Chapter 9 : Interactivity and Animation

"SVG content can be interactive (i.e., responsive to user-initiated events) by utilizing the following features in the SVG language:

- User-initiated actions such as button presses on the pointing device (e.g., a mouse) can cause animations or scripts to execute.
- The user can initiate hyperlinks to new Web pages (see Links out of SVG content: the 'a' element) by actions such as mouse clicks when the pointing device is positioned over particular graphics elements.
- In many cases, depending on the value of the zoomAndPan attribute on the 'svg' element and on the characteristics of the user agent, users are able to zoom into and pan around SVG content.
- User movements of the pointing device can cause changes to the cursor that shows the current position of the pointing device."

Extract of W3C specifications

Linking

Linking out of the svg content

SVG provides an 'a' element, analogous to HTML's 'a' element, to indicate links.
The destination for the link is defined by a URI specified by the xlink:href attribute on the 'a' element. The remote resource may be any Web resource (e.g., an image, a video clip, a sound bite, a program, another SVG document, an HTML document, an element within the current document, an element within a different document, etc.). By activating these links (by clicking with the mouse, through keyboard input, voice commands, etc.), users may visit these resources.
This is the syntax for 'a' element :

```
<a      id="name"
        xlink:href = '<uri>'
        xlink:type = 'simple'
        xlink:role = '<uri>'
        xlink:arcrole = '<uri>'
        xlink:title = '<string>'
        xlink:show = 'new | replace'
        xlink:actuate = 'onRequest'
        target = "<frame-target>" >
    <!-- svg objects to activate link -->
</a>
```

Attributes for 'a' are

Example of code for 'a' element :

```
<a xlink:href="MyFile.svg">
  <rect x="0" y="0" width="150" height="150" style="fill:green"/>
</a>
```

When pointer is on rectangle, cursor become a hand and if user click, MyFile.svg is open in same window.

Linking into svg content

We can link into any element using 'id' of element, but because SVG content often represents a picture or drawing of something, a common need is to link into a particular view of the document.

We can define in svg, a view element, give id and call it.

This is the syntax for view element

<**view** id="name"
viewBox="ViewBoxSpec"
preserveAspectRatio="PreserveAspectRatioSpec"
zoomAndPan="disable|magnify"
viewTarget = "XML_Name [XML_NAME] "/ >

To link into defined 'view' element :

```
<defs>
  <view id="MyView" viewBox="150 0 200 200"/>
</defs>
```

We can use

```
<a xlink:href="#MyView">
  <!-- target element  -->
</a>
```

or

```
<a xlink:href="#xpointer(id('MyView'))">
  <!-- target element  -->
</a>
```

We can also link into new view with

```
<a xlink:href="#svgView(viewBox(0,200,1000,1000))">
  <!-- target element  -->
</a>
```

Figure 9-1 show zoom on object by clicking it, without script :

Source code of this example (Example 9-1) :

```
<svg width="340" height="320" viewBox="-20 -20 340 320"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <defs>
    <linearGradient id="MyGradient1">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient2">
      <stop offset="20%" stop-color="red"/>
      <stop offset="80%" stop-color="yellow"/>
    </linearGradient>
    <view id="MyView" viewBox="140 0 170 150"/>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <a xlink:href="#MyView">
    <rect x="150" y="0" width="150" height="150"
          style="fill:url(#MyGradient2)"/>
  </a>
  <text x="75" y="175" style="text-anchor:middle">Offset 0 and 100</text>
  <text id="MyText" x="225" y="175" style="text-anchor:middle">
     Offset 20 and 80
  </text>
  <g style="stroke-dasharray:2 2;stroke:black">
    <path d="M180 0l0 150"/>
    <path d="M270 0l0 150"/>
    <path d="M0 0l0 150"/>
    <path d="M150 0l0 150"/>
  </g>
</svg>
```

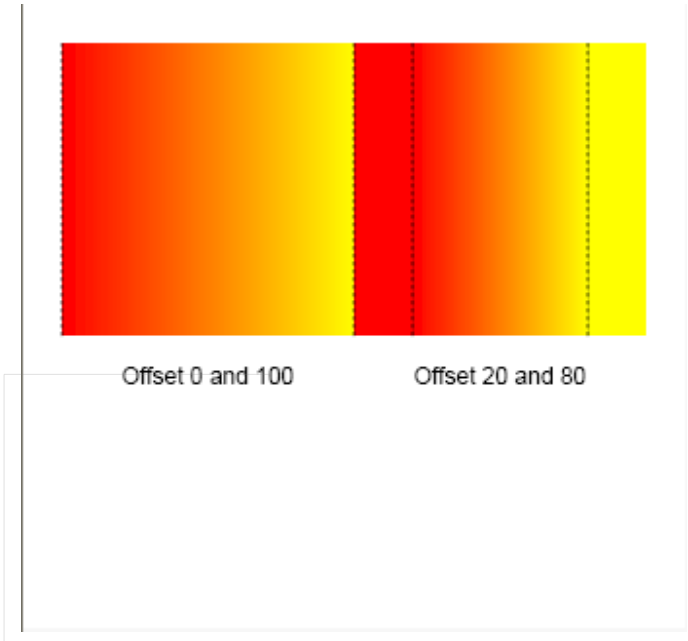

Figure 9-1. Zoom on object using view element in link

Supported events

Events can be used in script or in attributes as 'begin' or 'end' for animation elements.

First DOM2 category : UIEvent

Event name	Description	DOM2 name	Event attribute name
focusin	Occurs when an element receives focus, such as when a 'text' becomes selected.	DOMFocusIn	onfocusin
focusout	Occurs when an element loses focus, such as when a 'text' becomes unselected.	DOMFocusOut	onfocusout
activate	Occurs when an element is activated, for instance, thru a mouse click or a keypress. A numerical argument is provided to give an indication of the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter).	DOMActivate	onactivate

Second DOM2 category : MouseEvent

Event name	Description	DOM2 name	Event attribute name
click	Occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: mousedown, mouseup, click. If multiple clicks occur at the same screen location, the sequence repeats with the detail attribute incrementing with each repetition.	(same)	onclick
mousedown	Occurs when the pointing device button is pressed over an element.	(same)	onmousedown
mouseup	Occurs when the pointing device button is released over an element.	(same)	onmouseup
mouseover	Occurs when the pointing device is moved onto an element.	(same)	onmouseover
mousemove	Occurs when the pointing device is moved while it is over an element.	(same)	onmousemove
mouseout	Occurs when the pointing device is moved away from an element.	(same)	onmouseout

An event set designed for use with **keyboard** input devices will be included in a later version of the DOM and SVG specifications.

How use this events ?

We can start animation element on event, we use **event name** in begin attribute :
By example, animation will start by click on object with 'go' as id :

```
<animate begin="go.click"   ..................  />
<!--                            -->
<rect id="go" ........................... />
```

We can also start script on event, we use in this case **event attribute name** :

```
<rect onclick="MyFunction(evt)" ..................  />
```

Third DOM2 category : MutationEvent

Event name	Description	DOM2 name	Event attribute name
DOMSubtree Modified	This is a general event for notification of all changes to the document. It can be used instead of the more specific events listed below. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOMNode Inserted	Fired when a node has been added as a child of another node. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOMNode Removed	Fired when a node is being removed from another node. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOMNode RemovedFrom Document	Fired when a node is being removed from a document, either through direct removal of the Node or removal of a subtree in which it is contained. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOMNode InsertedInto Document	Fired when a node is being inserted into a document, either through direct insertion of the Node or insertion of a subtree in which it is contained. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOMAttr Modified	Fired after an attribute has been modified on a node. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none
DOM CharacterData Modified	Fired after CharacterData within a node has been modified but the node itself has not been inserted or deleted. (The normative definition of this event is the description in the DOM2 specification.)	(same)	none

Events on document

Event name	Description	DOM2 name	Event attribute name
SVGLoad	The event is triggered at the point at which the user agent has fully parsed the element and its descendants and is ready to act appropriately upon that element, such as being ready to render the element to the target device. Referenced external resources that are required must be loaded, parsed and ready to render before the event is triggered. Optional external resources are not required to be ready for the event to be triggered.	(same)	onload
SVGUnload	Only applicable to outermost 'svg' elements. The unload event occurs when the DOM implementation removes a document from a window or frame.	(same)	onunload
SVGAbort	The abort event occurs when page loading is stopped before an element has been allowed to load completely.	(same)	onabort
SVGError	The error event occurs when an element does not load properly or when an error occurs during script execution.	(same)	onerror
SVGResize	The resize event occurs when a document view is resized. (Only applicable to outermost 'svg' elements.)	(same)	onresize
SVGScroll	The scroll event occurs when a document view is shifted in X or Y or both, such as when the document view is scrolled or panned. (Only applicable to outermost 'svg' elements.)	(same)	onscroll
SVGZoom	Occurs when the document changes its zoom level based on user interaction. (Only applicable to outermost 'svg' elements.)	none	onzoom

Most used is SVGLoad, we can call function init(evt) on loading svg :

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

We use SVGResize, SVGScroll and SVGZoom when we need coordinates of pointer in viewport coordinate system for scripting.

With this code :

```
<svg width="100%" height="100%" viewBox="0 0 600 400"
     preserveAspectRatio="xMinYMin meet" onresize="coordinates(evt)" >
```

when user resize window, svg is redraw in window, we know coordinates of pointer in window with getClientX and getClientY but not in viewport.
We can use this code :

```
ratio=Math.min(window.innerHeight/400,window.innerWidth/600);
xm=evt.getClientX()/ratio;
ym=evt.getClientY()/ratio;
```

to get coordinates of pointer in viewport.

If origin in viewBox is not 0,0 we add coordinates of origin.
If user zoom or pan, we can use currentScale, currentTranslate.x and currentTranslate.y to get coordinates of pointer.

Figure 9-2 is screenshot of svg to test formula.
In this case, SVG is embedded in HTML with values for width and height.
We can use :

```
ratio=Math.min(width_svg/width_viewBox,height_svg/height_viewBox);
xm=x0_viewBox+
   (evt.getClientX()-svgdoc.currentTranslate.x)/ratio/svgdoc.currentScale;
ym=y0_viewBox+
   (evt.getClientY()-svgdoc.currentTranslate.y)/ratio/svgdoc.currentScale;
```

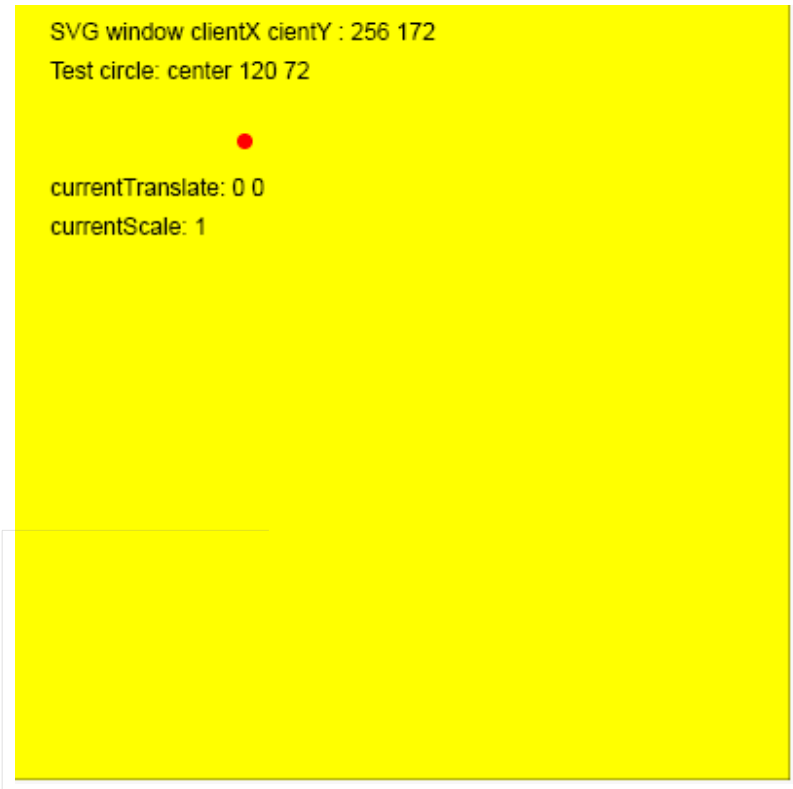

Figure 9-2. To get coordinates of pointer in viewport

Events about animations

Event name	Description	DOM2 name	Event attribute name
beginEvent	Occurs when an animation element begins. For details, see the description of Interface TimeEvent in the SMIL Animation specification.	none	onbegin
endEvent	Occurs when an animation element ends. For details, see the description of Interface TimeEvent in the SMIL Animation specification.	none	onend
repeatEvent	Occurs when an animation element repeats. It is raised each time the element repeats, after the first iteration. For details, see the description of Interface TimeEvent in the SMIL Animation specification.	none	onrepeat

Pointer-events property

The 'pointer-events' property specifies under what circumstances a given graphics element can be the target element for a pointer event.

'pointer-events'

Value:	visiblePainted | visibleFill | visibleStroke | visible |painted | fill | stroke | all | none | inherit
Initial:	visiblePainted
Applies to:	graphics elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Animatable:	yes

The given element can be the target element for pointer events

visiblePainted
when the visibility property is set to visible and when the pointer is over a "painted" area.

visibleFill
when the visibility property is set to visible and when the pointer is over the interior (i.e., fill) of the element.

visibleStroke
when the visibility property is set to visible and when the pointer is over the perimeter (i.e., stroke) of the element.

visible

when the visibility property is set to visible and the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element.

painted
when the pointer is over a "painted" area. The value of the visibility property does not effect event processing.

fill
when the pointer is over the interior (i.e., fill) of the element.

stroke
when the pointer is over the perimeter (i.e., stroke) of the element.

all
whenever the pointer is over either the interior (i.e., fill) or the perimeter (i.e., stroke) of the element.

none
The given element does not receive pointer events.

For text elements, hit detection is performed on a character cell basis:
The values visibleFill, visibleStroke and visible are equivalent
The values fill, stroke and all are equivalent

For raster images, hit detection is either performed on a whole-image basis (i.e., the rectangular area for the image is one of the determinants for whether the image receives the event) or on a per-pixel basic (i.e., the alpha values for pixels under the pointer help determine whether the image receives the event):
The values visibleFill, visibleStroke and visible are equivalent.
The values fill, stroke and all are equivalent.

Cursor property and element

Cursor Property

'cursor'

Value:	[[<uri> ,]* [auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help]] | inherit
Initial:	auto
Applies to:	container elements and graphics elements
Inherited:	yes
Percentages:	N/A
Media:	visual, interactive
Animatable:	yes

This property specifies the type of cursor to be displayed for the pointing device. Values have the following meanings:
auto The UA determines the cursor to display based on the current context.
crosshair A simple crosshair (e.g., short line segments resembling a "+" sign).

default The platform-dependent default cursor. Often rendered as an arrow.
pointer The cursor is a pointer that indicates a link.
move Indicates something is to be moved.
e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize
Indicate that some edge is to be moved. For example, the 'se-resize' cursor is used when the movement starts from the south-east corner of the box.
text Indicates text that can be selected. Often rendered as an I-bar.
wait Indicates that the program is busy. Often rendered as a watch or hourglass.
help Help is available for the object under the cursor. Often rendered as a question mark or a balloon.
<uri> The user agent retrieves the cursor from the resource designated by the URI. If the user agent cannot handle the first cursor of a list of cursors, it shall attempt to handle the second, etc. If the user agent cannot handle any user-defined cursor, it must use the generic cursor at the end of the list.

Cursor element

The 'cursor' element can be used to define a platform-independent custom cursor. A recommended approach for defining a platform-independent custom cursor is to create a PNG image and define a 'cursor' element that references the PNG image and identifies the exact position within the image which is the pointer position (i.e., the hot spot).

Syntax for 'cursor' element :

```
<cursor     id = "name"
            x = "number"
            y = "number"
            xlink:href = "URI" />
```

Attributes are
x : The x-coordinate of the position in the cursor's coordinate system which represents the precise position that is being pointed to.
y : The y-coordinate of the position in the cursor's coordinate system which represents the precise position that is being pointed to.
xlink:href : URI reference to the file or element which provides the image of the cursor

Animation

Common attributes

Target element for animation
We use '**xlink:href**' attribute as URI reference to the target of the animation.

If the xlink:href attribute is not provided, then the target element will be the immediate parent element of the current animation element.

Target attribute or property
We use **'attributeName'** and **'attributeType'** attributes to specify the name of the target attribute to be modified in animation.

Values for attributes :

attributeName="AttributeName"
attributeType="CSS|XML|auto"
CSS for a CSS property
XML for XML attribute
auto : The implementation must first search through the list of CSS properties for a matching property name, and if none is found, search the default XML namespace for the element.

Controling the timing

This attributes are common to all animation elements :

To define when animation begin :

begin= 'begin-value-list'
begin-value='offset-value | syncbase-value | event-value | repeat-value | accessKey-value | wallclock-sync-value | "indefinite'"

offset-value : for SVG, the implicit syncbase begin is defined to be relative to the document begin.

With begin="10" animation start 10 seconds after loading SVG.

syncbase-value : relative to begin or end of another animation

With begin="MyAnim.**end**+5" animation start 5 seconds after end of MyAnim animation.
event-value : begin on event

With begin="MyRect.**click**+3" animation start 3 seconds after click on MyRect object

repeat-value : begin after the repeat event is raised

With begin="MyAnim.**repeat**(2)" animation start when MyAnim is repeated 2 times.

accessKey-value : begin after key pressed

With begin="**accessKey**(g)" animation start when user press 'g' key

wall-clock-sync-value : begin at a real-world clock time

"indefinite" : begin is determined by beginElement() method call or hyperlink targeted to the element.

To define duration of animation

Specifies the simple duration

dur = 'Clock-value|"media"|"indefinite"'

Clock-value : specifies the duration in seconds
"media" is ignored for SVG animation elements
"indefinite" can be usefull for 'set' element

To define end of animation

Defines an end value for the animation that can constrain the active duration.

end = 'end-value-list'
end-value='offset-value | syncbase-value | event-value | repeat-value | accessKey-value | wallclock-sync-value | "indefinite"'

See begin-value for this values.
With end="indefinite" end is determined by endElement() method call

To define minimum and maximum values for duration of animation

min = 'Clock-value|"media"'
Value by default is 0
max = 'Clock-value|"media"'

To define when animation can restart

restart = '"always"|"whenNotActive"|"never"'

"always" : animation can restart at any time. It's default value.
"whenNotActive" : animation can restart when it is not active (after end)
"never" : animation cannot be restarted for the remainder of the document duration.

To define how repeat animation

We determine iterations of animation by number or duration :

repeatCount = 'Integer|"indefinite"'

repeatDur = 'Clock-value|"indefinite"'

With "indefinite", animation repeat indefinitely (until th document ends)

To define effect when animation stop

fill = '"freeze"|"remove"'

With fill="**freeze**", the animation effect is frozen for the document duration or until animation restart.

With fill="**remove**", the animation no longer affects the target. It is the default value.

Animation values over time

This attributes have no effect for **'set'** element except 'to' attribute.

To define interpolation mode for the animation

calcMode = "**discrete | linear | paced | spline**"

discrete : Animation function will jump from one value to the next without any interpolation.
linear : Simple linear interpolation between values is used to calculate the animation function. Except for 'animateMotion', this is the default value.
paced : Defines interpolation to produce an even pace of change across the animation. This is only supported for values that define a linear numeric range, and for which some notion of "distance" between points can be calculated (e.g. position, width, height, etc.). keyTimes or keySplines will be ignored. For 'animateMotion', this is the default value.
spline : Interpolates from one value in the values list to the next according to a time function defined by a cubic Bézier spline. The points of the spline are defined in the keyTimes attribute, and the control points for each interval are defined in the keySplines attribute.

To define values to use

We can use :

values = "list of values"
By example for 'viewBox' attribute,
we can write values="0,0,200,200;0,0,100,100;0,0,200,200" and in animation, viewBox will go from "0 0 200 200" to "0 0 100 100" and return to "0 0 200 200".
For filling color we can write values="red,yellow,green" and filling color will go from red to yellow and then to green.

from = "value"

We give value to start animation. In our example for viewBox, we can write from="0,0,200,200"

to = "value"
We give value to end animation.

by = "value"
We give relative offset value for animation.

To define pacing for animation

KeyTimes

keyTimes = "list of values between 0 and 1"
There must be exactly as many values in keyTimes list as in values list.
Each successive time value must be greater than or equal to the preceding time value.
Each value represents a proportional offset into the simple duration of the animation element.
With calcMode = 'paced', this attribute is ignored.

With this example :
keyTimes="0;0.195;0.405;0.7;1" values="0;400;47.5;450;500", if we have a duration of 10 seconds (dur="10"), and that values are for width attribute of a rectangle, we get this animation :
From 0 to 1.95s, width of rectangle increase from 0 to 400
from 1.95s to 4.05s width of rectangle decrease from 400 to 47.5
from 4.05s to 7s width of rectangle increase from 47.5 to 450
and
from 7s to 10s (end of animation) width increase from 450 to 500.
If we get no values for keySplines, speed of animation is the same on each part.

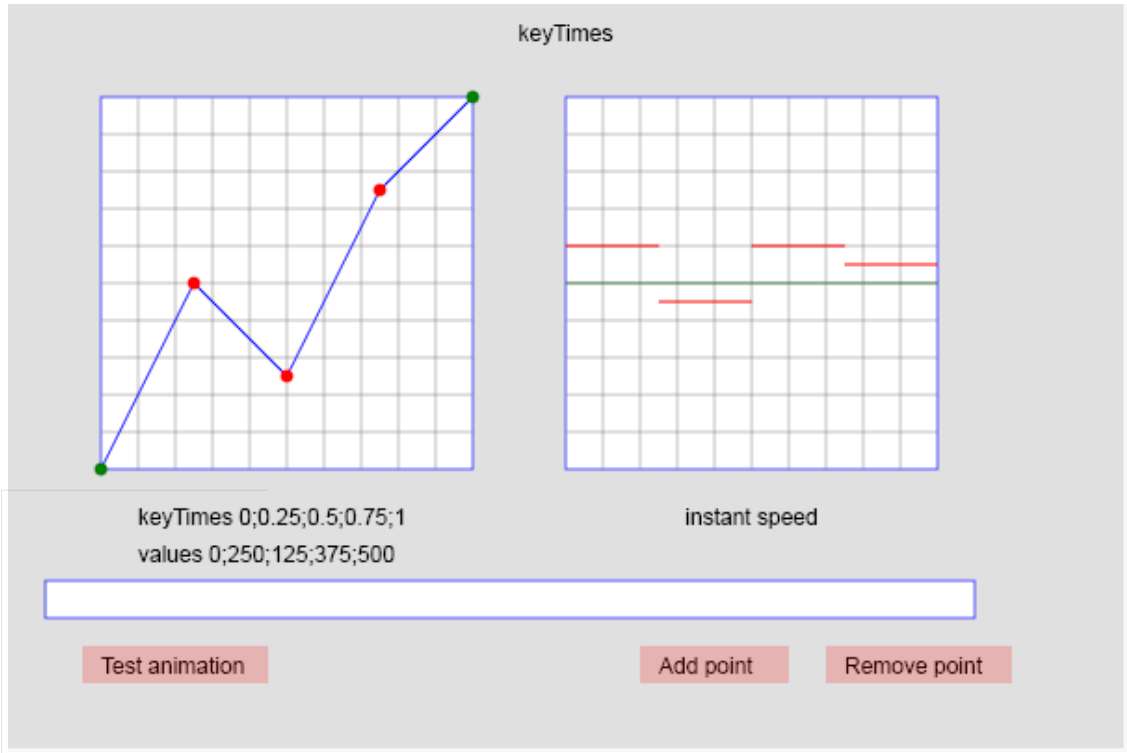

Figure 9-3. KeyTimes and values

KeySplines

keySplines = "list of list of 4 numbers between 0 and 1"
A set of Bézier control points associated with the keyTimes list, defining a cubic Bézier function that controls interval pacing.

If there is no value for keyTimes, we can use a cubic Bezier function that controls pacing for the whole animation.
If calcMode is not "spline", this attribute is ignored.

With our previous example :
keyTimes="0;0.195;0.405;0.7;1" values="0;400;47.5;450;500"
we add keySplines="0,0.5,0.5,1;0.5,0,1,0.5;0,0.5,0.5,1;0,0.5,0.5,1" and calcMode="spline"
we get same four steps in animation, but we have speed which increase on steps 1 3 4 and decrease on step 2

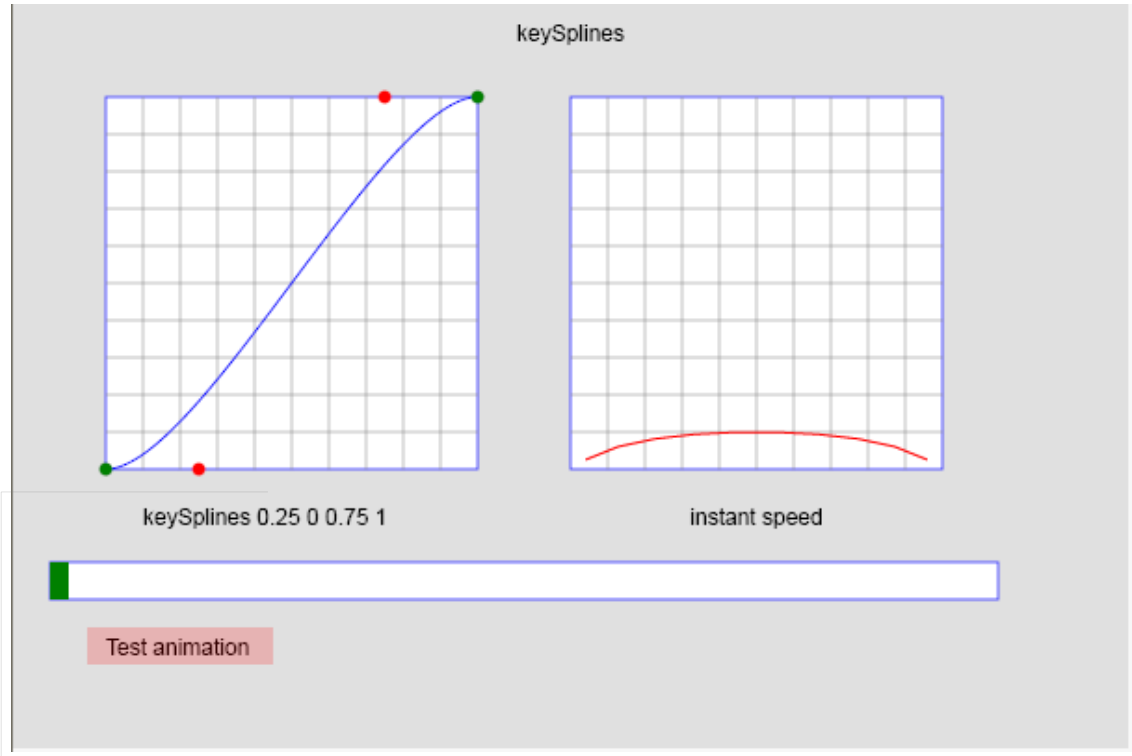

Figure 9-4. KeySplines and speed

Figures 9-4 and 9-5 show how is speed with two examples of values for keySplines

We can use **keyPoints** for **animateMotion** (see below)

To control if animations effects are additive

Values given for attribute can be added to existing value or replace it :

additive = "replace|sum"
replace : animation values replace existing value. It's default value.
sum : animation values added to existing value

When we repeat animation, effects can be cumulative or not :

accumulate = "none|sum"
sum : for each iteration of animation (with repeat) we start with new value of attribute.
none : effects are not cumulative. It's default value.

Data types used with 'additive' and 'accumulate' can be :
<angle>, <color>, <coordinate>, <integer>, <length>, <number>, <paint>, <percentage>, <uri> and <transform-list>.

Set element

The 'set' element provides a simple means of just setting the value of an attribute for a specified duration or until event.

This is the syntax of this element :

```
<set        id = "name"
            xlink:href = "URI"
            attributeName = "AttributeName"
            attributeType = "CSS|XML|auto"
            begin = 'begin-value-list'
            end = 'end-value-list'
            dur = 'Clock-value|"media"|"indefinite"'
            min = 'Clock-value|"media"'
            max = 'Clock-value|"media"'
            restart = '"always"|"whenNotActive"|"never"'
            repeatCount = 'Integer|"indefinite"'
            repeatDur = 'Clock-value|"indefinite"'
            fill = '"freeze"|"remove"'
            to = "value" />
```

All this attributes are explained above.

For 'to' attribute, data types can be :
<angle>, <color>, <coordinate>, <integer>, <length>, <list of xxx>, <number>, <paint>, <percentage>, <uri> or all other data types used in animatable attributes and properties.

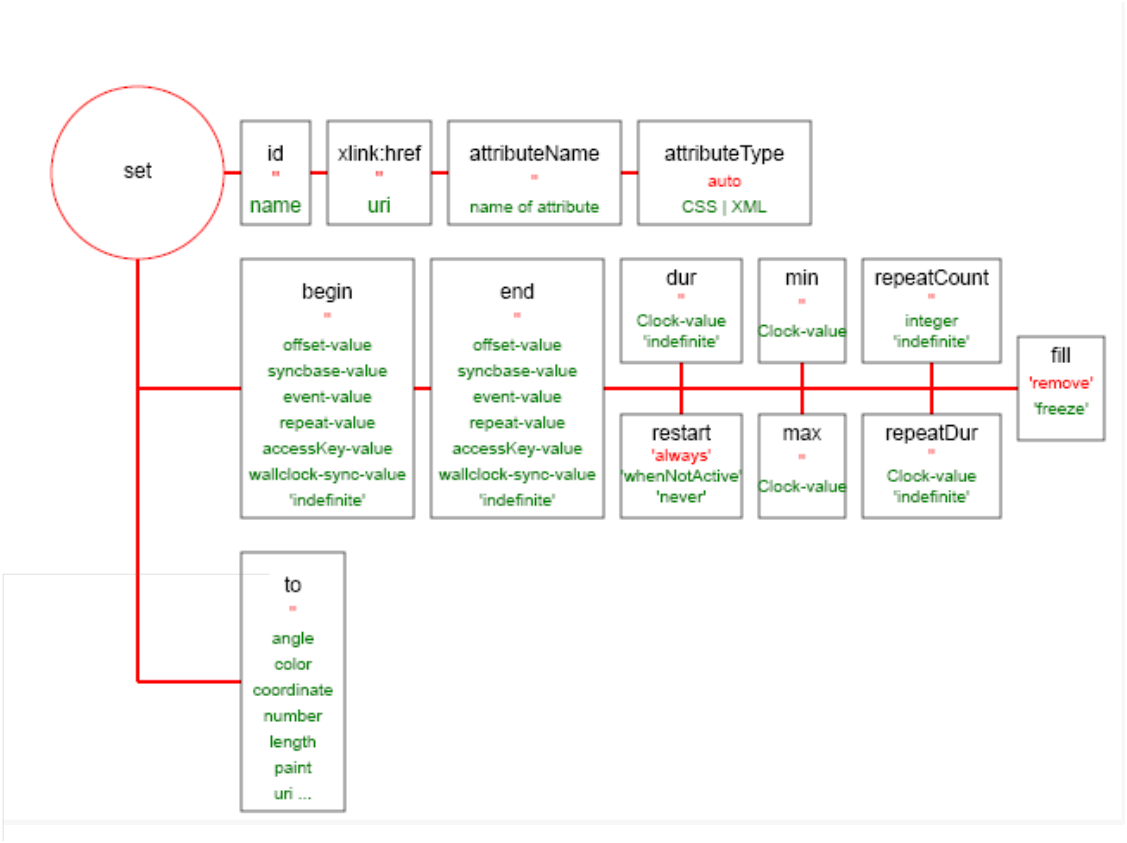

Diagram 9-1. Chart for 'set' syntax

To show utility of this element, we create a drop down list to select a color to fill a rectangle or other element.

When user click on ∇ , list of colors is writed, when pointer is on color, there is a gray rectangle on it to show that this color is selected. If user click on it, color is choosed, list of colors disappear. To cancel choice, user must click outside the list.

We use no script, only 'set' elements (Example 9-6) :

```
<svg width="400" height="400">
  <!-- rectangle to apply choice of color -->
  <rect x="40" y="100" width="320" height="80" fill="white">
    <set attributeName="fill" fill="freeze" to="red" begin="red.click"/>
    <set attributeName="fill" fill="freeze" to="yellow"
         begin="yellow.click"/>
    <set attributeName="fill" fill="freeze" to="blue" begin="blue.click"/>
    <set attributeName="fill" fill="freeze" to="fuchsia"
         begin="fuchsia.click"/>
  </rect>
  <rect x="120" y="270" width="100" height="15"
        style="fill:white;stroke:none"/>
  <text x="130" y="282" style="text-anchor:left;font-size:12">
    choose color
```

```
  </text>
  <rect x="220" y="270" width="15" height="15"
        style="fill:white;stroke:black"/>
  <path d="M222 272l11 0 -5.5 11z" style="fill:black"/>
  <!-- by click on rectangle list of colors is writed -->
  <rect id="go" x="220" y="270" width="15" height="15"
        style="fill:white;fill-opacity:0"/>
  <!-- list of colors, background, text, shadow for each -->
  <g display="none">
    <rect x="120" y="285" width="100" height="60" fill="white"/>
    <text x="130" y="297" style="text-anchor:left;font-size:12">
     red
    </text>
    <rect id="red" x="120" y="285" width="100" height="15"
          style="fill:black;fill-opacity:0">
      <!-- shadow when mouse is on color -->
      <set attributeName="fill-opacity" to="0.2"
           begin="red.mouseover" end="red.mouseout"/>
    </rect>
    <text x="130" y="312" style="text-anchor:left;
                                 font-size:12">yellow</text>
    <rect id="yellow" x="120" y="300" width="100" height="15"
          style="fill:black;fill-opacity:0">
      <set attributeName="fill-opacity" to="0.2"
           begin="yellow.mouseover" end="yellow.mouseout"/>
    </rect>
    <text x="130" y="327" style="text-anchor:left;font-size:12">
      blue
    </text>
    <rect id="blue" x="120" y="315" width="100" height="15"
          style="fill:black;fill-opacity:0">
      <set attributeName="fill-opacity" to="0.2"
           begin="blue.mouseover" end="blue.mouseout"/>
    </rect>
    <text x="130" y="342" style="text-anchor:left;font-size:12">
      fuchsia
    </text>
    <rect id="fuchsia" x="120" y="330" width="100" height="15"
          style="fill:black;fill-opacity:0">
      <set attributeName="fill-opacity" to="0.2"
           begin="fuchsia.mouseover" end="fuchsia.mouseout"/>
    </rect>
    <!-- show list of colors by click on go element, hide after choice -->
    <set attributeName="display" to="inline"
         begin="go.click"
         end="red.click;blue.click;yellow.click;fuchsia.click;bk.click"/>
  </g>
</svg>
```

Figure 9-5. Drop down list to select color

Animate element

The '**animate**' element is used to animate a single attribute or property over time.

This is the syntax of this element :

<**animate** id = "name"
xlink:href = "URI"
attributeName = "AttributeName"
attributeType = "CSS|XML|auto"
begin = 'begin-value-list'
end = 'end-value-list'
dur = 'Clock-value|"media"|"indefinite"'
min = 'Clock-value|"media"'
max = 'Clock-value|"media"'
restart = '"always"|"whenNotActive"|"never"'
repeatCount = 'Integer|"indefinite"'
repeatDur = 'Clock-value|"indefinite"'
fill = '"freeze"|"remove"'
calcMode = "**discrete | linear | paced | spline**"
values = "list of values"
from = "value"

to = "value"
by = "value"
keyTimes = "list of values"
keySplines = "list of values"
additive = "replace|sum"
accumulate = "none|sum" />

All this attributes are explained above.

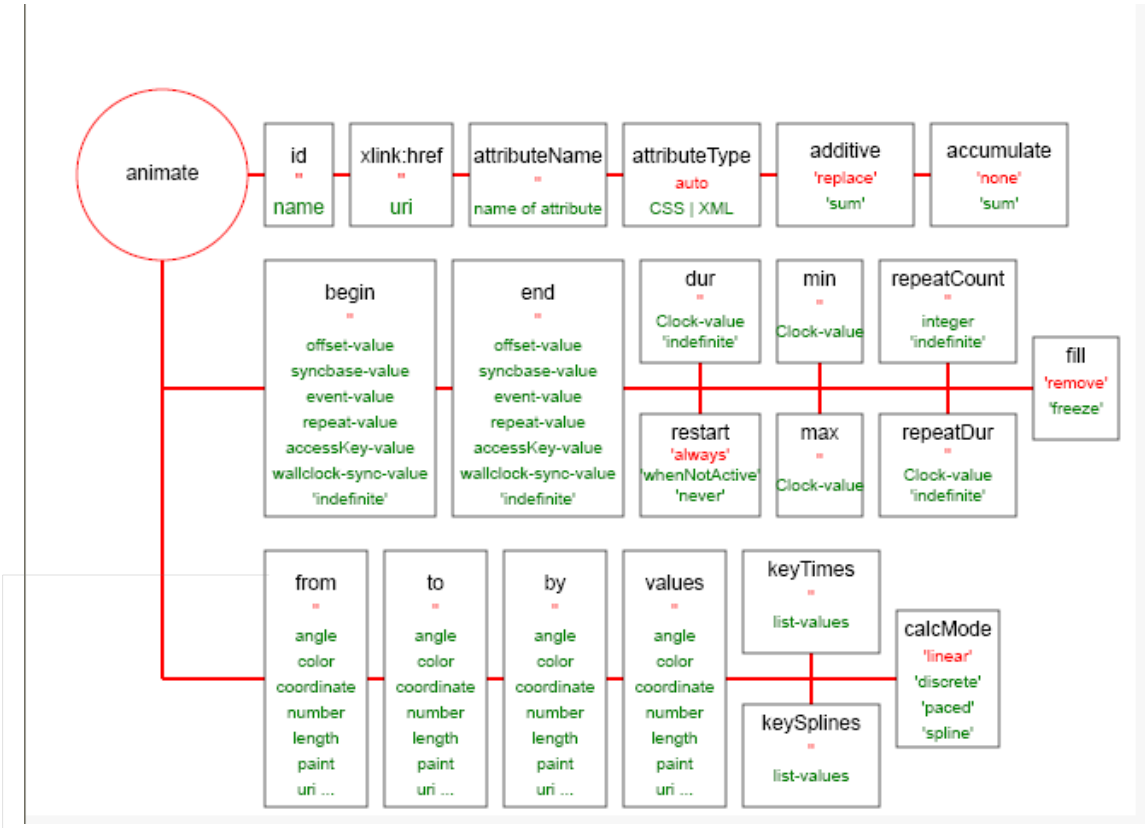

Diagram 9-2. Chart for 'animate' syntax

For 'values', 'from','to' and 'by' attributes, data types can be :
<angle>, <color>, <coordinate>, <integer>, <length>, <list of xxx>, <number>, <paint>, <percentage>, <uri> or all other data types used in animatable attributes and properties.

SVG allows both attributes and properties to be animated.

We see in chapter 7 example of animations for clipPath and in chapter 8 for filters.
Some other examples :

Animate stroke-dashoffset attribute of a basic shape

Some attributes give very interesting effect, by example stroke-dashoffset allow progressive drawing of any object (basic shape, path, text ...) .

To explain this, a very simple example, we take a line defined by a path :

```
<defs>
  <path id="line" style="stroke-width:2;stroke:black" d="m0 0l400 0"/>
</defs>
<use style="stroke-dasharray:400 400;stroke-dashoffset:400"
     xlink:href="#line" x="20" y="50"/>
```

As length of line is 400, with "stroke-dasharray:400 400;stroke-dashoffset:400" the line is not drawed.
If we change value for stroke-dashoffset, we see that if we go from 400 to 800, we get progressive drawing from right to left.
If we go from 400 to 0, we get progressive drawing from left to right.

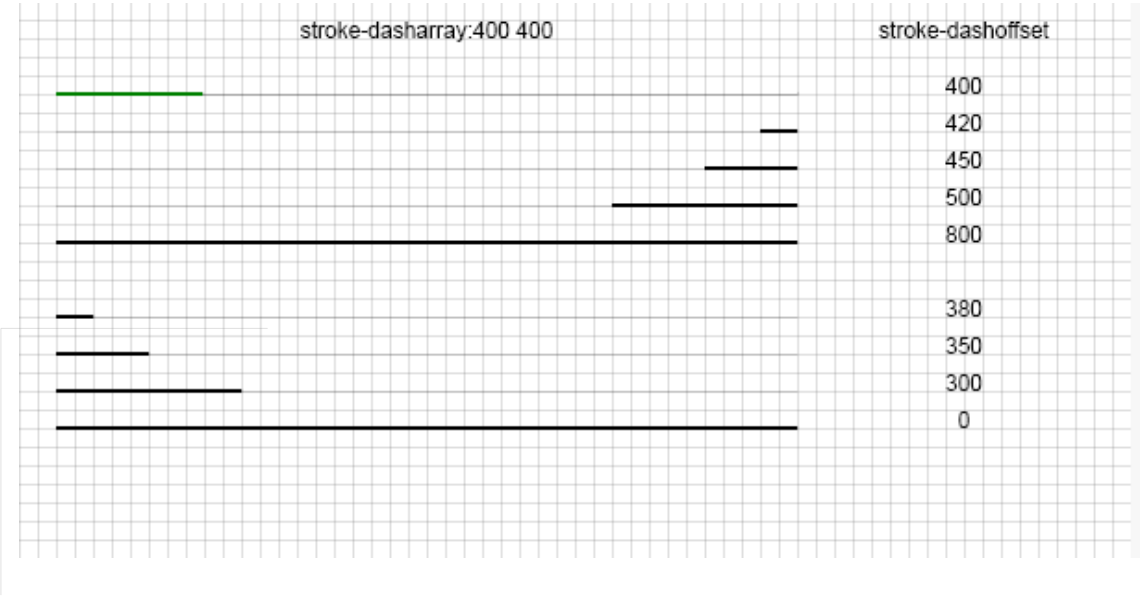

Figure 9-6. Effect of stroke-dashoffset

To get progressive drawing of line we can write :

```
<path d="M20 20l400 0" style="stroke-dasharray:400 400;stroke-
dashoffset:400">
  <animate attributeName="stroke-dashoffset" from="400" to="0"
           dur="4" begin="0" repeatCount="1" fill="freeze"/>
</path>
```

As xlink:href value is not provided, target of animate element is immediate parent, path element.
We use fill="freeze" to keep drawing of line at the end of animation.

Morphing with animate on 'd' attribute of path

If we give values over animation for 'd' attribute of a path, we can obtain 'morphing' in some cases, by example :

- we have only 'lineto' commands and there is same number in all values
- we have Bezier cubics or quadratics

First example with only 'lineto' commands in paths :
The square become a star, Figure 9-8 show some steps of morphing

Source code for this example (Example 9-8) :

```
<svg width="600" height="400" viewBox="-300 -200 600 400">
  <rect x="-300" y="-200" width="600" height="400" style="fill:#E0E0E0"/>
  <path d="M-100 -100l100 0 100 0 0 100 0 100 -100 0 -100 0 0 -100z"
        style="stroke:blue;fill-opacity:0.8; fill:blue">
    <animate begin="go.click" dur="10s" repeatCount="1" attributeName="d"
             values="M-100 -100l100 0 100 0 0 100 0 100 -100 0 -100 0 0 -
100z;
                     M-20 -20l20 -80 20 80 80 20 -80 20 -20 80 -20 -80 -80 -
      20z;
                     M-100 -100l100 0 100 0 0 100 0 100 -100 0 -100 0 0 -
      100z"/>
    <animate begin="go.click" dur="10s" repeatCount="1"
attributeName="fill"
             values="blue;green;blue"/>
  </path>
  <text x="250" y="180" style="font-size:25;fill:black;text-anchor:middle">
    GO
  </text>
  <rect id="go" x="220" y="155" width="60" height="30"
        style="fill:black;fill-opacity:0.1"/>
</svg>
```

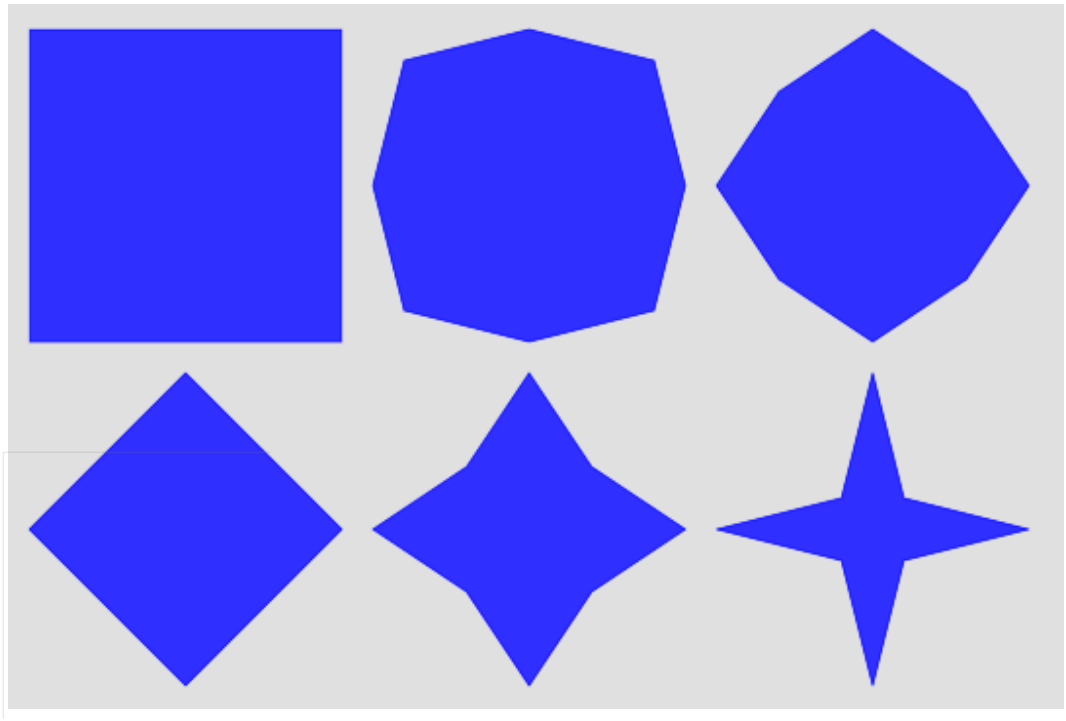

Figure 9-7. From square to star

Second example with quadratic Bezier curves and text on path :
We get morphing for text on path
Figure 9-9 show some steps for animation

Source code for this example (Example 9-9) :

```
<svg xml:space="preserve" width="600" height="400">
  <rect id="bkgrnd" x="0" y="0" width="600" height="400"
style="fill:#E0E0E0"/>
  <defs>
    <path id="courbe" d="M100 200Q200,200 300,200 T500,200"
          style="stroke:blue;fill-opacity:0.3;stroke-width:3;fill:none">
      <animate begin="go.click" dur="10s" repeatCount="1"
               attributeName="d"
              values="M100 200Q200,200 300,200 T500,200;
                      M100 200Q200,100 300,200 T500,200;
                      M100 200Q200,200 300,200 T500,200;
                      M100 200Q200,300 300,200 T500,200;
                      M100 200Q200,200 300,200 T500,200"/>
    </path>
  </defs>
  <text style="font-size:25;fill:red;text-anchor:middle">
    <textPath id="result" method="align" spacing="auto"
              startOffset="50%" xlink:href="#courbe">
      <tspan dy="-10">
        Textpath on morphing Bezier's curve
      </tspan>
```

```
    </textPath>
    <animate begin="go.click" dur="10s" repeatCount="1"
             attributeName="fill" values="red;green;red;green;red"/>
  </text>
  <use xlink:href="#courbe"/>
  <text x="550" y="380"
        style="font-size:25;fill:black;text-anchor:middle">
   GO
  </text>
  <rect id="go" x="520" y="355" width="60" height="30"
        style="fill:black;fill-opacity:0.1"/>
</svg>
```

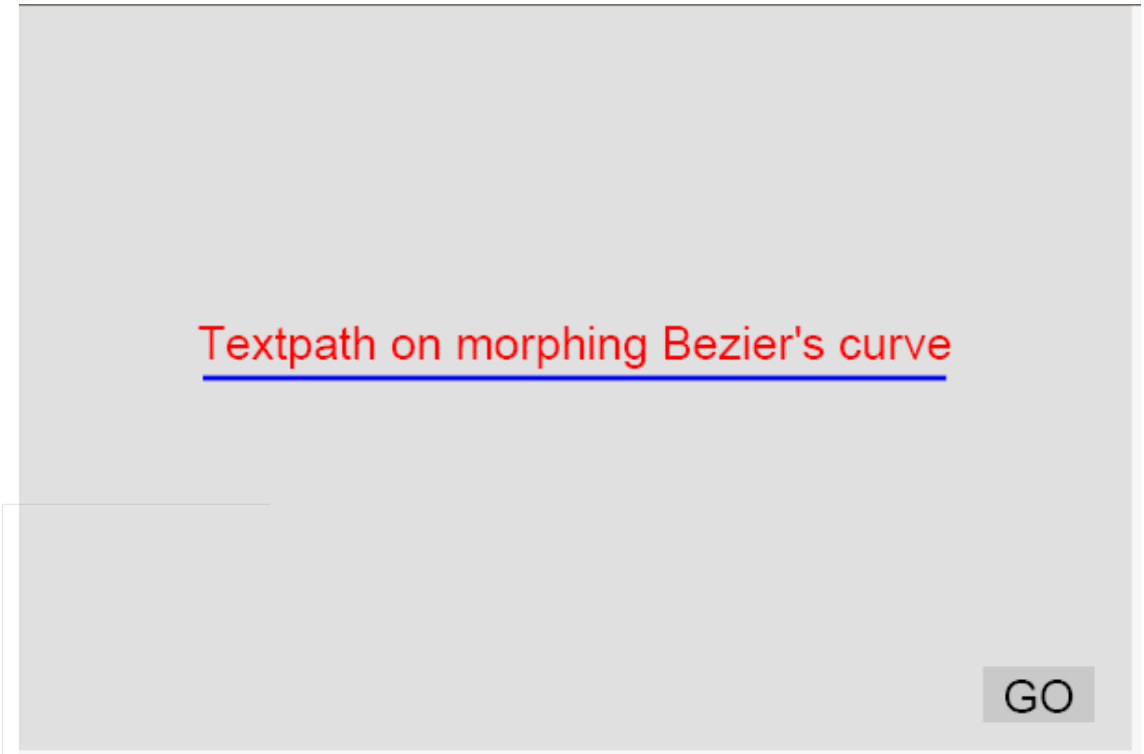

Figure 9-8. Morphing of text using animated textPath

Moving text on path with startOffset

We can animate 'startOffset' attribute, and text move from left to right along path.
If we put in 'd' attribute the path twice, text disappear by right and reappear on left.

Figure 9-10 show some steps of this animation.

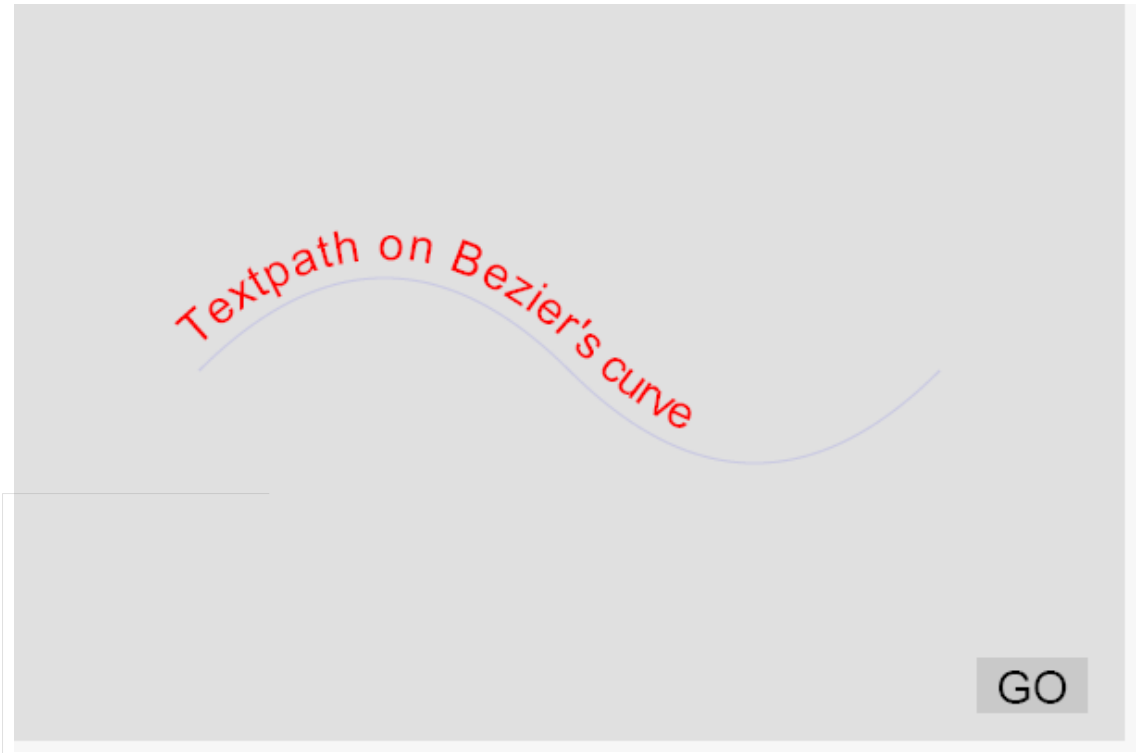

Figure 9-9. Text moving along Bezier's curve

Source code for this example (Example 9-10) :

```
<svg xml:space="preserve" width="600" height="400">
  <rect x="0" y="0" width="600" height="400" style="fill:#E0E0E0"/>
  <defs>
    <path id="courbe"
          d="M100 200Q200,100 300,200 T500,200"
          style="stroke:blue;stroke-opacity:0.1;stroke-width:1;fill:none"/>
  </defs>
  <text style="font-size:25;fill:red;text-anchor:left">
    <textPath id="result" method="align" spacing="auto"
              startOffset="0%" xlink:href="#courbe">
      <tspan dy="-10">
        Textpath on Bezier's curve
      </tspan>
      <animate begin="go.click" dur="5s" repeatCount="1"
               attributeName="startOffset" values="0%;100%"/>
    </textPath>
  </text>
  <use xlink:href="#courbe"/>
  <text x="550" y="380"
        style="font-size:25;fill:black;text-anchor:middle">
    GO
  </text>
  <rect id="go" x="520" y="355" width="60" height="30"
        style="fill:black;fill-opacity:0.1"/>
</svg>
```

AnimateColor element

The 'animateColor' element specifies a color transformation over time.

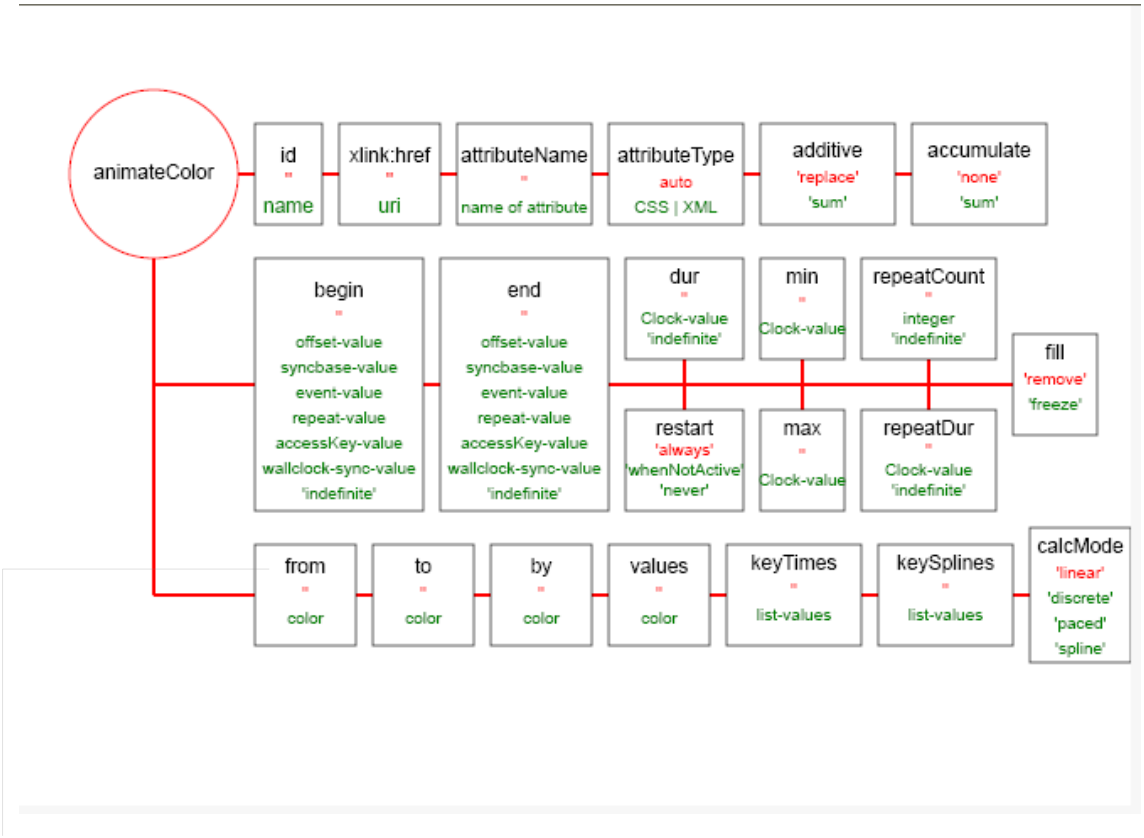

Diagram 9-3. Chart for 'animateColor' syntax

This is the syntax of this element :

<**animateColor** id = "name"
xlink:href = "URI"
attributeName = "AttributeName"
attributeType = "CSS|XML|auto"
begin = 'begin-value-list'
end = 'end-value-list'
dur = 'Clock-value|"media"|"indefinite"'
min = 'Clock-value|"media"'
max = 'Clock-value|"media"'
restart = '"always"|"whenNotActive"|"never"'
repeatCount = 'Integer|"indefinite"'
repeatDur = 'Clock-value|"indefinite"'
fill = '"freeze"|"remove"'
calcMode = "**discrete | linear | paced | spline**"
values = "list of values"

```
from = "value"
to = "value"
by = "value"
keyTimes = "list of values"
keySplines = "list of values"
additive = "replace|sum"
accumulate = "none|sum" />
```

All this attributes are explained above.

For 'values', 'from','to' and 'by' attributes, data types can be only <color>.

AnimateTransform element

The 'animateTransform' element animates a transformation attribute on a target element, we can use translation, scaling, rotation and/or skewing.

If we will animate 'matrix' transform, we have to use script because 'animate' can not use <transform-list> data.

This is the syntax of this element :

```
<animateTransform
        id = "name"
        xlink:href = "URI"
        attributeName = "AttributeName"
        attributeType = "CSS|XML|auto"
        begin = 'begin-value-list'
        end = 'end-value-list'
        dur = 'Clock-value|"media"|"indefinite"'
        min = 'Clock-value|"media"'
        max = 'Clock-value|"media"'
        restart = '"always"|"whenNotActive"|"never"'
        repeatCount = 'Integer|"indefinite"'
        repeatDur = 'Clock-value|"indefinite"'
        fill = '"freeze"|"remove"'
        calcMode = "discrete | linear | paced | spline"
        values = "list of values"
        from = "value"
        to = "value"
        by = "value"
        keyTimes = "list of values"
        keySplines = "list of values"
        additive = "replace|sum"
        accumulate = "none|sum"
        type = "translate | scale | rotate | skewX | skewY" />
```

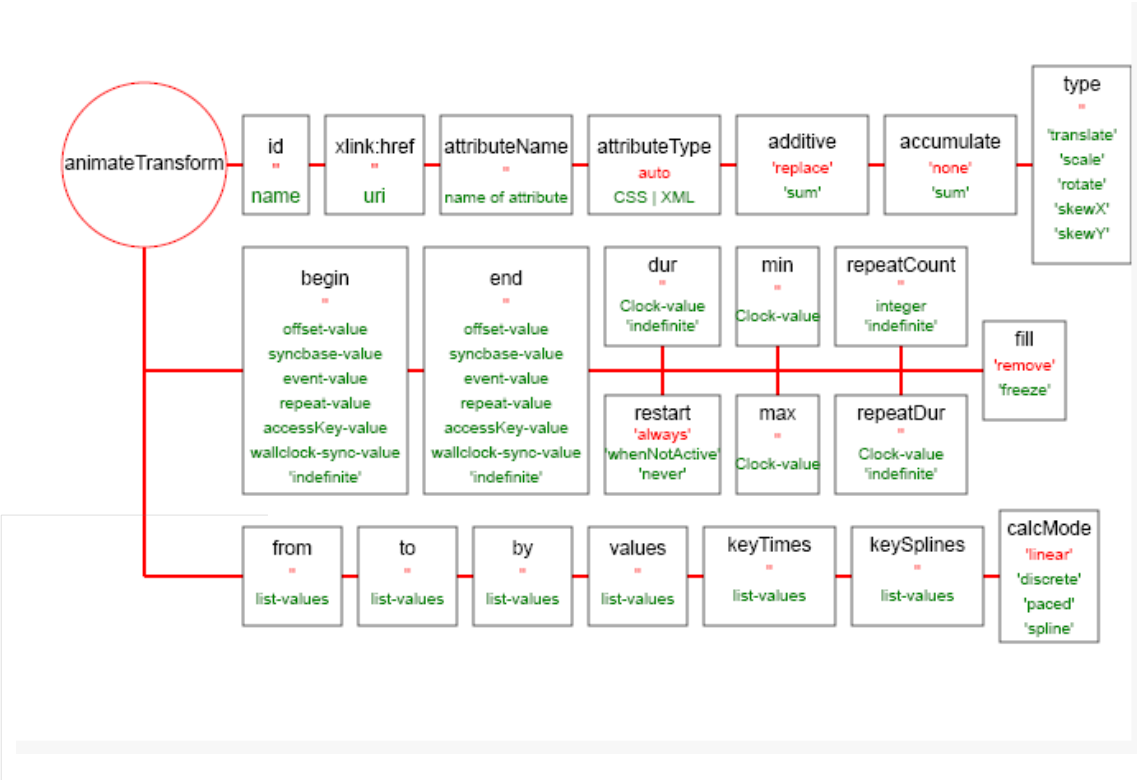

Diagram 9-4. Chart for 'animateTransform' syntax

All this attributes are explained above except 'type' .

Attribute '**type**' Indicates the type of transformation which is to have its values change over time.
Values in 'from' 'to' 'by' or 'values' have to be :
for a type="translate", as <tx> [,<ty>].
for a type="scale", as <sx> [,<sy>].
for a type="rotate", as <rotate-angle> [<cx> <cy>].
for a type="skewX" and type="skewY", as <skew-angle>.

Figure 9-10 show some examples using all type values.

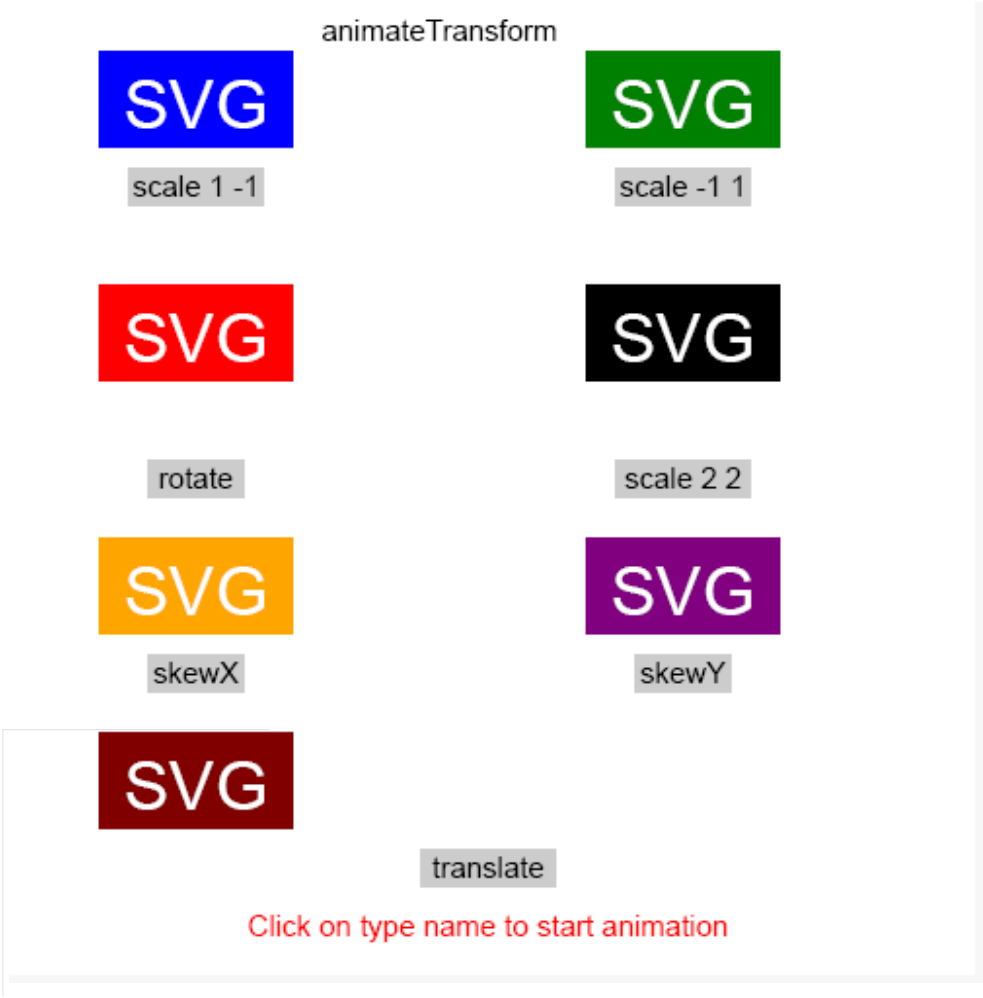

Figure 9-10. Effects using type="scale"

We can get rotation around horizontal or vertical axis, or decrease/increase size of object.

We can use 'animateTransform' element to create effect on letters of word 'SVG', each letter turn around vertical axis. We add 'animateColor' element to change color for filling letters to give impression that we see back of letters.
Figure 9-11 show some steps of animation.

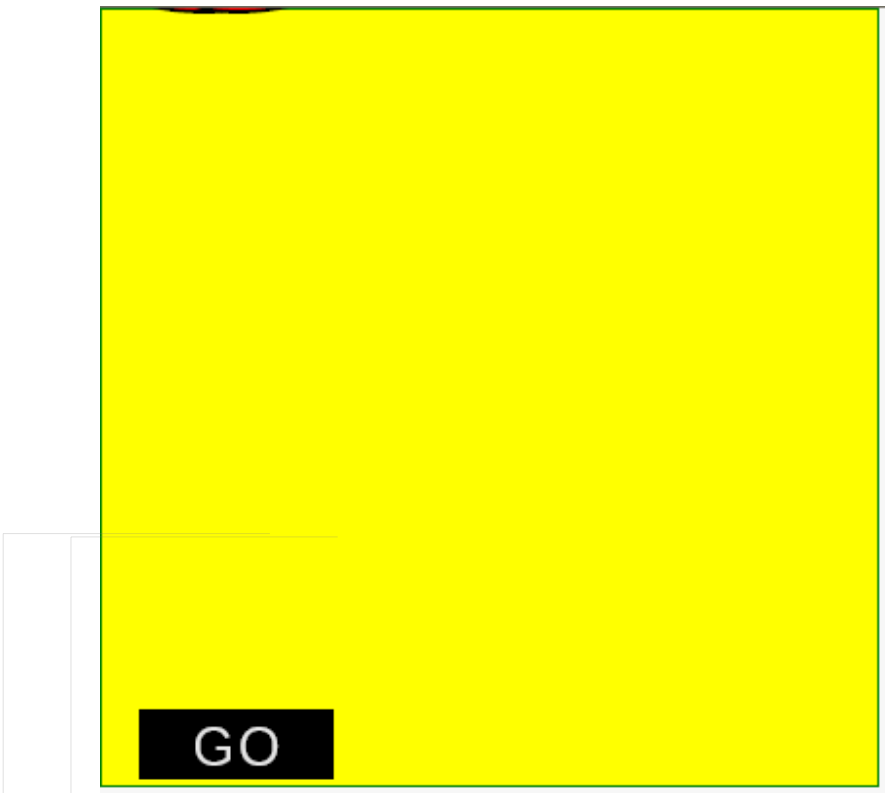

Figure 9-11. Letters of SVG turn around vertical axis

As we use 'scale' transformation, center is at origin, so we define letters with y="0" and put them in place in the word with a translation.

Animation begin by click on button (with "go" as 'id')

Source code for this example (Example 9-13) :

```
<svg width="200" height="200">
  <rect id='contour' x='0' y='0' width='200' height='200'
        style='stroke:green;fill:yellow'/>
  <g transform="translate(42,101)">
    <text x="-22.76" y="0" style="text-anchor:left;font-weight:bold;
          font-size:80;font-family:Verdana;fill:red;stroke:black;"
          transform="scale(1,1)" startOffset="0">
      S
      <animateTransform attributeName="transform" type="scale"
                        values="1,1;0,1;-1,1;0,1;1,1" begin="go.click"
                        repeatCount="1" dur="5s"/>
      <animateColor attributeName="fill" values="red;gray;gray;gray;red"
                    begin="go.click" repeatCount="1" dur="5s"/>
    </text>
  </g>
  <g transform="translate(98,101)">
    <text x="-26.1737" y="0" style="text-anchor:left;font-weight:bold;
          font-size:80;font-family:Verdana;fill:red;stroke:black;"
          transform="scale(1,1)" startOffset="0">
```

```
      V
      <animateTransform attributeName="transform" type="scale"
                        values="1,1;0,1;-1,1;0,1;1,1" begin="go.click"
                        repeatCount="1" dur="5s"/>
      <animateColor attributeName="fill" values="red;gray;gray;gray;red"
                    begin="go.click" repeatCount="1" dur="5s"/>
    </text>
  </g>
  <g transform="translate(153,101)">
    <text x="-26.4801" y="0" style="text-anchor:left;font-weight:bold;
          font-size:80;font-family:Verdana;fill:red;stroke:black;"
          transform="scale(1,1)" startOffset="0">
      G
      <animateTransform attributeName="transform" type="scale"
                        values="1,1;0,1;-1,1;0,1;1,1" begin="go.click"
                        repeatCount="1" dur="5s"/>
      <animateColor attributeName="fill" values="red;gray;gray;gray;red"
                    begin="go.click" repeatCount="1" dur="5s"/>
    </text>
  </g>
  <rect x="10" y="180" width="50" height="18" fill="black"/>
  <text x="35" y="195" style="text-anchor:middle;font-weight:bold;
        font-size:15;font-family:Arial;fill:white;stroke:black">
    GO
  </text>
  <rect id="go" x="10" y="180" width="50" height="18" opacity="0.1"/>
</svg>
```

With script, we can automatize this animation, we have only to give text to be animated. See below about script and SMIL animation.

AnimateMotion element

The 'animateMotion' element causes a referenced element to move along a motion path. Type of element can be :
'g' , 'defs' , 'use' , 'image' , 'switch' , 'path' , 'rect' , 'circle' , 'ellipse', 'line', 'polyline', 'polygon' , 'text' , 'clipPath' , 'mask' , 'a' or 'foreignObject'

This is the syntax of this element :
<**animateMotion**
id = "name"
xlink:href = "URI"
attributeName = "AttributeName"
attributeType = "CSS|XML|auto"
begin = 'begin-value-list'
end = 'end-value-list'
dur = 'Clock-value|"media"|"indefinite"'
min = 'Clock-value|"media"'
max = 'Clock-value|"media"'
restart = '"always"|"whenNotActive"|"never"'
repeatCount = 'Integer|"indefinite"'
repeatDur = 'Clock-value|"indefinite"'
fill = '"freeze"|"remove"'

```
calcMode = "discrete | linear | paced | spline"
values = "list of values"
from = "value"
to = "value"
by = "value"
keyTimes = "list of values"
keySplines = "list of values"
additive = "replace|sum"
accumulate = "none|sum"
path = "<path-data>"
keyPoints = "<list-of-numbers>"
rotate = "<angle> | auto | auto-reverse"
origin = "default">
        <mpath      id = "name"
                    xlink:href = "URI" />
< /animateMotion>
```

All this attributes are explained above except :

path : The motion path, expressed in the same format and interpreted the same way as the 'd' attribute on 'path' element.

keyPoints : a semicolon-separated list of floating point values between 0 and 1 and indicates how far along the motion path the object shall move at the moment in time specified by corresponding keyTimes value.

rotate values **auto** : the object is rotated over time by the angle of the direction.
auto-reverse : angle of the direction plus 180 degrees.
<angle> : angle relative to the x-axis. (0 by default)

origin : no effect for SVG

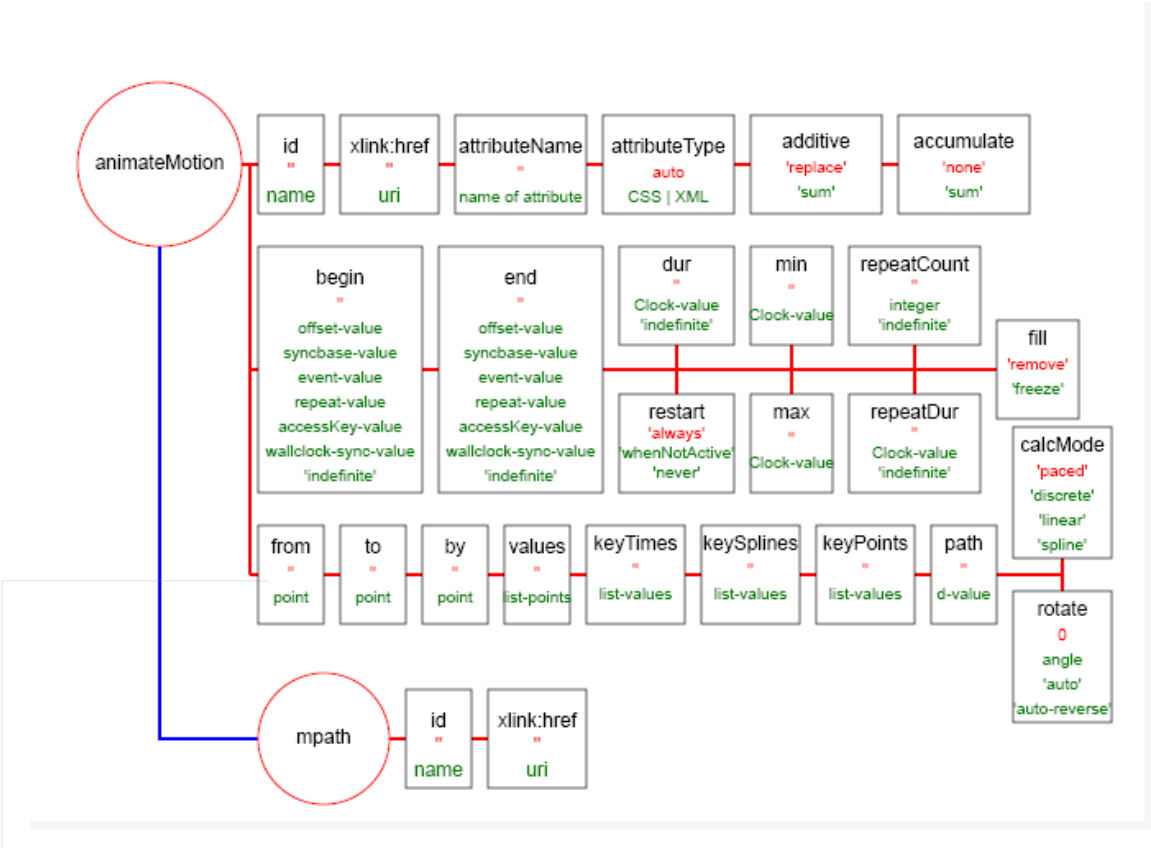

Diagram 9-5. Chart for 'animateMotion' syntax

To define path for 'animateMotion' we can write :

```
<animateMotion path="M100,250 C 100,50 400,50 400,250"  .......  />
```

or

```
<defs>
  <path id="MyPath" d="M100,250 C 100,50 400,50 400,250" />
</defs>
<animateMotion  ....................                                    >
  <mpath xlink:href="#MyPath" />
</animateMotion>
```

For keyPoints attribute, if we have
keyTimes = "0;0.25;0.5;0.7;1" keyPoints = "0;0.5;0.25;0.75;1" and dur = "10"

We have four steps in animation :
1) Object go from begin of trajectory to middle in 2.5 seconds
2) Object go back to quarter of trajectory in 2.5 seconds
3) Object go ahead to three quarters of trajectory in 2 seconds
4) Object go to end of trajectory in 3 seconds

Figure 9-12 show this example.

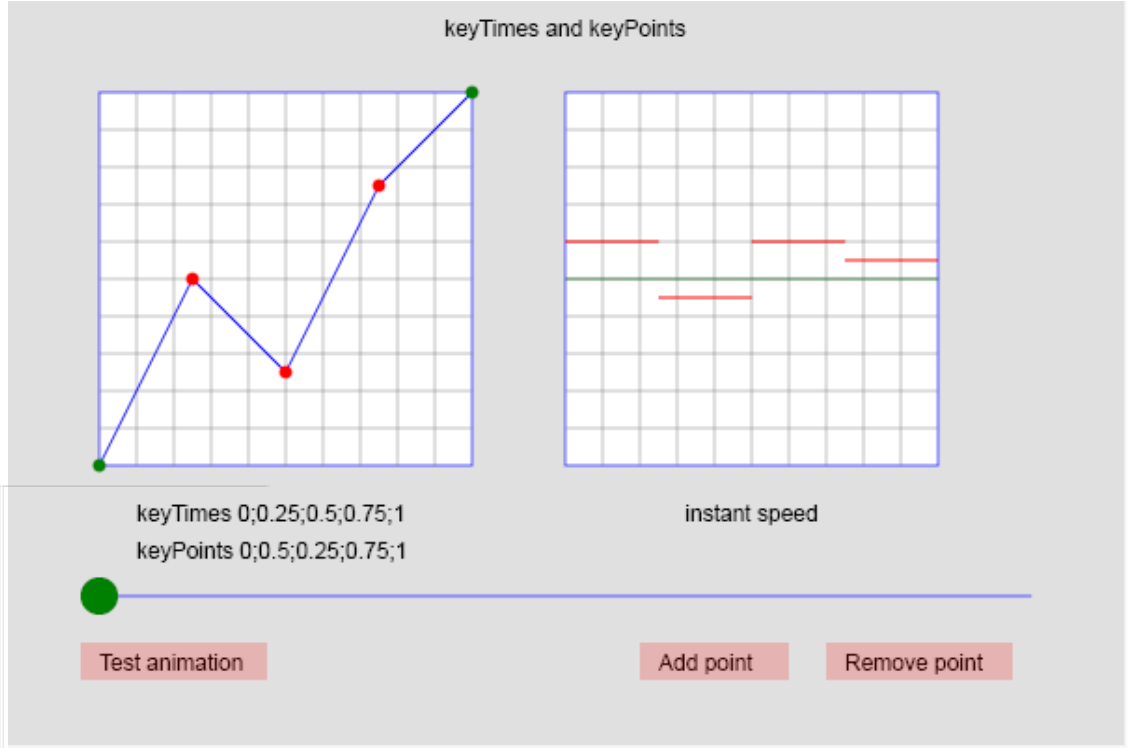

Figure 9-12. keyPoints and keyTimes

We can add keySplines to give various speed for each step.
Figure 9-13 show effect of values for 'rotate' attribute.

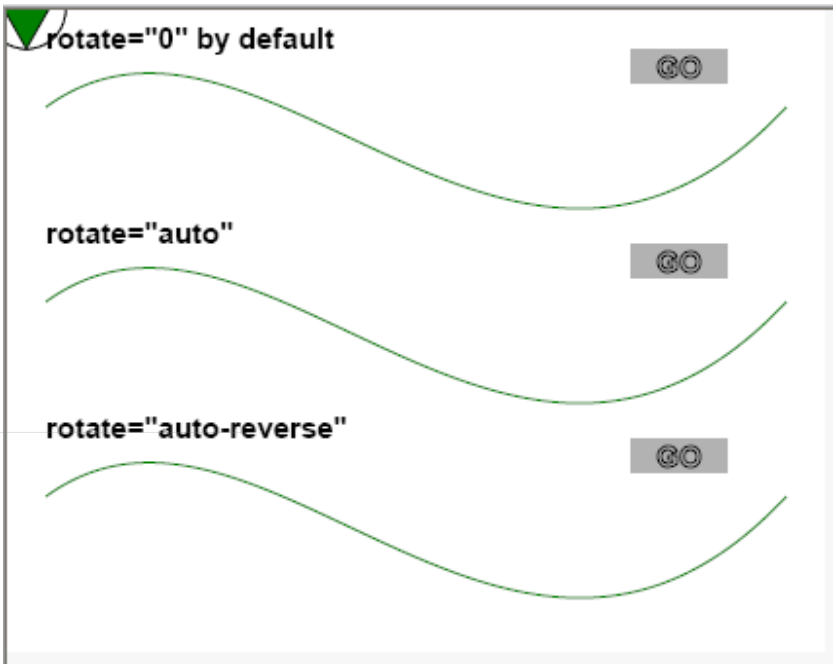

Figure 9-13. Different values for 'rotate' attribute

Animation and scripting

We can create animations by script or use script to create animated elements.

Script to create animated elements

With example of each letter of word 'SVG' turning around vertical axis, we can by script obtain all values to put letters in place and add animated elements.

For each letter there is two transformations, 'scale' to turn letter around vertical axis and 'translate' to put letter in place.

We must use additive='sum' to only modify values for 'scale' and keep same 'translate' transform.

Example 9-16 :

```
<svg width='200' height='200' onload="init(evt)">
  <script type="text/ecmascript">
    <![CDATA[
    // string to be animated
```

```
 var text_data='SVG';
 var svgdoc="";

// function to create letters and animated elements on loading svg


 function init(evt)
 {
 // create hidden text element with text_data as data

   svgdoc=evt.target.ownerDocument;
   node=svgdoc.createElement("text");
   node.setAttribute("x","100");
   node.setAttribute("y","100");
   node.setAttribute("style","visibility:hidden;text-anchor:middle;
   font-size:80;font-family:Arial;fill:black");
   node.setAttribute("id","texte");
   texte=svgdoc.createTextNode(text_data);
   node.appendChild(texte);
   where=svgdoc.getElementById('word');
   where.appendChild(node);

 // extract of hidden text element each letter to draw it and add animation

   objet=svgdoc.getElementById('texte');
   for (i=0;i<text_data.length;i++)
   {

   // draw each letter

         f=objet.getExtentOfChar(i);
         node=svgdoc.createElement('text');
         node.setAttribute('x',-f.width/2);
         node.setAttribute('y',"0");
         node.setAttribute('style','text-anchor:left;font-weight:bold;
         font-size:80;font-family:Verdana;fill:red;stroke:black');
         node.setAttribute('transform','translate('+(f.x+f.width/2)+','
         +(f.y+f.height)+') scale(1 1)');
         texte=svgdoc.createTextNode(text_data.substring(i,i+1));
         node.appendChild(texte);

  // add animateTransform using scale

         node_anim=svgdoc.createElement('animateTransform');
         node_anim.setAttribute('attributeName','transform');
         node_anim.setAttribute('type','scale');
         node_anim.setAttribute('additive','sum');
         node_anim.setAttribute('values','1,1;0,1;-1,1;0,1;1,1');
         node_anim.setAttribute('begin','go.click');
         node_anim.setAttribute('repeatCount','1');
         node_anim.setAttribute('dur','5s');
         node.appendChild(node_anim);

 // add animateColor element

         node_anim=svgdoc.createElement('animateColor');
         node_anim.setAttribute('attributeName','fill');
         node_anim.setAttribute('values','red;gray;gray;gray;red');
         node_anim.setAttribute('begin','go.click');
         node_anim.setAttribute('repeatCount','1');
```

```
            node_anim.setAttribute('dur','5s');
            node.appendChild(node_anim);
            where.appendChild(node);
        }
    };

    ]]>
  </script>
  <rect id='contour' x='0' y='0' width='200' height='200'
        style='stroke:green;fill:yellow'/>
  <g id='word'>
  </g>
  <rect x="10" y="180" width="50" height="18" fill="black"/>
  <text x="35" y="195" style="text-anchor:middle;font-weight:bold;
        font-size:15;font-family:Arial;fill:white;stroke:black">
   GO
  </text>
  <rect id="go" x="10" y="180" width="50" height="18" opacity="0.1"/>
</svg>
```

After loading svg, svg elements are the same that in first version of this example.

We can also use parseXML to create svg fragment and append it to DOM

```
      for (i=0;i<text_data.length;i++)
      {

      // draw each letter and add animate elements

            f=objet.getExtentOfChar(i);

      // build string to parse

            string_to_parse ="<text x='"
            +(-f.width/2)
            +"' y='0' style='text-anchor:left;font-weight:bold;
            font-size:80;font-family:Verdana;fill:red;stroke:black;'
            transform='translate("
            +(f.x+f.width/2)
            +","
            +(f.y+f.height)
            +") scale(1,1)' startOffset='0'>"
            +text_data.substring(i,i+1)
            +"<animateTransform attributeName='transform' type='scale'
            additive='sum' values='1,1;0,1;-1,1;0,1;1,1' begin='go.click'
            repeatCount='1' dur='5s'/>"
            +"<animateColor attributeName='fill'
            values='red;gray;gray;gray;red' begin='go.click'
            repeatCount='1' dur='5s'/></text>"

             // parse string and add it to svgdoc

            svg_fragment=parseXML(string_to_parse,svgdoc);

           // append elements to DOM

            where.appendChild(svg_fragment);
            }
```

We can add in string to parse "\n" for linefeed, so with "copy SVG" we can read easier code for svg elements.

Script to create animation

We can take same example and create animation only by script.

Source code for this example (Example 9-8) :

```
<svg width='200' height='200' onload="init(evt)">
  <script type="text/ecmascript">
    <![CDATA[
      var timevalue = 0;
      var timer_increment = 50; // each 50 milliseconds matrix change
      var max_time = 5000; // duration for animation in milliseconds
      var text_data='SVG'; // string to be animated
      var svgdoc="";
      var x_letter=new Array;
      var y_letter=new Array;

    // function to create object on loading svg

      function init(evt)
      {
     // create hidden text element with text_data as data
         svgdoc=evt.target.ownerDocument;
         node=svgdoc.createElement("text");
         node.setAttribute("x","100");
         node.setAttribute("y","100");
         node.setAttribute("style","visibility:hidden;text-anchor:middle;
         font-size:80;font-family:Arial;fill:black");
         node.setAttribute("id","texte");
         texte=svgdoc.createTextNode(text_data);
         node.appendChild(texte);
         where=svgdoc.getElementById('word');
         where.appendChild(node);

     // extract of hidden text element each letter to draw it

         objet=svgdoc.getElementById('texte');

         for (i=0;i<text_data.length;i++)
         {
              f=objet.getExtentOfChar(i);
              x_letter[i]=f.x+f.width/2;
              y_letter[i]=f.y+f.height;
              node=svgdoc.createElement('text');
              node.setAttribute("id","letter"+i.toString());
              node.setAttribute('x',-f.width/2);
              node.setAttribute('y',"0");
              node.setAttribute('style','text-anchor:left;font-weight:bold;
              font-size:80;font-family:Verdana;fill:red;stroke:black');
              node.setAttribute('transform',
              'translate('+x_letter[i]+','+y_letter[i]+') matrix(1 0 0 1 0 0)');
              texte=svgdoc.createTextNode(text_data.substring(i,i+1));
              node.appendChild(texte);
              where.appendChild(node)
         }
     };
```

```
    // function to create animation

    function anime(evt)
    {
       timevalue = timevalue + timer_increment;

// end of animation on condition

       if (timevalue > max_time)
              {
              timevalue=0;return
              };

// give value for matrix

       if (timevalue<=2500)
              {
              taille=(1250-timevalue)/1250
              }
              else
              {
              taille=(timevalue-3750)/1250
              };
       if (taille==0)
              {
              taille=0.001
              };

// affect value to matrix for each letter

       for (i=0;i<text_data.length;i++)
              {
              node=svgdoc.getElementById('letter'+i.toString());
              node.setAttribute("transform",
              'translate('+x_letter[i]+','+y_letter[i]+
              ') matrix('+taille+' 0 0 1 0 0)');

// change filling color to show back of letters

              if (timevalue==1250)
                     {
                     node.getStyle.setProperty("fill","gray")
                     };
              if (timevalue==3750)
                     {
                     node.getStyle.setProperty("fill","red")
                     };
              };

// recursive call to function

       setTimeout("anime()", timer_increment)
    };

// declare function as 'window' object

    window.anime = anime;

    ]]>
  </script>
  <rect x='0' y='0' width='200' height='200' style='stroke:green;fill:yellow'/>
  <g id='word'>
```

```
  </g>
  <rect x="10" y="180" width="50" height="18" fill="black"/>
  <text x="35" y="195" style="text-anchor:middle;font-weight:bold;
        font-size:15;font-family:Arial;fill:white;stroke:black">
   GO
  </text>
  <rect onclick="anime(evt)" id="go" x="10" y="180" width="50" height="18"
        opacity="0.1"/>
</svg>
```