Topic 1 - Graphics Output Primitives

- Graphics Output Primitives are fundamental elements or basic operations used in computer graphics to create images on a display device.

Importance in Computer Graphics and Visual Representation

▪ **Fundamental Building Blocks**: Graphics output primitives are foundational for creating complex graphical images.

▪ **Efficiency**: Graphics output primitives are often optimized for efficient rendering on various display devices, including monitors, projectors, and printers.

▪ **Flexibility and Control**: Graphics output primitives offer developers fine-grained control over the appearance and layout of graphical elements.

▪ **Interactivity**: Graphics output primitives enable the implementation of interactive graphics applications.

▪ **Compatibility and Portability**: Graphics output primitives are often standardized across different platforms and programming environments.

▪ **Scalability**: Graphics output primitives can be combined and transformed to create varying complexity and scale graphics.

▪ **Cross-Disciplinary Applications**: Graphics output primitives find applications not only in traditional computer graphics but also in fields such as data visualization, scientific computing, computer-aided design (CAD), virtual reality (VR), and simulation.

Types of Graphics Output Primitives

1. **Points**: The simplest primitive, representing a single pixel on the screen.
2. **Lines**: Defined by two endpoints in 2D space, lines are straight connections between two points.
3. **Curves**: Curves are smooth, continuous lines defined by mathematical equations or algorithms.
4. **Polygons**: Closed shapes with three or more straight sides, polygons are defined by a sequence of vertices connected by straight lines.
5. **Text**: Text primitives allow for rendering alphanumeric characters and symbols on the screen.
6. **Splines**: Splines are a type of curve defined by a set of control points that dictate the shape of the curve.
7. **Images**: Images can also be considered graphics output primitives, representing raster graphics composed of pixels.

Points as Output Primitives

- Points are one of the simplest output primitives in computer graphics. They represent individual pixels on a display device and are the building blocks for creating more complex graphical elements
- **Representation in 2D Space:**

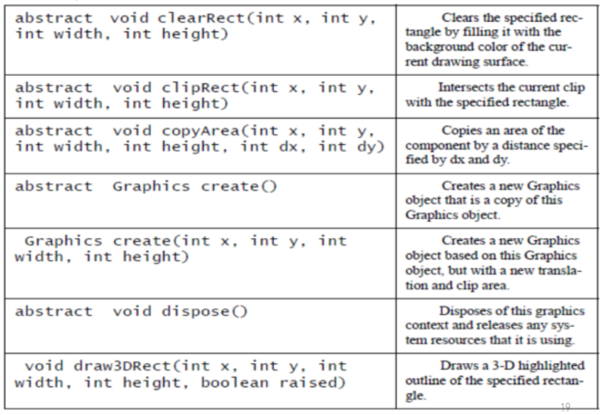
- In 2D space, points are represented by pairs of coordinates (x, y). The 'x' coordinate represents the horizontal position of the point, while the 'y' coordinate represents the vertical position.
- For example, the point (10, 5) in 2D space would be ten units to the right and five units up from the origin (0, 0).

- **Representation in 3D Space:**

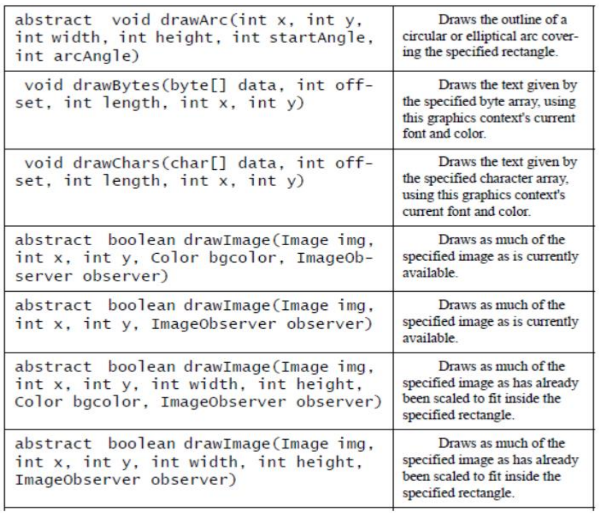
- In 3D space, points are represented by triples of coordinates (x, y, z). The 'x', 'y', and 'z' coordinates represent the position of the point along the three axes: x-axis (horizontal), y-axis (vertical), and z-axis (depth).
- For example, the point (10, -1, 5) in 3D space would be ten units to the right, 1 unit down, and five units away from the origin (0, 0, 0).

Graphics Primitives

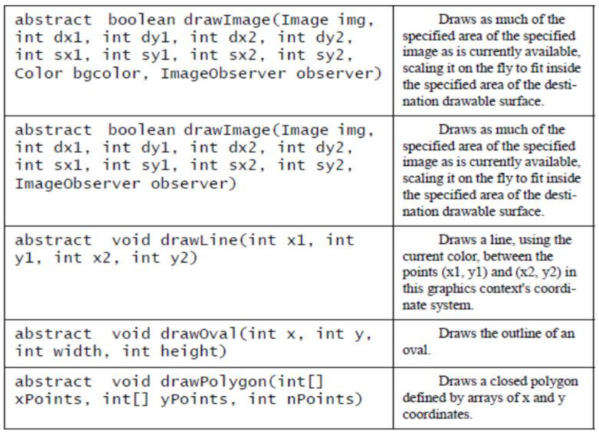
Table 1. Graphics class methods



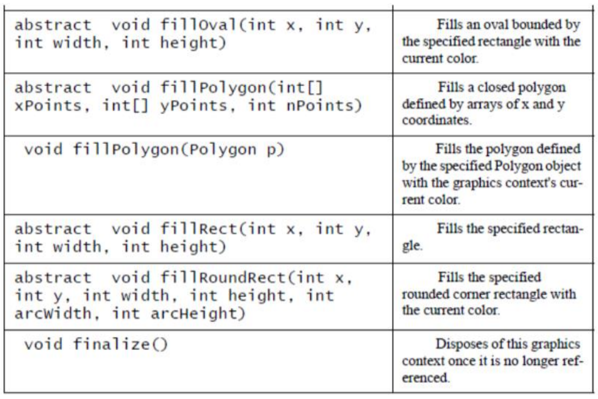
`abstract void clearRect(int x, int y, int width, int height)`	Clears the specified rectangle by filling it with the background color of the current drawing surface.
`abstract void clipRect(int x, int y, int width, int height)`	Intersects the current clip with the specified rectangle.
`abstract void copyArea(int x, int y, int width, int height, int dx, int dy)`	Copies an area of the component by a distance specified by dx and dy.
`abstract Graphics create()`	Creates a new Graphics object that is a copy of this Graphics object.
`Graphics create(int x, int y, int width, int height)`	Creates a new Graphics object based on this Graphics object, but with a new translation and clip area.
`abstract void dispose()`	Disposes of this graphics context and releases any system resources that it is using.
`void draw3DRect(int x, int y, int width, int height, boolean raised)`	Draws a 3-D highlighted outline of the specified rectangle.



`abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Draws the outline of a circular or elliptical arc covering the specified rectangle.
`void drawBytes(byte[] data, int offset, int length, int x, int y)`	Draws the text given by the specified byte array, using this graphics context's current font and color.
`void drawChars(char[] data, int offset, int length, int x, int y)`	Draws the text given by the specified character array, using this graphics context's current font and color.
`abstract boolean drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)`	Draws as much of the specified image as is currently available.
`abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)`	Draws as much of the specified image as is currently available.
`abstract boolean drawImage(Image img, int x, int y, int width, int height, Color bgcolor, ImageObserver observer)`	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.
`abstract boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)`	Draws as much of the specified image as has already been scaled to fit inside the specified rectangle.



`abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, Color bgcolor, ImageObserver observer)`	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
`abstract boolean drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)`	Draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface.
`abstract void drawLine(int x1, int y1, int x2, int y2)`	Draws a line, using the current color, between the points (x1, y1) and (x2, y2) in this graphics context's coordinate system.
`abstract void drawOval(int x, int y, int width, int height)`	Draws the outline of an oval.
`abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`	Draws a closed polygon defined by arrays of x and y coordinates.

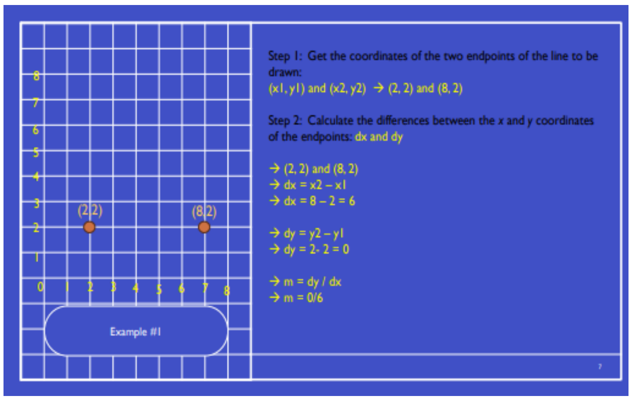


`abstract void fillOval(int x, int y, int width, int height)`	Fills an oval bounded by the specified rectangle with the current color.
`abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)`	Fills a closed polygon defined by arrays of x and y coordinates.
`void fillPolygon(Polygon p)`	Fills the polygon defined by the specified Polygon object with the graphics context's current color.
`abstract void fillRect(int x, int y, int width, int height)`	Fills the specified rectangle.
`abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`	Fills the specified rounded corner rectangle with the current color.
`void finalize()`	Disposes of this graphics context once it is no longer referenced.

Topic 2 - DRAWING LINE ALGORITHMS

- The **Digital Differential Analyzer (DDA)** algorithm is a method used for generating lines and curves in computer graphics.
- **DDA – Line (**y = mx + c)
  - Where:
    - - y is the dependent variable (usually the vertical axis),
    - - x is the independent variable (usually the horizontal axis),
    - - m is the slope of the line, which represents the rate of change of y with respect to x ,
    - - c is the y-intercept, which is the value of y when x is 0
- **SLOPE (**m = dy / dx**)**
  - The slope indicates how steep the line is:
    - A positive slope means the line is increasing from left to right.
    - A negative slope means the line decreases from left to right.
    - A zero slope means the line is horizontal.
- **DDA ALGORITHM Steps**
  1. Get the coordinates of the two endpoints of the line to be drawn: (x1,y1) and (x2,y2).
  2. Calculate the differences between the x and y coordinates of the endpoints.
     - 1. dx = x2 – x1
     - 2. dy = y2 – y1
  3. Determine the number of steps required to increment from the starting point to the ending point.
  4. Compute the incremental changes in x and y for each step:
  5. Starting from the first point (x1,y1), calculate the next point along the line using the incremental changes, repeat this process until the line reaches the endpoint (x2,y2).

6. DDA works with integer coordinates; you may need to round off the calculated coordinates to the nearest integer to plot them accurately.
7. Plot the calculated points on the display device.
8. Connect the plotted points to form the line.



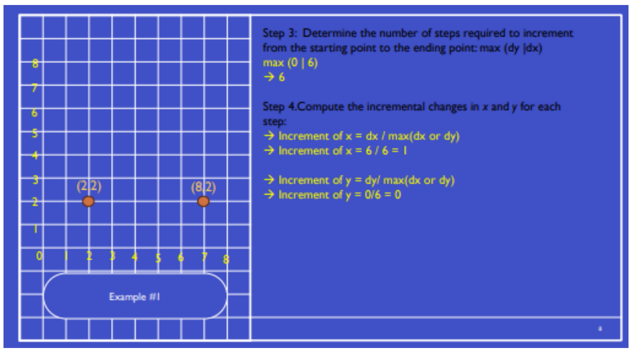
Step 1: Get the coordinates of the two endpoints of the line to be drawn:
(x1, y1) and (x2, y2) → (2, 2) and (8, 2)

Step 2: Calculate the differences between the *x* and *y* coordinates of the endpoints: dx and dy

→ (2, 2) and (8, 2)
→ dx = x2 – x1
→ dx = 8 – 2 = 6

→ dy = y2 – y1
→ dy = 2- 2 = 0

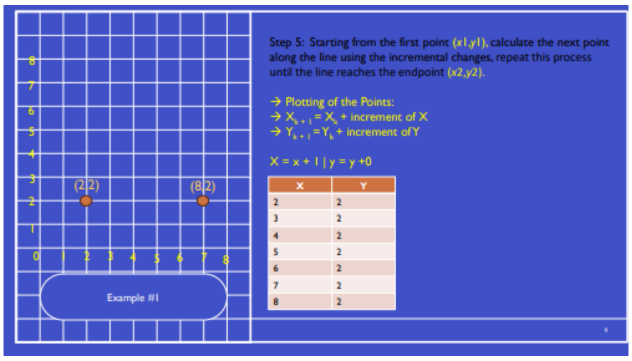
→ m = dy / dx
→ m = 0/6



Step 3: Determine the number of steps required to increment from the starting point to the ending point: max (dy |dx)
max (0 | 6)
→ 6

Step 4.Compute the incremental changes in *x* and *y* for each step:
→ Increment of x = dx / max(dx or dy)
→ Increment of x = 6 / 6 = 1

→ Increment of y = dy/ max(dx or dy)
→ Increment of y = 0/6 = 0

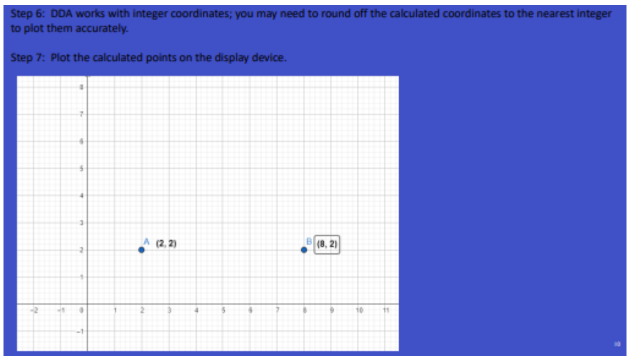


Step 5: Starting from the first point (*x*1,*y*1), calculate the next point along the line using the incremental changes, repeat this process until the line reaches the endpoint (*x*2,*y*2).

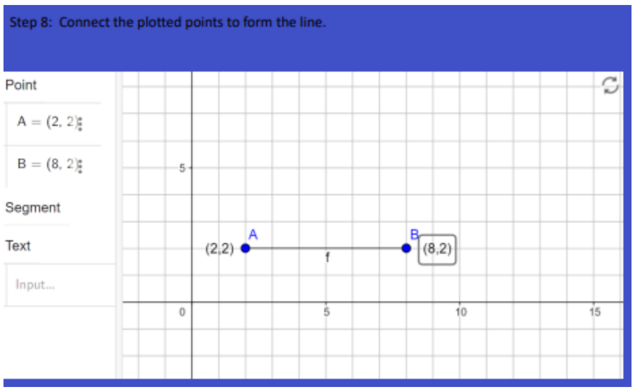
→ Plotting of the Points:
→ $X_{k+1} = X_k$ + increment of X
→ $Y_{k+1} = Y_k$ + increment of Y

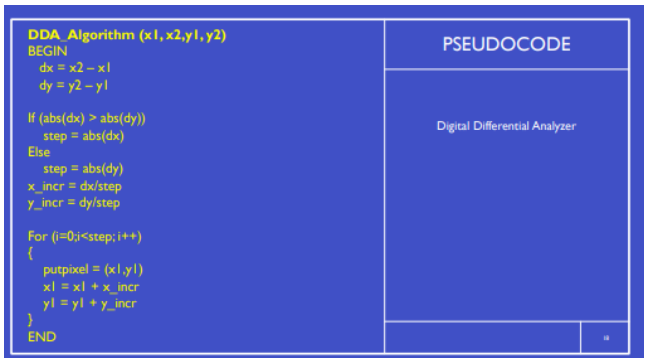
X = x + 1 | y = y +0

X	Y
2	2
3	2
4	2
5	2
6	2
7	2
8	2


Step 6: DDA works with integer coordinates; you may need to round off the calculated coordinates to the nearest integer to plot them accurately.

Step 7: Plot the calculated points on the display device.


Step 8: Connect the plotted points to form the line.

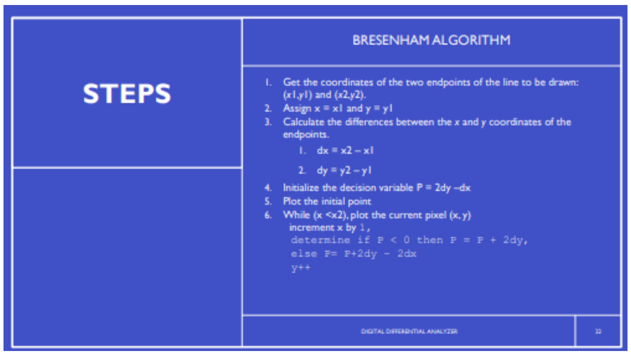


```
DDA_Algorithm (x1, x2,y1, y2)
BEGIN
  dx = x2 – x1
  dy = y2 – y1

If (abs(dx) > abs(dy))
   step = abs(dx)
Else
   step = abs(dy)
x_incr = dx/step
y_incr = dy/step

For (i=0;i<step; i++)
{
   putpixel = (x1,y1)
   x1 = x1 + x_incr
   y1 = y1 + y_incr
}
END
```

PSEUDOCODE

Digital Differential Analyzer

- **BRESENHAM ALGORITHM**
    - Bresenham's line algorithm efficiently draws a line between two points on a discrete grid, such as a computer screen. It avoids floating point arithmetic and only uses integer operations.
  - Between the Bresenham algorithm and the DDA algorithm for drawing lines on a discrete grid.
  - Bresenham algorithm is generally faster because it uses only integer operations and avoids floating-point arithmetic.
  - The DDA algorithm is simpler and easier to understand and may be more appropriate for certain applications where speed is not the primary concern.
- **SLOPE (m = dy / dx)**
  - The slope indicates how steep the line is:
    - A positive slope means the line is increasing from left to right.
    - A negative slope means the line decreases from left to right.
    - A zero slope means the line is horizontal.

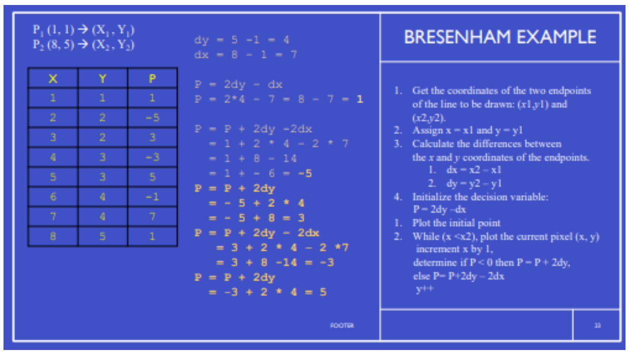


STEPS

BRESENHAM ALGORITHM

```
1. Get the coordinates of the two endpoints of the line to be drawn: (x1,y1) and (x2,y2).
2. Assign x = x1 and y = y1
3. Calculate the differences between the x and y coordinates of the endpoints.
    1. dx = x2 – x1
    2. dy = y2 – y1
4. Initialize the decision variable P = 2dy –dx
5. Plot the initial point
6. While (x <x2), plot the current pixel (x, y)
    increment x by 1,
    determine if P < 0 then P = P + 2dy,
    else P= P+2dy - 2dx
    y++
```

DIGITAL DIFFERENTIAL ANALYZER 22



BRESENHAM EXAMPLE

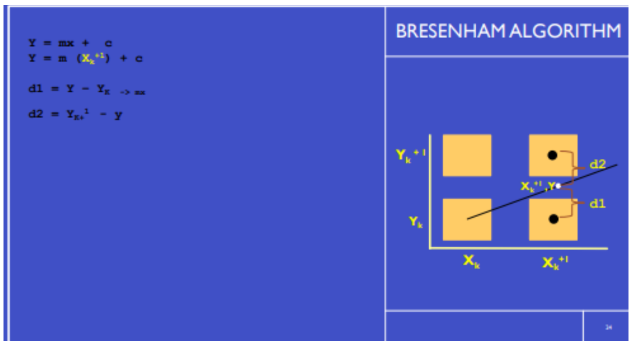
P_1 (1, 1) → (X_1, Y_1)
P_2 (8, 5) → (X_2, Y_2)

X	Y	P
1	1	1
2	2	-5
3	2	3
4	3	-3
5	3	5
6	4	-1
7	4	7
8	5	1

```
dy = 5 -1 = 4
dx = 8 - 1 = 7

P = 2dy - dx
P = 2*4 - 7 = 8 - 7 = 1

P = P + 2dy -2dx
  = 1 + 2 * 4 - 2 * 7
  = 1 + 8 - 14
  = 1 + - 6 = -5
P = P + 2dy
  = - 5 + 2 * 4
  = - 5 + 8 = 3
P = P + 2dy - 2dx
   = 3 + 2 * 4 - 2 *7
   = 3 + 8 -14 = -3
P = P + 2dy
  = -3 + 2 * 4 = 5
```

```
1. Get the coordinates of the two endpoints of the line to be drawn: (x1,y1) and (x2,y2).
2. Assign x = x1 and y = y1
3. Calculate the differences between the x and y coordinates of the endpoints.
    1. dx = x2 – x1
    2. dy = y2 – y1
4. Initialize the decision variable:
   P = 2dy –dx
1. Plot the initial point
2. While (x <x2), plot the current pixel (x, y)
    increment x by 1,
   determine if P < 0 then P = P + 2dy,
   else P= P+2dy – 2dx
    y++
```

FOOTER 23



BRESENHAM ALGORITHM

$Y = mx + c$
$Y = m\ (X_k^{+1}) + c$

$d1 = Y - Y_K$ -> mx

$d2 = Y_{K+}^{1} - y$

24

Topic 3 - Color Theory

- **Color**
  - Color is a property or description of light energy, and only with light do we see color.
  - For example, a tomato absorbs all but red light; therefore, the red is reflected light. For this reason, reflected color is also known as subtractive color.
  - White reflects all the wavelengths of the color spectrum.
  - Black absorbs all the wavelengths of the color spectrum.
  - The spectrum of colors is created by passing white light through a prism.
  - Pigments are the natural chemical substances within an object that interacts with light to determine the characteristic color perceived, such as the bright yellow of bananas, the reds of flowers, and the browns of fur.
  - Naturally or commercially produced pigmented surfaces are seen as reflected light.
  - The colors on a computer screen are light energy, a wavelength that we can refer to as digital color.
  - For example, when selecting a purely blue color in Adobe Photoshop (defined as Blue 255, Red 0, Green 0), the color seen is a blue wavelength of light itself. The digital colors seen in screen-based media are also known as additive colors.
  - Most people can sense color; it’s the sensation when our eyes are presented with different spectral mixes of light.
  - Light with a wavelength of near 400 nanometers makes most people experience “blue,” while light near 700 nm causes the sensation of “red.”
- **Grayscale**
  - Grayscale is the collection or the range of monochromic (gray) shades, ranging from pure white on the lightest end to pure black on the opposite end.
  - Grayscale only contains luminance (brightness) information and no color information; maximum luminance is white, and zero is black.
  - Grayscale is also known as achromatic.
  - Halftoning
  - Gamma Correction

- **Color Models**
  - Color models are mathematical models used to represent colors in an image.
  - They define rules for how colors can be mixed to create new colors.
  - Some of the most commonly used color models include:
    - RGB (Red, Green, Blue)
    - CMYK (Cyan, Magenta, Yellow, Key/Black)
    - HSV (Hue, Saturation, Value).

- **Color Depth**
  - Color depth, or bit depth, refers to the number of bits used to represent each color component in an image.
  - Common color depths include 8-bit (256 colors), 24-bit (true color), and 32-bit.
  - RGB model, if we represent each model by 8 bits = 2 8 = 256 colors
  - For example, we have an image with a color depth of 8 bits per channel. Each color channel (Red, Green, Blue) is represented using 8 bits, allowing for 256 possible intensity levels for each channel.
    - Total colors = (Number of intensity levels per channel) ^ (Number of channels)
    - Number of intensity levels per channel = 2^8 = 256
    - Number of channels = 3 (Red, Green, Blue)
    - Total colors = 256^3 = 16,777,216 colors

- **Color Theories**
  - Subtractive Theory
    - The subtractive, or pigment theory deals with how white light is absorbed and reflected on colored surfaces
  - Additive Theory
    - The Additive or light theory deals with radiated and filtered light.
  - Black absorbs the most light.
  - White reflects the most light.
  - Colored Pigments absorb light and reflect only the frequency of the pigment color.
  - For example, a computer monitor blends varying intensities of red, green, and blue light at its tiny pixels.
  - These pixels are so small and tightly packed that the eye’s RGB response is “fooled” into the perception of many different colors when there are only three.
- **Color Wheel**
  - The color wheel, sometimes called a color circle, is a circular arrangement of colors organized by their chromatic relationship.
  - The color wheel is an illustrative tool used to help us define colors and their relationships to one another.
- **Primary Color**
  - Further defining color helps to understand the role of basic colors called primary colors.
  - When working with light in screen-based media, the three primaries are red, green, and blue (RGB).
  - These primaries are also called the additive primaries because when added together in equal amounts, red, green, and blue create white light
- **Subtractive Primary Color**
  - The subtractive primary colors are red, yellow, and blue in paint or pigment such as watercolors, oils, or colored pencils. They are called primary colors because they cannot be mixed with other colors, yet other colors can be mixed from them.
  - In offset printing, the subtractive primary colors are cyan (C), magenta (M), and yellow (Y), plus black (K), or CMYK. Most often, black is added to increase contrast.
- **Secondary Color**
- **Tertiary Color**

- **Color Properties**
  - **HUE**
    - The name of the color
    - There are not many hues but there are many colors.
    - Pink, scarlet, maroon and crimson are colors, but they all have a hue of Red.
    - cadmium orange is a high saturation color, and burnt sienna is a low saturation color. Both colors are the same hue (orange).
  - **Value**
    - The relative lightness and darkness of a color/hue.
    - The most important of the three qualities. Hues cannot exist without value and different levels of saturation cannot be applied without hues.
    - **Tint** - adding white to a hue
    - **Shade** - adding black to a hue
    - Changing Color Value
      - When working with paint you can thin a color by adding medium.
      - You can also alter the value by mixing hues.
      - Value is changed by its surroundings.
    - Color Interaction
      - Colors change with context.
      - Amounts and repetition are also critical factors.
  - **Saturation**
    - Saturation refers to the brightness or dullness of a color or hue; a hue at its highest level of intensity is said to be purely saturated.
    - Brightness of a color (also called intensity)
    - For example, cadmium orange is a high saturation color, and burnt sienna is a low saturation. Both colors are the same hue (orange).
    - Yellow ochre is less saturated than cadmium yellow.
  - **Complimentary Color**
    - The opposite on color wheel
    - Mixing complementary colors will help you achieve more neutral, naturalistic tones.
    - Avoid using black, you can achieve darker and more neutral values by mixing complements. You will find that your painting will have stronger color interactions.
  - **Split Complimentary**
    - One color and the two hues adjacent to the complement .
    - Split complements function similarly to complementary colors when mixing and as a compositional tool .
    - It involves the use of three colors . Start with one color, find its complement and then use the two colors on either side of it .
    - For example, the complement of blue -green is red -orange and the split complement of blue -green would be red and orange .
  - **Texture**
    - The actual tactile quality of a surface or the simulation or representation of such a surface quality is a texture.
    - There are two categories of texture in the visual arts: tactile and visual. Tactile textures have an actual tactile quality and can be physically touched and felt; they are also called actual textures.

    - Several printing techniques can produce tactile textures on a printed design, including embossing and debossing, stamping, engraving, and letterpress.
- **Pattern**
    - The pattern is a consistent repetition of a single visual unit or element within a given area. In all cases, there must be systematic repetition with obvious directional movement. (An interesting aspect of the pattern is that the viewer anticipates a sequence.)
- **Monochromatic Color**
    - “Mono” means one, “chroma” means color
    - monochromatic color schemes have only one color and its values.
    - The painting has a **monochromatic** color scheme - blue and the values (tints and shades) of blue.
- **Analogous**
    - Analogous colors are three colors next to each other on the color wheel, composed of one dominant color (usually a primary or secondary color), then a supporting color (a secondary or tertiary color), and a third color that is either a mix of the two first colors or an accent color that pops.

- **COLOR PSYCHOLOGY**
    - **Color Temperature**
        - An artist may use warm and cool color relationships to create depth and volume
        - Color temperature is also used to create a strong sense of light.
        - Warm and Cool colors.
        - Warm colors are the colors red, orange, and yellow.
        - Blue and green are cool colors.
        - Violet/purple can be both warm and cool depending on how much red or how much blue is in the violet.
    - **Warm Color**
        - Red, Orange, Yellow
        - Warm colors advance spatially.
        - Represents – Fire, Sunlight
        - Implies – Happy, energy, anger
    - **Cool Color**
        - Blue, Green, Purple ▪ Cool colors recede spatially.
        - Represents – Sky, Water, Grass
        - Implies – Sadness, Depression, Night
- **Color Design**
    1. Color can create a focal point (low saturation color amid a field of highly saturated colors, and vice versa).
    2. Color is often used symbolically.
    3. Color can have cultural and emotional associations.
    4. Color can be associated with a brand and be chosen to express a brand’s personality—for example, Coca-Cola red or Tiffany blue.
    5. Color juxtaposition can create the illusion of space.
    6. Color selection should enhance the readability of the type.
    7. Ramped color, or a gradation of color, creates the illusion of movement. Color and Graphic Design Principles 48
    8. Color should always be selected with the other colors in the piece.
    9. There are established color schemes, such as monochromatic schemes, analogous colors, complementary colors, split complementary colors, triadic schemes, tetradic schemes, cool palettes, and warm palettes.

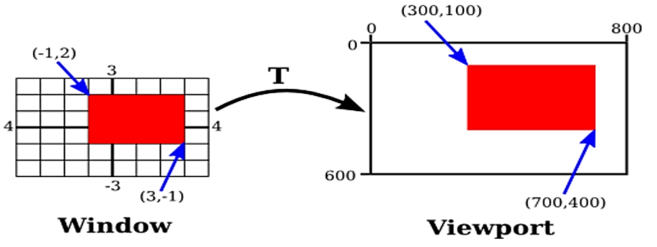
10. Color palettes are also associated with techniques, historical periods, and art and nature, such as batik colors, art deco, Victorian, retro palettes, ancient Chinese ceramic colors, and earth tones.
11. Grays can be warm or cool Color and Graphic Design Principles 49 Color and Graphic Design Principles
12. Do not use color indiscriminately.
13. Consider color as an aesthetic and cognitive design tool during every design stage.
14. Limit the use of different colors. Exercise simplicity, clarity, and consistency.
15. Do not rely on color alone. Color is best at redundantly emphasizing information.
16. Try to work in the same conditions your project will be used in the final form.
17. Experiment, experiment, experiment. Do not underestimate the potential of color.

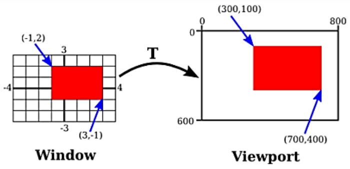
GEOMETRIC TRANSFORMATION

- Geometric Transformation is a process of modifying and repositioning the graphic objects.
- An *operation transforms* or changes a shape.
- *Changing* some graphics into something else by applying rule

Viewing and Modeling

- **T** is the coordinate transformation function that takes world coordinates (*x*,*y*) in some window and maps them to pixel coordinates **T**(*x*,*y*) in the viewport.





Viewing and Modeling

Let:

T(x,y) = (800*(x+4)/8, 600*(3-y)/6)

If (x,y)= (-1, 2) then

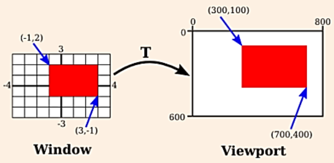
T(x,y) = (800 * (x + 4) / 8, 600 *(3 - y) / 6)

T(-1,2)= (800 * (-1+4) /8, 600 * (3- 2) /6)

= (800 * 3 / 8, 600 * 1 /6)

= (2400 / 8, 600 / 6)

= (300, 100)



Viewing and Modeling

Let:

T(x,y) = (800*(x+4)/8, 600*(3-y)/6)

If (x,y)= (3, -1) then

T(x,y) = (800 * (x + 4) / 8, 600 *(3 - y) / 6)

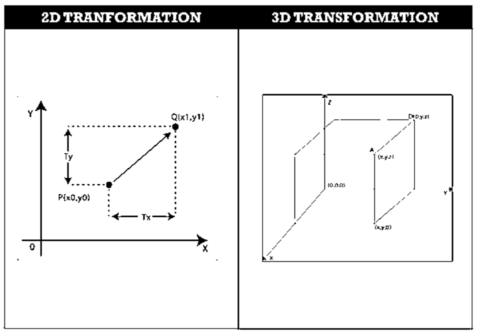
T(-1,2) = (800 * (3+4) /8, 600 * (3- (-1)) /6)

= (800 * 7 / 8, 600 * 4 /6)

= (5600 / 8, 2400 / 6)

= (700, 400)

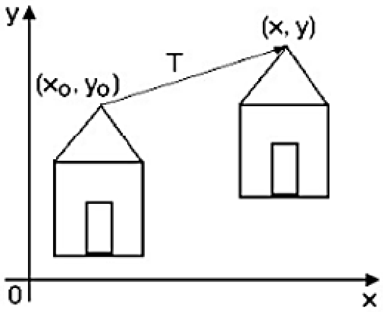
2D Transformation	3D Transformation
Translation	Translation
Scaling	Scaling
Rotation	Rotation
Reflection	Reflection
Shearing	Shearing

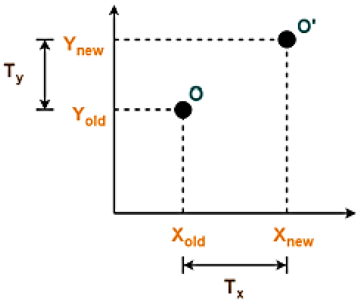


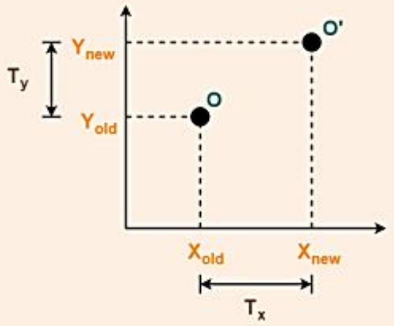
$$X_{new} = X_{old} + T_x$$
$$Y_{new} = Y_{old} + T_y$$
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

2D Translation

- Translation is moving an object from one location to another without altering its size, shape, or orientation. This transformation involves shifting all the points of the object by a constant distance and direction in a 2D coordinate system.





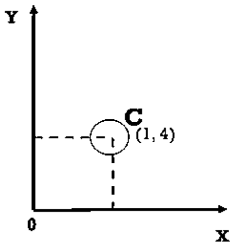


$$X_{new} = X_{old} + T_x$$
$$Y_{new} = Y_{old} + T_y$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Example:

Shape = Circle (C), radius = 10, center coordinates (1, 4). Apply the translation with distance 5 towards X axis and 1 towards Y axis. Obtain the new coordinates of C without changing its radius.



Solution:

$X_{new} = X_{old} + T_x$

$Y_{new} = Y_{old} + T_y$

Given:

Old Center coordinates of C $(X_{old}, Y_{old}) = (1, 4)$

Translation vector = $(T_x, T_y) = (5, 1)$

New Center Coordinates of C = X_{new}, Y_{new}

Apply the translation equation:

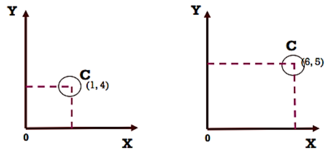
$X_{new} = X_{old} + T_x = 1 + 5 = 6$

$Y_{new} = Y_{old} + T_y = 4 + 1 = 5$

New Center Coordinates of C = (6, 5)

Example:

- ➢ Shape = Circle (C), radius = 10, center coordinates (1, 4).
- ➢ Apply the translation with distance 5 towards X axis and 1 towards Y axis.
- ➢ Obtain the new coordinates of C without changing its radius.



Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the translation with distance 1 towards X axis and 1 towards Y axis.
- ➢ Obtain the new coordinates of the square.

Solution:

$X_{new} = X_{old} + T_x$
$Y_{new} = Y_{old} + T_y$

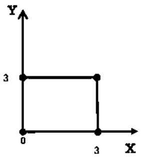
Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)

Translation vector = (T_x, T_y) = (1,1)

For P1(0,3):
New Center Coordinates of P1 = X_{new}, Y_{new}

Apply the translation equation:
$X_{new} = X_{old} + T_x = 0 + 1 = 1$
$Y_{new} = Y_{old} + T_y = 3 + 1 = 4$

New Center Coordinates of P1 = (1, 4)



Solution:

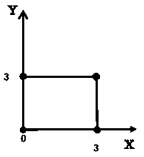
Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)

Translation vector = (T_x, T_y) = (1,1)

For P2(3,3):
New Center Coordinates of P2 = X_{new}, Y_{new}

Apply the translation equation:
$X_{new} = X_{old} + T_x = 3 + 1 = 4$
$Y_{new} = Y_{old} + T_y = 3 + 1 = 4$

New Center Coordinates of P2 = (4, 4)



Solution:

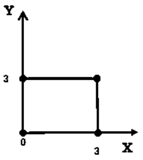
Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)

Translation vector = (T_x, T_y) = (1,1)

For P3(3,0):
New Center Coordinates of P3 = X_{new}, Y_{new}

Apply the translation equation:
$X_{new} = X_{old} + T_x = 3 + 1 = 4$
$Y_{new} = Y_{old} + T_y = 0 + 1 = 1$

New Center Coordinates of P3 = (4, 1)



Solution:

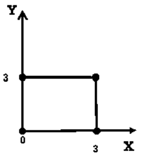
Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)

Translation vector = (T_x, T_y) = (1,1)

For P4(0,0):
New Center Coordinates of P4 = X_{new}, Y_{new}

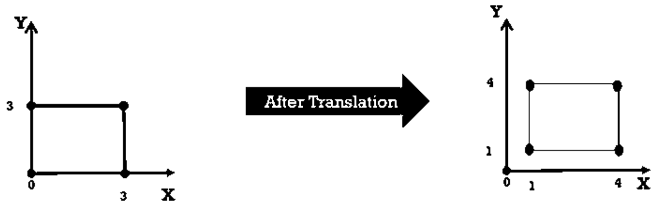
Apply the translation equation:
$X_{new} = X_{old} + T_x = 0 + 1 = 1$
$Y_{new} = Y_{old} + T_y = 0 + 1 = 1$

New Center Coordinates of P4 = (1, 1)



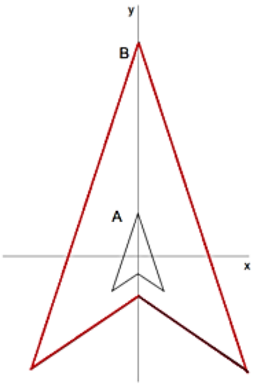
Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the translation with distance 1 towards X axis and 1 towards Y axis.
- ➢ Obtain the new coordinates of the square.
- ➢ New Coordinates of the square = P1(1,4), P2(4,4), P3(4,1), P4(1,1)



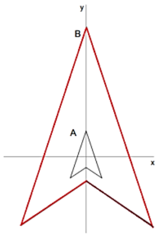
2D SCALING

- Scaling is the process of resizing an object in a two-dimensional coordinate system.
- In scaling, you either expand or compress the object's dimensions.
- The scaling factor determines whether the object size will be increased or reduced.
- If the scaling factor > 1, then the object size is increased.
- If the scaling factor < 1, then the object size is reduced.



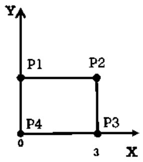
$X_{new} = X_{old}\ S_x$
$Y_{new} = Y_{old}\ S_y$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{x} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$



Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the scaling parameter 2 towards X axis and 3 towards Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
Scaling factor along X axis = 2
Scaling factor along Y axis = 3

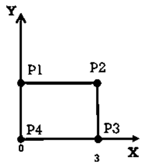
For P1(0,3)

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{x} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} \qquad \begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{x} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \text{x} \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the scaling parameter 2 towards X axis and 3 towards Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
Scaling factor along X axis = 2
Scaling factor along Y axis = 3

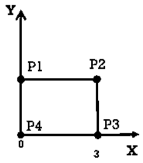
For P2(3,3)

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{x} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} \qquad \begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{x} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \text{x} \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the scaling parameter 2 towards X axis and 3 towards Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
Scaling factor along X axis = 2
Scaling factor along Y axis = 3

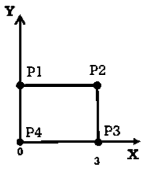
For P3(3,0)

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{x} \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} \qquad \begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{x} \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \text{x} \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the scaling parameter 2 towards X axis and 3 towards Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of S = P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
Scaling factor along X axis = 2
Scaling factor along Y axis = 3

For P4(0,0)

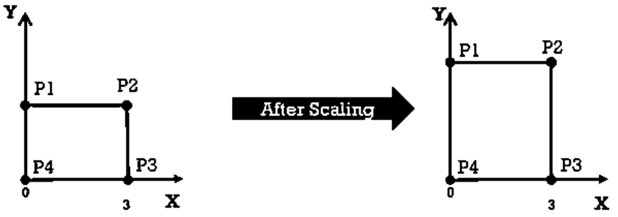
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} x \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} x \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

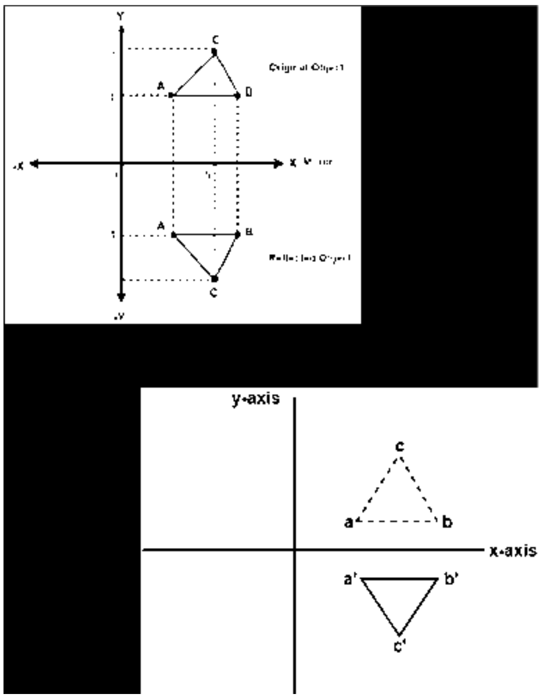
Example:

- ➢ Shape = Square (S), points P1(0,3), P2(3,3), P3(3,0), and P4(0,0)
- ➢ Apply the scaling parameter 2 towards X axis and 3 towards Y axis.
- ➢ Obtain the new coordinates of the square.
- ➢ New Coordinates of the square = (0,9),(6,9), (6,0),(0,0)



2D REFLECTION

- Reflection is a geometric transformation that mirrors an object or image across a specified axis in a two-dimensional coordinate system.
- Reflection is a rotation where the rotation angle is 180 degrees.
- The reflected object is always formed on the other side of the mirror
- The reflected object's size is the same as the original objects.



Example:

- ➢ Shape = Triangle (T), points P1 (3,4), P2(6,4), and P3(5,6).
- ➢ Apply the reflection on the X axis.
- ➢ Obtain the new coordinates of the square.

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Given:
Old Center coordinates of T = P1(3,4), P2(6,4), P3(5,6)
Reflections has to be taken on the X axis

For P1(3,4)

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

Example:

- ➢ Shape = Triangle (T), points P1(3,4), P2(6,4), and P3(5,6).
- ➢ Apply the reflection on the X axis.
- ➢ Obtain the new coordinates of the square.

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Given:
Old Center coordinates of T = P1(3,4), P2(6,4), P3(5,6)
Reflections has to be taken on the X axis

For P2(6,4)

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ -4 \end{bmatrix}$$

Example:

- ➢ Shape = Triangle (T), points P1(3,4), P2(6,4), and P3(5,6).
- ➢ Apply the reflection on the X axis.
- ➢ Obtain the new coordinates of the square.

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

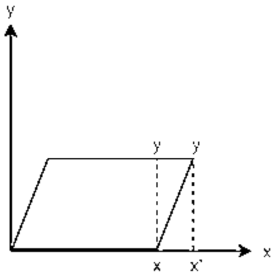
Given:
Old Center coordinates of T = P1(3,4), P2(6,4), P3(5,6)
Reflections has to be taken on the X axis

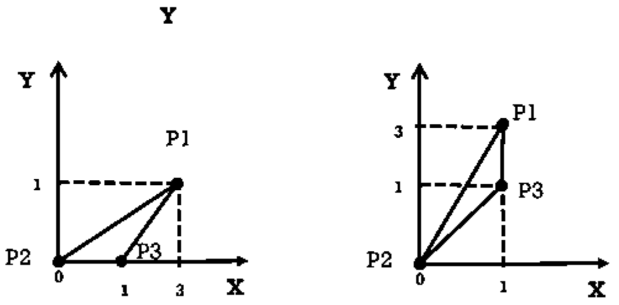
For P3(5,6)

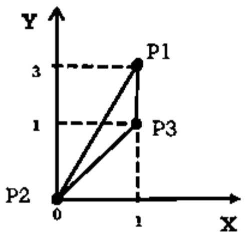
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} x \begin{bmatrix} 5 \\ 6 \end{bmatrix} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$$

2D SHEARING

- 2D Shearing is an ideal technique to change the shape of an existing object in a two dimensional plane.
- 2D Shearing is a transformation in 2D graphics where the position of each point in a shape is shifted along one axis, typically by a certain amount.

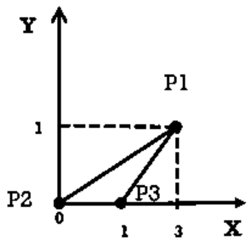






$$X_{new} = X_{old}$$
$$Y_{new} = Y_{old} + Sh_y \text{ x } X_{old}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \text{ x } \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

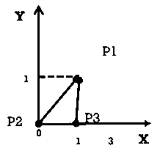


$$X_{new} = X_{old} + Sh_x \text{ x } Y_{old}$$
$$Y_{new} = Y_{old}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \text{ x } \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Example:

- ➢ Shape = Triangle (T), points P1(1,1), P2(0,0), and P3(1,0).
- ➢ Apply the shear parameter 2 on the X axis and 2 on the Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of T = P1(1,1), P2(0,0), P3(1,0)
Shearing parameter towards X direction (Sh_x) = 2
Shearing parameter towards X direction (Sh_y) = 2

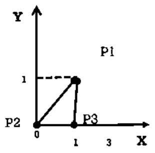
For P1(1,1)

$$X_{new} = X_{old} + Sh_x \text{ x } Y_{old}$$
$$Y_{new} = Y_{old}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \text{ x } \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Example:

- ➢ Shape = Triangle (T), points P1(1,1), P2(0,0), and P3(1,0).
- ➢ Apply the shear parameter 2 on the X axis and 2 on the Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of T = P1(1,1), P2(0,0), P3(1,0)
Shearing parameter towards X direction (Sh_x) = 2
Shearing parameter towards X direction (Sh_y) = 2

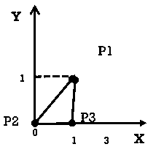
For P2(0,0)

$$X_{now} = X_{old} + Sh_x \text{ x } Y_{old}$$
$$Y_{new} = Y_{old}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \text{ x } \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Example:

- ➢ Shape = Triangle (T), points P1(1,1), P2(0,0), and P3(1,0).
- ➢ Apply the shear parameter 2 on the X axis and 2 on the Y axis.
- ➢ Obtain the new coordinates of the square.



Given:
Old Center coordinates of T = P1(1,1), P2(0,0), P3(1,0)
Shearing parameter towards X direction (Sh_x) = 2
Shearing parameter towards X direction (Sh_y) = 2

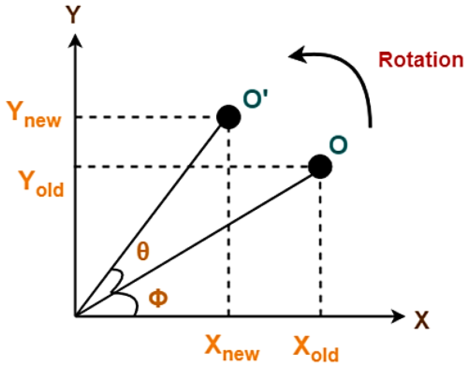
For P3(1,0)

$$X_{new} = X_{old} + Sh_x \text{ x } Y_{old}$$
$$Y_{new} = Y_{old}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \text{ x } \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

2D ROTATION

- 2D rotation is a transformation in two dimensional space where points, lines, shapes, or images are rotated around a fixed point called the center of rotation.
- 2D rotation is a process of rotating an object with respect to an angle in a two dimensional plane.



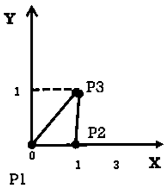
2D ROTATION

$$X_{new} = X_{old}\ \ COS\,\theta\ - YOL_D\ \sin\theta$$
$$Y_{new} = X_{old}\ \ \sin\theta\ + Y_{OLD}\ \cos\theta$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{ x } \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Problem:

Given a Triangle with the following corner coordinates (0,0), (1,0) and (1,1). Rotate the Triangle by 90-degree counterclockwise direction and determine the new coordinates.



P1 = (0,0):

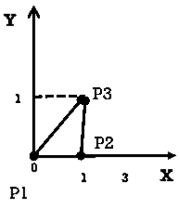
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{ x } \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} \text{ x } \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \text{ x } \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Problem:

Given a Triangle with the following corner coordinates (0,0), (1,0) and (1,1). Rotate the Triangle by 90-degree counterclockwise direction and determine the new coordinates.



P2 = (1,0):

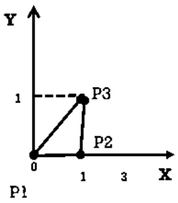
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{ x } \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} \text{ x } \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \text{ x } \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Problem:

Given a Triangle with the following corner coordinates (0,0), (1,0) and (1,1). Rotate the Triangle by 90-degree counterclockwise direction and determine the new coordinates



P3 = (1,1):

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{x} \begin{bmatrix} \mathrm{X_{old}} \\ \mathrm{Y_{old}} \end{bmatrix}$$

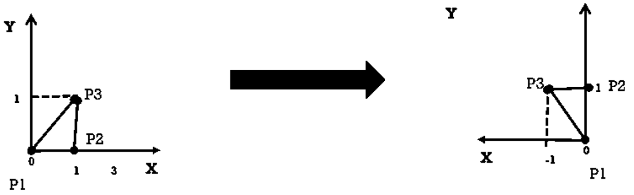
$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} \text{x} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \text{x} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

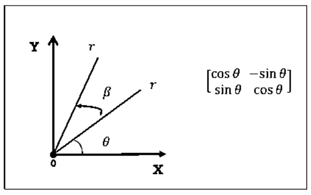
Problem:

Given a Triangle with the following corner coordinates (0,0), (1,0) and (1,1). Rotate the Triangle by 90-degree counterclockwise direction and determine the new coordinates.

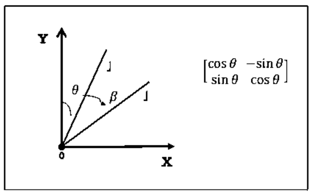
New Coordinates: (0,0),(0,1),(-1,1)



2D ROTATION



Counterclockwise Rotation



Clockwise Rotation

Three Dimensional (3D)

What is 3D?

- 3D means three-dimensional, i.e. something that has width, height and depth (length).
- 3D is a virtual – solid, surface form of shapes, objects, subject or picture with a three dimensional data value.
- It's a process of developing a mathematical representation of any shape into three dimensional Surface / Object.
- It can be displayed as a two-dimensional image through a process called 3D Rendering.

BASICS OF 3D

First full digital fully 3D animated movie toy story, which premiered in 1995.

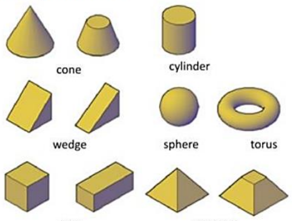
What is 3D modeling?

- 3D Modelling is a multi-stage process.
  - ➢ Representation
    - Build a model of 3D Objects
      → Shapes
      → Surface textures
  - ➢ Rendering
    - Geometric translations
    - Projection to 2D
    - Light representation

- 3D modelling is the process of developing a mathematical representation of any three dimensional object via specialized software.

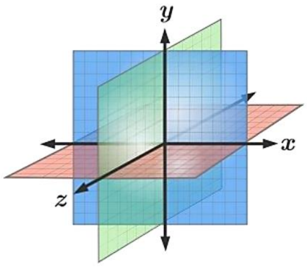
3D modeling

- Start from primitives



- Model based on a reference

- Cartesian Plane



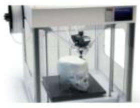
3d modeling in practice

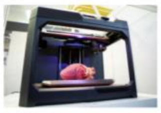
Internet Marketing

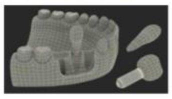
Architecture

Industrial Design

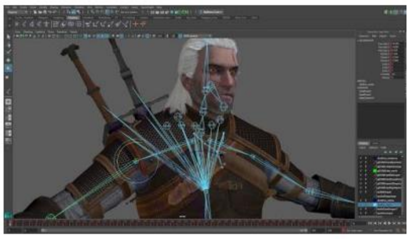
Medical Industry







Video Game Industry



3d Rendering

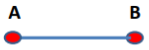


Figure 1

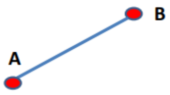


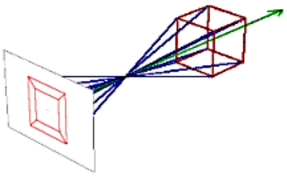
Figure 2. Manipulated line

3D Projection

- The process responsible for flattening 3D coordinates onto a 2D plane.

Wireframe rendering

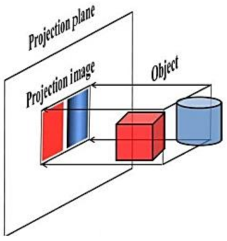
- Once all the 3D points have been converted to 2D, the regular 2D drawing function can be use to connect the dots or points.
- A wireframe is a three-dimensional model that only includes vertices and lines.
- It does not contain surfaces, textures, or lighting like a 3D mesh.
- Instead, a wireframe model is a 3D image comprised of only "wires" that represent three dimensional shapes.
- The process of taking the mathematical model of the world and producing the output image.
- The core of the rendering process involves projecting the 3D models onto a 2D image plane



3D Projection– Orthographic

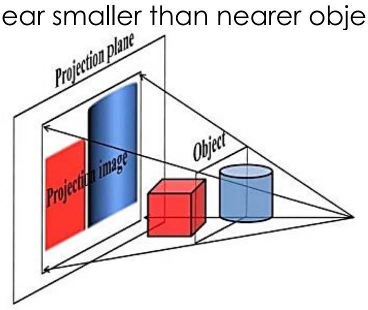
- Orthographic projection is a means of representing three-dimensional objects in two dimensions.
- It is a form of parallel projection, in which all the projection lines are orthogonal to

the projection plane, resulting in every plane of the scene appearing in affine transformation on the viewing surface.



3D Projection– Perspective

- Perspective projection or perspective transformation is a linear projection where three dimensional objects are projected on a picture plane.
- This has the effect that distant objects appear smaller than nearer objects.



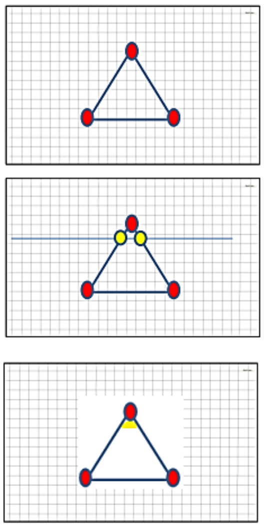
3D Rendering

- polygon mesh is the collection of vertices, edges, and faces that make up a 3D object.
- A polygon mesh defines the shape and contour of every 3D character & object, whether it be used for 3D animated film, advertising, or video games
- Game designers have to carefully balance ***model fidelity vs polygon count*.**
- if the count goes to high, the frame rate of an animation drops below what users perceive as smooth

3D Rendering – Scanline

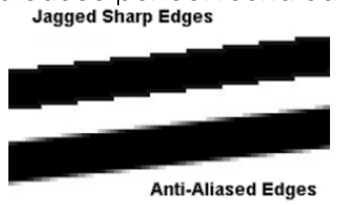
- Polygon-oriented rendering
- is an algorithm for visible surface determination, in 3D computer graphics, that works on a row-by-row basis rather than a polygon-by-polygon or pixel-by-pixel basis.

1. The scanline algorithm starts reading the three points that make up the polygon, and finding the lowest and highest Y values. It will only consider rows between these two points.
2. The algorithm works down one row at a time, in each row, it calculates a line – running through the center of a row, intersects with the side of the polygon.
3. On the first row, we look at the intersection. The algorithm then colors in all pixels between those intersections.
4. And step 2-3 continues row by row and when we reach the bottom of the polygon ,it stops



3D Rendering – Scanline

- **Jaggies** are stair-like lines that appear where there should be "smooth" straight lines or curves.
- **Anti-aliasing** is a technique used by users to get rid of jaggies that form on the screen. Since pixels are rectangular, they form small jagged edges when used to display round edges.
- **Anti-aliasing** tries to smooth out the shape and produce perfect round edges.



3D Rendering – Painter’s Algorithm

- Polygon-oriented.
- All the polygons are sorted by their depth and then displayed in this order.