Chapitre 4 : Courbes

En surfant, vous êtes sans doute tombé(e) sur quelque graphique SVG intéressant. Comme vous vous intéressez de près à ce langage, vous avez sans doute examiné le code et il est probable que vous avez remarqué la présence fréquente d'éléments <path>. Ceci indique souvent que le graphique a été généré par un programme plutôt que codé directement. Comme pour les pages HTML, il est plus pratique d'utiliser un logiciel pour créer des graphiques vectoriels. Le contenu peut être également généré par un script serveur ou traduit de documents XML par l'intermédiaire de XSLT. Les outils SVG connaissent l'importance des éléments <path> et optimisent leurs performances. Ceci dit, quand un programme exporte au format SVG, il utilise de préférence les éléments <path> plutôt que les formes de base. Comme nous voulons être capables de modifier leur code ou simplement de comprendre comment ils fonctionnent, il nous faut quelques lumières sur ces éléments. C'est ce que nous allons voir dans ce chapitre.

Qu'y a-t-il de spécial avec <path>?

1. L'élément <path> a un format très compact de données. C'est avantageux pour avoir un temps de chargement minimal.
2. L'élément <path> est très général. Il permet de dessiner lignes, cercles, rectangles, polygones et lignes brisées comme nous les avons vus au chapitre 2. Vous vous interrogez, pourquoi ne pas utiliser <path> à la place des formes de base?
3. L'élément <path> est très puissant mais aussi assez complexe. Il est plus intuitif et facile d'utiliser les formes de base quand c'est possible. D'un autre côté nous pouvons créer des courbes de Bézier avec <path> ou l'utiliser pour ajouter des trous aux formes, ce que nous ne pouvons faire avec les formes de base.

Je vous recommande d'utiliser les formes de base tant que la taille du document ne devient pas monstrueuse et que les formes répondent à votre besoin. Dans les autres cas, nous sommes obligés de nous pencher sur ces éléments.

Le concept de <path>

> *L'élément 'path' utilise le concept de point courant par analogie avec le tracé sur une feuille de papier, ce point courant étant la position de votre crayon.*

Voilà ce que les spécifications SVG nous disent sur ce concept.
Nous pouvons aussi faire une comparaison avec la tortue en langage Logo, les commandes d'un traceur ou le langage *PostScript*. Oui nous devons apprendre une syntaxe spéciale car le W3C a décidé de mettre toutes les informations géométriques dans un seul attribut. Ceci dans le but d'avoir une taille minimale pour les fichiers en vue de leur chargement. Mais le revers de la médaille est que les 'parsers' SVG non seulement doivent traiter un contenu XML, mais également les données complexes de l'attribut 'd' représentant la géométrie de l'élément. Passons à la syntaxe de l'élément <path>.

Syntaxe:

```
<path id="name"
      d="path-data"
```

```
            pathLength="length"
            marker="uri"
            marker-start="uri"
            marker-mid="uri"
            marker-end="uri"
            transform="transformation"
            fill-rule="nonzero | evenodd | inherit"
            style-attribute="style-value"
    />
```

L'attribut le plus important est d="*path-data*". Avec lui, nous définissons le contour de notre forme ou de notre courbe.
L'attribut '*pathLength*' ne change pas la longueur réelle de l'élément, il nous aide simplement à symboliser la longueur par une valeur, qui nous permet de repérer des points du tracé plus simplement. Nous pouvons dire: "Peu importe la longueur réelle, je lui donne la valeur 100. Et maintenant je peux retrouver la localisation de points à *0, 25, 50, 75* ou *100*." C'est une manière confortable de paramétrer la courbe que nous pouvons utiliser pour placer un texte sur la courbe ou créer une animation le long de cette courbe.

Tracer avec des lignes

Comment définir le tracé de notre élément? Quelques éléments à propos de l'attribut 'd':

1) Les données forment une série de commandes.
2) Une commande commence avec une lettre minuscule ou majuscule – la lettre de commande, accompagnée d'un certain nombre de valeurs numériques – les paramètres de commande.
3) Lettres de commande et paramètres sont séparés par des espaces ou des virgules.
4) Chaque commande s'applique au point courant, position du crayon imaginaire ou de la tortue Logo et le déplace à une nouvelle position suivant une trajectoire indiquée par les paramètres.
5) Le déplacement peut créer un tracé (crayon baissé) ou non (crayon levé).

L'objectif du groupe de travail SVG du W3C en utilisant cette syntaxe était de:

- Faciliter la programmation de ces éléments.
- Minimiser la taille des fichiers.

Voici la syntaxe des commandes usuelles.

Syntaxe: moveto
(se déplacer sans tracer et ouvrir un nouveau tracé)

```
M|m x, y
    x = x du nouveau point
    y = y du nouveau point
```

Syntaxe: lineto
(se déplacer avec un tracé rectiligne depuis le point courant)

```
L|l x, y
    x = x du nouveau point
    y = y du nouveau point
```

Syntaxe: closepath
(rejoindre le point de départ du tracé courant avec un tracé rectiligne)

```
Z|z
```

Voici un exemple avec la figure 4-1, nous avons un “U” avec un toit triangulaire, ceci étant tracé avec un seul élément <path>.

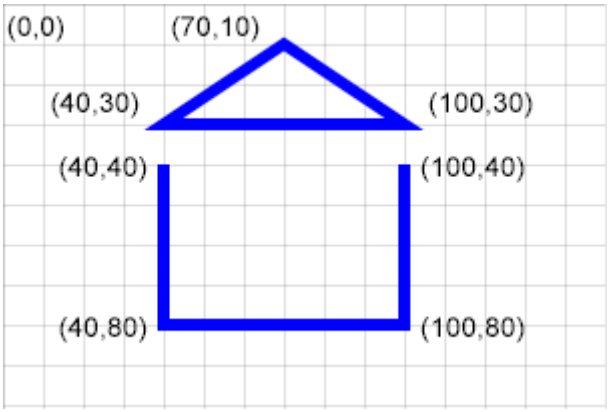


Figure 4-1. Path consisting of 6 lines

Voici le code correspondant.

```
<path stroke="blue" stroke-width="3" fill="none"
      d="M 40,40
         L 40,80
         L 100,80
         L 100,40
         M 40,30
         L 70,10
         L 100,30
         Z" />
```

Détaillons ces commandes.

Commande	Résultat
M 40,40	Le crayon se place en *(40,40)* et ouvre un nouveau tracé
L 40,80	Dessine un segment de droite depuis le point courant *(40,40)* jusqu'au nouveau point*(40,80)*
L 100,80	Dessine un segment de droite depuis le point courant *(40,80)* jusqu'au nouveau point *(100,80)*
L 100,40	Dessine un segment de droite depuis le point courant *(100,80)* jusqu'au nouveau point *(100,40)*
M 40,30	Déplace le crayon sans tracer au point (40,30) et ouvre un nouveau tracé
L 70,10	Dessine un segment de droite depuis le point courant *(40,30)* jusqu'au nouveau point *(70,10)*
L 100,30	Dessine un segment de droite depuis le point courant *(70,10)* jusqu'au nouveau point *(100,30)*
Z	Ferme la ligne en dessinant un segment de droite depuis le point courant*(100,30)* jusqu'au point de départ du tracé courant *(40,30)*

Table 4-1. Commandes et paramètres pour notre maison

Nous créons ce dessin avec les trois commandes que nous avons vues: *moveto*, *lineto* et *closepath*. Quelques remarques élémentaires sur ce tracé:

1) Chaque élément <path> doit commencer avec une commande *moveto*.
2) Dans cet exemple, nous utilisons des lettres majuscules pour indiquer que nous utilisons des coordonnées absolues, définies dans le système de l'élément. Nous verrons plus loin les coordonnées relatives à la position du point courant avec les commandes en lettres minuscules.
3) Nous séparons les valeurs de x et de y par une virgule et utilisons l'espace pour délimiter les autres paramètres. Ce n'est qu'une convention que nous discuterons un peu plus loin.

L'exemple de code de la figure 4-1 est écrit en mettant chaque commande sur une ligne, ce n'est que pour une meilleure lisibilité du code. Mais il est habituel de mettre le code sur une seule ligne comme ceci:

```
<path stroke="blue" stroke-width="3" fill="none"
    d="M 40,40 L 40,80 L 100,80 L 100,40 M 40,30 L 70,10 L 100,30 Z" />
```

Le concept de l'élément <path> étant de diminuer la taille du code, ne nous étonnons pas de l'existence de raccourcis pour les lignes horizontales et verticales.

Syntaxe: horizontal lineto
(déplacement horizontal avec tracé rectiligne)

H|h x = *x du nouveau point*

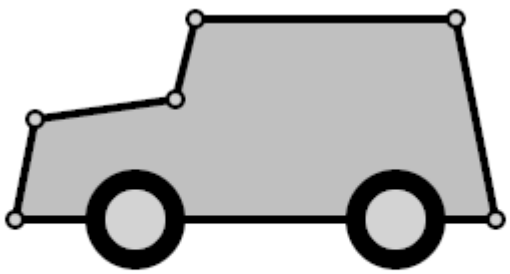
Syntaxe: vertical lineto
(déplacement vertical avec tracé rectiligne)

V|v y = *y du nouveau point*

Avec *horizontal lineto* la valeur de y pour le nouveau point n'a pas besoin d'être précisée, elle est celle du point courant. De même nous ne spécifions pas de valeur x pour *vertical lineto*. Ceci dit, nous pouvons récrire noter exemple en utilisant ces commandes pour obtenir un code plus compact.

```
<path stroke="blue" stroke-width="3" fill="none"
      d="M 40,40 V 80 H 100 V 40 M 40,30 L 70,10 L 100,30 Z" />
```

Utilisons ces commandes pour dessiner une voiture. Cette voiture sera améliorée au fur et à mesure de la découverte des autres commandes.



d="M 50,100 L 55,75 L 90,70 L 95,50 L 160,50 L 170,100 Z"

Figure 4-2. Voiture avec des segments de droite

Pour faire évoluer cette voiture, voici le code complet de ce document SVG.

```
<?xml version="1.0" ?>
<svg width="600" height="400" viewBox="0 0 300 200">
  <defs>
    <circle id="sommet" r="2" stroke="black" fill="lightgray" />
    <circle id="roue" r="10" stroke="black" stroke-width="5"
            fill="lightgray" />
  </defs>
  <path fill="silver" stroke="black" stroke-width="2"
        d="M 50,100 L 55,75 L 90,70 L 95,50 L 160,50 L 170,100 Z" />
  <use xlink:href="#sommet" x="50" y="100" />
  <use xlink:href="#sommet" x="55" y="75" />
  <use xlink:href="#sommet" x="90" y="70" />
  <use xlink:href="#sommet" x="95" y="50" />
  <use xlink:href="#sommet" x="160" y="50" />
  <use xlink:href="#sommet" x="170" y="100" />
  <use xlink:href="#roue"  x="80" y="100" />
  <use xlink:href="#roue"  x="145" y="100" />
  <text x="20" y="140" font-size="7">
    d="M 50,100 L 55,75 L 90,70 L 95,50 L 160,50 L 170,100 Z"
  </text>
</svg>
```

Dessiner des arcs

Vous devez vous étonner de ne pas trouver les arcs de cercles ou d'ellipses parmi les formes de base. En fait ce fut un sujet de discussion pour les spécifications. Peut-être aurons nous un élément <arc> dans une prochaine version. Pour le moment, prenons nous par la main pour dessiner ces arcs avec l'élément <path>.

Commençons par l'arc d'ellipse avec les commandes ***A*** (absolue) et ***a*** (relatif).

Syntaxe: elliptical arc
(trace un arc d'ellipse du point courant au nouveau point)

A|**a** rx, ry, x-axis-rotation, fA, fS, x, y
rx = demi-axe *x .. nombre positif*
ry = demi-axe *y .. nombre positif*
x-axis-rotation = angle de l'axe x avec l'horizontale
fA = *large-arc-flag .. 0, 1*
fS = *sweep-flag .. 0, 1*
x = *x du nouveau point*
y = *y du nouveau point*

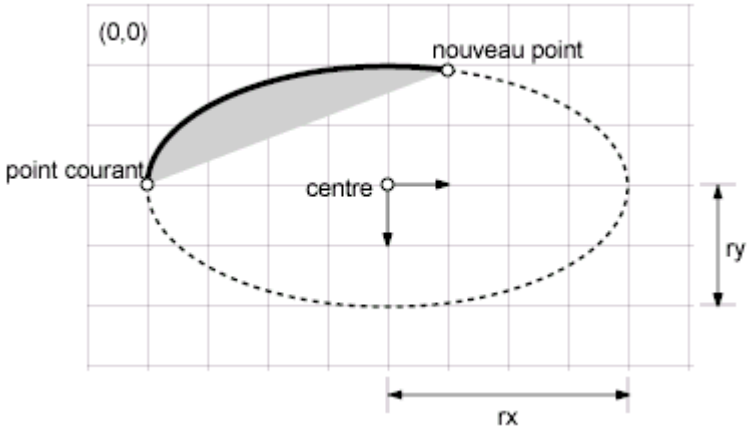


Figure 4-3. Arc d'ellipse

La figure 4-3 illustre les paramètres de la commande A. L'arc est une partie d'une ellipse, il débute au point courant et se termine au nouveau point dont les coordonnées sont les deux derniers paramètres. Les demi-axes de l'ellipse sont déterminés par les deux premiers paramètres.

Le troisième paramètre *x-axis*-rotation définit l'angle que fait l'axe en x de l'ellipse avec l'horizontale. Quand ce paramètre n'est pas nul, nous avons une transformation (rotation et translation) de l'ellipse pour qu'elle continue de passer par le point courant et le nouveau point (Figure 4-4).

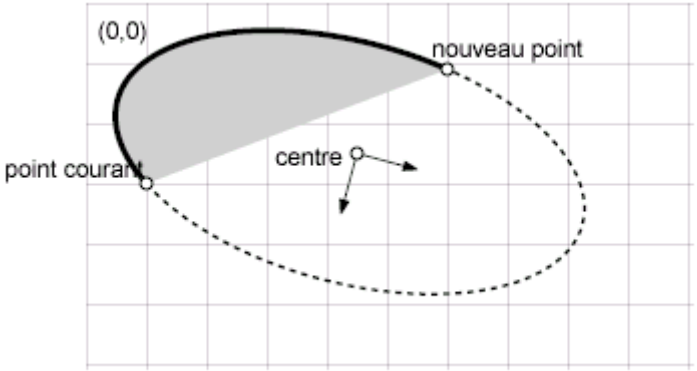


Figure 4-4. Avec x-axis-rotation non nul

Si les demi-axes de l'ellipse sont trop petits pour que l'ellipse puisse rejoindre les deux points; les demi-axes sont multipliés par un même coefficient pour que ce soit possible.

Figure 4-5. Rotation et ajustement pour ces ellipses

Les ellipses des figures 4-3 à 4-5 sont dessinées complètement, comment obtenir pour notre dessin les parties pointillées? Nous avons les quatrième et cinquième paramètres *large-arc-flag* et *sweep-flag*. En fait, connaissant les autres paramètres, nous n'avons pas une mais deux ellipses si les demi-axes sont suffisamment grands. La figure 4-6 nous le montre.

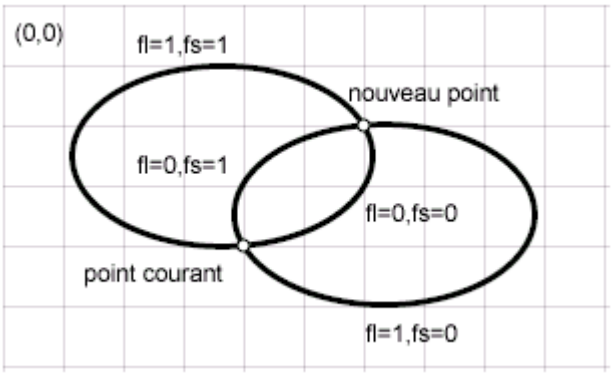


Figure 4-6. Quatre arcs possibles

Nous avons deux ellipses donc quatre arcs d'ellipse différents, deux petits et deux grands, deux orientés dans le sens positif (ici sens des aiguilles d'une montre contrairement au sens direct en mathématique) et deux dans le sens négatif. Voilà pourquoi nous avons deux paramètres supplémentaires pour choisir l'arc voulu. Le paramètre ***large-arc-flag*** signifie grand ou petit: fl = 0 pour petit, fl = 1 pour grand). Le paramètre ***sweep-flag*** définit l'orientation positive ou négative: fs = 0 pour positive, fs = 1 pour négative.

Signalons que si les points courants et nouveau coïncident, l'arc n'est pas rendu même avec *large-arc-flag* égal à 1, ceci devrait changer avec SVG 2.

Mettons ceci en pratique pour dessiner une spirale.

Figure 4-7. Spirale composée d'arcs d'ellipse

```
<svg width="600" height="400" viewBox="0 0 400 300">
  <path stroke="darkslategray" stroke-width="6" fill="none"
        stroke-linecap="round"
        d="M50,100
           A100,50 0 0 1 250,100
           A80,40  0 0 1  90,100
           A60,30  0 0 1 210,100
           A40,20  0 0 1 130,100
           A20,10  0 0 1 170,100" />
</svg>
```

Améliorons notre automobile en utilisant les arcs d'ellipse.

d="M 50,100 A 40,35 0 0 1 90,65 A 10,20 0 0 0 100 50 A 55 50 0 0 1 170 100 Z"

Figure 4-8. Automobile et arcs d'ellipse

Vous utiliserez rarement cet arc d'ellipse dans vos codages. Vous préférerez le support d'outils de dessin. Mais nous ne pouvons pas négliger le cas particulier des arcs de cercle.

Arc de cercle

L'arc de cercle n'est qu'un cas particulier de l'arc d'ellipse où les deux demi-axes sont égaux pour donner le rayon du cercle. Essayons de coder ces arcs de cercle très fréquents dans le dessin technique.

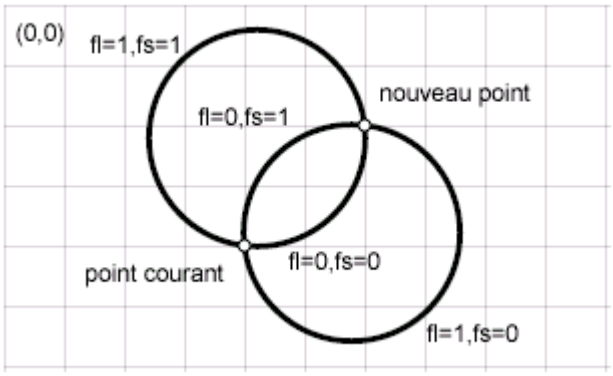


Figure 4-9. Quatre arcs de cercle possibles

Syntaxe:

A|**a** r, r, 0, fA, fS, x, y
r = *rayon .. nombre positif*
r = *rayon .. nombre positif*
0 = x-axis-rotation *angle .. toujours 0*
fA = *large-arc-flag .. 0, 1*
fS = *sweep-flag .. 0, 1*
x = *x du nouveau point*
y = *y du nouveau point*

Ainsi les deux premiers paramètres sont toujours égaux au rayon du cercle. D'autre part vous ne modifiez pas le cercle en donnant une valeur à 'x-axis-rotation', vous pouvez donc lui donner toujours la valeur 0.

La figure 4-9 nous montre les quatre arcs possibles.

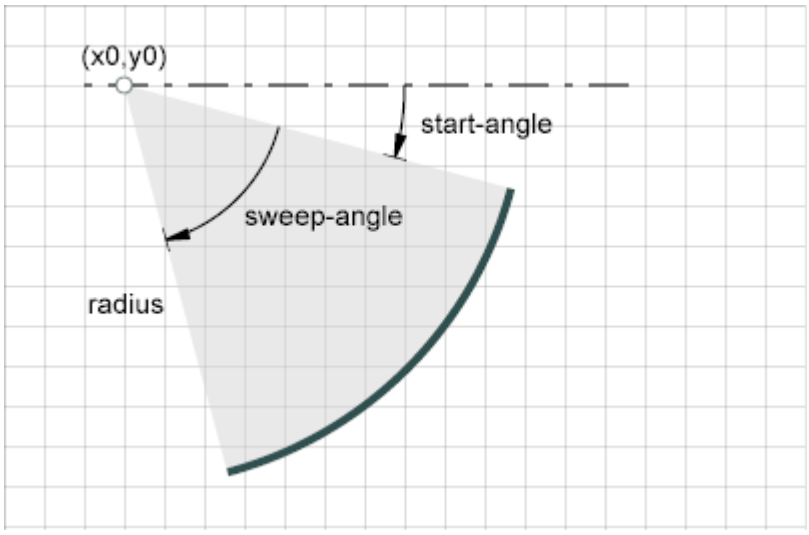


Figure 4-10. Arc de cercle défini par son centre et ses angles

Utilisant son concept de tracé au crayon, SVG décrit les arcs d'une façon peu orthodoxe pour un mathématicien.

Une méthode plus intuitive est de définir l'arc de cercle par le centre et le rayon du cercle, l'angle de départ et l'angle de l'arc comme dans la figure 4-10. Imaginons cet élément:

```
<circularArc cx="x0" cy="y0" r="radius" start="angle" sweep="angle" />
```

Espérons que SVG 2.0 nous apportera un tel élément.

Mais pour le moment, comment émuler cet élément avec les commandes 'A' ou 'a'.
Voici une solution possible:

```
<!-- arc -->
 <path transform="translate(x0,y0) rotate(start-angle) scale(radius)"
       d="M 1,0 A 1,1 0 fA fS cos(sweep-angle) sin(sweep-angle)"
       stroke-width="width/radius" />
```

Comme nous ne pouvons éviter tous les calculs, nous avons ici un minimum de mathématiques.
Voici les étapes de cette émulation:

1. Utiliser les coordonnées du centre comme vecteur de la translation.
2. L'angle de départ '*start-angle'* – en degrés – donne l'angle de rotation.
3. Le rayon nous donne le rapport de l'homothétie.
4. Le paramètre '*large-arc-flag fA'* vaut *0* si l'angle de l'arc '*sweep-angle'* est dans l'intervalle *–180 < sweep-angle < 180*, pour les autres valeurs, il vaut 1.
5. Le paramètre '*sweep-flag fS'* vaut 0 si '*sweep-angle'* est positif, sinon 1.
6. Les deux derniers paramètres sont les coordonnées calculées du point d'arrivée de l'arc sans tenir compte de l'angle de départ.
7. Comme '*stroke-width'* dépend du rapport de l'homothétie, nous devons compenser en divisant la largeur de tracé par le rayon.

Nous pouvons avec cette méthode, non seulement créer des arcs mais également des sections d'arcs ou des secteurs circulaires - portions de camembert - pour les représentations statistiques.

```
<!-- section d'arc -->
<path transform="translate(x0,y0) rotate(start-angle) scale(radius)"
      d="M 1,0 A 1,1 0 fA fS cos(sweep-angle) sin(sweep-angle) Z"
      stroke-width="width/radius" />

<!-- secteur circulaire -->
<path transform="translate(x0,y0) rotate(start-angle) scale(radius)"
      d="M 0,0 L 1,0 A 1,1 0 fA fS cos(sweep-angle) sin(sweep-angle) Z"
      stroke-width="width/radius" />
```

Voici un exemple:

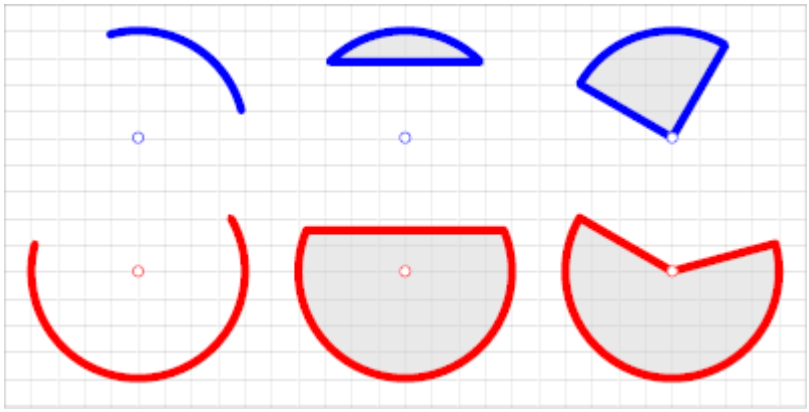
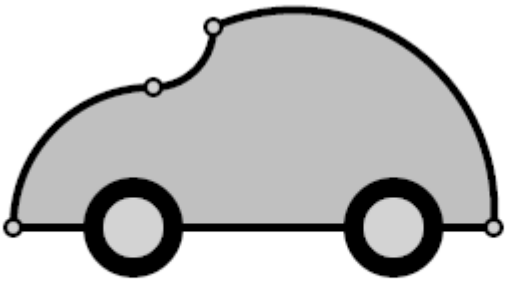


Figure 4-11. Secteurs d'arcs et secteurs circulaires

```
<svg>
  <path transform="translate(50,50) rotate(-15) scale(40)"
        d="M1,0 A1,1 0 0 0 0,-1"
        stroke-width="0.08" stroke="blue" fill="none" stroke-linecap="round" />
  <path transform="translate(50,100) rotate(-30) scale(40)"
        d="M1,0 A1,1 0 1 1 -0.707 -0.707"
        stroke-width="0.08" stroke="red" fill="none" stroke-linecap="round" />
  <path transform="translate(150,50) rotate(-45) scale(40)"
        d="M1,0 A1,1 0 0 0 0,-1 Z"
        stroke="blue" stroke-width="0.08" fill="silver" fill-opacity="0.5" />
  <path transform="translate(150,100) rotate(-22.5) scale(40)"
        d="M1,0 A1,1 0 1 1 -0.707 -0.707 Z"
        stroke="red" stroke-width="0.08" fill="silver" fill-opacity="0.5" />
  <path transform="translate(250,50) rotate(-60) scale(40)"
        d="M0,0 L1,0 A1,1 0 0 0 0,-1 Z"
        stroke="blue" stroke-width="0.08" fill="silver" fill-opacity="0.5" />
  <path transform="translate(250,100) rotate(-15) scale(40)"
        d="M0,0 L1,0 A1,1 0 1 1 -0.707 -0.707 Z"
        stroke="red" stroke-width="0.08" fill="silver" fill-opacity="0.5" />
</svg>
```

Nous ne pouvons terminer ce paragraphe sans revenir à notre automobile.



d="M 50,100 A 35,35 0 0 1 85,65 A 15,15 0 0 0 100,50 A 50 50 0 0 1 170 100 Z"

Figure 4-12. Automobile et arcs de cercle

Courbes

Approximation ou interpolation

Une courbe peut être définie par un ensemble de points de contrôle. Les segments joignant ces points constituent le polygone de contrôle.

Les méthodes pour définir la courbe sont variées, nous pouvons les diviser en méthodes d'interpolation où la courbe passe par tous les points de contrôle et méthodes d'approximation où la courbe est proche de ces points sans nécessairement passer par eux.

Le groupe de travail SVG a décidé pour la version 1.0 de se limiter à un seul type de courbe – les courbes de **Bézier**. Nous pouvons peut-être espérer une prise en compte plus générale dans les futures versions de SVG.

Courbes de Bézier

L'ingénieur Pierre Bézier travaillait pour Renault. Vers 1960 il eut à trouver une courbe qui soit facile à calculer mais suffisamment proche de la réalité de l'automobile. Il le fit avec l'aide des polynomes cubiques.

Les courbes de Bézier, dans le plan, sont à la base des programmes de dessin vectoriel (comme Adobe Illustrator ou Corel Draw), et un standard pour le language PostScript. La plupart des fontes de caractères, y compris les polices TrueType sont décrites comme des courbes de Bézier. Pour créer une courbe de Bézier, nous avons besoin d'au moins trois points. Pour être exact, deux peuvent suffire mais nous obtenons un simple segment de droite.

En théorie, ces courbes peuvent être définies pour un nombre quelconque de points. Mais pour la pratique, les courbes de Bézier quadratiques (trois points de contrôle) et cubiques (quatre points de contrôle) suffisent. Si nous avons plus de points, la courbe est décomposée en courbes quadratiques ou cubiques. Nous nommerons "**polybézier**" une telle courbe.

Courbe de Bézier quadratique

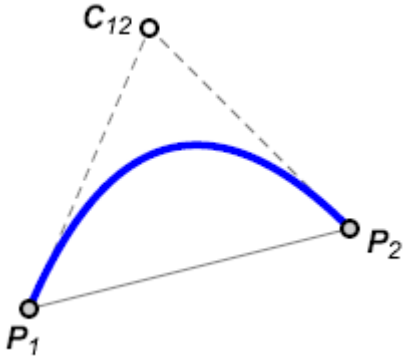


Figure 4-13. Courbe de Bézier quadratique

Pour mieux comprendre ce qu'est une courbe de Bézier quadratique, nous partons d'un segment de droite P_1 P_2 et ajoutons un troisième point C_{12} en dehors de ce segment. La courbe de Bézier passe par les points de contrôle P_1 et P_2, et approche seulement le point C_{12}. Imaginons que C_{12} exerce une certaine attraction sur le segment et le déforme en une courbe régulière.

Voici une approche possible pour définir la courbe obtenue.
Nous partons du triangle $P_1C_{12}P_2$.

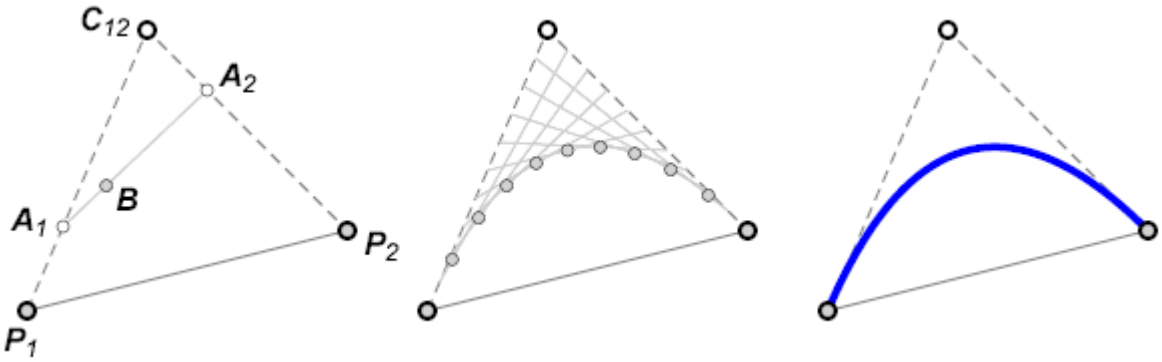


Figure 4-14. Construction de la quadratique de Bézier

Nous plaçons un point A_1 à *30%* du segment P_1 C_{12} ainsi que A_2 à *30%* de C_{12} P_2. Nous joignons ces points et plaçons B à *30%* du segment A_1 A_2 obtenu. Ce point B appartient à la courbe de Bézier.

En répétant cette construction en changeant le rapport de *0%* à *100%* avec un pas de *10%*, nous obtenons la figure centrale. Tous les segments construits sont des tangentes à la courbe – ils constituent l'enveloppe convexe de la courbe – et tous ces points B appartiennent à la courbe de Bézier qui n'est dans ce cas qu'une simple parabole. Voici ses caractéristiques:

- La tangente à la courbe en P_1 est dirigée vers C_{12}.
- La tangente en P_2 est également dirigée vers C_{12}.
- La courbe est contenue dans triangle $P_1C_{12}P_2$.
- Si les points P_1,C_{12},P_2 sont colinéaires (alignés), la courbe est un segment de droite.
- La courbe est convexe.

Utilisons maintenant cette courbe en SVG en la plaçant dans un élément <path>.

Nous avons donc de nouvelles commandes *quadratic Bézier curveto* **Q** et **q**.

Syntaxe:

```
Q|q x1, y1, x, y
    x1 = x du point de contrôle
    y1 = y du point de contrôle
    x = x du nouveau point
    y = y du nouveau point
```

La commande **Q** nécessite comme paramètres les coordonnées de deux points. La première paire de coordonnées définit notre point de contrôle C_{12} et la seconde le point de fin de tracé P_2 . P_1 est défini par le point courant de notre tracé. Utilisons cette commande pour dessiner un vase.

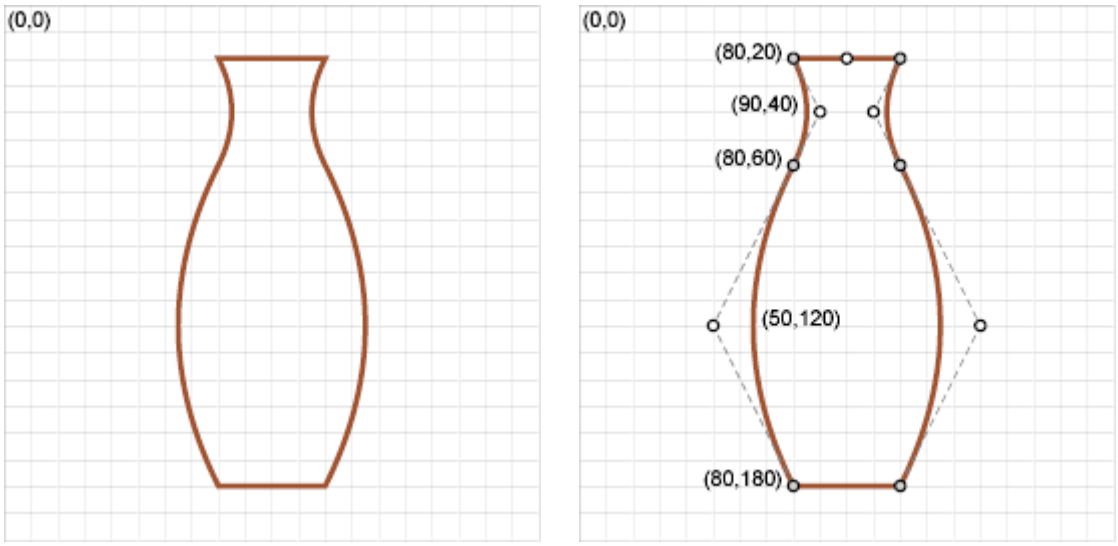


Figure 4-15. Vase fait de courbes de Bézier quadratiques

```
<path stroke="sienna" stroke-width="2" fill="none"
      d="M  80,180
         Q  50,120  80,60
         Q  90, 40  80,20
         Q 100, 20 120,20
         Q 110, 40 120,60
         Q 150,120 120,180
         Z" />
```

Nous démarrons notre tracé au point *(80,180)* avec la commande ***M*** . Ensuite nous traçons la courbe de Bézier contrôlée par le point *(50,120)* et se terminant en *(80,60)*. Nous continuons avec une autre commande ***Q*** avec le point de contrôle *(90,40)* et se terminant en *(80,20)*.
La commande suivante pourrait être remplacée par L120,20 pour tracer le haut du vase.

Notez que si nous voulons préserver la fluidité du tracé en passant d'une courbe de Bézier à la suivante, les tangentes doivent être les mêmes et les points de contrôle alignés.
La figure 4-16 illustre bien notre propos, seule la figure de gauche garde sa dérivabilité.

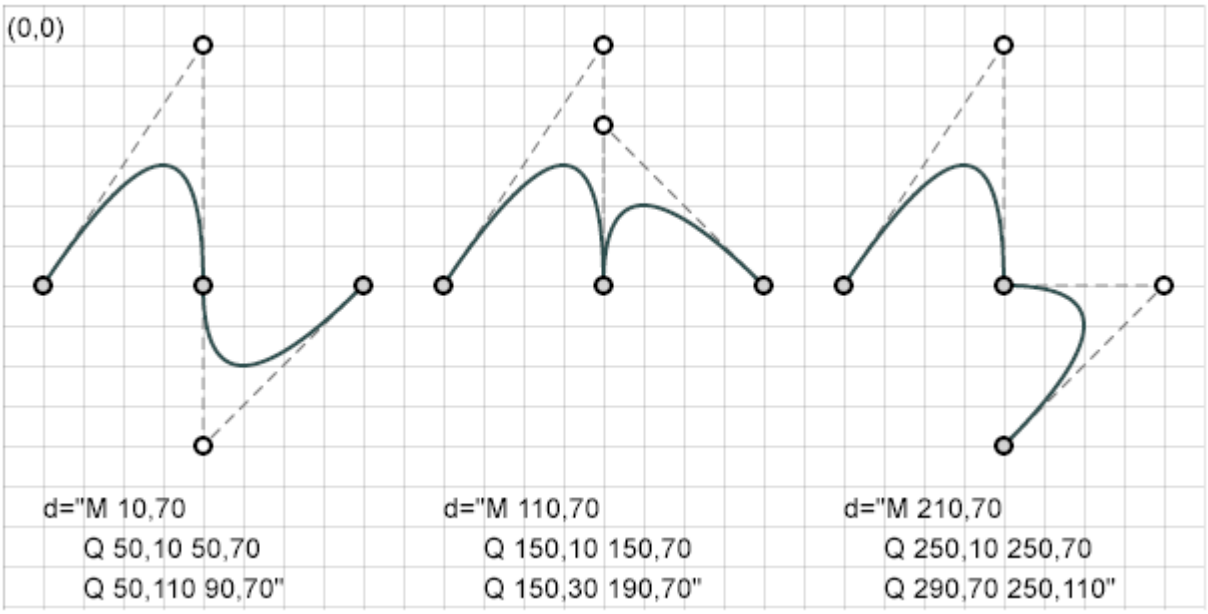


Figure 4-16. Tangentes aux points de raccordement

Comme nous avons souvent besoin de tels tracés continus, SVG propose une commande spécifique *smooth quadratic bézier curveto* ***T*** ou ***t***.

Syntaxe:

T|**t** x, y
x = *x du nouveau point*
y = *y du nouveau point*

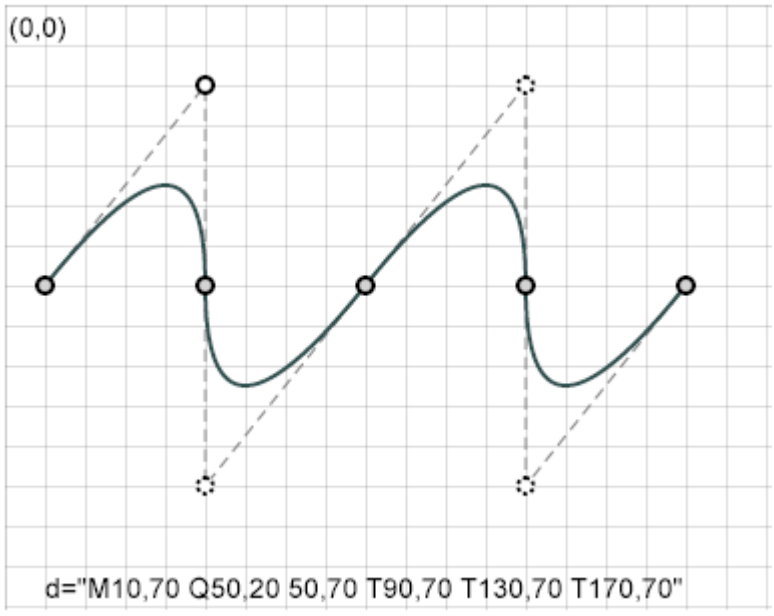
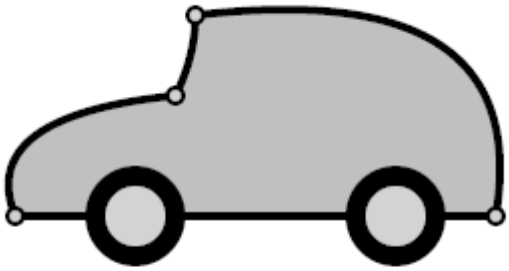


Figure 4-17. Points de contrôle automatiques

Cette commande **T** demande simplement les coordonnées du nouveau point, fin de tracé de la courbe de Bézier. Le point de contrôle pour ce tracé est créé automatiquement, il est le symétrique du précédent point de contrôle par rapport au point courant.
Ceci suppose évidemment qu'une commande **Q** soit utilisée avant les commandes **T.**

Et maintenant notre automobile avec des courbes de Bézier quadratiques.



d="M 50,100 Q 40,75 90,70 Q 95,60 95,50 Q 180,40 170,100 Z"

Figure 4-18. Automobile et quadratiques de Bézier

Courbe de Bézier cubique

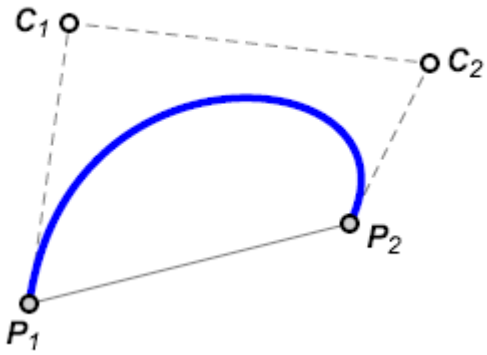


Figure 4-19. Courbe de Bézier cubique

La courbe de Bézier cubique est définie par un point de contrôle supplémentaire. Ce point de contrôle supplémentaire permet des formes beaucoup plus variées.

La courbe commence en P_1 en direction de C_1. Les points C_1 et C_2 sont approchés et la courbe se termine en P_2 venant de la direction de C_2. Ici aussi, nous pouvons imaginer que C_1 et C_2 exercent une attraction et déforment le segment P_1,P_2 pour donner cette courbe.

Comme pour la quadratique, nous allons illustrer une construction géométrique de cette courbe de Bézier cubique. Nous partons du quadrilatère $P_1C_1C_2P_2$.

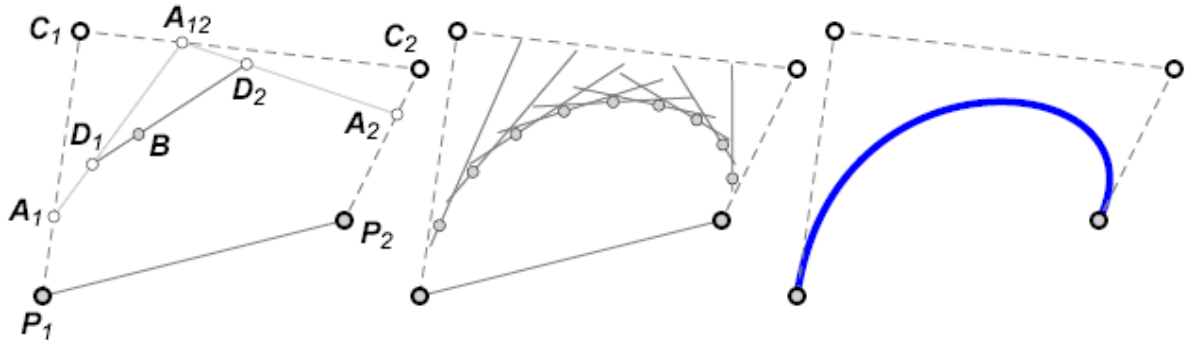


Figure 4-20. Construction géométrique de la cubique de Bézier

Nous marquons le point A_1 à *30%* du segment P_1C_1. Nous faisons de même sur C_1,C_2 et C_2,P_2 et obtenons les points A_{12} et A_2. Nous appliquons la construction de la quadratique au triangle $A_1A_{12}A_2$ pour obtenir les points D_1 et D_2 et enfin le point **B** qui est sur la cubique.

Nous changeons le rapport et sur la figure centrale, nous avons les points obtenus pour *10%,20%,…,90%* . Les derniers segments tracés D_1D_2 sont tangents à la courbe. La cubique est dessinée sur la figure de droite et présente quelques caractéristiques intéressantes:

- La tangente en P_1 a la direction C_1.
- La tangente P_2 a la direction de C_2.
- Si le quadrilatère $P_1C_1C_2P_2$ est convexe, la courbe est à l'intérieur de celui-ci.
- Si les points P_1,C_1,C_2,P_2 sont alignés, la courbe est un segment de droite.
- La courbe peut se recouper, elle n'est pas toujours convexe.
- La courbe peut avoir un point d'inflexion où elle change de sens de courbure.

Hmm, expliquons un peu ce dernier point. Que signifie changer de sens de courbure? Prenons la lettre ***C***, elle est toujours tracée en gardant le même sens, celui des aiguilles d'une montre si nous commençons par le bas ou plus souvent le sens anti-horaire en commençant par le haut. Nous pouvons dire que sa courbure est toujours positive ou négative, elle ne change pas au cours du tracé. Par contre si nous prenons la lettre ***S*** une partie de son tracé se fait dans un sens et l'autre partie dans l'autre sens – nous dirons que sa courbure change de sens et que le point où se fait ce changement est un point d'inflexion. Ceci permet à la courbe de Bézier cubique d'avoir des formes beaucoup plus variées que sa cousine quadratique.

Voyons son utilisation en SVG avec la commande **C**.

Syntaxe:

```
C|c x1, y1, x2, y2, x, y
    x1 = x du premier point de contrôle
    y1 = y du premier point de contrôle
    x2 = x du second point de contrôle
    y2 = y du second point de contrôle
    x = x du nouveau point
    y = y du nouveau point
```

La commande **C** nécessite trois couples de coordonnées comme paramètres. Les deux premiers couples définissent les points de contrôle C_1 and C_2 et le troisième le nouveau point courant P_2 , fin du tracé. Le point de départ P_1 est le point courant du tracé.

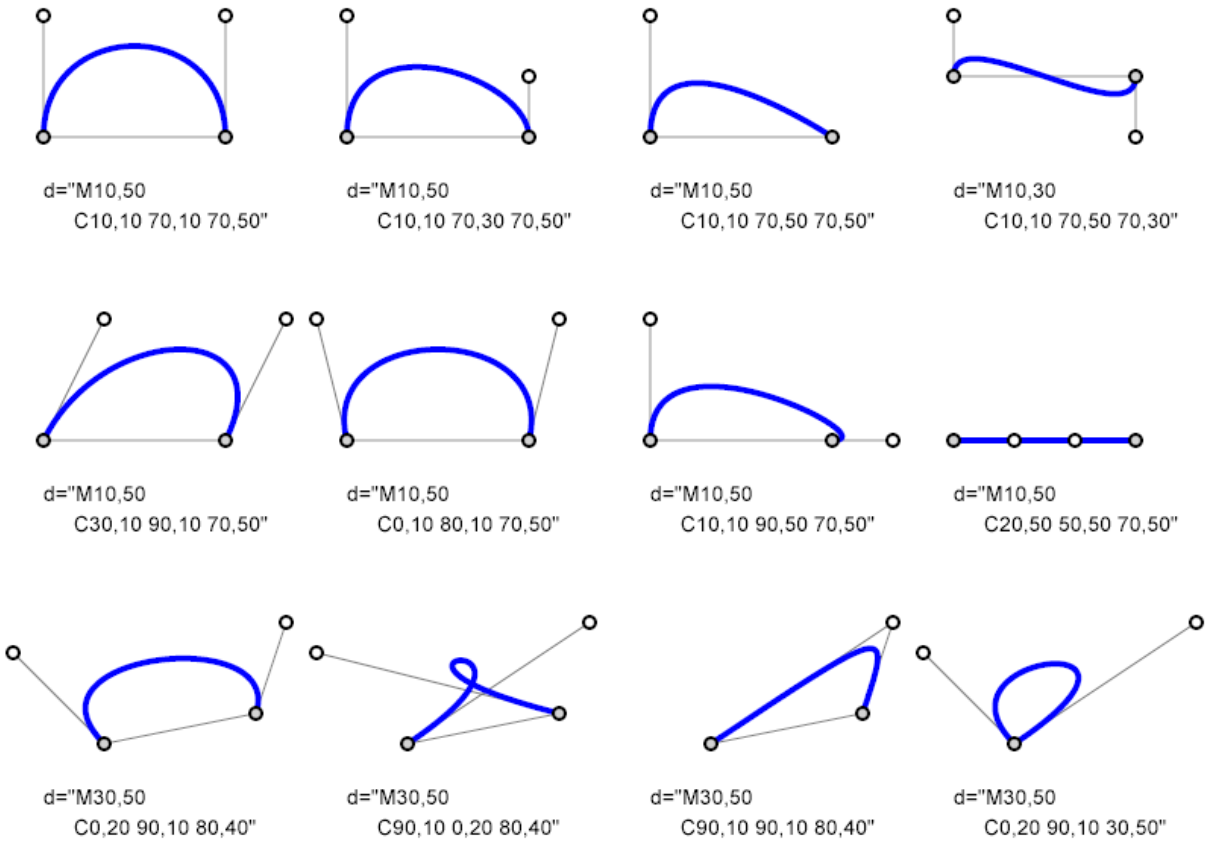


Figure 4-21. Quelques formes pour la cubique de Bézier

La figure 4-21 montre différentes dispositions des points de contrôle. Remarquez que si les quatre points forment un quadrilatère avec un axe de symétrie, la courbe admet également cet axe de symétrie. Nous pouvons aussi faire coïncider points de début et de fin pour fermer la courbe. Si les quatre points sont alignés, nous avons un segment de droite.

Mettons quelques cubiques à la suite.

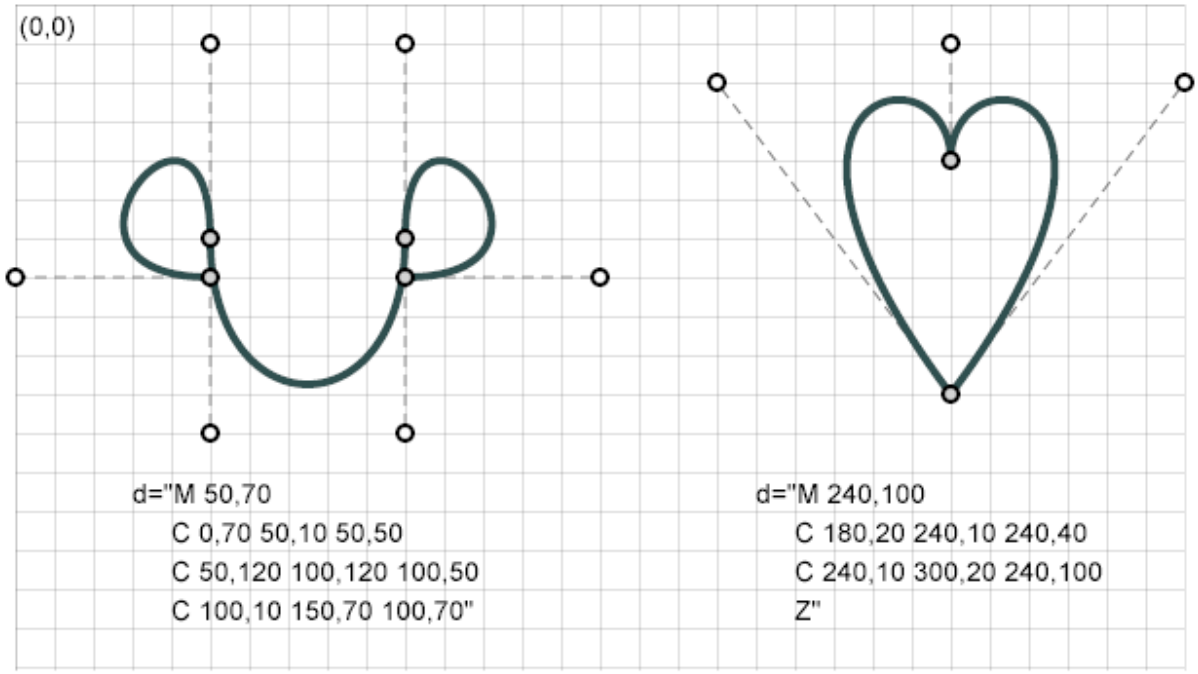


Figure 4-22. Polybézier cubiques

Comme pour la commande **T** de la quadratique, nous pouvons automatiser le premier point de contrôle pour la cubique avec la commande **S** et obtenir une continuité aux points de raccordement des tracés. Ici aussi, le premier point de contrôle est le symétrique du précédent par rapport au point courant. Nous devons également faire précéder cette commande d'une commande **C** ou **S** pour que le tracé soit rendu. Voici un exemple avec la figure 4-23, un trèfle à quatre feuilles.

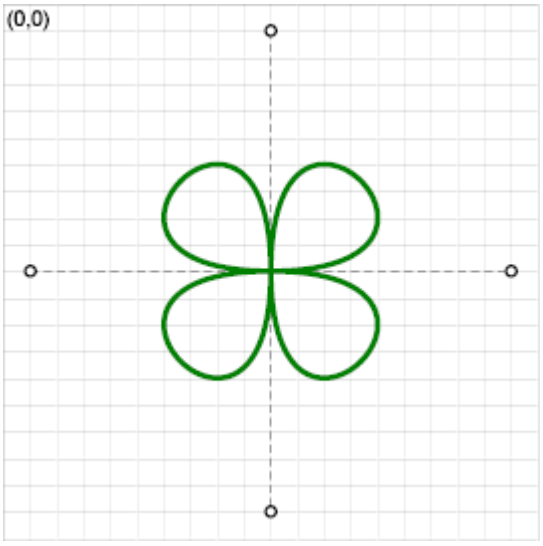
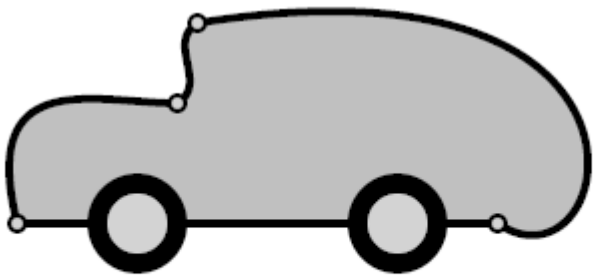


Figure 4-23. Utilisation de la commande S

```
<svg>
  <path stroke="green" stroke-width="2" fill="none"
        d="M100,100
           C100, 10 190,100 100,100
           S        100,190 100,100
           S         10,100 100,100
           S        100,190 100,100 Z"
        />
```

```
</svg>
```

Et notre automobile avec des cubiques de Bézier.



d="M 50,100 C 40,60 70,70 90,70 C 98,65 87,55 95,50 C 220,30 200,120 170,100 Z"

Figure 4-24. Automobile et cubiques de Bézier

Coordonnées relatives

Pour toutes ces commandes, nous avons utilisé les lettres majuscules, mais nous pouvons aussi utiliser les lettres minuscules. Chaque commande est double, **A** et **a**, **C** et **c** ….
La différence est très importante, avec les majuscules les paramètres sont définis dans le système de coordonnées de l'élément <path> alors que pour les minuscules les paramètres sont donnés dans un système de coordonnées relatif au point courant. Pour ces deux systèmes de coordonnées, seule l'origine n'est pas la même, mais ceci suffit à changer les valeurs des paramètres.

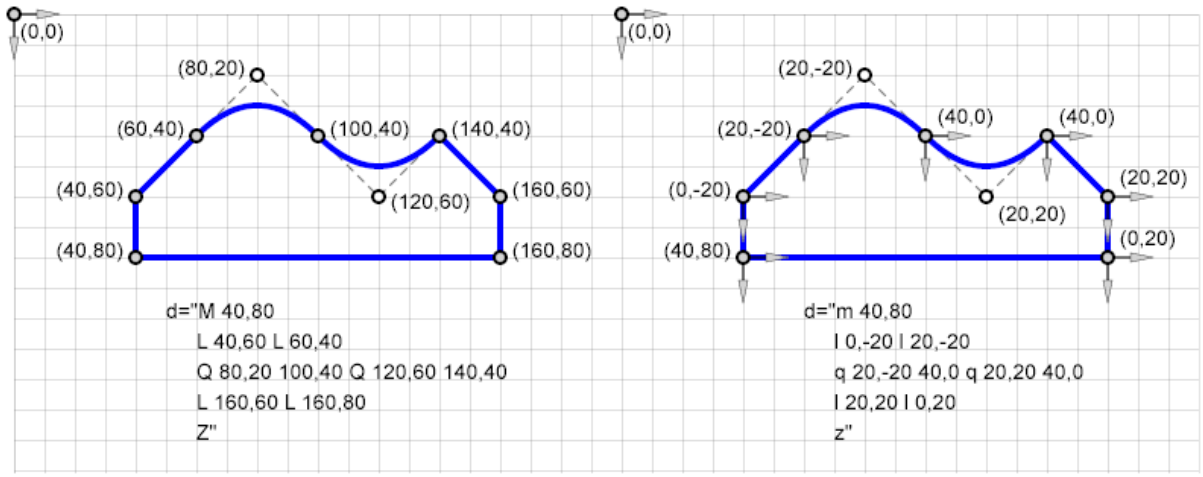


Figure 4-25. Commandes absolues et relatives

Nous avons deux fois le même tracé sur la figure 4-25. A gauche nous utilisons les majuscules, le repère est toujours le même. Sur la figure de droite, nous utilisons les minuscules et le repère change après chaque tracé.

Voyons de près cette figure. Nous commençons par une commande **m**. Comme c'est le début de notre élément, le point courant n'est pas défini, il coïncide avec l'origine du système de

coordonnées propre à l'élément <path>. Ici, il n'y a pas de différence entre **m** et **M**. Nous avons donc notre point courant *(40,80)* et définissons un repère provisoire pour la commande suivante. Dans ce repère le prochain point du tracé a les coordonnées *(0,-20)* d'où la commande **l 0,-20.** Nous avons un nouveau repère avec le point courant *(40,60)* comme origine. La figure montre ces repères successifs et les commandes utilisées.

Quelques remarques importantes pour ces commandes relatives:

1. Les commandes relatives ne changent que les coordonnées des points.
2. Angles, rayons et flags gardent les mêmes valeurs.
3. Les points de contrôle pour les courbes de Bézier ne changent pas le point courant, ils doivent être exprimés, ainsi que le nouveau point dans le repère défini par le point courant
4. Les commandes **z** et **Z** sont identiques. La commande **z** n'existe que par raison de cohérence.
5. La valeur du paramètre 'd' est plus compacte avec des commandes relatives.

*Soyez prudent(e), ne confondez pas la commande relative **lineto "l"** avec le nombre "1".*

Condenser les commandes

Après cette description des commandes, le choix de leur format a été dicté par une volonté de condenser les données. Voyons comment pousser cette concentration à son extrême.

Les valeurs numériques sont séparées entre elles et également des lettres de commande dans nos exemples. Nous pouvons éliminer certains séparateurs:

1. Nous pouvons supprimer des séparateurs superflus.
   d="M 10, 0, L 10, -20 L -10, 40Z"
   d="M 10,0 L 10,-20 L -10,40 Z"

2. Nous pouvons supprimer les séparateurs avant et après les commandes.
   d="M 10,0 L 10,-20 L -10,40 Z"
   d="M10,0L10,-20L-10,40Z"

3. Nous pouvons supprimer les séparateurs avant un nombre négatif.
   d="M10,0L10,-20L-10,40Z"
   d="M10,0L10-20L-10,40Z"

4. Nous pouvons supprimer les commandes si elles sont du même type que la précédente
   d="M10,0L10-20L-10,40Z"
   d="M10,0L10-20-10,40Z"

5. Après une commande **M,m,** des couples de coordonnées seront interprétés comme des commandes **L,l**.
   d="M10,0L10-20-10,40Z"
   d="M10,0 10-20-10,40Z"

6. Pensez à utiliser **H,h** et **V,v** pour des tracés horizontaux ou verticaux.
   d="M10,0L10-20-10,40Z"
   d="M10,0V-20L-10,40Z"

Courbes fermées

Souvent, un tracé se termine par un arc ou une courbe, nous pouvons fermer cette courbe ou non avec la commande ***Z.*** Même si le dernier point coïncide avec le premier point du tracé, ajouter la commande ***Z*** permet d'être certain que la jonction entre les points sera régulière comme sur cet exemple.

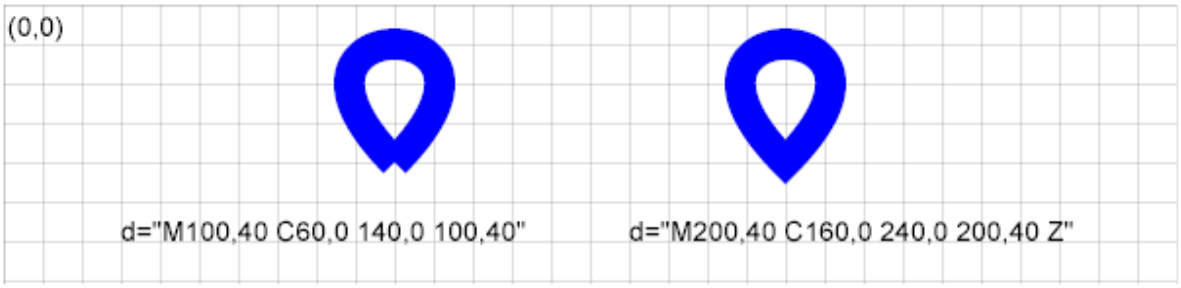


Figure 4-26. Terminer avec Z ou non

Remplir un élément "path"

Le remplissage d'un élément <path> n'est pas différent du remplissage d'un élément <polygon> ou <polyline>. Un tracé non fermé le sera automatiquement pour le remplissage, mais le segment ajouté ne sera pas dessiné (Figure 4-27).

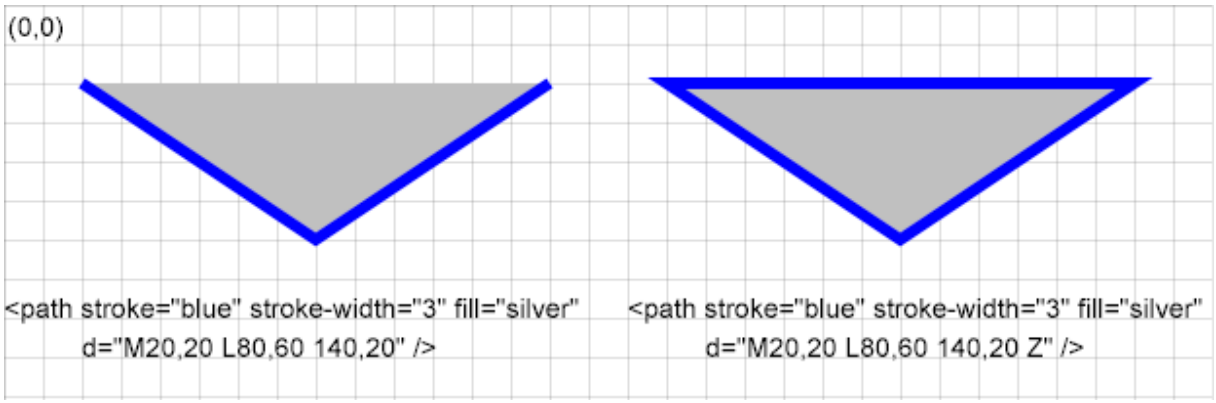


Figure 4-27. Remplissage de formes fermées ou non

Comme les éléments 'paths' peuvent se recouper ou avoir des tracés annexes, ils peuvent se recouvrir ou se contenir mutuellement. SVG propose plusieurs règles de remplissage pour définir l'intérieur de la forme. Pour établir ces règles:

> *Afin de savoir si un point est à l'intérieur du tracé, nous dessinons un rayon issu de ce point et partant dans une direction quelconque. Ensuite nous examinons les intersections de ce rayon avec le contour de la forme.*

Sur la base de ces algorithmes, SVG 1.0 definit l'attribut de présentation '*fill-rule'* qui peut prendre trois valeurs ***evenodd, nonzero*** ou ***inherit***. Avec SVG version 2.0, nous pourrions voir une valeur supplémentaire ***winding-count***.

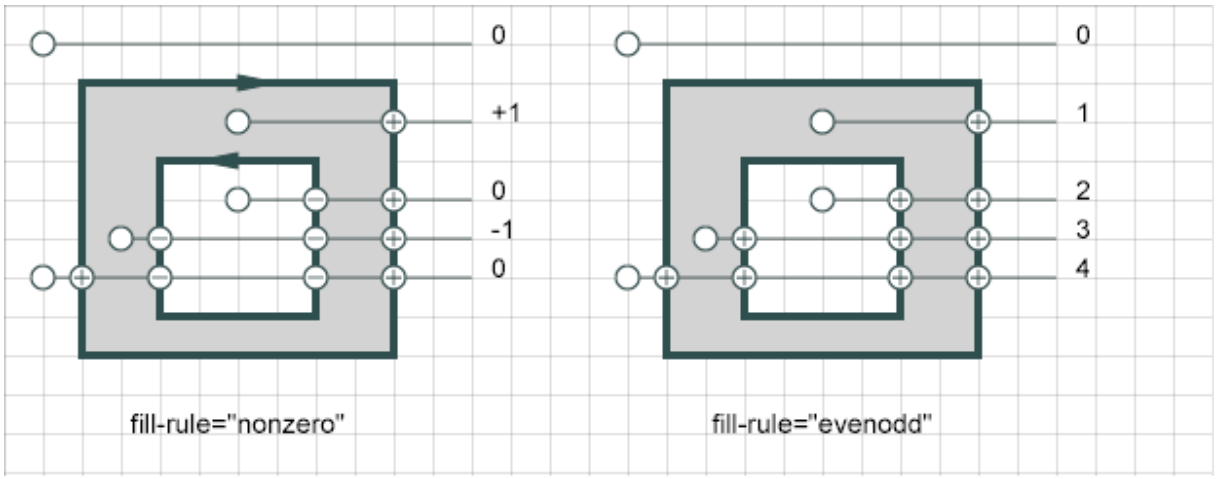


Figure 4-28. Valeurs pour 'fill-rule'

- ***nonzero:*** Quand le rayon rencontre un tracé orienté dans le sens des aiguilles d'une montre nous ajoutons **+1** au compteur. Si le tracé est dans le sens anti-horaire nous ajoutons **–1**. Si le total est *zero,* le point est à l'extérieur, dans le cas contraire, il est à l'intérieur.
- ***evenodd:*** Nous ajoutons simplement **+1** pour chaque rencontre. Si le résultat est pair le point est à l'extérieur, si le total est impair ('odd') le point est à l'intérieur.
- ***winding-count:*** Le décompte est le même que pour 'nonzero', mais si le résultat *n* est supérieur à 1, le point est rempli *n* fois avec la couleur de remplissage. Nous n'avons une traduction au niveau du dessin que si l'attribut *fill-opacity* est inférieur à 1.

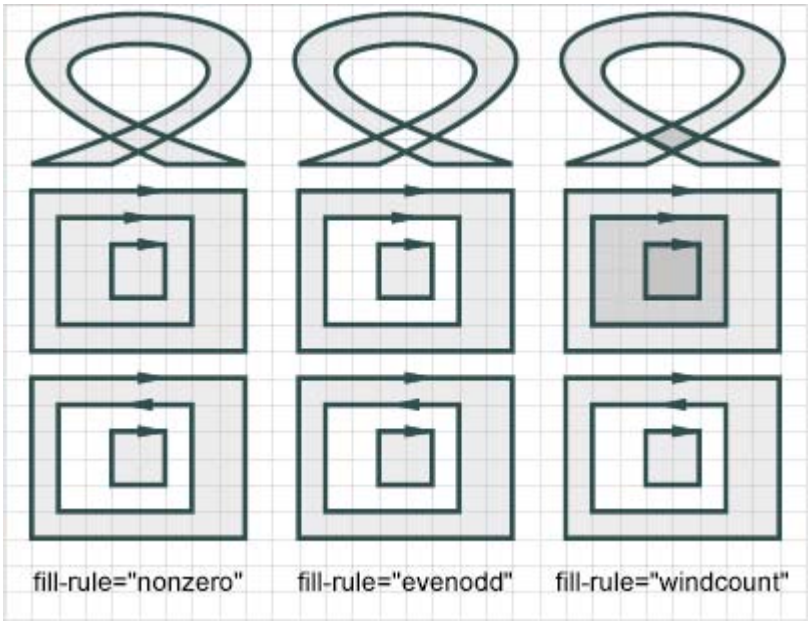


Figure 4-29. Valeurs de 'fill-rule' et dessin

Une conséquence de ces règles de remplissage est que nous pouvons créer des trous dans un objet. Appliquons ceci à notre automobile en lui ajoutant une vitre et nous voyons à travers celle-ci une partie de la seconde automobile. Nous avons un effet de masque.

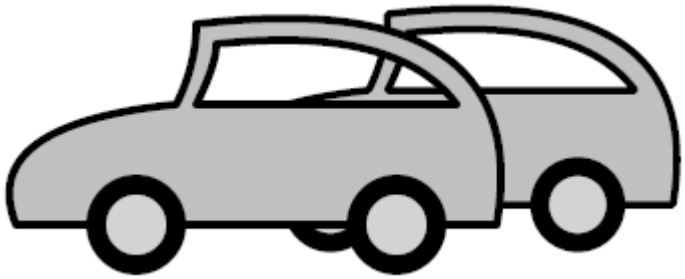


Figure 4-30. Automobiles et masque

Voici le code de la figure 4-30.

```
<svg width="600" height="400" viewBox="0 0 300 200">
  <defs>
     <g id="car">
       <path fill="silver" stroke="black" stroke-width="2"
             stroke-linejoin="round"
             d="M 50,100
                Q 40,75 90,70
                Q 95,60 95,50
                Q 180,40 170,100
                Z
                M 160,70
                Q 145,50 100,55
                Q 100,60 95,70
                Z" />
       <circle cx="80" cy="100" r="10" stroke="black"
               stroke-width="5" fill="lightgray" />
       <circle cx="145" cy="100" r="10" stroke="black"
               stroke-width="5" fill="lightgray" />
     </g>
  </defs>
  <use xlink:href="#car"
       transform="translate(100,95) scale(0.95) translate(-50,-100)"/>
  <use xlink:href="#car" />
</svg>
```

Marqueurs sur une courbe

Nous avons vu cet élément au chapitre 3, où nous avons créé des flèches pour les dimensions de notre rack. Voyons ce qu'il y a de nouveau à appliquer ces marqueurs aux courbes définies par 'path'.

Tout d'abord, nous ne pouvons placer ces marqueurs qu'à des points précis – les sommets de notre tracé. L'attribut *marker-start* place un marqueur au point de départ de notre tracé, *marker-end* le place à la fin du tracé et enfin *marker-mid* place un marqueur à chaque sommet intermédiaire. Si nous voulons les trois à la fois nous pouvons utiliser simplement le raccourci *marker*. Les spécifications ne traitent pas explicitement de l'ambiguité d'avoir à la fois 'marker-start' et 'marker-end' pour des courbes. Aussi le résultat peut dépendre du visualiseur utilisé, il peut y avoir deux marqueurs ou seulement un.

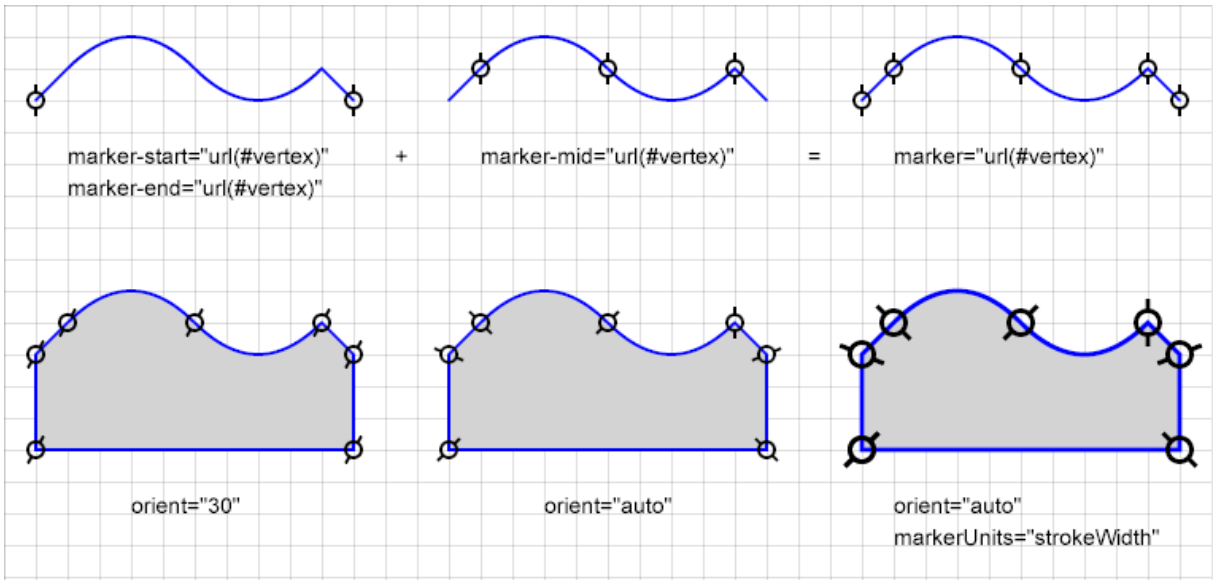


Figure 4-31. Marqueurs sur une courbe

Notez que *marker-start* et *marker-end* s'appliquent seulement à la première partie ou la dernière du tracé et non aux tracés intermédiaires

SVG permet de définir l'orientation des marqueurs. Nous pouvons fixer un angle fixe avec l'horizontale (voir *orient=”30”* sur la figure 4-31). Souvent nous avons besoin d'une orientation qui dépende de la direction de la tangente à la courbe, nous pouvons écrire *orient=”auto”*. Avec cette valeur le système de coordonnées du marqueur est orienté pour que l'axe des abscisses coïncide avec la tangente à la courbe. Cependant, il peut y avoir conflit entre la direction des deux tangentes aux points de raccordement des tracés, dans ce cas c'est la bissectrice des deux tangentes qui sert de direction. Ceci est visible sur le dessin central inférieur de notre figure 4-31.

Voici le code SVG pour notre dernière courbe (inférieure droite) de la figure 4-31.

```
<svg width="800" height="400" viewBox="0 0 400 200">
  <defs>
    <marker id="vertex" viewBox="-5 -5 10 10" markerWidth="10"
            markerHeight="10" markerUnits="strokeWidth" orient="auto">
      <path stroke="black" d="M0,-2.5 0,-5 M0,2.5 0,5" />
      <circle r="2.5" stroke="black" fill="none"/>
    </marker>
  </defs>
  <path marker-start="url(#scaleVertex)" marker-mid="url(#scaleVertex)”
        d="M10,60 L10,30 L20,20 Q40,0 60,20 T100,20 L110,30 L110,60 Z"
        stroke-width="1.5" stroke="blue" fill="lightgray"
        stroke-linecap="round" />
</svg>
```

Dans cet exemple, la taille du marqueur s'ajuste à l'épaisseur du tracé avec *markerUnits=”strokeWidth”*.

Peut-être pensez-vous qu'il serait utile que nous puissions choisir la place des marqueurs. En attendant une prochaine version de SVG, nous pouvons le faire avec un script.

Supposons que nous voulons disposer des marqueurs régulièrement sur une courbe comme sur cette figure:

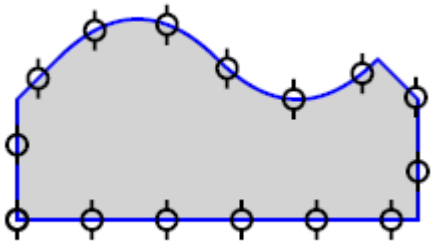


Figure 4-32. Marqueurs réguliers sur une courbe

Voici le code du document.

```
<svg width="800" height="400" viewBox="0 0 400 200"
                                          onload="Affiche_marqueurs(evt)">
  <defs>
    <g id="marqueur">
      <path stroke="black" d="M0-2.5 0-5 M0,2.5, 0,5" />
      <circle r="2.5" stroke="black" fill="none"/>
    </g>
  </defs>
  <path id="contour"
        d="M10,60 L10,30 L20,20 Q40,0 60,20 T100,20 L110,30 L110,60 Z"
        stroke="blue" fill="lightgray" stroke-linecap="round" />
  <script><![CDATA[
    function Affiche_marqueurs(evt)
    {
       var path = evt.target.ownerDocument.getElementById("contour"),
           pathLength = path.getTotalLength(),
           point = null,
           use = null,
           n = 15;

       for (var i=0; i<n; i++)
       {
          point = path.getPointAtLength(i/n*pathLength);
          use = evt.target.ownerDocument.createElement("use");
          use.setAttributeNS("http://www.w3.org/1999/xlink", "href",
                             "#marqueur");
          use.setAttribute("x", point.x);
          use.setAttribute("y", point.y);
          evt.target.ownerDocument.documentElement.appendChild(use);
       }
    }
  ]]></script>
</svg>
```

Pas de panique! Nous verrons en détail l'utilisation de script au chapitre 11. Voici le détail des opérations.

1) Nous plaçons l'élément *marqueur* dans un groupe.
2) Nous appelons la fonction quand le document est chargé.
3) Grâce à une boucle, nous parcourons la courbe régulièrement à 0/15, 1/15, 2/15, .., 14/15 de sa longueur totale.
4) Nous créons un élément *use* à chacun de ces points.

Nous pouvons facilement accéder aux objets du SVG DOM (voir le chapitre 11). Malheureusement, il n'y a pas de méthode simple pour connaître les tangentes en ces points et pouvoir orienter nos marqueurs.

Ne dessiner que les marqueurs

Nous pouvons utiliser l'élément <path> pour n'afficher que les marqueurs, il suffit de donner aux attributs *stroke* et *fill* la valeur *none*.

Si vos marqueurs sont de simples points, voici une méthode efficace où nous n'utilisons pas de marqueurs mais jouons sur les attributs de présentation.

- Donnez à *stroke-width* la valeur nécessaire pour la taille des points
- Donnez à *stroke-linecap* la valeur *round*.
- Placez des segments de longueur nulle en utilisant uniquement les commandes *moveto/closepath*.

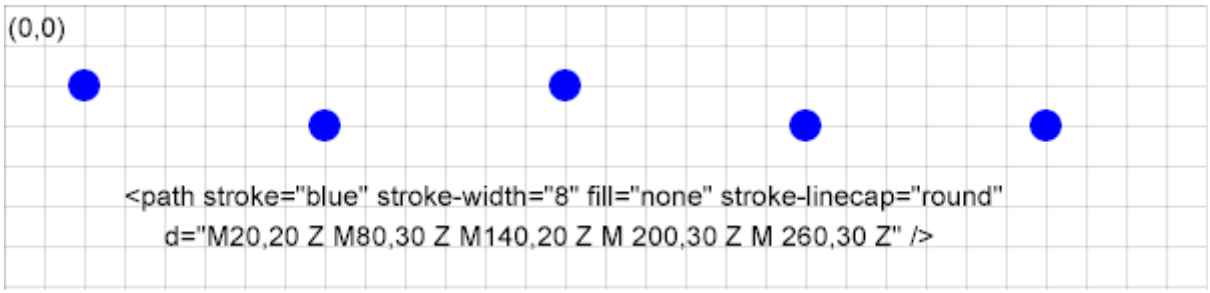


Figure 4-33. Rien que des points pour cette courbe

Nous obtenons ainsi des points, sans courbe les reliant.

Référence des commandes

Commandes & Paramètres		Résultat
M,m	***x, y***	Déplace le point courant en (*x, y*).
L,l	***x, y***	Trace un segment de droite du point courant au point (*x, y*).
H,h	***x***	Trace un segment horizontal du point courant au point (*x, y du point courant*).
V,v	***y***	Trace un segment vertical du point courant au point (*x du point courant, y*).
A,a	***rx, ry, x-axis-rotation, large-arc-flag, sweep-flag, x, y***	Trace un arc d'ellipse du point courant au point (*x, y*). L'ellipse a *rx* et *ry* comme demi-axes, l'axe principal fait un angle de *x-axis-roation* (en degrés) avec l'horizontale. Si *large-arc-flag* est 0 (zero), le petit arc (moins de 180°) est dessiné, pour une valeur de 1 c'est le grand arc arc (supérieur à 180°). Si *sweep-flag* est 0 l'arc tracé est dans le sens anti-horaire, sinon c'est l'autre arc qui est dessiné.
Q,q	***x1, y1 x, y***	Trace une courbe de Bézier quadratique du point courant au point (*x,y*) avec le point de contrôle (*x1,y1*).
T,t	***x, y***	Trace une courbe de Bézier quadratique du point courant au point (*x,y*) avec comme point de contrôle le symétrique du précédent point de contrôle par rapport au point courant.
C,c	***x1, y1 x2, y2 x, y***	Trace une courbe de Bézier cubique du point courant au point (*x,y*) avec les points de contrôle (*x1,y1*) et (*x2,y2*).
S,s	***x2, y2 x, y***	Trace une courbe de Bézier cubique du point courant au point (*x,y*) avec comme points de contrôle le symétrique du précédent point de contrôle par rapport au point courant et (*x2,y2*).

Table 4-2. Commandes et paramètres