Chapitre 7 : Remplir la forme

Opacité

Extrait des spécifications du W3C

"Il y a plusieurs propriétés d'opacité en SVG:

- Opacité pour le remplissage
- Opacité pour le tracé
- Opacité pour l'élément 'stop' dans un gradient
- Opacité pour un objet ou un groupe

Excepté cette dernière, les propriétés d'opacité sont traitées dans les opérations intermédiaires de rendu. L'opacité pour un objet ou un groupe peut être considérée comme une opération postrendu. Ce qui signifie qu'après que l'objet ou le groupe soit rendu dans une image RGBA, l'opacité indique comment intégrer l'image obtenue dans l'image courante."

Toute valeur en dehors de l'intervalle 0.0 (transparente) à 1.0 (opaque) sera ramenée dans cet intervalle.

Opacité de remplissage et de tracé

Nous pouvons indiquer une opacité pour le remplissage ou le tracé de n'importe quel objet forme, texte ....

'fill-opacity'

Valeur:	<opacity-value> | inherit
Par défaut:	1
S'applique à:	formes et texte
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel
Animable:	oui

'stroke-opacity'

Valeur:	<opacity-value> | inherit
Par défaut:	1
S'applique à:	formes et texte
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel
Animable:	oui

Nous pouvons écrire

```
<rect x="0" y="0" width="200" height="200" fill-opacity="0.5" fill="red"/>
```

ou

```
<rect x="0" y="0" width="200" height="200"
                  style="fill-opacity:0.5;fill:red"/>
```

Pour une opacité inférieure à 1, le résultat dépend de la couleur de fond.

Nous créons un fond avec des bandes de couleur, nous constatons que l'application de fill-opacity="0.5" pour un rectangle dans la Figure 7-1 donne des résultats différents pour chaque bande.

Source du fichier svg :

```
<svg width="600" height="110" viewBox="-25 -30 600 110">
<rect x="0" y="0" width="50" height="50" style="fill:black;fill-opacity:1"/>
<rect x="50" y="0" width="50" height="50" style="fill:black;fill-opacity:0.9"/>
<rect x="100" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.8"/>
<rect x="150" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.7"/>
<rect x="200" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.6"/>
<rect x="250" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.5"/>
<rect x="300" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.4"/>
<rect x="350" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.3"/>
<rect x="400" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.2"/>
<rect x="450" y="0" width="50" height="50"
style="fill:black;fill-opacity:0.1"/>
<rect x="500" y="0" width="50" height="50"
style="fill:black;fill-opacity:0"/>
<rect x="20" y="10" width="510" height="30"
style="fill:yellow;fill-opacity:0.5"/>
</svg>
```

Figure 7-1. Rectangle avec opacité de 0.5 sur des bandes

Opacité pour l'élément 'stop' dans un gradient

Voir plus loin dans le paragraphe sur les gradients

Opacité pour un objet ou un groupe

'opacity'

Valeur:	<alphavalue> | inherit
Par défaut:	1
S'applique à:	groupe et objets graphiques
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

La figure 7-2 montre qu'avec une opacité de 0.5 appliquée au groupe, le cercle jaune recouvre le cercle rouge et que l'opacité est appliquée à l'ensemble.
Avec une opacité appliquée à chaque cercle, la partie commune aux deux cercles est orange.

Source du fichier svg :

```
<svg width="450" height="250" viewBox="-25 -25 450 250">
  <g opacity="0.5">
    <circle cx="75" cy="100" r="50" fill="red" fill-opacity="1"/>
    <circle cx="125" cy="100" r="50" fill="yellow" fill-opacity="1"/>
  </g>
  <g>
    <circle cx="275" cy="100" r="50" fill="red" fill-opacity="0.5"/>
    <circle cx="325" cy="100" r="50" fill="yellow" fill-opacity="0.5"/>
  </g>
  <text x="100" y="180" style="text-anchor:middle">
      opacity on group
  </text>
  <text x="300" y="180" style="text-anchor:middle">
      opacity on elements
  </text>
</svg>
```

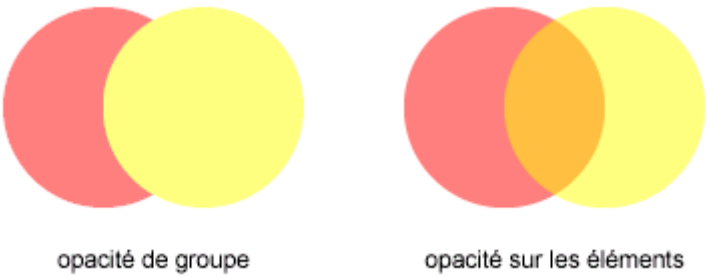

Figure 7-2. Opacité de groupe (gauche) sur chaque cercle (droite)

Gradients

Les gradients peuvent être utilisés pour remplir ou tracer des formes de base, des objets 'path' ou du texte. Ils utilisent plusieurs couleurs avec des effets de transition pour passer de l'une à l'autre.

Les couleurs ou les éléments 'stop'

Voici la syntaxe pour l'élément ‘stop’ :

<**stop** id="name"
offset="NumberOrPercentage"
stop-color="Color"
stop-opacity="Opacity-value" />

Note: Vous verrez de nombreux diagrammes aux chapitres 7-8 et 9. Les conventions sont les suivantes: élément dans un cercle, attribut dans un rectangle, propriété dans un rectangle aux coins arrondis, valeur par défaut en rouge, valeurs admises en vert. Les attributs et propriétés de l'élément sont alignés sur des lignes rouges et les descendants possibles sur une ligne bleue.

Diagramme 7-1. Diagramme de l'élément 'stop'

Les valeurs pour l'attribut 'offset' sont entre 0% et 100% ou entre 0 et 1.
Pour chaque élément 'stop', la valeur de 'offset' doit être supérieure ou égale à la valeur du précédent élément 'stop'.
Des valeurs inférieures à 0 (ou à 0%) sont ramenées à 0%. Des valeurs supérieures à 1 (ou à 100%) sont ramenées à 100%.

Pour commencer, nous créons un gradient linéaire avec deux couleurs seulement, ces couleurs sont définies dans les éléments 'stop'.

```
<defs>
   <linearGradient id="MyGradient">
        <stop offset="10%" stop-color="red"/>
        <stop offset="90%" stop-color="yellow"/>
   </linearGradient>
</defs>
```

Comprendre l'attribut 'offset' ?

Pour les gradients linéaires, la valeur de l'offset représente une position le long du vecteur défini par les valeurs des attributs 'x1' 'y1' 'x2' et 'y2'.
Pour les gradients circulaires, il représente un pourcentage de la distance du foyer (défini par 'fx' et 'fy') au bord du cercle le plus grand défini par 'cx' 'cy' et 'r'.

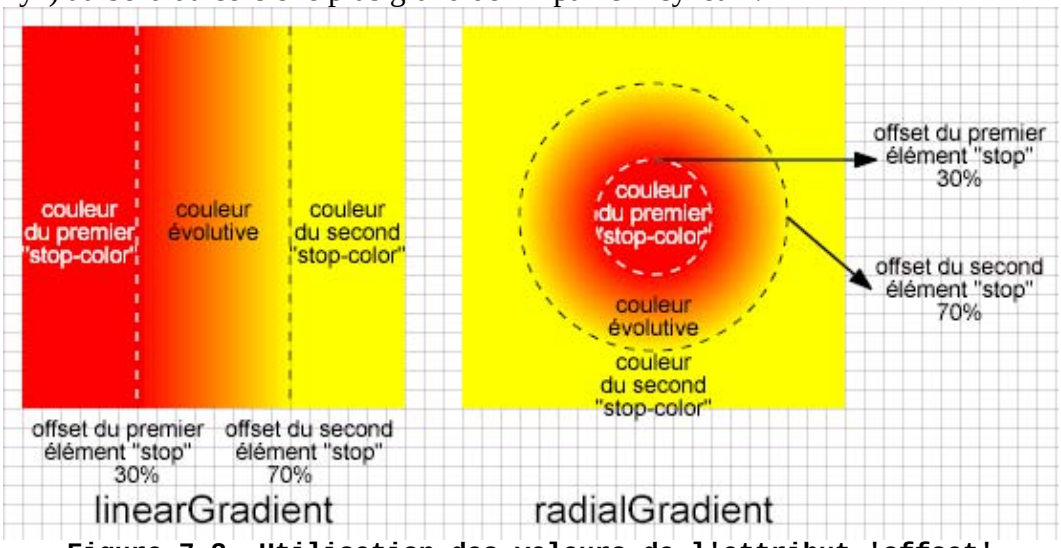

Figure 7-3. Utilisation des valeurs de l'attribut 'offset'

Si nous avons ce code :

```
<defs>
   <linearGradient id="MyGradient">
        <stop offset="10%" stop-color="red"/>
        <stop offset="90%" stop-color="yellow"/>
   </linearGradient>
```

```
</defs>
```

Pour remplir une forme, la couleur rouge est utilisée depuis le début de la forme jusqu'à 10% de sa longueur totale, la couleur jaune est utilisée de 90% à la fin, de 10% à 90% la couleur passe graduellement du rouge au jaune.
La figure 7-4 montre quelques valeurs pour les attributs 'offset' d'un gradient linéaire.
Pour 0% et 100%, tout le rectangle est rempli avec des dégradé. Pour 50% et 50%, nous avons deux zones, l'une rouge, l'autre jaune sans dégradés pour passer de l'une à l'autre. Nous aurions le même résultat avec 30% et 30% par exemple.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
<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>
        <linearGradient id="MyGradient3">
            <stop offset="30%" stop-color="red"/>
            <stop offset="70%" stop-color="yellow"/>
        </linearGradient>
        <linearGradient id="MyGradient4">
            <stop offset="50%" stop-color="red"/>
            <stop offset="50%" stop-color="yellow"/>
        </linearGradient>
</defs>
<rect x="0" y="0" width="150" height="150" style="fill:url(#MyGradient1)"/>
<rect x="150" y="0" width="150" height="150" style="fill:url(#MyGradient2)"/>
<rect x="300" y="0" width="150" height="150" style="fill:url(#MyGradient3)"/>
<rect x="450" y="0" width="150" height="150" style="fill:url(#MyGradient4)"/>
<text x="75" y="175" style="text-anchor:middle">Offset 0 and 100</text>
<text x="225" y="175" style="text-anchor:middle">Offset 20 and 80</text>
<text x="375" y="175" style="text-anchor:middle">Offset 30 and 70</text>
<text x="525" y="175" style="text-anchor:middle">Offset 50 and 50</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"/>
<path d="M345 0l0 150"/>
<path d="M405 0l0 150"/>
<path d="M525 0l0 150"/>
<path d="M525 0l0 150"/>
</g>
</svg>
```

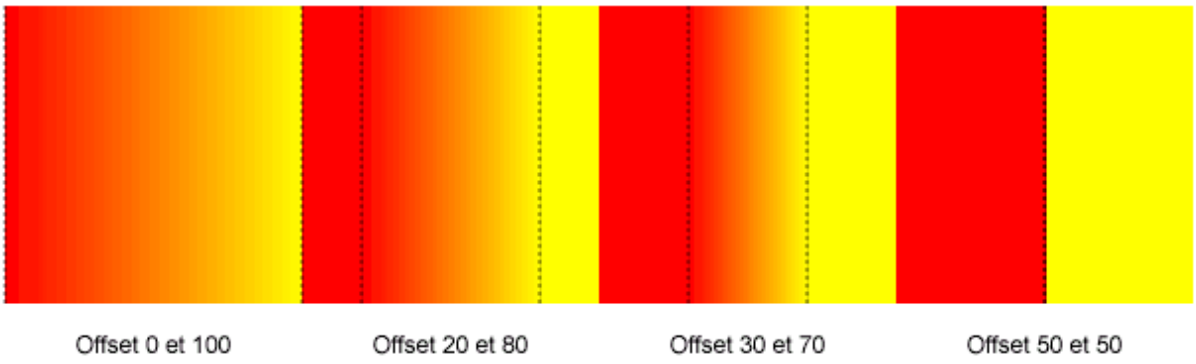

Figure 7-4. Quelques valeurs pour 'offset'

Avec ce code pour un gradient circulaire:

```
<defs>
   <radialGradient id="MyGradient">
        <stop offset="10%" stop-color="red"/>
        <stop offset="90%" stop-color="yellow"/>
   </radialGradient>
</defs>
```

Les valeurs de 'offset' s'appliquent aux rayons des cercles (nous pouvons choisir le foyer et le centre du plus grand cercle, voir dans la suite)
Pour remplir le rectangle, la couleur rouge est utilisée pour le cercle de rayon 10% du rayon du plus grand cercle, le jaune est utilisé dans la couronne entre 90% et le cercle maximum. Entre les deux, de 10% à 90%, nous avons un dégradé du rouge vers le jaune.

La figure 7-5 montre le résultat pour quelques valeurs de 'offset'.
Pour 0% et 100%, nous avons uniquement un dégradé à l'intérieur du grand cercle.
Pour 50% et 50%, nous retrouvons les deux couleurs sans dégradé.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <radialGradient id="MyGradient1">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </radialGradient>
    <radialGradient id="MyGradient2">
      <stop offset="20%" stop-color="red"/>
      <stop offset="80%" stop-color="yellow"/>
    </radialGradient>
    <radialGradient id="MyGradient3">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </radialGradient>
    <radialGradient id="MyGradient4">
      <stop offset="50%" stop-color="red"/>
      <stop offset="50%" stop-color="yellow"/>
    </radialGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">Offset 0 and 100</text>
  <text x="225" y="175" style="text-anchor:middle">Offset 20 and 80</text>
  <text x="375" y="175" style="text-anchor:middle">Offset 30 and 70</text>
  <text x="525" y="175" style="text-anchor:middle">Offset 50 and 50</text>
  <g style="stroke-dasharray:2 2;stroke:black;fill:none">
    <circle cx="75" cy="75" r="0"/>
    <circle cx="75" cy="75" r="75"/>
    <circle cx="225" cy="75" r="15"/>
    <circle cx="225" cy="75" r="60"/>
    <circle cx="375" cy="75" r="22"/>
    <circle cx="375" cy="75" r="52"/>
    <circle cx="525" cy="75" r="37.5"/>
    <circle cx="525" cy="75" r="37.5"/>
  </g>
</svg>
```

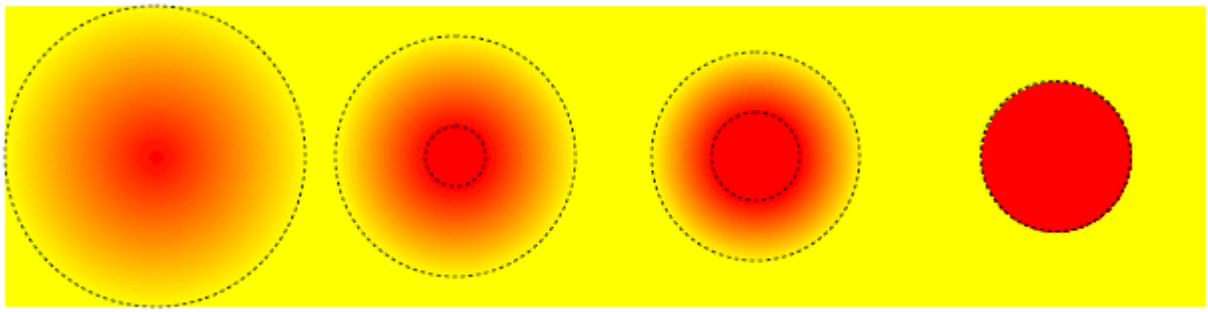

Figure 7-5. Quelques valeurs pour 'offset' dans un gradient circulaire

Autres attributs

Avec '**stop-color**' nous choisissons la couleur à utiliser (nom, valeur RGB ou hexadécimale). Il n'est pas possible de choisir "none", mais avec n'importe quelle couleur et avec stop-opacity="0" nous obtenons des parties transparentes très utiles pour utiliser avec les filtres d'éclairage pour obtenir des effets 3D.

Avec '**stop-opacity**' nous choisissons l'opacité pour la couleur entre 0 et 1.

Propriété 'stop-color'

Valeur:	currentColor |<color> [icc-color(<name>[,<icccolorvalue>]*)] | inherit
Par défaut:	black
S'applique à:	éléments 'stop'
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

Avec '**stop-opacity**' nous choisissons l'opacité pour la couleur entre 0 et 1.

Propriété 'stop-opacity'

Valeur:	<alphavalue> | inherit
Par défaut:	1
S'applique à:	éléments 'stop'
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

Propriétés pour le rendu des images

Pour les gradients, une propriété est importante : 'color-interpolation'.
Par défaut, 'color-interpolation' prend la valeur 'sRGB'

Propriété 'color-interpolation'

Valeur:	auto | sRGB | linearRGB | inherit
Par défaut:	sRGB
S'applique à:	groupe, objets graphiques et 'animateColor'
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel
Animable:	oui

'**auto**' signifie qu'aucun espace de couleurs particulier n'est requis .

Sur cet exemple (Figure 7-6), nous voyons que la répartition des couleurs n'est pas la même avec 'sRGB' et 'linearRGB'.

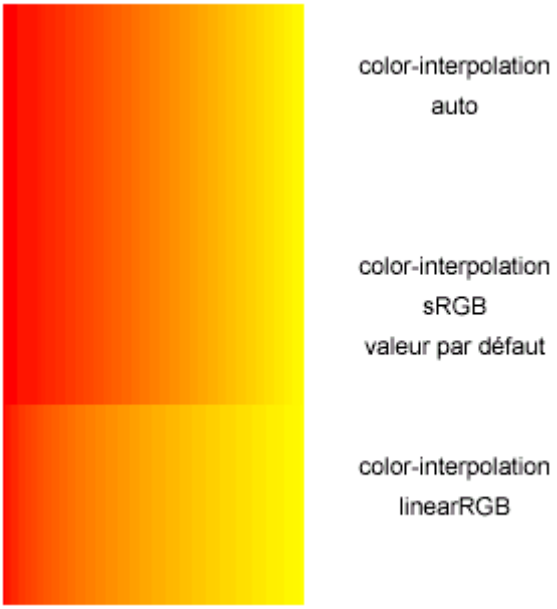

Figure 7-6. Propriété 'color-interpolation'

Attributs communs pour les gradients linéaire et radial

L'attribut '**gradientUnits**' peut être "userSpaceOnUse" ou "objectBoundingBox", il définit le système de coordonnées pour les attributs x1 y1 x2 y2 ('linearGradient') et cx cy r fx fy ('radialGradient').
Avec "userSpaceOnUse", les valeurs sont définies dans le repère courant (défini par éventuellement viewBox et les transformations sur les ascendants de l'élément auquel est appliqué le gradient).
Avec "objectBoundingBox", valeur par défaut, les valeurs sont définies dans le système de coordonnées défini par la trace ('bounding box') de l'élément auquel s'applique le gradient.
Dans ce cas il est plus facile d'utiliser des pourcentages pour les attributs.

Nous verrons ensuite que pour un gradient linéaire, ce choix ne donne pas les mêmes remplissages pour un carré et un rectangle par exemple.

L'attribut '**gradientTransform**' permet d'ajouter des transformations au système de coordonnées. Nous pouvons utiliser 'translate(tx,ty)' 'rotate(angle,cx,cy)' 'skewX(angle)' 'skewY(angle)' 'scale(sx,sy)' ou 'matrix(a b c d e f)'.

La figure 7-7 montre l'effet de trois transformations appliquées à un gradient linéaire.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <linearGradient id="MyGradient1">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient2" gradientTransform="rotate(45)">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient3" gradientTransform="scale(0.5)">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient4" gradientTransform="skewX(45)">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">identity</text>
  <text x="225" y="175" style="text-anchor:middle">rotate(45)</text>
  <text x="375" y="175" style="text-anchor:middle">scale(0.5)</text>
  <text x="525" y="175" style="text-anchor:middle">skewX(45)</text>
</svg>
```

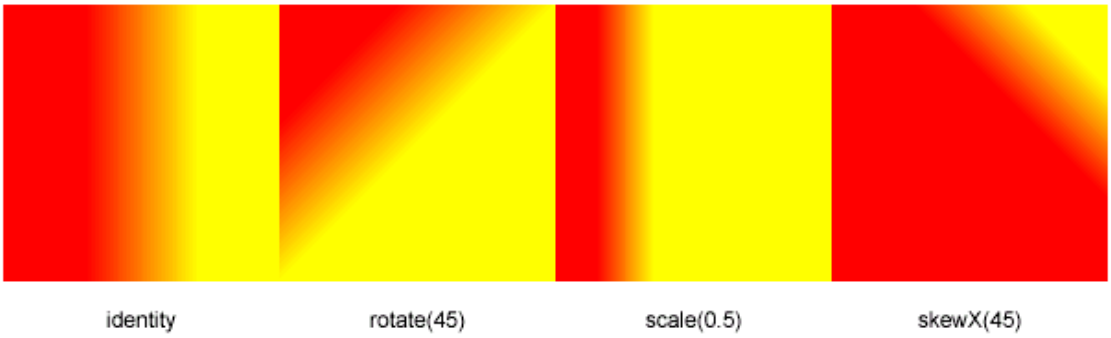

Figure 7-7. Trois transformations appliquées à 'linearGradient'

L'attribut '**spreadMethod**' peut être 'pad', 'reflect' ou 'repeat'. Cet attribut indique comment compléter le gradient quand il n'est défini que dans une partie de l'objet (avec x1>0 ou x2<100 par exemple pour un gradient linéaire).
Avec 'pad', valeur par défaut, la première ou la dernière couleur est utilisée pour compléter avant ou après les définitions du gradient.
Avec 'reflect', le gradient est répété du début à la fin puis de la fin au début et ainsi de suite pour compléter le gradient.
Avec 'repeat', le gradient est répété du début à la fin puis du début à la fin et ainsi de suite pour compléter le gradient.
Pour voir les effets produits, nous définissons ce gradient linéaire :

```
    <defs>
      <linearGradient id="MyGradient" x1="20%" y1="0%" x2="50%" y2="0%">
```

```
        <stop offset="30%" stop-color="red"/>
        <stop offset="70%" stop-color="yellow"/>
      </linearGradient>
    </defs>
```

Avec ces valeurs pour x1 y1 x2 et y2, le gradient commence à 20% de la forme et finit à 50%.
La figure 7-8 montre l'effet du choix de l'attribut 'spreadMethod', par défaut ou avec 'pad', la couleur rouge complète de 0% à 20% et le jaune de 50% à 100%.
Avec 'reflect' ou 'repeat', nous pouvons voir comment est complété le gradient.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <linearGradient id="MyGradient1" x1="20%" y1="0%" x2="50%" y2="0%">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient2" spreadMethod="pad"
                    x1="20%" y1="0%" x2="50%" y2="0%">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient3" spreadMethod="reflect"
                    x1="20%" y1="0%" x2="50%" y2="0%">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient4" spreadMethod="repeat"
                    x1="20%" y1="0%" x2="50%" y2="0%">
      <stop offset="30%" stop-color="red"/>
      <stop offset="70%" stop-color="yellow"/>
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">default</text>
  <text x="225" y="175" style="text-anchor:middle">pad</text>
  <text x="375" y="175" style="text-anchor:middle">reflect</text>
  <text x="525" y="175" style="text-anchor:middle">repeat</text>
</svg>
```

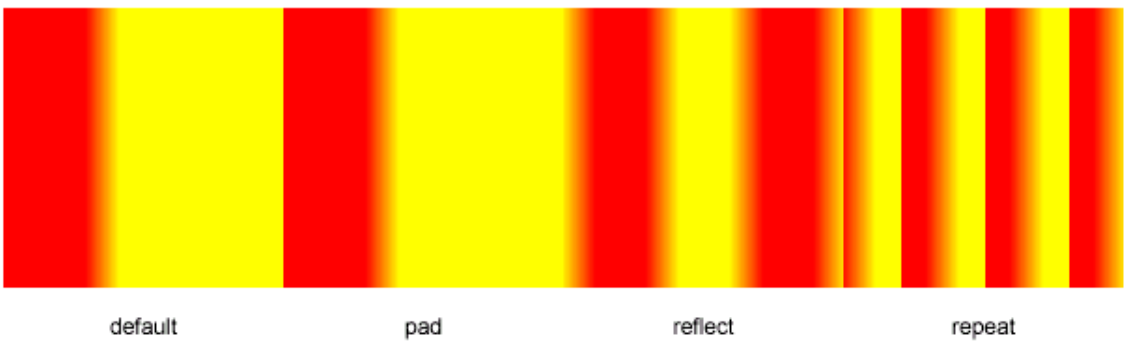

Figure 7-8. Différentes valeurs pour 'spreadMethod'

Comment appliquer un gradient à un objet ?

Nous définissons le gradient dans une section <defs>, lui donnons un 'id', ici "MyGradient".

```
    <defs>
        <linearGradient id="MyGradient" x1="20%" y1="0%" x2="50%" y2="0%">
            <stop offset="10%" stop-color="red"/>
            <stop offset="90%" stop-color="yellow"/>
        </linearGradient>
    </defs>
```

Nous pouvons alors utiliser ce gradient pour remplir ou tracer n'importe quel élément, forme élémentaire, objet 'path' ou texte.
Ce rectangle sera rempli avec le gradient :

```
<rect x='0' y='0' width='200' height='200' fill='url(#MyGradient)'/>
```

Nous pouvons également utiliser l'attribut 'style'

```
<rect x='0' y='0' width='200' height='200'
style='stroke:black;fill:url(#MyGradient)'/>
```

La figure 7-9 montre quelques exemples d'utilisation de gradient linéaire pour remplir ou tracer des objets.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <linearGradient id="MyGradient1" x2="5%" spreadMethod="reflect">
      <stop offset="10%" stop-color="red"/>
      <stop offset="90%" stop-color="yellow"/>
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="stroke:black;fill:url(#MyGradient1)"/>
  <circle cx="225" cy="75" r="60" style="fill:none;
          stroke-width:10;stroke:url(#MyGradient1)"/>
  <path d="M320 20l 70 0 0 60 20 50 -100 0z"
        style="stroke:black;fill:url(#MyGradient1)"/>
  <text x="525" y="100" style="stroke:black;fill:url(#MyGradient1);
        font-family:Balloon;text-anchor:middle;font-size:100">SVG</text>
  <text x="75" y="175" style="text-anchor:middle">fill rectangle</text>
  <text x="225" y="175" style="text-anchor:middle">stroke circle</text>
  <text x="375" y="175" style="text-anchor:middle">fill path</text>
  <text x="525" y="175" style="text-anchor:middle">fill text</text>
</svg>
```

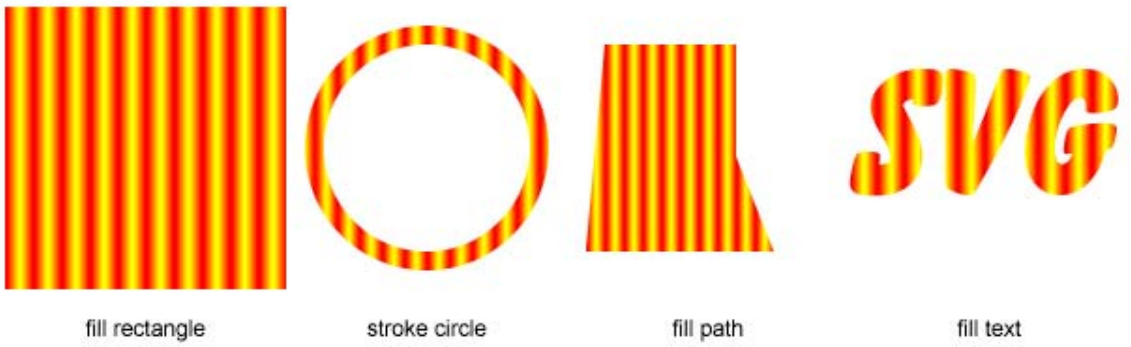

Figure 7-9. Exemples d'utilisation d'un gradient linéaire

Gradients linéaires

Voici la syntaxe pour l'élément ‘linearGradient’ :

```
<linearGradient id="name"
                gradientUnits="userSpaceOnUse|objectBoundingBox"
                gradientTransform="transform-list"
                spreadMethod="pad|repeat|reflect"
                x1="NumberOrPercentage"
                y1="NumberOrPercentage"
                x2="NumberOrPercentage"
                y2="NumberOrPercentage">
  <!--  éléments stop  -->
</linearGradient>
```

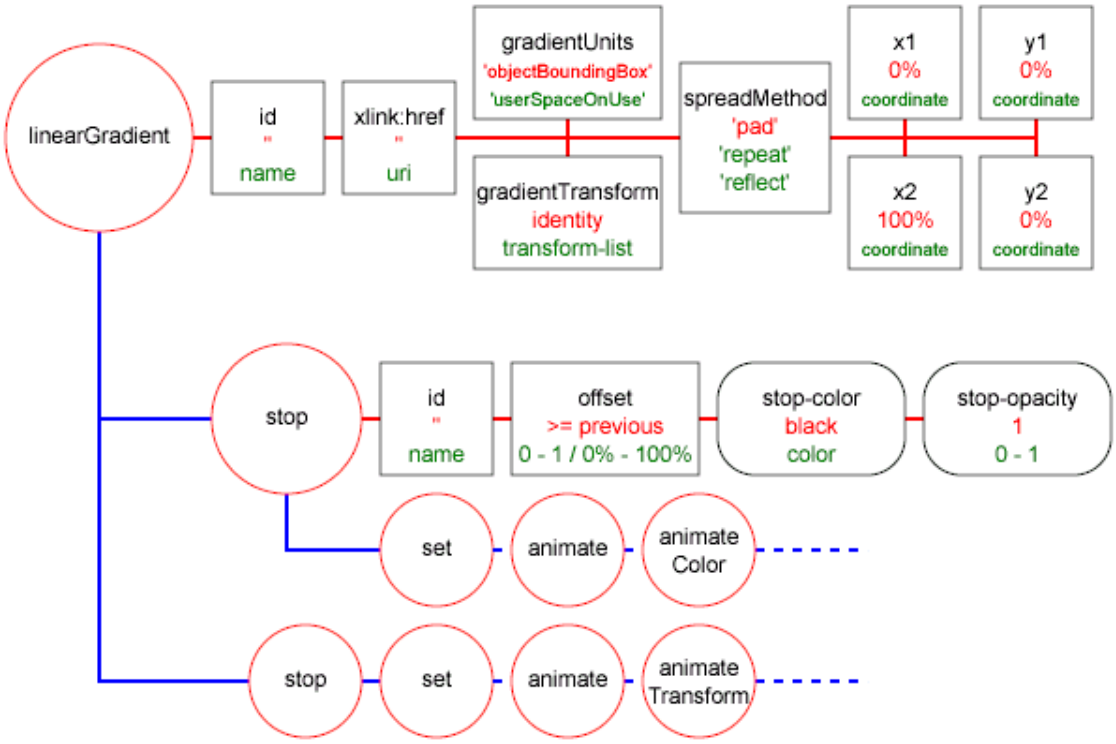

Diagramme 7-2. L'élément 'linearGradient'

La figure 7-10 montre les résultats pour les valeurs possibles de l'attribut 'spreadMethod'. Dans cet exemple, x1="10%" et x2="40%".

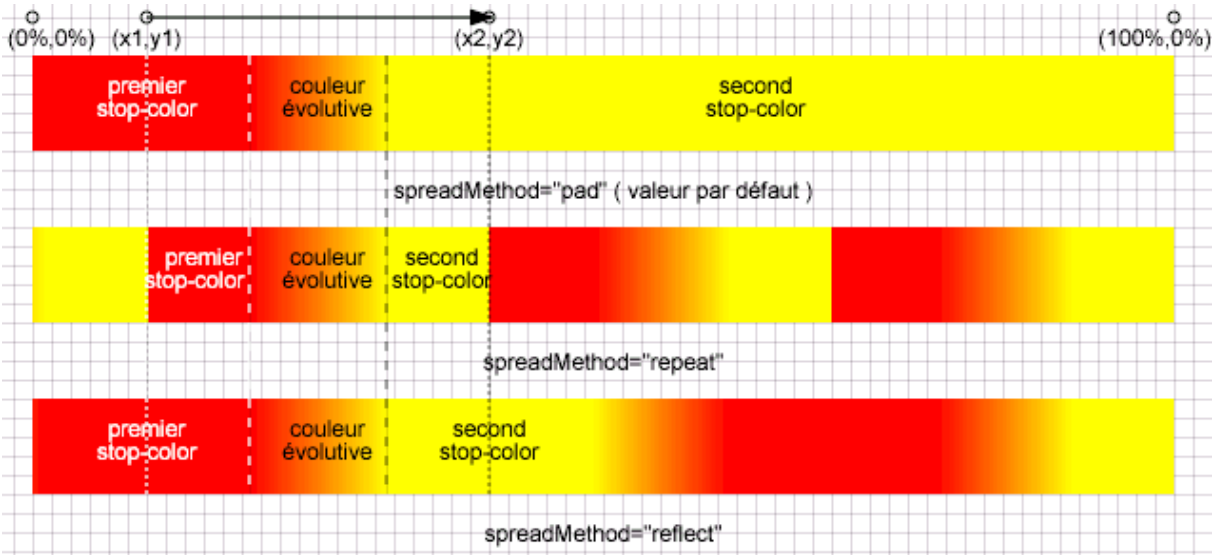

Figure 7-10. 'spreadMethod' en détail

Les attributs **'x1' 'y1' 'x2' 'y2'** définissent un vecteur pour le gradient linéaire.
Ce vecteur définit les points de début et de fin par rapport auxquels le gradient est dessiné, les couleurs sont appliquées suivant des bandes perpendiculaires au vecteur.

Les valeurs par défaut sont respectivement 0% 0% 100% et 0%, soit un gradient qui occupe tout l'espace avec des bandes verticales (par rapport au système de coordonnées courant)
La figure 7-11 montre quatre exemples :
- bandes verticales avec les valeurs par défaut 0 0 100 0, 0 x 100 x donnerait le même résultat
- bandes diagonales orientées NO-SE avec 0 0 100 100
- bandes diagonales orientées NE-SO avec 0 100 100 0
- bandes horizontales avec 0 0 0 100, x 0 x 100 donnerait le même résultat.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <linearGradient id="MyGradient1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient2" x1="0%" y1="0%" x2="100%" y2="100%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient3" x1="0%" y1="100%" x2="100%" y2="0%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient4" x1="0%" y1="0%" x2="0%" y2="100%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
```

```
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">0 0 100 0</text>
  <text x="225" y="175" style="text-anchor:middle">0 0 100 100</text>
  <text x="375" y="175" style="text-anchor:middle">0 100 100 0</text>
  <text x="525" y="175" style="text-anchor:middle">0 0 0 100</text>
</svg>
```

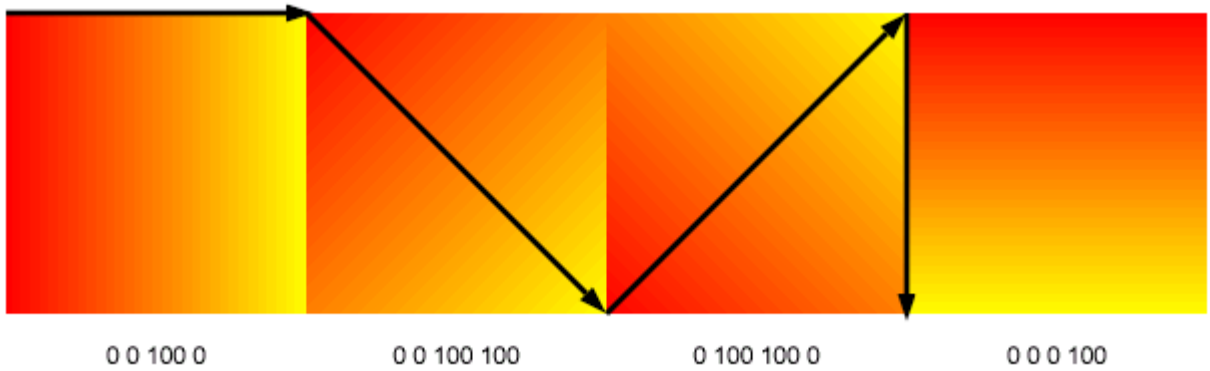

Figure 7-11. Différentes valeurs pour x1 y1 x2 y2

L'effet de x1, y1, x2 et y2 avec gradientUnits="objectBoundingBox" dépend de l'objet à remplir. Le résultat n'est pas le même sur un carré ou un rectangle.
Pour un rectangle, le système de coordonnées n'est pas orthogonal et les bandes de couleur, perpendiculaires au vecteur du gradient dans le calcul dans ce système ne le sont pas au sens commun.

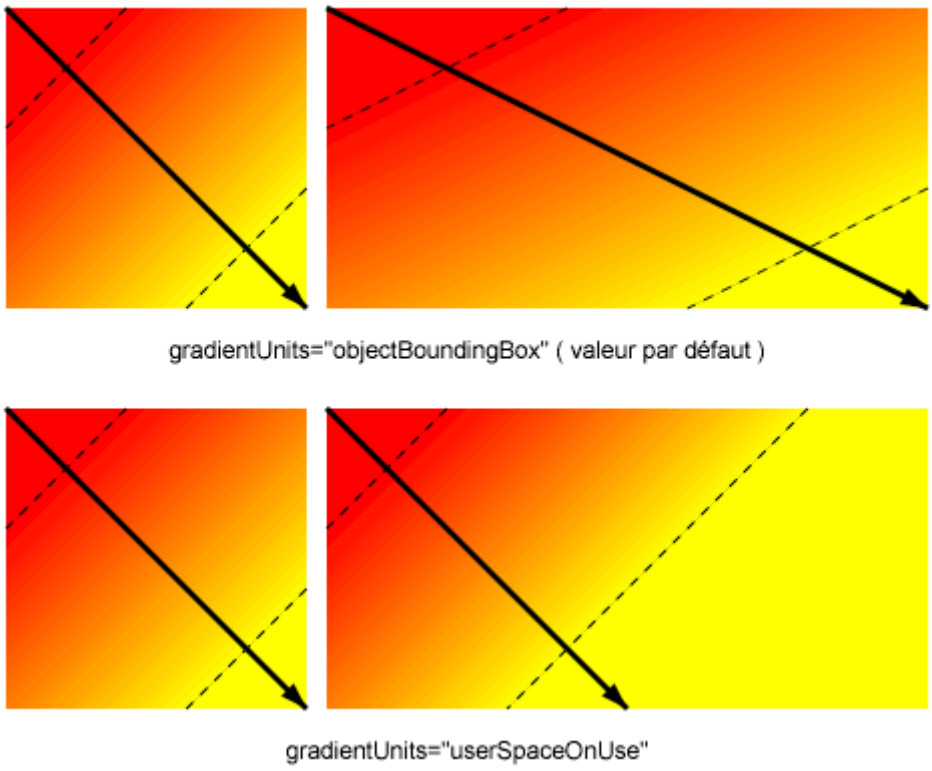

Figure 7-12. L'attribut gradientUnits et les formes

Par exemple, dans la figure 7-12, nous appliquons le même gradient avec gradientUnits = "objectBoundingBox" x1="0%" y1="0%" x2="100%" et y2="100%".
Les valeurs d'offset sont 20% et 80%, les limites des couleurs sont dessinées.
Si nous voulons la même orientation des bandes sur deux formes différentes, nous devons utiliser gradientUnits = "userSpaceOnUse" et définir un gradient pour chaque forme avec des valeurs différentes pour x1, y1, x2 et y2.

Nous aurons le même problème en utilisant gradientTransform.

La figure 7-13 permet une comparaison entre l'utilisation de x1 y1 x2 y2 et de 'gradientTransform' :

Source du fichier svg :

```
:<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <linearGradient id="MyGradient1" x1="0%" y1="0%" x2="100%" y2="100%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient2" gradientTransform="rotate(45)">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient id="MyGradient3" x1="0%" y1="100%" x2="100%" y2="0%">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
    <linearGradient gradientUnits="userSpaceOnUse" id="MyGradient4"
                    x1="450" y1="150" x2="600" y2="150"
                    gradientTransform="rotate(-45,450,150)">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="yellow"/>
    </linearGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">0 0 100 100</text>
  <text x="225" y="175" style="text-anchor:middle">rotate(45)</text>
  <text x="375" y="175" style="text-anchor:middle">0 100 100 0</text>
  <text x="525" y="175" style="text-anchor:middle">rotate(-45)</text>
</svg>
```

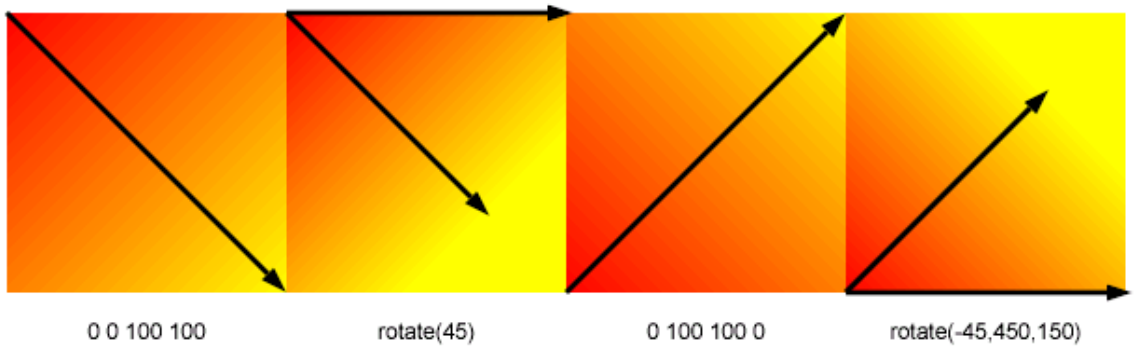

Figure 7-13. Utiliser x1 y1 x2 y2 ou 'gradientTransform'

Nous pouvons voir que gradientTransform="rotate(45)" ne donne pas le même résultat que 0 100 100 0 car dans la rotation le vecteur garde sa longueur et ne couvre pas toute la zone.
De même gradientTransform="rotate(-45)" et 0 100 100 0 sont très différents car le centre de la rotation est l'origine 0,0 et nous ne pouvons définir le centre de la rotation avec des

pourcentages par rapport à l'objet. Nous ne pouvons définir un tel gradient pour plusieurs objets.

Gradients circulaires

Voici la syntaxe pour l'élément ‘radialGradient’ :

```
<radialGradient id="name"
                gradientUnits="userSpaceOnUse|objectBoundingBox"
                gradientTransform="transform-list"
                spreadMethod="pad|repeat|reflect"
               cx="NumberOrPercentage"
               cy="NumberOrPercentage"
               r="NumberOrPercentage"
               fx="NumberOrPercentage"
               fy="NumberOrPercentage">
 <!--  éléments stop  -->
</radialGradient>
```

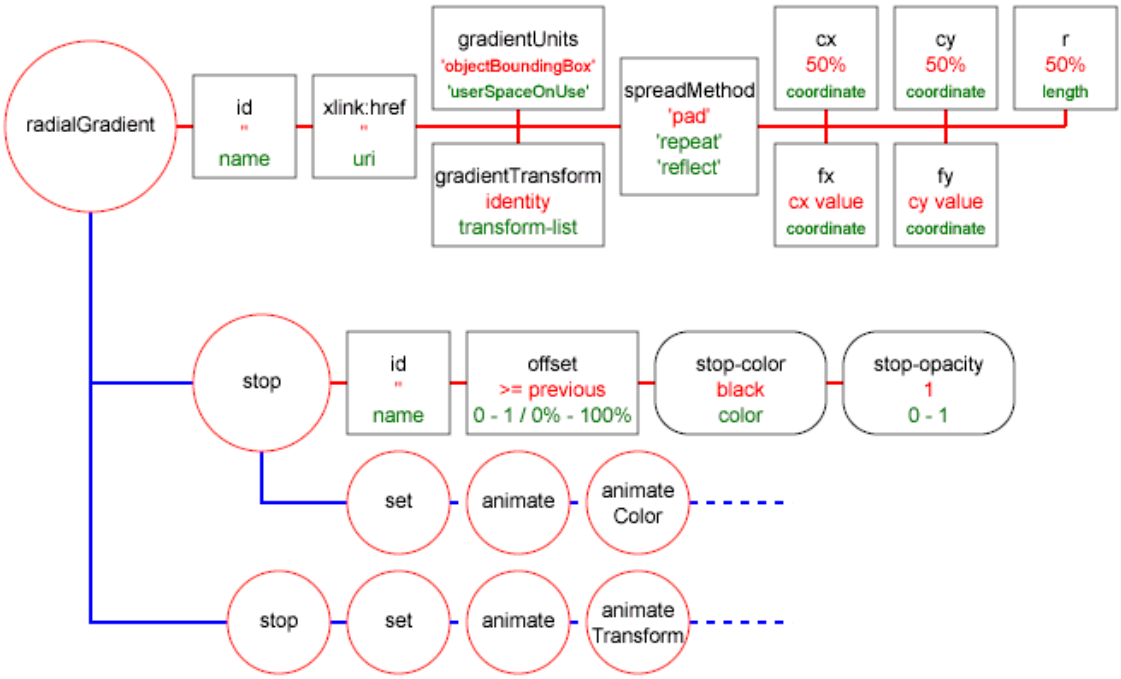

Diagramme 7-3. L'élément 'radialGradient'

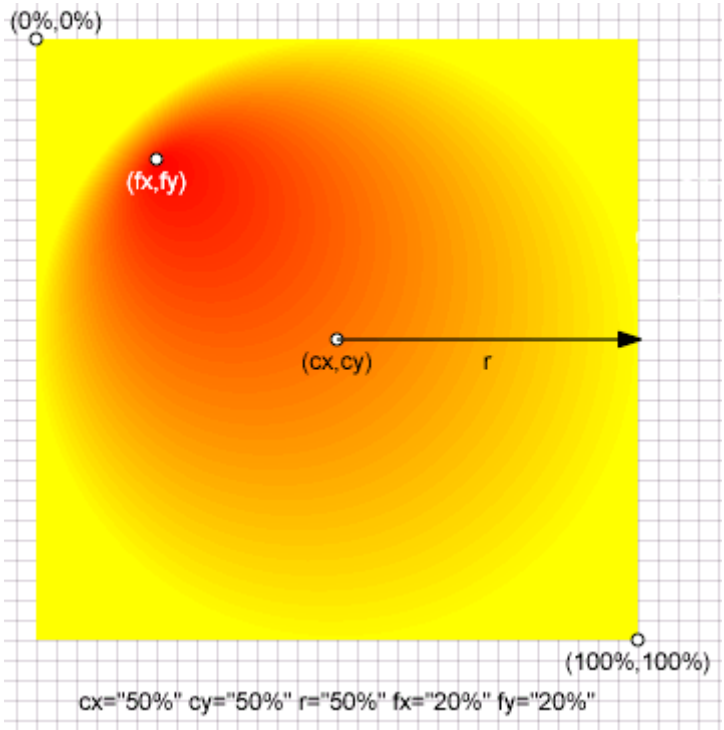

Figure 7-14. Attributs pour l'élément 'radialGradient'

cx, cy, r définissent le plus grand cercle pour le gradient (qui correspond à un 'offset' de 100%). Les valeurs par défaut sont de 50%.
fx, fy définissent le point focal pour le gradient (qui correspond à un 'offset' de 0%).
Les autres 'offset' seront définis par des cercles dont le centre se déplacera de (fx,fy) vers (cx,cy) avec un rayon variant de 0 à r.
Par défaut le point focal coïncide avec le centre du grand cercle (cx,cy).

La figure 7-14 montre un exemple avec cx="50%" cy="50%" r="50%" fx="20%" fy="20%" et offset="0%" pour le rouge et offset="100%" pour le jaune.

Nous pouvons déplacer le point focal et utiliser 'repeat' or 'reflect' for 'spreadMethod' pour obtenir des effets intéressants.
La figure 7-15 montre deux exemples où le point focal est au centre avec 'reflect' et 'repeat' pour 'spreadMethod'.
Pour les deux autres exemples le point focal est distinct du centre.

Source du fichier svg :

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <radialGradient id='MyGradient1' spreadMethod='reflect'
                    cx='50%' cy='50%' r='5%' fx='50%' fy='50%'>
      <stop id='c1' offset='5%' stop-color='red' stop-opacity='1'/>
      <stop id='c2' offset='95%' stop-color='yellow' stop-opacity='1'/>
    </radialGradient>
    <radialGradient id='MyGradient2' spreadMethod='repeat'
                    cx='50%' cy='50%' r='5%' fx='50%' fy='50%'>
      <stop id='c1' offset='5%' stop-color='red' stop-opacity='1'/>
      <stop id='c2' offset='95%' stop-color='yellow' stop-opacity='1'/>
    </radialGradient>
    <radialGradient id='MyGradient3' spreadMethod='reflect'
                    cx='50%' cy='50%' r='5%' fx='25%' fy='25%'>
      <stop id='c1' offset='5%' stop-color='red' stop-opacity='1'/>
      <stop id='c2' offset='95%' stop-color='yellow' stop-opacity='1'/>
    </radialGradient>
    <radialGradient id='MyGradient4' spreadMethod='repeat'
```

```
                    cx='50%' cy='50%' r='35%' fx='5%' fy='5%'>
      <stop id='c1' offset='5%' stop-color='red' stop-opacity='1'/>
      <stop id='c2' offset='95%' stop-color='yellow' stop-opacity='1'/>
    </radialGradient>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="fill:url(#MyGradient1)"/>
  <rect x="150" y="0" width="150" height="150"
        style="fill:url(#MyGradient2)"/>
  <rect x="300" y="0" width="150" height="150"
        style="fill:url(#MyGradient3)"/>
  <rect x="450" y="0" width="150" height="150"
        style="fill:url(#MyGradient4)"/>
  <text x="75" y="175" style="text-anchor:middle">
    reflect 50 50 5 50 50
  </text>
  <text x="225" y="175" style="text-anchor:middle">
    repeat 50 50 5 50 50
  </text>
  <text x="375" y="175" style="text-anchor:middle">
    reflect 50 50 5 25 25
  </text>
  <text x="525" y="175" style="text-anchor:middle">
    repeat 50 50 35 5 5
  </text>
</svg>
```

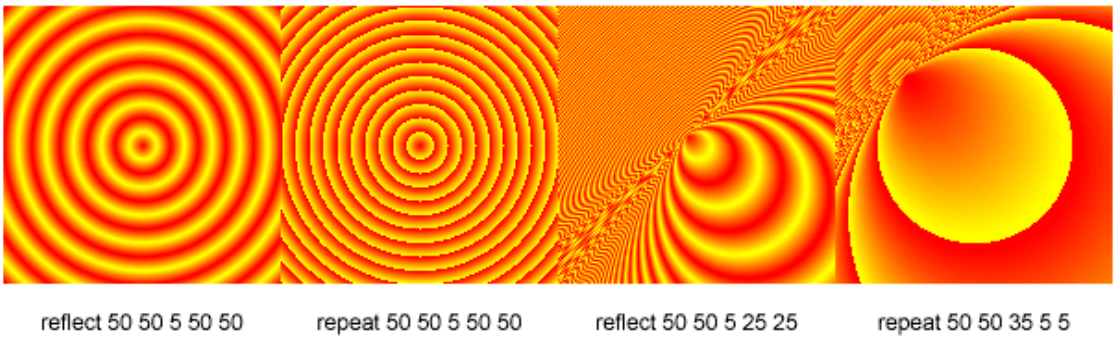

Figure 7-15. 'spreadMethod' et point focal

Gradients et filtres

Nous pouvons appliquer des filtres aux gradients. Pour avoir un effet de lumière et de relief nous pouvons utiliser stop-opacity="0" pour certains éléments 'stop'.
La figure 7-16 montre un gradient circulaire, l'éclairage de ce gradient et la combinaison des deux pour obtenir un effet de relief.

Source du fichier svg :

```
<svg width="660" height="300" viewBox="-30 -30 660 300">
  <defs>
    <radialGradient id='gradient' spreadMethod='repeat'
                    cx='50%' cy='50%' r='10%' fx='50%' fy='50%'>
      <stop id='c1' offset='5%' stop-color='red' stop-opacity='1'/>
      <stop id='c2' offset='95%' stop-color='yellow' stop-opacity='0'/>
    </radialGradient>
    <filter id='Filter0' filterUnits='objectBoundingBox'
            x='0%' y='0%' width='100%' height='100%'>
      <feImage result="pict0" xlink:href="#Image2"/>
      <feDiffuseLighting result='pict1' in='pict0'
                         lighting-color='white' diffuseConstant='1'
                         kernelUnitLength='1,1' surfaceScale='4.2'>
```

```
        <feDistantLight azimuth='60' elevation='25'/>
      </feDiffuseLighting>
      <feComposite in2='pict1' in='pict0' operator='xor'/>
    </filter>
    <filter id='Filter1' filterUnits='objectBoundingBox'
            x='0%' y='0%' width='100%' height='100%'>
      <feImage result="pict0" xlink:href="#Image1"/>
      <feDiffuseLighting result='pict1' in='pict0'
                         lighting-color='white' diffuseConstant='1'
                         kernelUnitLength='1,1' surfaceScale='4.2'>
        <feDistantLight azimuth='60' elevation='25'/>
      </feDiffuseLighting>
    </filter>
    <rect id="Image1" x="200" y="0" width='200' height='200'
          fill='url(#gradient)'/>
    <rect id="Image2" x="400" y="0" width='200' height='200'
          fill='url(#gradient)'/>
  </defs>
  <rect x='0' y='0' width='200' height='200' fill='url(#gradient)'/>
  <rect x="200" y="0" width='200' height='200' filter='url(#Filter1)'/>
  <rect x="400" y="0" width='200' height='200' filter='url(#Filter0)'/>
  <text x="100" y="225" style="text-anchor:middle">radial gradient</text>
  <text x="300" y="225" style="text-anchor:middle">lighting gradient</text>
  <text x="500" y="225" style="text-anchor:middle">
    composite pictures
  </text>
</svg>
```

Figure 7-16. Composer un gradient avec un éclairage

Comment créer des gradients ?

Dans la plupart des outils de dessin exportant en SVG, nous pouvons choisir un gradient parmi une palette, mais il est plus intéressant de pouvoir tester tous les attributs.
Aussi, vous trouverez en ligne (pilat.free.fr) un outil pour créer des gradients, tester les paramètres et récupérer le code (en utilisant PHP)
La figure 7-15 est une copie d'écran de cet outil pour la partie gradient circulaire.
Cet outil existe également comme composant que vous pouvez utiliser dans vos applications.

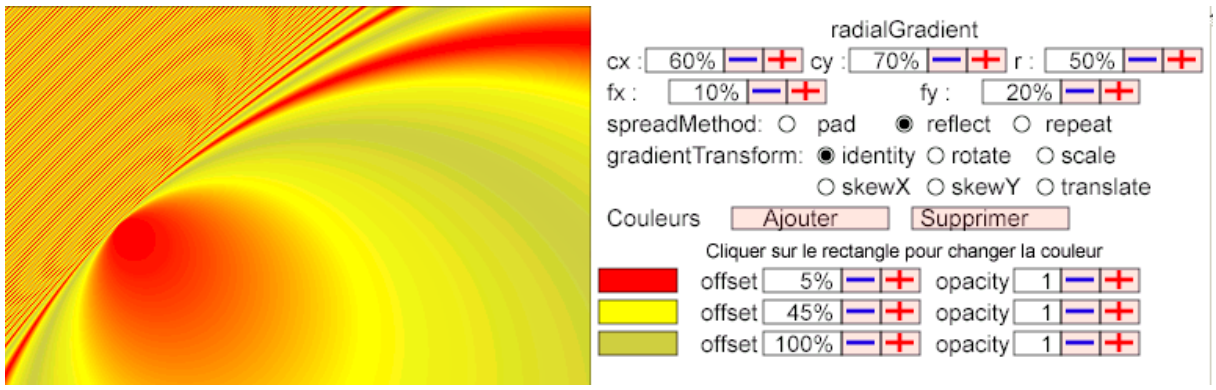

Figure 7-17. Outil pour créer des gradients circulaires

Motifs

L'élément 'pattern'

Un élément 'pattern' est utilisé pour remplir ou tracer un objet en utilisant un graphique prédéfini qui peut être répliqué pour couvrir toute la forme à peindre.

Voici la syntaxe pour l'élément 'pattern' :

```
<pattern  id="name"
          patternUnits=" userSpaceOnUse|objectBoundingBox"
          patternContentUnits=" userSpaceOnUse|objectBoundingBox"
          patternTransform="transform-list"
          viewBox="min-x min-y width height"
          x="NumberOrPercentage"
          y="NumberOrPercentage"
          width="NumberOrPercentage"
          height="NumberOrPercentage">
  <!--  some objects as pattern content  -->
</pattern>
```

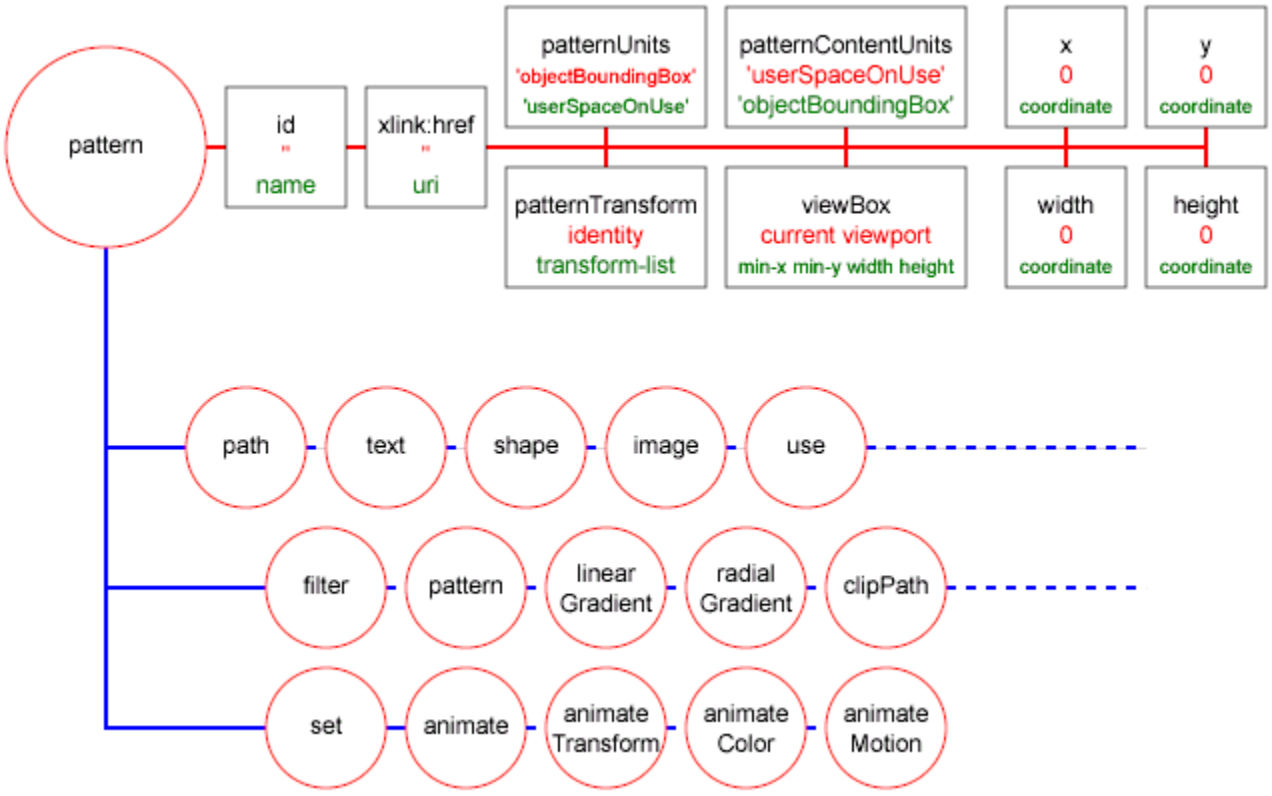

Diagramme 7-4. L'élément 'pattern'

Dans les chapitres précédents vous avez vu des figures avec un fond quadrillé.
Ce quadrillage est créé en utilisant un élément 'pattern' :

```
<pattern id="Pat01" width="10" height="10" patternUnits="userSpaceOnUse">
  <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
        stroke-width="0.1"/>
</pattern>
```

Nous définissons l'élément 'pattern', lui donnons les attributs x (0 par défaut), y (0 par défaut), width (0 par défaut) et height (0 par défaut) qui définissent la zone du motif de base comme objet rectangle et comme descendants donnons les éléments du dessin du motif.

L'attribut '**patternUnits**' peut être 'userSpaceOnUse' ou 'objectBoundingBox', il définit le système de coordonnées pour les attributs **x y width** et **height** du motif de base.
Avec 'userSpaceOnUse', les valeurs sont définies dans le repère courant.
Avec "objectBoundingBox", les valeurs sont définies dans le repère attaché à la trace de l'élément auquel est appliqué l'élément 'pattern'. Dans ce cas il est plus facile d'utiliser des pourcentages.

L'attribut '**x**' donne l'abscisse du sommet gauche supérieur du rectangle de base (0 par défaut)
L'attribut '**y**' donne l'ordonnée du sommet gauche supérieur du rectangle de base (0 par défaut)
L'attribut '**width**' donne la largeur du rectangle de base (0 par défaut)
L'attribut '**height**' donne la hauteur du rectangle de base (0 par défaut))

L'attribut '**patternContentUnits**' peut être 'userSpaceOnUse' ou 'objectBoundingBox', il définit le système de coordonnées pour le contenu de la forme de base.
Ici, 'userSpaceOnUse' est la valeur par défaut.

L'attribut '**viewBox'** s'applique au contenu de l'élément 'pattern'. Avec 'viewBox' défini, l'attribut 'patternContentUnits' n'a aucun effet.

L' attribut **'patternTransform'** ajoute une transformation au système de coordonnées de l'élément 'pattern'. Vous pouvez utiliser 'translate(tx,ty)' 'rotate(angle,cx,cy)' 'skewX(angle)' 'skewY(angle)' 'scale(sx,sy)' ou 'matrix(a b c d e f)'.

La figure 7-18 montre quelques effets de 'patternTransform' :

Source du svg:

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <pattern id="Pat01" width="10" height="10"
             patternUnits="userSpaceOnUse">
      <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
            stroke-width="0.1"/>
    </pattern>
    <pattern id="Pat02" width="10" height="10"
            patternUnits="userSpaceOnUse" patternTransform="rotate(45)">
      <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
            stroke-width="0.1"/>
    </pattern>
    <pattern id="Pat03" width="10" height="10"
             patternUnits="userSpaceOnUse" patternTransform="scale(2)">
      <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
            stroke-width="0.1"/>
    </pattern>
    <pattern id="Pat04" width="10" height="10"
             patternUnits="userSpaceOnUse" patternTransform="skewX(45)">
      <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
            stroke-width="0.1"/>
    </pattern>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="stroke:black;fill:url(#Pat01)"/>
  <rect x="150" y="0" width="150" height="150"
```

```
        style="stroke:black;fill:url(#Pat02)"/>
  <rect x="300" y="0" width="150" height="150"
        style="stroke:black;fill:url(#Pat03)"/>
  <rect x="450" y="0" width="150" height="150"
        style="stroke:black;fill:url(#Pat04)"/>
  <text x="75" y="175" style="text-anchor:middle">identity</text>
  <text x="225" y="175" style="text-anchor:middle">rotate(45)</text>
  <text x="375" y="175" style="text-anchor:middle">scale(2)</text>
  <text x="525" y="175" style="text-anchor:middle">skewX(45)</text>
</svg>
```

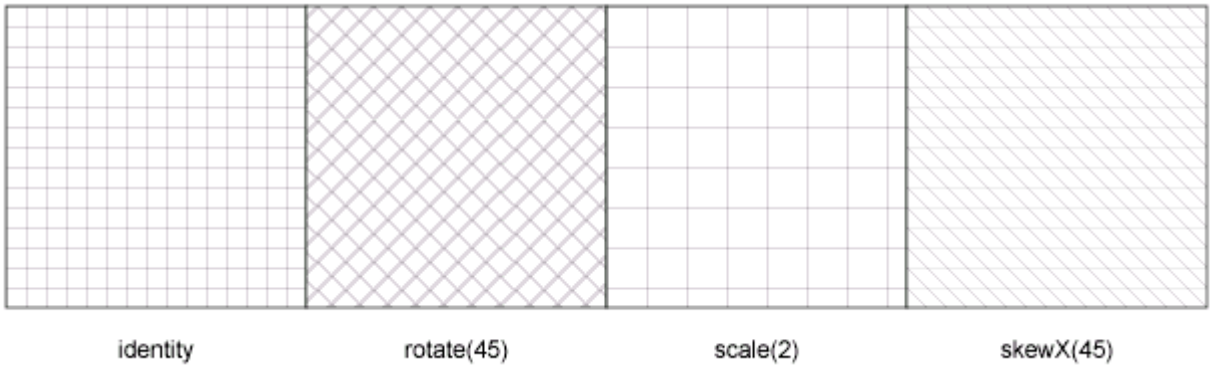

Figure 7-18. Effets avec 'patternTransform'

Le contenu de l'élément 'pattern' peut être formé de n'importe quels objets – forme de base, courbe, texte, gradient, filtre, masque, symbole, marqueur .....
Dans notre exemple, un simple rectangle pout tout contenu.

Comment utiliser les motifs?

Nous définissons l'élément 'pattern' dans une section <defs>, lui donnons un identificateur "motif1" par exemple.

```
<pattern id="motif1" width="10" height="10" patternUnits="userSpaceOnUse">
  <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
        stroke-width="0.1"/>
</pattern>
```

Nous utilisons ce motif pour remplir ou tracer un contour pour n'importe quel objet graphique ou texte. Ce rectangle sera rempli avec le motif:

```
<rect x='0' y='0' width='200' height='200' fill='url(#motif1)'/>
```

Vous pouvez aussi utiliser un attribut 'style':

```
<rect x='0' y='0' width='200' height='200'
style='stroke:black;fill:url(#motif1)'/>
```

La figure 7-19 montre quelques exemples d'utilisation de ce motif.

Source du svg:

```
<svg width="640" height="220" viewBox="-20 -20 640 220">
  <defs>
    <pattern id=" motif1 " width="10" height="10"
             patternUnits="userSpaceOnUse">
      <rect width="10" height="10" fill="#FFFFFF" stroke="#000000"
```

```
            stroke-width="0.1"/>
    </pattern>
  </defs>
  <rect x="0" y="0" width="150" height="150"
        style="stroke:black;fill:url(#motif1)"/>
  <circle cx="225" cy="75" r="60"
          style="fill:none;stroke-width:10;stroke:url(#motif1)"/>
  <path d="M320 20l 70 0 0 60 20 50 -100 0z"
        style="stroke:black;fill:url(#motif1)"/>
  <text x="525" y="100" style="stroke:black;fill:url(#motif1);
        font-family:Balloon;text-anchor:middle;font-size:100">SVG</text>
  <text x="75" y="175" style="text-anchor:middle">fill rectangle</text>
  <text x="225" y="175" style="text-anchor:middle">stroke circle</text>
  <text x="375" y="175" style="text-anchor:middle">fill path</text>
  <text x="525" y="175" style="text-anchor:middle">fill text</text>
</svg>
```

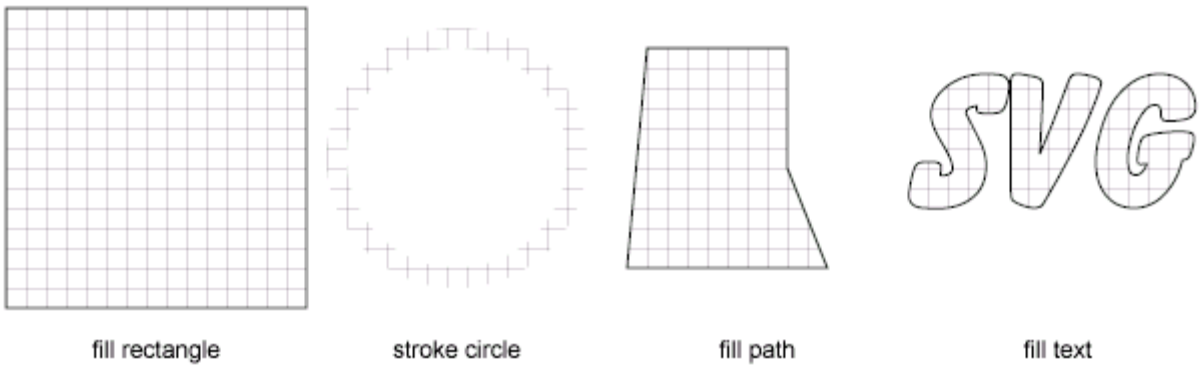

Figure 7-19. 'Fill' et 'stroke' avec 'pattern'

La figure 7-20 montre quelques exemples classiques de motifs utilisés en cartographie ou pour les schémas statistiques.

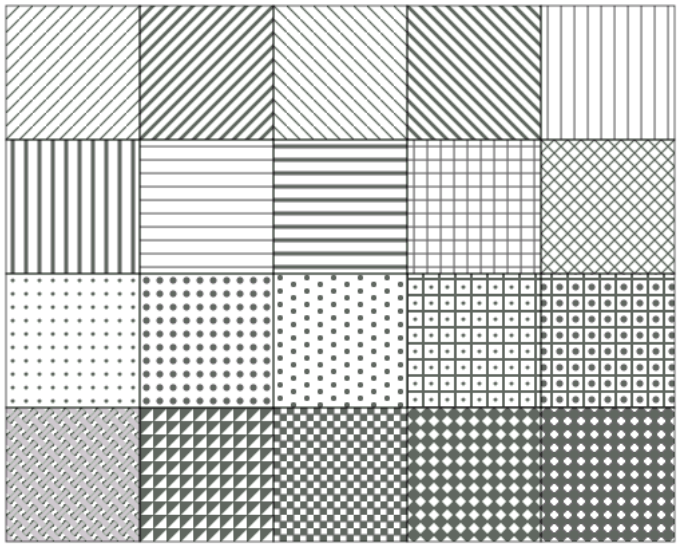

Figure 7-20. Quelques motifs classiques

Comment créer des motifs?

Dans les outils de dessin, vous pouvez choisir un motif, mais rarement créer votre propre motif. Vous pourrez trouver en ligne (pilat.free.fr) ou sur le CD accompagnant ce livre, un outil pour créer des motifs à partir d'éléments graphiques rectangle, ellipse, polygone, ligne brisée et courbe de Bézier, remplir ou tracer avec un gradient que vous créerez vous-même les éléments du motif, voir le résultat immédiatement et récupérer le code du motif. Cette application est 100% SVG-ECMAScript.

La figure 7-21 est une copie d'écran de cet outil

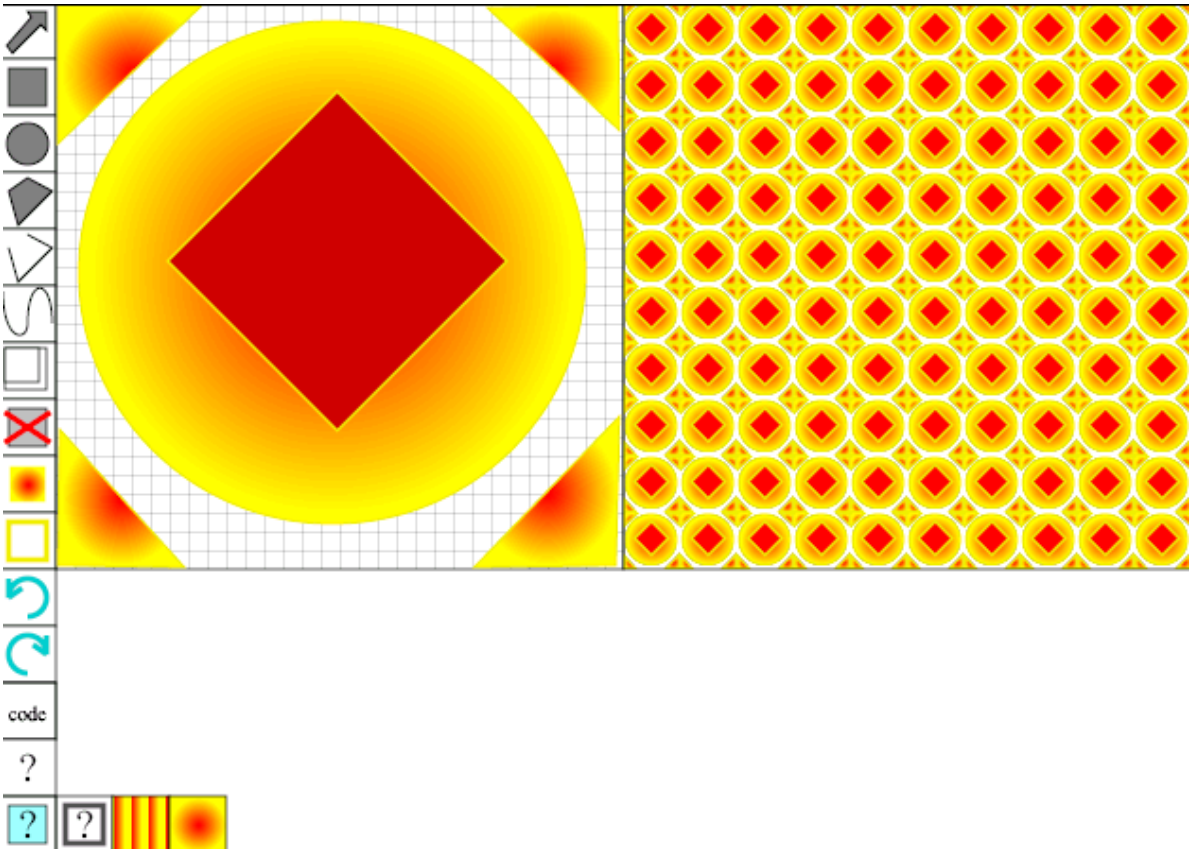

Figure 7-21. Outil de création de motifs

Mettre un filtre sur un motif

Vous pouvez ajouter un filtre dans le contenu de votre motif ou à l'objet qui l'utilise.

La figure 7-22 montre deux effets intéressants sur un motif très simple constitué d'un cercle. Nous utilisons un éclairage sur le motif que nous composons avec le motif lui-même.

Source du svg:

```
<svg width="650" height="270" viewBox="-30 -30 650 270">
  <defs>
    <pattern id='motif' x='0' y='0' width='20' height='20'
             patternUnits="userSpaceOnUse">
      <circle cx='5' cy='5' r='5' style='stroke:black;
            stroke-width:1;fill:red'/>
      <circle cx='15' cy='15' r='5' style='stroke:black;
            stroke-width:1;fill:red'/>
    </pattern>
    <filter id='Filter0' x="0%" y="0%" width="100%" height="100%">
      <feImage result='pict0' xlink:href='#Image1'/>
      <feDiffuseLighting result='pict1' in='pict0'
                         lighting-color='white' diffuseConstant='1'
                         kernelUnitLength='1,1' surfaceScale='1'>
        <feDistantLight azimuth='45' elevation='45'/>
      </feDiffuseLighting>
      <feComposite result='pict2' in2='pict0' in='pict1'
                   operator='over'/>
```

```
    </filter>
    <rect id='Image1' x="200" y="0" width='200' height='200'
         fill="url(#motif)"/>
    <rect id='Image2' x="400" y="0" width='200' height='200'
         fill="url(#motif)"/>
    <filter id='Filter1' x="0%" y="0%" width="100%" height="100%">
      <feImage result='pict0' xlink:href='#Image2'/>
      <feDiffuseLighting result='pict1' in='pict0'
                         lighting-color='white' diffuseConstant='1'
                         kernelUnitLength='1,1' surfaceScale='1'>
        <feDistantLight azimuth='95' elevation='35'/>
      </feDiffuseLighting>
      <feComposite result='pict2' in2='pict0' in='pict1' operator='in'/>
    </filter>
  </defs>
  <rect x="0" y="0" width="200" height="200" fill="url(#motif)"/>
  <rect x='200' y='0' width='200' height='200' filter='url(#Filter0)'/>
  <rect x='400' y='0' width='200' height='200' filter='url(#Filter1)'/>
  <text x="100" y="225" style="text-anchor:middle">pattern</text>
  <text x="300" y="225" style="text-anchor:middle">lighting</text>
  <text x="500" y="225" style="text-anchor:middle">other composite</text>
</svg>
```

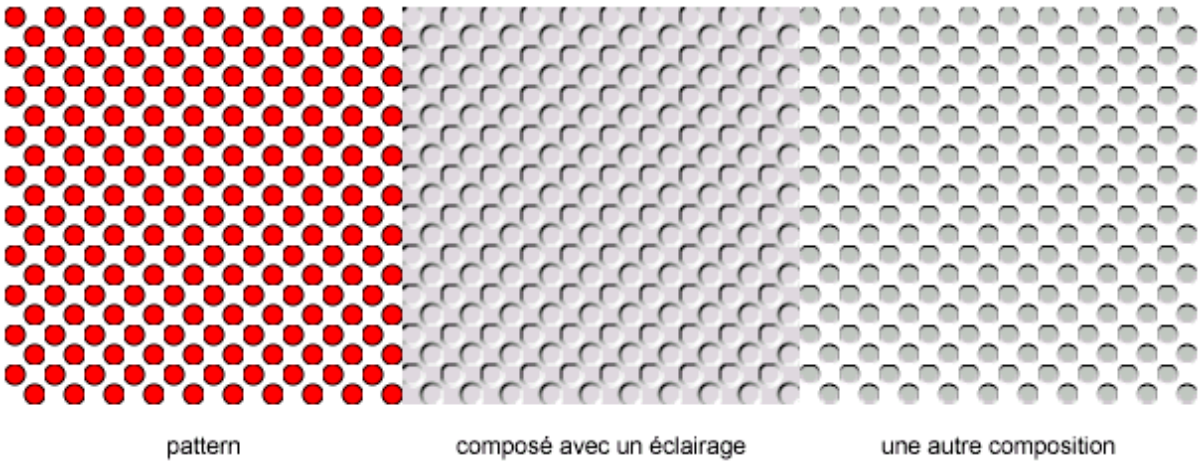

Figure 7-22. Motifs et filtres

La figure 7-23, montre une carte d'Europe utilisant ce motif pour donner un effet 'Lego' à la carte. Seule la couleur de la lumière change pour obtenir les deux motifs utilisés.

Figure 7-23. Carte d'Europe avec effet 'Lego'

Pour aller un peu plus loin avec les motifs

L'élément 'pattern' crée un pavage rectangulaire du plan, seule la translation est utilisée pour paver le plan. Pourquoi ne pas essayer de créer des pavages du plan plus complexes comme ceux de M.C. Escher ?

Chez le même fournisseur, vous trouverez un outil pour créer des pavages de types différents à partir d'objets graphiques de base et utilisant les transformations. La figure 7-24 en montre un exemple créé avec quelques courbes de Bézier, le pavage utilisant ensuite une combinaison de filtres pour donner un effet de relief.

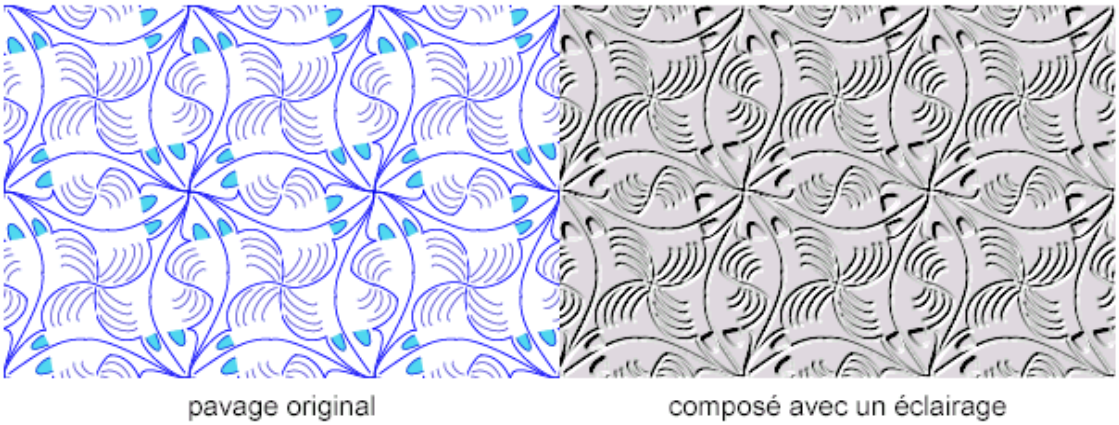

Figure 7-24. Pavage du plan et filtre

Masques

Le 'clipping path' restreint la région dans laquelle les objets seront dessinés. Toutes les parties des objets en dehors du masque ne seront pas rendues.

Définir un masque avec 'clipPath'

Nous utilisons l'élément **"clipPath"**.

La syntaxe:

```
<clipPath  id="name"
           clipPathUnits = "userSpaceOnUse | objectBoundingBox">
  <!-- formes courbes texte 'use' ... comme descendants -->
</clipPath>
```

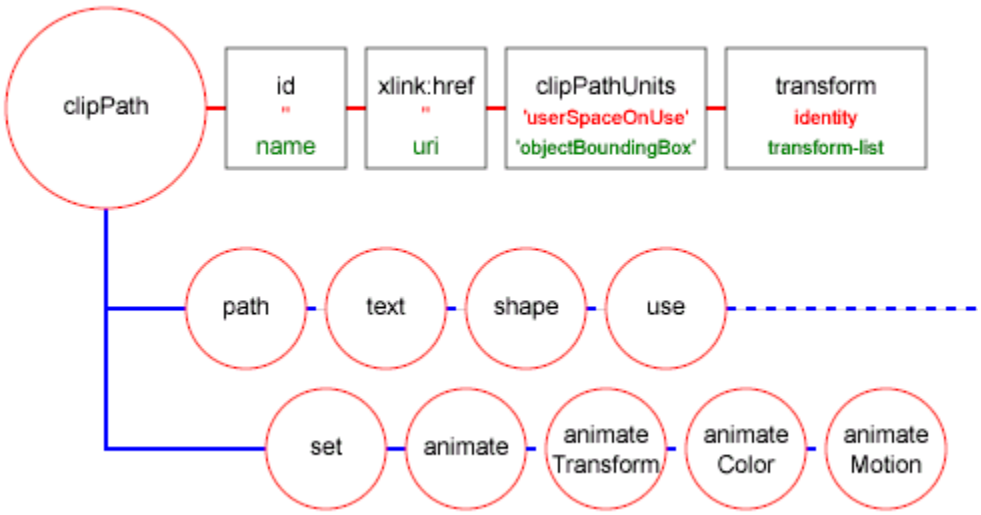

Diagram 7-5. Syntaxe de 'clipPath'

L'attribut **'clipPathUnits'** définit le système de coordonnées pour le contenu de l'élément 'clipPath'.
Avec "userSpaceOnUse", le repère sera le système de coordonnées de référence.
Avec "objectBoundingBox", valeur par défaut, le système sera défini par le rectangle dans lequel l'objet qui utilise 'clipPath' est inscrit.

Comment appliquer ce masque?

Pour des formes complexes et définir les points qui sont intérieurs, nous pouvons choisir pour la propriété **'clip-rule'** 'nonzero' ou 'evenodd'. Ce sont les mêmes valeurs que pour 'fill-rule' et ces paramètres sont expliqués en détail aux chapitres 2 et surtout 4.

'clip-rule'

Value:	nonzero | evenodd | inherit
Par défaut:	nonzero
S'applique à:	Eléments graphiques descendants d'un élément 'clipPath'
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel

Animable: oui

Nous utilisons la propriété '**clip-path**'
(ne pas confondre avec l'élément clipPath qui définit le masque):

Sa syntaxe:

'clip-path'

Valeur:	<uri> | none | inherit
Par défaut:	none
S'applique à:	containers et éléments graphiques
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

La figure 7-25 montre un cercle sur un texte, la couleur du texte à l'intérieur du cercle change comme si ce cercle était un spot éclairant une partie du texte. Ceci est de plus animé.

Dans une section <defs> nous définissons :
- le texte en noir
- l'élément clipPath defini par un cercle

Nous dessinons le rectangle noir, le texte en blanc et avec un élément <use> faisant référence au texte en noir et utilisant dans la propriété clip-path l'élément 'clipPath' nous affichons le texte de couleur noir dans le cercle délimité par le masque.

Code de cet exemple :

```
<svg width="550" height="130"  viewBox="-50 -30 550 130">
  <defs>
    <g id="texte_spot">
      <rect x="0" y="0" width="500" height="100" fill="white"/>
      <text x="250" y="60" style="text-anchor:middle;font-size:35;
            font-family:Arial;fill:black">Scalable Vector Graphics</text>
    </g>
    <clipPath id="spot">
      <circle cx="200" cy="50" r="50"/>
    </clipPath>
  </defs>
  <rect x="0" y="0" width="500" height="100" fill="black"/>
  <text x="250" y="60" style="text-anchor:middle;font-size:35;
        font-family:Arial;fill:white;fill-opacity:1">
    Scalable Vector Graphics
  </text>
  <use clip-path="url(#spot)" xlink:href="#texte_spot"/>
</svg>
```

Pour animer le spot, nous animons l'attribut 'cx' du cercle utilisé dans l'élément 'clipPath' de 50 à 450 ...

Voici le code pour l'animation, seule la définition de l'élément 'clipPath' change. Nous verrons les possibilités d'animation en détail au chapitre 9.

```
<clipPath id="spot">
      <circle cx="200" cy="50" r="50" clip-rule="evenodd">
```

```
            <animate attributeName="cx" from="0" to="500"
                     repeatCount="indefinite" begin="0" dur="5"/>
      </circle>
</clipPath>
```

Figure 7-25. Spot sur un texte

Une liste de sélection avec scrolling

Nous pouvons utiliser les masques pour une liste de sélection avec scrolling des options.

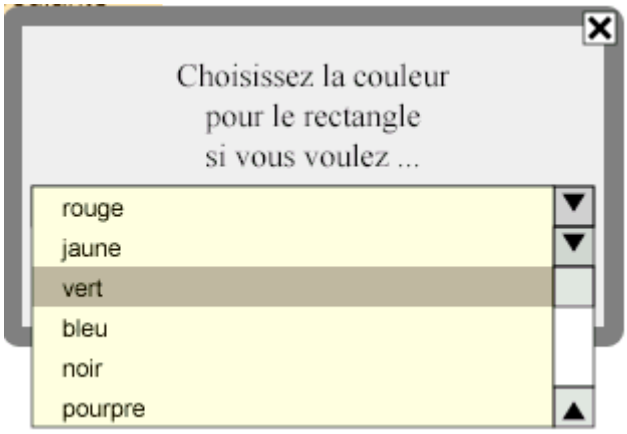

Figure 7-26. Liste de sélection avec scrolling

La figure 7-26 nous montre cette liste de sélection avec scrolling.
Ce composant SVG réutilisable est en deux parties, le script et les objets graphiques que l'utilisateur peut modifier.
Les options sont passées en paramètre et ajoutées par le script dans la section <defs>:

```
<defs>
  <g id="scroll_textes" transform="translate(0,90)"
     style="text-anchor:left;font-size:12;font-family:Arial;fill:black">
    <text x="10" y="15" startOffset="0">red</text>
    <text x="10" y="35" startOffset="0">yellow</text>
    <!--   autres éléments texte -->
   </g>
   <clipPath id="txt_spot">
     <rect id="txt_list" x="0" y="90" width="260" height="120"/>
   </clipPath>
</defs>
```

L'élément 'clipPath' définit la partie de la liste qui sera rendue.
Quand l'utilisateur clique sur les flèches de défilement ou déplace le curseur, le script change le paramètre 'y' de l'élément 'clipPath'.

Bandit manchot

Nous pouvons simuler une machine à sous ou bandit manchot en utilisant 'clipPath'.

La figure 7-27 montre cette animation qui fait défiler les chiffres sur les roues et sortir un nombre entre 0 et 999 de manière aléatoire.
Le script ne sert qu'à initialiser les roues pour avoir un résultat aléatoire.

Nous utilisons deux animations :
- la première s'applique à l'élément 'clipPath' pour changer les nombres visibles.
- la seconde s'applique au groupe utilisant 'clipPath' pour que les nombres visibles s'affichent toujours au même emplacement.

Code de cet exemple:

```
<svg width='200' height='200'>
  <script type="text/ecmascript">
    <![CDATA[
      function alea(evt)
      {
        svgdoc=evt.target.ownerDocument;
        for (i=1;i<=3;i++)
        {
        n=Math.floor(10*Math.random())-1;
        node=svgdoc.getElementById("move_"+i.toString());
        node.setAttribute("from",20+40*n);
        node.setAttribute("to",40*n+380);
        node=svgdoc.getElementById("number_"+i.toString());
        node.setAttribute("from",20+50*(i-1)+","+(-40*n));
        node.setAttribute("to",20+50*(i-1)+","+(-360-40*n));
        }
      }
    ]]>
  </script>
  <defs>
    <style type="text/css">
      <![CDATA[
        .lettre {text-anchor:middle;font-family:Verdana;font-weight:bold;
                 font-size:30;stroke:black}
      ]]>
    </style>
    <text id="ico1" class="lettre" x="25" y="30">1</text>
    <text id="ico2" class="lettre" x="25" y="30">2</text>
    <text id="ico3" class="lettre" x="25" y="30">3</text>
    <text id="ico4" class="lettre" x="25" y="30">4</text>
    <text id="ico5" class="lettre" x="25" y="30">5</text>
    <text id="ico6" class="lettre" x="25" y="30">6</text>
    <text id="ico7" class="lettre" x="25" y="30">7</text>
    <text id="ico8" class="lettre" x="25" y="30">8</text>
    <text id="ico9" class="lettre" x="25" y="30">9</text>
    <text id="ico0" class="lettre" x="25" y="30">0</text>
    <g id="rolling_numbers">
      <use x="0" y="0" xlink:href="#ico1"/>
      <use x="0" y="40" xlink:href="#ico2"/>
      <use x="0" y="80" xlink:href="#ico3"/>
      <use x="0" y="120" xlink:href="#ico4"/>
      <use x="0" y="160" xlink:href="#ico5"/>
      <use x="0" y="200" xlink:href="#ico6"/>
      <use x="0" y="240" xlink:href="#ico7"/>
      <use x="0" y="280" xlink:href="#ico8"/>
      <use x="0" y="320" xlink:href="#ico9"/>
      <use x="0" y="360" xlink:href="#ico0"/>
      <use x="0" y="400" xlink:href="#ico1"/>
      <use x="0" y="440" xlink:href="#ico2"/>
      <use x="0" y="480" xlink:href="#ico3"/>
      <use x="0" y="520" xlink:href="#ico4"/>
      <use x="0" y="560" xlink:href="#ico5"/>
      <use x="0" y="600" xlink:href="#ico6"/>
```

```
      <use x="0" y="640" xlink:href="#ico7"/>
      <use x="0" y="680" xlink:href="#ico8"/>
      <use x="0" y="720" xlink:href="#ico9"/>
      <use x="0" y="760" xlink:href="#ico0"/>
      <use x="0" y="800" xlink:href="#ico1"/>
      <use x="0" y="840" xlink:href="#ico2"/>
      <use x="0" y="880" xlink:href="#ico3"/>
    </g>
  </defs>
  <rect x='0' y='0' width='200' height='200' style='stroke:green;fill:yellow'/>
  <rect x="20" y="20" width="150" height="80" fill="white"/>
  <path d="M70 20l0 80 M120 20l0 80" stroke="black"/>
  <clipPath id="spot1">
    <rect x="0" y="20" width="50" height="80">
      <animate id="move_1" attributeName="y" dur="4s" repeatCount="2"
               fill="freeze" from="0" to="240" begin="go.click"/>
    </rect>
  </clipPath>
  <clipPath id="spot2">
    <rect x="0" y="60" width="50" height="80">
      <animate id="move_2" attributeName="y" dur="3s" repeatCount="3"
               fill="freeze" from="40" to="280" begin="go.click"/>
    </rect>
  </clipPath>
  <clipPath id="spot3">
    <rect x="0" y="100" width="50" height="80">
      <animate id="move_3" attributeName="y" dur="2s" repeatCount="4"
               fill="freeze" from="80" to="320"  begin="go.click"/>
    </rect>
  </clipPath>
  <g transform="translate(20,0)" clip-path="url(#spot1)">
    <use xlink:href="#rolling_numbers"/>
      <animateTransform id="number_1" attributeName="transform"
                        type="translate"
                        from="20,20" to="20,-220" fill="freeze" dur="4s"
                        repeatCount="2" begin="go.click"/>
  </g>
  <g transform="translate(70,-40)" clip-path="url(#spot2)">
    <use xlink:href="#rolling_numbers"/>
      <animateTransform id="number_2" attributeName="transform"
                        type="translate"
                        from="70,-20" to="70,-260" fill="freeze" dur="3s"
                        repeatCount="3" begin="go.click"/>
  </g>
  <g transform="translate(120,-80)" clip-path="url(#spot3)">
    <use xlink:href="#rolling_numbers"/>
      <animateTransform id="number_3" attributeName="transform"
                        type="translate"
                        from="120,-60" to="120,-300" fill="freeze" dur="2s"
                        repeatCount="4" begin="go.click"/>
  </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"
        onclick="alea(evt)"/>
</svg>
```

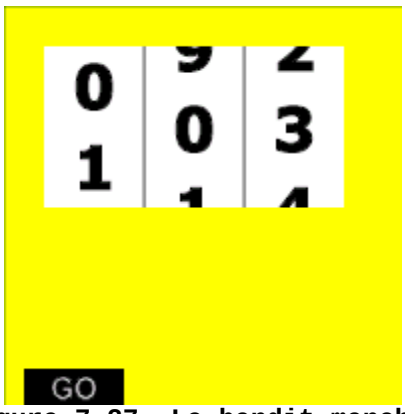

Figure 7-27. Le bandit manchot

Masque et puzzle

Nous créons un puzzle avec un peu de programmation. Dans cet exemple, l'utilisateur peut choisir l'image pour les pièces, le nombre de pièces, leur découpage.

Quand le svg est chargé, les éléments 'clipPath' sont créés dans une section <defs> sous forme d'un groupe "clips", les éléments 'path' qui les définissent sont dans le fichier "clip_paths.js". La function initiale crée aussi les pièces du puzzle dans le groupe "pieces".

Source complet de ce jeu:

```
<svg width="800" height="450" onload="init(evt)">
  <defs>
    <g id="picture">
      <image id="source" x="0" y="0" width="400" height="400"
             xlink:href="puzzle.jpg"/>
    </g>
    <g id="clips">
    </g>
  </defs>

// Chargement des découpes des pièces:

  <script type="text/ecmascript" xlink:href="clip_paths.js" />

  <script type="text/ecmascript">
    <![CDATA[


// Quelques variables à initialiser

      var svgdoc="",xd1,xd2,yd1,yd2,num;
      var click_piece=false,cible="", nb_trials=0;
      var played=new Array();

// Variables que l'utilisateur peut modifier pour avoir plus ou moins de pièces

      var size=5,nb_pieces=25,large=80,

// La fonction init définit svgdoc, la valeur de large et de nb_pieces.
// appelle create_pieces pour créer les pièces et distribution pour les ranger

    function init(evt)
      {
      svgdoc=evt.target.ownerDocument;
```

```
      large=Math.round(400/size);nb_pieces=size*size;
      create_pieces(evt);
      distribution(evt);
      }

// Crée pièces, les éléments clipPath puis les pièces avec <use>

    function create_pieces(evt)
      {
      contents = svgdoc.getElementById ('clips');
      contents2 = svgdoc.getElementById ('pieces');
      k=-1;
      for (i=0;i<size;i++)
        {
        for (j=0;j<size;j++)
          {
          k+=1;
          node=svgdoc.createElement("clipPath");
          name="clip_path"+(1+i+size*j).toString();
          node.setAttribute("id",name);
          contents.appendChild(node);
          node2=svgdoc.createElement("path");
          node2.setAttribute("d",chemin[k]);
          node.appendChild(node2);
          node=svgdoc.createElement("use");
          node.setAttribute("id","tile"+(1+i+size*j).toString());
          node.setAttribute("transform","translate(400,0)");
          name="url(#clip_path"+(1+i+size*j).toString()+")";
          node.setAttribute("clip-path",name);
          node.setAttributeNS('http://www.w3.org/2000/xlink/namespace/',
                              "xlink:href","#picture");
          contents2.appendChild(node);
          }
        }
      }

// Dispose les pièces de manière aléatoire

    function distribution(evt)
      {
      var place_allowed=new Array();
      for (i=0;i<size;i++)
        {
        for (j=0;j<size;j++)
          {
          place_allowed[i+size*j]=0
          }
        };
      for (i=1;i<=nb_pieces;i++)
        {
        played[i]=0
        };
      for (i=0;i<size;i++)
        {
        for (j=0;j<size;j++)
          {
          while (place_allowed[i+size*j]==0)
            {
            k=1+Math.floor(nb_pieces*Math.random());
            if (played[k]<1)
              {
              played[k]=played[k]+1;
              place_allowed[i+size*j]=k
              }
            };
            line=Math.floor((k-1)/size);
            row=k-1-size*line;
```

```
            dx=400-large*(line-i);
            dy=large*(j-row);
            transfo="translate("+dx.toString()+","+dy.toString()+")";
            node = svgdoc.getElementById("tile"+k.toString());
            node.setAttribute("transform",transfo);
          }
        }
      for (i=1;i<=nb_pieces;i++)
        {
        played[i]=0
        };
      nb_trials=0;
      node=svgdoc.getElementById("trials");
      child=node.firstChild;
      child.setData(nb_trials)
}


// Pour déplacer les pièces
// Désactive la pièce

    function mouse_up(evt)
      {
      click_piece=false
      }

// Active la pièce, incrémente nb_trials, écrit la nouvelle valeur
// Détermine la position de la pièce au moment du click


    function mouse_down(evt)
      {
      if (click_piece==false)
        {
        cible=evt.target.getAttribute("id");
        click_piece=true;
        nb_trials+=1;
        node=svgdoc.getElementById("trials");
        child=node.firstChild;
        child.setData(nb_trials);
        xm=evt.getClientX();
        ym=evt.getClientY();
        num=parseInt(cible.substring(4,cible.length));
        objet=svgdoc.getElementById("tile"+num.toString());
        transfo=new String(objet.getAttribute("transform"));
        posi2=transfo.lastIndexOf(",");
        tranx=transfo.substring(10,posi2);
        xd1=parseInt(tranx,10);
        longueur=transfo.length-1;
        trany=transfo.substring(posi2+1,longueur);
        yd1=parseInt(trany,10);
        xd2=xm;yd2=ym
        }
      }

// Déplace la pièce (x_pointer+xd1-xd2,y_pointer+yd1-yd2)
// Si la pièce est proche de sa position correcte – moins de 5 pixels
// la pièce est placée correctement, la variable played[num] mise à 1


    function mouse_move(evt)
      {
      xm=evt.getClientX();
      ym=evt.getClientY();
      if (click_piece==true)
        {
        depx=xm+xd1-xd2;
```

```
        depy=ym+yd1-yd2;
        if ((Math.abs(depx)<5)&&(Math.abs(depy)<5))
          {
          depx=0;
          depy=0;
          played[num]=1
          }
          else
          {
          played[num]=0
          };
        transfo="translate("+depx+","+depy+")";
        node = svgdoc.getElementById("tile"+num.toString());
        node.setAttribute("transform",transfo)
        }
      }


// Place les pièces correctement sur la demande de l'utilisateur

    function solution(evt)
      {
      for (i=1;i<=nb_pieces;i++)
        {
        node = svgdoc.getElementById("tile"+i.toString());
        node.setAttribute("transform","translate(0,0)");
        }
      }

// Donne le résultat du jeu

    function bilan(evt)
      {
      var win_game=true;
     for (i=1;i<=nb_pieces;i++)
        {
        if (played[i]==0)
          {
          win_game=false
          }
        };
      if (win_game==true)
        {
        alert("Vous gagnez en "+nb_trials+" coups")
        }
        else
        {
        alert("Vous n'avez pas fini!")
        }
      }

    ]]>
  </script>

  <rect x="0" y="0" width="800" height="450" fill="green" opacity="0.3"/>
  <rect x="10" y="410" width="100" height="20" style="fill:white"/>
  <text x="60" y="425" style="text-anchor:middle;font-size:15;
        font-family:Arial;fill:red">Result</text>
  <rect x="10" y="410" onclick="bilan(evt)" width="100" height="20"
        style="fill:green;fill-opacity:0"/>
  <rect x="120" y="410" width="120" height="20" style="fill:white"/>
  <text x="180" y="425" style="text-anchor:middle;font-size:15;
        font-family:Arial;fill:red">New game</text>
  <rect x="120" y="410" onclick="distribution(evt)" width="120" height="20"
        style="fill:green;fill-opacity:0"/>
  <rect x="250" y="410" width="120" height="20" style="fill:white"/>
  <text x="310" y="425" style="text-anchor:middle;font-size:15;
```

```
        font-family:Arial;fill:red">Solution</text>
  <rect x="250" y="410" onclick="solution(evt)" width="120" height="20"
        style="fill:green;fill-opacity:0"/>
  <text x="450" y="425" style="text-anchor:left;font-size:15;
        font-family:Arial;fill:black">Trials: </text>
  <text id="trials" x="580" y="425" style="text-anchor:right;font-size:15;
        font-family:Arial;fill:black">0</text>
  <g onmouseup="mouse_up(evt)">
    <use opacity="0.3" xlink:href="#picture"/>
    <g id="pieces" onmousemove="mouse_move(evt)"
       onmousedown="mouse_down(evt)">
    </g>
  </g>
</svg>
```

Quand le svg est chargé, la section <defs> est complétée:

```
<g id="clips">
  <clipPath id="clip_path1">
    <path d="M0 0 l80 0 c-10,10 -10,25 0,40 c10,10 10,30 0,40 c-5,-10
             -25,-10 -30,0 a10,10 0 1,1 -10,0 c-5,-5 -35,-5 -40,0 l0 -80z"/>
  </clipPath>
  <clipPath id="clip_path6">
    <path d="M80 0 c-10,10 -10,25 0,40 c10,10 10,30 0,40 c5,10 25,
             10 30,0 a10,10 0 1,1 10,0 c5,5 35,5 40,0 l-5 -35 a10,10 0 1,0
             0,-10 l5 -35z"/>
  </clipPath>
  <!--  autres clipPath     -->
</g>
```

ainsi que le groupe "pieces" :

```
<g id="pieces" onmousemove="mouse_move(evt)" onmousedown="mouse_down(evt)">
  <use id="tile1" transform="translate(400,80)"
       clip-path="url(#clip_path1)" xlink:href="#picture"/>
  <use id="tile6" transform="translate(320,160)"
       clip-path="url(#clip_path6)" xlink:href="#picture"/>
  <!--  autres pièces pour le puzzle  -->
</g>
```

En utilisant une forme HTML, l'utilisateur peu choisir une image pour les pièces sur son disque local.

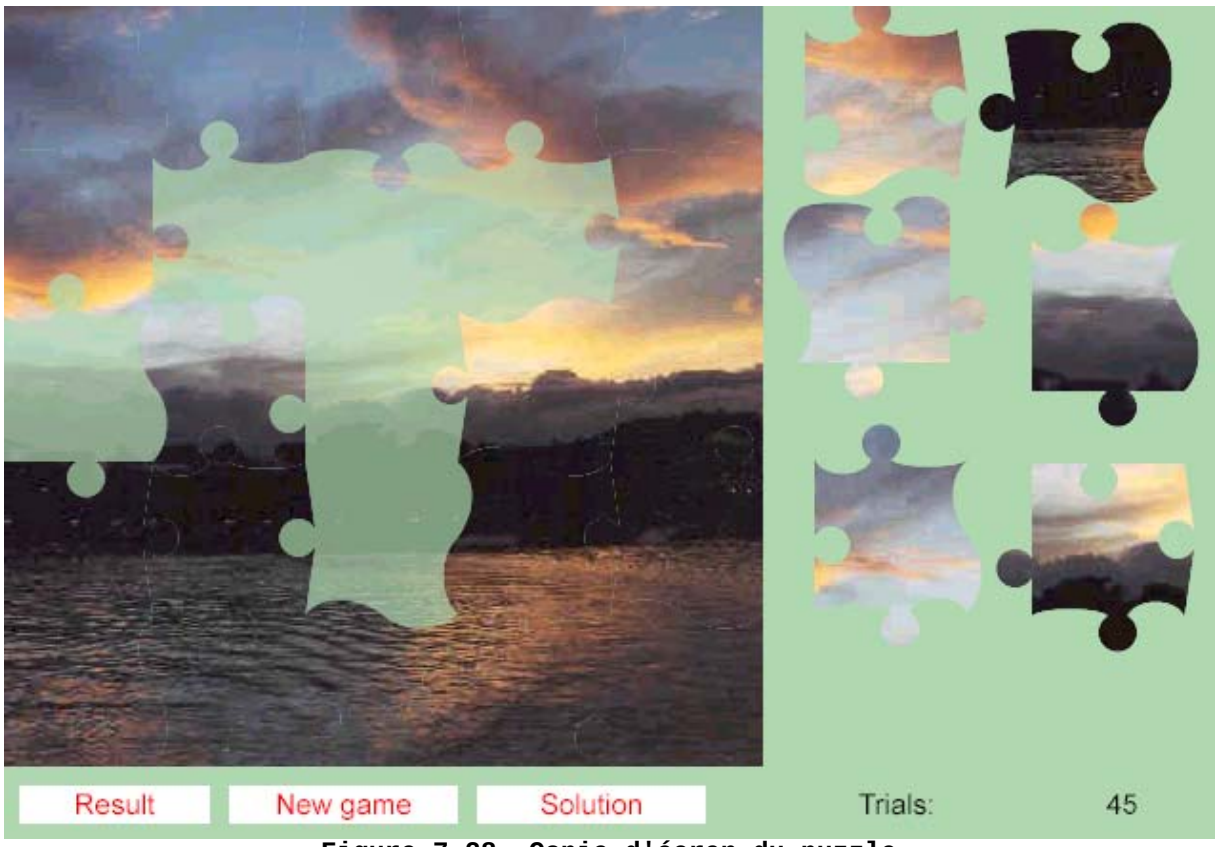

Figure 7-28. Copie d'écran du puzzle

Masques avec 'mask'

Nous verrons la différence entre les éléments 'mask' et 'clipPath'.

Voici la syntaxe de l'élément "mask":

```
<mask  id="name"
          maskUnits=" userSpaceOnUse|objectBoundingBox"
          maskContentUnits=" userSpaceOnUse|objectBoundingBox"
          x="NumberOrPercentage"
          y="NumberOrPercentage"
          width="NumberOrPercentage"
          height="NumberOrPercentage">
  <!--    contenu graphique du masque  -->
</mask>
```

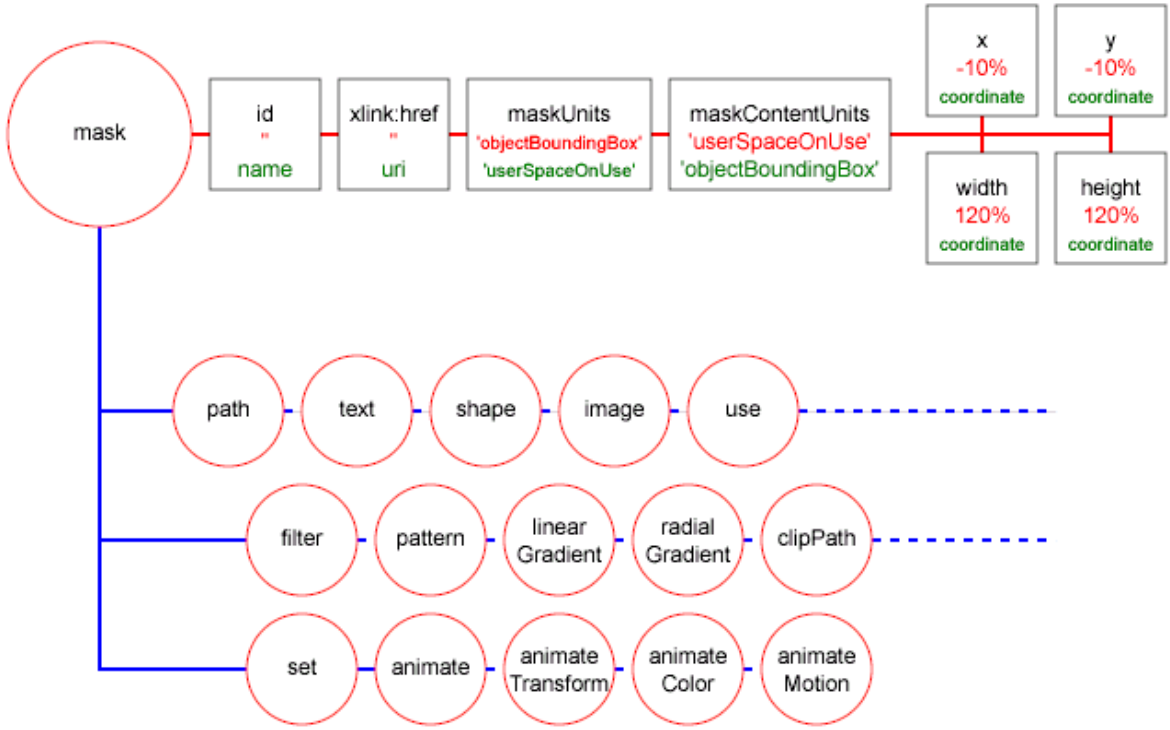

Diagram 7-6. Syntaxe de 'mask'

L'attribut '**maskUnits**' est "userSpaceOnUse" ou "objectBoundingBox", il définit le système de coordonnées pour les attributs '**x**' '**y**' '**width**' et '**height**' qui définissent la zone maximale où agira le masque, ceci est nécessaire car la zone doit être stockée dans un buffer avant l'application du masque.
Avec "userSpaceOnUse", les valeurs sont définies dans le repère de référence.
Avec "objectBoundingBox", valeur par défaut, le repère est défini par le rectangle dans lequel s'inscrit l'objet qui appelle le masque. Dans ce cas il est plus simple d'utiliser les pourcentages.

x abscisse du coin supérieur gauche de la zone (-10% by default)

y ordonnée du coin supérieur gauche de la zone (-10% by default)

width largeur de la zone (120% by default)

height hauteur de la zone (120% by default)

L'attribut '**maskContentUnits'** peut être "userSpaceOnUse" ou "objectBoundingBox", il définit le repère pour les coordonnées des objets descendants du masque qui forment son contenu. Ici la valeur par défaut est "userSpaceOnUse".

Pour appliquer le masque, nous utilisons la propriété 'mask':

'mask'

Valeur:	<uri> | none | inherit
Par défaut:	none
S'applique à:	container et éléments graphiques
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

Nous aurons le même résultat qu'à la figure 7-25 avec ce code:

```
<svg width="550" height="130"  viewBox="-50 -30 550 130">
  <defs>
    <g id="new_text">
      <rect x="0" y="0" width="500" height="100" fill="white"/>
      <text x="250" y="60" style="text-anchor:middle;font-size:35;
            font-family:Arial;fill:black">Scalable Vector Graphics</text>
    </g>
    <mask id="spot" maskUnits="userSpaceOnUse" x="0" y="0" width="500"
         height="100">
      <circle cx="200" cy="50" r="50" fill="white"/>
    </mask>
  </defs>
  <rect x="0" y="0" width="500" height="100" fill="black" fill-opacity="1"/>
  <text x="250" y="60" style="text-anchor:middle;font-size:35;
        font-family:Arial;fill:white;fill-opacity:1">
    Scalable Vector Graphics
  </text>
  <use mask="url(#spot)" xlink:href="#new_text"/>
</svg>
```

Quelles différences entre 'clipPath' et 'mask' ?

'**clipping path'** restreint la région où seront dessinés les objets graphiques qui l'utilisent.
Le résultat ne dépend que de la géométrie des objets qui composent l'élément 'clipPath'.
Si vous remplissez, appliquez un radient, un filtre à ces objets, le résultat est le même.

Pour l'élément '**mask',** les objets qui le composent servent de masque alpha composé avec les objets qui l'utilisent.
La géométrie des objets fixe les limites du dessin, mais le dessin est modifié par les caractéristiques des composants du masque (couleur de remplissage par exemple)

La figure 7-29 montre deux masques simples:

```
<mask id="spot" maskUnits="userSpaceOnUse" x="0" y="0" width="500"
      height="100">
```

```
  <circle cx="200" cy="50" r="50" fill="white"/>
</mask>
```

pour le haut

```
<mask id="spot" maskUnits="userSpaceOnUse" x="0" y="0" width="500"
      height="100">
  <circle cx="200" cy="50" r="50" fill="green"/>
</mask>
```

pour le bas

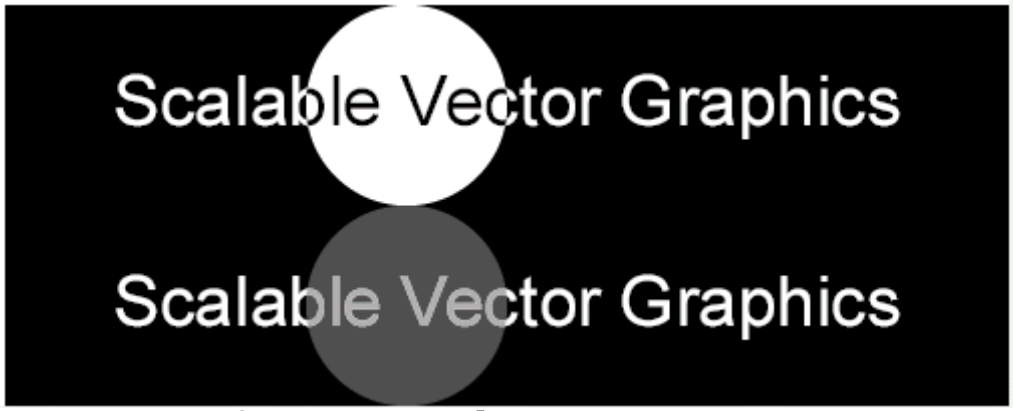

Figure 7-29. Couleurs et masques ...

La seule différence entre ces deux masques est la couleur de remplissage du cercle qui est le seul objet graphique composant le masque.