Timed-Fragmentation of SVG Documents to Control the Playback Memory Usage

Cyril Concolato Jean Le Feuvre Jean-Claude Moissinac

Multimedia Group, Department of Image and Signal Processing, GET-ENST, LTCI UMR 5141
46, rue Barrault, 75013 Paris, France
{concolato, lefeuvre, moissinac}@enst.fr

ABSTRACT

The Scalable Vector Graphics (SVG) language allows in its version 1.2 the description of multimedia scenes including audio, video, vector graphics, interactivity and animations. This standard has been selected by the mobile industry as the format for vector graphics and rich media content. For this purpose, additional tools were introduced in the language to solve the problem of the playback of long-running SVG sequences on memory-constrained devices like mobile phones. However, the proposed tools are not entirely sufficient and solutions outside the scope of SVG are needed.

This paper proposes a method, complementary to the SVG tools, to control the memory consumption while playing back long-running SVG sequences. This method relies on the use of an auxiliary XML document to describe the timed-fragmentation of the SVG document and the storage and streaming properties of each SVG fragment. Using this method, this paper shows that some SVG documents can be stored, delivered and played as streams, and that their playback as streams brings an important memory consumption reduction while using a standard SVG 1.2 Tiny player.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – *Animations;* H.3.2 [**Information Storage and Retrieval**]: Information Storage – *File organization*

General Terms

Measurement, Design, Standardization, Languages.

Keywords

Scalable Vector Graphics, Memory usage, Streaming, Timing, Fragmentation

1. INTRODUCTION

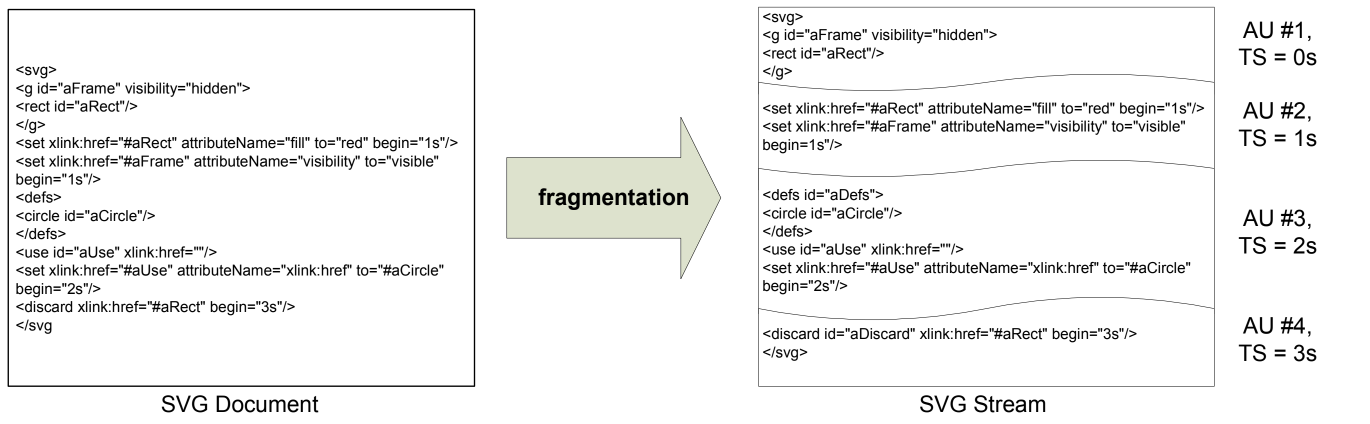
The Scalable Vector Graphics (SVG) Working Group is currently working on version 1.2 of the SVG language [1], which allows describing how a set of graphical, text and media (audio, video, images) primitives are used spatially and temporally to compose a multimedia presentation. The "Tiny" profile of SVG 1.2 is selected as a basis for the Rich Media format for mobile applications. Therefore, efficient consumption of long-running SVG animations on memory-constrained devices like mobile phones has become a requirement for SVG and an important challenge. Hence, some tools to improve the efficiency of the consumption of SVG documents have been defined. Firstly, the 'discard' element enables an SVG player to perform run-time garbage collection and reduce its memory usage during playback. Secondly, the progressive rendering process allows an SVG player to start rendering a scene even if the whole SVG document is not entirely received. The conjunction of both tools enables an SVG player to progressively render a scene when it is received and to remove data when it is not needed.

The problem with this situation is that usually SVG documents are transmitted as files over HTTP, where the data is sent as fast as possible, depending on the throughput of the server. If the bandwidth of the network is high, the whole document will be sent at the beginning of the download session, thus yielding to an important initial peak in the memory usage at the client side. This is also true for local file playback. There is therefore a need for a mechanism to control the timing of the data sent or read.

Traditional audio and video streaming offers such mechanism, but the streaming of SVG document is not an easy task. The REX working draft [2], the 'MORE' proposal [3] and the LASeR standard [4] allow streaming of SVG content, but require modifications of the SVG player at the scene level.

The method proposed in this paper only assumes a compliant SVG Tiny 1.2 player with no modification. It consists in using an auxiliary XML document to describe the timed-fragmentation of the SVG document to help servers control the sending and readers control the reading.

The rest of this paper is structured as follows. In Section 2, the proposed approach is presented and a language to describe the fragmentation of SVG documents is given. Some experiments using this approach are presented and discussed in Section 3. Finally, Section 4 concludes this paper.



```
<svg>
<g id="aFrame" visibility="hidden">
<rect id="aRect"/>
</g>
<set xlink:href="#aRect" attributeName="fill" to="red" begin="1s"/>
<set xlink:href="#aFrame" attributeName="visibility" to="visible"
begin="1s"/>
<defs>
<circle id="aCircle"/>
</defs>
<use id="aUse" xlink:href=""/>
<set xlink:href="#aUse" attributeName="xlink:href" to="#aCircle"
begin="2s"/>
<discard xlink:href="#aRect" begin="3s"/>
</svg
```

```
<svg>
<g id="aFrame" visibility="hidden">
<rect id="aRect"/>
</g>
```

```
<set xlink:href="#aRect" attributeName="fill" to="red" begin="1s"/>
<set xlink:href="#aFrame" attributeName="visibility" to="visible"
begin="1s"/>
```

```
<defs id="aDefs">
<circle id="aCircle"/>
</defs>
<use id="aUse" xlink:href=""/>
<set xlink:href="#aUse" attributeName="xlink:href" to="#aCircle"
begin="2s"/>
```

```
<discard id="aDiscard" xlink:href="#aRect" begin="3s"/>
</svg>
```

Figure 1 – Time fragmentation of an SVG document into SVG Access Units

2. TIMED-FRAGMENTATION OF SVG DOCUMENTS

In most cases, XML documents have to be fully parsed before any interpretation can be done, like rendering, unless progressive rendering is supported, as required by SVG and as many HTML browsers already do. Additionally, SVG scenes can be animated using SMIL [5] animation elements, like 'set' or 'animate'. Our approach considers the combination of progressive rendering and animations to envisage the timed fragmentation of SVG documents.

2.1 Principles of the approach

In our approach, we consider that SVG documents are made of timed rendering instructions, described by XML elements, and to each element, we associate a time instant. For graphical elements, this time instant is the first time it gets visible. For timed elements (animations or media), it is the first time they begin animating or playing. For discard elements, it is the time of execution. For other elements (e.g. scripts), it is the first time they are used.

We define an SVG Access Unit (AU), whose timestamp is T, as an XML document fragment (not necessarily well formed), where all the elements have timing information greater or equal to T. The timing information T associated with such SVG AU can be viewed as a transport time and used in MPEG-2 or in RTP. It describes when, at the latest, the elements are needed by the player. With this definition, an SVG scene is therefore "time-fragmentable" if 1) the XML document that describes it can be split into more than one SVG AU, each AU with a unique timestamp and, if 2) when sorted in increasing timestamp order, the concatenation of the sorted SVG AUs produces a document whose result is equivalent to the original document. This last condition is needed because SVG documents rely on the painter's algorithm for the rendering order.

Examples of "time-fragmentable" SVG scenes are vector graphics cartoons, video stream with subtitles, video with dynamic graphical overlay, or dynamic live scenes. Such scenes can be easily fragmented, as follows, because they have an inherent frame-based structure: an AU starts when a frame starts and ends when the next frame starts. This fragmentation can be described by the author, or could, in this case, be easily found by a simple analysis. Figure 1 illustrates the fragmentation of a simple SVG document into an SVG stream, made of four SVG AUs.

Reversely, it is clear that some SVG scenes are not fragmentable, for instance, if all elements are visible and all animations start from the beginning of the scene and if there is no deletion of elements. An interesting work would be to investigate static analysis algorithms to determine if and how a generic SVG scene can be fragmented, but such analysis is out of scope of this paper.

2.2 An XML language to describe the timed fragmentation

Following the previous approach, we have defined an XML language called NHML "Network Hint Markup Language" to describe, in an auxiliary document, the timed fragmentation of SVG documents. This auxiliary document can be used by multimedia packagers to store SVG content in the form of a stream. It may be also exploited by servers (like HTTP servers) to control the sending of file chunks. Finally, it may also be used directly by SVG players to determine how and when to read an SVG file. It may be compared to SMIL, or more generally to 'timesheets' as proposed in [6] with the following major differences: 1) the timing described in the NHML document is the transport timing of each AU, not the presentation timing; 2) the addressing mechanism identifies chunks of the XML document not necessarily well-formed. A simplified example of NHML document is given in Figure 2.

```
<stream timeScale="1.0" timeIncrement="1"
  baseMediaFile="flash2.svg">
<sample isRAP="yes" xmlFrom="doc.start"
                    xmlTo="aFrame.end"/>
<sample isRAP="no"  xmlFrom="aFrame.end "
                    xmlTo="aDefs.start"/>
<sample isRAP="no"  xmlFrom="aDefs.start"
                    xmlTo="aDiscard.start"/>
<sample isRAP="no"  xmlFrom="aDiscard.start"
                    xmlTo="doc.end"/>
</stream>
```

Figure 2 – Example of NHML code

The `stream` element declares a stream made from the SVG file referred to by the `baseMediaFile` attribute. The `timeScale` and `timeIncrement` attributes allows determining automatically the timestamps of all AU when the stream uses a fix frame rate, such as cartoons. Then, for each AU, a `sample` element is used. Each `sample` element indicates a timestamp in the TS attribute (optional) and the start and the end of AU payload using the `xmlFrom` and `xmlTo` attributes respectively. The syntax of these attributes uses an IDREF, with a special IDREF for the document root node (doc), followed by SAX-event names: start or end. A possible improvement could use XPath expressions instead of IDREF.

3. EXPERIMENTS AND RESULTS

3.1 Software environment and case studies

We have implemented the parsing and interpretation of the NHML language in the tool called MP4Box of the GPAC Open Source software [7]. This has allowed us to create ISO files with SVG tracks and to prepare them for streaming. The resulting files played locally as well as streamed from an unmodified Darwin Streaming Server [8]. We also used the Osmo4 player of the GPAC project, which includes streaming protocols support (e.g. RTP), SAX-based XML parsing capabilities; and SVG rendering capabilities compliant with SVG Tiny 1.1 (including progressive rendering) and with some SVG Tiny 1.2 features, like the 'discard' element and the 'audio' and 'video' elements. We have applied the proposed method to fragment SVG content into SVG streams on long-running animated vector graphics cartoons, resulting from the transcoding of Flash content, as described in [9] or in [10]. We have also applied it to the streaming of audiovisual content augmented with synchronized graphical overlays like subtitles or advertising. In both cases, NHML descriptions were generated automatically by the cartoon or subtitle transcoders.

3.2 Results

We measured the memory consumption during the reception and playback of SVG content. Figure 3 shows the memory usage as a function of time for four scenarios. In all cases, the same SVG file is read and in networking cases, a network which has a bandwidth of 800kbps is used. In this example, the content is a 14 seconds long cartoon sequence, with an average data rate of 500 kbps, and peaks at 1 Mbps. The time axes of the plots have been aligned so that rendering time of the first frame, named t_0, coincide.

We can see that, in all scenarios involving a non-fragmented XML file, the memory usage shows an initial peak and then starts decreasing. The initial peak is due to the loading of objects not needed at that time. It is at its highest when the full file is loaded before rendering and it is lower with the loading/reading speed. Obviously, the bigger the file (the document) is, the higher this peak is. The memory consumption decreases during playback due to the use of SVG 'discard' elements.

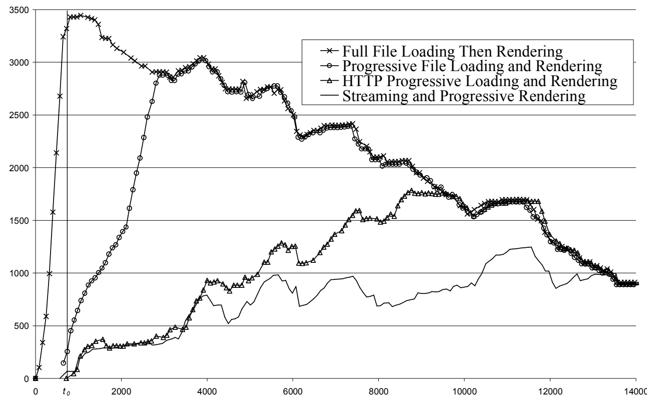


Figure 3 – Memory Usage (kB) vs. Scene Time (ms)

We can see that the scenario which requires the smallest amount of memory is the “Streaming and Progressive Rendering”. The memory consumption peak is reduced by 64%. We have made the same experiments on 23 SVG documents (cartoons and subtitles with various sizes, durations, and complexity) with a gain ranging from 10% to 93%, with an average of 64 %. This reduction is due to the fact that the objects are loaded only when they are needed, rendered and then deleted.

It has to be noted that, in this experiment, we have used the RTP protocol to deliver the SVG fragments to show that real streaming of SVG content is possible. However, the results are the same 1) if the HTTP protocol is used provided that the HTTP server uses the instructions given in the NHML, and 2) in the context of local file playback (i.e. without networking), if the SVG player is modified to read the SVG content either from a stream stored in a multimedia container file or if the SVG content is read directly using the NHML file.

4. CONCLUSION

In this paper, we have defined the notions of timed-fragmentation of SVG documents, of SVG Access Unit and of SVG stream. We have presented a new XML language which allows describing this timed-fragmentation as well as storage and streaming properties. We have shown that, using a timed-fragmented distribution of the SVG content, the required memory is greatly reduced (either from multimedia container files or streams) and that the SVG 'discard' element provides great help in keeping the memory usage low.

5. REFERENCES

[1] Scalable Vector Graphics (SVG) Tiny 1.2, W3C Working Draft, Dec. 2005, http://www.w3.org/TR/SVGMobile12

[2] Remote Events for XML (REX), W3C Working Draft 13 October 2006, http://www.w3.org/TR/rex/

[3] V.Setlur et al., "More: A Mobile Open Rich Media Environment", *IEEE International Conference on Multimedia and Expo*, July 2006, pp. 2029-2032

[4] J.-C. Dufourd, O. Avaro, C. Concolato, "An MPEG Standard for Rich Media Services", *Multimedia, IEEE*, Volume 12, Issue 4, pp. 60-68, Oct.-Dec 2005. 2005.

[5] D. Bulterman and L. Rutledge, "SMIL 2.0: Interactive Multimedia for Web and Mobile Devices", Springer-Verlag, Heidelberg, May 2004.

[6] W. ten Kate et al., "Timesheets – Integrating Timing in XML", *WWW9 Workshop: Multimedia on Web*, 2000

[7] GPAC Open Source Project, Multimedia Framework, http://gpac.sourceforge.net

[8] Darwin Streaming Server, http://developer.apple.com/darwin/projects/streaming

[9] S. Probets, J. Mong, D. Evans, D. Brailsford, "Vector graphics: from PostScript and Flash to SVG", *ACM Symposium on Document Engineering 2001*, pp. 135-143

[10] C. Concolato, J.-C. Moissinac, J.-C. Dufourd, "Representing 2D Cartoons using SVG", *Proceedings of SMIL Europe* 2003, Paris, Feb. 2003