COLOR AND GRAYSCALE LEVELS:

Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system. General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any. Color options are numerically coded with values ranging from 0 through the positive integers. For CRT monitors, these color codes are then converted to intensity level settings for the electron beams. With color plotters, the codes could control ink-jet deposits or pen selections.

COLOR AND GRAYSCALE LEVELS:

In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer Also, color-information can be stored in the frame buffer in two ways: We can store color codes directly in the frame buffer, or we can put the color codes in a separate table and use pixel values as an index into this table.

With the direct storage scheme, whenever a particular color code is specified in an application program, the corresponding binary value is placed in the frame buffer for each component pixel in the output primitives to be displayed in that color. A minimum number of colors can be provided in this scheme with 3 bits of storage per pixel, as shown in Table 4.1. Each of the three bit positions is used to control the intensity level (either on or off) of the corresponding electron gun in an RGB monitor..

COLOR AND GRAYSCALE LEVELS:

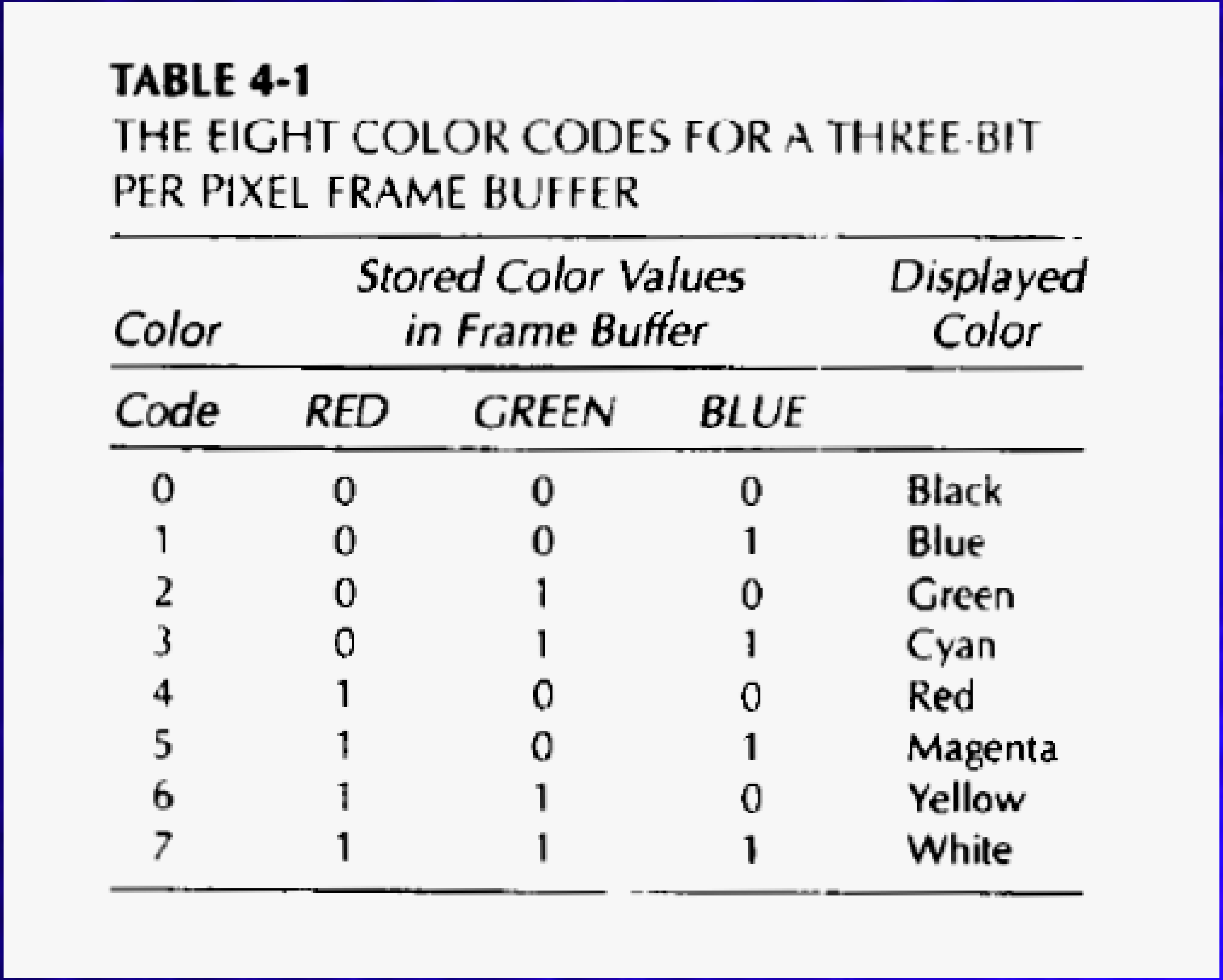


TABLE 4-1
THE EIGHT COLOR CODES FOR A THREE-BIT PER PIXEL FRAME BUFFER

Color	*Stored Color Values in Frame Buffer*			*Displayed Color*
Code	*RED*	*GREEN*	*BLUE*	
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

COLOR AND GRAYSCALE LEVELS:

The leftmost bit controls the red gun, the middle bit controls the green gun, and the rightmost bit controls the blue gun. Adding more bits per pixel to the frame buffer increases the number of color choices. With 6 bits per pixel, 2 bits can be used for each gun.

This allows four different intensity settings for each of the three color guns, and a total of 64 color values are available for each screen pixel. With a resolution of 1024 by 1024, a full-color (24bit per pixel) RGB system needs 3 megabytes of storage for the frame buffer. Color tables are an alternate means for providing extended color capabilities to a user without requiring large frame buffers. Lower-cost personal computer systems, in particular, often use color tables to reduce frame-buffer storage requirements.

COLOR AND GRAYSCALE LEVELS:

Color Tables:

Figure 4-16 illustrates a possible scheme for storing color values in a color lookup table (or video lookup table), where frame-buffer values are now used as indices into the color table. In this example, each pixel can reference any one of the 256 table positions, and each entry in the table uses 24 bits to specify an RGB color. For the color code 2081, a combination green-blue color is displayed for pixel location (x, y). Systems employing this particular lookup table would allow a user to select any 256 colors for simultaneous display from a palette of nearly 17 million colors. Compared to a full-color system. this scheme reduces the number of simultaneous colors that can be displayed, but it also reduces the frame buffer storage requirements to 1 megabyte. Some graphics systems provide 9 bits per pixel in the frame buffer, permitting a user to select 512 colors that could be used in each display.

COLOR AND GRAYSCALE LEVELS:

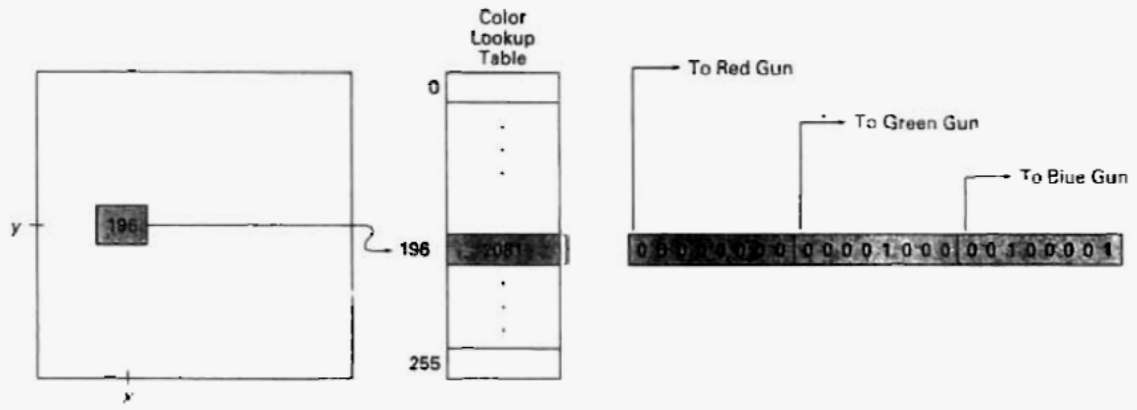
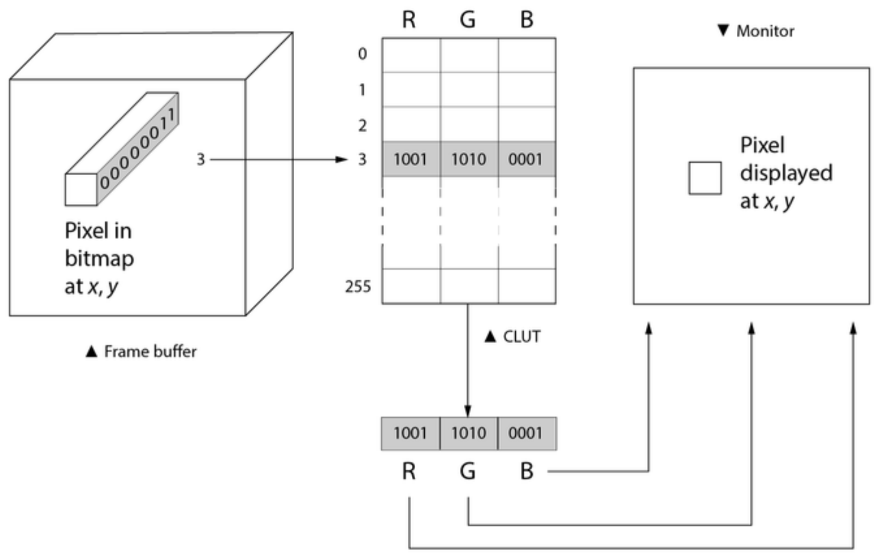


Figure 4-16
A color lookup table with 24 bits per entry accessed from a frame buffer with 8 bits per pixel. A value of 196 stored at pixel position (x, y) references the location in this table containing the value 2081. Each 8-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor.



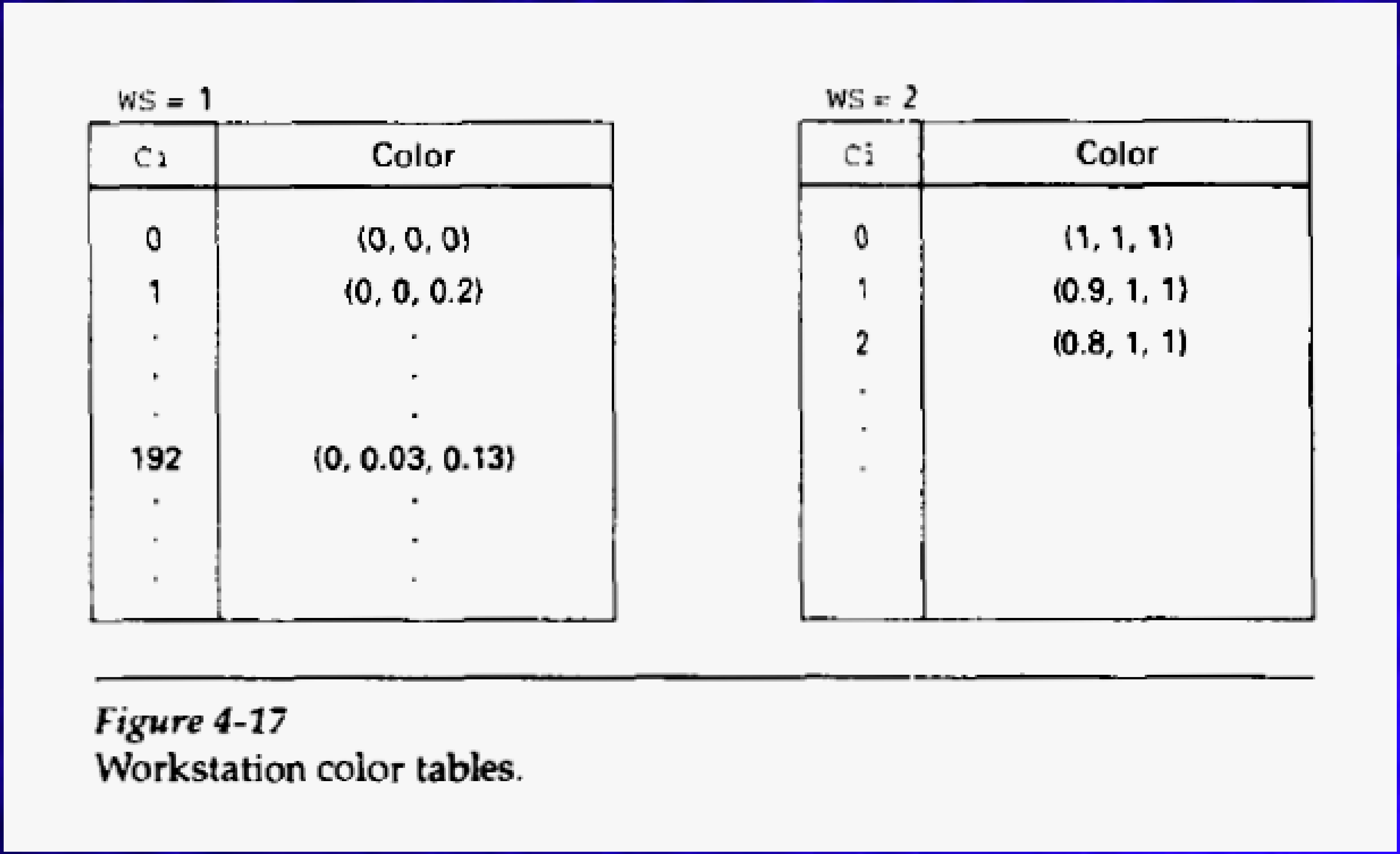
COLOR AND GRAYSCALE LEVELS:

A user can set color-table entries in a PHIGS applications program with the function

setColourRepresentation (ws, ci, colorptr)

Parameter ws identifies the workstation output device; parameter ci specifies the color index, which is the color-table position number (0 to 255 for the example in Fig. 4-16); and parameter colorptr points to a trio of RGB color values (r, g, b) each specified in the range from 0 to 1. An example of possible table entries for color monitors is given in Fig. 4-17.

COLOR AND GRAYSCALE LEVELS:



WS = 1

Ci	Color
0	(0, 0, 0)
1	(0, 0, 0.2)
.	.
.	.
.	.
192	(0, 0.03, 0.13)
.	.
.	.
.	.

WS = 2

Ci	Color
0	(1, 1, 1)
1	(0.9, 1, 1)
2	(0.8, 1, 1)
.	
.	
.	

Figure 4-17
Workstation color tables.

COLOR AND GRAYSCALE LEVELS:

Grayscale:

With monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale, for displayed primitives. Numeric values over the range from 0 to 1 can be used to specify grayscale levels, which are then converted to appropriate binary codes for storage in the raster. This allows the intensity settings to be easily adapted to systems with differing grayscale capabilities.

Table 4-2 lists the specifications for intensity codes for a four-level grayscale system. In this example, any intensity input value near 0.33 would be stored as the binary value 01 in the frame buffer, and pixels with this value would be displayed as dark gray. If additional bits per pixel are available in the frame buffer, the value of 0.33 would be mapped to the nearest level. With 3 bits per pixel, we can accommodate 8 gray levels; while 8 bits per pixel would give us 256 shades of gray. An alternative scheme for storing the intensity information is to convert each intensity code directly to the voltage value that produces this grayscale level on the output device in use.

COLOR AND GRAYSCALE LEVELS:

When multiple output devices are available at an installation, the same color-table interface may be used for all monitors. In this case, a color table for a monochrome monitor can be set up using a range of RGB values as in Fig. 4-17, with the display intensity corresponding to a given color index ci calculated as

$$\text{intensity} = 0.5[\min(r, g, b) + \max(r, g, b)]$$

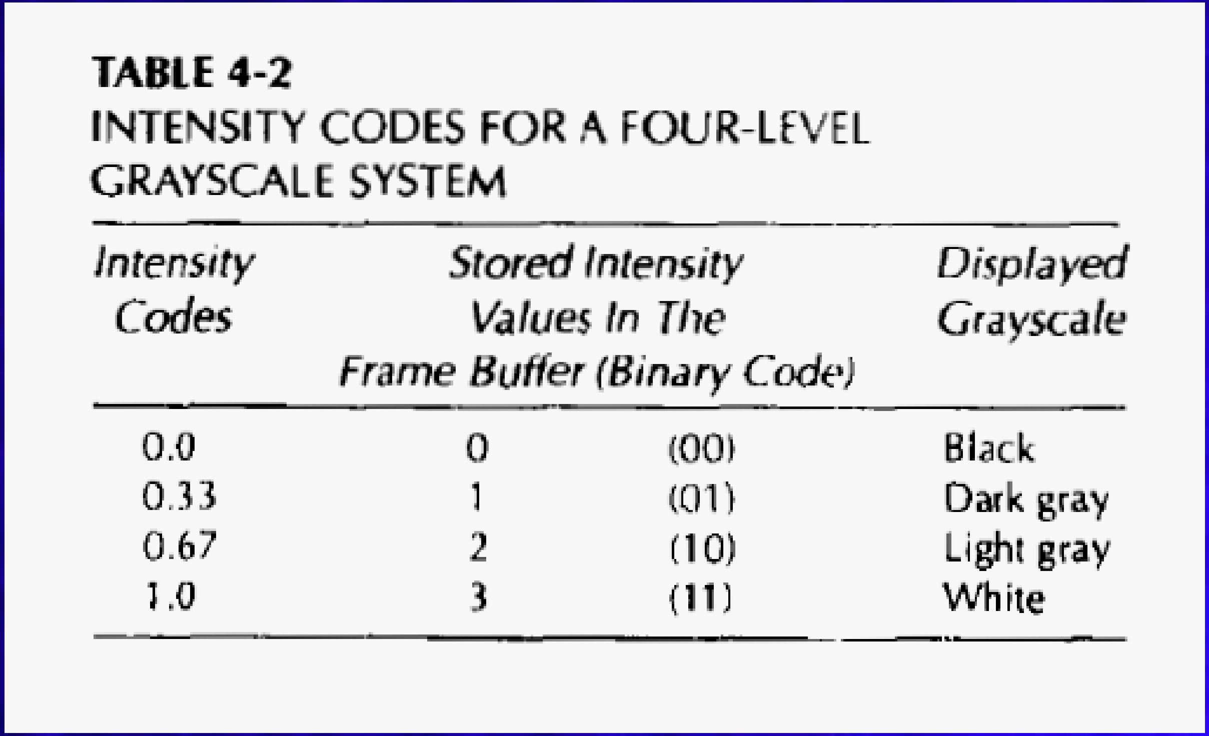


TABLE 4-2
INTENSITY CODES FOR A FOUR-LEVEL GRAYSCALE SYSTEM

Intensity Codes	*Stored Intensity Values In The Frame Buffer (Binary Code)*		*Displayed Grayscale*
0.0	0	(00)	Black
0.33	1	(01)	Dark gray
0.67	2	(10)	Light gray
1.0	3	(11)	White

COLOR AND GRAYSCALE LEVELS:

AREA-FILL ATTRIBUTES

Options for filling a defined region include a choice between a solid color or a patterned fill and choices for the particular colors and patterns. These fill options can be applied to polygon regions or to areas defined with curved boundaries, depending on the capabilities of the available package. In addition, areas can be painted using various brush styles, colors, and transparency parameters.

Fill Styles

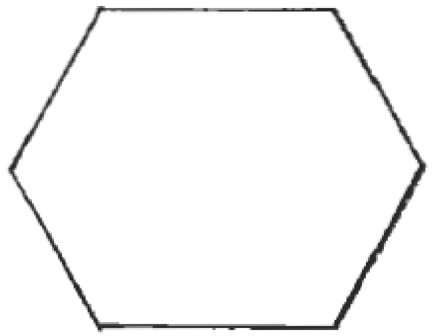
Areas are displayed with three basic fill styles: hollow with a color border, filled with a solid color, or filled with a specified pattern or design. A basic fill style is selected in a PHIGS program with the function

```
setInteriorStyle (fs)
```

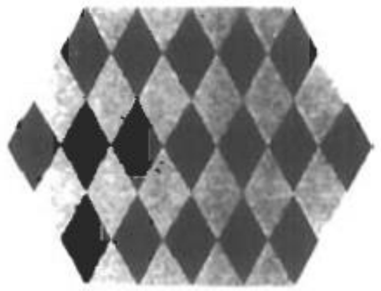
values for the fill-style parameter fs include hollow, solid, and pattern (Fig. 4-18).

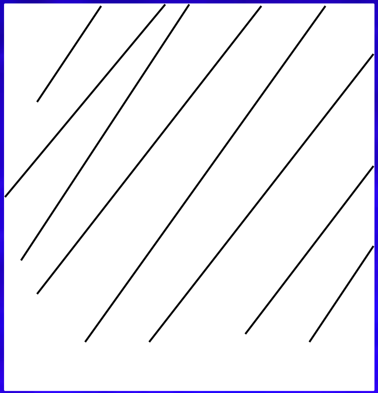
COLOR AND GRAYSCALE LEVELS:

Another value for fill style is hatch, which is used to fill an area with selected hatching patterns-parallel lines or crossed lines-as in Fig. 4-19. As with line attributes, a selected fill style value is recorded in the list of system attributes and applied to fill the interiors of subsequently specified areas. Fill selections for parameter fs are normally applied to polygon areas, but they can also be implemented to fill regions with curved boundaries.


Hollow
(a)

Solid
(b)


Patterned
(c)



COLOR AND GRAYSCALE LEVELS:

Hollow areas are displayed using only the boundary outline, with the interior color the same as the background color. A solid fill is displayed in a single color up to and including the borders of the region. The color for a solid interior or for a hollow area outline is chosen with

```
setInteriorColorIndex( fc )
```

where fill color parameter fc is set to the desired color code. A polygon hollow fill is generated with a line drawing routine as a closed polyline.

Other fill options include specifications for the edge type, edge width, and edge color of a region. These attributes are set independently of the fill style or fill color, and they provide for the same options as the line-attribute parameters (line type, line width, and line color). That is, we can display area edges dotted or dashed, fat or thin, and in any available color regardless of how we have filled the interior.

CHARACTER ATTRIBUTES:

The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation. Attributes can be set both for entire character strings (text) and for individual characters defined as marker symbols.

There are a great many text options that can be made available to graphics programmers. First of all, there is the choice of font (or typeface), which is a set of characters with a particular design style such as New York, Courier, Helvetica, London, 'Times Roman, and various special symbol groups. The characters in a selected font can also be displayed with assorted underlining styles (solid, dotted, double) in boldface, in italics. and in outline or shadow styles. A particular font and associated style is selected in a PHIGS program by setting an integer code for the text font parameter tf in the function

setTextFont (tf)

CHARACTER ATTRIBUTES:

Font options can be made available as predefined sets of grid patterns or as character sets designed with polylines and spline curves.

Color settings for displayed text are stored in the system attribute list and used by the procedures that load character definitions into the frame buffer. When a character string is to be displayed, the current color is used to set pixel values in the frame buffer corresponding to the character shapes and positions.

Control of text color (or intensity) is managed from an application program with

```
setTextColourIndex (tc)
```

where text color parameter tc specifies an allowable color code.

CHARACTER ATTRIBUTES:

We can adjust text size by scaling the overall dimensions (height and width) of characters or by scaling only the character width. Character size is specified by printers and compositors in points, where 1 point is 0.013837 inch (or approximately 1/72 inch). For example, the text you are now reading is a 10-point font Point measurements specify the size of the body of a character (Fig. 4-25), but different fonts with the same point specifications can have different character sizes, depending on the design of the typeface.

The distance between the bottomline and the topline of the character body is the same for all characters in a particular size and typeface, but the body width may vary. Proportionally spaced fonts assign a smaller body width to narrow characters such as i, j, 1, and f compared to broad characters such as W or M. Character height is defined as the distance between the baseline and the capline of characters. Kerned characters, such as f and j in Fig. 4-25, typically extend beyond the character-body limits, and letters with descenders (g, j, p, q, y) extend below the baseline. Each character is positioned within the character body by a font designer to allow suitable spacing along and between print lines when text is displayed with character bodies touching.

CHARACTER ATTRIBUTES:

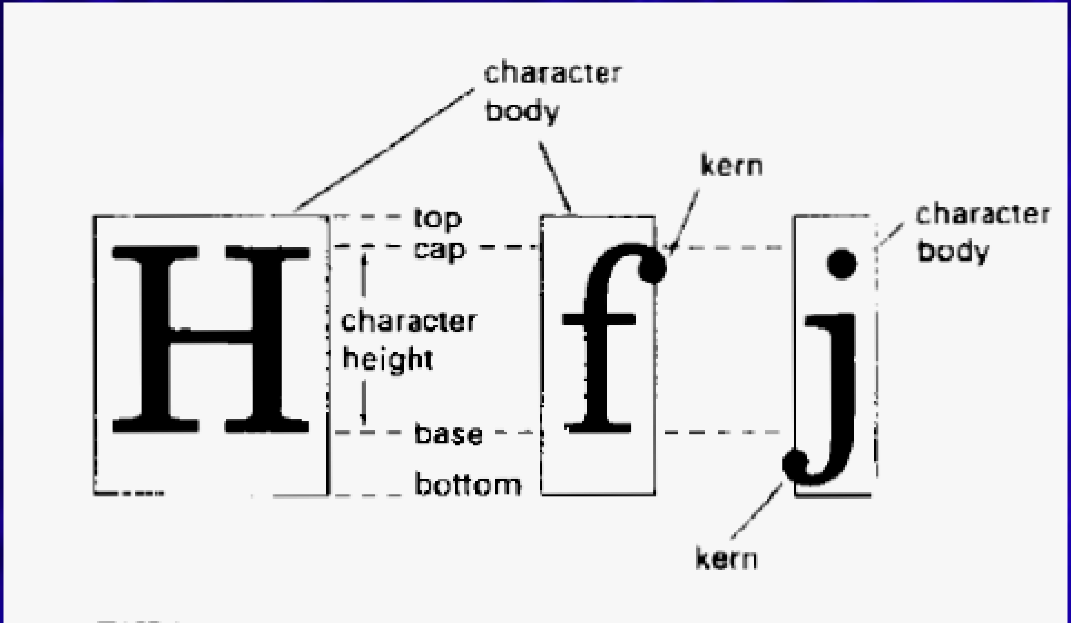
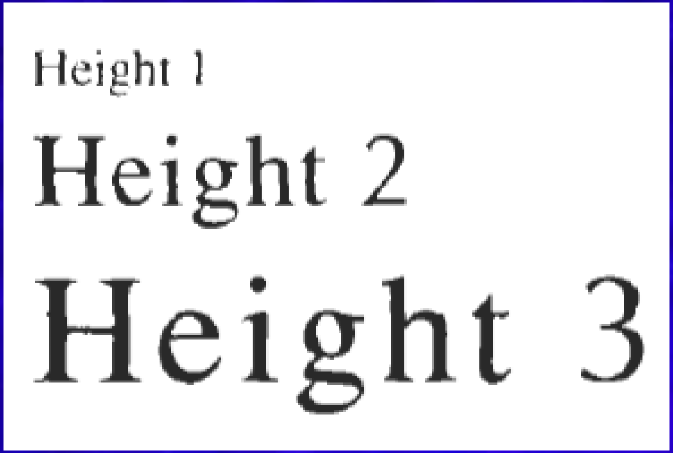


Figure 4-25
Character body.



The effect of different character height setting on displayed text

CHARACTER ATTRIBUTES:

Text size can be adjusted without changing the width-to-height ratio of characters with

```
setCharacterHeight (ch)
```

Parameter ch is assigned a real value greater than 0 to set the coordinate height of capital letters: the distance between baseline and capline in user coordinates. This setting also affects character-body size, so that the width and spacing of characters is adjusted to maintain the same text proportions. For instance, doubling the height also doubles the character width and the spacing between characters.

Figure 4-26 shows a character string displayed with three different character heights.

CHARACTER ATTRIBUTES:

The width only of text can be set with the function

setCharacterExpansionFactor (cw)

where the character-width parameter cw is set to a positive real value that scales the body width of characters. Text height is unaffected by this attribute setting. Examples of text displayed with different character expansions is given in Fig. 4-27.

Spacing between characters is controlled separately with

setCharacterSpacing (cs)

where the character-spacing parameter cs can be assigned any real value. The value assigned to cs determines the spacing between character bodes along print lines. Negative values for cs overlap character bodies; positive values insert space to spread out the displayed characters. Assigning the value 0 to cs causes text to be displayed with no space between character bodies. The amount of spacing to be applied is determined by multiplying the value of cs by the character height (distance between baseline and capline).

CHARACTER ATTRIBUTES:

In Fig. 4-28, a character string is displayed with three different settings for the character-spacing parameter.

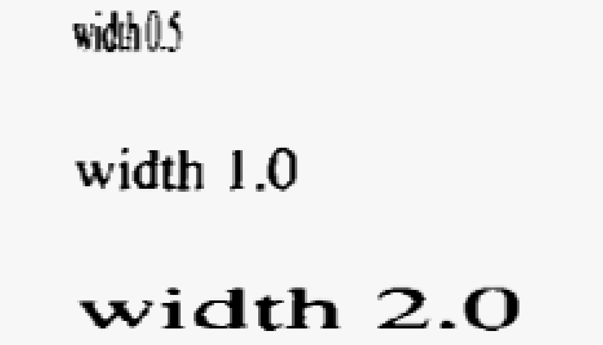


Figure 4-27
The effect of different character-width settings on displayed text.

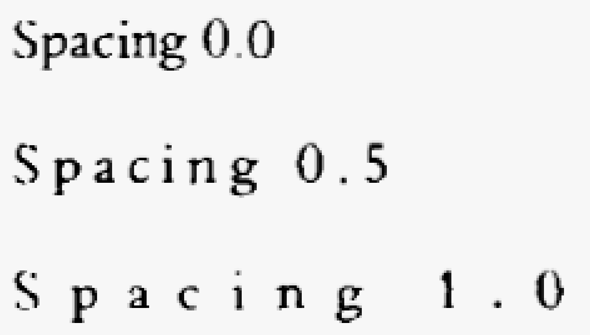


Figure 4-28
The effect of different character spacings on displayed text.

CHARACTER ATTRIBUTES:

The orientation for a displayed character string is set according to the direction of the character up vector. A procedure for orienting text rotates characters so that the sides of character bodies, from baseline to capline, are aligned with the up vector. The rotated character shapes are then scan converted into the frame buffer.

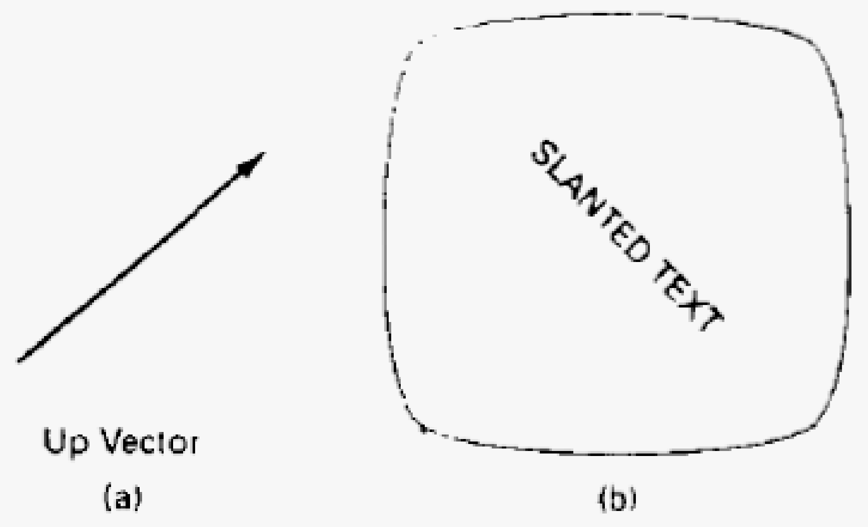


Figure 4-29
Direction of the up vector (a) controls the orientation of displayed text (b).

CHARACTER ATTRIBUTES:

It is useful in many applications to be able to arrange character strings vertically or horizontally (Fig. 4-30).

Character strings can also be oriented using a combination of up-vector and text-path specifications to produce slanted text.

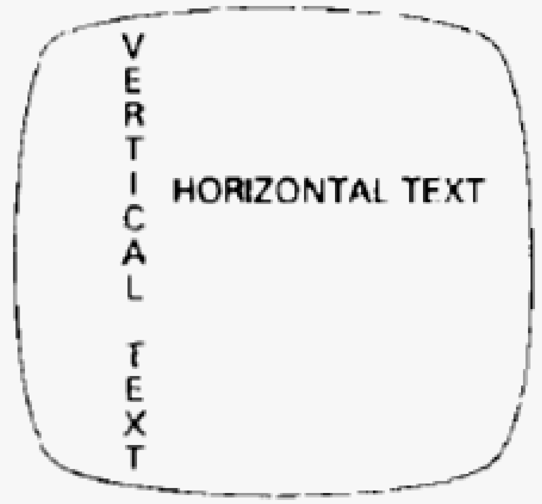


Figure 4-30
Text path attributes can be set to produce horizontal or vertical arrangements of character strings.

CHARACTER ATTRIBUTES:

Another handy attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates.

setTextAlignment (h, v)

Horizontal alignment is set by assigning h a value of left, centre, or right. Vertical alignment is set by assigning v a value of top, cap, half, base or bottom. The interpretation of these alignment values depends on the current setting for the text path.

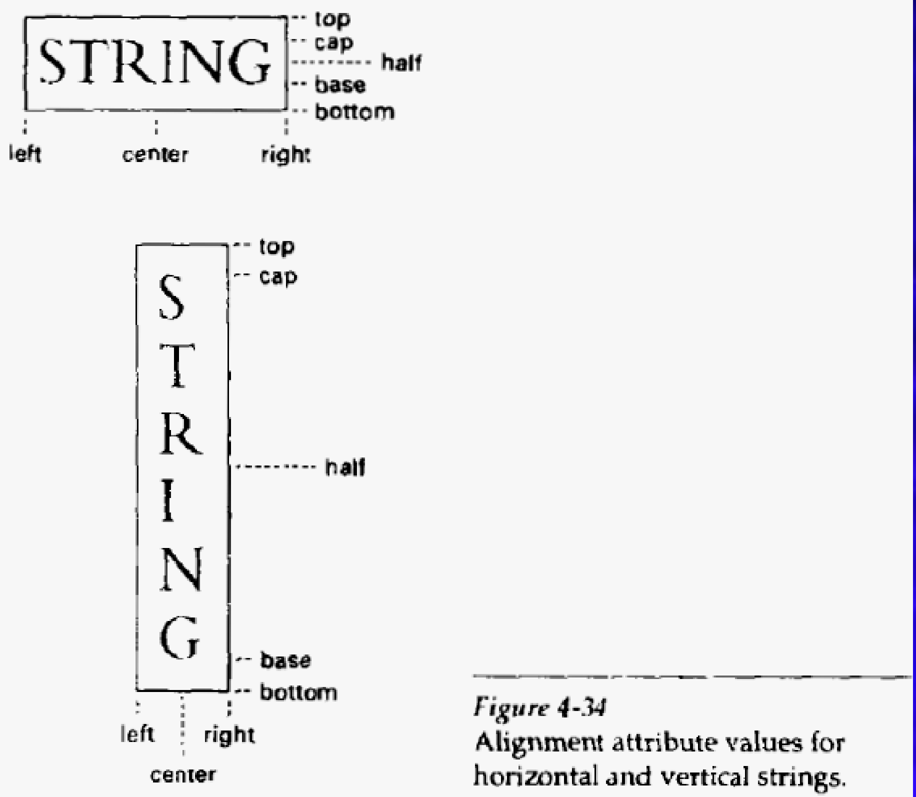


Figure 4-34
Alignment attribute values for horizontal and vertical strings.

BUNDLED ATTRIBUTES:

With the procedures we have considered so far, each function references a single attribute that specifies exactly how a primitive is to be displayed with that attribute setting. These specifications are called individual (or unbundled) attributes, and they are meant to be used with an output device that is capable of displaying primitives in the way specified.

If an application program, employing individual attributes, is interfaced to several output devices, some of the devices may not have the capability to display the intended attributes. A program using individual color attribute, for example, may have to be modified to produce acceptable output on a monochromatic monitor.

BUNDLED ATTRIBUTES:

Individual attribute commands provide a simple and direct method for specifying attributes when a single output device is used. When several kinds of output devices are available at a graphics installation, it is convenient for a user to be able to say how attributes are to be interpreted on each of the different devices.

This is accomplished by setting up tables for each output device that lists sets of attribute values that are to be used on that device, to display each primitive type. A particular set of attribute values for a primitive on each output device is then chosen by specifying the appropriate table index. Attributes specified in this manner called bundled attributes. The table for each primitive that defines groups of attribute values to be used when displaying at primitive on a particular output device is called a bundle table.

BUNDLED ATTRIBUTES:

Attributes that may be bundled into the workstation table entries are those that do not involve coordinate specifications, such as color and line type. The choice between a bundled or an unbundled specification is made by setting switch called the aspect source flag for each of these attributes:

setIndividualASF (attributeptr, flagptr)

where parameter attributeptr points to a list of attributes, and parameter flagptr points to the corresponding list of aspect source flags. Each aspect source flag can be assigned a value of individual or bundled.