Fonts optimization in PDF

Text is one of the fundamental tools for information interchange. In PDF the text is represented by text elements in page content **streams**. A text element specifies that characters should be drawn at certain positions. The characters are specified using various encoding techniques and font resource types. Fonts in PDF exist in two main categories. **Simple fonts** and **Composite fonts**.

In **Simple fonts**, the glyphs (outlines) are selected by single-byte character. These codes are indexed in a table of 256 glyphs. The mapping from codes to glyphs is called the font's encoding which is built in each font program.

Composite fonts use CID's (character identifier) as an intermediary step during processing. They are typically used to handle writing systems where there are a very large number of characters, such as in Japanese or Chinese writing systems.

Except for Type 3 fonts and standard Type 1 fonts, every font dictionary contains a **subsidiary dictionary**, the font descriptor, containing font-wide metrics and other attributes of the font. Among those attributes is an **optional font file stream** containing the font program.

This allows the font to be distributed with the document and avoids potential problems that can occur if the required fonts are not available on the system that is processing the PDF document. Because of this *many of the specialized PDF standards require all fonts to be embedded in the PDF document.*

This, however, comes with a potentially negative impact on document file size. Embedding typical TrueType font in a document can in many cases lead to inflating the file size by several orders of magnitude.

We can use several techniques to **reduce the size** of the font program and keep the document content and **quality unaffected**.

One of the most effective and universal is **font data stream compression**.

Font Data Stream Compression

Using this technique, we can effectively reduce the font file size without losing any information. Table 1. shows an example size ratio of an embedded font in PDF document.

Font used: Arial Regular Version 7.00 © 2017 The Monotype Corporation			
State	Size	Compressed Size (Deflate)	PDF file size (one page standard text)
Original - composite PDF font	1,036,584 Bytes	538,526 Bytes	562,266 Bytes
List of font data tables OpenType - DSIG, GDEF, GPOS, GSUB, JSTF, LTSH, PCLT, VDMX TrueType - cmap, cvt ,fpgm, gasp, glyf, hdmx, head, hhea, hmtx, kern, loca, maxp, meta, name, post, prep			

***Table 1**. Font file composition after embedding the complete font.*

Font Data Table Compression

Even after compression, the font program data can still constitute an unacceptable amount of file size. In most use cases the **OpenType type tables** are necessary during the creation of the text, and we can simply drop them from the font program. To reduce the size further, we can check the **TrueType** reference and PDF reference for minimum requirements of sfnt **data tables**. Table 2. shows the resulting font data table list along with updated size ratio to the PDF document file size.

Font used: Arial Regular Version 7.00 © 2017 The Monotype Corporation			
State	Size	Compressed Size (Deflate)	PDF file size (one page standard text)
Leave only minimum required tables for CID font	714,224 Bytes	420,000 Bytes	422,410 Bytes
OpenType - None TrueType - glyf, head, hhea, hmtx, loca, maxp			

***Table 2**. Font file composition after reducing font data tables to minimum.*

Font Substitution

Now that we have hit the limit of removing unused data tables we need to find another way to reduce the size. **Font substitution** is the next important step. Considering we are only using a limited set of characters in the PDF document we can rebuild the glyf and loca tables by **removing unused glyph definitions**. In the example, the original font program contains 4502 glyph definitions while the PDF document makes use of 28 of them. This means we can selectively remove 4474 glyph definitions from the font program while still retain the same functionality in the PDF document. Table 3. shows the resulting font program size after removing the unused glyph definitions.

Font used: Arial Regular Version 7.00 © 2017 The Monotype Corporation			
State	Size	Compressed Size (Deflate)	PDF file size (one page standard text)
Remove Unused glyphs	28,584 Bytes	19,264 Bytes	21,671 Bytes
OpenType - None TrueType - glyf, head, hhea, hmtx, loca, maxp			

***Table 3**. Font file composition after removing the unused glyph definitions.*

Removing Unused HMTX Table Entries

In the last step in font size reduction, we can take a look at the **hmtx table** which contains horizontal metrics data for each glyph definition.

Since we have reduced the number of glyph definitions, it means the **horizontal metrics** are also unused, and they constitute a recognizable amount of the new font data size. Table 4. shows the size of the font program after removing the unused hmtx table entries.

Font used: Arial Regular Version 7.00 © 2017 The Monotype Corporation			
State	Size	Compressed Size (Deflate)	PDF file size (one page standard text)
Reduce hmtx to used glyphs	12,005 Bytes	10,576 Bytes	14,411 Bytes
OpenType - None TrueType - glyf, head, hhea, hmtx, loca, maxp			

***Table 4**. Font file composition after removing unused hmtx entries.*

At this point, we have reached almost **99% reduction rate** of embedded font program while retaining the complete information in PDF file.

This technique can be applied both during PDF document creation and also for optimization of existing PDF documents. Many documents that are the result of collaboration workflow will contain full fonts embedded to enable easy modification and often these documents are not optimized when reaching the final stage and archiving. In this case, we can run **text parsing processes** to quantify the usage of fonts in documents and subsequently run optimizations that will enable lower document file size.

3 Optimization Processes

The optimization processes can be separated into these basic categories:

Font file optimization **Font de-duplication** **Font subset merging**

Font file optimization is achieved by **parsing** and **separating** text by fonts that are used for the visualization. In case the fonts are not subsetted we can apply already described process of font program reduction. This process is also useful in cases where fonts are already subsetted, but the font program still contains unused glyph definitions or unnecessary font data tables. The most common scenario is found in documents that are results of splitting/page extraction processes that do not optimize the output documents font programs.

Font de-duplication, on the other hand, is most common in document merging processes. Many times the separate pages will use the same set of fonts for text visualization and after merging them, the resulting document will use duplicate font resources for each merged page. This leads to incremental document file size with each added/merged page in the final document. Font de-duplication optimization process can **identify** these occurrences and **replace** the font program resources with shared/referenced resource for all pages that use that particular font. After successful de-duplication, font file optimization processes can be applied by taking into account all texts from merged pages to further reduce the final document file size.

Font subset merging is used in the same scenarios as font de-duplication with the difference that font programs are already optimized for particular pages. This process needs to **identify subsets** of the same parent font program and **merge them** as one single font program resource. This process can make use of available system fonts. If the parent font is available, the subset merging process can use it as base font and apply subset optimization based on the list of used characters/glyphs through the font file optimization process. This new font program can then replace the individual font subsets as single referenced/shared resource. In case the parent font is not available, font subsets can be merged in certain scenarios, depending on the font program subset integrity and encoding systems used.