Język opisu sceny SVG

March 10, 2019

1 Introduction

Celem jest stosowanie języka SVG przy opracowaniu grafiki 2D.

SVG to raczej *język opisu sceny* niż język programowania. Kiedy język programowania tworzy scenę, generując jej treść w sposób proceduralny, język opisu sceny określa scenę "deklaratywnie", wymieniając jej zawartość. Ponieważ SVG jest *językiem grafiki wektorowej*, zawartość sceny obejmuje kształty, atrybuty, takie jak kolor i szerokość linii oraz transformacje geometryczne.

SVG jest językiem XML, co oznacza, że ma bardzo ścisłą i dość obszerną składnię.

2 Struktura dokumentu SVG

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        width="4in" height="4in"
        viewBox="0 0 400 400"
        preserveAspectRatio="xMidYMid">

    <!-- The scene description goes here!  -->

</svg>
```

Składnia elementa XML

```
<elementname attrib1="value1" attrib2="value2">
      ...content...
</elementname>
```

"self-closing tag"

```
<circle cx="5" cy="5" r="4" fill="red"/>
```

Przykład dokumentu SVG

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     width="300px" height="200px"
     viewBox="0 0 3 2"
     preserveAspectRatio="xMidYMid">

<rect x="0" y="0" width="3" height="2"
        stroke="blue" fill="none" stroke-width="0.05"/>
<text x="0.2" y="0.5" font-size="0.4" fill="red">Hello World!</text>
<line x1="0.1" y1="0.7" x2="2.9" y2="0.7"
        stroke-width="0.05" stroke="blue"/>
<ellipse cx="1.5" cy="1.4" rx=".6" ry=".4" fill="rgb(0,255,180)"/>
<circle cx="0.4" cy="1.4" r="0.3"
    fill="magenta" stroke="black" stroke-width="0.03"/>
<polygon points="2.2,1.7 2.4,1 2.9,1.7"
    fill="none" stroke="green" stroke-width="0.02"/>

</svg>
```

3 Kształty, style oraz przekształcenia

W SVG podstawowy kształt jest określony przez element, w którym nazwa znacznika nadaje kształt, a atrybuty dają właściwości kształtu. Istnieją atrybuty określające geometrię, na przykład punkty końcowe linii lub promień okręgu. Inne atrybuty określają właściwości stylu, takie jak kolor wypełnienia i szerokość linii.

Element

```
<rect width = "3" height = "2" />
```

określa prostokąt z rogiem na (0,0), szerokości 3 i wysokości 2, natomiast

```
<rect x = "100" y = "200" height = "480" width = "640" />
```

daje prostokąt z rogiem na (100, 200), szerokości 640 i wysokości 480. (Uwaga, atrybuty w elemencie XML mogą być podane w dowolnej kolejności.)

Element `rect` ma również opcjonalne atrybuty `rx` i `ry`, które może być użyty do "roundRectów", a ich rogi zastąpione łukami eliptycznymi. Wartości `rx` i `ry` podają poziome i pionowe promienie łuków eliptycznych.

Oto kilka typowych atrybutów stylu:

- `fill` - określa sposób wypełniania kształtu. Wartość może być "none", co oznacza, że kształt nie jest wypełniony. Może to być kolor, w tym

samym formacie, co kolory CSS używane w interfejsie API HTML canvas. Na przykład, może to być zwykła nazwa koloru, na przykład "black" lub "red", lub kolor RGB, taki jak "rgb (255,200,180)". Są również wypełnienia gradientem i wzorami, ale nie będziemy ich tutaj omawiać.

- `stroke` - określa, jak obrysować kształt, z takimi samymi możliwymi wartościami jak "fill".
- `stroke-opacity` i `fill-opacity` - to liczby od 0,0 do 1,0 określające przezroczystość obrysu i wypełnienia. Wartości mniejsze niż 1,0 dają półprzezroczyste obrysy lub wypełnienia. Wartość domyślna 1,0 oznacza całkowicie nieprzezroczysty.
- `stroke-width` - to liczba określająca szerokość linii używaną podczas obrysu. Zwróć uwagę, że szerokość linii podlega przekształceniom. Wartością domyślną jest "1", co oznacza, że układ współrzędnych wykorzystuje piksele jako jednostkę miary, ale często jest zbyt szeroki w niestandardowych układach współrzędnych.
- `stroke-linecap` - określa wygląd punktów końcowych obrysu. Wartość może być "square", "round" lub "butt". Wartością domyślną jest "butt".
- `stroke-linejoin` - określa wygląd punktów, w których spotykają się dwa segmenty pociągnięcia. Wartości mogą być "miter", "round" lub "bevel". Wartością domyślną jest "miter".

Przykład

```
<rect x="-0.5" y="-0.5" width="1" height="1"
          rx="0.1" ry="0.1"
          fill="red" fill-opacity="0.5"
          stroke="gray" stroke-width="0.05" stroke-linejoin="round"/>
```

Atrybut `transform` może być użyty do zastosowania transformacji lub serii przekształceń do kształtu. Na przykład możemy zrobić prostokąt przechylony o 30 stopni od poziomu:

```
<rect width = "100" height = "50" transform = "rotate (30)" />
```

Wartość "rotate(30)" oznacza obrót o 30 stopni (nie radian!) o punkcie początkowym, (0,0).

Wartością atrybutu transformacji może być **lista transformacji** oddzielona spacjami lub przecinkami. Transformacje są stosowane do obiektu, jak zwykle, w przeciwieństwie do kolejności, w jakiej są wymienione. Więc,

```
<rect width = "100" height = "50"
transform = "translate (0,50) rotate (45) skewX (-30)" />
```

najpierw przekręciłby prostokąt w równoległobok, następnie obrócił równoległobok o 45 stopni wokół początku, a następnie przeniosł go o 50 jednostek w kierunku y.

Oprócz prostokątów, SVG ma linie, okręgi, elipsy i tekst jako podstawowe kształty. Oto kilka szczegółów.

- Element `<line>` reprezentuje segmentację linii i ma atrybuty geometryczne `x1, y1, x2` i `y2` w celu określenia współrzędnych punktów końcowych segmentu linii. Te cztery atrybuty mają zerową wartość domyślną, co ułatwia określenie linii poziomych i pionowych. Na przykład,

  ```
  <line x1 = "100" x2 = "300" stroke = "czarny" />
  ```

  Bez atrybutu obrysu nie zobaczysz linii, ponieważ domyślną wartością dla stroke jest "none".

- W przypadku elementu `<circle>` atrybuty geometryczne to `cx, cy` i `r`, podając współrzędne środka okręgu i promienia. Współrzędne środkowe mają domyślne wartości równe zeru.

- Dla elementu `<ellipse>` atrybuty to `cx, cy, rx` i `ry`, gdzie `rx` i `ry` podają promienie elipsy w kierunkach x i y.

- Element `<text>` jest trochę inny. Ma atrybuty `x` i `y`, z wartościami domyślnymi zero, aby określić położenie punktu bazowego tekstu. Jednak sam tekst jest podany jako treść elementu, a nie jako atrybut. Oznacza to, że element jest podzielony na znacznik początkowy i znacznik końcowy, a tekst, który pojawi się na rysunku, znajduje się między znacznikami początkowym i końcowym. Na przykład,

  ```
  <text x = "10" y = "30"> Ten tekst pojawi się na obrazku </ text>
  ```

  Zwykłe atrybuty `stroke` i `fill` odnoszą się do tekstu, ale tekst ma dodatkowe atrybuty stylu. Atrybut `font-family` określa samą czcionkę. Jego wartością może być jedna z ogólnych nazw czcionek `"serif", "sans-serif", "monospace"` lub nazwa określonej czcionki, która jest dostępna w systemie. Rozmiar czcionki (font-size) może być liczbą określającą (przybliżoną) wysokość znaków w układzie współrzędnych. (Rozmiar czcionki podlega przekształceniom współrzędnych i modelowaniu jak każda inna długość.) Możesz uzyskać tekst pogrubiony i pochyły, ustawiając font-weight równą "bold" i font-style równy "italic". Oto przykład, który wykorzystuje wszystkie te opcje i stosuje pewne dodatkowe style i transformację dla dobrego pomiaru:

```
<text x = "10" y = "30"
    font-family = "sans-serif" font-size = "50"
    font-style = "italic" font-weight = "bold"
    stroke = "black" stroke-width = "1" fill = "rgb(255,200,0)"
    transform = "rotate(20)"> Hello World </ text>
```

4 Wielokąty i ścieżki

SVG ma kilka fajnych funkcji do tworzenia bardziej złożonych kształtów.

- Element `<polygon>` ułatwia utworzenie wielokątu z listy par współrzędnych. Na przykład,

  ```
  <polygon points = "0,0 100,0 100,75 50,100 0,75" />
  ```

  tworzy pięciokątny wielokąt z wierzchołkami w (0,0), (100,0), (100,75), (50,100) i (0,75). Każda para liczb w atrybucie `points` określa wierzchołek. Liczby mogą być oddzielone spacjami lub przecinkami. Użyłem tutaj kombinacji spacji i przecinków, aby wyjaśnić, jak liczby się łączą.

- Element `<path>` jest o wiele bardziej interesujący. W rzeczywistości wszystkie inne podstawowe kształty, z wyjątkiem tekstu, mogą być wykonane przy użyciu elementów ścieżki. Ścieżka może składać się z odcinków linii, krzywych Beziera i łuków eliptycznych (chociaż tutaj nie będę omawiać łuków eliptycznych). Składnia określająca ścieżkę jest bardzo zwięzła i ma pewne cechy, których wcześniej nie widzieliśmy. Element ścieżki ma atrybut o nazwie `d` zawierający dane dotyczące ścieżki.

  Dane składają się z jednego lub więcej poleceń, gdzie każde polecenie składa się z pojedynczej litery, po której następują wszelkie dane niezbędne do wykonania polecenia. Polecenia moveTo, lineTo, cubic Bezier i quadratic Bezier, które są już znane, są kodowane przez litery `M, L, C, Q`. Polecenie do zamykania segmentu ścieżka jest `Z`, i nie wymaga żadnych danych. Na przykład, dane ścieżki `"M 10 20 L 100 200"` narysują odcinek linii od punktu (10,20) do punktu (100,200). Możesz połączyć kilka połączonych segmentów linii w jedno polecenie `L`. Na przykład powyższy przykład <polygon> może zostać utworzony przy użyciu elementu <path>

  ```
  <path d = "M 0,0 L 100,0 100,75 50,100 0,75 Z" />
  ```

  `Z` na końcu danych zamyka ścieżkę, dodając ostatnią stronę do wielokąta. (Zwróć uwagę, że jak zwykle możesz używać przecinków lub spacji w danych.)

  Polecenie `C` pobiera *sześć liczb* jako dane, aby określić dwa punkty kontrolne i końcowy punkt końcowy segmentu krzywej sześciennej Beziera.

Można również podać *wielokrotność sześciu wartości, aby uzyskać połączoną sekwencję segmentów krzywych.*

Podobnie, polecenie `Q` wykorzystuje *cztery wartości* danych do określenia punktu kontrolnego i końcowego punktu końcowego segmentu kwadratowej krzywej Beziera. Duży, krzywoliniowy, żółty kształt zostanie utworzony jako ścieżka z dwoma segmentami linii i dwoma segmentami krzywych Beziera:

```
<path
   d = "M 20,70 C 150,70 250,350 380,350
   L 380,380 C 250,380 150,100 20,100 Z"
   fill = "yellow" stroke-width = "2"
   stroke = "black" />
```

5 Modelowanie hierarchiczne

Potrzebujemy sposobu grupowania obiektów, aby mogły być traktowane jako jednostka. W tym celu SVG ma element `<g>`. Zawartość elementu `<g>` to lista elementów kształtu, które mogą być prostymi kształtami lub zagnieżdżonymi elementami `<g>`.

Możesz dodać atrybuty stylu i transformacji do elementu `<g>`. Głównym punktem grupowania jest to, że grupę można traktować jako pojedynczy obiekt. Atrybut `transform` w `<g>` przekształci całą grupę jako całość. Atrybut `style`, taki jak `fill` lub `font-family`, na elemencie `<g>` ustawi domyślną wartość grupy, zastępując bieżącą wartość domyślną. Oto przykład:

```
<g fill = "none" stroke = "black" stroke-width = "2"
                            transform = "scale(1, -1)">
      <circle r = "98" />
      <ellipse cx = "40" cy = "40" rx = "20" ry = "7" />
      <ellipse cx = "- 40" cy = "40" rx = "20" ry = "7" />
      <line y1 = "20" y2 = "- 10" />
      <path d = "M -40, -40 C-30, -50 30, -50 40, -40"
                                  stroke-width = "4" />
</g>
```

5.1 Wiele kopii obiektu w scenie

Załóżmy teraz, że chcemy uwzględnić wiele kopii obiektu w scenie. Nie powinno być konieczne powtarzanie kodu do rysowania obiektu. Byłoby miło mieć coś w rodzaju podprogramów wielokrotnego użytku. W rzeczywistości SVG ma coś bardzo podobnego: Możesz zdefiniować obiekty wielokrotnego użytku w elemencie `<defs>`. Obiekt zdefiniowany wewnątrz `<defs>` nie jest dodawany do sceny, ale kopie obiektu można dodać do sceny za pomocą pojedynczego polecenia. Aby to działało, obiekt musi mieć atrybut `id`, aby go zidentyfikować. Na przykład możemy zdefiniować obiekt, który wygląda jak znak plusa:

```
<defs>
   <g id = "plus" stroke = "black">
      <line x1 = "- 20" y1 = "0" x2 = "20" y2 = "0" />
      <line x1 = "0" y1 = "- 20" x2 = "0" y2 = "20" />
   </ g>
</ defs>
```

Element `<use>` może być następnie użyty do dodania kopii obiektu znaku plus do sceny. Składnia jest

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

Wartość atrybutu `xlink:href` musi być identyfikatorem obiektu, z dodanym na początku znakiem "". (Nie zapomnij o . Jeśli go opuścisz, element <use> zostanie po prostu zignorowany.)

Możesz dodać atrybut `transform` do elementu `<use>`, aby zastosować transformację do kopii obiektu. Można także zastosować atrybuty stylu, które będą używane jako wartości domyślne dla atrybutów w kopii. Na przykład możemy narysować kilka znaków plus z różnymi przekształceniami i szerokością obrysu:

```
<use xlink: href = "#plus" transform = "translate(50,20)"
                                    stroke-width = "5" />
<use xlink: href = "#plus" transform = "translate(0,30) rotate(45)" />
```

6 Animacja

Możliwe jest animowanie prawie każdej właściwości obiektu SVG, w tym geometrii, stylu i transformacji. Składnia animacji jest dość złożona. Tu omówimy tylko kilka przykładów.

- Wiele atrybutów elementu kształtu można animować, dodając element `<animate>` do zawartości elementu shape. Oto przykład, który sprawia, że prostokąt przesuwa się po obrazie od lewej do prawej:

  ```
  <rect x = "0" y = "210" width = "40" height = "40">
       <animate attributeName = "x"
       from = "0" to = "430" dur = "7s"
       repeatCount = "indefinite" />
  </ rect>
  ```

  Zauważ, że `<animate>` jest zagnieżdżony wewnątrz `<rect>`. Atrybut `attributeName` informuje, który atrybut animacji `<rect>` jest animowany, w tym przypadku `x`. Atrybuty `from` i `to` twierdzą, że `x` przyjmuje wartości od 0 do 430. Atrybut `dur` to "czas trwania", czyli czas trwania animacji; wartość "7s" oznacza "7 sekund". Atrybut `repeatCount` = "indefinite" oznacza, że po zakończeniu animacji rozpocznie się ona od nowa i będzie się powtarzać w nieskończoność, czyli tak długo, jak będzie wyświetlany

obraz. Jeśli atrybut `repeatCount` zostanie pominięty, to po uruchomieniu animacji jeden prostokąt przeskoczy z powrotem do pierwotnej pozycji i pozostanie tam. Jeśli parametr `repeatCount` zostanie zastąpiony przez `fill = "freeze"`, to po uruchomieniu animacji prostokąt zostanie zamrożony w ostatecznej pozycji, zamiast przeskoczyć z powrotem do pozycji wyjściowej. Animacja rozpoczyna się po pierwszym załadowaniu obrazu. Jeśli chcesz, aby animacja została uruchomiona później, możesz dodać atrybut `begin`, którego wartość określa czas rozpoczęcia animacji w kilka sekund po załadowaniu obrazu.

- keyframe animation:

```
<rect x="0" y="210" width="40" height="40">
     <animate attributeName="x"
     keyTimes="0;0.5;1" values="0;430;0" dur="7s"
     repeatCount="indefinite"/>
</rect>
```

- Transformacje mogą być również animowane, ale musisz użyć znacznika `<animateTransform>` zamiast `<animate>`, a musisz dodać atrybut `type`, aby określić transformację, którą animujesz, na przykład "rotate" lub "translate". Oto na przykład animacja transformacji zastosowana do grupy:

```
<g transform = "scale(0,0)">
     <animateTransform attributeName = "transform" type = "scale"
           from = "0,0" to = "0,4,0,7"
           begin = "3s" dur = "15s" fill = "freeze" />
     <rect x = "- 15" y = "0" width = "30" height = "40"
                           fill = "rgb (150, 100,0)" />
     <polygon points = "- 60,40 60,40 0,200" fill = "green" />
</ g>
```

Literatura

Uwaga!

Teoretyczne podstawy przekształceń geometrycznych umieszczone w prezentacji wykładu https://drive.google.com/drive/folders/0B0-jI5UeRsjcVExBTExOcXNlcGc

W języku angielskim

- książka interakcyjna: SVG http://math.hws.edu/graphicsbook/c2/s7.html (rozdział 2.7)

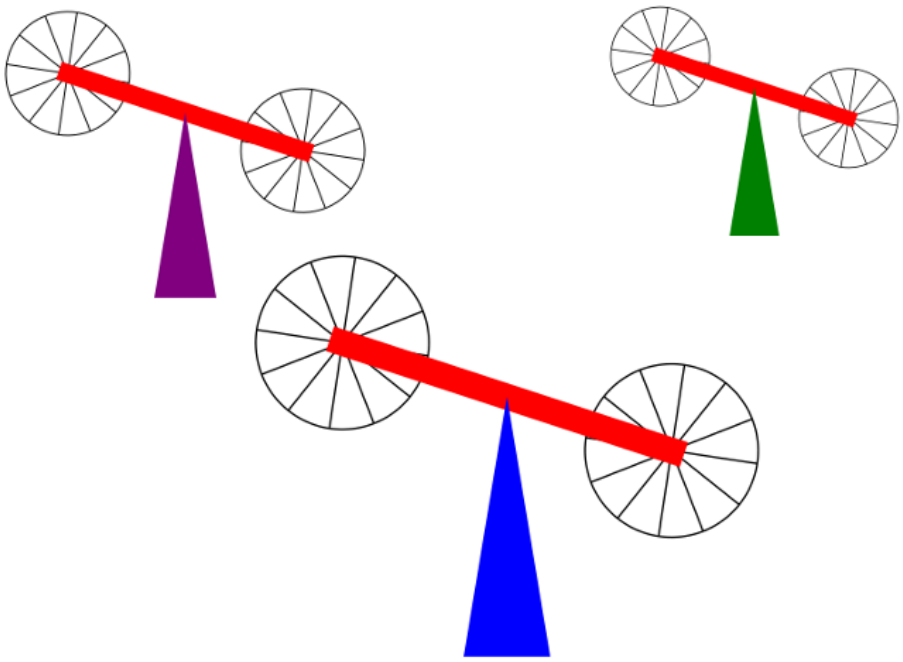

Figure 1: Obraz do zadania

7 Zadanie

Opracować scenę hierarchiczną zgodnie z obrazem używając zamiast kół wielokąty obracające się (animacja!) według wariantu. Opracowanie powinno być w języku SVG.