Chapitre 8 Utiliser les filtres

Les spécifications SVG définissent l'utilisation de filtres sur les objets graphiques, y compris les images bitmap.

L'élément 'Filter'

Voici la syntaxe de l'élément 'filter':

<**filter** id="name"
filterUnits="userSpaceOnUse|objectBoundingBox"
primitiveUnits=""userSpaceOnUse|objectBoundingBox"
filterRes="NumberOptionalNumber"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage">
<!--. primitives pour les filtres -->
</filter>

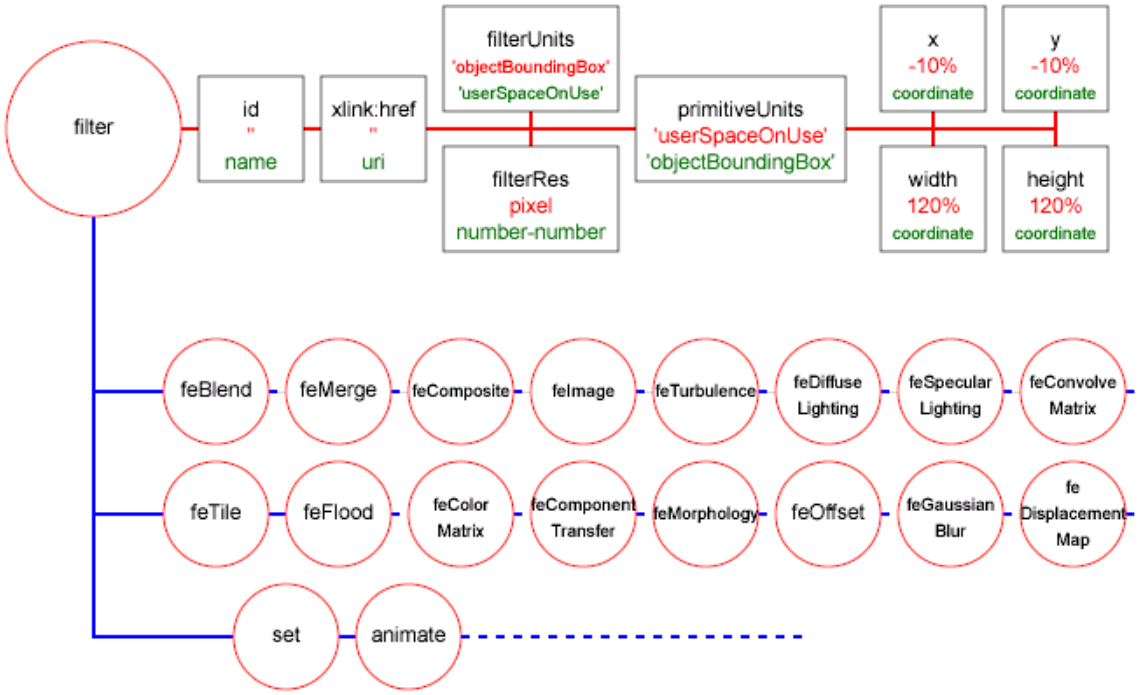

Diagram 8-1. Syntaxe de 'filter'

Il est recommandé de déclarer l'élément 'filter' dans une section <defs> et de lui donner un identificateur pour qu'il puisse être référencé:

```
<defs>
  <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0"
          width="400" height="400">
    <!-- primitives comme descendants de l'élément -->
  </filter>
</defs>
```

Attributs de l'élément 'filter'

L'attribut '**filterUnits**' est "userSpaceOnUse" ou "objectBoundingBox", il définit le système de coordonnées pour les attributs 'x' 'y' 'width' et 'height' qui définissent la zone d'action du filtre.
Avec "userSpaceOnUse", le repère est le repère de référence.
Avec "objectBoundingBox", valeur par défaut, le repère est défini par le rectangle dans lequel l'objet auquel s'applique le filtre est inscrit. Dans ce cas il est plus pratique d'utiliser des pourcentages pour 'x' 'y' 'width' et 'height'.

L'attribut '**primitiveUnits**' qui peut être "userSpaceOnUse" ou "objectBoundingBox" définira le repère pour les attributs 'x', 'y', 'width' et 'height' des primitives.

L'attribut '**filterRes**' définit la taille des images intermédiaires en pixels, images intermédiaires nécessaires au rendu du filtre. Par défaut, une valeur adaptée à la résolution du support sera donnée
.
La figure 8-1 montre une pixelisation des images dûe au choix des valeurs de 'filterRes'.
L'image traitée a une dimension de 200 pixels.

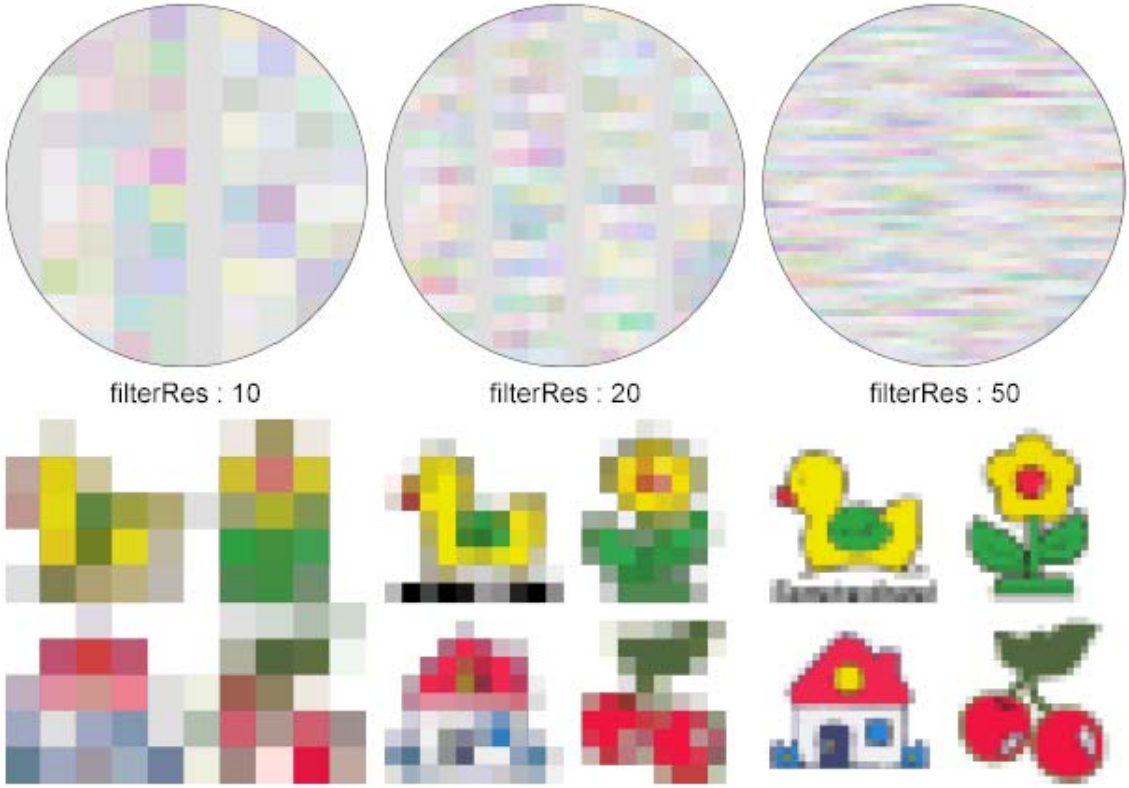

Figure 8-1. 10 20 et 50 pour 'filterRes'

Les attributs '**x**' '**y**' '**width**' '**height**' définissent une région rectangulaire à laquelle le filtre s'appliquera. Les valeurs peuvent être des nombres ou des pourcentages. Par défaut nous aurons –10% pour 'x' et 'y' et 120% pour 'width' et 'height'.

Primitives pour les filtres

Propriétés de rendu

Les primitives utilisent les propriétés de rendu des autres éléments et une propriété spécifique: 'color-interpolation-filters'

Syntaxe de 'color-interpolation-filters'

Valeur:	auto | sRGB | linearRGB | inherit
Par défaut:	linearRGB
S'applique à:	primitives des filtres
Transmissible:	oui
Pourcentages:	N/A
Media:	visuel
Animable:	oui

auto : Le visualiseur choisit 'sRGB' ou 'linearRGB' pour les effets de couleur des filtres.
sRGB : L'espace utilisé est 'sRGB'.
linearRGB : L'espace utilisé est 'linearized RGB color space'.

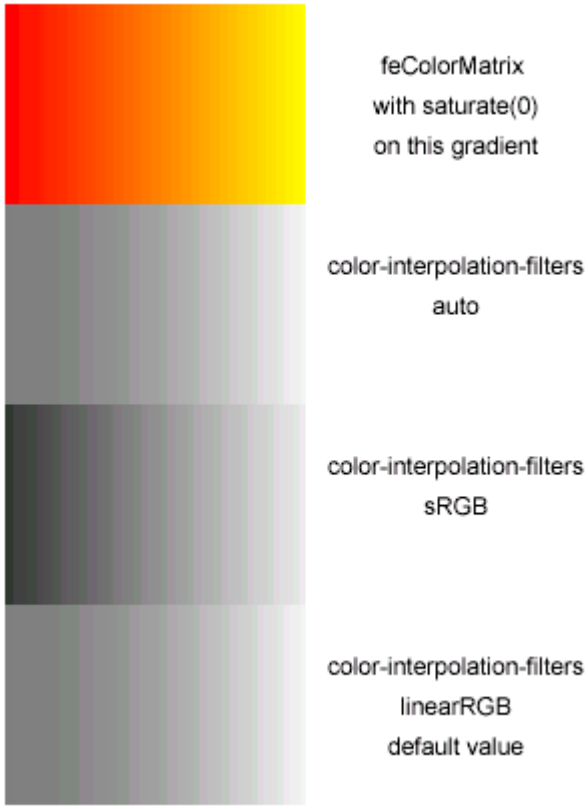

Figure 8-2. La propriété 'color-interpolation-filters'

La valeur par défaut de 'color-interpolation-filters' est 'linearRGB' alors que la valeur par défaut de 'color-interpolation' qui s'applique aux autres opérations est 'sRGB'.
Ceci peut causer quelques surprises ou résultats inattendus.

Attributs communs

'x' 'y' 'width' 'height' définissent la région rectangulaire où sera calculée et rendue la primitive. Nous pouvons donner des nombres ou des pourcentages. Par défaut nous aurons 0% pour 'x' et 'y' et 100% pour 'width' et 'height'.
L'attribut **'result'** permet de nommer le graphique résultat de l'application de la primitive. Ceci permet de le référencer dans une primitive suivante dans le même élément 'filter'.
Les attributs **'in'** '**in2'** définissent les sources pour la primitive. Leur valeur peut être l'une de ces valeurs prédéfinies - SourceGraphic | SourceAlpha | BackgroundImage | BackgroundAlpha | FillPaint | StrokePaint – ou le nom du résultat d'une précédente primitive dans le même élément 'filter'.

Si nous ne donnons pas de valeur à 'in', pour la première primitive du filtre 'SourceGraphic' sera utilisé, pour les autres primitives ce sera le résultat de la primitive précédente qui sera utilisé.

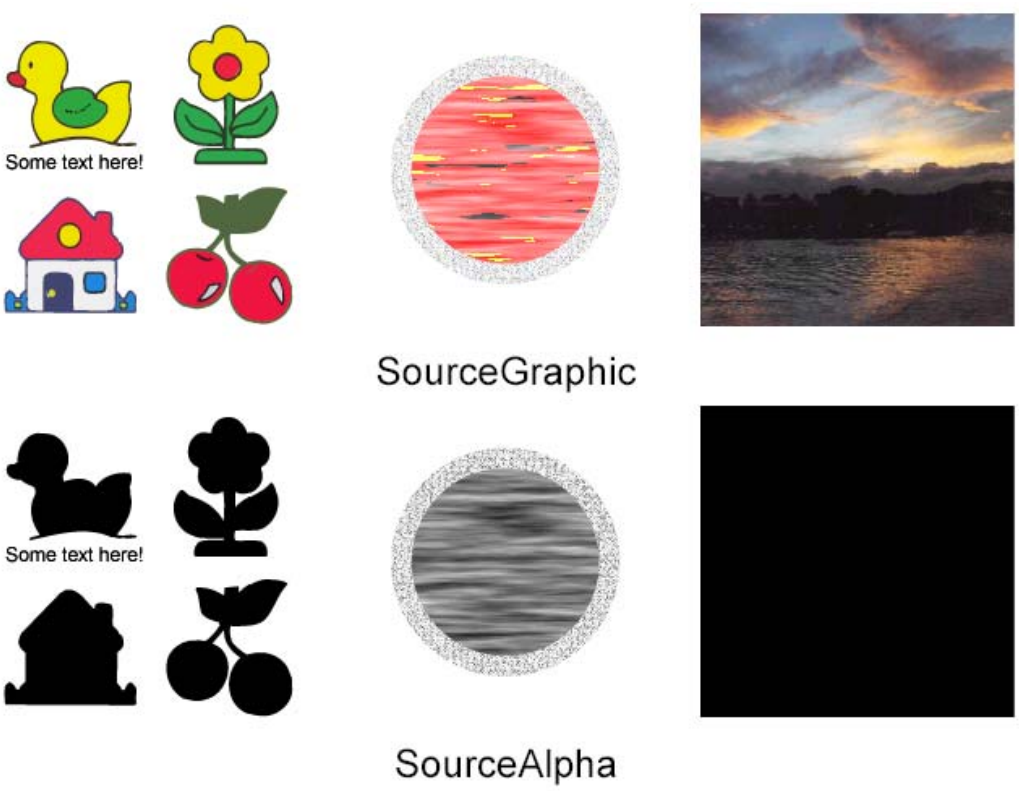

Figure 8-3. SourceGraphic et SourceAlpha

La figure 8-3 montre ce que sont 'SourceGraphic' et 'SourceAlpha' pour des objets SVG, une turbulence et une image bitmap.

Comment utiliser un filtre?

Nous utiliserons la propriété 'filter' qui peut s'appliquer à n'importe quel élément, container, graphisme, texte.

Nous ne devons pas confondre cette propriété avec l'élément 'filter'

'filter'

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

Si le filtre crée un graphique, vous pouvez écrire:

```
<rect filter="url(#MyFilter)" x='100' y='100' width='200' height='200'/>
```

Si vous voulez appliquer votre filtre à une image externe à votre document:

```
<image filter="url(#MyFilter)" x="0" y="0" width='400' height='400'
        xlink:href='fondu1.jpg'/>
```

Un filtre peut s'appliquer à un groupe:

```
<g filter="url(#MyFilter)">
  <!--  here your objects          -->
</g>
```

Vous pouvez utiliser le filtre dans un élément 'pattern' pour l'appliquer aux propriétés 'fill' et 'stroke' des objets:

```
<pattern id="motif" patternUnits="userSpaceOnUse" x="0" y="0" width="400"
         height="400">
  <rect filter="url(#MyFilter)" x="0" y="0" width="400" height="400"/>
</pattern>
<circle cx='200' cy='200' r='200' style='stroke:black;fill:url(#motif)'/>
```

Nous allons voir les différentes primitives en les classant par catégorie.

Primitives qui modifient les couleurs des objets

Nous avons deux primitives pour modifier la couleur des objets vectoriels ou des images bitmap 'feColorMatrix' et 'feComponentTransfer'.

La primitive 'feColorMatrix'

La primitive **'feColorMatrix'** utilise une matrice pour changer les couleurs RGB et le canal alpha pour chaque pixel:

| R' | | a00 a01 a02 a03 a04 | | R |

| G' | | a10 a11 a12 a13 a14 | | G |

| B' | = | a20 a21 a22 a23 a24 | * | B |

| A' | | a30 a31 a32 a33 a34 | | A |

| 1 | | 0 0 0 0 1 | | 1 |

Voici sa syntaxe:

<**feColorMatrix** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
type="matrix|hueRotate|saturate|luminanceToAlpha"
values="list of numbers"/>

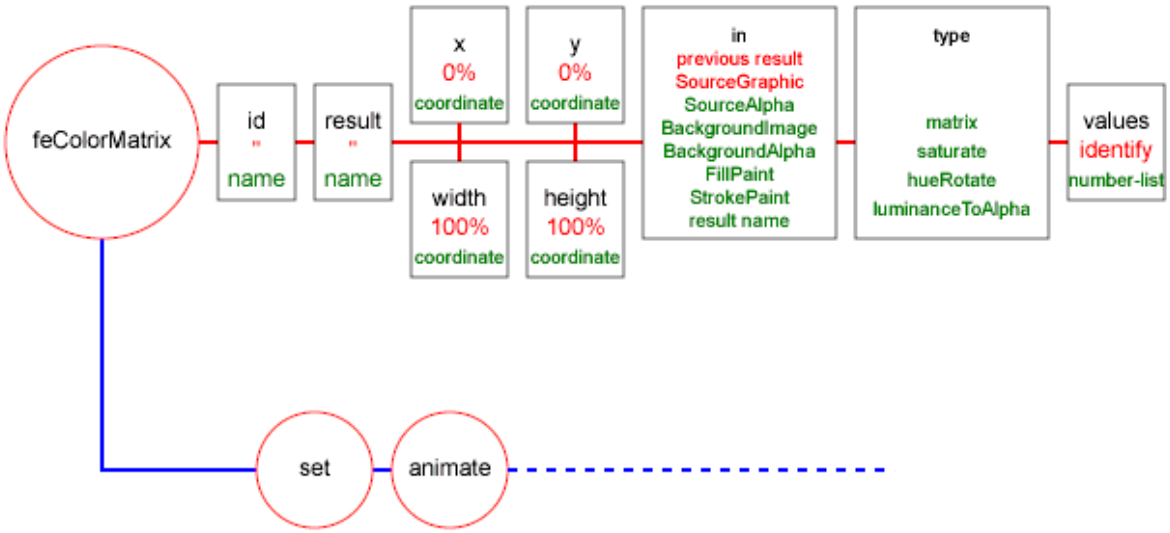

Diagram 8-2. Syntaxe de 'feColorMatrix'

L'attribut '**type**' peut être:
'**matrix**' : vous devez définir toutes les valeurs pour la matrice 5x4 (pour la plupart des termes de la matrice, seules des valeurs entre –1 et 1 donnent un effet pour les canaux RGB, et entre 0 et 1 pour le canal alpha)

'**hueRotate**' : Vous donnez un angle en degrés (0 n'a pas d'effet) et la matrice est

```
|| a00  a01  a02   0   0 |
|| a10  a11  a12   0   0 |
|| a20  a21  a22   0   0 |    avec
||  0    0    0    1   0 |
||  0    0    0    0   1 |

|| a00 a01 a02 |    [0.213   0.715   0.072]
|| a10 a11 a12 | =  [0.213   0.715   0.072] +
|| a20 a21 a22 |    [0.213   0.715   0.072]

                  [0.787   -0.715   -0.072]                    [-0.213   -0.715   0.928]
cos(angle) * [-0.213   0.285   -0.072]  + sin(angle) * [0.143   0.140    -0.283]
                  [-0.213   -0.715   0.928]                    [-0.787   0.715    0.072]
```

'**saturate**' : Vous donnez une valeur **s** entre 0 et 1
(1 ne change rien, 0 donne une image en tons de gris)
La matrice est

```
|| 0.213+0.787s  0.715-0.715s  0.072-0.072s   0   0 |
|| 0.213-0.213s  0.715+0.285s  0.072-0.072s   0   0 |
|| 0.213-0.213s  0.715-0.715s  0.072+0.928s   0   0 |
||          0             0             0        1   0 |
||          0             0             0        0   1 |
```

'**luminanceToAlpha**' : Aucun paramètre (donne un négatif noir et blanc de l'image)
La matrice est

```
||     0        0        0     0   0 |
||     0        0        0     0   0 |
||     0        0        0     0   0 |
|| 0.2125   0.7154   0.0721   0   0 |
||     0        0        0     0   1 |
```

Exemple de code avec type="matrix":

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feColorMatrix type="matrix" values="1 0 0 0 0
                                       0 1 0 0 0
                                       0 0 1 0 0
                                       0 0 0 .5 0"/>
</filter>
```

Exemple de code avec type="hueRotate":

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feColorMatrix type="hueRotate" values="45"/>
</filter>
```

La figure 8-4 montre des exemples d'effets sur une image bitmap.

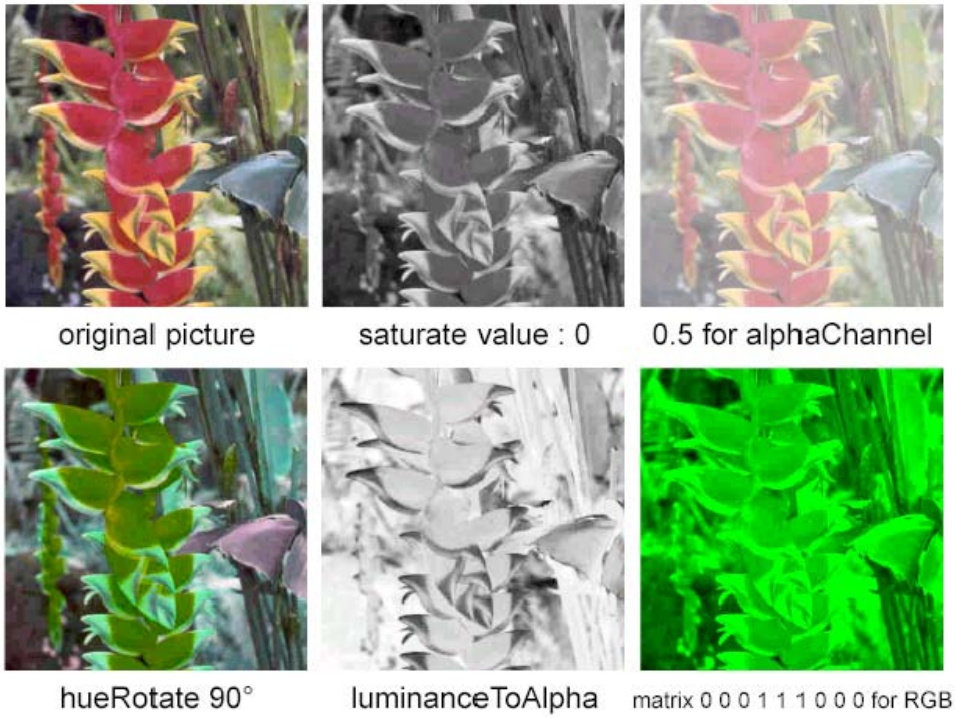

Figure 8-4. Une image bitmap avec 'feColorMatrix'

La primitive 'feComponentTransfer'

La primitive **'feComponentTransfer'** applique une fonction pour chacun des canaux RGBA en utilisant les éléments 'feFuncR', 'feFuncG', 'feFuncB' et 'feFuncA'. Cette primitive permet d'ajuster la luminosité, le contraste, labalance des couleurs pour une image.

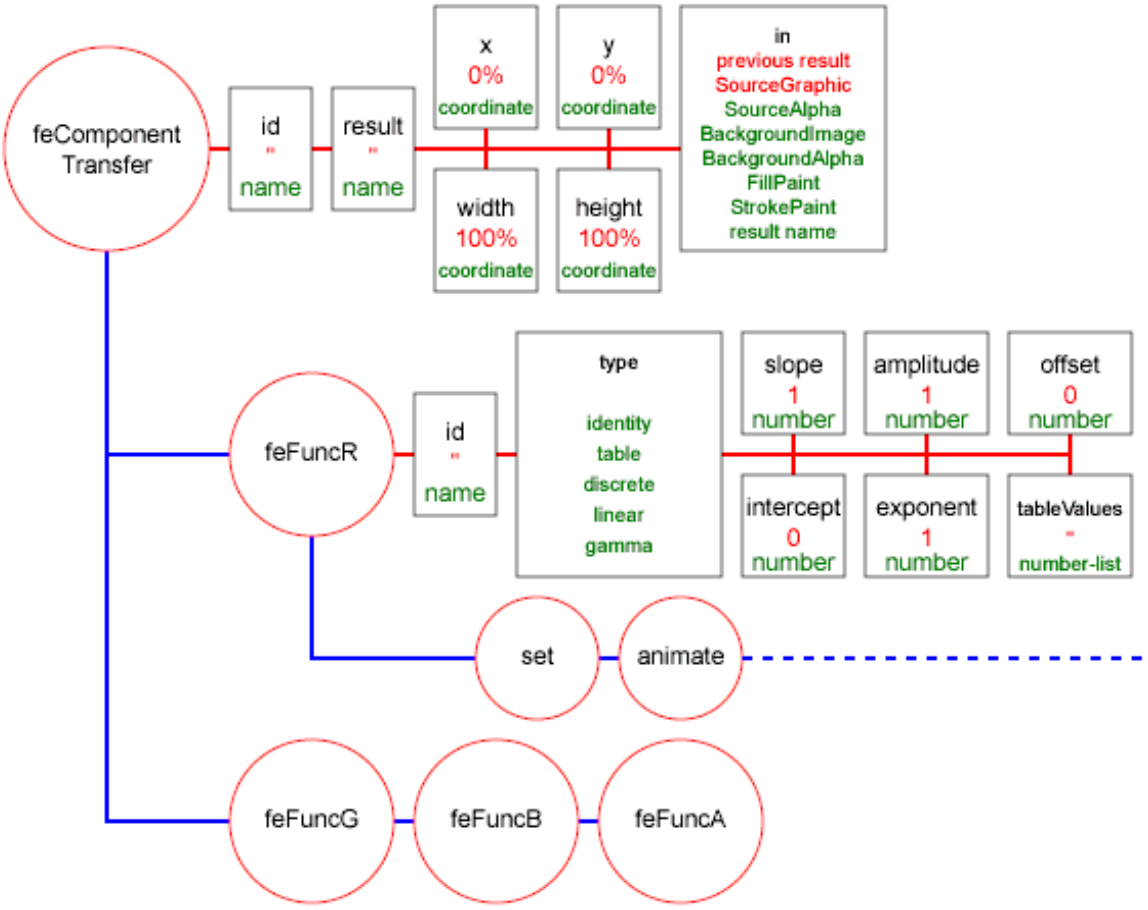

Diagram 8-3. Syntaxe de 'feComponentTransfer'

Sa syntaxe:

<**feComponentTransfer** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage">
<**feFuncR** **feFuncG** **feFuncB** or **feFuncA**
type="identity|linear|gamma|table|discrete"
slope="number"
intercept="number"
amplitude="number"
exponent="number"
offset="number"
tableValues="list of numbers"/>
</feComponentTransfer>

L'attribut '**type**' peut être:
'**identity**' : C'=C, aucun effet sur les couleurs.

'**linear**' : La fonction appliquée est C'=slope*C+intercept

Les attributs '**slope**' (1 par défaut) et '**intercept**' (0 par défaut) doivent être définis.

'**gamma**' : La fonction appliquée est C'=amplitude*pow(C,exponent)+offset
Les attributs '**amplitude**' (1 par défaut) '**exponent**' (1 par défaut) et '**offset**' (0 par défaut) doivent être définis.

'**table**' : Interpolation linéaire entre les valeurs entrées dans '**tableValues**'

'**discrete**' : Fonction en escalier définie par les valeurs entrées dans '**tableValues**'
Par défaut '**tableValues**' est vide

Voici un code où il n'y a aucune modification pour chacun des canaux :

```
<feComponentTransfer>
      <feFuncR type='identity' />
      <feFuncG type='linear' slope='1' intercept='0' />
      <feFuncB type='gamma' amplitude='1' exponent='1' offset='0'/>
      <feFuncA type='table' tableValues='1' >
</feComponentTransfer>
```

Si par exemple, il n'y a pas d'élément 'feFuncR', le canal R ne sera pas modifié.

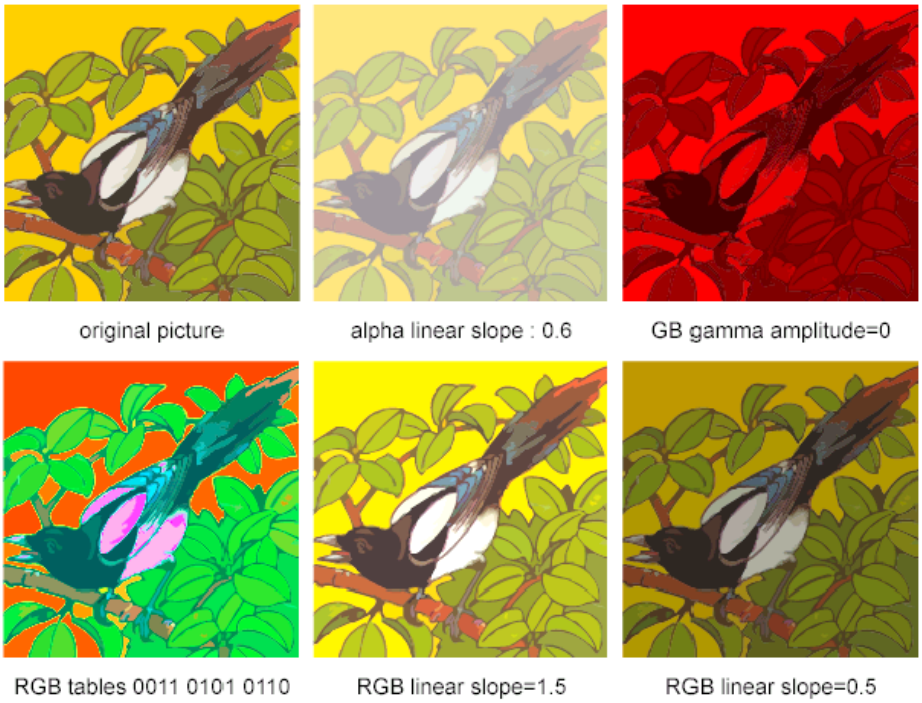

Figure 8-5. Exemples avec 'feComponentTransfer'

La figure 8-5 montre quelques effets sur des objets SVG (ce n'est pas une image bitmap).

Primitives pour créer un éclairage

Nous pouvons utiliser 'feSpecularLighting' ou 'feDiffuseLighting' après avoir choisi une ou plusieur sources lumineuses parmi 'feSpotLight', 'fePointLight' et 'feDistantLight'. Les sources lumineuses étant définies comme descendants de la primitive.

Source lumineuse 'feSpotLight'

Comme son nom l'indique, cette source simule un spot dont nous pouvons préciser la position (attributs 'x', 'y' et 'z') la direction ('pointsAtX', 'pointsAtY' et 'pointsAtZ') et l'ouverture du faisceau lumineux ('limitingConeAngle').

Syntaxe de l'élément 'feSpotLight':

<**feSpotLight** id="name"
x="number"
y="number"
z="number"
pointsAtX="number"
pointsAtY="number"
pointsAtZ="number"
specularExponent="number"
limitingConeAngle="number"/>

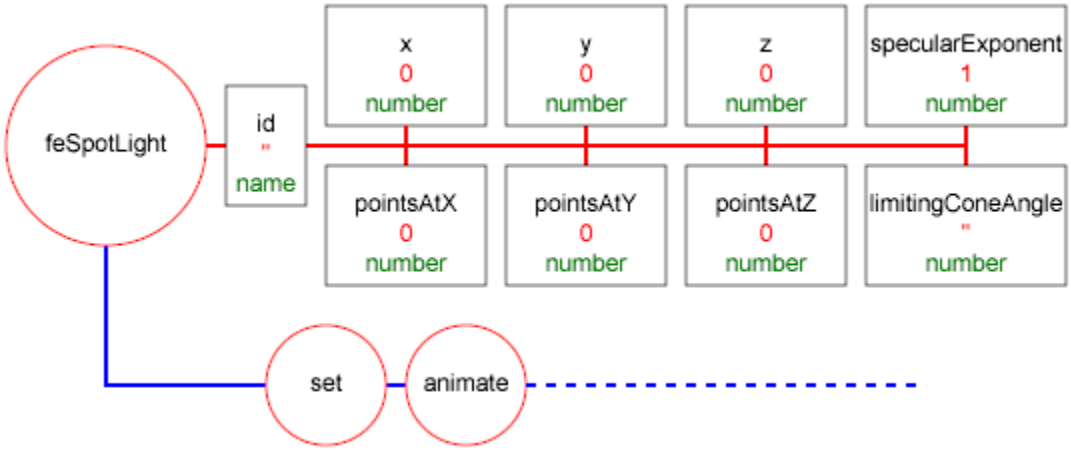

Diagram 8-4. Syntaxe de 'feSpotLight'

Les attributs de '**feSpotLight**' sont:
x = coordonnée x pour la source lumineuse dans le repère de référence (0 par défaut)
y = coordonnée y pour la source lumineuse dans le repère de référence (0 par défaut)
z = coordonnée z pour la source lumineuse dans le repère de référence étendu à l'espace en trois dimensions (0 par défaut)
pointsAtX = x du point vers lequel pointe le faisceau lumineux (0 par défaut)
pointsAtY = y du point vers lequel pointe le faisceau lumineux (0 par défaut)
pointsAtZ = z du point vers lequel pointe le faisceau lumineux (0 par défaut)
specularExponent = Exposant qui contrôle la concentration du faisceau (1 par défaut)
limitingConeAngle = Angle du cône lumineux en degrés (par défaut le cône n'est pas limité)

Source lumineuse 'fePointLight element'

La source lumineuse est symbolisée par un point d'où part la lumière. Nous ne pouvons choisir que la position de ce point.

Syntaxe de l'élément 'fePointLight':

<**fePointLight** id="name"
x="number"
y="number"
z="number"/>

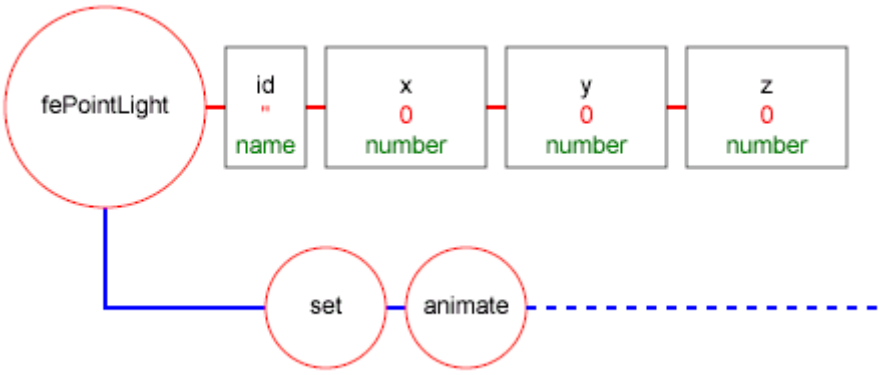

Diagram 8-5. Syntaxe de 'fePointLight'

Les attributs de '**fePointLight**' sont:
x = coordonnée x pour la source lumineuse dans le repère de référence (0 par défaut)
y = coordonnée y pour la source lumineuse dans le repère de référence (0 par défaut)
z = coordonnée z pour la source lumineuse dans le repère de référence étendu à l'espace en trois dimensions (0 par défaut)

Source lumineuse ' feDistantLight element'

Nous ne pouvons que choisir la direction d'où vient la lumière avec deux angles.
Syntaxe de l'élément 'feDistantLight':
<**feDistantLight** id="name"
azimuth="number"
elevation="number"/>

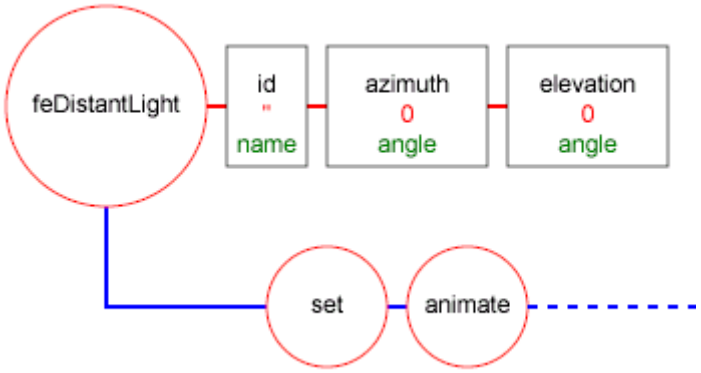

Diagram 8-6. Syntaxe de 'feDistantLight'

Les attributs de '**feDistantLight**' sont:
azimuth = Angle de la projection de la source sur le plan des objets en degrés (0 par défaut)
elevation = Angle de la source avec le plan des objets en degrés (0 par défaut)

Propriété 'lighting-color'

Cette propriété définit la couleur de la source lumineuse pour les primitives 'feDiffuseLighting' et 'feSpecularLighting'.

'lighting-color'
Valeur: currentColor |<color> [icc-color(<name>[,<icccolorValeur>]*)]

	|inherit
Par défaut:	white
S'applique à:	éléments 'feDiffuseLighting' et 'feSpecularLighting'
Transmissible:	non
Pourcentages:	N/A
Media:	visuel
Animable:	oui

La primitive 'feSpecularLighting'

Cette primitive 'feSpecularLighting' éclaire une source graphique en utilisant la canal alpha des objets. Elle crée une image RGBA basée sur la couleur de la lumière, cette image n'est pas opaque.

Syntaxe de l'élément 'feSpecularLighting':

<**feSpecularLighting** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
surfaceScale="Number"
specularConstant="Number"
specularExponent="Number"
lighting-color="property">
<éléments **feSpotLight feDistantLight** ou **fePointLight** />
</feSpecularLighting>

Nous pouvons utiliser plusieurs sources de lumière pour la primitive.

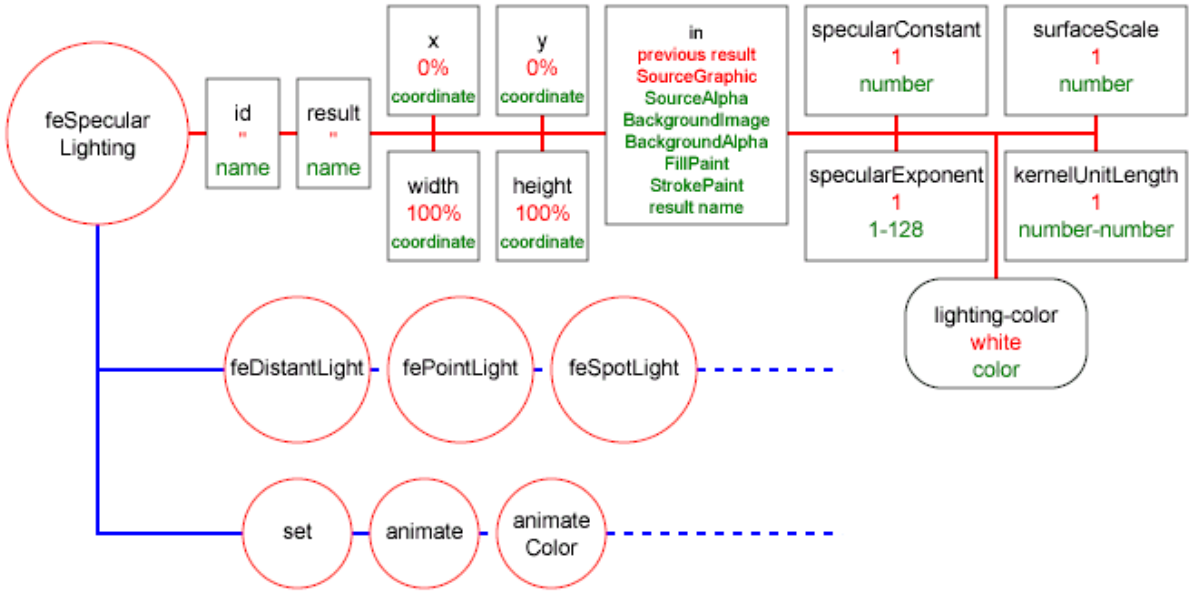

Diagram 8-7. Syntaxe de 'feSpecularLighting'

Les attributs de **'feSpecularLighting'** sont 'in', 'result' définis précédemment et:
surfaceScale : hauteur des reliefs créés (1 par défaut)
specularConstant : équivalent de 'ks' dans le modèle de Phong. En SVG, ce nombre doit être positif (1 par défaut)
specularExponent : Puissance de 2 (de 1.0 à 128.0).
Plus sa valeur est grande, plus l'éclairage est intense (1 par défaut)
lighting-color : Couleur de la lumière (blanc par défaut)

Voici un exemple de codage pour cette primitive:

```
<feSpecularLighting lighting-color='white' specularConstant ='1'
                    specularExponent='16'>
  <feSpotLight x='200' y='200' z='200' pointsAtX='200' pointsAtY='100'
               pointsAtZ='0' specularExponent='16' limitingConeAngle='45' />
</feSpecularLighting>
```

La primitive 'feDiffuseLighting'

Cette primitive 'feDiffuseLighting' éclaire une source graphique en utilisant le canal alpha des objets. L'image produite est une image RGBA opaque basée sur la couleur de la lumière. Si nous avons alpha = 1.0 sur toute l'image (une image bitmap par exemple) le résultat sera entièrement opaque.

Syntaxe de l'élément 'feDiffuseLighting':

<**feDiffuseLighting** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>″
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
surfaceScale="Number"
diffuseConstant="Number"
kernelUnitLength="NumberOptionalNumber"
lighting-color="property">
<éléments **feSpotLight feDistantLight** ou **fePointLight /**>
</feDiffuseLighting>

Nous pouvons ici aussi utiliser plusieurs sources lumineuses pour 'feDiffuseLighting'.

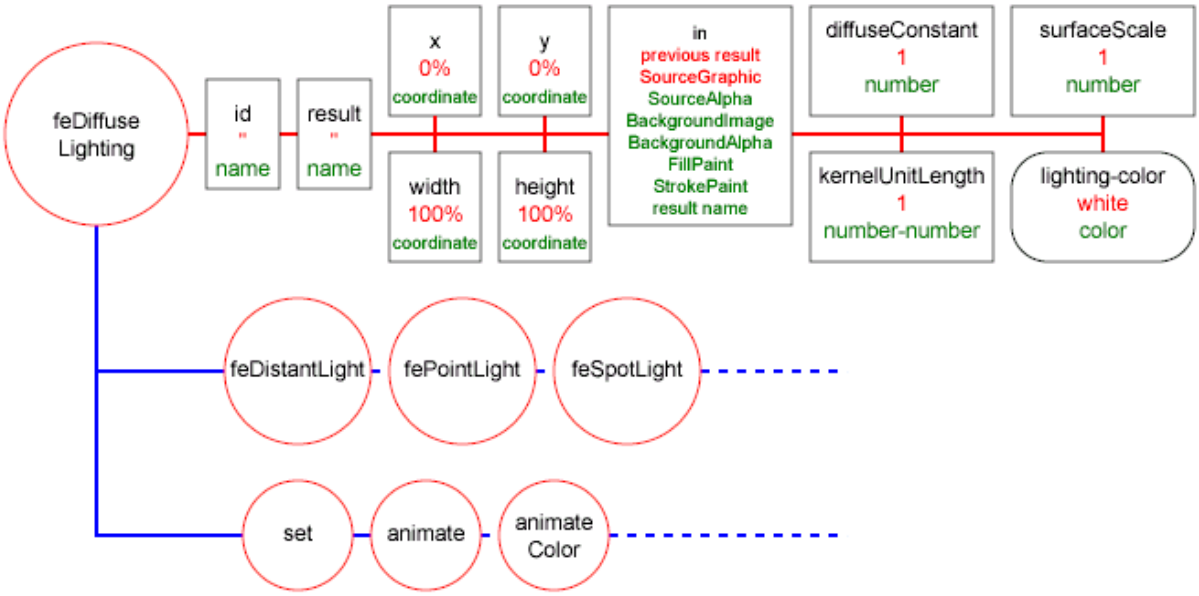

Diagram 8-8. Syntaxe de 'feDiffuseLighting'

Les attributs de '**feDiffuseLighting**' sont ' in', 'result' vus précédemment et:
surfaceScale : hauteur des reliefs créés (1 par défaut)
diffuseConstant : Un nombre positif. (1 par défaut)
kernelUnitLength : Un ou deux nombres positifs séparés par un espâce ou une virgule
lighting-color : Couleur de la lumière

Un exemple de code plus complet:

```
<filter id="MyFilter" filterUnits='objectBoundingBox' x='0%' y='0%'
        width='100%' height='100%'>
  <feColorMatrix in='SourceGraphic' result='pict0' type='luminanceToAlpha'/>
    <feDiffuseLighting in='pict0' result='pict1' lighting-color='white'
                       diffuseConstant='1' surfaceScale='9'>
      <feDistantLight azimuth='45' elevation='45'/>
  </feDiffuseLighting>
  <feComposite in2='pict1' in='SourceGraphic' operator='arithmetic'
               k1='0.6' k2='1.1' k3='0.6' k4='0'/>
</filter>

<image x="100" y="80" width='200' height='200' filter='url(#MyFilter)'
       xlink:href='fondu1.jpg'/>
```

La figure 8-6 montre l'influence des valeurs de l'attribut 'surfaceScale'. Nous utilisons ici l'éclairage d'une turbulence.

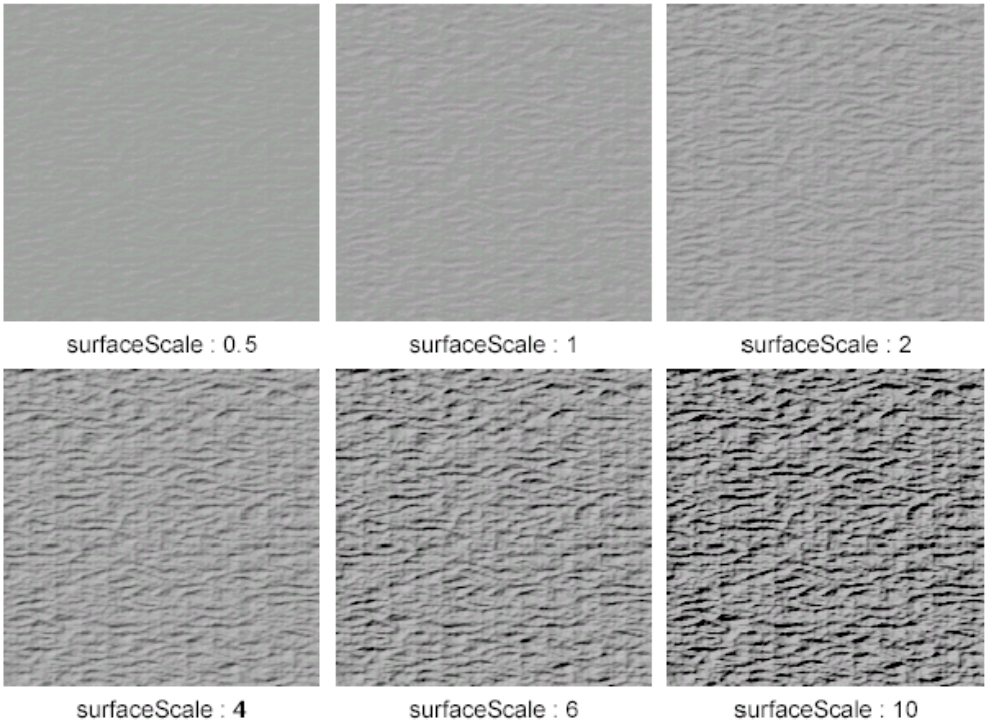

Figure 8-6. L'attribut 'surfaceScale'

La figure 8-7 montre quelques effets sur une image bitmap. Nous utilisons 'feColorMatrix' avec type="luminanceToAlpha" pour obtenir une nouvelle image, nous éclairons cette image et composons la lumière obtenue avec l'image originale.

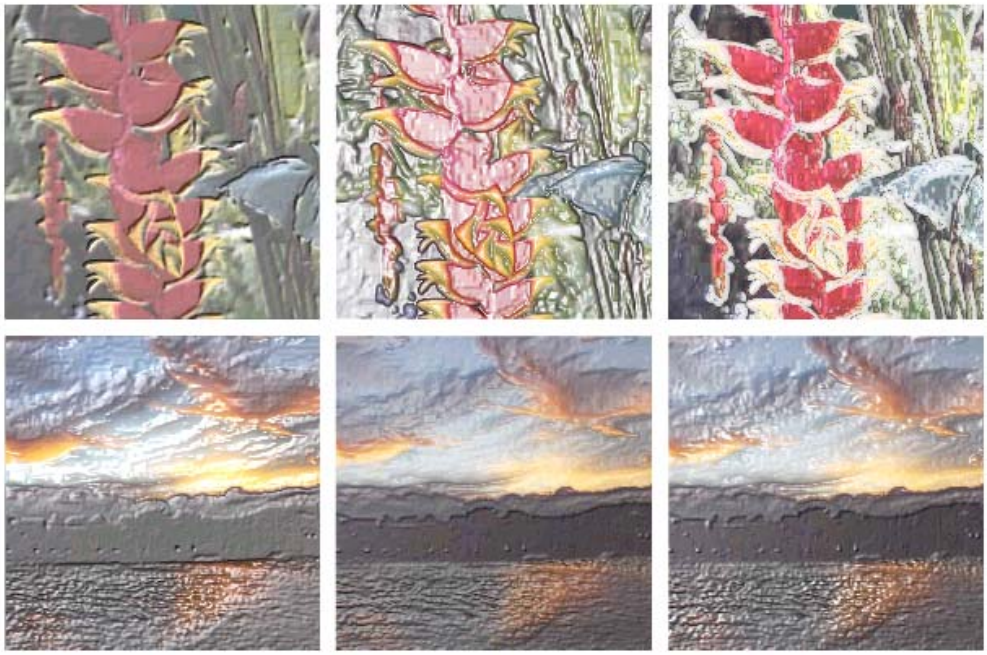

Figure 8-7. Effets avec 'feColorMatrix' et 'feDiffuseLighting'

Primitives pour composer des images

Trois primitives 'feBlend', 'feComposite' et 'feMerge' peuvent être utilisées pour composer des images (une image et son éclairage, un objet et son relief …).

La primitive 'feBlend'

La syntaxe de l'élément 'feBlend':

```
<feBlend  id="name"
          in="SourceGraphic | SourceAlpha | BackgroundImage |
              BackgroundAlpha | FillPaint | StrokePaint |
             <filter-primitive-reference>"
          in2="SourceGraphic | SourceAlpha | BackgroundImage |
              BackgroundAlpha | FillPaint | StrokePaint |
             <filter-primitive-reference>"
          result="<filter-primitive-reference>″
          x="NumberOrPercentage"
          y="NumberOrPercentage"
          width="NumberOrPercentage"
          height="NumberOrPercentage"
          mode="normal|multiply|screen|darken|lighten"/>
```

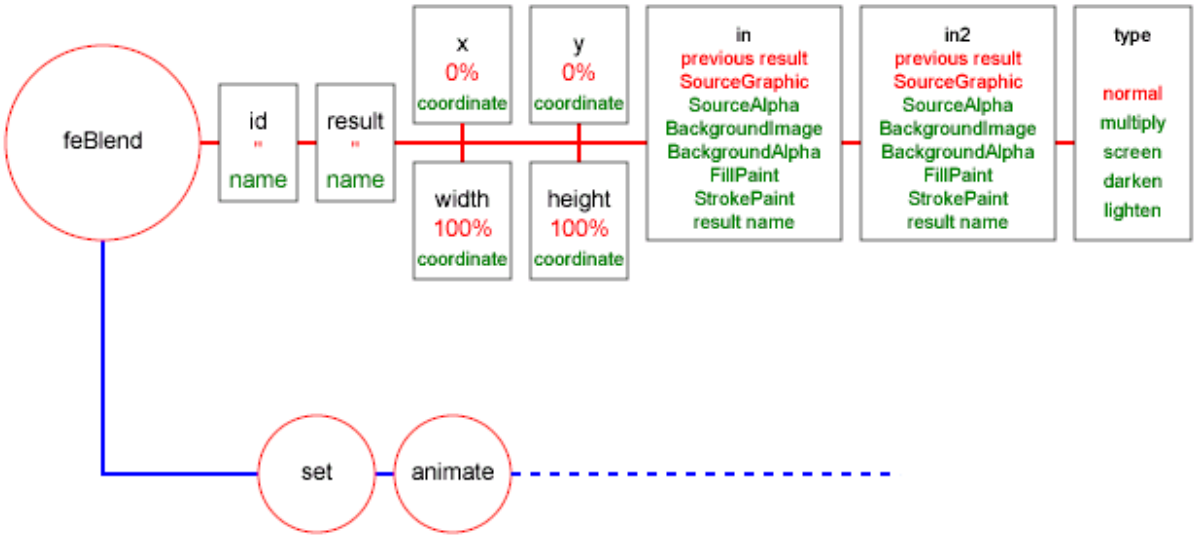

Diagram 8-9. Syntaxe de 'feBlend'

Pour '**feBlend**' en dehors des attributs communs, nous n'en avons qu'un seul spécifique:
mode : Les valeurs possibles sont "normal" "multiply" "screen" "darken" "lighten"

Exemple de code

```
<filter id='Filter2' filterUnits='userSpaceOnUse' x='0' y='0' width='400'
        height='400'>
  <feImage xlink:href='bateau.jpg' result='pict1'/>
  <feImage xlink:href='memo.svgz' result='pict2'/>
  <feBlend id='fbl' in='pict1' in2='pict2' mode='lighten'/>
</filter>
```

La figure 8-8. montre les différentes valeurs de l'attribut 'mode'. La première image ('in') est un bitmap et la seconde ('in2') est formée d'objets graphiques.

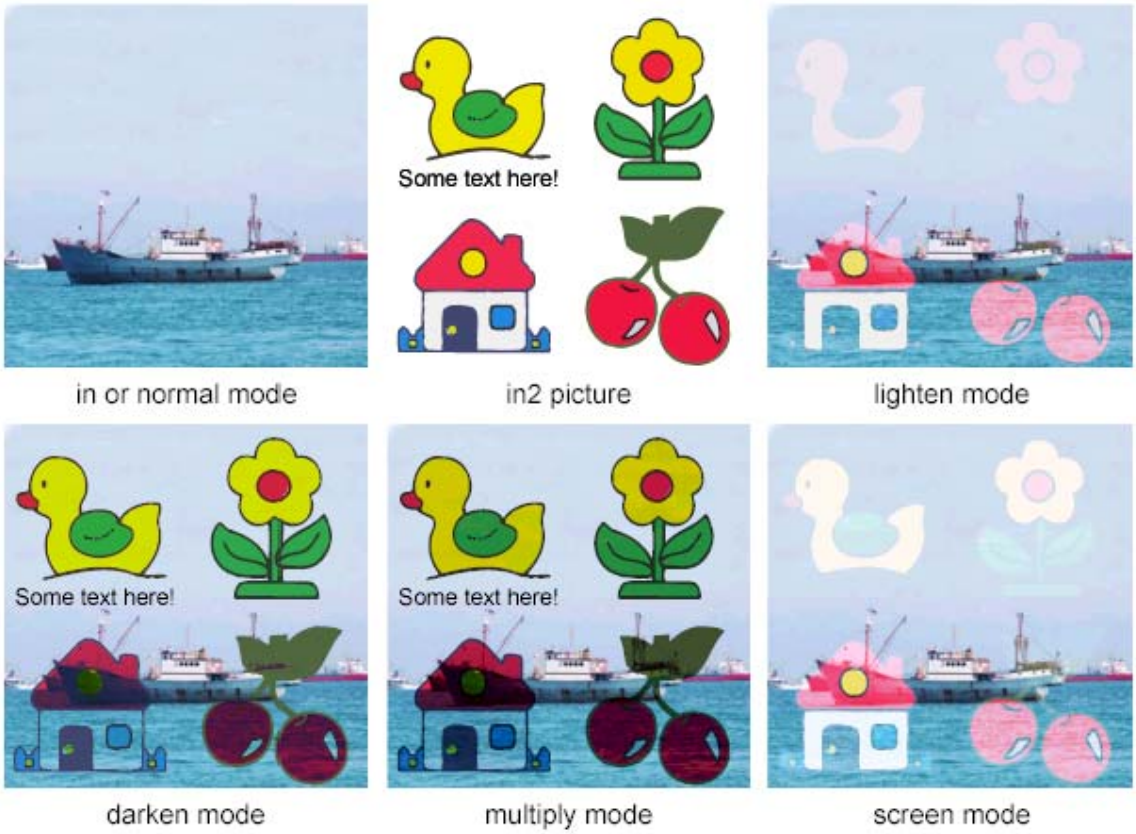

Figure 8-8. Valeurs de l'attribut 'mode'

La primitive 'feComposite'

La syntaxe de l'élément 'feComposite':

```
<feComposite  id="name"
              in="SourceGraphic | SourceAlpha | BackgroundImage |
                  BackgroundAlpha | FillPaint | StrokePaint |
                 <filter-primitive-reference>"
              in2="SourceGraphic | SourceAlpha | BackgroundImage |
                  BackgroundAlpha | FillPaint | StrokePaint |
                 <filter-primitive-reference>"
              result="<filter-primitive-reference>″
              x="NumberOrPercentage"
              y="NumberOrPercentage"
              width="NumberOrPercentage"
              height="NumberOrPercentage"
              operator="over|in|out|atop|xor|arithmetic"
              k1="number"
              k2="number"
              k3="number"
              k4="number"/>
```

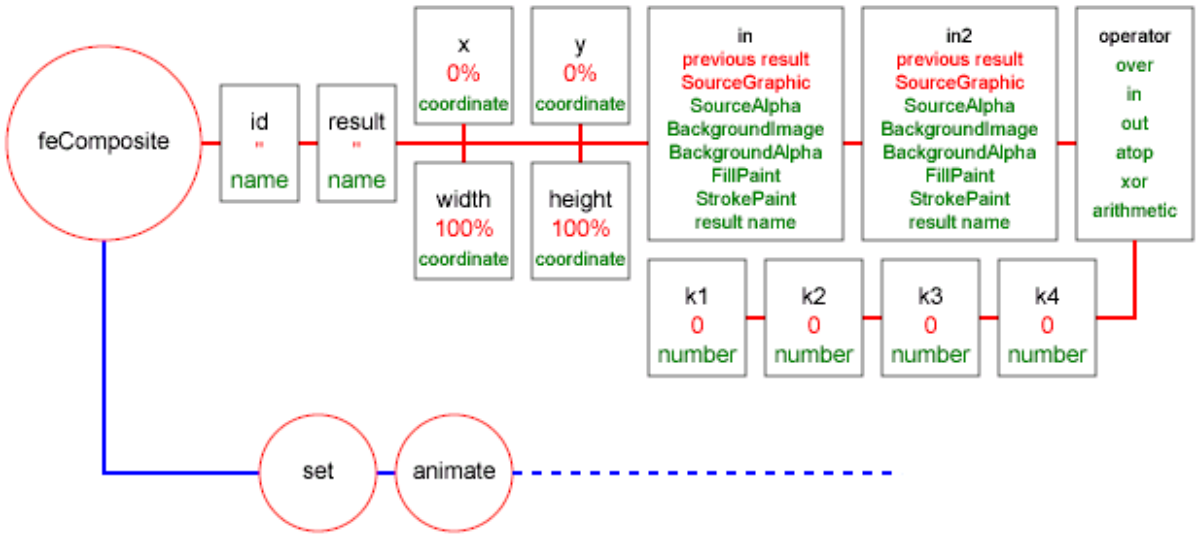

Diagram 8-10. Syntaxe de 'feComposite'

En dehors des attributs communs, '**feComposite**' n'a qu'un attribut spécifique
operator : Les valeurs admises sont "over" "in" "out" "atop" "xor" et "arithmetic".
Avec operator="arithmetic", nous devons donner des valeurs à '**k1**' '**k2**' '**k3**' et '**k4**' (0 par défaut pour chacun de ces attributs)

Code avec operator="atop"

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage xlink:href='memo.svgz' result='pict1'/>
  <feImage xlink:href='bandes.svg' result='pict2'/>
  <feComposite in='pict1' in2='pict2' operator='atop'/>
</filter>
```

Code avec operator="arithmetic"

```
<filter id='MyFilter' filterUnits='userSpaceOnUse' x='0' y='0' width='400'
        height='400'>
  <feImage xlink:href='memo.svgz' result='pict1'/>
  <feImage xlink:href='bandes.svg' result='pict2'/>
  <feComposite in='pict1' in2='pict2' operator='arithmetic' k1='0.4' k2='1'
               k3='0.5' k4='0'/>
</filter>
```

La figure 8-9 montre le résultat pour les différentes valeurs de 'operator', la première image ('in') est formée d'objets SVG, la seconde ('in2') est formée de bandes rouges.

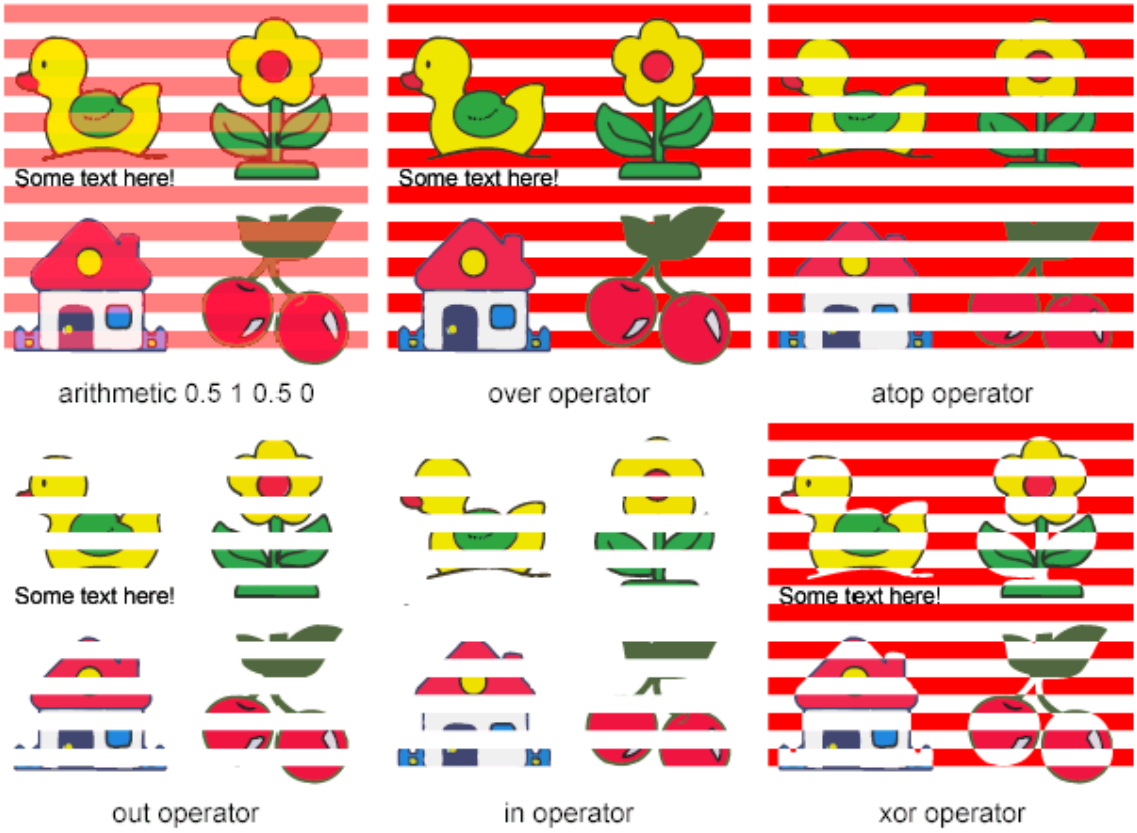

Figure 8-9. Les valeurs de l'attribut 'operator'

La primitive 'feMerge'

La syntaxe de l'élément 'feMerge':

```
<feMerge      id="name"
              result="<filter-primitive-reference>"
              x="NumberOrPercentage"
              y="NumberOrPercentage"
              width="NumberOrPercentage"
              height="NumberOrPercentage">
        <  éléments feMergeNode      />
</feMerge>
```

La syntaxe de l'élément 'feMergeNode'

```
<feMergeNode      id="name"
                  in="SourceGraphic | SourceAlpha | BackgroundImage |
                  BackgroundAlpha | FillPaint | StrokePaint |
                   <filter-primitive-reference>"/>
```

Pour '**feMerge**', les images à composer sont définies par les éléments '**feMergeNode**', descendants de 'feMerge'. Comme le nombre d'images que peut traiter 'feMerge' est quelconque, nous ne pouvons utiliser 'in' et 'in2'.

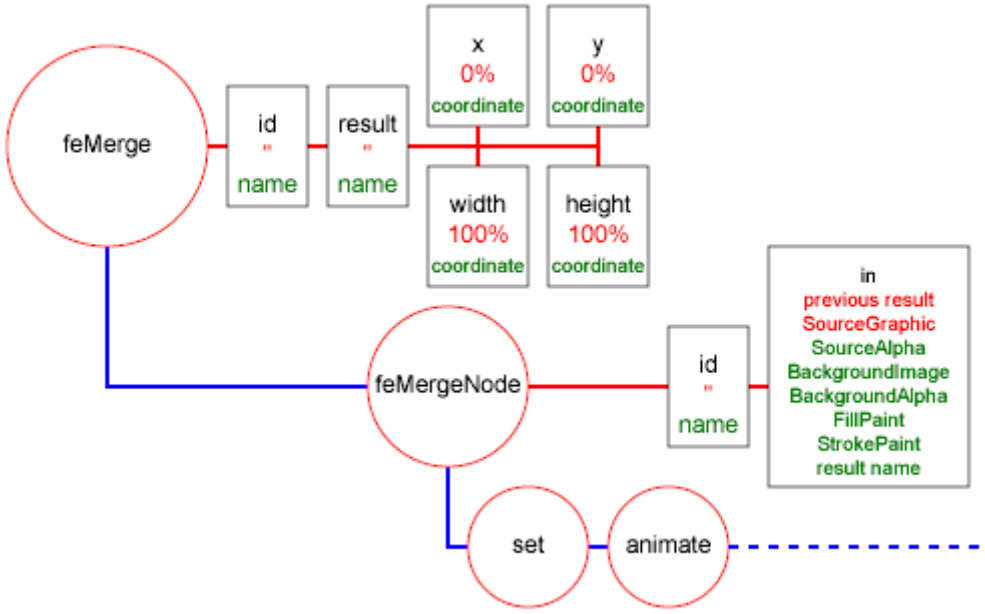

Diagram 8-11. Syntaxe de l'élément 'feMerge'

Avec la primitive 'feMerge' les images sont simplement affichées dans l'ordre des descendants 'feMergeNode'.
Nous ne pouvons pour des images opaques que jouer sur leur opacité.

Exemple de code :

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage xlink:href='fondu3.jpg' result='pict1'/>
  <feImage xlink:href='bateau.jpg' result='pict2'/>
  <feMerge>
    <feMergeNode in="pict1"/>
    <feMergeNode in="pict2"/>
  </feMerge>
</filter>
```

La figure 8-10 montre un exemple où nous jouons sur l'opacité de ces deux images bitmap.

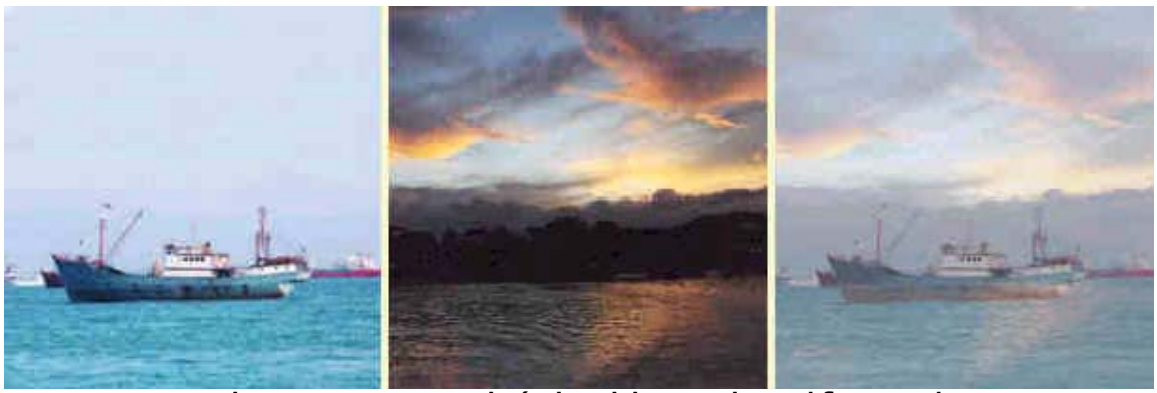

Figure 8-10. Opacité des bitmap dans 'feMerge'

Primitives qui créent des objets graphiques

La primitive 'feFlood'

Avec '**feFlood**', nous créons un rectangle coloré d'après les propriétés 'flood-color' et 'flood-opacity'.

La propriété 'flood-color' indique la couleur de remplissage du rectangle.

'flood-color'

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

La propriété 'flood-opacity' définit l'opacité pour le rectangle, la valeur est comprise entre 0 et 1.

'flood-opacity'

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

Syntaxe de l'élément 'feFlood':

```
<feFlood    id="name"
            result="<filter-primitive-reference>"
            x="NumberOrPercentage"
            y="NumberOrPercentage"
            width="NumberOrPercentage"
            height="NumberOrPercentage"
            flood-color="property"
            flood-opacity="property"/>
```

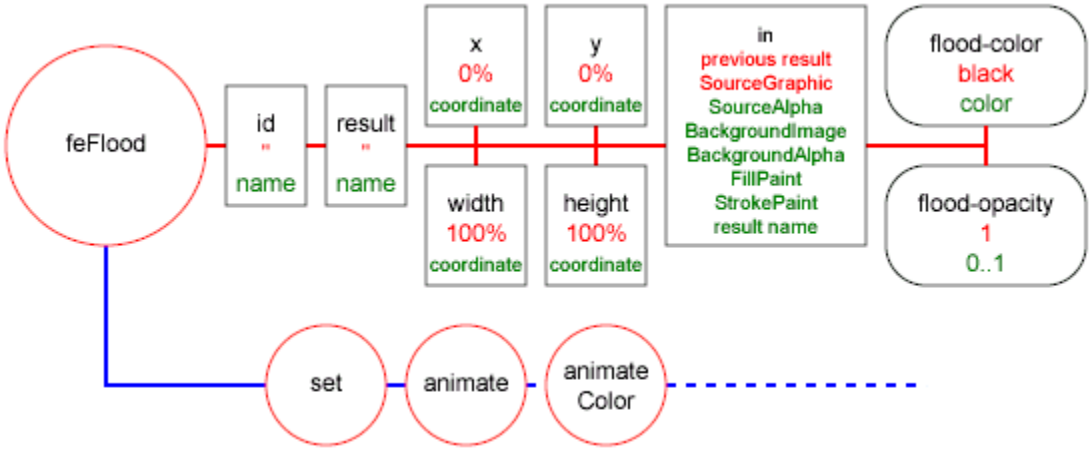

Diagram 8-12. Syntaxe de 'feFlood'

En plus des attributs 'id', 'x', 'y', 'width', 'height', 'result' nous avons deux propriétés:
flood-color : choix de la couleur de remplissage (noir par défaut)
flood-opacity : choix de l'opacité entre 0 et 1 (1 par défaut)

Exemple de code:

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feFlood flood-color="red" flood-opacity='0.3' result='pict1'/>
</filter>
```

La primitive 'feImage'

La primitive '**feImage**' crée un bitmap à partir d'un fichier externe ou d'éléments définis dans le document.

Ce bitmap est alors disponible pour une utilisation dans le filtre par d'autres primitives.

Syntaxe de l'élément 'feImage':

<**feImage** id="name"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
xlink:href="URI"/>

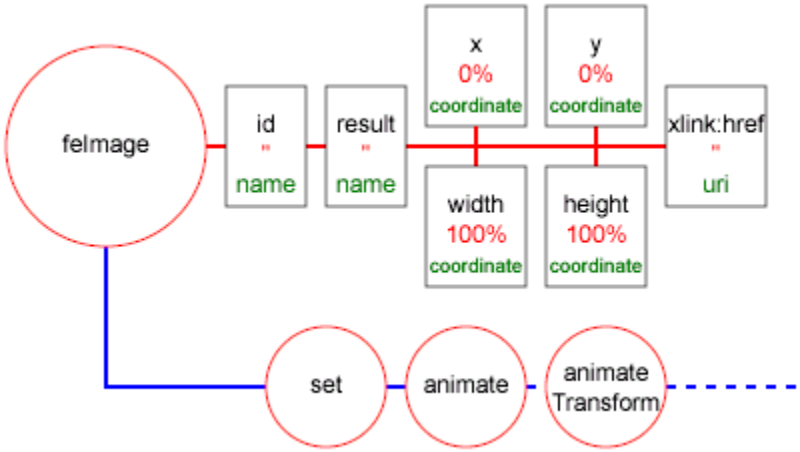

Diagram 8-13. Syntaxe de 'feImage'

En dehors des attributs communs aux primitives, nous avons un attribut spécifique:
xlink:href : adresse du graphique externe ou d'un élément du document.

Exemples de code:

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage xlink:href='fondu3.jpg' result='pict1'/>
</filter>
```

or

```
<filter id='Filter0' filterUnits='userSpaceOnUse' x='0' y='0' width='400'
        height='400'>
  <feImage xlink:href='#MyImage1' result='pict1'/>
</filter>
<g id='MyImage1'>
  <circle cx="200" cy="200" r="200" style="stroke:none;fill:red"/>
  <circle cx="200" cy="200" r="180" style="stroke:none;fill:white"/>
  <!--  autres éléments      -->
</g>
```

La primitive 'feTile'

Avec '**feTile**', nous créons un pavage de la zone affectée au filtre. C'est un pavage avec une forme rectangulaire qui est répétée par des translations.

Syntaxe de l'élément 'feTile':

<**feTile** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"/>
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"

width="NumberOrPercentage"
height="NumberOrPercentage"/>

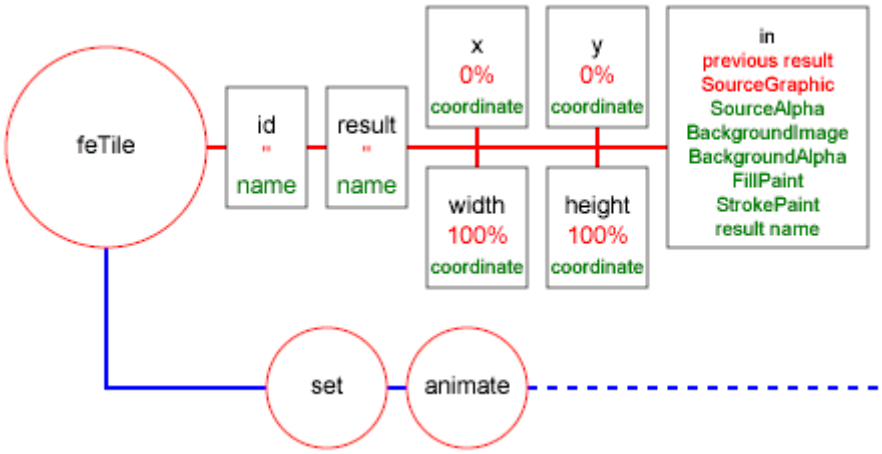

Diagram 8-14. Syntaxe de 'feTile'

Les attributs de 'feTile' sont les attributs communs '**x**' '**y**' '**width**' et '**height**' pour définir la région couverte par le pavage, '**in**' pour l'image qui sera répétée et '**result**' pour nommer le résultat obtenu.

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage x="0" y="0" width="100" height="100" xlink:href='memo2.svg'
          result='pict1'/>
  <feTile x="0" y="0" width="400" height="400" in="pict1"/>
</filter>
```

Nous pouvons obtenir le même résultat avec 'pattern', mais comme pour 'feImage', le réultat est utilisable par les primitives du filtre.

La primitive 'feTurbulence'

Avec '**feTurbulence**' nous pouvons créer des textures complexes en combinant cette primitive avec un travail sur les couleurs, un éclairage et en composant les images obtenues.

Syntaxe de l'élément 'feTurbulence':

<**feTurbulence** id="name"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
type="fractalNoise|turbulence"
baseFrequency="NumberOptionalNumber"
numOctaves="Integer"
seed="Number"
stitchTiles="stitch|noStitch"/>

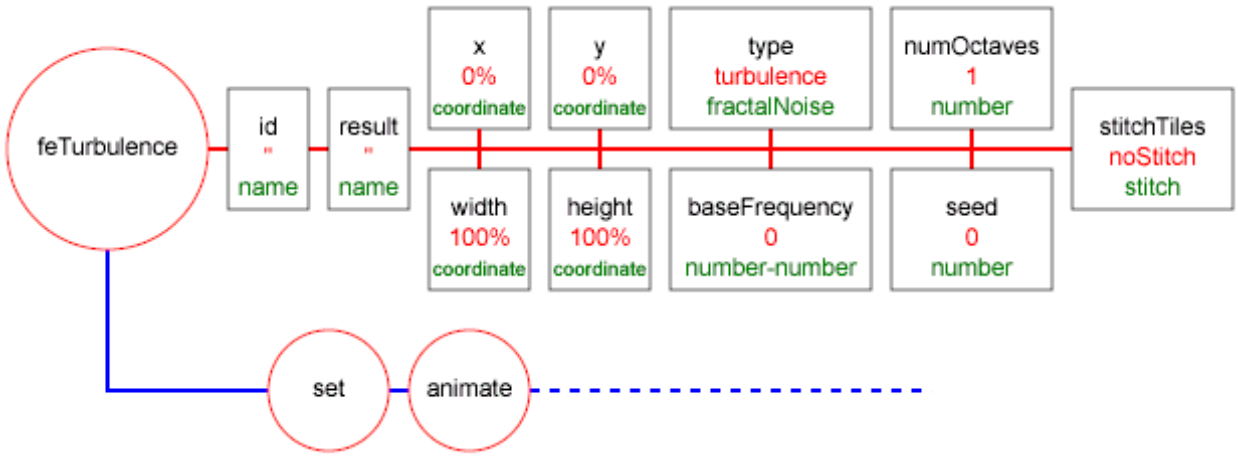

Diagram 8-15. Syntaxe de 'feTurbulence'

En dehors des attributs communs, nous avons:
type: deux valeurs possibles 'fractalNoise' ou 'turbulence' (valeur par défaut)
pour chacun de ces types:

baseFrequency : deux nombres positifs pour x et y. (0 par défaut)
numOctaves : un nombre entier positif (1 par défaut)
seed : un entier positif (change le départ pour les couleurs aléatoires) (0 par défaut)
stitchTiles : "stitch | noStitch" pour avoir une transition ou non sur les bords de la zone concernée.

Exemple de code:

```
<filter id="Filter5" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feTurbulence type="turbulence" baseFrequency="0.21,0.21" numOctaves="4"
                seed='10'/>
</filter>
```

La figure 8-11 montre l'influence des valeurs de 'baseFrequency' sur les graphiques obtenus.

Pour jouer sur les couleurs nous pouvons utiliser 'feColorMatrix' (Figure 8-13) ou 'feComponentTransfer' (Figure 8-12) comme dans cet exemple:

```
<filter id="Filter1" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feTurbulence type='turbulence' baseFrequency='0.01,0.09' numOctaves='4'
                seed='2'/>
  <feColorMatrix type='matrix'
                Valeurs='0.1  0  0  0  0
                         0  0.3  0  0  0
                         0   0  0.8 0  0
                         0   0   0  1  0 '/>
</filter>
```

Nous pouvons également ajouter un éclairage et le composer avec la turbulence (Figure 8-14) pour obtenir des textures complexes – nuages, granulations ...

```
<filter id="Filter5" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feTurbulence result='pict0' type='fractalNoise' baseFrequency='0.2,0.2'
                numOctaves='4' seed='0' stitchTiles='noStitch'/>
  <feDiffuseLighting result='pict1' in='pict0' lighting-color='#d7eb2f'
```

```
                    diffuseConstant='1' kernelUnitLength='1,1'
                    surfaceScale='12.7'>
    <feDistantLight azimuth='45' elevation='35'/>
  </feDiffuseLighting>
  <feComposite in2='pict1' in='pict0' operator='arithmetic' k1='0.4' k2='0.4'
               k3='1' k4='0'/>
</filter>
```

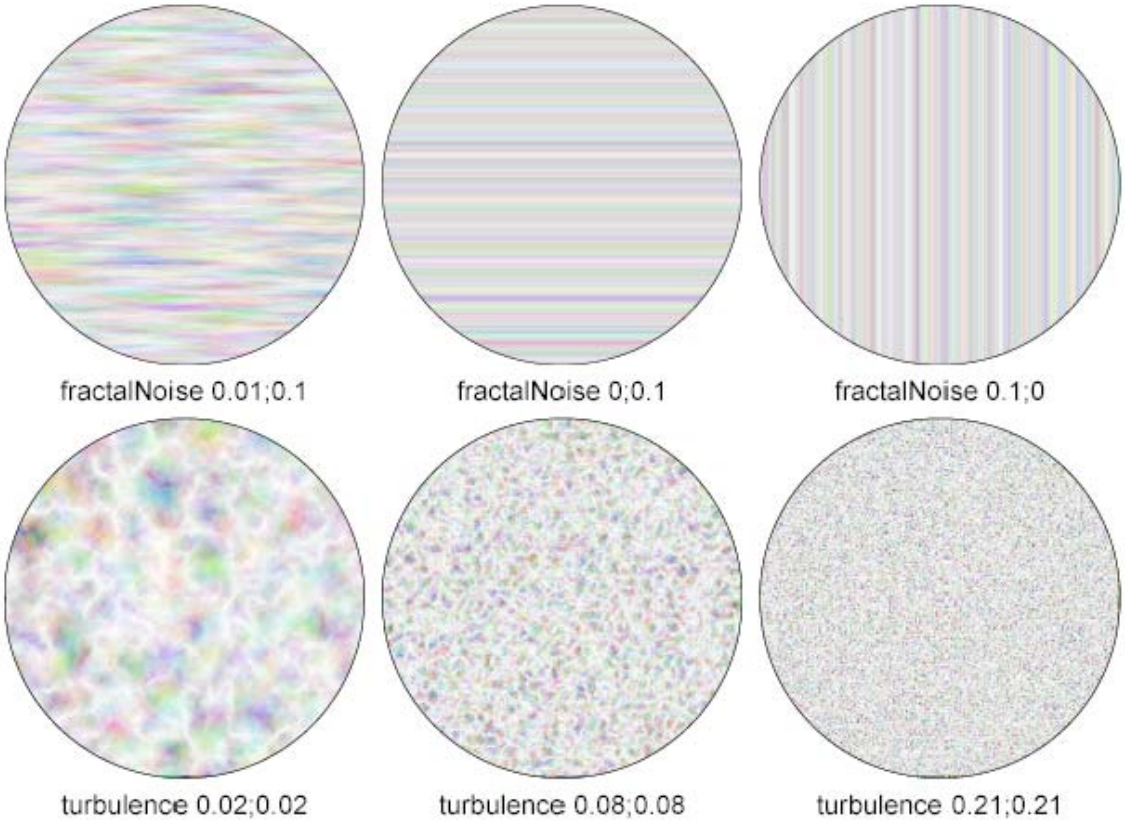

Figure 8-11. Exemples avec 'feTurbulence'

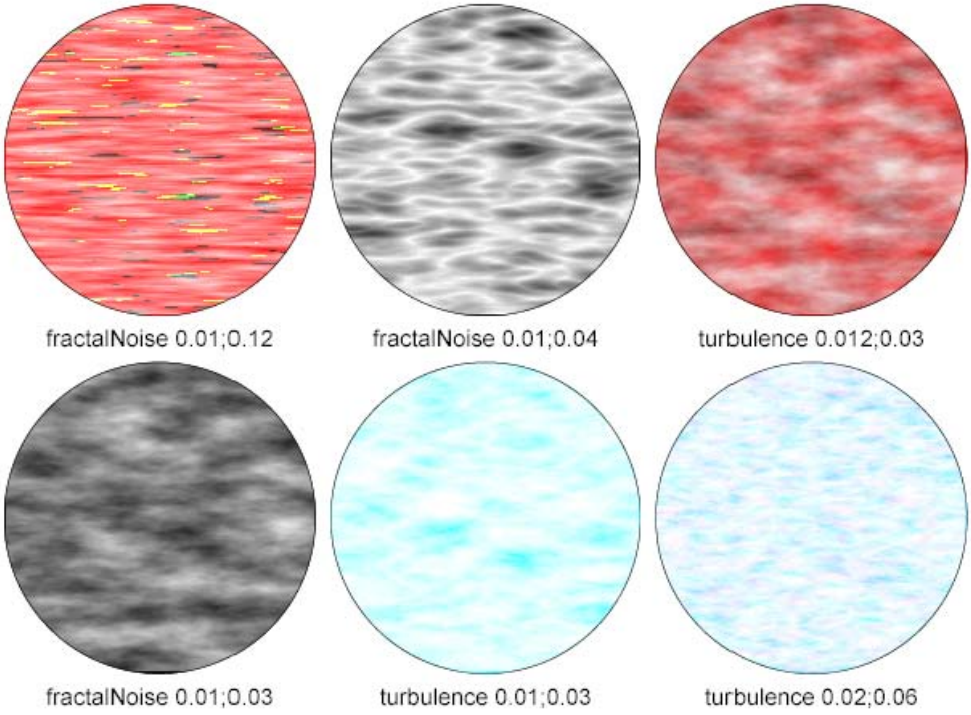

Figure 8-12. 'feTurbulence' et 'feComponentTransfer'

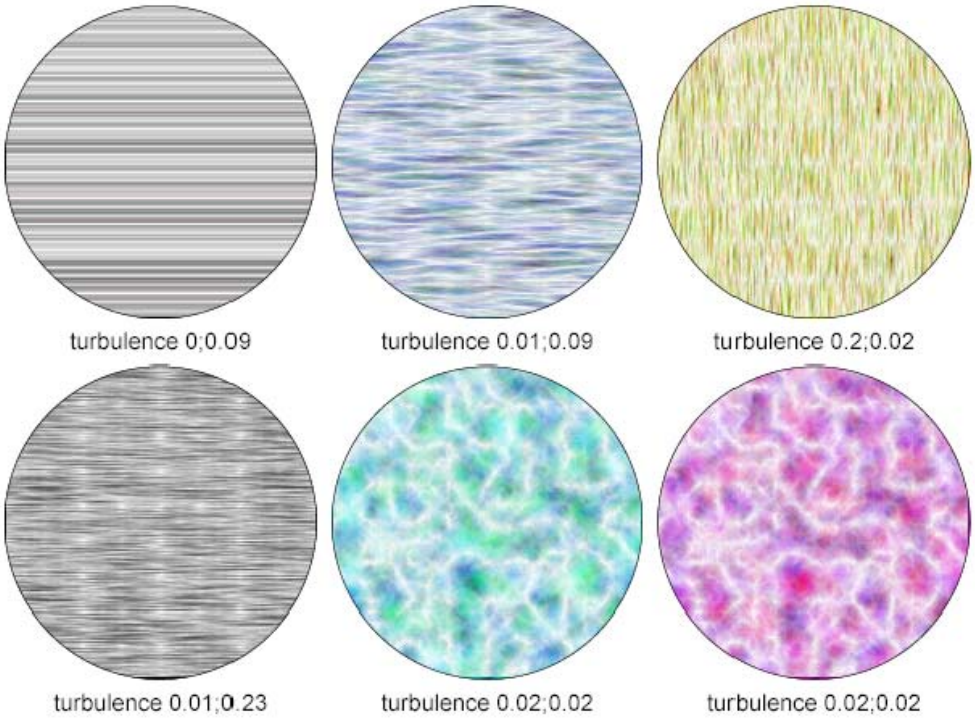

Figure 8-13. 'feTurbulence' et 'feColorMatrix'

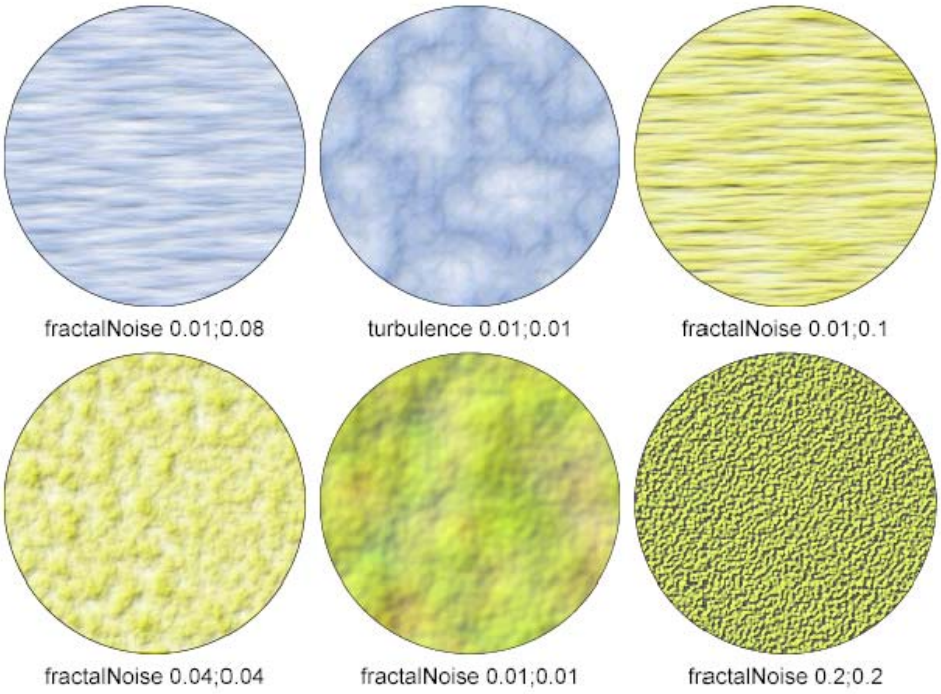

Figure 8-14. 'feTurbulence' et éclairages

Primitives qui modifient les objets graphiques

La primitive 'feGaussianBlur'

Avec 'feGaussianBlur' nous obtenons un flou sur l'image source.

Syntaxe de l'élément 'feGaussianBlur':

<**feGaussianBlur** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
stdDeviation="NumberOptionalNumber"/>

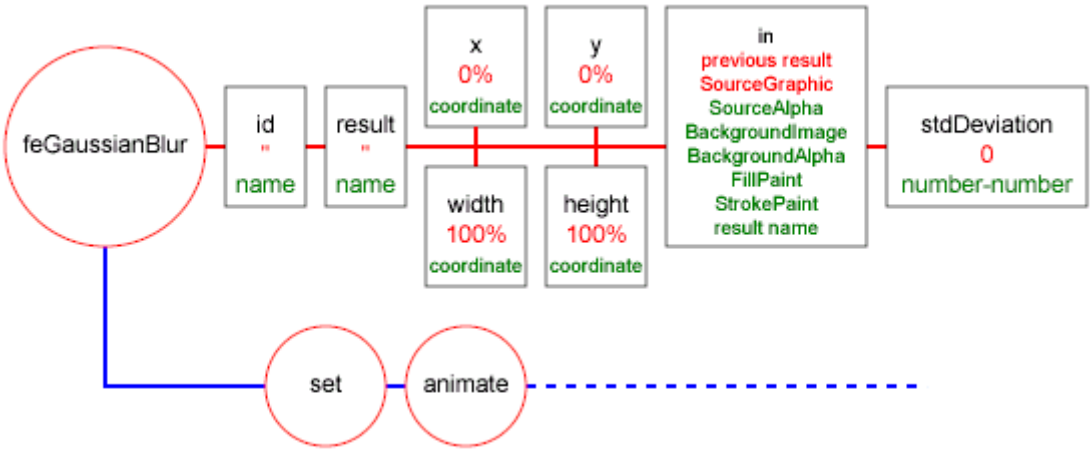

Diagram 8-16. Syntaxe de 'feGaussianBlur'

En dehors des attributs communs, nous avons pour 'feGaussianBlur' un attribut spécifique:

stdDeviation : deux entiers positifs en x et y (0 par défaut). Ces nombres définissent la dimension du flou créé suivant les deux directions du plan.'

Exemple de code:

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage xlink:href='pie.svgz' result='image1'/>
  <feGaussianBlur in='image1' stdDeviation='2,2' />
</filter>
```

La figure 8-15 montre l'effet de 'feGaussianBlur' sur des objets SVG en choisissant comme source 'SourceGraphic' ou 'SourceAlpha'.

Figure 8-15. 'feGaussianBlur' ('SourceGraphic' et 'SourceAlpha')

La primitive 'feMorphology'

La primitive 'feMorphology' produit un épaississement ou un rétrécissement des zones des objets.

Syntaxe de l'élément 'feMorphology':

<**feMorphology** id="name"
in="SourceGraphic | SourceAlpha | BackgroundImage |
BackgroundAlpha | FillPaint | StrokePaint |
<filter-primitive-reference>"
result="<filter-primitive-reference>"
x="NumberOrPercentage"
y="NumberOrPercentage"
width="NumberOrPercentage"
height="NumberOrPercentage"
operator="erode|dilate"
radius="NumberOptionalNumber"/>

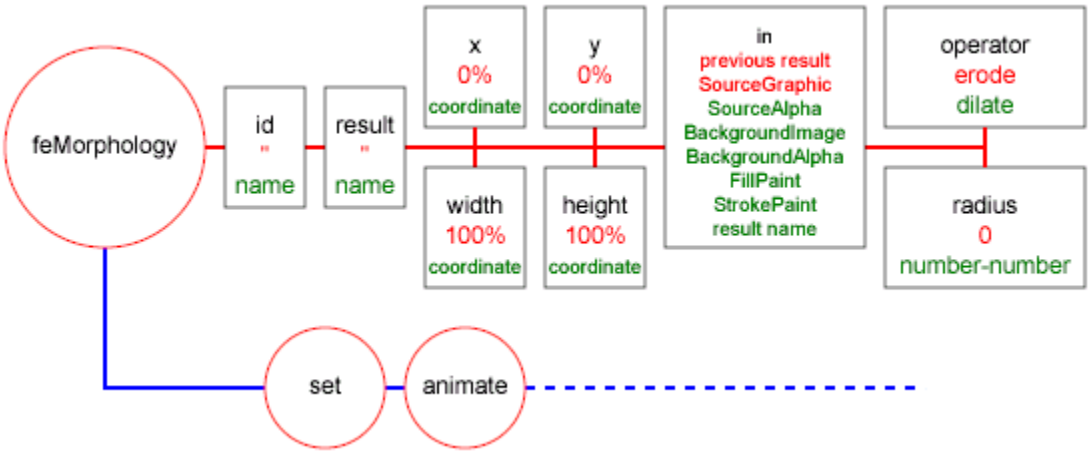

Diagram 8-17. Syntaxe de 'feMorphology'

En dehors des attributs communs nous avons pour 'feMorphology'

operator : deux valeurs possibles "erode" (valeur par défaut) ou "dilate"

radius : deux entiers positifs pour x et y (0 par défaut)

Exemple de code:

```
<filter id='Filter1' filterUnits='userSpaceOnUse' x='0' y='0' width='400'
        height='400'>
  <feImage xlink:href='memo.svgz' result='image1'/>
  <feMorphology in='image1' radius='4,4' operator='erode'/>
</filter>
```

La figure 8-16 montre l'effet de 'feMorphology' appliquée à des objets SVG ou un bitmap (avec 'erode' ou 'dilate' pour l'attribut 'operator').

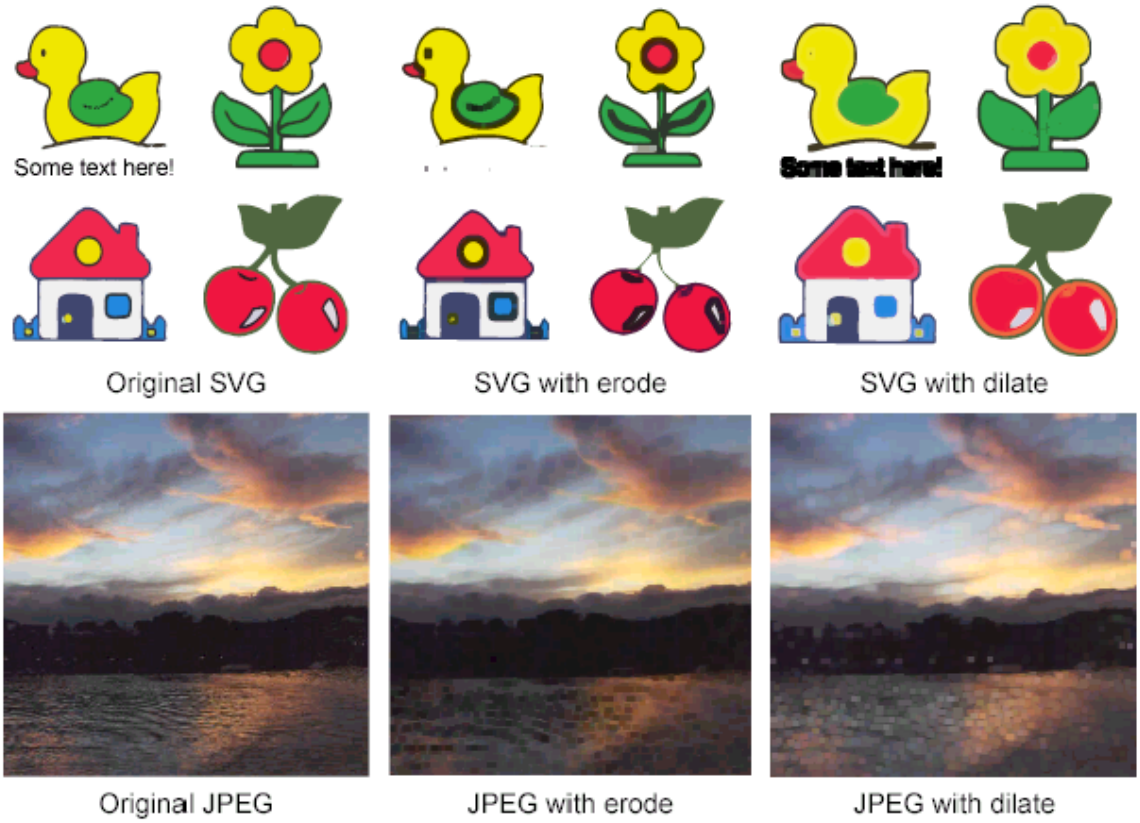

Figure 8-16. 'feMorphology' sur des objets SVG et un bitmap.

La primitive 'feDisplacementMap'

Cette primitive 'feDisplacementMap' utilise deux images, l'une définie par 'in2' détermine les pixels de l'autre définie par 'in' qui seront déplacés. De plus le déplacement sera commandé par le canal des pixels de 'in2' choisi, R G B ou A pour le déplacement en x et en y.

Syntaxe de l'élément 'feDisplacementMap':

```
<feDisplacementMap          id="name"
                    in="SourceGraphic | SourceAlpha | BackgroundImage |
                        BackgroundAlpha | FillPaint | StrokePaint |
                       <filter-primitive-reference>"
                    in2="SourceGraphic | SourceAlpha | BackgroundImage |
                        BackgroundAlpha | FillPaint | StrokePaint |
                       <filter-primitive-reference>"
                    result="<filter-primitive-reference>"
                    x="NumberOrPercentage"
                    y="NumberOrPercentage"
                    width="NumberOrPercentage"
                    height="NumberOrPercentage"
                    scale="Number"
                    xChannelSelector="R|G|B|A"
                    yChannelSelector="R|G|B|A"/>
```

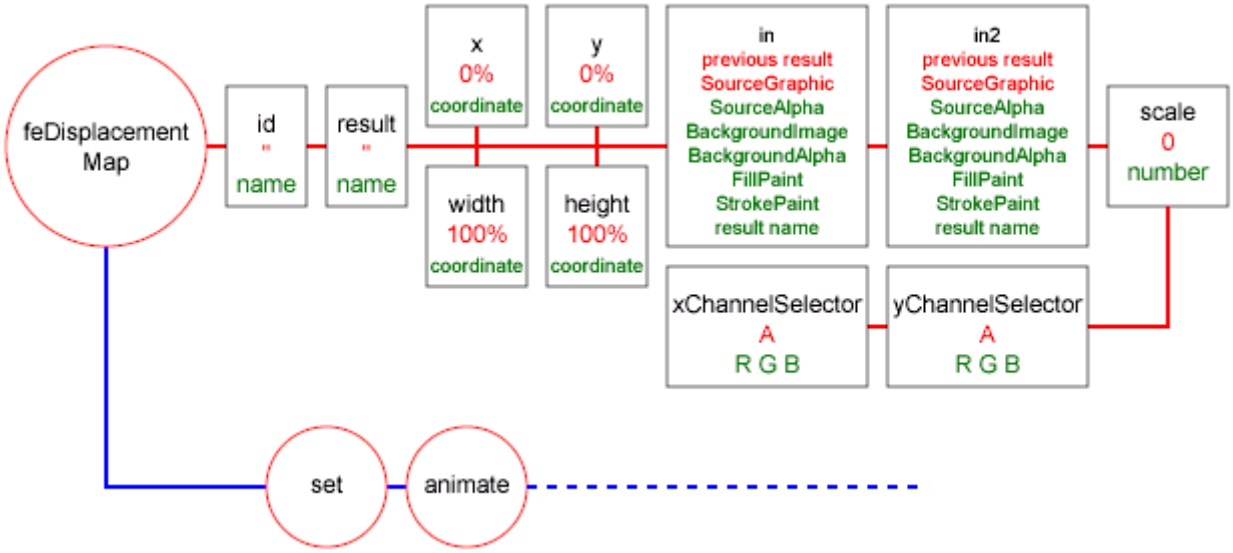

Diagram 8-18. Syntaxe de 'feDisplacementMap'

En dehors des attributs communs, nous avons pour 'feDisplacementMap':
scale : nombre (0 par défaut) qui donne l'amplitude du déplacement des pixels de 'in'
xChannelSelector : R G B ou A (définit le canal actif en x pour les pixels de 'in2')
yChannelSelector : R G B ou A (définit le canal actif en y pour les pixels de 'in2')

Dans ce code, nous créons avec 'pattern' l'image qui commandera les déplacements, en quelque sorte une grille, des pixels d'une image JPEG.
Ce code correspond à l'image "dots" de la figure 8-17.

```
<pattern id='motif1' x='0' y='0' width='40' height='40'
         patternUnits="userSpaceOnUse">
  <circle cx='10' cy='10' r='10' style='stroke:black;
         stroke-width:1;fill:red'/>
  <circle cx='30' cy='30' r='10' style='stroke:black;
         stroke-width:1;fill:red'/>
</pattern>
<rect id='MyGrid' x="0" y="0" width="400" height="400" fill="url(#motif1)"/>
<filter id='Filter1' filterUnits='userSpaceOnUse' x='0' y='0' width='400'
        height='400'>
  <feImage xlink:href='puzzle.jpg' result='pict1'/>
  <feImage xlink:href='#MyGrid' result='pict2'/>
  <feDisplacementMap scale='19' xChannelSelector='R' yChannelSelector='R'
                     in2='pict2' in='pict1'/>
</filter>
```

Nous pouvons également utiliser des gradients pour créer des grilles de déplacement comme nous le voyons dans la figure 8-17

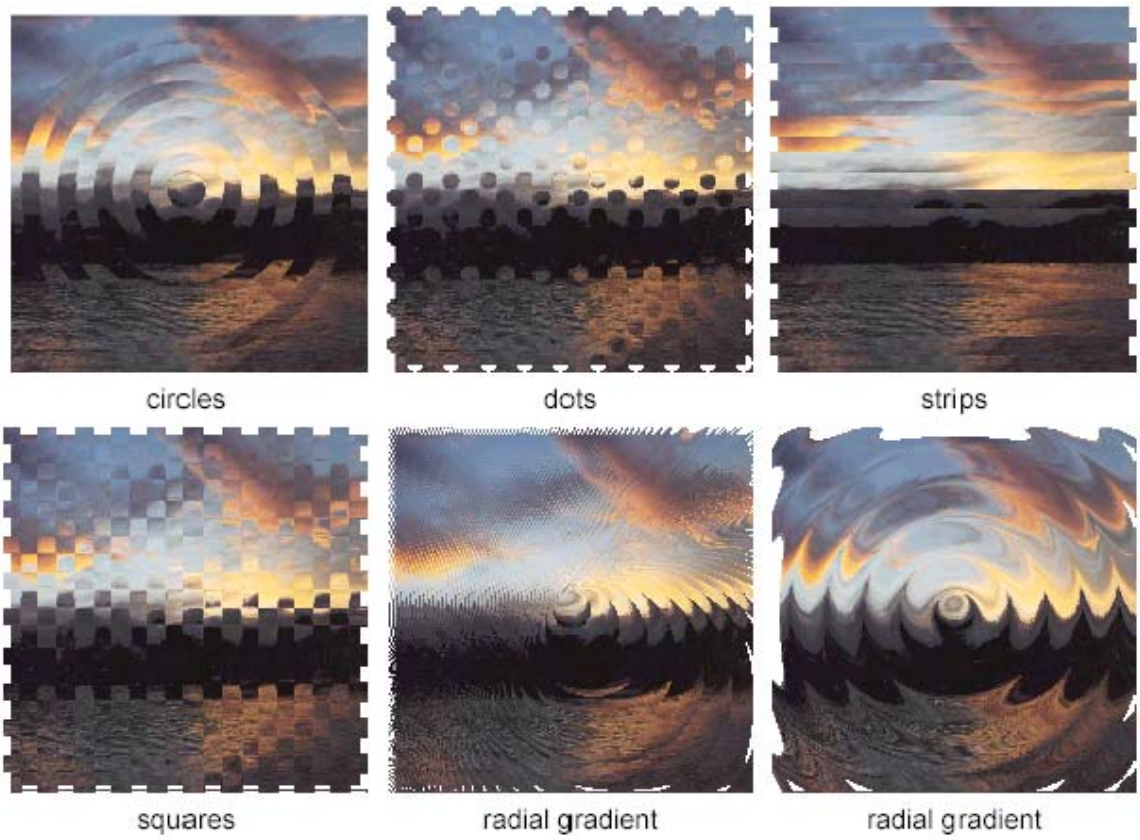

Figure 8-17. 'feDisplacementMap' appliqée à un bitmap

Avec 'feDisplacementMap', nous pouvons animer l'attribut 'scale' pour des effets de fondu-enchaîné sur des images.

La primitive 'feOffset'

La primitive 'feOffset' décale simplement l'image source du vecteur défini par 'dx' et 'dy'.

La syntaxe de l'élément 'feOffset':

```
<feOffset      id="name"
               in="SourceGraphic | SourceAlpha | BackgroundImage |
                   BackgroundAlpha | FillPaint | StrokePaint |
                  <filter-primitive-reference>"
               result="<filter-primitive-reference>″
               x="NumberOrPercentage"
               y="NumberOrPercentage"
               width="NumberOrPercentage"
               height="NumberOrPercentage"
               dx="Number"
               dy="Number"/>
```

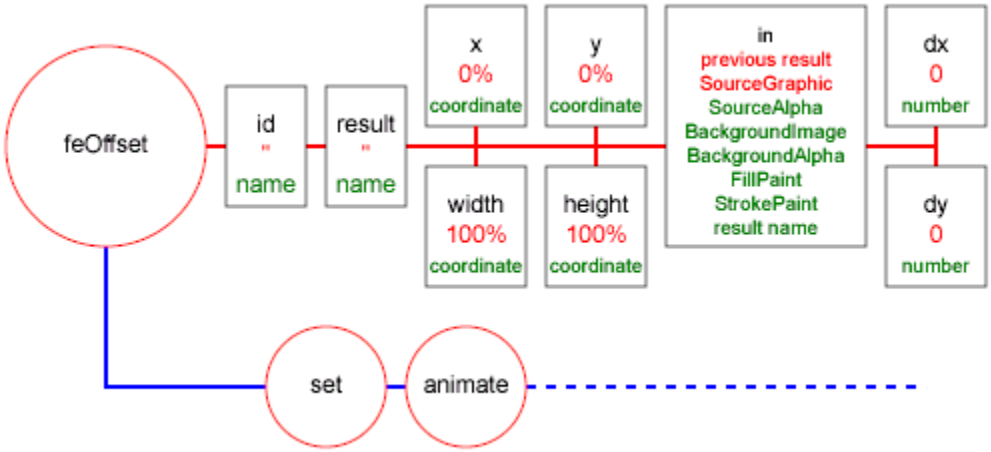

Diagram 8-19. Syntaxe de 'feOffset'

En dehors des attributs communs, nous avons pour 'feOffset':
dx : déplacement en x (0 par défaut)
dy : déplacement en y (0 par défaut)

La primitive 'feConvolveMatrix'

La primitive 'feConvolveMatrix' combine les pixels de l'image source avec les pixels voisins pour créer une nouvelle image. Cette primitive nécessite un temps de calcul très important.

Syntaxe de l'élément 'feConvolveMatrix':

```
<feConvolveMatrix  id="name"
                   in="SourceGraphic | SourceAlpha | BackgroundImage |
                       BackgroundAlpha | FillPaint | StrokePaint |
                      <filter-primitive-reference>"
                   result="<filter-primitive-reference>"
                   x="NumberOrPercentage"
                   y="NumberOrPercentage"
                   width="NumberOrPercentage"
                   height="NumberOrPercentage"
                   order="NumberOptionalNumber"
                   kernelMatrix="list of numbers"
                   bias="Integer"
                   targetX="Integer"
                   targetY="Integer"
                   preserveAlpha="true|false"
                   divisor="Integer"
                   edgeMode="duplicate|wrap|none"/>
```

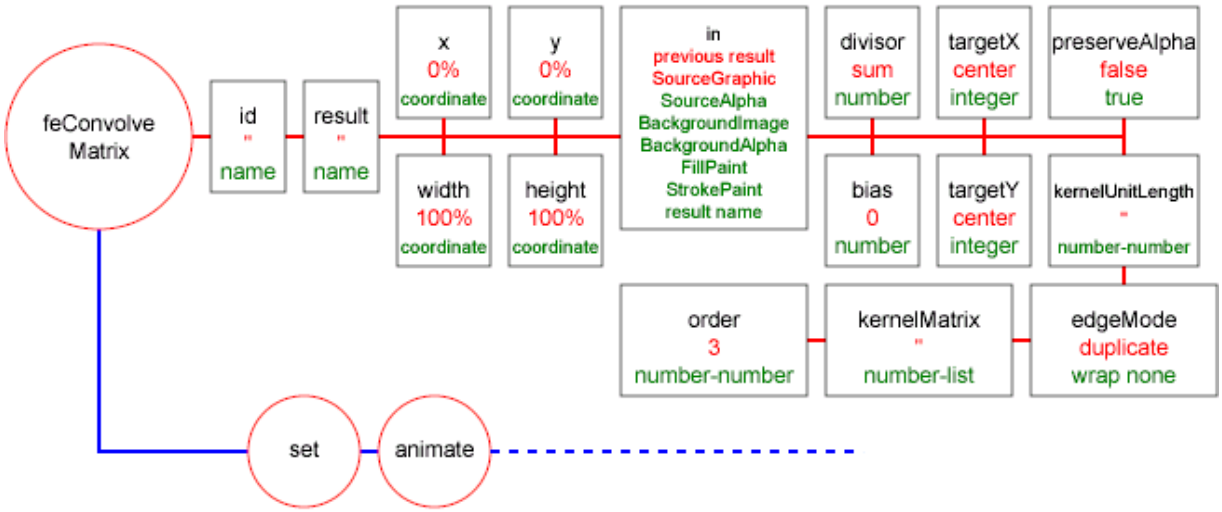

Diagram 8-20. Syntaxe de 'feConvolveMatrix'

En dehors des attributs communs, nous avons pour 'feConvolveMatrix':

order : deux entiers positifs pour x et y (3 par défaut)
kernelMatrix : order_x*order_y valeurs pour le calcul des pixels
targetX, targetY : deux entiers positifs pour centrer le calcul (order_x/2 et order_y/2 par défaut)
preserveAlpha : deux valeurs true ou false (false par défaut)
bias : valeur à ajouter dans le calcul (0 par défaut)
divisor : entier positif, par défaut ce sera la somme des termes de 'kernelMatrix'
edgeMode : trois valeurs duplicate | wrap | none (none par défaut) pour gérer les bords

La figure 8-18 montre le résultat sur des objets SVG

Source de l'exemple:

```
<svg width="450" height="250" viewBox="-50 -50 900 500">
  <defs>
    <filter id="MyFilter" filterUnits="userSpaceOnUse" x="400" y="0"
            width="400" height="400">
      <feImage xlink:href='memo.svgz' result='pict1'/>
      <feConvolveMatrix in='pict1' order="5 5" targetX="2" targetY="2"
                        kernelMatrix="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                                      0 0 0 0 0 1" preserveAlpha='true'/>
    </filter>
  </defs>
  <image x="0" y="0" width="400" height="400" xlink:href="memo.svgz"/>
  <rect filter="url(#MyFilter)" x="400" y="0" width="400" height="400"/>
</svg>
```

Figure 8-18. Exemple avec 'feConvolveMatrix'

Combinaisons de primitives

Nous avons vu quelques combinaisons de primitives comme feTurbulence, feColorMatrix et feDiffuseLighting pour créer des textures.

Voyons une combinaison classique pour créer un effet 3D

Nous pouvons utiliser:
'feGaussianBlur' appliquée à sourceAlpha pour créer les ombres
'feOffset' pour décaler cette ombre
'feSpecularLighting' pour créer un éclairage
'feComposite' pour composer ombre, lumière et image

Vous trouverez toujours à la même adresse (pilat.free.fr) des outils pour tester les paramètres des primitives et quelques combinaisons de primitives comme celle-ci.
La figure 8-19 est une copie d'écran de cet outil. Vous pouvez faire varier tous les paramètres et voir immédiatement le résultat, vous pouvez également choisir l'image de départ sur votre disque et récupérer le code de votre filtre (Utilisation de PHP pour retourner le code)

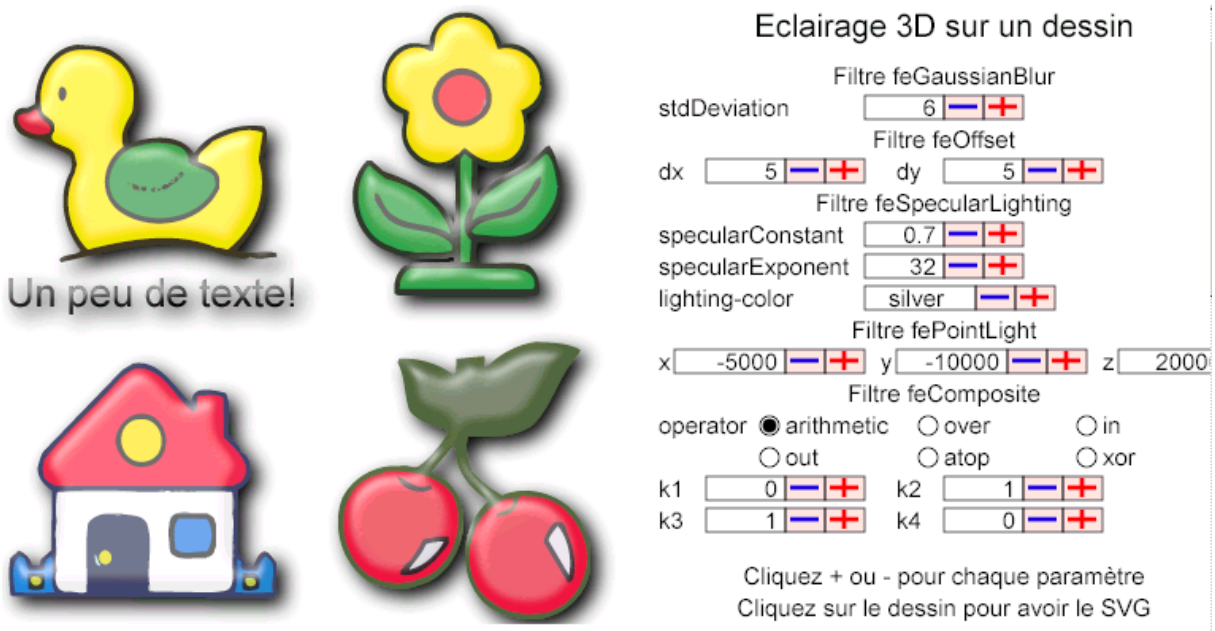

Figure 8-19. Tester les paramètres de cette combinaison de filtres

Comment créer des filtres?

Dans les outils de dessin, vous pouvez choisir un filtre avec quelques paramètres, mais vous ne pouvez pas créer votre filtre.
Vous trouverez donc en ligne (pilat.free.fr) ou sur le CD qui accompagne ce livre un outil où vous pouvez choisir vos primitives, vos images, modifier les paramètres, voir les résultats intermédiaires et évidemment récupérer le code du filtre (Merci PHP!)
Sur la figure 8-20 vous pouvez voir la liste des primitives et sur la figure 8-21 l'utilisation de composants SVG réutilisables pour modifier les valeurs des attributs.

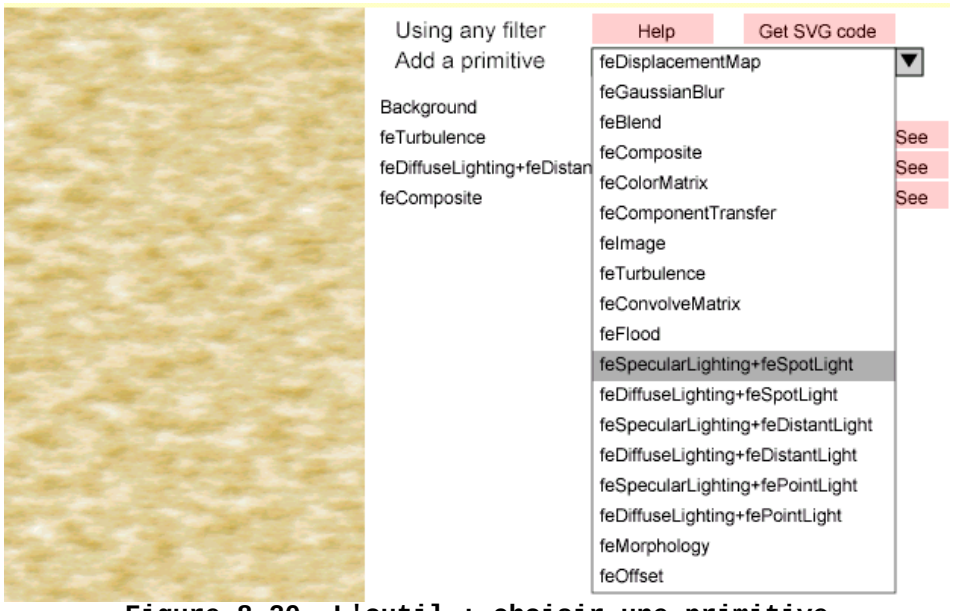

Figure 8-20. L'outil : choisir une primitive

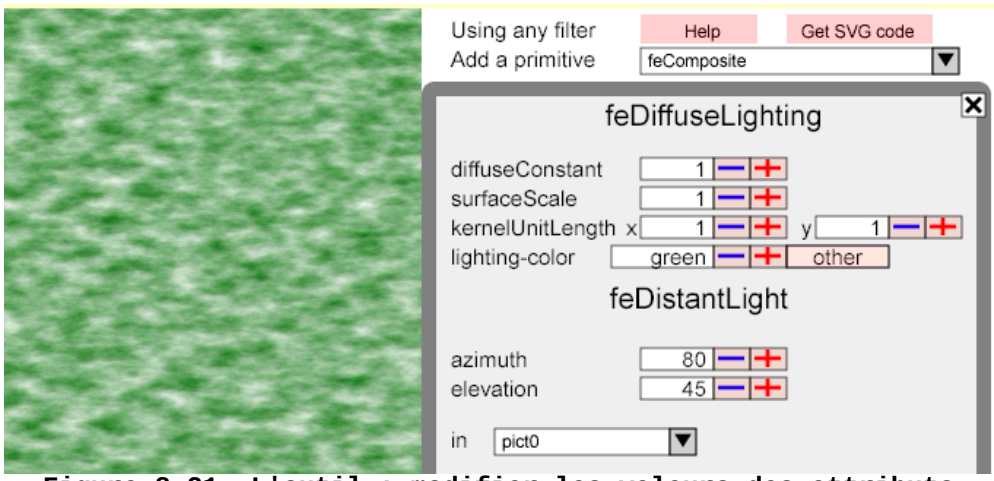

Figure 8-21. L'outil : modifier les valeurs des attributs

Filtres et animation

Tous les attributs des primitives peuvent être animés en recevant un élément 'animate' comme descendant de la primitive.

Premier exemple :
Mise au point sur une image en utilisant 'feMorphology', les valeurs de l'attribut 'radius' varieront de 8,8 à 0,0. L'image très déformée deviendra progressivement nette.
La figure 8-22 montre quelques étapes de l'animation.

Code pour cette animation :
Dans la section <defs>

```
<filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="400"
        height="400">
  <feImage xlink:href='puzzle.jpg' result='image1'/>
  <feMorphology in='image1' radius='8,8' operator='dilate'>
```

```
    <animate attributeName='radius' from='8,8' to='0,0' dur='5s' fill='freeze'
             begin='go.click' calcMode='paced'/>
  </feMorphology>
</filter>
```

Pour utiliser ce filtre

```
<rect filter="url(#MyFilter)" x='0' y='0' width='400' height='400'/>
```

Figure 8-22. Mise au point sur une image avec 'feMorphology'

Second exemple :

Pour simuler une éruption volcanique, en partant d'une photo prise lors d'une éruption sur l'île de la Réunion, nous utilisons la primitive 'feComponentTransfer' et dans son descendant 'feFuncR', l'attribut 'slope' varie de 0.2 à 2.2

Code pour cette animation :

Dans une section <defs>

```
<filter id='MyFilter' filterUnits='objectBoundingBox' x='0%' y='0%'
        width='100%' height='100%'>
  <feComponentTransfer>
    <feFuncR type='linear' slope='0.2' intercept='0'>
      <animate repeatCount="indefinite" attributeName='slope'
               Valeurs="0.2;2;0.2" dur='6s' begin='0s' calcMode='paced'/>
    </feFuncR>
  </feComponentTransfer>
</filter>
```

Pour utiliser ce filtre animé

```
<image filter='url(#MyFilter)' xlink:href='volcan.jpg' x='0' y='0'
       width="600" height="400"/>
```

La figure 8-23 montre quelques étapes de cette animation.

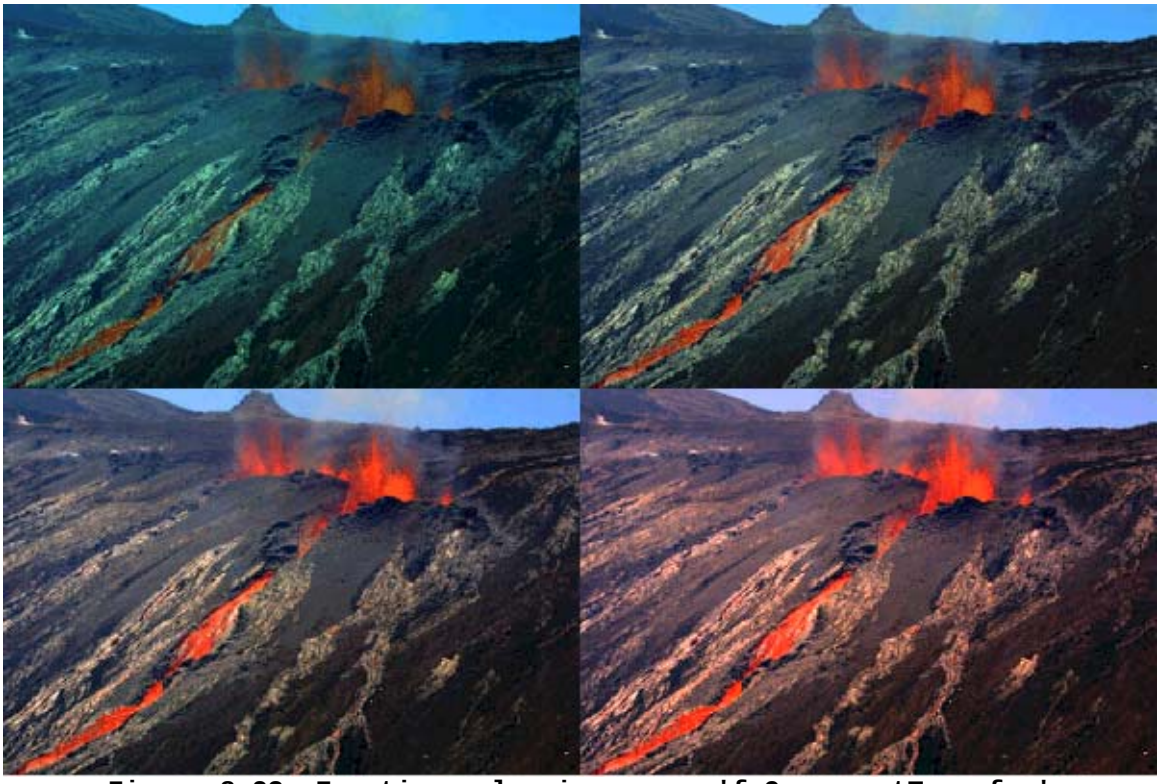

Figure 8-23. Eruption volcanique avec 'feComponentTransfer'

Et comme nous parlons d'animation, le prochain chapitre lui sera entièrement consacré.