Computer Graphics

Lecture 11

Curve Attributes: Parameters for curve attributes are the same as those for line segments. We can display curves with varying colours, widths, dot dash patterns, and available pen or brush options. Methods for adapting curve-drawing algorithms to accommodate attribute selections are similar to those for line drawing.

The pixel masks discussed for implementing line-type options are also used in raster curve algorithms to generate dashed and dotted patterns. For example, the mask 11100 produces the dashed circle shown in Fig. 4.12. We can generate the dashes in the various octants using circle symmetry, but we must shift the pixel positions to maintain the correct sequence of dashes and spaces as we move from one octant to the next. Also, as in line algorithms, pixel masks display dashes and inter-dash spaces that vary in length according to the slope of the curve. If we want to display constant-length dashes, we need to adjust the number of pixels plotted in each dash as we move around the circle circumference. Instead of applying a pixel mask with constant spans, we plot pixels along equal angular arcs to produce equal length dashes.

Raster curves of various widths can be displayed using the method of horizontal or vertical pixel spans. Where the magnitude of the curve slope is less than 1, we plot vertical spans; where the slope magnitude is greater than 1, we plot horizontal spans. Fig 4.13 demonstrates this method for displaying a circular arc of width 4 in the first quadrant. Using circle symmetry, we generate the circle path with vertical spans in the octant from x = 0 to x = y, and then reflect pixel positions about the line y = x to obtain the remainder of the curve shown. Circle sections in the other quadrants are obtained by reflecting pixel positions in the first quadrant about the coordinate axes. The thickness of curves displayed with this method is again a function of curve slope. Circles, ellipses, and other curves will appear thinnest where the slope has a magnitude of 1.

Another method for displaying thick curves is to fill in the area between two parallel curve paths, whose separation distance is equal to the desired width. We could do this using the specified curve path as one boundary

and setting up the second boundary either inside or outside the original curve path. This approach, however, shifts the original curve path either inward or outward, depending on which direction we choose for the second boundary. We can maintain the original curve position by setting the two boundary curves at a distance of one-half the width on either side of the specified curve path. An example of this approach is shown in Fig. 4-14 for a circle segment with radius 16 and a specified width of 4. The boundary arcs are then set at a separation distance of 2 on either side of the radius of 16. To maintain the proper dimensions of the circular arc, we can set the radii for the concentric boundary arcs at r = 14 and r = 17. Although this method is accurate for generating thick circles, in general, it provides only an approximation to the true area of other thick curves. For example, the inner and outer boundaries of a fat ellipse generated with this method do not have the same foci

Pen (or brush) displays of curves are generated using the same techniques discussed for straight line segments. We replicate a pen shape along the line path, as illustrated in Fig. 4-15 for a circular arc in the first quadrant.

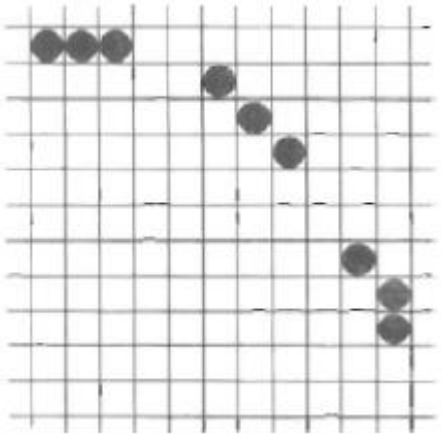
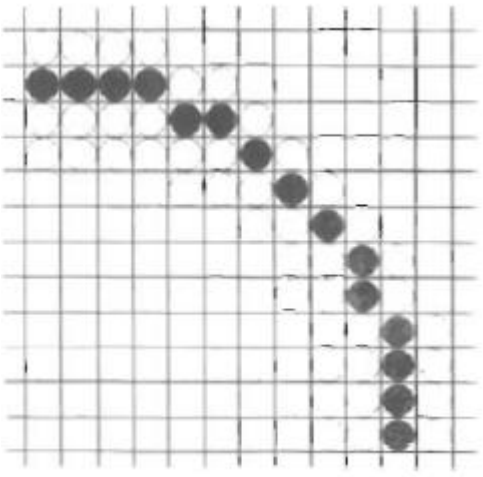


Figure 4-12
A dashed circular arc displayed with a dash span of 3 pixels and an *inter-dash spacing* of 2 pixels



Figurc 4-13
Circular arc of width 4 plotted with pixel spans.

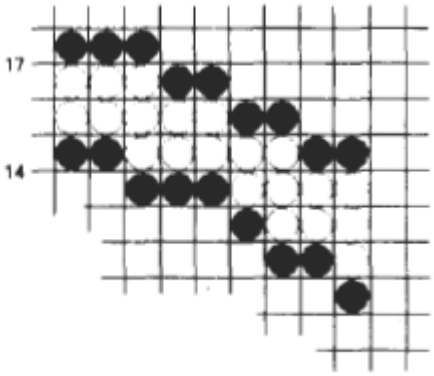


Figure 4.14
A circular arc of width 4 and radius 16 displayed by filling the region between two concentric arcs.

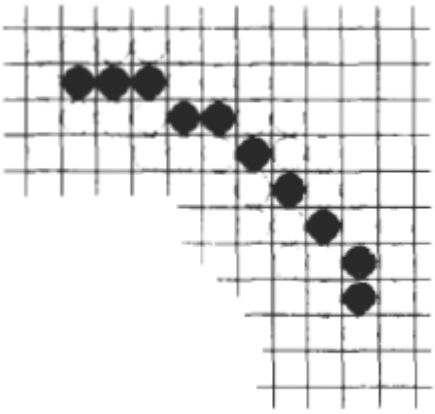


Figure 4.**15**
Circular arc displayed with rectangular pen.

Colour and Grayscale Levels: Various colour 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 colours, while random-scan monitors typically offer only a few colour choices, if any. Colour options are numerically coded with values ranging from 0 through the positive integers. For CRT monitors, these colour codes are then converted to intensity-level settings for the electron beams. With colour plotters, the codes could control ink-jet deposits or pen selections

In a colour raster system, the number of colour choices available depends on the amount of storage provided per pixel in the frame buffer. Also, colour-information can be stored in the frame buffer in two ways: We can store colour codes directly in the frame buffer, or we can put the colour codes in a separate table and use pixel values as an index into this table. With the direct storage scheme, whenever a particular colour 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 colour. A minimum number of colours can be provided in t h 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. 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 colour 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 colour guns, and a total of 64 colour values are available for each screen pixel. With a resolution of 1024 by 1024, a full-colour (24bit per pixel) RGB system needs 3 megabytes of storage for the frame buffer. Colour tables are an alternate means for providing extended colour capabilities to a user without requiring large frame buffers. Lower-cost personal computer systems, in particular, often use colour tables to reduce frame-buffer storage requirements.

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 Tables

Figure 4-16 illustrates a possible scheme for storing colour values in a colour look-up table (or video lookup table), where frame-buffer values art- now used as indices into the colour 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 colour. For the colour code 2081, a combination green-blue colour is displayed for pixel location (x, y). Systems employing this particular lookup table would allow a user to select any 256 colours for simultaneous display from a palette of nearly 17 million colours. Compared to a full-colour system, this scheme reduces the number of simultaneous colours 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 colours that could be used in each display. A user can set colour-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 colour index, which is the colour-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

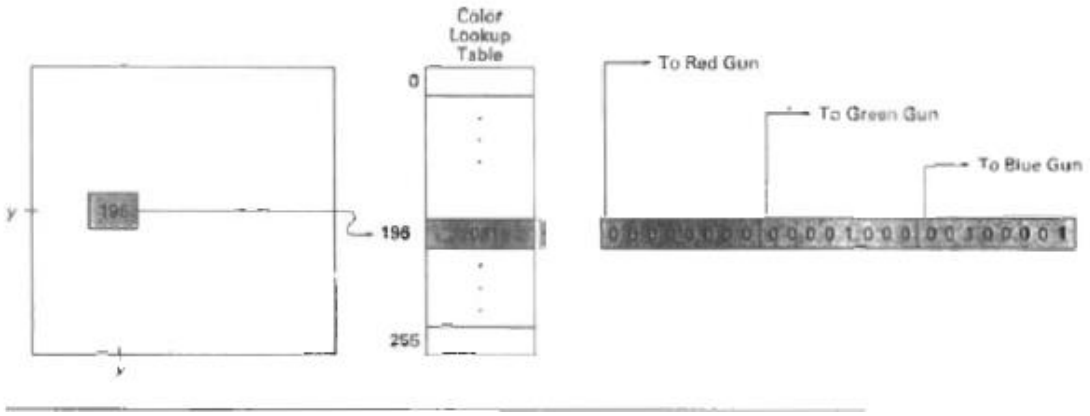


Figure 4-16
A color lookup table with 24 bits per entry accessed from a frame buffer with 8 bits per pixel. A value of 196 stored at pixel position (x, y) references the location in this table containing the value 2081. Each 8-bit segment of this entry control the intensity level of one of the three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

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.

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

$$Intensity = 0.5[min(r, g, b) + max(r, g, b)]$$

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

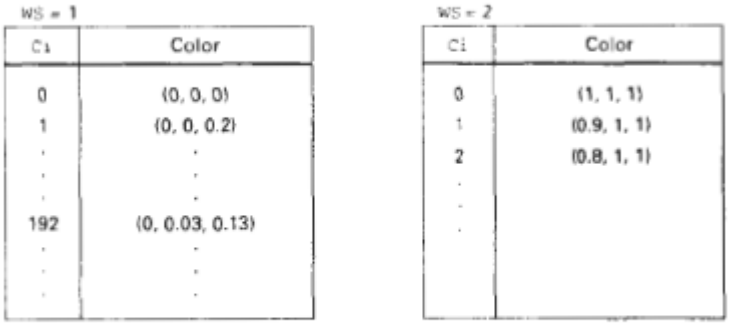


Figure 4-17
Workstation color tables.