

10. Graphiques Vectoriels SVG

10.1 Introduction

10.1.1 Qu'est-ce que SVG ?

SVG (*Scalable Vector Graphics*) est une application **XML** de graphiques vectoriels ayant fait l'objet de plusieurs recommandations du **W3C** :

● **SVG 1.0** (04/09/2001)

● **SVG 1.1** (14/01/2003 - 2^{ème} édition 16/08/11) - une modularisation de **SVG 1.0**,

🔗 W3C SVG Home Page

[<http://www.w3.org/Graphics/SVG/>]

SVG a initialement été promu par **Adobe**, qui mettait gratuitement à disposition un viewer sous forme de plugin pour les navigateurs les plus répandus de l'époque (*Netscape, Internet Explorer, Opera*) tout en commercialisant **Illustrator** permettant de créer des graphiques ou images dans ce format.

Heureusement, les navigateurs ont peu à peu fini par implémenter nativement les fonctionnalités de **SVG**, pour arriver finalement aux spécifications récentes qui intègrent directement les balises **SVG** au vocabulaire **HTML5**.

SVG est compatible **CSS** (i.e. on peut spécifier le style des graphiques grâce à une feuille de style) et scriptable à l'aide de **Javascript**.

10.1.2 Un format graphique pour le Web

> Documents SVG autonomes

Un document **SVG** est avant tout un document **XML** sur lequel il est possible de faire pointer un hyperlien. En ouvrant ce document dans un navigateur, celui-ci affichera le document **SVG** comme il le ferait d'une page **HTML** ou d'une image classique :

```
<a href="tigre.svg">Voir un tigre</a>
```

Voir un tigre :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple1-papier.svg>]

> La balise OBJECT

HTML possède d'autre part la balise **OBJECT**, qui est une balise historique pour l'inclusion de divers types de documents au sein d'une page. Cette balise pouvait être utilisée pour les images, les vidéos ou les sons. Elle s'applique également aux documents **SVG**, avec la syntaxe suivante :

```
<object type="image/svg+xml"
      data="tigre.svg" width="180" height="180" />
```

Remarquer ci-dessus le type mime d'un document **SVG** : **image/svg+xml**.

Tester l'exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple2.html>]

> La balise EMBED

EMBED est une autre balise historique, introduite par **Netscape** et non standard :

```
<EMBED TYPE="image/svg+xml"
      WIDTH="180" HEIGHT="180" SRC="tigre.svg"
      PLUGINSOURCE="http://www.adobe.com/svg/viewer/install/">
```

Tester l'exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple3.html>]

On a pu voir des constructions pour n'utiliser **EMBED** qu'en dernier recours pour les navigateurs qui ne reconnaissent pas **OBJECT** :

```
<object type="image/svg+xml"
      data="tigre.svg" width="180" height="180">
  <embed type="image/svg+xml"
        width="180" height="180" src="tigre.svg"
        pluginspage="http://www.adobe.com/svg/viewer/install/">
</object>
```

> La balise IMG

La possibilité la plus propre, implémentée le plus tardivement par les navigateurs, consiste à intégrer un graphique **SVG** directement dans une page **HTML** à l'aide de la classique balise **IMG**, dont l'attribut **SRC** pointe vers un document **SVG**.

```

```

Tester l'exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple4.html>]

> Les espaces de noms XML

Moyennant une gestion correcte des espaces de noms **XML**, il est possible d'intégrer directement du code source **SVG** au sein d'un document **XML** quelconque. Cette remarque s'applique bien entendu aux documents **XHTML**.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<h1>Scalable Vector Graphics</h1>
...
<svg:svg xmlns:svg="http://www.w3.org/2000/svg" ... >
  <svg:rect x="-600" y="-600" width="1800" ... />
  ...
</svg:svg>
<hr class="bottom"/>
...
</body>
</html>
```

Tester l'exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple5.xml>]

> HTML5

A l'heure actuelle, la méthode la plus simple consiste encore à faire confiance à **HTML5**, dont les spécifications intègrent nativement les balises **SVG**.

```
<!DOCTYPE html>
<h1>Scalable Vector Graphics</h1>
<hr class="top"/>
<svg viewBox="0 0 600 600" width="360" height="360" style="margin: auto;">
  <rect x="-600" y="-600" width="1800" height="1800" stroke="none"
  fill="#f0f0ea"/>
  <g transform="translate(200,200)" style="fill-opacity:1; fill:none;">
    ...
  </g>
</svg>
<hr class="bottom"/>
<address>JohnDoe@mycompany.com</address>
```

Tester l'exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/intro/slide2-exemple6.html>]

10.2 Principes généraux

10.2.1 SVG : une application XML

SVG est avant tout une application **XML**. A ce titre, un certain nombre d'éléments décrits ci-dessous peuvent être nécessaires.

> Déclaration XML

La déclaration **XML** est en toute rigueur optionnelle vis à vis des spécifications. Toutefois, elle peut être utile pour préciser le codage utilisé :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

Dans le cas d'un document **HTML5** contenant un ou plusieurs graphiques **SVG**, la déclaration **XML** est uniquement présente pour le document global, lorsque celui-ci est sérialisé en **XHTML5**.

> Déclaration de type de document

La déclaration de type de document indique la **DTD** (*Définition de Type de Document*) à laquelle se conforme le document. La déclaration ci-dessous correspond à un document **SVG 1.0** (noter les identificateurs **PUBLIC** et **SYSTEM**) :

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

N.B. Dans le cas d'un document **HTML5** contenant un ou plusieurs graphiques **SVG**, la déclaration de type de document serait différente :

```
<!DOCTYPE html>
```

> Élément racine

Comme l'indique la déclaration de type de document de **SVG** vue ci-dessus, l'élément racine d'un document **SVG** est **svg**.

Vis-à-vis de **XML**, **SVG** possède son propre espace de nom, qu'il est opportun de mentionner au niveau de l'élément racine :

```
<svg xmlns="http://www.w3.org/2000/svg">
...
</svg>
```

N.B. L'attribut **xmlns** ne doit pas être utilisé lorsque la balise **svg** est utilisée dans un document **HTML5**.

> Exemple complet

La structure générale d'un document **SVG** autonome est conforme à :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
...
</svg>
```

Vers un exemple complet :

<http://dmolinarius.github.io/demofiles/mod-84/svg/generalites/slide1-exemple1.html>

10.2.2 Echelles et coordonnées

> Coordonnées utilisateur

En théorie, **SVG** permet de représenter des éléments sur un plan de coordonnées allant de moins l'infini à plus l'infini suivant les deux axes **X** et **Y**.

En pratique, il est plus commode de préciser une fenêtre de coordonnées (*viewBox*) à travers laquelle on désire contempler le document. Cette fenêtre définit ce qu'il est convenu d'appeler le **système de coordonnées utilisateur** à l'aide de l'attribut **viewBox** de l'élément racine **svg**.

```
<svg viewBox="-500 -500 1000 1000">
...
</svg>
```

N.B. l'axe vertical **Y** est compté positivement de haut en bas. La syntaxe de l'attribut **viewBox** est :

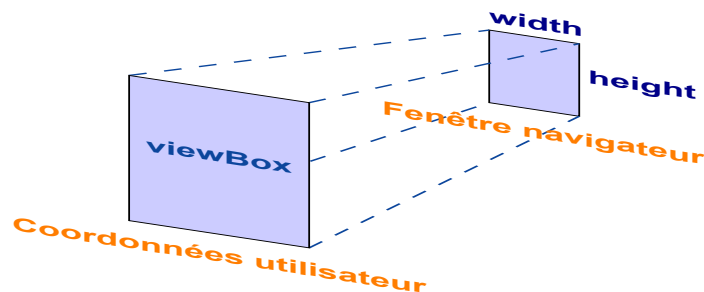
```
viewBox = "x_min y_min largeur hauteur"
```

Exemples :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/generalites/slide2-exemple1.html>]

> Préconisations d'échelle

Le passage des coordonnées utilisateur à un système de coordonnées réelles (*pixels*, *cm*, ...) est effectué par le dispositif de visualisation (*navigateur*, *convertisseur vers format papier*) en faisant coïncider la zone utilisateur avec l'espace disponible pour la représentation du graphique :



Les dimensions préconisées pour l'affichage peuvent être indiquées au sein du document lui-même :

```
<svg viewBox="-500 -500 1000 1000" width="400" height="400">
...
</svg>
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/generalites/slide2-exemple2.html>]

> Dimensions de la zone d'affichage

Toutefois, l'expérience prouve qu'il vaut mieux préciser les dimensions souhaitées du graphique au sein du document appelant, par exemple grâce aux attributs **width** et **height** de la balise **OBJECT** :

```
<object width="180" height="180"
        type="image/svg+xml" data="slide2-exemple3-1.svg">
</object>
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/generalites/slide2-exemple3.html>]

Remarque : Il n'est pas interdit de tenter d'afficher un document **SVG** en modifiant le rapport hauteur / largeur. Toutefois, le résultat obtenu peut dépendre du document lui-même, qui doit en principe explicitement autoriser un tel comportement :

```
<svg xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 180 180" preserveAspectRatio="none">
...
</svg>
```

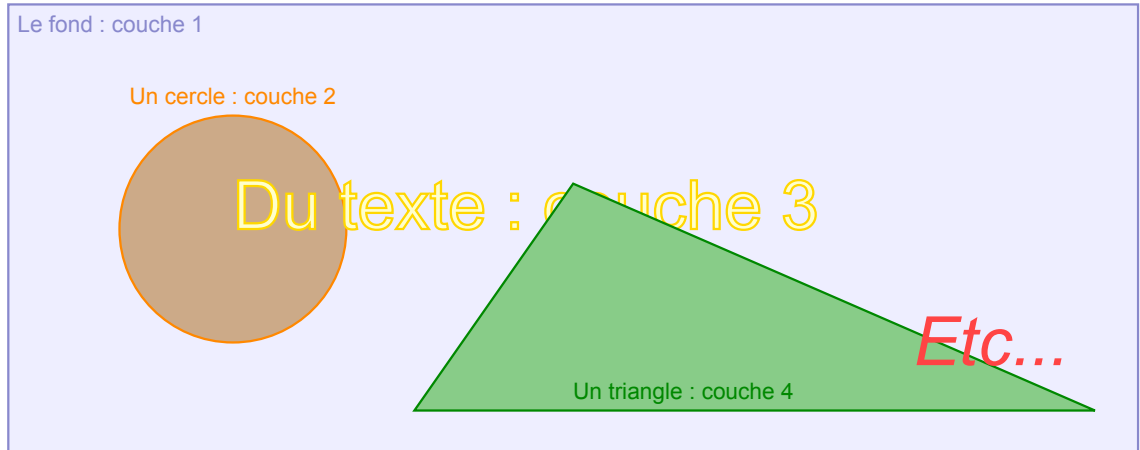
Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/generalites/slide2-exemple4.html>]

10.2.3 Troisième dimension

> L'algorithme du peintre

Pour gérer la troisième dimension (quel est l'objet représenté à l'avant-plan lorsque plusieurs éléments se trouvent au même endroit ?) **SVG** utilise l'algorithme du peintre: les éléments recouvrent les précédents, au fur et à mesure de leur tracé :



Le code **SVG** : (noter surtout l'ordre des éléments...)

```
<svg viewBox="0 0 500 200">
  <!-- le fond -->
  <rect x="1" y="1" width="498" height="198" ... />
  <!-- le cercle -->
  <circle cx="100" cy="100" r="50" ... />
  <!-- le texte -->
  <text x="100" y="100">Du texte : couche 3</text>
  <!-- le triangle -->
  <polygon points="180,180 250,80 480,180" .../>
  <!-- etc... -->
  <text x="400" y="160">Etc...</text>
</svg>
```

10.3 Formes de base

10.3.1 Attributs communs aux formes graphiques

L'ensemble des formes graphiques de **SVG** texte compris, partagent de nombreux attributs concernant leur représentation graphique, dont certains sont décrits ci-après.

> stroke

L'attribut **stroke** décrit la couleur d'un trait. Les valeurs possibles les plus couramment rencontrées sont **none** (pas de tracé), le nom d'une couleur prédéfinie, ou la valeur d'une couleur dans l'un des modes **CSS 2** :

```
<rect stroke="none" ... /> <!-- bords non tracés -->
<rect stroke="ivory" ... /> <!-- couleur prédéfinie -->
<rect stroke="#F00" ... /> <!-- #rgb -->
<rect stroke="#008800" ... /> <!-- #rrggbb -->
<rect stroke="rgb(255,127,0)" ... /> <!-- rgb(r,g,b) -->
<rect stroke="rgb(100%,50%,0%)" ... /> <!-- rgb(r%,g%,b%) -->
```

Voir les couleurs prédéfinies :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/couleurs.svg>]

La valeur par défaut de l'attribut **stroke** (si non spécifié) est **none**.

> stroke-width

La valeur de l'attribut **stroke-width** détermine la largeur du trait. Cette valeur peut être dépourvue d'unité (*coordonnées utilisateur, cf. viewBox*), être exprimée en % (*de la largeur de la fenêtre viewBox*) ou utiliser l'une des unités reconnues par **CSS 2** (*px = coords utilisateur, em, ex, pc, cm, mm ...*).

```
<line stroke-width="2" ... >
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/attr-stroke-width-papier.svg>]

La valeur par défaut de l'attribut **stroke-width** (si non spécifié) est **1**.

> stroke-opacity

Exprimé à l'aide d'une valeur comprise entre **0.0** (*ligne totalement invisible*) et **1.0** (*ligne totalement visible*) cet attribut contrôle la transparence du tracé.

```
<line stroke-opacity="0.5" ... >
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/attr-stroke-opacity-papier.svg>]

La valeur par défaut de l'attribut **stroke-opacity** (si non spécifié) est **1.0**.

> stroke-dasharray

Avec **SVG** il est possible de concevoir le pointillé de ses rêves... L'attribut **stroke-dasharray** peut prendre la valeur **none** (*ligne continue*) ou une liste de longueurs séparées par des virgules, donnant le dessin du pointillé :

```
<line stroke-dasharray="1,1,1,1,1,3" ...>
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/attr-stroke-dasharray-papier.svg>]

La valeur par défaut de l'attribut **stroke-dasharray** (si non spécifié) est **none**.

> fill

L'attribut **fill** décrit la couleur du remplissage d'une surface. Là encore, les valeurs les plus courantes sont **none** (*pas de tracé*), le nom d'une couleur prédéfinie, ou la valeur d'une couleur dans l'un des modes **CSS 2** :

```
<rect fill="orange" .../>
```

Revoir les couleurs prédéfinies ? :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/couleurs.svg>]

La valeur par défaut de l'attribut **fill** (si non spécifié) est **black**.

> fill-opacity

Comme **stroke-opacity**, cet attribut contrôle la transparence du remplissage, depuis un tracé totalement visible (*valeur 1.0*), jusqu'à un remplissage totalement transparent (*valeur 0.0*).

```
<rect fill-opacity="0.5" ... >
```

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/attr-fill-opacity-papier.svg>]

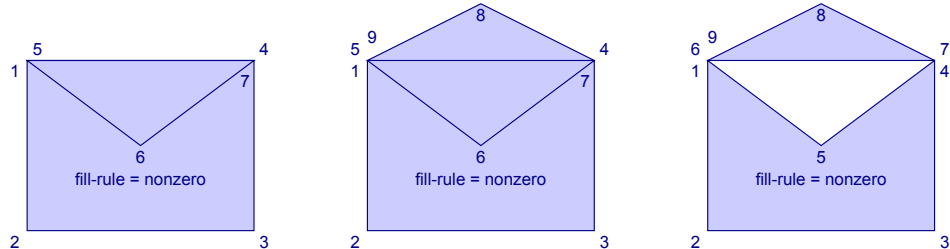
La valeur par défaut de l'attribut **fill-opacity** (si non spécifié) est **1.0**.

> fill-rule

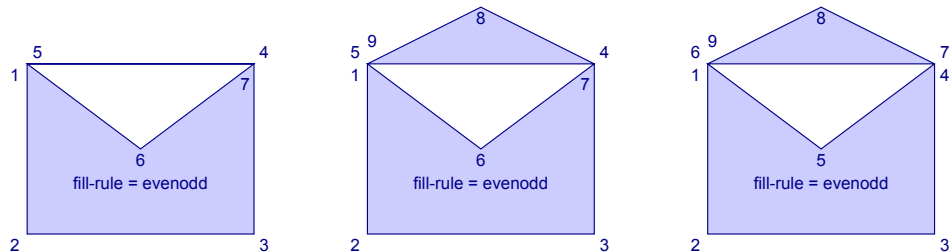
L'attribut **fill-rule** permet de spécifier comment se comporte le remplissage dans le cas de surfaces comportant des trous ou dont les bords présentent des intersections. Les valeurs autorisées sont **nonzero** ou **evenodd** en fonction de l'algorithme de remplissage désiré.

```
<polygon points="..." fill-rule="evenodd"/>
```

L'algorithme **nonzero** cherche à déterminer la position d'un point donné en lançant un rayon depuis ce point vers l'infini. Le long de ce rayon et partant de zéro, on compte +1 chaque fois qu'on croise un bord orienté de la gauche vers la droite, -1 pour un bord orienté de la droite vers la gauche. Le point est à l'intérieur (*donc coloré*) si le résultat final est non nul :



L'algorithme **evenodd** procède un peu de même, en comptant simplement le nombre d'intersections entre le rayon et les bords de la surface. Si ce nombre est impair le point est à l'intérieur (*i.e. coloré*), sinon il est à l'extérieur :



La valeur par défaut de l'attribut **fill-rule** (si non spécifié) est **nonzero**.

10.3.2 Segments et lignes

> line

line est la forme de base la plus simple.

Cet élément correspond à un simple segment, dont il suffit de donner les coordonnées à l'aide des attributs **x1**, **y1** (point de départ), et **x2**, **y2** (point d'arrivée) :

```
<line x1="10" y1="10" x2="100" y2="100" />
```

Remarque : en cas d'attribut non spécifié la valeur par défaut est : **0.0**

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur et le style de trait s'appliquent à l'élément **line**.

Exemple de figures à base de segments :



Un motif de la frise ci-dessus est ainsi tracé à l'aide des éléments suivants :

```
<line x1="0" y1="130" x2="0" y2="20"/>
<line x1="0" y1="20" x2="120" y2="20"/>
<line x1="120" y1="20" x2="120" y2="102.5"/>
<line x1="120" y1="102.5" x2="60" y2="102.5"/>
<line x1="60" y1="102.5" x2="60" y2="75"/>
<line x1="60" y1="75" x2="90" y2="75"/>
<line x1="90" y1="75" x2="90" y2="47.5"/>
<line x1="90" y1="47.5" x2="30" y2="47.5"/>
<line x1="30" y1="47.5" x2="30" y2="130"/>
<line x1="30" y1="130" x2="150" y2="130"/>
```

> polyline

Pour la représentation de graphiques du type de celui ci-dessus, il est évidemment assez fastidieux (*et volumineux en terme de taille de fichier*) de répéter deux fois les coordonnées de chacun des points (*point d'arrivée du segment n, et point de départ du segment n+1*).

L'élément **polyline** permet de représenter une ligne continue constituée de segments jointifs consécutifs :

```
<polyline points="0,130 0,20 120,20 120,102.5 ..."/>
```

Le motif élémentaire de la frise ci-dessus, se verra donc représenté par l'extrait de code suivant, qui est éventuellement un peu moins lisible, mais indéniablement plus concis !

```
<polyline points="0,130 0,20 120,20 120,102.5
60,102.5 60,75 90,75 90,47.5
30,47.5 30,130 150,130"/>
```

La frise avec polyline :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/formes/line-polyline.svg>]

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur et le style de trait, ainsi que ceux relatifs au remplissage s'appliquent à l'élément **polyline**.

N.B. Pour remplir une forme tracée avec **polyline**, **SVG** ferme le tracé avec un segment virtuel joignant le dernier point au premier...

10.3.3 Rectangles

> rect

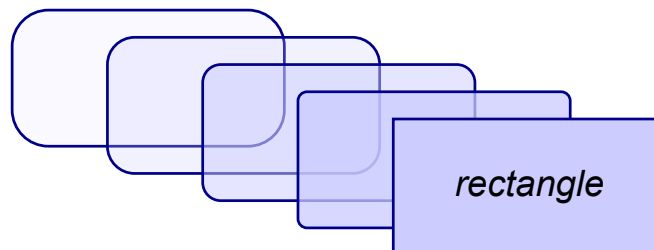
L'élément **rect** correspond à un rectangle. Il est décrit par les coordonnées de son coin haut gauche (*x min, ymin*) et par ses dimensions (*longueur et largeur*) :

```
<rect x="0" y="0" width="300" height="100"/>
```

La valeur par défaut des coordonnées **x** et **y** (*si non spécifiées*) est **0**. Si l'une des dimensions n'est pas spécifiée, le rectangle n'est pas tracé.

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur, le style de trait et le remplissage s'appliquent à l'élément **rect**.

Exemple de rectangles :



Ainsi qu'illustré par l'exemple ci-dessus, le rectangle **SVG** permet d'arrondir les angles !

Les attributs **rx** et **ry** donnent respectivement le rayon horizontal et le rayon vertical des portions d'ellipse servant à arrondir les angles :

```
<rect x="50" rx="40" ry="30" width="300" height="150"/>
```

Si aucun des attributs **rx** et **ry** n'est spécifié, leur valeur par défaut est **0** (*coins non arrondis*). Si l'un seulement est donné, l'autre est supposé avoir la même valeur.

La valeur de ces attributs ne peut être supérieure à la demi-dimension correspondante du rectangle, si c'est le cas elle sera limitée à cette valeur.

10.3.4 Polygones

> polygon

L'élément **polygon** représente un polygone. Un polygone n'est autre qu'une polyligne fermée.

Si la distinction **polygon** / **polyline** n'est pas significative pour ce qui concerne le remplissage, elle l'est pour le tracé des contours : un polygone est toujours fermé.

Pour le reste, les attributs d'un polygone sont les mêmes que ceux d'une polyligne :

```
<polygon points="0,130 0,20 120,20 120,102.5 ..."/>
```

Exemples de polygones :



10.3.5 Cercles

> circle

L'élément **circle** représente un cercle.

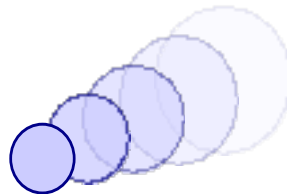
Un cercle est classiquement donné par les coordonnées de son centre **cx** et **cy**, ainsi que par la valeur de son rayon **r** :

```
<circle cx="150" cy="150" r="100"/>
```

La valeur par défaut des coordonnées **cx** et **cy** (si non spécifiées) est 0. Si le rayon **r** n'est pas spécifié, le cercle n'est pas tracé.

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur, le style de trait et le remplissage s'appliquent à l'élément **circle**.

Exemples de cercles :



Attention ! Le cercle est tracé dans le système de coordonnées utilisateur (cf. attribut *viewBox* de l'élément *racine svg*). Si le rapport hauteur / largeur n'est pas conservé pour la représentation finale, alors le cercle apparaîtra comme une ellipse...

10.3.6 Ellipse

> ellipse

L'élément **ellipse** représente une ellipse.

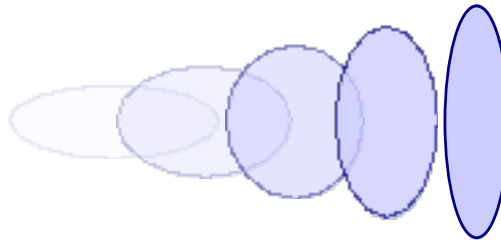
Une ellipse est donnée par les coordonnées **cx,cy** de son centre et par les rayons **rx** et **ry**.

```
<ellipse cx="150" cy="150" rx="100" ry="50"/>
```

La valeur par défaut des coordonnées **cx** et **cy** (si non spécifiées) est 0. Si l'un des rayons **rx** ou **ry** n'est pas spécifié, l'ellipse n'est pas tracée.

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur, le style de trait et le remplissage s'appliquent à l'élément **ellipse**.

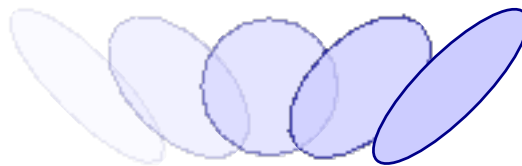
Exemples d'ellipses :



Remarque : Vue la manière dont elle spécifiée, en première approche les axes de l'ellipse ne peuvent évidemment qu'être horizontaux et verticaux.

SVG possède néanmoins des **transformations** (*vues plus loin*) qui permettront de faire tourner les ellipses...

Exemples d'ellipses tournées :



10.4 Texte

10.4.1 Élément texte

> text

L'élément **text** permet d'intégrer du texte aux graphiques **SVG**.

Les attributs permettant de positionner le texte sont ses coordonnées **x** et **y**. Le texte lui-même correspond au contenu de l'élément :

```
<text x="10" y="20">Un exemple de texte</text>
```

La valeur par défaut des coordonnées **x** et **y** (*si non spécifiées*) est **0**.

Les attributs communs [cf. paragraphe 10.3.1] aux formes graphiques concernant la couleur, le style de trait et le remplissage s'appliquent à l'élément **text** :

```
<text y="21" x="10">Un exemple de texte</text>
<text y="21" x="170" fill="#CCF" stroke-width="0.7" stroke="#008">SVG</text>
```

Un exemple de texte SVG

10.4.2 Polices de caractères

Les propriétés relevant de la description de la police de caractères sont à rapprocher de celles vues avec **CSS 2**.

> font-family

L'attribut **font-family** permet de spécifier la police à utiliser, à l'aide d'un nom (ou d'une liste de noms) de police générique (*serif, sans-serif, cursive, fantasy, monospace*) ou de police spécifique à la plateforme du client.

La valeur par défaut (*attribut non spécifié*) dépend du client.

:

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-font-family-generic-papier.svg>]

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-font-family-specific-papier.svg>]

> font-size

L'attribut **font-size** donne la taille de la police.

La valeur est une taille absolue à prendre dans une table gérée par le client (*xx-small, x-small, small, medium, large, x-large, xx-large*), une taille relative à celle héritée de l'élément parent (*smaller, larger*), une longueur (avec unité, *px = coordonnées utilisateur*), ou un pourcentage de la taille héritée de l'élément parent.

La valeur par défaut (*attribut non spécifié*) est **medium**.

> font-weight

Cet attribut permet de spécifier la graisse de la police.

Comme en **CSS 2** les valeurs possibles de l'attribut **font-weight** sont : **normal, bold, bolder, lighter** ou une valeur parmi **100, 200, ... 900**.

La valeur par défaut (*attribut non spécifié*) est **normal**.

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-font-weight-papier.svg>]

> font-style

L'attribut **font-style** indique s'il s'agit d'une police italique.

Les valeurs possibles sont **normal, italic** ou **oblique**.

La valeur par défaut (*attribut non spécifié*) est **normal**.

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-font-style-papier.svg>]

> font-variant

L'attribut **font-variant** possède deux valeurs : **normal** et **small-caps**.

La valeur par défaut (*attribut non spécifié*) est **normal**.

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-font-variant-papier.svg>]

> text-decoration

L'attribut **text-decoration** prend les valeurs suivantes : **none, underline, overline, line-through** ou **blink**.

La valeur par défaut (*attribut non spécifié*) est **none**.

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-text-decoration-papier.svg>]

> letter-spacing

L'espace entre les caractères peut être réglé grâce à l'attribut **letter-spacing**. Les valeurs possibles sont : **normal** ou une longueur munie d'une unité ou sans unité (*coordonnées utilisateur*)

La valeur par défaut (*attribut non spécifié*) est **normal**.

Exemple :

[<http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-letter-spacing-papier.svg>]

> word-spacing

L'espace entre les mots peut être réglé indépendamment du précédent à l'aide de l'attribut **word-spacing**. Les valeurs sont **normal** ou une longueur.

La valeur par défaut (*attribut non spécifié*) est **normal**.

Exemple :

[\[http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-word-spacing-papier.svg\]](http://dmolinarius.github.io/demofiles/mod-84/svg/texte/attr-word-spacing-papier.svg)

10.4.3 Attributs de l'élément texte

> x,y

Outre le fait de préciser la position de la chaîne de caractères, les attributs **x** et **y** peuvent être employés sous forme de liste donnant la position individuelle de chacun des caractères de la chaîne :

```
<text x="0,5.26,10.52,15.78,21.05, ..."
      y="0,4.75, 8.37, 9.96, 9.15, ...">
  Ceci est un exemple de texte chahuté !
</text>
```

Exemple :

Ceci est un exemple de texte chahuté !

> dx,dy

Plutôt que de donner une liste de positions absolues pour les caractères, il est possible de spécifier plutôt une position relative, par rapport à la position normale des caractères. Cette opération s'effectue à l'aide des attributs **dx** et **dy** :

```
<text dx="0, 0, -3, -3, 0, 0, 0, 0, 0, -3, 0"
      dy="0, 0, 4, -7, 3, 0, 0, 0, 0, 4, -4">
  axo2 + bxo + c = 0
</text>
```

Le texte correspondant à l'exemple ci-dessus :

$$ax_o^2 + bx_o + c = 0$$

N.B. Une fois le texte déplacé (*i.e. dx ou dy non nul*), la chaîne de caractères repart de la nouvelle position. Si l'effet recherché est de mettre comme ci-dessus un caractère en indice ou en exposant, il est nécessaire de refaire un décalage de signe opposé au caractère suivant pour revenir sur la ligne normale...

> rotate

L'élément **text** admet un autre attribut appelé **rotate** permettant de spécifier l'angle des caractères :

```
<text x="0" y="0" rotate="20">
  Autant en emporte le vent !
</text>
```

Autant en emporte le vent !

Comme les attributs précédents, **rotate** accepte également une liste s'appliquant séquentiellement à chacun des caractères de la chaîne.

Pour réorienter l'ensemble des caractères de la chaîne avec éventuellement un angle différent pour chacun d'eux, on peut écrire :

```
<text x="0" y="0" rotate="20, 20, 20, 20, 20, 20, 20, 20, 20 ...">
  Autant en emporte le vent !
</text>
```

Résultat :

Autant en emporte le vent !

N.B. Il est bien entendu possible de compléter l'attribut **rotate** avec les effets obtenus à l'aide des listes de coordonnées vues plus haut, afin d'obtenir du texte dont la position et l'angle varient de caractère en caractère.

10.4.4 Sous-chaînes

> tspan

Il existe de nombreux cas où l'on désire pouvoir affecter des caractéristiques particulières à une sous-chaîne d'un texte (*changement de police, couleur, indice, exposant...*).

L'obtention d'un tel résultat uniquement basé sur l'élément **text** serait difficile, car elle nécessiterait à chaque fois de calculer très exactement la position de départ (*attribut x*) de chaque nouvelle sous-chaîne.

Afin d'éviter ceci il existe un élément nommé **tspan** qui, apparaissant comme sous-élément d'un élément **text** ou d'un autre élément **tspan**, garde le compte de la position du caractère suivant :

```
<text>
  Du texte
  <tspan fill="#000088">en couleur</tspan>
  dans une phrase...
</text>
```

Exemple :

Du texte en couleur dans une phrase...

L'élément **tspan** est très utile pour la gestion du texte sur plusieurs lignes :

L'enfance de Zéphyrin.

C'est le 1er janvier, à minuit une seconde sexagésimale de temps moyen, que le jeune Brioché poussa ses premiers vagissements. A son baptême, il reçut les prénoms harmonieux, poétiques et distingués de Pancrace, Eusèbe, Zéphyrin, ce dont il parut se soucier comme un cloporte d'un ophicléide.

Consultée à son sujet, une somnambule extralucide, phrénologue distinguée, pédicure de nombreuses têtes couronnées, lui découvrit la bosse du mouvement perpétuel. D'où elle conclut logiquement qu'il serait un grand voyageur ou un grand mathématicien, à moins qu'il ne fût affligé de la danse de Saint-Guy.

Conformément à la prédiction de la somnambule, Zéphyrin montra, dès sa plus tendre enfance, des dispositions étonnantes pour les sciences expérimentales. Livré à lui-même, il s'ingéniait avec beaucoup de persévérance à résoudre les problèmes les plus compliqués.

Extrait de : l'idée fixe du Savant Cosinus.

L'exemple ci-dessus est obtenu à l'aide d'un seul élément **text** comportant un élément **tspan** par ligne :

```
<text>
  <tspan fill="#000088">L'enfance de Zéphyrin.</tspan>
  <tspan x="0" dy="6">C'est le 1er janvier ...</tspan>
  <tspan x="0" dy="3">poussa ses premiers ...</tspan>
  <tspan x="0" dy="3">et distingués de ...</tspan>
  ...
</text>
```

N.B. SVG ne gère pas les passages à la ligne. Il est du ressort de l'application générant le code source **SVG** de régler les points de césure et les coordonnées de début de ligne.

SVG possède également des mécanismes permettant de créer du texte justifié à droite (*attributs `textLength` et `lengthAdjust`*).

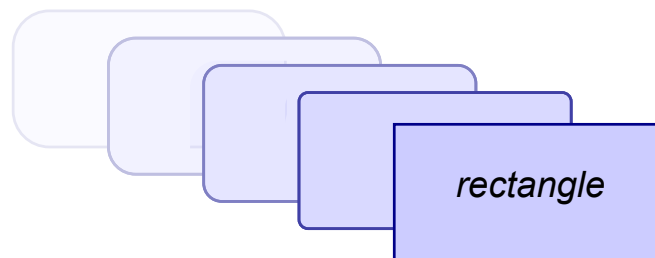
10.5 Groupes

> g

L'élément **g** est un élément structurel permettant de regrouper d'autres éléments **SVG** afin de les manipuler ensemble, en leur affectant par exemple des caractéristiques de présentation communes (*couleurs, traits, polices de caractères ...*) ou, comme il sera vu plus tard, de les positionner et de les orienter ensemble dans l'espace à l'aide d'une même transformation.

N.B. Un groupe peut contenir à peu près n'importe quel autre élément **SVG**, y compris un autre groupe.

Exemple :



Dans l'exemple ci-dessus, les rectangles héritent leur couleur et leur bordure de l'élément **g** englobant :

```
<g fill="#CCCCFF" stroke="#000050" stroke-width="3">
  <rect width="30" height="15" x="9" y="2" rx="4" opacity="0.1"/>
  <rect width="30" height="15" x="19" y="5" rx="3" opacity="0.25"/>
  <rect width="30" height="15" x="30" y="8" rx="2" opacity="0.5"/>
  <rect width="30" height="15" x="40" y="11" rx="1" opacity="0.75"/>
  <rect width="30" height="15" x="51" y="14" opacity="1.0"/>
</g>
```

Il faut toutefois noter que l'élément **g** ne transmet que les propriétés de présentation, mais pas les propriétés dimensionnelles (*width, height...*) ou de position (*x, y ...*).

10.6 Transformations

10.6.1 Principe des transformations

Soit à tracer la frise ci-dessous, visiblement composée d'un même motif plusieurs fois répété :



Cette frise est effectivement implémentée à l'aide d'éléments **polyline** correspondant au motif élémentaire, tous décrits à l'aide de la même liste de coordonnées (*sauf le premier et le dernier motif dont le segment d'extrémité est différent*).

La juxtaposition spatiale des motifs élémentaires est obtenue en spécifiant pour chacun d'eux une **translation** qui l'amène à la position souhaitée :

```
<polyline transform="translate(18.75,0) "
  points="0,130 0,20 120,20 120,102.5 60,102.5 60,75 ..."/>
<polyline transform="translate(168.75,0) "
  points="0,130 0,20 120,20 120,102.5 60,102.5 60,75 ..."/>
<polyline transform="translate(318.75,0) "
  points="0,130 0,20 120,20 120,102.5 60,102.5 60,75 ..."/>
```

La translation n'est qu'un exemple parmi l'ensemble des transformations géométriques dont dispose **SVG** (*translation, modification d'échelle, rotation, repères non orthogonaux, transformations quelconques...*).

Ainsi qu'illustré ci-dessus, ces transformations sont appliquées aux divers éléments **SVG** à l'aide de l'attribut **transform**.

N.B. Outre le texte et les divers éléments graphique, l'élément **g** est bien entendu une cible de choix pour les transformations, puisqu'il permet en une seule opération de modifier les caractéristiques géométriques de l'ensemble des éléments qu'il regroupe...

10.6.2 Translations

La translation est la plus simple des transformations. Elle consiste simplement à déplacer l'élément concerné dans la fenêtre de vue vers un nouvel emplacement.

La fonction **translate** spécifiée à l'aide de l'attribut **transform** prend deux arguments **tx** et **ty**, correspondant aux coordonnées du déplacement à effectuer :

```
transform="translate(tx,ty) "
```

N.B. le paramètre **ty** est optionnel. S'il est omis il est supposé valoir **0**.

La translation est une opération très utile pour représenter plusieurs fois un même objet éventuellement complexe car composé de multiples éléments :



Ainsi, le panneau **STOP** ci-dessus regroupe un polygone et un texte, groupés à l'aide d'un élément **g** puis représentés deux fois à des positions différentes grâce à une translation appliquée au groupe :

```
<g transform="translate(100,80) ">
  <polygon points="69.29,28.7 28.7,69.29 -28.7,69.29 -69.29,28.7 ..."/>
  <text dy="10">STOP</text>
</g>
<g transform="translate(900,80) ">
  <polygon points="69.29,28.7 28.7,69.29 -28.7,69.29 -69.29,28.7 ..."/>
  <text dy="10">STOP</text>
</g>
```

10.6.3 Modification d'échelle

> Modification d'échelle uniforme

La modification d'échelle s'effectue à l'aide de la fonction **scale**. Elle consiste à grossir ou à rapetisser l'objet considéré.

```
transform="scale(sx) "
```

Si **sx** est supérieur à 1, l'objet sera agrandi, dans le cas contraire il sera rapetissé.



L'image ci-dessus a été obtenue en réutilisant le groupe d'éléments correspondant au panneau (*polygone* + *texte*) et en lui faisant subir d'abord une diminution d'échelle (*scale*) puis une translation (*translate*) pour l'amener à l'endroit souhaité :

```
<g class="stop" transform="translate(800,300)">
  <polygone points="69.29,28.7 28.7,69.29 -28.7,69.29 -69.29,28.7 ..."/>
  <text dy="10">STOP</text>
</g>
<g class="stop" transform="translate(580,28)">
  <g transform="scale(0.3)">
    <polygone points="69.29,28.7 28.7,69.29 -28.7,69.29 -69.29,28.7..."/>
    <text dy="10">STOP</text>
  </g>
</g>
```

> Déformation

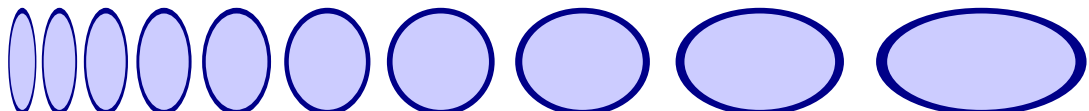
Il est également possible de spécifier un deuxième paramètre à la fonction **scale** :

```
transform="scale(sx,sy) "
```

De cette manière l'objet sera déformé d'un facteur différent suivant les deux dimensions **x** et **y**.

N.B. Lorsque **sy** est omis, il est de fait supposé égal à **sx**.

Exemple d'un cercle transformé en ellipses par redimensionnement non uniforme :



Une partie du code correspondant :

```
<g transform="scale(0.8,1)">
  <circle r="45" cx="500" cy="50"/>
</g>
<circle r="45" cx="500" cy="50"/>
<g transform="scale(1.25,1)">
  <circle r="45" cx="500" cy="50"/>
</g>
```


Noter en observant le graphique ci-dessus comment **SVG** redimensionne chacune des ellipses **y compris la largeur du trait**, qui n'est plus uniforme suivant les directions horizontales et verticales...

10.6.4 Rotations

> Rotation autour de l'origine

Une rotation s'effectue avec la fonction **rotate** :

```
transform="rotate(a)"
```

Cette opération aura pour effet de faire tourner la représentation de l'objet concerné de **a** degrés autour de l'origine du système de coordonnées utilisateur.

Exemple :



Afin de ne pas perdre l'objet dans l'espace, il est recommandé de le définir avec des coordonnées au voisinage de l'origine, de lui appliquer la rotation désirée, puis de le traduire ensuite à sa position finale :

```
<g transform="translate(100,100)">
  <g transform="scale(2)">
    <g transform="rotate(-45)">
      <circle transform="scale(1,0.25)" r="50"/>
      <text y="5.5">ellipses</text>
    </g>
  </g>
</g>
```

N.B. Comme l'axe vertical est compté du haut vers le bas, le sens de rotation positif correspond au sens horaire et non pas trigonométrique comme on pourrait s'y attendre...

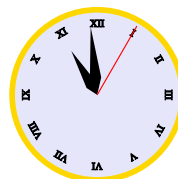
> Rotation autour d'un point arbitraire

La fonction **rotate** accepte en plus de l'angle **a** les coordonnées **cx,cy** du centre de rotation :

```
transform="rotate(a,cx,cy)"
```

Cette transformation est équivalente à une translation *translate(-cx,-cy)* pour amener le point (cx,cy) à l'origine, suivie par une rotation *rotate(a)* autour de l'origine, puis à nouveau par une translation *translate(cx,cy)* pour revenir à la position initiale.

Exemple :
(heure de création de ce document)



Les chiffres de cette horloge ont été tracés à l'aide d'une rotation autour du centre du cadran :

```
<circle cx="500" cy="100" r="75"/>
<text x="500" y="40">XII</text>
<text x="500" y="40" transform="rotate(30,500,100)">I</text>
<text x="500" y="40" transform="rotate(60,500,100)">II</text>
<text x="500" y="40" transform="rotate(90,500,100)">III</text>
...
```

10.6.5 Déformations

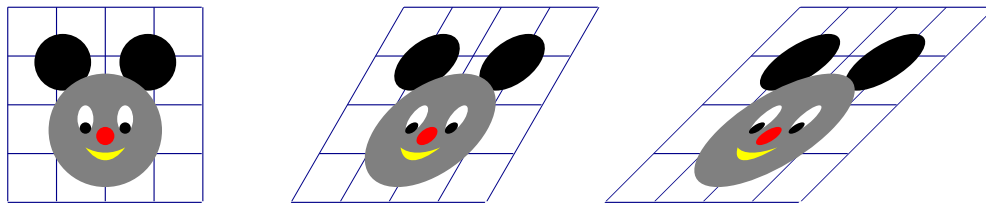
> Déformation suivant X

Une autre transformation élémentaire proposée par **SVG** est la déformation. Les déformations consistent à tracer les objets dans des repères non orthogonaux.

La déformation suivant **X** s'effectue à l'aide de la fonction **skewX** :

```
transform="skewX(a) "
```

La transformation **skewX** conserve l'axe **X**, tandis que l'axe **Y** est réorienté d'un angle **a** :



L'exemple ci-dessus correspond à des déformations de -30 puis -45 degrés.

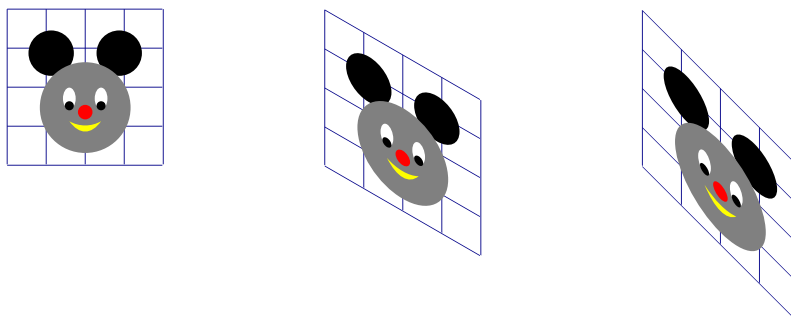
```
<g transform="skewX(-30)">
...
</g>
<g transform="skewX(-45)">
...
</g>
```

> Déformation suivant Y

De même, la déformation suivant **Y** s'effectue à l'aide de la fonction **skewY** :

```
transform="skewY(a) "
```

La transformation **skewY** conserve l'axe **Y**, tandis que l'axe **X** est réorienté d'un angle **a** :



L'exemple ci-dessus correspond à des déformations de 30 puis 45 degrés.

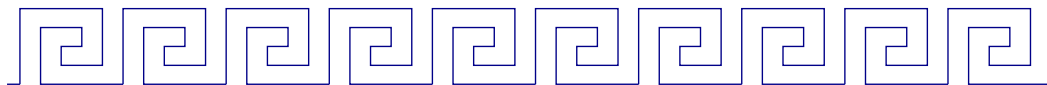
```
<g transform="skewY(30)">
...
</g>
<g transform="skewY(45)">
...
</g>
```

10.7 Réutilisation d'éléments

10.7.1 Réutilisation d'éléments

> use

Revenons une fois de plus à notre frise :



La répétition in extenso du motif élémentaire, déplacé d'intervalle en intervalle, conduit à un document dont le volume pourrait être réduit si l'on était capable de réutiliser simplement le motif.

Pour ceci, il faut lui donner un nom à l'aide d'un attribut **id**, puis y faire référence à l'aide d'un élément **use** pointant sur le motif à l'aide d'un fragment d'URL :


```
<polyline id="motif"
  transform="translate(-281.25,0)"
  points="0,130 0,20 120,20 120,102.5 ..."/>
<use x="150" xlink:href="#motif"/>
<use x="300" xlink:href="#motif"/>
<use x="450" xlink:href="#motif"/>
...
```

N.B. L'élément **use** utilise un lien **xml** défini par une spécification nommée **XLink**. Qu'il suffise ici de constater la nécessité de déclarer l'espace de noms correspondant, au niveau de l'élément **use** lui-même ou de l'un de ses parent, par exemple l'élément racine :

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  viewBox="-300 18 1650 150">
...
</svg>
```

 Les spécifications de XLink

[\[http://www.w3.org/TR/xlink/\]](http://www.w3.org/TR/xlink/)

 Les espaces de noms en XML

[\[http://www.w3.org/TR/REC-xml-names/\]](http://www.w3.org/TR/REC-xml-names/)

> symbol

L'inconvénient de l'exemple ci-dessus est que l'élément servant de modèle est visible sur la page, à sa position propre.

Il paraît tentant de pouvoir définir le motif élémentaire de manière générique, en l'absence de toute représentation graphique, puis de l'utiliser en le positionnant pour chacune des instances désirées.

Ce fonctionnement est possible, en définissant le modèle dans une section spéciale du document **SVG** précédant les éléments graphiques, appelée **defs**. Le modèle sera défini à l'aide de l'élément **symbol** :

```
<defs>
  <symbol id="motif" viewBox="0 0 150 150">
    <polyline points="0,130 0,20 120,20 120,102.5 60,102.5 ..."/>
  </symbol>
</defs>
```

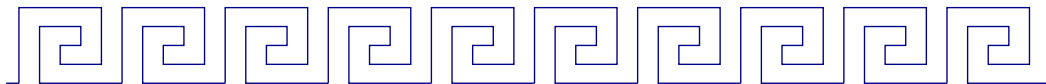
Remarquer l'attribut **viewBox** permettant de définir la zone utile du symbole.

La réutilisation d'un symbole se fait encore une fois à l'aide d'un élément **use** :

```
<use x="-281.25" width="150" height="150" xlink:href="#motif"/>
<use x="-131.25" width="150" height="150" xlink:href="#motif"/>
<use x="18.75" width="150" height="150" xlink:href="#motif"/>
...
```

Noter la nécessité d'indiquer l'échelle du symbole à l'aide des attributs **width** et **height**, en plus de sa position. Ces informations servent à recadrer la zone utile définie par l'attribut **viewBox** de l'élément **symbol**.

Le résultat est évidemment :



10.8 Feuilles de style CSS

10.8.1 SVG et CSS

> Exemple

Les avantages de séparer le fond de la forme, le contenu d'un document du style de la présentation ne sont plus à démontrer. On peut par exemple citer **HTML** et **CSS** ou dans une approche différente **XML** et **XSL**.

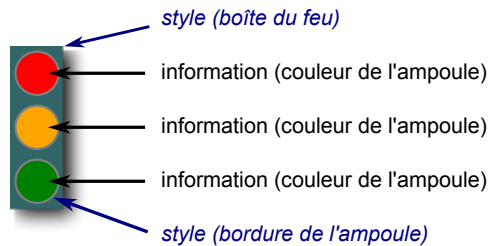
Ce raisonnement peut-il s'appliquer dans le domaine graphique ? La réponse est oui. **SVG** est compatible **CSS** :



Les trois ourses ci-dessus ont été tracés avec les mêmes primitives graphiques, faisant simplement référence à trois feuilles de style différentes.

> Position du problème

Le problème consistant à séparer le fond de la forme, l'information de sa présentation, est toutefois un peu plus délicat à régler pour un document graphique que pour un document texte.



En effet, la police de caractère est un attribut de style lorsqu'il s'agit de l'appliquer à un paragraphe, mais devient de l'information dans le cas d'un logo. La couleur relève du style dans le cas d'un graphe de données (*courbes, camemberts...*) mais est chargée d'information sur les représentations cartographiques (*routes, chemins, voies ferrées...*) ou dans le cas d'un feu de signalisation.

> La solution SVG

C'est pour cette raison qu'en **SVG** une propriété peut être spécifiée indifféremment à l'aide d'un **attribut** (*information*) ou d'une règle **CSS** (*style*) :

Code **SVG** :

```
<rect width="50" height="150"/>
<circle fill="red" cx="25" cy="25" r="20"/>
<circle fill="orange" cx="25" cy="75" r="20"/>
<circle fill="green" cx="25" cy="125" r="20"/>
```

Règles **CSS** :

```
rect { stroke:none; fill:#336666; }
circle { stroke:grey; stroke-width:2 }
```

N.B. La possibilité de spécifier des valeurs à l'aide d'une feuille de style est limitée aux **propriétés de présentation**.

10.8.2 Feuille de style interne ou externe

> Élément style

La méthode la plus simple pour adjoindre des règles **CSS** à un document **SVG** consiste, comme avec **HTML**, à utiliser une feuille de style interne.

L'élément **style** devra se trouver dans la section des définitions **defs** (*comme l'élément symbol*).

```
<defs>
  <style type="text/css"><![CDATA[
    rectangle { stroke:none; fill:#336666; }
    circle { stroke:grey; stroke-width:2 }
    text { stroke:none; font-family: Arial; font-size:20; }
  ]]></style>
</defs>
```

N.B. Comme on peut le noter ci-dessus il sera utile, par acquis de conscience, de protéger le contenu de la balise **style** l'aide d'un section **CDATA** (*ou pas*)...

L'usage d'un feuille de style interne n'est pas idéal. En effet, si elle permet de partager un même style pour de nombreux éléments d'un même document, elle ne peut être réutilisée dans un autre document que par recopie...

> Feuille de style externe

La seconde méthode consiste à faire référence à une feuille de style externe, localisée dans un document **CSS** à part entière.

La syntaxe à utiliser est la syntaxe classique permettant de faire référence à une feuille de style depuis un document **XML** :

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

Cette instruction de traitement est à placer au sein du document **SVG** dans la zone des entêtes, avant l'élément racine **svg** :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<?xml-stylesheet href="smile1.css" type="text/css"?>
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 180 180">
...
</svg>
```

10.8.3 Sélection d'éléments

> Sélection par famille

Avec **SVG** comme avec **HTML**, les sélecteurs **CSS** permettent de désigner tous les éléments d'une famille donnée :

```
rect { fill:blue; stroke:yellow; }
circle { stroke-width: 2; }
```

Dans l'exemple ci-dessus tous les éléments **rect** auront un fond bleu et une bordure jaune, tandis que les cercles auront une bordure de largeur 2.

> Eléments identifiés

Les éléments identifiés de manière unique à l'aide de l'attribut **id**, peuvent être désignés de la même façon que lorsque **CSS** s'applique à un document **HTML** :

```
#face { fill:black; }
#eyes ellipse { fill:white; stroke:black; }
```

Dans ce exemple, l'élément dont l'attribut **id** vaut *face* sera rempli en noir, et les ellipses enfant de l'élément identifié par *eyes* seront remplies en blanc avec une bordure noire.

> Classes d'éléments

Les concepteurs de **SVG** ont également repris la possibilité apparue avec **HTML** d'attacher un attribut **class** à l'ensemble des éléments graphiques.

Une série d'éléments peut ensuite être sélectionnée grâce à la valeur de leur attribut **class** :

```
.body { stroke:none; fill:#336666; }
.lamp { stroke:grey; stroke-width:2 }
```

Tous les éléments de la classe *body* seront remplis d'une couleur cyan foncé, sans bordure, et les éléments de la classe *lamp* auront une bordure grise d'une largeur de 2.

> Sélecteurs CSS

Pour le reste, la syntaxe des sélecteurs est celle spécifiée par **CSS 2**.