Colour Spaces and Colour Contrast

Avi Naiman

Department of Computer Science
University of Toronto
Toronto, Ontario, M5S 1A4

ABSTRACT

With the introduction of low-cost colour graphics systems come a host of problems specifically concerned with the colour aspect of the system. This paper discusses two of these problems: the selection and manipulation of colours by (possibly) inexperienced users, and the automatic selection of colours by the system to achieve high contrast effects on the screen. A new colour model based on colour opponency theory is described. This model is useful both in the user interface and in the automatic selection of high contrast colours.

RESUME

L'introduction de systèmes graphiques en couleur de bas prix introduit une pléthore de problèmes associés à l'apparition de la couleur dans le système. Cet article discute deux de ces problèmes: la sèlection et la manipulation des couleurs par les utilisateurs (possiblement) inexpèrimentés, et la sèlection automatique des couleurs par le système pour obtenir un haut contraste sur l'écran. Nous décrivons aussi un nouveau modèle de couleurs, fondé sur la théorie des couleurs opposantes. Ce modèle est utile à la fois pour l'interface à l'utilisateur et pour la sèlection automatique de couleurs à haut contraste.

KEYWORDS: Colour, Colour Contrast, Colour Models, Colour Selection, Colour Spaces, Colour Specification, User-Interface

Introduction

Colour is an increasingly affordable and common feature of graphics displays — be it in educational environments, professional workstations, or home computer systems. It has already become a significant component of computer aided visualization of information, concepts, and ideas. As the proliferation of colour raster CRT systems continues, colour becomes increasingly tantalizing to programmers with little or no background in the technology and techniques of colour computer graphics. Both computer graphics programmers and users — typically novices with respect to colour — find themselves faced with the problems of colour specification and modification.

Most Computer Graphics applications available today offer the user little or no freedom in selecting colours. Often, only a predefined palette of colours is available from which the user must make a selection; no method is provided for the modification of the selected colours. Even those applications which do provide more useful tools typically restrict the flexibility of colour specification by allowing the user only to specify relative amounts of the red, green, and blue components of the colour — directly corresponding to the values to be supplied to the digital-to-analog-converters which control the three electron guns sweeping out the image on the monitor. It has been recognized for a while that this limitation can considerably hamper a user's creativity in using the colour dimension of system [BERK82b, FOLE82, JOBL78, MEYE80, SMIT78, SCHW84, and TANN83].

In this paper, we review the more common colour spaces used in Computer Graphics today and discuss the problems associated with using these spaces for specification and selection of colour. Furthermore, as it is often most important that certain foreground objects (e.g., tracking symbols) be highly visible, we analyze the usefulness of the various colour spaces in the context of automatically selecting the colour of the foreground objects against coloured backgrounds. Finally, we describe a new colour space we have developed based on opponent colour theory.

Colour Spaces

For the casual or inexperienced user, the process of specifying colours to be used in a scene, report, or package must be easy to learn, use, and understand. This goal is better achieved when the underlying specification model is intuitively clear.

A *colour space*† is a specification of a 3D coordinate system and a 3D subspace within which each displayable colour is represented by a point. The purpose of a colour space is to allow for the convenient specification of colours within some colour gamut — the range of colours producible on or recognizable by some medium [FOLE82]. Computer Graphics is primarily concerned with presentation of coloured output in video, film, and print, each of which is a subspace of the human colour gamut. Our prime concern here is the gamut for colour video as defined by the red, green, and blue primaries corresponding to the red, green, and blue phosphors in a colour display device. All colour spaces we use, then, must be invertible functions of the valid red, green, and blue inputs to the DACs.‡

In order to specify a colour, the user need not know the details of the space being used; only the concepts behind each of the dimensions in the space and how the dimensions interact to produce a colour need be understood. For instance, how the colour will be realized on the monitor (i.e., the transformation that must be applied to the colour space in order to produce appropriate voltages for the electron guns) need not be understood by the user.

Three issues affect the usefulness of a colour space: *ease of implementation* (the difficulty of developing conversion algorithms between the colour space and one which can be used to drive the display's electron guns), *efficiency of implementation* (in terms of the speed and accuracy of the conversion algorithms), and *ease of use* (measured in terms of the ease with which a wide selection of colours can be found and the naturalness of the dimensions of the colour space). Unfortunately, these issues are usually considered in decreasing order of importance when designing graphics applications today. It would be more appropriate for the user's flexibility to be the primary consideration in providing colour manipulation tools — within the limitations of the system.

Common Colour Spaces

The RGB (for, red, green, blue) colour space was originally implemented in hardware (i.e., the electron guns) because of its relation to the red, green, and blue cone systems in the human eye. This subsequently influenced the selection of the RGB space in software as an interaction tool — despite its unintuitiveness. From a programming standpoint, the RGB space is very appealing, as the only transformation needed to obtain the requested colour on the monitor is quantization.

All other colour spaces must be specified in terms of invertible functions of RGB.† In addition to RGB, the following alternative colour spaces have been used with some success in recent applications and research environments. More in-depth treatments of the various spaces can be found in [FOLE82, NAIM85, and SMIT82].

The YIQ space is an encoding of RGB established by the National Television Standards Committee (NTSC) in 1953 for the transmission primaries of broadcast colour television in North America. The Y dimension, called *luminance*,‡ measures the

---

† In the Computer Graphics literature, the terms *colour model* and *colour space* are often used interchangeably. To avoid ambiguity, we will reserve the term *model* to refer to a model of vision and use *space* to refer to a 3-dimensional system of colour representation.

‡ Note that, if images originally created on a video monitor must be faithfully reproduced on film or by printing, the full video gamut of colours must not be used since the gamuts for films and inks are considerably smaller than the video gamut.

† Note that, since CRT phosphors produce intensity levels which exhibit a non-linear relationship to the DACs' inputs, we must compensate by employing a mapping function (via a lookup table) between the computed intensities and those actually displayed on the CRT. Techniques for compensating for these hardware non-linearities, often called *gamma correction*, can be found in [CATM79 and COWA83].

‡ As is traditional in the Computer Graphics literature, we use the term *brightness* loosely, and interchangeably with lightness, whiteness, and luminosity.

brightness perceived by a human watching a typical home television receiver and matches the human luminosity response curve. The I and Q dimensions contain the *chrominance* components of chromatic colour. Though this space is merely a linear transformation of the RGB space, it was developed to meet certain criteria for TV transmission; it is not at all intuitive and is attractive neither to a programmer nor a user.

The HSV space, proposed by Smith, is the first colour space for computer graphics to be user-oriented, being based on the intuitively appealing notions of the artist's tint, shade, and tone concepts [SMIT78]. In this system, each colour is considered to be a pure *hue* modified by a *saturation* and a *value*. Hue is the dimension with points on it normally called red, yellow, blue-green, etc. Saturation measures the departure of a hue from achromatic (i.e., from white or grey). Value measures the departure of a hue from black. Though this space needs a fairly intricate transformation in order to specify appropriate voltages to the monitor's electron guns, from a user's point of view it is quite easy to learn and, once mastered, very easy to use for colour specification and selection.

The CNS colour space is based on commonly understood English words, whereby a colour is specified by a structure that follows a few simple, systematic rules [BERK82a]. Hue, lightness, and saturation are allowed for in the syntax, corresponding to the HLS colour space, which is a deformation of the HSV space. Aside from the fact that the CNS space is language dependent, two major problems exist with it. First, each person's preconceived notion of what colour a particular colour name represents is slightly different (in fact, there can be significant differences around greens [BOYN79]). Second, it is not well-suited for the fine tuning of colours, necessitating the use of one of the other colour spaces when further modification is needed. However, users of CNS achieve considerably higher accuracy when naming displayed colours than users of RGB or HSV [BERK82a].

Recently, Schwarz et. al., proposed a new colour space based on opponent colour theory [SCHW84]. This model of human colour perception, accepted today by most colour vision scientists, states that the red, green, and blue cones in the retina of the eye feed into a second stage of processing in which their values are summed and differenced to produce three new channels: an *achromatic*, or *luminance*, channel, a red-green opponent channel, and a yellow-blue opponent channel (figure 1).

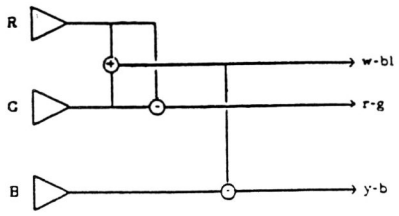

Figure 1. The opponent colours model.

The justification for introducing a colour space based on this model is twofold. First, this space, like YIQ and, to a limited extent, HSV and HLS, separates the luminance information in a colour from the chrominance content. Second, opponent colour theory explains a number of visual phenomena, such as why colours are never described as being composed of red and green or yellow and blue [BOYN79]. Furthermore, as red and green as well as yellow and blue are complementary colour pairs — that is, an appropriate mixture of the two colours can produce white — it is useful to separate them to opposite ends of colour dimensions.

One major drawback of the colour space described in [SCHW84] is that it requires a significant amount of computational power to make a real-time environment possible. Therefore, we have implemented a modified version of their colour space, which we call ARgYb (for Achromatic, Red-Green, Yellow-Blue).

We compute the intensity channel as a weighted summation of the red, green, and blue components of the colour as determined by the electron-gun stimulus values. This allows us to encode the luminance channel in exactly the same way as the Y channel in the YIQ space. Recall that this transformation matches the human luminosity response curve.

We then use a simplified formula to map the chrominance into the red-green and yellow-blue channels. The formula assigns equal weights to the red and green values in determining the red-green channel, and equal weights to the red plus green and blue inputs in determining the yellow-blue channel. The major benefit of our implementation is that it is very efficient for converting to and from RGB, while still providing an intuitive input mechanism related to the opponent colour theory model. However, it does not accurately implement the colour opponency model. While we feel that this colour space provides an advantageous tradeoff between accuracy and efficiency, it remains to be thoroughly tested to judge its usefulness.

The transformations can be written as follows:

$$\begin{bmatrix} A \\ Rg \\ Yb \end{bmatrix} = \begin{bmatrix} 0.3 & 0.59 & 0.11 \\ 0.5 & -0.5 & 0 \\ 0.25 & 0.25 & -0.5 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.29 & 0.22 \\ 1 & -0.71 & 0.22 \\ 1 & 0.29 & -1.78 \end{bmatrix} \begin{bmatrix} A \\ Rg \\ Yb \end{bmatrix}$$

The User Interface for Colour Selection

A common user interface, then, is a triplet of numbers corresponding to the three values of the colour in whatever colour space is implemented, which the user is allowed to modify, either through keyboard entry, or, more effectively, through the control of input devices such as potentiometers or sliders (physical or virtual). We have implemented such an interface in a colour experimentation package which allows for the simultaneous representation of the current colour in two colour spaces through virtual potentiometers on the screen. As the user modifies the colour using one set of potentiometers, the representation in the alternate colour space is dynamically updated. This often provides insight into the relationships between various dimensions. For example, when the selected colour spaces are RGB and HSV, it becomes immediately apparent to the user that the Value dimension of HSV is computed as the maximum of the R, G, and B components of the colour.

The experiment reported in [SCHW84] attempted to show which dimensions from different colour spaces, and what interaction techniques for altering those dimensions, are most useful as colour selection tools. Although not yet conclusive, their data indicates that spaces which separate the luminance from the chrominance are more useful than those that do not, and using a two-dimensional chrominance input technique is more useful than two one-dimensional input channels [COWA84]. The first conclusion appears well-founded, as there are only about 200 distinguishable levels of hue and 20 distinguishable steps of saturation, whereas there are about 500 distinguishable steps of brightness for every hue and grade of saturation [BOYN79].

Unfortunately, the opponent colour space which they implemented did not perform significantly better than other, similar colour spaces. This may be because, although we process colour stimuli in one manner (that proposed by colour opponency supporters), we do not necessarily *think* about colours in the same terms. However, we will see that the concept of a colour space based on opponency theory can be very useful in terms of automatic selection of optimally visible colours.

As users gain familiarity with the various colour spaces, situations crop up where the dimensions from a particular colour space are the most appropriate ones to use in modifying colours; what is easy to do in one space can be very difficult in another. Therefore, until a more all-purpose colour space is developed, it is necessary to provide a *number* of colour spaces which can be alternatingly employed. Such a system must provide a consistent interface across the various spaces, so as not to confuse the user in what must done.

Interpolation

Another useful tool in colour selection comes about from the fact that the user may only have a vague notion of the colour to be selected. Quite often, this uncertainty can be expressed in the form of looking for colour A that lies ***between*** colours B and C. In these situations, it is useful to provide an

interpolation mechanism whereby the colours of two ends of an interpolation bar can be specified, and the system automatically fills in intermediate colours (at some appropriate step resolution) between the two specified ones.

Due to the differing geometries among the colour spaces, straight-line interpolation in various colour spaces will not produce identical results. For example, interpolating between pure red and pure green in RGB, will produce a midpoint interpolation value of half red and half green (i.e., fully-saturated, half-intense yellow), whereas interpolating between the same two colours in HSV, will provide a midpoint interpolation value which is half-saturated, fully-valued yellow.

Many applications in Computer Graphics require the computation of interpolation values (e.g., shading, anti-aliasing, and image blending). Where interpolation is basically an additive process, a colour space which simulates additive colour mixing — such as RGB — should be used to interpolate between two colours [FOLE82]. However, Fishkin has shown that the HLS colour space is the only one (of those discussed) appropriate for the simulation of pigment mixing [FISH83].

Unfortunately, as a colour selection tool, it is neither obvious what the intermediate interpolation values should be (except in a limited set of special cases), nor is it clear whether any particular space is always better suited than others. What does seem clear though, is that, in the context of providing a rich set of manipulation tools, it is useful to provide the user with an interpolation mechanism and allow for interpolation in any of the available colour spaces.

Visibility Considerations

A problem which is common to many colour graphics environments concerns the visibility of foreground 'objects' against coloured backgrounds. These objects can be anything from iconic tracking symbols (more commonly referred to as cursors), to text, to an arbitrary shape being dragged around the screen. If the colour of the foreground object is not selected with care, it is easy for there to be an insufficient level of contrast between the foreground and background, causing the object to blend intc the image and, seemingly, disappear.

Consider the case of moving a tracking symbol around on the screen. The image itself may be complex (in terms of its colour composition). Assuming that the primary concern is for the tracker to remain visible at all times (as opposed to being composed of uniform colour), each time the tracker is relocated, some analysis should be performed to modify the colour of each of its pixels so that none of them blend into the background.

A common technique that is used is to one's-complement the bits of the background colours on a pixel-by-pixel basis to obtain the colours for the tracking symbol's pixels. There are two problems with this scheme. First, complementing the bits of a mid-range value results in another mid-range value — not far off from the original! Second, complementing the frame buffer memory values (typically) implies that an RGB colour space underlies the operation. However, since the RGB space contains little information about the contrast of two colours, it is a poor vehicle for visibility tests. For example, although blue and black lie at opposite ends of the blue dimension, they provide little contrast against each other, unless there is significant contrast along the red and green dimensions as well.

We have already seen how to solve the latter problem. That is, we can use a different colour space for computing the tracker's colours. In this case, we are not interested in the intuitiveness or ease-of-use of the colour space since no user interaction is required. What is important is the contrast provided by the colour space. Since luminance plays such an important role in colour discrimination by humans [BOYN79], a colour space which uses luminance as one of its dimensions is more suitable for this task than one which does not.

The former problem can be solved by implementing a simple *distance* algorithm which, for each of the dimensions of the colour space, simply chooses the minimum or maximum value possible in that dimension. The decision of the minimum vs. the maximum is a trivial test of the position of the background colour's value for that

dimension: if it is above the half-way mark, then the minimum is chosen; below the half, the maximum is chosen.† Although this scheme is rather more complicated than merely a complement operation, it is far more reliable in choosing contrasting colours.

It must be pointed out that this scheme does not perform well in all colour spaces. For example, in HSV the optimally visible colour chosen for any of the fully-valued colours (e.g., pure blue) is black, since black lies at the opposite end of the value dimension. However, black on blue is a very poor choice of contrasting colours, as there is little luminosity contrast between them. This demonstrates that the value dimension is *not* perceptually based (though it is intuitively based) and cannot be used for visibility testing.

We can now see a useful property of the ARgYb colour space. Not only does it separate the luminance from the chroma (like in YIQ), but its other two dimensions position complementary colours at the ends of their ranges. Therefore, by selecting minimum or maximum values along each of its dimensions, *a* highly contrasting colour is guaranteed, if not *the* optimal one (which would be based more soundly on colour theory and would be much more expensive to compute).

This scheme has a number of drawbacks, of which the first concerns the amount of computation that needs to be carried out. If the object being displayed is static, like text, then the algorithm does not add much overhead to the rendering requirements. However, since a tracking symbol is constantly repositioned, these calculations have to be performed for each pixel every time the tracker is moved. If no assumptions about the background's colour composition can be made (e.g., a uniform colour in certain regions of the screen), then the computation can be costly.

† Note that dimensions which are circular (e.g., hue in HSV) require the choosing of diametrically opposite values and that implicit constraints must be enforced so that when the colour is transformed back into RGB, a valid triplet is obtained.

Fortunately, many manufacturers of graphics systems now provide hardware for tracking symbols which includes separate colour lookup tables. In these systems, it is a trivial task to precompute the tracking symbol lookup table based on the frame buffer lookup table, using the method described above. Modifications for the tracking symbol need only be made when the frame buffer's lookup table is modified. When the tracking symbol is moved around the screen, the colour changes happen automatically and at refresh rate.

However, for an arbitrary shape which must be moved around the screen, the computation can be prohibitive. Since high contrast is not as critical for the visibility of objects in motion, this technique can be bypassed while the object is being moved, and invoked only when it comes to rest, guaranteeing that it does not then disappear into the background.

A second problem of this scheme is that it does not take into consideration the possible lack of coherence of the background. This problem manifests itself in two ways: if the background is very 'noisy', then the foreground object will also be noisy, and there is the danger of the pixels seeming to be unrelated to each other. However, since there are only eight colours from which the foreground pixels can be selected (the minimum or maximum from each of three dimensions), whereas the background pixels usually contain a much wider selection of colours, coherence of the foreground object is almost always assured. The second way this problem manifests itself is in the fact that we should not be concerning ourselves with the contrast between the foreground object and the pixels which are going to covered up, but rather with the pixels *surrounding* the ones which will be overwritten. Again, we are relying on the coherence of the background by assuming that the pixels in the neighborhood of the ones being covered up are of similar colour composition.

Visibility Testing

Although the scheme described above is quite useful, it limits the variety of colour in foreground objects; for each colour space, there are only eight possible colours from which foreground colours will be chosen.

This may be adequate in systems which, at any rate, provide only a limited colour selection, or in situations where the object visibility is more important than the object colour (e.g., the tracking symbol). However, other applications (e.g., text display), require a more generous selection range. Furthermore, the application may require — as text display often does — that the foreground object be of uniform colour rather than of uniform visibility.

One possible approach is to modify the distance algorithm discussed above to determine a single foreground colour that is 'sufficiently' visible against *all* of the background colours. However, there is no a priori guarantee that such a colour exists, and the algorithm for trying to find one would be very complex. Furthermore, even if one does exist and is found, there is nothing to guarantee that it will appeal to the user.

Therefore, the following approach is much more viable. The distance algorithm is only slightly modified to perform a 'contrast sufficiency' test between any pair of colours. In this test, the algorithm incorporates a similarity criterion analogous to, but simpler than, MacAdam ellipses [WYSC67]. The user specifies a colour for the foreground object, and the system tests it against a set of user-specified background colours. If the foreground colour passes the test for each of the background colours, then the colour is used. Otherwise, the problem colours are reported back to the user with modification suggestions for repeating the test.

Conclusions

A useful set of tools is needed in the user-interface to allow for interactively manipulating user-selectable colours. Furthermore, since no single colour space satisfies all users all of the time, various colour spaces should be provided — with a consistent interface — to ensure flexibility in the selection task. Along these lines, we have designed the ARgYb colour space, based on opponency theory but developed as a simple transformation from RGB in order to be efficient.

An algorithm was given for automatic colour selection of foreground objects in order to guarantee visibility against coloured backgrounds. It was shown that this algorithm provides a sufficiently contrasting colour for the object on a pixel-by-pixel basis. However, more complicated algorithms are needed to provide a uniform colour on a complex background and determine if there are contrast problems in the area surrounding the foreground object.

The following table summarizes the performance of the various colour spaces based on the judgement criteria discussed in this paper:

	Ease of Impl.	Eff. of Impl.	Ease of Use	Visib. Testing
RGB	Great	Great	Poor	Fair
CMY	Great	Great	Poor	Fair
YIQ	Good	Fair	Bad	Good
HSV	Bad	Poor	Good	Poor
HLS	Bad	Poor	Fair	Poor
CNS	Bad	Good	Fair	Bad
Schwarz	Bad	Bad	Good	Great
ARgYb	Good	Fair	Good	Great

References

BERK82a Berk, T., L. Brownston, and A. Kaufman, "A New Colour-Naming System for Graphics Languages," *IEEE Computer Graphics and Applications*, May 1982, pp. 37-44.

BERK82b Berk, T., L. Brownston, and A. Kaufman, "A Human Factors Study of Colour Notation Systems for Computer Graphics," *Communications of the ACM*, Volume 25, Number 8, August 1982, pp. 547-550.

BOYN79 Boynton, R. M., *Human Colour Vision*, Holt, Rinehart, and Winston, New York, 1979.

CATM79 Catmull, E., "A Tutorial on Compensation Tables," *Computer Graphics*, Volume 13, Number 2, August 1979, pp. 1-7. SIGGRAPH 1979 Proceedings.

COWA83 Cowan, W. B., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates," *Computer Graphics*, Volume 17, Number 3, July 1983, pp. 315-321. SIGGRAPH 1983 Proceedings.

COWA84 Cowan, W. B., *Private Communication.*

FISH83 Fishkin, K. P., "Applying Colour Science to Computer Graphics," M. Sc. Thesis, Computer Science Division, University of California, Berkeley, California, 1983.

FOLE82 Foley, J. D. and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., Menlo Park, California, 1982.

JOBL78 Joblove, G. H. and D. Greenberg "Colour Spaces for Compute Graphics," *Computer Graphic* Volume 12, Number 3, August 197[illegible], pp. 20-27. SIGGRAPH 1978 Proceedings.

MEYE80 Meyer, G. W. and D. P. Greenberg, "Perceptual Colour Spaces for Computer Graphics," *Computer Graphics*, Volume 14, Number 3, July 1980, pp. 254-261. SIGGRAPH 1980 Proceedings.

NAIM85 Naiman, A. C., "High-Quality Text for Raster Displays," M. Sc. Thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, 1985.

SCHW84 Schwarz, M. W., J. C. Beatty, W. B. Cowan, and J. F. Gentleman, "Towards an Effective User Interface for Interactive Colour Manipulation," *Graphics Interface 1984*, May 1984, pp. 187-196.

SMIT78 Smith, A. R., "Colour Gamut Transform Pairs," *Computer Graphics*, Volume 12, Number 3, August 1978, pp. 12-19. SIGGRAPH 1978 Proceedings.

SMIT82 Smith, A. R., "Colour Tutorial Notes," SIGGRAPH 1982 Tutorial Notes on Two-Dimensional Computer Animation, August 1982, pp. 71-81.

TANN83 Tanner, P., W. Cowan, and M. Wein, "Colour Selection, Swath Brushes and Memory Architecture for Paint Systems," *Graphics Interface 1983*, May 1983, pp. 171-180.

WYSZ67 Wyszecki, G. and W. S. Stiles, *Colour Science*, Wiley, New York, 1967.