Multimedia, PDF and ConTEXt

Michal Vlásak

The possibility of inserting multimedia (audio, video, 3D) into PDF files has been here for a long time, albeit in different forms. Traditionally ConTEXt has had support for it, but the support in PDF viewers was dubious. What is the situation today, and is it worth all the hassle?

1. Introduction

With features like multimedia there has always been a circular problem – the features are mostly unknown, hence not demanded by users, thus not implemented by PDF writers, nor PDF viewers. It doesn't help that for multimedia, the standard defines as much as five different mechanisms for including them. What do these mechanisms offer? What does ConTEXt support? What can the viewers handle? And does it even make sense today?

Recently, as part of my bachelor thesis, I looked into these problems with the hope of presenting my findings to you. I will introduce and compare the various mechanisms the PDF standard offers, as well as discuss their support in real world.

2. PDF mechanisms for multimedia

Over the years, various versions of the PDF standard have developed several ways of including "multimedia" into PDF files. The simplest is XObjects which allow raster and vector graphics – this is a well-known and well-supported feature in both PDF writers and viewers.

However, later revisions of the PDF standard added several different mechanisms for including video, audio or 3D files (each mechanism supports a different subset of these three). They all have in common the capability of showing and playing multimedia as part of the PDF page. However, in choosing the right one, we might want to delve into the details. For the purpose of evaluating these mechanisms from the perspective of ConTEXt, it is possible to devise the following criteria:

a. support in the PDF standard (too new, deprecated, etc.),

b. supported media types (audio, video, 3D),

c. support for different source types (embedded file, external file, URL file)

d. what is possible to achieve ("usefulness") and at what cost ("complexity"),

e. current support in ConTEXt,

f. and the most important: support in PDF viewers.

The "source types" need a bit of explanation. Essentially all file references in PDF files can be one of three types:

1. *Embedded file*. The referenced file is literally embedded *into* the PDF file, which means that it can also be compressed as part of it (although you don't gain much by compressing multimedia). This is nice because the re-

sult is integral – the media file can't get lost and there is only a single file to distribute.

2. *File URL*. The reference to the file is solely the URL. While this takes almost no space at all in the PDF file, it means that the availability of the media file cannot be guaranteed since it is not tied to the PDF file.

3. *External file*. The PDF includes only a file path, not the full file. Compared to file URLs, the file doesn't have to be available over the internet, but has to be distributed along with the PDF file (*and* the relative path has to match).

While the possibility of embedding the media file alone may be more interesting than just side stepping the PDF viewer altogether and using another application to play the file, with tighter integration the demands are increasing. Hence, the "usefulness" aspect includes the possibility of interaction or scripting e. g. using media player buttons ("controls"), scripting with JavaScript or control with PDF actions (ConTEXt's `\goto`, and triggers like "page open", which may allow autoplay).

Several PDF viewers were tested: Acrobat Reader (AR), Foxit Reader (FR), SumatraPDF (SU) on Windows and Evince (EV), Okular (OK), Xpdf (XP), MuPDF (MU), Firefox (FF), Google Chrome (CR) on Linux. But only four (Acrobat, Foxit, Evince and Okular) were found to support multimedia (at least partially).

2.1 Movies

Movies first appeared in PDF 1.2 (1996), but had since been deprecated (PDF 1.5, 2003) and became unsupported (PDF 2.0, 2017). The mechanism supports video *and* audio from any source (either embedded, external or file URL).

The mechanism is relatively simple and allows some customization and control (media player controls, PDF actions).

This is the backing mechanism for including most video and audio in ConTEXt:

```
\externalfigure[video.mp4]
  [width=\textwidth,
   height=.461\textwidth]

\useexternalsoundtrack
  [myaudio][audio.mp3]
\checksoundtrack{myaudio}
```

Here, only the "external file" method is allowed in ConTEXt.

Only Evince and Okular support this mechanism today (with their usual quirks, see further). Notably Acrobat Reader no longer supports movies.

2.2 Multimedia ("Renditions")

This mechanism first appeared in PDF 1.5 (2003). It is not officially deprecated, but Acrobat considers it "legacy".

The mechanism in theory supports any multimedia type (even Flash and images), but in practice only video and audio make sense today.

This mechanism was supposed to replace sound and movie objects, hence its deprecation. The mechanism is very complex (the spec is ten times longer than the one for movie objects). It expects the PDF viewers to work with plugins and introduces ways for determining if a media file is really playable in some plugin. It is allowed to even include more media files (to serve as fallback should the primary one be unsupported by the viewer). Another complexity is that the concept of the rectangle where the media will be played ("screen") is separated from the media itself ("rendition"). In theory this allows mixing and matching, but in practice is a lot of unnecessary complexity.

This mechanism is the most extensible though. It allows multimedia player controls, as well as PDF actions. The PDF action can be either one of the predefined ones or entirely programmed in JavaScript using the related API.

ConTEXt makes the mechanism available to its fullest extent, meaning that a bit of the complexity leaks to the interface:

```
\useexternalrendering[embedded]
  [video/mp4][video.mp4][embed=yes]

\useexternalrendering[URL]
  [video/mp4]
  [https://example.com/video.mp4]

\useexternalrendering[external]
  [video/mp4][video.mp4]

\definerenderingwindow[mywin]
  [width=\textwidth,
   height=\textwidth]

\placerenderingwindow[mywin][embedded]
\placerenderingwindow[mywin][URL]
\placerenderingwindow[mywin][external]
```

In the example above, all three file sources are showcased. The complexity leak is apparent, because one has to manually specify the MIME type (`video/mp4`). Also the page area and media file are defined separately. Flash files (`.swf`) could also be inserted this way, but that would be pointless today.

Evince and Okular support this mechanism (with the usual quirks), Acrobat and Foxit do as well, but Acrobat sadly only allows embedded files.

2.3 3D art

This is the first mechanism that facilitates 3D files. It first appeared in PDF 1.6 (2004) and allows the inclusion of U3D and later PRC files. The 3D objects described in the files are shown in a scene whose parameters (like camera position/direction, background color, lighting, etc.) can be configured.

The flaw in this mechanism is that the source is not a file, but a “PDF stream”, which is essentially an embedded file with different metadata. It should also allow “external files” to contain the stream data but this is not used much in practice though.

This 3D functionality is really nice. It allows a great amount of interactivity (playing with the camera, selectively disabling 3D objects, etc.) and also scripting, allowing switching between predefined “views” with PDF actions, and *many* other possibilities with JavaScript. One can even program animations that allow user interactions (e. g. buttons for changing the animation parameters).

ConTEXt handles both PRC and U3D files with `\externalfigure`. Although a very basic usage is possible, one would usually set some of the many parameters using the `display` and `controls` Lua parameter sets.

Adobe Acrobat has full support for this mechanism; the only flaw is that it doesn’t allow external streams (but this isn’t what one would want in most cases anyway). Foxit Reader also has support for 3D but it is more limited (e. g. no support for JavaScript and printing previews).

2.4 Rich Media

Rich media first appeared in Adobe extension level 3 to PDF 1.7 (2008), but was later included in PDF 2.0 (2017). It was meant to replace both the “renditions” (audio, video) and the 3D art mechanisms, with an unified approach based on Flash; that also means support for arbitrary Flash applications. Only embedded files are supported.

One distinct feature of this mechanism is that it allows the playing of multimedia in a customizable window. It even supports a full screen window, not just part of the page like the other mechanisms do.

While the mechanism is heavily based on Flash (which has been obsolete since December 2020), it does allow the direct playing of media files *without* Flash (this could be called “plain” rich media).

The initial idea was that the PDF viewer would support Flash (and playing its video as well as mp4), but the audio/video wouldn’t be played directly by the PDF viewer. Instead it would be played by an intermediate Flash application (embedded in the PDF along with the media file) that would provide the graphical user interface (controls) as well as the scripting capabilities. This means that this mechanism has some inherent complexity that is not justifiable nowadays.

Fortunately 3D files have always been used in the “plain” way and haven’t suffered much from additional complexity in rich media. In the end, the only difference between 3D Rich Media and 3D art is the internal wrapping structure (3D rich media uses normal files instead of streams, which is more consistent and also allows more than one script file to be used).

Unfortunately with the death of Flash, there remains no scriptability for audio and video. There is also no way to display multimedia player controls (although there is a working hack for Acrobat).

The use of this mechanism in ConTEXt is based on Flash. For example, to include a video, one would embed a Flash video player like `vplayer.swf` together with a media file, passed as an argument to the player (but the process is simplified for the user).

With Flash being dead, this approach is no longer useful, but you can still find the details in `java-imp-vplayer.mkiv`.

Surprisingly, Acrobat and Okular still support this method! Both have a compatibility layer that detects an embedded Flash media player file and instead of using it to play the video, they play the video natively. This is good because you can find a lot of documents that use Flash-based rich media. But there is absolutely no need to create *new* documents with embedded Flash player applications – it only takes additional space and isn’t even used. The different compatibility layers sometimes pickup other information (like show controls or loop the multimedia), and still there is no possibility of scripting.

Okular notably doesn’t support plain rich media. The support would be easy to add, but my proposed patch[1] depends on changes[2] to the `poppler` library, which I haven’t finished yet.

Support similar to Okular’s should be relatively easy to add to Evince as well. The hard work of supporting video in the first place is already done.

The 3D support is the same as with 3D art for Acrobat Reader. Weirdly, Foxit Reader doesn’t support 3D files wrapped in Rich Media, even though there doesn’t seem to be any good reason not to.

3. PDF viewer quirks

While the previous descriptions might have sounded positive, the situation is not very good. There are many PDF viewer quirks that prevent the pleasant consumption of multimedia in PDF.

3.1 Okular, Evince

The support for video and audio in Okular and Evince is based on GStreamer. GStreamer is a framework that allows a range of different me-

PDF feature	Criteria	AR	FR	EV	OK
Movies	Embedded file	✗	✗	✓	✓
	External file	✗	✗	✓	✓
	URL file	✗	✗	✓	✓
	Action	✗	✗	✗	✗
	Controls	✗	✗	✓	✓
Renditions	Embedded file	✓	✓	✓	✓
	External file	✗	✓	✓	✓
	URL file	✗	✓	✓	✓
	Action	✓	✓	✗	✗
	Controls	✓	✓	✓	✓
	Sections	✓	✗	✗	✗
	Customization	✓	✓	✗	✗
Rich Media	Embedded file	✓	✗	✗	✓
	Plain	✓	✗	✗	✗
	Flash compatibility	✓	✗	✗	✓
	Autoplay	✓	✗	✗	✓
3D art	External stream	✗	✗	✗	✗
	PDF 1.7 extensions	✓	✓	✗	✗
	Action	✓	✗	✗	✗
	JavaScript control	✓	✗	✗	✗
	Interactivity	✓	✓	✗	✗
	Preview printing	✓	✗	✗	✗
3D Rich Media	Same as 3D art	✓	✗	✗	✗
Formats	MP3	✓	✓	~	~
	Opus	✗	✗	~	~
	AVI	✓	✓	~	~
	MP4	✓	✓	~	~
	MPG	✓	✓	~	~
	MOV	✗	✗	~	~
	WMV	✓	✓	~	~
	U3D	✓	✓	✗	✗
	PRC	✓	✓	✗	✗

Table 1. Multimedia support in PDF viewers

dia processing. Okular and Evince use it to play the media files. Since GStreamer's media type support is based on plugins, the right plugin for the media type has to be installed. These plugins are distributed in bundles and three of them cover just about any reasonable format and more.

But while the media file format support is great, both of these PDF viewers don't really support multimedia PDF actions or JavaScript for more control over the media playback. One can then ask what is the benefit of playing the multimedia in the PDF viewer in first place.

3.2 Acrobat, Foxit

Acrobat and Foxit both use Windows Media Player for playing videos. Both support controls, but behave differently – Acrobat displays the controls *outside* of the multimedia annotation, Foxit *within*, making the actual result less predictable.

Both viewers (for security reasons) also nag you to allow the media playback every time (even multiple times in a single document). You can select to trust a file either this time, or from now on, but if somebody opens a foreign PDF with video, they are not going to get smooth experience.

Another thing is that there is a check box in Acrobat Reader, which enables the "legacy" Multimedia mechanism. Different versions, of course, may have the checkbox either checked or unchecked by default, further limiting the chance of a pleasant user experience.

4. Conclusion

All in all, of the three mechanisms, one is deprecated and unfortunately no longer supported in the most common PDF viewers, while the other two mechanisms are needlessly complex and in reality limited. For example, while the multimedia mechanism supports JavaScript, depending on it essentially narrows the choice to Acrobat Reader.

The old multimedia ("renditions") mechanism is the most widely supported across PDF viewers, but the "plain rich media" support could be brought up to par.

In theory, despite the loss of Flash, rich media seems like a potential future investment, because it is embraced by recent PDF standards, and apart from handling audio and video, it is also the best way of inserting 3D files (albeit with no controls/scripting).

5. ConTEXt future

What should ConTEXt do? On one hand all available mechanisms are flawed in one way or another. On the other hand some users may still find the functionality useful.

I see two opposing possibilities in the area of audio/video:

1. Give up on the functionality completely. This seems to be what the users did, because as it turns out, for some time now, multimedia content inserted by ConTEXt was not playable for one reason or another anyway (although some bugs have been fixed recently).

2. Keep the functionality, but clean up the code to remove what is now obsolete. The figure system could change the backing PDF mechanism to work with rich media.

Note that 3D support in ConTEXt is completely fine and works in Acrobat and Foxit. There are even open source tools one can use to create these 3D files – Meshlab, which can convert other formats to U3D, and Asymptote, which can generate PRC files.

6. Summary

The support of multimedia in PDF viewers is summarized in table 1 (p. 92). As a bonus, support of a few other features I have tested is in table 2 (p. 96). "✓" means full support, "∼" partial support (supported, but not fully / with some quirks) and "✗" means no support.

Most partial support designations are for the different multimedia formats in Evince/Okular (where they need the right plugin installed). `/GoToR` actions differ wildly across viewers (e. g. some don't support URLs, some don't support page numbers, etc.). `/Text` annotations have only partial support in most viewers because they implement a reasonable behaviour, instead of what the standard specifies (that is what the other viewers do).

7. Closing words

This was a dump of knowledge that I gained from writing my thesis.[3] Sadly its in Czech, but a large part of what I deem practical today is implemented and documented in English[4] (mostly plain TEX macros). Other resources for this topic were linked on the mailing list.[5]

Sadly, while working on this, I didn't have access to the PDF 2.0 standard. My information mostly comes from the PDF 1.7 standard and publicly known information about PDF 2.0 – the rich media mechanism has been included in PDF 2.0, but I am not sure to what extent the Flash part got implemented. I also don't know if there really is anything new, but nothing I have read suggests it. Regardless, viewer support isn't complete for something standardized over 20 years ago, so a sudden revolution in PDF viewers is not to be expected.

1 https://invent.kde.org/graphics/okular/-/merge_requests/426
2 https://gitlab.freedesktop.org/poppler/poppler/-/merge_requests/855
3 https://dspace.cvut.cz/handle/10467/95065
4 http://mirrors.ctan.org/macros/luatex/optex/pdfextra/pdfextra-doc.pdf
5 https://mailman.ntg.nl/pipermail/ntg-context/2021/103011.html

PDF feature	Criteria	AR	FR	EV	OK	SU	XP	MU	FF	CR
Bookmarks	Are shown	✓	✓	✓	✓	✓	∼	✓	✓	✓
	Non-`GoTo` actions	✓	✓	✓	✗	✗	✓	✗	✓	✓
Annotations	Comments (`/Text`)	✓	✓	∼	∼	✗	✗	∼	∼	∼
	Attachments (`FileAttachment`)	✓	✓	✓	✓	✗	∼	✓	✓	✗
`/EmbeddedFiles`		✓	✓	✓	✓	∼	✓	✗	✓	✗
Actions	`/GoTo`	✓	✓	✓	✓	✓	✓	✓	✓	✓
	`/GoToR`	∼	✓	∼	∼	✗	∼	✗	∼	✗
	`/URI`	✓	✓	✓	✓	✓	✓	✓	✓	✓
`/OpenAction`		✓	✓	✗	✗	✗	✗	✗	∼	✓

Table 2. Interactive features supported in PDF viewers

Photo: Hraban Ramm