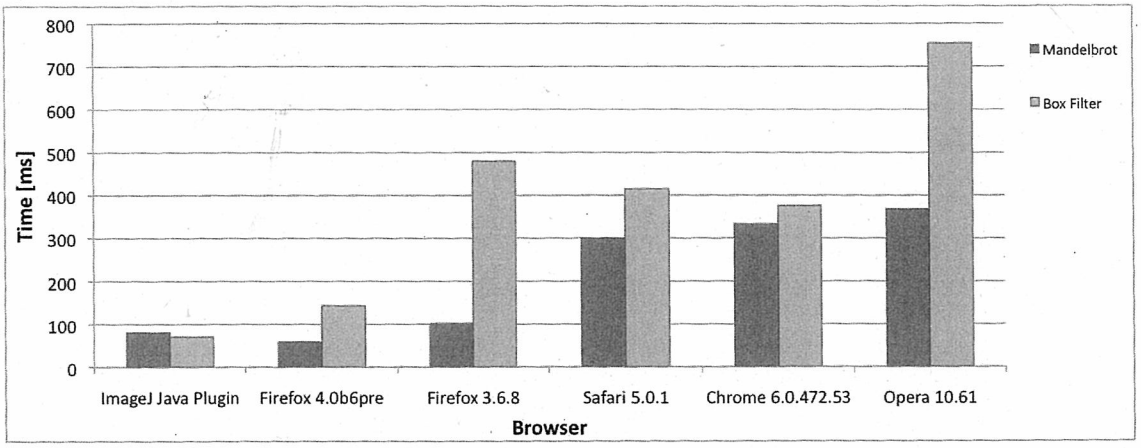

Figure 5. Comparison of browser speeds (Mandelbrot: image size = 500 pixels, Box Filter: radius = 3 pixels)

The results show that the HTML5 canvas-code is more than 25 times faster than the ImageJ macro code. This may be due to the ability to read and write all pixels with one function call. In addition the JavaScript interpreter of the browser is heavily optimized. Comparing the speed of the HTML5 code to the speed of the ImageJ plugins there is no constant speed ratio. For the box filter the ImageJ plugin is 3 times faster using a filter radius of 1. This ratio gets better for larger radii. Using a radius of 6 pixels the plugin is ten times faster. For the Mandelbrot set the results are similar. The plugin is more than 5 times faster for an image size of 600 pixels. These experiments were performed using a Macbook Pro (2.93 GHz) using Safari 5.0.1. Figure 5 compares the execution speeds of different browsers. The source code used for the experiments may be downloaded from [5].

4. CONCLUSION

We think that using the HTML5 canvas object for image processing is an interesting option because it runs pretty fast and image processing algorithms can easily be adapted. Support and performance will definitely improve in the near future. Switching to HTML5 is pretty easy because the JavaScript language is very similar to the ImageJ macro language. For the moment the biggest disadvantage of HTML5 image processing is the fact that there are no image processing frameworks (like ImageJ) that can be used. However it should not be too difficult to port parts of the ImageJ framework to JavaScript. Especially the fact that WebGL will also be supported in the next versions of the browsers makes the canvas element a very interesting option for online image processing. In [6] very promising results are shown for surface and volume renderings in the browser.

5. REFERENCES

[1] http://dev.w3.org/html5/spec/Overview.html

[2] http://www.khronos.org/webgl/

[3] https://developer.mozilla.org/en/Canvas_tutorial/Using_images

[4] Jerome Mutterer, ImageJ macro: "Mandelbrot.txt", 2003

[5] http://www.f4.fhtw-berlin.de/people/barthel/ImageJ/conference2010.html

[6] WebGLot – High Performance Visualization in the Browser
http://dan.lecocq.us/wordpress/tag/webglot/