UNSW High Schools Programming Competition

Information for Finalists: SVG Preparation

Introduction

Traditionally the ProgComp finals include a task requiring teams to process a image, either a bitmap using the PPM text format, or a vector image, static or animated. This year, it's back to animation.

Vector graphics

Bitmaps are just a rectangular collection of pixels. Vector images contain shapes described mathematically in terms of geometry (size, position) and attributes such as fill and stroke (or outline), width, colour and transparency. Only when rendered on a specific display do they become bitmaps.

SVG

The notation used for the task is SVG (Scalable Vector Graphics), a non-proprietary web standard supported to varying degrees by all browsers. As well as containing static images, SVG supports both declarative animation and Javascript manipulation of objects. Declarative animation specifies progressive changes to certain attributes of an object along with absolute timing information. That's the kind to be used: for the ProgComp task it requires the entire animated process to be pre-planned and written to the file object-by-object.

SVG structure

An SVG document is an XML file containing one well-formed `svg` element. Within the element is (usually) one set of definitions, often including CSS styling, and any number of drawing elements, such as lines, rectangles, ellipses, arbitrary paths, and text elements. They are initially displayed according to their explicit or implied attributes, but these can be altered over time by specifying embedded `animate` elements. Objects are drawn in the order they occur in the file.

SVG conventions

The coordinate system is Cartesian (X-Y), with the origin at the top left. X increases to the right and Y down. Default units are pixels, expressed as real or integer values.

All elements can have a unique `id` attribute for referencing from Javascript or between elements.

The accompanying document is an animation that applies insertion sort to 13 values in arbitrary order. The following sections summarise the document content.

XML Header

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
 "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
```

The header above must be provided for compliant browsers to recognise the content.

SVG element

```
<svg width="600" height="400"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
```

Although there are other ways of describing the overall size of the image, specifying width and height in pixels is sufficient.

Definitions

```
<defs>
 <style type="text/css">
 <![CDATA[
    text.C { text-anchor:middle; }
    text.R { text-anchor:end; }
    text.title { font-size:16px; font-weight:bold;
           font-family:Arial,Helvetica,sansserif;
    }
    .bar { stroke:#888888; stroke-width:0.5px; }
    .unordered { fill:#E7CACA; stroke:#B54040; }
    .sep { stroke:#444444; stroke-width:0.8px; stroke-dasharray:5,3; }
    .border { stroke: #AAAAAA; stroke-width:3px; fill:#FFFFF0; }
 ]]>
 </style>

 <filter id="bevel"> ... </filter>
</defs>
```

Protects CSS from being interpreted as XML

CSS is applied to drawing elements via the `class` attribute as with HTML elements. The `filter` element specifies a complex shading scheme to simulate a 3D bevel effect on drawing elements: details are unimportant.

Text elements

```
<text class="title C" x="300" y="30">Insertion Sort</text>
```

Text must have position (x, y attributes) and may be styled by CSS. The displayed text is within the body of the element.

Line elements

```
<line class="sep" x1="540.0" x2="540.0" y1="370.0" y2="40.0" />
```

Lines have start and end points. The element is self-closing as shown unless it requires an embedded element such as `animate`.

Rectangles

```
<rect id="bar0" class="bar unordered" filter="url(#bevel)"
    x="30.0" y="295.7"
    width="20" height="74.35" />
```

Rectangle coordinates refer to the the top left corner. Width and height must be specified. The optional `filter` attribute applies the filter identified in the definition section.

Circles and ellipses

```
<circle id="point99" class="dot" cx="90" cy="120" r="12.5" />
<ellipse class="oval" cx="90" cy="120" rx="12.5" ry="15"/>
```

Coordinates are of the centre of the figure, with one or two radii specified. Note that the ellipse major axis is parallel to one of the coordinate axes.

Paths

A `path` element describes an arbitrary set of lines (straight or curved) or polygons. You won't have to create any new paths but you should recognise path definitions (there's one in the sample file).

Groups

A group (g) element combines any number of elements so they can be processed as a unit. Unlike primitive elements that have x and y properties for positioning, groups are modified through a `transform` attribute. Transformations comprise `translate` (reposition), `scale` and `rotate`.

The `use` element

An object (usually a group) that may need to occur multiple times in the document can be defined once in the defs section and then called up in the main section with

```
<use xlink:href="#defID" />
```

where `defID` is the value of the `id` attribute on the definition. The copy can then be styled as desired. The triangle in the demo is an example.

Animation

You need to know about two types of declarative animation:

- Embedding one or more `animate` elements in a drawing element or group ,or
- Embedding the `animateTransform` element in a group.

Each `animate` element changes one attribute of the parent object over a set period of time. For the sample file, the transformations are generally moving or re-styling rectangles:

```
<rect id="bar0" ... >
    <animate attributeName="x" fill="freeze"
        begin="23.2s" dur="0.4s" to="550" />
        ...
    <animate attributeName="fill" fill="freeze"
        begin="23.6s" dur="0.4s" to="#FFFF66" />
</rect>
```

The `animate` element specifies what to change, what it changes to, when to begin the transformation and how long it should take.

`attributeName`	attribute to vary, can be geometric (x, y, width, height) or styled (opacity, fill, stroke etc).
`to`	the final value of the attribute name. If `from` is also provided, sets the value at the begin point, but that's likely to introduce a visible glitch.
`begin`	when the transformation starts, can be absolute time from when the document was loaded (as shown in these examples) or relative to the last animation applied to another element (`begin="element42.end+2s"`).
`dur`	duration of animation, usually absolute time.
`fill="freeze"`	indicates that the transformation should be retained after it is applied.

To demonstrate `animateTransform`, the title has been embedded in a group and its position and size changed. Following that the triangle is rotated.

What to do next

1. Try to understand how the sample SVG document is structured, and how each element is displayed in the browser (Chrome is recommended, Firefox is OK but doesn't realise when the animation is finished, Safari is probably OK, and of course forget IE and Edge).

2. Think about how a program to generate the SVG would look: it needs to store the results of the simulation before producing the animated elements.
3. Try to produce your own sorting demonstration, using selection sort or (if you must) bubble sort.
4. Bring your solution to the finals, it may be adaptable to the set task or it may not, but at least it will have a basic framework that you can build a new solution with. The task will require a program to read data (if appropriate) and generate a valid SVG representation of some prescribed simulated actions. You will need to demonstrate its correctness by showing the file working on your browser.

Notes and references

Official SVG Reference: http://www.w3.org/TR/SVG/

Tutorial (not much on animation): http://www.w3schools.com/svg/default.asp

Animation: http://tutorials.jenkov.com/svg/svg-animation.html

This idea can be taken to extremes: the ABC re-published AEC senate above-the-line election counts during the 2013 elections. It was in a form easily parsed, so I animated the whole fascinating preference distribution process, it's still at http://www.grwpub.info/senate/ See with your own eyes how preference deals won the Motoring Enthusiasts Party a Senate seat in Victoria with a tiny number of first-preference votes.

Geoff Whale, ProgComp Convenor