SVG 2 Proposals for Engineering Diagrams

Thomas Smailus
October 2014

These are proposals for SVG 2 based on use cases and needs in the creation and usage of technical diagrams within an engineering industrial environment, as experienced by Boeing. This set of proposals is by no means meant to be all inclusive of technical diagram or engineering use cases.

Non-scaling line patterns (dash patterns)

There needs to be an option for not scaling the dashed patterns, referred to here as an *absolute* mode. The dashed example contains 2 lines, the top one in *scaled* mode, and the bottom one in *absolute* mode. When zooming in, the scaled line's pattern also scales up (with the line thickness). When zooming out, the scaled line's pattern scales down. The absolute line's patter does not change pattern, regardless of zoom level. Note that the line width is also zoom vs. absolute in this case.

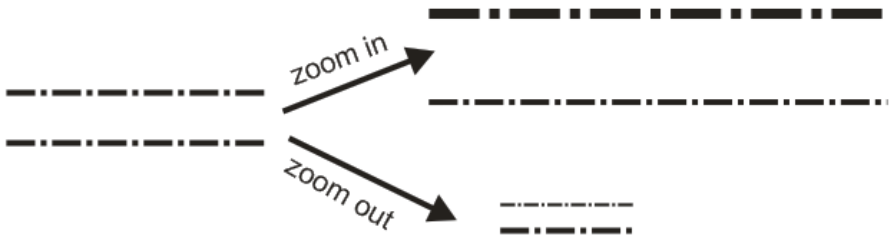


Figure 1. Effects of zooming on line width and dashed patterns

Proposed Usage:

```
<polyline stroke-width="8" stroke="rgb(0,0,0)" fill="none"
      vector-effect="non-scaling-dasharray"
      stroke-dasharray="120,40"  points="766,9807 3066,9807 "
      stroke-linejoin="miter"  stroke-miterlimit="1"  />
```

Proposal: add vector-effect option non-scaling-dasharray to vector-effect and renaming 12.6 as follows:

12.6. Non-scaling stroke and dasharray

Sometimes it is of interest to let the outline of an object keep its original width no matter which transforms are applied to it. For example, in a map with a 2px wide line representing roads it is of interest to keep the roads 2px wide even when the user zooms into the map.

For example, in technical and engineering diagrams, line patterns carry important meaning and the pattern must be recognizable at all levels of zoom. That is, the pattern should not change appearance if the zoom or scale transform on the stroked entity changes.

To achieve this, SVG Tiny 1.2 introduces the 'vector-effect' property. Future versions of the SVG language will allow for more powerful vector effects through this property but this version restricts it to being able to specify the non-scaling stroke, *limited-scaling-stroke*[1], and non -scaling-dasharray4 below behavior.

Name:	vector-effect
Value:	non-scaling-stroke | *limited-scaling-stroke*[1] | non-scaling-dasharray | none
Initial:	none

non-scaling-dasharray

Modifies the way the object is stroked. Computation of stroked region is done as if 'non-scaling-stroke' were specified. Additionally, the dasharray pattern is rendered in host coordinate space, and then transformed back to user coordinate system.

The resulting visual effect of this modification is that stroke width and stroke dash pattern are not dependent on the transformations of the element (including non-uniform scaling and shear transformations) and zoom level.

If '*non-scaling-dasharray*' is specified with no '*stoke-dasharray*' defined on the same element, the client should render as if '*non-scaling-stroke*' were specified.

[1] Definition added in another proposal

Non-Scaling fill / hatch patterns

There needs to be an option for not scaling the fill patterns, referred to here as an *absolute* mode. Two rectangles filled with a hatch pattern on the left of the figure. The upper one is in *scale* mode, the lower one in *absolute* mode. As one zooms in, the upper pattern changes with the zoom level, while the *absolute* mode pattern stays the same. As one zooms out, the same holds true, the upper *scaled* mode pattern gets smaller, while the *absolute* mode pattern stays the same.

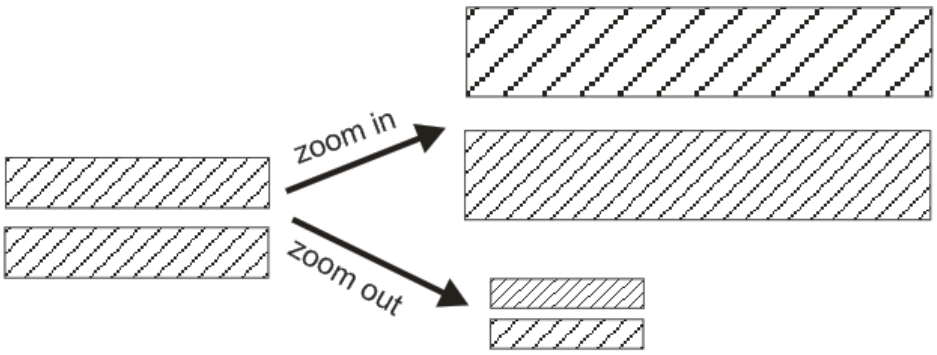


Figure 2. Effect of zooming on hatch fill patterns

Proposed usage:

```
<pattern id="HATCH1" patternUnits="userSpaceOnUse"
      pattern-effects=”non-scaling-pattern”
      x="0" y="0" width="104.05" height="104.05" >
      <line x1="0" y1="52.025" x2="104.05" y2="52.025"
            stroke-width="6.93667" stroke="rgb(0,0,0)" />
</pattern>
```

Proposal: add a pattern-effect attribute to pattern, in 14.4 of the spec, which mimics the vector-effect attribute of strokes:

Name:	pattern-effect
Value:	non-scaling-pattern | none
Initial:	none

none

Specifies that no pattern effect shall be applied, i.e. the default rendering behaviour from SVG 1.1 is used which is to first fill the geometry of a shape with a specified pattern.

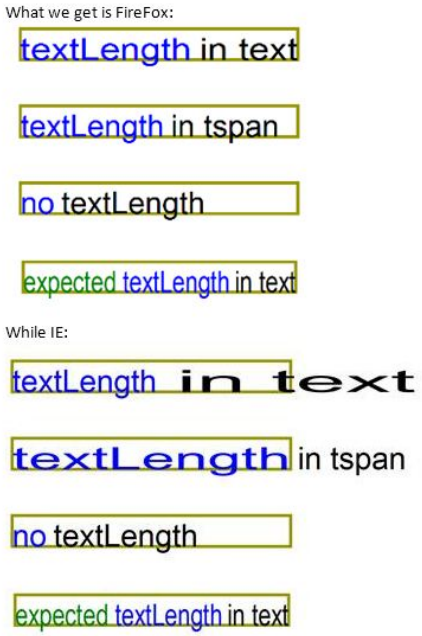
non-scaling-pattern

Modifies the way an object is filled and how that fill pattern is stroked. Normally filling involves calculating the fill outline of the shape's path in current user space and filling that shape with the fill pattern. With the non-scaling-pattern pattern effect, fill pattern and fill pattern stroke shall be calculated in the "host" coordinate space instead of user coordinate space. The pattern definition’s stroke-width specifications would be in “host”

coordinate space. More precisely: a user agent establishes a host coordinate space which in SVG Tiny 1.2 is always the same as "screen coordinate space". The fill pattern and the stroke of the pattern is calculated in the following manner: first, the shape's path is transformed into the host coordinate space. Stroke outline is calculated in the host coordinate space. The fill pattern is calculated in host coordinate space. The resulting outline and fill pattern is transformed back to the user coordinate system. (Stroke outline is always filled with stroke paint in the current user space). The resulting visual effect of this modification is that stroke width and the fill pattern are not dependent on the transformations of the element (including non-uniform scaling and shear transformations) and zoom level.

Control of text rendering width via *textLength* over text & tspan elements

At present the rendering of text with tspan children, where textLength is specified, is inconsistent, possibly due to some ambiguity in the SVG 1.1 SE spec. For example, the same diagram viewed in Firefox and Internet Explorer:



The specification of `textLength` in a text element should determine the extents of the contents of text, including all of its children tspan elements. `textLength` in a tspan child should specify the extents of that child `tspan`. Unspecified text content or tspan children should be shortened or lengthened to meet the overall `textLength` target on the parent `text` element.

```
<text textLength="100">Base <tspan
textLength="60">Child</tspan></text>
```

Here "Child" would be 60 units long, and "Base " would be 40 units long.

The question arises what to do with a no-solution text-tspan combination, where all tspan children have specified lengths that do not add up to the specified text length.

e.g. `<text textLength="100"><tspan textLength="60">long</tspan><tspan textLength="50">short</tspan></text>`

Proposal is that in this case, an error be thrown (OR, the `tspan`'s be proportionally scaled to meet the overall goal of 100 (i.e. long gets 60/110 of 100, and short gets 50/110 of 100).)

The standard language would be, with new language in orange:

Text: attribute **textLength**:

> *The author's computation of the total sum of all of the advance values that correspond to character data within this element, including the advance value on the glyph (horizontal or vertical), the effect of properties 'letter-spacing' and 'word-spacing' and adjustments due to attributes 'dx' and 'dy' on 'tspan' elements. This value is used to calibrate the user agent's own calculations with that of the author.*
>
> *The purpose of this attribute is to allow the author to achieve exact alignment, in visual rendering order after any bidirectional reordering, for the first and last rendered glyphs that correspond to this element; thus, for the last rendered character (in visual rendering order after any bidirectional reordering), any supplemental inter-character spacing beyond normal glyph advances are ignored (in most cases) when the user agent determines the appropriate amount to expand/compress the text string to fit within a length of 'textLength'.*
>
> *If attribute 'textLength' is specified on a given element and also specified on any descendants, the adjustments on all character data within this element and it's descendants are controlled by the value of 'textLength' on this element such that the 'textLength' specifications of all*

descendants are honored, as well as the 'textLength' of the given element, with any necessary adjustment needing to solve the solution to all adjustments on this element and its descendants being uniformly applied to this or decedent elements for which no 'textLength' is specified. The possible side effect being that the adjustment ratio for elements without textLength specification would likely be different than for the elements with textLength specification.

A negative value is an error (see Error processing).

*A combination of a given element specified value with a fully specified value on **all** descendants such that the sum of the descendants values does not equal this value, is an error (See Error processing).*

If the attribute is not specified anywhere within a 'text' element, the effect is as if the author's computation exactly matched the value calculated by the user agent; thus, no advance adjustments are made.

Value	*<length>*
lacuna value	*See above.*
Animatable	*yes*

And for tspan: attribute **textLength**: no change necessary

The author's computation of the total sum of all of the advance values that correspond to character data within this element, including the advance value on the glyph (horizontal or vertical), the effect of properties 'letter-spacing' and 'word-spacing' and adjustments due to attributes 'dx' and 'dy' on this 'tspan' element or any descendants. This value is used to calibrate the user agent's own calculations with that of the author.

The purpose of this attribute is to allow the author to achieve exact alignment, in visual rendering order after any bidirectional reordering, for the first and last rendered glyphs that correspond to this element; thus, for the last rendered character (in visual rendering order after any bidirectional reordering), any supplemental inter-character spacing beyond normal glyph advances are ignored (in most cases) when the user agent determines the appropriate amount to expand/compress the text string to fit within a length of 'textLength'.

If attribute 'textLength' is specified on a given element and also specified on an ancestor, the adjustments on all character data within this element are controlled by the value of 'textLength' on this element exclusively, with the possible side-effect that the adjustment ratio for the contents of this element might be different than the adjustment ratio used for other content that shares the same ancestor. The user agent must assume that the total advance values for the other content within that ancestor is the difference between the advance value on that ancestor and the advance value for this element.

A negative value is an error (see Error processing).

If the attribute is not specified anywhere within a 'text' element, the effect is as if the author's computation exactly matched the value calculated by the user agent; thus, no advance adjustments are made.

Value	*<length>*
lacuna value	*See above.*
Animatable	*yes*

The SVG 2 test for textLength will be modified to test for more complex variations.

Control of text rendering height

The existing implementation in SVG 1.1 of the use of *font-size* is sufficient to almost satisfy the CGM 1 and CGM 3 'basic' method of restricted text which only require the resulting text fill but not exceed the bounding box (decenders go below the box, as in Boxed-cap). Industrial users of SVG render diagrams in CAD systems, export them into CGM format typically (as this is an aerospace interchange format under S1000D) and then convert those CGM diagrams to SVG as end-state documents, to be consumed by the end-user. Being able to precisely control the sizing of text in important in engineering diagrams in order for the SVG diagram to convey the meaning intended by the diagram author upstream.

If controlling restricted text precisely is critical to the end user, then SVG will need a more precise way of constraining the height of text than *font-size* by itself. A method that can at least implement the "Boxed-Cap" and "Boxed-all" CGM text specifications is proposed.

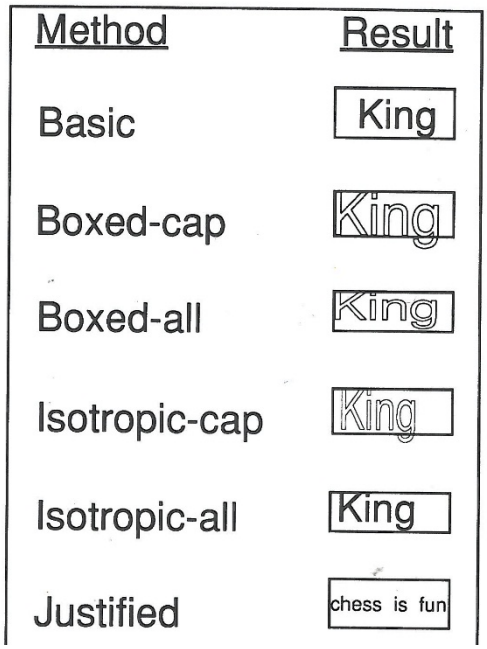


Figure 3. CGM Restricted Text Type definitions and their effects[2]

To exactly specify the stroked height of the glyphs from base to cap, specifying a font-size is an approximation because font-size includes cap to top and base to bottom.

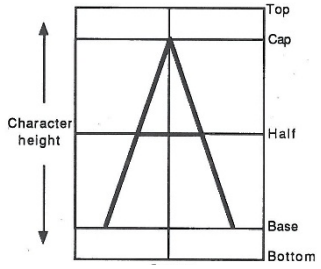


Figure 4. CGM terminology with respect to text glyphs[2]

textHeight with textLength in conjunction with length-Adjust and height-Adjust would provide a mechanism for implementing CGM Restricted Text (ISO/IEC 8632-1:1999(E) 7.6.5) in CGM 3 files with restriction methods (Restricted Text Type). The types CGM 3 supports are shown in the accompanying figure[2].

Proposal: adopt the textHeight and heightAdjust attributes and their definitions given below.

textHeight

> The author's computation of the total height value of the tallest glyph within the text element or its descendants. This value is used to calibrate the user agent's own calculations with that of the author.
>
> The purpose of this attribute is to allow the author to achieve exact alignment and sizing in the glyph height dimension.
>
> A negative value is an error (see Error processing).

[2] Figures from Henderson, Mumford, "The CGM Handbook," Academic Press, 1993

If the attribute is not specified anywhere within a ‘text’ element, the effect is as if the author's computation exactly matched the value calculated by the user agent; thus, no adjustments are made.

Value	*<length>*
lacuna value	*Current ‘font-size’.*
Animatable	*yes*

heightAdjust

Indicates the type of adjustments which the user agent shall make to make the rendered length of the text match the value specified on the ‘textLength’ attribute.

Value	none|cap|all
lacuna value	none
Animatable	*yes*

none

Indicates no changes to the effective font-size are made.

cap

Indicates that distance from the baseline to the cap line is specified by the textHeight value, overriding any current font-size. Decenders will extend below the textHeight extents, and diacritics can appear below the base line and above the cap line.

All

Indicates that the distance from the bottom line to the cap line is specified by textHeight, all glyph rendering is limited to within the textHeight value (diacritics as well?)

The user agent is required to achieve correct start and end positions for the text strings, but the locations of intermediate glyphs are not predictable because user agents might employ advanced algorithms to stretch or compress text strings in order to balance correct start and end positioning with optimal typography.

Minimal rendered stroke width

In very large diagrams, where the extents of the full diagram are orders of magnitude more than the size or stroke of the smallest elements in the diagram, when this diagram is viewed fully, the fine lines and elements can become impossible to see. A possible solution from SVG Tiny is the use of `vector-effect="non-scaling-stroke"`

Example Diagram, if rendered without non-scaling-stroke setting, and text rendered only with the textLength attribute set: http://jsfiddle.net/thomassmailus/0dcs7dng/

Same Diagram with non-scaling-stroke and text resized to fill a specified region (text length and height): http://jsfiddle.net/thomassmailus/84mfxkea/

However, when the user zooms in on a particular region, the stroke thickness maters, relative to elements around it. Thus the needed solution is something that will not render the stroke width less than a single pixel.

e.g. `stroke-width="27" vector-effect="limited-scaling-stroke"`

The stroke width is computed in screen (client?) coordinates and if is below 1, a value of 1 is used.

> **Proposal**: add a vector-effect option "*limited-scaling-stroke*" to section 12.6 so it is changed as follows:
>
> **12.6. Non-scaling stroke *and dasharray*[3]**
>
> Sometimes it is of interest to let the outline of an object keep its original width no matter which transforms are applied to it. For example, in a map with a 2px wide line representing roads it is of interest to keep the roads 2px wide even when the user zooms into the map.

[3] Added in another proposal

In technical and engineering diagrams, line patterns carry important meaning and the pattern must be recognizable at all levels of zoom. That is, the pattern should not change appearance if the zoom or scale transform on the stroked entity changes.[4]

To achieve this, SVG Tiny 1.2 introduces the 'vector-effect' property. Future versions of the SVG language will allow for more powerful vector effects through this property but this version restricts it to being able to specify the non-scaling stroke *and limited-scaling-dasharray*[4] below behavior.

Name:	vector-effect
Value:	non-scaling-stroke | limited-scaling-stroke | *non-scaling-dasharray*[4] |none
Initial:	none

limited-scaling-stroke

Modifies the way the object is stroked. Computation of stroked region is done as usual (as if *none* were specified), and if that computation results in a stroke in screen/host coordinate space stroke width of less than 1, a stroke-width of 1 is used in screen/host coordinate space. The screen/host stroke width is lower bound by 1.

[4] Definition added in another proposal

Browser/Viewer Pan/Zoom/Rotate/Print

The viewer pan/zoom/rotate ability seemed to disappear – no longer supported.

Currently pan/zoom/rotate is implemented via embedded script modifying the scale/translate/rotate transformations at the top level. Should this be the norm? If so, drop `zoomAndPan` attribute on SVG?

Current SVG 2 draft indicates pan/zoom will be implemented. Pan and zoom has substantial value to large and detailed technical diagrams.

> **Proposal**: add 'rotate' as a supported transform, allowing the user to rotate the graphic through 90 degree orthogonal rotations in clockwise or counterclockwise direction.

Print – there is a need to generate a 'print of the current view' as well as the 'print of viewBox/entire diagram', in technical diagrams were the user would pan/zoom/rotate the diagram to obtain a specific visible rendering.

> **Proposal**: add 'print current view' as a supported feature.

Multiple picture support

A single SVG graphics contains multiple ‘picture’ elements, of which only one, or several, or all, are visible at once. Similar to the content of each ‘picture’ being contained within a ‘group’, but these g-type elements are a special element that allows viewers to control their display (all at one, or paged, or selective activation).

Seeking input on support and usefulness in other domains.