8. Introduction to Color

Color is a difficult but important topic. In this lecture we will discuss some of the basic properties of color, the mechanism used by display devices to model color, and some of the approaches used by operating systems to specify color.

8.1 Properties of Color

Color is light, and the properties of what we percieve to be color are directly related to the properties of light: *wavelength, purity* and *amplitude*. In terms of color, these characteristics of light describe *hue, saturation* and *brightness*, respectively. As seen in **Figure 8-1**, brightening a

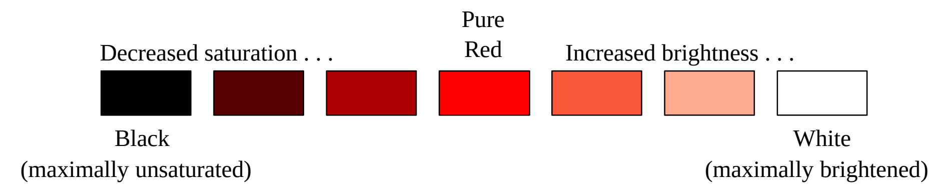

Figure 8-1 Hue, Saturation and Brightness

hue lightens the color; a hue at miximum brightness is white. Reducing the saturation of a hue, or *unsaturating* the hue, darkens the color; a maximally unsaturated hue is black.

8.2 Primary Colors

A set of *primary colors* consists of two or more colors that are mixed together to obtain other colors. The range of colors that can be produced by mixing colors in the set is known as the *gamut* for the set. Having many colors in a set of primaries broadens the set's gamut, but no finite set of primaries can produce all possible colors.

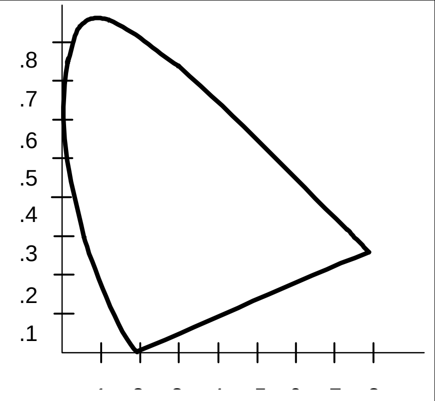

Figure 8-2 The CIE Chromaticity Diagram

In 1931, the International Commission on Illumination developed an idealized model that plots all visible colors in three dimensions. By normalizing the model so that all colors with the same brightness map to the same pont, the two-dimensional graph shown in **Figure 8-2**[2], known as the *CIE Chromaticity Diagram*, is obtained. To discover the subset of colors representing the gamut for some set of primaries, locate each primary color within the diagram, then join them to enclose the largest possible area; the area then represents the primaries' gamut.

[2] Please note that this figure was sketched by hand, and is likely inaccurate. For an accurate representation of the figure, please consult your text book.

8.3 The RGB Color Model

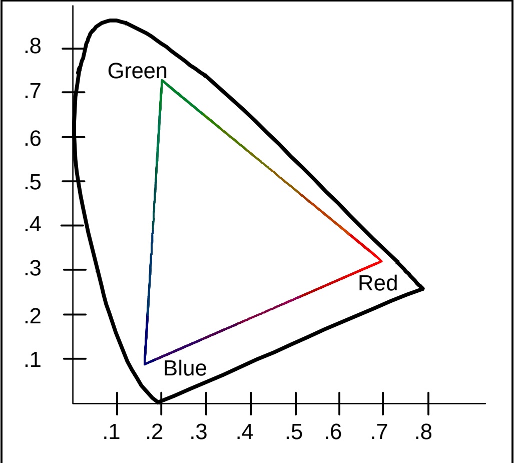

Figure 8-3 The RGB Gamut

Most luminescent graphics devices (for example, your monitor) display colors by dynamically mixing them from a set of primaries; the primaries of choice are most often *red*, *green* and *blue*. The colors that can be formed from this set of primaries, along with the method for mixing colors, is known as the *RGB Color Model*. The gamut for the model, as plotted within the CIE Chromaticity diagram, is shown in **Figure 8-3**.

The RGB Color Model is *additive*, that is, red green and blue frequencies are added together to produce additional colors; red plus blue produces magenta, red plus green produces yellow, and so forth. The full RGB gamut is usually visualized as a cube plotted in three-dimensional cartesian coordinates, with red on the X axis, green on the Y axis and blue on the Z axis. As seen in **Figure 8-4**, the corners of the cube are occupied by red, green and blue; the principle additive colors, cyan, magenta and yellow; plus white, obtained by adding all three colors together, and black, the absence of any of the colors.

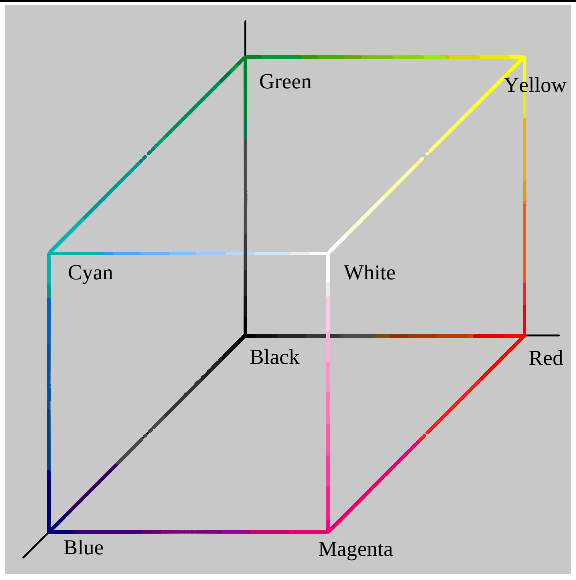

Figure 8-4 The RGB Color Cube

An RGB color can now be specified as a cartesian coordinate triple representing a point inside the color cube. If we choose the length of a side of the cube to be 1, red would be (1, 0, 0), cyan would be (0, 1, 1), and dark magenta would be (.5, 0, .5). Gray values occupy the line, known as the *gray scale*, connecting the black and white corners of the cube, so that red, green and blue values are all equal; 60% gray (Gray-60) would then be (.4, .4, .4), and 35% gray would be (.65, .65, .65).

The RGB model in which red, green and blue intensities are determined by a floating point value between 0 and 1 is known as the *idealized* model. However, since computer operations are usually more efficient when conducted in integers, there are two other common models; in the *scaled 255* model, a side of the color cube is 255, and RGB intensities are encoded as integer values between 0 and 255; in the *scaled 65535* model, a side of the cube is 65535, and intensities are encoded as integer values between 0 and 65535. A few examples of RGB values in all three models are shown in **Figure 8-5**.

		Idealized	Scaled 255	Scaled 65535
⬜	White	(1, 1, 1)	(255, 255, 255)	(65535, 65535, 65535)
⬛	Black	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
🟩	Green	(0, 1, 0)	(0, 255, 0)	(0, 65535, 0)
🟨	Yellow	(1, 1, 0)	(255, 255, 0)	(65535, 65535, 0)
🟪	Magenta	(1, 0, 1)	(255, 0, 255)	(65535, 0, 65535)
▩	Gray-50	(.50, .50, .50)	(127, 127, 127)	(32767, 32767, 32767)
▩	Gray-75	(.25, .25, .25)	(63, 63, 63)	(16383, 16383, 16383)
🟧	DarkOrange	(.97, .50, .09)	(248, 128, 23)	(63568, 32767, 5898)

Figure 8-5 RGB Values

8.4 Color Displays

Frame Buffers

Color displays store images in a *frame buffer.* The frame buffer is a block of memory associated with your graphics hardware, usually resident on your video card.

255	16	123	255	49
12	96	255	127	33
14	97	129	231	1
66	241	201	111	135
14	56	56	201	135

Figure 8-6 Frame Buffers

As shown in **Figure 8-6**, a frame buffer can be visualized as a two dimensional array of integers. Each integer corresponds to a pixel on the associated display, and the value of the integer encodes an RGB value that determines the color of the pixel. The range of colors that can be displayed by a pixel is determined by the range of values that can be stored in a memory location in the frame buffer; this in turn is determined by the *depth* of the frame buffer; that is, the number of bits assigned to each memory location. Monochrome displays have a depth of one, restricting them to displaying images in black or white. Color displays have a minimum depth of 3, allowing them to display up to 8 colors; *true color* displays have a minimum depth of 24, and can display up to 16 million different colors.

How an integer value in a frame buffer maps to a color is dependent on the specific type of graphics hardware. Consider a theoretical color display, with a depth of three. Each bit in an integer will control one of the three electron guns in the monitor; when a bit is turned on, the associated gun fires at full intensity, and when the bit is off, the gun doesn't fire at all. If *red* is 4, *green* is 2 and *blue* is 1, then 5 corresponds to magenta. The original CGA (for *color graphics adaptor*) devices have a depth of four; bits 4/2/1 correspond to red/green/blue, respectively, and bit 8 represents a *multiplier* or *intensity* bit. When a red, green or blue bit is on, and the intensity bit is off, the corresponding electron gun fires at half intensity; when the intensity bit is on, the gun associated with any other bit that is also on fires at full intensity. In a CGA device, a value of 3 would be dark cyan, and 12 (8 plus 4) would be light red. EGA/VGA devices have a depth of 6, and assign two bits to each of the electron guns, allowing red, green and blue component to have a value of *off* (0), *light intensity* (1), *medium intensity* (2) or *high intensity* (3). **Figure 8-7** demonstrates the range of physical RGB values for CGA devices, and our theoretical depth 3 device.

	R G B	Color
⬛	0 0 0	Black
🟥	1 0 0	Red
🟩	0 1 0	Green
🟦	0 0 1	Blue
🟪	1 0 1	Magenta
🟨	1 1 0	Yellow
🟦	0 1 1	Cyan
⬜	1 1 1	White

Depth 3 Device

	i R G B	Color
⬛	0 0 0 0	Black
🟥	0 1 0 0	Dark Red
🟩	0 0 1 0	Dark Green
🟦	0 0 0 1	Dark Blue
🟪	0 1 0 1	Dark Magenta
🟫	0 1 1 0	Brown
🟦	0 0 1 1	Dark Cyan
⬜	0 1 1 1	Light Gray
⬛	1 0 0 0	Dark Gray
🟥	1 1 0 0	Light Red
🟩	1 0 1 0	Light Green
🟦	1 0 0 1	Light Blue
🟪	1 1 0 1	Light Magenta
🟨	1 1 1 0	Yellow
🟦	1 0 1 1	Light Cyan
⬜	1 1 1 1	White

CGA (Depth 4) Device

Figure 8-7 Devices and RGB Values

Color Maps

The successor to VGA, Super VGA or SVGA, extended the depth of the frame buffer to eight, allowing up to 256 different colors to be assembled simultaneously. It also added an element of flexibility to color specification by taking advantage of a *color map* (also called a *color lookup table* or *palette*).

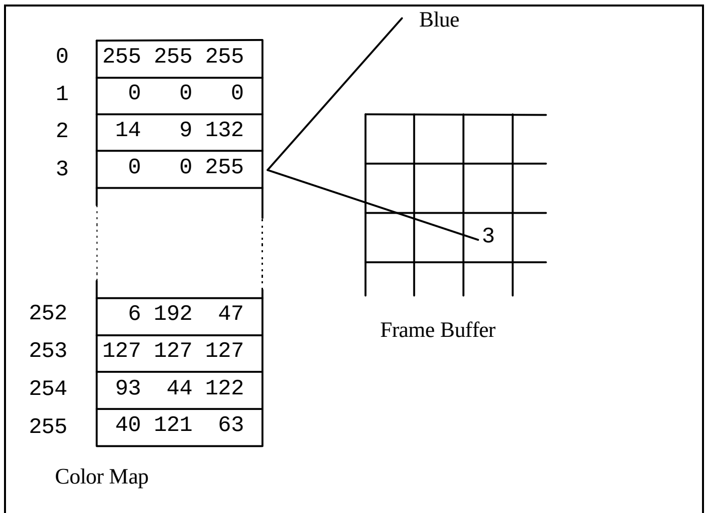

Figure 8-8 Color Maps

As seen in **Figure 8-8**, a color map is an array of RGB triples maintained by an operating system in "normal" memory (i.e. not on the video card). For a frame buffer of depth 8 the array can contain up to 256 colors, but may be shorter depending on operating system constraints; for frame buffers of greater depth the array can be longer. The RGB values contained in the array are usually stored as scaled 255 or scaled 65535 values. An integer stored in the frame buffer now becomes an index into the color map, and changing the color map will change the colors displayed on the associated monitor. Some systems allow applications to have private color maps, but in most systems only one color map can be active at once; for this reason, most systems provide a default color map, which is shared by all applications.

Display Modes

Since the capabilities of a display device are largely a function of the amount of memory contained in the frame buffer, many such devices provide different operating *modes* in which they trade resolution for color. For example, a device with 1.4 megabytes of video memory can provide true color (depth 24) at a resolution of 800 x 600, but only 256 colors (depth 8) at a resolution of 1024 x 1024.

Color Planes

So far we have been discussing frame buffers as though they were (at least virtually) two dimensional in nature, but, as suggested by **Figure 8-9** they are actually three dimensional. For a

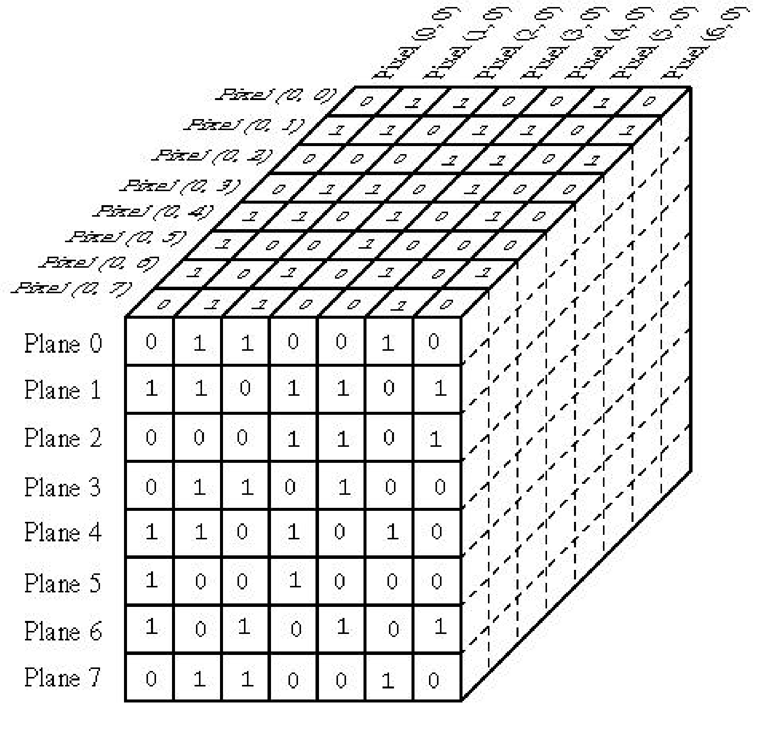

Figure 8-9 Color Planes

particular location in the frame buffer, imagine that the individual bits assigned to the location are lined up along the Z-axis of a cube (if you think about it, this explains why the number of bits assigned to a location in the frame buffer is referred to as the frame buffer's depth). Now imagine a slice taken out of the cube consisting of bit number 0 for every frame buffer location; this would be the frame buffer's zeroth *color plane*.

A frame buffer has as many planes as it has depth, and some video systems are capable of addressing individual planes in the frame buffer. For example, with the right hardware you could store, say, 8 different monochrome images in eight different planes of the frame buffer, and instruct your system to display images from a single plane at a time; then by rapidly switches planes you could perform a sort of animation.

The most common use of color planes is made in systems employing so-called twelve plane display devices. In these systems, the planes of the device are divided into one set of four planes, and one set of eight, and the system assigns a different color map to each. Typically, the four plane color map is declared public, and used for things like window decorations, and the twelve plane color map becomes the private domain of a large graphics application; then the graphics application can change its color map in any way that best suits its purpose without affecting peripheral applications using the public color map.

Dithering

Often a system will attempt to display more colors than the hardware actually supports. This is

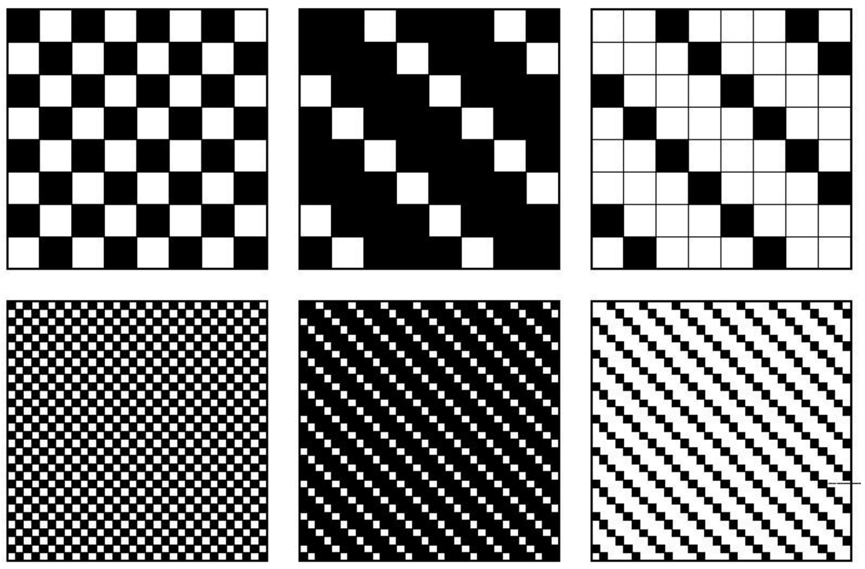

Figure 8-10 Dithering

done via the process of*dithering*, which combines different colors in a small area to simulate the effect of producing a third color.

8.5 Assembling Colors

DOS/CGA	i	R	G	B	
Brown	0	1	1	0	(0x6)
Light Blue	1	0	0	1	(0x9)

DOS/EGA	r	g	b	R	G	B	
Light Red	1	1	1	1	0	0	(0x3C)
Light Magenta	1	1	1	1	0	1	(0x3D)

Windows	m	G	B	R
Light Red	0x00	00	00	00
Dark Red	0x00	00	00	77
Light Magenta	0x00	FF	00	FF

Figure 8-11 RGB Values in DOS and Windows

In order to be able to specify a color in some system, you have to know how to assemble an RGB value in that system. In DOS first you have to know what kind of hardware you are addressing, and how a color is represented using that hardware; brown in CGA, for example, is 0x6 (red and green on at half intensity) and light magenta in EGA is 0x3D (red and blue on at full intensity).

MS Windows uses a more flexible and portable approach. RGB values are encoded as scaled 255 values in the low order three bytes of a long integer, so that, to the programmer, 0x00FF00FF is light magenta no matter what type of hardware you have; the system takes responsibility for converting the value to a device specific representation when necessary.

In the X Window System, RGB values are encoded as scaled 65535 values in three fields of an XColor structure.

```
XColor brown,
       magenta;

brown.red = 32768;
brown.green = 32768;
brown.blue = 0;

magenta.red = 65535
magenta.green = 0;
magenta.blue = 65535;
```

Figure 8-12 RGB Values in X

In Windows colors are extracted from a *palette*, usually the system palette, but possibly a custom or *logical* palette created for a specific application. Windows' default or *stock* palette contains at least 20 colors: the 16 colors normally supported by CGA systems, plus four additional colors, dithered if necessary. (If you would like to investigate building a logical palette for your own application, check your reference pages for *CreatePalette*, *SelectPalette* and *RealizePalette*.)

A Windows palette is addressed using the *COLORREF* data type. This is a 32 bit integer which can be used in three different contexts, or *modes*. The high order byte of a COLORREF value encodes the mode, and the low order bytes are interpreted as follows:

COLORREF

	Name	Value	Macro
MODE0	Absolute	0x00000000	RGB
MODE1	Palette Index	0x01000000	PALETTEINDEX
MODE2	Palette RGB	0x02000000	PALETTERGB

(MODE0 | 0x00FFFF)	Light magenta
(MODE1 | 0x000010)	The seventeenth color in a logical palette
(MODE2 | 0x770077)	The color closest to brown in a logical palette

Figure 8-13 Windows COLORREF Modes

Absolute Mode (mode 0): The low order three bytes encode a scaled 255 RGB value. This value is compared to the values in the stock palette, and the closest color is used.

Palette Index Mode (mode 1): The low order three bytes encode an index into an application's logical palette.

Palette RGB Mode (mode 2): The low order three bytes encode a scaled 255 RGB value. This value is compared to the values in an application's logical palette, and the closest color is used.

In X, colors are stored in a *color map*. This is usually a resource that is shared among all applications. It is an array of 16 bit RGB triples, big enough to store all the values a display system is capable of handling (2^{depth}). An element in the array is called a *color cell*.

Depending on device and operating system, an X color map may be *read/write* or *read only*. A read only color map is fully loaded when the system starts, and can't be changed. Indexes into a read only color map can be requested by calling a system routine and passing an RGB value; the routine interrogates the color map, and returns an index to the color cell that most closely matches the requested color.

Status	Red	Green	Blue
shared	0x7777	0x7777	0x0000
shared	0xFFFF	0x0000	0xFFFF
free			
shared	0x0000	0xFFFF	0x0000
private	0x0210	0x7575	0x7300
. . .	. . .	. . .	. . .

Figure 8-14 A Read/Write X Color Map

A read/write color map is initially empty, and is loaded as needed by allocating color cells as *shared* or *private*. An application can request allocation of a private color cell (which will be denied if the color map is full) and then load it with any RGB value. More often an application will request allocation of a shared color cell by calling a system routine and passing an RGB value, to which X responds as follows:

- If the color is already allocated in a shared color cell, X returns an index to the existing cell.
- If the color is not already allocated, and there is a free cell in the color map, X allocates a free cell, stores the requested value in the cell, and returns an index to it.
- If the color is not already allocated and the color map is full, X returns an index to the shared cell that contains the color that most closely matches the requested color.

As seen in **Figure 8-15**, X also allows you to request a color using a name in the X *color name data base*. A name in the data base is associated with a scaled 255 RGB triple.

Depending on your hardware, X may allow you to create and select different color maps. However, except in the case of unusual hardware, you can only have one color map (X) or one logical palette (Windows) selected, so switching a color map or logical palette is usually considered antisocial.

```
typedef struct
{
    unsigned long  pixel;
    unsigned short red;
    unsigend short green;
    unsigned short blue;
    char           flags;
} XColor;
void change_colors( Widget widget )
{
    Display  *display = XtDisplay( widget );
    Colormap colormap = None;
    XColor   backg,
             foreg_actual,
             foreg_exact;
    XtVaGetValues( widget, XmNcolormap, &colormap, NULL );

    /* Background: light gray */
    backg.red = 0x7777;
    backg.green = 0x7777;
    backg.blue = 0x7777;
    if ( !XAllocColor( display, colormap, &backg ) )
        raise_error( “Couldn’t allocate color” );

    if ( !XAllocNamedColor( display,
                            colormap,
                            “Aquamarine”,
                            &foreg_actual,
                            &foreg_exact
                          )
       )
        raise_error( “Couldn’t allocate Aquamarine” );

    XtVaSetValues( widget, XmNbackground, backg.pixel,
                           XmNforeground, foreg_actual.pixel,
                           NULL
                 );
}
```

Figure 8-15 Allocating Color Cells in X