Chapitre 3 : Structure du document

Les avantages de documents structurés

Quand nous créons un document avec SVG, nous pouvons, le plus souvent, le faire de différentes manières. Je ne veux pas dire remplacer un élément graphique par un autre, par exemple utiliser un élément ‘path’ pour dessiner un rectangle. Je ne parle pas non plus des différentes possibilités d’affecter les attributs de présentation..

Essentiellement je fais référence à l’utilisation d’éléments structurés, que l’on peut nommer **container d’éléments**. Prenons directement un exemple en nous mettant à la place d’un fabricant de racks pour ranger des palettes qui envisage une présentation graphique de ses produits.

Voici l’un de ses produits.

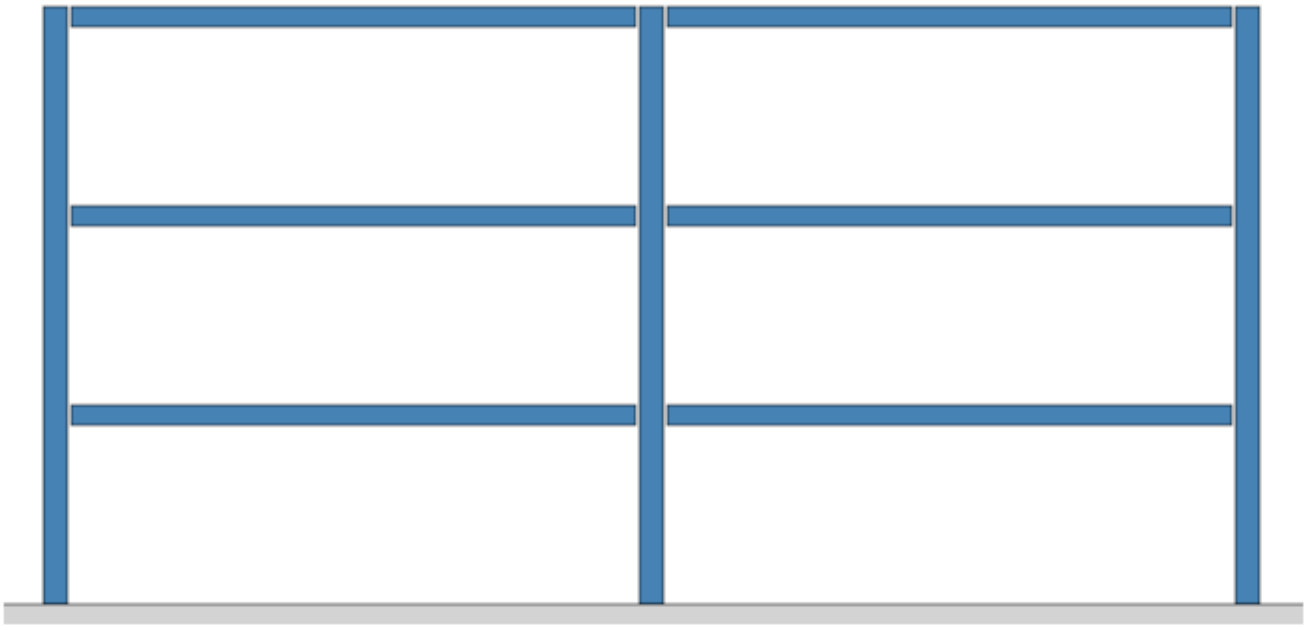

Figure 3-1. Le rack à palettes

Vous devez créer cette image en codant le document SVG:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
         "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <rect x="50" y="20" width="12" height="300" stroke="black" fill="steelblue"
        />
  <rect x="64" y="20" width="284" height="10" stroke="black" fill="steelblue"
        />
  <rect x="64" y="120" width="284" height="10" stroke="black"
        fill="steelblue" />
  <rect x="64" y="220" width="284" height="10" stroke="black"
        fill="steelblue" />
  <rect x="350" y="20" width="12" height="300" stroke="black"
        fill="steelblue" />
  <rect x="364" y="20" width="284" height="10" stroke="black"
        fill="steelblue" />
  <rect x="364" y="120" width="284" height="10" stroke="black"
        fill="steelblue" />
  <rect x="364" y="220" width="284" height="10" stroke="black"
```

```
        fill="steelblue" />
  <rect x="650" y="20" width="12" height="300" stroke="black"
        fill="steelblue" />
  <rect x="30" y="320" width="654" height="10" stroke="none" fill="lightgray"
        />
  <line x1="30" y1="320" x2="684" y2="320" stroke="black" />
</svg>
```

Nous avons un document constitué essentiellement d'éléments `<rect>`. Il est compact et produit le graphique escompté. Mais vous êtes fabricant et pensez en termes de racks, supports, montants et palettes. De ce point de vue, le document n'est pas très explicite. Pour identifier les différentes pièces comme les supports, nous devons chercher les rectangles qui ont une plus grande largeur que hauteur, ce qui n'est pas très pratique. En dépit de la simplicité de ce document, nous avons beaucoup de mal à savoir qui fait quoi.

Créons un document pour le même résultat graphique, mais mieux organisé et structuré. Examinez ce code:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
         "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="12" height="300" />
    <rect id="support" x="14" y="0" width="284" height="10" />
    <g id="colonne">
      <use xlink:href="#support" x="0" y="0" />
      <use xlink:href="#support" x="0" y="100" />
      <use xlink:href="#support" x="0" y="200" />
      <use xlink:href="#montant" x="300" y="0" />
    </g>
    <g id="rack">
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
      <g id="sol">
        <rect x="-20" y="300" width="662" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-20" y1="300" x2="642" y2="300" stroke="black" fill="none"
              />
      </g>
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" fill="steelblue" stroke="black" />
</svg>
```

Quel document! Ce document a presque le double de la taille du précédent. Mais si nous voulons un document plus explicite en termes de **racks**, **montants** et **supports**, celui-ci l'est. Nous allons examiner les éléments SVG utilisés dans la suite du chapitre.

Modifier le document

Pour prouver le caractère fonctionnel de ce code, supposons que nous voulons faire les changements suivants:

- Un rack avec des supports plus épais
- Un rack rouge
- Un rack avec une colonne de plus

Figure 3-2. Les racks modifiés

Nous réalisons la première modification en changeant simplement la valeur de l'attribut `height` dans une seule ligne:

```
<rect id="support" x="14" y="0" width="284" height="20" />
```

Pour la seconde, nous changeons la couleur dans:

```
<use xlink:href="#rack" x="50" y="20" fill="red" stroke="black" />
```

Pour la troisième variante, nous ajoutons une simple ligne de code:

```
<g id="rack">
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
       <use xlink:href="#colonne" x="600" y="0" />
```

Essayez maintenant de créer ces variantes avec le premier code. Ce n'est pas une sinécure!

Ainsi, pour apporter des modifications, il faut structurer les documents.

Une autre option

Maintenant que vous êtes impressionné(e) par les documents structurés, mais toujours peiné(e) par la taille du code, regardez ce troisième code qui nous donne toujours le même graphique.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <g id="rack" externalResourcesRequired="true">
      <use xlink:href="RackParts.svg#xpointer(id('montant'))" x="0" y="0" />
      <use xlink:href="RackParts.svg# xpointer(id('colonne'))" x="0" y="0" />
      <use xlink:href="RackParts.svg# xpointer(id('colonne'))" x="300" y="0"
          />
      <g id="sol">
        <rect x="-20" y="300" width="662" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-20" y1="300" x2="642" y2="300" stroke="black" fill="none"
              />
      </g>
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" fill="steelblue" stroke="black" />
</svg>
```

Je ne veux pas expliquer le contenu en détail, mais examinons quelques caractéristiques.

- La taille du fichier est même plus petite que celle du premier code.

- Nous définissons `montant`, `support` et `colonne` dans un fichier externe `RackParts.svg`.
- Nous faisons simplement référence aux groupes de ce fichier externe.

Vous devez penser — *C'est du bluff. Comme nous avons besoin d'un fichier externe, nous ne réduisons pas la taille du code.*
Vous avez raison dans un sens, mais vous pouvez partager ce fichier externe. Le fabricant de palettes n'a pas qu'un rack à son catalogue, mais plusieurs centaines peut-être et tous ces documents SVG feront référence au même fichier externe. C'est bien un gain pour la taille du code.

Réutilisation des éléments

Voyons comment fonctionne le principe "définir une fois, utiliser de multiples fois" en SVG. Nous allons rebâtir le rack graduellement. Nous partons avec deux montants et trois supports.

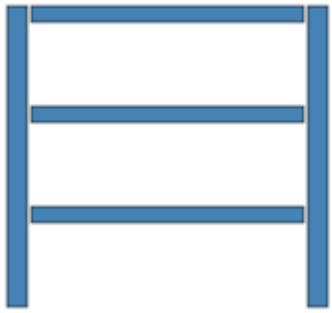

Figure 3-3. Elément de base

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <!-- les montants -->
  <rect x="50" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <rect x="200" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <!-- les supports -->
  <rect id="support" x="62" y="20" width="136" height="8" fill="steelblue"
        stroke="black" />
  <rect id="support" x="62" y="70" width="136" height="8" fill="steelblue"
        stroke="black" />
  <rect id="support" x="62" y="120" width="136" height="8" fill="steelblue"
        stroke="black" />
</svg>
```

Constatant que les trois rectangles des supports sont identiques à l'exception de leur position, nous essayons de réutiliser le rectangle.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <!-- les montants -->
  <rect x="50" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <rect x="200" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <!-- les supports -->
  <rect id="support" x="62" y="20" width="136" height="8" fill="steelblue"
        stroke="black" />
```

```
  <use xlink:href="#support" x="0" y="50" />
  <use xlink:href="#support" x="0" y="100" />
</svg>
```

Ceci fonctionne grâce à l'élément **<use>.** Le premier rectangle est simplement cloné – dupliqué – et positionné. Nous avons la même image.

Il est temps d'étudier cet élément <use>:

Syntaxe

```
<use xlink:href="uri"
     x="coordinate"
     y="coordinate"
     width="length"
     height="length"
     style-attribute="style-attribute"
>
```

L'attribut xlink:href référence l'élément à utiliser grâce à son id, nous devons définir un attribut id pour tous les éléments auxquels nous souhaitons faire référence dans <use>. La valeur de l'identificateur doit être unique dans le document. Comme cette valeur doit être du type XML **URI** (**uniform resource identifier**), nous pouvons référencer n'importe quel élément dans n'importe quel document SVG sur tout le Web. Le plus souvent nous faisons référence à des éléments du fichier local (local URI reference) avec

```
xlink:href="#name"
```

mais nous pouvons également faire référence à des éléments dans des ressources externes (non-local URI reference) avec une URI absolue comme

```
xlink:href=
"http://www.w3.org/TR/2001/REC-SVG-20010904/images/struct/Use01.svg/#xpointer(id('MyRect'))"
```

ou une URI relative comme

```
xlink:href="../svglib/Vehicels.svg#xpointer(id('Motorcycle'))"
```

Avec les attributs x et y, nous positionnons l'élément cloné dans le dessin. L'interprétation de ces attributs sera vue dans le chapitre 6 avec les systèmes de coordonnées et les transformations. Pour l'instant, donnons simplement des valeurs à x et y pour positionner correctement l'objet. Si nous ne donnons pas de valeur à x et y comme ici:

```
<use xlink:href="#support" />
```

les valeurs par défaut seront zéro, aussi l'élément <use> est positionné à la même place que l'original.

Une conséquence de l'utilisation de l'élément <use> est que les ajustements que nous apportons à celui-ci n'affectent pas l'original.

Un petit résumé des propriétés de <use>

- L'élément <use> crée une copie d'un autre élément.
- Les éléments clonés doivent avoir un identificateur. L'attribut id s'applique à tous les éléments SVG.
- L'élément cloné peut être dans le même fichier ou dans un fichier externe. Ceci permet la création de libraries de symboles par exemple.

- L'élément `<use>` ne peut changer les attributs de l'élément référencé.
- L'utilisation de `<use>` sur des éléments complexes – un élément `<path>` avec beaucoup de données par exemple – peut amener une réduction sensible de la taille du fichier.

Les utiliser tous ou aucun

Revenons à notre exemple de rack

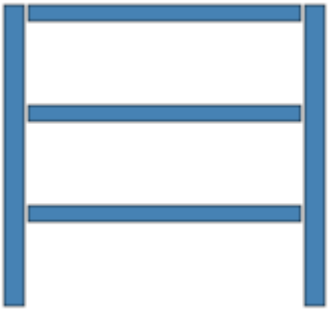

Figure 3-4. L'élément de base

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
<svg>
  <!-- les montants -->
  <rect x="50" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <rect x="200" y="20" width="10" height="150" fill="steelblue"
        stroke="black" />
  <!-- les supports -->
  <rect id="support" x="62" y="20" width="136" height="8" fill="steelblue"
        stroke="black" />
  <use xlink:href="#support" x="0" y="50" />
  <use xlink:href="#support" x="0" y="100" />
</svg>
```

Regardons la structure des supports.

Regarder la différence

Sur l'image il est impossible de savoir lequel est l'original et quels sont les clones. Nous devons examiner le code pour répondre.

Quelle est l'importance de connaître l'original?

La réponse est simplement "Aucune". L'importance est dans la structure du groupe. Cependant ceci peut être important dans certaines situations.

Supposons que nous voulons rehausser le dernier support de 20 unités. Aucun problème, nous changeons l'attribut `y` du dernier élément `<use>`.

```
<use xlink:href="#support" x="0" y="90" />
```

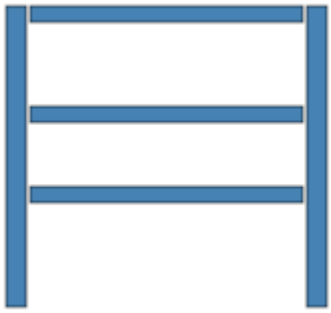

Figure 3-5. Support du bas rehaussé

Supposons maintenant que nous voulons modifier la position du support supérieur.

Nous devons modifier l'original. Mais … tous les éléments `<use>` dépendent de lui et un changement affectera tous les clones. Nous devrons alors reprendre tous les clones pour compenser la modification.

```
<rect id="support" x="62" y="30" width="136" height="8" fill="steelblue"
      stroke="black" />
<use xlink:href="#support" x="0" y="40" />
<use xlink:href="#support" x="0" y="80" />
```

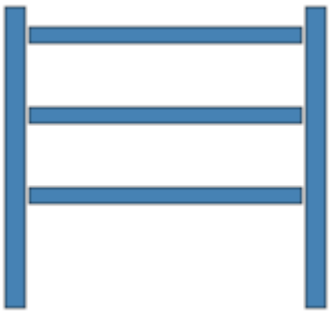

Figure 3-6. Modification de l'élément original

Oui, nous y arrivons, mais quel travail! Soyons soulagés de ne pas avoir à modifier des centaines de supports!

Que devons nous retenir? L'avantage que la modification de l'original s'applique à tous les clones devient ici un inconvénient.

Il serait bon de ne pas voir l'élément original. Aussi nous n'aurions pas à modifier sa position et nous n'aurions pas besoin de faire tout ce travail. Nous pouvons le faire en mettant cet élément dans une section `<defs>`

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <!-- les montants -->
  <rect x="50" y="20" width="10" height="150" fill="steelblue" stroke="black"
        />
  <rect x="200" y="20" width="10" height="150" fill="steelblue"
        stroke="black" />
```

```
  <defs> <!-- le support -->
    <rect id="support" width="136" height="8" fill="steelblue" stroke="black"
          />
  </defs>
  <use xlink:href="#support" x="62" y="30" />
  <use xlink:href="#support" x="62" y="70" />
  <use xlink:href="#support" x="62" y="110" />
</svg>
```

Nous avons toujours la même image, tous les rectangles visibles sont des éléments `<use>` clones de l'original stocké dans la section `<defs>`. Nous sommes maintenant certains que toute modification que nous ferons n'affectera pas les autres éléments.

> *Si vous voulez réutiliser des éléments SVG, placez les dans une section `<defs>`. C'est un point important. Comme les spécifications SVG nous le conseillent — "For understandability and accessibility reasons, it is recommended that, whenever possible, referenced elements be defined inside of a 'defs."*

Groupes

Abordons l'élément le plus important en terme de structure – l'élément groupe `<g>`. L'élément groupe n'est pas un élément graphique, il n'ajoute rien au graphisme par lui-même. C'est un container comme l'élément `<defs>` — il regroupe différents éléments, principalement d'autres éléments graphiques mais aussi d'autres containers.

Syntaxe

```
<g id="<name>"
   style-attribute="style-attribute"
   transform="transformation commands">
  <!-- contenu du groupe -->
</g>
```

L'attribut `id` est l'attribut standard XML pour assigner un nom unique à un élément. Il est très utilisé pour les groupes. Cet attribut `id` doit être connu des autres éléments qui l'utilisent comme référence. Les attributs de présentation seront discutés plus loin dans ce chapitre.

Nous allons construire un groupe `colonne` composé de trois supports et d'un montant qui pourra être réutilisé dans la construction du rack.

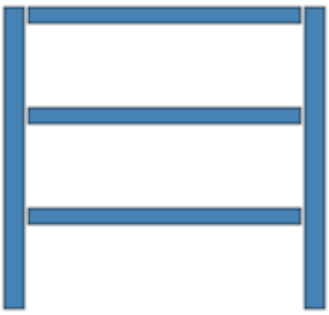

Figure 3-7. Le groupe "colonne"

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" fill="steelblue"
          stroke="black" />
    <rect id="support" width="136" height="8" fill="steelblue" stroke="black"
```

```
        />
  </defs>
  <use xlink:href="#montant" x="50" y="20" />
  <g id="colonne">
    <use xlink:href="#support" x="62" y="20" />
    <use xlink:href="#support" x="62" y="70" />
    <use xlink:href="#support" x="62" y="120" />
    <use xlink:href="#montant" x="200" y="20" />
  </g>
</svg>
```

Comme attendu, nous ne voyons rien de nouveau. Cependant nous verrons bientôt son intérêt. Un coup d'œil au code, pour définir un groupe il suffit d'insérer les éléments graphiques entre les balises `<g> .. </g>`.

Il y a plusieurs raisons d'utiliser les groupes.

- Nous voulons créer un objet complexe à réutiliser.
- Nous voulons que le code soit plus explicite et regrouper logiquement les éléments.
- Nous voulons appliquer à tous les éléments du groupe les mêmes attributs de présentation.
- Nous voulons simuler des couches et maîtriser leur visibilité.

Utiliser les groupes

L'utilité de l'élément `<g>` est accrue avec son compagnon — l'élément `<use>`. Nous utilisons `<use>` pour dessiner des objets comme des groupes. Nous pouvons ainsi créer un rack bien organisé.

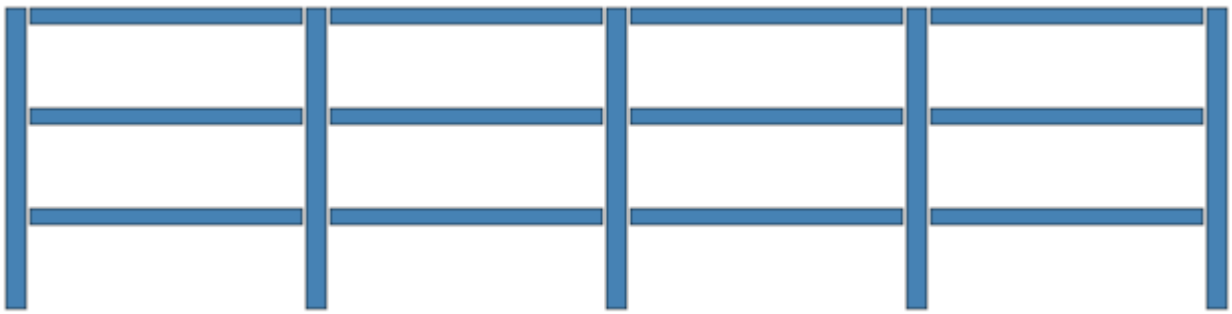

Figure 3-8. Plus d'éléments

Voici le code:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" fill="steelblue"
stroke="black" />
    <rect id="support" width="136" height="8" fill="steelblue" stroke="black"
          />
    <g id="colonne">
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
  </defs>
  <use xlink:href="#montant" x="50" y="20" />
  <use xlink:href="#colonne" x="50" y="20" />
  <use xlink:href="#colonne" x="200" y="20" />
```

```
    <use xlink:href="#colonne" x="350" y="20" />
    <use xlink:href="#colonne" x="500" y="20" />
  </svg>
```

Comme nous l'avons vu, nous mettons le groupe `colonne` dans une section `<defs>`. Nous n'avons que cinq éléments pour créer notre graphique, un clone de `montant` et quatre clones de `colonne`.

Comme c'est important, répétons:

> *Tout groupe qui sera réutilisé devra être dans une section `<defs>`.*

Il est courant de réutiliser des groupes dans différents fichiers SVG. Nous savons que SVG permet le référencement d'éléments d'autres fichiers locaux ou non avec l'attribut `xlink:href="`*`uri`*`"`.
Nous voulons l'appliquer à notre rack. Nous avons deux fichiers:

RackParts.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" fill="steelblue"
          stroke="black" />
    <rect id="support" width="136" height="8" fill="steelblue"
          stroke="black"/>
    <g id="colonne">
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
  </defs>
</svg>
```

et

Rack.svg

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" externalResourcesRequired="true"
     xmlns="http://www.w3.org/2000/svg">
  <use xlink:href="../parts/RackParts.svg#xpointer(id('montant'))" x="50"
       y="20" />
  <use xlink:href="../parts/RackParts.svg#xpointer(id('colonne'))" x="50"
       y="20" />
  <use xlink:href="../parts/RackParts.svg#xpointer(id('colonne'))" x="200"
       y="20" />
  <use xlink:href="../parts/RackParts.svg#xpointer(id('colonne'))" x="350"
       y="20" />
  <use xlink:href="../parts/RackParts.svg#xpointer(id('colonne'))" x="500"
       y="20" />
</svg>
```

Le fichier `RACKPARTS.SVG` est localisé dans un sous-répertoire "parts" parallèle à celui de `RACK.SVG`.

Nous sommes désormais incapables de nous passer de groupes dans nos documents SVG. Voyons comment structurer nos documents avec un grand nombre de groupes.

Groupes inclus

L'élément groupe est un container d'éléments SVG, non seulement des objets graphiques mais aussi d'autres containers, en particulier d'autres groupes. Nous avons donc des groupes à l'intérieur d'autres groupes que nous nommerons **sous-groupes** . Comment référencer ces sous-groupes ou même pourquoi utiliser des sous-groupes. Voyons ces sous-groupes sur notre exemple de rack.

Utilisons des sous-groupes pour notre exemple

Les images de la brochure de notre compagnie présente toujours le sol sous le rack. Pour visualiser ce sol, nous utilisons une ligne noire et un rectangle gris. D'autre part il nous manque quelques détails pour nos supports — les cales.

Construisons ces cales.

1. Nous raccourcissons les supports et plaçons deux petits rectangles aux extrémités.
2. Mettons ces trois éléments dans un groupe nommé '`support`', ainsi le reste du document ne sera pas modifié.
3. Nous définissons également un groupe '`rack`' avec les colonnes et le montant de départ. Nous ajoutons les éléments du sol également.

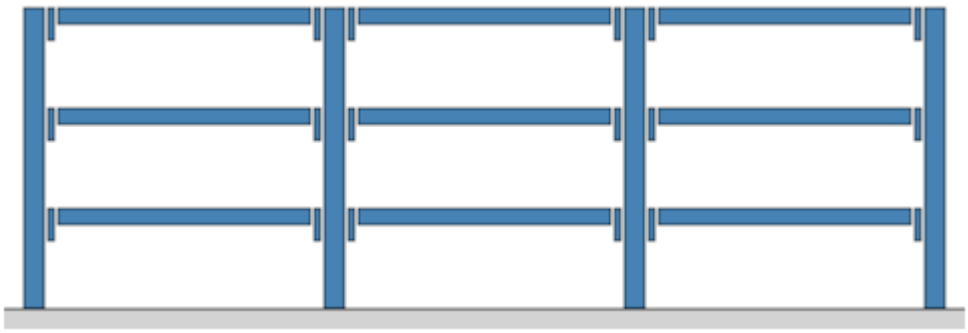

Figure 3-9. Rack avec cales et sol

4. Ce qui nous donne ce code:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" fill="steelblue"
          stroke="black" />
    <g id="support">
      <rect x="0" width="3" height="16" fill="steelblue" stroke="black" />
      <rect x="5" width="126" height="8" fill="steelblue" stroke="black" />
      <rect x="133" width="3" height="16" fill="steelblue" stroke="black" />
    </g>
    <g id="colonne">
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
    <g id="rack">
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="150" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
  <rect x="40" y="170" width="480" height="10" stroke="none"
```

```
        fill="lightgray" />
    <line x1="40" y1="170" x2="520" y2="170" stroke="black" />
</svg>
```

L'image est correcte mais pourquoi ne pas mettre les éléments du sol dans un groupe '`sol`'.

1. Nous créons ainsi ce groupe

```
<use xlink:href="#rack" x="50" y="20" />
<g id="sol">
  <rect x="40" y="170" width="480" height="10" stroke="none"
        fill="lightgray" />
  <line x1="40" y1="170" x2="520" y2="170" stroke="black" />
</g>
```

Cela semble mieux structuré. Devons nous laisser le groupe '`sol`' où il est ou est-il préférable de le mettre dans la section `<defs>` ? Nous ne réutilisons pas ce groupe, il n'est donc pas nécessaire de le mettre dans la section `<defs>`.

La question est "Est-ce que chaque rack a son propre sol?"

Vous, expert es rack, répondez immédiatement: "Oui absolument — car il est peu probable que des racks partagent le même sol ". Bien, le groupe 'sol' peut entrer dans le groupe 'rack'. C'est le bon sens!

2. Nous modifions donc notre code

```
<g id="rack">
  <g id="sol">
    <rect x="-10" y="150" width="480" height="10" stroke="none"
          fill="lightgray" />
    <line x1="-10" y1="150" x2="470" y2="150" stroke="black" />
  </g>
  <use xlink:href="#montant" x="0" y="0" />
  <use xlink:href="#colonne" x="0" y="0" />
  <use xlink:href="#colonne" x="150" y="0" />
  <use xlink:href="#colonne" x="300" y="0" />
</g>
```

Si vous vous demandez pourquoi je mets le groupe sol au début plutôt qu'à la fin, je pourrais vous répondre que c'est par habitude de codage car ici il n'y a pas de problème de superposition des objets (les objets sont rendus dans l'ordre du code et l'objet suivant peu recouvrir le précédent). Notez également que nous devons modifier les positions des éléments 'line' et 'rectangle' (nous soustrayons les coordonnées de l'élément `<use>` qui affiche le rack). Nous verrons ceci en détail au **chapitre 6**, où nous aborderons les systèmes de coordonnées et les transformations.

Après avoir examiné le code, vous pouvez poser la question: "Pourquoi ne pas définir le groupe 'support' à l'intérieur du groupe 'colonne'? "

Bonne question!. Il n'y a pas d'objection à le faire.

3. Nous modifions le groupe 'colonne'.

```
<g id="colonne">
  <defs>
    <g id="support">
      <rect x="0" width="3" height="16" fill="steelblue" stroke="black" />
      <rect x="5" width="126" height="8" fill="steelblue" stroke="black" />
      <rect x="133" width="3" height="16" fill="steelblue"
            stroke="black" />
```

```
      </g>
    </defs>
    <use xlink:href="#support" x="12" y="0" />
    <use xlink:href="#support" x="12" y="50" />
    <use xlink:href="#support" x="12" y="100" />
    <use xlink:href="#montant" x="150" y="0" />
  </g>
```

Nous ne pouvons simplement déplacer le groupe ‘support’ dans le groupe ‘colonne’, nous devons créer une section `<defs>` locale. Pourquoi? Bien, nous avons vu que les éléments des sections `<defs>` ne sont pas rendus, ils ne servent que de référence. Si nous n’utilisons pas de section `<defs>` le groupe ‘support’ sera dessiné.

Cela peut vous paraître confus. Pourquoi ne pas laisser le groupe ‘support’ où il est? Mais si nous voulons avoir une structure complète pour le document, il faut le faire.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
         "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" fill="steelblue"
          stroke="black" />
    <g id="colonne">
      <defs>
        <g id="support">
          <rect x="0" width="3" height="16" fill="steelblue"
                stroke="black" />
          <rect x="5" width="126" height="8" fill="steelblue"
                stroke="black" />
          <rect x="133" width="3" height="16" fill="steelblue"
                stroke="black" />
        </g>
      </defs>
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
    <g id="rack">
      <g id="sol">
        <rect x="-10" y="150" width="480" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-10" y1="150" x2="470" y2="150" stroke="black" />
      </g>
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="150" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
</svg>
```

Fantastique — une place pour chaque chose et chaque chose à sa place.

Quelques règles pour les groupes

Voici un petit résumé des règles applicables aux groupes:

- Des groupes qui serviront de référence dans le même ou d’autres documents doivent être placés dans une section `<defs>` principale *(groupe public)*.
- Des groupes qui seront uniquement utilisés dans le groupe parent, seront définis dans ce groupe parent *(groupe privé)*. Ceci permet de modifier le groupe totalement.

- Eviter de référencer un tel sous-groupe à l'extérieur de son groupe parent *(accès public à un groupe privé)*.
- Ne jamais référencer un groupe à l'intérieur de lui-même *(auto-référence)*. Il ne pourrait être défini.

Attributs de présentation et groupes

Les spécifications SVG indiquent que le groupe transmet ses attributs à son contenu. Ceci vient des propriétés de transmissibilité pour les règles CSS2.

Illustrons par un exemple.

1. Entrons ce code dans un document. Deux cercles bleus forment le contenu d'un groupe.

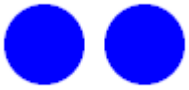

Figure 3-10. Deux cercles bleus

```
<g>
  <circle cx="50" cy="50" r="20" fill="blue" />
  <circle cx="100" cy="50" r="20" fill="blue" />
</g>
```

2. Maintenant nous pouvons faire de cette couleur de remplissage un attribut du groupe.

```
<g fill="blue">
  <circle cx="50" cy="50" r="20" />
  <circle cx="100" cy="50" r="20" />
</g>
```

Nous voyons que les éléments du groupe héritent leur attribut de celui du groupe.

3. Que se passe-t-il si un élément a également une couleur définie comme son propre attribut?

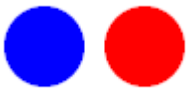

Figure 3-11. Groupe bleu et cercle rouge

```
<g fill="blue">
  <circle cx="50" cy="50" r="20" />
  <circle cx="100" cy="50" r="20" fill="red" />
</g>
```

Nous constatons que seuls les éléments pour lesquels la couleur de remplissage n'est pas définie héritent de la valeur donnée pour le groupe.

Si nous avons des éléments qui ont des attributs de présentation communs, c'est intéressant de les définir au niveau global. Mais il faut alors mettre ces éléments dans un groupe.

4. Vérifions que ceci est vrai également pour les sous-groupes avec cet exemple:

Figure 3-12. Groupe vert et cercle rouge

```
<g fill="green">
  <g>
    <circle cx="50" cy="50" r="20" />
    <circle cx="100" cy="50" r="20" fill="red" />
  </g>
  <rect x="130" y="30" width="40" height="40" />
</g>
```

Pas de surprise ici. le sous-groupe, aussi bien que les éléments qui le composent héritent de la couleur du groupe.

Ici – pour plus de simplicité – nous utilisons les attributs de présentation. Si nous utilisions les styles CSS à la place, le résultat serait le même. Simplement vous devez savoir:

> *Quand nous mêlons attributs de présentation et styles CSS, les styles CSS ont la priorité et définissent la présentation.*
>
> *Vous pouvez annuler cette priorité avec la déclaration* `"!important"`.

Nous verrons en détail les propriétés des styles CSS.

Revenons à notre rack

4. Appliquons à notre code ce que nous venons de voir.

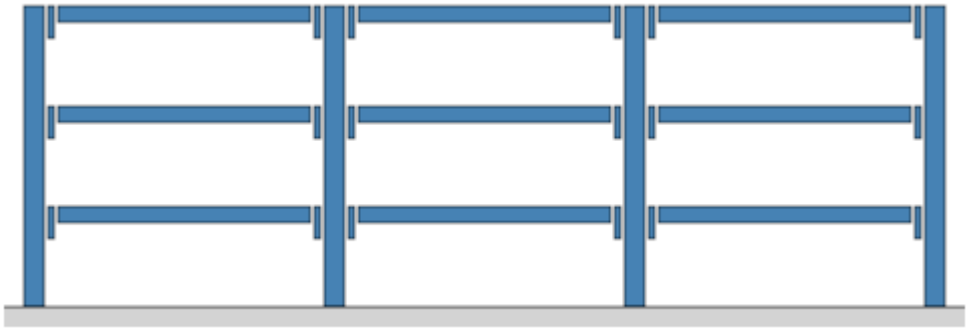

Figure 3-13. Notre rack …

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
        "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <rect id="montant" width="10" height="150" />
    <g id="colonne">
      <defs>
        <g id="support">
          <rect x="0" width="3" height="16" />
          <rect x="5" width="126" height="8" />
          <rect x="133" width="3" height="16" />
        </g>
      </defs>
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
```

```
    <g id="rack" fill="steelblue" stroke="black" >
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="150" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
      <g id="sol">
        <rect x="-10" y="150" width="480" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-10" y1="150" x2="470" y2="150" stroke="black" />
      </g>
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
</svg>
```

Nous avons supprimé toutes les déclarations de `fill="steelblue" stroke="black"` et simplement donné ces attributs au groupe rack. Ceci fonctionne très bien et diminue sensiblement la taille de notre fichier.

Mais si nous voulons avoir la colonne du milieu d'une autre couleur, 'moccasin' par exemple?

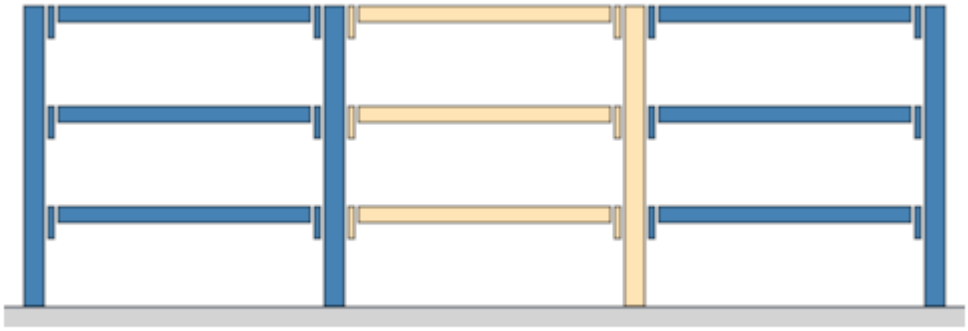

Figure 3-14. Changement de couleur pour la colonne centrale

Il suffit de donner un attribut 'fill' à cette colonne.

```
<use xlink:href="#colonne" x="0" y="0" />
<use xlink:href="#colonne" x="150" y="0" fill="moccasin" />
<use xlink:href="#colonne" x="300" y="0" />
```

Bien, mais nous n'avons pas donné la couleur au groupe colonne mais à un de ses clones. Comment l'élément `<use>` gére-t-il le style du groupe? Voyons un exemple très simple.

Figure 3-15. Couleur et élément "use"

```
<g fill="green">
  <defs>
    <g id="inner">
      <circle cx="50" cy="50" r="20" />
      <circle cx="100" cy="50" r="20" fill="red" />
    </g>
  </defs>
  <rect x="130" y="30" width="40" height="40" />
  <use xlink:href="#inner" fill="moccasin" />
</g>
```

Un groupe fait passer les attributs de présentation d'un de ses clones défini avec `<use>` aux éléments qui le composent aussi longtemps que le groupe lui-même n'a pas d'attributs de présentation. Dans ce cas, le groupe est hermétique au style attribué au clone. Les règles d'héritage CSS s'appliquent aussi aux clones.

Résumons!

- Les éléments du groupe héritent du style du groupe parent.
- Les éléments du groupe héritent également des clones du groupe parent.
- Si nous voulons avoir la même présentation pour tous les clones, nous affectons les attributs au groupe.
- Si nous voulons des clones avec des présentations différentes, nous affectons les attributs à chaque clone.

Symboles

SVG permet l'utilisation d'un autre élément de structure, l'élément `<symbol>`. C'est aussi un container comme `<g>` et `<defs>`. En fait il tient un peu des deux.

1. L'élément `<symbol>` contient des éléments graphiques et éventuelleemnt d'autres containers. Il ressemble à l'élément `<g>`.
2. Le contenu de l'élément `<symbol>` n'est pas dessiné. Il doit être référencé par un élément `<use>` pour être rendu. Dans ce sens, il ressemble à l'élément `<defs>`.
3. L'élément `<symbol>` est toujours rendu dans une zone rectangulaire. L'élément `<use>` qui le référence doit donc avoir les attributs `width` et `height`.

Syntaxe

Voyons la syntaxe de l'élément 'symbol'

```
<symbol id="name"
        viewbox="min-x min-y width height"
        preserveAspectRatio="align [meetOrSlice]"
        style-attribute="style-attribute"
>
  <!-- contenu graphique -->
</symbol>
```

et rapidement un exemple.

Utiliser des symboles

Comment utiliser dans la pratique ces symboles.

1. Commençons avec ce code:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <symbol id="icon" viewBox="0 0 100 100" preserveAspectRatio="none"
          stroke="blue" stroke-width="5" fill="blue">
    <circle cx="50" cy="14" r="14" />
    <path fill="none"
          d="M0,0 L50,40 50,60 0,100 M100,0 L50,40 M100,100 L50,60" />
  </symbol>
  <rect x="50" y="50" width="100" height="100" fill="wheat" stroke="black" />
  <use xlink:href="#icon" x="50" y="50" width="100" height="100" />
</svg>
```

Nous obtenons l'image suivante.

Figure 3-16. Un symbole

Ce symbole est constitué d'un cercle – la tête – et d'un élément 'path' – le corps et les membres. Nous affichons ce symbole avec un élément `<use>` qui définit une zone rectangulaire de 100 unités sur 100 unités. Nous avons un élément <rect> pour montrer le périmètre de la zone utilisée par ce symbole.

L'attribut `viewBox` établit un nouveau système de coordonnées, dans lequel seront définis les éléments graphiques qui constituent le symbole. Nous discuterons en détail l'attribut `viewBox` au chapitre 6. Remarquons simplement que le système défini par l'attribut `viewBox` recouvre exactement le rectangle défini dans l'élément `use`.

2. Jouons sur les dimensions données à 'use'. Elargissons ce rectangle de 50 unités

```
<symbol id="icon" viewBox="0 0 100 100" preserveAspectRatio="none"
…
<use xlink:href="#icon" x="50" y="50" width="150" height="100" />
```

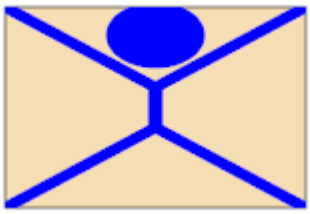

Figure 3-17. Le symbole dans un rectangle élargi

Comme vous le remarquez, le contenu graphique du symbole est déformé, élargi, pour remplir complètement le rectangle alloué.

Pour éviter cette déformation, SVG nous permet de jouer sur les valeurs de l'attribut `preserveAspectRatio`.

3. Utilisons une autre valeur pour `preserveAspectRatio`

```
<symbol id="icon" viewBox="0 0 100 100" preserveAspectRatio="xMidyMid meet"
  …
  <use xlink:href="#icon" x="50" y="50" width="150" height="100" />
```

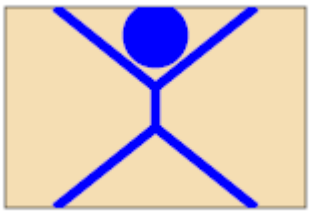

Figure 3-18. "xMidyMid meet" pour preserveAspectRatio

Si nous donnons à `preserveAspectRatio` une autre valeur que `"none"` le symbole ne sera pas déformé.

La valeur `"xMidyMid"` indique que les milieux du symbole et du rectangle défini par `viewBox` coïncident. La seconde valeur `"meet"` indique que tous les éléments graphiques seront dessinés.

4. Remplaçons "meet" par "slice"

```
<symbol id="icon" viewBox="0 0 100 100" preserveAspectRatio="xMidyMid slice"
  …
  <use xlink:href="#icon" x="50" y="50" width="150" height="100" />
```

Figure 3-19. "xMidyMid slice" pour preserveAspectRatio

Cette valeur “`slice`” provoque un agrandissement du symbole sans déformation pour que le rectangle défini par l'attribut 'viewBox' contienne entièrement le rectangle défini par 'use'.

Nous pouvons combiner toutes ces différentes valeurs pour `preserveAspectRatio` . Vous trouverez dans les spécifications SVG une explication et une illustration de ces différentes combinaisons.

Un point de vue

Je pense qu'il y a peu de chance que vous confondiez le symbole avec un groupe. Concevez un élément 'symbol' comme une zone rectangulaire où vous pouvez dessiner. Ensuite vous projetez ce dessin sur un autre rectangle pour qu'il soit rendu.

Personnellement, je n'utilise pas souvent cet élément. Cependant il est utile de le connaître car les éléments `<pattern>` et `<marker>` utilisent les mêmes concepts.

Enfin, je vous recommande de définir un élément 'symbol' dans une section `<defs>` pour rester cohérent avec notre maxime *tout de qui est réutilisé par d'autres éléments doit être dans une section* `<defs>`.

L'élément 'pattern'

Revenons à notre exemple de rack. Peut-être pouvons nous améliorer l'apparence des montants? Les montants ne sont pas dans la réalité de simples rectangles. Ils présentent plutôt des trous pour les alléger.

Hmm, l'élément 'pattern' pourrait nous être utile. Les spécifications nous disent:

> *Un élément 'pattern' est utilisé pour remplir ou tracer un objet en utilisant un graphique pré-défini qui est répété à intervalles réguliers en x et en y pour couvrir toute la zone concernée.*

Voyons la syntaxe de cet élément 'pattern'

Syntaxe

```
<pattern id="name"
         x="coordinate"
         y="coordinate"
         width="length"
         height="length"
         patternUnits="userSpaceOnUse"
         style-attribute="style-attribute"
>
  <!-- contenu pour l'élément 'pattern' -->
</pattern>
```

Avec cet élément `<pattern>` nous avons un autre container. Il est utile de le représenter comme un pavage régulier avec des rectangles.Voyons un exemple.

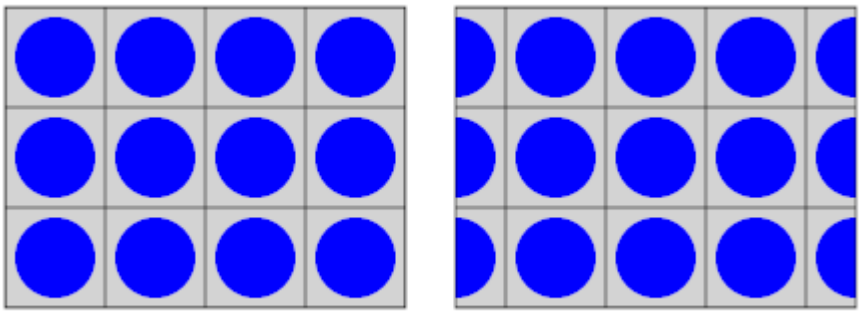

Figure 3-20. Deux rectangles remplis avec un motif

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <pattern id="holes" x="0" y="0" width="50" height="50"
             patternUnits="userSpaceOnUse">
      <rect x="0" y="0" width="50" height="50" stroke="black"
            fill="lightgray" />
      <circle cx="25" cy="25" r="20" fill="blue" />
    </pattern>
  </defs>
  <rect x="50" y="50" width="200" height="150" fill="url(#holes)"
        stroke="black" />
  <rect x="275" y="50" width="200" height="150" fill="url(#holes)"
        stroke="black" />
</svg>
```

Un élément du pavage est formé d'un carré gris de 50 unités de côté avec un cercle bleu centré dans ce carré. Quand nous remplissons les deux rectangles de 200 unités sur 150 unités avec ce pavage, nous sommes un peu surpris du résultat.

Ceci est dû au fait que le premier carré commence à l'origine du système de coordonnées qui est aussi le coin supérieur gauche du premer rectangle. Pour vérifier ceci, décalons les deux rectangles vers la droite.

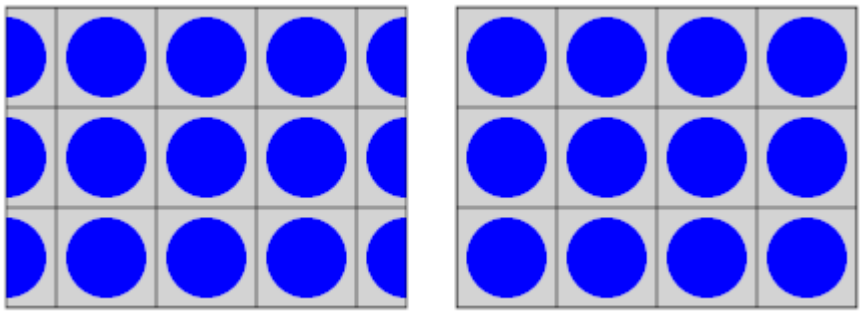

Figure 3-21. Les deux rectangles décalés

```
<rect x="75" y="50" width="200" height="150" fill="url(#holes)"
      stroke="black" />
<rect x="300" y="50" width="200" height="150" fill="url(#holes)"
      stroke="black" />
```

Comme prévu, les carrés ne couvrent pas de la même manière nos rectangles, mais ce n'est pas ce que nous souhaitions. Pour que les carrés coïncident avec les deux rectangles, il faudrait que les carrés soient dessinés dans un système de coordonnées spécifique à chaque rectangle. Nous pouvons également créer un groupe, et le réutiliser deux fois.

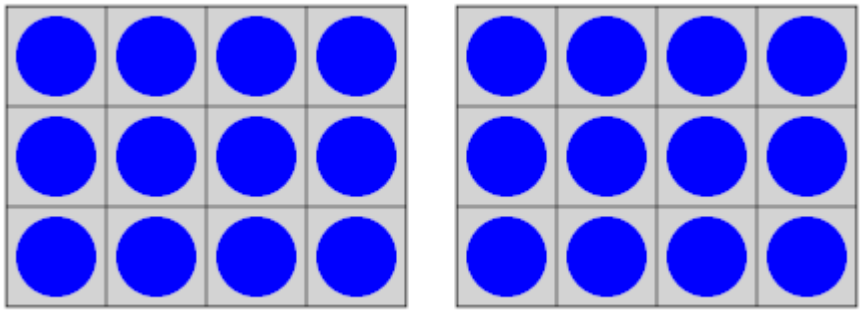

Figure 3-22. Le même rectangle réutilisé

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <pattern id="holes" width="50" height="50" patternUnits="userSpaceOnUse">
      <rect x="0" y="0" width="50" height="50" stroke="black"
            fill="lightgray" />
      <circle cx="25" cy="25" r="20" fill="blue" />
    </pattern>
    <g id="rect">
      <rect width="200" height="150" fill="url(#holes)" stroke="black" />
    </g>
  </defs>
  <use xlink:href="#rect" x="50" y="50" />
  <use xlink:href="#rect" x="275" y="50" />
</svg>
```

L'origine de 'pattern' peut-être déplacée en donnant des valeurs aux attributs `x` et `y`.

```
<pattern id="holes" x="10" y="10" width="50" height="50"
         patternUnits="userSpaceOnUse">
```

Nous pourrions également changer la valeur de l'attribut 'patternUnits', nous verrons ceci au chapitre 7.

Nous pouvons maintenant ajouter des trous à nos montants.

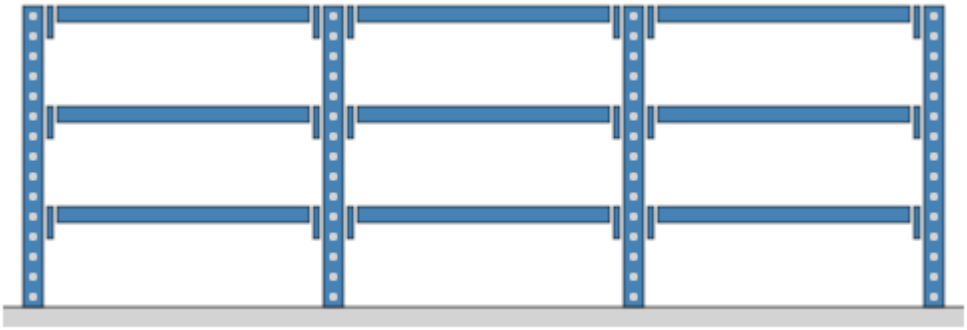

Figure 3-23. Rack avec des montants à trous

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <pattern id="holes" width="10" height="10" patternUnits="userSpaceOnUse">
      <rect width="10" height="10" stroke="none" fill="steelblue" />
      <circle cx="5" cy="5" r="2" fill="lightgray" />
    </pattern>
    <g id="montant">
      <rect width="10" height="150" fill="url(#holes)" stroke="black" />
    </g>
    <g id="colonne">
      <defs>
        <g id="support">
          <rect x="0" width="3" height="16" />
          <rect x="5" width="126" height="8" />
          <rect x="133" width="3" height="16" />
        </g>
      </defs>
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
    <g id="rack" fill="steelblue" stroke="black" >
      <g id="sol">
        <rect x="-10" y="150" width="480" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-10" y1="150" x2="470" y2="150" stroke="black" />
      </g>
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
      <use xlink:href="#colonne" x="150" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
</svg>
```

L'élément 'marker'

Très contents de ce nouveau look pour le rack: *“ Et si nous pouvions indiquer les dimensions dans notre brochure?.”*

Utilisons l'élément 'marker'. Les spécifications SVG précisent:

> *Un élément 'marker' est un symbole lié à un ou plusieurs points d'un élément ‘path’, ’line’, ’polyline’ ou ‘polygon’.*

Ceci semble intéressant. Voici la syntaxe pour l'élément `<marker>`.

Syntaxe

```
<marker id="name"
        refX="coordinate"
        refY="coordinate"
        markerWidth="length"
        markerHeight="length"
        markerUnits="strokeWidth | userSpaceOnUse"
        viewBox="min-x min-y width height"
        orient="auto | angle"
        style-attribute="style-attribute"
>
  <!-- contenu pour l'élément 'marker' -->
</marker>
```

L'élément `<marker>` est lui aussi un élément container. Comme nous voulons dessiner des flèches aux extrémités de nos lignes, nous allons définir la forme de ces flèches en utilisant un élément 'path'.

```
<path d="M0,0 L20,-4 20,4 z M0,-10 L0,10" stroke="black" />
```

Pour intégrer cette forme à notre élément 'marker', définissons la en détail.

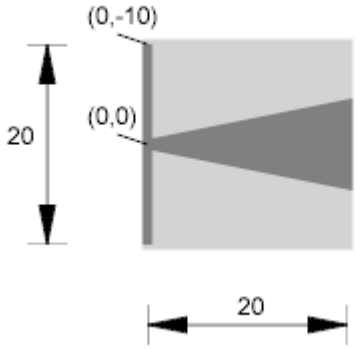

Figure 3-24. Une flèche détaillée

```
<marker id="arrow" viewBox="0 -10 20 20" markerUnits="strokeWidth"
        markerWidth="20" markerHeight="20" orient="auto">
  <path d="M0,0 L20,-4 20,4 z M0,-10 L0,10" stroke="black" />
</marker>
```

Comme souvent avec SVG il y a plusieurs manières de procéder. Examinons ma préférée.

Voyons les détails du graphique. L'origine de notre système de coordonnées pour l'élément 'marker' est repérée par (0,0). C'est aussi le point du marqueur qui sera placé au début ou à la fin de nos lignes. Nous donnons les dimensions du rectangle du tracé et l'angle supérieur gauche de ce tracé.

1. Nous complétons l'attribut `viewBox` avec les coordonnées du coin supérieur gauche et les dimensions du rectangle de tracé.
2. Nous donnons à l'attribut `markerWidth` comme valeur la largeur de notre tracé.
3. De même pour l'attribut `markerHeight` nous donnons comme valeur la hauteur de notre tracé.
4. Nous définissons le contenu de `<marker>` en utilisant les coordonnées dans le système propre au marqueur..

Maintenant que nous savons définir des marqueurs, nous pouvons les placer sur des éléments 'path', 'line', 'polyline', etc..
Pour celà, ces éléments supportent les attributs `marker-start`, `marker-mid`, et `marker-end`. Avec ces attributs nous pouvons placer des marqueurs aux points de départ, de fin et aux milieux de nos objets.

Voici donc une ligne avec une flèche à chaque extrémité.

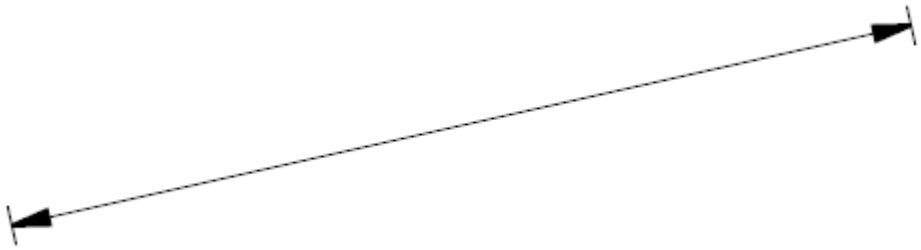

Figure 3-25. Ligne fléchée

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="startArrow" viewBox="0 -10 20 20"
            markerUnits="strokeWidth"
            refX="0" refY="0" markerWidth="20" markerHeight="20"
            orient="auto">
      <path d="M0,0 L20,-4 20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
    <marker id="endArrow" viewBox="-20 -10 20 20" markerUnits="strokeWidth"
            refX="0" refY="0" markerWidth="20" markerHeight="20"
            orient="auto">
      <path d="M-20,-4 L0,0 -20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
  </defs>
  <line x1="50" y1="150" x2="500" y2="50" stroke="black"
        marker-start="url(#startArrow)" marker-end="url(#endArrow)" />
</svg>
```

Comme les flèches de début et de fin peuvent avoir différentes directions, nous devons définir deux marqueurs différents `"startArrow"` et `"endArrow"`. Nous utilisons l'attribut `orient="auto"` pour que les marqueurs soient automatiquement alignés avec la direction des lignes.

Nous pouvons enfin faire apparaître les dimensions de notre rack.

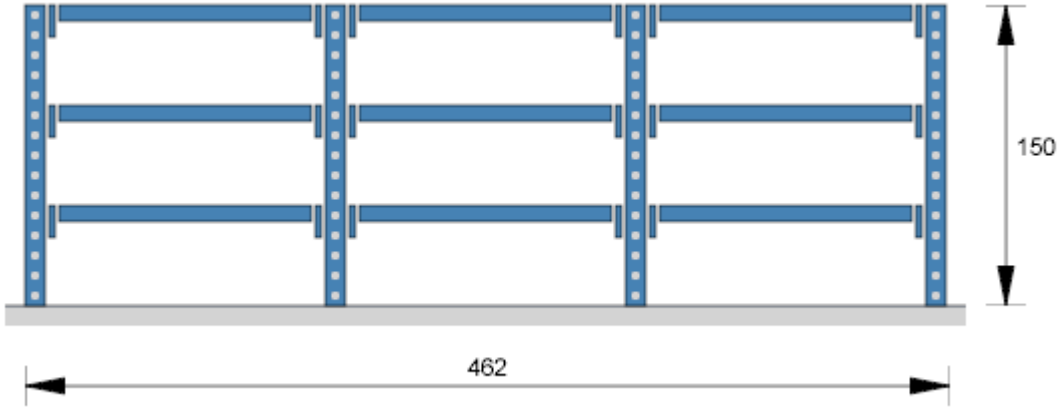

Figure 3-26. Rack et ses dimensions

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
<svg>
  <defs>
    <marker id="startArrow" viewBox="0 -10 20 20"
            markerUnits="strokeWidth" refX="0" refY="0" markerWidth="20"
            markerHeight="20" orient="auto">
      <path d="M0,0 L20,-4 20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
    <marker id="endArrow" viewBox="-20 -10 20 20" markerUnits="strokeWidth"
            refX="0" refY="0" markerWidth="20" markerHeight="20"
            orient="auto">
      <path d="M-20,-4 L0,0 -20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
    <!-- éléments du rack -->
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
  <g id="dimensions">
    <line x1="50" y1="210" x2="512" y2="210" stroke="black"
          marker-start="url(#startArrow)" marker-end="url(#endArrow)" />
    <text x="281" y="205" text-anchor="middle">462</text>
    <line x1="540" y1="20" x2="540" y2="170" stroke="black"
          marker-start="url(#startArrow)" marker-end="url(#endArrow)" />
    <text x="545" y="95">150</text>
  </g>
</svg>
```

Dans ce cas, comme nous voulons que les marqueurs soient proportionnels à l'épaisseur du tracé de nos lignes, nous utilisons `markerUnits="strokeWidth"`.

Visibilité des groupes

Nous ne voulons pas toujours que les éléments de notre document soient tous visibles au même moment. Pour contrôler la visibilité de certains éléments nous changeons la structure de notre document. Nous constituons des couches qui contiennent les éléments liés par la logique.

Pour ce faire, l'élément <g> est idéal. Nous regroupons dans un groupe tous les élément dont la visibilité peut changer avec les interventions de l'utilisateur.

SVG ou plus exactement CSS2 nous fournit deux attributs de présentation `display` et `visibility` pour contrôler la visibilité des éléments. Je vous conseille d'utiliser l'attribut `visibility` avec un groupe pour contrôler cette visibilité. Nous pouvons alors modifier la valeur de l'attribut `visibility` pour les éléments de ce groupe.

Au chapitre 11.5 des recommandations SVG, vous pouvez en savoir plus sur la différence entre les attributs `display` et `visibility`.

Nous désirons contrôler la visibilité des dimensions du rack. Nous devons

1. Ajouter l'attribut `visibility` au groupe.
2. Nous assurer qu'aucun des éléments du groupe n'a son propre attribut `visibility` défini.

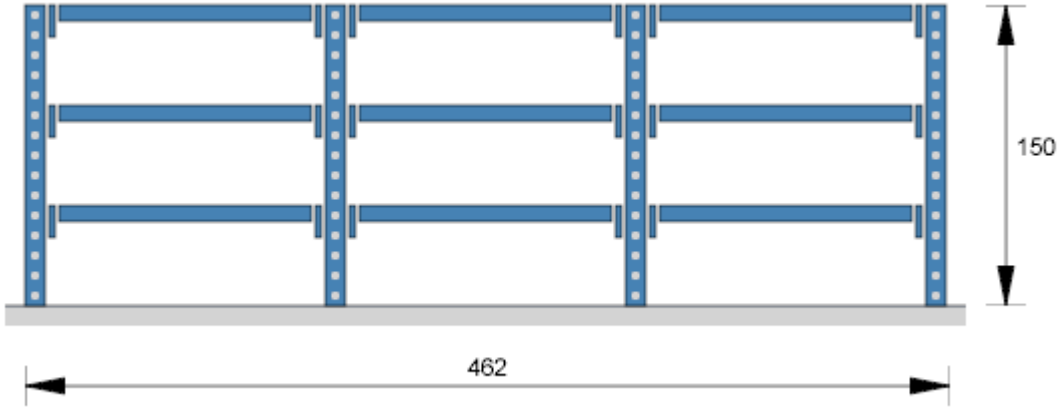

Figure 3-27. Montrer/cacher les dimensions du rack

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/SVG/DTD/svg10.dtd">
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="startArrow" viewBox="0 -10 20 20"
            markerUnits="strokeWidth" refX="0" refY="0" markerWidth="20"
            markerHeight="20" orient="auto">
      <path d="M0,0 L20,-4 20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
    <marker id="endArrow" viewBox="-20 -10 20 20" markerUnits="strokeWidth"
            refX="0" refY="0" markerWidth="20" markerHeight="20"
            orient="auto">
      <path d="M-20,-4 L0,0 -20,4 z M0,-10 L0,10" stroke="black" />
    </marker>
    <pattern id="holes" width="10" height="10" patternUnits="userSpaceOnUse">
      <rect width="10" height="10" stroke="none" fill="steelblue" />
      <circle cx="5" cy="5" r="2" fill="lightgray" />
    </pattern>
    <g id="montant">
      <rect width="10" height="150" fill="url(#holes)" stroke="black" />
    </g>
    <g id="colonne">
      <defs>
        <g id="support">
          <rect x="0" width="3" height="16" />
          <rect x="5" width="126" height="8" />
          <rect x="133" width="3" height="16" />
        </g>
      </defs>
      <use xlink:href="#support" x="12" y="0" />
      <use xlink:href="#support" x="12" y="50" />
      <use xlink:href="#support" x="12" y="100" />
      <use xlink:href="#montant" x="150" y="0" />
    </g>
    <g id="rack" fill="steelblue" stroke="black" >
      <g id="sol">
        <rect x="-10" y="150" width="480" height="10" stroke="none"
              fill="lightgray" />
        <line x1="-10" y1="150" x2="470" y2="150" stroke="black" />
      </g>
      <use xlink:href="#montant" x="0" y="0" />
      <use xlink:href="#colonne" x="0" y="0" />
```

```
      <use xlink:href="#colonne" x="150" y="0" />
      <use xlink:href="#colonne" x="300" y="0" />
    </g>
  </defs>
  <use xlink:href="#rack" x="50" y="20" />
  <g id="dimensions" visibility="visible">
    <line x1="50" y1="210" x2="512" y2="210" stroke="black"
          marker-start="url(#startArrow)" marker-end="url(#endArrow)" />
    <text x="281" y="205" text-anchor="middle">462</text>
    <line x1="540" y1="20" x2="540" y2="170" stroke="black"
          marker-start="url(#startArrow)" marker-end="url(#endArrow)" />
    <text x="545" y="95">150</text>
  </g>
</svg>
```

Avec ce document complet, non seulement vous avez une image pour votre brochure, mais aussi un document dynamique pour vos projets de site Web. Le visiteur de votre site pourra voir apparaître comme par magie les dimensions du rack en cliquant sur un bouton ou en passant son pointeur sur le rack.

Comment réaliser cette interactivité? Nous le verrons aux chapitres 9 et 10, avec des éléments d'animation ou un petit script.