Unit -1

COMPUTER GRAPHICS

-INTRODUCTION, OUTPUT PRIMITIVES & 2-D TRANSFORMATION

Applications of Computer Graphics and Multimedia

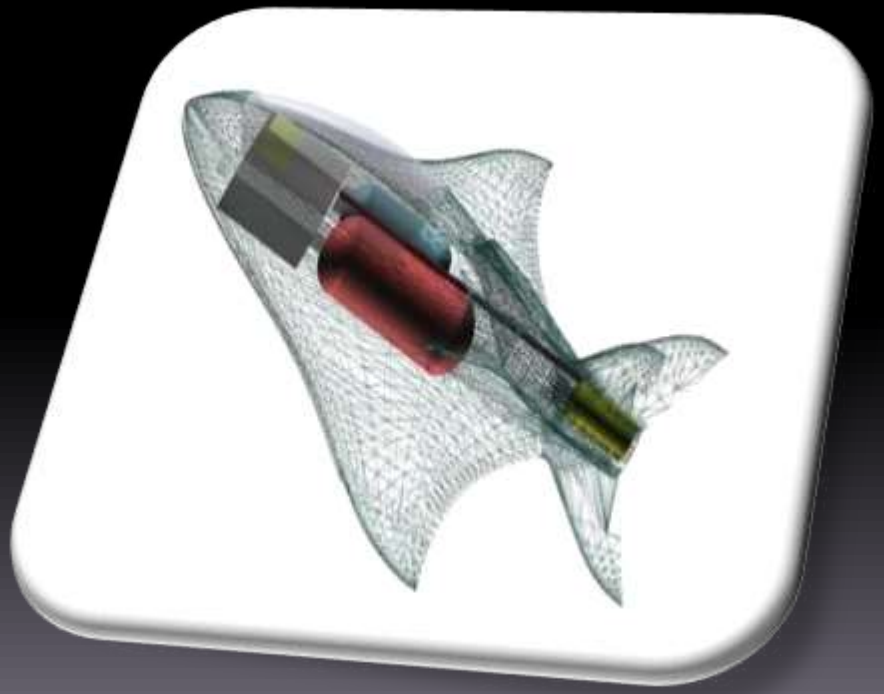
- Computer-Aided Design (CAD)
- Presentation Graphics
- Computer Art
- Entertainment
- Education and Training
- Visualization
- Image processing
- Graphical User Interfaces

Computer Aided Design

- For engineering and architectural systems, a major use of computer graphics is in design processes.
- CAD methods are used in the design of buildings, automobiles, aircraft, watercraft, space craft, computers, textiles and many other products.
- For some design applications, objects are first displayed in a wireframe out-line form, that shows the overall shape and internal features of objects.

- Wireframe display for Automobile

- Wireframe display for Aircraft



- Circuits and networks for communications, water supply, or other utilities are constructed with repeated placement of a few graphical shapes.
- Standard shapes for electrical, electronic, and logic circuits are often supplied by the design package.
- Facilities are provided to designer to try out alternate circuit schematics for minimizing the number of components or the space required for the system.
- Animations are often used in CAD applications.
- Software packages for CAD applications typically provide the designer with a multi-window environment.
- Almost in every branch of engineering, the graphics are used in different fashion.

Presentation Graphics

- Presentation graphics are used to produce illustrations for reports or to generate slides or transparencies for use with projectors.
- They are commonly used to summarize financial, statistical, mathematical, scientific, and economic data for research reports, managerial reports, consumer information bulletins, and other types of reports.
- Examples are bar charts, line graphs, surface graphs, pie charts, and other displays showing relationships between multiple parameters.

Computer Arts

- Computer graphics methods are widely used in both fine art and commercial art applications.
- Artists use special purpose hardware, software packages or programs that allow them to paint pictures on the screen of a video monitor.
- The picture is usually painted electronically on a graphics tablet (digitizer) using a stylus, which can simulate different brush strokes, brush widths, and colors.
- The stylus translates changing hand pressure into variable line widths, brush sizes, and color gradations.
- Computer arts are also used in commercial art for logos and other designs, page layouts combining text and graphics, TV advertising, and other areas.
- ***Morphing*** is a common graphics method employed in many commercials, where one object is transformed into another.

Art design using paint brush

Graphics design using mathematical functions

Entertainment

?

?

?

?

?

?

?

?

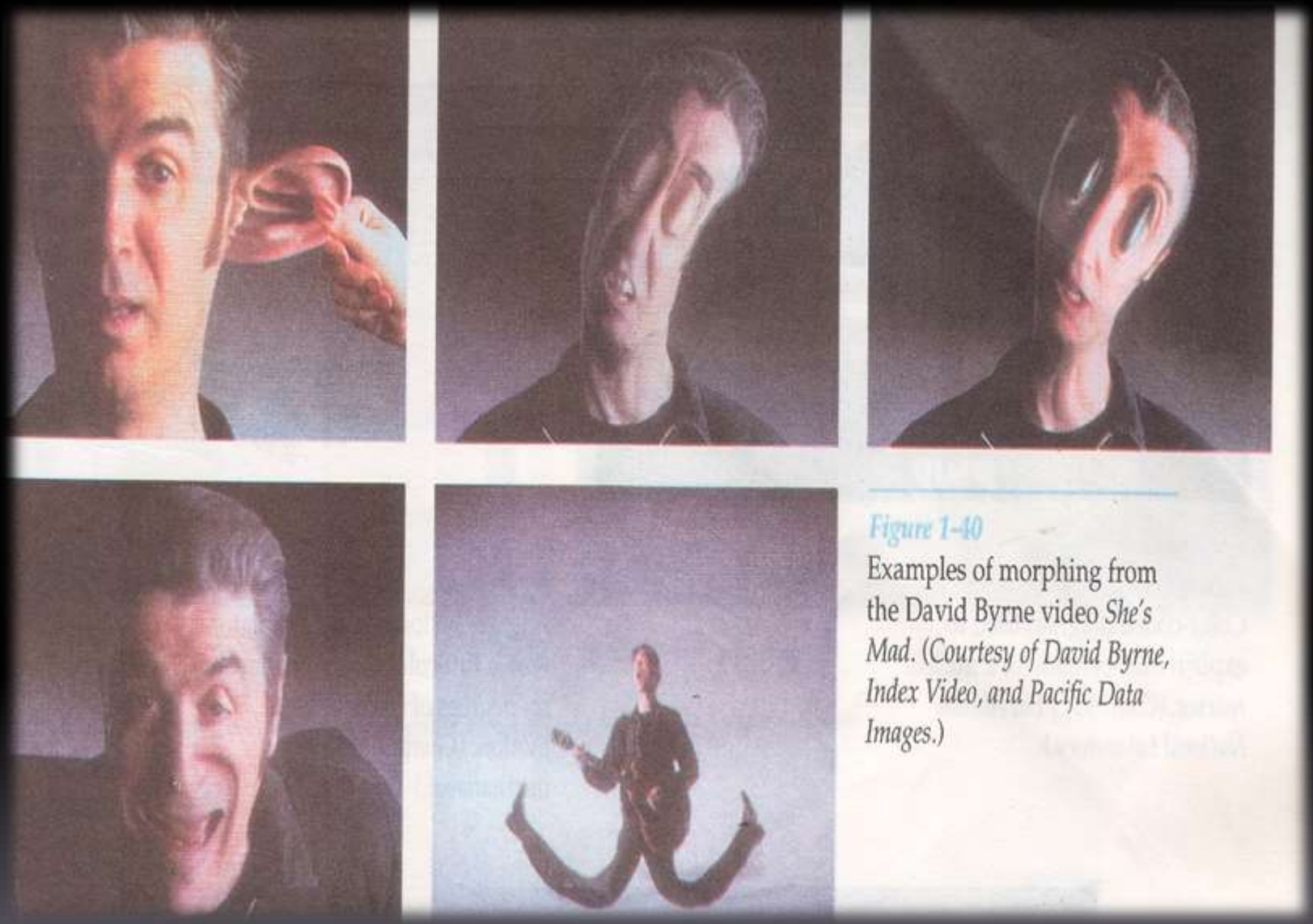


Figure 1-40
Examples of morphing from the David Byrne video *She's Mad*. (*Courtesy of David Byrne, Index Video, and Pacific Data Images.*)

Education and Training

- Computer-generated models of physical, financial, and economic systems are often used as educational aids.
- Models can help trainees to understand the operation of the system.
- For some training applications, special systems are designed. For example, the simulators for practice session or training of ship captains, aircraft pilots, heavy-equipment operators, and air traffic control personnel.
- Most simulators provide graphics screens for visual operation.

Visualization

- Scientists, business analysts or some people who often need to analyze large amounts of information or to study the behavior of certain process are benefited by graphics.
- Numerical simulations carried out on supercomputers frequently produce data files containing thousands and even millions of data values.
- Scanning these large sets of numbers to determine trends and relationships is a tedious and ineffective process. But if the data are converted to a visual form, the trends and patterns are often immediately apparent.

Image Processing

- In computer graphics, a computer is used to create a picture. Image processing, on the other hand, applies techniques to modify or interpret existing pictures, such as photographs and TV scans.
- Principal applications of image processing are

1. Improving picture quality and
2. Machine perception of visual information

Graphical User Interfaces

- Menus, icons, windows etc.

Example of VFX (Visual Effects)

Example of VFX (Visual Effects)

Example of VFX (Visual Effects)

Example of VFX (Visual Effects)

Example of VFX (Visual Effects)

Example of VFX (Visual Effects)

Graphics Software

- There are two general classifications of graphics software: (i) General programming packages and (ii) Special-purpose application packages.

1. General programming package provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN. Example of a general graphics programming package is the GL(Graphics Library).
2. Application graphics packages are designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work. Examples of such applications packages are the artist's painting programs, CAD systems, medical applications etc.

Software Standards

1.Graphical Kernel System (GKS)

- **GKS** was the first ISO standard for low-level computer graphics, introduced in 1977.
- GKS provides a set of drawing features for two-dimensional vector graphics suitable for charting and similar duties.
- The calls are designed to be portable across different programming languages, graphics devices and hardware, so that applications written to use GKS will be readily portable to many platforms and devices.
- Next version of GKS supported 3D pictures.
- A descendant of GKS was PHIGS.

2.PHIGS

- **PHIGS** (**Programmer's Hierarchical Interactive Graphics System**) is an API standard for rendering 3D computer graphics.
- PHIGS was designed in the 1980s, inheriting many of its ideas from the Graphical Kernel System of the late 1970s.
- It include some extra features like color specification, picture manipulation, surface rendering etc.
- It provide basic graphics function, but it does not specify any standard method to interface with devices.
- It does not support any method to store and transmission of images.
- Next version is PHIGS+. It include 3D surface shading capability.

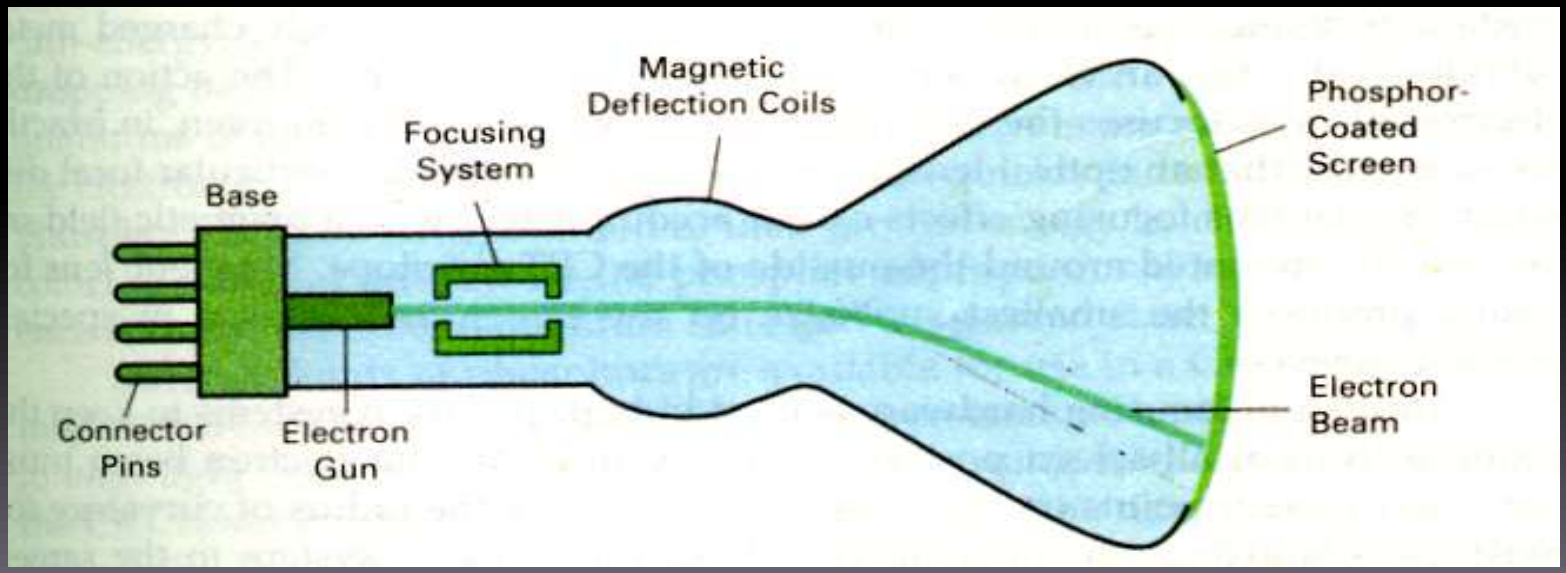
- To overcome the limitations of PHIGS, separate standards have been developed.
- Standardization for device interface methods is given in the ***Computer Graphics Interface (CGI)*** system and the ***Computer Graphics Metafile (CGM)*** system specifies standards for archiving and transporting pictures.

3. OpenGL

- **OpenGL** (**Open G**raphics **L**ibrary) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics.
- The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.
- OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels.
- OpenGL's "low-level" API allowed the programmer to make dramatic improvements in rendering performance by first examining the data on the CPU-side before trying to send it over the bus to the graphics engine. For instance, the programmer could "cull" the objects by examining which objects were actually visible in the scene, and sending only those objects that would actually end up on the screen. This was kept private in PHIGS, making it much more difficult to tune performance.

Video Display Devices

- The primary output device in a graphics system is a video monitor.
- The operation of most video monitors is based on the standard cathode-ray tube (CRT).



Raster-Scan Displays

- The most common type of graphics monitor employing a CRT is the raster scan display, based on television technology.
- In raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom.
- As the electron beam moves across each row, the beam intensity is turned on and off the create a pattern of illuminated spots.
- Picture definition is stored in memory area called the ***refresh buffer*** or ***frame buffer***.
- This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row(scan line) at a time.
- Each screen point is referred to as a ***pixel*** or ***pel*** (picture element).

- Intensity range for pixel positions depends on the capability of the raster system.
- In a simple black and white system, each screen point is either on or off, so only one bit per pixel is needed to control the intensity of screen positions.
- Bit value of 1 indicates that the electron beam is to be turned on at that position, and a value of 0 indicates that the beam intensity is off.
- On a black and white system with one bit per pixel, the frame buffer is commonly called a ***bitmap***.
- Additional bits are needed when color and intensity variations can be displayed (up to 24 bits per pixel).
- A system with 24 bits per pixel and a screen resolution of 1024 by 1024 requires 3 MB of storage for frame buffer.
- For systems with multiple bits per pixel, the frame buffer is often referred to as a ***pixmap***.

- Refreshing on raster scan displays is carried out at the rate of 60 to 80 frames per second or higher.
- Refresh rates are described in units of cycle per second, or Hertz (Hz), where cycle correspond to one frame (i.e 60 Hz).
- At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.
- The return to the left of the screen, after refreshing each scan line, is called the ***horizontal retrace*** of the electron beam. And at the end of each frame, the electron beam returns to the top left corner of the screen to begin the next frame is called ***vertical retrace***.

Random Scan Display

- When operated as a random scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn.
- Random scan monitors draw a picture one line at a time and for this reason are also referred to as ***vector displays*** or ***stroke-writing*** or ***calligraphic displays***.
- The component lines of a picture can be drawn and refreshed by a random-scan system in any specified order.
- Refresh rate on a random scan system depends on the number of lines to be displayed.
- Picture definition is now stored as a set of line drawing commands in an area of memory referred to as the ***refresh display file***. Sometimes it is also called the ***display list***, ***display program***, or simply ***refresh buffer***.

- To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn.
- After all line drawing commands have been processed, the system cycles back to the first line command in the list.
- Random scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.
- When a small set of lines is to be displayed, each refresh cycle is delayed to avoid refresh rates greater than 60 frames per second. Otherwise, faster refreshing of the set of lines could burn out the phosphor.

Random-Scan Systems v/s Raster-Scan Systems

- Random scan systems are designed for line drawing applications and cannot display realistic shaded scenes as Raster scan systems.
- Since picture definition is stored as a set of line-drawing instructions and not as a set of intensity values for all screen points, vector displays generally have higher resolution than raster systems.
- Vector displays produce smooth line drawings because the CRT beam directly follows the line path. A raster system in contrast, produces jagged lines that are plotted as discrete point sets.

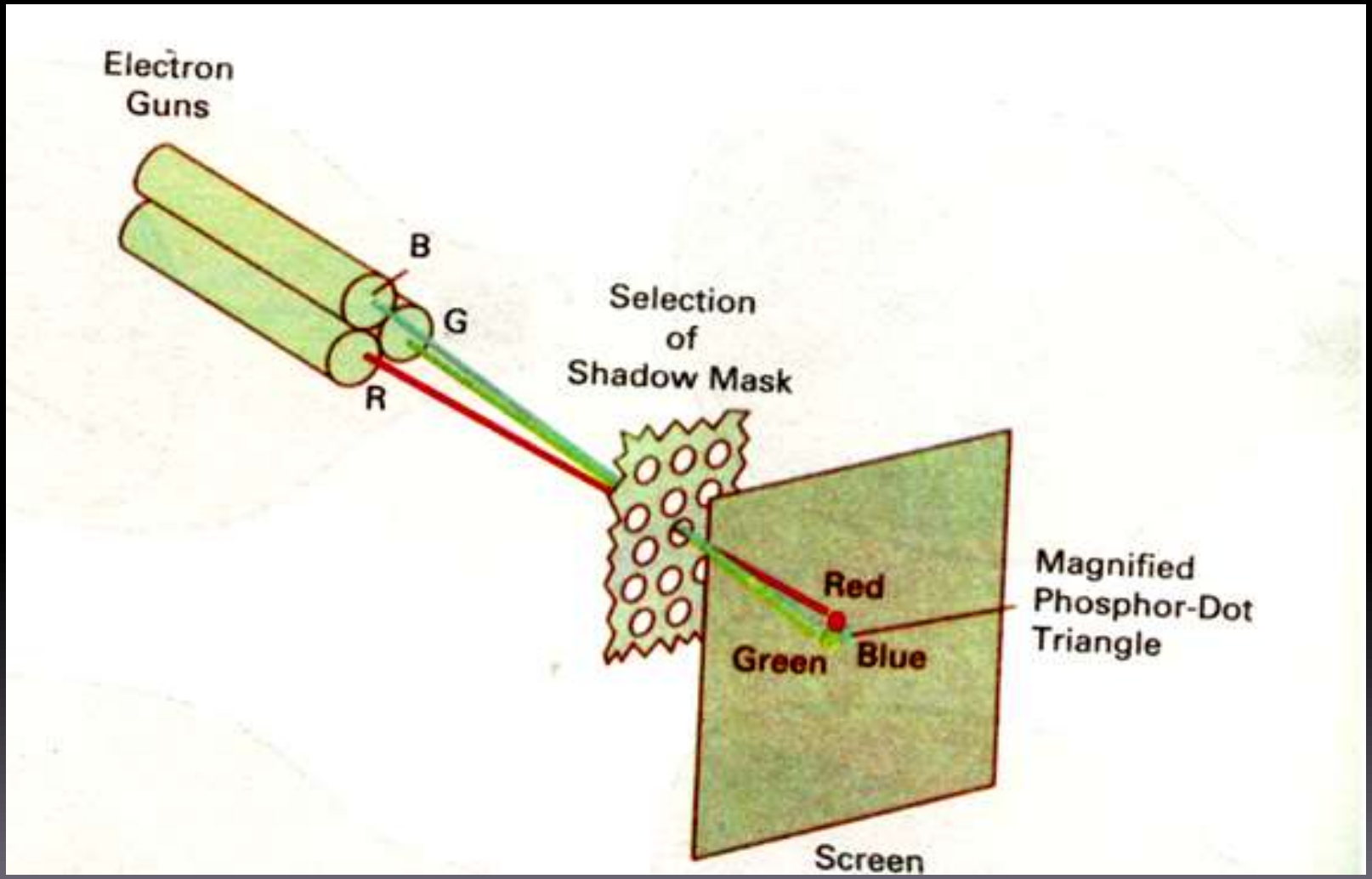
Color CRT Monitors

- A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light.
- By combining the emitted light from the different phosphors, a range of colors can be generated.
- The two basic techniques for producing colors in CRT are the beam penetration method and the shadow-mask method.

Beam penetration method

- This method for displaying color pictures has been used with random scan monitors.
- Two layers of phosphor, usually red and green, are coated onto the inside of the CRT screen, and the displayed color depends on how far the electron beam penetrates into the phosphor layers.
- A beam of slow electrons excites only the outer red layer.
- A beam of very fast electrons penetrates through the red layer and excites the inner green layer.
- At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow.
- The speed of the electrons, and hence the screen color at any point, is controlled by the beam-acceleration voltage.
- This method is inexpensive, but produces only four colors. Quality of picture is not good as other methods.

Shadow Mask method



- Shadow-mask methods are commonly used in raster-scan systems because they produce a much wider range of colors than the beam penetration method .
- A shadow mask CRT has three phosphor color dots at each pixel position.
- One phosphor dot emits a red light, another emits a green light, and the third emits a blue light.
- This type of CRT has three electron guns, one for each color dot, and a shadow mask grid just behind the phosphor coated screen.
- Figure shows the ***delta-delta*** shadow-mask method, commonly used in color CRT systems.
- Three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of holes aligned with the phosphor-dot patterns.
- When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small color spot on the screen.

- The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.
- Another configuration for the three electron guns is an ***in-line*** arrangement in which the three electron guns, and the corresponding red-green-blue dots on the screen, are aligned along one scan line instead of in a triangular pattern.
- This in-line arrangement of electron guns is easier to keep in alignment and is commonly used in high-resolution color CRTs.
- We obtain color variations in shadow-mask CRT by varying the intensity levels of the three electron beams.
- By turning off the red and green guns, we get only the color coming from the blue phosphor. Likewise, we can get different colors.

- Color CRTs in graphics systems are designed as RGB monitors.
- These monitors use shadow mask methods and take the intensity level for each electron gun directly from the computer system without any intermediate processing.
- High quality raster graphics systems have 24 bits per pixel in the frame buffer, allowing 256 voltage settings for each electron gun and nearly 17 million color choices for each pixel.
- An RGB color system with 24 bits of storage per pixel is generally referred to as a ***full-color system*** or ***true-color system***.

Simple Raster Display System

- The simplest and most common raster display system organization is shown in figure 4.18.
- A portion of the memory also serves as the pixmap.
- The video controller displays the image defined in the frame buffer, accessing the memory through a separate access port as often as the raster-scan rate dictates.
- In many systems, a fixed portion of memory is permanently allocated to the frame buffer.
- Some systems have several interchangeable memory areas.
- In the case where system can designate any part of memory for the frame buffer, the system may be organized as shown in figure 4.19.
- The simple raster display system organizations are used in many inexpensive personal computers, but has a number of disadvantages.

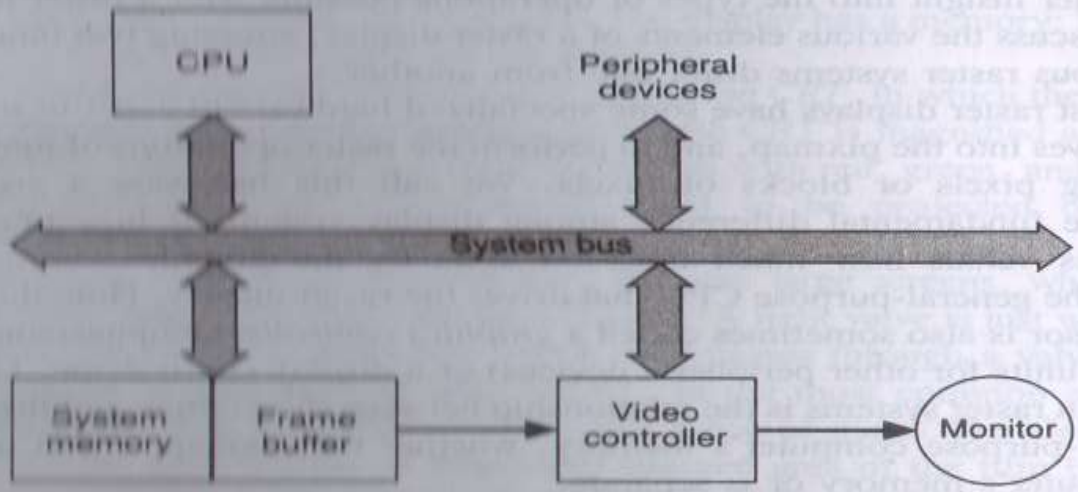


Fig. 4.18 A common raster display system architecture. A dedicated portion of the system memory is dual-ported, so that it can be accessed directly by the video controller, without the system bus being tied up.

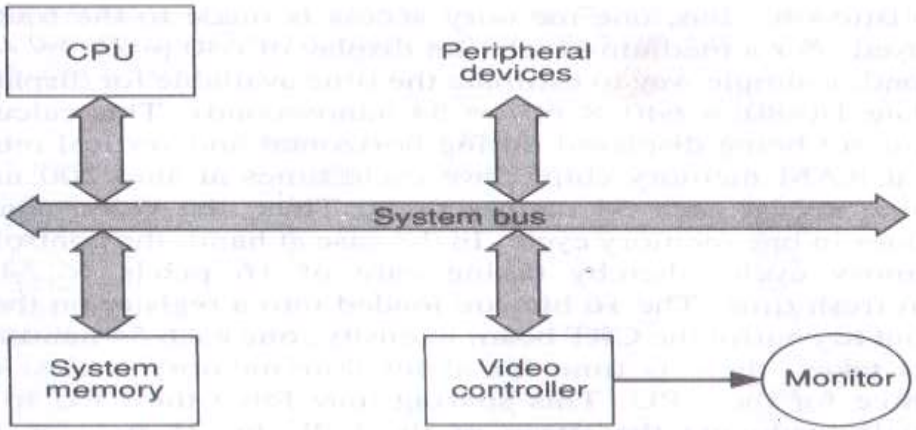


Fig. 4.19 A simple raster display system architecture. Because the frame buffer may be stored anywhere in system memory, the video controller accesses the memory via the system bus.

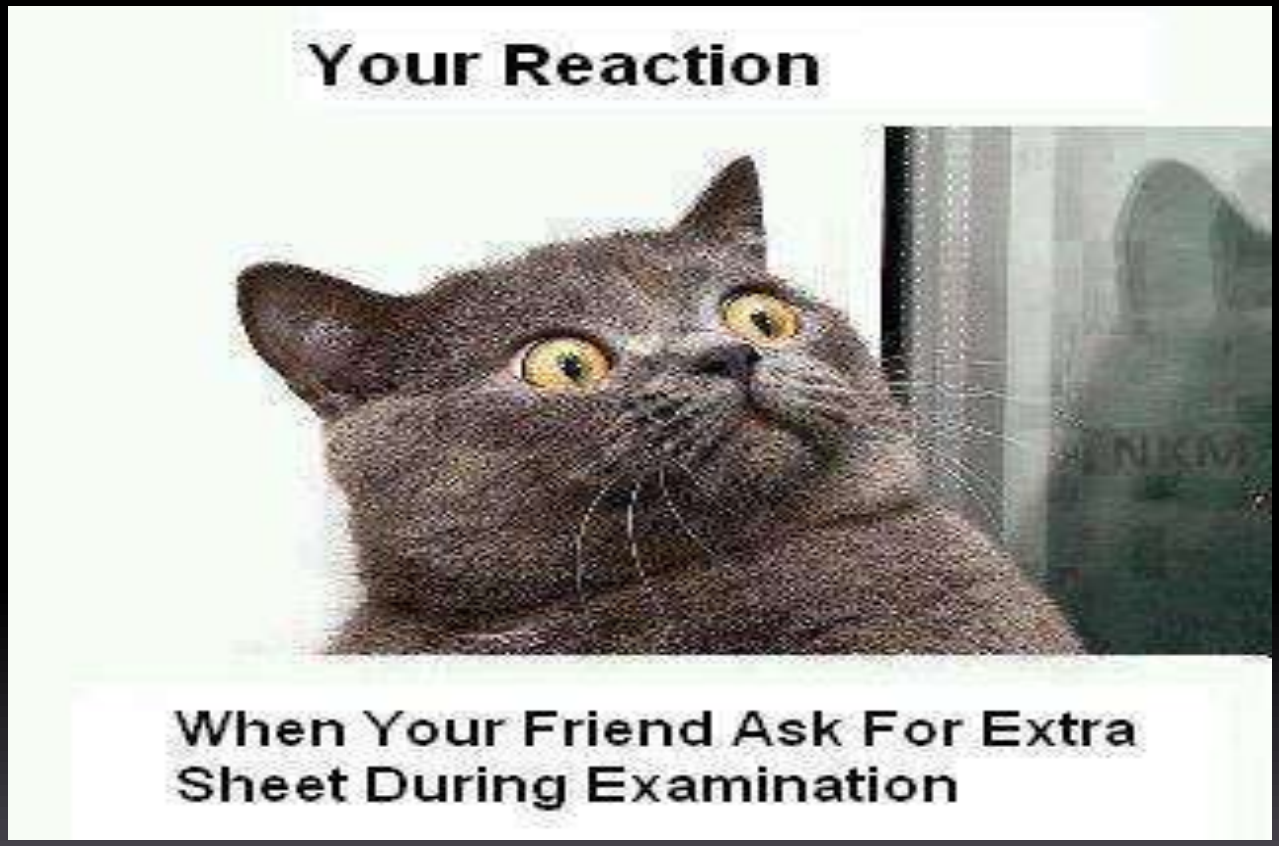
- In other words, a simple raster-display system contains a CPU, system bus, main memory, frame buffer, video controller, and CRT display.
- In such a system, the CPU performs all the modeling, transformation, and display computations, and writes the final image to the frame buffer.
- The video controller reads pixel data from the frame buffer in raster-scan order, converts digital pixel values to analog, and drives the display.

Problems in SRDS

- SRDS is inexpensive to build, but has a number of disadvantages.
- First, scan conversion in software is slow. For example, the (x,y) address of each pixel on a line must be calculated, then must be translated into a memory address consisting of a byte and bit-within-byte pair.
- Each of the individual steps is simple, each is repeated many times.
- The second disadvantage of this architecture is that as the refresh rate of the display increases, the number of memory accesses made by the video controller also increases, thus decreasing the number of memory cycles available to the CPU.
- The CPU slows down, especially with the architecture in fig 4.19. With the dual-porting of part of the system memory shown figure 4.18, the slowdown occurs only when the CPU is accessing the frame buffer, usually for scan conversion or raster operations.

Solutions

- A partial solution, suitable for low-resolution displays, is to place the frame buffer on an isolated bus. This allows video scanout to occur simultaneously with CPU operations that do not affect the frame buffer.
- Second partial solution is to take advantage of DRAM's page-mode feature. Since video scanout accesses memory locations in order, DRAM page misses are infrequent, so the frame buffer can run at almost twice its normal speed. We can also use dedicated Cache.
- Third approach is to duplicate the frame-buffer memory, creating a double-buffered system in which the image in one buffer is displayed while the image in the other buffer is computed.
- Video RAM can also be used.


Your Reaction
When Your Friend Ask For Extra Sheet During Examination

Coordinate Representations

- Graphics packages require **coordinate specifications** to be given with respect to Cartesian reference frames.
- Several different Cartesian reference frames are used to construct and display a scene.
- We can construct the **shape of individual objects** in a scene within **separate coordinate** reference frames called **Modeling coordinates** OR **Local Coordinates** OR **Master coordinates.**
- Once individual objects shapes have been specified, we can **place the objects into appropriate positions** within the scene using a reference frame called **World Coordinates** .

- Finally, the world-coordinate description of the scene is transferred to one or more output device reference frames for display. These display coordinate systems are referred to as **Device Coordinates** or **Screen coordinates** in case of a video monitor.
- Generally, a graphics system first converts world-coordinate positions to Normalized device coordinates, in the range from 0 to 1, before final conversion to specific device coordinates. This makes system independent of the various devices that might be used at a particular workstation.

$$(X_{mc},Y_{mc}) \rightarrow (X_{wc},Y_{wc}) \rightarrow (X_{nc},Y_{nc}) \rightarrow (X_{dc},Y_{dc})$$

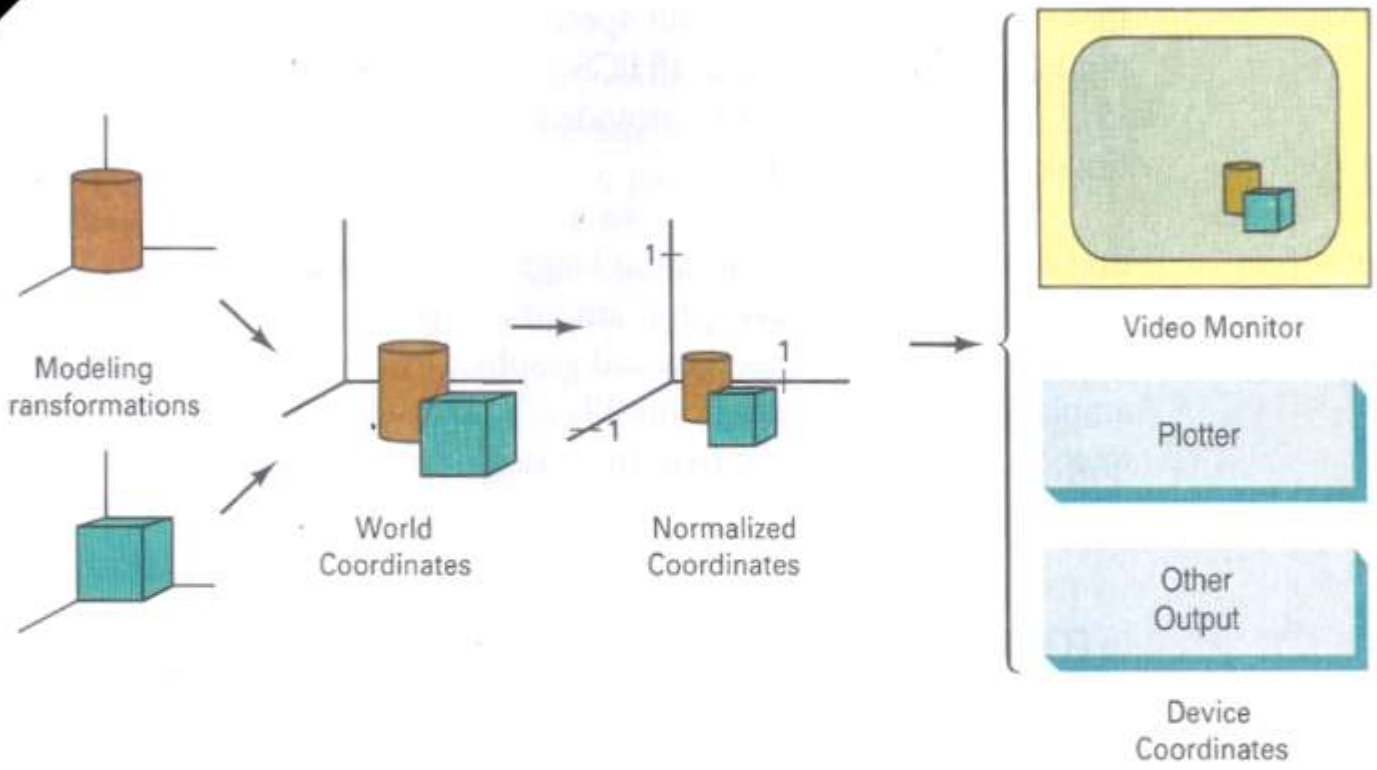


Figure 2-65
The transformation sequence from modeling coordinates to device coordinates for a two-dimensional scene. Object shapes are defined in local modeling-coordinate systems, then positioned within the overall world-coordinate scene. World-coordinate specifications are then transformed into normalized coordinates. At the final step, individual device drivers transfer the normalized-coordinate representation of the scene to the output devices for display.

Graphics Functions

- **Output primitive:** The most basic shape that can be printed as an output by the output device is called as output primitive. For e.g. Point, Line, Circle.
- **Attributes:** The properties of an output primitive such as color, width, height of text, etc. is called attribute.
- **Modeling transformation:** It is used for representation of modeling coordinates. They are used to construct a scene using object descriptions given in modeling coordinates.
- **Geometric transformation:** It is used for representation of world coordinates. We can change the size, position, or orientation of an object within a scene using it.

- **Viewing transformation:** It is used for representation of device coordinates. They are used to specify the view that is to be presented and the portion of the output display area that is to be used.
- **Input function:** The function which is used to create and draw the output primitive on the screen, is called input function. They are used to control and process the data flow from interactive devices like mouse, or joystick etc.
- **Control operations:** The operation performed on the output primitive to make any changes or make the control on it, is called control operation. A graphics package contains a number of tasks, such as clearing a display screen and initializing parameters which fall under the category of control operations.

Line Drawing Algorithms

- The Cartesian equation for a straight line is given by

  $y = m.x + b$ **-- (1)**

- Where x, y are co-ordinate points, m is the slope of line, and b is known as y intercept.
- Thus, value of slope for the line with two endpoints as

  (X_1, Y_1) and (X_2, Y_2) is given as:

  $m = Y_2 - Y_1 / X_2 - X_1$ **-- (2)**

  From eqn. (1), value of **$b = y_1 - mx_1$**
- Now the change in y is given by

  $\Delta y = m \Delta x$ **-- (a)**

  $\Delta x = \Delta y / m$ **--(b)**

- These equations form the basis for determining deflection voltages in analog devices.
- On raster systems, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations.
- That is, we must "sample" a line at discrete positions and determine the nearest pixel to the line at each sampled position.

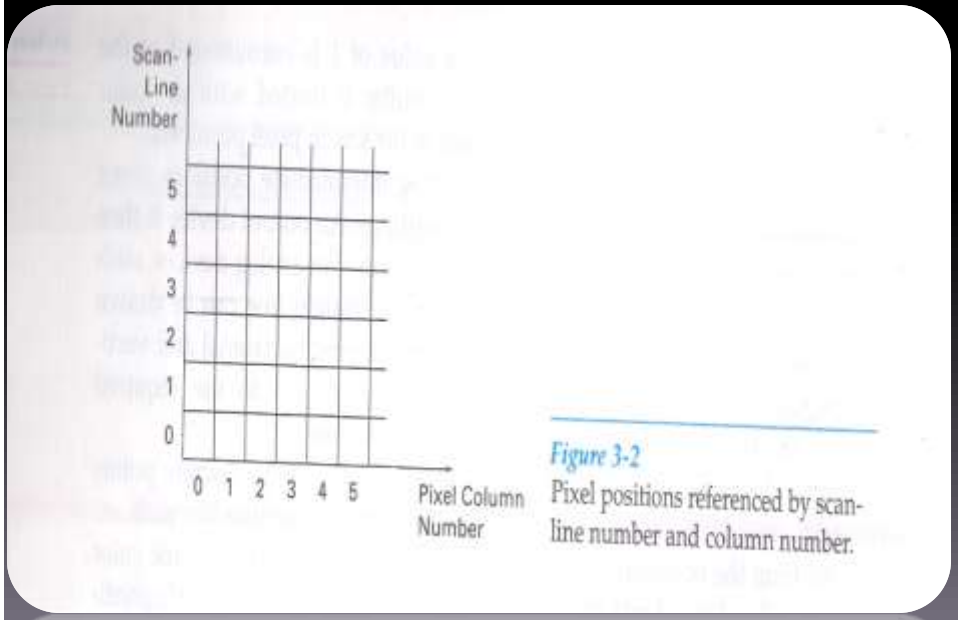


Figure 3-2
Pixel positions referenced by scan-line number and column number.

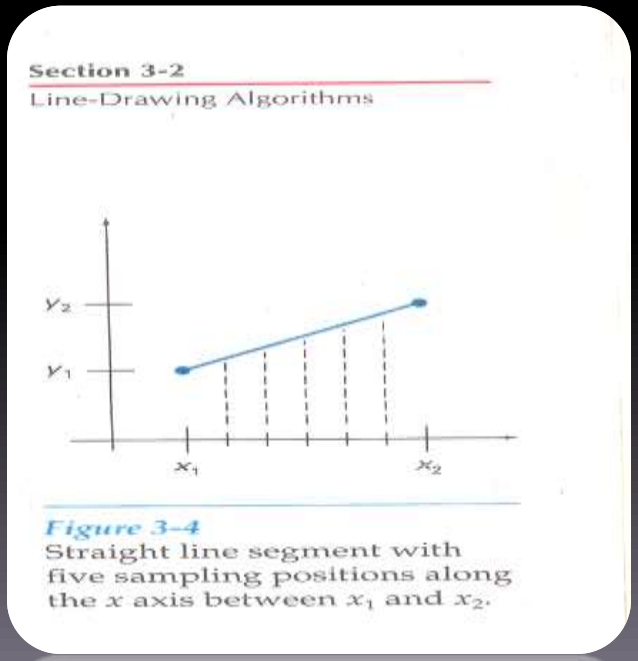
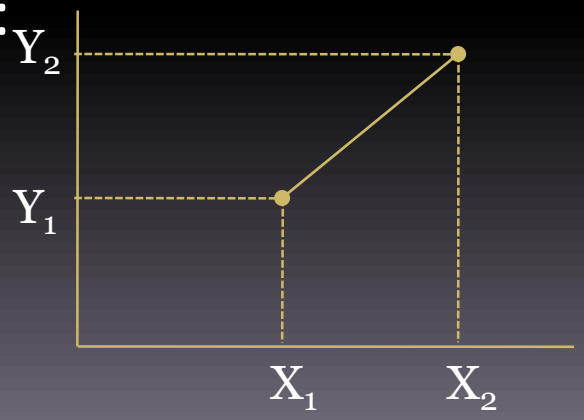


Figure 3-4
Straight line segment with five sampling positions along the x axis between x_1 and x_2.

DDA algorithm

- The Digital Differential Analyzer is a scan-conversion line algorithm based on calculating either **Δy** or **Δx**.
- We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate.
- The line to be drawn may have the slope as positive or negative.
- Positive slope means that values of X and Y are increasing as in below line:



Line with positive slope

- If the **value of slope is less or equal to 1**, we sample at unit x intervals and compute each successive y value as:

  $$Y_{k+1} = Y_k + m \qquad \text{-- (1)}$$

  Here, k starts from 1 and increases by 1 until endpoint is obtained. Since m can be any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.

- If the v**alue of slope is greater than 1**, we sample at unit y intervals and calculate each succeeding x values as:

  $$X_{k+1} = X_k + (1/m) \qquad \text{-- (2)}$$

  the above two equations are based on the assumption that lines are to be processed from the left endpoint to the right endpoint.

- If this processing is reversed, so that the starting endpoint is at the right, then either we have $\Delta x = -1$ and

$$Y_{k+1} = Y_k - m \qquad \text{-- (3)}$$

- Or, when the slope greater than 1, we have $\Delta y = -1$ and

$$X_{k+1} = X_k - (1/m) \qquad \text{-- (4)}$$

- Equations 1,2,3 and 4 can also be used to calculate pixel positions along a line with negative slope.
- If the absolute value of the slope is less than 1 and the start endpoint is at the left, we set $\Delta x = 1$ and calculate y values with equation(1).
- When the start endpoint is at the right, we set $\Delta x = -1$ and obtain y positions from equation(3).
- Similarly, when the absolute value of a negative slope is greater than 1, we use $\Delta y = -1$ and equation(4) or we use $\Delta y = 1$ and equation(2)

Algorithm

1: Accept two endpoint pixel positions, say ($\mathbf{X_a}$, $\mathbf{Y_a}$) and ($\mathbf{X_b}$, $\mathbf{Y_b}$)

2: **dx** = Xb - Xa ; **dy** = Yb - Ya [Find the difference of endpoints]

3: if abs(**dx**) > abs(**dy**) then

steps = abs(dx)

else **steps** = abs(dy)

4: **xIncrement** = dx / steps;

yIncrement = dy / steps [Find total intermediate points]

5: setPixel(ROUND(Xa),ROUND(Ya)) [Plot the first point]

6: Repeat step 7 **steps** times

7: **x** = x + xIncrement ;

y = y + yIncrement;

setPixel(ROUND(x), ROUND(y)) [Plot all points]

- The DDA algorithm is faster method for calculating pixel positions than the direct use of line equation.
- It eliminates the multiplication in line equation by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to step to pixel positions along the line path
- The access of roundoff error in successive additions of the floating-point increment, can cause the calculated pixel positions to drift away from the true line path for long line segments.
- Rounding operations and floating point arithmetic are still time consuming.
- We can improve the performance of the DDA algorithm by separating the increments m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts line using only incremental integer calculations that can be adapted to display circles and other curves.
- Following figures shows sections of a display screen where straight line segments are to be drawn.

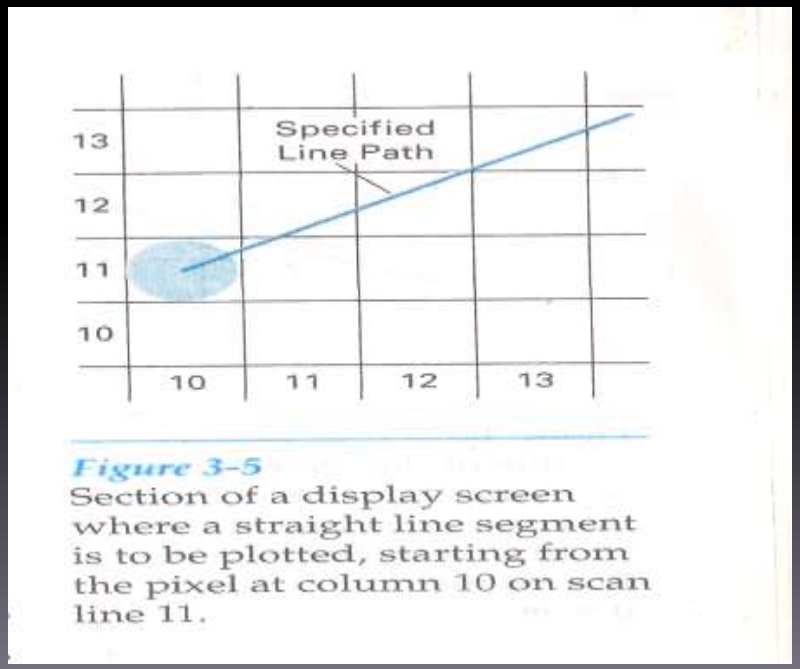


Figure 3-5
Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

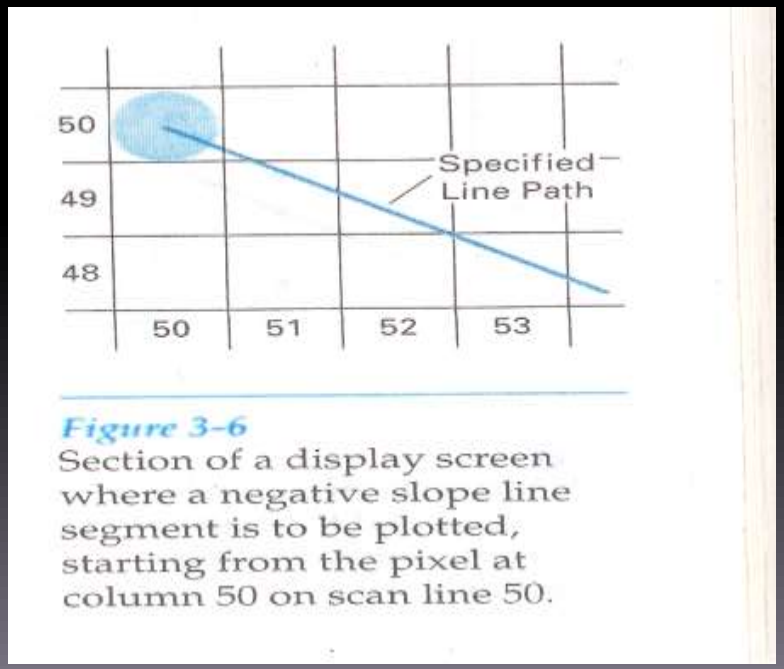


Figure 3-6
Section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

- The vertical axes show scan-line positions, and the horizontal axes identify pixel columns.
- Sampling at unit x intervals in these examples, we need to decide which of two possible pixel positions is closer to the line path at each sample step.
- To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.
- Pixel positions along a line path are then determined by sampling at unit x intervals.
- Starting from the left end-point (x_0,y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.
- Following figure shows the kth step in this process.

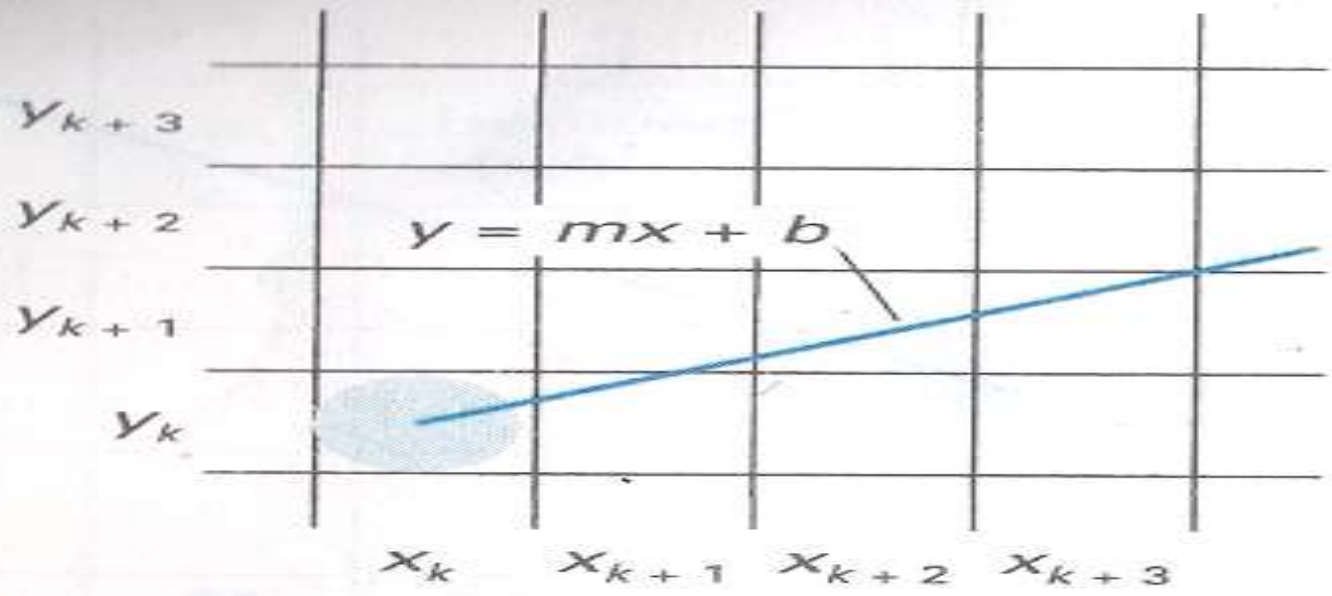


Figure 3-7
Section of the screen grid showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

- Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column x_{k+1}.
- Our choices are the pixels at positions $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$.
- At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path as d1 and d2.

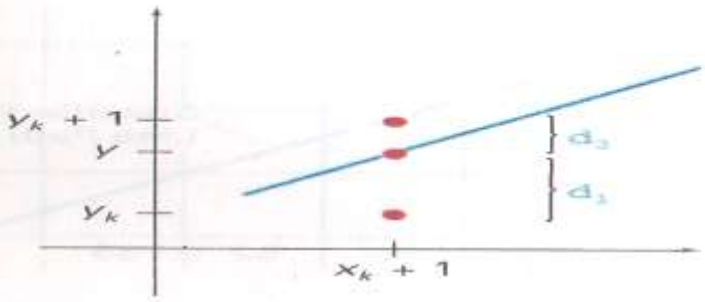


Figure 3-8
Distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

- The y coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as
- $y = m(x_k + 1) + b$ …(1)

Then

$d1 = y - y_k$

$d2 = (y_k + 1) - y$

$d1 - d2 = [m(x_k + 1) + b - y_k] - [y_k + 1 - m(x_k + 1) - b]$

$d1 - d2 = 2m(x_k + 1) - 2y_k + 2b - 1$ …(2)

- The decision parameter P_k for k^{th} step in line algorithm can be obtain by rearranging equation no. (2) so it involve only integer calculation.

$m = \Delta y / \Delta x$

where Δy = vertical separation and Δx = horizontal separation

$P_k = \Delta x (d1 - d2)$

$P_k = 2\Delta y\, x_k - 2y_k \Delta x + C$ …(3)

Where parameter C is constant and has value $2\Delta y + \Delta x$ (2b-1)

- The sign of P_k is the same as the sign of $d_1 - d_2$, since $\Delta x > 0$.
- Parameter C is independent of pixel position and will be eliminated in the recursive calculations for P_k.
- If the pixel at y_k is closer to the line path than the pixel at $y_k + 1$ ($d_1 < d_2$), then decision parameter P_k is negative.
- In that case, we plot the lower pixel; otherwise, we plot the upper pixel.
- Coordinate changes along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations.
- At step k + 1, the decision parameter is evaluated from eq. 3 as
- $P_{k+1} = 2\,\Delta y\, x_{k+1} - 2\, y_{k+1}\, \Delta x + C$

- Subtracting eq.3 from the preceding equation, we have,

$$P_{k+1} - P_k = 2\Delta y\,(x_{k+1} - x_k) - 2\Delta x\,(y_{k+1} - y_k)$$

- But, $x_{k+1} = x_k + 1$, so that,

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x\,(y_{k+1} - y_k) \quad \ldots. (4)$$

where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter P_k.

- This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoint of the line.
- The first parameter, P_0, is evaluated from eq. 3 at the starting pixel position (x_0, y_0) and with m evaluated as $\Delta y/\Delta x$:

$$P_0 = 2\Delta y - \Delta X \quad \ldots. (5)$$

- We can summarize Bresenham line drawing for a line with a positive slope less than 1 in the following listed steps.
- The constants **2 Δy and 2 Δy - 2 Δx** are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these two constants.

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test: If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 Δx times.

$P_k = \Delta X (d1 - d2)$

$= \Delta X (2\, m(X_k + 1) - 2\, Y_k + 2b - 1)$

$= \Delta X (2mX_k + 2m - 2\, Y_k + 2b - 1)$

$= \Delta X (2mX_0 + 2m - 2\, Y_0 + 2b - 1)$ [put k=0]

$= \Delta X (2(mX_0 + b - Y_0) + 2m - 1)$

$= \Delta X(2m - 1)$ (Because $mX_0 + b - Y_0 = 0$)

$= \Delta X (2\, \Delta Y / \Delta X - 1)$

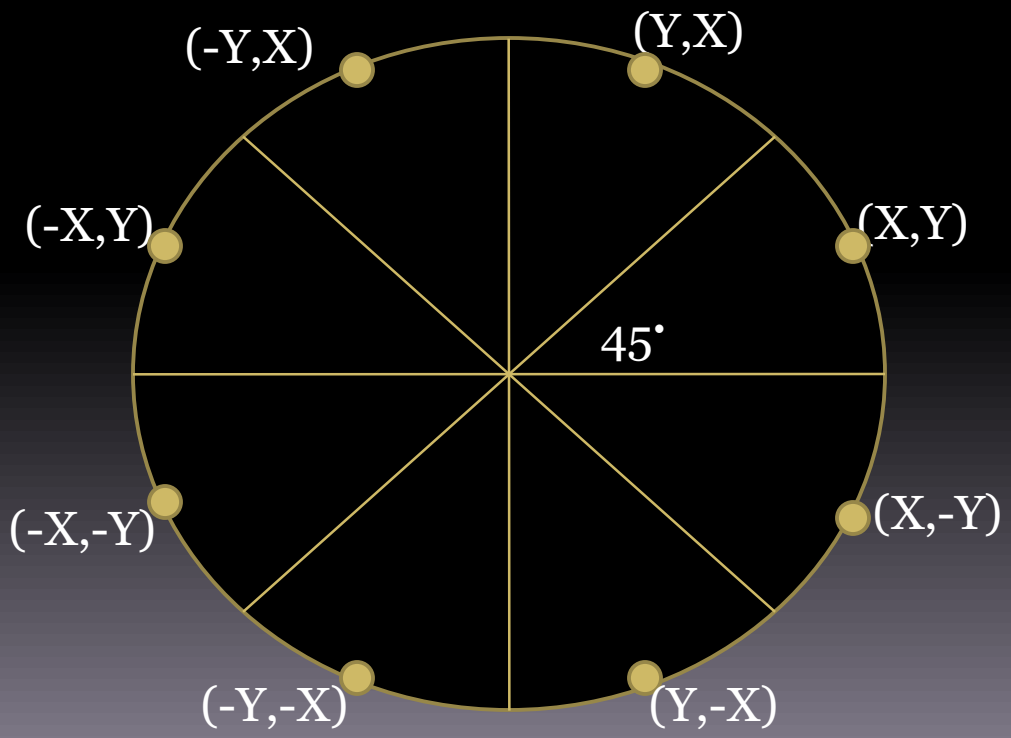
$P_0 = 2\, \Delta y - \Delta X$ [decision parameter]

Circle Generating Algorithm

- Circle is a set of points that all are at a given distance ‘r’ from center position (Xc, Yc). If ‘r’ is the radius and (Xc, Yc) are central point then Cartesian formula of circle is

$$(X - Xc)^2 + (Y - Yc)^2 = r^2$$

- **Symmetric nature of the circle:**

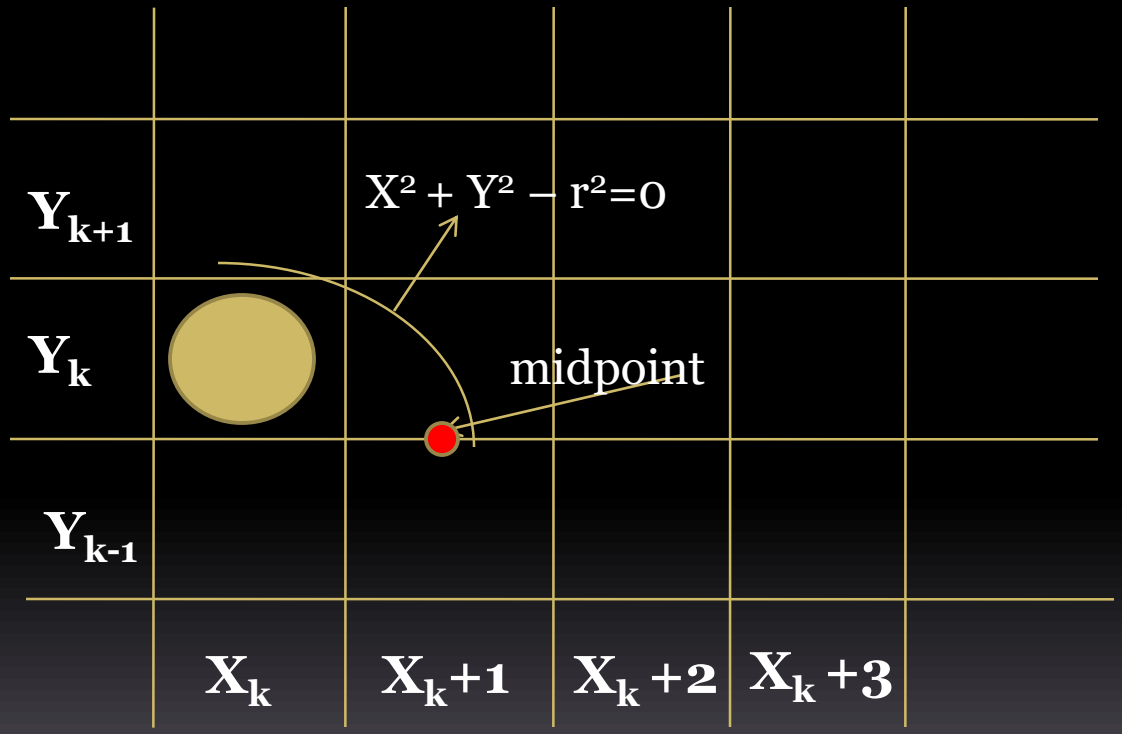


Circle Generating Algorithm

- Bresenham's line algorithm for raster display is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.
- A method for direct distance comparison is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

Midpoint Circle Algorithm

- Assumption: Assume that origin of circle is (0,0)



- To apply the midpoint method, we define a circle function

$$f_{circle}(x,y) = x^2 + y^2 - r^2 \quad .....(1)$$

- If f(X,Y) =< 0, pixel is inside the circle boundary
  = 0, pixel is on the circle boundary
  > 0, pixel is outside the circle boundary … (2)
- Here assume that we have already find out pixel position for X_k and Y_k
- Now find the pixel for X_k+1 column, y – coordinate may be either Y_k or Y_{k-1}.
- But this algorithm is known as midpoint algorithm so evaluate decision parameter between Y_k and Y_{k-1} scanned lined

- Find out decision parameter for X_k+1column , coordinate is $(X_k+1, Y_k - ½)$

  Therefore ,

  $$P_k = (X_k+1)^2 + (Y_k - ½)^2 - r^2 \quad \ldots(3)$$

- Now if $P_k < 0$ then midpoint is inside the circle and pixel on scan line Y_k is closer to the circle boundary. So select upper pixel. i.e. Y_k
- Now if $P_k \geq 0$ then mid point is on the circle boundary or outside the circle boundary. So select lower pixel. i.e.Y_{k-1}

Now find successive decision parameter (P_{k+1})

Replace k as k+1 in eq. no (3)

$$P_{k+1} = (X_{k+1}+1)^2 + (Y_{k+1} - ½)^2 - r^2$$

$$P_{k+1} = (X_k+2)^2 + (Y_{k+1} - ½)^2 - r^2 \quad \ldots(4)$$

Now,

$$P_{k+1} - P_k = [(X_k+2)^2 + (Y_{k+1} - ½)^2 - r^2] - [(X_k+1)^2 + (Y_k - ½)^2 - r^2]$$

$$P_{k+1} = P_k + 2(X_k+1) + (Y^2_{k+1} - Y^2_k) - (Y_{k+1} - Y_k) + 1 \quad \ldots(5)$$

Where Y_{k+1} is either Y_k or Y_{k-1}

- if $P_k < 0$ then midpoint is inside the circle and pixel on scan line Y_k is closer to the circle boundary. So select upper pixel. i.e. Y_k

Then from eq. no (5)

$$P_{k+1} = P_k + 2(X_k + 1) + (Y^2_{k+1} - Y^2_k) - (Y_{k+1} - Y_k) + 1$$

$$P_{k+1} = P_k + 2(X_k + 1) + (Y^2_k - Y^2_k) - (Y_k - Y_k) + 1$$

$$= P_k + 2(X_k + 1) + 1$$

$$\mathbf{P_{k+1} = P_k + 2X_k + 3}$$

- Now if $P_k \geq 0$ then mid point is on the circle boundary or outside the circle boundary. So select lower pixel. i.e.Y_{k-1}

Then from eq. no (5)

$$P_{k+1} = P_k + 2(X_k + 1) + (Y^2_{k+1} - Y^2_k) - (Y_{k+1} - Y_k) + 1$$

$$P_{k+1} = P_k + 2(X_k + 1) + (Y_k - 1)^2 - Y^2_k) - (Y_k - 1 - Y_k) + 1$$

$$\mathbf{P_{k+1} = P_k + 2(X_k - Y_k) + 5}$$

- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$.

First decision parameter coordinate at (0,r)

$P_0 = f_{circle}(1, r - \frac{1}{2})$

$P_0 = (0 + 1)^2 + (r - \frac{1}{2})^2 - r^2$

$P_0 = 5/4 - r$

but Bresenham algorithm involve only integer value

$$P_0 = 1 - r \quad \ldots(6)$$

Midpoint Circle Algorithm

1. Input radius **r** and circle center **(Xc, Yc)**, and obtain the first point in circumference of a circle centered on origin as

   $$(Xo, Yo) = (0, r)$$

2. Calculate the initial value of decision parameter as

   $$Po = 5/4 - r \text{ or } Po = 1 - r$$

3. At each X_k position, starting at k = O, perform the following test:

   If $P_k < 0$, the next point along circle centered on (0,0) is (X_{k+1}, Y_k) and

   $$P_{k+1} = P_k + 2X_k + 3 \text{ or } P_{k+1} = P_k + 2X_{k+1} + 1$$

   Otherwise, the next point is (X_{k+1}, Y_{k-1}) and

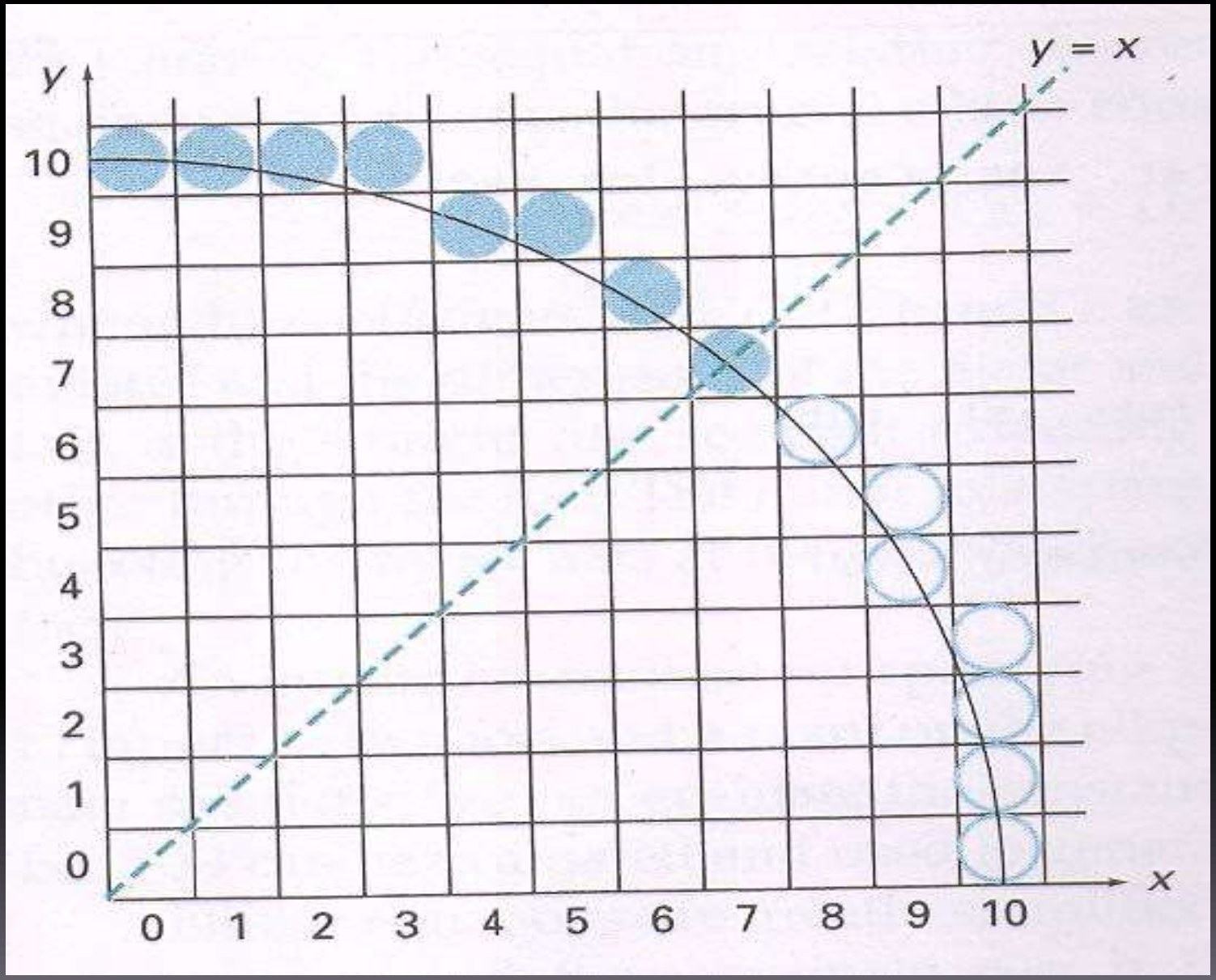
   $$P_{k+1} = P_k + 2(X_k - Y_k) + 5 \text{ or } P_{k+1} = P_k + 2X_{k+1} + 1 - 2Y_{k+1}.$$

   where $2X_{k+1} = 2X_k + 2$ and $2Y_{k+1} = 2Y_k - 2$.

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c,y_c) and plot the coordinate values:

   $X = X + Xc$ and $Y = Y + Yc$

6. Repeat steps 3 through 5 until X>=Y.

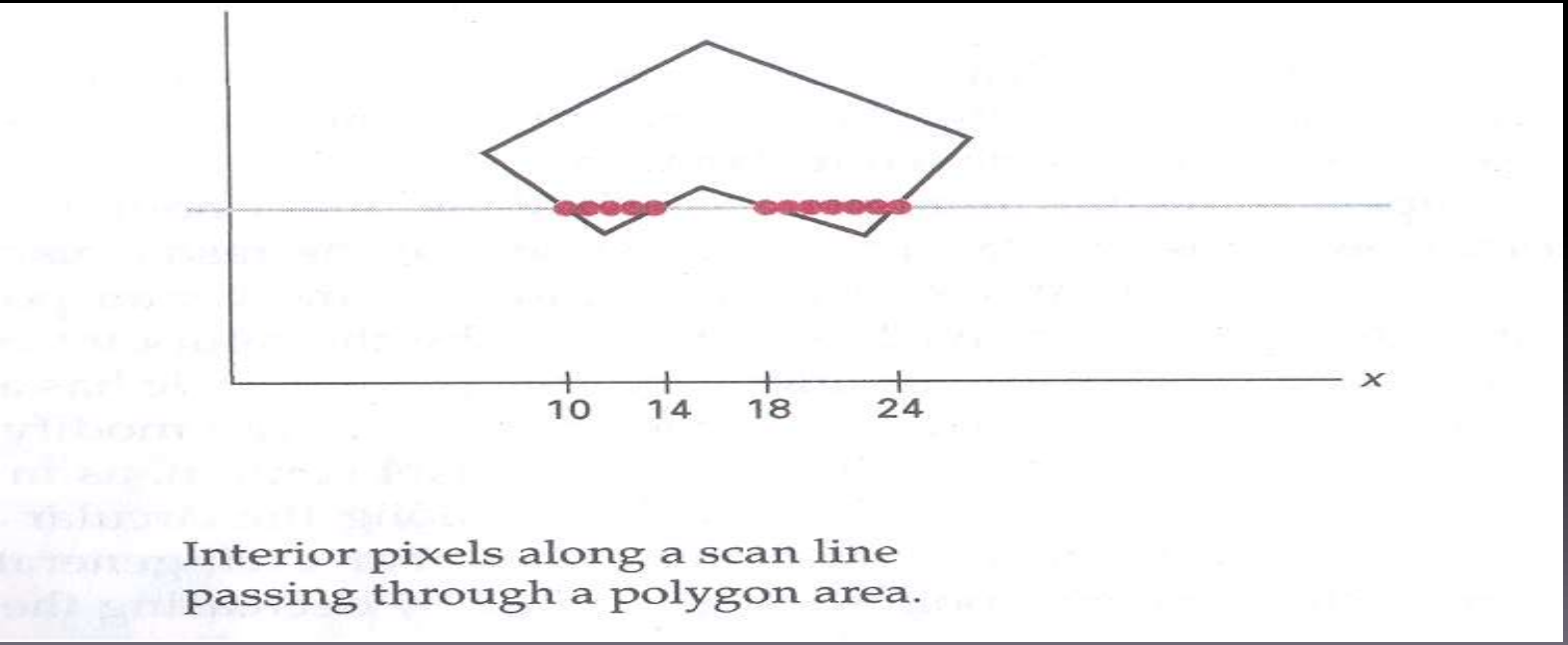

y = x
y
10
9
8
7
6
5
4
3
2
1
0
x
0 1 2 3 4 5 6 7 8 9 10

Filled-Area Primitives

- A standard output primitive in general graphics packages is a solid-color or patterned polygon area.
- Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries.
- There are two basic approaches to area filling on raster systems.
- One way to fill an area is to determine the overlap intervals for scan lines that cross the area.
- Another method for area filling is to start from a given interior position and paint outward from this point until we encounter the specified boundary conditions.

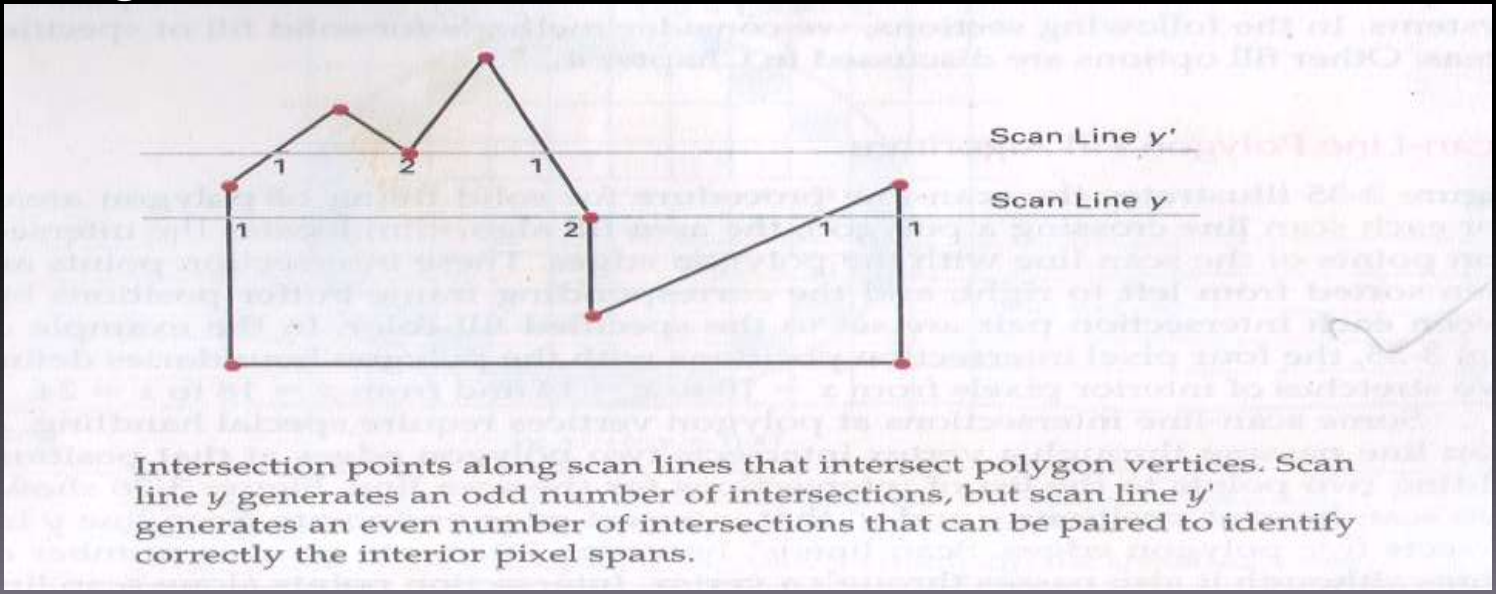
Scan-Line Polygon Fill Algo.

- For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges.
- These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color.



Interior pixels along a scan line passing through a polygon area.

- A scan line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersections for the scan line.
- Following figure shows two scan lines at positions y and y′ that intersect edge endpoints.
- Scan line y intersects five polygon edges. Scan line y′ intersects an even number of edges although it also passes through a vertex.



Intersection points along scan lines that intersect polygon vertices. Scan line *y* generates an odd number of intersections, but scan line *y'* generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

- Intersection points along scan line y' correctly identify the interior pixel spans.
- But with scan line y, we need to do some additional processing to determine the correct interior points.
- For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line.
- But, for scan line y', the two intersecting edges are both above the scan line.
- Thus, the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line.
- We can identify these vertices by tracing around the polygon boundary either in clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next.
- If the endpoint y values of two consecutive edges monotonically increase or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex.
- Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.

Inside-Outside Tests

- Area-filling algorithms and other graphics processes often need to identify interior regions of objects.
- In elementary geometry, a polygon is usually defined as having no self-intersections. For example, triangle, rectangles, octagons.
- The component edges of these objects are joined only at the vertices, and otherwise the edges have no common points in the plane.
- Identifying the interior regions of standard polygons is generally a straightforward process. But in most graphics applications, we can specify any sequence for the vertices of a fill area, including sequences that produce intersecting edges, as in following figure.

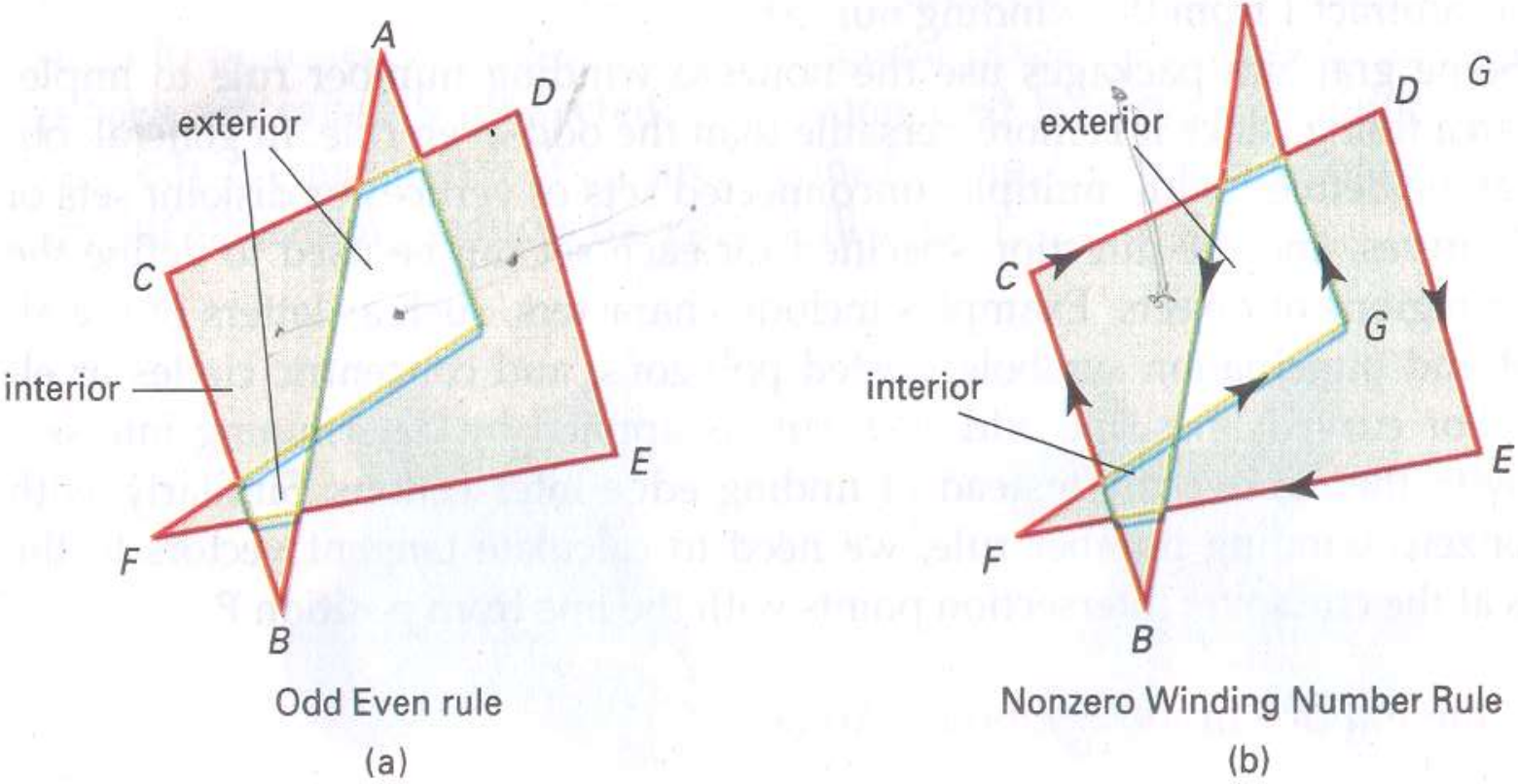


Figure 3-40
Identifying interior and exterior regions for a self-intersecting polygon.

Odd-Even Rule

- It is also known as **odd parity rule** or **even parity rule**
- **Rules:**

  1. Draw a line from the given point **P** to the point, which is outside the polygon co-ordinates.

  2. If the total no. of edges that cross (intersect) this line is **odd**, then point P is **interior** point. If it is **even**, then P is **exterior**.

- Here, we should take care that the line do not pass through the vertices. The **scan-line polygon fill** is example of odd-even parity rule.

Nonzero Winding number rule

- In **non-zero winding no.**, we find the no. of polygon edges which are winding (surrounding) the given point in counter-clockwise direction.
- The count obtained is known as **winding number**.
- The point whose winding no. is non-zero is considered as interior point, and otherwise, it is exterior.

1. Take one counter point, initialize it zero.
2. Draw a line from any position P to a distant point.(Line must not pass through any vertices.)
3. Process all edges that across the line in each direction.

   if direction is from right → left then +1

   if direction is from left → right then -1
4. The final value of the winding number, after all edge crossings have been counted, determines the relative position of P.

Scan-Line Fill of Curved Boundary Areas

- In general, scan-line fill of regions with curved boundaries requires more work than polygon filling, since intersection calculations now involve nonlinear boundaries.
- Another approach to area filling is to start at a point inside a region and paint the interior outward toward the boundary.
- If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered.
- This method is called the boundary-fill algorithm, is particularly useful in interactive painting packages, where interior points are easily selected.
- If solid-color is required (no border), Then user should choose the fill color same as the boundary color.

- A boundary Fill procedure accept
  1. Boundary Color
  2. Coordinates of interior point
  3. Fill color
- The boundary fill algorithm takes input the (X,Y) position of the interior point, the fill color and the boundary color.
- It start by painting the neighboring pixels with fill color if they do not have boundary color.
- If pixels are found with boundary color, it will stop.

There are two methods for filling area:

- 4 connected which uses only left, right, top and bottom pixels.
- 8 connected which uses 8 pixels surrounding.

The area filled by 8-connected is correct fill, while previous one uses partial fill.

Procedure

```
void boundaryfill4(int x, int y, int fill, int boundary)
{
     int current;
     current = getpixel(x,y);
     if ((current !=fill) && (current!=boundary))
   {
            setcolor(fill);
            setpixel(x,y);
            boundaryfill4(x,y+1,fill,boundary);
            boundaryfill4(x,y-1,fill,boundary);
            boundaryfill4(x+1,y,fill,boundary);
            boundaryfill4(x-1,y,fill,boundary);
   }
}
```

Drawback of Boundary-Fill Algorithm

1. If the color of any one of the interior point is same as fill color at that time this algorithm will not fill all interior pixels. Because the algorithm checks next pixels both for boundary color and fill color.

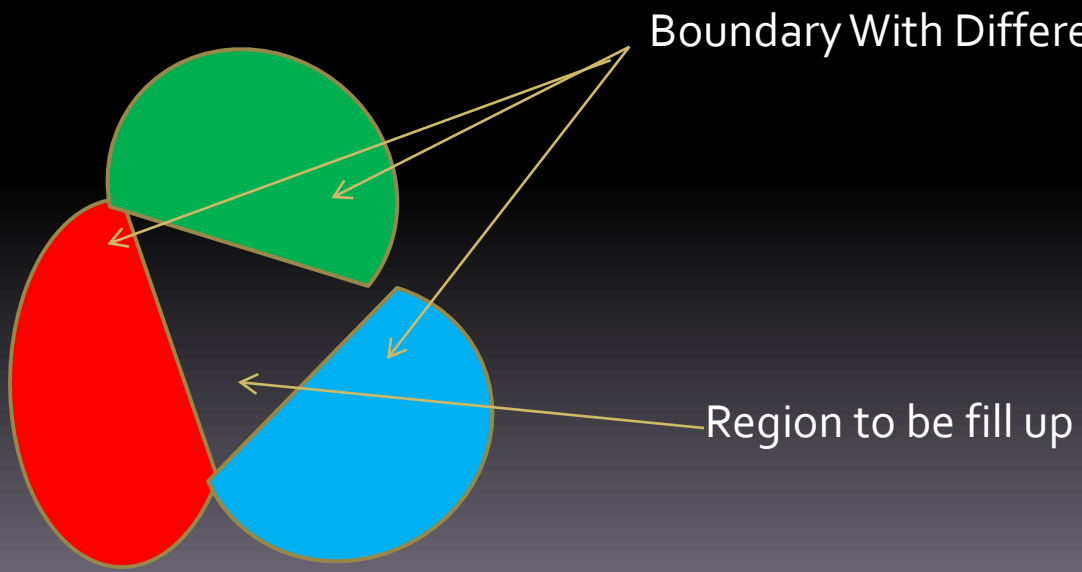
Solution: Change the color of any interior pixels that are initially set to the fill color before applying the boundary-fill procedure.

2. It requires large number of PUSH and POP operation.

Solution: Fill horizontal pixels spans across scan line. So that we have to push beginning position of the horizontal scan line.

Flood Fill algorithm

- It is used when boundary color is more than one color.
- We can paint such area by replacing specified interior color instead of searching for boundary color value. This approach is known as Flood-Fill algorithm.



- We start from a specified interior point(x,y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
- If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color.
- Using 4-connected or 8-connected approach, we then step through pixel position until all interior points have repainted.

Procedure

```
void floodfill4(int x, int y, int fillcolor, int oldcolor)
{   if (getpixel(x,y) == oldcolor)
    {    setcolor(fillcolor);
         setpixel(x,y);
         floodfill4(x,y+1,fillcolor,oldcolor);
         floodfill4(x,y-1, fillcolor,oldcolor);
         floodfill4(x+1,y, fillcolor,oldcolor);
         floodfill4(x-1,y, fillcolor,oldcolor);
    }
}
```

Character Generation

- There are various types of characters that we can display.
- The overall design of set of characters is called **typeface.** There are hundreds of typefaces available today. They are also referred as **fonts.**
- Originally, the term font referred to a set of cast metal character forms in a particular size and format, such as 10-point Courier Italic
- Typefaces or Fonts can be classified mainly into two types:

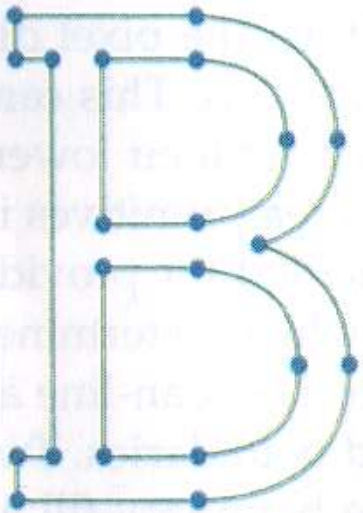
  Serif and **Sans serif**.
- **Serif font** consists of the accents (small lines) for each character, while **sans serif** does not.

- Serif type is generally more readable; that is, it is easier to read in longer blocks of text. On the other hand, the individual characters in sans-serif type are easier to recognize.
- The character can be represented in **two ways**.
  1. Bit-mapped font
  2. Outline font
- **Bitmap font** uses the frame buffer to store the value for point to be plotted as 1, and others as O.
- It is the simple method for representing character shape in particular font.
- It uses rectangular grid patterns to represent the character shape.
- Bit –mapped fonts are simplest to define and display.
- The character grid only needs to be mapped to a frame buffer position.
- Bitmaps fonts requires more space because all character must be stores in a font cache.

- It is possible to generate different sizes and variations, such as bold and italic, from one set, but produced result is not good.
- **Outline font**: It is second more flexible scheme to describe character shape using straight line and curve sections.
- The outline of character must be filled using the scan-line fill procedure.
- Outline fonts required less storage since each variation do not required individual font cache.
- We can produce bold, italic or different sizes by manipulating the curve definitions for the character outlines.
- But, It takes more time to process the outline fonts because they must be scan converted into frame buffer.

1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

(a)



(b)

Figure 3-48

The letter B represented in (a) with an 8 by 8 bilevel bitmap pattern and in (b) with an outline shape defined with straight-line and curve segments.

Attributes of Output Primitives

- **Attribute Parameter:** Any parameter that affects a primitive is to be displayed is known as attribute parameter.

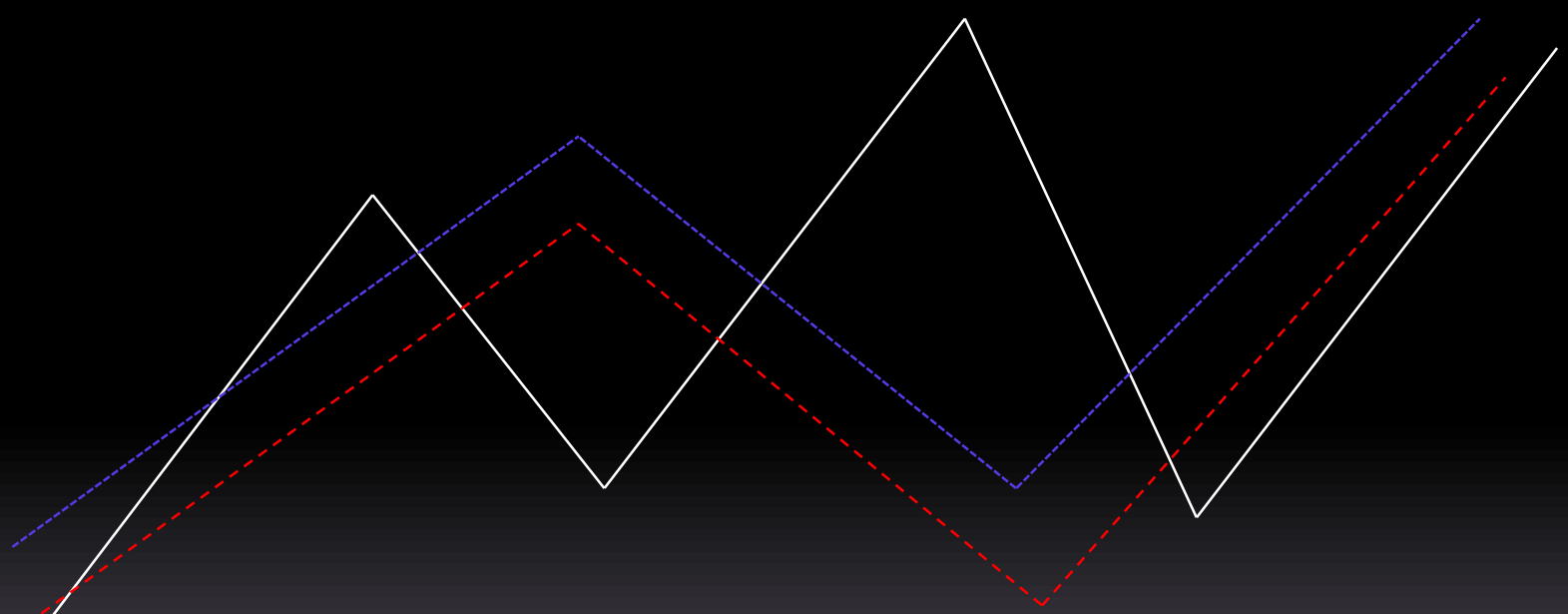
- With GKS and PHIGS standards, attributes settings are accomplished with separate functions that updates a system attribute list.
- **Four types of attributes:**
  1. **Line Attribute**
  2. **Color And Grayscale Level Attribute**
  3. **Area Fill Attribute**
  4. **Character Attributes**

Line Attribute

- There are three basic Line Attributes :

1. **Line type (Solid, Dashed, Dotted)**
2. **Line width**
3. **Line color**

1. Line Types



2. Line Width :

- The implementation of line width option depends on the capabilities of the output devices.
- In Raster Scan, a standard width is generated with single pixel at each sample position.
- In raster scan, the lines of width> 1 are drawn by plotting the additional pixel which are adjacent to the line.
- If the width is 2, the line with (x,y) values is drawn with one more line with (x,y+1) values.
- If lw is greater then 2, we select alternatively, the lines to right and left of the standard line.

Problems:

1. The horizontal and vertical lines can be perfectly drawn, but the diagonal lines will be drawn thinner.
2. Another problem is that the shape of line at the ends is horizontal or vertical regardless of the slope of the line.

Solution:

We can adjust the shape of line ends by using the line caps.

- **Line caps:** Three types of line caps.
  1. **Butt cap**
  2. **Round cap**
  3. **Projection square cap**

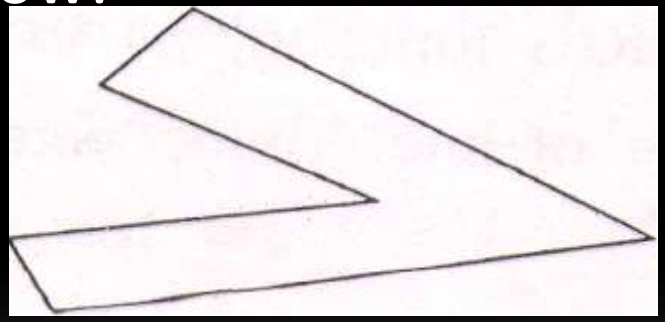
Butt cap: Butt cap is obtained by adjusting the end position of the component parallel lines, so that the thick line is displayed with square ends that are perpendicular to the line path.

Round cap : Obtained by adding a filled semicircle to each butt cap. The circular arcs are centered on the line endpoints and have a diameter equal to the line thickness**.**

Projecting square cap: Obtained by extending the line, and adding butt cap That are positioned one-half of the line width beyond the specified endpoints.

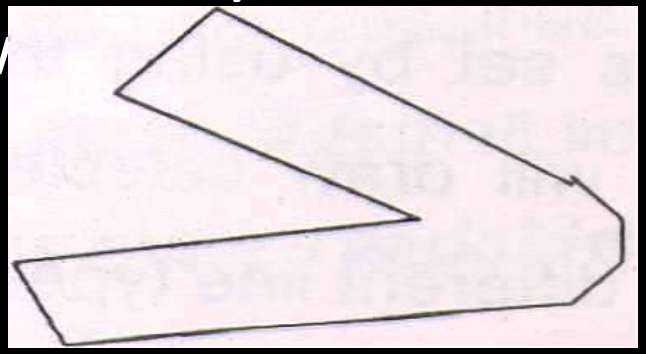
- Displaying thick lines using horizontal and vertical pixel spans, leaves pixel gaps at the boundaries between lines of different slopes where there is a shift from horizontal spans to vertical spans.
- When two thick lines are intersecting, then also we may want to set the join into required shape. The **three types of joins** are as below:

  1. Miter join
  2. Round join
  3. Bevel join

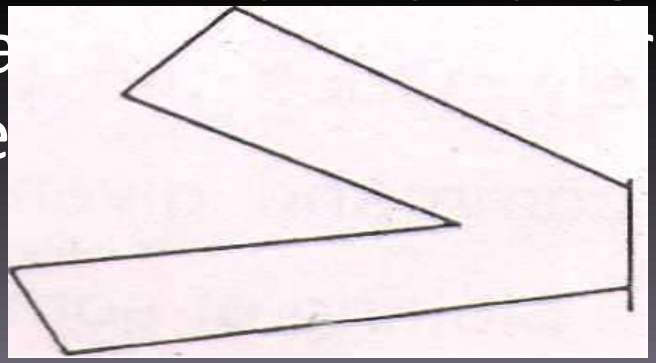


1. Miter join: Accomplished by extending the outer boundary of each of the two line until they meet.

2. Round join: It is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the line-w



3. Bevel join: It is generated by displaying the line segment with butt ca iangular gap where the segme



3. **Line color :** The number of color options depends on the number of bit per pixel in the frame buffer.

Color and Grayscale level Attribute

Color Level Attribute:

- Different color and intensity levels can be provided to the user, depending on the design objectives and capability of a particular system.
- The color options are numerically coded with the value ranging from 0 to positive integer.
- In color raster scan, number of colors available depends on the storage capacity per pixel in the frame buffer.
- 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.
- **Minimum number** of colors can be provided by using **3 bits per pixel**, each of which represents the presence of R, G and B, a frame buffer of which is as below.

The 8-color codes for a 3-bit/pixel frame buffer				
Color	Stored color values in the frame buffer			Displayed Color
	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

Grayscale Level Attribute:

- The monitors which do not have capability of producing colors, uses the different shades of **gray, or *grayscale*** for displayed output primitive.
- Color functions can be used in an application program to set the shades of gray or grayscale for display primitives.
- Numeric values over the range from 0 – 1 can be used to specify gray scale level, which are then converted into appropriate binary code for storage in the raster.
- If we use 2 bits per pixel, there are four possible colors as shown in the table.

Intensity	Binary Code	Color Value	Display Color
0.0	00	0	Black
0.33	01	1	Dark Gray
0.67	10	2	Light Gray
1.0	11	3	White

Area-Fill Attributes

- Used for filling a defined region include an option for choice between an solid color or a patterned fill.
- It also include choice between a 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.
- Areas can be painted using various brush styles, colors, and transparency parameters.

Three types of attributes:

1. **Fill Style**
2. **Pattern Fill**
3. **Soft Fill**

1. Fill Style :

Areas are displayed with three basic fill styles:

i. Hollow with a color border
ii. Filled with a solid color
iii. Filled with a specified pattern or design.

2. Pattern Fill :

In filling the area with a pattern, we may want to start pattern with some specific point, the pattern is then replicated in the x and y direction until the defined area is covered by non-overlapping copies of the pattern array. This procedure is k**nown as *tiling*** (like putting tiles from left top corner of room, or from centre).

3. Soft Fill :

- Modified boundary-fill and flood-fill procedures that are applied to repaint areas so that the fill color is combined with the background colors are referred to as soft-fill or tint-fill algorithms.
- One use for these fill methods is to soften the fill colors at object borders that have been blurred to antialias the edges.
- Another is to allow repainting of a color area that was originally filled with a semitransparent brush, where the current color is then a mixture of the brush color and the background colors “behind” the area.

- The linear soft-fill algorithm repaints an area that was originally painted by merging a foreground color F with a single background color B, where $F \neq B$.
- Assuming we know the values for F and B, we can determine how these colors were originally combined by checking the current color contents of the frame buffer.
- The current RGB color P of each pixel within the area to be refilled is some linear combination of F and B.

$$P = tF + (1 - t)B$$

where the "transparency" factor t has a value between 0 and 1 for each pixel.

e.g. for t=0.2 , the new color is combination of 20% of F + 80% of B

- For values of t less than 0.5, the background color contributes more to the interior color of the region than does the fill color.

- Above Vector equation holds for each RGB component of the colors, with

P = (Pr, Pg, Pb) , F = (Fr, Fg, Fb) , B = (Br, Bg, Bb) ,

We can thus calculate the value of parameter t using one of the RGB color components as …

$$t = \frac{P_k - B_k}{F_k - B_k}$$

Where, k = R, G, or B, intensity of any one color. i.e. take highest intensity value ; and F_k is not equal to B_k.

- similar soft-fill procedures can be applied to an area whose foreground color is to be merged with multiple background color areas, such as checker board pattern.

Two Background colors B_1 and B_2 are mixed with Foreground color F, the resulting pixel color P is

$$\mathbf{P} = t_0\mathbf{F} + t_1\mathbf{B_1} + (1 - t_0 - t_1)\mathbf{B2}.$$

Where the sum of the coefficients t_0, t_1 and $(1-t_0-t_1)$ on the color terms must equal 1 .

$t_0 = \dfrac{P_k - B_{1k}}{F_k - B_{1k}}$	$t_1 = \dfrac{P_k - B_{2k}}{F_k - B_{2k}}$

Two dimensional Geometric Transformations

- Changes in orientation, size, and shape are accomplished with geometric transformations that alter the coordinate descriptions of objects.
- The basic geometric transformations are translation, rotation, and scaling.
- Other transformations that are often applied to objects include reflection and shear.

1. Translation:

"A translation is applied to the object by repositioning it along a straight line path, from one co-ordinate location to another."

- We translate a 2D point by adding translation distances, t_x and t_y, to the original co-ordinate position (x,y) to make the point to a new position (x′,y′).
- Thus new co-ordinates are

$$X' = X + t_x \qquad y' = y + t_y \qquad \ldots(1)$$

The pair (t_x, t_y) is called **translation vector** or **shift vector**.

- We can express the above translation equations as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- The general eqn. can now be written as

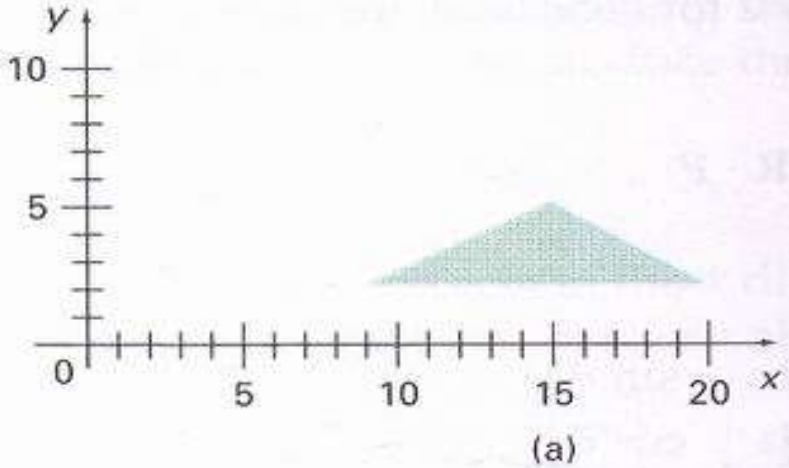
$$\mathbf{P' = P + T} \qquad \text{e.q ...(2)}$$

- Sometimes matrix-transformation equations are expressed in terms of coordinate row vectors instead of column vectors. In this case, we would write the matrix representations as:

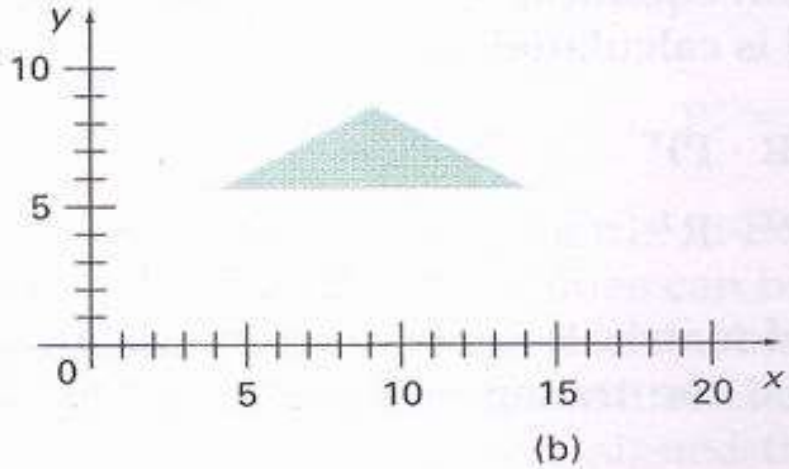
$$P = [\, X \;\; Y \,] \quad P' = [\, X' \; Y' \,] \text{ and } T = [\, t_x \; t_y \,]$$

- Many graphics packages like PHIGS and GKS also use the column-vector representation.
- Translation is a rigid-body transformation that moves objects without change it.

  i.e. Every point on the object is translated by the same amount.
- A straight line segment is translated by applying the transformation e.q …(2) to each of the line endpoints and redrawing the line between the new endpoint position.



(a)



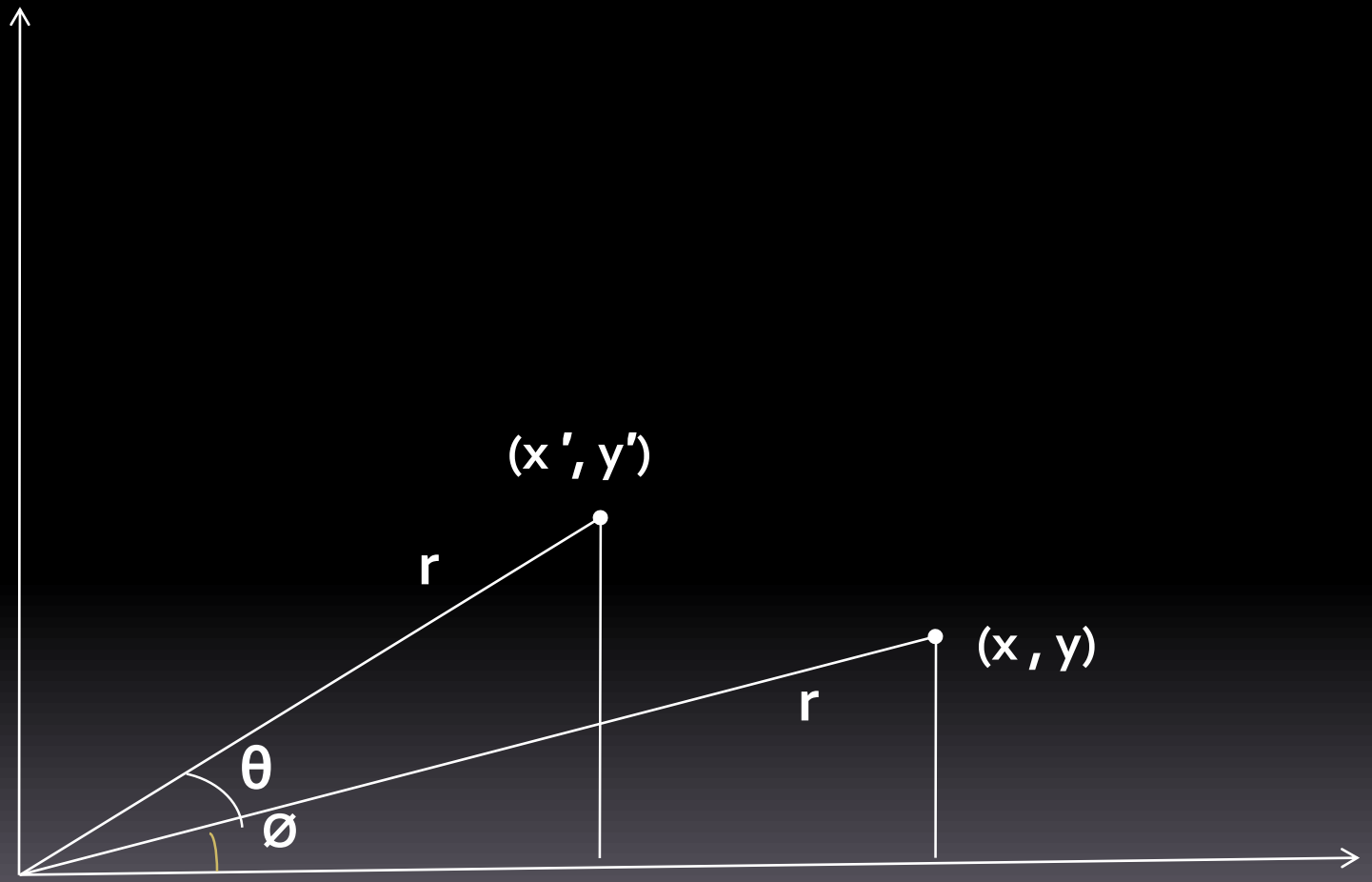
(b)

Figure 5-2
Moving a polygon from position (a) to position (b) with the translation vector (-5.50, 3.75).

Rotation

- A two dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane.
- To generate a rotation, we specify a rotation angle θ and the position (x_r, y_r) of the rotation point (pivot point) about which the object is to be rotated.
- Positive values for the rotation angle define counterclockwise rotations about the pivot point, and negative values rotate objects in the clockwise direction.
- This transformation can also be described as a rotation about a rotation axis that is perpendicular to the xy plane and passes through the pivot point.

- Equations for rotation of a point position P when the pivot point is at the coordinate origin:



- In this figure, r is the constant distance of the point from the origin. Angle Ø is the original angular position of the point from the horizontal, and θ is the rotation angle.
- Using standard trigonometric identities, we can express the transformed coordinates in terms of angles θ and Ø.
- $x' = r \cos (Ø + θ)$ and $y' = r \sin (Ø + θ)$

◉ By expanding the above eqn.,

$$x' = r\cos Ø \cos θ - r\sin Ø \sin θ$$

$$y' = r\cos Ø \sin θ + r\sin Ø \cos θ \qquad \text{e.q. ...(1)}$$

- Original co-ordinate of the point in polar forms are

$$x = r \cos \phi$$
$$y = r \sin \phi \qquad \text{e.q. ...(2)}$$

- Putting the values of x and y from eqn. (2) in eqn.(1), we get

$$x' = x \cos \theta - y \sin \theta$$
$$y' = x \sin \theta + y \cos \theta \qquad \text{e.q. ...(3)}$$

- We can write rotation e.q in matrix form as

$$\mathbf{P' = R.P} \qquad \text{e.q. ...(4)} \quad \text{where R = rotation matrix}$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

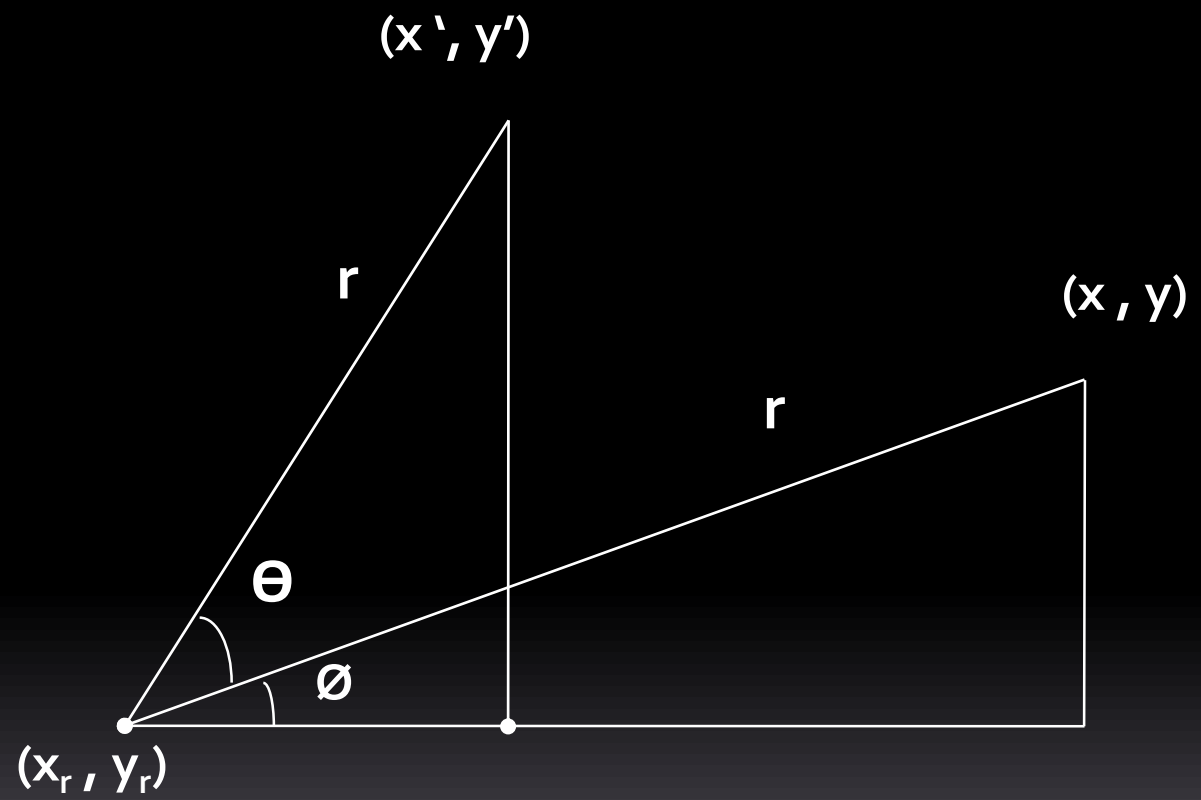
- If we want to represent co-ordinate position as row vectors, then matrix product in e.q. (4) is transposed.
- Transformed row co-ordinate vector is calculated as

  $$P'^{T} = (R.P)^{T}$$

  $$= P^{T}.R^{T}$$

  where $P^{T} = [x\ y]$

  R^{T} of matrix R is obtained by interchanging rows and columns.
- Rotation of a point about an arbitrary pivot position is illustrated in following figure.
- Using the trigonometric relationships in this figure, we can generalize eq. (3) to obtain the transformation equations for rotation of a point about any specified rotation position (x_r, y_r).


(x ‘, y’)
r
(x , y)
r
ϴ
ø
(x_r , y_r)

- The rotation about any random point (Xr,yr) is given by

$$X' = X_r + (X - X_r)\cos\Theta - (Y - Y_r)\sin\Theta$$
$$Y' = Y_r + (X - X_r)\sin\Theta + (Y - Y_r)\cos\Theta \quad \text{e.q. ...(5)}$$

- As with translation, rotation are rigid-body transformation that move objects without deformation.
- Every point on an object is rotated through the same angle.

Scaling

- A scaling transformation alters the size of an object.
- This operation can be carried out for polygons by multiplying the coordinate values (x , y) of each vertex by scaling factors S_x and S_y to produce the transformed coordinates (x' , y').
- So the transformed eq. is

  $x' = x \bullet S_x$, $y' = y \bullet S_y$ eq. …(1)
- The scaling factor S_x is used to scale object in x direction and S_y to scale in y direction.

- The general equation is ,

  $$P' = S \bullet P$$

  where value for scaling factors is any number. S is the 2 by 2 scaling matrix.

  - if value is less than 1, it will reduce the size of the object.

  - if value is greater than 1, it will enlarge the size of an object.

  - if value is equal to 1, it will have no change .

Types of Scaling

- Uniform Scaling : If the values of both scaling factors are same, such scaling is called uniform scaling.
- Differential Scaling: if the values of both scaling factors are different, such scaling is called differential scaling.
- Polygon Scaling: All the vertices of polygon are transformed according to scaling and then polygon is redrawn.
- Circle Scaling: Only radius is changed, and circle is redrawn.
- Ellipse Scaling: In ellipse, both the axes are transformed and ellipse is redrawn.

- Objects transformed with above equation are both scaled and repositioned.
- We can control the location of scaled object by choosing a position, called the fixed point, that to remain unchanged after the scaling transformation.
- Coordinates for the fixed point (xf,yf) can be chosen as one of the vertices, the object centroid, or any other position.
- A polygon is then scaled relative to the fixed point by scaling the distance from each vertex to the fixed point.
- For vertex with coordinates (x,y), the scaled coordinates (x' , y') are calculated as:

$$x' = x_f + (x - x_f) \bullet S_x \;,\; y' = y_f + (y - y_f) \bullet S_y$$

OR

eq. ...(2)

$$x' = x_f \bullet (1 - S_x) + x \bullet S_x \;,\; y' = y_f \bullet (1 - S_y) + y \bullet S_y$$

- Where the additive terms $x_f(1-s_x)$ and $y_f(1-s_y)$ are constant for all points in the object.

Matrix Representations and homogeneous coordinates

- Many graphics applications involve sequence of geometric transformations.
- For example, in animation, translation and rotation are required for each increment for the motion.
- So how the matrix representation of translation, rotation and scaling can be reformulated, so that such transformation sequences can be efficiently processed.
- Basic transformations can be expressed in general matrix form :

$$P' = M1 \bullet P + M2$$

where P and P' are coordinate positions represented as column vectors.

M1 is 2 x 2 matrix containing multiplicative factors.

M2 is 2 element column matrix containing translation terms.

- For translation , M1 is the identity matrix .
- For rotation or scaling, M2 contains the translational terms associated with the pivot point or scaling factors point (zero matrix).
- If we want to produce a sequence of transformations like scaling followed by rotation then translation, we must calculate the transformed coordinates one step at a time.
- First, perform scaling then that scaled coordinates are rotated and finally the rotated coordinates are transformed.
- A more efficient approach would be to combine the transformations so that the final coordinates positions are obtained directly from the initial coordinates, so it eliminates the calculations of the intermediate coordinate values.
- We can combine the multiplicative and translation terms for 2D geometric transformation into a single matrix representation by expanding 2 x 2 matrix to 3 x 3 matrix representation.

- To express any 2D transformation as a matrix multiplication, we can represent each Cartesian coordinate position (x , y) with the homogeneous coordinate triple (x_h, y_h, h)

  Where $\mathbf{x = x_h / h}$

  $\mathbf{y = y_h / h}$ e.q … (1)

  where h is nonzero value.
- A convenient choice is simply to set h = 1.
- Each 2D position is then represented with homogeneous coordinates (x, y,1).
- The term homogeneous coordinates is used in mathematics to refer to the effect of this representation on Cartesian equations.
- When Cartesian point (x,y) is converted to a homogeneous representation (x_h, y_h, h), equations containing x and y, such as f(x,y) = 0, become homogeneous equations in the three parameters x_h, y_h, and h.
- Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations as matrix multiplications.

- We can represent 2D transformations in homogeneous coordinates are as follows :

1. **Translation:**

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

i.e. $x' = x + t_x$, $y' = y + t_y$

- General formula. $\mathbf{P' = T\,(t_x\,,\,t_y\,) \bullet P}$

  Where $T\,(t_x\,,\,t_y\,)$ is 3 x 3 translation matrix.

- The inverse of the translation matrix is obtained by replacing the translation parameters ($-t_x\,,\,-t_y$)

2. Rotation

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

OR

$$\mathbf{P' = R(\theta) \bullet P}$$

Where rotation transformation operator $R(\theta)$ is the 3×3 matrix.

- We get inverse rotation matrix by replacing θ by $-\theta$.

3. Scaling

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

OR

$$\mathbf{P' = S(S_x, S_y) \bullet P}$$

Where $\mathbf{S(S_x, S_y)}$ is the 3 x 3 matrix.

- For inverse scaling S_x and S_y are replaced by $1/S_x$ and $1/S_y$

Composite Transformation

- We can set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformation.
- Forming products of transformation matrices is often referred to as a concatenation, or composition, of matrices.
- For column matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left.
- That is, each successive transformation matrix pre-multiplies the product of the preceding transformation matrices.

1. Translations

- If two successive translation vectors (t_{x1} , t_{y1}) and (t_{x2} , t_{y2}) are applied to a co-ordinate position P, the final transformed location P’ is calculated as :

$$P' = T(t_{x2}, t_{y2}) \bullet \{ T(t_{x1}, t_{y1}) \bullet P \}$$
$$= \{T(t_{x2}, t_{y2}) \bullet T(t_{x1}, t_{y1}) \} \bullet P \qquad \text{e.q. ...(1)}$$

where P and P’ are represented as homogeneous coordinate column vectors.

- The composite transformation matrix for this sequence of translation is ,

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x2} + t_{x1} \\ 0 & 1 & t_{y2} + t_{y1} \\ 0 & 0 & 1 \end{bmatrix}$$

i.e. $T(t_{x2}, t_{y2}) \bullet T(t_{x1}, t_{y1}) = T(t_{x2} + t_{x1}, t_{y2} + t_{y1})$

- **Which demonstrates that two successive translations are additive.**

2. Rotations

- Two successive rotations applied to point P produce the transformed position:

$$P' = R(\theta_2) \bullet \{ R(\theta_1) \bullet P \}$$
$$= \{R(\theta_2) \bullet R(\theta_1) \} \bullet P$$

- **By multiplying the two rotation matrices, we can verify that two successive rotations are additive :**

$$R(\theta_2) \bullet R(\theta_1) = R(\theta_1 + \theta_2)$$

- So that the final rotated coordinates can be calculated with the composite rotation matrix as

$$P' = R(\theta_1 + \theta_2) \bullet P$$

3. Scalings

- Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix:

$$\begin{bmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_{x1} \cdot S_{x2} & 0 & 0 \\ 0 & S_{y1} \cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or

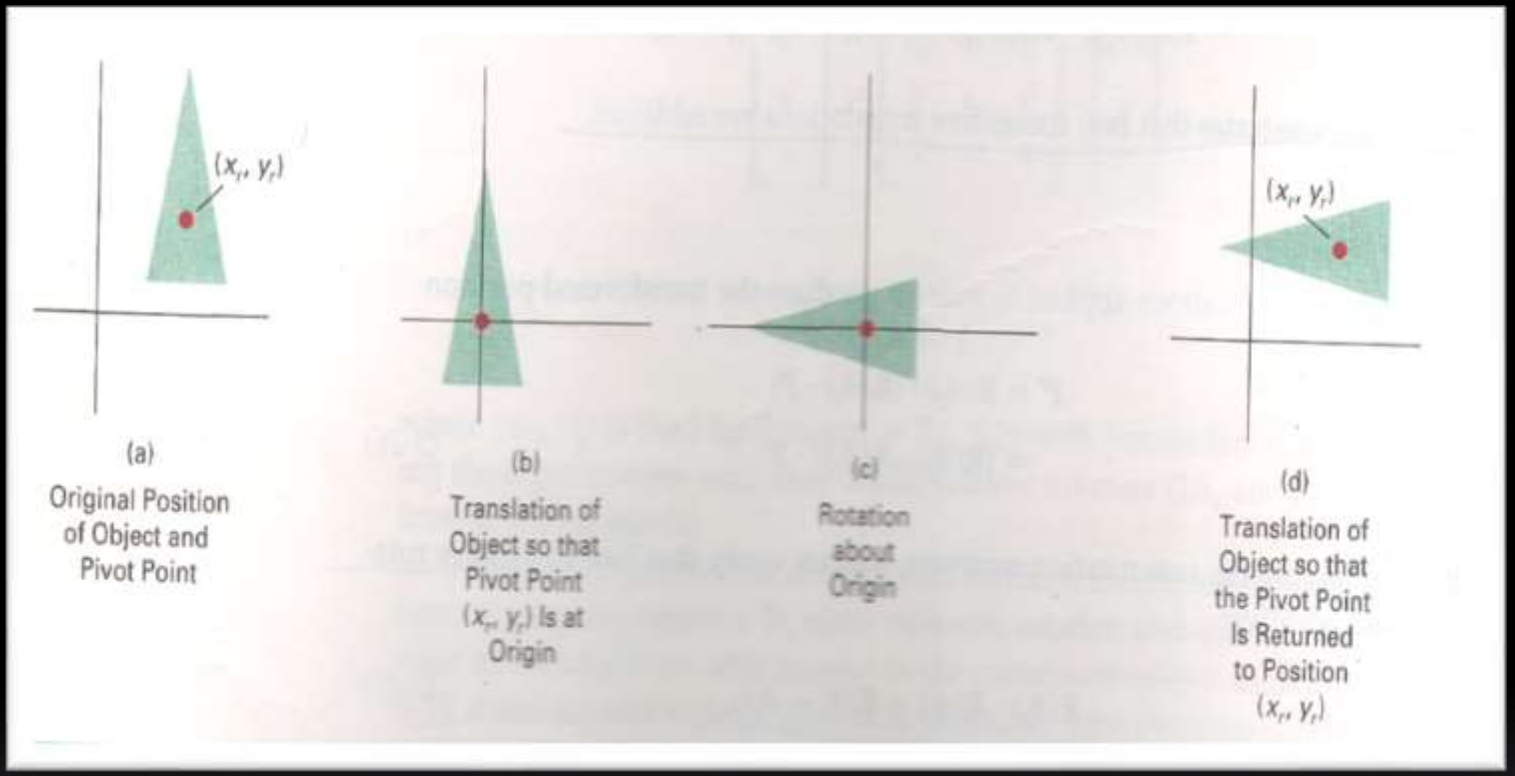
$$S(S_{x2}, S_{y2}) \cdot S(S_{x1}, S_{y1}) = S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{y2})$$

The resulting matrix in this case indicates that successive scaling operations are multiplicative.

General Pivot-Point Rotation

- With a graphics package that only provides a rotation function for revolving object about the coordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of translate-rotate-translate operation:

1. Translate the object so that the pivot-point position is moved to the coordinate origin.
2. Rotate the object about the coordinate origin.
3. Translate the object so that the pivot-point is returned to its original position.


(x_r, y_r)
(a)
Original Position of Object and Pivot Point
(b)
Translation of Object so that Pivot Point (x_r, y_r) Is at Origin
(c)
Rotation about Origin
(x_r, y_r)
(d)
Translation of Object so that the Pivot Point Is Returned to Position (x_r, y_r)

- The composite transformation matrix for this sequence is obtained with the concatenation

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Which can be expressed in the form

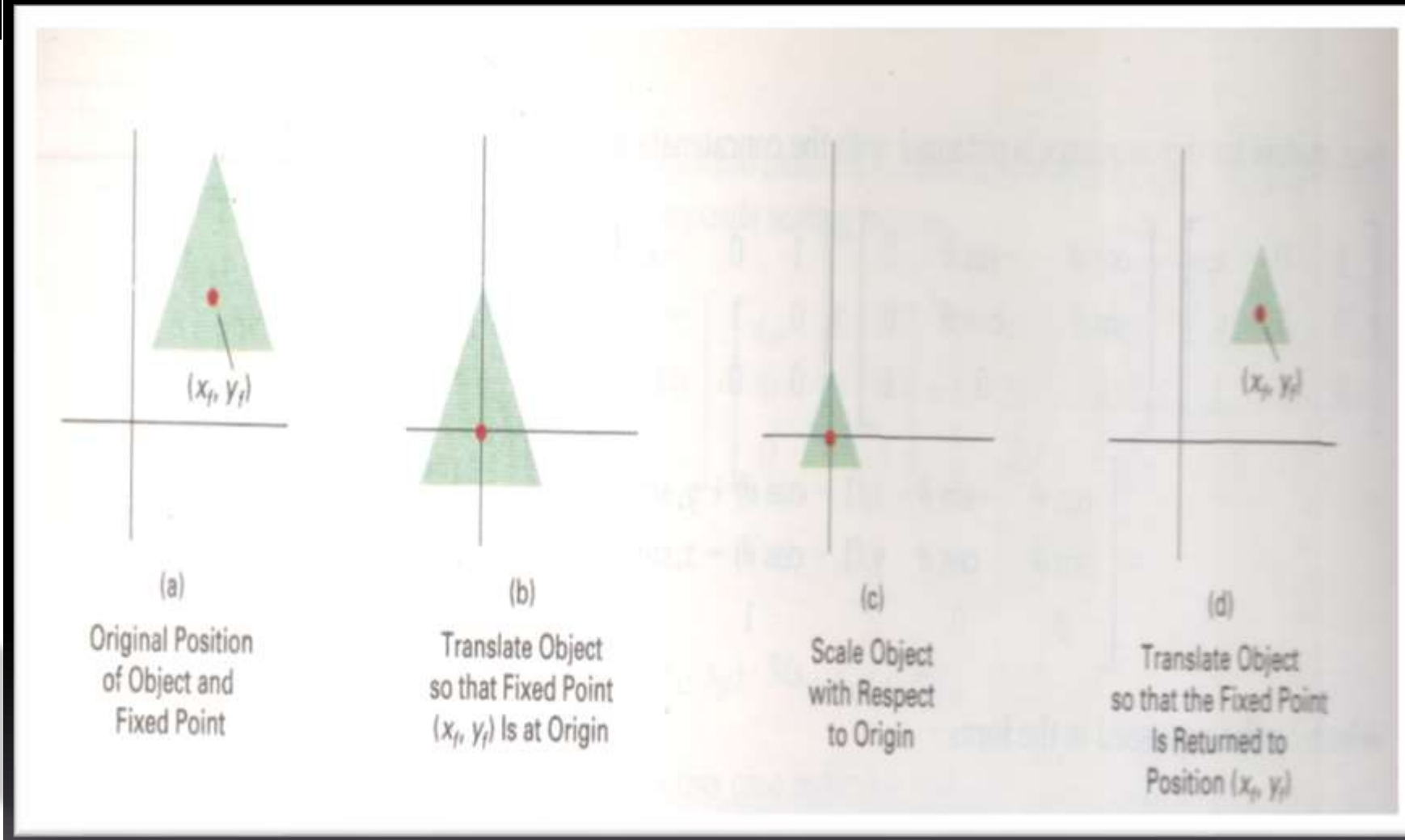
$$T(x_r, y_r) \bullet R(\theta) \bullet T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

General Fixed-Point Scaling

- A transformation sequence to produce scaling with respect to a selected fixed position (x_f ,y_f) using a scaling function that can be only scale relative to the coordinate origin.

  1. Translate the object so that the fixed point is moved to the coordinate origin.
  2. Scale the object with respect to the coordinate origin.
  3. Use the inverse translate of step 1 to return the object to its original position.

- Concatenating the matrices for these three operation produce the required scaling matrix.


(x_f, y_f)
(a)
Original Position of Object and Fixed Point
(b)
Translate Object so that Fixed Point (x_f, y_f) Is at Origin
(c)
Scale Object with Respect to Origin
(x_f, y_f)
(d)
Translate Object so that the Fixed Point Is Returned to Position (x_f, y_f)

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & x_f (1 - S_x) \\ 0 & S_y & y_f (1 - S_y) \\ 0 & 0 & 1 \end{bmatrix}$$

or

$$T (x_f , y_f) \bullet S (s_x , s_y) \bullet T (- x_f , - y_f) = S (x_f , y_f , S_x , S_y)$$

Examples

4.5 Perform a 45° rotation of triangle $A(0, 0)$, $B(1, 1)$, $C(5, 2)$ (*a*) about the origin and (*b*) about $P(-1, -1)$.

SOLUTION

We represent the triangle by a matrix formed from the homogeneous coordinates of the vertices:

$$\begin{matrix} A & B & C \end{matrix}$$
$$\begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

(*a*) The matrix of rotation is

$$R_{45^\circ} = \begin{pmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So the coordinates $A'B'C'$ of the rotated triangle ABC can be found as

$$[A'B'C'] = R_{45^\circ} \cdot [ABC] = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{matrix} A' & B' & C' \end{matrix} \begin{pmatrix} 0 & 0 & \frac{3\sqrt{2}}{2} \\ 0 & \sqrt{2} & \frac{7\sqrt{2}}{2} \\ 1 & 1 & 1 \end{pmatrix}$$

Thus $A' = (0, 0)$, $B' = (0, \sqrt{2})$, and $C' = (\frac{3}{2}\sqrt{2}, \frac{7}{2}\sqrt{2})$.

Examples

(*b*) From Prob. 4.4, the rotation matrix is given by $R_{45°,P} = T_{\mathbf{v}} \cdot R_{45°} \cdot T_{-\mathbf{v}}$, where $\mathbf{v} = -\mathbf{I} - \mathbf{J}$. So

$$R_{45°,P} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{pmatrix}$$

Now

$$[A'B'C'] = R_{45°,P} \cdot [ABC] = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -1 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & (\sqrt{2}-1) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -1 & -1 & (\frac{3}{2}\sqrt{2}-1) \\ (\sqrt{2}-1) & (2\sqrt{2}-1) & (\frac{9}{2}\sqrt{2}-1) \\ 1 & 1 & 1 \end{pmatrix}$$

So $A' = (-1, \sqrt{2}-1)$, $B' = (-1, 2\sqrt{2}-1)$, and $C' = (\frac{3}{2}\sqrt{2}-1, \frac{9}{2}\sqrt{2}-1)$.

Examples

4.8 Magnify the triangle with vertices $A(0, 0)$, $B(1, 1)$, and $C(5, 2)$ to twice its size while keeping $C(5, 2)$ fixed.

SOLUTION

From Prob. 4.7, we can write the required transformation with $\mathbf{v} = 5\mathbf{I} + 2\mathbf{J}$ as

$$S_{2,2,C} = T_{\mathbf{v}} \cdot S_{2,2} \cdot T_{-\mathbf{v}}$$
$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

Representing a point P with coordinates (x, y) by the column vector $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$, we have

$$S_{2,2,C} \cdot A = \begin{pmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -5 \\ -2 \\ 1 \end{pmatrix}$$
$$S_{2,2,C} \cdot B = \begin{pmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -3 \\ 0 \\ 1 \end{pmatrix}$$
$$S_{2,2,C} \cdot C = \begin{pmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}$$

So $A' = (-5, -2)$, $B' = (-3, 0)$, and $C' = (5, 2)$. Note that, since the triangle ABC is completely determined by its vertices, we could have saved much writing by representing the vertices using a 3×3 matrix

$$[ABC] = \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

and applying $S_{2,2,C}$ to this. So

$$S_{2,2,C} \cdot [ABC] = \begin{pmatrix} 2 & 0 & -5 \\ 0 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -5 & -3 & 5 \\ -2 & 0 & 2 \\ 1 & 1 & 1 \end{pmatrix} = [A'B'C']$$

Other transformation

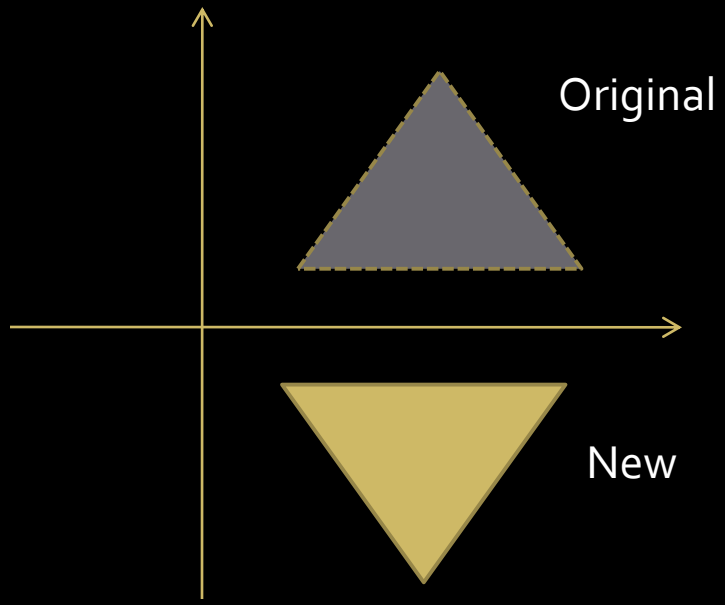
- Most of the graphics packages provide basic transformations. But some packages provides few additional transformations that are useful in certain application.

Reflection :

- Reflection transformation produce a mirror image of an object.
- The mirror image for a 2D reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

- The reflection about x – axis is given by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

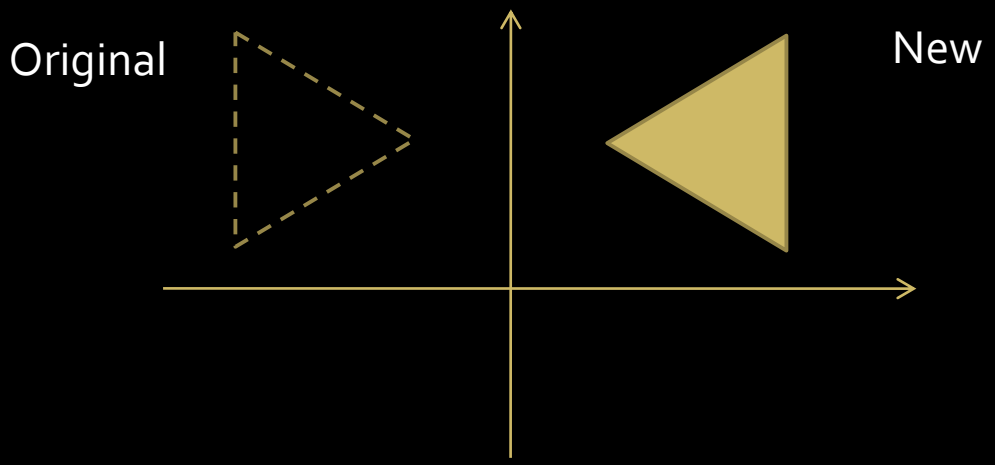


This transformation keeps x values the same, but flips the y values of the coordinate positions.

The resulting orientation of an object after it has been reflected about the x axis is shown in above figure.

- The reflection about y – axis is given by

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

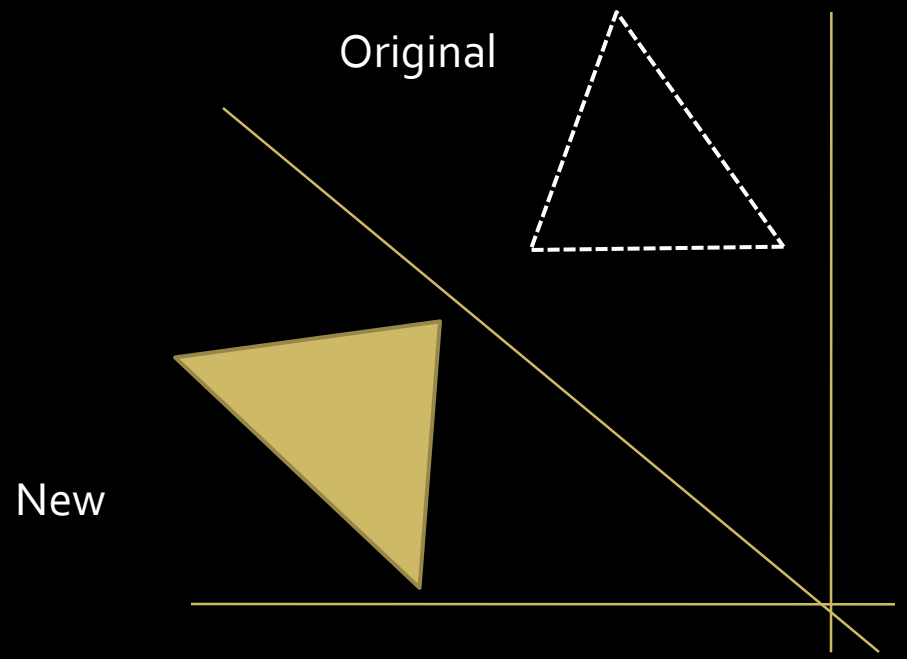


- The reflection about y = x is given by

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

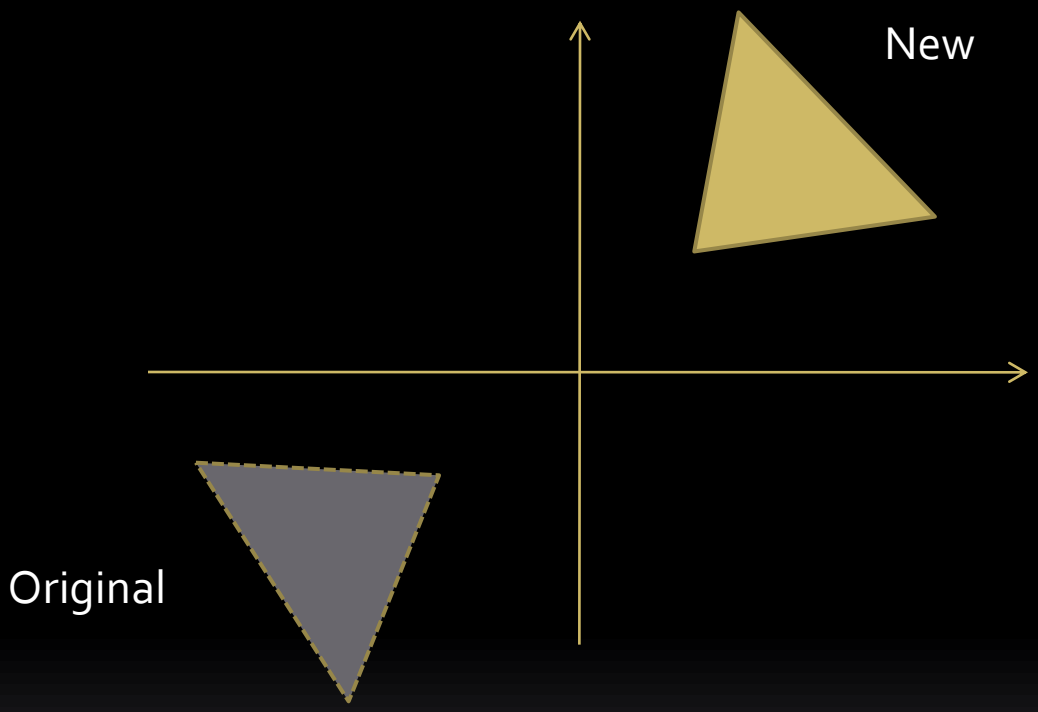
- Reflection about line y = -x is given by

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- The reflection about origin is given by

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



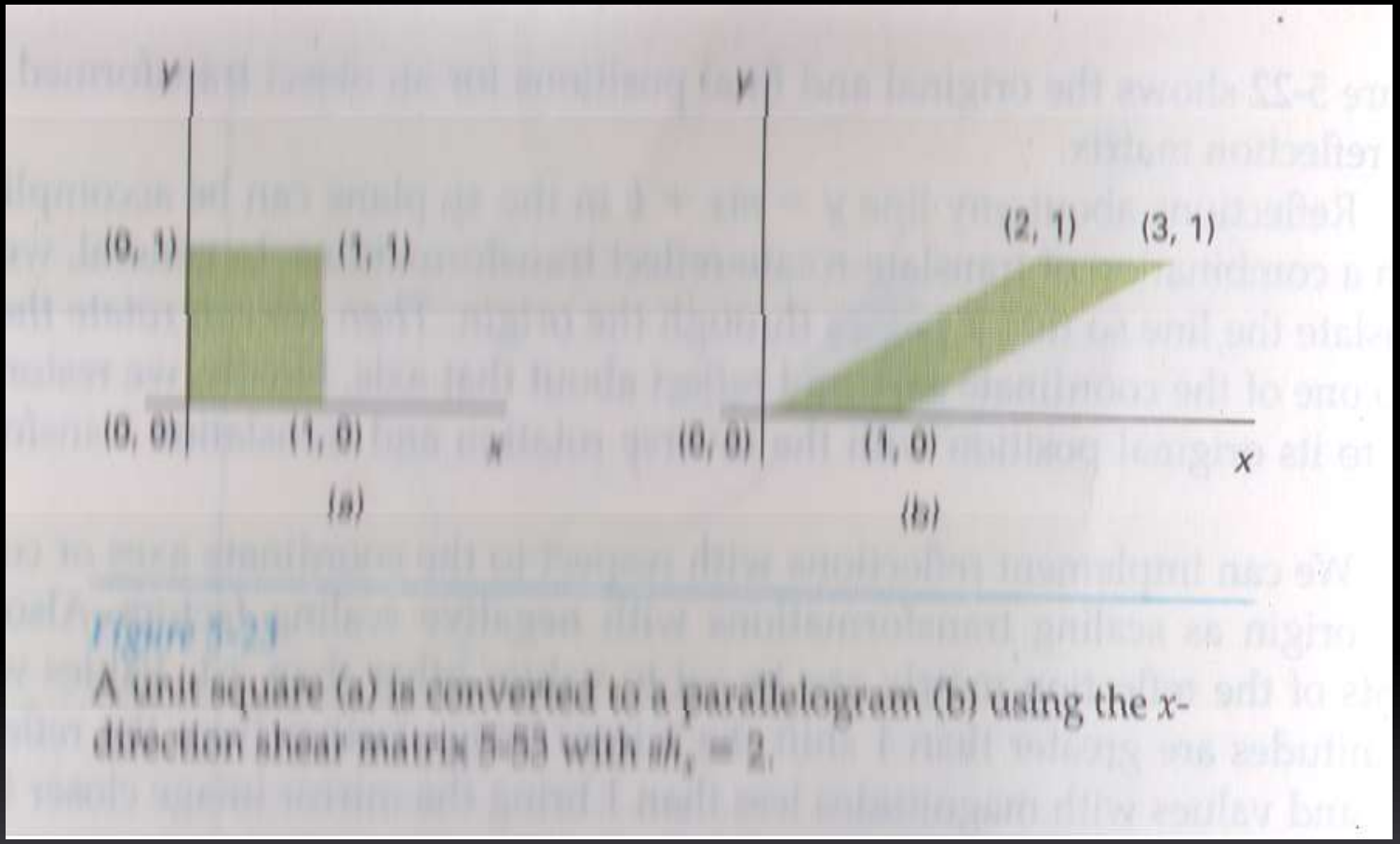
Shear Transformation

- A transformation which distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each-other is called a shear.
- Two common shearing transformations are :
  1. x – direction shear (Shift x – values)
  2. y – direction shear (Shift y – values)
- An x – direction shear relative to the x – axis is produced with the transformation matrix is given as below:

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms coordinate position as

$$x' = x + sh_x \bullet y \ , \ y' = y$$



A unit square (a) is converted to a parallelogram (b) using the x-direction shear matrix 5-53 with $sh_x = 2$.

- Any real number can be assigned to the shear parameter shx.
- A coordinate position (x,y) is then shifted horizontally by an amount proportional to its distance (y value) from the x-axis (y = 0).
- Setting shx = 2 changes the square in to a parallelogram.
- Negative values of shx shift coordinate positions to the left.
- An x – direction shear relative to other reference line with the transformation matrix is given as below:

$$\begin{bmatrix} 1 & sh_x & -sh_x \bullet y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms coordinate position as

$$x' = x + sh_x \bullet (y - y_{ref}) \, , \; y' = y$$

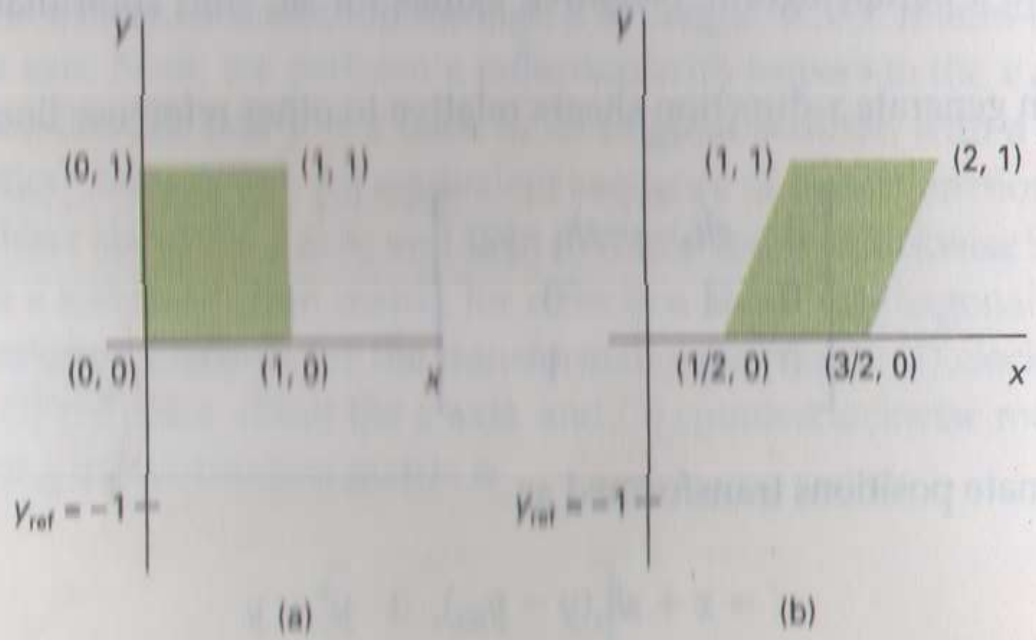


Figure 5-24
A unit square (a) is transformed to a shifted parallelogram (b) with $sh_x = 1/2$ and $y_{ref} = -1$ in the shear matrix 5-55.

- A Y – direction shear relative to the line $x = x_{ref}$ is produced with the transformation matrix is given as below:

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \bullet x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms coordinate position as

$$x' = x, \quad y' = sh_y \bullet (x - x_{ref}) + y$$

- Shearing operations can be expressed as sequences of basic transformations. The x-direction shear matrix can be written as a composite transformation involving a series of rotation and scaling matrices that would scale the unit square along its diagonal, while maintaining the original lengths and orientations of edges parallel to the x axis.
- Shifts in the positions of objects relative to shearing reference lines are equivalent to translations.

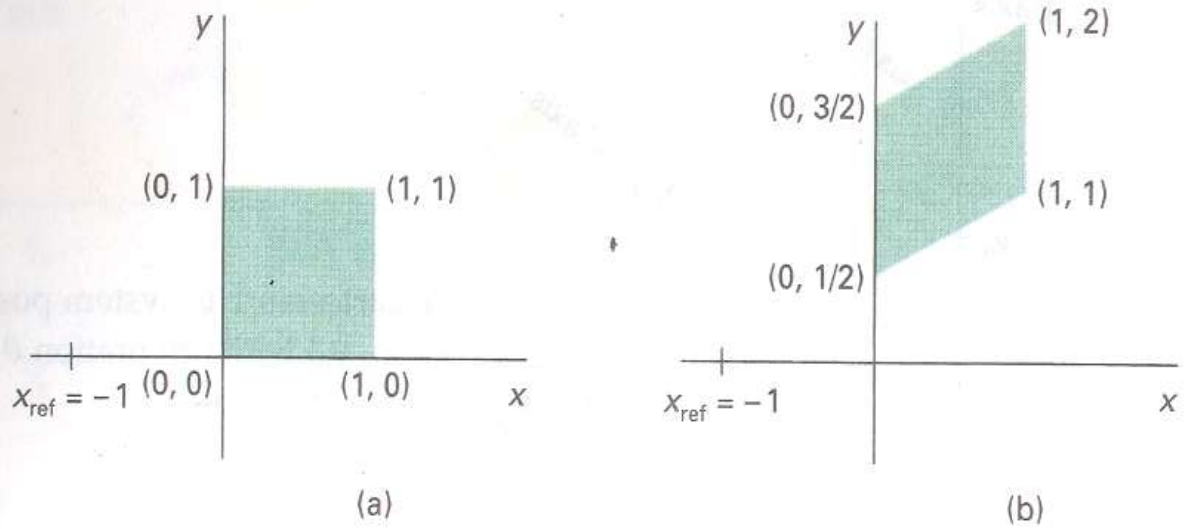
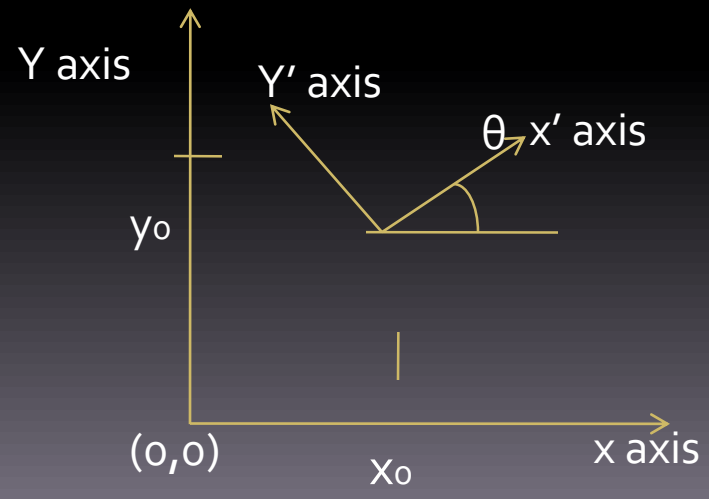


Figure 5-25

A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 1/2$ and $x_{ref} = -1$ in the y-direction using shearing transformation 5-57.

Transformations between coordinate systems

- Graphics applications often require the transformation of object descriptions from one coordinate system to another.
- Figure shows two Cartesian systems, with the coordinate origins at (0,0) and (x_0,y_0) and with an orientation angle θ between the x and x′ axes.



- To transform object descriptions from xy coordinates to x'y' coordinates, we need to set up a transformation that superimposes the x'y' axes onto the xy axes. This is done in two steps:

1. Translate so that the origin (x_0,y_0) of the x'y' system is moved to the origin of the xy system.
2. Rotate the x' axis onto the x axis.

- Translation of the coordinate origin is expressed with the matrix operation and the orientation of the two systems after the translation operation would appear as in figure:

$$T(-x_0,-y_0) = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

- To get the axes of the two systems into coincidence, we then perform the clockwise rotation.
- $R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Concatenating these two transformations matrices gives us the complete composite matrix for transforming object descriptions from the xy system to the x′y′ system:
- $M_{xy,x'y'} = R(-\theta) \bullet T(-x_0,-y_0)$

Raster Methods for transformations

- The particular capabilities of raster systems suggest an alternate method for transforming objects.
- Raster systems store picture information as pixel patterns in the frame buffer.
- Therefore, some simple transformations can be carried out rapidly by simply moving rectangular arrays of stored pixel values from one location to another within the frame buffer.
- Raster functions that manipulate rectangular pixel arrays are generally referred to as ***raster ops***.
- Moving a block of pixels from one location to another is also called a ***block transfer*** of pixel values.
- On bilevel system, this operation is called ***bitBlt (bit-block transfer***), particularly when the function is hardware implemented.
- The term ***pixBlt*** is sometimes used for block transfers on multi level systems (multiple bits per pixels).

- All bit settings in the rectangular area shown are copied as a block into another part of the raster.
- We accomplish this translation by first reading pixel intensities from a specified rectangular area of a raster into an array, then we copy the array back into the raster at the new location.
- The original object could be erased by filling its rectangular area with the background intensity.
- Typical raster functions often provided in graphics packages are:

1. Copy – move a pixel block from one raster area to another.
2. Read – save a pixel block in a designated array.
3. Write – transfer a pixel array to a position in the frame buffer.