PDF Document Rendering on Mobile Devices in the Case of Aurora OS

Alexey Fedchenko
Open Mobile Platform
St.Petersburg, Russia
a.fedchenko@omprussia.ru

Kirill Chuvilin
Open Mobile Platform
Moscow, Russia
k.chuvilin@omprussia.ru
Moscow Institute of Physics and Technology
Moscow, Russia
kirill.chuvilin@phystech.edu

***Abstract*—The article is devoted to the problem of displaying the PDF documents on mobile devices in the case of Aurora OS. The performance problem becomes extremely important for mobile applications. Other technical requirements are also formulated. In case of using third-party libraries it is necessary to take into account the licenses under which they are distributed. The basic requirements are met by Poppler, PDFium and muPDF libraries. This article describes the API of these libraries and analyzes the speed of rendering and the quality of the result.**

I. Introduction

Aurora operating system [1] is designed for the B2B and B2G segments. One of the most common tasks for these areas is document handling: viewing, editing, validating, signing. The PDF is a very common format along with office document formats (docx, odt). It allows you to transfer documents between devices without loss of formatting. Therefore, some Aurora OS applications need to display documents in the PDF format. Solutions of this type include: office and mail applications, automation of document management, information applications.

The PDF processing libraries are needed to meet the needs of third-party developers of such applications. A united system component allows developers not to waste time on choosing the library to use, but to start implementing the application right away. This approach also makes it easier to keep the library up to date, which has a positive effect on the security of the system.

The PDF format is complex. Its documentation is nearly 800 pages long [2], [3].

Each PDF document contains:

- The header specifies the version of the used PDF specification.
- The body of the document contains text streams, images, other multimedia elements, etc. The body section is used to store all the document data that is visible for a user.
- The cross-reference table refers to all the elements from the body that are used on the pages.
- The PDF trailer specifies how a reader application should find the cross-reference table and other special objects.

The PDF document contains eight basic types of objects: booleans, numbers, strings, names, arrays, dictionaries, streams, and the null object. Objects may be labeled so that they can be referenced by other objects (Fig. 1) [4].

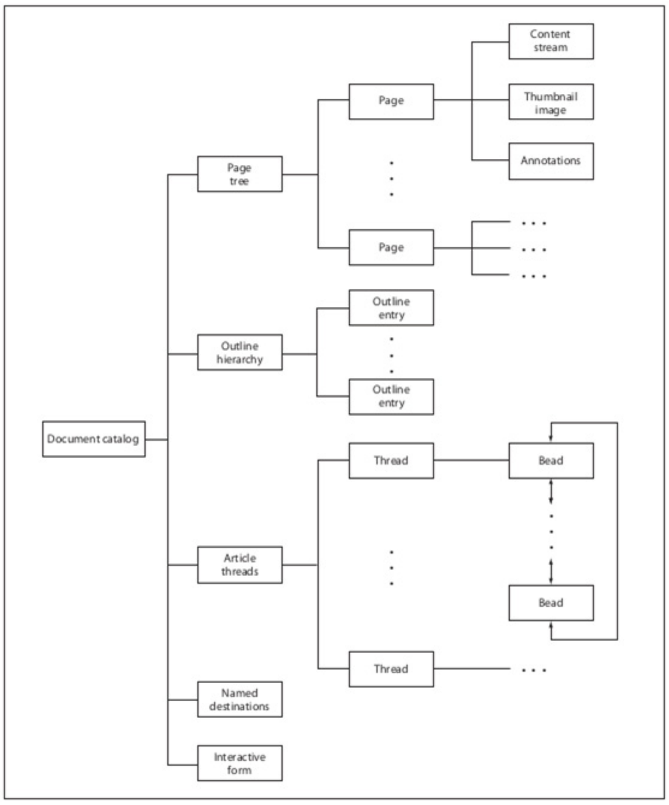

Fig. 1. PDF format structure

Implementation of a custom solution to work with such a complex standard is time-consuming and rarely feasible. It is convenient to use one of the already existing libraries to work with the PDF [5]. It is important to consider the following features. Aurora OS is POSIX-compatible [6], so solutions for Android and iOS cannot be reused. On the other hand, devices running Aurora OS are smartphones and tablets, and this imposes performance limitations compared to solutions for Linux distributions designed to run on desktops.

The peculiarities of the platform form the requirements for the suitable libraries:

1) *Work speed* is the minimum time spent on document operations:
    a) document object creation (including file loading);
    b) page object creation;
    c) page rendering;
    d) loading of annotations, text, and other additional information.
2) Criteria of document *processing quality* are:
    a) dimensional accuracy;
    b) page content rendering correctness;
    c) stability.
3) The *development technologies* requirement is about what tools are used to build the library, and what programming languages the library supports. Not all technologies can be effectively adapted to the mobile platform. In the case of Aurora OS, it is required that the library is compatible with C++17 (gcc compiler) [7]. It is also convenient if there is support for the Qt framework [8].
4) The *license* requirement reason is that the terms of use are defined for libraries intended to be used in third-party projects. It is important to keep in mind that there are not only licenses that prevent the use of the library in third-party applications, but also variants that require disclosure of the source code. For example, if a library with GPL [9] is used in a project, it is necessary to disclose the source code of the entire application. And this is unacceptable for a significant part of third-party projects. On the other hand, the MIT [10] or BSD [11] licenses allow use it without such restrictions.
5) The *functionality* of a library in each case is determined by the requirements of the application that uses it. But to summarize all the cases we can say, that the following functions should be provided:
    a) open document and password protected document;
    b) extract meta information about a document such as an author, creation and modification date, page count;
    c) load pages;
    d) render pages;
    e) render the page at arbitrary scale so that thumbnail preview or zoom can be implemented;
    f) render arbitrary part of the page to opportunity realize tile render;
    g) obtain page information such as number of the page, page size;
    h) obtain annotations, information of annotations: type, size, and position on the page, the target of a reference;
    i) save changes made by a user to insert comments or fill forms;
    j) obtain page text for the select and copy opportunity.

In the context of the functionality it is worth to note that it is not always possible to display all the nuances of a PDF document, especially on a mobile device due to the limited performance, screen size, etc. In this case, it is easier to display an image rendered from the document page. Text selection, annotations and other content could be overlaid if necessary.

When rendering a document, it is often necessary to show its contents at different scales: displaying page thumbnails, zooming in on a particular part of the page. To create thumbnails, it is possible to scale the resulting page render to the desired size, but it is much more convenient when the library provides methods to generate the page image at an arbitrary scale. For larger scale, the situation is slightly different. Usual scaling of rendered page image is fraught with loss of quality. On the other hand, on mobile devices at high values of approximation, there is a possibility of exceeding the maximum size of the texture buffer [12]. For example, on the Inoi R7 [13] the buffer is limited to 4096 points per dimension, Having the original page size of 512x512 pixels it gives a maximum of eight times the magnification. Also the rendering time of the page is multiplied, which leads to display delays.

To bypass these restrictions we use tile rendering: small parts of the page are rendered to construct the needed view. With such approach the time of rendering decreases, because the parts out view bounds are not needed. Consumed memory also decreases, for the same reason. To implement tile rendering, the library must provide an API for rendering an arbitrary portion of the page. All libraries in this article have the corresponding methods.

II. Environment

All actual devices working on Aurora OS, using ARM architecture.

Aurora SDK is used to develop applications for Aurora OS. It provides both IDE and build tools. The supplied compiler is Linaro GCC [14].

Linaro GCC is GCC optimization for ARM platform use. Aurora OS 3.2.2, current today, uses GCC version 4.9.4.

Besides SDK provides a device Aurora OS emulator, working on host architecture, for the developing application launch and debug opportunity, without a physical device.

Since development is done on x86 workstations, cross-compilation is used to build source code into binaries for the ARM architecture. To make it easier to build source code for a target platform (for example, an emulator using x86 or a physical device using ARM), the SDK uses the Scratchbox2 [15] toolkit, which automatically configures the build environment.

The Qt framework is used in the development of application software. It involves the use of C++ to implement the logic, and QML to describe the user interface [16], [17]. Qt also makes efficient use of C++ objects in QML, and data binding provides the ability to use the MVVM [18] pattern. This provides opportunities to use C++ libraries to implement graphics-related functions, including the PDF rendering.

Qt provides API for asynchronous operations and comfort work with threads besides convenient and effective using of MVVM. The asynchronous approach to software development can significantly increase the performance of applications on devices with multi-core CPUs, including modern mobile devices. On the other hand, the intensive use of multithreading for heavy computations leads to a higher battery drain rate, which can negatively affect the user experience.

The highest available version of Qt for Aurora OS is 5.6 at the moment. This limit is due to the license restrictions.

To test the speed and quality of the libraries we use Inoi R7 smartphone: Qualcomm Snapdragon 212 4 cores 1.2GHz, 2GB RAM [13], Aurora OS 3.2.2.20.

III. Available solutions

One of the requirements for libraries is that they support the C++ language. There are only a few projects that provide the corresponding API and are suitable for use in third-party applications in terms of licenses: poppler, PDFium, muPDF. These are the libraries which will be discussed in this article.

Development using a particular library requires a set of files consisting of header files containing descriptions of types, structures, functions, and object files in the form of a static or shared library. All libraries provide the ability to build into either a shared library or a static library. To get such a set of files, the source code could be built using one of the build systems: make [19], cmake [20], ninja [21], qmake [22]. These are the most common build systems. There are more exotic ones, for example, gn is the build system used by Google for their products.

The considered libraries use different build systems: Poppler — make, mupdf — cmake, PDFium — gn. The fact that PDFium uses gn makes it a bit more complicated to build than other libraries.

All libraries have a C-style interface. This implies the need for careful handling of allocated memory, because otherwise there is a possibility of the target application memory leaks. On the other hand, using Qt with its memory management features, can minimize the risks. An appropriate wrapper providing the necessary functionality is required to make full use of the Qt. Currently, only Poppler has official support for Qt [23]; the rest have only third party wrappers, with no performance guarantees. However, a custom wrapper may well be implemented if necessary.

Qt provides the QtPDF module for displaying and interacting with PDF files [24]. Unfortunately, this module cannot be used in conjunction with the Aurora OS in third-party applications for the following reasons:

1) The standard QtPDF builds are done since of Qt 5.15, whereas only Qt 5.6 is available for the Aurora OS.
2) QtPDF is distributed under a dual license: GPLv3 or commercial. Both of the approaches are inconvenient.

It is interesting to note that QtPDF uses PDFium to handle PDF format.

A. Poppler

Poppler [25] appeared as a fork of the Xpdf program in 2005, but it was not until 2011 (with version 0.18) that it began to be a full implementation of ISO 32000-1, the PDF format standard [26], [27]. It is the first major free library for working with PDF and is currently being developed by the freedesktop.org project. The library is distributed under GPLv2 [28].

The Poppler can use one of two backends to generate images: Cario [29] or Splash [30]. Depending on the backend you use, the available rendering functions will also change. Despite the backend choice, the speed of the Poppler is considered slow. The advantages are stability, prevalence.

Poppler is included as standard in many Linux distributions. The source code can be found in the official repository [31]. Builds are done with cmake, which is also available on most distributions. Poppler comes with utilities to work with PDF files:

- `pdfdetach` extracts embedded documents;
- `pdffonts` enumerates all the fonts used in the document;
- `pdfimages` extracts all the embedded images in the source resolution;
- `pdfinfo` listings all the meta information;
- `pdfseparate` extracts single pages;
- `pdftocairo` converts pages to vector graphics or bitmaps using cairo;
- `pdftohtml` converts to HTML saving the formatting;
- `pdftoppm` converts a page to a bitmap;
- `pdftops` converts the document to the PS format useful for printing;
- `pdftotext` extracts the document text;
- `pdfunite` unites several documents.

Poppler library provides functions to open a document, render pages, work with annotations, extract text from the page, and extract meta information.

The code for rendering a document page looks like this:

```
auto *popplerDocument = Poppler::Document::
    load(qUtf8Printable(documentPath));
auto *popplerPage = popplerDocument->page(0);
QImage pageImage =
    popplerPage->renderToImage();
pageImage.save(savePath, "PNG");
```

Poppler supports render settings:

- smoothing text and graphics;
- rotate a page to a fixed angle;
- rendering a part of a page.

In addition to C++, there are Poppler bindings for other programming languages: JavaScript, C#, php, Lisp.

B. PDFium

The PDFium library was originally developed by Foxit Software Incorporated as an open source version of their commercial Foxit PDF SDK [32], and was opened by Google to the community in 2014. It is part of Chromium and is actively supported [33]. It is distributed under the Apache v2.0 license [34], which allows you to use it in any project.

The source code of PDFium is publicly available in the repository [35]. To synchronize the source code and build the library, custom tools are used: gclient and gn. It also requires ninja build system, which is more exotic than make and is available not for all distributions by default.

PDFium provides not only the functions of rendering PDF documents, but also the ability to edit and create new documents. The library implements methods of opening password-protected documents, providing detailed information about the file. Work with annotations and text on pages is also supported. PDFium API is stable and contains the necessary set of functions for full-scale work with documents.

Library methods use the `FPDF_*` prefix. Any work with PDFium begins with initializing the library and ends with closing it:

```
FPDF_InitLibrary();
...
FPDF_DestroyLibrary();
```

The `FPDF_Load*` functions are used to load a document and a page:

```
auto document =
    FPDF_LoadDocument(path, password);
auto page =
    FPDF_LoadPage(document, pageNumber);
```

When you are done with a document or page, you must close them with the `FPDF_Close*` functions:

```
FPDF_CloseDocument(document);
FPDF_ClosePage(page);
```

The `FPDF_GetPage*` methods of obtaining page dimensions are provided:

```
auto pageWidth = FPDF_GetPageWidthF(page);
auto pageHeight = FPDF_GetPageHeightF(page);
```

The page is rendered into an image using the buffer created by the `FPDFBitmap_CreateEx` function:

```
FPDF_BITMAP bitmap = FPDFBitmap_CreateEx(...);
FPDF_RenderPageBitmap(bitmap, page, 0, 0,
    pageWidth, pageHeight, 0, 0);
```

Render functions also accept setup flags, for example:

- smoothing of text and graphic;
- discolouration;
- rotation of a page to any angle
- mirroring.

In addition to C++, there are PDFium wrappers for other programming languages: C#, Python, JavaScript, etc.

C. MuPDF

MuPDF has been developed by Artifex Software, Inc since 2005 [36]. The library emphasizes speed, lightweighting, and rendering quality. It is included in many distributions by default. In addition to the library, a set of command line utilities is also supplied.

The MuPDF implementation is mostly in C. The source code is distributed under the AGPLv3 [37]. All that is required to build it is make.

MuPDF supports PDF 1.7, XPS documents and CBZ archives. Document attributes are also supported: transparency, encryption, hyperlinks, annotations, etc. There is support for text extraction, document editing, and creating new documents.

Document page rendering is done with minimal code:

1) context is created to store the cache and exception stack and types are registered:

```
auto *context = fz_new_context(...);
fz_register_document_handlers(context);
```

2) the document and page are opened:

```
auto *document =
    fz_open_document(context, path);
auto *page =
    fz_load_page(context, document, 0);
```

3) the buffer is created to store the result of the rendering:

```
auto *pixmap =
    fz_new_pixmap_from_page(...);
```

Like the other libraries, MuPDF supports various rendering settings.

Unlike PDFium, MuPDF was originally developed with multi-threading support, so rendering speed increases when there are multiple processing cores [38].

IV. Comparison

The following criteria will be used for the functional library comparison:

- rendering speed;
- rendering quality.

A. Rendering speed

PDF documents were used to estimate the rendering speed:

- 37 pages, the pages contain many vector primitives that negatively affect the rendering speed,
- 1713 pages, consisting mainly of text and tables.

The documents were processed by each library. Each page was rendered in its original size. The testing was done without using multithreading.

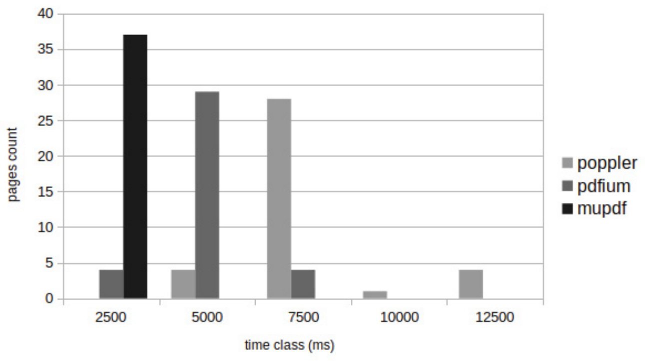

Fig. 2. Document pages rendering speed: 37 heavy pages

The results of document processing are shown in the graphs (Fig. 2 and Fig. 3). MuPDF processes document pages noticeably faster than other libraries (2.5 seconds for all pages of the first document and 10 seconds for almost all pages of the second document). For light pages, Popple and PDFium are almost equal in speed. At the same time, Poppler requires

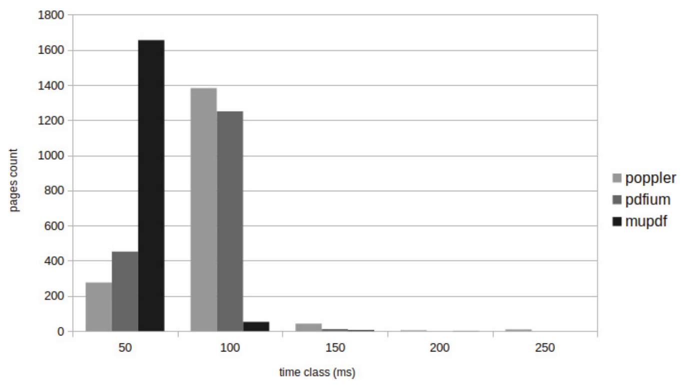

Fig. 3. Document pages render speed: 1713 light pages

TABLE I. DOCUMENT PAGES RENDER TIME (MS): 37 HEAVY PAGES

	Poppler	PDFium	MuPDF
min	2590	1525	308
max	12350	7050	1538
mean	5879	3476	511
total	235766	137053	24060

significantly more time to render heavy pages. Similar results can be seen in the summary tables I and II.

The summary of this criterion is as follows. Poppler is slower than the other libraries, regardless of document content. PDFium renders faster than Poppler, but the simpler document content makes the gap smaller. MuPDF is the fastest of the libraries, no matter what the content of the document is, it takes less time to process it.

B. Rendering quality

To evaluate the quality of rendering, a document was prepared using the Inkscape graphic editor [39] and exported in the PDF and PNG formats. The PSNR (peak signal-to-noise ratio) [40]–[42] value was calculated for the PNG file obtained from Inkscape, and images obtained using libraries from the PDF file. The higher the PSNR value, the more accurate the rendering is. The console utility magick [43] was used to calculate it. The pages for comparison were rendered with the default library settings.

Running the comparison (using PDFium as an example) was done in a such way:

```
libs@test magick compare -metric PSNR  pdfium/
    render_page_0.png pdfium/original.png pdfium/
    different.png
```

For each of the libraries, the collages of images were prepared:

- top image is the result of rendering;
- center image is the reference image,
- bottom image is the difference map generated by magick.

The result for Poppler is shown in Fig. 4. The output is a grainy picture with coarse curves. The PSNR value is the lowest, equal to 17.47.

TABLE II. DOCUMENT PAGES RENDER TIME (MS): 1713 LIGHT PAGES

	Poppler	PDFium	MuPDF
min	37	20	4
max	247	146	164
mean	60	58	19
total	109377	99515	38040

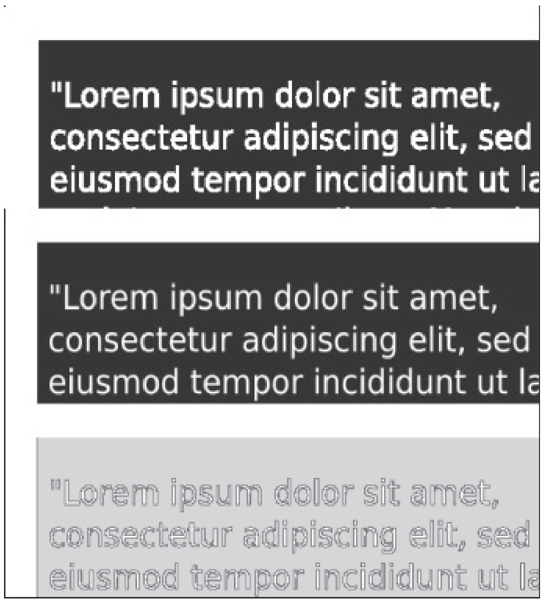
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la

Fig. 4. Poppler collage

The result for PDFium is shown in Fig. 5. This text is rendered quite well, The curves are smoothed, the lines are of the right thickness. The PSNR value is 29.85.

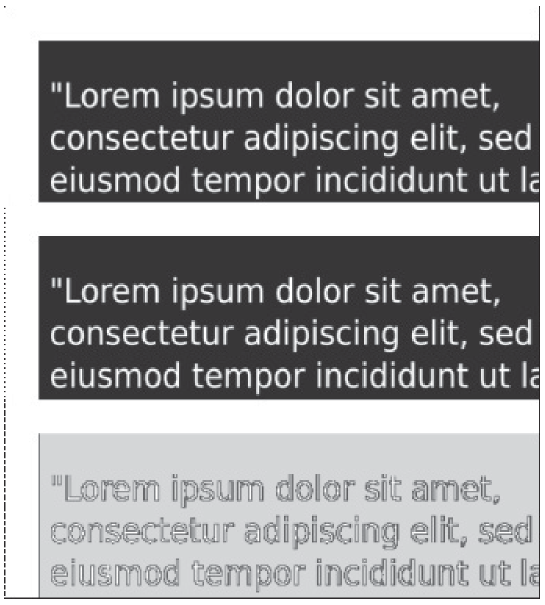
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la

Fig. 5. PDFium collage

The result for MuPDF is shown in Figu. 6. The MuPDF rendering quality is very close to that of PDFium, same neat work with curves, correct line thicknesses. The PSNR value of 24.02 confirms these conclusions.

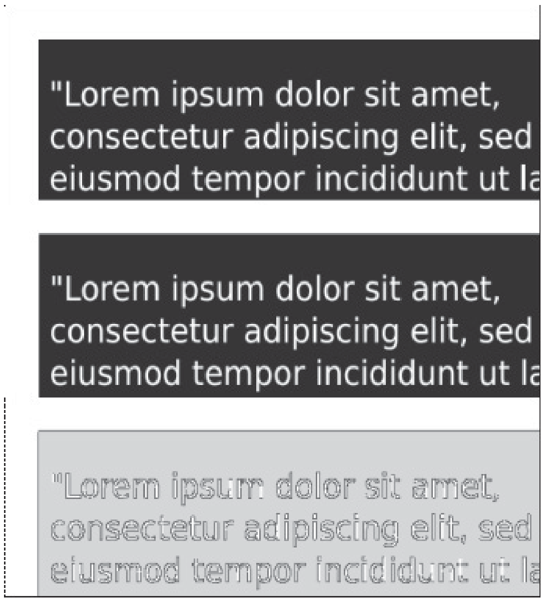
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la
"Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
eiusmod tempor incididunt ut la

Fig. 6. MuPDF collage

It is worth noting that the size of rendered images for all libraries except Poppler differs from the original. The sizes are given in the table III.

TABLE III. THE PSNR VALUES AND SIZES OF THE RENDERED IMAGES

	Original	Poppler	PDFium	MuPDF
PSNR		17.47	29.85	24.02
Size	595x842	595x842	595x841	596x842

Based on the results of the tests (see PSNR values in table III) we can conclude, that the rendering quality of PDFium and MuPDF is high. However, the size of the resulting image slightly differs from the size of the original page, which can potentially affect the proportions. At the same time Poppler, in spite of the correct size processing, is significantly inferior in quality.

V. CONCLUSION

Three libraries are available for work with PDF files using C++ in Aurora OS: Poppler, PDFium, MuPDF. They offer similar functionality to satisfy the requirements of application software.

Poppler, despite being used in a large number of Linux distributions, has a slow rendering speed and insufficient image quality. In addition, its license requires disclosure of the application source code, which is not always acceptable.

MuPDF provides good quality and very high speed. However, the license also restricts its free use in third-party software.

PDFium offers the best rendering quality and a respectable speed. In addition, its license is free enough for third-party applications to use this library. We recommend to use it to work with PDF files in Aurora OS.

REFERENCES

[1] Aurora os website. [Online]. Available: https://auroraos.ru/
[2] Document management — portable document format — part 1: Pdf 1.7. [Online]. Available: https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/PDF32000_2008.pdf
[3] K. Thomas, “Portable document format: An introduction for programmers,” *MacTech Magazine*, vol. 15, no. 9, 1999.
[4] D. Lukan. Pdf file format: Basic structure [updated 2020] - infosec resources. [Online]. Available: https://resources.infosecinstitute.com/topic/pdf-file-format-basic-structure/
[5] List of pdf software - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/List_of_PDF_software#Linux_and_Unix
[6] Posixp1003.1 - standard for information technology–portable operating system interface (posix(tm)) base specifications, issue 8. [Online]. Available: https://standards.ieee.org/project/1003_1.html
[7] Posixworking draft, standard for programming language c++. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf
[8] Qt — cross-platform software development for embedded & desktop. [Online]. Available: https://qt.io
[9] The gnu general public license v3.0 - gnu project - free software foundation. [Online]. Available: http://www.gnu.org/licenses/gpl-3.0.html
[10] Mit license definition. [Online]. Available: http://www.linfo.org/mitlicense.html
[11] Bsd license definition. [Online]. Available: http://www.linfo.org/bsdlicense.html
[12] Bsdbuffer texture - opengl wiki. [Online]. Available: https://www.khronos.org/opengl/wiki/Buffer_Texture
[13] Inoi r7 product page. [Online]. Available: https://inoi.com/2344/inoi-r7/
[14] Accelerating deployment of arm-based solutions - linaro. [Online]. Available: https://www.linaro.org
[15] Scratchbox2 on mer wiki. [Online]. Available: https://wiki.merproject.org/wiki/SB2
[16] Qml applications — qt 5.15. [Online]. Available: https://doc.qt.io/qt-5/qmlapplications.html
[17] D. Laure, A. Vasilyev, I. Paramonov, and N. Kasatkina, “Pdfcross-platform development for sailfish os and android: Architectural patterns and “dictionary trainer” application case study,” in *19th Conference of Open Innovations Association*. FRUCT, 2016, pp. 145–150.
[18] Model–view–viewmodel - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
[19] Gnu make. [Online]. Available: https://www.gnu.org/software/make/manual/make.html
[20] Cmake. [Online]. Available: https://cmake.org/
[21] Ninja, a small build system with a focus on speed. [Online]. Available: https://ninja-build.org/
[22] qmake manual. [Online]. Available: https://doc.qt.io/qt-5/qmake-manual.html
[23] Poppler qt5: The poppler qt5 interface library. [Online]. Available: https://poppler.freedesktop.org/api/qt5/
[24] Qt pdf — qt marketplace. [Online]. Available: https://marketplace.qt.io/products/qtpdf
[25] Poppler. [Online]. Available: https://poppler.freedesktop.org
[26] Pdfiso - iso 32000-1:2008 - document management — portable document format — part 1: Pdf 1.7. [Online]. Available: https://www.iso.org/standard/51502.html
[27] B. Fanning, “Pdf standards....transitioning the pdf specification from a de facto standard to a de jure standard,” *MacTechAIIM Magazine*, July/August, 2007.
[28] Gnu general public license version 2 — open source initiative. [Online]. Available: https://opensource.org/licenses/GPL-2.0
[29] cairographics.org. [Online]. Available: https://www.cairographics.org/
[30] cairographicssplash - a javascript rendering service. [Online]. Available: https://splash.readthedocs.io/en/stable/
[31] poppler / poppler · gitlab. [Online]. Available: https://gitlab.freedesktop.org/poppler/poppler

[32] Foxit pdf sdk. [Online]. Available: https://developers.foxitsoftware.com/resources/pdf-sdk/c_api_reference_pdfium/index.html
[33] The chromium project · github. [Online]. Available: https://github.com/chromium
[34] Apache license, version 2.0 — open source initiative. [Online]. Available: https://opensource.org/licenses/Apache-2.0
[35] Github - chromium/pdfium: The pdf library used by the chromium project. [Online]. Available: https://github.com/chromium/pdfium
[36] Mupdf. [Online]. Available: https://mupdf.com/
[37] Gnu affero gpl version 3 and the ”asp loophole” — open source initiative. [Online]. Available: https://opensource.org/node/152
[38] Pdfium thread safety. [Online]. Available: https://groups.google.com/g/pdfium/c/HeZSsM_KEUk
[39] Draw freely — inkscape. [Online]. Available: https://inkscape.org/
[40] Peak signal-to-noise ratio - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
[41] S. T. Welstead, “Fractal and wavelet image compression techniques,” *SPIE Publication*, pp. 155–156, 1999.
[42] R. Hamzaoui and D. Saupe, *Document and Image Compression. Signal Processing and Communications*, M. Barni, Ed. CRC Press, 2018.
[43] Imagemagick - command-line tools: Compare. [Online]. Available: https://imagemagick.org/script/compare.php