Representing 2D Cartoons using SVG

Cyril Concolato Jean-Claude Moissinac Jean-Claude Dufourd
Ecole Nationale Supérieure des Télécommunications
46 rue Barrault 75013 Paris
cyril.concolato@enst.fr, dufourd@enst.fr, moissinac@enst.fr

Abstract
Our work focuses on the encoding of 2D graphics animated cartoons with SVG. We expected this could be an efficient and useful coding with good versatility for various types of terminals.

First, we present some technical specification for such cartoons. Next, we give an overview of SVG support for cartoons and present methods best suited for a such task; some of our proposals could be useful for other 2D graphics application than animated cartoon. We also give some considerations on the compression and streaming of such cartoons.

Keywords
SVG, SMIL Animation, 2D Vector Graphics, cartoon

1 Introduction

With the emergence of powerful mobile devices, there is a need for low bitrate and high quality animations to be streamed over the Internet or LANs (W-Lan, GPRS…). In particular, the need for efficient coding and streaming solutions for high quality cartoons is real. As a response for this demand, a lot of proprietary solutions and a lot of standards are being developed. Of course it is possible to code a cartoon or an animation as a video. But this solution lacks flexibility and presents some poor performances in the trade-off between quality and bit rate. On the opposite, 2D vector graphics encoding brings a lot of flexibility because an animation can be easily resized and adapted to the capabilities of the terminal. A 2D vector graphics animation can even be customized for the user. We will see in this paper that vector coding is adapted to the coding of cartoons and graphics animation.

One of the candidate for the encoding of cartoons is the SVG-SMIL couple. SVG [1] is the candidate for the encoding of the graphics while SMIL [2] is the candidate for encoding the synchronization of the graphics and the audio. We will show in this paper that SVG presents some good possibilities for the encoding of cartoons, but need to be finely used. Moreover, SVG is an international standard and so this solution offers the benefits of interoperability. We also present some preliminary results about bit rate.

The rest of this paper is organized as follows. Section 2 gives an overview of the cartoon design techniques. Section 3 describes the techniques used to represent such cartoons using SVG. Section 4 explains the research environment and raises some remaining problems. Section 5 suggests some solutions to solve the compression and streaming problems Finally, Section 6 concludes this paper.

2 The Design of Cartoons

In this section, we describe the hypotheses made on some features which are relevant for efficient encoding of cartoons of a rather classic kind.

Classic cartoons like the ones produced by Hanna and Barbera, are generally made of a background on which some characters move. The traditional manner to compose the pictures of such a cartoon consists in drawing the background and the characters on different transparent celluloid sheets. Some sheets are then put on top of each other and filmed.

The drawings representing the characters are generally made of zones of uniform colors and surrounded with a border line of a different color. Sometimes, a zone is defined by a color and a transparency. The characters are drawn separately and their compositing is specified by the exposure sheet.

The exposure sheet describes, for each frame, the way to compose the final picture. It tells which background to use and what are the celluloid sheets to be stacked on top of the background. A short note indicates if an element of the previous frame is to be reused in the next one.

In classic cartoons, most of the animations are non parametric. The animation is the result of the replacement of the elements of the previous frame by new graphical elements drawn on different celluloid sheets. This is illustrated in Figure 1.

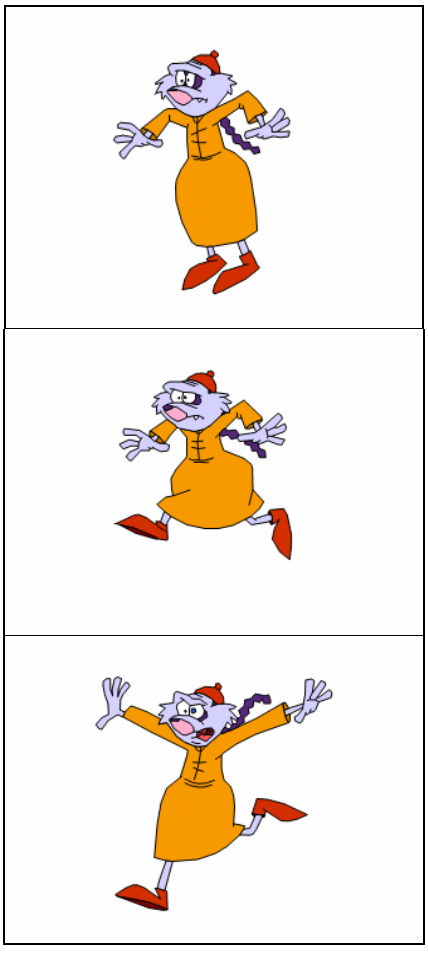

Figure 1 - Consecutive frames showing a non-parametric animation of a character

From a technical perspective, this means that the exposure sheet consists in a list of actions to perform for each frame:

- addition of a new element to be displayed,
- removal of an element of the previous frame,
- application of a parametric transformation to an element of the previous frame;

In the cartoons we are dealing with, all the graphical elements are either polygons filled with a uniform color and surrounded with an other color or polylines drawn with a uniform color. More elaborate cartoons may use gradients or textures to fill the polygons. Hence, the major part of the description of these elements is a list of 2D coordinates.

3 Representing Cartoons in SVG

The previous section gave an overview of the generic principles used in the cartoon design. This section shows how these principles can be mapped on SVG and SMIL concepts.
As described previously, there are two important parts in a cartoon: the characters and their behavior. The SVG representation of these two parts is described hereafter.

3.1 Representing the Cartoon Characters

In order to understand how the cartoon characters are represented in SVG, two aspects need to be described. This section will first describe how a single character could be efficiently represented in SVG and in a second part how the dictionary of characters is handled.

3.1.1 A Character: a list of shapes

A cartoon character is usually composed of several shapes. The set of shapes that compose the character may change over time. Each shape is described by an outline. The outline is a sequence of edges of the following types: straight lines, cubic or quadratic Bézier curves. Using the outline, a shape may be filled using a solid color, a gradient color, a texture, or not filled. It may also have holes in its filling. The

holes are themselves specified by a sequence of edges. The outline may be drawn with different width. Finally, some of the shapes may be used to clip other shapes.
The composition of a cartoon character is illustrated by Figure 2 and Figure 3 below.

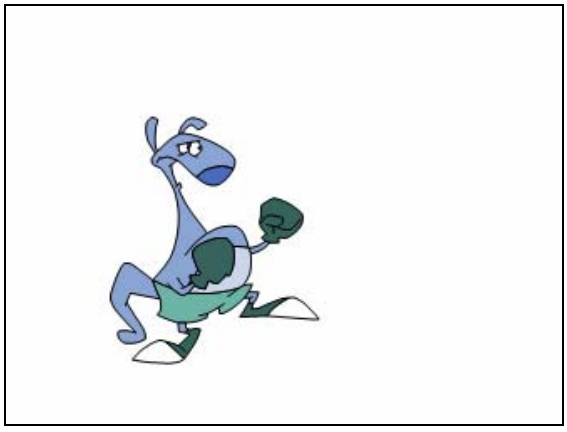

Figure 2 - A cartoon character

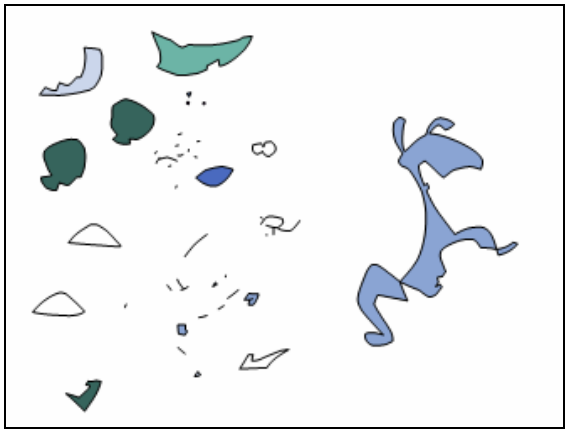

Figure 3 - Split elements composing a cartoon character

All the requirements of the representation of cartoon shapes are fulfilled by SVG. Indeed, the *path* element allows to draw all the types of edges described above. A *path* element may be filled or not. The filling can be of the types described also above. Holes can be achieved using *move* commands in the *path* data and the *fill-rule* property. Paths can be reused to do some clipping.

3.1.2 Managing the dictionary of characters

Hence, a cartoon character is a list of *path* elements with different filling, stroking, clipping properties. In order to reuse and/or move this character during the cartoon, all the path elements forming a character need to be grouped in a *g* element. This element is given an identifier that uniquely identifies this state of the character in the cartoon. It may also have spatial transformations such as rotation, translation and color transformations such as transparency, color shifting.

3.2 Representing the animations

The animation is what gives life to the cartoon characters. During the cartoon play-out, a character will move, change color, appear or disappear. Moreover, the set of shapes that represent this character may be replaced by a new set of shapes. This reflects the non-parametric feature of classic cartoons. This section explains how to represent all these changes in SVG.

3.2.1 To script or not to script

SVG offers a lot of scripting possibilities thanks to the Document Object Model. It is therefore possible to use a scripting language like JavaScript to perform the animation of the object. This approach has been chosen in [3]. However, scripts require the terminal to have a scripting engine implemented as well as an implementation of the Document Object Model. This increases the memory footprint of the player and therefore limits the number of devices that can play such cartoons.
Our approach is to represent the cartoons in SVG without using any script. Hence, they could be played on limited devices like cell phones. Our approach uses the animation capabilities of SVG, which is inherited from the SMIL Animation module.

3.2.2 Representing the timeline

In a cartoon, several character properties like the position, the color, the appearance or the set of shapes that represent this character can be changed at some precise point in time and for a certain duration. Two ways have been studied to represent the animation of the character properties. The first one was dropped because it suffered some limitations but we believe it is

useful to present it in order to understand the second one.

3.2.2.1 A timeline per cartoon character

The first approach consisted in inserting the SMIL animation elements such as *set*, *animate*, *animateMotion* within the *g* element that defines the graphical properties of the character. This approach is illustrated in the SVG code below.

```
<g id="Character1" visible="hidden">
  <path stroke="none" fill="red" d="…">
  <path stroke="4" fill="black" d="…">
  <set attributeName="visibility" to="visible" begin="1s"
end="2s"/>
  <animateTransform attributeName="transform"
type="rotate" from="0" to="90" dur="5s"
additive="replace" fill="freeze"/>
</g>
```

The visibility of a character is controlled using the *visibility* attribute. This means that a character can appear or disappear throughout the cartoon.

Finally, the whole cartoon is described by an SVG document of the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg width="384" height="288" viewBox="0 0 384 288"
>
<g id="Character2" visible="hidden">
  <path stroke="none" fill="yellow" d="…"/>
  <path stroke-width="1" fill="green" d="…"/>
  <set attributeName="visibility" to="visible" begin="1s"
end="2s"/>
  <animateTransform attributeName="transform"
type="rotate" from="0" to="90" dur="5s"
additive="replace" fill="freeze"/>
</g>
<g id="Character1" visible="hidden">
  <path stroke="none" fill="red" d="…"/>
  <path stroke-width="4" stroke="blue" d="…"/>
  <set attributeName="visibility" to="visible" begin="1s"
end="2s"/>
</g>
<g id="Character3" visible="hidden">
…
</g>
…
</svg>
```

The characters are laid out in the document according to their initial depth in the exposure sheet. The problem with this approach arises when the depth of a character changes during the cartoon. According to the SVG specification, the elements are rendered on top of each other in the order they appear in the document. Take the following cartoon scenario:

- at time T, Character A is displayed on top of Character B
- at time T+dt, Character B is displayed on top of Character A

The current approach cannot handle that scenario. That is one of the reason why we moved to the second approach.

3.2.2.2 A frame-based document

This approach solves the problem stated in the previous section and also offers a document structure which is closer to the exposure sheet.
The notion of frame is introduced. A frame is a composed of graphical objects that should be displayed at some point in time and for a certain duration. It contains all the graphical objects that need to be displayed during this time interval. A frame can be represented in SVG as follows:

```
<g id="frame_N" display="none" >
  <set attributeName="display" to="inline"
begin="N/FrameRate" end=" (N+1)/FrameRate"/>
  <defs>
    <g id="Character1">
      <path stroke="none" fill="yellow" d="…"/>
      <path stroke="green" fill="none" d="…"/>
    </g>
    <g id="Character2">
      <path stroke="none" fill="red" d="…"/>
      <path stroke-width="2" stroke="green" d="…"/>
      <path stroke="none" fill="yellow" d="…"/>
    </g>
  </defs>
</g>
```

Several points have to be noted in the above code. First, a frame is by default not displayed. The *visibility* attribute has been replaced by the *display* attribute for performance issues in the rendering. Then, all the characters are grouped within a *defs* element and their *animation* elements have been removed.

A consequence of using the *defs* element is that the characters are not displayed but instead defined and ready for being used via a *use* element. A character can even be reused outside the current frame. This is illustrated in the SVG code below.

```
<g id="frame_5" display="none" >
<set attributeName="display" to="inline"
begin="0.3846154" end="0.46153846"/>
```

```
<use xlink:href="#Character4"
transform="translate(0.0 288.0) scale(0.5 0.5) " />
<use xlink:href="#Character5"
transform="translate(0.0 288.0) scale(0.5 0.5) " />
<use xlink:href="#Character6"
transform="translate(32.0 215.0) scale(0.5 0.5) " />
</g>
```

4 Tests and Results

4.1 Environment of the experiment

We have worked on two different set of cartoons. The first one provided as a set of XML files produced by the Pegs software, from the company MediaPegs [5]. The second one gathered as a set of Flash files from various cartoon Web sites such as [4].

We have developed a Java program to translate the Flash files into SVG as well as into MPEG-4 BIFS.

The results of the conversion of either the XML files or the Flash files into SVG files were played in the Adobe SVG Viewer 3.0 running on Windows 2000. The conversion of the SWF file into SVG for the Butch Cassidy sequence which is 200 seconds long takes approximately 30 seconds on a Pentium IV 2.0GHz PC.

4.2 Results

In previous works [6] [7], we have shown that it is possible to use MPEG-4 BIFS [9] to represent classical cartoons. We also shown that the resulting BIFS file size is similar to the original Flash file size.

Using the techniques exposed in this paper, and adding a ZLIB compression using the gzip software, the results we have show that the gzipped SVG is usually less than twice the size of the SWF. In that case, the gzipped SVG files can still be played by the Adobe viewer. Some results are given in Table 1.
We have to explore some simple ways to reduce the size, mainly using less digit for each 2D coordinate and id name.

Sequence	SWF	SVGZ	SVG
Kangourou	78	137	770
Karate	91	208	819
Butch Cassidy	851	1863	10 902
Captain Distraction	753	1436	9 246
Pitch	558	1342	20025

Table 1 - File size in KB of different cartoons in different formats

However, even if the file sizes are similar between the three technologies, the performances of the playing process are quite different. Since Flash and MPEG-4 BIFS allow for progressive loading of local files and different levels of streaming, the playing starts as soon as the first frame is loaded. But, since SVG uses XML, the whole SVG cartoon has to be loaded before the rendering of the first frame. An immediate consequence of that statement is that SVG seems suitable for the representation of short cartoons. Possible solutions to solve this problem may be found with the help of XML fragments [8] or in the next section.

5 Considerations about Streaming and Compression

We have seen in the previous section that progressive loading of SVG documents is a challenge for long cartoons. In this section we propose two approaches to solve this problem. The first one suggests to use the MPEG-7 standard while the other suggests to define an update mechanism within SVG.

5.1 MPEG-7 Approach

MPEG-7 [10] has developed a technology to binarize and stream XML descriptions for which an XML schema exists. Within this technology called BiM, streamability of XML documents is achieved through a set of commands (insert, delete, replace or modify) that closely fit the actions to perform on each frame of the cartoon. Using BiM within SVG would achieve a triple benefit:

- Compression: the compression ratio achieved by BiM is generally better than ZLIB.
- Streamability: the XML document is split into semantically meaningful pieces of descriptions
- Memory foot print reduction: the whole document does not need to be loaded into memory.

These three advantages are very important for mobile devices which are very memory-bound.

5.2 Adding updates to SVG

This second approach does not deal with the compression problem but rather suggests to add some mechanism to SMIL and/or SVG in order to solve the streamability problem. This mechanism is a document update mechanism similar to the one found in BIFS. BIFS defines an update mechanism whereby any element in the scene can be added, deleted, replaced or have any property changed at any time. The timing of the update commands is expressed within the MPEG-4 Systems framework. Such a mechanism could be achieved in SMIL/SVG using *par* elements for the timing, *set* for the type of updates and SVG elements for the payload. This is illustrated in the code below:

```
<par begin="N/FrameRate" end="
(N+1)/FrameRate">
  <set target="Character1" fill="replace">
    <g>
      <path stroke="none" fill="yellow" d="…"/>
      <path stroke="green" fill="none" d="…"/>
    </g>
  </set>
    <set target="Character2" fill="remove"/>
</par>
```

6 Conclusion

In this paper, we have shown how to represent some classic cartoons using SVG and one of the problems we faced when producing such cartoons. We have also raised some problems related to compression and streaming and suggested some possible solutions that need to be further studied.

Bibliography

[1] *Scalable Vector Graphics 1.1,* http://www.w3.org/TR/SVG11/

[2] *Synchronized Multimedia Integration Language,* http://www.w3.org/TR/smil20/

[3] *Flash and SVG,* http://www.eprg.org/projects/SVG/flash2svg

[4] *Cartoon Network*, http://www.cartoonnetwork.com/watch/web_shows/

[5] http://www.mediapegs.com/

[6] *Codage MPEG-4 de dessins animés,* CORESA'01

[7] *Encoding of Cartoons Using MPEG-4 BIFS: Methods and Results,* IEEE T-CSVT

[8] *XML Fragment Interchange*, http://www.w3.org/TR/xml-fragment

[9] *Coding of audio-visual objects – Part 1: Systems*, ISO/IEC 14496-1:2000

[10] *MPEG-7 Overview*, http://mpeg-industry.com/mp7a/w4980_mp7_Overview1.html