Overview of JavaScript Application Development

Juho Vepsäläinen
Mathematical Information Technology
University of Jyväskylä
Tietotekniikan teemaseminaari (TIEA255)

April 5, 2011

Abstract

This article will provide an overview of JavaScript application development. CanvasDraw, an actual web based drawing application, will be used to highlight various technologies available. These technologies include HTML5 Canvas and two JavaScript libraries: RightJS and RequireJS. In addition the feasibility of JavaScript based application development is discussed in some detail.

1 Introduction

Software development has gone through radical changes during the last few decades. As minituarization and competition has driven the progress of hardware it has become commoditized, particularly in the Western world.

Before computing resources were scarce. Now they have become abundant. These days there is often more computing power available than we are able to properly utilize.

This development has made the role of a *software developer* crucial, even more so than before. These days it is possible even for small, agile teams to succeed. They just need to work the right way on the right market.

Web development provides one feasible target for these kind of teams. Thanks to the prevalence of web it's now possible to reach millions of potential users with a relative ease. You could say web browser has subsumed the role of operating system and blurred the distinction between various platforms.

Almost ubiquitously supported *JavaScript* programming language and still developing *HTML5* standard combined together form the dynamic duo of web development. Throw in CSS3 and you have got a winner. In some cases amalgamation of this technology provides a significant alternative to established Flash and Java based solutions.

JavaScript – a Joke Language? Before, in the 90s, developers may have sneered at JavaScript and treated it as a language for script kiddies. Perhaps the language deserved some of that.

Surprisingly, the core of the language is quite powerful. Perceived problems often stem from various cross-browser incompabilities and poor browser APIs. Particularly *DOM* is a good example of this.

JavaScript Libraries Various JavaScript libraries have arisen to work around these problems. *jQuery* in particular has become a shining example

of one. It manages to hide nasty cross-browser issues and make it possible for a casual coder to focus on getting things done.

There is a huge collection of libraries like this out there just waiting for you. There is rarely any good reason to stick with vanilla JavaScript.

Goals The main purpose of this article is to provide some insight to JavaScript based app development. I will cover very basics of JavaScript, the programming language.

After that I will have a look at it as compared to a full web development stack ranging from client to server.

Once I'm done with JavaScript, I will move onto HTML5 and discuss the concept a bit further. It is way too broad to cover in full detail in this context. I do aim to provide some starting points for a reader interested in the subject.

Finally I will discuss a JavaScript and HTML5 Canvas based application, CanvasDraw.

I will revisit the concepts discussed in a brief conclusion.

2 JavaScript – Programming Language

> "Java is to JavaScript as ham is to hamster."

Often people not familiar with JavaScript think it's the same thing as Java. The truth could not be further apart.

The name JavaScript has been derived through a poorly thought out decision by marketers at Netscape [6]. It was probably thought to give the language some kind of leverage since Java was the hip thing in the golden 90s.

Oh well, even despite this poor decision the language has managed to thrive, or at least become popular. It is easily one of the most deployed languages out there. Almost every web browser provides some kind of support for JavaScript. This makes it a truly ubiquitous platform to develop for.

2.1 History

Brendan Eich started developing JavaScript in 1995 for Netscape's then popular Navigator web browser. The idea was to provide some kind of scripting interface to make it easier for programmers to develop various functionality to otherwise static webpages. [6]

Microsoft responded by implementing a similar language, JScript. Even though the languages nowadays look pretty much the same, there are some minor differences between them. They are not big enough for me to worry about, though, so I won't bug you about them.

Since then the specification of the language has been formalized. This specification is also known as ECMAScript [1]. Various implementations are based on it. Perhaps the most known ones besides JavaScript is Adobe's ActionScript [2] and its versions.

2.2 Outlook

At the time Eich was particularly impressed by Self, Scheme and similar languages. From Java 1.0 he took Math and Time libraries. As a result he integrated various features from these languages to his mighty little hack of a week [6].

Overall the language is quite simple. It looks a bit like C and Java. Particularly bracketed syntax is probably familiar to friends of these languages. Similarity-wise that's about it, though.

2.3 Typing

JavaScript uses weak, dynamic typing [1]. Since the language accepts almost anything except for the most blatant errors, this makes it quite flexible. It also means that it's quite easy to make inadvertent errors. These will get caught later on as some poor user does some nasty thing the developer didn't anticipate properly.

Especially comparison operator (==) may be tricky for a beginning programmer. It coerces the comparable items to the same type by default and executes the comparison based on that. In some cases this may yield unexpected result. This is the reason why some favor using non-coercing operator (===) instead.

2.4 Basic Types

The language provides basic types including *Object*, *Array*, *String*, *Number*, *null* and *undefined* [1].

Object is the core type of the language. It is simply a hash (key-value pairs).

In most, if not all JavaScript implementations, it is also ordered. This means the keys of an Object appear in given order when iterated. Note that

this is not guaranteed by the ECMAScript specification [1] so it may not be quite safe to rely on this behavior.

Here's a small example of what Object looks like:

```
var duck = {
    name: 'donald',
    age: 42
};

// let's print out some values
console.log(duck.name, duck['age']);

// set some property
duck.height = 123;
```

Note that "duck.name" and "duck['age']" achieve pretty much the same thing. The latter syntax is used particularly when key happens to clash with some JavaScript keyword. It is also handy while iterating values of an Object.

Array in JavaScript is indexed from zero just like in most popular languages perhaps with Lua as an exception. They may also be treated as queues or stacks using the API.

An Array can look like this:

```
var lotteryNums = [12, 25, 5, 2, 6, 3, 21];

// extra number
lotteryNums.push(22);

// not gonna need it. got good numbers already
lotteryNums.pop();
```

String simply signifies an array of characters. It is possible to iterate it just like a regular Array.

A very simple String:

```
var name = 'Joe';
```

Number type is a bit special. Even though the language contains functions such as parseInt, it stores all numbers using some kind of floating point presentation. This is a good enough solution as long as you are not doing

any high precision mathematics. There are libraries such as BigNumber[1] that work around this issue, though.

A couple of Numbers:

```
var a = 2;
var b = 5.2;
```

null and undefined null and undefined are more or less equivalent semantically. Of these particularly undefined is used. It comes around in many places. JavaScript Garden [8] lists its usages as follows:

- Accessing the (unmodified) global variable undefined.
- Implicit returns of functions due to missing return statements.
- return statements which do not explicitly return anything.
- Lookups of non-existent properties.
- Function parameters which do not had any explicit value passed.
- Anything that has been set to the value of undefined.

Often null may be replaced simply by using undefined. Some parts of JavaScript's internals rely on the usage of null, though [8].

The following example shows null and undefined in action:

```
var c;
var d = undefined;
var e = null;

console.log(c); // undefined
console.log(d); // undefined
console.log(e); // null
```

2.5 Basic Structures

JavaScript includes a variety of handy language structures. I will cover basic conditionals, exceptions, loops and functions next.

[1] `http://jsfromhell.com/classes/bignumber`

Conditionals form the core of most modern languages. JavaScript provides very standard conditionals.

The basic one looks like this:

```
var a = true;
var b = 21;
var result;

if(a) {
    result = 'got a';
}
else if(b == 13) {
    result = 'no a, b matched';
}
else {
    result = 'no match;
}

console.log(result);
```

In addition it is possible to use Cish *ternary operator* like this:

```
var a = '13';
var b = a == 14? 'got a': 'noo';
```

There are also the usual *and* and *or*:

```
var a, b = 1, 0;

// evaluates as false, selects b
var c = a && b;

// evaluates as true, selects a
var d = a || b;
```

The language contains also a switch statement:

```
var favoriteMovie = 'Rambo';
var result;

switch(favoriteMovie) {
    case 'Rambo':
        result = 'Awesome!';
        break;
    case 'Water World':
```

```
        result = 'That sucks.';
        break;
    default:
        result = 'Ok...';
}

console.log(result);
```

Exceptions work as well:

```
var triggerError = function() {
    throw new Error('Uh oh, did something nasty!');
};

try {
    // do something nasty to trigger exception
    triggerError();
} catch(e) {
    // catch it
    console.log(e);
}
```

Looping Arrays is quite simple in JavaScript. It contains the usual *for*, *while* and *do-while*.

```
var i;
var lotteryNums = [12, 25, 5,
    2, 6, 3, 21];
var amount = lotteryNums.length;

var printNumber = function() {
    console.log(lotteryNums[i]);
}

for(i in lotteryNums) {
    printNumber();
}

i = 0;
while(i++ < amount) {
    printNumber();
```

```
}

i = 0;
do {
    printNumber();
} while(i++ < amount);
```

Some people prefer to use an *each* or *forEach* method like this:

```
// another way (JS 1.6+)
lotteryNums.forEach(
    // it's possible to omit i
    // and array if needed
    function(number, i, array) {
        console.log(number);
    }
);
```

In case the browser doesn't support this yet, it is possible to add it there simply by attaching a suitable function to the prototype of Array 2.10.

One example of this may be found at `https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/forEach`.

Looping Objects is relatively straightforward. As can be seen below, the syntax resembles Array one quite a bit:

```
var duck = {
    name: 'donald',
    age: 42
};

for(var k in duck) {
    var v = duck[k];

    console.log(k, v);
}
```

Of course longer term writing something like that each time you want to loop through your keys and values gets boring. Some people use a solution like this:

```
Object.forEach(duck,
    function(k, v) {
        console.log(k, v);
```

```
    }
);

// compact alternative that just might work in this case
Object.forEach(duck, console.log);
```

It is important to know that it is not advisable to attach forEach or a similar method to Object's prototype directly [8]. This **will** most likely break some library code depending on a vanilla Object. Instead it is preferable to attach this kind of methods directly to the Object itself.

In case you just need to access just keys or values, you could implement Object.keys and Object.values. It is possible to iterate contents of that just like above in the case of Arrays.

Some people work around the prototype issue simply by wrapping Object in a Hash like this:

```
// some wrap Object in Hash class
var hashDuck = new Hash(duck);

// now we can use this
hashDuck.each(
    function(k, v) {
        console.log(k, v);
    }
);

// as above, this should work took
hashDuck.each(console.log);
```

2.6 Functions

are treated as first class citizens in JavaScript. A basic function definition may look something this:

```
function hello() {
    console.log('hello world!');
}

var helloKitty = function() {
    console.log('Hello kitty!');
};
```

```
// invocations
hello();
helloKitty();
```

It is quite common to pass them around and use them as callbacks. This is particularly true when JavaScript is used in event driven programming. Here's a small example showing that:

```
var amountOfCats = 0;

// just some selector and method
// ~not~ a part of JS core
$('catCounter').on('click',
    function() {
        amountOfCats++;
    }
);
```

Function arguments may be accessed also using a specific Arguments object. Despite looking like an Array, it is not one so don't expect it to work exactly like one. Here's a quick example:

```
var showArgs = function() {
    console.log(
        'first arg', arguments[0],
        'all', arguments
    );
};

showArgs('shoe', 'cat', 'pie');
```

Even though the language doesn't support named arguments officially, it is possible to mimic this behavior simply by passing an Object.

This is particularly useful in case the function has lots of arguments. Keeping track of them gets kind of tricky once you pass the limit of two or three. Optimally your functions should take only one or two arguments at maximum.

2.7 Scoping

Scoping-wise it uses a bit different kind of system than many other languages. Instead of having a block scope, it uses function one.

Obviously this means that even variables defined within blocks within your main scope are visible to it. This is the reason why some developers make their variable definitions at the beginning of a function as a precaution. Here's a small example illustrating this:

```
var random = function() {
    var a = 5; // random as any

    var secret = function() {
        // not visible to parent
        var b = 10;

        // access parent scope
        return a;
    };

    return secret();
};

random();
```

2.8 this

this is one of the interesting features of JavaScript. It simply points to the current parent. Commonly this is problematic especially when dealing with callbacks. The following example illustrates this:

```
var Button = {
    initialize: function() {
        var self = this;

        this._d = false;

        function toggle() {
            self._d = !self._d;
        }

        // external lib!
        $E('div').on('press', toggle);
    }
}
```

This solution exploits the way JavaScript's scoping works. An alternative would be to pass reference to parent as an argument.

2.9 Global variables

By default variables are treated as global. In browser environment they are bound to the window Object.

In case the developer want to use a local variable, she has to declare it using specific *var* keyword. It is highly recommended that global variables are avoided since they are quite brittle and prone to errors.

In case global scope within a script or an app is needed, a specific global Object is often defined. After that it is just a matter of referring to it instead.

The main benefit of this is that the real global scope won't get unnecessarily polluted. The next example illustrates this:

```
// stash for globals of our app
APP = {};

APP.amountOfBananas = 13;

// local
var animal = 'monkey';
```

2.10 Inheritance

One of the main differences between JavaScript and many other Object based languages is the fact that JavaScript provides prototypal inheritance instead of a class based one. It is, however, possible to implement a class based system using this scheme.

In prototypal inheritance scheme the system keeps track of prototype chains and provides means of seeking Object properties using it. In case an attribute is not found within the current Object, the system will traverse to a parent, seeks there and so on. In case nothing is found, undefined is returned.

An example showing how to deal with inheritance in JavaScript may be found at `http://phrogz.net/js/classes/OOPinJS2.html`. Various JavaScript frameworks provide some kind of solution of their own so there is rarely reason to bake your own solution unless you happen to need something special.

The main advantage of JavaScript's approach is that it allows you to extend existing functionality with little effort.

2.11 Programming paradigms

As JavaScript is a very flexible language it is possible to use common programming paradigms in it. It is no problem to mix some functional code with object oriented one for instance.

It is particularly well suited for event driven programming due to its nature as seen in the examples above.

2.12 Problem Spots

Even though JavaScript is a decent language it has some shortcomings. As mentioned earlier weak typing may be troublesome especially if you are not too aware of how it's coercing your types.

The fact that all variables are global by default is another nasty detail. More often than not you want to explicitly use local variables, not global. It would be much nicer to provide a global keyword and treat all variables as local instead.

The language doesn't provide any kind of support for operator overloading. This is particularly troublesome if you need to develop custom types that blend in with the language. As it is you have to handle your operations using explicit methods.

In case of Microsoft's JScript it is not possible to use trailing commas within Array and Object definitions like in regular implementations of JavaScript. This can be somewhat annoying especially if you have gotten used to putting them there in other languages.

It is possible to work around these issues using a precompiler. There are actual languages that have been built upon JavaScript that solve some of these issues in their own way. CoffeeScript[2] provides perhaps the most known example of these.

2.13 Further Resources

Further resources related to JavaScript may be found at `https://github.com/bebraw/jswiki/wiki`. A quick search should yield more starting points.

I do recommend checking out Douglas Crockford's lectures[3] in particular. They may be a bit opinionated but even still contain a lot of eye opening pointers.

[2] `http://jashkenas.github.com/coffee-script/`.

[3] http://developer.yahoo.com/yui/theater/

3 JavaScript in Web Development Stack

Traditionally web development has been split into two separate wholes: *server* and *client* side. Of these server makes sure the pages are shown correctly. On static pages client side does virtually nothing except for rendering the page.

The introduction of JavaScript and similar technologies, such as Flash, made it possible to do certain kind of processing on the client side as well.

Flash in particular is used for implementing full scale applications and games running on top of web browser. JavaScript and various HTML5 APIs enable developers to do the same.

3.1 AJAX

In the early 2000's Microsoft introduced a way to make asynchronous calls to the server using some JavaScript code. Other browsers quickly mimicked this API. As a result now very popular concept *AJAX* was born.

AJAX enables web developers to create interactive pages with ease [10]. As a result features, such as suggesting search, are now commonplace.

3.2 Cloud Computing

As web has become more ubiquitous it has transformed into an application platform of sorts. The core idea this *cloud* way of thinking is that the users may access their data as long as they have access to some kind of client. The user's data is stored on the server. [5]

This is very different compared to the way personal computing is thought of traditionally. Before applications were something that had to be specifically installed on the user's operating system. Cloud based applications mitigate this entirely.

Even though they seem to present a step forward to *ubiquitous computing* [13], they do raise several issues. It is possible the application server suffers from downtime. In this case access to the application may be restricted.

Another issue has to do with data and its privacy. It is true the data will be probably better backed up than on normal desktop use. Who can guarantee that only the right persons have access to it?

In case the data is not confidential this may be a small price to pay for the advantages.

3.3 Client-Server Architecture

As mentioned in the introduction of this section client-server architecture forms the core of all web applications. It is always there in one way or another. From developers point of view this forms an interesting dilemma.

There are a wide range of solutions out there, way too to enumerate properly. Commonly a specific framework is used to ease the burden of the developer.

These frameworks take care of some common problems developers have to deal with on daily basis. They usually contain some kind of easy access to database, way to handle what is shown to the user using templating and some way to route URIs to these templates.

Usually they provide a certain kind of architecture on top of which to build your site or application. Particularly MVC[4] and its variants are popular.

3.4 Common Solutions

It is common to use a language such as Ruby, Python, PHP or Java on the server side and write interactive bits using some JavaScript. This incurs some mental overhead to the developer as he needs to be proficient in many languages at once.

Even if the developer is proficient switching between languages always leads to some kind of context switch. This is turn may lead to inadvertent errors.

There are solutions available that try to mitigate this problem by generating the JavaScript code needed based on a snippet written in the host language. Pyjamas[5] provides on solution such as this for Python.

Of course this means access to some specific JavaScript based functionality may be limited. You might for example use some very specific widget you found. In this case you may have to implement some kind of kludge to get it working the way you want to.

3.5 Pure JavaScript Stack

Recently another way of thinking has arisen. What if instead of trying to convert some code from another language to JavaScript we just used JavaScript on the server side as well? As it happens this is a workable idea.

[4] Model, View, Controller [12]

[5] `http://pyjs.org/`.

node[6], a JavaScript server library built on top of Google's blazingly fast V8 engine, is perhaps the most interesting step towards this direction. It provides an asynchronous way, similar to Erlang, to implement server side functionality.

It works on quite low-level so a variety of higher level libraries and frameworks have been built on top of it. A listing of these may be found at `https://github.com/joyent/node/wiki/modules`.

3.6 Web Based Development

There are various tools available that might revolutionize the way we think about development. Traditionally code is written using an Integrated Development Environment (IDE) or text editor running directly on operating system and then tested using a web browser or some terminal based tool that emulates browser environment. What if development was done directly on the web instead?

Various web based JavaScript editors provide an answer to this question. They range from simple solutions such as jsFiddle[7] to more robust ones such as Cloud9[8] and Akshell[9]. Akshell in particular promises to cover the whole stack so you can develop whole application without ever leaving the browser.

The main advantage of this approach is that it decouples the IDE from your computer making it possible to access it almost anywhere without additional trickery. Solutions such as this might also enable some interesting team based functionality to be developed.

One example of this could be collaborative code editing and pair coding via web. Currently Google Docs applications give a nice idea how collaborative work like his could work out in practice.

4 HTML5

Traditionally HTML has been designed to be a content sharing platform, not application one. This unfortunately shows. Even though various frameworks try to hack around the limitations, there is only so much they can do.

HTML5 specification makes a few steps to amend this. It exposes new APIs for developers that enable them to do more than before. The specification is still more or less in flux. There are various parts that are quite well

[6] `http://nodejs.org/`
[7] `http://jsfiddle.net/`
[8] `http://cloud9ide.com/`
[9] `http://www.akshell.com/`

supported already by modern web browsers. Canvas API [7] in particular provides a good example of this.

4.1 Available Features

In addition to Canvas, HTML5 provides APIs for Audio, Video, 3D, Web Workers, Web Sockets, History and such. There's simply too much in the spec [3] to enumerate here.

There are of course improvements to the HTML markup that make it possible to express semantics in more accurate manner. CSS3 includes some new features that complement HTML5 well.

In some way the specification takes browsers closer to Adobe's Flash. Some functionality that required Flash before may be implemented using native ways. HTML5 has not been designed to replace Flash entirely. It will rather complement it and work as a fallback in some cases.

4.2 Canvas API

To quote the specification [7], Canvas API provides an immediate-mode API and associated utility methods for drawing two-dimensional vector graphics to a raster rendering area. In order to do allow this it provides a specific Canvas element. It may be defined like this using HTML markup:

```
<canvas id="myCanvas" width="640" height="480"></canvas>
```

The canvas provides a 2D context that may be used to manipulate its content. The following JavaScript example demonstrates this:

```
var canvas = document.getElementById("myCanvas").
var ctx = canvas.getContext('2d');
```

2D Context may be thought as a state machine. It allows you to transform the output space before actually drawing on it. For instance in order to draw a clock you might just draw straight lines while rotating the context every once in a while. This makes it possible to avoid some tricky math.

The API is quite low-level by its nature. Even though it provides some useful abstractions in some ways it isn't that useful out of the box. Various wrappers[10] aim to work around its handicaps.

In order to give some kind of idea of the API I've listed its attributes and methods in the following list based on the official specification [7]:

[10] `https://github.com/bebraw/jswiki/wiki/Canvas-wrappers`

- canvas - back-reference to the actual canvas
- save, restore - state related methods
- rotate, scale, setTransform, transform, translate - various transformation methods
- globalAlpha, globalCompositeOperation - compositing attributes
- fillStyle, strokeStyle, createLinearGradient, createRadialGradient, createPattern - colors and styles
- lineCap, lineJoin, lineWidth, miterLimit - line styles
- shadowBlur, shadowColor, shadowOffsetX, shadowOffsetY - shadows
- clearRect, fillRect, strokeRect - rectangles
- arc, arcTo, beginPath, bezierCurveTo, clip, closePath, fill, lineTo, moveTo, quadraticCurveTo, rect, stroke, isPointInPath - paths
- font, textAlign, textBaseline, fillText, measureText, strokeText - text
- drawImage - drawing images
- createImageData, getImageData, putImageData - pixel manipulation

Example The following example shows how you might draw a linear gradient using the API:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");

// set up gradient
var grad = ctx.createLinearGradient(0, 0,
    canvas.width, canvas.height);
grad.addColorStop(0, 'yellow');
grad.addColorStop(0.4, 'green');
grad.addColorStop(0.8, 'black');
grad.addColorStop(1, 'black');

// draw gradient
ctx.fillStyle = grad;
ctx.fillRect(0, 0, canvas.width, canvas.height)
```

Even though the example itself is quite trivial, it shows the main steps. First you have to set up the context. After that you have to set some states, fillStyle in this case, and finally execute some operation (fillRect).

As mentioned earlier it would be possible to transform the output space before rendering. In addition it's possible to set global alpha and compositing mode that will be used when applying new data on the canvas.

It is notable, however, that globalCompositeOperation does **not** apply for pixel-wise operations, such as putImageData. Considering this it is often desirable to use drawImage method instead since it composites properly.

4.3 Starting Points

"Dive into HTML5" [11] by Mark Pilgrim provides one starting point. Various demos may be found at `http://html5demos.com/`. A quick search should yield more interesting resources to study.

5 CanvasDraw

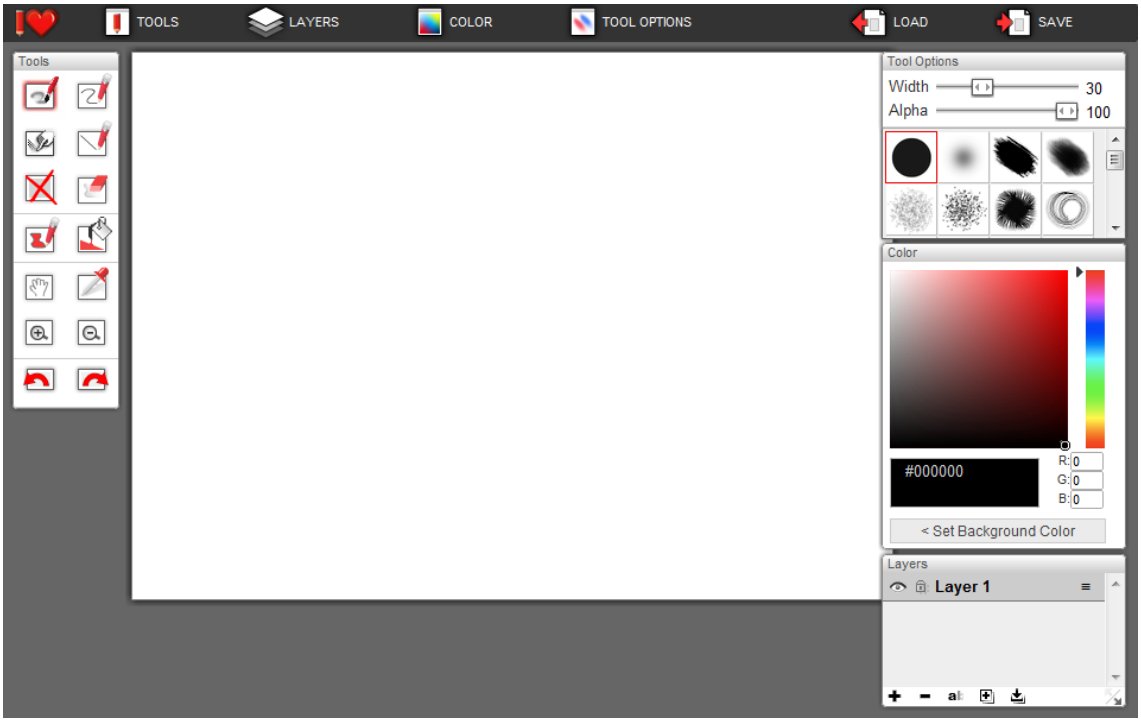

Figure 1: Default View of the Application

CanvasDraw (figure 1) is a drawing application built using JavaScript and HTML5 technologies. It utilizes Canvas API in particular. On JavaScript side it relies on RightJS and RequireJS libraries.

It has been designed for `http://www.ratemydrawings.com` site. Rate My Drawings (RMD) is a drawing community aimed for people of all ages. As can be inferred from the name it makes it possible for the users to share and rate their drawings. The drawings are drawn using a set of applications provided by the site itself.

In addition to drawing applications the site provides basic forum functionality. Competitions and other community events are arranged on regular basis.

5.1 Overview

The current drawing tools available are based on Java and Flash. CanvasDraw is meant to complement, and possibly replace, these tools.

Compared to earlier tools CanvasDraw has some unique advantages and taps into HTML5 development happening. Browsers have improved their capabilities by leaps and bounds making implementation of applications like this possible.

In some ways it cannot quite match existing tools. Java tool in particular is simply "too good" in some respects. Some features, such as layer blending modes, are not feasible due to the limitations of the available APIs.

There are already HTML5 Canvas based tools, such as DeviantArt's Muro[11], Mugtug's Sketchpad[12] or Mr.doob's Harmony[13] out there. All applications like this are bound by the same API. Fortunately there are some ways that may be used to differentiate from the competition.

This is true particularly when it comes to surrounding services, brand and overall architecture of the application. With some right choices a significant difference may be made.

5.2 Libraries

The application relies heavily on RightJS[14] and RequireJS[15]. Both libraries have proven to be quite invaluable in practice even though they are not that widely used yet.

RightJS provides various utilities and a simple implementation of classes. The library contains also a handy variety of user interface plugins.

The library aims to provide some kind of "right" way to develop JavaScript. It can be considered as a some kind of mixture of more well known alternatives such as jQuery, MooTools or Prototype. There are also some Rubyish concepts in the library.

The following example shows how to implement a simple cat counter using RightJS:

```
var amountOfCats = 0;
var catCount = $E('div').
    text('No cats!').
    insertTo(document.body);

var addCat = function() {
```

[11] http://muro.deviantart.com/
[12] http://mugtug.com/sketchpad/
[13] http://mrdoob.com/projects/harmony/
[14] http://rightjs.org/
[15] http://requirejs.org/

```
    amountOfCats++;

    catCount.text('Amount of cats: ' + amountOfCats);
};

var addCat = $E('div').
    text('addCat').
    on('click', addCat).
    insertTo(document.body);
```

RequireJS JavaScript does not have concept of modules by default. This is particularly troublesome in any larger scale development.

Fortunately RequireJS manages to provide a neat solution for this problem. It provides a nice module loader and a very basic build system that utilizes various minifiers[16] available.

The cool thing about RequireJS is that it makes it easy to generate specific debug and release builds of the application. There are also various handy build options designed particularly for development work.

Check out the following example to better understand what RequireJS modules are all about. Note how dependencies are injected to the module and how it is possible to expose the interface of the module.

```
// module for some app conf,
// items may contain funcs too
define({
    debug: true,
    logUndo: false,
});

// more complicated definition
// example of a plugin
define({
    meta: {
        tooltip: 'Kickass',
        attributes: {
            width: 5
        }
    },
    execute: function(opts) {
```

[16] Google Closure Compiler, UglifyJS

```
        // do something fab now
    }
});

// def with deps
define(['utils/math'],
        function(math) {
    // stash local utils here

    // and define what to share
    return {
        execute: function() {
            // use the math Luke!
            return math.sqrt(123);
        }
    };
});
```

Compared to normal JavaScript there is some overhead. It is definitely worth it, though. A more complete example of how to use RequireJS in your application may be found at `http://bit.ly/fL2jzH`.

5.3 Architecture

The application has been designed to be easily configurable. This has been made possible by implementing the application more as a plugin platform.

Every panel and tool of the application is a plugin. Even tools may contain plugin systems within them. For instance various brush variants have been implemented as plugins.

Each plugin contains at least some callbacks and possibly some metadata describing it further. Panel plugins contain user interface definitions. These definitions are further styled by using some CSS.

At the time of writing I'm developing an iPad specific port of the application that utilizes the same functional core as the desktop version. Even though desktop version works on iPad, it does not provide quite optimal experience due to limitations of a touch interface.

5.4 Functionality

The application provides functionality you might expect to see in a drawing application. In addition to hotkeys, layers, zoom and undo it provides basic

drawing tools. These tools include brush, pen, eraser, blender, shape, bucket fill and eyedropper. It is also possible to load and save drawings.

Hotkeys The architecture of the application makes it easy to bind "press" and "drag" hotkeys. The former ones of these invoke the tool instantly. Latter ones are invoked as long as the key is pressed after which it switches back to the previously selected tool. The actual binding looks something like this:

```
hotkeys: {
    press: {
        1: 'brush',
        2: 'pen',
        q: 'blend',
        w: 'line',
        ...
    },
    drag: {
        c: 'drag',
        ...
    }
}
```

It simply maps a hotkey to some tool based on its name. Previously I had actual bindings in the tool themselves. I feel it is easier to manage this way.

Since dealing with shortcuts is quite messy in JavaScript as evidenced by [14]. I ended up using an external library, shortcut.js[17] for handling actual binding. The library allows one to use key combinations and special characters in addition to single keys. I use a state pattern for dealing with "drag" hotkeys. I have documented this solution in more detail at `http://bit.ly/ih9JN5`.

Layers The layer system of CanvasDraw is based on CSS z-index property and absolute positioning. This makes it possible to overlay multiple canvasii on top of each other while letting the browser handle compositing. It is also possible to modify opacity of these canvasii thanks to the widely implemented CSS3 opacity property. For now that has not been exposed via the user interface, though.

In addition to actual image data each layer contains name, visibility and alpha lock data. Alpha lock makes it possible to constrain drawn strokes to

[17] `http://www.openjs.com/scripts/events/keyboard_shortcuts/`

currently visible strokes meaning it can be used as a sort of a mask. This can be highly useful for coloring.

The layers may be sorted freely. There is also copying and flattening functionality.

The only major feature missing are layer blending modes. At the time being it simply is not possible to implement this standard functionality without a significant performance overhead.

Zoom Finding a good way to implement zoom performing well enough on low-end systems provided quite a challenge. The easiest way to implement zoom using Canvas is simply to use CSS. Unfortunately that incurs some overhead.

While being accurate that simply is not acceptable. I ended up going entirely another route. I have split layer data to two data structures. One contains actual, "original" data while the other contains the data shown to the user.

In case the user operates on zoom level 1 (no zoom), the application will simply modify original data directly. In case the user zooms things get interesting. In this case the canvas is rescaled and the contents of original layer are drawn on zoom layer using Canvas native method known as drawImage. This performs actual scaling.

Thanks to this painting on the zoomed layer is effectively as fast as it is in the non-zoomed case. There is some overhead caused by the need to redraw the strokes to the original canvas after each stroke on zoomed layer. In practice this hasn't proven to be a real issue. It is interactive performance that counts.

Undo The zoom system provided by the application spans a wide range of layer related functionality and tools modifying canvas. The system is based on a simple queue keeping track of performed commands. For each undoable command an anti-command has been defined.

In case of "move layer" the actual command (redo) can be stated as "move from n slot to m slot" while anti-command may be stated simply as "move from m slot to n slot". There are cases that are more complicated as they need to keep track of actual image data.

There are some hard undo limits in the system after which it will simply start to discard data. In this case undo branching has been disabled as well meaning that in case you undo a few times and perform an entirely new operation it will simply discard the old items that could have been redone before.

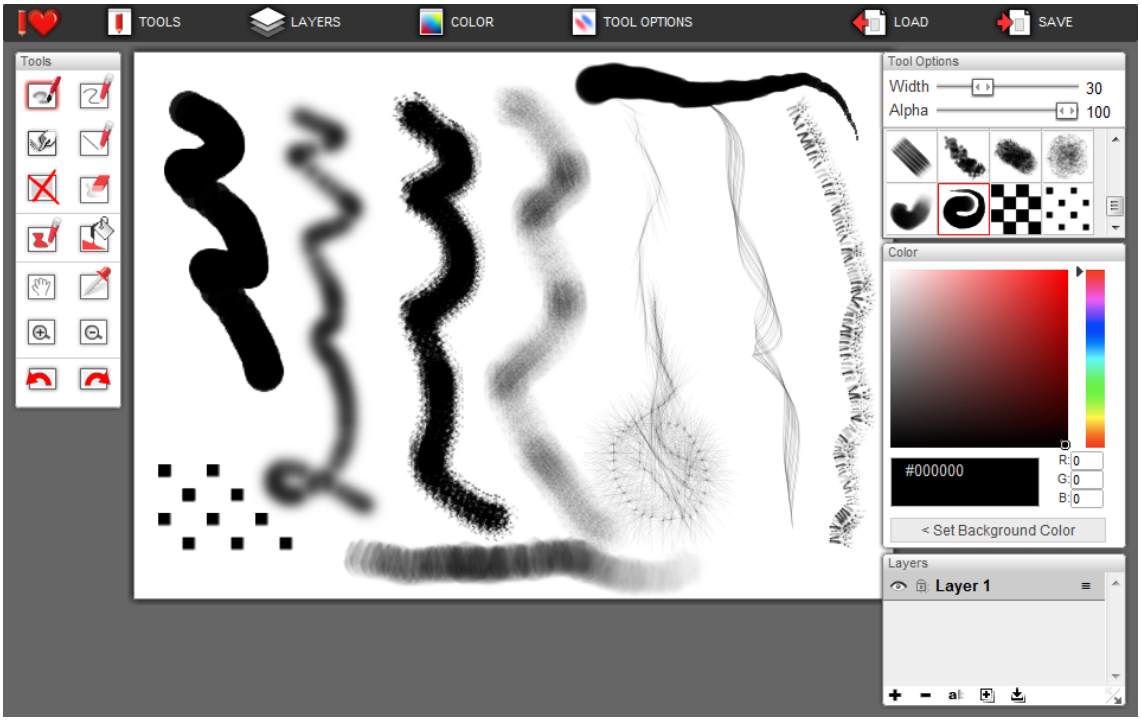

Figure 2: Some Brushes of the Application

Brushes The basic brush system (figure 2) provided by the application is fairly standard. It uses a method also known as "stamping". In this solution a drawn stroke is imagined to consist of individual dabs. Each dab is then rendered on the canvas using drawImage method of the Canvas API [7].

In order to make it easier to develop new brushes I designed a special architecture that makes it possible to vary various properties of a stamp based brush based on callbacks. These properties include stamp texture, rotation, scale, translation, alpha, color and spacing.

The following snippet should give a better idea of what stamp brushes look like:

```
define(['./common'], function(common) {
    return common.createStampBrush({
        name: 'swirly', // name of the brush,
        // maps to default stamp
        meta: {
            tooltip: 'Grass_brush',
            attributes: { // default values for attributes
                spacing: 4
            }
```

```
        },
        callbacks: {
            ...
        }
    });
});
```

There is also an advanced system that allows to create special passed brushes. In this case each resulting dab is treated as stack of these individual stamps. This makes it possible to implement various interesting effects including some kind of shadow effects and such. The following snippet should give a better idea how this works out in practice:

```
define(['./common'], function(common) {
    return common.createStampBrush({
        name: 'swirly', // name of the brush,
        // maps to default stamp
        // names of stamp images to use. overrides "name"
        stamps: ['swirl1', 'swirl2', 'swirl3'],
        meta: {
            tooltip: 'Grass brush',
            attributes: { // default values for attributes
                spacing: 4
            }
        },
        callbacks: {
            stamp: function(opts) {
                // by default the system will use just
                // the first image if you want to cycle
                // the images,
                // you can do something like this
                return opts.stamps[
                    opts.index % opts.stamps.length
                ];

                // opts.stamps is just a list
                // containing stamp names.
                // based on this you could just
                // return something like "swirl3"
            }
        }
    });
```

});

In addition to stamp architecture there are several purely procedural brushes in the application. They are procedural in the sense that they do not utilize any predefined textures. Instead they just draw their result using the available APIs. In case of fur brush it actually takes the current stroke data in count and uses it to perform some extra shading in case the current dab is near enough existing ones.

All brushes of CanvasDraw are compatible with Wacom digitizer pressure. They are particularly useful for achieving painterly effects and mimicking natural media.

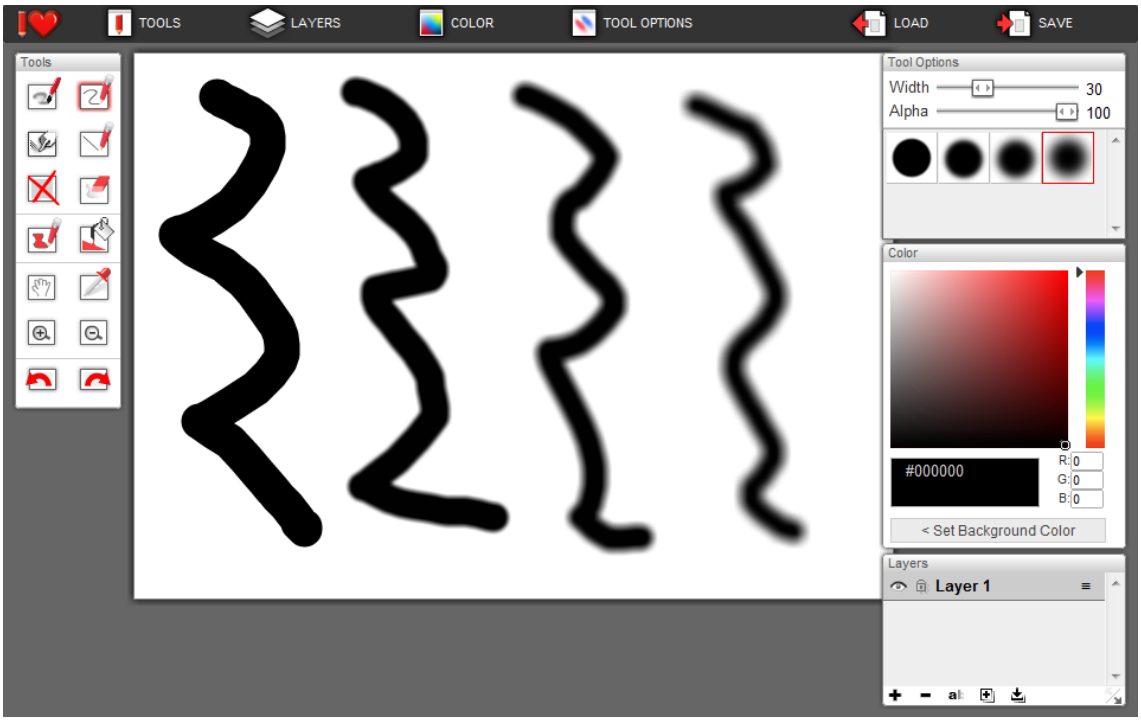

Figure 3: Pens of the Application

Pens Compared to brushes pens (figure 3) provide a different kind of way of drawing. They are drawn procedurally and utilize shadow API of Canvas [7]. Shadow API in particular makes it possible to achieve smooth, continuous results. Pens are useful for lineart and sketching in particular.

Performance-wise pens available currently cannot match brushes. This is due to the fact that they are drawn using a buffer that is cleared and then redrawn at the beginning of each draw cycle. There are various ways to

speed it up. These rely on minimizing the amount of clearing and drawing performed.

Due to the way the pens work they are not unfortunately compatible with Wacom digitizer pressure.

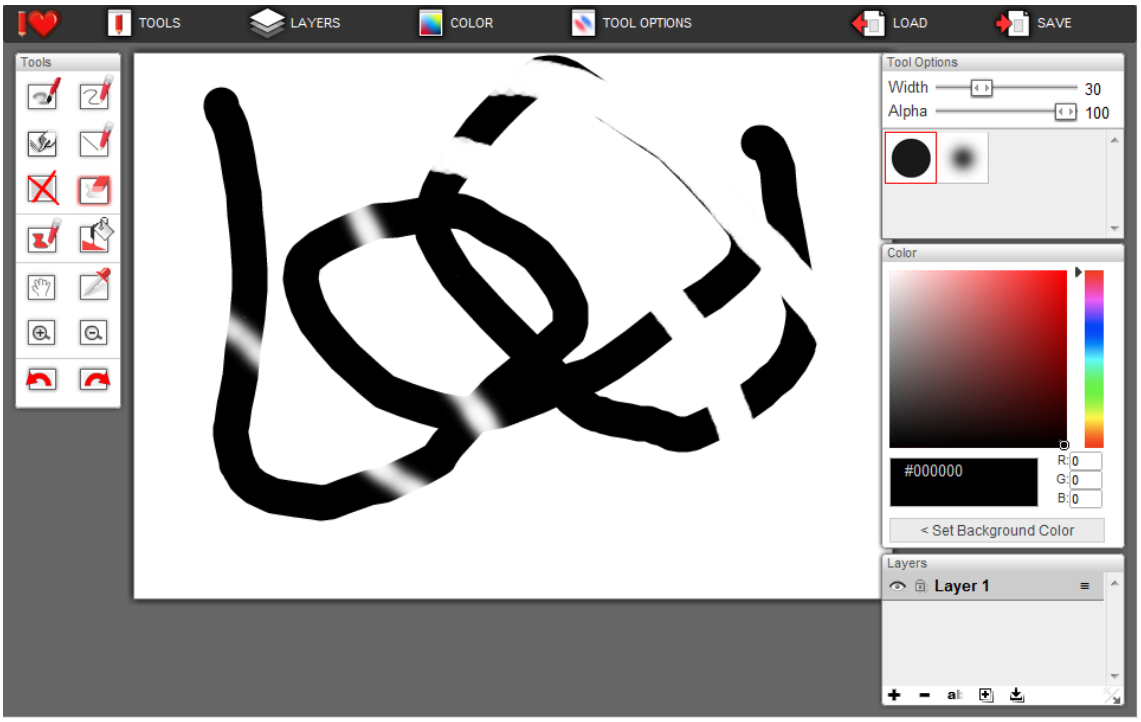

Figure 4: Erasers of the Application

Erasers Erasers (figure 4) may be considered a special case of brushes. Instead of adding color to the canvas they just remove it. This is achieved using "destination-out" globalCompositeOperation [7] instead of the default ("source-in") one.

Blenders Currently CanvasDraw provides three special blenders (figure 5) that may be used to manipulate existing color data. They work like stamping brushes except for the fact that they sample the existing data and then manipulate it somehow.

In the case of blur variants (two leftmost) the data is simply blurred by using Mario Klingemann's implementation of stackblur for Canvas[18].

[18] `http://www.quasimondo.com/StackBlurForCanvas/StackBlurDemo.html`

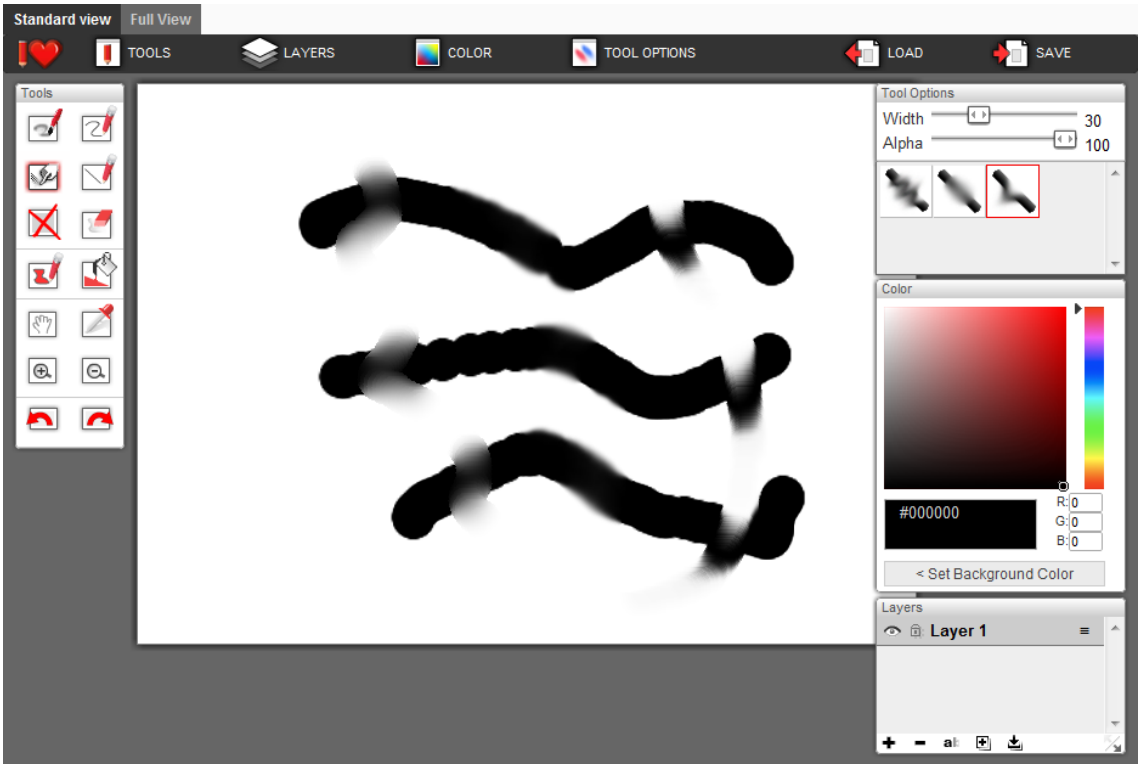

Figure 5: Blenders of the Application

Smudge (the rightmost) makes it possible to literally push image data around. The basic idea is very simple. It is based on sampling and then applying a "lighter" version of the sample on the next dab. This is commonly done using *alpha blending*. After this has been done, sample and mix again at next dab till done.

Shapes The application provides a set of basic shapes (figure 6) including filled and outlined versions of rectangle and ellipse. In addition a special *scanfill* based "fill" tool is provided.

Scanfill makes it possible to draw shapes while adding and subtracting in an interactive manner.

Besides these basic shapes there is a separate "line" tool that could be treated as a shape as well. It has been just separated in the user interface considering it is quite commonly used.

Bucket Fill Given bucket fill is a fairly standard feature the application provides it. Since Canvas API does not actually include an implementation

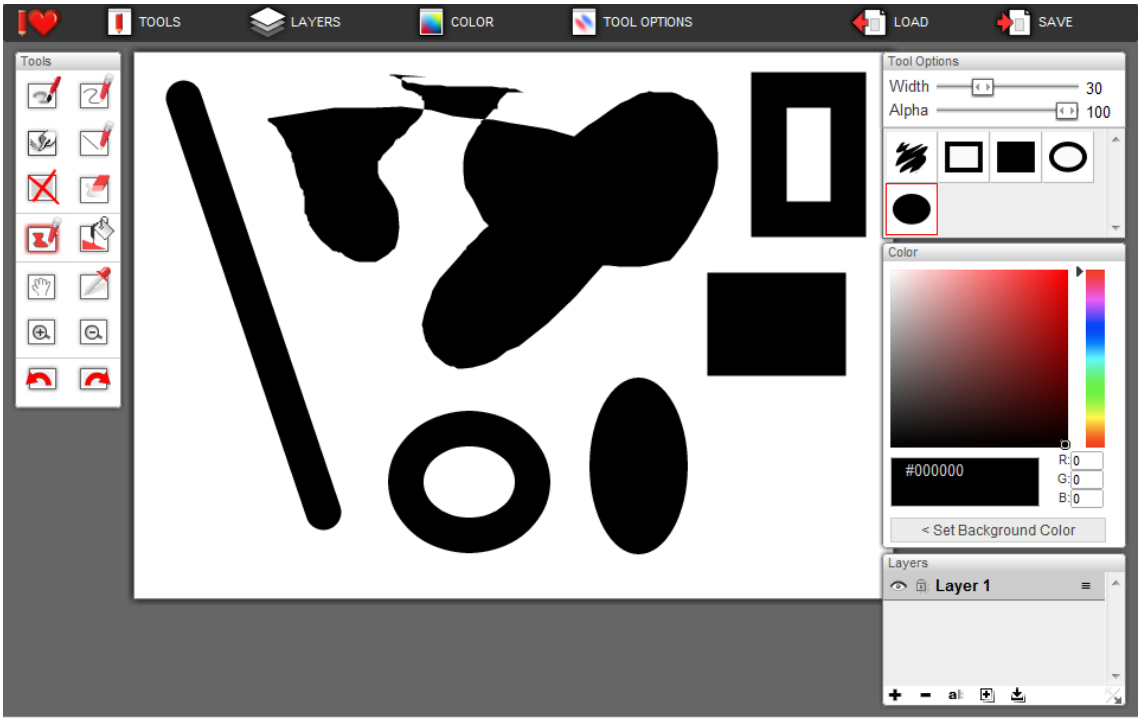

Figure 6: Shapes of the Application

of it one had to be implemented using the slow pixel-wise API. Since it can take quite a few seconds to execute the tool on slow systems a special progress dialog system had to be implemented.

The current implementation if based on William Malone's work [9]. I have no doubt there are faster ways to implement it. This solution is passable for now, though.

Eyedropper The application includes standard eyedropper tool. It simply samples the pixel color under the cursor and sets current color to it.

Save/Load It is possible to save drawings either locally or to finalize the results and send them to the server. In the latter case the application just flattens layer data. Local save dumps the data to a file in JSON[19] format. The contents of this file may then be easily restored.

[19] `http://www.json.org/`

5.5 Experiences

Developing CanvasDraw might not have been as easy as initially envisioned. This is partly due to my relative inexperience as a frontend and JavaScript development.

My earlier efforts definitely helped a lot, though. Some of the issues encountered would have been tough to handle in any case.

To give a recent example the release of Firefox 4 broke blending tools due to the way it deals with clipping areas. The bug has been acknowledged by the development team. Now it's just a matter of waiting for a fix or finding some kind of workaround. At the worst case I have to write a patch to fix the issue myself and then hope that it gets applied.

Now that CanvasDraw is in a relatively stable state I have a nice architecture to build new projects upon. This, in addition to the knowledge gained, will be a huge boon for further development.

Canvas Issues In some ways HTML5 Canvas managed to reach my expectations. In some cases I did have to work around its limitations. Zoom and layer system in particular were quite time consuming to implement.

The API is pretty low-level by its nature. Unfortunately it is missing some functionality that would be nice to have. In some cases browsers implement its functionality in a different manner.

Sometimes it is possible to workaround browser specific issues but not always. This can be particularly frustrating. In some cases working around some issue is simply not feasible due to the performance. If you are willing to restrict your application to work in a small set of browsers, or even one, you will have less to worry about.

One major issue I came upon has to do with the way the API deals with color. Color channels have been limited to 8 bits per channel. This leads to very noticeable rounding errors while compositing. It would be possible to keep track of accurate color data internally. This comes with some development overhead, though. Fortunately this is something not noticeable in common use cases.

5.6 Further Development

The development of the application will likely continue. Various functionality may be added. It is also possible that the application will fork into a few separate applications based on a common core.

One interesting possibility would be to implement the drawing core of the application using WebGL [4] and use that where available. This should

make it possible to reach better performance since it has been accelerated by GPU better than Canvas at the moment.

There have been talks of open sourcing the core. I retain the rights to the source except for certain very specific parts so it is quite possible we will see an open source version of the application at some point.

Open sourcing is not simply a matter of putting the source available somewhere. More thought than that should be put to it.

It is quite possible this could lead to some new, interesting innovation. Perhaps it is possible a healthy developer community could be built around CanvasDraw.

6 Conclusion

It seems JavaScript development has hit mainstream, at least in smaller scale. It is commonly used to enhance web sites.

The fact that the technology was not initially designed for web applications unfortunately shows. The whole infrastructure has been built upon hacks. Fortunately these are often hidden from a developer. Especially various libraries have helped a lot in this regard.

Google's applications provide perhaps the most prominent example of wide-scale usage of JavaScript. They definitely show it is possible to come up with impressive applications that in some ways put their desktop brethren in shame.

In addition to pure frontend development JavaScript has started to make inroads to the server side as well. This development has been made possible especially by node and similar projects during the past few years.

It looks like the future of JavaScript will be bright, at least for the time being. It is easily the most used and deployed language out there that is often perhaps overlooked as an alternative. If you are willing to overcome the initial learning curve, you will be rewarded. The web is yours.

It is easy to compare JavaScript to another seminal language, C. In my mind it fits the exact same purpose but for web instead of systems. I bet Brendan Eich didn't anticipate that when he spent a week designing the first version of it.

References

[1] "ECMA-262 ECMAScript Language Specification", 2009.
URL `http://www.ecma-international.org/publications/files/`

ECMA-ST/ECMA-262.pdf

[2] "ActionScript 3.0 Language Specification", 2011.
URL http://livedocs.adobe.com/specs/actionscript/3/as3_specification.html

[3] "HTML5 Specification", 2011.
URL http://dev.w3.org/html5/spec/Overview.html

[4] "WebGL Specification", 2011.
URL https://www.khronos.org/registry/webgl/specs/1.0/

[5] M. Armbrust et al.: "A view of cloud computing", *Communications of the ACM*, 53(4), ss. 50–58, 2010, ISSN 0001-0782.

[6] Brendan Eich: "JavaScript at ten years", teoksessa "Proceedings of the tenth ACM SIGPLAN international conference on Functional programming", (s. 129), ACM, 2005, ISBN 1595930647.

[7] W3C HTML Working Group: "Canvas 2D API Specification 1.0", 2009.
URL http://dev.w3.org/html5/canvas-api/canvas-2d-api.html

[8] Zhang Yi Jiang Ivo Wetzel: "JavaScript Garden", 2011.
URL http://bonsaiden.github.com/JavaScript-Garden/

[9] William Malone: "Create a Paint Bucket Tool in HTML5 and JavaScript", 2011.
URL http://www.williammalone.com/articles/html5-canvas-javascript-paint-bucket-tool/

[10] L.D. Paulson: "Building rich web applications with Ajax", *Computer*, 38(10), ss. 14–17, 2005, ISSN 0018-9162.

[11] Mark Pilgrim: "Dive Into HTML5", 2011.
URL http://diveintohtml5.org/

[12] T. Reenskaug: "The model-view-controller (mvc) its past and present", *JavaZONE, Oslo*, (ss. 2007–03), 2003.

[13] Mark Weiser: "Ubiquitous Computing", *Computer*, 26, ss. 71–72, 1995, ISSN 0018-9162.

[14] Jan Wolter: "JavaScript Madness: Keyboard Events", 2011.
URL http://unixpapa.com/js/key.html