Dynamic SVG generation under Firefox 1.5 using JavaScript, XML and XSLT

Dr. Thomas Meinike

Merseburg University of Applied Sciences
Department of Computer Science and Communication Systems

thomas.meinike@hs-merseburg.de
http://www.hs-merseburg.de/~meiniket/

XTech 2006 Conference, Amsterdam

2006-05-18

About the referent

- university teacher for online documentation and web development

- interest in computer science and programming since 1987

- studies in natural sciences until 1996 (diploma + PhD)

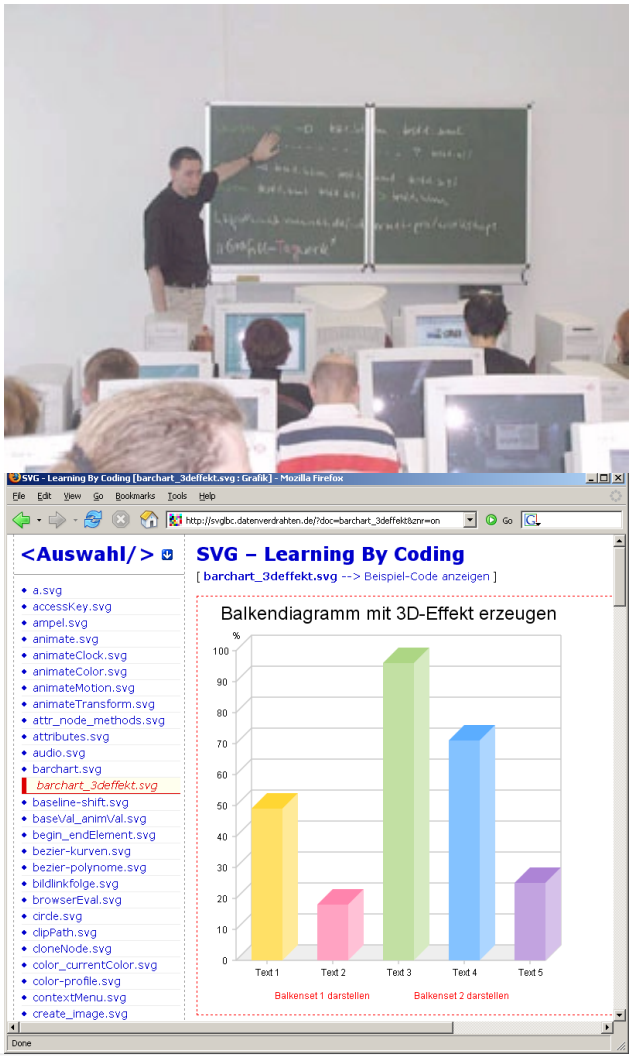
- „web guy“ since the early 1990ies

- XML worker since the first days

- five years experience in SVG development

- webmaster of „SVG – Learning By Coding“
  http://svglbc.datenverdrahten.de

- author for (german) print magazines
  Entwickler Magazin, Internet Professionell



Agenda

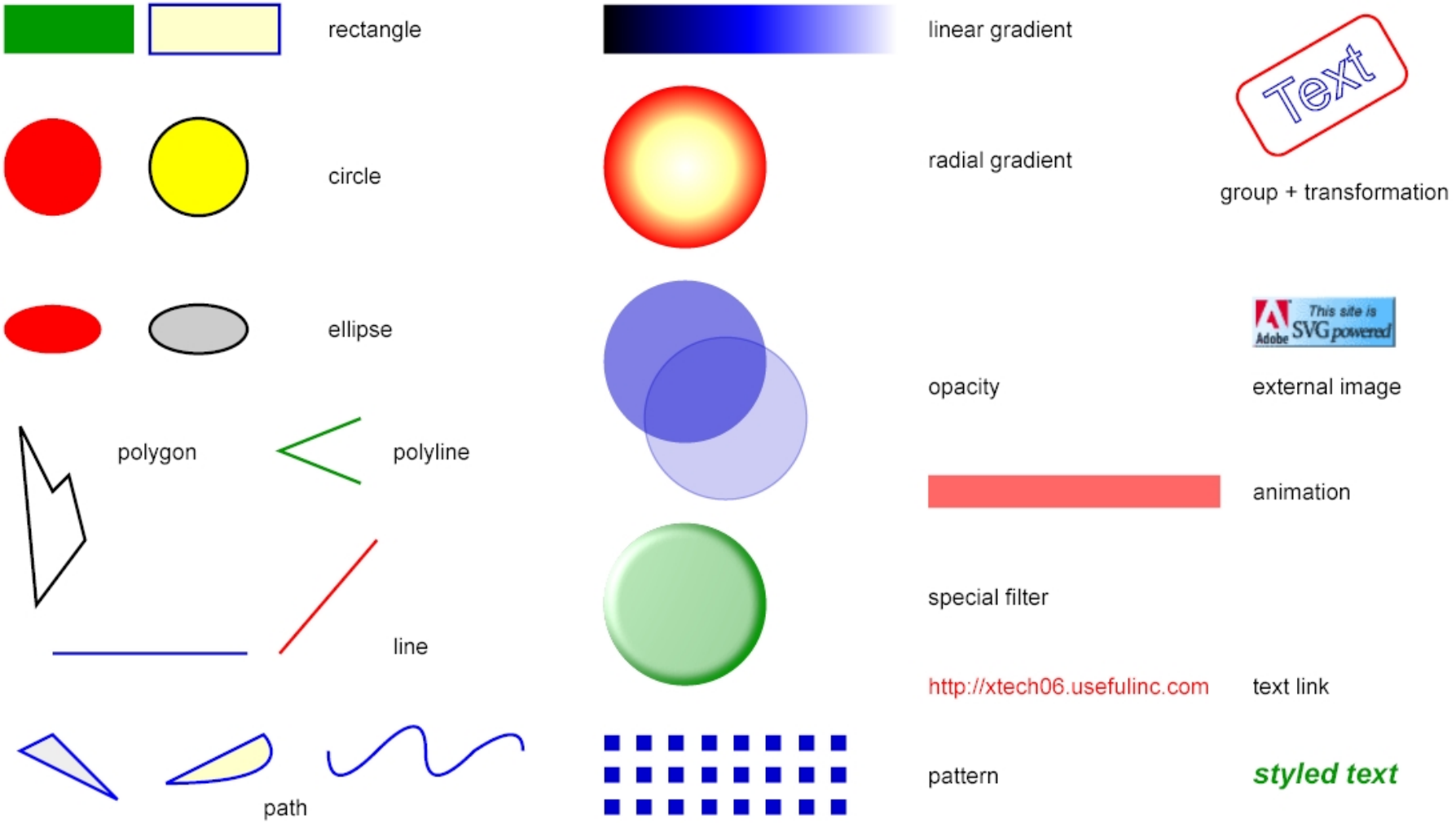
- **Introduction**
  - What is SVG?
  - Structure of SVG documents
  - The coordinate system
  - Basic shapes and techniques
  - Special operations
  - DOM Scripting
- **Firefox 1.5 SVG status**
  - (Un)Implemented features
  - Enhancements for dynamic code generation/manipulation
- **Practical demonstrations**

Introduction – What is SVG?

- W3C specification for vector graphics
  versions: 1.0 (2001), 1.1 (2003), 1.2 (2006?)
- XML based vocabulary and grammar (DTDs for 1.0/1.1)
- SVG is like HTML for vector graphics
- Provides vector shapes, text placement and special techniques
  (clipping, pattern, symbols, gradients, filters, transformations, ...)
- SMIL based animation elements are included
- Content is stylable using CSS or presentation attributes
- Event driven DOM scripting for enhanced dynamics and interactivity
- SVG is ready for mobile applications (SVG 1.1 Tiny and Basic profiles)
- SVG in browsers: native (Opera 8/9, Firefox 1.5+, ...) or with plug-in (ASV)

Introduction – What is SVG?

SVG – objects and effects



Introduction – Structure of SVG documents

⇨ SVG skeleton:

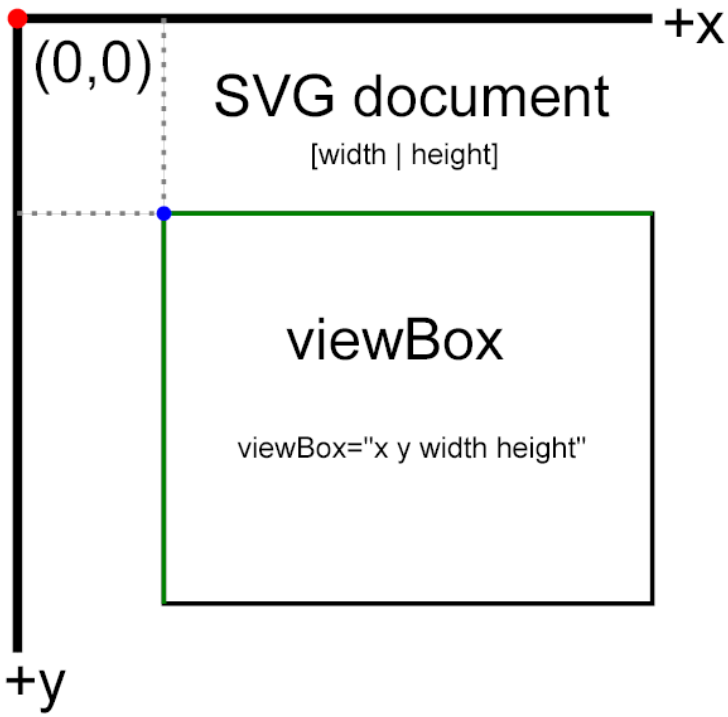
```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg" [*]
     xmlns:xlink="http://www.w3.org/1999/xlink">

  <title>optionally title</title>
  <desc>optionally description</desc>
  <defs>stylesheets, scripts, references</defs>

  <!-- more SVG content -->

</svg>
```

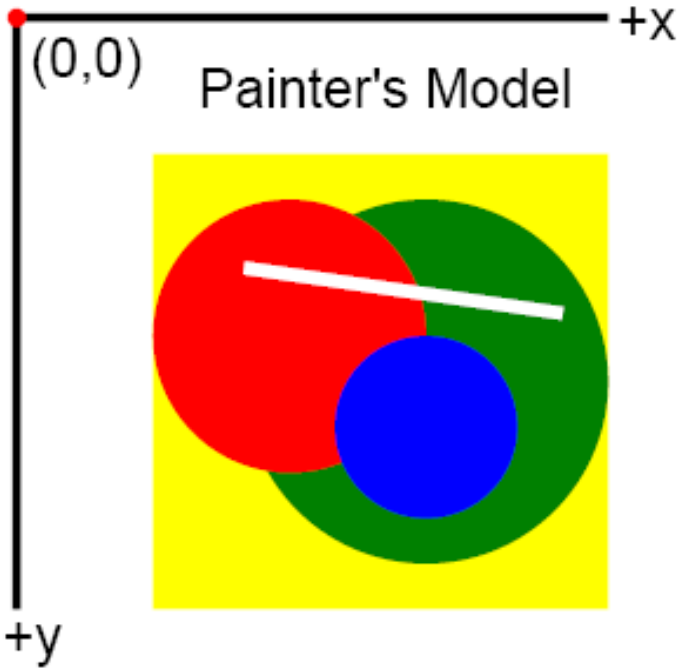
[*] more attributes available:

height, width, viewBox, version, ...

Introduction – The coordinate system

⇨ Coordinate system with negative y axis:



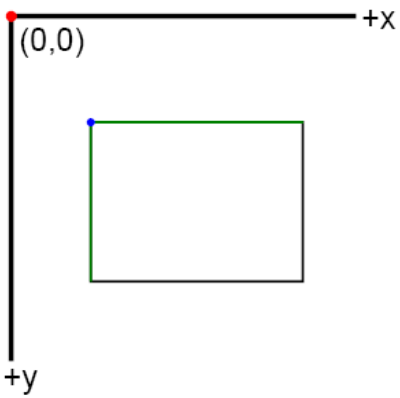


⇨ Image rendering order follows the XML document order:

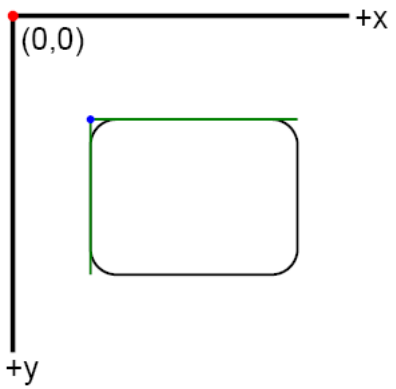
Introduction – Basic shapes and techniques

- Element rect:

```
<rect x="..." y="..." width="..." height="..."
      rx="... " ry="..."/>

 x, y   = coordinates of the upper left corner
 rx, ry = radii for rounded corners (optionally)
```




```
<rect x="30" y="40"
 width="80" height="60"/>
```



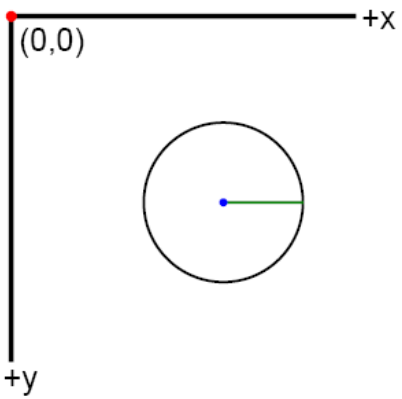

```
<rect x="30" y="40" width="80"
 height="60" rx="10" ry="10"/>
```

Introduction – Basic shapes and techniques

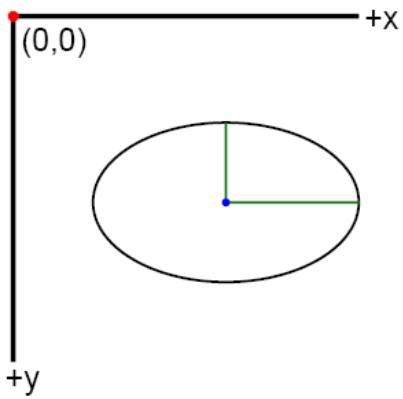
- Elements circle and ellipse:

```
<circle cx="..." cy="..." r="..."/>

<ellipse cx="..." cy="..." rx="..." ry="..."/>
```

cx, cy = coordinates of the center point
r, rx, ry = radii




```
<circle cx="80" cy="70" r="30"/>
```

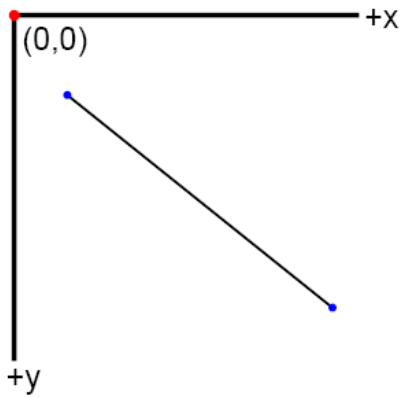



```
<ellipse cx="80" cy="70"
 rx="50" ry="30"/>
```

Introduction – Basic shapes and techniques

- Element line:

```
<line x1="..." y1="..." x2="..." y2="..."/>

 x1, y1 = coordinates of the first point
 x2, y2 = coordinates of the second point
```



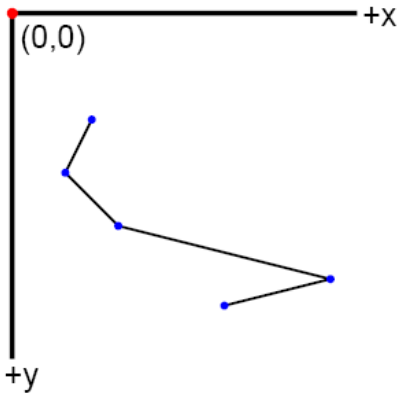

```
<line x1="20" y1="30"
 x2="120" y2="110"/>
```

Introduction – Basic shapes and techniques

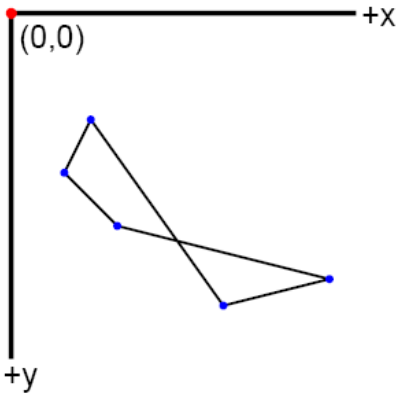
- Elements polyline and polygon:

```
<polyline points="x1,y1 x2,y2 ... xn,yn"/>

<polygon points="x1,y1 x2,y2 ... xn,yn"/>
```

points = point pairs describe a series of connected lines (separated by spaces and/or commas)




```
<polyline points="30,40 20,60
 40,80, 120,100 80,110"/>
```




```
<polygon points="30,40 20,60
 40,80, 120,100 80,110"/>
```

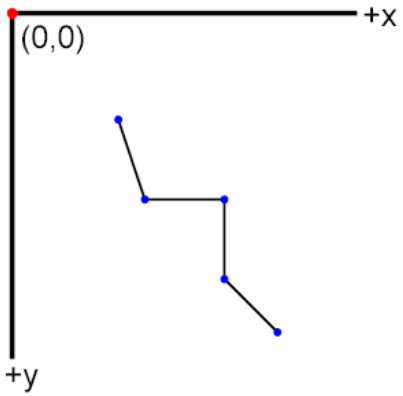
Introduction – Basic shapes and techniques

- Element path:

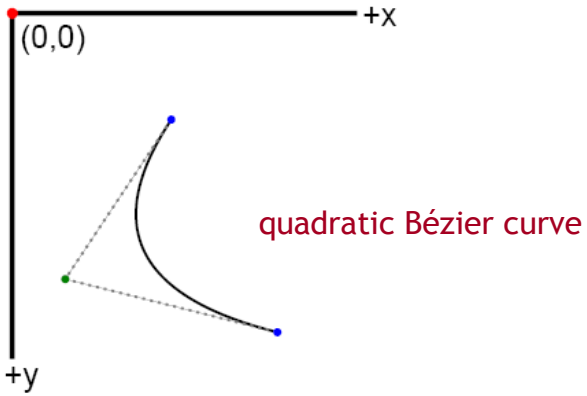
```
<path d="..."/>

 d = path data:

 [M,m]x,y                    : moveto
 [L,l]x,y                    : lineto
 [H,h]x                      : horizontal lineto
 [V,v]y                      : vertical lineto
 [C,c]x1,y1 x2,y2 x,y        : cubic Bézier curveto
 [S,s]x2,y2 x,y              : smooth cubic curveto
 [Q,q]x1,y1 x,y              : quadratic Bézier curveto
 [T,t]x,y                    : smooth quadratic curveto
 [A,a]rx,ry x-axis-rotation large-arc,sweep x,y : elliptical arc
 [Z,z]                       : closepath

 [...]upper letters = absolute coordinates
      lower letters = relative coordinates
```

Introduction – Basic shapes and techniques

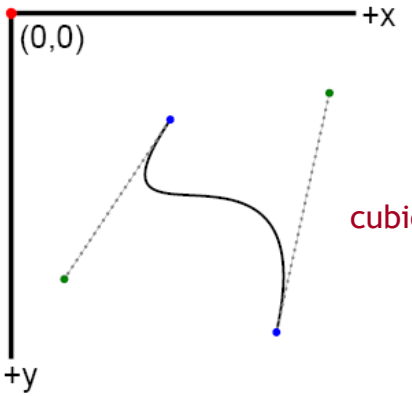
⇨ Element path:




```
<path d="M40,40 L50,70 H80 V100 L100,120"/> (abs.)

<path d="M40,40 l10,30 h30 v30 l20,20"/>    (rel.)
```



quadratic Bézier curve



cubic Bézier curve

```
<path d="M60,40 Q20,100 100,120"/>
```

```
<path d="M60,40 C20,100 120,30 100,120"/>
```

Introduction – Basic shapes and techniques

- Element text:

```
<text x="..." y="...">text content</text>
```

- Element a (hyperlinks, text or shape elements):

```
<a xlink:href="...">
  <text x="..." y="...">Linktext</text>
</a>

<a xlink:href="..."><circle ... /></a>
```

- Element image (JPEG, PNG, SVG):

```
<image xlink:href="..."
  x="..." y="..." width="..." height="..."/>
```

Introduction – Basic shapes and techniques

⇨ Presentation attributes vs. styling with CSS:

- red circle with a blue 2px border:

```
<circle cx="..." cy="..." r="..." fill="red"
  stroke="blue" stroke-width="2px"/>

<circle cx="..." cy="..." r="..." style="fill: red;
  stroke: blue; stroke-width: 2px"/>
```

Use presentation attributes or separated CSS rules!

- style element or <?xml-stylesheet ... ?> PI
- Note: No CSS support in SVG 1.1 Tiny!

```
circle
{
  fill: red;
  stroke: blue;
  stroke-width: 2px;
}
```

Introduction – Basic shapes and techniques

⇨ Some attributes / CSS properties:

- fill color: fill (value *none* for non-filling)
- outline color: stroke
- outline width: stroke-width
- dashed lines: stroke-dasharray

ways to define colors:
red
#F00
#FF0000
rgb(255,0,0)
rgb(100%,0%,0%)

- opacity/transparency: opacity (0...1, 1 = full opaque)
  also defined fill-opacity and stroke-opacity

- font informations: font-size, font-family, font-style
- text markup: text-decoration, text-transform

- show or hide objects: display, visibility

- more known properties from the CSS 2 specification and
  special SVG related definitions

Introduction – Basic shapes and techniques

⇨ Including SVG in HTML documents:

- W3C conform ways with **object** or **iframe** elements:

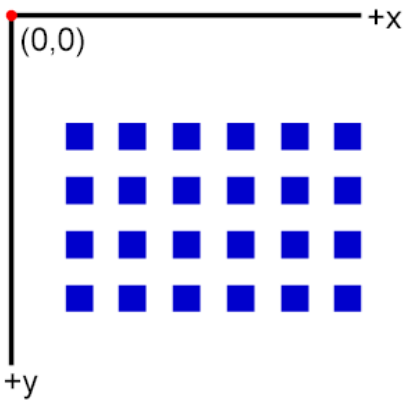
```
<object data="file.svg" width="..." height="..."
  type="image/svg+xml">
  <!-- alternative content -->
</object>

<iframe src="file.svg" width="..." height="..."
  frameborder="0">
  <!-- alternative content -->
</iframe>
```

- embed element, maybe useful for scripting access to the embedded SVG:

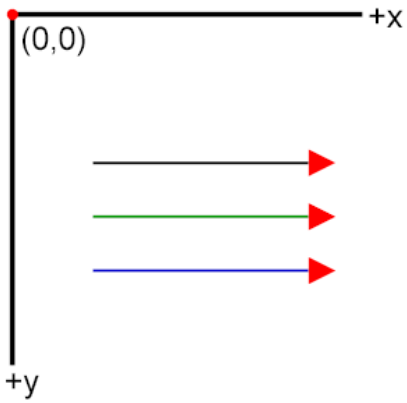
```
<embed src="file.svg" width="..." height="..."
  type="image/svg+xml">
  <!-- alternative content -->
</embed>
```

Introduction – Special operations

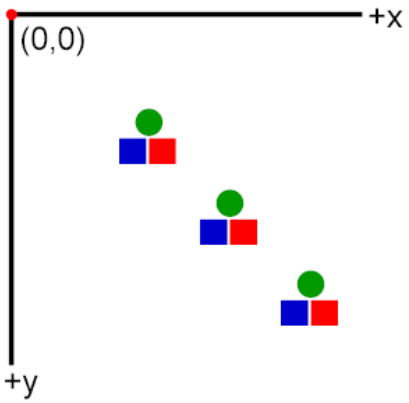
pattern



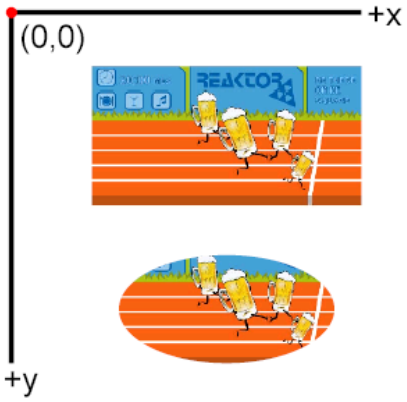
marker



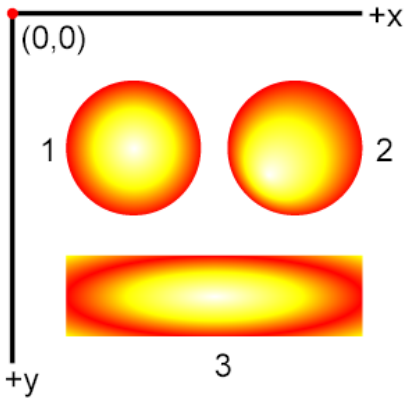
symbol



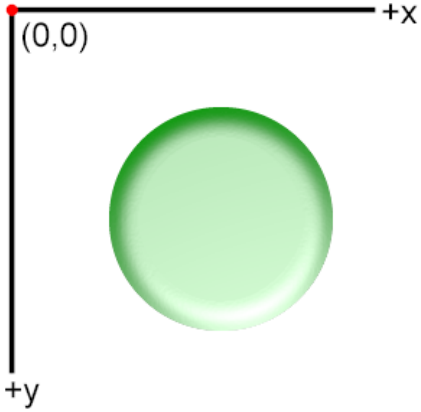
clipPath



gradient
(linear | radialGradient)



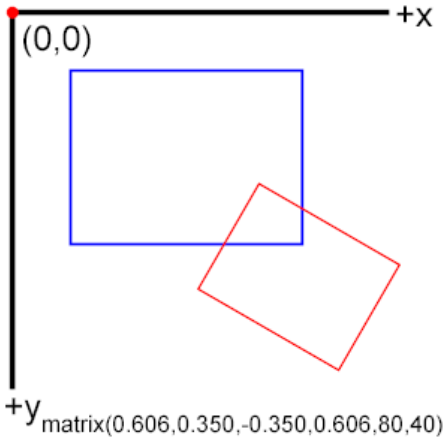
filter (fe*)



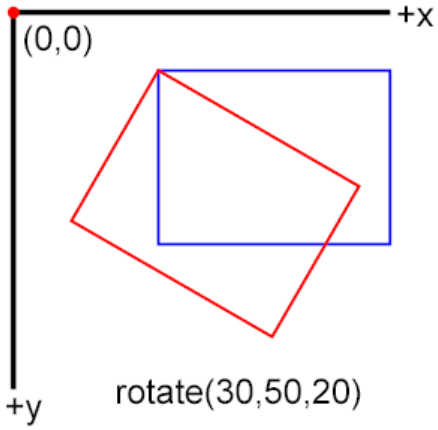
Enhanced painting/rendering

Introduction – Special operations

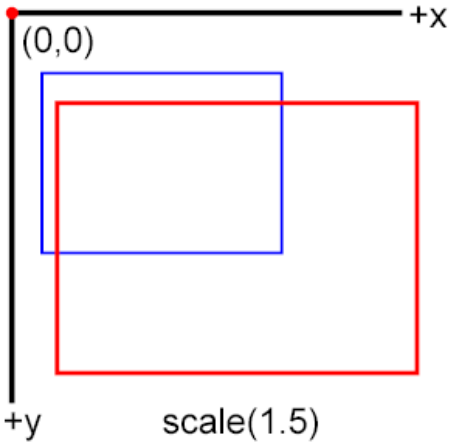
matrix



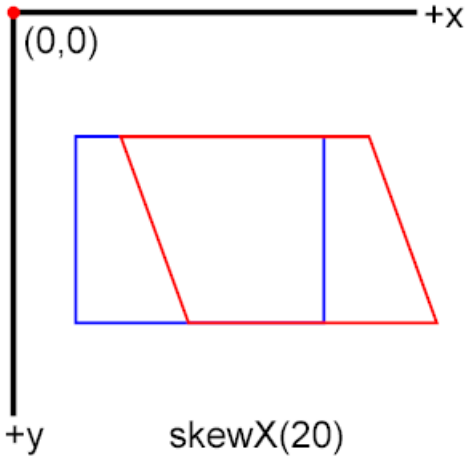
rotate



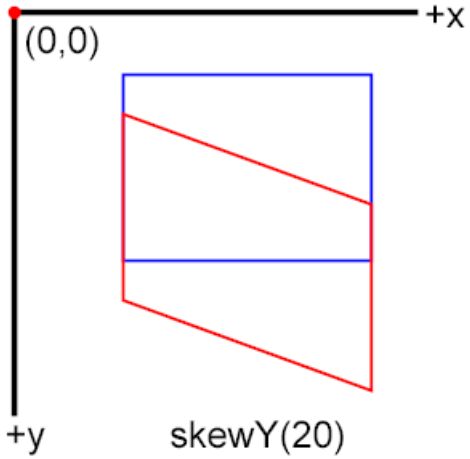
scale



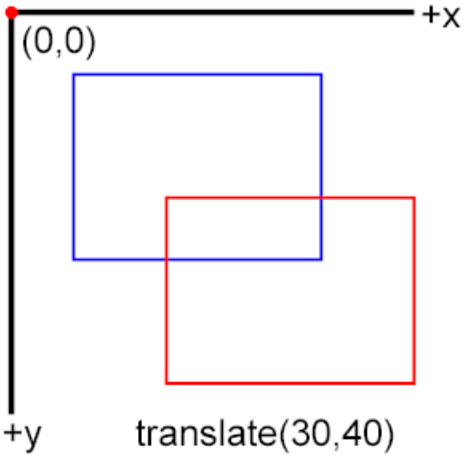
skewX



skewY



translate



Coordinate system transformations

Introduction – DOM Scripting

⇨ Code should be placed in defs-elements

- Internally (CDATA section):

```
<defs>
   <script type="text/ecmascript">
   <![CDATA[

     /* script Code */

   ]]>
   </script>
</defs>
```

- Externally (text files or compressed text files; building reusable libraries):

```
<script xlink:href="file.js" type="text/ecmascript"/>
<script xlink:href="file.js.gz" type="text/ecmascript"/>
```

Introduction – DOM Scripting

⇨ Definition of variables and functions

- Functions encapsulate the code for a specific job:

```
var ...; // global variables


function name_of_function1(arg)
{
  var ...; // local variables
  /* more function code */
}


function name_of_function2(arg1,arg2,...)
{
  var ...; // local variables
  /* more function code */
}
```

Introduction – DOM Scripting

⇨ Definition of variables and functions

- Functions will be called event-based (onclick, onload, ...):

```
<circle cx="50" cy="50" r="20" fill="#F00"
  onclick="myFunction(evt)"/>
```

```
function myFunction(evt)
{
  var obj;
  obj=evt.target;
  obj.setAttributeNS(null, "fill", "#090");
}
```

Introduction – DOM Scripting

⇨ Useful techniques

- Get access to the SVG document:

```
<svg ... onload="init(evt)">...

 <script ...>

  var svgdoc, svgroot; // global variables

  function init(evt)
  {
    svgdoc=evt.target.ownerDocument;
    svgroot=svgdoc.documentElement;
  }
 </script>

...</svg>
```

svgdoc → binding for getElementById(...) access

svgroot → binding for properties like currentScale, currentTranslate and appendChild(...) operations to the root element.

Introduction – DOM Scripting

⇨ Useful techniques

- Some methods and properties from W3C DOM Level 2:

createElementNS(), createTextNode()
appendChild(), cloneNode(), insertBefore()
getAttributeNS(), setAttributeNS()
getElementById(), getElementsByTagNameNS()
getPropertyValue(), setPropertyValue()
firstChild, lastChild, childNodes
...

- Some methods and properties from the separate SVG DOM:

getComputedTextLength() [of a text]
getTotalLength() [of a path]
getBBox() [bounding box of an element]
currentScale, currentTranslate [zoom and pan informations]
...

Firefox 1.5 SVG status

⇨ Supported features (elements)

- Clip Module: clipPath
- Conditional Processing Module: switch
- Gradient Module: linearGradient, radialGradient, stop
- Hyperlinking Module: a
- Image Module: image
- Marker Module: marker
- Scripting Module: script
- Structure Module: defs, desc, g, metadata, svg, symbol, title, use
- Shapes Module: circle, ellipse, line, path, polygon, polyline, rect
- Style Module: style
- Text Module: text, tspan

Firefox 1.5 SVG status

⇨ Unsupported features (elements)

- Animation Module:
  animate, animateColor, animateMotion, animateTransform, mpath, set
- Color Profile Module: color-profile
- Cursor Module: cursor
- Extensibility Module: foreignObject (implemented, but not built)
- Filter Module: filter + 24 filter elements (fe*)
- Font Module: font, font-face, font-face-src, glyph, missing-glyph, ...
- Mask Module: mask
- Pattern Module: pattern
- Text Module: altGlyph, altGlyphDef, altGlyphItem, glyphRef, textPath, tref
- View Module: view

Firefox 1.5 SVG status

⇨ Enhancements for dynamic code generation/manipulation

- AJAX techniques
  XMLHttpRequest()

- XML Parser „expat“
  DOMParser(), XMLSerializer(), parseFromString(), serializeToString()

- XSLT Processor „TransforMiiX“
  XSLTProcessor(), transformToDocument(), transformToFragment()

- XPath techniques
  XPathEvaluator(), createNSResolver(), evaluate()

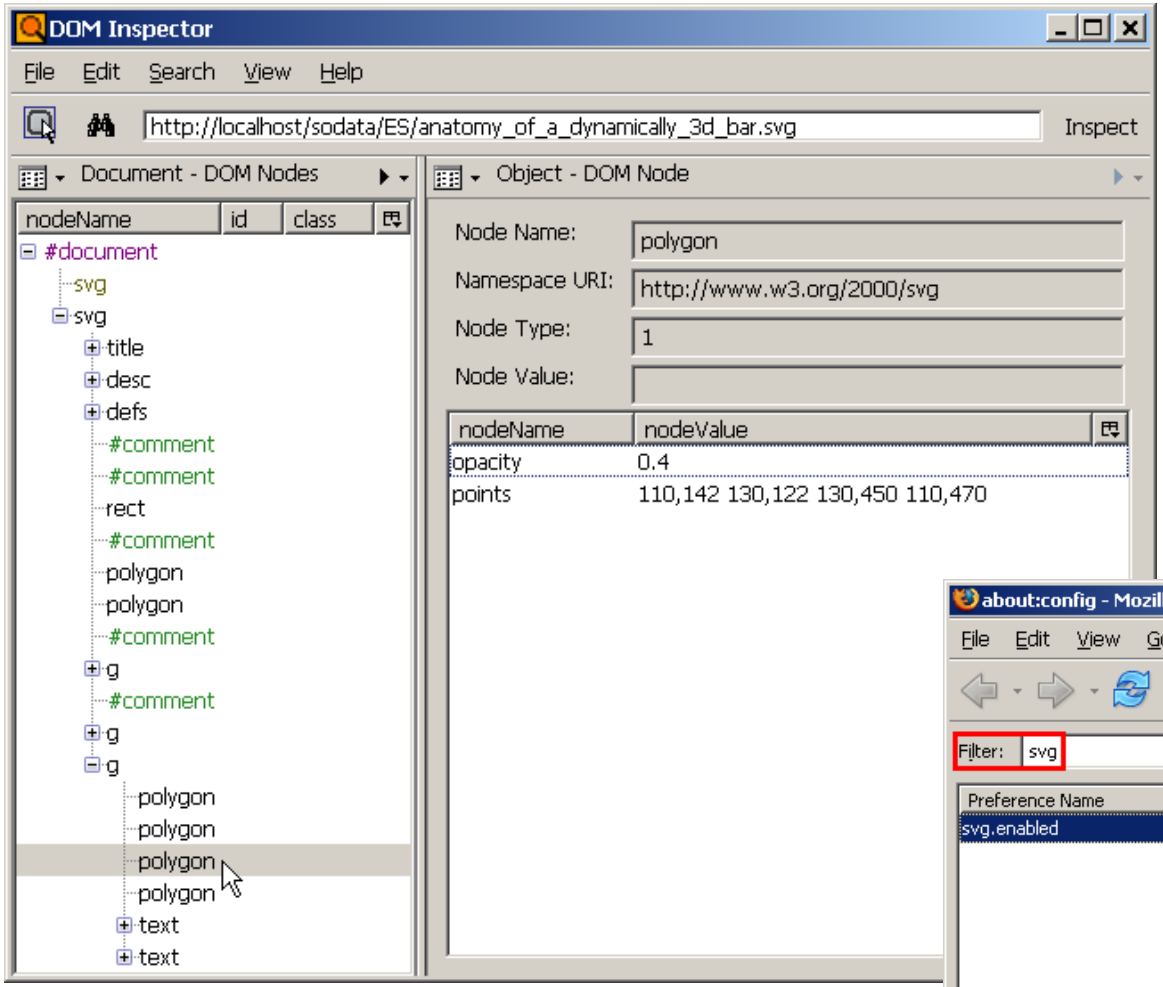
- ECMAScript for XML (E4X)

- Integration of SVG content in XHTML content
  (using namespaces)

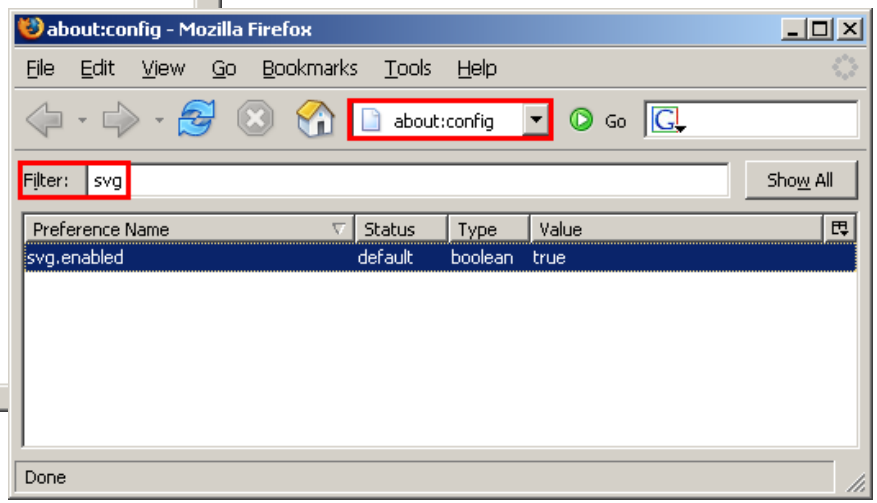
See the examples for more details!

Firefox 1.5 SVG status

⇨ Tools → DOM Inspector



⇨ Is SVG enabled?

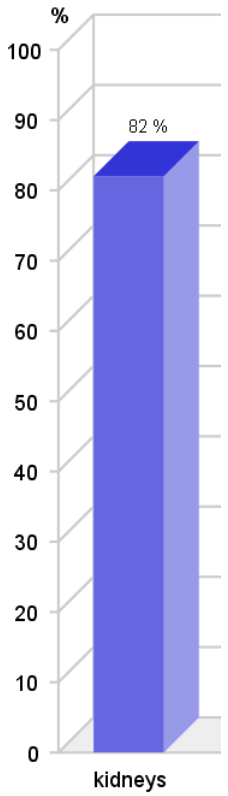


Practical demonstrations

⇨ Creating a 3D bar chart with polygons from XML data

- XML data source collected from a survey

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE data SYSTEM "data.dtd">
<data infotext="Survey: The most transplanted organ(s)?">
<!-- values in percent -->
  <set>
    <value>82</value>
    <descr>kidneys</descr>
    <color>#00C</color>
  </set>
  <set>
    <value>12</value>
    <descr>liver</descr>
    <color>#090</color>
  </set>
  <set>
    <value>4</value>
    <descr>heart</descr>
    <color>#F00</color>
  </set>
  <set>
    <value>2</value>
    <descr>lungs</descr>
    <color>#FF0</color>
  </set>

</data>
```



DTD:

```
<!ELEMENT data   (set*)>
<!ELEMENT set    (value, descr, color)>
<!ELEMENT value  (#PCDATA)>
<!ELEMENT descr  (#PCDATA)>
<!ELEMENT color  (#PCDATA)>
<!ATTLIST data   infotext CDATA #IMPLIED>
```

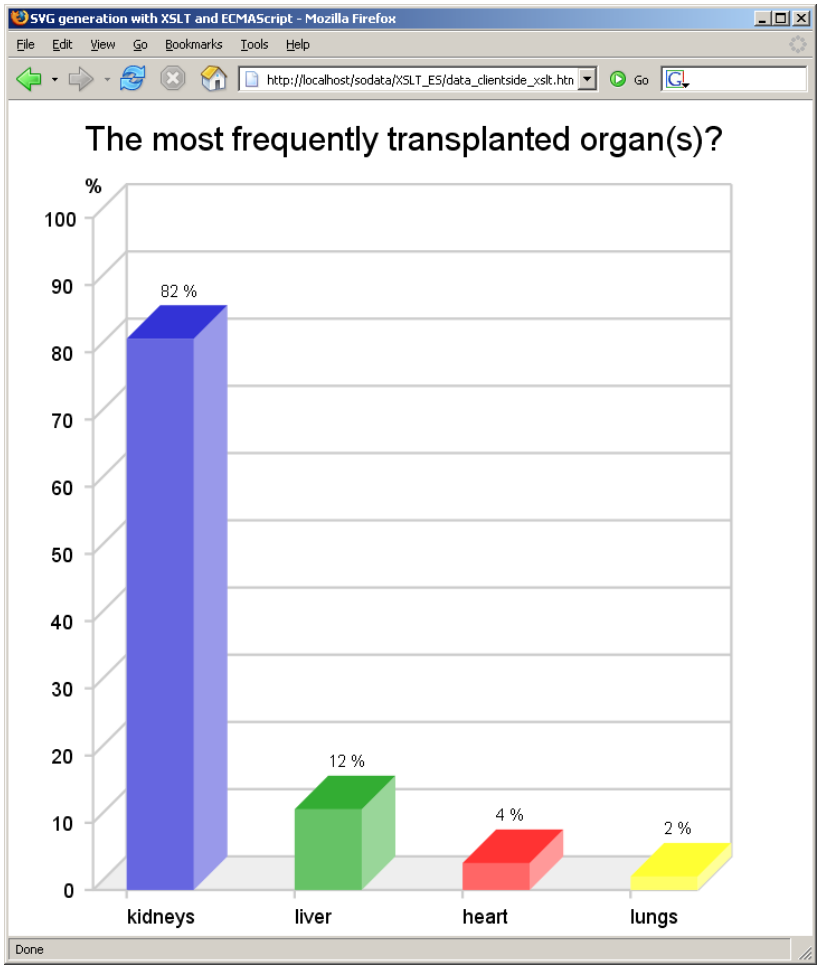
Practical demonstrations

⇨ Creating a 3D bar chart with polygons from XML data

- Used 3D bars consist of three polygons placed side by side

- SVG generation with DOM Scripting only
  [with „create/append“ methods
  as library create3DBars.es;
  data access using XMLHttpRequest(),
  and some of the additional XML features]

- XSL Transformation of the XML data into SVG code
  at the client side (using ECMAScript)
  at the server side (using PHP)
  using animation library „SmilScript“
  from http://vectoreal.com/smilscript



Practical demonstrations

⇨ Creating a 3D bar chart with polygons from XML data

- ECMAScript code (main steps)

```
function create3DBars()
{
  var ...; // definition of variables ...

  // building headline ...

  for(i=0; i<max; i++)
  {
    new_g=svgdoc.createElementNS(svgns,"g");
    new_g.setAttributeNS(...); // id, fill, stroke

    // white area back polygon (6 points) ..., front polygon (4 points) ...
    new_front_polygon=svgdoc.createElementNS(svgns,"polygon");
    new_front_polygon.setAttributeNS(...); // points, opacity

    // side polygon (4 points) ..., top polygon (4 points) ...
    // x-axis marker ..., x-axis text ...
    // value on top of the bars (with background rectangle) ...

    // append element nodes as child nodes of the new_g element
    new_g.appendChild(new_back_polygon);
    ...
    new_g.appendChild(new_bar_text);

    // append the new_g node as child node of the root element
    svgroot.appendChild(new_g);
  }
}
```

Practical demonstrations

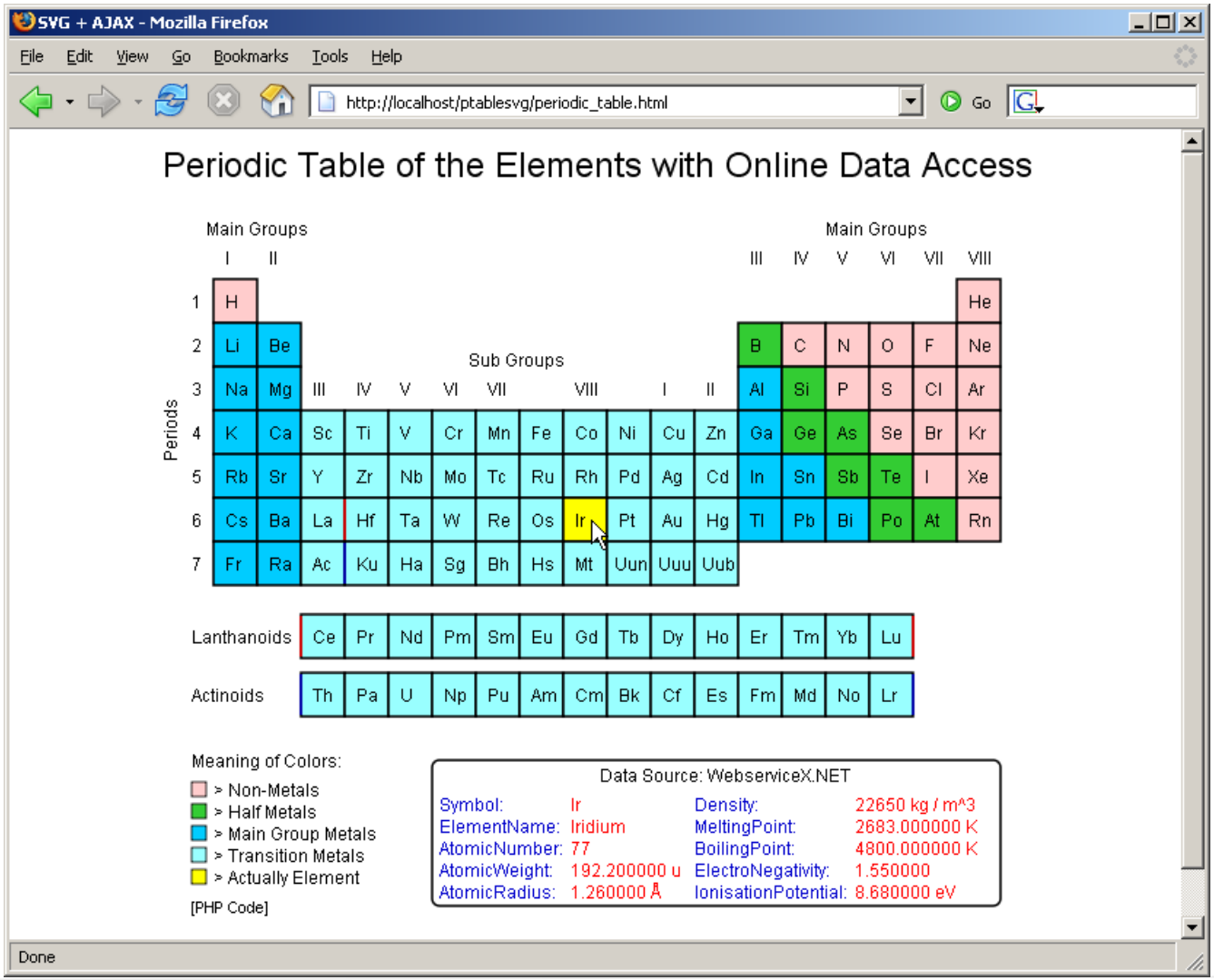
⇨ Creating a 3D bar chart with polygons from XML data

- XSL stylesheet (basic structure)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" ... />
<xsl:template match="data">

<!-- definition of variables ... -->
<svg version="1.0"
     xmlns="http://www.w3.org/2000/svg"
     viewBox="..." width="100%" height="100%">

<!-- building the static part of the diagram ... -->
<!-- data analysis -->
<xsl:for-each select="set">
  <xsl:sort select="value" data-type="number" order="descending"/>
  <xsl:variable name="i" select="position() - 1"/>

  <g id="b{$i + 1}" fill="{color}" stroke="none">
    <!-- processing the content of all value and descr
       elements into polygon and text elements ... -->
  </g>

</xsl:for-each>

</svg>

</xsl:template>
</xsl:stylesheet>
```

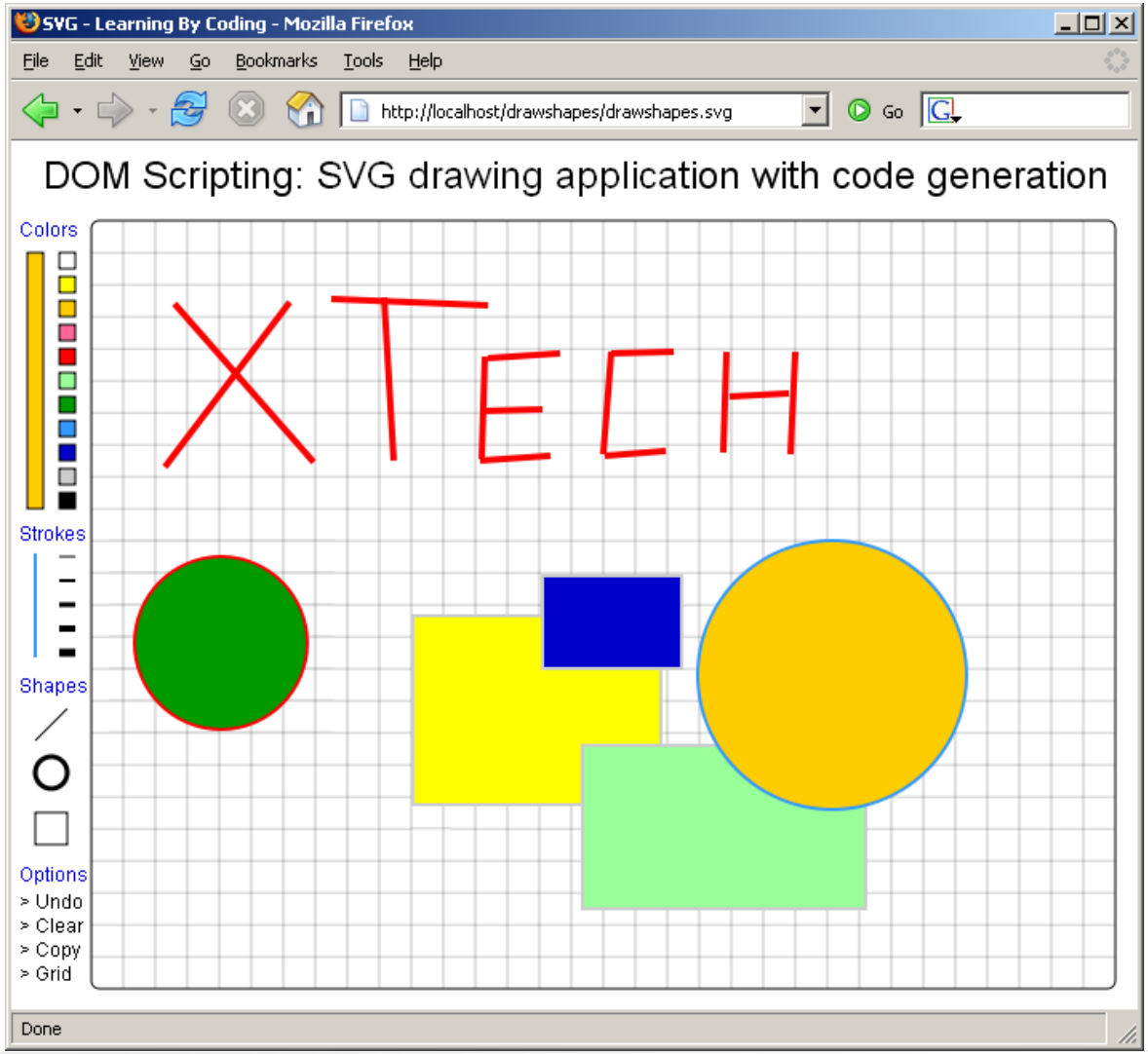
Practical demonstrations

⇨ SVG + AJAX based data connection to a web service



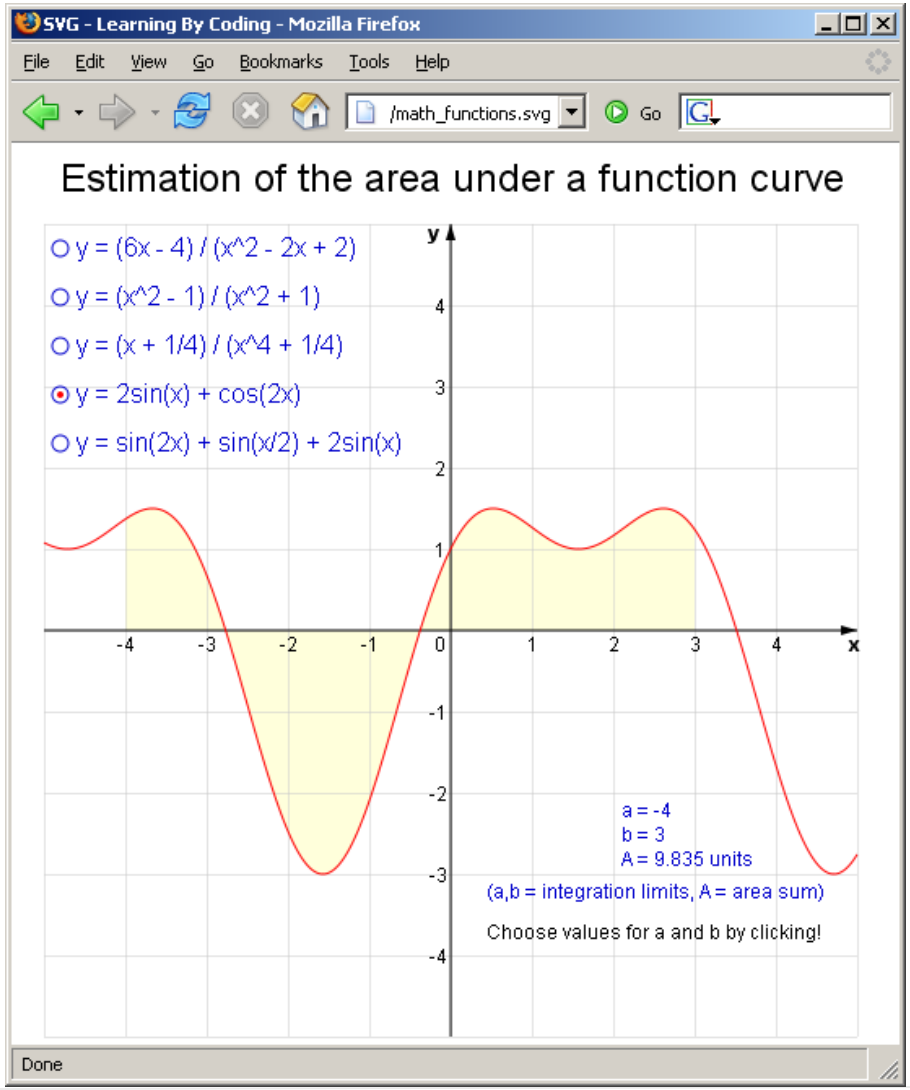
Practical demonstrations

⇨ Simple SVG drawing application with code generation



Practical demonstrations

⇨ Interactive mathematical example



References

- W3C, Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/DOM-Level-2-Core/ (2000)

- W3C, Scalable Vector Graphics (SVG) – XML Graphics for the Web, http://www.w3.org/Graphics/SVG/ (2001)

- Mozilla Developer Center: SVG in Firefox 1.5, http://developer.mozilla.org/en/docs/SVG_in_Firefox_1.5 (2005)

- Mozilla Developer Center, http://developer.mozilla.org and MozillaZine Knowledge Base, http://kb.mozillazine.org/Knowledge_Base (2006)

- T. Meinike, Fuchsschlaue Vektorgrafiken – Zur SVG-Entwicklung mit Firefox 1.5, Entwickler Magazin 2.2006, p. 132-135 (2006)

- T. Meinike, XTech 2006 conference material, http://svglbc.datenverdrahten.de/xtech06/ (2006)