

Contenu embarqué dans le SVG

Français ▼

[« Précédent](#)[Suivant »](#)

En plus des formes graphiques simples comme les rectangles et les cercles, le format SVG permet d'ajouter d'autres types de contenu aux images.

Embarquer des images

De la même façon qu'il est possible d'utiliser la balise **img** en HTML, le format SVG possède un élément **image** qui a la même utilité. Vous pouvez l'utiliser pour insérer des images bitmap ou vectorielles dans votre image SVG. La spécification définit que les formats PNG, JPEG et SVG au moins doivent être supportés.

L'image embarquée devient un élément SVG normal. Cela implique que vous pouvez utiliser le découpage, les masques, les filtres, les rotations et toute la panoplie des outils svg sur ce contenu embarqué :


```
<svg version="1.1"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
  width="200" height="200">
  <image x="90" y="-65" width="128" height="146" transform="rotate(45)"
    xlink:href="https://developer.mozilla.org/media/img/mdn-logo.png"/>
</svg>
```



Embarquer du contenu XML quelconque

Étant donné que le SVG est un document XML, il est toujours possible d'adjoindre un contenu XML quelconque n'importe où dans le document. Mais il n'y a évidemment aucun moyen de savoir comment l'élément SVG encadrant votre contenu réagira à ce qui aura été inséré. En fait, un lecteur SVG correct ne réagira d'aucune façon particulière et ignorera purement et simplement ce contenu. Si la spécification ajoute l'élément SVG **foreignObject**, son utilité est essentiellement d'être une coquille pour d'autres balises et de permettre d'adjoindre des attributs de style (comme par exemple la *largeur* et la *hauteur* de l'objet embarqué afin de définir la place que celui-ci occupera).

L'élément `foreignObject` est donc la bonne méthode pour embarquer du **XHTML** dans du SVG. Si le SVG doit contenir du texte de longueur conséquente, la disposition HTML est bien plus pratique et utilisable que l'élément SVG `text`. Une autre utilisation bien pratique de cet élément est l'adjonction de formules avec MathML. Pour des applications scientifiques utilisant le SVG, c'est un bon moyen de permettre la communication entre ces deux univers.

 **Note:** Gardez à l'esprit que le contenu du `foreignObject` doit pouvoir être analysé et pris en compte par votre lecteur SVG. Il y a peu de chances qu'un lecteur SVG autonome soit capable de restituer du contenu HTML or MathML.

Etant donné que le `foreignObject` est un élément SVG comme un autre, vous pouvez, comme dans le cas de l'élément `image`, utiliser toute la panoplie des attributs SVG qui pourrait s'appliquer au contenu embarqué.

[« Précédent](#)

[Suivant »](#)

🕒 **Dernière modification** : 23 mars 2019, by [MDN contributors](#)

×

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

[Sign up now](#)

[Technologies ▼](#)[Guides et références ▼](#)[Votre avis ▼](#)

Découpages et masquages

[Français ▼](#)[« Précédent](#)[Suivant »](#)

Effacer une partie de ce que l'on a créé précédemment peut paraître maladroit, voire totalement contradictoire. Mais cela peut se révéler très utile, par exemple quand vous essayez de dessiner un demi-cercle.

Le **découpage** (*clipping*) correspond au fait d'enlever des morceaux d'élément. Dans ce cas là, les effets de transparence ne sont pas permis, il s'agit d'une approche du tout-ou-rien.

D'un autre côté, le **masquage** (*masking*) permet plus de souplesse en prenant en compte la transparence et les niveaux de gris.

Découper


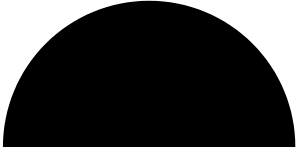
Pour créer un demi-cercle, on définit d'abord un élément `circle`:

```
1 <svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
2   <defs>
3     <clipPath id="cut-off-bottom">
4       <rect x="0" y="0" width="200" height="100" />
5     </clipPath>
6   </defs>
7
8   <circle cx="100" cy="100" r="100" clip-path="url(#cut-off-bottom)" />
9 </svg>
```

On dessine ici un cercle d'un rayon de 100 pixels, dont le centre est placé au point (100,100). L'attribut `clip-path` fait référence à l'élément `clipPath` définit plus haut, qui est généralement placé dans la section `defs`.

L'élément `clipPath` contient un simple rectangle qui, seul, remplirait en noir la moitié supérieur du canvas. Le rectangle ne sera pas dessiné, parce qu'il est définit dans un élément `clipPath`, il a pour effet de déterminer quels pixels seront affichés ou non dans le dessin final. Le rectangle ne couvrant que la partie supérieure du cercle, la partie inférieure du cercle ne sera pas affichée:

[Screenshot](#)[Live sample](#)

| Screenshot | Live sample |
|---|---|
|  |  |

Nous avons maintenant un demi-cercle, sans avoir à passer par un arc dans un élément `path`. Pour le découpage, chaque forme à l'intérieur de `clipPath` est inspecté et évalué avec ses propriétés et ses transformations. Chaque zone transparente dans `clipPath` aura pour effet de masquer le contenu. La couleur, l'opacité et autres n'ont pas d'effet tant qu'ils ne rendent pas les formes complètement transparentes.

Masquage

Le masquage, contrairement au découpage permet de travailler avec des gradients. Si vous voulez qu'un élément disparaisse progressivement, vous y parviendrez en utilisant des masques.

```

1  <svg width="200" height="200" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/19
2  <defs>
3    <linearGradient id="Gradient">
4      <stop offset="0" stop-color="white" stop-opacity="0" />
5      <stop offset="1" stop-color="white" stop-opacity="1" />
6    </linearGradient>
7    <mask id="Mask">
8      <rect x="0" y="0" width="200" height="200" fill="url(#Gradient)" />
9    </mask>
10 </defs>
11
12 <rect x="0" y="0" width="200" height="200" fill="green" />
13 <rect x="0" y="0" width="200" height="200" fill="red" mask="url(#Mask)" />
14 </svg>

```

Vous pouvez voir qu'on a défini un rectangle vert en-dessous d'un rectangle rouge. Ce dernier a un attribut `mask` qui pointe vers le masque situé dans les définitions. Le contenu du masque est un simple élément `rect`, qui est rempli d'un gradient transparent-vers-blanc. Les pixels du rectangle rouge héritent de la valeur alpha (la transparence) du contenu du masque, si bien que le rectangle rouge est progressivement masqué et laisse voir le rectangle vert en-dessous:

| Screenshot | Live sample |
|------------|-------------|
|------------|-------------|

| Screenshot | Live sample |
|---|---|
|  |  |

Transparence avec `opacity`

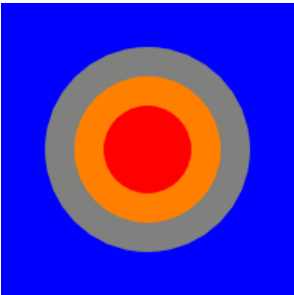
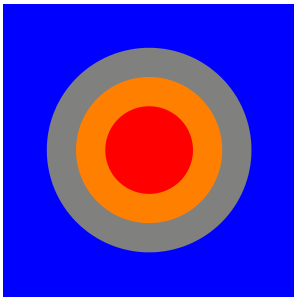
Pour définir la transparence d'un élément entier, on peut utiliser l'attribut `opacity`:

```
1 | <rect x="0" y="0" width="100" height="100" opacity=".5" />
```

Le rectangle ci-dessus sera dessiné semi-transparent.

On peut également utiliser deux attributs distincts pour le remplissage et le contour: `fill-opacity` et `stroke-opacity`, pour contrôler l'opacité des propriétés `fill` et `stroke` respectivement. Notez que le contour est dessiné au-dessus du remplissage. Ainsi, si vous rendez le contour semi-transparent et non le remplissage, celui-ci sera visible à travers le contour:

```
1 | <svg width="200" height="200" version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
2 |   <rect x="0" y="0" width="200" height="200" fill="blue" />
3 |   <circle cx="100" cy="100" r="50" stroke="yellow" stroke-width="40" stroke-opacity=".5" fill="red" />
4 | </svg>
```

| Screenshot | Live sample |
|---|---|
|  |  |

Vous pouvez voir dans cet exemple un cercle rouge sur un fond bleu. Le contour jaune a une opacité de 50%, si bien qu'on se retrouve avec une partie du remplissage en orange.

Utilisation de techniques CSS bien connues

Un des outils les plus puissants parmi l'arsenal du développeur web est `display: none`. Il n'est donc pas étonnant qu'il ait été décidé que cette propriété CSS serait également intégrée à SVG, de même que `visibility` et `clip` définis en CSS 2. Pour ré-afficher un élément précédemment caché avec `display: none` il est important de savoir que la valeur initiale des éléments SVG est `inline`.

🕒 Dernière modification : 23 mars 2019, by [MDN contributors](#)

Utilisation de techniques CSS bien connues



Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

[Sign up now](#)

SVG: Élément image

Français ▼

[« Précédent](#)[Suivant »](#)

L'élément SVG `<image>` permet d'afficher des images pixélisées au sein d'un objet SVG.

Dans cet exemple basique, une image JPG liée par l'attribut `xlink:href` sera rendue à l'intérieur d'un objet SVG.

```
1 <?xml version="1.0" standalone="no"?>
2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4 <svg width="5cm" height="4cm" version="1.1"
5     xmlns="http://www.w3.org/2000/svg" xlink:href="http://www.w3.org/1999/xlink">
6   <image xlink:href="firefox.jpg" x="0" y="0" height="50px" width="50px"/>
7 </svg>
```

Il faut prendre note de quelques point essentiels (donnés par les [spécifications W3](#)):

- Si les attributs x ou y ne sont pas spécifiés, ils vaudront 0.
- Si les attributs height ou width ne sont pas spécifiés, ils vaudront 0.
- Si l'attribut height ou l'attribut width est initialisé à 0, cela désactivera l'affichage de l'image.

[« Précédent](#)[Suivant »](#)

🕒 **Dernière modification** : 23 mars 2019, by [MDN contributors](#)

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.



[Technologies ▼](#)[Guides et références ▼](#)[Votre avis ▼](#)

Filtres

[Français ▼](#)[« Précédent](#)[Suivant »](#)

Dans certaines situations, les formes de base n'offrent pas la flexibilité nécessaire pour obtenir un certain effet. Par exemple, les ombres portées ne peuvent raisonnablement pas être créées avec des gradients. Les filtres sont des mécanismes SVG qui permettent de créer effets plus sophistiqués.

Un exemple de base consiste à ajouter un effet de flou au contenu du SVG. Bien que des effets de flou simples peuvent être obtenus avec les gradients, le filtre est nécessaire pour quelque chose de plus complexe.

Exemple

Les filtres sont définis par l'élément `<filter>`, qui doit être placé dans la section `<defs>` de votre fichier SVG. Entre les balises du filtre, se placent une liste de *primitives*, des opérations basiques qui s'ajoutent aux opérations précédentes (tel que du flou, de la lumière, etc). Pour appliquer le filtre créé sur un élément graphique, on définit l'attribut `filter`.

```
1 <svg width="250" viewBox="0 0 200 85"
2   xmlns="http://www.w3.org/2000/svg" version="1.1">
3 <defs>
4   <!-- Déclaration du filtre -->
5   <filter id="MyFilter" filterUnits="userSpaceOnUse"
6     x="0" y="0"
7     width="200" height="120">
8
9     <!-- offsetBlur -->
10    <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
11    <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
12
13    <!-- litPaint -->
14    <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
15      specularExponent="20" lighting-color="#bbbbbb"
16      result="specOut">
17      <fePointLight x="-5000" y="-10000" z="20000"/>
18    </feSpecularLighting>
19    <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
20    <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
21      k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
22
23    <!-- fusionne offsetBlur + litPaint -->
24    <feMerge>
```



```

25     <feMergeNode in="offsetBlur" />
26     <feMergeNode in="litPaint" />
27   </feMerge>
28 </filter>
29 </defs>
30
31 <!-- Éléments graphiques -->
32 <g filter="url(#MyFilter)">
33   <path fill="none" stroke="#D90000" stroke-width="10"
34     d="M50,66 c-50,0 -50,-60 0,-60 h100 c50,0 50,60 0,60z" />
35   <path fill="#D90000"
36     d="M60,56 c-30,0 -30,-40 0,-40 h80 c30,0 30,40 0,40z" />
37   <g fill="FFFFFF" stroke="black" font-size="45" font-family="Verdana" >
38     <text x="52" y="52">SVG</text>
39   </g>
40 </g>
41 </svg>

```



Étape 1

```

1 <feGaussianBlur in="SourceAlpha"
2     stdDeviation="4"
3     result="blur" />

```

`<feGaussianBlur>` prend en entrée (`in`) "SourceAlpha", qui est la couche alpha de l'élément source, applique un flou de 4, et stocke le résultat (`result`) dans un buffer temporaire nommé "blur".

Étape 2

```

1 <feOffset in="blur"
2     dx="4" dy="4"
3     result="offsetBlur" />

```

`<feOffset>` prend en entrée (`in`) "blur", qu'on a créé précédemment, le décale de 4 vers la droite et 4 vers le bas, et stocke le résultat (`result`) dans le buffer "offsetBlur". Les deux premières primitives viennent de créer une ombre portée.

Étape 3

```

1 <feSpecularLighting in="blur"
2     surfaceScale="5" specularConstant=".75"
3     specularExponent="20" lighting-color="#bbbbbb"
4     result="specOut">
5

```

```
6 | <fePointLight x="-5000" y="-10000" z="20000" />
   | </feSpecularLighting>
```

`<feSpecularLighting>` prend en entrée (`in`) "blur", génère un effet d'éclairage, et stocke le résultat (`result`) dans le buffer "specOut".

Étape 4

```
1 | <feComposite in="specOut" in2="SourceAlpha"
2 |           operator="in"
3 |           result="specOut" />
```

Le premier `<feComposite>` prend en entrée (`in`, `in2`) "specOut" et "SourceAlpha", masque le résultat de "specOut" de telle sorte qu'il ne soit pas plus grand que "SourceAlpha" (l'élément graphique d'origine), et remplace le résultat (`result`) "specOut".

Étape 5

```
1 | <feComposite in="SourceGraphic" in2="specOut"
2 |           operator="arithmetic"
3 |           k1="0" k2="1" k3="1" k4="0"
4 |           result="litPaint" />
```

Le second `<feComposite>` prend en entrée (`in`, `in2`) "SourceAlpha" et "specOut", ajoute le résultat "specOut" au-dessus de "SourceAlpha", et stocke le résultat (`result`) dans "litPaint".

Étape 6

```
1 | <feMerge>
2 |   <feMergeNode in="offsetBlur" />
3 |   <feMergeNode in="litPaint" />
4 | </feMerge>
```

Finalement, `<feMerge>` fusionne ensemble "offsetBlur", qui est l'ombre portée, et "litPaint", qui est l'élément d'origine avec l'effet d'éclairage.



Élément d'origine

Primitive 1

Primitive 2

Primitive 3

Primitive 4

Primitive 5

Primitive 6

[« Précédent](#)

[Suivant »](#)

Technologies ▼

Guides et références ▼

Votre avis ▼

Formes de base

Français ▼

« Précédent

Suivant »

Il existe tout un ensemble de formes de base utilisées pour faire du dessin via SVG. Le but de ces formes assez transparent, si on regarde attentivement les noms de chaque élément. Des attributs permettent de configurer leur position et leur taille, mais vous pourrez retrouver les détails de chaque élément avec tous ses attributs à [la page des références SVG](#). Nous nous contenterons ici de couvrir les fonctions de base qui nous sont nécessaires, car elles sont utilisées dans la plupart des documents SVG.

Ajout de formes

Pour insérer une forme, vous devez ajouter un élément dans un document. Des éléments différents correspondent à des formes différentes et ont des attributs différents pour décrire leur taille et leur position. Certaines déclarations sont très fortement redondantes en ce qu'elles peuvent être créées par d'autres formes, mais elles sont toutes là de manière à faciliter votre vie et à rendre le document SVG aussi court et lisible que possible. Toutes les formes de bases sont affichées sur l'image de gauche. Le code pour générer tout cela ressemble à cela :



```
1 <?xml version="1.0" standalone="no"?>
2 <svg width="200" height="250" version="1.1" xmlns="http://www.w3.org/2000/svg">
3
4   <rect x="10" y="10" width="30" height="30" stroke="black" fill="transparent" stroke-width="5"/>
5   <rect x="60" y="10" rx="10" ry="10" width="30" height="30" stroke="black" fill="transparent" stroke-width="5"/>
6
7   <circle cx="25" cy="75" r="20" stroke="red" fill="transparent" stroke-width="5"/>
8   <ellipse cx="75" cy="75" rx="20" ry="5" stroke="red" fill="transparent" stroke-width="5"/>
9
10  <line x1="10" x2="50" y1="110" y2="150" stroke="orange" fill="transparent" stroke-width="5"/>
11  <polyline points="60 110 65 120 70 115 75 130 80 125 85 140 90 135 95 150 100 145"
12    stroke="orange" fill="transparent" stroke-width="5"/>
13
14  <polygon points="50 160 55 180 60 190 65 205 70 180 75 160 80 190 85 205 90 180 95 160 100 190 105 180 110 160"
15    stroke="green" fill="transparent" stroke-width="5"/>
```

```
16 | <path d="M20,230 Q40,205 50,230 T90,230" fill="none" stroke="blue" stroke-width="5"/>
17 | </svg>
18 |
```

Note : les attributs `stroke`, `stroke-width` et `fill` sont détaillés plus loin dans ce tutoriel.

Figures de bases

Rectangles

L'élément `rect`, comme son nom ne l'indique peut-être pas, dessine à l'écran des rectangles. Il existe 6 attributs de base qui contrôlent la position et la forme du rectangle dessiné ici. L'image précédente affichait 2 rectangles, ce qui est un peu répétitif. Celui de droite possède des attributs `rx` et `ry` définis, ce qui lui donne des coins arrondis. Si ces attributs ne sont pas définis, leur valeur par défaut est de 0, ce qui a pour résultats d'afficher un rectangle avec des angles droits.

```
1 | <rect x="10" y="10" width="30" height="30"/>
2 | <rect x="60" y="10" rx="10" ry="10" width="30" height="30"/>
```

x

Position du rectangle sur l'axe horizontal par rapport au coin supérieur gauche.

y

Position du rectangle sur l'axe vertical par rapport au coin supérieur gauche.

width

Largeur du rectangle.

height

Hauteur du rectangle.

rx

Rayon x des coins du rectangle.

ry

Rayon y des coins du rectangle.

Cercles

De la même manière, il est facile de deviner la fonction de l'élément `circle`. Il dessine à l'écran un cercle. Seuls 3 attributs peuvent être définis pour cet élément.

```
1 | <circle cx="25" cy="75" r="20"/>
```

r

Rayon du cercle.

cx

Position x du centre du cercle.

cy

Position y du centre du cercle.

Ellipses

Les **ellipses** sont juste des sortes de cercles bien particuliers, où l'on peut modifier les rayons x et y séparément l'un de l'autre (les mathématiciens appellent ces rayons le grand axe et le petit axe).

```
1 | <ellipse cx="75" cy="75" rx="20" ry="5" />
```

rx

Rayon x de l'ellipse.

ry

Rayon y de l'ellipse.

cx

Position x du centre de l'ellipse.

cy

Position y du centre de l'ellipse.

Figures complexes

Lignes

Les lignes droites permettent de créer des figures plus complexes, en les additionnant les unes avec les autres. L'élément **line** en SVG correspond au segment que l'on apprend en géométrie traditionnelle : c'est une portion de droite délimitée par 2 points. Donc pour définir une droite en SVG, il va falloir lui donner pour attribut les coordonnées des deux points qui la définissent.

```
1 | <line x1="10" x2="50" y1="110" y2="150" />
```

x1

Position x du premier point.

x2

Position x du deuxième point.

y1

Position y du premier point.

y2

Position y du deuxième point.

Lignes brisées

Les lignes brisées, aussi appelées lignes polygonales, sont définies par l'élément **polyline** en SVG. Elles sont constituées d'un ensemble de lignes droites connectées entre elles, donc d'un ensemble de points se reliant entre eux suivant un ordre défini. Comme ce lot de points peut être assez conséquent à déclarer, un seul attribut est utilisé pour déclarer l'ensemble de points :

```
1 | <polyline points="60 110, 65 120, 70 115, 75 130, 80 125, 85 140, 90 135, 95 150, 100 145" />
```

points

Liste des points, chaque pair de nombres correspondant aux coordonnées x et y de chaque point. Chaque position x est séparée de la position y par un espace. Chaque ensemble de coordonnées est séparé du suivant par une virgule.

Polygones

Le `polygone` fonctionne exactement de la même manière que la ligne brisée. Au final, un polygone n'est rien d'autre qu'une ligne brisée qui relie une série de points. Toutefois, pour les polygones, le chemin de cette ligne retourne automatiquement au point de départ, créant ainsi une forme fermée. Il est à noter que le rectangle est un type de polygone particulier. Il est donc possible, pour des besoins de flexibilité, de déclarer un rectangle en utilisant l'élément `polygone`.

```
1 | <polygone points="50 160, 55 180, 70 180, 60 190, 65 205, 50 195, 35 205, 40 190, 30 180, 45 180"/>
```

points

Idem que l'attribut `points` de l'élément `polyline`. Liste des points, chaque paire de nombres correspondant aux coordonnées x et y de chaque point. Chaque position x est séparée de la position y par un espace, chaque ensemble de coordonnées est séparé du suivant par une virgule. Une dernière ligne ferme automatiquement la forme en retournant au point de départ.

Chemins

L'élément pour tracer les chemins, très logiquement nommé `path`, est sûrement la forme la plus généraliste qui peut être utilisée en SVG. Avec un élément `path`, vous pouvez dessiner un rectangle (avec ou sans coins arrondis), des cercles, des ellipses, des lignes brisées et des polygones. De manière plus basique, il est aussi possible de dessiner d'autres types de formes, comme des courbes de Bézier, des paraboles, et bien plus encore. Pour cette raison, l'élément `path` en lui-même sera un chapitre entier de ce tutoriel, mais pour le moment, nous allons juste voir comment définir cet élément.

```
1 | <path d="M 20 230 Q 40 205, 50 230 T 90230"/>
```

d

Un ensemble d'information définissant le chemin à dessiner. Pour en savoir plus, allez à la page à propos des [Chemins](#).

[« Précédent](#)

[Suivant »](#)

© Dernière modification : 23 mars 2019, by [MDN contributors](#)

×

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Technologies ▼

Guides et références ▼

Votre avis ▼

Gradients SVG

Français ▼

[« Précédent](#)[Suivant »](#)

Probablement plus excitant qu'un simple remplissage et contour, est le fait de pouvoir créer et appliquer des dégradés comme remplissage ou contour.

Il y a deux types de dégradés: linéaire et radial. Les dégradés sont définis dans la section `defs` et non sur les formes elles-mêmes — cela favorise leur réutilisabilité. Vous **devez** donner au dégradé un attribut `id`; autrement, il ne pourra pas être utilisé par les autres éléments à l'intérieur du fichier SVG.

Dégradé Linéaire

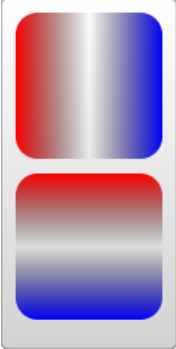
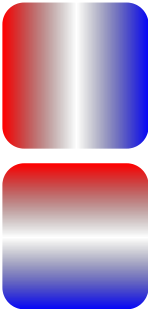
Les dégradés linéaires (*linear gradient* en anglais) changent de couleur le long d'une ligne droite. Pour en insérer un, on crée un élément `<linearGradient>` dans la section des définitions du fichier SVG.

Exemple

Un exemple de dégradé linéaire appliqué à un élément `<rect>`:

```
1 <svg width="120" height="240" version="1.1" xmlns="http://www.w3.org/2000/svg">
2   <defs>
3     <linearGradient id="Gradient1" x1="0" x2="0" y1="0" y2="1">
4       <stop offset="0%" stop-color="red"/>
5       <stop offset="50%" stop-color="black" stop-opacity="0"/>
6       <stop offset="100%" stop-color="blue"/>
7     </linearGradient>
8     <linearGradient id="Gradient2">
9       <stop class="stop1" offset="0%"/>
10      <stop class="stop2" offset="50%"/>
11      <stop class="stop3" offset="100%"/>
12    </linearGradient>
13    <style type="text/css"><![CDATA[
14      #rect1 { fill: url(#Gradient2); }
15      .stop1 { stop-color: red; }
16      .stop2 { stop-color: black; stop-opacity: 0; }
17      .stop3 { stop-color: blue; }
18    ]]></style>
19  </defs>
20
21  <rect x="10" y="120" rx="15" ry="15" width="100" height="100" fill="url(#Gradient1)"/>
```

```
22 | <rect x="10" y="10" rx="15" ry="15" width="100" height="100" id="rect1" />
23 | </svg>
```

| Screenshot | Live sample |
|---|---|
|  |  |

Définir le dégradé

À l'intérieur du dégradé, il y a divers noeuds `<stop>`. Ces noeuds disent au dégradé quelles couleurs doivent être affichées à quelles positions, en spécifiant les attributs `offset` pour la position et `stop-color` pour la couleur. On peut également le définir avec CSS. Les deux méthodes ont été utilisées dans l'exemple pour le démontrer.

Dans cet exemple, on dit au dégradé de commencer en rouge, de passer au noir transparent au centre et de terminer par la couleur bleue. Vous pouvez ajouter autant de couleurs que vous le souhaitez, pour créer un dégradé aussi beau ou aussi laid que vous le souhaitez, mais les positions (`offset`) doivent toujours être incrementées de 0% (ou 0) à 100% (ou 1). Si des valeurs sont dupliquées, la couleur définie la plus en bas de la définition sera utilisée.

Aussi, comme pour le remplissage et le contour, vous pouvez spécifier un attribut `stop-opacity` pour définir l'opacité de la couleur à cette position (encore une fois, à partir de FF3 vous pouvez utiliser les valeurs rgba pour le même effet).

```
1 | <stop offset="100%" stop-color="yellow" stop-opacity="0.5" />
```

Utiliser le dégradé

Pour utiliser le dégradé, vous devez le référencer avec l'attribut `fill` ou `stroke` d'un objet. On référence un élément SVG de la même manière que l'on référence des éléments en CSS, via `url()`. Dans notre cas, l'url est juste une référence vers le dégradé avec l'ID "Gradient". Pour le référencer, on définit `fill="url(#Gradient)"`, et voilà! Notre objet est maintenant multi-couleur. Vous pouvez faire de même avec `stroke`.

Orientation du dégradé

L'élément `<linearGradient>` peut également prendre différents attributs pour spécifier la taille et l'apparence du dégradé. L'orientation du dégradé est contrôlé par deux points, désignés par les attributs `x1`, `x2`, `y1`, et `y2`. Ces attributs définissent la ligne le long de laquelle le dégradé est tracé. Par défaut, le dégradé est horizontal, mais il peut être orienté autrement grâce à ces attributs. "Gradient2" dans l'exemple précédent crée un dégradé vertical.


```
1 | <linearGradient id="Gradient2" x1="0" x2="0" y1="0" y2="1">
```

xlink:href

Vous pouvez également utiliser l'attribut `xlink:href` sur les dégradés. Quand il est utilisé, les attributs et stops d'un dégradé peuvent être réutilisé sur un autre. Ainsi, dans l'exemple précédent, on aurait pu ne pas redéfinir tous les stops dans Gradient2, comme ceci:

```
<linearGradient id="Gradient1">
  <stop id="stop1" offset="0%"/>
  <stop id="stop2" offset="50%"/>
  <stop id="stop3" offset="100%"/>
</linearGradient>
<linearGradient id="Gradient2" x1="0" x2="0" y1="0" y2="1"
  xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#Gradient1"/>
```

Ici, le namespace `xlink` est inclut directement sur le noeud, bien qu'il soit généralement défini en haut du document, comme dans [l'exemple avec les images](#)

Dégradé Radial

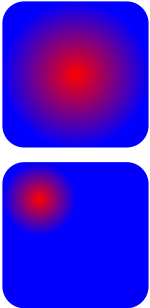
Les dégradés radiaux (*radial gradient* en anglais) sont similaires aux dégradés linéaires à la différence près qu'ils irradient autour d'un point. Pour en créer un, on crée un élément `<radialGradient>` dans la section de définitions du document SVG.

Exemple

```
1 | <svg width="120" height="240" version="1.1" xmlns="http://www.w3.org/2000/svg">
2 |   <defs>
3 |     <radialGradient id="RadialGradient1">
4 |       <stop offset="0%" stop-color="red"/>
5 |       <stop offset="100%" stop-color="blue"/>
6 |     </radialGradient>
7 |     <radialGradient id="RadialGradient2" cx="0.25" cy="0.25" r="0.25">
8 |       <stop offset="0%" stop-color="red"/>
9 |       <stop offset="100%" stop-color="blue"/>
10 |    </radialGradient>
11 |  </defs>
12 |
13 |  <rect x="10" y="10" rx="15" ry="15" width="100" height="100" fill="url(#RadialGradient1)"/>
14 |  <rect x="10" y="120" rx="15" ry="15" width="100" height="100" fill="url(#RadialGradient2)"/>
15 | </svg>
```

Screenshot

Live sample

| Screenshot | Live sample |
|---|---|
|  |  |

Définir le dégradé

Les stops utilisés ici sont les mêmes que précédemment, la différence étant que désormais l'objet sera rouge en son centre, et que la couleur changera progressivement vers le bleu en approchant des contours. Comme pour les dégradés linéaires, le noeud `<radialGradient>` peut prendre différents attributs pour décrire sa position et son orientation. Cependant, la définition est un peu plus complexe. Le dégradé linéaire est défini par deux points, qui déterminent où sont situé le centre et les bords:

- Le premier point définit le cercle où le dégradé se termine. Il requiert un point central, spécifié par les attributs `cx` et `cy`, et un rayon, `r`. Définir ces trois attributs vous permettra de déplacer le dégradé et de changer sa taille, comme illustré dans le deuxième `rect` de notre exemple.
- Le second point est appelé le point focal et il est défini par les attributs `fx` et `fy`. Tandis que le premier point décrit où sont les bords du dégradé, le point focal décrit où est son centre. C'est plus facile à voir avec un exemple (voir la section qui suit).

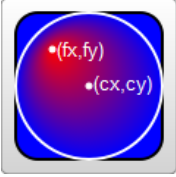
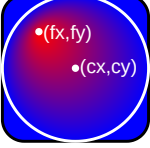
Centre et point focal

```

1 <svg width="120" height="120" version="1.1"
2   xmlns="http://www.w3.org/2000/svg">
3   <defs>
4     <radialGradient id="Gradient"
5       cx="0.5" cy="0.5" r="0.5" fx="0.25" fy="0.25">
6       <stop offset="0%" stop-color="red"/>
7       <stop offset="100%" stop-color="blue"/>
8     </radialGradient>
9   </defs>
10
11 <rect x="10" y="10" rx="15" ry="15" width="100" height="100"
12   fill="url(#Gradient)" stroke="black" stroke-width="2"/>
13
14 <circle cx="60" cy="60" r="50" fill="transparent" stroke="white" stroke-width="2"/>
15 <circle cx="35" cy="35" r="2" fill="white" stroke="white"/>
16 <circle cx="60" cy="60" r="2" fill="white" stroke="white"/>
17 <text x="38" y="40" fill="white" font-family="sans-serif" font-size="10pt">(fx,fy)</text>
18 <text x="63" y="63" fill="white" font-family="sans-serif" font-size="10pt">(cx,cy)</text>
19
20 </svg>

```

| Screenshot | Live sample |
|------------|-------------|
|------------|-------------|

| Screenshot | Live sample |
|---|---|
|  |  |

Si le point focal est déplacé en dehors du cercle décrit précédemment, il est impossible que le dégradé s'affiche correctement, le point focal sera donc supposé être à l'intérieur du bord du cercle. Si le point focal n'est pas du tout indiqué, il sera supposé être au même endroit que le point central.

Attributs additionnels

Les dégradés linéaires et radiaux peuvent également prendre quelques autres attributs pour décrire les transformations qu'ils peuvent subir.

spreadMethod

Cet attribut contrôle ce qu'il arrive quand le dégradé arrive à sa fin, mais que l'objet n'est pas encore rempli. Trois valeurs sont possibles: "pad", "reflect", ou "repeat".

- "pad" est la valeur par défaut. Quand un dégradé arrive à sa fin, la dernière couleur est utilisée pour remplir le reste de l'objet.
- "reflect" a pour effet de poursuivre le dégradé, mais en sens inverse: de la dernière couleur (offset 100%) à la première (offset 0%), puis de nouveau de la première à la dernière, etc.
- "repeat" poursuit également le dégradé, mais au lieu de revenir en arrière, il revient au début et est exécuté de nouveau.

```

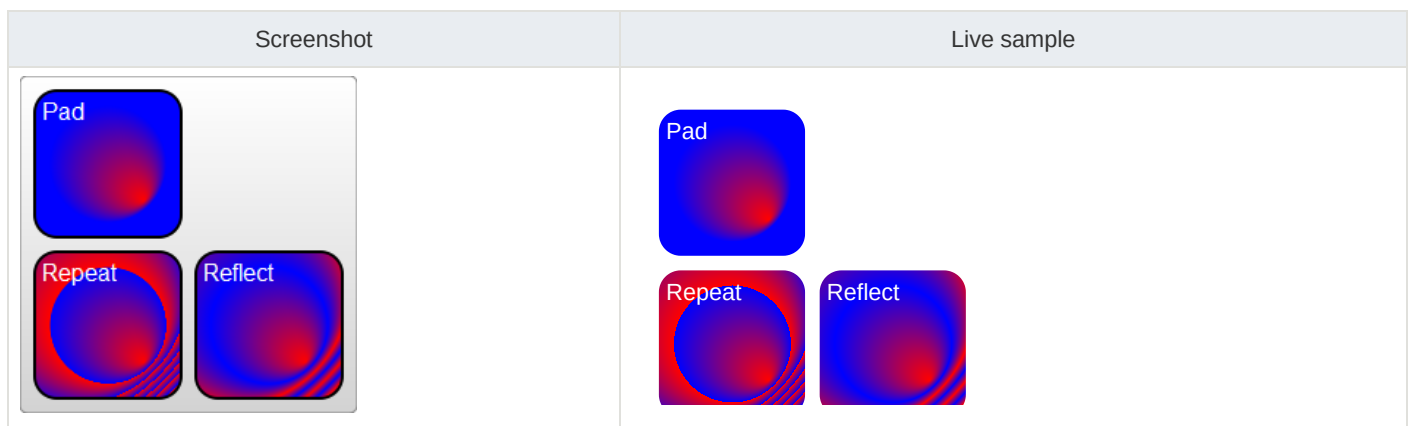
1 <svg width="220" height="220" version="1.1" xmlns="http://www.w3.org/2000/svg">
2   <defs>
3     <!-- pad -->
4     <radialGradient id="GradientPad"
5       cx="0.5" cy="0.5" r="0.4" fx="0.75" fy="0.75"
6       spreadMethod="pad">
7       <stop offset="0%" stop-color="red"/>
8       <stop offset="100%" stop-color="blue"/>
9     </radialGradient>
10
11    <!-- repeat -->
12    <radialGradient id="GradientRepeat"
13      cx="0.5" cy="0.5" r="0.4" fx="0.75" fy="0.75"
14      spreadMethod="repeat">
15      <stop offset="0%" stop-color="red"/>
16      <stop offset="100%" stop-color="blue"/>
17    </radialGradient>
18
19    <!-- reflect -->
20    <radialGradient id="GradientReflect"
21      cx="0.5" cy="0.5" r="0.4" fx="0.75" fy="0.75"
22      spreadMethod="reflect">

```

```

22     <stop offset="0%" stop-color="red"/>
23     <stop offset="100%" stop-color="blue"/>
24   </radialGradient>
25 </defs>
26
27 <rect x="10" y="10" rx="15" ry="15" width="100" height="100" fill="url(#GradientPad)"/>
28 <rect x="10" y="120" rx="15" ry="15" width="100" height="100" fill="url(#GradientRepeat)"/>
29 <rect x="120" y="120" rx="15" ry="15" width="100" height="100" fill="url(#GradientReflect)"/>
30
31 <text x="15" y="30" fill="white" font-family="sans-serif" font-size="12pt">Pad</text>
32 <text x="15" y="140" fill="white" font-family="sans-serif" font-size="12pt">Repeat</text>
33 <text x="125" y="140" fill="white" font-family="sans-serif" font-size="12pt">Reflect</text>
34
35 </svg>
36

```



gradientUnits

Les deux types de dégradés ont également un attribut `gradientUnits`, qui indique l'unité utilisée pour décrire la taille et l'orientation du dégradé. Deux valeurs sont possibles: `userSpaceOnUse` ou `objectBoundingBox`.

- `objectBoundingBox` est la valeur par défaut, c'est ce qu'on a vu jusqu'à présent. Le dégradé est automatiquement redimensionné à la taille de l'objet sur lequel il est appliqué, vous n'avez donc qu'à spécifier les coordonnées de zéro à un (ou de 0% à 100%), et les coordonnées sont automatiquement redimensionnée à la taille de l'objet.
- `userSpaceOnUse` indique que les valeurs sont absolues. Vous devez donc savoir où se situe l'objet, et placer le dégradé à la même position. Le dégradé radial précédent devrait être ré-écrit comme suit:

```

1   <radialGradient id="Gradient"
2       cx="60" cy="60" r="50"
3       fx="35" fy="35"
4       gradientUnits="userSpaceOnUse">

```

Il y a quelques subtilités concernant l'utilisation de `gradientUnits="objectBoundingBox"` quand les limites de l'objet ne sont pas carrées, mais elles sont assez complexes et nous attendrons quelqu'un de plus au courant pour les expliquer.

gradientTransform

Vous pouvez également appliquer une transformation au gradient en utilisant l'attribut `gradientTransform`, mais puisque nous n'avons pas encore introduit les [transformations](#), nous le laisserons de côté pour l'instant.

[« Précédent](#)

[Suivant »](#)

🕒 **Dernière modification :** 18 mars 2019, by MDN contributors

x

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

vous@example.com

Sign up now

Motifs

Français ▼

[« Précédent](#)[Suivant »](#)

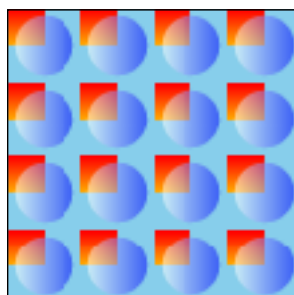
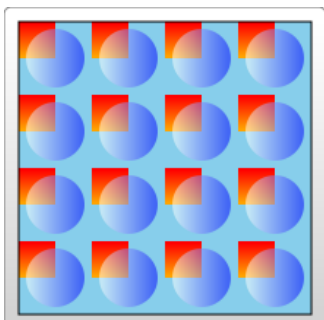
Les motifs (*patterns* en anglais) sont sans aucun doute les types de remplissages les plus complexes à utiliser en SVG. Ce sont également des outils très puissants, ils méritent donc d'être abordés pour que vous en connaissiez les fondamentaux. Comme les dégradés, l'élément `<pattern>` doit être placé dans la section `<defs>` du fichier SVG.

Exemple

```
1 <svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
2   <defs>
3     <linearGradient id="Gradient1">
4       <stop offset="5%" stop-color="white"/>
5       <stop offset="95%" stop-color="blue"/>
6     </linearGradient>
7     <linearGradient id="Gradient2" x1="0" x2="0" y1="0" y2="1">
8       <stop offset="5%" stop-color="red"/>
9       <stop offset="95%" stop-color="orange"/>
10    </linearGradient>
11
12    <pattern id="Pattern" x="0" y="0" width=".25" height=".25">
13      <rect x="0" y="0" width="50" height="50" fill="skyblue"/>
14      <rect x="0" y="0" width="25" height="25" fill="url(#Gradient2)"/>
15      <circle cx="25" cy="25" r="20" fill="url(#Gradient1)" fill-opacity="0.5"/>
16    </pattern>
17  </defs>
18
19  <rect fill="url(#Pattern)" stroke="black" width="200" height="200"/>
20 </svg>
```

Screenshot

Live sample



À l'intérieur de l'élément `pattern`, vous pouvez inclure toutes les formes de bases de SVG et les styliser de la même manière que d'habitude (remplissage, contour, dégradés, opacité, etc). Dans notre exemple, on a dessiné un cercle et deux rectangles (qui se chevauchent et dont l'un est deux fois plus grand que l'autre pour remplir le motif en entier).

La partie pouvant apporter le plus de confusion avec les motifs est le système d'unité et la taille des éléments.

Unités du motif: `objectBoundingBox`

Les attributs `width` et `height` sur l'élément `pattern` décrivent jusqu'où le motif doit aller avant de se répéter. Les attributs `x` et `y` sont également disponibles si vous souhaitez décaler le point de départ du motif à l'intérieur du dessin.

Même principe que l'attribut `gradientUnits` (que nous avons vu précédemment avec les dégradés), les motifs peuvent prendre un attribut `patternUnits`, pour spécifier l'unité utilisée par le motif. La valeur par défaut est `"objectBoundingBox"`, ainsi une taille de 1 remplira entièrement la hauteur/largeur de l'objet auquel le motif est appliqué. Puisque dans notre cas, on veut que le motif se répète 4 fois horizontalement et verticalement, on a défini `height` et `width` à 0.25. Cela signifie que la hauteur et largeur du pattern sera de 25% celle de l'objet.

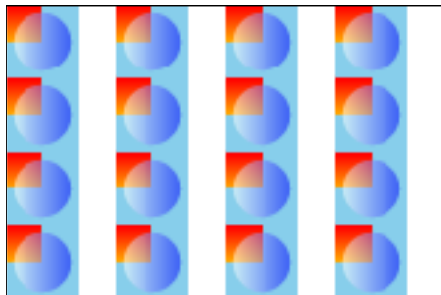
De même, pour que le motif commence à 10 pixels du bord supérieur-gauche de l'objet, il faudrait définir les valeurs de `x` et `y` à 0.05 ($10/200 = 0.05$).

Unités du contenu: `userSpaceOnUse`

Contrairement aux dégradés, les motifs ont un deuxième argument, `patternContentUnits`, qui lui spécifie l'unité utilisée par les formes à l'intérieur du motif. La valeur par défaut est `"userSpaceOnUse"`, l'opposé de l'attribut `patternUnits`. Cela signifie qu'à moins de définir ces attributs autrement (`patternContentUnits` et/ou `patternUnits`), les formes que vous dessinez à l'intérieur du motif ont un système de coordonnées différent du motif, ce qui peut rendre les choses un peu déroutantes si vous écrivez le code à la main.

Pour que cela fonctionne dans l'exemple ci-dessus, nous avons dû prendre en compte la taille du rectangle sur lequel est appliqué le motif (200px) et le fait que l'on veut répéter le motif 4 fois horizontalement et verticalement, donc que le motif sera un carré de 50x50. Les deux rectangles et le cercle à l'intérieur du motif ont été dimensionnés pour tenir dans un carré de 50x50. Tout ce qui sortirait en dehors ne serait pas affiché.

La chose à retenir est que si l'objet change de taille, le motif lui-même sera mis à l'échelle mais les objets à l'intérieur non. Ainsi, alors qu'on aura toujours 4 motifs qui se répètent horizontalement et verticalement, les objets à l'intérieur du motif garderont la même taille, et une zone vide sera affichée.



Edit Reset

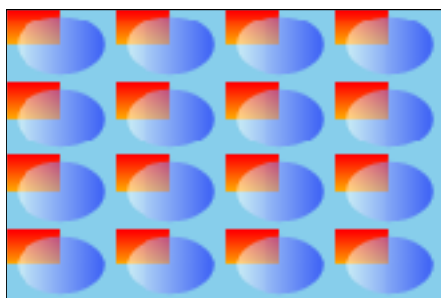
```
rect.setAttribute('width', 300);
```

Unités du contenu: `objectBoundingBox`

En changeant l'attribut `patternContentUnits`, on peut utiliser le même système d'unité pour tous les éléments:

```
1 <pattern id="Pattern" width=".25" height=".25" patternContentUnits="objectBoundingBox">
2   <rect x="0" y="0" width=".25" height=".25" fill="skyblue"/>
3   <rect x="0" y="0" width=".125" height=".125" fill="url(#Gradient2)"/>
4   <circle cx=".125" cy=".125" r=".1" fill="url(#Gradient1)" fill-opacity="0.5"/>
5 </pattern>
```

Maintenant, parce le contenu du motif utilise le même système d'unité que le motif, le motif redimensionne automatiquement son contenu. Cela contraste avec le système "userSpaceOnUse" par défaut, où lorsque le motif change la taille, le contenu garde la même taille.



Edit Reset

```
rect.setAttribute('width', 300);
```

Note: Dans Gecko, les cercles semblent avoir du mal à être dessinés si le rayon est inférieur à 0.075 (on ignore s'il s'agit d'un bug de l'élément `pattern` ou non). Pour contourner ce

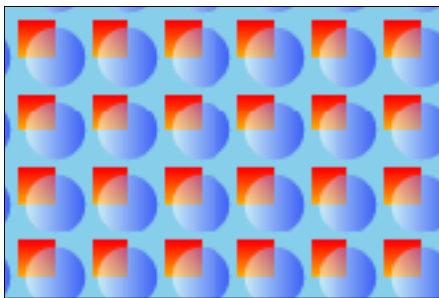
problème, il est probablement préférable d'éviter de dessiner des cercles dans des unités "objectBoundingBox".

Unités du motif: userSpaceOnUse

Aucune des utilisations vu jusqu'ici ne correspond à l'usage habituel des motifs (tel qu'on le ferait en CSS): les motifs ont généralement une taille définie et se répètent indépendamment de la taille de l'objet sur lequel il est appliqué. Pour créer quelque chose comme ça, le motif et le contenu doivent être dessiné en mode "userSpaceOnUse":

```
1 <pattern id="Pattern" x="10" y="10" width="50" height="50" patternUnits="userSpaceOnUse">
2   <rect x="0" y="0" width="50" height="50" fill="skyblue" />
3   <rect x="0" y="0" width="25" height="25" fill="url(#Gradient2)" />
4   <circle cx="25" cy="25" r="20" fill="url(#Gradient1)" fill-opacity="0.5" />
5 </pattern>
```

Bien sûr, cela veut dire que le motif ne sera pas mis à l'échelle si vous modifiez la taille de l'objet ultérieurement.

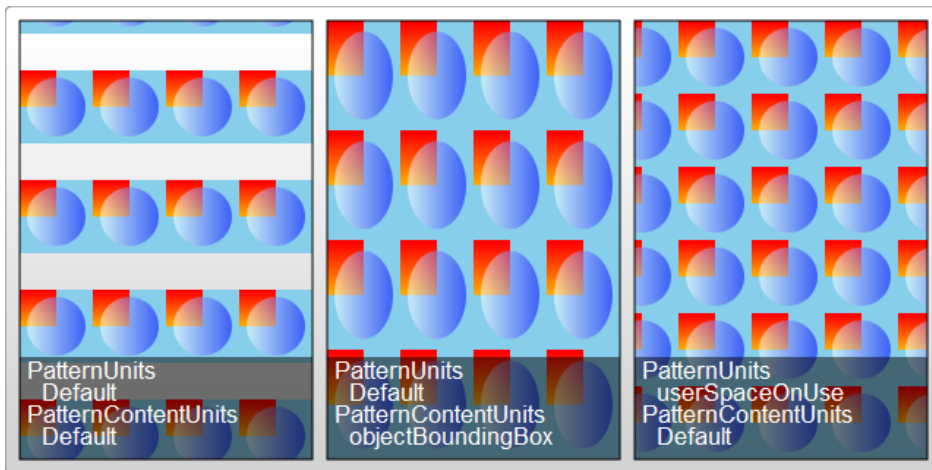


Edit Reset

```
rect.setAttribute('width', 300);
```

Récapitulatif

Les trois exemples sont illustrés ci-dessous sur un rectangle allongé à une hauteur de 300px:



« Précédent

Suivant »

🕒 **Dernière modification** : 18 mars 2019, by [MDN contributors](#)

Exemple

Unités du motif: `objectBoundingBox`

Unités du contenu: `userSpaceOnUse`

Unités du contenu: `objectBoundingBox`

Unités du motif: `userSpaceOnUse`

Récapitulatif

×

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

Sign up now

[Technologies ▼](#)[Guides et références ▼](#)[Votre avis ▼](#)

Paths

[Français ▼](#)[« Précédent](#)[Suivant »](#)

L'élément `<path>` (*chemin* en français) est le plus versatile des éléments de la bibliothèque SVG parmi les [formes basiques](#). Vous pouvez l'utiliser pour créer des lignes, des courbes, des arcs et autres.

Les chemins créent des formes en combinant plusieurs lignes droites ou courbes. Les formes composées uniquement de lignes droites peuvent être créées avec des [lignes brisées](#) (*polylines*). Bien que les lignes brisées et les chemins peuvent tout deux créer des formes d'apparence similaire, les lignes brisées nécessitent un grand nombre de petites lignes pour simuler des courbes, et qui ne s'adaptent pas bien aux grandes tailles. Une bonne compréhension des chemins est importante pour dessiner en SVG. Bien qu'il ne soit pas recommandé d'éditer des chemins complexes avec un éditeur XML ou texte (on utilisera plutôt un éditeur SVG tel que Inkscape ou Adobe Illustrator), comprendre comment un chemin s'écrit vous permettra éventuellement d'identifier et de corriger des erreurs d'affichage dans un SVG.

La forme d'un élément path est définie par son attribut `d`. Celui-ci prend pour valeur une série de commandes suivi de paramètres utilisés par ces commandes.

Chacune des commandes est instanciée par une lettre spécifique. Par exemple, pour se positionner aux coordonnées (10, 10), on utilise la commande `M` (pour *MoveTo*, « aller à ») suivi des coordonnées: "M 10 10". Quand l'interpréteur rencontre une lettre, il comprend que vous invoquez une commande, et les nombres qui suivent sont les paramètres de la commande.

De plus, toutes les commandes se présentent sous deux formes: une **lettre majuscule** spécifie des coordonnées absolues dans la page, une **lettre minuscule** spécifie des coordonnées relatives (par exemple, « aller à 10px vers le haut et 7px vers la gauche depuis le point précédent »).

Les coordonnées dans l'attribut `d` sont **toujours sans unité** et par conséquent dans le système de coordonnées utilisateur. Par la suite, nous apprendrons comment les chemins peuvent être transformés pour répondre à d'autres besoins.

Commandes pour les lignes

Il existe cinq commandes pour tracer des lignes avec un élément `<path>`. Ces commandes permettent de tracer une ligne droite entre deux points.

MoveTo

La première commande, « aller à », invoquée avec **M** (*MoveTo*), a été décrite ci-dessus. Elle prend en paramètres les coordonnées **x** et **y** où se rendre. Aucun trait n'est dessiné, le curseur est simplement déplacé dans la page. La commande « aller à » apparaît au début d'un chemin pour spécifier à quel endroit le dessin doit commencer. Par exemple :

```
1 | M x y
```

ou

```
1 | m dx dy
```

Dans l'exemple suivant, on se place au point (10, 10). Notez cependant qu'à ce stade rien n'est dessiné, on a manuellement ajouté un cercle pour indiquer la position:



```
1 | <svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 10" />
3 |
4 |   <!-- Indique la position -->
5 |   <circle cx="10" cy="10" r="2" fill="red" />
6 | </svg>
```

LineTo, Horizontal LineTo, Vertical LineTo

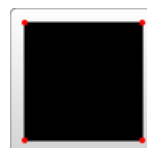
Il y a trois commandes qui dessinent des lignes. La plus générique est la commande « ligne vers », invoquée avec **L** (*LineTo*). **L** prend deux paramètres, les coordonnées **x** et **y**, et dessine une ligne depuis la position actuelle vers la nouvelle position.

```
1 | L x y (ou l dx dy)
```

Il existe deux formes abrégées pour dessiner des lignes horizontales ou verticales. **H** (*Horizontal LineTo*) dessine une ligne horizontale, et **V** (*Vertical LineTo*) dessine une ligne verticale. Ces deux commandes ne prennent qu'un seul argument car elles ne se déplacent que le long d'une direction.

```
1 | H x (ou h dx)
2 | V y (ou v dy)
```

Afin de commencer facilement, nous allons dessiner une forme simple, un rectangle (qu'on aurait aussi pu dessiner avec un élément `<rect>`). Il est composé uniquement de lignes horizontales et verticales :



```
1 | <svg width="100" height="100" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 10 H 90 V 90 H 10 L 10 10" />
```

```

3
4  <!-- Indique les points -->
5  <circle cx="10" cy="10" r="2" fill="red"/>
6  <circle cx="90" cy="90" r="2" fill="red"/>
7  <circle cx="90" cy="10" r="2" fill="red"/>
8  <circle cx="10" cy="90" r="2" fill="red"/>
9  </svg>

```

ClosePath

On aurait pu raccourcir un peu la déclaration de l'exemple ci-dessus en utilisant la commande « fermer le chemin », invoquée avec `Z` (*ClosePath*). Cette commande dessine une ligne droite entre la position actuelle et le premier point du chemin. Elle est souvent placée à la fin du `path`, mais pas toujours. Il n'y a pas de différence entre la commande en majuscule et en minuscule.

```

1 | Z (ou z)

```

Ainsi, notre chemin précédent peut se raccourcir comme ceci:

```

1 | <path d="M10 10 H 90 V 90 H 10 Z" fill="transparent" stroke="black"/>

```

Commandes relatives

On aurait également pu utiliser des commandes relatives pour dessiner la même image.

Les commandes relatives sont invoquées en utilisant des lettres minuscules. Plutôt que de déplacer le curseur vers des coordonnées absolues, elles le déplacent relativement à sa dernière position. Par exemple, puisque notre boîte est de dimension 80x80, l'élément `path` aurait pu être écrit:

```

1 | <path d="M10 10 h 80 v 80 h -80 Z" fill="transparent" stroke="black"/>

```

Le chemin va se positionner au point (10, 10), se déplacer horizontalement de 80 points vers la droite, puis de 80 points vers le bas, de 80 points vers la gauche, et enfin revenir à son point de départ.

Dans ces exemples, il serait probablement plus simple d'utiliser un élément `<polygon>` ou `<polyline>`. Cependant, les chemins sont si couramment utilisés en dessin SVG qu'un développeur peut se sentir plus à l'aise avec eux. Il n'y a pas de réel avantage ou inconvénient à utiliser l'un ou l'autre.

Commandes pour les courbes

Il existe trois commandes différentes pour créer des courbes. Deux d'entre elles sont des courbes de Bézier, et la troisième est un « arc » ou section de cercle. Il se peut que vous ayez déjà acquis une expérience pratique avec les courbes de Bézier en utilisant les outils de

chemins avec Inkscape, Illustrator ou Photoshop. Pour une description complète des concepts mathématiques sous-jacents, vous pouvez consulter la [page Wikipedia Courbe de Bézier](#).

Il existe une infinité de courbes de Bézier, mais seulement deux des plus simples d'entre elles sont disponibles dans les éléments `path`: l'une cubique, invoquée avec `C`, et l'autre quadratique, invoquée avec `Q`.

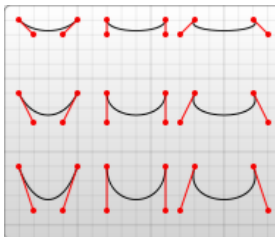
CurveTo

La courbe de Bézier cubique, `C` (*CurveTo*), est la forme de courbe Bézier la plus complexe. Ce type de courbe nécessite deux points de contrôle. Ainsi, pour créer une courbe de Bézier cubique, vous devez spécifier trois paires de coordonnées.

```
1 | C x1 y1, x2 y2, x y (or c dx1 dy1, dx2 dy2, dx dy)
```

Les deux premières paires de coordonnées sont les points de contrôle: le point de contrôle pour le début de la courbe est (x1, y1), et (x2, y2) est celui de la fin de la courbe. La dernière paire de coordonnées (x, y) est l'endroit où vous voulez que la ligne se termine.

Les points de contrôle décrivent, pour faire simple, la pente de la courbe pour le point de départ et pour le point d'arrivée. La fonction Bézier crée ensuite une courbe lisse faisant le lien entre la pente que vous avez établie au début de votre ligne, et celle à l'autre extrémité.



```
1 | <svg width="190" height="160" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 10 C 20 20, 40 20, 50 10" stroke="black" fill="transparent"/>
3 |   <path d="M70 10 C 70 20, 120 20, 120 10" stroke="black" fill="transparent"/>
4 |   <path d="M130 10 C 120 20, 180 20, 170 10" stroke="black" fill="transparent"/>
5 |   <path d="M10 60 C 20 80, 40 80, 50 60" stroke="black" fill="transparent"/>
6 |   <path d="M70 60 C 70 80, 110 80, 110 60" stroke="black" fill="transparent"/>
7 |   <path d="M130 60 C 120 80, 180 80, 170 60" stroke="black" fill="transparent"/>
8 |   <path d="M10 110 C 20 140, 40 140, 50 110" stroke="black" fill="transparent"/>
9 |   <path d="M70 110 C 70 140, 110 140, 110 110" stroke="black" fill="transparent"/>
10 |  <path d="M130 110 C 120 140, 180 140, 170 110" stroke="black" fill="transparent"/>
11 | </svg>
```

L'exemple ci-dessus crée neuf courbes de Bézier cubiques. De gauche à droite, les points de contrôle sont de plus en plus espacés horizontalement. De haut en bas, ils sont de plus en plus éloignés des extrémités. La chose à remarquer ici est que la courbe commence dans la direction du premier point de contrôle, puis se courbe de manière à terminer le long de la direction du second point de contrôle.

Shorthand CurveTo

Vous pouvez lier ensemble plusieurs courbes de Bézier pour créer des formes harmonieuses étendues. Souvent, le point de contrôle d'un côté d'une extrémité sera une réflexion du point de contrôle utilisé de l'autre côté, afin de garder une pente constante. Dans ce cas, vous pouvez

utiliser une version raccourcie de la courbe cubique, désignée par la commande **S**, ou **s** (*Shorthand CurveTo*).

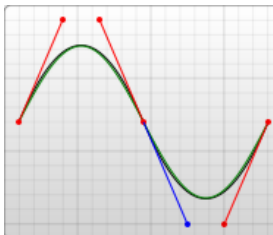
```
1 | S x2 y2, x y (ou s dx2 dy2, dx dy)
```

S dessine une courbe de Bézier cubique entre le point actuel et (x, y).

- Si elle suit une autre commande **S** ou **C**, le premier point de contrôle est calculé pour être le reflet du point de contrôle précédent.
- Si la commande **S** ne suit pas une autre commande **S** ou **C**, la position actuelle du curseur est utilisée comme premier point de contrôle. Dans ce cas, le résultat est le même que ce que la commande **Q** aurait produit avec les mêmes paramètres.

(x2, y2) est le second point de contrôle.

Un exemple de cette syntaxe est montré ci-dessous. Dans la figure associée, les points de contrôle spécifiés sont indiqués en rouge, et le point de contrôle inféré, en bleu.



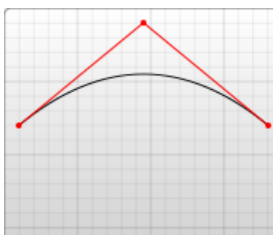
```
1 | <svg width="190" height="160" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 80 C 40 10, 65 10, 95 80 S 150 150, 180 80" stroke="black" fill="transparent"/>
3 | </svg>
```

Quadratic Bezier CurveTo

L'autre type de courbe, la courbe de Bézier quadratique, est invoquée avec **Q** (*Quadratic Bezier CurveTo*). Elle est plus simple que la version cubique puisqu'elle ne nécessite qu'un point de contrôle. Le point de contrôle détermine la pente de la courbe à la fois au point de départ et au point d'arrivée.

```
1 | Q x1 y1, x y (ou q dx1 dy1, dx dy)
```

(x1 y1) est la position du point de contrôle, et (x y) est le point d'arrivée de la courbe.



```
1 | <svg width="190" height="160" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 80 Q 95 10 180 80" stroke="black" fill="transparent"/>
3 | </svg>
```

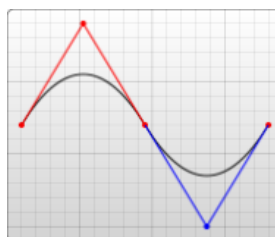
Shorthand Quadratic Bezier CurveTo

Comme pour la courbe cubique, il existe un raccourci pour lier ensemble plusieurs courbes quadratiques, invoqué avec **T** (*Shorthand Quadratic Bezier CurveTo*).

```
1 | T x y (ou t dx dy)
```

Ce raccourci examine le précédent point de contrôle utilisé et en infère un nouveau à partir de celui-ci. Cela signifie qu'après un premier point de contrôle, vous pouvez créer des formes assez complexes en spécifiant seulement les points d'extrémités.

Note: Ce raccourci fonctionne uniquement si la commande précédente est une commande **Q** ou **T**. Dans le cas contraire, le point de contrôle est considéré comme le même que le point précédent, et vous ne dessinerez que des lignes.



```
1 | <svg width="190" height="160" xmlns="http://www.w3.org/2000/svg">
2 |   <path d="M10 80 Q 52.5 10, 95 80 T 180 80" stroke="black" fill="transparent"/>
3 | </svg>
```

Les deux courbes produisent des résultats similaires, bien que les courbes cubiques vous offrent une plus grande liberté dans l'apparence exacte que vous voulez donner à votre courbe. Le choix du type de courbe de Bézier à utiliser se fait au cas par cas, et dépend du nombre de symétries que présente votre ligne.

Elliptical Arc

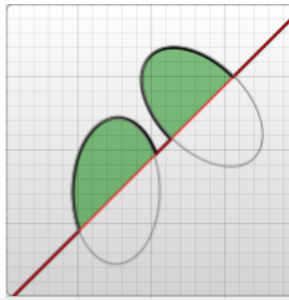
Le dernier type de ligne courbe que vous pouvez créer avec SVG est l'arc, invoqué avec **A** (*Elliptical Arc*). Les arcs sont des sections de cercles ou d'ellipses.

L'élément arc part du point actuel vers le point d'arrivée (x, y) en parcourant la ligne le long d'une ellipse définie par **rx** et **ry**. Le centre de l'ellipse (cx, cy) est calculé automatiquement pour satisfaire les contraintes imposées par les autres paramètres. Si vous avez besoin d'un rappel sur les ellipses, voyez les [formes de base](#). Ensemble, ces quatre valeurs définissent la structure de base de l'arc.

```
1 | A rx ry x-axis-rotation large-arc-flag sweep-flag x y
2 | a rx ry x-axis-rotation large-arc-flag sweep-flag dx dy
```

x-axis-rotation

x-axis-rotation décrit la rotation de l'arc. Il s'explique plus facilement avec un exemple:



```

1 <svg width="320" height="320" xmlns="http://www.w3.org/2000/svg">
2   <line x1="10" y1="315" x2="315" y2="10" stroke="black" stroke-width="2" />
3
4   <path d="M110 215      a 30 50  0 0 1 52.55 -52.45" fill="#7FBF7F" stroke="black" stroke-width="2" />
5   <path d="M172.55 152.45 a 30 50 -45 0 1 42.55 -42.55" fill="#7FBF7F" stroke="black" stroke-width="2" />
6 </svg>

```

Cet exemple montre deux arcs elliptiques de rayon `dx` 30 et rayon `dy` 50.

- Pour le premier arc, le paramètre `x-axis-rotation` a été laissé à 0, et l'ellipse autour de laquelle passe l'arc (montrée en gris) est orientée verticalement.
- Pour le second arc en revanche, `x-axis-rotation` est passé à -45 degrés. Cela pivote l'ellipse, de telle sorte que son petit axe (`dy`) est aligné avec la direction du chemin, comme illustré par la seconde ellipse dans l'image.

sweep-flag

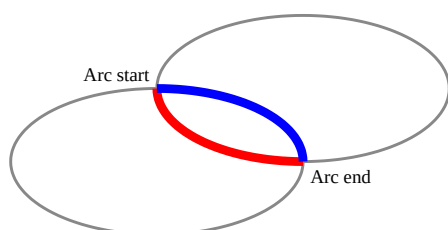
Pour un rayon `rx` et un rayon `ry` donnés, il existe deux ellipses pouvant connecter deux points quelconques.

`sweep-flag` détermine si l'arc doit commencer son mouvement à un angle négatif ou positif, permettant ainsi de choisir lequel des deux cercles est parcouru.

```

1 <!-- sweep-flag: 0 -->
2 <path d="M 125,75 a100,50 0 0,0 100,50"
3       stroke="red" stroke-width="6" fill="none" />
4
5 <!-- sweep-flag: 1 -->
6 <path d="M 125,75 a100,50 0 0,1 100,50"
7       stroke="blue" stroke-width="6" fill="none" />

```



large-arc-flag

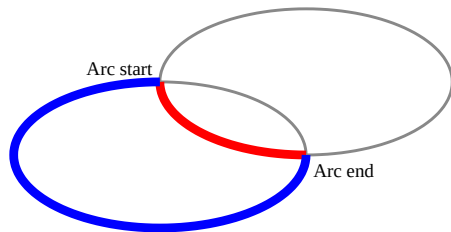
Pour chacune des deux ellipses, il existe deux chemins possibles, ce qui donne quatre chemins possibles.

`large-arc-flag` détermine simplement si l'arc doit être supérieur ou inférieur à 180 degrés ; au final, il détermine dans quelle direction l'arc va parcourir une ellipse donnée.

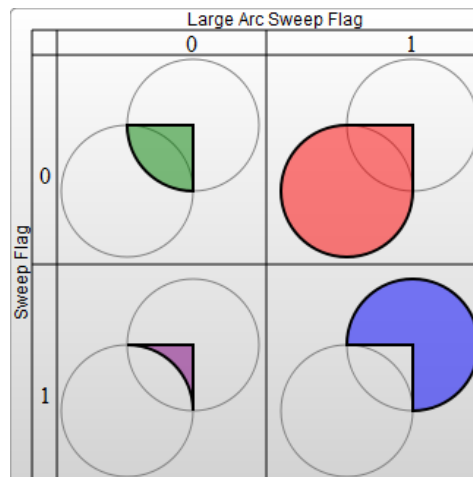
```

1 <!-- large-arc-flag: 0 -->
2 <path d="M 125,75 a100,50 0 0,0 100,50"
3     stroke="red" stroke-width="6" fill="none" />
4
5 <!-- large-arc-flag: 1 -->
6 <path d="M 125,75 a100,50 0 1,0 100,50"
7     stroke="blue" stroke-width="6" fill="none" />

```



L'exemple ci-dessous montre les quatre combinaisons possibles avec `sweep-flag` et `large-arc-flag`:



```

1 <svg width="325" height="325" xmlns="http://www.w3.org/2000/svg">
2   <path d="M80 80
3       A 45 45, 0, 0, 0, 125 125
4       L 125 80 Z" fill="green"/>
5   <path d="M230 80
6       A 45 45, 0, 1, 0, 275 125
7       L 275 80 Z" fill="red"/>
8   <path d="M80 230
9       A 45 45, 0, 0, 1, 125 275
10      L 125 230 Z" fill="purple"/>
11  <path d="M230 230
12      A 45 45, 0, 1, 1, 275 275
13      L 275 230 Z" fill="blue"/>
14 </svg>

```

Conclusion

Les arcs sont un moyen facile de créer des portions de cercle ou d'ellipse dans vos dessins. Par exemple pour dessiner un graphique en camembert. Si vous êtes en train de migrer vers SVG depuis [Canvas](#), les arcs peuvent être la partie la plus difficile à appréhender, mais sont également bien plus puissants.

Comme les points de départ et d'arrivée de tout chemin parcourant un cercle sont confondus, un nombre infini de cercles peuvent être choisis, par conséquent le chemin est indéfini. Il est possible d'en faire une approximation en prenant des points de départ et d'arrivée légèrement décalés, puis de les connecter à l'aide d'un autre segment de chemin. Dans ces conditions, il est souvent plus facile d'utiliser un véritable élément cercle ou ellipse à la place.

Vous pouvez trouver une démo interactive à l'adresse suivante, pour vous aider à comprendre les concepts derrière les arcs SVG: <http://codepen.io/lingtalfi/pen/yaLWJG> (testé avec Chrome et Firefox seulement, peut ne pas marcher avec votre navigateur).

[« Précédent](#)

[Suivant »](#)

🕒 **Dernière modification** : 10 sept. 2019, by [MDN contributors](#)

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

[Sign up now](#)



Polices SVG

Français ▼

[« Précédent](#)[Suivant »](#)

Lorsque SVG a été spécifié, le support des polices d'écriture pour le web n'était pas répandu dans les navigateurs. Comme l'accès au fichier de la police adéquate est cependant crucial pour afficher correctement le texte, une technologie de description des polices a été ajoutée à SVG pour offrir cette capacité. Elle n'a pas été conçue pour la compatibilité avec d'autres formats tels que le PostScript ou OTF, mais plutôt comme un moyen simple d'intégration des informations des glyphes en SVG lors de l'affichage.

Les Polices d'écritures SVG sont actuellement supportées uniquement sur Safari et le navigateur Android.

Internet Explorer [n'a pas envisagé de les implémenter](#), la fonctionnalité a été [supprimée de Chrome 38](#) (et Opera 25) et Firefox a [reporté sa mise en œuvre indéfiniment](#) pour se concentrer sur [WOFF](#). Cependant, d'autres outils comme le plugin [Adobe SVG Viewer](#), Batik et des modèles de document d'[Inkscape](#) supportent l'incorporation des Police d'écriture SVG.

La base pour définir une police SVG est l'élément ``.

Définir une police

Quelques ingrédients sont nécessaires pour intégrer une police en SVG. Prenons un exemple de déclaration (celle [de la spécification](#)), et expliquons-en les détails.

```
1 <font id="Font1" horiz-adv-x="1000">
2   <font-face font-family="Super Sans" font-weight="bold" font-style="normal"
3     units-per-em="1000" cap-height="600" x-height="400"
4     ascent="700" descent="300"
5     alphabetic="0" mathematical="350" ideographic="400" hanging="500">
6     <font-face-src>
7       <font-face-name name="Super Sans Bold"/>
8     </font-face-src>
9   </font-face>
10  <missing-glyph><path d="M0,0h200v200h-200z"/></missing-glyph>
11  <glyph unicode="!" horiz-adv-x="300"><!-- Outline of exclamation. pt. glyph --></glyph>
12  <glyph unicode="@"><!-- Outline of @ glyph --></glyph>
13  <!-- more glyphs -->
14 </font>
```

Nous commençons avec l'élément ``. Il contient un attribut `id`, ce qui permet de le référencer via une URI (voir plus bas). L'attribut `horiz-adv-x` définit sa largeur moyenne,

comparée aux définitions des autres glyphes individuelles. La valeur 1000 définit une valeur raisonnable. Plusieurs autres attributs associés précisent l'affichage de la boîte qui encapsule le glyphe.

L'élément `<font-face>` est l'équivalent SVG de la déclaration CSS `@font-face`. Il définit les propriétés de base de la police finale, telles que 'weight', 'style', etc. Dans l'exemple ci-dessus, la première et la plus importante est `font-family` : Elle pourra alors être référencée via la propriété `font-family` présente dans les CSS et les SVG. Les attributs `font-weight` et `font-style` ont la même fonction que leurs équivalents CSS. Les attributs suivants sont des instructions de rendu, pour le moteur d'affichage des polices ; par exemple : quelle est la taille des jambages supérieurs des glyphes ([↗ ascenders](#)).

Its child, the `<font-face-src>` element, corresponds to CSS' `src` descriptor in `@font-face` declarations. You can point to external sources for font declarations by means of its children `<font-face-name>` and `<font-face-uri>`. The above example states that if the renderer has a local font available named "Super Sans Bold", it should use this instead.

Following `<font-face-src>` is a `<missing-glyph>` element. This defines what should be displayed if a certain glyph is not found in the font and if there are no fallback mechanisms. It also shows how glyphs are created: By simply adding any graphical SVG content inside. You can use literally any other SVG elements in here, even `<filter>`, `<a>` or `<script>`. For simple glyphs, however, you can simply add a `d` attribute — this defines a shape for the glyph exactly like how standard SVG paths work.

The actual glyphs are then defined by `<glyph>` elements. The most important attribute is `unicode`. It defines the unicode codepoint represented by this glyph. If you also specify the `lang` attribute on a glyph, you can further restrict it to certain languages (represented by `xml:lang` on the target) exclusively. Again, you can use arbitrary SVG to define the glyph, which allows for great effects in supporting user agents.

There are two further elements that can be defined inside `font`: `<hkern>` and `<vkern>`. Each carries references to at least two characters (attributes `u1` and `u2`) and an attribute `k` that determines how much the distance between those characters should be decreased. The below example instructs user agents to place the "A" and "V" characters closer together the standard distance between characters.

```
1 | <hkern u1="A" u2="V" k="20" />
```

Référencer une police

Lorsque vous avez mis en place votre déclaration de police comme décrit ci-dessus, vous pouvez utiliser un simple attribut `font-family` pour réellement appliquer la police à un texte SVG:

```
1 | <font>
2 |   <font-face font-family="Super Sans" />
3 |   <!-- ... -->
4 | </font>
5 |
6 | <text font-family="Super Sans">My text uses Super Sans</text>
```

Cependant, vous êtes libre de combiner plusieurs méthodes pour une plus grande liberté de où et comment définir la police.

Option: Utiliser le CSS @font-face

Vous pouvez utiliser `@font - face` pour les polices externes de référence :

```
1 <font id="Super_Sans">
2   <!-- ... -->
3 </font>
4
5 <style type="text/css">
6 @font-face {
7   font-family: "Super Sans";
8   src: url(#Super_Sans);
9 }
10 </style>
11
12 <text font-family="Super Sans">My text uses Super Sans</text>
```

Option: Référencer une police externe

L'élément mentionné `font - face - uri` vous permet de référencer une police externe, permettant donc une plus grande réutilisabilité :

```
1 <font>
2   <font-face font-family="Super Sans">
3     <font-face-src>
4       <font-face-uri xlink:href="fonts.svg#Super_Sans" />
5     </font-face-src>
6   </font-face>
7 </font>
```

« Précédent

Suivant »

🕒 **Dernière modification :** 23 mars 2019, by [MDN contributors](#)

[Technologies ▼](#)[Guides et références ▼](#)[Votre avis ▼](#)

Remplissages et contours

[Français ▼](#)[« Précédent](#)[Suivant »](#)

Il y a différentes manières de colorer des formes: utiliser différents attributs SVG sur l'objet, utiliser du [CSS](#) en ligne, une section CSS ou un fichier CSS externe. La plupart des [SVG](#) que vous trouverez sur le Web utilisent du CSS en ligne, mais il y a des avantages et inconvénients pour chaque manière.

Attributs Fill et Stroke

Colorer

La coloration peut être faite en définissant deux attributs sur l'objet: `fill` et `stroke`. `Fill` définit la couleur de remplissage et `stroke` définit la couleur de la bordure. Vous pouvez utiliser la même convention de nommage des couleurs que CSS, que ce soit les noms (comme *red*), les valeurs rgb (comme *rgb(255,0,0)*), les valeurs hexadécimales, rgba, etc.

```
1 <rect x="10" y="10" width="100" height="100"
2     stroke="blue" fill="purple"
3     stroke-opacity="0.8" fill-opacity="0.5"/>
```

De plus, vous pouvez spécifier l'opacité de `fill` et/ou `stroke`. Celle-ci est contrôlé par les attributs `fill-opacity` et `stroke-opacity` respectivement.

Note: Dans Firefox 3+, les valeurs rgba sont autorisés, ce qui donne le même effet qu'utiliser les attributs d'opacité. En revanche, pour être compatible avec les autres navigateurs, il est souvent préférable de spécifier fill/stroke-opacity séparément. Si vous spécifiez à la fois une valeur rgba et fill/stroke-opacity, les deux seront appliquées.

Options du contour

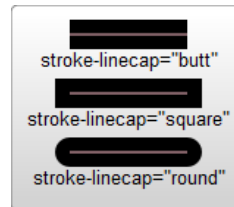
Outre les propriétés de couleur, il existe quelques attributs additionnels pour contrôler la manière dont le contour est dessiné.

stroke-width

La propriété `stroke-width` définit la taille du contour. La ligne du contour est centrée autour du remplissage (si le contour vaut 10, 5 pixels du contour chevauchent le remplissage).

stroke-linecap

Le second attribut affectant le contour est la propriété `stroke-linecap`. Elle contrôle la forme des fins de ligne. Dans l'image ci-dessous, le chemin est dessiné en rose et le contour en noir.



```
1 <svg width="160" height="140" xmlns="http://www.w3.org/2000/svg" version="1.1">
2   <line x1="40" x2="120" y1="20" y2="20"
3       stroke-linecap="butt" stroke="black" stroke-width="20" />
4   <line x1="40" x2="120" y1="60" y2="60"
5       stroke-linecap="square" stroke="black" stroke-width="20" />
6   <line x1="40" x2="120" y1="100" y2="100"
7       stroke-linecap="round" stroke="black" stroke-width="20" />
8 </svg>
```

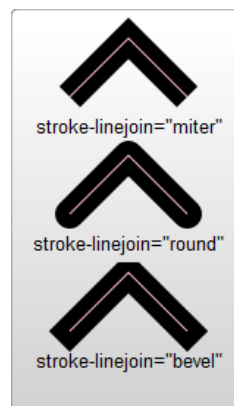


Il y a trois valeurs possibles pour `stroke-linecap` :

- `butt` (valeur par défaut) ferme la ligne avec un bord droit, à 90 degrés à l'endroit où la ligne se termine.
- `square` a la même apparence mais termine au delà de la ligne. La distance ajoutée est la moitié de `stroke-width`.
- `round` produit un effet arrondi à la fin du trait. Le rayon de cette courbe est également contrôlé par `stroke-width`.

stroke-linejoin

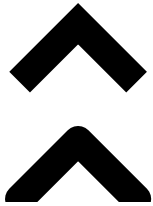
La propriété `stroke-linejoin` permet de contrôler la manière de dessiner la liaison entre deux segments de ligne.




```

1 <svg width="160" height="280" xmlns="http://www.w3.org/2000/svg" version="1.1">
2   <polyline points="40 60 80 20 120 60" stroke="black" stroke-width="20"
3     stroke-linecap="butt" fill="none" stroke-linejoin="miter"/>
4
5   <polyline points="40 140 80 100 120 140" stroke="black" stroke-width="20"
6     stroke-linecap="round" fill="none" stroke-linejoin="round"/>
7
8   <polyline points="40 220 80 180 120 220" stroke="black" stroke-width="20"
9     stroke-linecap="square" fill="none" stroke-linejoin="bevel"/>
10 </svg>

```

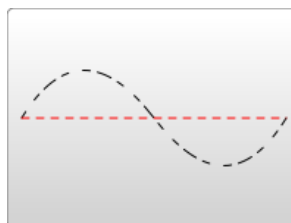


Chacune des ces polygones est composée de deux segments de lignes. La liaison entre les deux est contrôlée par l'attribut `stroke-linejoin`. Il y a trois valeurs possibles pour cet attribut:

- `miter` (valeur par défaut) prolonge légèrement la ligne au-delà de sa largeur normale pour créer un coin carré, de telle sorte qu'il n'y ait qu'un seul angle.
- `round` crée un coin arrondi.
- `bevel` crée un nouvel angle pour faciliter la transition entre les deux segments.

stroke-dasharray

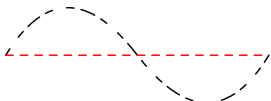
Finalement, vous pouvez également créer des lignes pointillées en spécifiant l'attribut `stroke-dasharray`.



```

1 <svg width="200" height="150" xmlns="http://www.w3.org/2000/svg" version="1.1">
2   <path d="M 10 75 Q 50 10 100 75 T 190 75" stroke="black"
3     stroke-linecap="round" stroke-dasharray="5,10,5" fill="none"/>
4   <path d="M 10 75 L 190 75" stroke="red"
5     stroke-linecap="round" stroke-width="1" stroke-dasharray="5,5" fill="none"/>
6 </svg>

```



L'attribut `stroke-dasharray` prend une série de nombres séparés par une virgule en argument.

Note: Contrairement aux `paths`, ces nombres **doivent** être séparés par des virgules (les espaces sont ignorés).

Le premier nombre spécifie la distance du trait et le second la distance de l'espace. Dans l'exemple précédent, la ligne rouge commence par un trait de 5 suivi d'un espace de 5 (`5, 5`), motif qui se répète sur le reste de la ligne. Vous pouvez spécifier davantage de nombres pour créer un motif de pointillés plus complexe. Pour la ligne noire on a spécifié trois nombres (`5, 10, 5`), ce qui a pour effet d'alterner le motif: (5 trait, 10 espace, 5 trait), (5 espace, 10 trait, 5 espace), etc.

Autres

Il existe d'autres propriétés disponibles:

- `fill-rule`, spécifie la règle de remplissage pour les formes où des chemins se chevauchent.
- `stroke-miterlimit`, détermine à partir de quel angle une liaison de segment de type `miter` sera affichée en `bevel`.
- `stroke-dashoffset`, définit à partir d'où commencer les pointillés sur la ligne.

Utiliser CSS

En plus de définir des attributs sur des objets, vous pouvez également utiliser CSS pour styliser les remplissages et les contours. Tous les attributs ne peuvent pas être définis via CSS. Ceux qui traitent le remplissage et le contour le sont généralement, `fill`, `stroke`, `stroke-dasharray`, etc... peuvent donc être définis de cette manière. Les attributs tels que `width`, `height`, ou les commandes des `paths`, ne peuvent pas être définis par CSS. Le plus simple est de tester pour découvrir ce qui est disponible et ce qui ne l'est pas.

Note: La [spécification SVG](#) décide strictement entre les attributs qui sont des *propriétés* et les autres. Les premiers peuvent être modifiés avec CSS, les derniers non.

En ligne

CSS peut être inséré en ligne avec l'élément via l'attribut `style`:

```
1 | <rect x="10" height="180" y="10" width="180" style="stroke: black; fill: red;"/>
```

Dans un section style

Sinon, il peut être déplacé vers une section `style`. Au lieu de l'insérer dans une section `<head>` comme vous le feriez en HTML, on la place dans la zone `<defs>` du SVG. `<defs>` (abréviation de *definitions*) est l'endroit où vous placez les éléments qui n'apparaissent pas dans le SVG directement, mais qui sont utilisés par les autres éléments.

```

1 <?xml version="1.0" standalone="no"?>
2 <svg width="200" height="200" xmlns="http://www.w3.org/2000/svg" version="1.1">
3   <defs>
4     <style type="text/css"><![CDATA[
5       #MyRect {
6         stroke: black;
7         fill: red;
8       }
9     ]]></style>
10  </defs>
11  <rect x="10" height="180" y="10" width="180" id="MyRect" />
12 </svg>

```

Déplacer les styles dans une zone comme ceci peut rendre les choses plus simples pour ajuster les propriétés d'un grand nombre d'éléments. Vous pouvez également utiliser les **pseudo-classes comme `hover`** pour créer des effets:

```

1 #MyRect:hover {
2   stroke: black;
3   fill: blue;
4 }

```

Dans un fichier externe

Ou vous pouvez spécifier une feuille de style externe pour vos règles CSS avec la [syntaxe XML pour les stylesheets](#):

```

1 <?xml version="1.0" standalone="no"?>
2 <?xml-stylesheet type="text/css" href="style.css"?>
3
4 <svg width="200" height="150" xmlns="http://www.w3.org/2000/svg" version="1.1">
5   <rect height="10" width="10" id="MyRect" />
6 </svg>

```

où `style.css` ressemble à ça:

```

1 #MyRect {
2   fill: red;
3   stroke: black;
4 }

```

[« Précédent](#)

[Suivant »](#)

[Technologies ▼](#)[Guides et références ▼](#)[Votre avis ▼](#)

Textes

[Français ▼](#)[« Précédent](#)[Suivant »](#)

Lorsqu'on parle de texte en SVG, on doit différencier deux choses pratiquement complètement séparées: 1. l'inclusion et l'affichage de texte dans une image, 2. les polices SVG. Un article séparé sera dédié aux polices SVG, celui-ci se concentrera uniquement sur le fait d'insérer du texte.

Les bases

Nous avons vu dans l'[exemple de l'introduction](#) que l'élément `text` peut être utilisé pour mettre du texte dans des documents SVG:

```
1 | <text x="10" y="10">Hello World!</text>
```

Les attributs `x` et `y` déterminent où le texte apparaîtra dans la fenêtre. L'attribut `text-anchor` spécifie l'alignement horizontal du texte (si ce point doit être le côté gauche, droit ou le centre du texte) et l'attribut `dominant-baseline` l'alignement vertical (si ce point est le haut, le bas ou le centre).

De même que les formes basiques, la couleur des éléments texte peut être modifié avec l'attribut `fill` pour le remplissage ou `stroke` pour le contour. Tout deux peuvent également faire référence à un dégradé ou motif, ce qui rend la coloration de texte SVG beaucoup plus puissante que CSS 2.1.

Définir la police

Une partie essentielle d'un texte est la police dans laquelle il est affiché. SVG offre un ensemble d'attributs pour spécifier la police, dont beaucoup sont similaires à leurs équivalents CSS. Chacune des propriétés suivantes peut être définie en tant qu'attribut ou via une déclaration CSS: `font-family`, `font-style`, `font-weight`, `font-variant`, `font-stretch`, `font-size`, `font-size-adjust`, `kerning`, `letter-spacing`, `word-spacing` et `text-decoration`.

Autres éléments liés au texte

tspan

Cet élément est utilisé pour baliser des sous-parties d'un texte. Il doit s'agir d'un enfant d'un élément `text` ou d'un autre élément `tspan`. Un cas typique consiste à écrire un mot d'une phrase en gras:

```
1 <text>
2   This is <tspan font-weight="bold" fill="red">bold and red</tspan>
3 </text>
```

This is **bold and red**

L'élément `tspan` peut prendre les attributs personnalisés suivants:

x

Définit une nouvelle coordonnées absolue pour le texte qu'il contient. Cela écrase la position par défaut du texte. Cet attribut peut également contenir une liste de nombres, qui sont appliqués un par un à chaque caractère du `tspan`.

dx

Définit un décalage horizontal relatif à la position par défaut du texte. Ici aussi, vous pouvez fournir une liste de valeurs qui seront appliquées consécutivement à chaque caractère.

y et **dy** sont utilisés de la même manière mais pour le déplacement vertical.

rotate

Applique une rotation aux caractères, avec le nombre de degrés donné. Donner une liste de nombres aura pour effet d'appliquer une rotation à chaque caractère respectif, la dernière valeur sera appliquée aux caractères restants.

textLength

Un attribut quelque peu obscur qui donne la longueur calculée de la chaîne. Il est destiné au moteur de rendu pour lui permettre d'affiner la position des glyphes, lorsque la longueur de texte mesurée ne correspond pas à celle qui est indiquée.

xref

L'élément `xref` permet de référencer un texte déjà défini, et recopier le texte à sa place. Vous devez utiliser l'attribut `xlink:href` pour définir l'élément à copier. Vous pouvez ensuite styliser le texte et modifier son apparence indépendamment de la source.

```
1 <text id="example">This is an example text.</text>
2
3 <text>
4   <xref xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#example" />
</text>
```

textPath

Cet élément récupère via son attribut `xlink:href` un chemin arbitraire et aligne ses caractères le long de ce chemin:

```
1 <path id="my_path" d="M 20,20 C 80,60 100,40 120,20" fill="transparent" />
2 <text>
3   <textPath xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="#my_path">
4     A curve.
5   </textPath>
6 </text>
```

[<< Précédent](#)[Suivant >>](#)

© Dernière modification : 18 mars 2019, by [MDN contributors](#)

✕

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

[Sign up now](#)

Technologies ▼

Guides et références ▼

Votre avis ▼

Transformations de base

Français ▼

[<< Précédent](#)[Suivant >>](#)

Maintenant, nous sommes prêts à tordre nos images dans tous les sens. Mais avant toute chose, il faut vous présenter l'élément `<g>`. Cet assistant va vous permettre d'assigner des attributs à un ensemble d'éléments. En fait, c'est bien son seul rôle. Par exemple :

```
1 <svg width="30" height="10">
2   <g fill="red">
3     <rect x="0" y="0" width="10" height="10" />
4     <rect x="20" y="0" width="10" height="10" />
5   </g>
6 </svg>
```

Toutes les transformations suivantes sont résumées dans l'attribut `transform` de l'élément. Les transformations peuvent être mises les unes à la suite des autres, tout simplement en les écrivant toutes dans cet attribut, séparées par des espaces.

Translation

Il peut être nécessaire de décaler un élément, même s'il est possible de définir sa position dans ses attributs. Pour ce faire, la fonction `translate()` est parfaite.

```
1 <svg width="40" height="50" style="background-color:#bff;">
2   <rect x="0" y="0" width="10" height="10" transform="translate(30,40)" />
3 </svg>
```

Cet exemple a pour résultat un rectangle, qui est déplacé du point (0,0) au point (30,40).



Si la deuxième valeur de `translate()` n'est pas définie, elle sera par défaut assignée à 0.

Rotation

Appliquer une rotation à un élément est assez simple : il suffit d'utiliser la fonction `rotate()`.

```
1 <svg width="31" height="31">
2   <rect x="12" y="-10" width="20" height="20" transform="rotate(45)" />
3 </svg>
```

Cet exemple montre un carré pivoté de 45°. La valeur de la rotation doit être définie en degrés.



Transformations multiples

Les transformations peuvent être concaténées, séparées par des espaces. Par exemple, `translate()` et `rotate()` sont couramment utilisées ensemble:

```
1 <svg width="40" height="50" style="background-color:#bff;">
2   <rect x="0" y="0" width="10" height="10" transform="translate(30,40) rotate(45)" />
3 </svg>
```



Cet exemple montre un carré déplacé et pivoté de 45 degrés.

Déformation

Pour transformer un rectangle en un losange, vous pouvez utiliser les fonctions `skewX()` et `skewY()`. Chacun prend pour attribut un angle qui détermine le biais de l'élément transformé.

Agrandissement et réduction

`scale()` modifie la taille d'un élément. Cette fonction prend en paramètre 2 valeurs de transformation, la première pour celle des X et la deuxième pour celle des Y. Ces valeurs sont écrites sous forme de ratio : 0.5 correspond à une réduction à 50%, 1.5 à une augmentation de 50%. Attention, c'est le système de chiffre anglo-saxon qui est ici utilisé, il faut donc déclarer un nombre réel en utilisant un point et non une virgule. *Si la deuxième valeur n'est pas déclarée, elle est considérée par défaut comme égale à la première.*

Transformations complexes avec matrice

Toutes les transformations détaillées ci-dessous peuvent être décrites dans une matrice de passage 3 par 3. Il est alors possible de combiner plusieurs transformations en appliquant directement la matrice de transformation `matrix(a, b, c, d, e, f)` qui mappe les coordonnées d'un système de coordonnées précédent en un nouveau système de coordonnées par

$$\begin{matrix} \left\{ \begin{matrix} x_{\mathrm{prevCoordSys}} \\ y_{\mathrm{prevCoordSys}} \end{matrix} \right\} = a \ x_{\mathrm{newCoordSys}} + c \\ y_{\mathrm{newCoordSys}} + e \\ y_{\mathrm{prevCoordSys}} = b \ x_{\mathrm{newCoordSys}} \\ + d \ y_{\mathrm{newCoordSys}} + f \end{matrix} \right. \end{matrix}$$

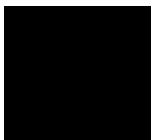
Voici un [exemple concret sur la documentation de transformation SVG](#). Pour plus de renseignements, veuillez vous référer à [la page de recommandation SVG](#).

Effets sur les systèmes de coordonnées

Quand vous utilisez une transformation, vous définissez un nouveau système de coordonnées dans l'élément que vous transformez. Cela signifie que vous appliquez la transformation à tous les attributs de l'élément transformé et donc que cet élément n'est plus dans une carte de pixel d'échelle 1:1. Cette carte est également déplacée, déformée, agrandie ou réduite selon la transformation qui lui est appliquée.

```
1 <svg width="100" height="100">
2   <g transform="scale(2)">
3     <rect width="50" height="50" />
4   </g>
5 </svg>
```

Cet exemple aura pour résultat un rectangle de 100 par 100 pixels. Les effets les plus étonnants apparaissent lorsque vous utilisez des attributs tels que `userSpaceOnUse`.



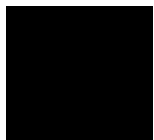
Embarquer du SVG dans SVG

Par opposition au HTML, le SVG peut embarquer d'autres éléments `svg` déclarés de manière tout à fait transparente. De cette façon, vous pouvez très simplement créer de nouveaux systèmes de coordonnées en utilisant `viewBox`, `width` et `height` de l'élément `svg`.

```
1 <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
2   <svg width="100" height="100" viewBox="0 0 50 50">
3     <rect width="50" height="50" />
4   </svg>
5 </svg>
```

```
3 | </svg>
4 | </svg>
5 |
```

Cet exemple a le même effet que celui vu précédemment, soit un rectangle deux fois plus grand que ce qu'il est défini.



[« Précédent](#)

[Suivant »](#)

🕒 **Dernière modification** : 23 mars 2019, [by MDN contributors](#)

x

Recevez le meilleur du développement web

Get the latest and greatest from MDN delivered straight to your inbox.

Cette lettre d'information est uniquement disponible en anglais pour l'instant.

[Sign up now](#)