The HINT Project: Status and Open Questions

Martin Ruckert and Gudrun Socher

Abstract

The HINT file format is intended as a replacement of the DVI or PDF file format for on-screen reading of TEX output. This presentation gives an overview of the current state of the project and solicits answers to open questions that might influence further development.

1 Current State

1.1 Version 1.0

Version 1.0 of the HINT file format [**?**] was published in August 2019 and presented [**?**] at the TUG meeting 2019. In March 2020, the first version of HiTEX and two HINT viewers, one for Windows and one for Android, went online on `http://hint.userweb.mwn.de`. The performance of these programs was better than expected in time as well as in space.

HiTEX runs as fast as other implementations of TEX, after all, it skips the breaking of paragraphs into lines and the building of pages. The size of the HINT files it produces are similar to the size of PDF files produced from the same source. Even smaller files can be expected in the future because the current implementation of HiTEX does not use the more compact "text format" defined as part of the HINT specification.

The first viewer for version 1.0 HINT files became operational in Fall 2019. It runs under the Windows operating system using the WIN32 API. While it is not optimized for speed, its performance is still good and the rendering time does not depend on the file size nor on the position of the page inside a large file. Large pages on big screens at tiny font sizes are shown with a small, noticeable delay, but average pages — comparable to letter size paper at 10pt — pop up at once.

When we started to build the first viewer for the Android operating system, we were aware of the limited computational resources available on mobile devices and curious how the user experience would turn out. We decided to use Open EGL to delegate the rendering of glyphs to the GPU and were positively surprised that the new implementation — even on low-cost mobile phones — clearly outperformed the Windows implementation running on a standard PC. It turned out that the computational demands of the TEX based backend are very low and fast rendering depends almost exclusively on the ability to accelerate the transfer of bitmaps to the screen. As a side effect, a group of (lazy) students, charged with implementing various features of the user interface as part of a comutergraphics course, successfully used the full power of the TEX backend to implement a two finger zoom gesture: rerendering the complete page with every movement of the fingers including line breaking and page building. Just to demonstrate the backend-performance this unusual way of zooming is left as an option — named "TEX Zoom" — in the Android implementation.

Both viewers, for Windows and for Android, rest on a shared backend, written as a literate program in cweb (3475 lines of change files for TEX plus 6109 lines of cweb code).

The Windows viewer is written in C (1056 lines plus 176 lines for print support); its user interface is mouse- and keyboard-based. The Android viewer is written in C++ (947 lines) with some embedded Open EGL and Java (1201 lines); its user interface is touch-screen based. In both cases, additional libraries are used for decompression and rendering of fonts and images.

1.2 Version 1.1

Since the publication in March 2020, new features were added to the software and we expect Version 1.1 to be ready for publication in fall 2020. Unfortunately the Version 1.1 HINT file format is not compatible with the 1.0 Version. But the HINT project is a research project and it seemed reasonable to make these changes.

First, a new tag was introduced to indicate the current language. Furthermore, font descriptions now indicate the encoding. This information was not available in Version 1.0 because it is not needed to render HINT files on screen. Knowing language and encoding is, however, very important for language translation or for text to speech conversion to make texts accessible to the visually-impaired.

Second, HiTEX was extended by special syntax to define page templates and the viewer's backend was extended to use page templates. So now it is possible to display footnotes and floating insertions. In the process of implementing these features, the representation of insertions and page templates in the HINT file format was slightly redesigned.

The new Windows viewer now has a "Print" feature. Taking a TEX file that is designed for, let's say, letter paper, producing a HINT file from it, and printing it on letter paper should give exactly the same result as producing a PDF file and printing

the PDF. The "Print" feature is not (yet) available on Android, because the interface between Android's print manager and Open EGL turned out to be challenging.

The LaTeX support of HiTeX is still incomplete because HiTeX is still based on Knuth's TeX distribution and the extensions of ε-TeX are not yet part of it. HiTeX's memory model is still based on Knuth's 16 bit pointers. We hope, however, that in the near future, HiTeX and the HINT viewers can become a standard part of the TeXlive distribution and as easy to use as any other TeX engine.

2 Open Questions

Now to the questions. Two basic goals of the HINT project are:

1. HINT is a TeX independent format. It should lend itself as an output format to all kinds of document processors.
2. HiTeX is source-compatible to other common versions of TeX like pdfTeX.

As a consequence, it is important to use common standards both inside and outside the TeX universe.

2.1 Language Information and Character Encoding

To represent language information, the world wide web has set universally accepted standards. The Internet Engineering Task Force IETF has defined in BCP 47 [**?**] tags for identifying languages: short strings like "en" for English or "de" for Deutsch, and longer ones like "sl-IT-nedis", for the specific variant of the Nadiza dialect of Slovenian that is spoken in Italy. A HINT file should contain these language tags to enable tools, for example a text to speech converter, to process a HINT file. The open question is: How can I obtain this information from a TeX source file?

The babel package is the de facto standard handling language selection in TeX. It provides a mechanism to map the tags from BCP 47 to TeX's language numbers and back again. Adding these tags to an output file is, however, less simple: When TeX's whatsit nodes with subtype language arrive at the page builder, only the language number is left. I think all TeX engines that want to embed language information in their output would benefit from a simple standard mechanism (like adding a TeX string number pointing to the BCP 47 tag to the language whatsit nodes).

The situation with character encodings is similar. In a HINT file, text is represented as a sequence of numbers called character codes. HINT files use the UTF-8 character encoding scheme (CES) to map these numbers to their representation as byte sequences. For example the number "`0xE4`" is encoded as the byte sequence "`0xC3 0xA4`". The same number `0xE4` now can represent different characters depending on the coded character set (CCS). For example in the common ISO-8859-1 (Latin 1) encoding the number `0xE4` is the umlaut "ä" whereas in the ISO-8859-7 (Latin/Greek) it is the Greek letter "δ" and in the EBCDIC encoding, used on IBM mainframes, it is the upper case letter "U".

The character encoding is irrelevant for rendering a HINT file as long as the character codes in the HINT content section are consistent with the character codes used in the font file, but the character encoding is necessary for all programs that need to "understand" the content of the HINT file. For example programs that want to translate a HINT document to a different language or for text-to-speech conversion.

The Internet Engineering Task Force IETF has established a character set registry [**?**] that defines an enumeration of all registered coded character sets. The coded character set numbers are in the range 1–2999. This encoding number is required in a HINT file as part of a font definition. Is there a method to obtain this number in a standard way when processing a TeX source file? Again, all TeX engines that want to produce accessible output will need that information.

2.2 Images

This section is only about still images, not animated images. Still there are many different file formats to store images, but most of them exist only for historic reasons, marketing considerations, or patent rights. If you look at the WWW, only a few image file formats are around: JPEG for photographs; PNG for clip-art, button faces, and other simple graphical objects; and SVG for resolution-independent vector graphics. Image formats that are commonly used with TeX, like EPS or PDF, are not found among them. Further, TeX engines and TeX viewers support different sets of image formats. There are very good programs available to convert basically any image format into any other image format, and therefore it seems reasonable to define one basic set of image formats that are supported by any TeX engine and any TeX viewer. I assume that JPEG and PNG are most likely part of such a set. The open question is: What kind of image format should be used for vector graphics?

From the different versions of SVG, probable only compressed SVG Tiny is a candidate for the

HINT file format because the HINT file format is specifically designed for mobile devices, like eBook readers, with severely limited capabilities. But even for SVG Tiny, it seems there is no simple, high-performance library that would do the job. Of course there are interpreters for SVG, EPS, or PDF graphics, but unless you need such an interpreter anyway for viewing your TEX output, these interpreters add considerably to the footprint of your TEX viewer.

The only extra interpreter that is (planed to be) part of the HINT viewer is the FreeType library. It is needed for PostScript Type 1 fonts. Therefore the cheapest alternative would be a program that converts vector graphics to glyphs in such a font. Is this a viable idea?

Finally, someone might be able to answer this question easily, but I have not investigated it yet: What is the minimal set of primitives I have to implement in HiTEX to make the typical LATEX graphics packages happy?

2.3 Links

For an output format like HINT that supports on-demand page building, the usual method of books — and TEX — to implement references, namely page numbers, does not work. Therefore links are necessary to navigate large documents. Generating links is already common practice when TEX's output is a PDF file. But note, that for example the *TUGboat* sample article template starts with

```
\usepackage{ifpdf}
\ifpdf
\usepackage[...]{hyperref}
\else
\usepackage{url}
\fi
```

A clear indication that there is no common set of primitives to implement links. Is the only solution another `ifhint` package?

Many of the new primitives of PDFTEX write their arguments directly to the PDF output file. Is it not better to define a new, standardized subtype of whatsit nodes for representing links, together with a common set of primitives to generate these nodes, and postpone the generation of output format dependent code?

2.4 The Viewer API

Unlike previous questions, this question is mainly, but not exclusively, a HINT specific question. When viewing TEX output, be it a HINT file, a PDF file, or some other format used for example in a WYSIWYG editor, the viewer must support interactions between user and document. In simple cases this is just paging forward or backwards, and in more sophisticated cases following a link or zooming. In contrast to DVI or PDF output, the document representation in a HINT file is more or less the representation that TEX uses internally for the document. Therefore the question of how to interact with such a document might be of interest for all programs that interactively manipulate TEX documents.

The HINT project separates the document handling from the graphical user interface (GUI). It provides a generic backend program with a clear API that should be flexible enough to support any GUI frontend. Currently the following operations are supported: opening and closing a document, setting the size of the output area (true size and resolution), rendering the current page, obtaining the top-left position of the current page, and moving to a new page given the top-left or bottom-right position.

Other operations are under consideration. For example, consider that the user interface wants to implement searching for words. Opening a window and entering the search term is entirely the responsibility of the GUI. The backend needs to supply a function to find the position of the next occurrence of the search term. But just displaying a page that starts or ends with the given position is probably not what the user wants. A new function is needed that moves to a new page that contains the given position somewhere in the middle. Another function should probably be available to test whether the new position is already on the current page.

To interact with images, floating insertions, and links might require many more additional functions. Since there is no point in reinventing the wheel, we ask: Is there an established set of functions or pattern to accomplish such tasks which is as simple as possible and as powerful as necessary to support the user interfaces that — hopefully — will be written in the future?

2.5 Change Files as Literate Programs

Large parts of the HINT project are written as literate programs using `cweb`. The special situation with HINT is, however, that important parts of the code are taken directly from Knuth's TEX implementation, written as a `WEB` itself, but with many, many small modifications. For example, TEX's dimensions become extended dimensions in HINT. An extended dimension is a linear function of `\hsize` and `\vsize` and hence every occurrence of `cur_val` needs to be supplemented by an occurrence of `cur_hfactor` and `cur_vfactor`. Similar supplements are needed for the table of equivalents or the save stack.

The traditional method for such modifications are change files or slightly less traditional, but more convenient, patch files. Large change files or patch files tend to be dull reading material up to the point where they must be considered unreadable. Two main problems affect the readability of change files: the the lack of context and the necessity to order changes by their appearance in the original. The situation can be alleviated somewhat by using the `tie` program and organizing the changes into collections of related changes. For the HINT documentation, from these change files reasonable TEX output is generated. But overall, the result is still less than satisfactory, and this raises the question: Is there a good way to present changes to a literate program as a literate program?

3 Conclusion

Considering the scale of the project and the complexity of the involved software, the HINT project has moved forward with surprising speed. While there are still many open questions and missing pieces, the available HINT prototypes are already usable for small projects and provide a testbed to explore the advantages and the challenges of on-demand paging with TEX.

Version 1.1 will provide a more stable basis and offer enough flexibility to make HINT files a viable alternative to the DVI or PDF format for on-screen reading of TEX output. In the long run, however, a new document format will only survive if it is either widely used or if its infrastructure is sufficiently superior to existing formats. Neither is currently the case. Therefore the HINT project is still looking for partners in the industry that have the will and the necessary resources to turn the HINT project from a research effort into a product.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
martin.ruckert@hm.edu

◇ Gudrun Socher
Hochschule München
Lothstrasse 64
80336 München
Germany
gudrun.socher@hm.edu