Sharing Image Colors in X

Author:
Robert NC Shelley

February 10, 1989

Abstract

Displaying images (natural or synthetic) may require an enormous number of colors. A 24-bit deep TrueColor workstation can simultaneously display a large enough palette of colors to faithfully render almost any image. When working with depth-limited PseudoColor displays, however, techniques such as judicious color selection or dithering must be employed to reduce the number of unique colors in an image to a workable set. This paper addresses a strategy for allocating and sharing colors and grayscale shades within the limitations of the PseudoColor and GrayScale visual classes of the X Window System. All applications requiring static shared colors or grayscale shades can benefit from this strategy. An abridged example of this algorithm is available in the server display module of the version 3 X Image Extension sample implementation from Digital Equipment Corporation.

The Problem

In the X Window System the palette of colors, displayable at any given instant, is determined by color definitions in the installed colormap. Many X servers only support one active colormap at a time.

The X11 PseudoColor visual class offers a wide gamut of colors to choose from (2^{48}). Most hardware platforms offer a subset of the full X11 gamut, often 2^{24} colors. However, the palette of colors that can be displayed simultaneously may be much more limited. For example, the gamut of over 16 million colors may be limited to a palette of 256 simultaneously displayable colors on an 8 plane system, or as few as 16 colors on a 4 plane system.

Color hungry applications may attempt to hog the default colormap's color cells as private read/write color cells. This strategy tends to starve late comers who find an inadequate number of color cells remaining. It is equally unsociable to use a private colormap. The application owning the colormap controls all the colors it wants, while the windows of all other applications displayed on the same screen go "Technicolor". The advantage in using private color cells or a private colormap is that the application has full control over the choice of displayable colors and may change colormap entries dynamically.

A more sociable strategy involves sharing colors from the default colormap. In many cases applications sharing the default colormap would be willing to compromise their choice of exact colors for colors which are perceived by the human visual system to be "close enough" to the desired color.

The X Window System reference manual states, "XAllocColor returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the RGB values actually used". This sounds like the desired sociable color sharing strategy. However, the meaning of this statement lies in the fact that most servers use only the upper 'n' bits of each of the RGB values (n often equals 8). When allocating the closest color supported by the hardware, the server makes no attempt to "search" for the closest color. The "RGB values actually used" are the values requested, disregarding the lower (16-n) bits.

The Solution

To encourage the sociable use of color resources among applications we have developed color allocation routines which, when given a list of preferred colors, will return colors which are perceptually close enough to those requested, and allocate new sharable colors when no acceptable substitute can be found. Each application defines what constitutes "close enough".

Since the routines simultaneously manage sharing colors and grayscale shades, applications requiring colors can coexist with applications exclusively requiring shades of gray.

What's an application to do?

Prior to calling the color allocation routines, a typical application would create the list of preferred colors. If the colors are being allocated for display of an image this list may be created by generating a histogram of color usage within the image. The color allocation algorithm gives a limited priority to allocating colors according to their order within the list, therefore the list should be arranged in an order that will produce the most pleasing affect for the application (e.g. decreasing frequency of usage) in case the colormap becomes full.

Along with the list of colors, the application supplies parameters which limit the definition of an acceptable color match, and specifies a color space to be used during color match calculations.

The preferred color list is then processed by the color allocation routines which will return the allocated colors. The application would then create its final image by re-mapping each image color to the pixel index of the color returned.

Algorithm overview

The color allocation algorithm gives priority to allocating new sharable colors in a sequence that will provide an even distribution of sharable colors throughout the specified color space. This iterative process begins by dividing the color space into sub-volumes and determining the existence or need for sharable colors within each sub-volume. The sub-volume size is determined dynamically such that the size is reduced on each iteration.

Secondary priority is given to allocating colors according to their order within the list of requested colors. Each iteration of the algorithm processes the preferred color list in order. Sub-volumes of the color space, centered around each requested color, are examined to see if any sharable color exists within the sub-volume. If none is found, the new sharable color is allocated. This process proceeds until the end of the list is reached, at which time a new sub-volume size is calculated.

The color allocation algorithm ends when: all colors have been allocated, or the sub-volume size has reached the limit specified by the application, or a sharable color allocation fails due to the colormap being full.

All remaining colors are assigned the pixel index of their closest match within the specified color space.

Parameters required

The parameters to the color allocation routine include: the screen on which the image will be displayed, the colormap to allocate colors from, a list (and count) of colors to allocate, the color space in which to match colors, and a pair of control parameters.

Color match space

The color allocation routines allow the application to choose one of the following color spaces: HLS, Lab, L*U*V*, RGB, U*V*W*, and YIQ. During color allocation the color space is used for choosing colors that have the greatest distance from any existing color. During color matching the color space is used to find the minimum distance to an existing color.

Control parameters — what's close enough

The cube in figure 1, represents the RGB color space — all possible colors that can be defined using the primary spectral components of visible light: red, green, and blue. All the pure shades of gray lie along a diagonal line running between black and white.

The human visual system is unable to perceive any difference (in hue, saturation or intensity) between closely spaced points within the cube. Thus there is a sphere of points surrounding each color, any of which could be an acceptable substitute for the central color. Furthermore, there is a cylinder of points surrounding the gray shade line, each of which is close enough to its closest gray shade on the line to satisfy the human eye.

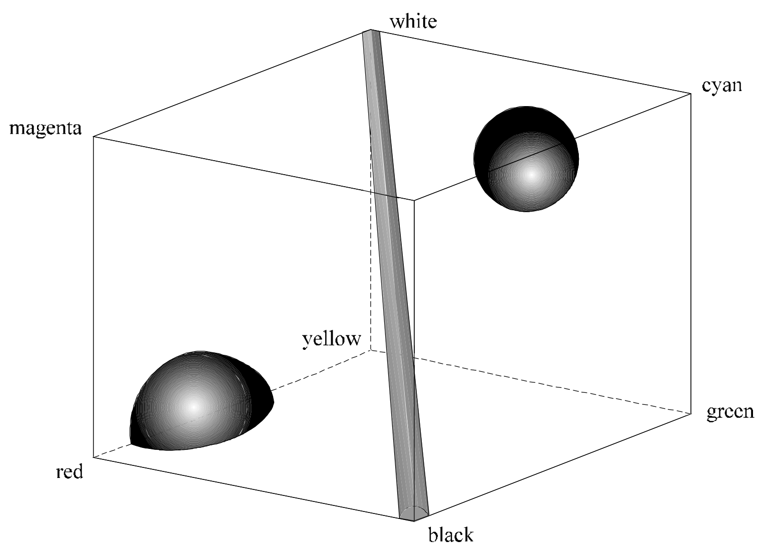

Figure 1: RGB color cube showing color match spheres and gray shade cylinder

The RGB color space was chosen here to simplify illustration. The other color spaces are not cubic in shape. Neither are the volumes representing acceptable colors and gray shades spherical and cylindrical, but rather they are distorted by the shape of the color space.

Match-limit

The radius of the color match spheres (parameter **match_limit**), determines the degree of error to be tolerated when searching for perceptually close colors. The minimum value of **match_limit** allows only exact matches (within the limits of the hardware), while the maximum value encompasses the entire color space (no new colors would be allocated).

Figure 1 illustrates a pair of colors to be allocated: a saturated orange (centered within the quarter sphere) and a blue-green shade (centered within the full sphere). The color allocation routine will locate the color existing within each sphere that has the shortest distance vector to the desired color, or allocate the exact color if no color has been allocated within the sphere.

Gray-limit

Most people will accept a decreased resolution in intensity rather than tolerate inaccuracies in hue when viewing a grayscale image. Therefore a separate parameter (**gray_limit**) is provided in order to define a relatively narrow cylinder of gray shades while tolerating larger intensity errors along the axis of the cylinder (specified using the **match_limit** parameter).

The radius of the gray shade cylinder (parameter **gray_limit**), determines the degree of error to be tolerated when searching for shades of gray. The minimum value of **gray_limit** allows only pure shades of gray to be shared from the colormap, while the maximum value (which must be specified when allocating colors rather than gray shades) includes the entire gray shade line.

Special effects

If the colormap contains a grayscale ramp, a color image can be displayed as grayscale by using a low value for **gray_limit** and the maximum value, for **match_limit**. Likewise, a grayscale image can be displayed with a "digital art" effect, if the colormap contains only a distribution of colors and both **match_limit** and **gray_limit** are specified at their maximum values.

Many happy returns

The color allocation routine modifies each XColor structure in the supplied **colors** list to contain the pixel value of the allocated color, the actual RGB values assigned, and a symbol in the **pad** field that indicates whether the exact color, or the best match was allocated. Or the **pad** field may indicate failure in the event that the supplied colormap was completely allocated with private read/write color cells.

Each color allocated must be freed using XFreeColors. Each pixel index must be freed as many times as it appears in the **colors** list.

A closer look

The following is a summary of the color allocation algorithm:

- discover the state of each colormap cell: available, sharable, private,
- exclude sharable colors which deviate too far from **gray_limit**,
- convert each sharable colormap cell into the color space of choice,
- convert each requested color into the color space of choice,
- find the distance to the closest existing match for each color requested,
- allocate new colors for those having no acceptable match,
- share existing colors that are perceptually close enough,
- if the colormap becomes full, assign the remaining colors to their closest match.

What state is your colormap in?

Each colormap cell is tested to determine which colors are sharable. However, the algorithm makes allowance for the volatility of the colormap resource as other clients also have access to the colormap while the color allocation routine executes.

If the supplied colormap does not allow any sharable colors to be allocated, each of the requested colors are marked as failing to be allocated.

How gray it is

For each sharable color absolute value of the difference between RGB components is computed. If any difference exceeds the application's **gray_limit** parameter the colormap cell is excluded from being a color match candidate.

Finding the best mate

Initially each requested color is assigned the colormap index of the closest existing color; the distance between them is also recorded. The distance between colors is computed as a space diagonal within the color space requested:

$$distance = \sqrt{\left(C1_0 - C1_1\right)^2 + \left(C2_0 - C2_1\right)^2 + \left(C3_0 - C3_1\right)^2}$$

Where C1, C2, and C3 are the three components of the requested color space. Since only relative distances are required in determining the closest match, the square root is not computed (reducing computational overhead).

Private rooms for distant cousins

A new color is allocated for each color that has a match distance exceeding the **match_limit**. Color allocations proceed in groups that will distribute them throughout the color space. Within each group the colors are allocated by their order within the list of requested colors.

The mean distance between all pending allocations is calculated. All colors exceeding this mean distance form a group of colors to be allocated. For each new color allocated the distance to the remaining colors is tested — the new color may be a better match than was previously found. While this process eliminates many colors from the group under consideration, it ensures an equitable distribution of colors through out the color space.

Allocation of new colors is complete when: all colors have been allocated, or all the remaining colors have a close enough match, or the colormap becomes full.

We're room mates — like it or not

Any remaining colors share the allocation of the closest existing color within the colormap. While colors are being allocated other clients may also be allocating and deallocating colors. In case this activity causes the allocation of the closest matching color to fail, the next best match will be allocated instead.