German version

www.walterzorn.de

Home DHTML-Drag&Drop-Library Tooltips Function-Grapher

.drawPolygon( ) .drawRect( ) .fillEllipse( ) .fillPolygon( ) .fillArc( ) .drawLine( ) .drawPolyline( ) .drawEllipse( )

These are not image files.
Shapes & text have been dynamically drawn with wz_jsgraphics.js.

DHTML, JavaScript
Draw Line, Circle, Ellipse (Oval), Polyline, Polygon, Rectangle.
High Performance JavaScript Vector Graphics Library.
Developed by Walter Zorn

Performance?
Cross Browser?
Documentation
Download

This JavaScript VectorGraphics library provides graphics capabilities for JavaScript: functions to draw circles, ellipses (ovals), oblique lines, polylines and polygons (for instance triangles, rectangles) dynamically into a webpage. Usage of this Vector Graphics library should be easy even if you don't have JavaScript experience. Documentation. Another goal during development of this JavaScript Draw Shapes Vector Graphics Library was to achieve optimized performance and cleanly arranged pixel stair-step patterns (pixel-optimization).

Try it out:

drawLine() x1 50 y1 -330 x2 500 y2 800
drawRect() x 130 y 20 w 350 h 60
fillRect() x -70 y 130 w 160 h 100
fillPolygon() x 80,126,-30,-60 y 0,130,40,100
drawEllipse() x 320 y -20 w 250 h 140
fillEllipse() x 75 y 140 w 240 h 400
fillArc() x 350 y 140 w 240 h 400 arc begin 25 ° arc end 290 °
setColor("#00aaaa") setStroke(4) setStroke(Stroke.DOTTED ☐)
clear() this test area

Test Canvas

Here's an example of how to animate (rotate) a vectorgraphic.

Or see how to draw onto an image, using the image itself as canvas (this example uses my DHTML-Drag&Drop-Library).

Top of Page
Performance?
Cross Browser?
Documentation
Download

Performance

In HTML there are no such elements as oblique lines, circles, ellipses or other non-rectangularly bounded elements available. For a workaround, pixels might be painted by creating small background-colored layers (DIV elements), and arranging these to the desired pattern. Creating a separate DIV for each pixel of the pattern, however, would be very inefficient since each of these DIVs inevitably had to lug its complete browser-internal object overhead.

Image dynamically drawn with wz_jsgraphics.js

To minimize the amount of such overhead and to optimize performance to what's possible, this Draw Shapes JavaScript VectorGraphics Library, besides using fast algorithms (based on Bresenham algorithms) to compute the shapes, minimizes the number of generated DIVs by combining as many pixels into each DIV as possible. Therefore, as illustrated by the picture on the left, the number of DIV elements is decreased [1] from one per pixel to one per pixel stair-step. For filled shapes, the advantage of the wz_jsgraphics.js approach is even bigger, as illustrated by the varicoloured filled ellipse below these paragraphs - imagine, by comparison, an ellipse composed of 1*1 pixel DIVs...

worst bad optimal wz_jsgraphics.js

However, don't compare the performance with Java or stand-alone applications!
Making a browser create DIV elements is of course much slower than just colouring pixels [2]. Even this JavaScript Vectorgraphics Library can't escape from this fundamental restriction - it just tries to squeeze out the maximum of what's possible. It's recommended to refrain from creating shapes that extend more than 600 - 1000 pixels in both dimensions.

Alternative to SVG or <canvas> tag?
Both techniques are presently not natively supported by IE. IE's proprietary VML, on the other hand, is not supported by the other browsers. Therefore you might use this Vector Graphics Library as an non-proprietary alternative for small or not-too-complex graphic elements, presumably running on the machines of more than 95% of all internet visitors. Moreover, as this vectorgraphics library allows graphics to overflow the borders of their "canvases", it can render to almost anywhere on a page.

Cross Browser Functionality?

Linux:
Browsers with Gecko-Engine (Mozilla, Netscape 6+, Galeon), Konqueror, Opera 5, 6 and 7+.

Windows:
Gecko-Browsers, IE 4, 5 and 6, Opera 5, 6 and 7+.

Mac:
Safari, Gecko-Browsers, Opera, partially IE.

The functionality "Draw into html elements even after the page has fully loaded" isn't available for Opera prior to version 7, whereas "Draw into the document while the page is parsed" is cross-browser capable.

Documentation: How to Use the JavaScript Graphics Library

1. Download the Library

Download the JavaScript Vector Graphics Library. **Note: Please use the downloaded file only**, do not directly link wz_jsgraphics.js from my site into your site.

2. Include the Library

Un-zipp the file into the same directory as your html file. Insert the following code into the head-section of the html file, between <head> and </head> - if you'd like to have the library in a different directory, adapt the path src="wz_jsgraphics.js":

```
<script type="text/javascript" src="wz_jsgraphics.js"></script>
```

3. HTML: Elements As Canvases

This step isn't required for the mode "Draw directly into the document while the page is parsed". To draw into DIV elements even after the page has fully loaded, however, relatively as well as absolutely positioned layers (DIV elements) are appropriate to serve as canvases. Each of these layers must have a unique ID:

```
<div id="myCanvas" style="position:relative;height:250px;width:100%;"></div>
...
<div id="anotherCanvas" style="position:relative;height:100px;width:300px;"></div>
```

Top of Page
Performance?
Cross Browser?
Documentation
Download

How to Draw Shapes

1. Create a jsGraphics Object

a) Draw into DIV elements even after the page has fully loaded:

(This mode doesn't work in Opera 5 and 6.)

See the example below for how to create a jsGraphics object for a certain DIV element intended to serve as canvas. This code should be inserted into the html code past the concerned DIV element, but in any case ahead of the closing body tag. The constructor function `new jsGraphics()` requires as parameter either the ID of the canvas DIV in quotation marks, or a direct reference to the DIV:

```
<script type="text/javascript">
<!--
var jg = new jsGraphics("myCanvas");
//-->
</script>
```

or

```
<script type="text/javascript">
<!--
var cnv = document.getElementById("myCanvas");
var jg = new jsGraphics(cnv);
//-->
</script>
```

If there are multiple separate DIV elements, each requires its own jsGraphics instance:

```
<script type="text/javascript">
<!--

var jg = new jsGraphics("myCanvas");
var jg2 = new jsGraphics("anotherCanvas");

//-->
</script>
```

b) Draw directly into the document while the page is parsed:

Works even in Opera 5/6. Instead of the ID of a DIV element, nothing (or the value null) must be passed to the constructor function:

```
<script type="text/javascript">
<!--

var jg_doc = new jsGraphics();

//-->
</script>
```

Instead of jg, jg2 or jg_doc you may choose any other variable name, provided it follows the rules for variable names in JavaScript.

2. Functions to Draw Shapes

Once generated, these graphics objects (in this expample jg, jg2 and/or jg_doc) can be used to call the Draw Shapes methods. The shapes generated by a certain graphics object will be drawn into the concerned html element, or, with mode b) used, into the document itself:

```
<script type="text/javascript">
<!--
function myDrawFunction()
{
  jg_doc.setColor("#00ff00"); // green
  jg_doc.fillEllipse(100, 200, 100, 180); // co-ordinates related to the document
  jg_doc.setColor("maroon");
  jg_doc.drawPolyline(new Array(50, 10, 120), new Array(10, 50, 70));
  jg_doc.paint(); // draws, in this case, directly into the document

  jg.setColor("#ff0000"); // red
  jg.drawLine(10, 113, 220, 55); // co-ordinates related to "myCanvas"
  jg.setColor("#0000ff"); // blue
  jg.fillRect(110, 120, 30, 60);
  jg.paint();

  jg2.setColor("#0000ff"); // blue
  jg2.drawEllipse(10, 50, 30, 100);
  jg2.drawRect(400, 10, 100, 50);
  jg2.paint();
}

var jg_doc = new jsGraphics(); // draw directly into document
var jg = new jsGraphics("myCanvas");
var jg2 = new jsGraphics("anotherCanvas");

myDrawFunction();

//-->
</script>
```

As illustrated in this example, at first the color of the "pen" should be set. Otherwise the default color black will be used. The coordinates passed to the Draw Shapes methods refer either, if mode b) used, to the left-top corner of the document itself, or to the left-top corner of the concerned DIV. For each canvas (graphics object) separately, its paint() method must be called explicitly to render the internally-generated graphics html. Otherwise nothing will happen on your screen.

Name of function (method)

Code example as to be used with the above example object 'jg' (which is the context of the DIV called "myCanvas")

General Notes

1.) Numbers passed to these functions must always be **integer numbers**, rather than decimal point numbers (floating point numbers), characters or strings. For instance, values obtained from formular inputs are always strings, and results from previous JavaScript calculations typically floating point numbers. Use the predefined `parseInt()` or `Math.round()` JavaScript functions to convert such values to integers. Example:

```
jg.setStroke(parseInt(document.MyForm.Linewidth.value));
```

2.) Consider that co-ordinates lie between pixels, not on them, and that the drawing "pen" hangs beneath and to the right of the path specified by co-ordinates passed to the functions.

`setColor("#HexColor");`

Specifies the color of the drawing "pen". Once set, this color will be used by the subsequently called drawing methods until it will be overridden through another call of setColor(). The value must be enclosed in quotation marks, and should be hexadecimal following the pattern "#rrggbb" (as usual with HTML). Color names available in HTML (for instance "maroon") may be used as well.

```
jg.setColor("#ff0000");
```

or with identical result

```
jg.setColor("red");
```

`setStroke(Number);`

Specifies the thickness of the drawing "pen" for lines and bounding lines of shapes. Once set, this thickness will be used by the subsequently called drawing methods until it will be overridden through another call of setStroke(). Default line thickness is 1 px, as long as .setStroke() has not yet been invoked.

To create dotted lines, `setStroke()` requires the constant `Stroke.DOTTED` as argument. Width of dotted lines is always 1 pixel.

```
jg.setStroke(3);
```

or

```
jg.setStroke(Stroke.DOTTED);
```

`drawLine(X1, Y1, X2, Y2);`

Line from the first to the second pair of coordinates. Line thickness is either 1 px or the value most recently specified by **`.setStroke()`**.

```
jg.drawLine(20,50,453,40);
```

`drawPolyline(Xpoints, Ypoints);`

A polyline is a series of connected line segments. Xpoints and Ypoints are arrays which specify the x and y coordinates of each point as follows:
`var Xpoints = new Array(x1,x2,x3,x4,x5);`
`var YPoints = new Array(y1,y2,y3,y4,y5);`
Instead of Xpoints and Ypoints you may of course use other names provided these follow the rules for JavaScript variable names.

```
var Xpoints = new Array(10,85,93,60);
var YPoints = new Array(50,10,105,87);
jg.drawPolyline(Xpoints,Ypoints);
```

Line thickness is either 1px or the value most recently specified by **`.setStroke()`**.

`drawRect(X, Y, width, height);`

Outline of a rectangle. X and Y give the co-ordinates of the left top corner. Line thickness is either 1px or the value most recently specified by **`.setStroke()`**.

```
jg.drawRect(20,50,70,140);
```

`fillRect(X, Y, width, height);`

Filled rectangle. X and Y give the co-ordinates to the left top corner.

```
jg.fillRect(20,50,453,40);
```

`drawPolygon(Xpoints, Ypoints);`

Polygon. Xpoints and Ypoints are arrays which specify the x and y coordinates of the polygon's corners as follows:
`var Xpoints = new Array(x1,x2,x3,x4,x5);`
`var Ypoints = new Array(y1,y2,y3,y4,y5);`
The polygon will be automatically closed if the first and last points are not identical.

```
var Xpoints = new Array(10,85,93,60);
var Ypoints = new Array(50,10,105,87);
jg.drawPolygon(Xpoints, Ypoints);
```

Instead of Xpoints and Ypoints you may of course use other names provided these follow the rules for variable names.

Line thickness is either 1px or the value most recently specified by **`.setStroke()`**.

`fillPolygon(Xpoints, Ypoints);`

Filled Polygon. Parameters as for drawPolygon()

```
jg.fillPolygon(new Array(10,85,93,60),
new Array(50,10,105,87));
```

`drawEllipse(X, Y, width, height);`

Outline of an ellipse. Values refer to the bounding rectangle of the ellipse, X and Y give the co-ordinates of the left top corner of that rectangle rather than of its center. Line thickness is either 1px or the value most recently specified by **`.setStroke()`**.

```
jg.drawEllipse(20,50,70,140);
```
or
```
jg.drawOval(20,50,70,140);
```

`fillEllipse(X, Y, width, height);`

Filled ellipse. Values refer to the bounding rectangle of the ellipse, X and Y give the co-ordinates of the left-top corner of that rectangle rather than of its center.

```
jg.fillEllipse(20,50,71,141);
```
or
```
jg.fillOval(20,50,71,141);
```

`fillArc(X, Y, width, height, start-angle, end-angle);`

Fills a pie section of an ellipse. Start-angle and end-angle may be integer numbers or decimalpoint values. Like with the other ...Ellipse() functions, X and Y specify the left-top corner of the bounding rectangle.

```
jg.fillArc(20,20,41,12,270.0,220.0);
```

`setFont("font-family", "size+unit", Style);`

This method can be invoked prior to drawString() to specify or change font-family, -size and -style. Values or font-family and -size may be whatever possible in HTML, and must be enclosed in quotation marks.

Available font styles:
Font.PLAIN for normal style (not bold, not italic)
Font.BOLD for bold fonts
Font.ITALIC for italics
Font.ITALIC_BOLD or Font.BOLD_ITALIC to combine the latters.

Example: see `drawString()` below

`drawString("Text", X, Y);`
I've been drawn with the Vectorgraphics Library...
Writes text to the location specified by X and Y. Differently from Java, these coordinates refer to the left top corner of the first line of the text. The string passed to drawString() must be enclosed in quotation marks. (Non-escaped) HTML tags inside the string will be interpreted. For example, "Some Text
more Text" would indeed create a line break.

```
jg.setFont("arial","15px",Font.ITALIC_BOLD);
jg.drawString("Some Text",20,50);
```

`drawStringRect("Text", X, Y, Width, Alignment);`

A text drawn by using drawStringRect() which allows to set the width of the text rectangle and to align the text horizontally (in this case "right")

```
jg.setFont("verdana","11px",Font.BOLD);
jg.drawStringRect("Text",20,50,300,"right");
```

Like drawString. Allows however to set the width of the text rectangle and to specify the horizontal text-alignment. Text-alignment value must be a string (i.e. enclosed in quotation marks or apostrophes) and can be either "left", "center", "right" or "justify".

`drawImage("src", X, Y, width, height);`

Draws image at the specified location. The "src" parameter specifies the file path. The width and height parameters are optional (may be 0, null or omitted, in which case the image is displayed in its default size), but provide the option to stretch the image (almost) arbitrarily.

```
jg.drawImage("friendlyDog.jpg", 20,50,100,150);
```

Optionally, drawImage() accepts a fifth parameter which you can use to insert an eventhandler into the generated image tag. Example:
`jg.drawImage('anImg.jpg',8,5,95,70,'onmouseover="YourFunc()"');`

`paint();`

Must be envoked explicitly to draw the internally-generated graphics into the html page. To optimize performance it's recommended to restrain from calling `paint()` in unnecessarily short intervals.

```
jg.paint();
```

Avoid something like:

```
jg.drawEllipse(0, 0, 100, 100);
jg.paint();
jg.drawLine(200, 10, 400, 40);
jg.paint();
...
```

The following will be faster:

```
jg.drawEllipse(0, 0, 100, 100);
jg.drawLine(200, 10, 400, 40);
/*...further drawing methods... */
jg.paint();
```

clear();

Any content created by the Graphics JavaScript Library will be deleted (within the canvas the graphics object refers to). The default content of the canvas (content not created by the script) will remain untouched, i.e. neither be changed nor be deleted.

```
jg.clear();
```

Any stuff the script has drawn to "myCanvas" is deleted (assuming that "myCanvas" is the DIV for which 'jg' has been created).

setPrintable(true**);**

By default, printing shapes isn't feasible since default printing settings of browsers usually prevent background colors from being printed. Invoking setPrintable() with the parameter true enables wz_jsgraphics.js to draw printable shapes (at least in Mozilla/Netscape 6+ and IE). However, at the price of a slightly decreased rendering speed (about 10% to 25% slower).

```
jg.setPrintable(false);
```

The parameter false switches wz_jsgraphics.js back to non-printable mode. The benefit from this, however, will be re-optimized rendering performance.

Top of Page
Performance?
Cross Browser?
Documentation
Download

Download

wz_jsgraphics.js 3.05, published under the **LGPL**:
wz_jsgraphics.zip (7 KB)

History of Updates (it's recommended to read this, especially if you aren't sure about the benefits from an update :-)).

Donation

Financial support for the very numerous hours of development and for the costs of this website is of course welcome.

Notes

[1]
If we assume the probability of line angles on the screen to be equally distributed over 360°:
Then for 1 px wide lines the number of layers is averagely **sin(45°) / (1 - sin(45°)) = 2.41** times larger if for each pixel a separate layer is created, compared with a solution that requires merely one layer per pixel stair-step (as is the case with this library). This is true under the assumption that both algorithms let pixel stair-steps touch each other corner-by-corner rather than side-by-side. Otherwise the number of pixels to be colored would be larger, and therefore the disadvantage of an unfavorable one-layer-per-pixel algorithm even bigger.
Back

[2]
Something like a really lightweight <pixel color="#ff9900" left="127" top="63"> tag would be nice.
Back

Walter Zorn, Munich
Impressum and about this page