The (Pre)History of Color in Rokicki's `dvips`

James Lee Hafner
IBM Research Division, Almaden Research Center, K53/802, 650 Harry Road, San Jose, CA 95120-6099, USA
hafner@almaden.ibm.com

Abstract

In this paper I give an abbreviated history of the current scheme for using color with Rokicki's `dvips` program up to the end of 1993. The real story begins in early 1990, when a local user asked if I could add to the fledgling FoilTEX project support for color to take advantage of our new color printers. This started a major effort, in collaboration with Tom Rokicki, to find an efficient and simple method for specifying color in TEX documents.

Introduction

The `\special` command enables you to make use of special equipment that might be available to you, e.g., for printing books in glorious TEXnicolor.

D. E. Knuth

As the quote above indicates, the grand wizard himself expected that color could (would?) be incorporated into TEX. He expected that this would be done through the use of `\special` commands to the dvi driver. In spite of this, not much was done with color for many years. Even SLITEX, where color is very desirable, was written to handle color in a rather cumbersome way.

In this paper, I will describe the efforts that went into the design and development of the current color support in Tom Rokicki's `dvips` program. I consider this the prehistory of true color support because only some of the real color issues were addressed (and many of these were done via simple hacks). Before we go into `dvips`'s method, let me set the stage.

The availability of color PostScript printers created a need for a better method to handle color. In stepped a number of people, including Leslie Lamport who wrote `color.sty` and Timothy Van Zandt who wrote PStricks. These all use literal PostScript commands passed to the dvi driver and then to the output PostScript file to create color effects. Unfortunately, there are problems with the use of literal PostScript. Namely, since each page is generally a self-contained graphics object, color effects on one page would not readily pass over to the next. Furthermore, effects at the end of a current page might trickle into the footnotes or page footer. This forces the use of these color utilities to be limited to very small parts of documents, e.g., single boxes. On the positive side, most dvi-to-PostScript drivers handle these kinds of literal PostScript `\special`s, so usability/portability was not an issue.

My character enters the story in early 1990, when a local user asked if I could add to the fledgling FoilTEX project support for color to take advantage of our new color printers. (At the time, I was not aware of the two packages mentioned above.) Using `dvi2ps` and `dvialw`, I massaged some primitive color support into these programs but certain obstacles came immediately to light. For example, in `dvialw`, large characters and all rules are placed immediately on the first pass of the page, and then the graphics environment is set up for the main characters. This is efficient for memory use but not for consistent color. If one tries to set a large square root sign in color, the opening check mark is fine but the long rule above the enclosed formula will always be black. Similar splits of colors occur for large brackets.

Finally, I came across Rokicki's `dvips` and determined that this is very well suited for color. Some of the reasons for this are stated below. This started a collaboration with Tom about how one could achieve the desired effects. In the next section I discuss the relevant issues. Later I talk about the first real attempts at getting at the problem. Finally, we describe the current system in some detail and discuss some of the limitations.

The Issues

There were a number of issues that we had to deal with at three different levels of the process. At the TEX-level (i.e., for the user macros) we wanted them to work across formats so that they could be used in FoilTEX as well as Plain TEX and LATEX, for example. We had two somewhat conflicting requirements at this level as well. We wanted to allow the naive user to specify colors without having to know a specific color model (do you know what RGB=`(1,.5,.2)` will look like? do you even know what RGB stands for or the notation `(1,.5,.2)`?). At the same time we wanted enough functionality in the underlying system to let sophisticated color experts use a broad range of color models and effects. Furthermore, the macros should lend themselves to the kind of effects one would expect with regards to TEX's grouping. For example, it should be possible to nest colors with the expected results.

Equally important from our point of view, the macros should be *device-independent*. In particular, they should not be written so that only PostScript printers could handle them. This means that the \special keywords should not invoke literal PostScript, but be generic. The transformation from these generic keywords to the device language (e.g., PostScript) should be handled by the driver itself. As far as I know, at present only dvips and TeXview on a NeXT handle these \specials. Hopefully, in the next era in this history (see Rokicki's article in these proceedings), more drivers will be added to this list.

Furthermore, nesting information should not get lost at each new page or other structural break, nor should the order of the pages matter when processing. This of course requires careful handling at the driver level. It must track this nesting information and be able to restore state for any specific page. Structural breaks include but are not limited to margin paragraphs, footnotes, headers and footers.

On the other hand, it is important to note when dealing with color that different rendering devices (even if they are POSTSCRIPT devices) can produce dramatically different perceptual colors on the same input. For example, on a Tektronix wax printer, green is dark and rich whereas on an X-display the same color is much lighter and even phosphorescent. Ideally the driver should be able to customize itself for this discrepancy, at least on named colors.

Dvips's prescanning processes and its ability to modify its behavior for different printers were ideally suited to these ends. (Besides, it is well written code and so easy to dive into to add modifications.)

There is one issue that we did not address. That is the issue of "floats". By floats, we mean anything that appears in some place other than at the current point where TeX encounters it. This includes the obvious floats like figures and tables as well as the more subtle issue of footnotes (particularly long footnotes that might get split across pages) and saved boxes. The problem here is that color attributes at the time the float is processed may conflict with color attributes at the time the float is placed in the document. For example, a float that is encountered when text is blue and background is yellow may float to a page that has a yellow background. There are two possible approaches to this, namely, the float picks up the attributes on the page on which it is set or it takes its attributes (and the surrounding attributes) with it to the float page. In this case, the float may have a boxed background that differs from the main page on which it is set. As should be obvious, this problem is very subtle and it is not clear what approach is the best to take. Some local grouping *à la* the current scheme may provide a partial solution to the problem using the first approach, though we have not experimented with it at all.

A First Pass

In the first attempts at addressing this problem of color, we ignored the device-independence of the \special keywords and attempted to find a solution that required very minimal (if any) changes to the original dvips code.

We used literal PostScript strings in \special macros. There were two kinds of macros. Ones that just set the color state, and another that tried for nesting. This saved the current color state on the PostScript stack, set the color and at the end of the grouping, restored the color state from the stack. For example,

```
\def\textRed{%
      % set color to Red
    \special{ps:1 0 0 setrgbcolor}}
      % save current color
\def\Red#1{\special{ps: currentrgbcolor}
      % set color and typeset #1
            \textRed #1
      % restore old color
            \special{ps: setrgbcolor}}
```

To help with changes across page boundaries, we made a small modification to the bop (BeginOfPage), eop (EndOfPage), and start in the header files. Basically, eop saved the current color, bop restored the color and start initialized the color on the PostScript operand stack. Tom suggested that we do this on a separate color stack, an idea we never implemented because we soon abandoned this approach. We realized that this method was inherently flawed because it was too much tied to PostScript and it only worked if the document was processed front to back with all pages printed. We thought about storing more of the color stack information in the PostScript itself, but this still suffered from a number of limitations, not the least of which is the first one mentioned above.

The Current Scheme

After realizing that any attempt to do this work in the PostScript code was either doomed or too costly in terms of PostScript resources, we determined that it would be best to have dvips track everything internally, primarily during the prescan and then when a color is changed (either by starting a new color region or closing one), simply output a "set color" command in PostScript.

Below are the basic features of this scheme.

The \special Keywords. All color \specials begin with the keyword color (with one exception). The "parameters" to this keyword and their desired effects are described below:

ColorName
: Declare the current color to be the specified color. Furthermore, drop all current color stack

history, and tell the driver to set the new color state to *ColorName* (see the section on Header Files).

Model Parameters
: Same effect on the color stack history as above. The new color state will have model determined by *Model*, e.g., rgb, cmyk, cie, lab, etc., and parameters *Parameters* in that model.

push *ColorName*
: Append or save the current color to the color stack, set the new color to *ColorName.*

push *Model Parameters*
: Same as above but build the new color state from the model and parameters.

pop
: Take previous color off the color stack and tell the driver to use this for the new color state (used for closing of group).

There is one additional color special keyword. This is background. It is used with either the options *ColorName* or *Model Parameters* to tell the driver to set the background color of the current page and subsequent pages.

Some things to note about this system. First, it is completely generic, with no reliance on PostScript. Second, it assumes that some color names are known to the driver or are defined in the output file. For example, in PostScript, dvips could predefine Red to be 1 0 0 setrgbcolor or 0 1 1 0 setcmykcolor. (In fact it uses the latter.) The user might also be able to use the drivers' literal strings mechanism to predefine their own color names. Third, there are two types of color settings. The first is just a "set color and forget the stack." The other "push"es the current color on the stack, sets a new color, and (presumably at the end of a group) "pop"s the last pushed color off the stack to restore. This is the basic nesting mechanism. It is limited only by the resources dvips uses. Fourth, the parsing of the flags is in a hierarchical order. First comes the color keyword to indicate a color special. Next is either a known color name or a color model. After the color model are the parameters for the chosen color. This is in slight contrast to PostScript itself which is more stack oriented and expects the parameters first. We felt that if the driver didn't understand a particular model it should recognize this in the order it parses the \special string.

This functionality of being able to specify the model and parameters allows sophisticated color users a simple option to get special effects.

Header Files. As mentioned above, we assume that a certain set of color names is already known either to the driver internally or is passed to the output file for the printer interpreter. In dvips this is done in the second manner via the color.pro header file. This is prepended automatically to the output stream as soon as any color special is encountered. In this file, two things are defined. First, the PostScript command setcmykcolor is defined in terms of setrgbcolor in order that the output can be processed by some old PostScript interpreters, i.e., ones that do not recognize this function. More precisely, this is done only if the current interpreter requires a definition for this function. Other color models could also be defined here if necessary. Second, the predefined color names are defined in terms of the CMYK (Cyan, Magenta, Yellow, Black) color model. The reasons for this choice are that most color printers use this physical model of printing. This is a subtractive color space as opposed to the additive color (RGB — Red, Green, Blue) of most displays. Another reason is that I had a good template for matching color names to parameters in the CMYK color space for a particular printer.

These colors are only conditionally defined in color.pro. If they are known by the userdict, then no new definition is added. The reason for this is that a particular device might need to have different parameters set. Dvips's configuration file mechanism can then easily be used to customize the color parameters for a particular device by inclusion of a special device header file.

I emphasize at this point the distinction between physical device and output data stream. The latter is PostScript or HPGL or PCL or some such. The former is the actual physical device. These devices can vary widely even under the same data stream. An analogy is the difference between a write-white or a write-black printer and the need for finely tuned bitmap fonts for each. They may both be PostScript printers, but they print differently. The driver should be able to compensate for the physical characteristics of a given device, if at all possible.

The Color Macros. The color macros, defined in the style file colordvi.sty, come in basically two flavors. One kind sets a new color by issuing a \special{color *ColorName*} or \special{color *Model Parameters*}. The second is a combination of pushes, sets and pops for nested local colors. Furthermore, there are user definable colors of both types, where the user declares the color model and the parameters. Finally, there is a \background macro for setting the background color. For example, the revised version of the \textRed and \Red macros defined above are:

```
     % set color to Red
\def\textRed{\special{color Red}}
     % save current color and set Red
\def\Red#1{\special{color push Red}
     % typeset #1 and restore old color
          #1 \special{color pop}}
```

These are described in more detail in Hafner (1992) as well as in the documentation for dvips and for FoilTEX. Note, that these macros are completely device independent, hence the name of the style file. The macros are all in plain TEX form, so that they can be used in any format or macro package. In other words, they are not LaTeX specific.

As for the color names, we used most of the names from the Crayola crayon box of 64 colors with a couple of additions and some deletions. Additions were the named Pantone colors not already included (e.g., RubineRed) and a couple of other well-rendered colors which we named ourselves (e.g., RoyalBlue). Deletions were mostly for colors that did not render well on our printers. In particular, the new fluorescent colors were eliminated. We chose these color names over, say, the X11 colors for a couple of reasons. First, we originally tried the X11 colors but they suffered from bad rendering on all devices tested. They just did not match their names (at least to me on my display or printer). Second, we could match the crayon names to the Pantone tables for a particular printer, and so give an approximate Pantone match to the color names as well as a good set of parameters. This information could (should?) be used at printer setup time to fine tune the parameters of the predefined colors to nearly match a Pantone (via special header files as mentioned above). In this way, output devices can be approximately calibrated to produce similar and expected results. The color names were also very descriptive of the actual color and very familiar to (at least) the North American TeXies. So, naive users have some idea of what to expect from certain color macros.

Tracking the Color Stack. The color stack or history is tracked by dvips in an internal structure. During the prescan which always goes from front to back on the dvi file, the color stack is tracked and a snapshot is taken at the beginning of every page. During the output pass, regardless of what pages are being processed, the driver knows the state of the stack at the beginning of every page. First one outputs both the background color (if necessary) and the top color on the color stack (i.e., the current color active at the beginning of the page) for the page being processed. Then color pushes just augment the color stack. Color pops just drop colors off the stack. Skipped pages are handled in the same way. This tracking keeps everything consistent from page to page.

The Remaining Issues. We have discussed almost every issue that was raised in the beginning. These included the simplicity of the macros themselves so they can work with any format, can be used by naive users with simple and generally recognizable names (Crayola crayons), still fully support arbitrary color models (if the driver can handle them), and their independence of the particular output data stream. We also discussed, in our implementation in dvips, how nesting and crossing of page boundaries are handled in a clean way. Furthermore, the implementation also can be easily customized for device-dependent differences, even within the same data stream.

The only remaining issue is how other structural problems, like margin paragraphs, headers and footers, itemize tags, floats and the like deal with color changes. Other than floats, these can be handled with simple redesign of the basic macros that deal with these page layout areas. Namely, they simply need to protect themselves with some local color macros. Unfortunately, most formats were written before this issue of color came up, so certain problems can arise. As far as I know, FoilTeX is the only format that has the color macros integrated into it. For example, the header and footer macros have their regions wrapped in a local color command that defaults to the root color of the document. So, for example, if the root color is blue, and there is some green text that gets split across page boundaries, then the text will resume green at the beginning of the next page and furthermore, the footer of the current page and header of the the next page will still be set in blue.

The mechanism described here is basically a hack to deal with these problems. A more structural approach at the driver level will be described by Tom Rokicki. At the user level, there is now some color support (e.g., `\normalcolor` and the color package by David Carlisle) in the new version of LaTeX that is designed to help deal with some of these problems in the context of existing drivers. It should be noted that the user interface in the color package is very different from the one we have presented.

Concluding Remarks

The story doesn't end here, of course. We don't claim to have solved all the problems (there are still many) nor to have provided the functionality that a professional publisher might want (refer to the paper by Michael Sofka on that point). The next era in the story is for Tom Rokicki to write (see his paper in this proceedings). Hopefully, Rokicki's new developments will provide a basis for a very powerful mechanism for setting color and one that can be easily integrated into plain TeX and the next generations of LaTeX (and other formats).

Acknowledgements

We thank Tom Rokicki for his comments on this paper as well as for his acceptance of support for color in dvips and his continued interest in the subject. Thanks also to Sebastian Rahtz for the invitation to participate in this session.

References

Hafner, James L., "FoilTeX, a LaTeX-like system for typesetting foils", *TUGboat*, **13** (3), pages 347-356, 1992.