A color concept for HiTEX

Martin Ruckert

Abstract

While using colors is expensive in print, it is available at no cost on screen. Therefore HiTEX and the HINT file format need support for colors. The design, its objectives, and specification is described in the article. The result is illustrated by examples.

1 Designing a programming API

The current design of color primitives in the various TEX engines is strongly influenced by the available primitives of pdfTEX. The design space available when pdfTEX came into being was limited by the features that the target PDF output format could provide, which in turn was limited by the capabilities of the graphics hardware at that time. The graphics hardware available in the 1990s on personal computers is no match to the capabilities we have now. Even the capabilities of expensive workstations or the dedicated hardware of professional digital printers at that time fall short of what we have today on a mobile phone. The limitations given by the output format were not negotiable. PDF started as a proprietary format of Adobe and then evolved into an international standard. A comparatively small user community like ours had little to no influence on the decisions made.

The necessity to design an new color API for HiTEX offers me a unique opportunity: To design an API with hardly any restrictions. It starts with the available graphics hardware which is dimensioned to handle gaming workloads that far exceed anything that we might need for displaying TEX output. Further, the graphic cards are programmable for example with OpenGL. This puts the raw capabilities of the hardware at the disposal of the programmer without the limitations of a predefined API like, for instance, the Microsoft Windows graphics device interface (GDI). Since I am the designer and implementer of the viewer applications for HINT files as well as the HINT file format, I can freely choose what features the file format should offer. As the designer and implementer of HiTEX, I can start with designing its programming API and work my way back from HiTEX's requirements to the graphics hardware. To be honest, I had to do this journey three times before arriving at the present design. Each time the struggle for an efficient and elegant solution taught me new insights that convinced me to start over again.

2 Design objectives

Of course the new API should be powerful and flexible, elegant and easy to use, but at the same time allow for efficient implementations. Efficiency means that colors should not take up too much space in the HINT file, and rendering the HINT file should not require expensive operations. The efficiency of rendering a HINT file is far more important than the efficiency of creating a HINT file because the former might run on a small mobile, battery powered, device with severe restrictions on memory and energy consumption, while the latter usually runs on a personal computer with lots of memory and processing power of a different order of magnitude.

There were two more complex objectives: The new design should respect the structure and spirit of TEX, and it should be largely compatible with the existing designs of TEX color primitives. The latter is crucial for user acceptance; no one will rewrite or adapt an existing macro package if the benefits do not warrant the effort. The former is crucial for an intuitive use of the primitives: few people read manuals, most people follow their instincts when they modify and extend pre-existing documents. So what the primitives do should be in line with what users generally expect of TEX.

The most important design decision that followed from these objectives was to associate background colors with boxes, not rules. Rules are colored with the foreground color so that for example the fraction line naturally gets the same color as numerator and denominator. Boxes can be stored in registers and should take their background color with them when taken out of the register. Boxes can be nested, and therefore transparent colors must reflect the nesting of boxes.

3 Use of colors in text

Three different decision makers might determine the color of any word or glyph: the document author, the reader of a document, and the rendering application. One important design goal of the HINT file format was to give the document author ultimate control over the appearance of the document. This is in contrast to the HTML file format, where the rendering application, the browser, can decide the fonts or colors, along with many other aspects of the document. The browser in turn offers the user various settings to adapt the appearance to personal preferences. This leads to a permanent fight between document designers and document consumers over how a document should look.

The reader of a document has legitimate requirements for the look and feel of a document. It is

doi.org/10.47397/tb/45-3/tb141ruckert-hicolor

common for user interfaces to offer a "Dark" mode that changes the colors to reflect a dark reading environment. Other users might need a "High Contrast" mode or a "Color Blind" mode. So the `HINT` file format is capable of offering various modes. Currently there are only two modes implemented, "Dark" mode and "Normal" or "Day" mode; there might be more modes in the future. The user interface of the rendering application should enable choosing between the different modes available to the user.

Also the rendering application might have the need to make a color change. For example if the reader uses a search form, the application might want to highlight all matching words on the page. Just using a yellow foreground color might work sometimes but not if the background happens to be yellow as well. If the background is green, the highlighting might still be visible, but hard to read, and incredibly ugly. And what might work in day mode might no longer work in dark mode. Besides "highlighting", the renderer also has the option to "focus" a part of the document. There might be more choices for the renderer in future versions.

To give the document author full control over the colors used, while still allowing the user and renderer to initiate color changes, a color change in a document, as given by the document author, must specify colors for all possible modes, and for each mode, colors for normal, highlighted, and focus text must be given. Together this ensures that a document might use colors in a decent way, that works well with the user's choices and the renderer's necessities. Of course it is essential that useful default colors ease the burden of specifying all these colors. While the design presented here offers some support for default values, the main responsibility to compute and provide useful defaults lies with high-level macro packages.

4 The specification

This section tries to give a complete specification of syntax and semantics of the primitives envisioned for the color support of HiTEX. To describe the syntax, I will use the extended Backus-Naur form (EBNF). For those not familiar with this formalism, here are a few explanations: A syntactic element is represented by a "symbol". I use ⟨*italics*⟩ enclosed in pointed brackets to denote symbols, and I use rules to define the meaning of symbols. A rule starts with the symbol to be defined, followed by a colon "`:`", and then the text that this symbol stands for. A rule ends with a period "`.`". Some symbols refer to text that is defined as part of standard TEX. These symbols are explained by an informal description. For example:

⟨*integer*⟩`:` an integer as in `\penalty`⟨*integer*⟩`.`
⟨*number*⟩`:` a number as in `\kern`⟨*number*⟩`pt.`

Optional parts of the rule's text are enclosed in [square brackets]. Alternatives are separated by a vertical bar "|". For text that must occur verbatim in the TEX source file, I use a `typewriter font`.

4.1 Single colors

The internal representation of a color in HiTEX as well as in the `HINT` file format uses three bytes to represent a color using the sRGB IEC61966-2.1 color space specification and a fourth byte for an alpha channel, which specifies the level of transparency. The sRGB color space is neither the most modern nor the one preferred by professional artists, but it is available (at least approximately) on all computer monitors and these are the main target for displaying `HINT` files. There are alternative external formats possible, for example using high precision floating point numbers or alternative color space definitions, but at the end, every color needs to be rounded to the best possible internal representation just described.

Now let's consider the external representation of a color when using the HiTEX color primitives.

The most common color specification is the specification of a foreground color. Because we use one byte for each of the four values that define a color, it is common to specify the three color components, red, green, blue, and the transparency component alpha (in this order) using integer values in the range 0 to 255. Using this representation, a foreground color can be specified using the following syntax:

⟨*foreground*⟩`:` `FG {` ⟨*integer*⟩ ⟨*integer*⟩ ⟨*integer*⟩ [⟨*integer*⟩] `}.`

For convenience, the alpha value is optional; if no alpha value is given, the value 255 will be used and the color is completely opaque.

Here are some examples: `FG{255 0 0}` and `FG{255 0 0 255}` both specify the same plain and opaque red; `FG{0 0 255}` is plain blue; `FG{255 255 0 127}` is a transparent yellow. Because each value fits in a single byte, the values are often given in hexadecimal notation. In TEX, hexadecimal values are written with a `"` prefix. The same colors as before are then written `FG{"FF 0 0}`, `FG{"FF 0 0 "FF}`, `FG{0 0 "FF}` and `FG{"FF "FF "7F}`. Values greater than 255 or less than 0 are not allowed.

A common alternative to the color representation just described is a device-independent notation, where each value is a real number in the interval from 0 to 1. To keep both representations unambiguous, the device-independent representation (with the smaller numbers) uses the lowercase keyword `fg` instead of `FG`. Here is the syntax:

⟨*foreground*⟩: `fg {` ⟨*number*⟩ ⟨*number*⟩ ⟨*number*⟩ [⟨*number*⟩] `}`.

Using the new syntax, the colors above are written `fg{1 0 0}`, `fg{1 0 0 1}`, `fg{0 0 1}` and `fg{1 1 0 0.5}`. Values greater than 1 or less than 0 are not allowed.

The big difference between `fg` and `FG` is the possible range of values. This can be especially confusing when using the value 1, which belongs to both value ranges; in the first case 1 is the maximum value, and in the second, the smallest above zero. So `fg{1 1 1}` is pure white, while `FG{1 1 1}` is the darkest possible gray, which on most devices is indistinguishable from pure black.

4.2 Color pairs, sets, and specifications

In the HINT file format colors always are given as a pair: foreground and background.

⟨*color*⟩: ⟨*foreground*⟩ [⟨*background*⟩].
⟨*background*⟩: `BG {` ⟨*integer*⟩ ⟨*integer*⟩ ⟨*integer*⟩ [⟨*integer*⟩] `}`.
⟨*background*⟩: `bg {` ⟨*number*⟩ ⟨*number*⟩ ⟨*number*⟩ [⟨*number*⟩] `}`.

The rules for the background color mirror the rules for the foreground. Note that the specification of the background is optional. Here and in the following, default values are used if optional values are not given.

Here are some examples: `fg{0 0 0}` specifies the foreground color black which is then used for rules and glyphs. In addition to the foreground color, you can specify a background color. For example, black text on white background is specified by `fg{0 0 0} bg{1 1 1}` or, using hexadecimal, `fg{0 0 0} BG{"FF "FF "FF}`.

Colors always come as a color set where a color set consists of three color pairs: for normal text, for highlighted text, and for text that has the focus.

⟨*color set*⟩: ⟨*color*⟩ [⟨*color*⟩ [⟨*color*⟩]].

The second and third color pair are again optional.

Finally, a call to the `\HINTcolor` primitive that is used in HiTEX to change the current color has the format: `\HINTcolor {` ⟨*color specification*⟩ `}`. A ⟨*color specification*⟩ will comprise colors for all color modes available to the reader.

⟨*color specification*⟩:
⟨*color set*⟩ [`dark` ⟨*color set*⟩].

For example, if in dark mode you like white letters on a deep blue background you can write `\HINTcolor {fg{0 0 0} bg{1 1 1} dark fg{1 1 1} bg{0 0 0.3}}`. Such a color specification needs $4*2*3*2 = 48$ bytes, and with more modes, even more bytes. To reduce the memory use for colors, color specifications are listed in the definition part of a HINT file and referenced in the content part of the HINT file by a one byte number. This limits the number of different color specifications (not colors) in a single document to 256.

4.3 Colors and boxes

The renderer of a HINT file will render boxes, glyphs, and rules from left to right and top to bottom (right-to-left languages are not currently supported in HINT). The colors given with the `\HINTcolor` primitive will have an immediate effect on all boxes, rules, and glyphs that follow in the direction of rendering. At any point inside a box there is exactly one current foreground color and one current background color.

Rules and glyphs are rendered with the foreground color on top of the background color. The effect of a color change will persist until the next change of colors or until the end of the box — whichever occurs first. Restricting the effect of a color change to the enclosing box implies a natural nesting of colors that follows the nesting of boxes: An inner box is rendered on top of the outer box.

The implementation of the renderer must follow this rendering order. While the rendering of glyphs and rules in the current foreground color is simple, rendering the background requires extra effort because the background must be rendered before rendering glyphs, rules, and inner boxes on top of it. A new background color will start where the color change is found and fills a horizontal box from top to bottom and a vertical box from left to right.

To render the background, the renderer needs to know where the new background color should end. There are two choices: searching the document for the next explicit or implicit color change, or buffering all rendering activities until the end of the current background is reached. The first alternative is easy to implement and since the rendering is always concerned with a single page the search is quite fast. The second alternative is more complex to implement, but possibly much more efficient because transferring a whole array of rendering data from main memory to the graphic card tends to be much faster than writing each item separately.

4.4 HiTEX's color stack

While the HINT file format can represent nested boxes, which affects the rendering of colors as just described, the color model of the HINT file system within a box is completely flat. But TEX is a document description where nearly everything can be nested inside everything else. So having a color stack that allows the author or macro programmer to say: "return to the previous color" seems indispensable. So HiTEX maintains a color stack and a few other features that allow for more convenient programming.

The `\HINTendcolor` primitive of HiTEX relies on this color stack. It will look up the color specification that was valid just before the matching use of `\HINTcolor` and insert in the HINT file a color change that returns to this previous color. If there is no matching `\HINTcolor` primitive, the `\HINTendcolor` primitive is silently ignored. Note that within a single box, there is at any point only a single background color: The color stack will switch from one background color to another background color but will not overlay an "inner" background color over an "outer" background color. This is the case only when multiple boxes are nested as described above.

Further, splitting off the initial part of a vertical box with `\vsplit` will insert a color node in the remaining part if necessary to keep the color consistent across the split.

Complications arise from color changes in the top-level vertical list, which is split into pages in the HINT file viewer at runtime. Because the page builder in the viewer has no global information and should not need global information, HiTEX will insert copies of the local color information after every possible breakpoint in the top level vertical list. This will ensure that page breaks will not affect the colors of the displayed material.

However, since TEX considers glue (and kerns) as discardable, it removes such items from the top of a new page. Because glue and kern items are colored using the current background color, they might be visible on a page but disappear when they follow immediately after a page break. So, if you want the effect of a colored glue or kern that is not affected by a page break, you should include it inside a box or use a colored rule instead.

The line breaking algorithm that is part of the HINT file renderer also tracks changes in color within a paragraph and reinserts an appropriate color change at the start of every `\hbox` that contains a new line. In this way local color changes inside a paragraph can span multiple lines but do not affect the interline glue or material that is inserted with `\vadjust`.

4.5 Colors and links

The most common change in color is caused by the use of links. To support this changing of colors, the primitives `\HINTstartlink` and `\HINTendlink` cause an automatic color change. Whenever the `\HINTstartlink` primitive is used, its effect on the colors is equivalent to executing the `\HINTcolor` primitive using the current link color, which is set with the primitive `\HINTlinkcolor {` ⟨*color specification*⟩ `}`. This implies that the color change caused by `\HINTstartlink` is local to the enclosing box.

Whenever the `\HINTendlink` primitive is used, it will restore the color stack of HiTEX to its state before the matching `\HINTstartlink`. It is the responsibility of the TEX source code to keep sequences of `\HINTstartlink`, `\HINTendlink`, `\HINTcolor`, and `\HINTendcolor` properly nested. A sequence like `\HINTstartlink` ... `\HINTcolor` ... `\HINTendlink` ... `\HINTendcolor` is not an error, but will cause `\HINTendlink` to restore the colors to those in effect before the `\HINTstartlink`. Then, the following `\HINTendcolor` will either restore the color of a matching `\HINTcolor` preceding the link in the same box or it will restore the color in the outer box, or it will be ignored. In short, color changes inside a link stay local to the link.

4.6 Color defaults

The HINT file format specifies default values for all colors. To override these defaults, HiTEX provides the primitive `\HINTdefaultcolor {` ⟨*color specification*⟩ `}`. This primitive must be used before defining any custom colors using `\HINTcolor`.

The HINT format specifies the following default colors: Normal text is black `FG{0 0 0}`, highlight text is slightly dark red `FG{"EE 0 0}`, and focus text is slightly dark green `FG{0 "EE 0}`. The background is transparent white `bg{1 1 1 0}`. In dark mode the background is transparent black `bg{0 0 0 0}`, normal text is white `fg{1 1 1}`, and a lighter red `FG{"FF "11 "11}` and green `FG{"11 "FF "11}` are used for highlighted and focus text.

For convenience, the HINT file format specifies default colors for links as well: in normal mode, links use dark blue `FG{0 0 "EE}` instead of black; in dark mode, links use light blue `FG{"11 "11 "FF}` instead of white. The primitive `\HINTdefaultlinkcolor` `{` ⟨*color specification*⟩ `}` can partly or completely redefine these defaults.

5 Examples

The first example illustrates a common application: Color is used to emphasize a word and accentuate selected paragraphs.

```
\def\redTeX{%
  \HINTcolor{fg{1 0 0}}\TeX\HINTendcolor}
\def\note{\HINTcolor{fg{0.3 0.3 0.3}}}
```

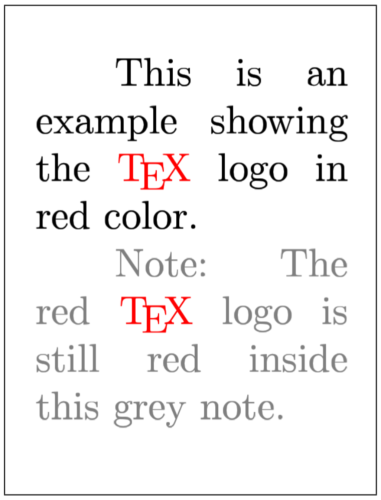
This is an example showing the TEX logo in red color.
Note: The red TEX logo is still red inside this grey note.

Figure 1: Interacting colors

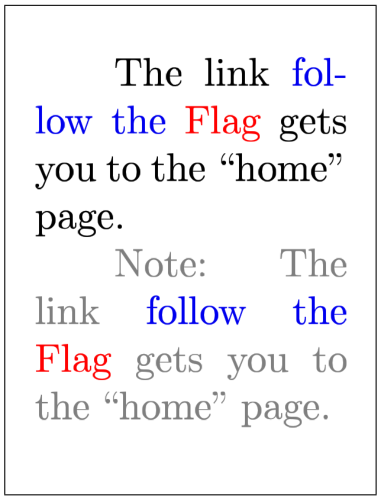
The link follow the Flag gets you to the "home" page.
Note: The link follow the Flag gets you to the "home" page.

Figure 2: Link colors

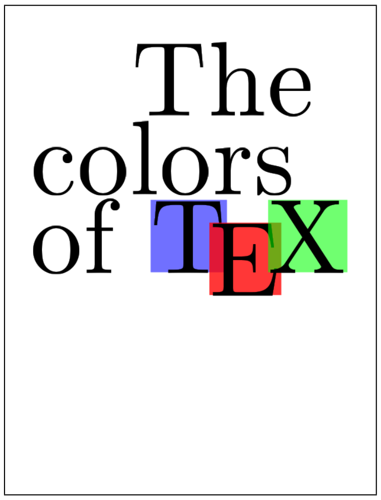

Figure 3: Transparency

```
\def\endnote{\HINTendcolor}

This is an example showing
the \redTeX\ logo in red color.

\note\ Note: The red \redTeX\ logo is still
red inside this grey note.\endnote
```

I show the TEX logo in red and use a `\note` environment that renders text in light gray. The example shows how to implement the color changes, and Figure 1 shows how they interact.

The next example uses `\red` to change the color to red and the "Note" environment from the previous example. It illustrates the interaction with links that are rendered by default in blue.

```
\def\red{\HINTcolor{fg{1 0 0}}}
\def\home#1{%
  \HINTstartlink goto name {HINT.home}
  #1\HINTendlink}

The link \home{follow the \red Flag}
gets you to the ``home'' page.

\note Note: The link \home{follow the
\red Flag} gets you to the ``home'' page.
\endnote
```

Figure 2 shows how the end of the link automatically restores the color that was in effect before the link and that hyphenation and line breaking interacts smoothly with color changes.

The last example illustrates the use of transparent backgrounds.

```
\def\blue{\HINTcolor
   {fg {0 0 0} bg{0 0 1 0.3}}}
\def\red{\HINTcolor
   {fg {0 0 0} bg{1 0 0 0.6}}}
\def\green{\HINTcolor
   {fg {0 0 0} bg{0 1 0 0.3}}}
\def\TeX{\hbox{\blue T}\kern-.1667em
  \lower.5ex\hbox{\red E}%
  \kern-.125em\hbox{\green X}}

The colors of \TeX
```

This defines a TEX logo that uses three different background colors for the letters, using alpha values. As seen in Figure 3, the normal kerning and shifting of the letters in the logo causes the backgrounds and the glyphs to overlap producing the effect of overlapping pieces of colored glass.

6 Conclusion

The HiTEX color support is planed to be released with TEX Live 2025 as an experimental release. It will require not only a new version of HiTEX but also an update of the viewer applications for various operating systems (GNU/Linux, Windows, macOS, iOS, Android) so there is still a lot of work to do.

To use the color support with LATEX, driver files need to be written for at least the most important packages that deal with colors. The help of the LATEX team and package maintainers would be greatly appreciated. I assume that these activities will result in new insights that will lead to an improved API and an improved implementation, which might be available as soon as the 2026 TEX Live release.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
martin.ruckert (at) hm dot edu