ACE: A Color Expert System for User Interface Design

Barbara J. Meier

Department of Computer Science
Brown University
Box 1910
Providence, RI 02912

Abstract

Color is used in computer graphics to code information, to call attention to items, to signal a user, and to enhance display aesthetics, but using color effectively and tastefully is often beyond the abilities of application programmers because the study of color crosses many disciplines, and many aspects, such as human color vision, are not completely understood. We compiled a comprehensive set of guidelines for the proper use of color, but even these guidelines cannot provide all of the aesthetic and human factors knowledge necessary for making good color selections. Furthermore, programmers may misinterpret or ignore the guidelines. To alleviate some of these problems, we have implemented ACE, A Color Expert system which embodies the color rules and applies them to user interface design. The goal of the implementation was to test whether an automated mechanism would be a viable solution to the problem of choosing effective and tasteful colors.

Our implementation is written in OPS5, a production system programming language, which allowed us encode rules in a similar fashion to our existing set of guidelines. ACE takes a user interface specification and uses our color rules as constraints to determine the best colors for particular items. While ACE is only a prototype, we learned that an expert system is a viable method for choosing an initial set of colors that can be "tweaked" by a human expert. We also learned that much more research needs to be performed in the areas of visual color relationships and how they can be used to provide the most effective user interface.

1. Introduction

1.1. Why color is used

Color is used in computer graphics for a variety of reasons ranging from the aesthetic to the utilitarian. Color can be used to group similar things, to distinguish dissimilar things, to show temporal or magnitude differences, and to label items. In a user interface, color can be used to call attention to an item, to signal the user (as in green for acceptable, yellow for caution, red for error or stop), and to show logical relations between parts of the user interface. Color can also be used to influence, to convey a mood, and to enhance recall. Color helps with perceptual organization as in realistic imagery, and in trying to visualize multidimensional data sets [Marcus, 1982].

A review of the experimental literature on the effects of color coding [Christ, 1975] revealed that color is superior to size, brightness, and shape in searching for and identifying items that vary in only one aspect (e.g., only in color or in size), but that color cannot be identified as accurately as text. This review also showed that for some specific tasks, people remember color longer than size, orientation, or shape. In another study, Tullis [1981] found that although subjects performed tasks equally well with black-and-white graphics as with color graphics, when subjects were surveyed about their preferences in formats, most selected color graphics as their first choice, citing aesthetic advantages over black-and-white formats. Thus, even if color doesn't enhance performance, it may have an effect on user acceptance of a system. Both Christ and Tullis found that subjects believe they perform better when using color; they find color to be less monotonous and believe it causes less fatigue and eyestrain.

1.2. The problem of designing with color

Unfortunately, user interface designers and implementors typically do not have expertise in the science, theory, art, or pragmatics of color use. Colors are often selected for user interfaces in an ad hoc fashion, without considering their physical or psychological effects, and without taking into account design principles concerning legibility and readability. Programs that provide color selection capabilities typically allow the user to select colors freely without regard to their application; they provide no constraints, guidelines, or "templates" to help ensure that the image will make intelligent use of color. Likewise, programmers usually do not have available to them good tools for selecting colors for an image or application program. They use

primary or secondary colors or even use random ones without regard to the effect the colors will have on viewers.

Color improperly used can be worse than no color at all. It can cause confusion or eyestrain; logical relationships can be implied where they don't exist or a viewer may perceive unintended depth differences. If redundancy coding for color deficiency is not considered in a design, some color-deficient users may have difficulties using an interface or perceiving an image. With detailed images or text, incorrectly chosen colors may detract from legibility [Marcus, 1982].

1.3. Why color is difficult

The study of color crosses multiple disciplines: physics, physiology, psychology, art, and graphic design. Each field employs its own terminology; in some cases, a particular word will have one meaning in one field and a different meaning in another field. Since human color vision is not well understood, much of the study of color is theoretical and experimental. As mentioned above, researchers have performed some studies to learn more about how we perceive and use color and have developed theories for explaining color perception, but no comprehensive model of vision exists and some perceptual phenomena have yet to be understood [Boynton, 1979].

Researchers in all of the disciplines in which color is studied agree that a color can be described by a set of three independent parameters; however, each field uses a different set. Some are physical measurements while others are perceptual measurements. To add to the confusion, some are based on reflected color while others are based on emitted color. Colors can be specified using any of these parametric models; although, some models are clearly more appropriate than others for given applications. Computer graphics programmers and users may be familiar with several color systems, but the transformations between them are not always trivial.

More to the point, however, color models contain no information about the effective use of color; they are only a framework on which to build this information, which is based on theory, formal experiments, experience (i.e., informal experiments), and aesthetic judgment. There are no established algorithms that can be applied to choosing colors, only heuristics and rules of thumb.

2. Guidelines to the use of color

Researchers in different disciplines have published information about the effective use of color in computer displays to try to bring together the existing, but scattered color rules [Davis and Swezey, 1983], [Frome, 1984], [Heath and Flavell, 1985], [Krebs, Wolf, and Sandvig, 1978], [Marcus, 1982, 1986], [Murch, 1984a, 1984b, 1984c], [Murch and Taylor, 1986], [Oborne, 1985], [Robertson, 1976, 1980, 1981, 1982], [Shneiderman, 1987], [Teichner, Christ, and Corso, 1977], [Truckenbrod, 1981]. Many of the guidelines, heuristics, and rules of thumb discussed in the these sources have been compiled, edited, and synthesized into a set of prescriptive rules [Meier, 1987]. The need for a compilation arose because much of the published information consists of descriptions of experimental studies in which a particular use of color has been tested, but programmers and designers have difficulty generalizing from the specific results of these experiments. Prior to our compilation, existing lists of rules presented a few important rules appropriate for programmers who occasionally must make a few color selections, but these lists were not adequate for the user interface designer. Moreover, many of the existing rules suggested using or not using a particular color for a specific application; they did not suggest general strategies and design guidelines for selecting colors. Our compilation is focused on color in the user interface and provides both strategic and tactical rules for design synthesis.

3. How an expert system can alleviate the problems of written guidelines

A set of guidelines such as ours can provide readers with many of the rules and strategies for proper use of color. However, because there are so many complex relationships, special cases to general rules, and seeming contradictions, the guidelines cannot capture all the subtleties of the subject. Furthermore, the rules may be misinterpreted, misapplied, or not applied at all. An expert system that helps programmers and users select colors could alleviate many of these problems. ACE, A Color Expert, is a research implementation of an expert system that selects colors for user interfaces.

Three criteria determine the suitability of an expert system for solving a class of problems [Prerau, 1985]. First, the domain should be fairly narrow. The problems presented to ACE are specific enough so that the knowledge needed to solve them can be encoded in a reasonably-sized program.

Second, tasks that are suitable for an expert system should require the use of heuristics and strategies based on the experience of the expert, rather than well-developed models and algorithms. As mentioned above, since a comprehensive model of human color vision does not exist, experts cannot predict exactly how users will react to color; they must rely on experimental results and their own experience and aesthetic judgment when solving color problems.

Third, appropriate tasks for expert systems require expertise, can be solved by existing experts, and are performed better by experts than by amateurs. Expert user interface designers do solve color selection problems, but there are few experts in the field because the knowledge is scattered across many disciplines and the use of color for computer displays is largely an unexplored area of study. Moreover, the importance of using color properly is often underestimated by managers that appropriate resources for user interface design and implementation. Many of the expert user interface designers that are making color decisions are basing their selections on their expertise in another discipline such as graphic design for print media. Much more information and experimentation is needed in order to evaluate current solutions.

4. Previous automatic display design and color design work

In several experimental projects researchers implemented systems that encoded design rules and used this knowledge to synthesize some part of a computer display. Several automated design systems work in the domain of charts, tables, and graphs. Bharat [Gnanamgari, 1981] is a system that chooses the most appropriate chart style, such as bar, pie, or line, and its attributes to display tabular data. Beach [1985] automated the layout of tables that are provided by the user. Unlike Bharat, Beach's system does not work on the semantic level, but instead it uses a

design database to choose a low-level typographical style for a table. APT, A Presentation Tool, developed by Mackinlay [1986], is another system that designs and renders graphical presentations such as bar charts, scatter plots, and connected graphs.

None of these systems have the ability to make decisions about the effective use of particular colors, with the exception of the limited abilities of Bharat. A system that does select particular colors for coding based on constraints was developed by De Corte [1986]. De Corte's algorithm produced a set of colors in which each color is perceived to be different from all other members of the set. The algorithm used minimum between-color distance formulae (based on the 1976 CIELUV color space) and some constraints based on human factors studies for improving user performance in color coding situations. De Corte started with random colors and maximized the minimum color distance by iteratively relaxing the maximum distance constraints, while insuring that the colors lay within the human factors constraints.

A knowledge-based system for solving coloring problems in cartography, by Samson and Poiker [1985], is the most similar project to ACE. Color rules (e.g., "use desaturated color for large areas and saturated color for small areas") and map-coloring rules (e.g., "use blue for water") are used in conjunction with a table that encodes color relationships derived from the Goethe color chart to decide what colors to assign to regions of a map. ACE uses some of these same ideas, particularly the use of a table to encode relations between colors, in its knowledge base.

5. Goals of ACE

The long range goal of this project was to have an implementation of ACE that chooses appropriate, effective, and tasteful colors for user interfaces. The selected colors should be suited for both the task being performed and the output medium; they should make good use of human visual abilities; and, at the same time, the colors should be aesthetically pleasing.

Expert systems typically take many years to design, implement, evaluate, and refine; therefore, we also had more immediate goals for this project. We wanted to determine if automating the color selection process were possible by implementing a prototype, and thus determine if the existing guidelines from the literature were complete enough to enable the automatic synthesis of designs, and whether an expert system were an effective medium for encoding and automating the rules. We also hoped to learn which areas of color selection and user interface design need more study and experimentation.

6. Domain: types of user interfaces for which ACE can select colors

ACE's knowledge base encodes information about particular objects in user interfaces. Most of these objects, which are characterized by their function, are standard items found in the Xerox STAR and Apple Macintosh desktop environments. They include the desktop or screen background, windows, icons, dialog boxes, cursors, and menus. Some of these objects have other objects associated with them such as backgrounds, borders, text, symbols and other details, and highlights. Objects about which ACE has information are shown in Figure 1.

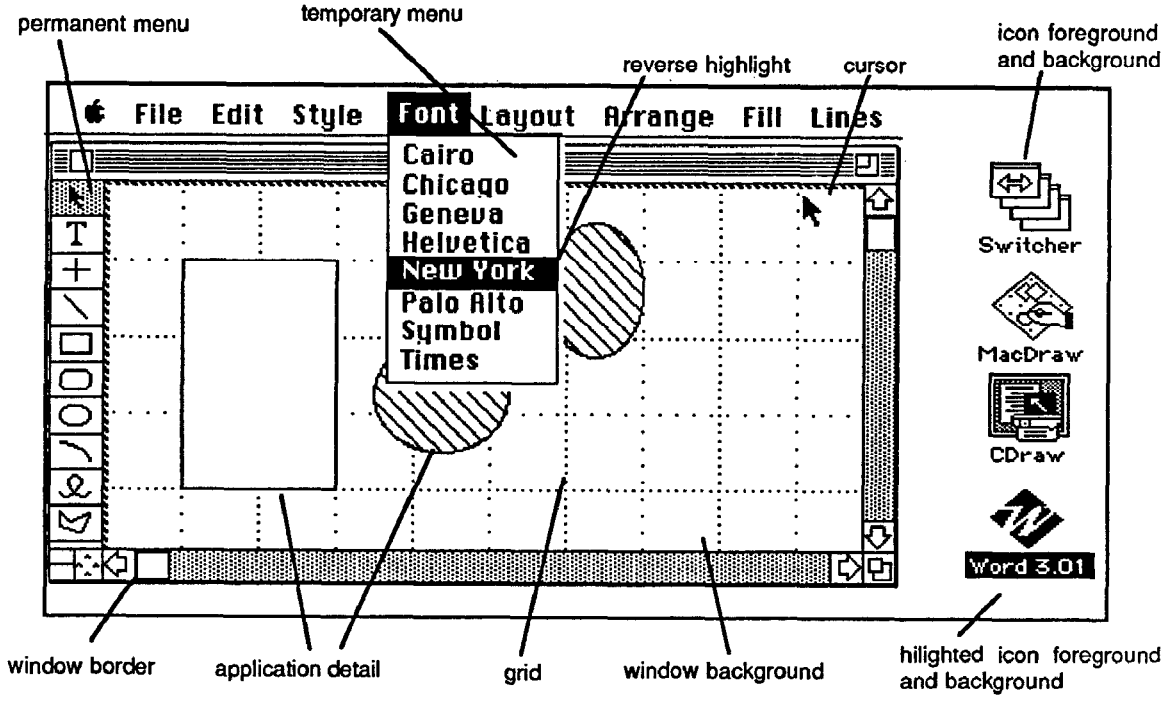

Figure 1: User interface objects that ACE understands

If a user interface includes a non-standard item, the user can select properties that describe the item's physical and functional characteristics, including size, shape, and how it is used. Usually ACE is better at selecting colors for standard items than for non-standard items, because it includes specific rules based on the functions of standard items.

7. User interface to ACE

The first interface to ACE is textual and consists of ACE asking questions about user interface objects and their interrelationships and users selecting answers from menus. After an object's description has been entered, ACE asks the user to indicate how the new object physically relates to all of the previously entered objects. The possible physical relations are next to, on, behind, on screen at same time, or not on screen at same time.

In the future, we would like to have a graphical interface in which there are pictures of the standard user interface objects, similar to Figure 1. Users would be able to point to the objects they wanted and could drag objects to different positions on the screen to show their relationships. Ideally, ACE should get its specification of the user interface directly from a user interface design and synthesis system, relieving users from specifying the interface more than once. As well, this type of interface would prevent users from misinterpreting the functions of objects. For example, a user could mistakenly describe the icon of a window as a window with a border and text instead of as an icon. Since ACE uses different rules for windows and icons, the result might not be appropriate.

8. Expert systems and production system programming languages

8.1. Expert systems

An expert system consists of a knowledge base of rules and an inference-making capability that can put the rules together to draw conclusions. Rules are conditional statements that may perform an action or draw a conclusion. Using a rule-based system for prototyping an expert system allows the programmer to incorporate knowledge in a format that is very similar to that expressed by domain experts. One of our goals in implementing ACE was to understand more about the problem itself. If the implementation mirrors the knowledge base, it can be built, modified, and tailored incrementally as we accrue rules and guidelines. By taking advantage of incremental implementation, prototypes of rule-based systems can be written very quickly. Unlike programs written in procedural programming languages, rule-based systems do not typically require lots of flow-of-control constructs and support routines that must be coded before the system can be tested. A rule-based system can be tested after only a few rules are coded; more rules can be added to improve the results. This is important when the exact rules that will produce the desired results are unknown.

8.2. OPS5 programming language

Our implementation of ACE uses OPS5, a production system programming language developed at Carnegie-Mellon University [Forgy, 1981; Brownston et al., 1985]. A production system consists of a set of productions, which are simply if-then rules, and a global data base called the working memory. The flow of control in a production system is not sequential as in conventional programming languages. Instead, a program executes in cycles that consist of determining which productions may be fired (executed), selecting one (based on a strategy inherent in the language), and firing the production. This is called the recognize-act cycle.

8.2.1. Production system implementations and strategies

In general, production systems which use the recognize-act cycle, such as OPS5, are better suited for synthesis problems, which have many possible solutions, than for diagnosis problems, which usually have one acceptable solution. ACE is a synthesis problem – there are many ways to color a particular user interface – so it uses a "data-driven," forward-chaining strategy instead of the "goal-driven," backward-chaining strategy used for diagnosis problems. A forward-chaining architecture must rely heavily on the heuristic information encoded in its rules to proceed toward the solution, thus the rules often contain control or strategic information.

9. ACE's knowledge base

ACE's rule base contains three types of knowledge. One type is knowledge about user interface components and their interrelationships while another type is knowledge about relations between colors. The third type is control knowledge, the rules that determine how the first two types of knowledge will be used.

9.1. User interface knowledge

These rules constrain the colors of user interface components based on their *functions* and their *relations* to one another. An example of a *relational constraint* between user interface components is that text in a window should have a color that contrasts in brightness with the window color. A *functional constraint* would be that the color selected for text should not be pure blue, because human eyes cannot focus on small blue objects. Another example is that windows and other areas whose function is that of "paper" are generally given a light color, while text and graphics that appear on such paper areas are given darker colors.

Some of the rules in this part of the knowledge base, such as the "paper" rule, are taken from the literature on human factors and user interface design. Unfortunately, most of this literature contains only general principles and very few concrete and specific recommendations and these are often what *not* to do instead of what to do. Thus, most of ACE's rules about user interface design are based on our extensions of rules from the literature, our experience, and our aesthetic judgment. For example, we recommend that the screen background or desktop be a dark color, and that menus be given a color that has a brightness between that of the desktop and that of the window backgrounds. These suggestions are based on the principles that items that will appear on top of each other should have different brightnesses and that background items should have darker colors than foreground items. Thus, there are many ways to apply the rules from the literature; the rules in ACE's knowledge base are not the only way to apply them, but we had to make some decisions in order to use them at all.

9.2. Color knowledge

The second type of knowledge in ACE is information about particular colors and their interrelationships. In a later section, we will discuss how ACE uses constraints on these relationships to select colors, just as a CAD program might use geometric relationships (e.g., distance or parallelism) to constrain positions of geometric primitives. Many color theorists have devised rules for color harmony and contrast, particularly for pure hues. These rules usually state that adjacent colors on a color wheel harmonize while opposite colors contrast. Harmony/contrast rules are helpful – in fact Samson and Poiker's map-coloring system [Samson and Poiker, 1985] is based on them – but they do not state which color combinations are the most attractive or what happens to color relationships when the colors have different brightnesses and saturations. We tried to discover a general relation between any two colors in a three-dimensional color space that would show whether the two colors harmonized or contrasted and how attractive they appeared together. This involved extending the one-dimensional (hue only) harmony/contrast relation of the color wheel to three dimensions and adding attractiveness relations as well. Unfortunately, we were unable to find any general relations, so we selected a discrete set of three-dimensional colors and explicitly tabulated the relations between them based on aesthetic judgment. These relations influence greatly the colors that ACE selects for a user interface design and may be tuned to produce better results without interfering with the user interface rules and control rules.

An earlier implementation of ACE used the HSV (hue, saturation, and value) color space [Smith, 1978] and for each color decision, it attempted to find a subset of the HSV space by eliminating inappropriate colors according to a few user interface and color rules. The final color was chosen randomly from the HSV subset. This scheme did not work because the subset was always too large; typically, it was about half of HSV space, and therefore, the final color selections were almost entirely random.

The current implementation narrows the set of possibilities from infinite to a discrete set of colors from the HSL (hue, saturation, and lightness) color space. [Note: The correct term for the third parameter, and the one that we will use, is brightness, not lightness, which usually refers to reflected light.] In particular, the set consists of ten perceptually different hues, fifteen brightnesses between black and white for each hue, and three saturations for each hue/brightness combination. This may seem limiting at first glance, but, in fact, it encompasses 450 different colors, 150 of which are shades of grey. User interfaces require perceptually different colors unlike realistic imagery, which requires continuous tone colors. A typical interface might have a dozen different colors, half of which might be white, black, or shades of grey. The hues in our set have mnemonic names: red, orange, yellow, green, cyan, light blue, blue, purple, magenta, and rose. The brightness levels are denoted by integers from one to fifteen, and the saturations have the names saturated, desaturated, and grey. Each particular hue, brightness level, and saturation was selected to be perceptually different from its neighbors and to be aesthetically pleasing.

There are several color relations that are encoded in ACE's knowledge base. Ideally, we would have either a general or an explicit relation between each pair of the 450 colors for each relationship we wanted to know about (e.g., contrast or attractiveness), but, as mentioned above, there is no general relation, and encoding explicit relations would require too much space. We compromised by explicitly encoding the relationships between hues and by relying on general relations for brightnesses and saturations. For example, a general relation about brightness levels is that a low brightness (dark) color always contrasts with a high brightness (pastel) color.

There are four hue relations encoded in ACE's knowledge base. Each relation is between two colors. Two encode harmony/contrast and the other two encode attractiveness. Of the first two, the *adjacent contrast* relation is used if the two colors will be adjacent on the display, and the other, *screen contrast*, is used if the two colors will not be adjacent, but will appear at the same time. Similar to Samson and Poiker's system, the value of the harmony/contrast relations is an integer from one to five: one means harmony and five means contrast. The adjacent contrast value for two colors can also be negative which means that they clash and should not be used next to each together.

Of the attractiveness relations, one is used if the first color will be darker than the second color; the other relation is used if the second color will be darker. The reason for having two attractiveness relations is that some pairs of colors are much more attractive if one of the colors is darker than the other. The attractiveness value is also an integer from one to five: one is unattractive and five is most attractive.

Relations between different brightnesses and between saturations are encoded as well. Brightness contrast is simply the difference between the two brightness levels. Sometimes ACE does not need to know the exact brightness of a color; it only needs an approximation so the fifteen brightness levels are broken into three groups: dark (brightnesses 1-5), bright (6-10), and pastel (11-15).

Relations between colors with different saturations are slightly more complicated. Saturated and desaturated hues are treated alike, but the third category, grey, is handled in a different way which is similar to the way hues are handled. Grey colors in ACE appear as greys that have a slight tint of color; thus there are ten "tints" of grey – one for each hue – in fifteen brightness levels (producing 150 different greys). Fifteen greys might have been adequate, but we found that grey is one of the most common colors in user interfaces and that variations in the color of grey can make a mundane-looking screen attractive and sophisticated. Greys are assumed to be attractive with any hue; however, some tints of grey are more attractive than others with a particular hue. The particular tint of grey that is most attractive with each of the ten basic hues is encoded explicitly. This relation also depends on the approximate brightness of the grey, so for each hue, there is an optimal tint for dark grey, medium grey, and light grey. For example, warm greys are attractive with red when the grey is pastel or light, so yellow is encoded as the most attractive light grey tint for red. This relation is encoded as a set of pairs similar to the hue pairs described earlier. Each of the ten hues is matched with its most attractive tint of grey for each of the three approximate brightnesses so there are thirty pairs.

The color relations and the reasoning behind them will become more clear in the sections that explain how they are used. At this point it should be noted that all of ACE's color relationship knowledge is based on our own interpretation of classic color theory and our own aesthetic judgments. A less-

biased implementation could incorporate the taste and judgments of several designers, but it could also run the risk of having inconsistent and muddled relations, or the color selections might be uninteresting and without style. We would like to experiment with several types of color relations and their values and perhaps allow users to choose the style they prefer.

9.3. Control knowledge

The third type of knowledge in ACE is control knowledge. The other two types of knowledge can be thought of as encoding data while the control knowledge encodes the color selection algorithm. These rules determine how the user interface design constraints will be applied and how the color relationships will be used to choose colors.

Since choosing colors is a visual process, it is very difficult to get designers to articulate their color selection process. Most say that they select a few basic colors and then tweak them until they "look right." ACE's control knowledge is based on an analysis of how we solve user interface coloring problems and on color theory. As with both the user interface knowledge and the color knowledge, this method can be changed or tuned without changing the other two types of knowledge. Since ACE has a forward-chaining architecture and most of its inference-making capability is encoded in the control knowledge, changing the method would be more difficult than changing the other types of knowledge simply because it is more pervasive in the system.

The problem-solving strategy is broken into eight discrete steps which are shown in Figure 2. The OPS5 productions used by each of these steps have no specific ordering, but some steps have substeps. The control knowledge formalizes and provides a strategy for color selection, but it does not specify a rigid algorithmic flow of control.

10. How ACE works

10.1. Overview

The goal is for ACE to assign a color to each object described to it. ACE uses monotonic reasoning, so the overall method is to order the color selection decisions in such a way that none ever has to be changed. The first task that ACE performs upon receiving a user interface specification is to determine what functional and relational constraints will act on each object. Next, it orders the objects for color selection. The order is roughly back to front; objects that are in the background of the display will be assigned colors first and objects that are nearest will be assigned colors last. ACE then enters a loop and selects a color for each object, called the target object. ACE uses the functional constraints that apply to the target object to narrow down the possibilities for candidate colors to form a color pool. Next, each object that has already had a color assigned to it and that imposes a relational constraint on the target object proposes a candidate color from the color pool for the target object. If a perfect candidate cannot be found in the color pool, the relational constraints are relaxed gradually until a candidate color is found.

When all candidates colors have been proposed, each candidate is graded against all of the object colors that have proposed candidates. The grade determines "how far" the candidate is from the grading object's version of an ideal color for the target object. The lower the grade, the closer the candidate is to an ideal color. The grades for each candidate are summed and the candidate with the lowest total grade is assigned to the target object. When all object colors have been selected, ACE prints out the results. The basic steps are shown in Figure 2 and the details of the entire process will be explained in the following sections.

Read user interface specifications
Create functional constraints
Create relational constraints
Order objects for selection
For each object
  Choose possible colors by applying functional constraints
  Propose candidates by applying relational constraints
  Grade candidates
Print results

Figure 2: Basic steps in the color selection process

The idea is that all of the previously selected colors should have some influence on the color of the target object. It would be too hard, however, to evaluate simultaneously the relations between all the members of a large group of colors; therefore, the candidates and previously selected colors are compared pair-wise.

10.2. Constraints

In a previous section, we discussed knowledge about the functions of user interface objects, the relations between objects, and the control knowledge that describes how colors are selected. The following sections will show how the control knowledge uses the functional and relational knowledge to select colors. The set of permissible colors for the target object are constrained by the object's function and by the relations the object's color shares with other objects' colors. These two types of constraints are applied one at a time since they are based on different types of knowledge. A third type of constraint, global constraints, may be added in the future.

10.2.1. Functional constraints

Functional constraints are imposed by the function of an object and act on a single object at a time. When functional constraints are applied, they determine the possibilities from which candidate colors may be chosen. These constraints can never be relaxed or violated – a color that has been ruled out for a particular target object cannot be used as a candidate for it. Functional constraints are imposed on one of the color parameters (hue, saturation, or brightness) and yield the approximate ranges, *dark, bright,* and *pastel,* for brightness, *saturated, desaturated,* and *grey* for saturation, and the general categories, *warm* and *cool* for hue. *Particular hues* may also be specified by functional constraints. For example, the desktop is static, in the background, and large in size; therefore, according to our color rules, it should be be a dark, desaturated color. Another color rule states that passive items, such as backgrounds, should have cool colors, so cool is also imposed as a functional constraint on the desktop color.

Functional constraints	Relational constraints
apply to color parameters (h,s,b) of one object	apply to color relationships between two objects
are never relaxed or violated	can be relaxed
based mostly on user interface knowledge	based mostly on color knowledge although some on user interface knowledge
purpose: to initially narrow down the set of all possible colors	purpose: to provide data for selecting candidates from the narrowed down set of possibilities

Figure 3: Differences between functional and relational constraints.

10.2.2. Relational constraints

Relational constraints, the second type of constraint in ACE, are imposed by the physical and functional relationships between two objects. Unlike functional constraints, relational constraints can be relaxed and, in fact, have to be since a perfect candidate cannot always be found in the color pool. Figure 3 shows the differences between functional and relational constraints.

Each relational constraint is weighted according to how much influence it should have over the final color selection for the target object. When the constraints are applied during candidate proposal, they are summed to form a total weight. The weights are assigned when the constraints are created and are used in the grading process which will be explained in a later section.

The first two types of relational constraints we will describe, *adjacent contrast* and *screen contrast*, are constraints on hue relationships and they describe how much the colors of two objects must contrast. These are values from one to five, one meaning harmony or very little contrast, and five meaning a lot of contrast. An adjacent contrast constraint is created if the two objects will appear next to each other or one atop the other, whereas the screen contrast constraint is created if the two objects will appear on the screen at the same time.

Another relational constraint, *relative brightness*, is a constraint on the brightness relationship between two object colors. It is the most common of the relational constraints, because, in general, brightness is the most important color parameter for distinguishing objects; humans can detect brightness differences much more easily than hue or saturation differences. The relative brightness constraint describes how much brightness contrast should exist between two objects and is given by a number from one to fifteen. A *relative brightness direction* can also be specified which describes which of the two colors should be darker. The direction can also be unknown. In this case, ACE knows that the two objects should contrast in brightness, but cannot yet specify which will be darker than the other.

Relational constraints have to be coordinated with the functional constraints, because, in this implementation, the functional constraints cannot be relaxed. For example, a functional constraint on window backgrounds is that they be pastel. The brightness contrast between window backgrounds and the text that will appear in them must be fairly large, thus the text must be bright or dark in order to contrast with the pastel background. In this case, we can specify the direction of the brightness contrast: the text should contrast with and be darker than the window background.

There are two other types of relational constraints which constrain some of the color parameters of the two objects to be the same. One of them, *hue contrast*, asserts that the brightnesses and saturations of the two objects are the same, but the hue should be different. The other constraint, *same*, asserts that the two objects will have exactly the same color. An example of a situation where this constraint would be created is highlighting schemes where the highlight colors are reversed from the non-highlighted colors.

10.2.3. Global constraints

Global constraints are not yet implemented, but will be described here for completeness. These constraints will be imposed by the way in which the user interface is to be used and will apply to all objects. For example, if the interface will be used for long periods of time, then the colors should be less saturated so that they will not cause eyestrain. Another global constraint is that the colors used for a business application should be subtle while those used for a video game can be bright and saturated.

There are at least two ways that these constraints could be applied. First, they could limit the available colors for all objects. Another way is that they could limit the number of colors in a particular category. For example, they could limit the number of very bright, saturated colors for a conservative business application. Neither of these methods seem satisfactory at this time since almost any color can be appropriate if used properly. We need to look into other ways that these global constraints can be applied.

10.3. Ordering color selection

After the functional and relational constraints have been created for each specified object, but before any color selections are made, ACE orders the selection decisions. The general strategy is to select colors for objects that are in the background first and then work forward to the foreground until all object colors are selected. Each of the types of objects that ACE knows about are placed in one of five partitions where the first partition is for background items and the fifth partition is for foreground items. Some objects are assigned to partitions based on their relations

to other objects. For example, text color is always selected after the background color on which it will appear.

All objects in the first partition are assigned selection order numbers starting with number one; those in the second partition are assigned next, and so on. The order within each partition is random. The reasoning for the ordering scheme is that background items are usually large and pervasive, thus they have more presence and should have the most influence over other color selections. Background items also tend to carry the basic color scheme of the interface while foreground items provide accents. When choosing colors for any application, designers usually select a few basic colors for the fundamental items and then select accent colors for the more temporary or special items. The colors that are selected first have fewer constraints acting on them; therefore, it is easier to satisfy their constraints than it is for colors that are selected later in the process. The result is that the first few colors form an aesthetically-pleasing combination and determine the color scheme for the rest.

10.4. Narrowing down possible colors: applying functional constraints

After the objects have been ordered for color selection, ACE enters a loop that iterates for each object. The object whose color is being selected is called the *target* object. The first thing that ACE does for a new target object is apply the functional constraints to the set of all available colors. This has the effect of ruling out all possibilities that are not specified in a functional constraint and of consolidating all of the functional constraint information in one data structure. The data structure, called the color pool, encodes which hues, brightnesses, and saturations may be used for the final color. Hues are marked explicitly as to whether they may be used, brightnesses are specified by approximate range, dark, bright or pastel, and saturations by their mnemonic names, saturated, desaturated, and grey.

10.5. Proposing candidates: applying relational constraints

Each object whose color has been previously selected and who imposes at least one relational constraint on the target object proposes a candidate color. For example, if ACE is selecting a color for the fourth object, and the first and the third impose a relational constraint on the target object then only these two objects will propose candidates. Only objects that are affected by the target object are able to influence its color.

An object proposing a candidate color for the target object is called a *proposing* object. The candidate colors are based on the proposing object's color and on the relations between the proposing object and the target object as expressed in the relational constraints. The candidate is selected from the color pool, so the functional constraints have already had their effect on the decision. There are two steps in proposing a candidate. The first step is assembling the relational constraint information into one data structure, just as ACE assembled the functional constraints in the color pool. In addition to consolidating the constraint information, the weights of all the constraints are summed to produce a total weight for the candidate. Typically, the candidate of a proposing object that is behind or next to the target object will have the most total weight because it imposes the most constraints and the most important constraints on the target object. The second step in the proposing process is using the consolidated constraints to select a hue, saturation, and brightness for the candidate.

The constraints that are consolidated in one data structure include adjacent contrast, screen contrast, and relative brightness and its associated direction. The information in the color pool, i.e., the brightness range, the saturation, and the hue possibilities, also influences the proposal of a candidate. The last pieces of information are the proposing object's hue, saturation, and brightness. Figure 4 summarizes where these pieces of data come from and which color parameters of the candidate they are used to influence.

Source	Data	Parameters affected
functional constraints imposed on target object	possible hues saturation brightness range	hue saturation brightness, hue
relational constraints between proposing object and target object	adjacent contrast screen contrast relative brightness and direction	hue hue brightness, hue
proposing object's color	hue brightness saturation	hue brightness, hue saturation

Figure 4: Information that influences candidate selection.

10.5.1. Proposing a brightness

The candidate's brightness depends on the brightness range specified in the color pool, the proposing object's brightness, and the relative brightness constraint and its direction. The relative brightness is added (or subtracted, depending on the relative brightness direction) to the proposing object's brightness. This value is the brightness that is suggested by the proposing object. If the suggested value is within the color pool range, then all constraints are satisfied and the suggested value is assigned to the candidate's brightness parameter. If the suggested value is outside the brightness range, then the value that is within the range, but closest to the suggested value is assigned to the brightness parameter because the color pool brightness range cannot be violated. If the direction of the relative brightness is not specified, then ACE tries both and chooses the one that is closest to the one specified in the color pool. Note that the proposing object's suggested value, called the *ideal brightness*, is used later in the grading process.

10.5.2. Proposing a saturation

There are no relational constraints that influence saturation so candidates get this parameter directly from the color pool. If there is no saturation value in the color pool because there were no functional constraints on saturation, then the saturation defaults to saturated. The *ideal saturation* is whatever saturation was found in the color pool.

10.5.3. Proposing a hue

Unless there is a *same* relational constraint, the selection of the candidate's hue is the most complicated of the three color parameters. Several relational constraints, which include screen and adjacent contrast and the direction of the relative brightness, work together to influence the selection of a hue from amongst those available in the color pool. A proposing object may impose only one or two of these constraints.

ACE tries to find a hue relationship pair in the color pool that matches the proposing object's hue, the adjacent and screen contrast values, the relative brightness direction, and that also has the highest attractiveness value. This set of constraints defines the *ideal hue* and will be used later in the grading process. The ideal hue may not exist, but if all of these constraints can be satisfied exactly, then the best match occurs. The second best match occurs if either of the screen or adjacent contrast values in the color pool are off by one from what the proposing object suggests and the attractiveness level is still at the highest level. If a hue that satisfies the first or second best match cannot be found, then ACE tries relaxing the attractiveness constraint.

If the attractiveness level reaches a threshold and ACE still has not found a hue, it will suggest using a grey for the candidate because grey is considered attractive with any hue. ACE uses the grey pairs to determine which tint of grey will be most attractive with the hue of the proposing object. The hue parameter of the candidate is set to the tint of the grey, and the saturation parameter is set to grey.

There are many exceptional cases to the above scenario that occur in the cases of the first target object, a target object on which no relational constraints are imposed, or a target object on which not all of the basic relational constraints are imposed. For the first target object, all three color parameters are selected based only on the functional constraints, i.e., the color pool. The hue is selected randomly from those in the color pool. The brightness is set to the middle value within its approximate range. If no range is specified, then it is set to the middle value. The color for an object on which there are no relational constraints, and therefore no proposing objects, is selected in the same way. After one or two object colors have been selected, this rarely happens.

10.6. Grading candidates

Each of the objects that proposed a candidate assigns a grade to each of the other candidate colors. The reasoning is that each object that imposes any relational constraints on the target object should have some influence on the target object's color. Thus the proposing objects from the last phase are now *grading* objects. All of the grades that a candidate receives are summed; the candidate color with the lowest total grade is assigned to the target object.

The grade is a number that describes the difference between a candidate and the grading object's version of an ideal color for the target object. The lower the number, the closer the candidate is to an ideal color. Each candidate has a weight associated with it which is used to affect the other candidates' grades. Before a grade is added to the candidate's total grade, it is multiplied by the weight of the grading object's candidate. An object with a large weight has the effect of hurting every other candidate's grade while an object with a small weight does not hurt other candidates' grades very much. Thus, the weight is not used to improve one's own grade (by lowering it), but to raise every other candidate's by an amount proportional to the weight.

Note that the grading object's candidate color is not necessarily the same as its version of an ideal color for the target object. The constraints that define the ideal color were probably relaxed before the candidate color was found. The grade expresses the difference between an existing color that satisfies the constraints and the nonexistent ideal color.

There are four basic components to the grade: *brightness difference, adjacent contrast difference, screen contrast difference,* and *attractiveness difference.* The *brightness difference* is the difference between the grading object's ideal brightness and the candidate's brightness. The ideal versions of adjacent and screen contrast are those that the grading object suggested in the proposing phase. ACE finds the real contrast relations between the grading object's hue and the candidate's hue from the hue pair data structure that contain them. The *contrast difference* components of the grade are the differences between the real values and the ideal ones. The last component, *attractiveness difference,* is computed by taking the difference between the highest possible attractiveness value and the real attractiveness value.

If one of the candidate or the grading object's color is grey, a slightly different scheme is used to compute the grade. The brightness difference component is retained, but the contrast differences and attractiveness difference are omitted while a *tint difference* component is added. An ideal tint of grey from the point of view of the non-grey color (either the candidate or the grading object) is determined from the grey pairs. The *tint difference* is the difference between the ideal tint and the real tint for the grey. The difference between tints is the number of steps between them on the color wheel of the ten basic hues. If both the grading object's color and the candidate are grey, then the tint difference is simply the difference between the tints of grey.

After a grade is computed, it is multiplied by the grading object's candidate's weight and added to the candidate's total weight. A candidate does not grade itself since it was already selected to be the closest match to the proposing object's ideal color for the target object.

When all candidates have been graded, then ACE finds the candidate with the minimum grade. The winning candidate color is assigned to the target object. The minimum average grade is determined by dividing the minimum grade by the number of objects that graded each candidate. If this grade is not over a certain threshold, a warning message is printed. The reasoning behind the grading process is that ACE finds the candidate which is the least offensive to the most objects. That the lowest average grade is not over the threshold implies that even the best candidate may still not be very appropriate. In a later section we propose several possible ways to deal with this problem other than just printing a warning message.

10.7. Output of results

The loop described above iterates for each user-specified object in the order that is determined near the beginning of ACE's execution. When every object has been assigned a color, ACE prints the results in a file, cleans up its working memory, and halts execution.

11. Implementation

ACE is written in OPS5 with a few routines in Lisp. Input and display programs are written in C. The OPS5 interpreter is written in Franz Lisp and runs on a DEC Microvax II running Ultrix 1.2. ACE contains 170 OPS5 productions.

12. Preliminary results

Our first prototype of ACE has been in existence for only a short time and has not undergone rigorous testing, but we have run it on some sample user interfaces. This section will present the results of running ACE on a test case.

The test case consists of a desktop, windows with borders and text, icons that can appear on either the desktop or the windows, and a temporary menu. Both the icons and the menu have background and foreground colors. Figure 5 shows the user interface items, their selection order, and the colors that ACE assigned to them. ACE took about nine minutes of real time to find this solution.

Objects in order of color selection	Color (saturation hue brightness)
desktop	desaturated light blue 3 (dark)
window background	desaturated yellow 11 (pastel)
window border	desaturated light blue 5 (dark)
window text	saturated purple 5 (dark)
icon background	grey orange 9 (medium)
temporary menu background	grey yellow 11 (pastel)
icon foreground	grey red 1 (dark)
temporary menu text	grey red 5 (dark)

Figure 5: Results from test case.

An expert user interface designer would probably not select these same colors, but they follow all of the rules that ACE currently knows about. For example, the desktop is a dark, desaturated, cool color. The colors follow ACE's attractiveness rules well. Each of the color pairs that appear adjacent on the display is attractive together. Contrast rules are generally followed; for example, the pale yellow window color contrasts well with the dark purple text. Other foreground and background colors, such as menu background and text and icon background and detail, have lots of brightness contrast. The backgrounds of icons have a middle brightness since they must contrast both with the desktop and with the windows. The only colors that do not provide adequate brightness contrast are the window color and the temporary menu background color. In this example, a menu may appear on top of the window so they should contrast in brightness. At least they contrast in saturation, so they will not totally blend in color.

13. Areas for improvement

The most significant problem with ACE's algorithm is that a candidate that is not appropriate can receive the lowest total grade. Typically, one or two objects should have the most influence over a color and, indeed, their candidates have the largest weights. The other candidates, however, usually all have approximately the same relation to the target object, so they all propose the same or a similar color, but a color that is different from the candidates of objects that should have the most influence. Individually, they have small weights but the result is that their candidate gets the lowest grade because they are all voting for approximately the same thing. Hence, the candidate that eventually wins is wrong, but gets elected by sheer numbers.

Associating weights with the relational constraints was supposed to handle this problem, and it does for small numbers of user interface objects. ACE could increase the weights of the most influential objects as the number of ordinary proposing objects increase. A better way of handling the problem is to not allow objects that have a small number of relational constraints with the target object to propose candidates if more influential objects exist. In this scenario, the most influential objects would fight among themselves without corruption from a lot of ordinary candidates.

Another drawback to the way that ACE selects colors is that both selections and evaluations are performed pair-wise, but the effect of a group of colors is as a whole. We might be able to devise color relations between three, and maybe even four or five colors, but more than this would be extremely difficult. Adding these kinds of relations would require a fairly major overhaul of the candidate proposal and grading phases. Before attempting this, we will see what changes can be made in the existing color knowledge that may help produce better results.

In general, the visual effect of color is hard to encode. Color relationships are very subtle and a small change can make a big difference. A designer might spend several hours tweaking the colors of a user interface. As well, it is difficult to encode effects such as simultaneous contrast in which colors look different depending on their surroundings. There seems to be a limitation on the aesthetic quality of ACE's output; ACE can select reasonably attractive colors, but it probably will not be able to perform the same fine adjustments that a human expert can. We may be able to incorporate some of these fine adjustments, but first we need to examine the ways that different experts solve the problem.

As mentioned in the results section, ACE uses its rules well, but the amount of knowledge we have encoded thus far has been limited. If more knowledge were encoded, ACE might be able to make better color choices. On the other hand, if too many constraints are added then ACE might reproduce the same results for many user interfaces. This could be advantageous if we wanted to develop a style of interface; other styles could be introduced as well. One simple way to introduce different color styles would be to provide different initial sets of colors or to change the attractiveness relations between the colors.

In general, there are many ways that ACE can be improved. Each of the types of knowledge, user interface, color relations, and control, can be tuned and augmented to produce better results.

14. Conclusions

ACE has a long way to go before it is a robust, useful tool, but we have accomplished our immediate goals. From our prototype, we have learned that an expert system is an appropriate way of selecting colors for a user interface, but some tweaking by a human expert will be necessary in order to achieve the

most aesthetically-pleasing colors. Certainly, the colors ACE selects are better than a random selection and also better than what a naive programmer with little aesthetic experience would choose. The process of synthesis is usually considered more difficult than that of analysis. ACE is a useful synthesizer of colors, while human experts can be called in for analysis and successive refinement; thus, ACE provides a user with a head start, a first approximation to build upon.

Another of our goals was to discover areas in which more studies and research need to be performed. One of these areas is visual color relationships. Humans develop senses of harmony/contrast and of attractiveness throughout their lives, and each individual has a different idea of which sets of colors harmonize or contrast or are attractive. This type of information is the hardest to encode and is the reason why expert systems usually take years to write and refine. Ultimately, user interface colors that are produced by ACE should be tested on users.

15. Acknowledgements

Many thanks to David Laidlaw for ideas and suggestions for the ACE algorithm and to Ed Chang, David Laidlaw, and Andy van Dam for critically reading earlier drafts of this paper. Thanks also to Norm Cox, Steve Feiner, Jim Foley, David Laidlaw, Aaron Marcus, Dick Shirley, David Smith, and Andy van Dam for critical feedback on the color rules work on which ACE is based. The color rules work was supported in part by Apple Computer, Inc.

16. References and Bibliography

[Beach, 1985]
Beach, R., Setting tables and illustrations with style, Ph.D. Thesis, Department of Computer Science, University of Waterloo, Ontario, 1985.

[Boynton, 1979]
Boynton, R., *Human Color Vision*, Holt, Rinehart, and Winston, 1979.

[Brownston, et al., 1985]
Brownston, L., R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, MA, 1985.

[Christ, 1975]
Christ, R., Review and analysis of color coding research for visual displays, *Human Factors*, 17, 1975, 542-570.

[Davis and Swezey, 1983]
Davis, E. and R. Swezey, Human factors guidelines in computer graphics: a case study, *International Journal of Man-Machine Studies*, 18, 1983, 113-33.

[De Corte, 1986]
De Corte, W., Finding appropriate colors for color displays, *COLOR Research and Application*, 11:1, Spring, 1986, 56-61.

[Forgy, 1981]
Forgy, C. L. The OPS5 users's manual. Technical Rept. CMU-CS-81-135. Department of Computer Science, Carnegie-Mellon University, 1981.

[Frome, 1984]
Frome, F., Improving color CAD systems for users: some suggestions from human factors studies, *IEEE Design and Test of Computers*, 1:1, February, 1984, 18-27.

[Gnanamgari, 1981]
Gnanamgari, S., Information presentation through default displays, Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1981.

[Heath and Flavell, 1985]
Heath, A., and R. Flavell, Colour coding scales and computer graphics, *Proc. Graphics Interface '85*, June, 1985, 321-328.

[Krebs, Wolf, and Sandvig, 1978]
Krebs, M., J. Wolf, and J. Sandvig, Color display design guide, ONR Report No. ONR-CR213-136-2F, NTIS AD No. A066630, October, 1978.

[Mackinlay, 1986]
Mackinlay, J., Automating the design of graphical presentations of relational information, *ACM Transactions on Graphics*, 5:2, April, 1986, 110.

[Marcus, 1982]
Marcus, A., Color: a tool for computer graphics communication, in *The Computer Image*, Addison-Wesley, Reading, MA, 1982, 76-90.

[Marcus, 1986]
Marcus, A., The ten commandments of color, *Computer Graphics Today*, 3:10, November, 1986, 7.

[Meier, 1987]
Meier, B., Effective use of color in user-computer interface design, Brown University, 1987.

[Murch, 1984a]
Murch, G., Physiological principles for the effective use of color, *IEEE Computer Graphics and Applications*, 4:11, November, 1984, 49-54.

[Murch, 1984b]
Murch, G., The effective use of color: perceptual principles, *TEKniques* 8:1, Spring, 1984, p. 4-9.

[Murch, 1984c]
Murch, G., The effective use of color: cognitive principles, *TEKniques* 8:2, Summer, 1984, p. 25-31.

[Murch and Taylor, 1986]
Murch, G., and J. Taylor, The effective use of color in computer graphics applications, Proc. Computer Graphics '86 Conference, Vol. 3, National Computer Graphics Association, Anaheim, Ca., May, 1986, 515-521.

[Oborne, 1985]
Oborne, D., *Computers at Work: A Behavioural Approach*, John Wiley & Sons Ltd., 1985.

[Prerau, 1985]
Prerau, D., Selection of an appropriate domain for an expert system, *AI Magazine*, 6:2, (Summer 1985), 26-30.

[Robertson, 1976]
Robertson, P.J., The use of colour for computer displays, MS Thesis, University of Aston at Birmingham, U.K., Department of Applied Psychology: published as IBM Hursley Human Factors Laboratory Report No. HF005, IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hampshire, U.K., October, 1976.

[Robertson, 1980]
Robertson, P.J., A guide to using color on alphanumeric displays, Technical Report, IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hampshire, U.K., June, 1980.

[Robertson, 1981]
Robertson, P.J., A survey of users of the IBM 3279 color display station, Technical Report TR.12.193, IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hampshire, U.K., August, 1981.

[Robertson, 1982]
Robertson, P.J., Review of color display benefits, IBM Hursley Human Factors Laboratory Report No. HF056, IBM United Kingdom Laboratories Ltd., Hursley Park, Winchester, Hampshire, U.K., January, 1982.

[Samson and Poiker, 1985]
Samson, L. and T. Poiker, Graphic design with color using a knowledge base, Simon Fraser University, 1985.

[Shneiderman, 1987]
Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction* Addison-Wesley, 1987.

[Smith, 1978]
Smith, A., Color gamut transform pairs. *Computer Graphics* (Proc. SIGGRAPH 78), 12:3, July, 1978, 12-19.

[Teichner, Christ, and Corso, 1977]
Teichner, W., R. Christ, and G. Corso, Color research for visual displays, ONR Report No. ONR-CR213-102-4F, NTIS AD No. A043609, June, 1977.

[Truckenbrod, 1981]
Truckenbrod, J. R., Effective use of color in computer graphics, *Computer Graphics* (Proc. SIGGRAPH 81), 15:3, August, 1981, 83-90.

[Tullis, 1981]
Tullis, T., An evaluation of alphanumeric, graphic, and color information displays, *Human Factors* 23:5, October, 1981, 541-550.