Exploring the Capabilities of PostScript in Generating Color Images

Michael Blum[*] and David Norwood[**]

Abstract

Since its introduction in 1984 PostScript has had a significant impact on both desktop publishing and traditional typesetting. Today it is starting to be used in the creation of color images as well, including high resolution color separation films and full-color output.

This paper covers how the PostScript page description language deals with color, the advantages and disadvantages involved, and the various methods available for generating color images with PostScript. Looking to the future, the impact of this maturing technology on desktop publishers and graphic arts professionals is discussed.

Background

In the past typesetting machines have been controlled by command languages which were specific to a given manufacturer and product line. These command languages allowed the users to specify typesetting characteristics such as font, line length, point size and leading. Problems arose when converting files created for one system to another. Such command languages worked best when a user bought a turn-key system in which all elements were designed to work together. Particularly with the advent of personal computers and the wide-spread use of software for imaging text and graphics it became an enormous task for software developers to write device drivers to support the large variety of printers and typesetters which a given user might select. It was in this historical context that the PostScript page description language was introduced in late 1984 by John Warnock and Chuck Geschke of Adobe Systems.

PostScript

PostScript was developed as a page description language for a number of reasons: to meet the growing need for control over the imaging of graphics; to deal with the imaging requirements of raster devices, such as laser printers, which of necessity must image an entire page at a time, rather than one character or one line at a

*California Polytechnic State University
**Information International Incorporated

time; and to allow software developers the freedom to create programs which would work with a variety of output devices.

Both hardware and software components make up the PostScript standard. The hardware component of the standard is the PostScript interpreter. The PostScript interpreter is a raster image processor (RIP) licensed by Adobe Systems to printer or imagesetter manufactures to include with their devices. Its function is to interpret PostScript page descriptions and create a bit map to be imaged by a raster printing engine. The PostScript RIP is currently found in at least 27 output devices. These raster output devices include laser printers, LED-array, ink jet printing devices, and laser imagesetters. These devices range in resolution from 300 to 2540 dots per inch; however, the same page data can be imaged on any of them without modification. This is because in normal PostScript page definitions no device coordinates are specified. It is the job of the interpreter to convert the coordinates defined in the page description into the device coordinates.

The software component of the PostScript standard is the PostScript page description language. In order to promote the language as a graphics imaging standard, Adobe Systems has placed it in the public domain and has published three manuals describing its usage. Programming in PostScript is much like programming in other languages. It is a full featured language including arithmetic, boolean, string, program control, and file operators. These operators are present to complement its powerful type and graphics operators.

The design of the PostScript imaging model allows for a number of powerful graphics capabilities. The user has the ability to manipulate the page coordinate system in a variety of ways including translation, rotation, scaling, and even odd effects like skewing. Changes to the page coordinate system affect all elements on the page including type, rules, tints, line art, and photos.

In normal circumstances it is the job of an application to automatically generate the PostScript page description which is in turn interpreted by the RIP. Application programs that can output PostScript page descriptions are available today on a variety of computer systems, including the Apple Macintosh, IBM personal and mainframe computers, Digital Equipment Corporation VAX computers, and various UNIX machines. Usually the user never has to work with the page description coding at all, since the application takes care of this. It is, however, also possible for a user to write PostScript programs directly which will result in printed pages. This is typically done to achieve graphic effects not available in commercial software.

The strength of PostScript is that it is designed to be able to handle any type of text or graphics regardless of complexity and process that data for output to any PostScript printer with little or no configuring or setup by the user. Because of this, PostScript is significant for individuals in many different areas of endeavor.

In an office environment PostScript laser printers are used for printing business documents which can easily integrate graphics such as charts along with the text. PostScript has also been instrumental in the rise of the popular phenomenon known as desktop publishing wherein individuals with typically little or no background in graphic arts use personal computers, page makeup software and laser printers to produce documents such as newsletters and manuals.

Applications capable of generating PostScript page descriptions can also be used by graphic arts professionals to create high-resolution typeset text. The manner in which the PostScript imaging model treats text as a graphic element makes it particularly useful for creating special text-effects. Such effects are difficult to achieve on traditional typesetting devices without resorting to labor-intensive manual techniques including cutting and pasting, airbrushing, process camera work and image assembly manipulation. These text-effects include the ability to rotate, slant, stretch, skew, mirror, reverse, curve, shade, shadow, outline, etc.

PostScript's ability to render lines, curves, and shapes in different levels of gray have made it an excellent tool for creating detailed illustrations. Application programs such as Adobe Illustrator allow graphic artists to specify line weights in increments finer than a printer's point and screen tint values in fractions of one percent dot. Curves and shapes can be drawn using Bezier control points. Like the PostScript fonts which are also defined by these control points, they are capable of being manipulated using all of the effects mentioned above for text. These illustrations can often be created and edited in a fraction of the time required using manual methods such as technical pens, ruby masking film, screen tints, etc.

Emerging standardization of file formats such as the Encapsulated PostScript Format (ESPF) allow graphics created with an illustration program to be easily imported into a page makeup program. This is possible even across different hardware platforms since PostScript page descriptions are stored as ASCII text. The ESPF format includes a screen representation of the image in addition to the PostScript page description. Because of the device- and resolution-independent nature of PostScript it is easy for individuals to use a 300 dpi laser printer to proof these illustrations and send the final output to a 2540 dpi PostScript imagesetter.

PostScript and Color

Although Postscript is being used mainly to produce only black and white images, the industry is starting to see it has a great potential for color. Since its introduction, PostScript has contained operators for specifying and using color even though there were no color output devices available that could utilize them. As of this writing, color PostScript devices are just becoming available and with the introduction of these devices the PostScript page description language is being enhanced to provide more color capabilities. Application developers are also enhancing their products to take full advantage of color PostScript imaging.

The original set of four color operators (as documented in the *PostScript Language Reference Manual*) allowed colors to be specified in either the hue, saturation, brightness (HSB) model or the red, green, blue (RGB) model. Supplying HSB values to the sethsbcolor operator will set up the current color to have those specified hue, saturation, and brightness characteristics. Like most parameters in PostScript, these values are normalized between 0 and 1. A hue of 0 corresponds to red, .33 corresponds to green, and .67 corresponds to blue. For saturation, 0 yields a gray while 1 yields a pure color. Likewise, a 0 brightness represents black while a 1 represents white. On a black and white printer it is the brightness component of the color which is used to represent the color.

Supplying RGB values to the setrgbcolor operator will set up the current color to have those specified red, green, and blue intensities. Again, the three values presented to this operator are normalized between 0 and 1. The remaining two color operators report the current color in one of the two formats. Executing the currenthsbcolor operator will return the hue, saturation, and brightness components of the current color, while executing the currentrgbcolor operator will return the red, green, and blue components of the current color. By allowing you to define colors in one model and report that color back in another model, PostScript provides a way to convert colors from one model to another.

With the introduction of color PostScript devices, the Cyan, Magenta, Yellow, and Black (CMYK) model is also being supported. Other enhancements include the ability to image color photos and alter the mapping of colors, as discussed later in this paper.

An Experimental PostScript Color Implementation

As with black and white PostScript images, color separations can either be generated automatically by software applications or directly by manually writing PostScript programs. In order to explore the capabilities of PostScript in generating color images, the Graphic Communication Department at Cal Poly, San Luis Obispo, produced an experimental process color poster commemorating donations made to the department by Apple Computer Inc. and the Linotype Company.

The method used to generate the color separations for the Cal Poly poster involved direct programming in PostScript using a Macintosh Plus computer. Proofing during the coding stage was done on an Apple LaserWriter Plus. Final film separation positives with all images in place were output on a Linotronic 300. Color proofs were made using positive-acting Dupont Cromalin. The positives were then contacted to make negatives for platemaking. Printing of the poster was performed at Cal Poly on a four-color Heidelberg MOVP. No camera work or paste-up was required at any stage of the production of this poster.

By programming the color separation function entirely in PostScript, complete control and flexibility was obtained and graphic effects were possible that could not be obtained through the use of any commercial software. The drawback to this method, however, is the time it takes to write and debug even a simple page description.

Although is not feasible to use this approach for most color applications, the research demonstrates the flexibility of PostScript in creating sophisticated color images of graphic arts quality and points out some limitations of the current PostScript implementation. The PostScript code generated in this research could be used in a "cookbook" fashion by other researchers who might desire to build upon the color routines implemented here. These routines could also be incorporated into software which would automatically generate the appropriate PostScript code for a given color effect.

The way this method is used is as follows. All elements of the page description are in one file. The bulk of the program consists of a prologue where user routines are defined. A small section of code at the end of the program makes calls these routines and controls the actual imaging of separations. This section of code, which we will call the imaging loop, is executed four times. Each time it is run, several settings are modified and, in turn, the yellow, magenta, cyan, and black separations are produced.

The name of the routine used to define colors in this implementation is cdef. It takes the subtractive components of the color and stores them in a one-by-four array. Because colors need only be defined once, execution of this operator occurs in the prologue. For example, to define a color called "green1" which has 90% yellow, 0% magenta, 75% cyan, and 15% black, the following command would be given:

```
/green1 [ 90 0 75 15 ] cdef
```

The values for the subtractive components range from 0 to 100 and represent the percentages of yellow, magenta, cyan and black that make up that color. Before storing these values into the array, the cdef routine converts them into PostScript intensity units which range from 0 to 1. Intensity units in PostScript are inverted from the percent screen units normally used in graphic arts. On an output positive, for example, areas of intensity 0 will be solid (100% dot), while areas of intensity 1 will be clear (0% dot). The cdef routine accomplishes this conversion using the following equation:

$$\text{intensity} = \frac{100 - \text{percent dot value}}{100}$$

To use a color, the setcolor routine is used. This routine takes a color defined using cdef and makes it the current color. All images placed on the page from then on will be in that color until the next setcolor is executed. Calls to setcolor must occur within the imaging loop because its action depends on which separation is being produced at that time. An example of how this might be used is to create a one inch green square which is one inch from the left and two inches up from the bottom of the page.

```
/greensquare {
        72 144 moveto
        0 72 rlineto
        72 0 rlineto
        0 -72 rlineto
        -72 0 rlineto
        closepath
        /green1 setcolor
        fill
} def
```

The imaging loop performs all tasks necessary to create a separation. It is repeated four times by use of a for-next structure to output the yellow, magenta, cyan and black separations in turn. Before each separation is started the imaging loop does several things to initialize the environment.

First, the routine setprintercolor is called. Its purpose is to set up the variable holding the current separation color. The index provided by the for-next structure is used to represent the current printer (separation) being produced. This value is stored in the variable currentprintercolor.

This variable is used in the next step which is setting the screen angle. Screen angles of 90 degrees (yellow), 75 degrees (magenta), 105 degrees (cyan), and 45 degrees (black) are stored in an array. The routine setscreenangle uses the currentprintercolor value as an index into this array. It then calls PostScript's setscreen operator to replace the screen angle to be used for this separation.

The last step in preparation is to image the register and trim marks for the page, as well as the color name tags which will identify the separation, using the domarks routine. Each of the above initializing operations is done once at the beginning of each separation.

With the preparation work done, the actual images can be added to the page. Routines are called within the imaging loop to perform this function. After these are completed, a showpage is executed (which records the page on the printing engine) and the next separation is started. This continues until all of the separations are completed. An example of an imaging loop that creates the separations of the green square from above might look like:

```
0 1 3{
        setprintingcolor
        setscreenangle
        domarks
        greensquare
        showpage
} for
```

With these few user-defined routines added to those built-in to the PostScript Language, basic color separations of type and tint matter can be produced easily. By adding other routines, discussed in the next few sections, special color effects such as neon letters, gradations, and posterizations also can be achieved.

Special Color Effects

Creating a gradation is a time consuming task when approached by conventional methods. Gradations are fairly easy to produce in PostScript. In fact, it is possible to perform some complicated gradation effects that would be too difficult to do by hand.

The method used to produce gradations utilizes the image operator. Although it is normally used for rendering photographic images, this operator can be used with synthesized data which will allow it to be used for rendering gradations.

When imaging a photograph, the image operator reads in a string of bytes representing the gray values of individual picture elements (pixels). From these data it creates a halftone representation of the image and adds it to the page. Instead of feeding the image operator actual picture data, we can synthesize our own string values using any mathematical formula we choose. The image operator will treat the string just like actual picture data and place a halftone representation of it on the page. The image operator will also scale the data to fill the area desired making it necessary to define only one row of string values.

To fill irregular shapes, such as type characters, the clip operator is employed. First, a path defining the outline of the shape is constructed. Then, using the clip operator we limit the imageable area to the inside of this path. Last, we use the image operator as described above to add the gradation over the clipped shape. The result of this routine is an irregular shape containing the desired gradation.

The type of gradation achieved will depend on the function used to create the image string. The only requirement of this string is that it contain integer values ranging from 0 to 255. Any function can be used providing its output can be scaled to this range. Following is a discussion of different types of functions that can be used for gradations.

For a normal gradation—that is, one that gradually changes value from light to dark—there are two functions possible: linear and logarithmic. The logarithmic function yields a more evenly graduated appearance, whereas the linear function yields a graduation which seems to progress faster at the higher densities. The better appearance of the logarithmic function is a result of the way humans perceive light. Our eyes, like most of our senses, react in a logarithmic fashion, which means we are able to distinguish more gray levels at lower intensities than at higher ones. Using a logarithmic function accounts for this, because its slope is greater at lower values than at higher ones. Although a linear function might be used for a certain type of effect, a logarithmic function is the better choice for most purposes.

Figure 1 shows an example of a gradation made using a sinusoidal function. This rippled effect is caused by the wave-like shape of a sine curve. Used with gold and silver colors, this type of gradation gives the appearance of a metallic surface.

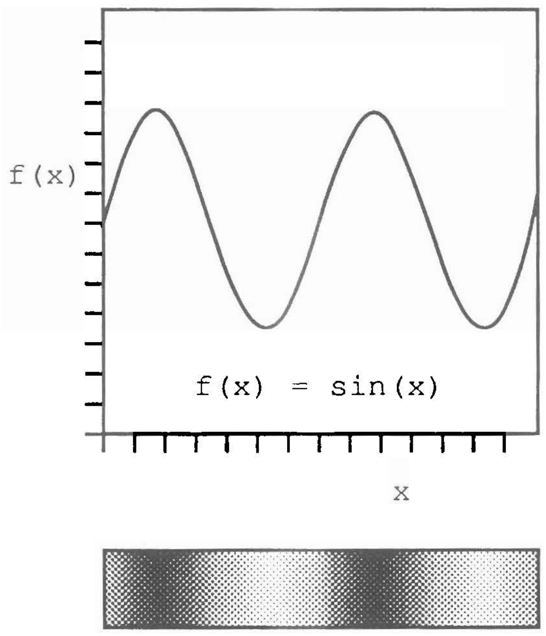

Figure 1 Sine Function

A neon type effect, which is essentially a gradation radiating around each letter of type, is achieved by stroking the outline of the type several times while decreasing the width and changing the color of the line.

Figure 2 shows a progression illustrating the creation of this effect. Starting at the far left, a letter is shown that has been spread 4 points and imaged in a 80 percent tint. Moving toward the right, the next letter has been imaged the same way as the first, except that the character path has been stroked and filled again, this time spread 3 points at a 50 percent tint. On the next character over this has been repeated, this time adding the outline spread 2 points at a 20 percent tint. Finally, on the left, the character is finished off with a thin white 1 point stroke and a black fill.

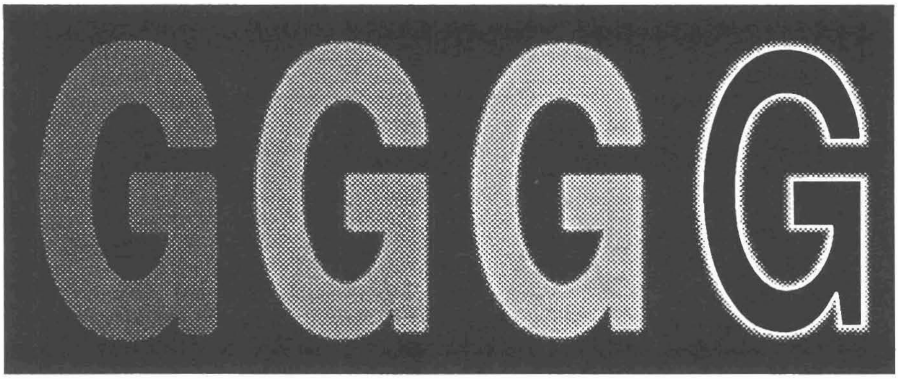

Figure 2 Creating Neon Type

To simplify this task, the routine glowletters was written. This routine utilizes a for-next loop to image the letter. Inside this loop the character path is repeatedly stoked while the width and color of the stroke changes. The color starts out the same as the background and is at its maximum width. As the loop is repeated the color goes to white and the width of the line goes to zero. Finally the type is imaged in its normal color. Figure 3 shows how this effect can be used.

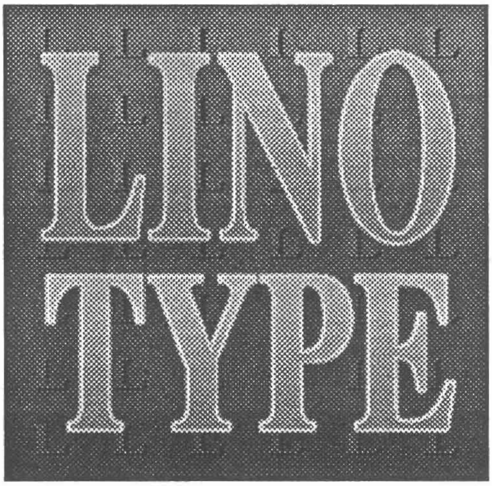

Figure 3 Neon Type Example

The way PostScript handles images allows for much flexibility. It is possible by modifying several parameters in the graphics state to alter greatly the way an image will appear. For this study monochromatic images were manipulated to create color posterizations and tri-tones. The settransfer operator is what makes such effects as posterizations and tri-tones possible. This operator allows the user to change the relationship between the scanned gray level information and the output gray levels. This relationship is similar to a gamma curve used in photography.

Figure 4 shows a linear transfer function and a photo that was imaged with it. This is the normal transfer function as it gives a one-to-one relationship between input and output values. If we reverse the slope of the function as in Figure 5 we get a negative image of the photograph. By manipulating the transfer function we can achieve some very interesting effects.

The idea of a posterization is to reproduce a continuous tone photograph using only several tones. In effect what happens is the tonal range of the original photograph is divided into several sub-ranges. Every tone in a sub-range is assigned to one corresponding tone in the resulting posterization. Figure 6 shows an example of a transfer curve that would be used to create a posterization. The stair step shape of the curve is what causes the posterization effect.

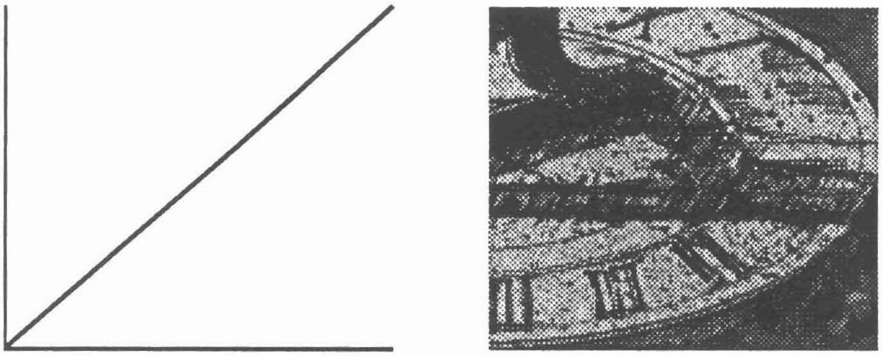
Figure 4 Linear Transfer Function

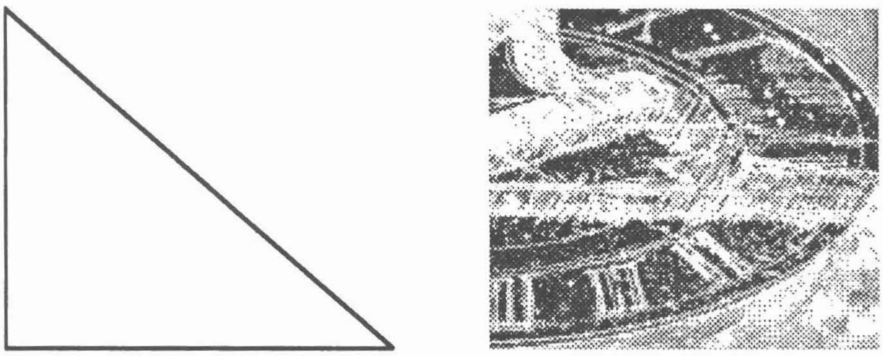
Figure 5 Negative Transfer Function

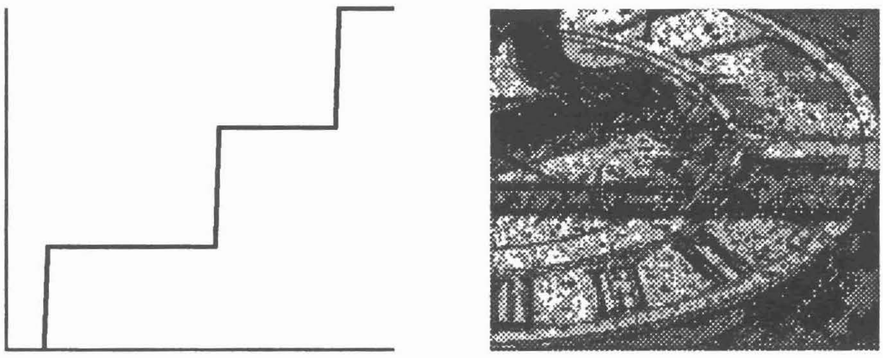
Figure 6 Posterization Transfer Function

The way this input/output relationship is established is by supplying a procedure to the settransfer operator that defines the shape of this curve. A procedure is set up using a nested if-else structure. For example, if a value coming into the procedure is less than 1.0 (0%) and equal to or greater than .85 (15%), a value of 1.0 (0%) is returned. If it is within the range of .85 (15%) to .54 (46%) a value of .65 (35%) is returned. Between .54 (46%) and .10 (90%) a value of .30 (70%) is returned. And between .10 (90%) and .0 (100%) a value of .0 (100%) is returned.

The same principles are applied in producing color posterizations. The only difference is that each of the tonal sub-ranges of the original is assigned to a color tint rather than just a tone. To achieve this, a different transfer curve is used for each separation.

A tri-tone effect can be created to add a color tint to the reproduction of a black and white photograph. The result is still monochromatic but with a greater range of tones. The key to producing this effect is determining the correct transfer curves to be used in rendering the image for the individual separations. This can be done by evaluating the color balance of the printing press being used and the desired tone reproduction of the image. These curves are then used to produce the separations.

Problems Associated with PostScript Color

With present technology only "mechanical" separations are feasible. These effects include spot color, screen tints, line art, tinted type, and gradations. Reproducing full process color images such as color photographs is still problematic due to memory and speed limitations.

Like all data in PostScript, the data format for PostScript images is limited to valid ASCII printable characters. Because of this design it takes two bytes of data to represent 256 levels of grey, whereas in some other data formats it would take only one byte. Also, no data compression techniques are used in any of the present implementations of PostScript. Therefore, images can take up to twice as much space as with other systems and require excessive transmission times. Once the data arrives at the interpreter, ripping times can be excessive compared to normal color separation production.

Another problem area is color trapping. Performing this function is possible in PostScript, however techniques to accomplish this are lacking from present software packages. In conventional image assembly, photomechanical techniques are used to create a slight overlap or trap between adjacent colors. When misregister does occur on press this overlap acts as a buffer area so the unwanted gaps will not appear. Spreads are used to fatten a color area to overlap it with the surrounding background. Chokes are used to shrink a color area relative to its background.

It is normal to modify the thickness of whichever image has the least dominant color, in order to minimize this effect from visual detection.

Performing spreads and chokes of geometric and curved shapes can accomplished in PostScript by utilizing the stroke and setlinewidth operators. This is done by stroking the outline of these types of objects with different line widths depending on which color separation is being produced. When spreading is required the outline of the shape is stroked using the same tint value as the object itself. For a choke, the outline is stroked using the same tint value as the background. The width of the stroke in both cases is set to a thickness equal to twice the amount of overlap desired, since the stroke operator centers the line it creates on the outline of the object.

Two methods were used in the Cal Poly poster to achieve spreads and chokes of text. The first is easiest to implement and uses the PostScript charpath operator, which takes a string of text and creates a path defining its outline. Figure 7 illustrates how charpath can be used to produce spreads. Figure 7(a) shows a character we want to spread. The charpath operator creates a path which follows the outline of the character which can then be imaged using the stroke operator. In Figure 7(b) we see the path created by charpath represented by a solid line. A line applied to this path using the stroke operator will be centered on it. The resulting line is represented by the dotted lines in the figure.

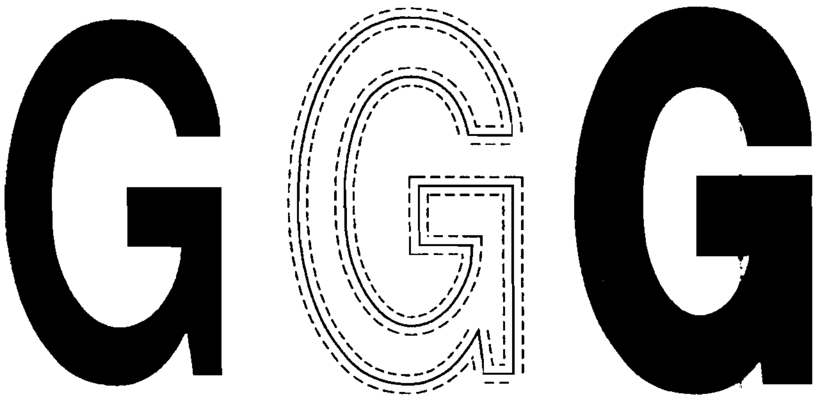

Figure 7 Creating Spreads and Chokes

To accomplish spreading as in figure 7(c), we first image the characters normally with the show operator. Next a path following the outline of the characters is created with the charpath operator. Finally the current line width parameter is set to twice the amount of desired overlap and the path is stroked.

This method has two drawbacks which will keep it from being able to work effectively with larger amounts of text. First, with current implementations of PostScript it takes a great deal of time to create character paths. Secondly, there is a finite limit to the number of characters that can be outlined with the charpath operator at one time because of the memory it takes to hold character paths.

A way around these drawbacks is to create an outlined version of the desired font. Outlined fonts are imaged very quickly and, because they are treated like normal fonts, are not limited in the number of characters that can be imaged at a time. Spreads and chokes can be created with outline fonts much the same way charpath was used. A spread is created by imaging the characters once with the normal font and again in the same color with the outline font. A choke can be created by imaging the characters with the normal font and then changing the current color to that of the background before imaging the characters with the outline font.

Emerging PostScript Color Hardware and Software

At the time this paper is being written there are a number of developments on the immediate horizon which will have a significant impact on the use of color in PostScript. The first color PostScript output device, the ColorScript 100, will be shipped by the QMS Company in mid-May of this year. It is a thermal transfer color printer capable of outputting approximately one 11" x 17" color page per minute at a resolution of 300 x 300 dots per inch. The Color Script 100 controller comes with 8 MB RAM, 1 MB ROM, and a 20 MB hard disk. The controller is based on Version 49 of PostScript, and has RS-232, Centronics parallel, and RS-422/AppleTalk interfaces.

Adobe Systems is currently introducing a number of new extensions to PostScript for the support of color. To complement the existing setrgbcolor and sethsbcolor operators, a setcmykcolor operator has been added. The currentcmykcolor operator returns the four components of the current color in the graphics state. The color equivalents of the existing setscreen, settransfer, and image operators are being added as well. The setcolorscreen operator will allow the specification of screen angle, screen frequency, and halftone cell pattern for each color. The currentcolorscreen operator returns all twelve current halftone screen parameters in the graphics state. The setcolortransfer operator is designed to map the user values of the color components to the actual device color components. Similar to the settransfer operator it may be used to provide gamma correction for a printer's halftoning response. As demonstrated earlier in this paper, settransfer may also be useful for various effects beyond its original intent, such as posterizations. The currentcolortransfer operator returns the current transfer functions in the graphics state for each of the four colors.

The addition of a colorimage operator will now make it possible for PostScript to render sampled color images in several data formats which are specified by Adobe along with the new color operator definitions. The new operators also provide a means of converting from the RGB color model to the CMYK color model, and allow the generation of a black printer based on RGB values. It is possible using the new setblackgeneration operator to specify a procedure that maps the user color components to the output device black values. Undercolor removal can be set similarly using a procedure that is called by the new setundercolorremoval operator. There are also operators to return the current function of each of these two aspects of the graphics state: currentblackgeneration and currentundercolorremoval.

Adobe is working on providing backward compatibility of these operators in existing black and white printers through PostScript language implementations. New black-and-white printers will have the capability of producing color separations from color PostScript files. This color separation capability will not be backward compatible as the frame buffer of existing machines would be too small and other significant changes to the PostScript interpreter would be required. Rather, the front-end will have to create the separations and send each to the output device followed by a separate showpage command, in a manner similar to the method used in the Cal Poly poster.

The Scitex Corporation has recently announced an agreement with Adobe Systems to build a PostScript RIP to work with the Scitex Raystar film recorder. It has also announced the intention of utilizing the emerging Display PostScript standard in its Response color workstations. Display PostScript is being developed by Adobe Systems to create a true WYSIWYG (What You See Is What You Get) display standard, which will exactly match the screen image to the output printed on a PostScript device, except in terms of resolution.

There have been a number of developments regarding companies which are attempting to "clone" the PostScript interpreter. Several of these are high-resolution imagesetters which are promising significantly faster processing speeds. These may have a positive effect on the development of color PostScript applications if the implementations are truly compatible.

The Quark Corporation currently sells a page makeup program for the Macintosh known as XPress 1.1 which is capable of separating spot color. They are currently completing version 2.0 which has extensive color support. In this new version the user will have a choice of one of four color models to work with: HSB, RGB, CMYK, and PANTONE. A palette of up to 256 colors per job may be defined using any of the above models. It is possible to switch color models at any time in a job. Selected objects are assigned to any one of the colors from the palette, which is displayable on a color monitor. The output of this software has been calibrated by Quark to the QMS ColorScript 100 printer using the new PostScript setcolortransfer operator to color correct for hue error in the printer's ribbon.

Additionally, Quark is preparing to release another version of the software known as Handshake XPress which has been developed to pass text, graphics, and page geometry to Scitex Response Systems. This means that it will be possible for a desktop publishing system to create a complete layout of a color page with type, color tints, and low-resolution scans (used as position prints), which can in turn be transferred to a CEPS (Color Electronic Pre-press System) where high-resolution scans may be merged with the text and page geometry; retouching, blending, trapping and the like may be accomplished; and finally the file may be output to a film recorder for final separations.

Future of PostScript Color

The new PostScript color operators will provide a standard way of communicating color definitions between applications such as word processors, illustration programs, and page makeup packages. The flow of data in a theoretical PostScript desktop color imaging system is illustrated in Figure 8. Components of a color page could be generated by a word processor, illustration program, or color scanner. Text files could contain colored initial capitals, subheads, or other color emphasis. Color images from the scanner could be brought up and altered in a pixel editing program or used as a tracing template in a color illustration program. The files created by these programs then would be combined into complete pages with a color page makeup program.

The advantages of device independence are very important when dealing with color. Color output will be possible at several stages. From the word processor, pixel editor, or illustration program, users can create output on the color printer. Also, completed color pages will be sent to this device. While some people will use color printers to output their final pages, for others these will only be proofs. With the same data they will be able to use a high resolution PostScript imagesetter and create graphic arts quality separations.

To the graphic arts professional PostScript has the potential of being more than just a page description language useful in the office environment and desktop publishing systems. As evidenced by this paper, color systems based on PostScript have the potential to create professional-quality color images. Recent developments in the industry indicate that PostScript is likely to help integrate inexpensive personal computers with high-end CEPS to create a more efficient and cost-effective color imaging environment.

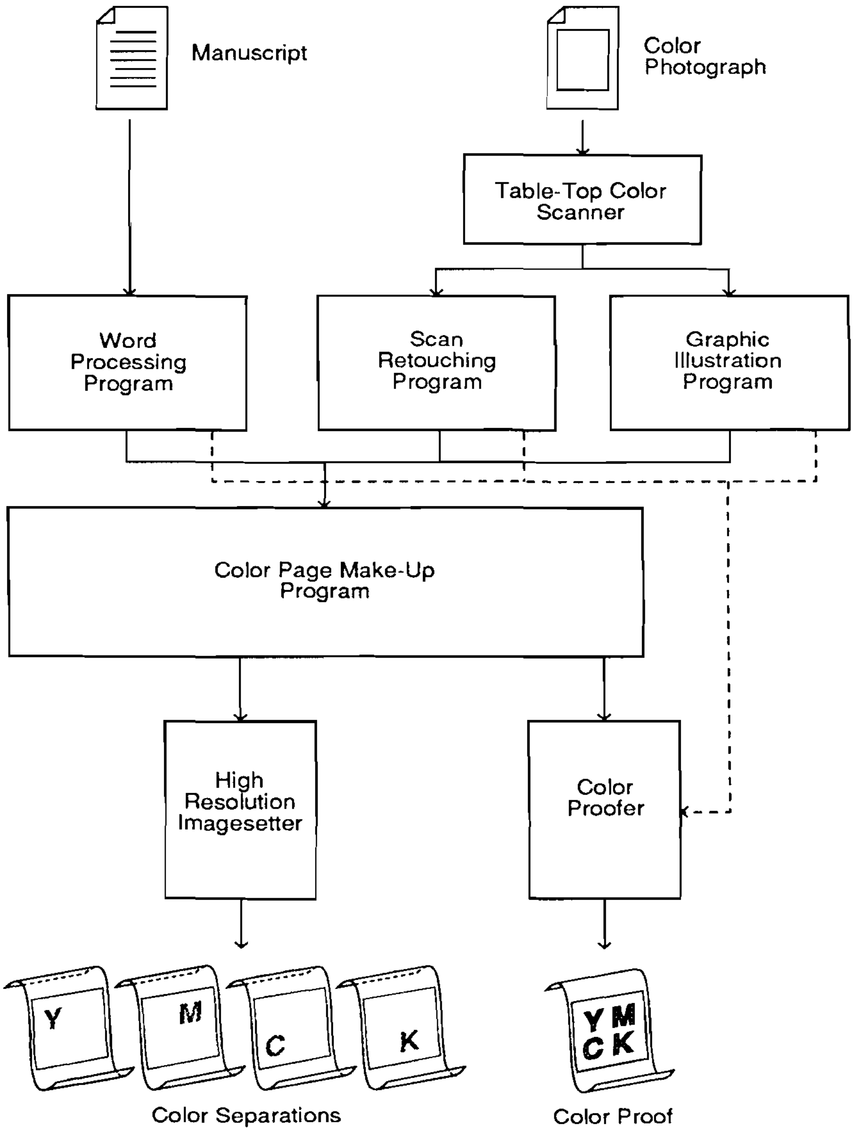

Figure 8
A Color Imaging System

Selected Bibliography

Adobe Systems Inc.

1988a. *PostScript Language Color Operator Definitions*, (Adobe Systems Inc., Palo Alto, CA) 21 pp.

1988b. *PostScript Language Program Design*, (Addison-Wesley Publishing Co., Reading, MA) 224 pp.

1988c. *Colophon: Adobe Systems News Publication*, vols.1-5, (October 1985-April 1988).

1986. *PostScript Language Supplement for the Linotype Series 100: Version 42*, (Adobe Systems Inc., Palo Alto, CA) 65 pp.

1985a. *PostScript Language Tutorial and Cookbook*, (Addison-Wesley Publishing Co., Reading, MA) 244 pp.

1985b *PostScript Language Reference Manual*, (Addison-Wesley Publishing Co., Reading, MA) 321 pp.

Holzgang, David A.

1987. *Understanding PostScript Programming*, (Sybex Inc., Alameda, CA) 459 pp.

Smith, Alvy Ray

1978. "Color Gamut Transfer," *Computer Graphics*, vol. 12, no. 3, pp. 12-19 (August 1978).

Warnock, John, and Wyatt, Douglas

1982. "A Device Independent Graphics Model for use with Raster Devices," *Computer Graphics*, vol. 16, no. 3, pp. 313-320 (July 1982).