Interactive SVG with JSXGraph

Bianca Valentin and Michael Gerhäuser

August 30, 2009

Contents

List of Figures

Abstract

JSXGraph is a open-source client-side web library for displaying interactive mathematics and drawings in a web browser. This article demonstrates what can be done with JSXGraph and explains how these things are implemented. The latter focuses on SVG and VML.

1 Introduction

JSXGraph[1] is a client-side web application/library for advanced vector graphics in a web browser. Its range of applications starts from the display of simple bar charts and pie charts up to the interactive display of function graphs and geometric constructions used in schools and universities. Even more advanced features of JSXGraph are the on-the-fly solving of Differential Equation Systems, the display of Lindenmayer systems[2], which are of importance in Biology, and dynamic computation of regression polynomials.

JSXGraph is a library which may be used in various situations. The main application is to serve as a client-side viewer for educational mathematics in schools and universities. Usually these constructions, especially from euclidean geometry, are available in different proprietary XML based file formats, and are intended to be displayed by their associated Java applet. In order to compensate the slow demise of Java applets in the web browsers, JSXGraph may prove to be valuable. At the moment JSXGraph supports the file format of GEONExT[GXT], Intergeo[I2GEO], and - at least partially - GeoGebra[GGB]. Since JSXGraph comes with its own implementation of unzip and gunzip, it should be easy to adapt JSXGraph to a whole variety of even more file formats.

Beside using it as a viewer for existing material, the JSXGraph library may be used as an API for advanced special purpose visualizations. JSXGraph enables interactive

[1] http://jsxgraph.org
[2] Inspired by the work of Sam Ruby: http://intertwingly.net/blog/2006/07/06/Penrose-Tiling

experiments with sophisticated mathematical problems in the web browser. One example is the simulation of the epidemiology model for the swine flu[3]. The possibilities of JSXGraph in this respect exceed by far the options offered by software which is common in schools today.

Another possible application is to use it as a front end for server based applications. One example is the widely-spread free statistics software R[R]. After the mouse up event is fired in this application, the coordinates of the points are sent via AJAX to the R interface on server side. After doing some computations in R, the web server sends the results back to the browser. There the display is updated by JSXGraph. A more simple application of JSXGraph can be the creation of charts on server side. For example PHP could be used to generate JSXGraph charts which are sent to the web browser. Hence the high quality of the vector graphics display can be fully exploited. The additional overhead of downloading JSXGraph is less than 100 kByte, which is smaller than most bitmap images. Furthermore, the JavaScript source is static and can be cached by the browser, which is not possible with non static chart pictures.

From a technical point of view, JSXGraph is implemented in JavaScript. It generates SVG output on Mozilla-, Webkit-, and Presto-based browsers, and VML on the Internet Explorer. Therefore, JSXGraph runs on every modern browser.

JSXGraph is open source software, released under the Lesser GNU General Public License (LGPL). The source code is hosted by sourceforge.

2 Mathematics

The originally intended purpose of JSXGraph was having a JavaScript library for displaying geometric constructions in a web browser. So it is possible to visualize geometric elements that are taught in schools and universities like points, lines, circles, arcs, and angles, as well as predefined composition elements like orthogonal lines, midpoints, and triangles. JSXGraph not only provides tools for displaying geometric constructions but also for doing calculus like plotting curves starting from function graphs with tangents up to parametric and polar curves, cubic splines or other interpolation methods.

2.1 JSXGraph as a web viewer for dynamic geometry software

Dynamic geometry software (DGS) are computer programs which allow one to create and then manipulate geometric constructions, primarily in plane geometry. In most DGS, one starts a construction by putting a few points and using them to define new objects such as lines, circles or other points e.g. intersections of lines and circles. After some construction is done, one can move the points one started with and see how the construction changes.

There are lots of different available software packages to play around dynamically with geometry. Most of them are written in Java, such as GeoGebra, Cinderella[CDY], Compass and Ruler[CaR] or GEONExT, others like The Geometer's Sketchpad[TGS]

[3] http://jsxgraph.uni-bayreuth.de/wiki/index.php/SIR_model:_swine_flu

or Cabri[CABRI] offer a Java based web viewer. If you want to integrate a construction that was created with one of these DGS in a web page, it takes some time to load the corresponding Java applet before you can use it. There are moreover big doubts about the future of Java in web browsers because Java applets are vanishing more and more from the internet and are actually not supported in browsers on modern mobile internet devices. As a result, the idea of JSXGraph was to read GEONExT Files and display them in the browser using JavaScript to manage the DOM elements of SVG or VML, depending on what browser is used. JavaScript is furthermore used to realize the dynamic part in DGS by capturing the browser's mouse events.

2.2 Currently supported file formats

Since JSXGraph has evolved from the GEONExT project, the support of its file format is almost complete. Other file formats are under development. For example JSXGraph is involved in the European Intergeo project which unites several dynamic geometry systems (GeoGebra, Cabri, GEONExT, Cinderella) to agree on a standardized file format that can be opened and edited in any of the participating DGS. Such files can be displayed within a browser using JSXGraph's Intergeo reader. Support for reading GeoGebra files directly is under development in cooperation with the GeoGebra developer team.

Example 2.1 Loading a GEONExT construction from a file named geonextConstruction.gxt

```
<div id="box" class="jxgbox" style="width:500px; height:500px ↩
    ;"></div>
<script type="text/javascript">
   var b2 = JXG.JSXGraph.loadBoardFromFile('box', ' ↩
       geonextConstruction.gxt', 'Geonext');
</script>
```

2.3 Constructing from scratch

The JSXGraph API used in the readers for the different file formats can also be used to create a construction from scratch or to manipulate a construction loaded from a given file. JSXGraph contains functions for compass and ruler constructions in euclidean geometry basing on points, lines, circles, arcs, and angles. Those can be intersected or combined in predefined ways. For example one can construct a line that is parallel to another one or a point that is the midpoint between two other points by using integrated JSXGraph methods. Additionally, one can easily implement own composition elements with JavaScript and the above-mentioned functions.

Example 2.2 Constructing the euler line of a triangle from scratch

```
/* triangle */
A = brd.createElement('point',[1,0]);
B = brd.createElement('point',[-1,0]);
C = brd.createElement('point',[0.2,1.5]);
triangle = brd.createElement('polygon',[A,B,C]);

/* heights */
hc = brd.createElement('perpendicular',[pol.borders[0],C],{ ↩
    name:['','H_c']});
ha = brd.createElement('perpendicular',[pol.borders[1],A],{ ↩
    name:['','H_a']});
hb = brd.createElement('perpendicular',[pol.borders[2],B],{ ↩
    name:['','H_b']});
h = brd.createElement('intersection',[hc[0],hb[0],0],{name:'H ↩
    '});

/* median lines */
mc = brd.createElement('midpoint',[A,B],{name:'M_c'});
ma = brd.createElement('midpoint',[B,C],{name:'M_a'});
mb = brd.createElement('midpoint',[C,A],{name:'M_b'});
sa = brd.createElement('segment',[ma,A]);
sb = brd.createElement('segment',[mb,B]);
sc = brd.createElement('segment',[mc,C]);
/* centroid */
s = brd.createElement('intersection',[sa,sc,0],{name:'S'});

/* circumcircle */
[cPoint,cCirc] = brd.createElement('circumcircle',[A,B,C],{ ↩
    name:['U','']});
cCirc.setProperty({strokeColor:'#000000',dash:3,strokeWidth ↩
    :1});

eulerLine = brd.createElement('line',[s,cPoint],{strokeWidth ↩
    :2,strokeColor:'#901B77'});
```

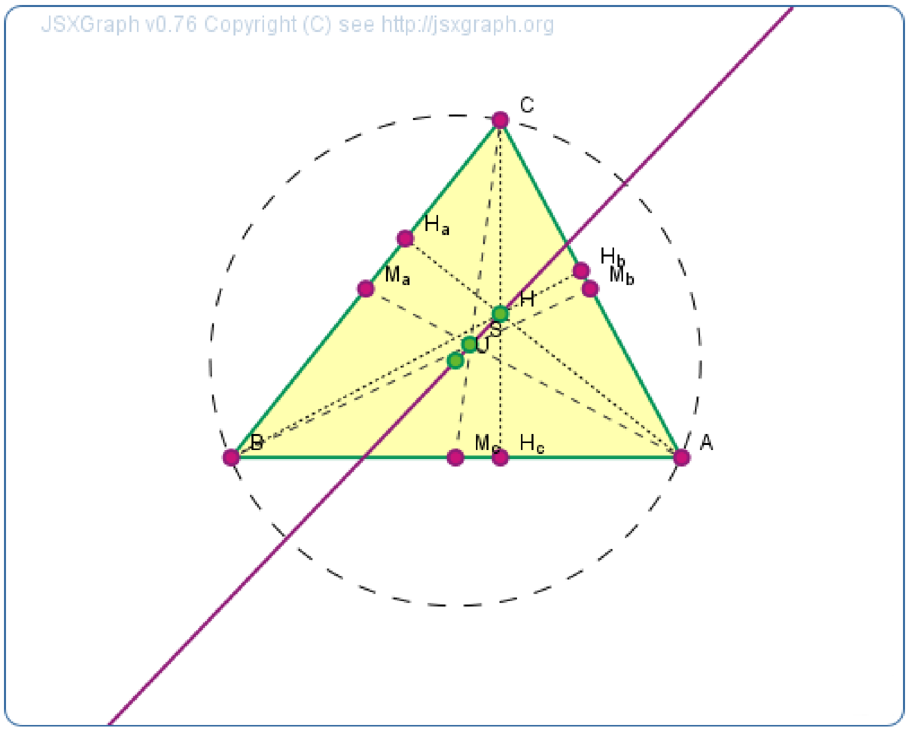

Figure 1: The Euler Line [4]

2.4 Calculus

Beyond the capabilities of JSXGraph with regard to geometric constructions, one is also able to visualize calculus. The possibilities start from plotting simple function graphs to displaying more complex graphs like parametric and polar curves which can be analyzed using the implemented methods for tangents, integrals or spline interpolation.

Example 2.3 Sourcecode to the construction shown in Figure **??**

```
var brd = JXG.JSXGraph.initBoard('jxgbox', {originX: 400,  ↩
    originY: 200, grid:true, unitX: 50, unitY: 50});
brd.createElement('axis', [[0,0], [1,0]], {strokeColor:'black ↩
    '});
brd.createElement('axis', [[0,0], [0,1]], {strokeColor:'black ↩
    '});
var b = brd.createElement('slider ↩
    ',[[1,3.5],[5,3.5],[0,1,3]],{name:'a', strokeColor:'black ↩
    ', fillColor:'white'});
var f = function(x){ return b.Value()*Math.sin(x); }
var plot = brd.createElement('functiongraph',[f,-7,7], { ↩
    strokeColor:'#32CD32', strokeWidth:'4px'});
var g = brd.D(f);
var plot2 = brd.createElement('functiongraph',[g,-7,7], { ↩
    strokeColor:'#9370DB', strokeWidth:'2px'});
var os = brd.createElement('riemannsum',[f, 35, 'middle', -7, ↩
     7], {fillColor:'#B22222', fillOpacity:0.3, strokeColor ↩
    :'#8B1A1A', strokeWidth:'2px'});
```

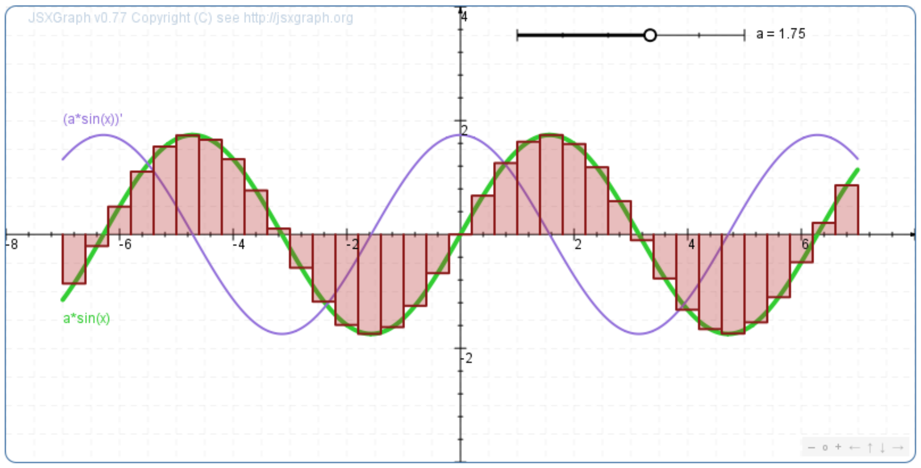

Figure 2: Sine with it's derivative and riemann sum. [5]

With an implementation of the explicit Runge-Kutta method, JSXGraph can also be used to solve ordinary differential equations numerically on the fly with the possibility to change the parameters on runtime and watch the results.

3 Turtle graphics

Turtle graphics as a method of drawing offer a wide variety of applications. So JSX-Graph provides an easy-to-use interface to them.

3.1 Turtle graphics in mathematical modelling

Mathematical modelling with differential equations can also be done using turtle graphics. Exponential population growth models can be simulated easily by using the discrete analogue of the corresponding differential equation and computing the movement of the turtle in equidistant small time intervals.

Example 3.1 Simulating exponential population growth using turtle graphics [6]

```
var board = JXG.JSXGraph.initBoard('box', {originX: 10,  ↩
    originY: 250, unitX: 40, unitY: 20, axis:true});
var t = board.createElement('turtle',[4,3,70]);
...
var dy = alpha*t.Y()*dx;   // Exponential growth
t.moveTo([dx+t.X(),dy+t.Y()]);
```

In a similar way, logistic or autocatalytic population growth processes can be modelled. A further application which can be simulated with turtle graphics is epidemiology where a set of differential equations is given to predict the immediate consequences of a epidemic like the Hong Kong flu or the swine flu. With only a few lines of code the chronological sequence of the rate of susceptible, infected and recovered population can be studied.

3.2 Drawing with turtle graphics

An advantage of using turtle graphics is the easy way to draw. The basic commands are "Go forward a given number of units" and "Rotate left resp. right a given angle". The following programm listing draws "SVG" shown in the picture below.

Example 3.2 Drawing "SVG" using a turtle.

The rt(), lt(), fd(), and bk() methods are abbreviations for right(), left(), forward(), and back(). A full list of commands can be found in the JSXGraph wiki[7]

```
var t = board.createElement('turtle');
// Draw the S,
t.rt(90); t.fd(100); t.lt(90); t.fd(100); t.lt(90); t.fd(100) ↩
    ; t.rt(90); t.fd(100); t.rt(90); t.fd(100);
// the V, and
t.rt(70); t.fd(210); t.lt(140); t.fd(210);
// the G
t.rt(70); t.fd(100); t.bk(100); t.rt(90); t.fd(200); t.lt(90) ↩
    ; t.fd(100); t.lt(90); t.fd(100); t.lt(90); t.fd(40);
```

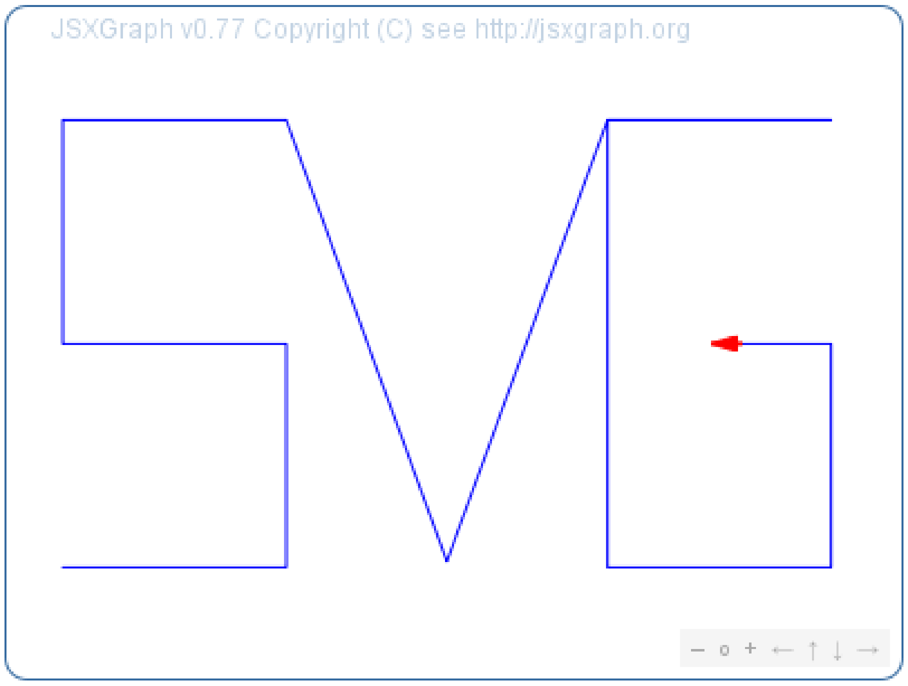

Figure 3: SVG written by a turtle. Result of Example **??** [8]

This approach can be used to easily produce beautiful Lindenmayer systems like plants or penrose tilings which are known from chaos theory . Similarly, well-known fractals like the sierpinski triangle or the Koch snowflake can be drawn. Other applications can be found in statistics where one can visualize random walks with turtles.

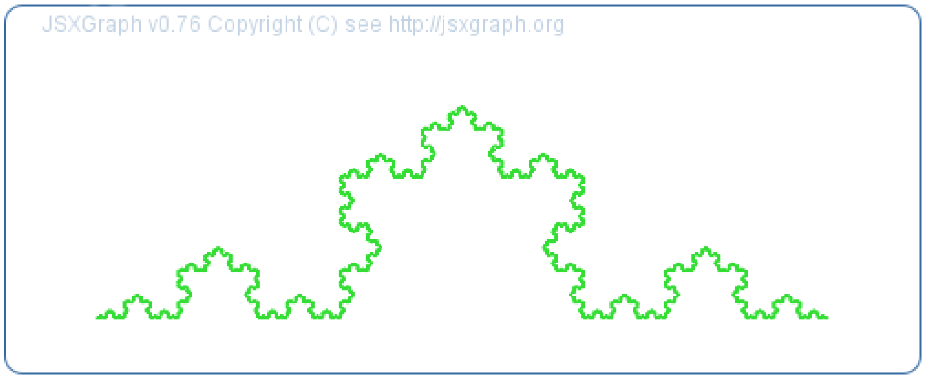

Figure 4: The Koch curve with recursion level 7 [9]

4 Charts

After implementing all the geometric objects and calculus in JSXGraph it was not difficult to extend the library to support charts. For example line charts and cubic spline interpolated charts are made of function graphs, sectors are used to display pie charts, and horizontal resp. vertical bar charts are based on polygons. Of course all of the visual options from the geometric elements like fill and stroke colors, fill gradients, line styles, shadows or special highlighting effects are passed on to the charts. But also the dynamics is still available, so even interactive charts can be done with JSXGraph which is useful for forecasts with varying parameters to model different scenarios.

Example 4.1 Drawing an interactive bar chart

```
board = JXG.JSXGraph.initBoard('box', {originX: 50, originY:  ↩
    450, unitX: 50, unitY: 50, axis:false});
board.suspendUpdate();
board.createElement('axis', [[0,0], [1,0]], {strokeColor:' ↩
    black'});
board.createElement('axis', [[0,0], [0,1]], {strokeColor:' ↩
    black'});

var s = board.createElement('slider',  ↩
    [[8,7],[11,7],[1,1,1.5]], {name:'S',strokeColor:'black', ↩
    fillColor:'white'});
var f = [function(){return this.board.round(s.Value()*4.5,2) ↩
    ;},
          function(){return this.board.round(s.Value()*(-1),2) ↩
              ;},
         /* add other bars similarly */
          function(){return this.board.round(s.Value()*(-1.25) ↩
              ,2);}
          ];
var chart = board.createElement('chart', [f], {chartStyle:' ↩
    bar',width:0.8,labels:f});
/* ...
   set additional style properties
   ...*/
```

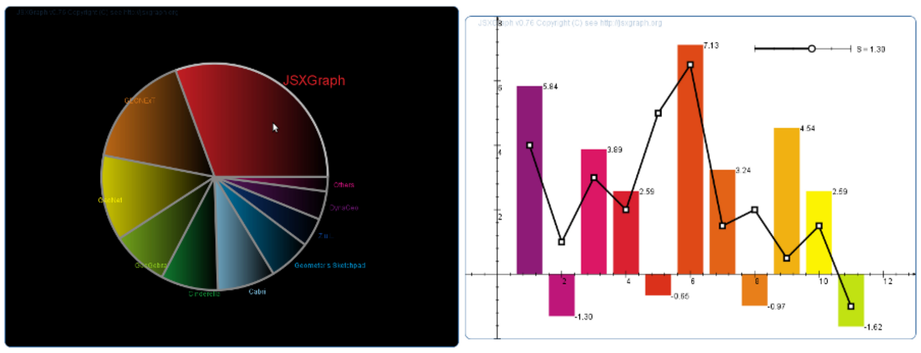

Figure 5: A pie chart on the left hand side[12] and an interactive bar chart together with a static line chart on the right hand side[13].

5 JSXGraph as a front end to server side applications

JavaScript enables us to use the XmlHTTPRequest object to interact with server side applications. Currently one can use our AJAX wrapper JXG.FileReader or the one provided by Prototype[14] or jQuery[15]. So JSXGraph can be used to visualize the output of server side calculations and provides new input data for server applications.

5.1 Just AJAX

For example it is possible to fetch share prices from a web site like Yahoo finance. To do this a script on your own webserver is required because of security restrictions concerning the XmlHTTPRequest object which cannot be used in conjunction with a url that is not on the same server as the web site starting the request. After starting the script from JavaScript it sends back the data from Yahoo which then is parsed by the JavaScript and visualized using a curve.

[14] http://prototype.org
[15] http://jquery.org

Example 5.1 JavaScript invoking a PHP script using AJAX [16]

```
var hi, lo, brd, periodical,
    brd, g, txt, val,
    x = [],
    y = [];

// Update method, called periodically every second
fetchData = function() {
    new Ajax.Request('/ajax/stockquotes.php', {
        onComplete: function(transport) {
            var t, a;
            if (200 == transport.status) {
                t = transport.responseText;
                a = t.split(',');
                x.push(x.length+1);
                y.push(a[1]);
                val = a[1];  // set the text
                if (!g) {
                    g = brd.createElement('curve', [x,y],{ ↩
                        strokeWidth:3, strokeColor:'green', ↩
                        shadow:true});
                    txt = brd.createElement('text', [3,(hi+lo ↩
                        )*0.5,function(){return 'GDAXI = '+ ↩
                        val;}],{fontSize:'14px'});
                } else {
                    g.dataX = x;
                    g.dataY = y;
                }
                brd.update();
            };
    }});
};

// Fetch maximum and minimum values, initialize the board and ↩
     start the periodical update.
new Ajax.Request('/ajax/stockquotes.php', {
        onComplete: function(transport) {
            var a, t;
            if (200 == transport.status) {
                t = transport.responseText;
                a = t.split(',');
                hi = a[6]*1.001;
                lo = a[7]*0.999;
                brd = JXG.JSXGraph.initBoard('jxgbox', {axis: ↩
                    true, boundingbox:[0,hi,200,lo]});
                brd.createElement('axis',[[0,lo],[1,lo]]);
            };
    }});

periodical = setInterval(fetchData,1000);  // Start the  ↩
    periodical update
```

```
<?php
  $fp = fopen ("http://finance.yahoo.com/d/quotes.csv?s=^ ↩
      gdaxi&f=sl1d1t1c1ohgv&e=.csv","r");
  echo fgets ($fp, 1024);
  fclose ($fp);
?>
```

5.2 JXG.Server

For quick'n'dirty solutions the AJAX interface is nice. But if you want to implement an interface to a server side program and want to use it with other worksheets or share it with other people, you'd better take a look at JXG.Server. This is a frontend to a python based plugin system. These plugins are called modules and can be loaded easily with just one function call. After initializing the functionality is available through an interface in JSXGraph and all the AJAX stuff like parameter handling and return data parsing is done by the module.

Example 5.2 Python module on the server which multiplies a given number by 3

```
from JXGServerModule import JXGServerModule
import JXG

class JXGModuleMultiply(JXGServerModule):

    def init(self, resp):
        resp.addHandler(self.mult, 'function(data) { alert( ↩
            data.y); }')
        return

    def mult(self, resp, x):
        resp.addData('y', 3*x)
        return
```

This module is in a file called jxgmodulemultiply.py and can be loaded with JXG.Server through

```
JXG.Server.loadModule('jxgmodulemultiply');
```

After the module has been loaded the python function can be accessed with

```
JXG.Server.modules.jxgmodulemultiply.mult(5);
```

which will result in an alert notification window displaying "15".

6 Plugins for integration in other web applications

To ease the integration of JSXGraph into other web applications we provide some plugins for a couple of them. Those are the blogging software Wordpress[17], the basis software for Wikipedia called MediaWiki[18], and the course management system Moodle[19]. The plugin for the latter one ist still experimental. The user only has to provide a few options for the drawing panel and the JavaScript construction code or a url to a file in a format supported by JSXGraph. The rest is done by the plugin, like loading

[17] http://wordpress.com
[18] http://mediawiki.org
[19] http://moodle.org

the necessary files, initializing the division element containing the drawing panel and finally loading the file or the javascript construction code.

Example 6.1 Include a JSXGraph construction on a MedaWiki page using the MediaWiki plugin

```
[... wiki page content ...]
<jsxgraph height="500" width="600" board="board"  box="box1">
brd = JXG.JSXGraph.initBoard('box1', {originX: 10, originY:  ↩
    250, unitX: 40, unitY: 20, axis:true});
var t = brd.createElement('point',[4,3]);
</jsxgraph>
[... wiki page content ...]
```

7 Implementation details

The main focus of JSXGraph lies on interactivity. The JSXGraph user should experience an immediate feedback to the dragging of points with the mouse. So, special attention has to be paid to fast reactions to mouse move and mouse up events. On the other hand we want to exploit the superb output quality of vector graphics as offered by SVG. These two controversial goals - speed versus quality - have to be balanced.

7.1 Cross-browser development

JSXGraph works on every major browser. The cross-browser issues are handled by an external library, where the web developer who uses JSXGraph can choose between either the Prototype JavaScript library or the jQuery library. JSXGraph can be used with either of these two libraries equally good. Only very few features of these libraries are used and the access is capsuled in some lines in the source code file jsxgraph.js. Mainly the cross-browser handling of events is delegated to the external libraries. Everything else in JSXGraph is built up from scratch and does not rely on any third-party library. So, it should be easy to port JSXGraph to a further framework, if necessary.

7.2 SVG vs. VML

The Microsoft Internet Explorer has still the largest market share of all browsers. Since we do not want to exclude the users of the Internet Explorer we have to adapt JSXGraph to this browser as well. The Internet Explorer can not display SVG, but since 1998 Microsoft[VML1998] supports VML - vector markup language - as vector graphic language in the Internet Explorer. Early in the development process of JSXGraph we decided to implement SVG and VML drawing separately, without using one of the existing abstraction layers. The reason for that decision was performance, the existing layers were too slow at that time.

SVG and VML are similar in many aspects, but different in many details. In order to realize visual properties VML heavily makes use of subnodes of XML nodes.

SVG however mostly uses attributes for the same visual properties. Therefore, the JSXGraph source contains the two files SVGRenderer.js and VMLRenderer.js which control the different graphics languages. The user working with JSXGraph should not get aware of this separation at all because all methods for accessing the drawing layer are encapsulated by JSXGraph.

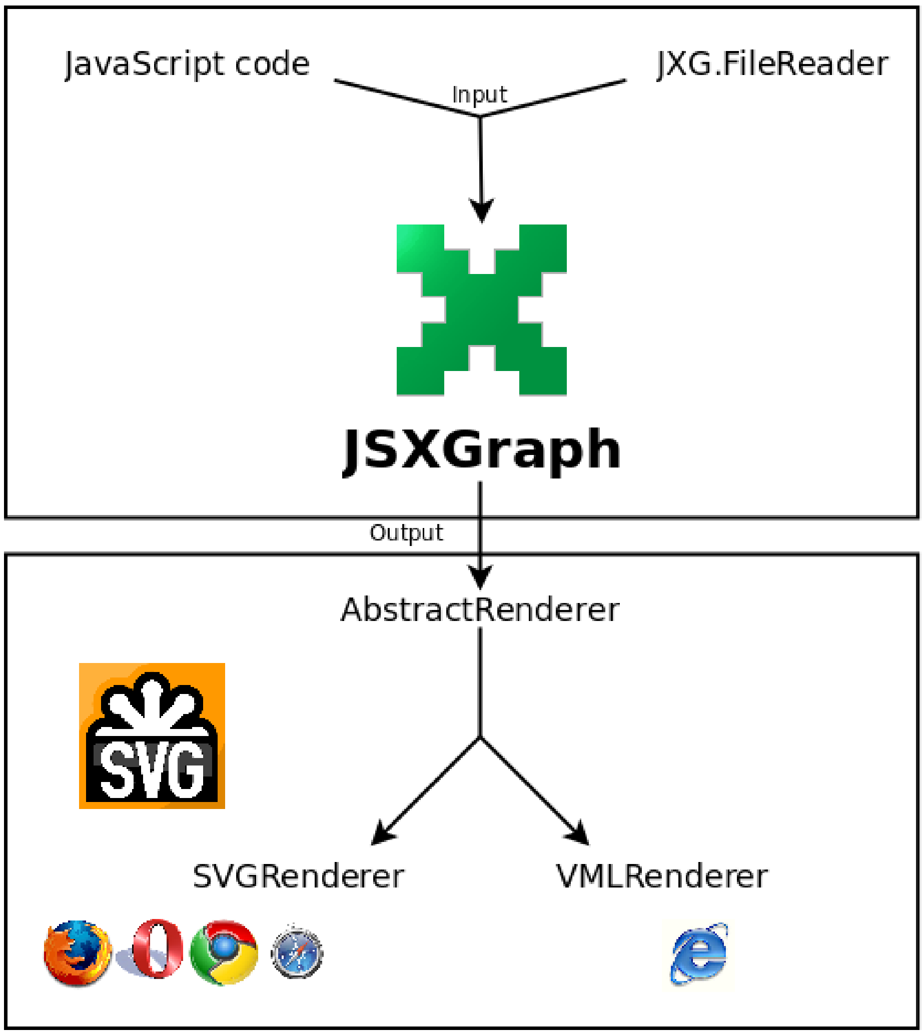

Figure 6: Scheme showing the renderer split in JSXGraph

For SVG drawing JSXGraph rarely uses formatting through CSS. The reason is that in the beginning of the development process the use of CSS styles consumed to much computing resources, especially setting the opacity in Firefox. So, nearly all visual properties are realized by setting attributes and applying filters. Even if filters in SVG are still not implemented satisfactorily in all browsers, JSXGraph supports shadows and gradients via filters since version 0.76. At the time of writing shadows slow down the Firefox browser considerably. Further, there seems to be a bug for horizontal and vertical lines. In these cases neither the line nor its shadow is visible. In Webkit based browsers like Chrome and Safari these filters are not implemented, yet. In Opera it looks ugly. On the other hand, VML realizes shadows as a subnode. There, no noticable speed reduction is observable.

7.3 HTML elements

For the display of text, like labels of points or results of computations, JSXGraph does not use the text element of SVG. Instead, div tags in HTML are used. That means the browsers display the text as HTML elements above the SVG image. In order to update the text the innerHTML property of HTML is used. This is slow, but since many texts are never changed, it is used scarcely. We decided to use HTML elements for SVG, because at that time HTML tags and HTML entities in texts were not supported in the Firefox implementation.

7.4 Speed improvements

In order to enable fast reactions to user interactions like mouse moves, JSXGraph contains many tricks to speed up the computations. Even if the JavaScript interpreters, respectively Just-in-time-compilers, have made huge progress since the development of JSXGraph has been started, special care about implementation details is still necessary. Here, we list a few of the tricks, which proved to be useful for our implementation. Of course, the benefits of most of these tricks depend on the browser implementations and may change with new versions of the browsers.

7.4.1 Avoid DOM accesses with getElementById()

When a new SVG node is created, JSXGraph keeps an internal reference to this DOM node. Later on, access to this node is realized via this pointer variable, exclusively. Using getElementById() or the $() operator of Prototype or jQuery proved to be much too slow.

7.4.2 Suspend the update of SVG nodes

Before a construction update is triggered by a mouse move or mouse up event, the redrawing of the SVG nodes is suspended, until the position of all elements has been recalculated. Then, the redrawing of the SVG nodes is forced.

```
this.suspendHandle = this.svgRoot.suspendRedraw(10000);
// Expensive recalculations ...
this.svgRoot.unsuspendRedraw(this.suspendHandle);
this.svgRoot.forceRedraw();
```

This seems to speed up at least the linux version of Firefox considerably.

7.4.3 Use memoizers

Common functions like binomial() and factorial() are realized with memoizer techniques, as descripted in [Crockford2008], page 44. Thus, the expensive computation of each value of these functions is done only once.

7.4.4 Distinguish mouse move and mouse up events

For curves in JSXGraph there are two update modes, resulting in different output resolutions. During the update following a mouse move event, curves are plotted evaluating only few data points. The default value here is to use 400 points. In the update following a mouse up event curves are plotted with much more points. The default value now is to use 2000 points. These values can be adapted in Options.js.

7.4.5 Reduce the download and initialization time

The JavaScript source code of JSXGraph version 0.76 consists of 20 individual files, which add up to about 980 kBytes including comments. These 20 files are bundled together in one big file which is subsequently compressed by the YUI compressor[20] from Yahoo!. During this process comments and superfluous whitespaces are removed. The resulting file jsxgraphcore.js only consists of 380 kBytes. If the web server delivering jsxgraphcore.js has DEFLATE enabled, which means it compresses its output additionally with gzip, the web browser has to download about 80 kByte. In many cases this is still less than delivering the same construction as a non-interactive PNG image.

7.4.6 Use closures

Sometimes function graphs are expensive to compute. One example is a regression polynomial through a given set of points. On update the position of all points has to be determined, the resulting linear system of equations has to solved. Its solutions give the coefficients of the polynomial. We certainly do not want to do this computation for determining every single function value, since during every update of the function graph we have to compute more than a thousand values of that function.

We apply the following strategy: When the function graph is updated, we compute the coefficients of the polynomial only once. Then we use these coefficients for the computation of all function values. But where to store the coefficients? Using the "this" keyword does not give the desired results in a function. But fortunately, we can use the function invocation pattern, see [Crockford2008], page 28. In the example below, the function which is returned by the call of "generateFunction()" still has access to the variable "coeffs". This is based on the powerful concept of closures, which is part of JavaScript and comes from functional programming languages. So, we can compute many function values very fast, and we do not have to do the bookkeeping, where these coefficients are stored.

[20] http://developer.yahoo.com/yui/compressor/

Example 7.1 Use of the function invocation pattern

```
generateFunction = function(degree,dataX,dataY) {
    var coeffs = [];
    var fct = function(x,suspendedUpdate){
            var i, j, M, MT, y, B, c, s,
                len = dataX.length;

            if (!suspendedUpdate) {  // Recomputation of the  ↩
                cooefficents: expensive
                M = [];
                for (j=0;j<len;j++) {
                    M.push([1]);
                }
                for (i=1;i<=degree;i++) {
                    for (j=0;j<len;j++) {
                        M[j][i] = M[j][i-1]*datax[j];
                    }
                }
                y = dataY;
                MT = JXG.Math.Matrix.transpose(M);

                B = JXG.Math.matMatMult(MT,M);
                c = JXG.Math.matVecMult(MT,y);
                coeffs = JXG.Math.Numerics.Gauss(B, c);
            }

            // Horner's scheme to evaluate polynomial: cheap
            s = coeffs[degree];
            for (i=degree-1;i>=0;i--) {
                s = (s*x+coeffs[i]);
            }
            return s;
        };
        return fct;};
```

7.4.7 An Internet Explorer trick

A very special, but decisive speed improvement has been done for the Internet Explorer versions 6 and 7. With the following piece of code which is executed at initialization time the update speed of the Internet Explorer has been improved by an order of magnitude.

```
function MouseMove(e) {
        document.body.scrollLeft;
        document.body.scrollTop;
    }
    document.onmousemove = MouseMove;
```

We are not aware that this trick has been mentioned before anywhere else.

8 References

8.1 References

[Crockford2008] Douglas. Crockford, *JavaScript: The Good Parts*, ISBN 978-0-596-51774-8, O'Reilly.

[SVG2009] Ola Andersson, Phil Armstrong, Henric Axelsson, Robin Berjon, Benoît Bézaire, John Bowler, Craig Brown, Mike Bultrowicz, Tolga Capin, Milt Capsimalis, Mathias Larsson Carlander, Jakob Cederquist, Charilaos Christopoulos, Richard Cohn, Lee Cole, Don Cone, Alex Danilo, Thomas DeWeese, David Dodds, Andrew Donoho, David Duce, Jerry Evans, Jon Ferraiolo, Darryl Fuller, &#x6df3; &#x85e4;&#x6ca2;, Scott Furman, Brent Getlin, Peter Graffagnino, Rick Graham, Vincent Hardy, &#x8cb4;&#x4e5f; &#x7aef;&#x5c71;, Lofton Henderson, Jan Christian Herlitz, Alan Hester, Bob Hopgood, &#x96c5;&#x5eb7; &#x77f3;&#x5ddd;, Dean Jackson, Christophe Jolif, Lee Klosterman, &#x4e9c;&#x4ee4; &#x5c0f;&#x6797;, Thierry Kormann, Yuri Khramov, Kelvin Lawrence, Håkon Lie, Chris Lilley, Philip Mansfield, Kevin McCluskey, &#x5145; &#x6c34;&#x53e3;, Luc Minnebo, Tuan Nguyen, &#x4fee;&#x4e00;&#x90ce; &#x5c0f;&#x91ce;, Antoine Quint, &#x6bc5; &#x76f8;&#x826f;, Troy Sandal, Peter Santangeli, Haroon Sheikh, Brad Sipes, Peter Sorotokin, Gavriel State, Robert Stevahn, Timothy Thompson, &#x5b8f;&#x9ad8; &#x4e0a;&#x7530;, Rick Yardumian, Charles Ying, and Shenxue Zhou, *Scalable Vector Graphics (SVG) 1.1 Specification*, Copyright © 2009 W3C, ISBN , http://www.w3.org/TR/SVG11/.

[VML1998] John Bowler, Howard Cooperstein, Brian Dister, Ajay Jindal, Daniel Lee, Brian Mathews, Tuan Nguyen, Troy Sandal, and Peter Wu, *Vector Markup Language (VML)*, Copyright © 1998 W3C, ISBN , http://www.w3.org/TR/1998/NOTE-VML-19980513.

8.2 Websites

[CABRI] *http://cabri.com*

[CDY] *http://cinderella.de*

[CaR] *http://zirkel.sourceforge.net/*

[GGB] *http://www.geogebra.org/*

[GXT] *http://www.geonext.de/*

[I2GEO]	*http://i2geo.net*
[R]	*http://r-project.org*
[TGS]	*http://www.dynamicgeometry.com/*