Graphics

A Solution to the Color Separation Problem

Daniel Levin

Color is one of the hallmarks of modern publishing, yet TEX makes no provisions for it. This is unfortunate, although understandable. Until the age of desktop publishing, colors were not part of the domain of typesetters. If a print job involved colors, the help of others was required: graphic artists to do the paste-up, photographers to do the separations, press operators to mix and apply the ink.

While TEX is an exceptional computerized typesetter, it has definitely not kept up with other programs in terms of color capabilities. In the TEX community, there is much concern that the program is losing its position as the premier typesetter for books and journals. I believe this process will not only continue, but accelerate, unless TEX adapts to the changes that have taken place on desktops and in printshops around the world. The changes are centered around new technology, such as color scanners and printers. But that is not all: the greatest change may be in people's expectations of what a typeset document should look like.

Perhaps some will argue that color is not an important issue, since the needs are different between academic presses, where TEX has the upper hand, and commercial operations, where desktop publishing is taking over. The differences are not so great, however. The argument may be based on the view that color is just an embellishment and is best left to graphic artists and photographers. In my opinion, such a view patronizes the arts and constrains the written word.

Color is helpful, if not essential, in many publications. It can highlight parts of a text, clarify certain points, and distinguish phrases and examples. Who has not seen a bible annotated or a textbook printed with answers in red? Who has not written a paper and wished that tables and graphs could have multiple colors? Who has not read a book or magazine and been drawn in by colorful headlines and diagrams, not to mention advertisements? Color printing has arrived and its role is widening, to bulletins, newspapers, even scholarly journals.

This article does not deal with all of the issues surrounding color. Instead, it focuses on just one: how to separate colors — or more specifically, how to apply different colors to characters and rules. Also, while the title reads "A Solution...", the article is more of a proposed solution. It mentions a superset of TEX called Vector TEX (or VTEX), which uses scalable font technology.[1] So as to avoid the debate about names, we should think of VTEX's newer capabilities as proposed extensions to TEX.[2]

Another focal-point of this article is on a certain type of publication: mathematics textbooks. The discussion is limited not only because of space but because of the author's experience in that field. Obviously there are other important uses of color, and those deserve careful attention before one can really claim to have solved the color separation problem. In particular, there needs to be an agreed-upon model of color that is convenient for both authors and publishers. The model must work with different tints or saturation levels; it should accept both process and spot colors; and it should allow colors to be layered (printed on top of one another) or "cancel out" those underneath. These are interesting topics, but they will have to be dealt with at another time.[3]

1. The traditional approach

As mentioned at the outset, TEX does not make any special provisions for color. You can say that it is a beautiful and flexible system, but when you get right down to it, everything is done in black and white. Nevertheless, there are two possibilities open to someone who wants to create a document with more than one color. They are not very satisfactory methods, but they do work in certain situations. The first method involves 'hboxes' and 'vboxes' and makes use of TEX's `\phantom` command; the second relies on the METAFONT program to create invisible fonts.

a. Boxes. This method involves putting small amounts of text in boxes, and then keeping or discarding the boxes, depending on whether they match a given color. For instance, suppose you wanted to print a headline in blue and a short paragraph in black. That would require putting the headline in an 'hbox' and the paragraph in a 'vbox'. Then, to separate colors, you process the document twice, each time hiding one of the objects.

Let's say you wanted to print the headline and hide the paragraph. That is accomplished by typing `\phantom{\vbox{paragraph material}}`. To do the reverse, you remove the `\phantom` command from the paragraph (but keep the paragraph in its 'vbox') and type `\phantom{\hbox{headline}}`. In both cases, the `\phantom` command measures the enclosed box, discards it, and puts an empty,

similarly-sized box in its place. *All of the boxes are necessary.* They assure that the document will be laid out the same way each time it is processed.

The box method works reasonably well if it is done in vertical mode and with short amounts of text (or in horizontal mode with single words or characters). It does, however, have serious limitations. First, since all text is boxed, there is no way for page breaks to occur in the middle of a paragraph. Second, any phrase inside an 'hbox' is typeset at its natural width; no glue is allowed to stretch or shrink. Third, TEX spends extra time building and sizing boxes. Fourth, the method gets increasingly complicated with each additional color.

A fifth problem relates to the design and proofreading stages: there is no way to approximate how colors look next to each other. Your choices are to print colors one at a time or altogether (presumably in black). Referring to the previous example, it would be nice if you could print the paragraph normally and the headline at, say, 50% gray — in other words, give the headline a rough approximation to blue. (It would be even nicer, if one has a specially-equipped printer, to see a document's true colors.) Not being able to see colors, or at least approximate them, makes page layout more difficult and color assignment a lot of guesswork.

b. METAFONT. Instead of putting text in a box and then discarding it, you can use the METAFONT program to create 'invisible' fonts. These fonts have the same dimensions as visible ones, but they have neither an outline nor a fill — which is to say, they do not show up in print. By substituting invisible for visible fonts, you can hide any amount of text. This method overcomes most of the problems of the box method; glue can stretch and shrink, and TEX can calculate optimal breakpoints. But it still leaves you with an 'all or nothing' proposition: either things are printed in black or they are completely hidden.

Invisible fonts are often included in LATEX and SliTEX packages. They are easy to spot, because they have an 'I' as the first character of their names (e.g., the invisible roman font is `ICMR10`). If you can obtain invisible fonts, you may find they are not a complete set. Fortunately, the method of creating them is simple. You take a METAFONT file like `CMEX10.MF` and make a copy with the name `ICMEX10.MF`. Then, just before the last line of the file, type

```
extra_endchar:=
  extra_endchar&"clearit";
```

(so that what you type becomes the second-to-last line). Then run the file through the METAFONT program. Everything is done as usual, except that METAFONT erases all character representations; TFM files and "empty" font files are still created.

When dealing with ordinary text, invisible fonts are a reasonably effective way to separate colors. But they do not offer a complete solution. Surprisingly, they work least well with mathematics, because they do not hide rules. You see, rules are not characters, so there is no way to make them invisible using METAFONT. Even if all fonts are made invisible, rules still show up as horizontal bars in fractions and radicals, and as 'overlines' and 'underlines'.

Perhaps the best thing to do with standard implementations of TEX is to combine invisible fonts with the box method (above). If something involves a horizontal or vertical rule, you simply put it inside a box and, when the time comes, hide it using the `\phantom` command. Fractions, radicals, underlined words, etc., are always typeset at their natural width, regardless of whether they are boxed. So the major problems with boxes do not apply here.

With a good font scheme and judicious use of the `\phantom` command, you can effectively separate colors. You still run into the proofing and design problem mentioned earlier, since you cannot see colors next to each other. Also, you may run into memory problems, because you need to load most fonts twice (a visible and invisible version). And, of course, you must keep many more TFM and PK files on your system. If you are not bothered by such things, it is possible — though still difficult — to print colors one at a time.

2. A newer approach

The VTEX program offers the best method for separating colors that this author has seen. As stated above, VTEX uses scalable font technology, which has all kinds of advantages over traditional bitmapped fonts (not the least of which is a significant reduction in the number of files). In addition to offering limitless font sizes, VTEX can fill characters and rules with a variety of patterns and grayscales. This gives one the capability of separating colors and simulating them in black and white.[4]

VTEX's method is straightforward. Using a `\special` command, you specify a 'fillpattern' for both characters and rules. The syntax is `\special{F#}`, where # is a number corresponding to a particular pattern. VTEX has 22 built-in

patterns and supplies an editor for creating new ones. Of special interest are patterns 0 (black), 8 (about 50% gray) and 22 (white). With those three patterns, you can readily design and print a two-color document. (To simulate additional colors, you need to select more patterns.)

a. Implementation. One can think of various schemes for assigning colors with the `\special` command. A simple approach, which also resembles TEX's `\font` command, uses the following macros:

```
\def\color#1{\toks@={#1}
  \afterassignment\color@\count@}
\def\color@{\expandafter\edef
  \the\toks@{\special{F\the\count@}}}
```

Briefly, the `\color` command matches up a color with a particular fillpattern. The syntax is `\color`⟨*name*⟩`=#`, where the name is written as a control-sequence and the number corresponds to a fillpattern.[5] For example, if you wanted to assign fillpatterns to two colors, blue and black, you could type `\color\blue=8` and `\color\black=0`. Then, to typeset characters and rules normally, you give the command `\black`. To approximate the other color, you give the command `\blue`. Thus the line

```
\black Something old. \blue
  Something new. \black
```

yields

Something old. Something new.

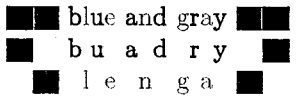
Note: The second `\black` command *is* necessary. Color changes cannot be confined to a group because the `\special` command is inherently global (it sends messages directly to the device driver).

The previous example illustrates colors side by side. By now the reader has probably figured out how to separate them. The trick is to re-define colors. To be specific, any color that should not be printed is assigned fillpattern 22 (white). Following this approach, the lines

```
\def\civilwar{{
  \blue \square \gray \square\
  \blue b\gray l\blue u\gray e
  \blue a\gray n\blue d
  \gray g\blue r\gray a\blue y\
  \gray \square \blue \square \par}}
\def\square{\vrule width1em}

\color\blue=0  \color\gray=8  \civilwar
\color\blue=0  \color\gray=22 \civilwar
\color\blue=22 \color\gray=8  \civilwar
```

result in



1. Simplify: $(3xy^{-2})(\frac{1}{6}x^3y^2)$ $\frac{1}{2}x^4$

2. Solve using the quadratic formula: $2a^2 - 4a - 5 = 0$ $\frac{2\pm\sqrt{14}}{2}$

3. Explain why the sum of two odd numbers is always even.

   If the first number is $2a + 1$ and the second is $2b + 1$, then the sum is $2(a+b)+2$ or $2(a+b+1)$, which is even.

4. Write an equation for the parabola whose x-intercepts are -3 and 3 and which contains the point (0,6).

   $3y + x^2 - 18 = 0$

Figure 1. Black = fillpattern 0 ; Red = fillpattern 0.

1. Simplify: $(3xy^{-2})(\frac{1}{6}x^3y^2)$ $\frac{1}{2}x^4$

2. Solve using the quadratic formula: $2a^2 - 4a - 5 = 0$ $\frac{2\pm\sqrt{14}}{2}$

3. Explain why the sum of two odd numbers is always even.

   If the first number is $2a + 1$ and the second is $2b + 1$, then the sum is $2(a+b)+2$ or $2(a+b+1)$, which is even.

4. Write an equation for the parabola whose x-intercepts are -3 and 3 and which contains the point (0,6).

   $3y + x^2 - 18 = 0$

Figure 2. Black = fillpattern 0 ; Red = fillpattern 8.

b. Textbook Example. Another, more realistic example of color separation has to do with school textbooks. It is customary with teachers' editions of textbooks to print answers next to problems (exercises). Usually the answers are printed in a color other than black, in order to distinguish them from problems. This is an especially common practice with mathematics textbooks, and it makes up a good case study for TEX. While TEX is superb at arranging problems and answers, it does require extensions to handle the color separation.

Let us now consider the situation in a little more detail. Interestingly, color separation is not the toughest obstacle to overcome. With VTEX's `\special` command, as described above, one can easily assign different colors to problems and answers. The real task is in positioning answers next to problems. Ideally, one would like to automate the process, and be economical about it — that is to say, put answers as close to problems as possible. This requires some ingenuity. One needs to figure out how many lines are in a problem, how wide is the last line, and whether an answer will fit in the available space.

In the appendix to this article are some macros which make reasonably intelligent decisions about the placement of answers. There are four primary commands, called `\beginproblem`, `\beginanswer`, `\endanswer`, and `\endproblem`. They are used in the following manner.

```
\beginproblem
general text (problem)
\beginanswer
general text (answer)
\endanswer
\endproblem
```

The 'begin' and 'end' commands form boundaries around problems and answers. Incidentally, the `\beginanswer...\endanswer` field is optional.

The `\beginproblem` command serves several purposes: it assigns a color (normally black), it numbers problems automatically, and it puts problems inside 'vboxes' so that they can appear next to each other. The `\beginanswer` command assigns a color (red) and looks at current conditions. If TEX is in vertical mode, the answer simply begins a new paragraph. Otherwise, the answer is put inside an 'hbox' and the last line of the paragraph is looked at more closely.

What happens next depends on several things: the 'hsize' and number of lines in the current paragraph (h and n), the width of the last line (ℓ), and the width of the answer box (a). Roughly speaking, if $a < h - \ell$, the answer is placed at the end of the last line of the problem. Otherwise, the answer is 'unhboxed' and begins a new paragraph.[6]

1. Simplify: $(3xy^{-2})(\frac{1}{6}x^3y^2)$

2. Solve using the quadratic formula: $2a^2 - 4a - 5 = 0$

3. Explain why the sum of two odd numbers is always even.

4. Write an equation for the parabola whose x-intercepts are -3 and 3 and which contains the point (0,6).

Figure 3. Black = fillpattern 0 ; Red = fillpattern 22.

$\frac{1}{2}x^4$

$\frac{2\pm\sqrt{14}}{2}$

If the first number is $2a + 1$ and the second is $2b + 1$, then the sum is $2(a + b) + 2$ or $2(a + b + 1)$, which is even.

$3y + x^2 - 18 = 0$

Figure 4. Black = fillpattern 22 ; Red = fillpattern 0.

There is just one more determination: if the answer fits on the last line of the problem, and if $n > 1$, the answer box is lowered a little bit. This helps to distinguish the answer from any text which appears above it.

The effect of the macros can be seen in figure 1. Notice that problems and answers are given fillpattern 0 (black). In figures 2–4, the problems and answers are given different fillpatterns; in other words, the commands `\black` and `\red` are re-defined. No other changes are made to the input lines. Finally, notice that all text appears in exactly the same position, regardless of its color (or more precisely, its fillpattern). The color separation is achieved without any extra boxes or font assignments, and with very little fuss!

Conclusion

The importance of color in modern publishing has already been emphasized. The author hopes that the textbook example shows how useful is a good color separation scheme. The macros presented in this article are by no means a complete solution, but they enable TEXnicians to perform many important tasks. Also, they introduce some ideas about color that warrant further discussion. Here are three more points that ought to be debated:

1. The separation of colors is best handled by device drivers; TEX does not need to be concerned with the fillpattern of characters and rules. The only situation where TEX may need to be actively involved is in the layering of colors. For example, a perfectly adaptable system should allow yellow letters to be printed on a cyan background, without the cyan showing through.[7]

2. A color assignment scheme should be part of plain TEX (or LATEX), so that full-color documents can be shared between various platforms and implementations. This can be done in many different ways, perhaps with a font-like command (as in this article) or else with a numbering system (as in `\newtoks`, `\newfam`, etc.).

3. Not all device drivers need to be color capable, but they need to be color *aware*. Even if a device is monochrome, it can still simulate colors with grayscales and other patterns. In short, no driver should be tripped up by color; it should be able to turn hues into black and white.

Finally, it is hoped that members of the TEX community will join together in developing new standards for the program, especially in regard to color. Extensions should not be dismissed out of hand, as if they were unneeded or contrary to the goals of TEX. Rather, they should be looked at real innovations, or as springboards for improving the program, or at the very least as a reflection of the needs of publishers. Sooner or later those needs will be met by a computer system. Wouldn't it be great if that system were TEX?

Appendix

The textbook macros are relatively easy to follow, except for `\measurelastline`. This performs some measurements which are used in the positioning of an answer (`\placeanswerbox`). The macro is derived from an example given by Frank Mittelbach.[8] As he indicates, the only way to measure a line is to go into display mode temporarily. TEX then sets `\predisplaysize` equal to the width of the last line (plus two ems). Simultaneously, `\prevgraph` is set to the number of completed lines in the paragraph.

Once the measurements are made, it is necessary to undo the skips and penalties associated with display mode. This sends TEX back to the end of the last line, where horizontal (paragraph) mode can begin again, virtually uninterrupted. There is one major complication: when TEX begins display mode and then reverses its steps (does an 'unskip'), the depth of the last line is lost. That is why a 'vrule' is added to the list. It acts like a 'strut' — in other words, it guarantees that the last line has the proper depth.

Note: Another fix is needed if an answer is unusually tall and is placed on the first line of a problem. In that case, a correction must be made to the height of the 'vbox' which contains the problem and answer. That is not difficult to do and is left to the reader as an exercise.

```
\newdimen\probwidth \probwidth=12pc
\newdimen\gutterwidth \gutterwidth=2pc
\newdimen\answerskip \answerskip=1em
\newdimen\numberwidth \numberwidth=1.5em
\newcount\probnumber \probnumber=1
\newdimen\lastlinewidth
\newdimen\lastlinedepth
\newcount\lastlinenumber
\newbox\answerbox

\def\beginproblem{\black
  \ifodd\probnumber \allowbreak
    \else \hskip\gutterwidth \fi
  \hbox to \numberwidth{%
    \the\probnumber.\hfil}\nobreak
  \vtop\bgroup \hsize=\probwidth
  \noindent}
```

```
\def\beginanswer{\red
  \ifvmode
    \setbox\answerbox=\box\voidb@x
    \else \measurelastline
    \setbox\answerbox=\hbox \fi
  \bgroup}

\def\endanswer{\egroup
  \ifvoid\answerbox
    \else \placeanswerbox \fi}

\def\endproblem{\par
  \vskip2\parskip \egroup
  \advance\probnumber by 1}

\def\placeanswerbox{%
  \dimen@=\hsize
  \advance\dimen@-\lastlinewidth
  \advance\dimen@-\answerskip
  \ifdim\wd\answerbox<\dimen@
    \hskip\answerskip
    \ifnum\lastlinenumber>1
      \lower\ht\answerbox \fi
    \box\answerbox
    \else \par \normalbaselines
    \noindent \unhbox\answerbox \fi}

\def\measurelastline{%
  $$\nodisplaylineskip
    \global\lastlinenumber=\prevgraf
    \dimen@=\predisplaysize
    \advance\dimen@-2em
    \global\lastlinewidth=\dimen@ $$%
  \par \unskip \unpenalty
  \setbox0=\lastbox
  \lastlinedepth=-\lastskip
  \vskip-\parskip
  \baselineskip=\z@
  \lineskiplimit=-\maxdimen
  \noindent \hskip\lastlinewidth
  \vrule width\z@ height\z@
    depth\lastlinedepth}

\def\nodisplaylineskip{%
  \abovedisplayskip=\z@
  \belowdisplayskip=\z@
  \abovedisplayshortskip=\z@
  \belowdisplayshortskip=\z@
  \baselineskip=\z@
  \lineskiplimit=-\maxdimen}
```

Notes/References

1. VTEX is a trademark of MicroPress, Inc. See the listing in TUG Resource Directory, *TUGboat* 12, no. 2, supplement, p. 129.
2. A similiar point was made by Nelson Beebe in *TUGboat* 11, no. 3, p. 335. He noted that MicroPress "has done very interesting things with extensions to TEX," and these should be studied as "a pilot implementation of some ideas for TEX's evolution."
3. Color graphics are something else to consider. Donald Knuth and others have suggested using a "grayscale font" to create halftoned pictures. This is one way to separate colors, as pointed out by Adrian Clark in *TUGboat* 12, no. 1, pp. 157–165. The method, however, requires a lot of processing power and is device-specific. Also, it does not help someone who wants to apply a shade of gray to a regular font.
4. For a review of VTEX, see *AMS Notices*, 38(2), February 1991, pp. 105–109. The program is also discussed by Al Cameron in *Personal Workstation*, June 1990.
5. This is something like the method suggested by Robert Adams, in *TUGboat* 11, no. 3, pp. 405–406. He suggests using commands such as `\black` and `\red`, and tying them into the PostScript operator `setgray`.
6. This approach is not without complications. For an interesting discussion of line-breaks in 'unhboxed' text, refer to an article by Michael Downes, *TUGboat* 11, no. 4.
7. Knuth implies as much about device drivers: with the `\special` command, one can take advantage of any equipment that might be available — e.g., "for printing documents in glorious TEXnicolor" (TEXbook, p. 229). But he also says the `\special` command leads to incompatibilities. This author agrees completely, and thinks that is the best argument for extending TEX. `\special` commands — or at least the *explicit* use of `\special` — can be avoided only if there emerges a new set of standards for color printing.
8. Refer to Mittelbach's article, "ETEX: Guidelines for Future TEX Extensions," *TUGboat* 11, no. 3, p. 344.