



Ghostscript and MuPDF Status

OpenPrinting summit August 2014

Michael Vrhel, Ph.D.
Artifex Software Inc.
San Rafael CA



Ghostscript overview

What is new...

Color Architecture Details

Threading and GS

MuPDF



The Basics

Ghostscript is a document conversion and rendering engine.

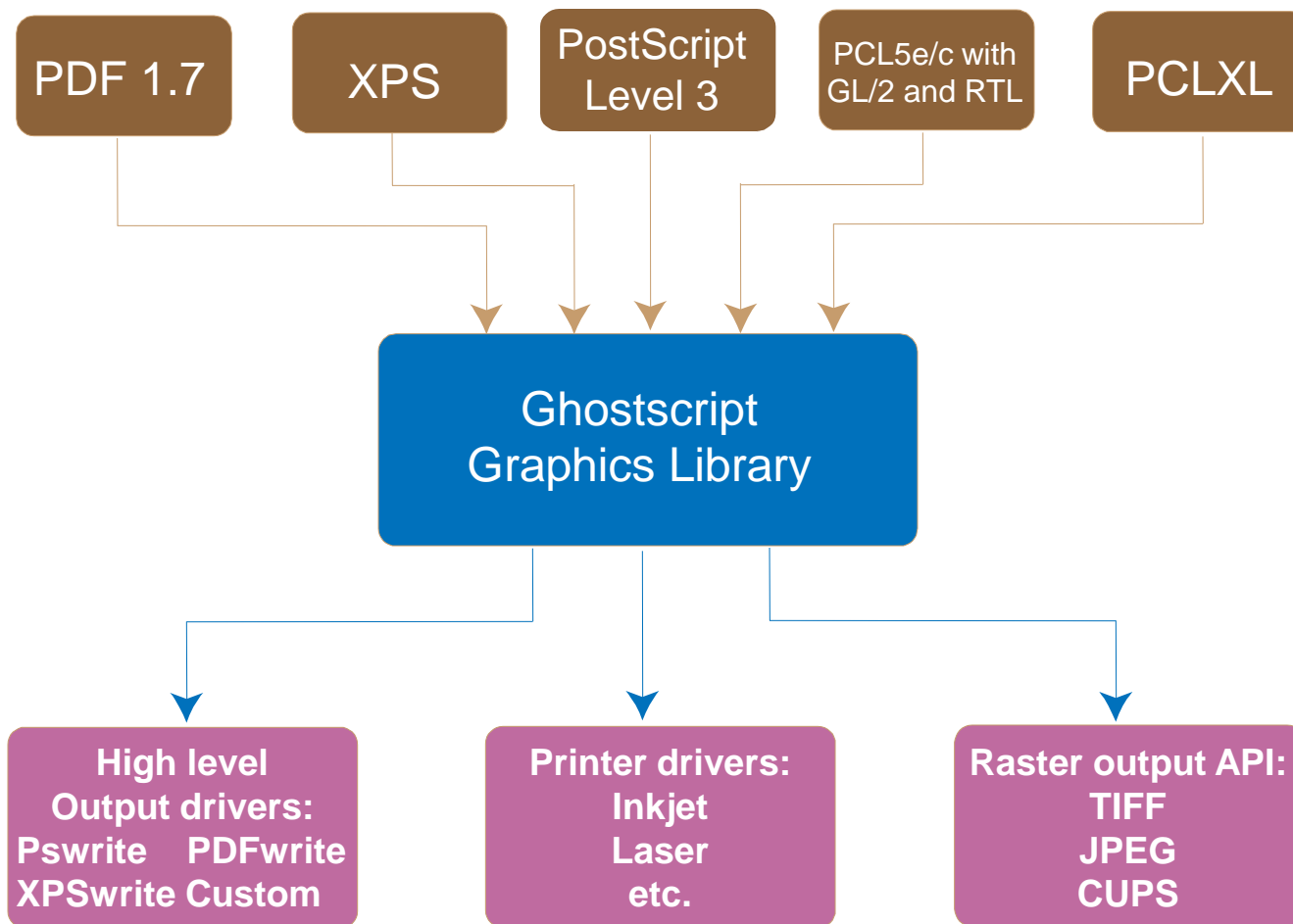
Written in C ANSI 1989 standard (ANS X3.159-1989)

Essential component of the Linux printing pipeline.

Dual AGPL/Proprietary licensed. Artifex owns the copyright.

Source and documentation available at www.ghostscript.com

Graphical Overview



Understanding devices is a major key to understanding Ghostscript.

Devices can have high-level functionality. e.g. pdfwrite can handle text, images, patterns, shading, fills, strokes and transparency directly.

Devices may be set up to handle only certain high-level operations.

Graphics library has “default” operations. e.g. text turns into bitmaps, images decomposed into rectangles.

In embedded environments, calls into hardware can be made.

Raster devices require the graphics library to do all the rendering.

Relevant Changes to GS since last meeting....

The upcoming page can now be interpreted to a display list while the current page display list is being rendered on multiple threads (9.10)

`-dBGPrint = true/false`

It is now possible to monitor the content for the presence of color. If no color is detected the buffers are converted to gray scale. This is primarily aimed at workflows where saving ink (especially color inks) is required. (9.10)

`-dGrayDetection=true/false`

Support added to build Ghostscript DLL for WinRT for x86, x64 and ARM (Requires MS Visual Studio 2012 Pro). (9.10)

Relevant Changes to GS since last meeting....

The URW Postscript font set has been updated to the latest version, fixing many compatibility problems with the Adobe fonts. (9.10)

Large transparency flattening cases now handled by using a display list for the transparency enabling the use of small bands, reducing memory use. (9.14)

pdfwrite now uses the same color management engine as Ghostscript rendering devices (by default LCMS2). (9.14)

A new device 'eps2write' has been added which allows for the creation of EPS files using the ps2write device instead of the old (deprecated and removed) pswrite device. (9.14)

Old epswrite device is now deprecated and will be removed in a future release.

Relevant Changes to GS since last meeting....

Ghostscript can now collect information for pages in temp files (in Ghostscript's clist format), then render and output pages for the job in arbitrary order, including normal, reverse, odd, even, or subset of the pages. (9.14)

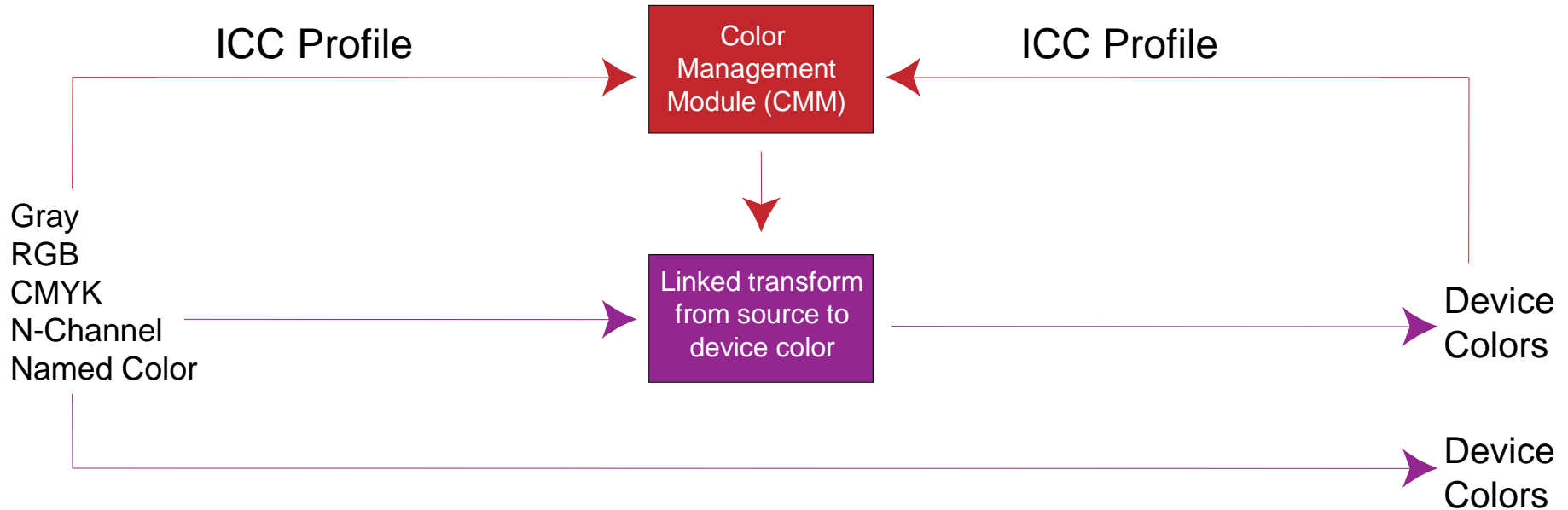
--saved-page=___ option.

The Ghostscript device architecture has been extended so that, when rendering bands into multiple threads, possible to perform post-processing in multiple threads, such as downscale, post-render halftoning, or compression. (9.14)

The CUPS device now has improved support for PPD-less printing (9.14)

Upcoming: Full ICC color support in pdfwrite device. (9.15)

Ghostscript Color Flow



Ghostscript Color Architecture

- Easy to interface different CMM with Ghostscript.
- ALL color spaces defined in terms of ICC profiles.
- Linked transformations and internally generated profiles cached.
- Easily accessed manager for ICC profiles.
- Easy to specify default profiles for DeviceGray, DeviceRGB and DeviceCMYK.
- Devices communicate their ICC profiles and have their ICC profile set.
- Operates efficiently in a multithreaded environment.
- Handles named colors with ICC named color profile or proprietary format.
- ICC Color management of Device-N colors or customizable spot handling.
- Includes object type (e.g. image, graphic, text) and rendering intent into the computation of the linked transform. Maintained with transparency.

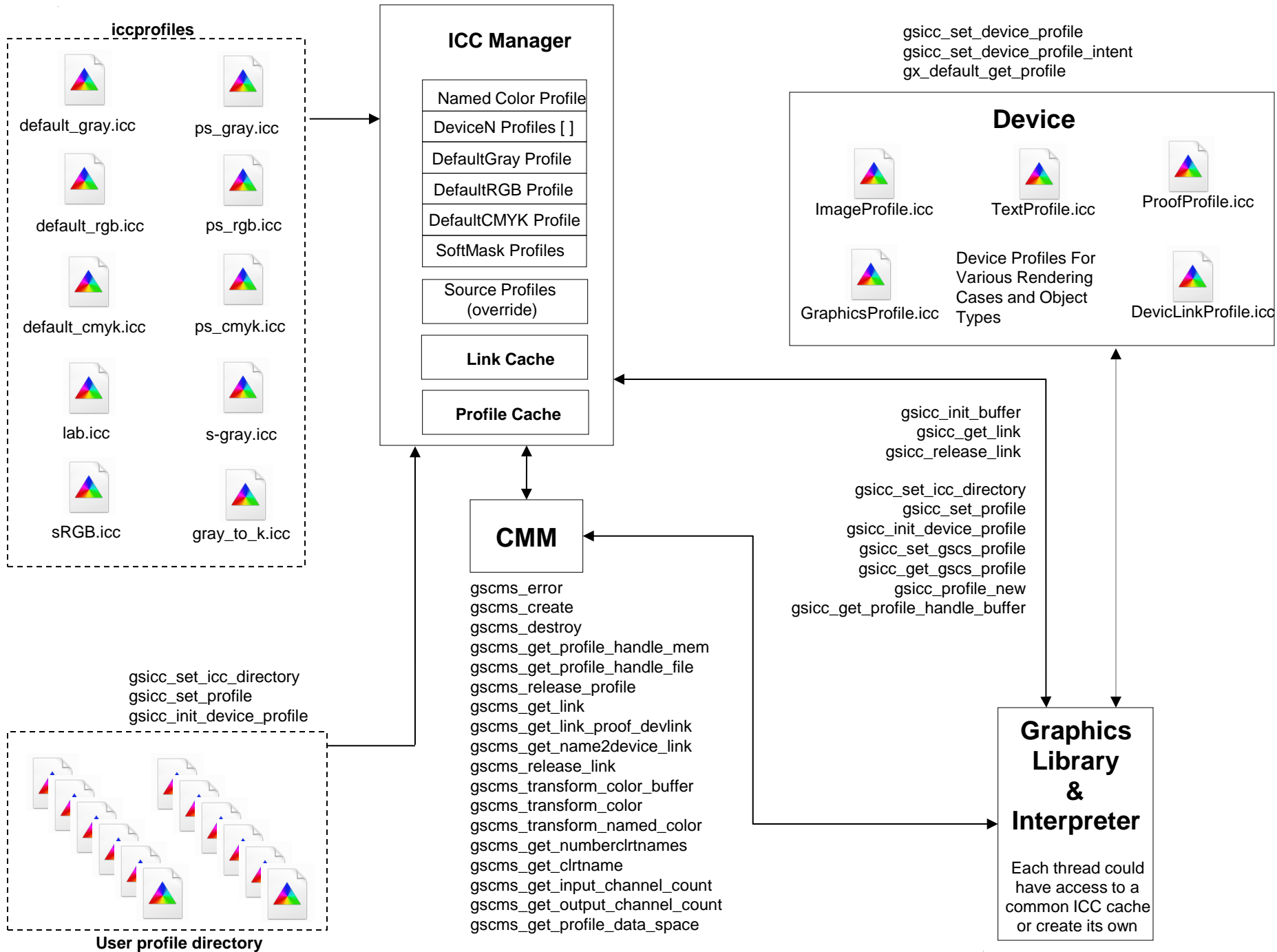
Ghostscript Color Architecture

- Ability to override document embedded ICC profiles with Ghostscript's default ICC profiles.
- Easy to specify unique **source** ICC profiles to use with CMYK and RGB graphic, image and text objects.
- Easy to specify unique **destination** ICC profiles to use with graphic, image and text objects.
- Easy to specify different rendering intents (perceptual, colorimetric, saturation, absolute colorimetric) and black point comp. for graphic, image and text objects.
- Control to force gray source colors to black ink only for devices that support black ink (e.g. CMYK).



Ghostscript Color Architecture

- Make use of PDF output intent ICC profile.
- Use an NCLR ICC output profile when rendering to a separation device.
- Make use of device link ICC profiles for direct mapping of source colors to the device color space.
- Ability to make use of device link ICC profiles for retargeting from SWOP/Fogra standard color space to a specific device color space.



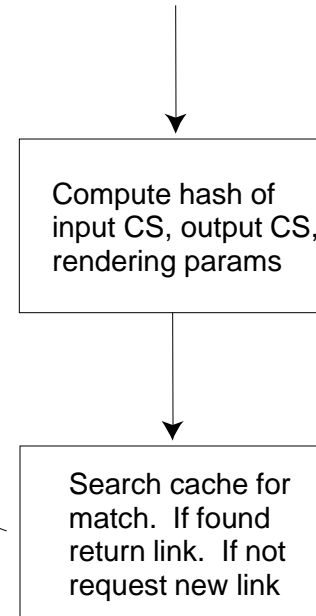
Link Cache

GRAPHICS LIBRARY

```
gsicc_get_link(* pis, *input_colorspace, *output_colorspace, *rendering_params,  
memory, include_softproof)
```

Link Cache

Hash Code	Ref Count	Link Structure
Hash Code	Ref Count	Link Structure
Hash Code	Ref Count	Link Structure
Hash Code	Ref Count	Link Structure
	.	
	.	
	.	
	.	
Hash Code	Ref Count	Link Structure



Link entries are reference counted.

Conversion of PS and PDF Color Spaces

- Ghostscript creates ICC profiles from PDF and PS CIE colorspace definitions (e.g. CalRGB, CIEABC, CIEDEFG)
- To avoid repeated creations, these profiles are cached based upon a hash code that is related to the resource ID.
- Cache is designed such that MRU item is at the top of the list.

Device N color spaces (PDF and PS)

- For Device N output, very simple to provide capability for N-color ICC profile.
- Many desire to have CM with CMYK and to pass additional spot colors unmolested.
- For DeviceN input color, XPS requires ICC profile. PDF and PS use an alternate tint transform.
- Architecture provides capability to define N-color ICC profile for DeviceN input colors to replace the alternate tint transform if desired.
- Named color custom support is possible for DeviceN source colors. Example implementation is given in `gs\toolbin\color\named_color` folder

Current Color Command Line Interface

Source Default Profiles

-sDefaultGrayProfile = my_gray_profile.icc
-sDefaultRGBProfile = my_rgb_profile.icc
-sDefaultCMYKProfile = my_cmyk_profile.icc
-sDeviceNProfile = my_devicen.icc
-sNamedProfile = my_namedcolor_profile.icc

Device Profile

-sOutputICCPProfile = my_device_profile.icc

ICC Search Directory

-sICCProfilesDir = c:/my_iccprofiles/

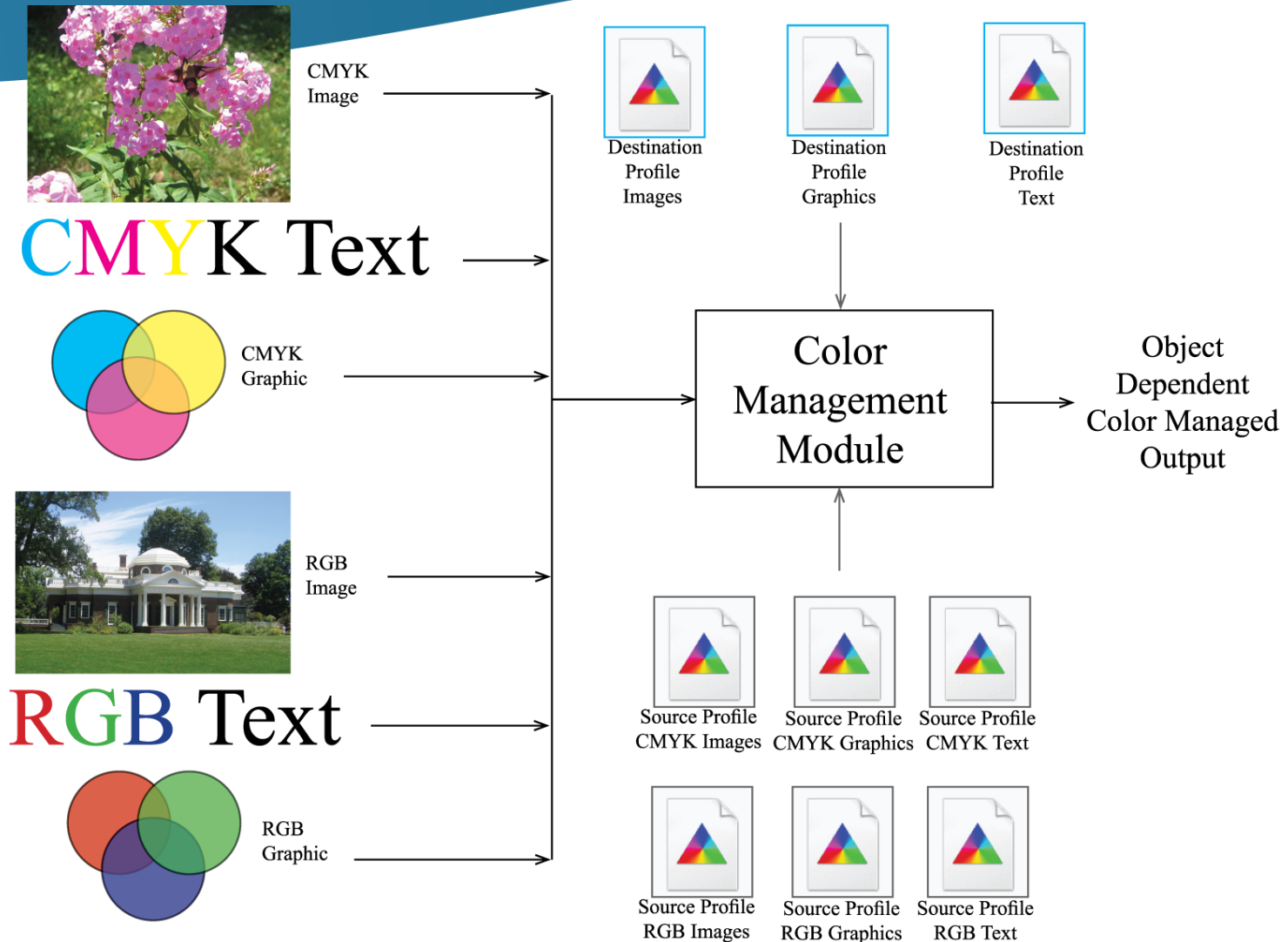
Current Color Command Line Interface

Other Settings

- sProofProfile = my_proof_profile.icc
- sDeviceLinkProfile = my_link_profile.icc
- dRenderIntent = intent (0, 1, 2, 3)
- dOverrideICC = true/false
- dDeviceGrayToK = true/false
- dUseFastColor = true/false
- dBlackPtComp = 0 / 1
- dKPreserve = 0 / 1 / 2
- dSimulateOverprint = true/false
- dUsePDFX3Profile = int

- sICCTransformColors = "Cyan, Magenta, Yellow, Black, Orange, Violet"

Object Dependent Color Management



Object Dependent Color Management Source Profiles

Source object dependent control achieved through the command line Specification:

`-sSourceObjectICC` = filename

Contents of this file define what source profiles and settings should be used with what objects

Key	Profile	Intent	BlackPtComp	Override	BlackPreserve
Graphic CMYK	cmyk_src_graphic.icc	0	1	0	0
Image CMYK	cmyk_src_image.icc	0	1	0	0
Text CMYK	cmyk_src_text.icc	0	1	0	0
Graphic RGB	rgb_source_graphic.icc	0	1	0	
Image RGB	rgb_source_image.icc	0	1	0	
Text RGB	rgb_source_text.icc	0	1	0	

Object Dependent Color Management Destination Profiles

Destination object dependent control achieved through the command line

-sTextICCProfile = my_device_text_profile.icc
-sGraphicICCProfile = my_device_graphic_profile.icc
-sImageICCProfile = my_device_image_profile.icc

-dTextIntent = intent (0, 1, 2, 3)
-dGraphicIntent = intent (0, 1, 2, 3)
-dImageIntent = intent (0, 1, 2, 3)

-sTextBlackPt = 0/1
-sGraphicBlackPt = 0/1
-sImageBlackPt = 0/1

-sTextKPreserve = 0/1/2
-sGraphicKPreserve = 0/1/2
-sImageKPreserve = 0/1/2

Example: Object Dependent CM Default Profiles



RGB Image



RGB Graphic

Source file includes RGB and CMYK
Images, graphics and text.

RGB TEXT



CMYK Image



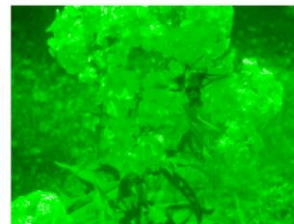
CMYK Graphic

CMYK TEXT

Example: Object Dependent CM Source Profiles Vary

In this case, different ICC profiles were specified to be used with RGB and CMYK graphic, image, and text objects via the Ghostscript command line with `-sSourceObjectICC = filename`.

Graphic_CMYK	cmyk_src_cyan.icc	0 1 0 0
Image_CMYK	cmyk_src_magenta.icc	0 1 0 0
Text_CMYK	cmyk_src_yellow.icc	0 1 0 0
Graphic_RGB	rgb_source_red.icc	0 1 0
Image_RGB	rgb_source_green.icc	0 1 0
Text_RGB	rgb_source_blue.icc	0 1 0

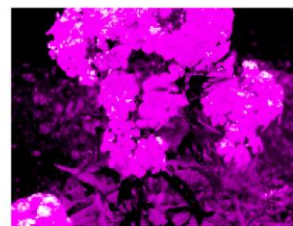


RGB Image



RGB Graphic

RGB TEXT



CMYK Image



CMYK Graphic

CMYK TEXT

Example: Object Dependent CM

Source CMYK rendering intent varies

In this case, a special source ICC profile for CMYK objects was specified via the Ghostscript command line. The profile was designed to give radically different results in different rendering intents.

Different rendering intents used for CMYK graphics, images and text

Graphic_CMYK	cmyk_src_renderintent.icc	0	1	0	0
Image_CMYK	cmyk_src_renderintent.icc	1	1	0	0
Text_CMYK	cmyk_src_renderintent.icc	2	1	0	0



RGB Image



RGB Graphic

RGB TEXT



CMYK Image



CMYK Graphic

CMYK TEXT

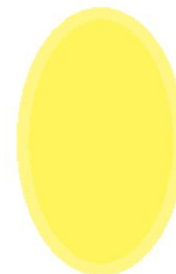
Example: Object Dependent CM Destination Profile varies

Different destination profiles
specified for different objects

- sGraphicICCProle = yellow_output.icc
- sImageICCProle = magenta_output.icc
- sTextICCProle = cyan_output.icc



RGB Image



RGB Graphic

RGB TEXT



CMYK Image



CMYK Graphic

CMYK TEXT

Example: Object Dependent CM Destination Intent varies

In this case, a special source ICC profile for CMYK objects was specified via the Ghostscript command line.

Different rendering intents used for graphics, images and text

```
-sGraphicICCProle = cmyk_des_renderintent.icc  
-sImageICCProle  = cmyk_des_renderintent.icc  
-sTextICCProle   = cmyk_des_renderintent.icc  
-dImageIntent    = 0  
-dGraphicIntent  = 1  
-dTextIntent     = 2
```



RGB Image



RGB Graphic

RGB TEXT



CMYK Image



CMYK Graphic

CMYK TEXT

Proof and DeviceLink ICC Profile Usage

Two situations:

- 1) Can I print (or display) on device B what my output will look like if I were to print on device A?

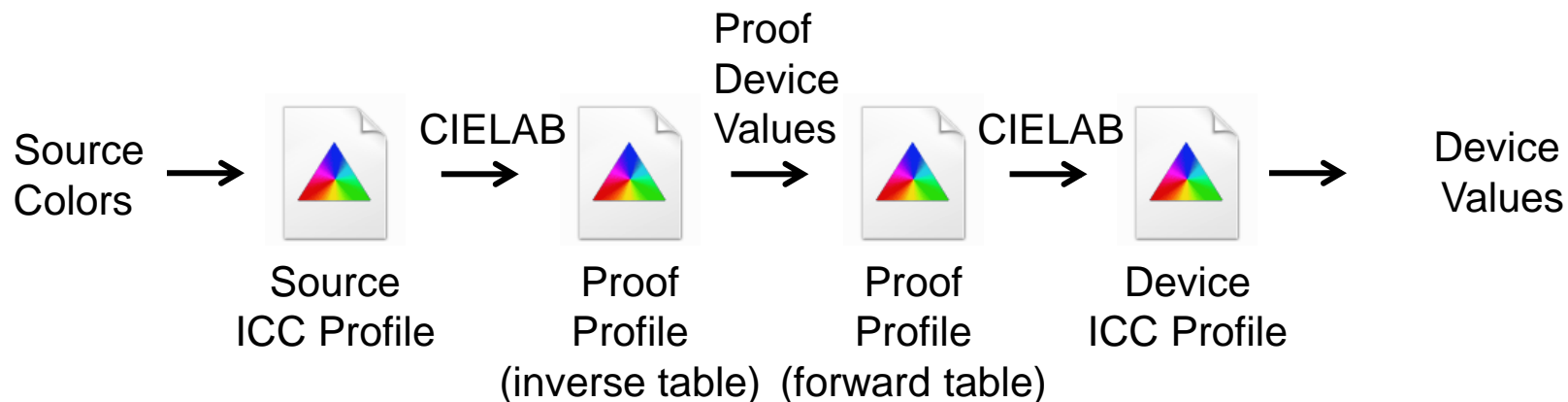
Use a proofing profile.

- 2) Can I map my output to a common standard space (e.g. Forgra39) and then perform a device link transform to my actual device values?

Use a device-link profile.

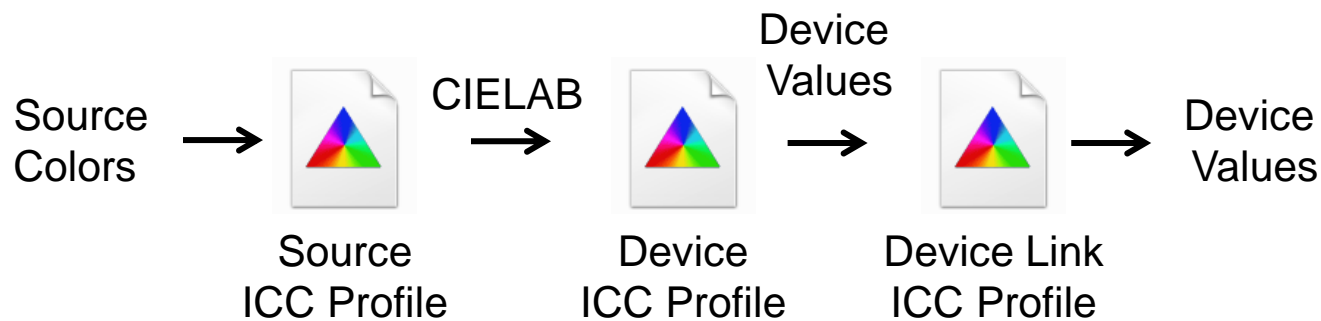
Proof and DeviceLink ICC Profile Usage

Proof Profile Only Case:



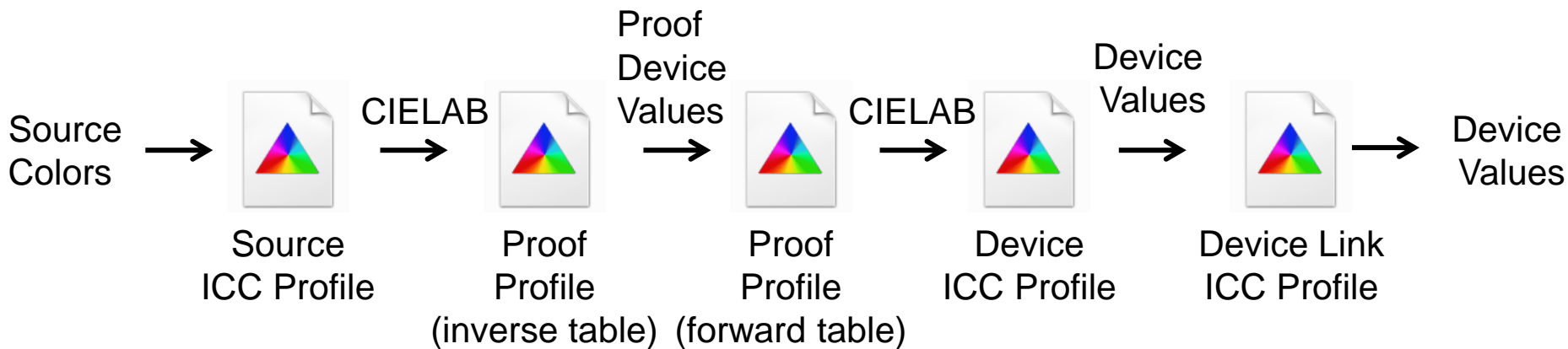
Proof and DeviceLink ICC Profile Usage

Device Link Profile Only Case:



Proof and DeviceLink ICC Profile Usage

Both proofing and device-link profile.

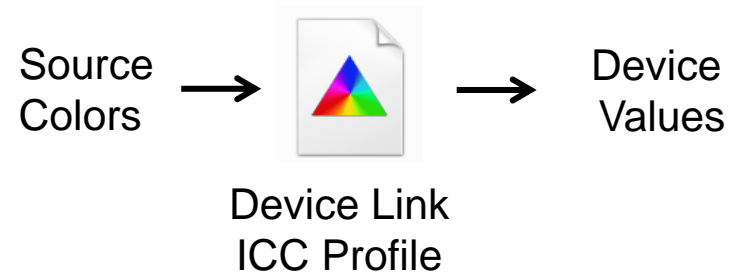


DeviceLink ICC Profile Usage

Use Device Link Profile Directly to output: Specified with `-sSourceObjectICC`

Graphic_RGB linkRGBtoCMYK.icc 0 1 0

Note output can be DeviceN.



Note that if `-sDeviceLinkProfile` is also used then you have:



Special Color Handling

The color transformation operations have a simple API.

A link (which transforms colors from one space to another) has a structure called `gscms_procs_t` which contains three functions.

```
void (*gscms_trans_color_proc_t) (gx_device*, gsicc_link_t*,  
                                   void *inputcolor, void *outputcolor, int num_bytes);
```

```
void (*gscms_trans_buffer_proc_t) (gx_device*, gsicc_link_t*,  
                                   gsicc_bufferdesc_t *input_buff_desc,  
                                   gsicc_bufferdesc_t *output_buff_desc,  
                                   void *inputbuffer, void *outputbuffer);
```

```
void (*gscms_link_free_proc_t) (gsicc_link_t *icclink);
```

Special Color Handling Continued

It is simple to replace these procedures with other ones to do special color handling/customization with minimal code writing.

Ghostscript includes three examples:

`gsicc_nocm.c`

This contains code which uses 255-X style color management.

`gsicc_monitorcm.c`

This contains code to monitor the colors encountered during management. Used to monitor for non-gray content.





`gsicc_replacecm.c`

This contains code to replace the incoming colors in some “special” way.

PDF Output Rendering Intent

GS supports PDF Output Rendering Intent usage Passes all Ghent tests and these are included in the testing suite.

GWG 13.1 - Rendering Intents




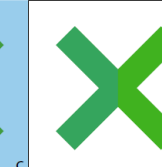
Perceptual	relative Colorimetric	Absolute Colorimetric	Saturation
			
a	b	c	d
0/0/87/0 (yellow)	80/0/00/0 (cyan)	35/0/0/0 (light cyan)	0/0/0/0 (white)

Output Intent: RenderingIntentRestProfile
File is not suitable for amalgamation - see notes for further information

If colored **X**'s appears Rendering Intent was not respected.

02 Nov 2007 Ghent PDF Workgroup © www.gwg.org 13.1

GWG 13.1 - Rendering Intents

Perceptual	relative Colorimetric	Absolute Colorimetric	Saturation
			
a	b	c	d
0/0/87/0 (yellow)	80/0/00/0 (cyan)	35/0/0/0 (light cyan)	0/0/0/0 (white)

Output Intent: RenderingIntentRestProfile
File is not suitable for amalgamation - see notes for further information

If colored **X**'s appears Rendering Intent was not respected.

02 Nov 2007 Ghent PDF Workgroup © www.gwg.org 13.1

OutputIntents array (Optional; PDF 1.4) An array of output intent dictionaries describing the color characteristics of output devices on which the document might be rendered (see "Output Intents" on page 970).

PDF Output Rendering Intent

`-dUsePDFX3Profile = #`

Where # defines which output intent to use in the order that they occur in the document. If no number specified, first one encountered is used.

If no profile is present in the intent dictionary, a warning is displayed and the rendering intent is ignored.

If the output intent ICC profile does not match the process color model of the output device, then the output intent ICC profile is used as a proofing profile.

Ghostscript's Display List and Threads

On memory limited devices ghostscript will subdivide a job into bands to reduce peak memory usage. This will occur for raster output devices.

Following our device-centric world, the display list is a high-level device. output can be stored in memory or on disk.

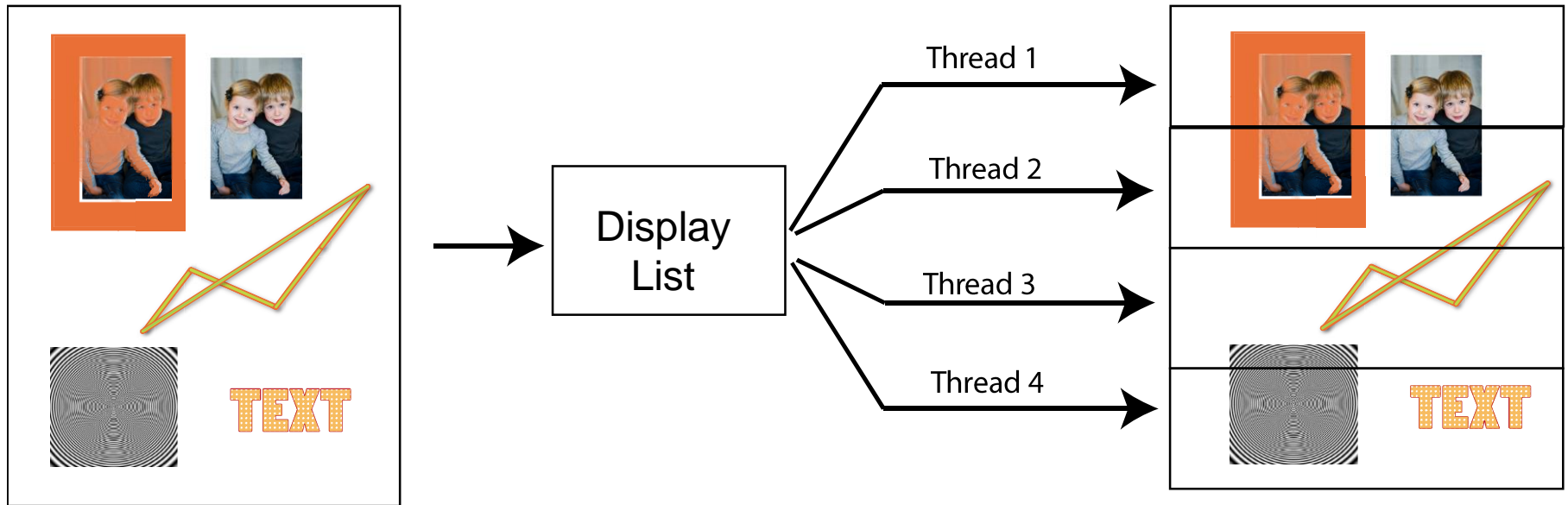
Memory-based display list can be compressed if RAM resources become critical. LRU cache is used with decompression to maximize performance.

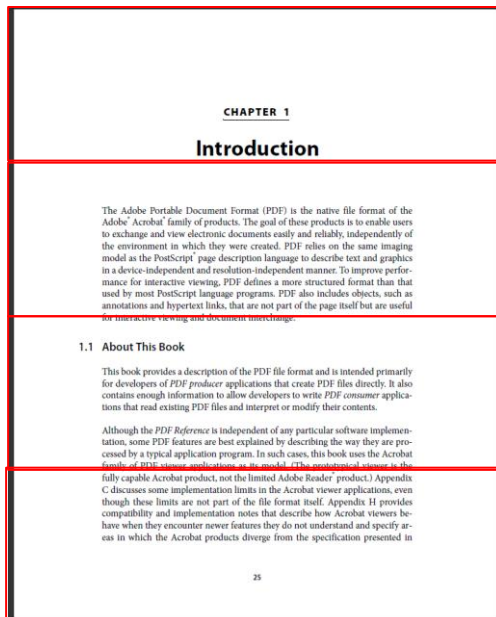
Disk-based display list is optimized to prevent seeking during reading.

Encoded in the display list output are commands and the bands in which they occur.

Each band can be rendered as a separate thread, providing significant performance improvement for multi-core systems.

Display List Multithreaded Rendering





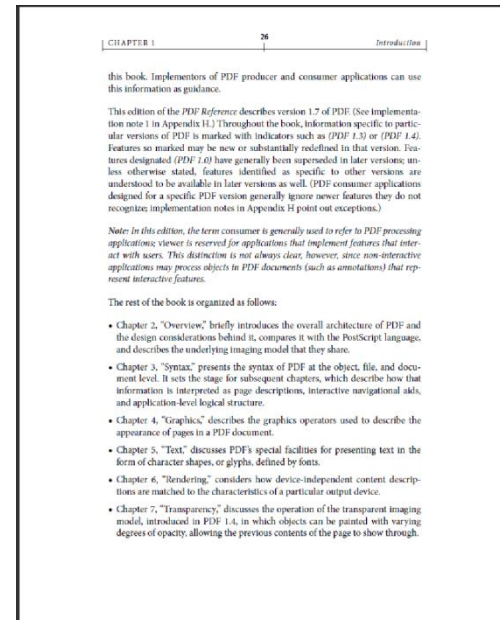
→ Thread 1

→ Thread 2

→ Thread 3

→ Thread 4

Display list of page 25 being rendered into bands by 4 threads. Note that numbers bands need not equal number of threads.



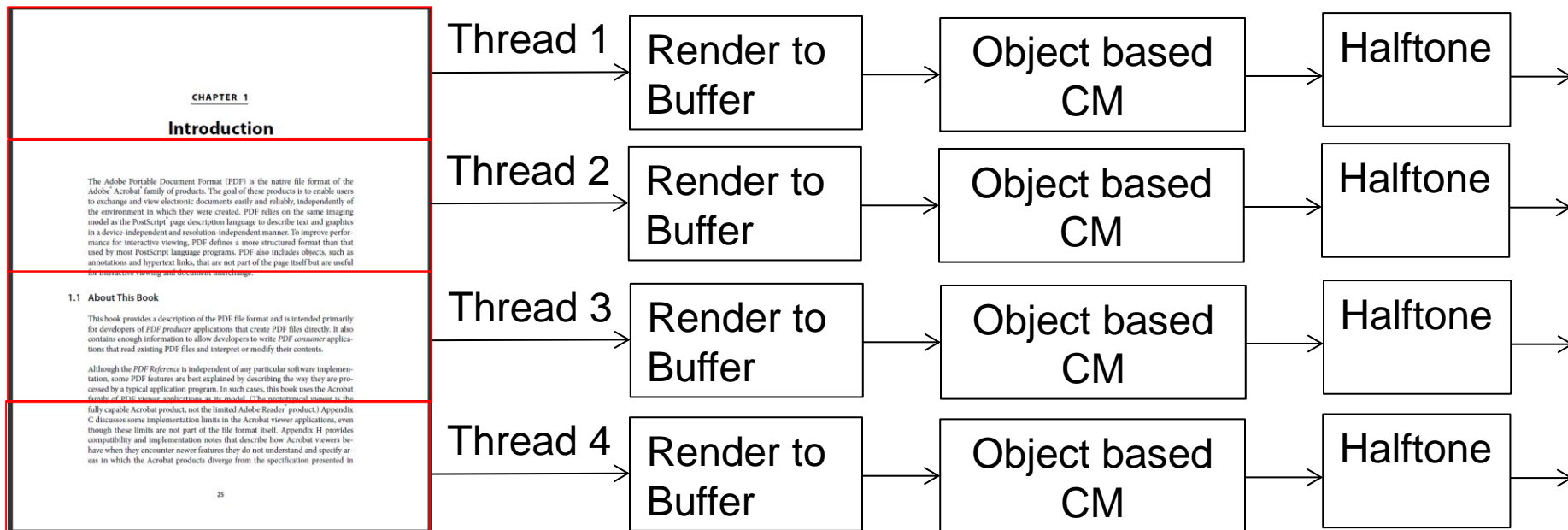
→ Thread 5

Page 26 being interpreted and display list created with another thread.



GS and threads

Post processing on rendered band buffer can be performed on thread used to render the particular band. These methods are defined in the device. `gdevcmymkog.c` provides an example implementation.



Bug Tracking

<http://bugs.ghostscript.com/>

Bugzilla – Main Page version 4.2.5

[Home](#) | [New](#) | [Browse](#) | [Search](#) | [?] | [Reports](#) | [Preferences](#)
| [Administration](#) | [Log out michael.vrhel@artifex.com](#)

Welcome to Bugzilla



[File a Bug](#)



[Search](#)



[User Preferences](#)

[Quick Search help](#)

[Bugzilla User's Guide](#) | [Release Notes](#)

[Home](#) | [New](#) | [Browse](#) | [Search](#) | [?] | [Reports](#) | [Preferences](#)
| [Administration](#) | [Log out michael.vrhel@artifex.com](#)

[My Bugs](#) | [All Mine](#) | [My Closed](#) | [My Customer Bugs](#) | [regresssions_mjv](#)
[customer bugs](#)



MuPDF



What is MuPDF?



- **A core set of libraries**
 - + Entirely written in C
 - + Very portable
 - We've done: Windows/Linux/MacOS/iOS/Android
 - Third parties: BB10/QNX/others
- Various example tools that use these libraries:
 - + Simple viewers for Linux/Android/MacOS/iOS/Windows/WinRT.
 - + Command line tools for rendering PDF pages.
 - + Command line tools for manipulating PDF files
 - page extraction
 - decompression
 - repair
 - resource extraction
- Licensing the same as Ghostscript. Dual AGPL/Proprietary licensed. Artifex owns the copyright.

Why MuPDF when we have Ghostscript?



For printing on large machines - use Ghostscript.

- * Postscript
- * PCL
- * Spot colors
- * Extreme level of color management control
- * Massive range of output devices

Why MuPDF when we have Ghostscript?



For screen use or embedded devices - use MuPDF.

*** Fast**

- + PDF Parser in C.
- + AA Rendering designed in from the ground up.

*** Small**

- + Much smaller ROM footprint.

*** Simple**

- + No complex garbage collector to maintain
- + Small set of dependent libraries
- + Simpler to port

*** Interactive features**

- + More suitable for building viewers
- + Searching
- + Zooming
- + Form filling
- + Transitions

Features of MuPDF



"Complete" PDF support

- + Transparency
- + Patterns/Shadings
- + Fonts (all kinds)
- + Image formats
- + Decryption
- + Interaction (more later)

Features of MuPDF



Not just "PDF" - Other formats too:

- + XPS
- + CBZ/JPEG/PNG
- + Extensible system

"Device" interface

- + Separates interpretation from rendering
- + Allows display lists
 - interpret page once, render many times at different zooms
- + Allows format conversions (more later)

Clever memory management

- + Caching of objects (both raw and decoded)
- + Memory scavenging (throw objects away just in time)

Optional Multi-core/threading support

- + Not tied to any one threading implementation
 - All we need is locks
- + Interpretation happens on 1 thread
- + Rendering can happen on many.
 - Thumbnails rendered in the background.
 - Banded rendering of pages.
 - Resources decoded on one thread can be used by others.

Recent changes: Interactivity



Form filling

- + Javascript for validation
 - Not tied to any one javascript implementation
 - Thin veneer to Googles 'v8' engine supplied
 - muJS is our own small javascript library
- + Ability to save files back with data in them.

Google Cloud Print support added to Android app

Reflow View

- + Pages extracted to HTML (text and images).
- + Rudimentary layout detection (tables, indents, etc.)

Digital Signatures

- + Verify signatures
- + Sign documents
- + Re-sign documents after form filling

Submission of filled in forms.

- + Several different ways of doing this.
 - Send the whole filled in file.
 - Extract data as XML and send that.

Improve page extraction for reflow.

- + Improve column detection
- + Reorder lines
- + Spot captions on images
- + Preserve "line art" areas of the page as images

Output devices for format conversion.

- + PDF output (prototype code exists)
- + SVG output (prototype code exists)
- + Other possibilities include XPS or PCL.

More input devices

- + SVG seems most likely.

Color Management

- + Plan to use LCMS to enable color management.



Get Involved



If any of this sounds interesting, we'd love to hear from you.

Our development direction is driven by customers and users.

Contributions are always welcome. We have a bug bounty scheme.

Repository located at

[git://git.ghostscript.com/mupdf.git](https://git.ghostscript.com/mupdf.git)

MuPDF discussions on IRC freenode #ghostscript channel

Additional information at www.mupdf.com



Thank you for your attention!



michael.vrhel (at) artifex.com