Peter James Rowlett

MathML/XML series

A simple example of dynamic graphics

Peter James Rowlett
School of Computing & Informatics
Nottingham Trent University
peter.rowlett@ntu.ac.uk

Generating a triangle with SVG

In SVG a polygon can be specified using the polygon tag with the attribute points listing the coordinates of the vertices, e.g.:

```
<polygon points="50,250 200,250 200,125"/>
```

in fact, it is sensible to add some style information, so use:

```
<polygon points="50,250 200,250 200,125" style="fill:#cccccc;stroke:#000000;stroke-width:1"/>
```

Putting this in an SVG document gives:

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="230" height="300" x="0" y="0">
<polygon points="50,250 200,250 200,125" style="fill:#cccccc;stroke:#000000;stroke-width:1"/>
</svg>
```

Which can be saved as an SVG file (.svg or .xml) and loaded into an SVG capable browser (such as Firefox, or Internet Explorer with an SVG plugin).

Using PHP to dynamically define a triangle in SVG

The angle θ is determined by a pseudo-random number generator in PHP, and is set between 30 and 70.

```
$theta = rand(30,70);
```

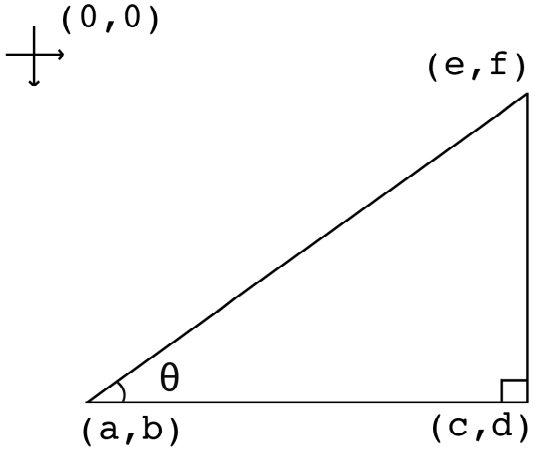

Fig 1 - Triangle with coordinates marked

The origin is at the top left of the stage and y increases downwards. The triangle is defined by coordinates (a,b), (c,d), (e,f) (see Fig 1). If we decide on a length for the hypotenuse (200, say) then we can determine the lengths of the adjacent and opposite sides from θ (200cos(θ) and 200sin(θ), respectively). If we set a=50 and b=250 to place the triangle a reasonable distance from the top left of the stage, then the remaining coordinates are determined as:

$c = a+200\cos(\theta)$

$d = b = 250$

$e = c$

$f = b-200\sin(\theta)$

Encoding this in PHP has the complication of some syntax changes and that the functions cos and sin deal in radians (of course, useful if you wish your example to be in radians). Then, the definitions of the coordinates are:

```
$a = 50;
$b = 250;
$c = $a + 200*cos(deg2rad($theta));
$d = $b;
$e = $c;
$f = $b - 200*sin(deg2rad($theta));
```

Using PHP to insert the values $a-$f into the definition of the triangle above gives:

```
echo "<polygon points=\"$a,$b $c,$d $e,$f\"
style=\"fill:#cccccc;stroke:#000000;stroke-
width:1\"/>";
```

Some notes on using PHP to generate SVG

Firstly, PHP is normally used to generate webpages. When a page is sent from a web server to a web browser it is sent with a MIME type, which allows the browser to determine how to interpret the file. By default, PHP sends the MIME type for a HTML page. Here, we want it to send the MIME type for SVG, image/svg+xml [1].

Additionally, note that the XML declaration at the top of the file starts "<?". In PHP, this is reserved for PHP code. Thus the XML declaration must be output from PHP code.

Labels

Finally, the rect tag is used to define a rectangle to mark the right angle and the text tag used to add labels to the other two angles.

The rect tag is fairly easy to use. A 10 by 10 rectangle is placed 10 up and 10 across from the vertex of the right angle. Again some style information is given. width and height define the width and the height, and x and y define the position of the upper-left point.

```
<rect width="10" height="10" x="<?php
echo $c-10; ?>" y="<?php echo $d-10; ?>"
style="fill:rgb(240,240,240);stroke-
width:1;stroke:rgb(0,0,0)"/>
```

For the labels, the text tag uses attributes x and y to define the position of a text string. The labels must move to be a suitable distance from the vertices to which they relate: too far away and they get close to other corners, introducing ambiguity; too close, and the label overlaps the edge of the triangle. PHP is used to position the labels a certain percentage of the distance along each line (determined by trial and error), in order that they are kept in a sensible position.

```
<text x="<?php echo ($a+17*($c-$a)/100);
?>" y="<?php echo ($b-5); ?>" style="font-
family:Verdana;font-size:12"><?php echo
$theta; ?></text>
<text x="<?php echo ($e-15*($e-$a)/100);
?>" y="<?php echo ($f+17*($d-$f)/100);
?>" style="font-family:Verdana;font-
size:12">a</text>
```

Final code

The full code is given below. This can be saved as a PHP file (.php) and run on a server that has PHP installed to give an SVG output viewable in an SVG-capable browser.

```
<?php
$theta = rand(30,70);

$a = 50;
$b = 250;
$c = $a + 200*cos(deg2rad($theta));
$d = $b;
$e = $c;
$f = $b - 200*sin(deg2rad($theta));

// MIME type
header("content-type: image/svg+xml");

// XML information
echo "<?xml version=\"1.0\"?".">";
?>
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
width="230" height="300" x="0" y="0">
<?php echo "<polygon points=\
"$a,$b $c,$d $e,$f\" style=\
"fill:#cccccc;stroke:#000000;stroke-
width:1\"/>"; ?>
<text x="<?php echo ($a+17*($c-$a)/100);
?>" y="<?php echo ($b-5); ?>" style="font-
```

```
family:Verdana;font-size:12"><?php echo
$theta; ?></text>
<text x="<?php echo ($e-15*($e-$a)/100);
?>" y="<?php echo ($f+17*($d-$f)/100);
?>" style="font-family:Verdana;font-
size:12">a</text>
<rect width="10" height="10" x="<?php
echo $c-10; ?>" y="<?php echo $d-10; ?>"
style="fill:rgb(240,240,240);stroke-
width:1;stroke:rgb(0,0,0)"/>
</svg>
```

The result

Putting this together and running it on a web server with PHP, we get a triangle (see Fig 2) and each time the page is reloaded the angles of that triangle change.

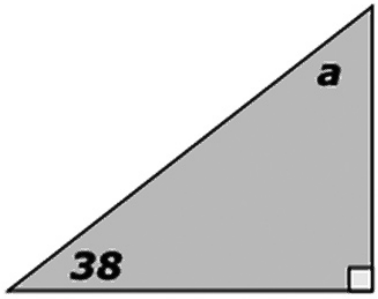

Fig 2 - A dynamically generated triangle

This could be included in a webpage with a suitable question ("If a right-angled triangle has an angle 38°, what is the value of the remaining angle (a)?", say) and answer mechanism. In this case, the question and answer mechanism is unrelated to the graphic, which is really just an illustration. Of course, a similar effect could be produced with much less effort by simply having a static image of a triangle whose angles are labelled with θ and a and telling the user that angle θ is 38°. However, it is believed that there is an advantage in ensuring that the angle θ actually looks like it is 38°. For example, if the triangle in Fig 2 was so marked, and a student was told angle θ is 70°, and their calculations told them angle a was not 20° it might be useful if the angle a looked as though it were 20°, so that an error in calculation would (should) be more obvious to the student.

Non-SVG capable browsers

SVG support in web browsers is not yet ubiquitous. The java-based Batik toolkit [2] contains a program SVG Rasterizer which can be used to convert SVG graphics into PNG, JPEG, TIFF or PDF files, which can be served as an alternative to browsers as yet without SVG support.

Further applications

Of course, a similar effect could easily be replicated with a server side scripting language other than PHP.

It should be obvious that this is a simple example, which produces a simple graphic. It is hoped the techniques used here give a useful insight into what might be possible with SVG.

References

1. Mozilla SVG Project Frequently Asked Questions [online]. Mozilla, 2006. Available at: http://www.mozilla.org/projects/svg/faq.html [Last accessed: 27 July 2006].
2. Batik SVG Rasterizer [online]. Apache, 2005. Available at: http://xmlgraphics.apache.org/batik/svgrasterizer.html [Last accessed: 27 July 2006].