IMPROVED COLOR FOR THE WORLD WIDE WEB: A CASE STUDY IN COLOR MANAGEMENT FOR DISTRIBUTED DIGITAL MEDIA

Todd Newman

Keywords: Communication, Images, Characterization, Colorimetry

Abstract: Distributed digital media need color management just as traditional printing does. However, no tools or practices exist for managing color on the World Wide Web. Consider a GIF image in a page of HTML on the Web. Pixel colors in the image are implicitly tied to characteristics -- such as phosphor chromaticity, gamma, and white point -- of the device on which it was created. Unless the display device miraculously happens to have exactly the same characteristics, image color is not preserved.

Color management can solve this problem, even though the color characteristics of the display system are not, and cannot be, known at the time the Web page is created. A technique is presented to embed in the GIF image an International Color Consortium (ICC) device color profile describing the source device. Armed with this and an ICC profile for the display device, the Web browser can then create and display a GIF image in the device color space of the display device. The benefits of applying this technique and some of the pitfalls are discussed.

Why the Web Needs Color Management

The World Wide Web has been tremendously successful. Both the number of users and the amount of material available on it have grown at an astonishing rate. There's hardly a reason to think about improving upon it. But as the Web grows, people are increasingly looking at the Web as a tool for commerce. And here is one shortcoming of the Web today: color reproduction on the Web is not nearly at the quality level needed for color-sensitive material, such as catalogue sales. Many products, such as lipstick and clothing, are purchased primarily or exclusively because of their color. If that cannot be reproduced with accuracy rivaling that achieved on paper today, the Web cannot successfully replace or augment paper-based sales channels.

Todd Newman is a Member of the Technical Staff at Silicon Graphics, Inc.
Mailing address: Mailstop 1L-945, 2011 N. Shoreline Blvd., Mt. View, CA 94043.
Email: tdn@sgi.com

Case Study

This paper describes a project to add color management support to the production and distribution of documents on the World Wide Web. The primary goal of the project was to improve the reproduction of color. To keep the project practical, I functioned under the following restraints:

- Existing established Web tools (in specific, the Netscape™ browser) were not to modified. I wanted to integrate cleanly into the Web as it now exists, to keep the scope of the project manageable, and to enhance the chances for acceptance. People are used to the look and behavior of their tools. I wanted to improve them, but not to change them.

- Color management had to be added in a way that was also compatible with browsing existing Web pages that did not support color management. Having to run the tools in one mode for most pages and in a different mode for color managed pages was not appealing.

- I favored designs that would work well for personal computers. This constraint acknowledged that most of the current users of the Web are on machines with relatively slow network connections, and moderate amounts of main memory. If the addition of color management slowed down browsing by too much, it would not be deemed worthwhile by many users.

The reminder of this paper explains the problem, including descriptions of the contents of HTML pages and how Web browsers currently present these pages. I include a section devoted to GIF image files. Next is a discussion of existing color management practices and an introduction of the concept of distributed color management. This is followed by a description of the color management solution to the problem. The paper ends with identifying areas for further research and a brief summary.

Presenting a Web Page

Pages on the Web are written in HTML, the HyperText Markup Language. (The specification for HTML is available, in HTML, on the Web at http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html) HTML consists of text, document structuring commands, and hypertext references. It is the job of the HTML browser to decide how to process hypertext references and to determine how to present the document based on the document structuring commands. (I use the term "present" is used because it sounds odd to speak of "displaying" audio data, which is often part of HTML pages.) Document structuring commands denote parts such as section headings, lists, extended quotations, and the body of the document. The browser determines how to present each of those sections by selecting the appropriate font, page layout, and so on.

In the source file, hypertext references look like the token "http://" followed by something that resembles a UNIX® file path. These are called "universal resource locators" ("URLs". The Web is worldwide, but the resources are universal.) Again, the Web browser determines how each reference is handled. References to other HTML documents are usually handled by displaying some highlighted text. If that text is selected with a mouse, the current page is replaced with the referenced page. References to images are handled either by bringing up an image viewer or by displaying the image integrated into the document. References to audio files invoke a sound player; references to movie files, a movie player. The mechanism that allows browsers to

determine how to handle references is very flexible, at least on UNIX systems. It is that flexibility that allowed my project to succeed.

The Web uses the MIME format to transport referenced data across the Internet. This format tags the data with a "Content" field that indicates the type of the data. UNIX-based Web browsers then use a **mailcap** file to determine how to present data based on its type. Users may provide their own mailcap file. Any data types not supported in the user's mailcap file will be searched for in a system default mailcap file. (Actually, there is a hierarchy of default files. The system search through a list of mailcap files, stopping when it encounters a rule covering the data type.) Any types still not recognized cannot be presented by the Web browser. mailcap files are plain text files. Each line of the file specifies a media content type and the UNIX program to be used to present that type. For example, my mailcap file contains the following lines:

```
audio/*; playaiff %s
video/mpeg; movieplayer %s
application/postscript; ghostview %s
message/rfc822; xterm -e metamail %s
application/x-tardist; tardist %s
```

If an audio file is found, the program **playaiff** is invoked with the audio file as an argument. If an MPEG movie is found, **movieplayer** is invoked with the movie file as an argument.

This scheme makes it easier to port Web browsers to different UNIX systems. The implementer of the browser does not have to write a new set of multimedia tools, or include them in the browser program. Different tools of equivalent functionality can be substituted on different vendors' systems or at the whim of the skilled user. (UNIX systems are very big on catering to the whims of skilled users.) For example, Silicon Graphics (SGI) provides multiple ways to view GIF images; among them are the applications **xv** and **imgview**. The default configuration of our system level mailcap file invokes imgview when presented with a GIF image file. But by putting the line:

```
image/gif; xv %s
```

in my mailcap file, I can choose to invoke the program xv instead.

Images can be referenced in HTML in two ways. One is with the tag IMG, which explicitly marks the referent as an image. The other way is with the tag HREF, which can be used for any hypertext reference. This leaves it up to the MIME mechanism to determine the type of the reference and to the mailcap mechanism to determine how to present the referent. The default browser on SGI's system, Netscape's Navigator recognizes IMG and displays the image in the same window as the rest of the document. This makes for a better-looking document. The browser is not capable of recognizing when a generic HREF refers to an image file; such references are left to be handled by the mailcap mechanism. Figure 1 shows how a Web browser determines how to present HTML.

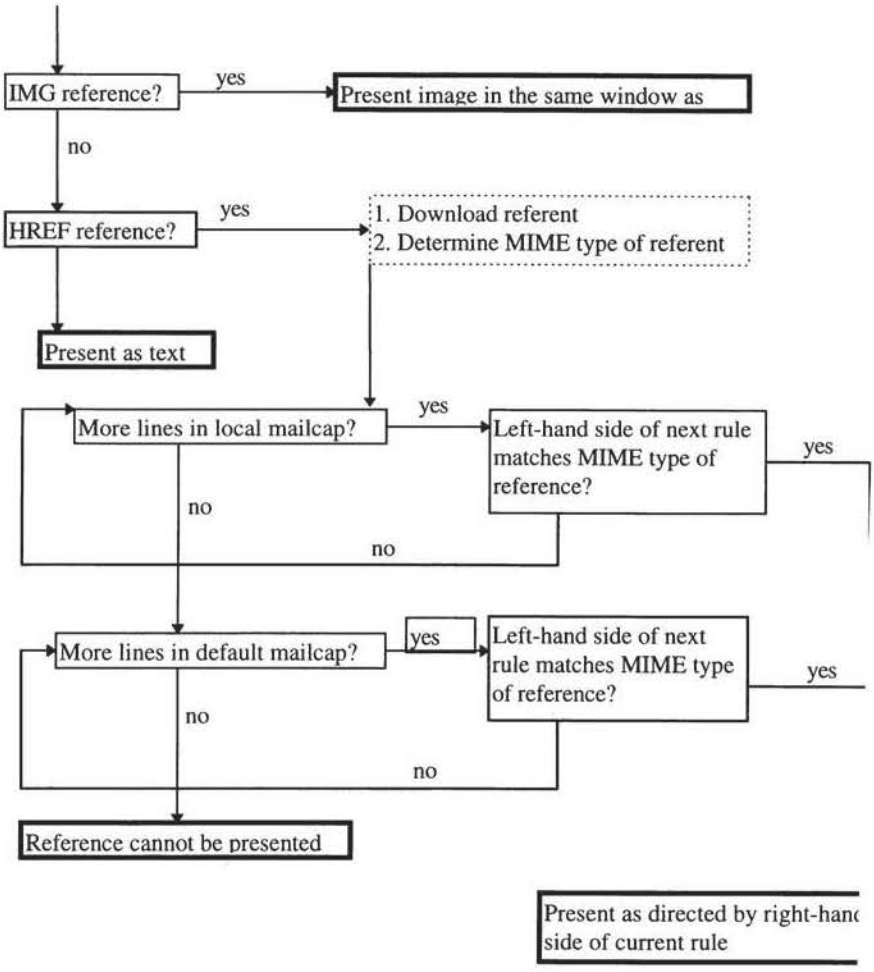

Figure 1. How a browser determines HTML presentation.

GIF Image Files

While any image type can supported in an HTML document, the vast majority of images availal on the Web are GIF (Graphics Interchange Format ©) images. GIF images are either bitonal or colored. A GIF image is a two-dimensional array of pixels. The pixels do not directly represent colors, but are indices into a color lookup table. The size of a color lookup table must be a pow of 2 between 1 and 8. In other words, color tables have between 2 and 256 colors. All the color

in the table are specified with three components: red, green, and blue. The components are each 8 bits deep. Thus, an image can have up to 256 colors from a palette of approximately 16 million.

For the purposes of this paper, the most important thing to remember about the GIF format is that all the pixels in images are described as red, green, and blue values. This is called an RGB color space. However, the meanings of "red," "green," and "blue" are not well defined. Most software interprets the colors as if they were in the RGB space of the display monitor. This may or may not resemble the RGB space of the monitor on which the image was created, depending on the color characteristics of the two monitors. Often images are created on desktop scanners. While these are RGB devices, the spectral response of the red, green, and blue primaries, the white point, and the tone response curves are often different from those of a computer monitor. Whatever the source of the image, any resemblance between the display device's color space and the source's is purely fortuitous.

Web pages contain a mixture of text and references to data in other formats such as images, movies, and sound. We have seen that the color for referenced GIF images is based on the device (monitor or scanner) that the image was created on, but the exact meaning of the colors is no longer available when the image is displayed. The next section describes how this problem relates other color management problems.

Traditional Color Management

Color management for the Web is different from more traditional pre-press color management. Most pre-press systems are tightly coupled. That is, the scanner, the monitor, and all output devices (such printers or proofers) are all running on the same system. This means that the color characteristics of the destination device are fixed at the time the image is acquired. There is not much difference between adjusting the color as it comes out of the scanner or adjusting it before it is output. See Figure 2.

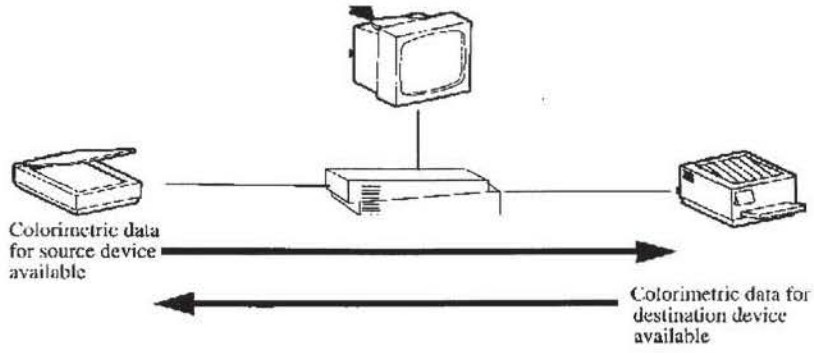

Figure 2: A tightly coupled system. Colorimetric data for any device is available at any point.

The situation is similar when a large-scale sheet- or web-fed printing press is the ultimate target. The press operator is required to maintain the press in close adherence to an industry standard

such as SWOP or Euroscale. Once the standard and paper type are known, the system, although geographically distributed, is again tightly coupled. The color management system can correct to the reference standard as early as desired. In fact, it is not uncommon for the scanner to output image files in the printing press' CMYK color space.

In contrast, the Web is loosely coupled. The creator of a document does not know how many different systems will ultimately present that document, what type of devices will be used to present the document, nor even what on what medium the document will be presented. Most Web browsers can print color images so an image may be viewed on a monitor, printed on paper, or both. And the number of different kinds of printer, paper, and ink used is impossible to calculate.

Colors cannot be adjusted for final display as they are scanned. Nor can they be adjusted at any point during document creation. The only time this can be done is at document presentation–when the final presentation medium and device are determined. But just as the creator of the document does not know the color characteristics of the device on which the image will be displayed, the displayer of the document does not know the color characteristics of the device on which the image was created. See Figure 3.

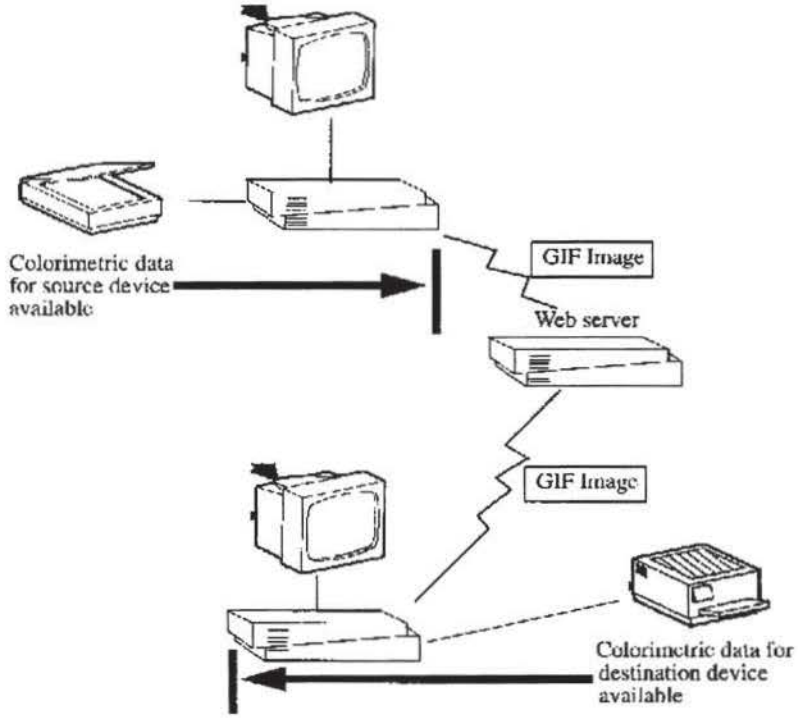

Figure 3: A loosely coupled system. Colorimetric data cannot move between the source and destination machines.

The solution to the problem of the presenter not knowing how an images was created is in two parts:

- At document creation time, map the source devices color space into a well-know color space
- At document presentation time, map from the well-known color space into the display device's color space.

With this solution, the creator of the document does not need to know how it will be presented or the color characteristics of devices on the other side of Web. By the same token, the person displaying does not need to know how the document was created, but knows the display device and has enough information about the color space of the source device. The solution distributes the color management task between the creator and the presenter.

The best way to provide the mapping of the source device's color space is to embed the mapping information into the source image itself. The mapping information could be put in a separate file, but it is quite likely that the two files would be separated at some time. If the information is part of the image, there is only one file to manage and much less chance of losing the color information.

This solution also works for documents that include more than one image and whose images may have been created on different devices. One document may reference many images created on different devices, each with its own device-dependent color space. The images should, of course, be color-adjusted separately. So an overall document color profile is not adequate. Each image can have its own embedded color information, and the color adjustment can be done on each image. See Figure 4.

This is not the only possible approach. All the images can be translated into a reference color space and stored in the document that way. This is how the tightly coupled systems work: they move everything into SWOP CMYK (or CIELAB) and every image in the document is in the same color space. SWOP and Euroscale are not be appropriate spaces for GIF files, however, because GIF supports only RGB. The same incompatibility precludes storing the images in one of the CIE color spaces. None of the Web browsers support CMYK- or CIE-based image formats directly. Since compatibility with existing software was a project requirement, extending the GIF format by embedded color descriptions seemed like the best solution.

Another approach is to provide all the image data in both the RGB color space, for compatibility with existing software, and in a CIE color space, to support color management. This doubles the size of the image, which is unacceptable since it is unnecessary. Instead, store the data once in the traditional device-dependent RGB space, then provide additional information to allow mapping from that space into a CIE space later on.

Color management for the Web presents a new paradigm for color management. The system used to create documents may be separated from the system used to present the documents. Solve the color management problem by embedding a description of the source device's color space in the image itself. Then the display system can map device colors into its own display device's color space. In this way, what the creator saw will closely match what the ultimate reader sees.

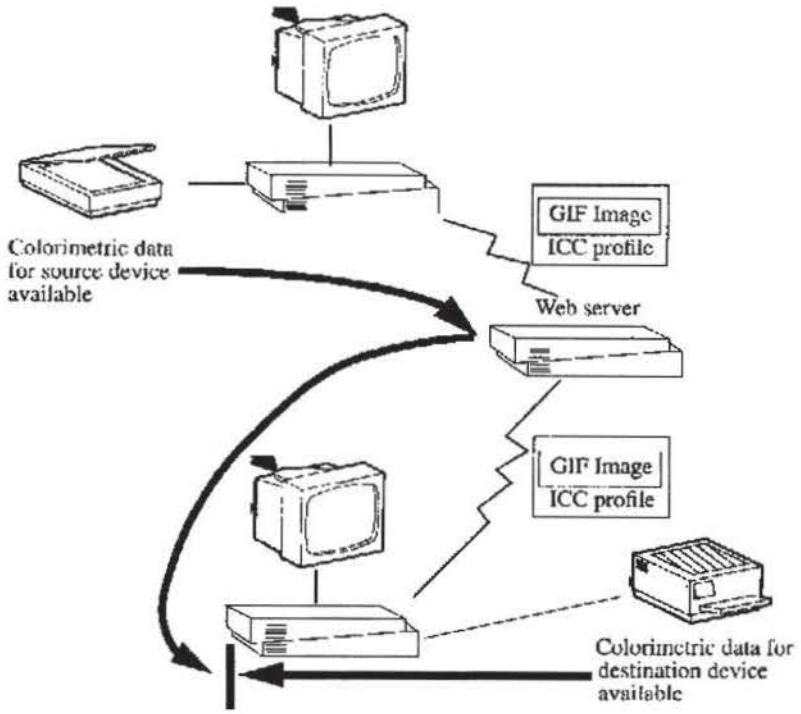

Figure 4. Embedding profiles in images. Colorimetric data from the source is available on the destination machine.

Color Management for GIF Images

Since distributed color management is made up of two parts, two tools were needed: one to run on the source system to tag a GIF image with a device color description, and one on the destination system to process the tagged image and adjust the color for the display device. Each tool posed its own problems. For the tagging tool, the problem was to find a way to extend the GIF format in a manner compatible with existing tools. For the color adjustment tool, the problems were to find a way to intervene in the browser's image display process, to find a characterization of the display device's character space, and to perform the actual color adjustment. Obviously, the tagging and adjusting tools had to use the same method of describing the device color space.

First, I will present the common method for describing device color spaces, the ICC device profile. Then I will show how the device profiles can be embedded into GIF image files. Once that is possible, actually writing the tagging and adjustment tools is quite straightforward. The only remaining challenge was finding a opportunity to apply the adjustment tool within the browsing process.

Characterizing Colors: ICC Device profiles

I chose to use the International Color Consortium (ICC) device profiles, both for the embedded characterization of the source device's color space and for the characterization of the display device's color space. The ICC profiles provide a mapping between the device color space and either CIEXYZ or CIELAB. Although the format is quite new, I had access to a wide variety of profiles for scanners, monitors, and printers. The profiles are quite portable. The same profile can be used under the Macintosh™, Windows 95™, Solaris™, or SGI's IRIX™ operating systems. While porting to other platforms was outside the scope of this project, selecting a mechanism that would have been easy to port seemed a good plan.

The color management system under development at SGI supports the ICC profiles. It also provides a mechanism to find the profile for the workstation's display. It was a matter of a day's work to write a simple program, **cmdecodegif**, that adjusted the colors of a GIF image, once I had figured out how to embed an ICC profile. Embedding was done with a program called **taggif**.

Extending the GIF Format

To understand how the GIF file format was extended, it is necessary to understand in some detail the structure of a GIF file. The following description is much simplified, but suffices for the purposes of understanding this project. GIF files are designed to serve a number of different purposes. The most common use is for a file to contain a single image, and a Color Table. A Color Table is a one-dimensional array of three-component entries. Each entry has a red, a green, and a blue component. The table provides the mapping between the index stored in the image and the device-dependent color space of the display device. A file may also contain multiple images, each with its own Local Color Table, or it may default to use the Global Color Table. Within a single file, some images may use the Global Color Table and some their own Local Color Table. Further, it is possible to define a file that contains no images, but only sets the contents of the Global Color Table for use by subsequent files. Unfortunately, all that flexibility does come at the expense of increased code complexity.

A GIF file is composed of a series of typed blocks. Some blocks contain image data, some contain color table data, and some control the interpretation of subsequent block or set state needed for processing other blocks. The sequencing of blocks is defined by a grammar (GIF 1989). The GIF grammar uses the following set of symbols:

<>	indicates a defined symbol in the grammar
::=	defines a symbol
*	indicates zero or more occurrences
|	indicates an alternate element
[ ]	indicates an optional element

The grammar is then presented as follows:

```
<GIF Data Stream> :: =          Header  <Logical Screen>  <Data>*  Trailer
<Logical Screen> ::=            Logical Screen Descriptor  [Global Color Table]
<Data> ::=                      <Graphic Block>  |  <Special-Purpose Block>
<Graphic Block> ::=             [Graphic Control Extension]  <Graphic-Rendering Block>
<Graphic-Rendering Block> ::=   <Table-Based Image>  |  Plain Text Extension
<Table-Based Image> ::=          Image Descriptor  [Local Color Table]  Image Data
<Special Purpose Block> ::=     Application Extension  |  Comment Extension
```

This grammar is fairly easy to read. The first line may be interpreted as follows: a "GIF Data Stream" (or GIF file) is composed of a Header, followed by a Logical Screen, followed by zero or more instances of Data, followed by a Trailer. Each of the terminal symbols (Header, Trailer, Logical Screen Descriptor, Plain Text Extension, etc.) denotes a different kind of block. The contents of each of the block types is defined elsewhere in the GIF specification.

Color Table blocks (both global and local) were the focal point for this project. The tagging tool needed a way to associate a device profile with each Color Table block. The adjustment tool needed to find the profile and the Color Table and create a new adjusted Color Table. In fact, both tools are written as UNIX filter programs which read in a GIF file and command line arguments as inputs and write out a suitably modified GIF file as output.

The only user-definable block offered is the Application Extension. Different user-defined blocks are indicated by an Application Identifier that begins the Application Extension block. So that was where the embedded profile information had to be placed. Since this was only an experiment, I created an Application Identifier, the string "ICCRGBG1012", but did not register it with CompuServe. It was an unfortunate complication that the block had to be placed after a Global Color Table but before a Local Color Table. Examining the grammar, the grammatical production that places an Application Extension block near a Global Color Table works as follows: each line in the example below is produced from the preceding by expanding one non-terminal symbol according to the rules of the grammar:

```
Header  <Logical Screen>                <Data>*
Header  Logical Screen Descriptor  Global Color Table  <Data>*
Header  Logical Screen Descriptor  Global Color Table  <Special Purpose Block>  <Data>*
Header  Logical Screen Descriptor  Global Color Table  Application Extension  <Data>*
```

The Application Extension block containing the embedded profile immediately follows the Global Color Table. But examine the following production:

```
<Table-Based Image> ::=  Image Descriptor  [Local Color Table]  Image Data
```

If I place an Application Extension block containing a profile after the Table-Based Image, it not only follows the Local Color Table, which is convenient, but it also follows all the image data. To process any Color Table with a profile, both the color table and the profile must be buffered. If the profile followed the image data as well, that too would have to be buffered. But an image file can easily be a million bytes long. That is a lot of buffer space to require. It might be feasible on a UNIX workstation but seems ludicrous for a personal computer. The grammatical production that places the Application Extension as close as possible to a Local Color Table is:

```
<Data>*
<Special Purpose Block>  <Table-Based Image>  <Data>*
Application Extension  Image Descriptor  Local Color Table  Image Data  <Data>*
```

Even though this leads to an asymmetry in processing global and local and much easier to buffer than the Image Data color tables, it keeps the relevant data at hand. The Image Descriptor is small and much easier to buffer than the Image Data.

Intervening in Image Display

At this point, I knew how to embed a profile into a GIF file, which solved the encoding problem. I also knew how to perform color management on a GIF file with an embedded image. The next problem was how to get that color adjustment to be invoked by Web browser. Fortunately, most of the mechanism was already in place. The solution described below works on SGI systems, and I believe it can work on any UNIX system. Implementing it on other types of operating systems should be feasible, but the details may vary.

The first step to intervention is to ensure that all images for which colors are to be adjusted are referenced as HREF and not as IMG. As explained earlier, the Netscape browser automatically integrates images referenced as IMG, but those that are simply marked as HREF are processed according to the rules established by the mailcap mechanism. The Mosaic™ browser never integrates images and processes only according to the dictates of the mailcap mechanism. There is no mechanism provided within the Netscape browser to provide access to images if they are integrated in the main document window. So no color management can be performed. This is unfortunate, and should be corrected in a future version of the browser .

The mailcap file is consulted for all HREF references. In particular, it is consulted for the images that are to have their color adjusted. In theory, the UNIX pipe facility here could be used. Assume that the color adjustment program is named cmdecodegif, that it is a filter that inputs and outputs GIF files, and that SGI's standard image viewing program is named imgview. I could simply create a mailcap file with the line:

```
image/gif; cmdecodegif %s | imgview
```

which says that to display a GIF image, run the cmdecodegif program with the image file as input data, and then take the output of that and feed it as input to the imgview program. Unfortunately, the SGI imgview program cannot read its input from a UNIX pipe. (This is probably a bad design decision.) So this theoretically simple approach was not what I implemented. Instead, I wrote a simple UNIX shell script. A shell script is a series of UNIX commands which operate as if they were a single program. The entire shell script reads:

```
~tdn/bin/cmdecodegif -i $* > /usr/tmp/tdntest
imgview /usr/tmp/tdntest
```

The only difference between this and the UNIX pipe command described above is that the output of cmdecodegif is placed in a temporary file called tdntest. This is sufficient for testing purposes, but a better method for generating temporary filenames should be used. The shell script was named cmview.

Given cmview, the mailcap file I actually use contains the line:

```
image/gif; cmview %s
```

That invokes the shell script that invokes the program cmdecodegif. That program reads the image that the browser passed, adjusts the color, and writes the output to a temporary file. Then the regular image viewer, imgview, is used to display the color-adjusted image on the monitor.

An Enhancement

An ICC device profile can be as small as 500 bytes for a minimal monitor profile. But it is not uncommon for the profile for a scanner to exceed 20000 bytes. While this is only 2 percent of a one megabyte file, which is not uncommon on UNIX workstations, and still only 6 percent of a 640 by 480 full screen PC file, it could be a significant factor in small image files. What I needed was a way to maintain the benefits of sending a full ICC profile and not incur such an increase in file size.

The solution was simple: translate each color in the palette into a reference color space, place the translated palette in an Application Extension block, and do not send the ICC profile. The only reason for embedding the profile was to allow the adjusting tool to understand the color space of the device on which the image was created. Because the color tables are available in a reference color space, no more information is needed about the source device. The CIELAB data is as device-independent and portable as an ICC profile. We would be sending at most 256 CIELAB values. It seems that 8-bit CIELAB data is sufficiently accurate, so this is only one byte times 3 components times 256 entries, or 768 bytes of data. While this is larger than the smallest monitor profile, it is much smaller than a typical scanner profile. Both techniques are easily supported in the same tagging and decoding tools. So I can choose whichever technique creates the smallest resulting image.

It is important for compatibility that the translated colors be stored in an Application Extension block and not in the original color tables. Most of the existing GIF applications and image viewers do not support CIELAB, nor would there be a way to signal to them the color space of the GIF data. We have to use the Application Extension block to maintain compatibility.

Results

Normally, a results section in a technical paper would be filled with measurements and quantified data. However, this project was more qualitative than quantitative in nature. It was not a goal to provide an assessment of how well color management works, or how closely the image on the display monitor could be brought to resemble the image on the source monitor or that being scanned. Those are tests of the quality of the underlying color management system and might be a fit subject for another research project. Indeed, it has doubtless been so.

Instead, this was a project to assess the feasibility of providing color management in a loosely coupled distributed system and to do so compatibly with a large base of installed software. The primary results are therefore a binary choice: either it was achievable or it was not. This paper has shown that it is feasible and how it can be achieved.

Nevertheless, I felt a strong desire to see whether the addition of color management to the Web would produce images that "looked better." Two factors were of particular concern: the relative scarcity of color calibrated monitors, and the fact that GIF images have a palette of only 256 colors.

Of course, the best results require monitor calibration. There are two ways to calibrate the monitor. The first is to generate an ICC profile that describes the current state of the monitor. One of the test monitors was a Barco Reference calibrator. Software provided with the monitor provided information such as the phosphor chromaticities, the white point, and a gamma curve for each color channel. From this, it was easy to generate an ICC profile. The second approach to calibration is to bring the monitor's response into line with a stable ICC profile. This approach

Dithering

As mentioned above, the color quality of images can be traded against spatial resolution by using standard dithering techniques such as ordered dithering or error diffusion. Images with a small enough palette of colors will look the same. Depending on the content of the image, images with larger palettes may or may not be noticeably less detailed. For some purposes, the loss of detail will prohibit dithering. For others, the improvement in color accuracy will make dithering an attractive alternative.

Web pages often contain multiple images. All of these images will be displayed on one computer screen in one window. If that window uses an indexed color map, then all the pixels in all the palettes for all the images in the window must be allocated out of that same color map. It is not difficult to run out of color table space in an 8 bit window, even if no individual image uses more than 256 colors. Because this has been a problem, SGI's WebMagic™ authoring tool offers a way to allocate a color cube within that color table and then use error diffusion to encode all the images for the page. Results are astonishingly good with only a 4 entry color cube. With a 64 entry cube, results are indistinguishable from the original for all but the most demanding images. Combining this approach with a color management solution to adjust the axes of the color cubes could produce even better results.

Standard RGB Space

If a device color space could be adequately specified as a standard, then colors could be transferred in that space without the need to pass information describing that space with every image. Ralf Kuron, of FOGRA, has suggested that a standard RGB space be defined. This standard could then be used for transferring GIF images. He proposes that this space be a representation of the "average" monitor.(Kuron 1995) The document creator would use a color management system to convert from the source device's color space into this standard RGB. If the person viewing the document has a calibrated monitor and CMS, then the image could be adjusted at display time. If not, the uncorrected image display would be "good enough," or at least it could not be improved upon.

The color management industry has frequently searched for acceptable image interchange color spaces. To date, no consensus has been reached. It is not clear whether the search for a standard RGB space would do better. Any standard choice of phosphor chromaticities, white point, and gamma will favor some vendors at the expense of others, because either no conversion, or only minimal conversions, would be needed. Also, given the extent and speed with which devices drift from calibration, it is quite possible that results would be no more accurate than they are today. Nevertheless, the idea of standardized image interchange color spaces has great appeal and should be investigated further.

Summary

The project showed that it is possible to introduce color management into a distributed document production and viewing environment. Color management on the World Wide Web is feasible and can be added in a manner compatible with current tools. The GIF extension requires a small amount of work. It can be implemented to add only a very small amount of data per image. Because the processing is relatively lightweight, it has no noticeable performance impact. Finally, and most importantly, colors look better. Given this initial success, further research should be done in several of the indicated areas.

Acknowledgments

I would like to thank Victor Reilly and Ashmeet Sidana of Silicon Graphics for their assistance in this project.I would also like to thank Michael Has and Ralf Kuron of FOGRA for comments on earlier drafts of the document.

Literature Cited

(1) Graphics Interchange Format, Version 89a, CompuServe Incorporated, Columbus Ohio, 1989.

(2) Ralf Kuron, "Accurate Colors in Online Systems," unpublished draft, private communication, August 10, 1995.