Projecto de programação II

Programação (L.EIC009)

Eduardo R. B. Marques, DCC/FCUP

Sumário

Neste projecto terá de definir um conversor de imagens do formato vectorial SVG para o formato “raster” PNG. Para o efeito terá de definir uma hierarquia de classes C++ correspondentes a elementos SVG e código de leitura do formato SVG.

O código a desenvolver compreende a definição da classe `svg::shape` para representar formas geométricas SVG, leitura de ficheiros SVG usando a biblioteca TinyXML2, e conversão para PNG.

É fornecido um projecto CMake para desenvolvimento com a estrutura detalhada abaixo neste documento, e uma versão com programas solução (binários) no Replit.

Exemplo de ficheiro SVG

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg":

    <circle cx="100" cy="100" r="95" fill="red"/>
    <circle cx="100" cy="100" r="80" fill="#A00000"/>
    <circle cx="50" cy="50" r="25" fill="white"/>
    <circle cx="50" cy="50" r="10" fill="black"/>
    <circle cx="150" cy="50" r="25" fill="white"/>
```

```
    <circle cx="150" cy="50" r="10" fill="black"/>
    <circle cx="100" cy="150" r="40" fill="white"/>
    <circle cx="100" cy="150" r="30" fill="black"/>
</svg>
```

Imagem resultante

Material de apoio

- Documentação da biblioteca TinyXML2: veja pasta `tinyxml2/docs` no projecto CMake.
- SVG tutorial
- Referência SVG (mozilla.org)
- Try It editor W3 Schools

Realização e entrega

O trabalho pode ser realizado individualmente ou em grupos de 2 alunos e ser entregue até **24 de Janeiro de 2022**.

1. Preencha o arquivo `README.md` identificando o grupo e fazendo um sumário das tarefas que conseguiu completar ou não.

2. Derive um arquivo ZIP para entrega contendo o ficheiro README.md e a pasta `svg` com o código fonte principal. Em Linux pode por exemplo executar a partir do directório raíz do projecto:

```
zip -9r delivery.zip README.md svg
```

3. Um formulário para submissão do arquivo ZIP será anunciado posteriormente.

Projecto CMake

Ficheiro/directório	Descrição
`CMakeLists.txt`	**Configuração do projecto CMake.**
`svg`	**Código fonte.**
`elements.hpp elements.cpp`	Elementos SVG - subclasses de `shape` (**a completar**)
`svg_to_png.hpp svg_to_png.cpp`	Leitura de SVG e conversão de SVG para PNG (**a completar**)
`svg.hpp`	Header file global para inclusão por programas (**código já fornecido que não deverá modificar**).
`shape.hpp shape.cpp`	Classe base para elementos SVG `svg::shape` (**código já fornecido que não deverá modificar**).
`png_image.hpp png_image.cpp`	Classe para imagens PNG `svg::png_image` (**código já fornecido que não deverá modificar**)

`color.hpp`	Representação de cores RGB `struct color` (**código já fornecido que não deverá modificar**)
`point.hpp`	Representação de pontos 2D `struct point` (**código já fornecido que não deverá modificar**)
`programs`	**Programas utilitários (poderão ser convenientes para desenvolvimento/debugging)**
`convert.cpp`	Converte imagem SVG em PNG.
`png_diff.cpp`	Compara duas imagens PNG.
`png_dump.cpp`	Lista valores de pixeis para uma imagem PNG.
`xmltest.cpp`	Ilustra uso da biblioteca TinyXML2.
`test`	**Testes unitários (usando GoogleTest).**
`test_ellipse.cpp`	Definição de `<ellipse>` e `<circle>`.
`test_polygon.cpp`	Definição de `<polygon>` e `<rect>`.
`test_polyline.cpp`	Definição de `<polyline>` e `<line>`.
`test_transform.cpp`	Testes para transformações `translate`, `scale` e `rotate`.
`test_group.cpp`	Testes para grupos de

	elementos (`<g>`)
`test_use.cpp`	Testes para duplicação de elementos (`<use>`)
`data`	**Pasta com exemplos de teste**
`data/input`	Ficheiros SVG de input
`data/output`	Directório para imagens PNG produzidas por testes unitários.
`data/expected`	Directório para imagens PNG de validação. As imagens geradas em `output` deverão ser equivalentes às imagens correspondentes neste directório.
`external`	**Bibliotecas auxiliares**
`external/gtest`	Código fonte da biblioteca GoogleTest, usada para programação de testes.
`external/stb`	Código fonte da biblioteca stb, usada para para ler e escrever imagens no formato PNG.
`external/tinyxml2`	Código fonte da biblioteca TinyXML2, usado para leitura de ficheiros SVG.

Critérios de avaliação

1. O seu código deverá estar correcto atendendo aos requisitos detalhados neste enunciado.

- Use os programas de teste fornecidos para validar a sua implementação.
- Além dos aspectos funcionais, o código não deverá apresentar deficiências no uso de memória, por exemplo em termos de “buffer overflows”, “dangling references”, ou “memory leaks”.

2. O seu código deverá estar bem estruturado e ser tão simples quanto possível.
3. O código que escrever **NÃO PRECISA** de ser documentado no formato Doxygen.
4. O código de todos os grupos será analisado por uma ferramenta de detecção automática de cópia. Trabalhos com similaridades para além da dúvida razoável serão anulados.

Fluxo de trabalho

Sugere-se o seguinte fluxo de trabalho para a implementação de código (ver detalhes no resto do texto):

1. Comece por definir suporte para `<circle>` até que os testes de `test_ellipse` passem todos (suporte para `<ellipse>` já é dado).
2. Proceda de forma similar para `<polygon>`/`<rect>`/`test_polygon` e `<polyline>`/`<line>`/`test_polyline`.
3. Habilite as transformações a elementos segundo o atributo `transform`. Teste o código com `test_transform`.
4. Habilite grupos de elementos (`<g>`) e teste o seu código com `test_group`.
5. Habilite duplicação de elementos e teste o seu código com `test_use`.

SVG

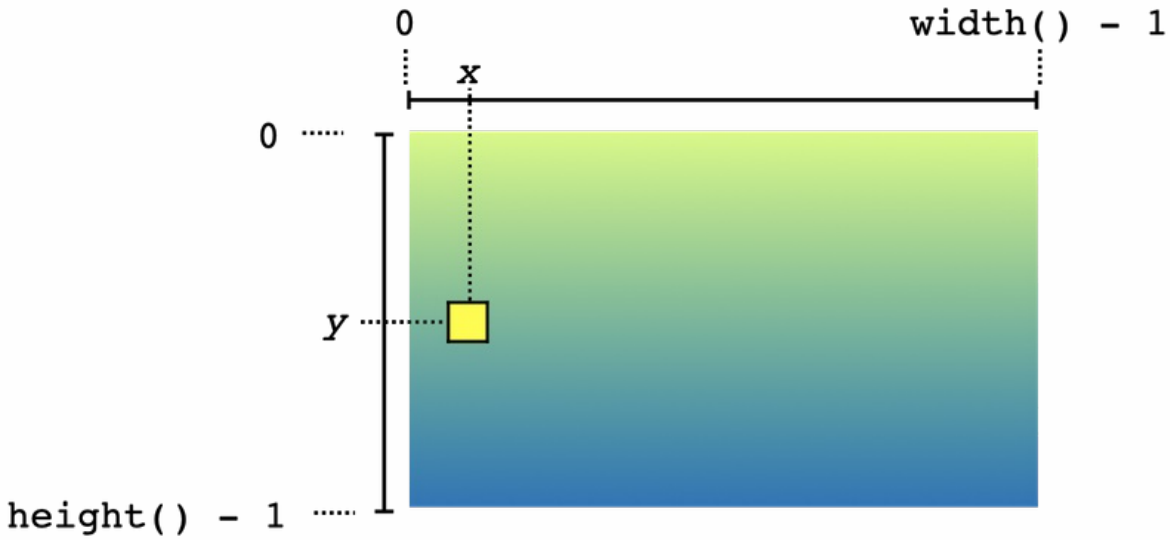
Faz-se a seguir um sumário do subconjunto do standard SVG a considerar para o projecto e descrição das tarefas associadas. Note-se que apenas **uma pequena**

parte da funcionalidade SVG é coberta pelo projecto e em alguns casos de forma aproximada.

Elemento raíz (`<svg>`)

Imagens SVG são expressas em XML por elementos `<svg>`. No projecto consideram-se (apenas) elementos do tipo:

```
<svg width="@valor_inteiro" height="@valor_inteiro">
  ...
</svg>
```



Os atributos `width` e `height` são valores inteiros que expressam respectivamente a largura e altura da imagem. Como no caso de outros formatos de imagem (ex. também PNG) a convenção é que o canto superior esquerdo da imagem tem coordenada `(0,0)` e o canto inferior direito tem coordenada `(width-1,height-1)`.

Nota: Nos exemplos fornecidos consta também o atributo `xmlns` apenas para o efeito de visualização de imagens SVG em browsers. O atributo não tem de resto qualquer impacto no desenvolvimento do trabalho. Sugere-se o uso do Firefox ou Chrome caso queira visualizar ficheiros SVG. O Safari tem alguns problemas na visualização de SVGs, e o mesmo acontece no CLion.

Suporte dado

Em `svg_to_png.cpp` encontra suporte inicial para leitura de ficheiros SVG e sua conversão para PNG:

- `svg_to_png()`: função de entrada para conversão de SVG para PNG;
- `parse_shapes()`: leitura de sub-elementos (formas) de `<svg>`;
- `parse_ellipse()`: leitura de elementos `ellipse` já dada;
- `parse_color()`: leitura de cores;
- `parse_transform()`: leitura de transformações.

Formas geométricas SVG

Formas e atributos a considerar

- `ellipse` - elipse (**código de suporte já fornecido**)
  - `cx` e `cy`: coordenadas do centro do círculo;
  - `rx` e `ry`: raios da elipse para cada eixo;
  - `fill`: cor de preenchimento;
- `circle` - círculo
  - `cx` e `cy`: coordenadas do centro do círculo;
  - `r`: raio;
  - `fill`: cor de preenchimento;
- `polyline` - linha formada por uma sequência de pontos
  - `points`: sequência de pontos;
  - `stroke`: cor da linha;
- `line` - segmento de recta definido por 2 pontos
  - `x1`, `y1` : coordenadas X e Y do primeiro ponto;
  - `x2`, `y2` : coordenadas X e Y do primeiro ponto;
  - `stroke`: cor da linha;
- `polygon` - polígono
  - `points`: sequência de pontos do polígono;

    - `fill`: cor da preenchimento;
- `rect` - rectângulo
    - `x` e `y`: coordenadas do canto superior esquerdo do rectângulo;
    - `width` e `height`: largura e altura do rectângulo;
    - `fill`: cor da preenchimento;

Adicionalmente e de forma opcional cada um dos elementos pode ter um atributo `id` em suporte a elementos `<use>` (duplicação de elementos; ver à frente).

Tarefas

- Tome como referência inicial o código já dado em `elements.hpp`, `elements.cpp`, e `svg_to_png.cpp` nomeadamente o suporte para elementos do tipo `ellipse`.
- Para cada forma `x` defina em `elements.hpp`/`elements.cpp` uma classe `x` correspondente tal que:
    - `x` seja subclasse de `shape` (casos de `polyline` e `polygon`) ou de outra sua sub-classe (`circle` deverá ser subclasse de `ellipse`, `rect` de `polygon`, e `line` de `polyline`);
    - o construtor, em linha com os atributos SVG considerados:
    - a função membro `draw`, redefinida de `shape`;
    - correspondente suporte para leitura do formato SVG em `svg_to_png.cpp`.

Transformações (atributo `transform`)

Elementos SVG podem ser transformados com uma ou mais operações especificadas com o atributo `transform`.

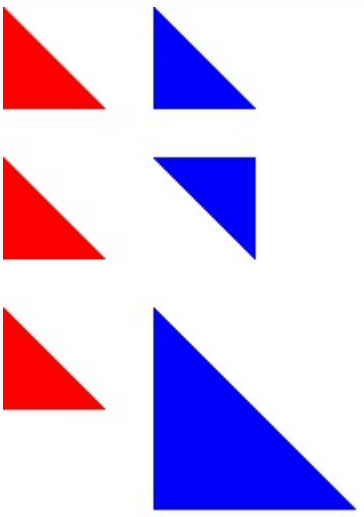
Exemplo

```
<svg width="180" height="260"  xmlns="http://www.w3.org/2000/svg'

  <polygon points="0,0 0,50 50,50" fill="red"/>
  <polygon points="0,0 0,50 50,50" fill="blue"
```

```
    transform="translate(75 0)"/>
  <polygon points="0,75 0,125 50,125" fill="red"/>
  <polygon points="75,75 75,125 125,125" fill="blue"
    transform-origin="100 100" transform="rotate(180)"/>
  <polygon points="0,150 0,200 50,200" fill="red"/>
  <polygon points="75,150 75,200 125,200" fill="blue"
    transform-origin="75 150" transform="scale(2)" />
</svg>
```

Imagem resultante



Transformações consideradas

Vamos considerar apenas o caso em que `transform` codifica no máximo uma transformação de um dos seguintes tipos (como ilustrado no exemplo):

- `translate(x y)`: translação do elemento segundo coordenadas `x` e `y`;
- `scale(v)`: escala com factor `v` em função de uma referência
  - referência é dado por coordenadas expressas pelo atributo `transform-origin` se definido, caso contrário a referência é o ponto `(0,0)`
  - vamos considerar no projecto apenas casos em que o valor `v` de escala é inteiro e maior ou igual a `1` e aplicado aos dois eixos X e Y;
- `rotate(v)`: escala com ângulo `v` (no sentido do ponteiro dos relógios para um valor positivo) em função de uma referência
  - como em `scale`, referência é dada por `transform-origin` se definido,

caso contrário a referência é o ponto `(0,0)`

Desvios ao standard SVG no projecto

- Para `polyline` e `line` a operação `scale` altera (escala também) a grossura das linhas desenhadas (`stroke-width`). No nosso caso **não vamos** considerar esse atributo pelo que o resultado nesses casos não será totalmente compatível com SVG (a linha terá sempre espessura 1 independentemente da escala).
- A operação `rotate` sobre `ellipse` roda toda a elipse em SVG. Por simplicidade vamos considerar apenas rodar o centro da elipse em função da referência de rotação e não de todos os pontos da elipse.

Tarefa

- A leitura das transformações já é feita em `parse_transform`.
- O código de transformação de coordenadas individuais já é dado por funções de `point` (veja `point.hpp` e exemplo do seu uso para `ellipse`);
- Precisa só de redefinir apropriadamente as funções membro `translate`, `scale`, `rotate` em subclasses de `shape`.

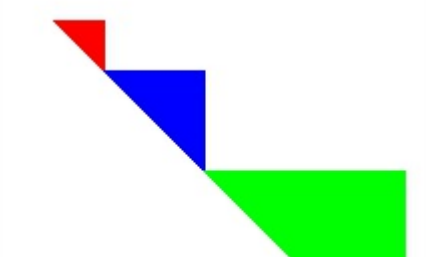
Exemplos de `scale` e `rotate` sem `transform_origin`

```
<svg width="210" height="210"  xmlns="http://www.w3.org/2000/svg'

  <polygon points="25,25 50,25 50,50" fill="red"/>
  <polygon points="25,25 50,25 50,50" fill="blue" transform="sca

  <polygon points="25,25 50,25 50,50" fill="green" transform="sc

</svg>
```

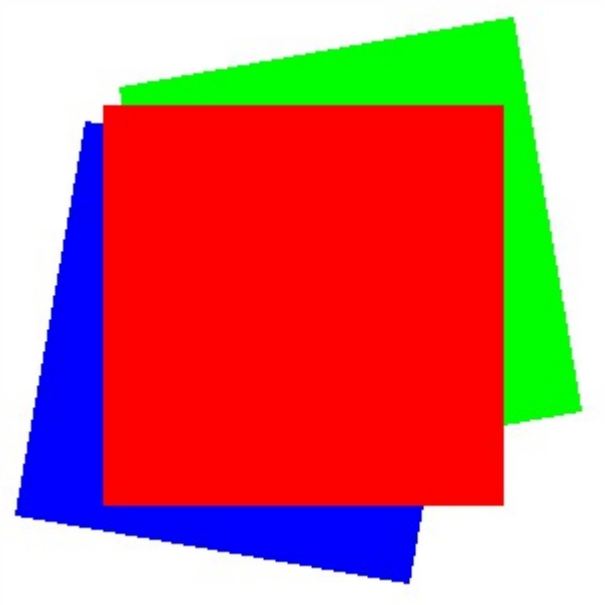


```
<svg width="300" height="300"  xmlns="http://www.w3.org/2000/svg'

  <rect x="50" y="50" width="200" height="200" transform="rotate

  <rect x="50" y="50" width="200" height="200" transform="rotate

  <rect x="50" y="50" width="200" height="200" fill="red"/>
</svg>
```



Grupos de formas (elementos <g>)

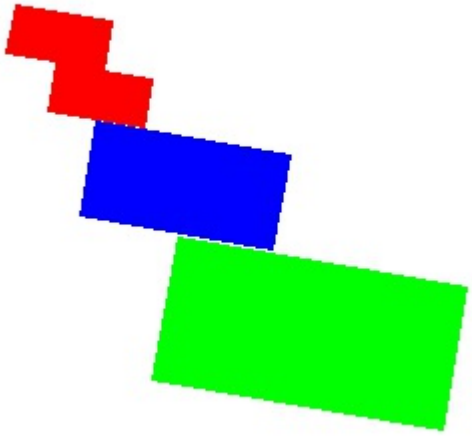
Um elemento <g> representa um grupo de formas definidos como sub-elementos. Por exemplo o seguinte grupo contém 2 círculos:

Exemplo:

```
<svg width="300" height="300" xmlns="http://www.w3.org/2000/svg"

  <g transform="rotate(10)">
    <rect x="50" y="50" width="50" height="25" transform="scale(
```

```
    <rect x="50" y="50" width="50" height="25" transform="scale(
    <rect x="50" y="50" width="50" height="25" fill="red"/>
    <rect x="50" y="50" width="50" height="25" transform="transl
  </g>
</svg>
```



Atributos a considerar

- `transform`: quando o atributo `transform` está definido, as transformações expressas devem aplicar-se a todos os elementos do grupo.
- `transform-origin`: poderá estar definido como para os elementos de forma;
- `id`: para duplicação de elementos (descrito a seguir).

Tarefa

- Crie uma subclasse de `shape` para representar um grupo de formas
- Esquema de implementação sugerido:
  - O grupo não terá uma cor global associada mas pode usar uma qualquer para passar ao construtor de `shape`;
  - Pode usar um “container” (ex. `std::vector`) para guardar apontadores

para as formas do grupo, mas tenha atenção em libertar estas no destructor do grupo.
    - Cada operação (`draw`, `translate`, …) deve iterar as formas no grupo e aplicar a operação individualmente a cada forma.
    - Para leitura do grupo deverá ser conveniente invocar `parse_shapes` recursivamente.

Duplicação de elementos (elementos `<use>`)

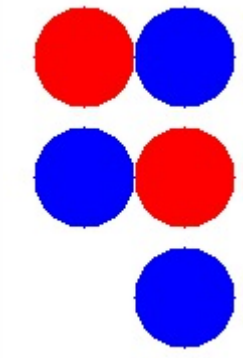
Um elemento `<use>` representa a duplicação (cópia de um elemento).

Exemplo:

```
<svg width="120" height="200" xmlns="http://www.w3.org/2000/svg":

  <circle id="c1" cx="40" cy="40" r="25" fill="red"/>
  <circle id="c2" cx="40" cy="40" r="25" fill="blue" transform="t

  <!-- Cópia de c1 e c2 (transladadas) -->
  <use href="#c1" transform="translate(50 60)"/>
  <use id="cc" href="#c2" transform="translate(-50 60)"/>
  <!-- "Cópia de cópia" -->
  <use href="#cc" transform="translate(50 60)"/>
</svg>
```



Atributos a considerar

- O elemento a copiar é expresso pelo atributo `href` que terá um valor da forma `#identificador` (note o uso do caracter `#` como prefixo). O valor `identificador` deverá ter sido associado a um elemento definido anteriormente através do atributo `id`.
- Quando o atributo `<transform>` está definido, as transformações expressas aplicam-se à cópia (e não ao elemento original).
- `id` (opcional): o próprio elemento pode ser "re-duplicado".

Tarefa

- Em sub-classes de `shape` re-defina a função membro `duplicate()` apropriadamente.
- Ao ler elementos SVG verifique se atributo `id` está definido, e mantenha registo da associação entre identificadores e formas, por exemplo usando um "container" `std::map`.