Rendu SVG piloté par le réseau

Florent Becker

23 janvier 2020

1 Organisation du projet

Vous allez réaliser au cours des 8 prochaines semaines un projet par groupes de 3 étudiants·es. Le projet de cette année est un moteur de rendu svg avec la possibilité de piloter certaines données de l'image représentée par le fichier depuis le réseau.

L'organisation de votre groupe de travail est libre, **mais** il sera tenu compte dans votre évaluation de la façon dont vous vous êtes organisés *telle qu'elle transparait dans votre rendu*. Le rendu de votre projet se fera sous la forme d'un projet git dont vous enverrez l'url à `florent.becker@univ-orleans.fr` dès sa création.

Vous avancerez sur le projet pendant votre temps de travail personnel, ainsi que pendant les séances de TP prévues à cet effet dans votre emploi du temps.

Un des objectifs de ce projet est de vous faire travailler avec plusieurs bibliothèques, écrites en C et en C++. Vous pouvez réaliser votre projet avec les bibliothèques de votre choix (tant que le projet est en C ou C++). Une sélection de bibliothèques conseillées vous est donnée dans le sujet.

2 Introduction

Le format svg est un format vectoriel d'image fondé sur xml. Il s'agit du format d'image vectoriel le plus commun pour la publication web. Une image svg est un document xml contenant des éléments géométriques (rectangles, ellipses, courbes...) qui peuvent être groupés (à l'aide de l'élément `<g>` ou transformés géométriquement.

Voici par exemple un document svg représentant une maison vue par un enfant de trois ans.

```
<svg width="100" height="100">

  <g id="sol">
    <rect x="0" y="80" width="100" height="20" fill="green"/>
  </g>

  <g id="ciel">
```

```
    <rect x="0" y="0" width="100" height="80" fill="blue"/>
    <g id="soleil">
      <circle cx="35" cy="35" r="20" stroke="yellow" fill="orange" stroke-width="5"/>
    </g>
  </g>


  <g id="maison">
    <rect x="60" y="65" width="30" height="23" fill="gray"/>
    <polyline points="60 65 75 55 90 65"
      stroke="black" fill="red" stroke-width="1"/>
  </g>

</svg>
```

Dans ce document svg, la position du soleil ($cx = 50, cy = 25$) correspond à 12h (à l'heure solaire). En revanche, à 7h, le soleil est plutôt à la position ($cx = 20, cy = 45$). L'objet de ce projet est d'avoir une application de rendu svg dans laquelle on puisse modifier la position du soleil pour le faire se déplacer en temps réel dans le ciel.

Pour cela, on va introduire un élément `driven`, qui ne fait pas partie de la norme svg. Voici un exemple d'image svg utilisant cet élément `driven`. L'élément `driven` contient deux attributs :

— `target` est le nom de l'attribut dont on contrôle la valeur. Cet attribut est pris parmi les attribut de l'élément père de `driven`,

— `by` est le nom de l'entrée utilisée pour contrôler cet attribut.

L'application que vous réaliserez devra ensuite écouter sur une socket udp (port 6789) des messages indiquant quelles valeurs donner aux attributs liés par `driven`.

```
<svg width="100" height="100">

  <g id="sol">
    <rect x="0" y="80" width="100" height="20" fill="green"/>
  </g>

  <g id="ciel">
    <rect x="0" y="0" width="100" height="80" fill="blue"/>
    <g id="soleil">
      <circle cx="50" cy="25" r="20" stroke="yellow" fill="orange" stroke-width="5">
<driven target="cx" by="sun_x"/>
<driven target="cy" by="sun_y"/>
      </circle>
    </g>
  </g>
```

```
  <g id="maison">
    <rect x="60" y="65" width="30" height="23" fill="gray"/>
    <polyline points="60 65 75 55 90 65"
      stroke="black" fill="red" stroke-width="1"/>
  </g>

</svg>
```

Si on ouvre cet image avec un logiciel svg classique, par exemple inkscape, on obtient le même affichage que précédemment. En revanche, puisqu'il y a deux éléments `driven` (dans le groupe `#soleil`) qui dépendent des paramètres `sun_x` et `sun_y`, notre application de rendu pilotable va attendre dessus des messages de la forme `{ "sun_x" : 17.9 }` ou `{"sun_y": 23.4, "sun_x": 32.1}`. À la réception de chacun de ces messages, elle ajustera la position du soleil.

On aura ainsi la possibilité, en connectant via la socket udp des capteurs bien choisis, de réaliser des animations de la température, du volume sonore, du trafic réseau ou encore de la charge processeur de la machine.

3 Les bibliothèques recommandées

Votre application sera une application graphique, qui affiche un document dont le chemin est passé en argument sur la ligne de commande. Si votre application s'appelle `tartempion_viewer`, la commande

```
tartempion_viewer toto.svg
```

doit ouvrir une fenêtre et afficher le document toto.svg. La bibliothèque `gtk` vous donne permet d'ouvrir une fenêtre avec, mais ça n'est pas nécessaire pour le moment, des boutons, menus... Cette bibliothèque `gtk` s'accompagne de la bibliothèque `cairo` qui vous donne un canevas sur lequel dessiner. La bibliothèque `librsvg` permet de prendre un document svg (depuis un fichier ou une chaîne de caractères) et de l'afficher sur une surface Cairo.

Pour que votre application fonctionne, vous aurez besoin de manipuler le document `svg` qui vous est donné en entrée en fonction des messages qui sont reçu. Plutôt que de tenter de modifier à la main le contenu du fichier `svg` à chaque mise à jour, vous pouvez utiliser une bibliothèque XML telle que `tinyxml-2` pour accéder aux éléments et à leurs attributs.

Une application `ex_gtk_svg` qui affiche un document svg dans une fenêtre gtk vous est fourni comme point de départ.

4 Les entrées

Pour que les images affichées par votre application soient pilotables, il faut que celle-ci accepte des entrées. Vous implémenterez obligatoirement un type d'entrée générique que

sont les sockets udp. Votre application devra donc écouter en udp sur le port 6789 et réagir aux messages entrants en animant le dessin qu'elle affiche. Naturellement, il faut pour cela que les messages reçus aient une forme correcte.

4.1 Pilotage réseau : le format de messages cbor

Les messages que vous recevrez sur la socket seront au format cbor (défini sur `https://cbor.io/`). Ce format est une sorte de json binaire. Il permet en particulier de délimiter simplement la taille des champs dans les données reçues en entrée. L'utilisation d'une bibliothèque telle que `libcbor` ou `cbor11` est recommandée.

Les messages cbor accepté seront des dictionnaires («map») dont les clés sont les noms des entrées (c'est à dire les attributs `by` des éléments `driven` cible), et dont les valeurs sont les valeurs à leur affecter.

5 Améliorations possibles

5.1 Gestion des types des attributs

Chaque attribut en svg a un type (par exemple, une `opacity` doit être un nombre entre 0 et 1). Si un message est envoyé qui donne une valeur à cet attribut qui n'est pas du bon type, votre application doit ignorer ce message.

5.2 Gestion de l'attribut style

L'attribut `style` est un attribut composite, qui comprend la couleur de fond, la couleur de bordure, l'opacité... Définir une syntaxe pour rendre chacun des sous-attributs de `style` pilotable.

5.3 Animation

Quand la valeur d'un attribut change parce qu'un message est reçu, cela peut provoquer un mouvement de grande amplitude (ou un changement de couleur immédiat). Ajouter dans l'élément `driven` un attribut `delay` qui indique en combien de temps l'attribut doit passer de sa valeur actuelle à la nouvelle valeur. Afficher pendant la transition une animation aussi fluide que possible.

5.4 Valeurs calculées

Il peut être intéressant d'utiliser pour les valeurs des attributs non pas directement celles qui sont reçues dans les messages, mais des valeurs calculées à partir de celles-ci. Donner la possibilité d'utiliser des opérations dans les éléments driven, notamment :

— Les opérations arithmétiques : `<driven target="opacity" by="input1 - input2"/>`
— Les fonctions flottantes de base : log, fonctions trigonométriques
— La longueur (pour une chaîne de caractères)
— min, max, abs, sign

— des conditionnelles

6 Grille (indicative) d'évaluation

Les éléments suivants participeront à l'évaluation de votre projet :
— Votre application fonctionne correctement
— Un manuel d'utilisation est fourni
— Les améliorations proposées sont implémentées
— Un jeu de test est fourni (et le comportement sur ces tests est correct)
— Votre application donne des messages d'erreurs compréhensibles si un fichier ou un message réseau est incorrect
— La compilation de votre application se fait par un Makefile
— Le dépôt git montre des contributions régulières de chaque membre du projet
— Le dépôt git contient les documents qui vous ont permis d'organiser le travail