Chapitre 9 Interactivité et Animation

"Le contenu SVG peut être interactif (autrement dit, répondre aux événements déclenchés par l'utilisateur) par l'utilisation des fonctionnalités suivantes du langage SVG:

- Les actions de l'utilisateur, comme presser un bouton du pointeur (par exemple, une souris) peuvent démarrer des animations ou déclencher l'exécution d'un script.
- L'utilisateur peut suivre des liens vers d'autres pages Web (voir les liens externes en SVG: l'élément 'a') par des actions telles qu'un click de souris quand le pointeur est sur des éléments graphiques particuliers.
- Dans de nombreux cas, suivant les valeurs de l'attribut 'zoomAndPan' de l'élément 'svg' et les caractéristiques du rendu, les utilisateurs peuvent zoomer ou faire un panoramique sur le contenu SVG.
- Les déplacements du pointeur peuvent provoquer des changements dans le dessin du curseur indiquant la position du pointeur."

Extrait des spécifications SVG du W3C

Créer des liens

Liens externes

SVG propose un élément 'a', analogue à l'élément 'a' de HTML, pour définir un lien. L'adresse du lien est définie par une URL, valeur de l'attribut 'xlink:href' de l'élément 'a'. Le lien peut être n'importe quelle ressource Web (par exemple, une image, un clip video, un extrait sonore, un programme, un autre document SVG, un document HTML, un élément du document, un élément d'un autre document, etc.). En activant ces liens (par un click de la souris, par une touche du clavier, par une commande vocale, etc.), l'utilisateur accéde à ces ressources.

Syntaxe de l'élément 'a':

```
<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>" >
    <!-- objets svg pour activer le lien -->
</a>
```

Exemple de code:

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

Quand le pointeur est sur ce rectangle, il prend la forme d'une main et si l'utilisateur clique, le fichier "MyFile.svg" est ouvert dans la même fenêtre.

Liens internes au document

Nous pouvons choisir comme lien n'importe quel élément du document en utilisant son attribut 'id', mais comme le contenu SVG représente souvent un dessin, une image, il est intéressant de pouvoir accéder à une vue particulière du document.

Nous pouvons définir un élément 'view', lui donner un identificateur et y faire référence dans un lien.

Syntaxe de l'élément 'view'

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

Pour définir l'élément dans une section <defs>:

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

Le lien est alors codé comme ceci:

```
<a xlink:href="#MyView">
  <!-- élément activateur  -->
</a>
```

ou

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

Nous pouvons également définir directement la nouvelle vue:

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

La figure 9-1 montre un zoom sur un rectangle quand on clique sur lui:

code de cet 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>
```

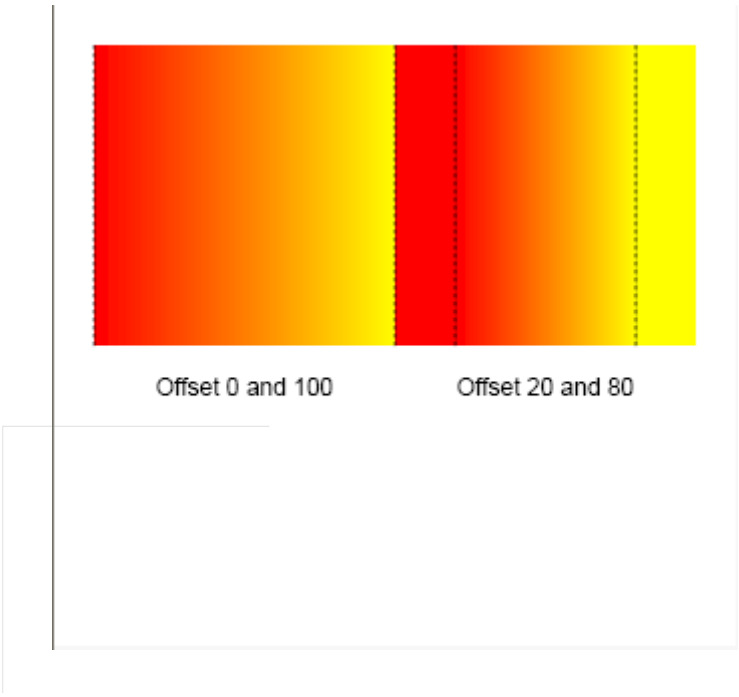

Figure 9-1. Cliquez pour avoir une nouvelle vue

Evénements supportés

Les événements peuvent être utilisés dans un script ou comme valeurs des attributs 'begin' ou 'end' d'éléments d'animation.

DOM2 catégorie UIEvent

nom	Description	nom DOM2	nom de l'attribut
focusin	Quand un élément est activé, par exemple un texte est sélectionné	DOMFocusIn	onfocusin
focusout	Quand un élément est désactivé, par exemple un texte est désélectionné.	DOMFocusOut	onfocusout
activate	Quand un élément est activé par un click de souris ou une touche du clavier. Un nombre est renvoyé pour donner le type d'activation 1 pour une simple activation (un click ou la touche 'Entrée'), 2 pour une hyper activation (un double click ou les touches 'Shift'+ 'Entrée').	DOMActivate	onactivate

DOM2 catégorie MouseEvent

nom	Description	nom DOM2	nom de l'attribut
click	Quand le bouton du pointeur est pressé et relâché sur un élément. Il est défini par un 'mousedown' et un ' mouseup' à la même position du pointeur. L'ordre des événements est: 'mousedown', 'mouseup', 'click'. pour de multiples clicks, la séquence se répéte et l'attribut 'detail' est incrémenté pour chaque répétition.	(idem)	onclick
mousedown	Quand le bouton du pointeur est pressé sur un élément.	(idem)	onmousedown
mouseup	Quand le bouton du pointeur est relâché sur un élément.	(idem)	onmouseup
mouseover	Quand le pointeur est sur un élément.	(idem)	onmouseover
mousemove	Quand le pointeur est déplacé sur un élément.	(idem)	onmousemove
mouseout	Quand le pointeur n'est plus sur un élément.	(idem)	onmouseout

Un ensemble d'événements pour gérer le clavier sera inclus dans les prochaines versions du DOM.

Comment utiliser ces événements?

Nous pouvons démarrer une animation avec un événement placé dans l'attribut 'begin'.
Par exemple, cette animation commencera quand l'utilisateur cliquera sur l'objet 'go':

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

Nous pouvons aussi déclencher l'exécution d'une fonction par un click sur un objet:

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

Dans le premier exemple nous utilisons 'click' nom de l'événement et dans le second 'onclick' qui est le nom de l'attribut événement.

DOM2 catégorie MutationEvent

nom	Description	nom DOM2	nom de l'attribut
DOMSubtree Modified	Pour tout changement dans le document.	(idem)	none
DOMNode Inserted	Quand un noeud est ajouté à un autre nœud	(idem)	none
DOMNode Removed	Quand un noeud est enlevé à un autre noeud	(idem)	none
DOMNode RemovedFrom Document	Quand un noeud est enlevé du document	(idem)	none
DOMNode InsertedInto Document	Quand un noeud est ajouté au document	(idem)	none
DOMAttr Modified	Quand un noeud est modifié	(idem)	none
DOM CharacterData Modified	Quand 'CharacterData' d'un noeud est modifié (contenu d'un nœud 'text' par exemple)	(idem)	none

Evénements liés au document

nom	Description	nom DOM2	nom de l'attribut
SVGLoad	Quand le document a été chargé et parsé	(idem)	onload
SVGUnload	Quand un document est supprimé dans la fenêtre ou la 'frame'	(idem)	onunload
SVGAbort	Quand le chargement échoue	(idem)	onabort
SVGError	En cas d'erreur au chargement ou dans l'exécution d'un script	(idem)	onerror
SVGResize	Quand la fenêtre est redimensionnée	(idem)	onresize
SVGScroll	Quand il y a scrolling de la fenêtre	(idem)	onscroll
SVGZoom	Quand il y a un zoom sur la fenêtre	aucun	onzoom

Le plus utilisé est 'SVGLoad', nous appelons la fonction init(evt) au chargement du svg:

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

Nous utilisons 'SVGResize', 'SVGScroll' et 'SVGZoom' quand nous voulons traduire les coordonnées du pointeur dans le système de coordonnées du SVG pour un script:

Avec ce code:

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

Quand l'utilisateur redimensionne la fenêtre, les objets graphiques sont redessinés mais le système de coordonnées a changé, clientX et clientY donnent la position du pointeur dans la fenêtre mais nous avons surtout besoins des coordonnées dans le système de coordonnées SVG pour par exemple modifier ou créer un objet répondant à la position du pointeur.
Nous pouvons utiliser ce code :

```
ratio=Math.min(window.innerHeight/400,window.innerWidth/600);
```

```
xm=evt.clientX/ratio;
ym=evt.clientY/ratio;
```

pour avoir les coordonnées SVG.

Si l'origine dans 'viewBox' n'est pas 0,0 nous ajoutons leurs valeurs.
Si l'utilisateur zoome ou fait un panoramique, nous utilisons 'currentScale', 'currentTranslate.x' et 'currentTranslate.y' pour retrouver notre système de coordonnées.

La figure 9-2 montre un test pour connaître les coordonnées SVG du pointeur dans tous les cas.
Pour cet exemple, le document SVG est 'embedded' dans un fichier HTML avec des valeurs fixées pour 'width' et 'height'.
Nous utiliserons ces formules:

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

Figure 9-2. Test pour calculer les coordonnées SVG du pointeur

Evénements pour les animations

nom	Description	nom DOM2	nom de l'attribut
beginEvent	Quand l'animation débute	aucun	onbegin
endEvent	Quand l'animation se termine	aucun	onend
repeatEvent	Quand l'animation se répète	aucun	onrepeat

La propriété 'pointer-events'

La propriété 'pointer-events' précise dans quelles circonstances un objet graphique déclenche l'événement.

'pointer-events'

Valeur: visiblePainted | visibleFill | visibleStroke | visible |painted | fill | stroke | all | none | inherit
Par défaut: visiblePainted
S'applique à: éléments graphiques
transmissible: oui
Pourcentages: N/A
Media: visuel
Animable: oui

visiblePainted
Quand la propriété 'visibility' vaut 'visible' et que le pointeur est sur une aire peinte.

visibleFill
Quand la propriété 'visibility' vaut 'visible' et que le pointeur est sur une aire remplie.

visibleStroke
Quand la propriété 'visibility' vaut 'visible' et que le pointeur est sur le contour.

visible
Quand la propriété 'visibility' vaut 'visible' et que le pointeur est sur l'intérieur ou le contour.

painted
Quand le pointeur est sur une aire peinte. La valeur de 'visibility' ne compte pas.

fill
Quand le pointeur est sur une aire remplie. La valeur de 'visibility' ne compte pas.

stroke
Quand le pointeur est sur le contour. La valeur de 'visibility' ne compte pas.

all
Quand le pointeur est sur le contour ou l'intérieur

none
L'élément ne déclenche aucun événement. Nous pouvons utiliser cette valeur pour qu'un objet soit totalement inerte.

Pour les éléments 'text':
Les valeurs 'visibleFill', 'visibleStroke' et 'visible' sont équivalentes.
Les valeurs 'fill', 'stroke' et 'all' sont équivalentes.

Pour les images bitmap:
Les valeurs 'visibleFill', 'visibleStroke' et 'visible' sont équivalentes.
Les valeurs 'fill', 'stroke' et 'all' sont équivalentes.

Propriété et élément 'cursor'

Propriété 'Cursor'

Sa syntaxe:

'cursor'

Valeur:	[[<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
Par défaut:	auto
S'applique à:	container et éléments graphiques
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel, interactif
Animable:	oui

Cette propriété définit le dessin du curseur.
Les valeurs possibles:
auto Le visualiseur choisit la forme du curseur suivant le contexte.
crosshair Une croix (deux segments formant le symbole +).
default Dépend de la plateforme. Souvent une flèche.
pointer Indique un lien.
move Indique que quelque chose se déplace.
e-resize, ne-resize, nw-resize, n-resize, se-resize, sw-resize, s-resize, w-resize
Indique qu'un sommet est déplacé. Par exemple 'se-resize' est utilisé pour un mouvement démarrant du coin inférieur droit.
text Indique qu'un texte peut être sélectionné. Souvent une barre "I' majuscule.
wait Indique que le programme s'affaire. Souvent une montre, un sablier.
help Une aide est disponible. Souvent un "?" ou une bulle.
<uri> L'URL désigne une ressource pour dessiner le curseur.

L'élément 'cursor'

L'élément 'cursor' peut définir un dessin de curseur indépendant de la plateforme. Il est recommandé de partir d'une image PNG et de définir un élément 'cursor' qui référence l'image PNG et précise la position du pointeur dans l'image.

Syntaxe de l'élément 'cursor':

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

Les attributs:
x : La coordonnée en x dans le système de coordonnées du curseur qui donne la position du pointeur.
y : La coordonnée en y dans le système de coordonnées du curseur qui donne la position du pointeur.
xlink:href : URI qui référence l'élément ou le fichier qui permettra de dessiner le curseur.

Animation

Nous avons différents éléments d'animation, 'animate', 'animateMotion', 'animateColor', 'animateTransform' et 'set'. Ils seront en général placés comme descendants de l'objet graphique à animer.

Attributs communs

Elément cible et 'xlink:href'

Nous utilisons l'attribut **'xlink:href'** pour référencer l'élément cible de l'animation. Si nous ne donnons pas de valeur à cet attribut, la cible sera l'élément parent de l'élément d'animation.

Exemple de code avec une valeur pour 'xlink:href':

```
<rect id="MyRect" x="40" y="100" width="320" height="80" fill="white"/>

<!-- d'autres éléments -->

<set xlink:href="#MyRect" attributeName="fill" fill="freeze" to="red"
      begin="red.click"/>
```

Ou en utilisant le parent comme cible:

```
<rect x="40" y="100" width="320" height="80" fill="white">
    <set attributeName="fill" fill="freeze" to="red" begin="red.click"/>
</rect>
```

Une animation consiste à modifier les valeurs d'un attribut de l'élément cible.
Dans notre exemple, quand l'utilisateur clique sur l'objet qui a l'identificateur 'red' (événement 'red.click'), le rectangle qui était blanc devient rouge.

Attribut ou propriété visée

Nous utilisons **'attributeName'** pour le nom de l'attribut ou de la propriété et **'attributeType'** pour son type.

Valeurs de ces attributs:

attributeName="AttributeName"

attributeType="CSS|XML|auto"
CSS pour une propriété CSS
XML pour un attribut XML
auto : le visualiseur cherche parmi les propriétés CSS et s'il n'a pas trouvé la propriété, il continue dans l'espace XML de l'élément.

Contrôler le timing

Ces attributs sont communs à tous les éléments d'animation:

Quand commence l'animation?

L'attribut 'begin' définit le moment où commence l'animation:

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

Voyons les valeurs possibles en détail:

offset-value : pour SVG, le temps commence avec le chargement du document.

Avec begin="10" l'animation démarre 10 secondes après le chargement du SVG.

syncbase-value : durée relative au début ou à la fin d'une autre animation

Avec begin="MyAnim.**end**+5" l'animation commence 5 secondes après la fin de l'animation MyAnim.

event-value : l'animation démarre sur un événement

Avec begin="MyRect.**click**+3" l'animation commence 3 secondes après que l'utilisateur ait cliqué sur l'objet 'MyRect'.

repeat-value : l'animation démarre après un certain nombre de répétitions d'une autre animation.

Avec begin="MyAnim.**repeat**(2)" l'animation démarrera quand MyAnim aura été répété deux fois.

accessKey-value : l'action démarre sur une touche du clavier.

Avec begin="**accessKey**(g)" l'animation commence quand l'utilisateur presse la touche 'g' du clavier.

wall-clock-sync-value : l'action démarre à une date donnée en temps universel.

"indefinite" : Le début de l'animation sera commandé par l'appel de la méthode beginElement() dans un script ou par un lien pointant sur l'élément d'animation.

Comment définir la durée de l'animation?

Nous utilisons l'attribut 'dur' pour définir la durée de l'animation

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

Clock-value : définit la durée de l'animation en secondes
"media" est ignoré pour SVG
"indefinite" utilisable surtout avec l'élément 'set'

Comment définir la fin de l'animation?

L'attribut 'end' permet de forcer la fin de l'animation indépendamment de la valeur de 'dur'.

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

Les types de valeurs sont les mêmes que pour 'begin'.
Avec end="indefinite" la méthode appelée est endElement()

Définir des valeurs extrêmes pour la durée de l'animation

Nous avons les attributs 'min' et 'max'
min = 'Clock-value|"media"'
La valeur par défaut est 0
max = 'Clock-value|"media"'

Pour savoir quand l'animation peut redémarrer

Nous avons l'attribut 'restart':
restart = '"always"|"whenNotActive"|"never"'

"always" : L'animation peut recommencer n'importe quand, c'est la valeur par défaut de l'attribut.
"whenNotActive" : L'animation peut recommencer si elle n'est pas en cours d'exécution.
"never" : L'animation ne peut jamais recommencer.

Pour définir le nombre de répétitions de l'animation

Ce nombre de répétitions peut-être défini par un nombre - 'repeatCount' ou une durée - 'repeatDur' :

repeatCount = 'Integer|"indefinite"'
repeatDur = 'Clock-value|"indefinite"'

Avec "indefinite", l'animation se répéte indéfiniment (jusqu'à la fermeture du document)

Pour définir l'état de l'objet graphique à la fin de l'animation

Nous avons l'attribut 'fill' (à ne pas confondre avec la propriété 'fill' de remplissage)

fill = '"freeze"|"remove"'

Avec fill="**freeze**", l'effet de l'animation est gelé jusqu'à la répétition éventuelle de celle-ci.

Avec fill="**remove**", l'objet graphique retrouve son état initial à la fin de l'animation.

L'évolution des choses au cours de l'animation

Ces attributs - à l'exception de 'to' - ne sont pas gérés par l'élément **'set'**.

Pour définir le type d'interpolation au cours de l'animation

Nous avons l'attribut 'calcMode'

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

discrete : Nous passons d'une étape à l'autre sans interpolation. Si par exemple un rectangle s'étire au cours de l'animation, nous aurons une animation saccadée.
linear : Interpolation linéaire entre les valeurs calculées au cours de l'animation. C'est la valeur par défaut sauf pour l'élément 'animateMotion'.
paced : Permet d'avoir un mouvement régulier durant l'animation. Ceci n'est possible que pour des valeurs numériques - position de l'objet, distance, dimensions ...
Les attributs 'keyTimes' et 'keySplines' seront ignorés.
Pour l'élément 'animateMotion', c'est la valeur par défaut.
spline : Interpolation fonction d'une liste de valeurs correspondant à une fonction temps définie par une courbe cubique de Bézier. Cette courbe est définie par ses points de contrôle dans l'attribut 'keySplines'. L'attribut 'keyTimes' permet de fractionner l'animation en étapes de durées différentes et nous pouvons avoir une courbe de Bézier pour chaque intervalle. Si nous utilisons ces attributs, il est impératif de choisir calcMode="spline".

Pour définir les valeurs utilisées pour l'animation

Nous avons les attributs 'values', 'from', 'to' et 'by' :

values = "liste de valeurs"
Si nous voulons animer l'attribut 'viewBox' de l'élément 'svg', nous pouvons indiquer values="0,0,200,200;0,0,100,100;0,0,200,200" et dans l'animation, 'viewBox' variera de "0 0 200 200" à "0 0 100 100" (nous aurons un zoom x2) et revenir à "0 0 200 200".
Remarquez que les valeurs sont séparées par un point-virgule ";", chacune des valeurs étant formée de quatre nombres séparés par des virgules ou des espaces.
Pour des couleurs de remplissage nous pouvons écrire values="red;yellow;green" et l'objet passera du rouge au jaune, puis au vert.

from = "value"
Pour l'exemple de 'viewBox' précédent nous pouvons écrire from="0,0,200,200"

to = "value"
Pour l'exemple de 'viewBox' précédent nous pouvons écrire to="0,0,200,200".

by = "value"
Pour l'exemple de 'viewBox' précédent nous pouvons écrire by="0,0,100,100"

Pour définir la vitesse de chacune des étapes de l'animation

L'attribut 'keyTimes'

keyTimes = "liste de valeurs entre 0 et 1"
Il doit y avoir autant de valeurs dans 'keyTimes' que dans 'values'.
Ces valeurs doivent être en ordre croissant et représentent un pourcentage de la durée totale de l'animation
Avec calcMode = 'paced', l'attribut est ignoré.

Prenons un exemple:
Nous écrivons keyTimes="0;0.195;0.405;0.7;1" values="0;400;47.5;450;500".
Si la durée de l'animation est de 10 secondes (dur="10"), et que les valeurs concernent la largeur d'un rectangle, nous aurons les étapes suivantes dans l'animation:

De 0 à 1.95s, la largeur du rectangle croît de 0 à 400
de 1.95s à 4.05s la largeur du rectangle décroît de 400 à 47.5
de 4.05s à 7s la largeur du rectangle croît de 47.5 à 450
et enfin
de 7s à 10s (fin de l'animation) la largeur du rectangle croît de 450 à 500.

Si nous ne donnons pas de valeur à l'attribut 'keySplines', la vitesse de l'animation est constante pour chacune de ces étapes.

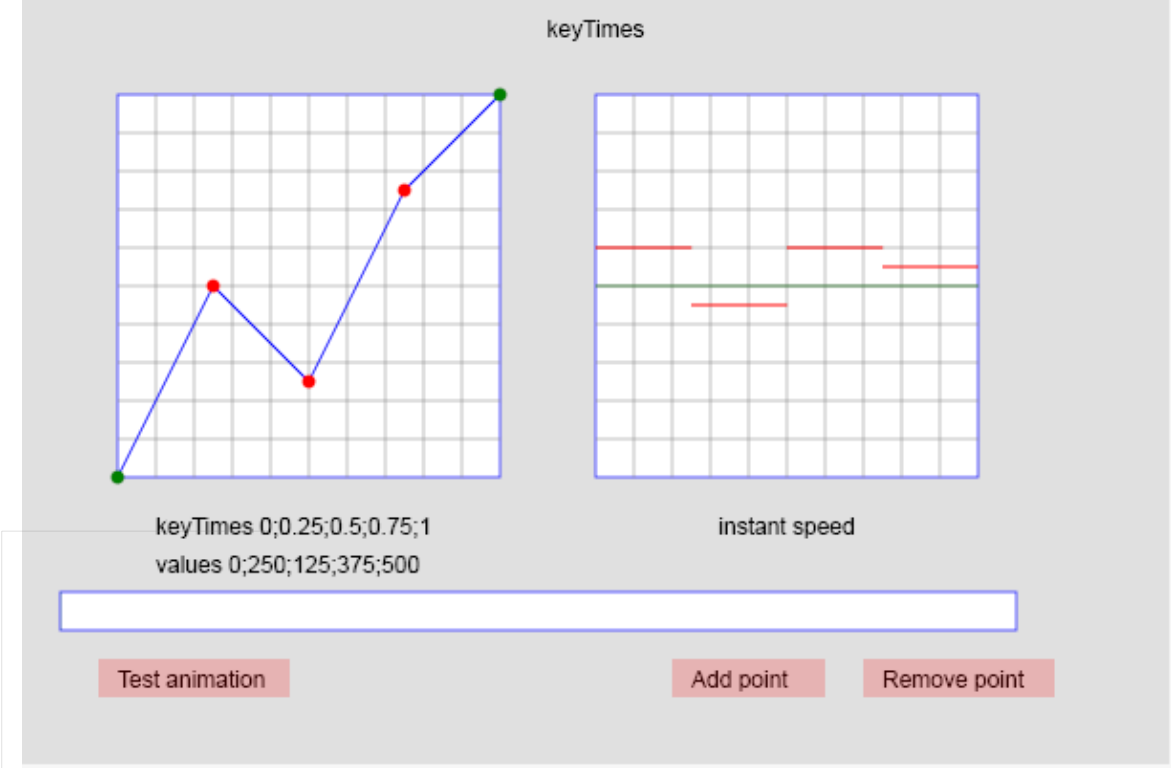

Figure 9-3. 'keyTimes' et 'values'

Déplacez les cercles rouges pour modifier keyTimes et values, et testez l'animation obtenue.

L'attribut 'keySplines'

keySplines = "liste de liste de 4 nombres entre 0 et 1"

Les quatre nombres sont les coordonnées des deux points de contrôle de la cubique de Bézier, celle-ci commençant au point (0,0) et se terminant en (1,1). Cette courbe de Bézier contrôle la vitesse pour une étape, si nous donnons une valeur à 'keyTimes' ou pour l'animation complète et dans ce cas une série de quatre nombres suffit.
Si 'calcMode' n'a pas la valeur "spline", l'attribut est ignoré.

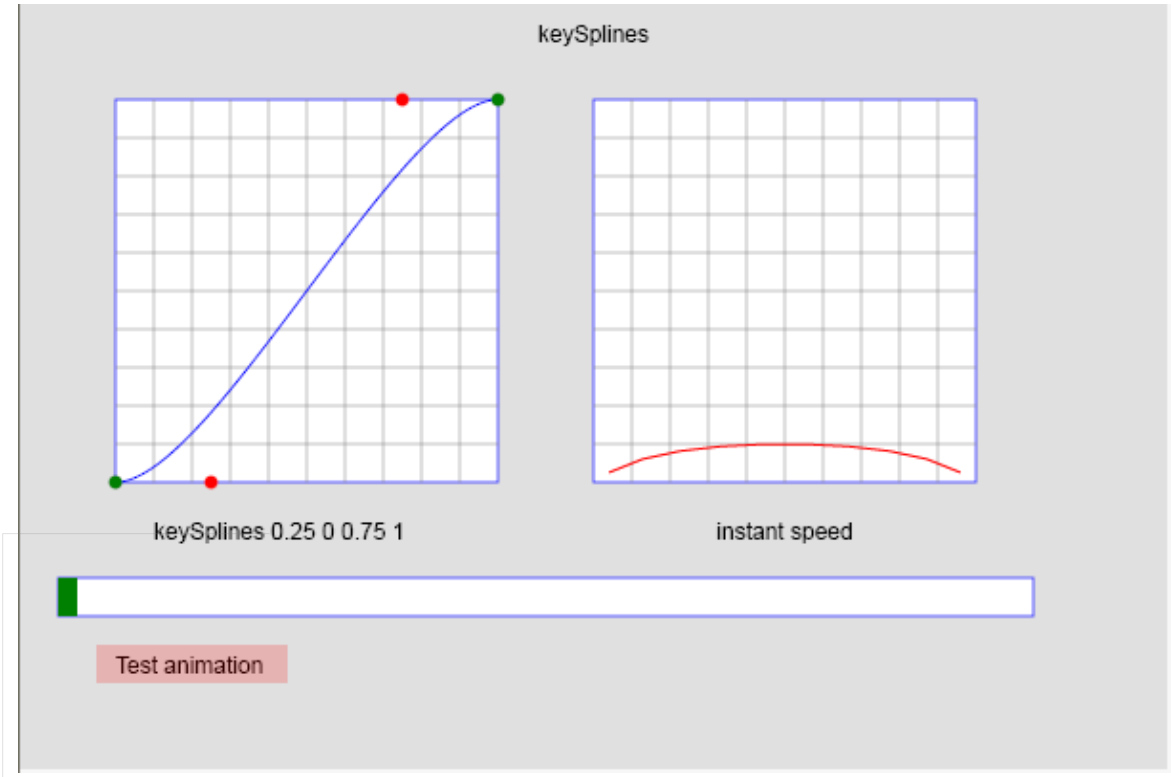

Figure 9-4. 'keySplines' et vitesse

Déplacez les cercles rouges pour modifier la courbe de Bézier, et testez l'animation obtenue.

Revenons à notre exemple précédent:
En plus de keyTimes="0;0.195;0.405;0.7;1" values="0;400;47.5;450;500" nous ajoutons 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" et calcMode="spline".
Nous avons toujours les mêmes étapes pour l'animation, mais la vitesse pour chacune de ces étapes est contrôlée par la courbe de Bézier correspondante. La vitesse augmente du début à la fin pour les étapes 1, 3 et 4 alors qu'elle décroît pour la seconde étape.

Il nous reste un dernier attribut '**keyPoints**' mais il ne s'applique qu'à l'élément '**animateMotion**' nous le verrons dans le paragraphe consacré à cet élément.

Effets de l'animation sur les valeurs initiales

Les valeurs données peuvent remplacer ou s'ajouter aux valeurs initiales suivant la valeur de l'attribut 'additive':

additive = "replace|sum"
replace : les valeurs données dans l'animation remplacent les valeurs existantes. C'est la valeur par défaut.
sum : les valeurs données dans l'animation s'ajoutent aux valeurs initiales - si la somme a un sens pour les valeurs concernées.

Effets de l'animation dans la répétition

Quand nous répétons une animation, les effets peuvent se cumuler ou non suivant la valeur de l'attribut 'accumulate':

accumulate = "none|sum"
none : Nous repartons chaque fois de la valeur initialee Les effets ne sont pas cumulés. C'est la valeur par défaut.
sum : Pour chaque répétition de l'animation, nous repartons avec la nouvelle valeur, les effets sont cumulés.

Les attributs 'additive' et 'accumulate' n'ont de sens que pour des valeurs de type :
<angle>, <color>, <coordinate>, <integer>, <length>, <number>, <paint>, <percentage>, <uri> et <transform-list>.

L'élément 'set'

L'élément 'set' permet simplement d'attribuer une nouvelle valeur à un attribut pour une durée spécifiée ou sur un événement

La syntaxe de cet élément:

```
<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" />
```

Tous ces attributs ont été détaillés.

Pour l'attribut 'to', les types de valeurs possibles sont:
<angle>, <color>, <coordinate>, <integer>, <length>, <list of xxx>, <number>, <paint>, <percentage>, <uri> ou tout autre type utilisé par des attributs ou propriétés animables.

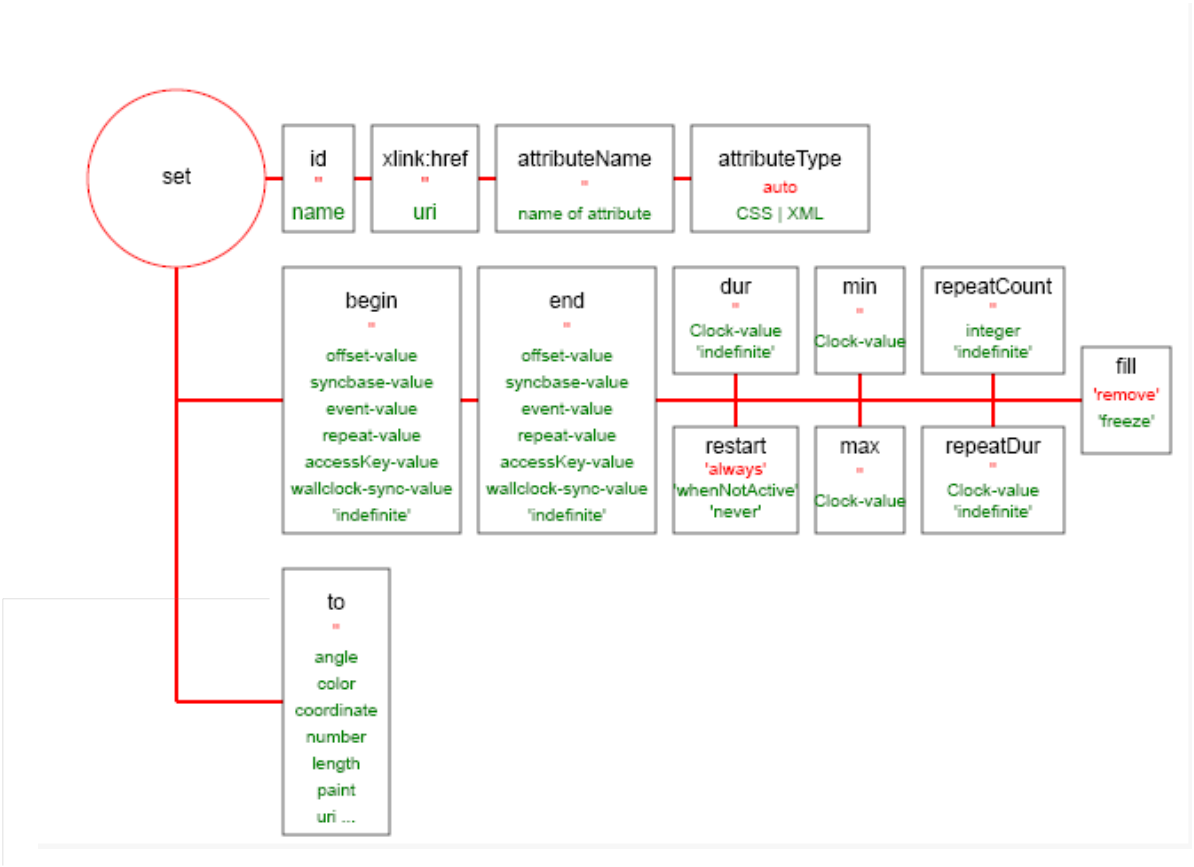

Diagram 9-1. Syntaxe de 'set'

Pour montrer l'intérêt de cet élément, nous créons un liste déroulante où l'utilisateur choisira une couleur pour le rectangle ou un autre objet graphique.

Quand l'utilisateur clique sur ∇ , la liste des couleurs s'affiche, quand le pointeur est sur une couleur, un rectangle grisé signale qu'elle est sélectionnée. Si l'utilisateur clique sur cette couleur, la liste disparaît, le rectangle est coloré. Pour renoncer à choisir une couleur, l'utilisateur peut cliquer en dehors de la liste.

Il n'y a aucun script seulement des éléments 'set'. Voici le code:

```
<svg width="400" height="400">
  <!-- rectangle à colorier -->
  <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">
    Couleur?
  </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"/>
  <!-- rectangle qui provoque l'affichage de la liste -->
  <rect id="go" x="220" y="270" width="15" height="15"
```

```
        style="fill:white;fill-opacity:0"/>
  <!-- fond de la liste, noms des couleurs, rectangles grisés -->
  <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">
     rouge
    </text>
    <rect id="red" x="120" y="285" width="100" height="15"
          style="fill:black;fill-opacity:0">
      <!-- ombre quand la couleur est sélectionnée -->
      <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">jaune</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">
      bleu
    </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>
    <!-- élément set qui affiche/cache la liste -->
    <set attributeName="display" to="inline"
         begin="go.click"
         end="red.click;blue.click;yellow.click;fuchsia.click;bk.click"/>
  </g>
</svg>
```

Figure 9-5. Liste déroulante pour choisir une couleur

L'élément 'animate'

L'élément **'animate'** s'applique à n'importe quel attribut ou propriété qui est animable, c'est à dire toutes celles où la notion d'animation a un sens.

Syntaxe de cet élément:

<**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" />

Tous ces attributs ont été vus précédemment.

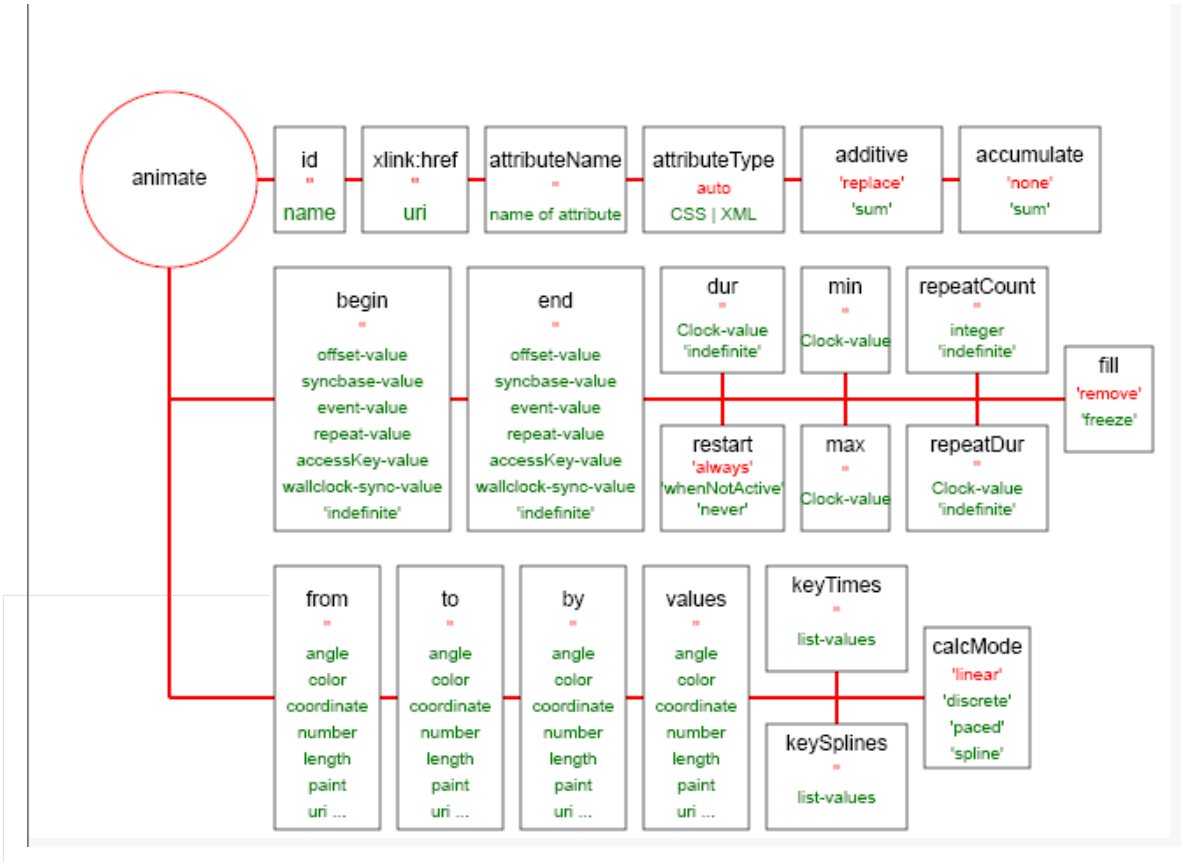

Diagram 9-2. Syntaxe pour 'animate'

Pour les attributs 'values', 'from','to' et 'by', les types de données possibles sont:
<angle>, <color>, <coordinate>, <integer>, <length>, <list of xxx>, <number>, <paint>, <percentage>, <uri> ou tout autre type utilisé par un attribut ou une propriété animable.

Nous avons vu des exemples au chapitre 7 avec 'clipPath' et au chapitre 8 avec les filtres.

Voyons quelques autres exemples :

Tracé progressif en animant l'attribut 'stroke-dashoffset'

Quelques attributs permettent des effets intéressants, c'est le cas de 'stroke-dashoffset' qui permet un tracé progressif d'un objet graphique.

Pour expliquer ceci, prenons un exemple très simple:

```
<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"/>
```

Notre ligne a une longueur de 400 unités (pixels), "stroke-dasharray:400 400" signifie que nous la dessinons en pointillé, mais le pointillé est un peu spécial, nous dessinons sur une longueur de 400 unités, nous ne dessinons plus sur 400 unités et ainsi de suite.
Si nous ajoutons "stroke-dashoffset:400" l'application de ce pointillé commencera 400 unités après le début du tracé, ce qui fait que la ligne n'est pas dessinée.
Si nous changeons la valeur de la propriété 'stroke-dashoffset', nous voyons que si la valeur dépasse 400, la ligne commence à être dessinée à partir de sa fin, elle est dessinée entièrement pour une valeur de 800.
Si la valeur de 'stroke-dashoffset' diminue de 400 à 0, la ligne est dessinée à partir du début, le dessin est complet pour la valeur 0.

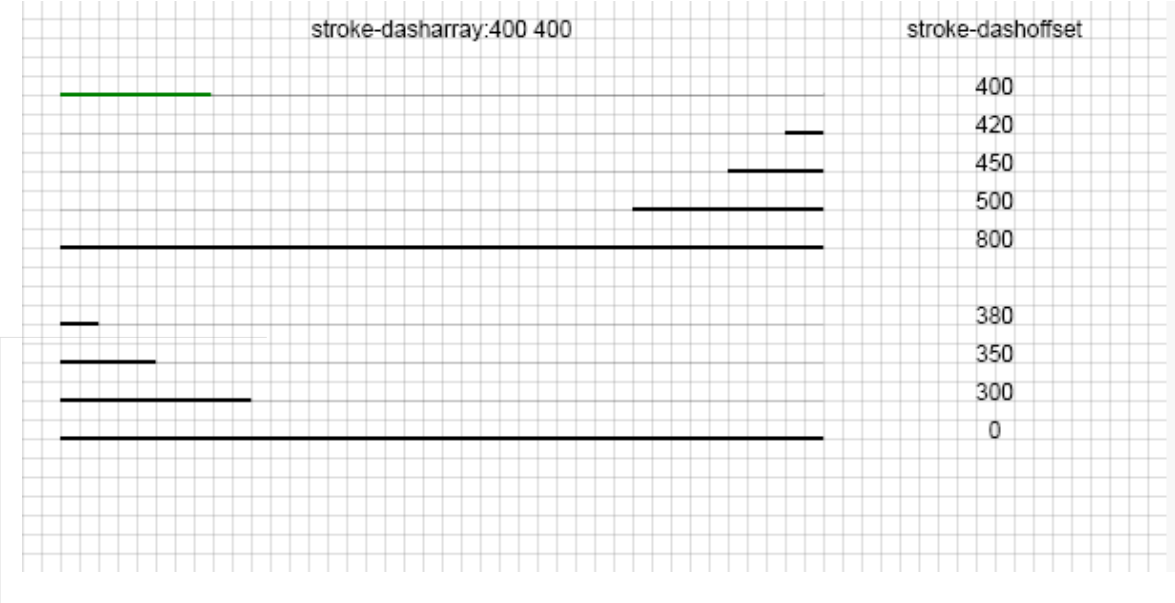

Figure 9-6. Valeurs de 'stroke-dashoffset'

Pour obtenir un tracé progressif de cette ligne à partir de son début à gauche, nous utilisons ce code:

```
<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>
```

Ici, l'élément 'animate' est placé comme descendant de l'élément cible 'path'.
Nous écrivons fill="freeze" pour que le tracé reste visible à la fin de l'animation.

Morphing d'une courbe en animant l'attribut 'd'

Nous pouvons faire évoluer les valeurs de l'attribut 'd' d'un élément 'path', la courbe se déformera. Nous pouvons même avoir un morphing - passage continu d'une forme à la suivante avec quelques conditions:

- Nous n'avons que des commandes 'lineto' et il y en a le même nombre pour toutes les valeurs données à 'd'
- Nous n'avons que des courbes de Bézier cubiques ou quadratiques et en même nombre.

Premier exemple avec les commandes 'lineto':
Le carré devient progressivement une étoile.
Cliquez sur le rectangle "go" pour voir l'animation dans la figure 9-7.

Code pour cet exemple:

```
<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>
```

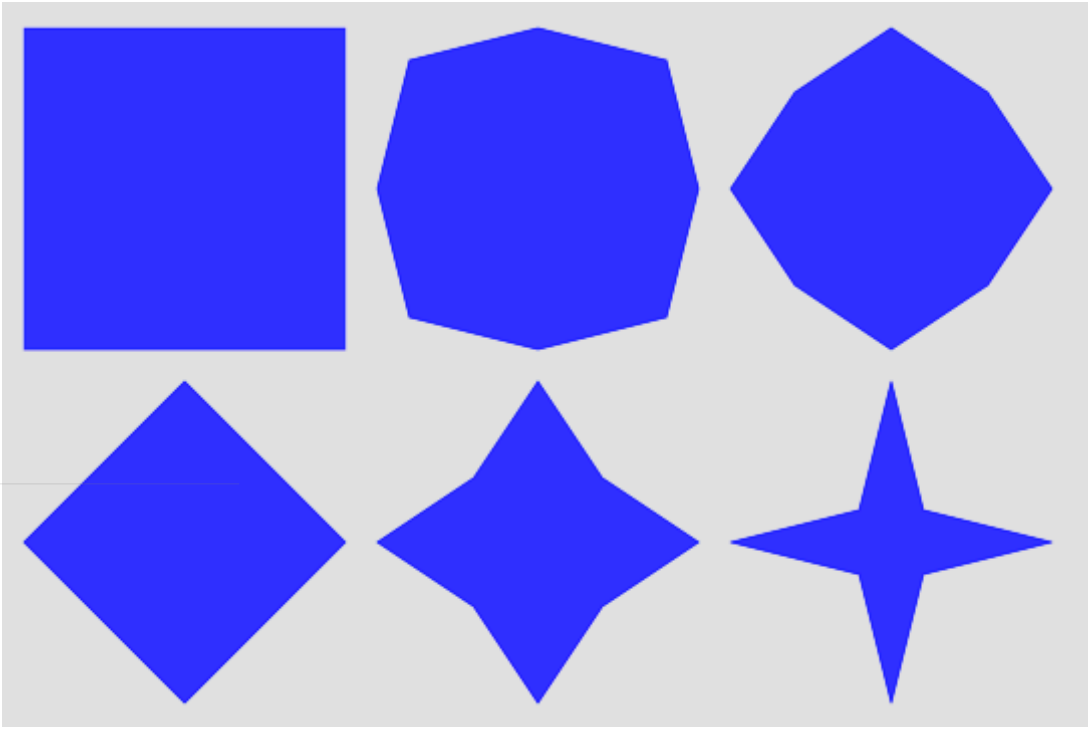

Figure 9-7. Du carré à l'étoile

Second exemple avec deux courbes de Bézier quadratiques. Nous en profitons pour mettre un texte sur la courbe et ce texte suivra les évolutions de la courbe.
Cliquez sur le rectangle "go" pour voir l'animation.

Nous pourrions cacher la courbe en supprimant la ligne

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

et obtenir un texte ondulant du plus bel effet

Code de cet exemple:

```
<svg 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 200 Q200,200 300,200 T500,200;
                      M100 200 Q200,100 300,200 T500,200;
                      M100 200 Q200,200 300,200 T500,200;
                      M100 200 Q200,300 300,200 T500,200;
                      M100 200 Q200,200 300,200 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>
  </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 texte et courbe

Déplacer un texte le long d'une courbe avec 'startOffset'

Profitons de ce texte sur une courbe pour essayer une autre animation. Faisons défiler ce texte le long de la courbe en animant l'attribut 'startOffset', le texte de déplacera le long de la courbe.
Si nous doublons les données de 'd' dans l'élément 'path', le texte disparaîtra sur la droite et réapparaîtra sur la gauche. Nous aurons un texte défilant sur une courbe, ce qui est tout de même moins monotone que ce que nous voyons sur les pages Web.

Cliquez sur le rectangle "go" pour voir l'animation dans la figure 9-9.

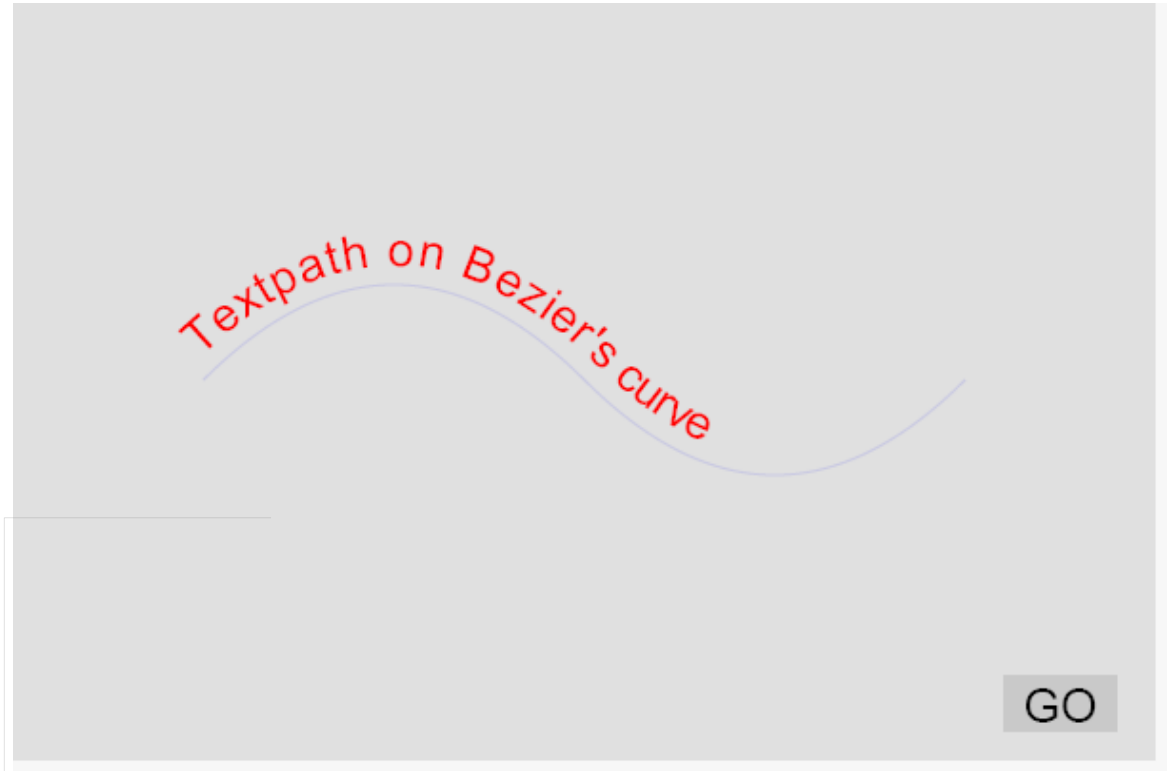

Figure 9-9. Texte défilant le long d'une courbe de Bézier

Code de cet exemple:

```
<svg 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>
```

L'élément 'animateColor'

Nous allons voir des éléments d'animation spécifique pour certains types d'attributs ou de propriéés. Le premier 'animateColor' s'applique uniquement aux couleurs.

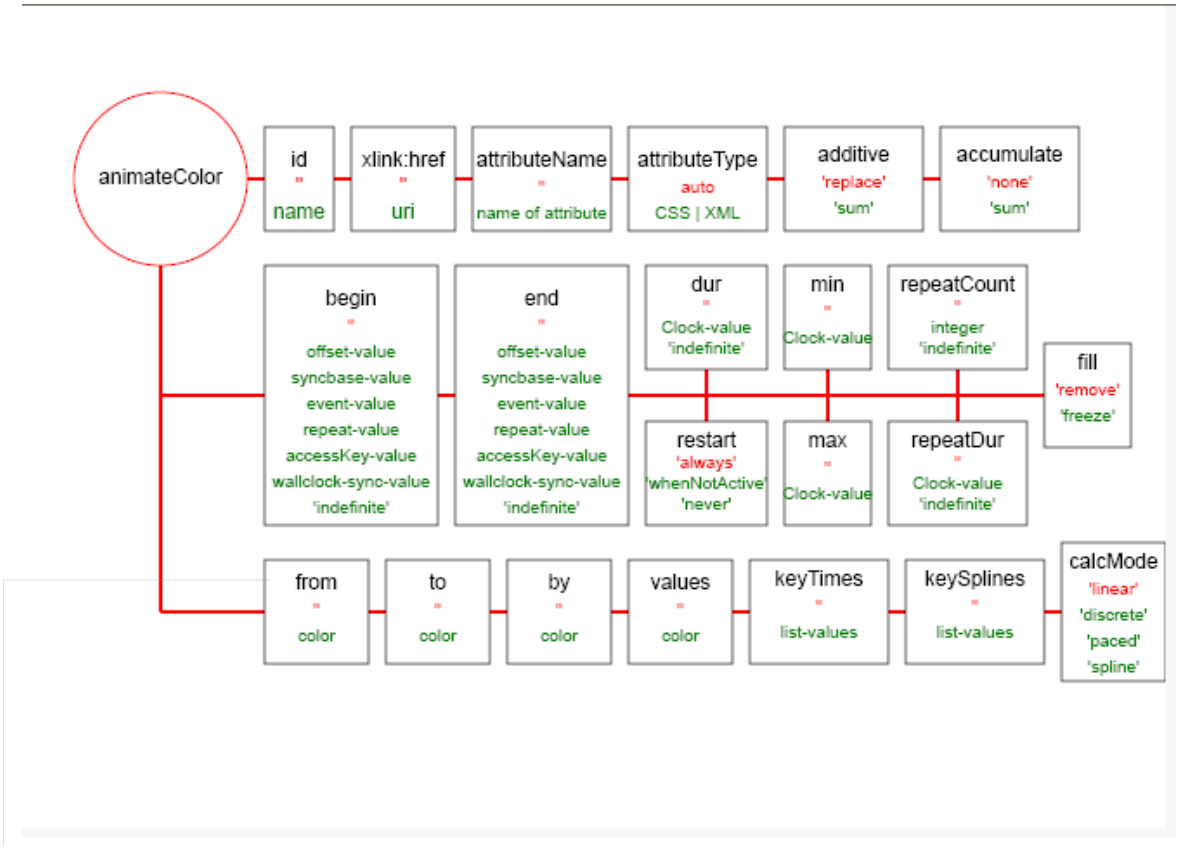

Diagram 9-3. Syntaxe de 'animateColor'

Syntaxe de cet élément :

<**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" />

Tous les attributs ont été expliqués.

Pour les attributs 'values', 'from', 'to' et 'by', le seul type de données accepté est <color>.

Nous pouvons obtenir les mêmes effets avec l'élément 'animate' appliqué à des propriétés comme 'fill' ou 'stroke'.
Dans le paragraphe suivant, nous verrons un exemple utilisant 'animateColor' et 'animateTransform' sur les lettres de SVG.

L'élément 'animateTransform'

L'élément 'animateTransform' s'applique uniquement à l'attribut 'transform' et les valeurs qu'il peut utiliser sont 'translate(tx,ty)', 'rotate(r,cx,cy)', 'scale(sx,sy)', 'skewX(a)' et 'skewY(a)'.

Si nous voulons utiliser les transformations définies par 'matrix(a b c d e f)' nous devons le faire par script.

Syntaxe de cet élément:

<**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" />

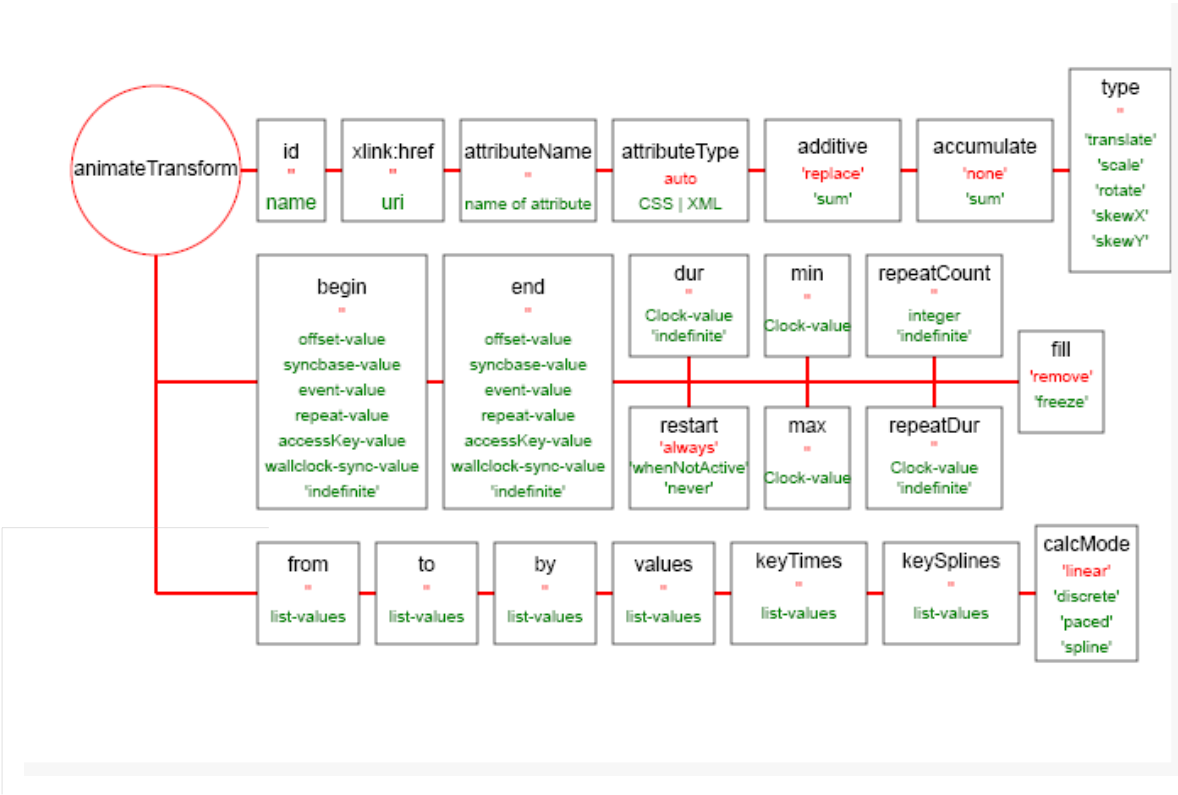

Diagram 9-4. Syntaxe pour 'animateTransform'

Tous ces attributs ont été expliqués sauf 'type'.

L'attribut **'type'** indique quelle est la transformation dont les valeurs seront modifiées durant l'animation.

Les valeurs de 'from' 'to' 'by' et 'values' doivent être:
pour type="translate", de la forme <tx> [,<ty>].
pour type="scale", de la forme <sx> [,<sy>].
pour type="rotate", de la forme <rotate-angle> [<cx> <cy>].
pour type="skewX" ou type="skewY", de la forme <skew-angle>.

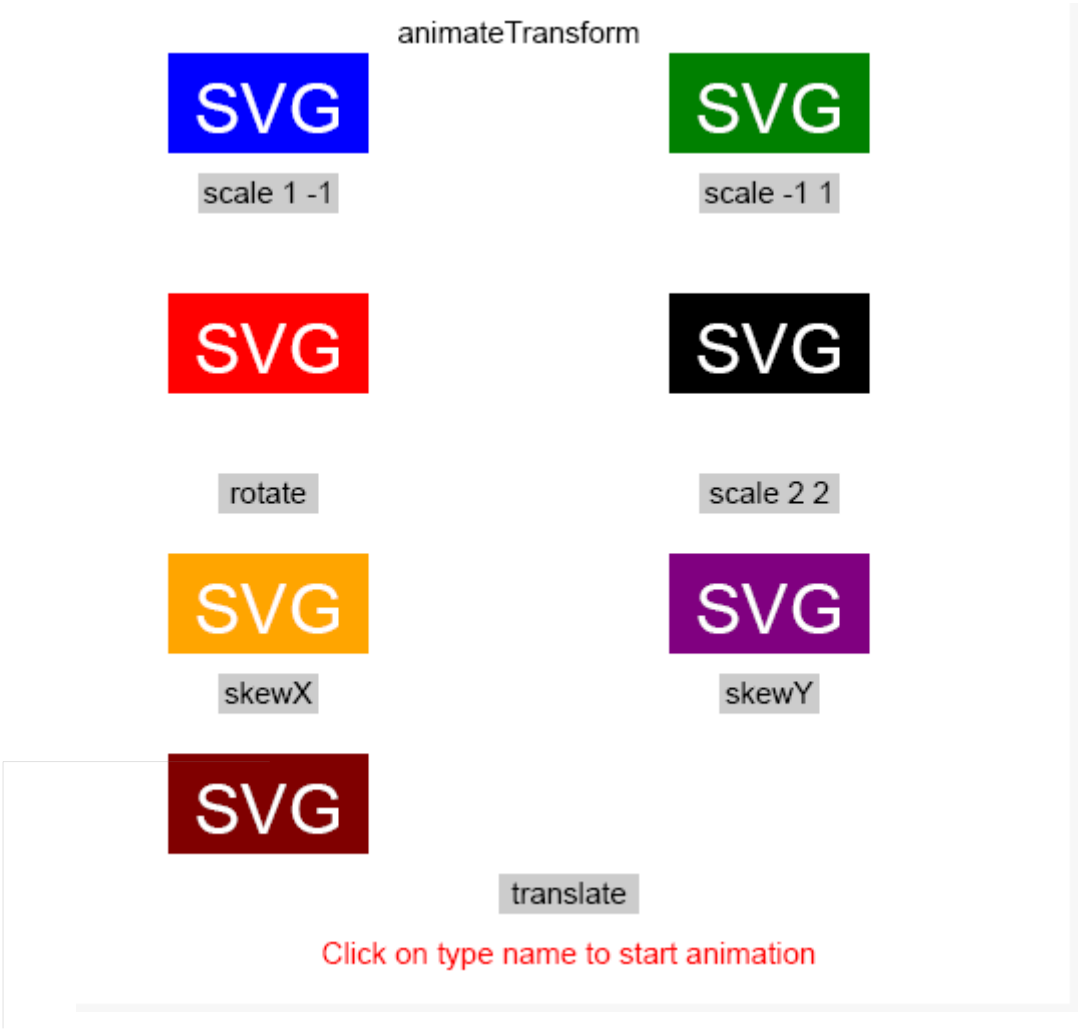

Figure 9-10. 'animateTransform' pour type='scale'

Dans la figure 9-10 cliquez sur le type pour voir l'animation.
Une rotation du texte autour d'un axe vertical, une rotation autour d'un axe horizontal et enfin la taille de l'objet décroît.
Nous utilisons l'élément 'animateTransform' pour faire tourner chacune des lettres de 'SVG' autour d'un axe vertical. Nous ajouterons l'élément 'animateColor' pour que nous ayons l'impression de voir le dessous des lettres

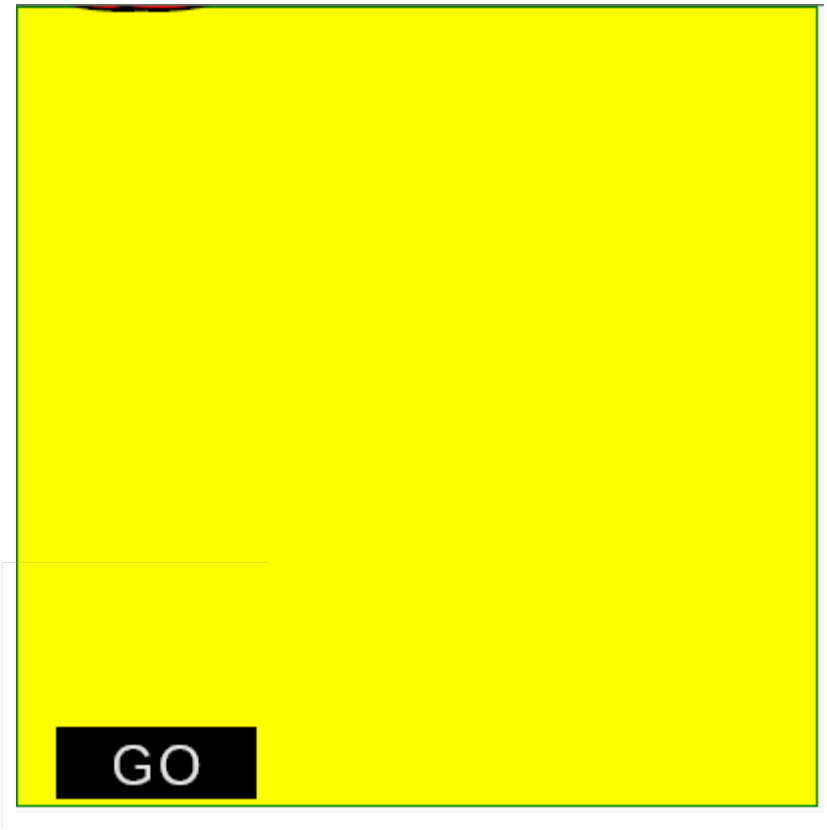

Figure 9-11. Chaque lettre de 'SVG' tourne autour d'un axe vertical

Comme nous utilisons la transformation 'scale' qui est obligatoirement centrée à l'origine du repère, les lettres seront positionnées avec y="0" et x égal à la moitié de la largeur de la lettre. Ces lettres seront ensuite positionnées avec une translation
L'animation est déclenchée par un click sur un bouton (avec "go" comme identificateur - 'id')

Code de cet exemple:

```
<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>
```

Ce code suppose que nous connaissions la largeur des lettres et leur position pour former le mot. Avec un script, nous pouvons connaître toutes ces données et créer les éléments d'animation corrects. Nous verrons cet exemple, automatisé, dans la partie de ce chapitre consacrée à l'utilisation de scripts pour créer une animation.

L'élément 'animateMotion'

L'élément 'animateMotion' provoque le déplacement d'un objet le long d'une courbe définie par 'path'.

Les types d'objets qui peuvent être déplacés sont:
'g' , 'defs' , 'use' , 'image' , 'switch' , 'path' , 'rect' , 'circle' , 'ellipse', 'line', 'polyline', 'polygon' , 'text' , 'clipPath' , 'mask' , 'a' et 'foreignObject'

La syntaxe de l'élément 'animateMotion' :

<**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>

Tous les attributs ont été expliqués sauf :
path : cet attribut définit la trajectoire de l'objet, celle ci doit être exprimée sous la même forme que l'attribut 'd' d'un élément 'path'.

keyPoints : cet attribut est défini par une liste de valeurs entre 0 et 1, séparées par des ';' qui représentent des pourcentages de la longueur de la trajectoire. Associé à 'keyTimes', cet attribut permet un découpage de l'animation - comme 'keyTimes' et 'values' pour les éléments 'animate' et 'animateTransform' - en étapes et l'utilsiation de 'keySplines' pour définir la vitesse du mouvement.

rotate peut prendre les valeurs 'auto', 'auto-reverse' ou <angle>
auto : l'objet est orienté suivant la normale à la trajectoire.
auto-reverse : une rotation de 180° supplémentaire est appliquée.
<angle> : l'objet garde le même angle avec l'axe horizontal. (0 par défaut)

origin : n'a pas d'effet en SVG (conformité aux spécifications SMIL)

Pour définir la trajectoire, nous pouvons utiliser l'élément 'mpath' descendant de l'élément 'animateTransform' qui fera référence à un élément 'path' défini dans le document avec l'attribut 'xlink:href'. Au lieu de

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

Nous pouvons écrire

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

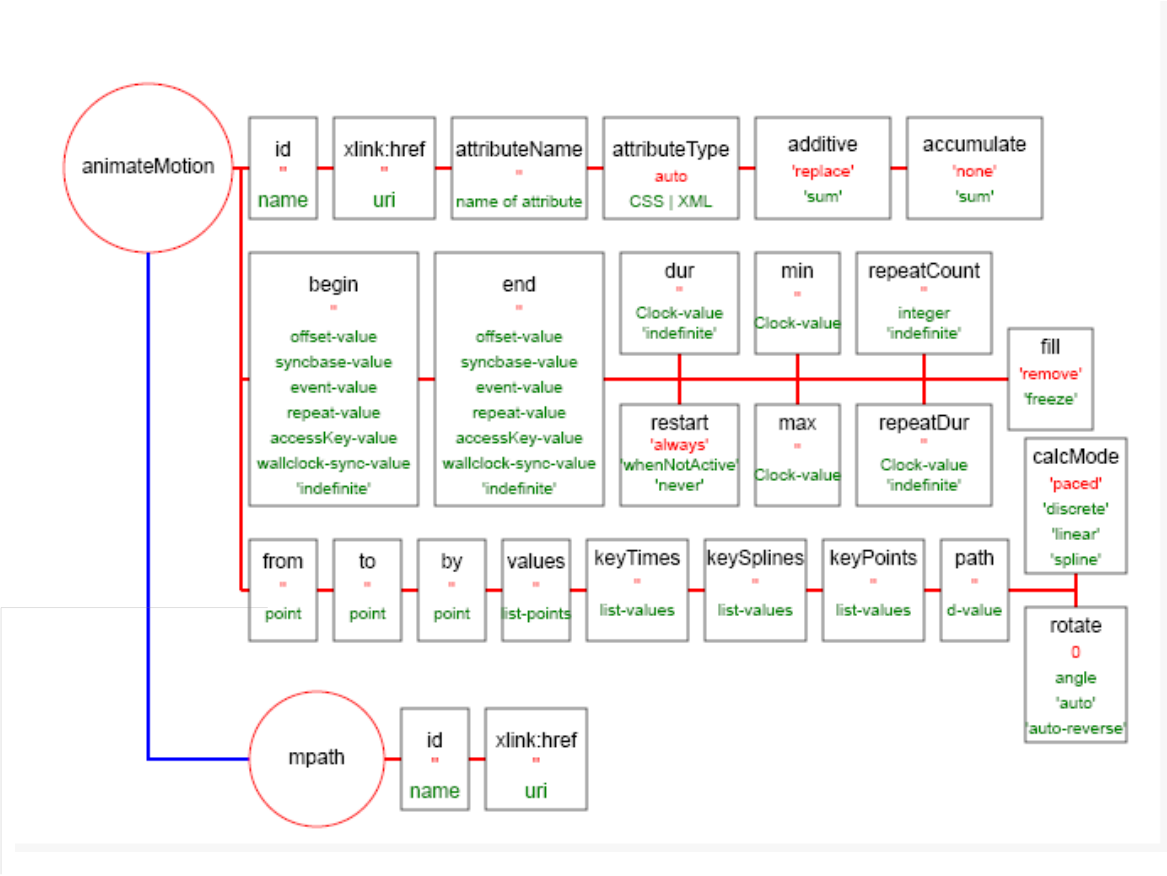

Diagram 9-5. Syntaxe pour 'animateMotion'

Revenons rapidement à 'keyPoints'.
Si nous avons
keyTimes = "0;0.25;0.5;0.7;1" keyPoints = "0;0.5;0.25;0.75;1" dur = "10"

Nous définissons quatre étapes pour l'animation :

- L'objet va du début de la trajectoire au milieu de celle-ci en 2.5 secondes
- L'objet revient en arrière jusqu'au quart de la trajectoire en 2.5 secondes
- L'objet repart jusqu'aux trois quarts de la trajectoire en 2 secondes
- L'objet parcourt le quart restant en 3 secondes

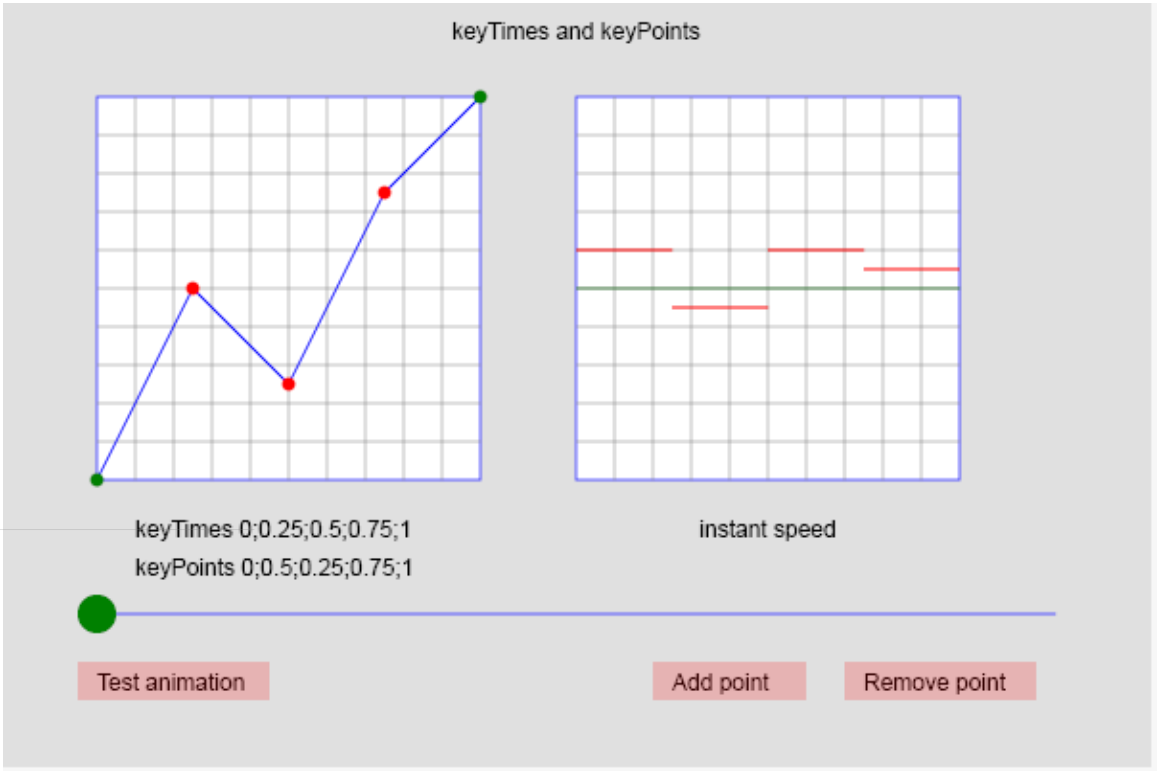

Figure 9-12. 'keyPoints' et 'keyTimes'

Sur la figure 9-12 déplacez les cercles rouges. La vitesse est constante sur chacune des étapes, nous pouvons utiliser 'keySplines' pour que la vitesse soit variable au long de chaque étape.

La figure 9-13 illustre les valeurs de l'attribut 'rotate'.

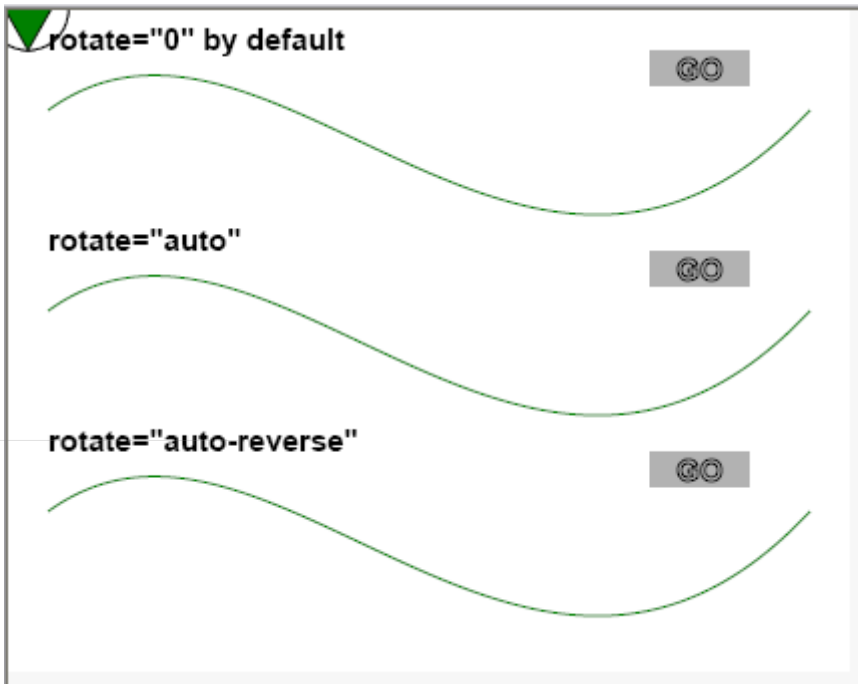

Figure 9-13. Valeurs de l'attribut 'rotate'

Animation et script

Le script peut intervenir à deux niveaux, créer les éléments d'animation et les ajouter au DOM du document ou créer entièrement cette animation en modifiant les attributs et propriétés des objets graphiques

Script pour créer les éléments d'animation

Reprenons l'exemple des lettres de 'SVG' tournant autour d'un axe vertical. Nous pouvons, grâce à un script appliquer cette animation à n'importe quel mot. Le script fera les calculs nécessaires pour définir correctement les éléments d'animation et les ajouter au DOM du document. Nous commençons par créer un élément texte caché avec la chaîne souhaitée.

Pour créer un nœud texte, nous créons le nœud texte avec
node=svgdoc.createElement('text') qui crée l'élément texte
data=svgdoc.createTextNode('tra la la) qui crée la chaîne à afficher qui est ajoutée au nœud comme descendant avec
node.appendChild(data)
le noeud ainsi complet est ajouté au DOM avec
where.appendChild(node)
Dans le chapitre 10, nous détaillons ces manipulations du DOM avec ECMAScript

Pour récupérer la position d'un caractère de la chaîne d'un élément 'text' nous utilisons
texte.getExtentOfChar(i)
renvoie la place occupée par la lettre sous forme d'objet '**SVGRect**' qui a quatre paramètres 'x' et 'y' coordonnées du coin supérieur gauche de l'espace occupé, 'width' et 'height' dimensions de l'espace occupé.
Aussi la $i_{ème}$lettre sera placée pour que x = - texte.getExtentOfChar(i).width / 2

Pour chaque lettre, nous avons deux transformations, 'scale' pour l'animation et 'translate' pour mettre la lettre en place.
Nous utiliserons additive='sum' pour uniquement modifier les paramètres de 'scale' sans toucher à la translation.

Nous ajoutons les éléments d'animation comme descendants du nœud texte.

Code de l'exemple:

```
<svg width='200' height='200' onload="init(evt)">
  <script type="text/ecmascript">
    <![CDATA[
    // chaîne à animer
    var text_data='SVG';
    var svgdoc="";

   // fonction qui crée les éléments d'animation au chargement du SVG

    function init(evt)
    {
    // crée un élément 'text' caché pour récupérer les données

       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");
      data=svgdoc.createTextNode(text_data);
      node.appendChild(data);
      where=svgdoc.getElementById('word');
      where.appendChild(node);

    // extrait chaque lettre du texte

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

      // place chaque lettre avec x égal à - la moitié de la largeur

            place=texte.getExtentOfChar(i);
            node=svgdoc.createElement('text');
            node.setAttribute('x',-place.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('+(place.x+place.width/2)
      +','+(place.y+place.height)+') scale(1 1)');
            data=svgdoc.createTextNode(text_data.substring(i,i+1));
            node.appendChild(data);

     // crée l'élément d'animation pour '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);

    // crée l'élément d'animation 'animateColor'

            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);

// ajoute le noeud au DOM

            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>
```

Après le chargement et l'exécution de la fonction init(evt), les éléments ajoutés au groupe 'word' sont les mêmes que dans notre première version de cet exemple.

Nous pouvons également utiliser la fonction 'parseXML' pour, à partir d'une chaîne, créer les éléments à ajouter au DOM

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

// extrait chaque lettre

      f=objet.getExtentOfChar(i);

// forme la chaîne à parser

      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 la chaîne et l'ajoute au document

      svg_fragment=parseXML(string_to_parse,svgdoc);

  // intégre les éléments parsés au DOM

      where.appendChild(svg_fragment);
      }
```

Nous pouvons ajouter dans la chaîne des séquences "\n" pour marquer des fins de ligne qui nous seront bien utiles pour examiner le code du document chargé (avec "copier SVG" dans le plugin d'Adobe par exemple).

Script pour créer une animation

Toujours le même exemple pour créer l'animation, les attributs de 'transform' seront modifiés directement par le script. Et comme ici nous pouvons le faire, la transformation sera définie par une matrice.

Une nouveauté par rapport au script précédent:

```
setTimeout("anime()", timer_increment)
```

Cette fonction appelle de manière récursive la fonction anime(evt) quand la durée exprimée par timer_increment en millisecondes est écoulée.

Code de cet exemple:

```
<svg width='500' height='200' onload="init(evt)">
```

```
  <script type="text/ecmascript">
    <![CDATA[
      var timevalue = 0;
      var timer_increment = 50; // chaque 50 millisecondes la matrice change
      var max_time = 5000; // durée de l'animation en millisecondes
      var text_data='SVG'; // chaîne à animer
      var svgdoc="";
      var x_letter=new Array;
      var y_letter=new Array;

    // Création des objets nécessaires au chargement du document

      function init(evt)
      {
     // crée le texte - caché - pour retrouver les paramètres
         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);

     // extrait chaque lettre du texte et la dessine indépendamment

         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',
              'matrix(1 0 0 1 '+y_letter[i]+' '+y_letter[i]+')');
              texte=svgdoc.createTextNode(text_data.substring(i,i+1));
              node.appendChild(texte);
              where.appendChild(node)
         }
     };

    // création de l'animation

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

// fin de l'animation quand le temps est écoulé

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

// valeurs pour la matrice

       if (timevalue<=2500)
              {
              taille=(1250-timevalue)/1250
              }
              else
```

```
              {
              taille=(timevalue-3750)/1250
              };
       if (taille==0)
              {
              taille=0.001
              };

// affecte les valeurs à la matrice de transformation de chaque lettre

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

// change la couleur des lettres pour simuler l'envers des lettres

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

// appel récursif à la fonction après l'écoulement de timer_increment millisecondes

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

// déclare la fonction comme un objet 'window'

    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>
<!-- un click sur ce rectangle lance l'animation -->
  <rect onclick="anime(evt)" id="go" x="10" y="180" width="50" height="18"
        opacity="0.1"/>
</svg>
```

Dans le chapitre suivant, nous revenons en détail avec ECMAScript sur les manipulations du DOM (créer, modifier, cloner, supprimer des objets) et nous verrons également des exemples d'animation par script.