Unit 6:

Colour Models, Digital Images and Graphical User Interface

Mobile Application Development (MAD)
CCIT4059, 2025-2026

Outline

- Colour Models
  - RGB Model
  - CMY Model
  - HSV Model
- Digital Images
  - Image Resolution, Colour Depth, Compression
  - Image Transparency
  - Raster vs. Vector Graphics
- Developing Android App
  - Graphical User Interface

Colour Models

Revisit

- In computer graphics development, it is important that students also understand the concepts behind, including common colour models and digit image representations.
- There are different ways to represent colours
- Colour Model is an abstract mathematical model describing the way colours can be represented with numbers, often as three or four values or colour components
- 3 common colour models:
  - RGB
  - CMY
  - HSV (also known as HSB)

Colour Models
(RGB Colour Model)

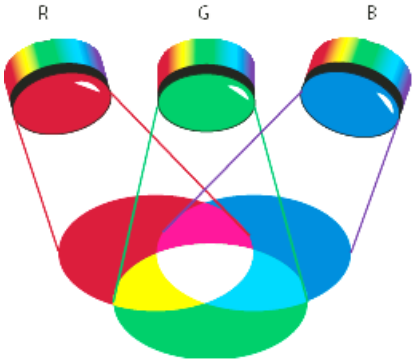
- Light is made up of energy waves that are grouped together on a spectrum called the “electromagnetic energy spectrum”.
- At one end are shorter electromagnetic waves that we perceive as blue; on the other end are longer waves that we perceive as red.
- Additive colours begin as black and become white as more red, blue, or green light is added. TVs, computer monitors, and other electronics use additive colour.

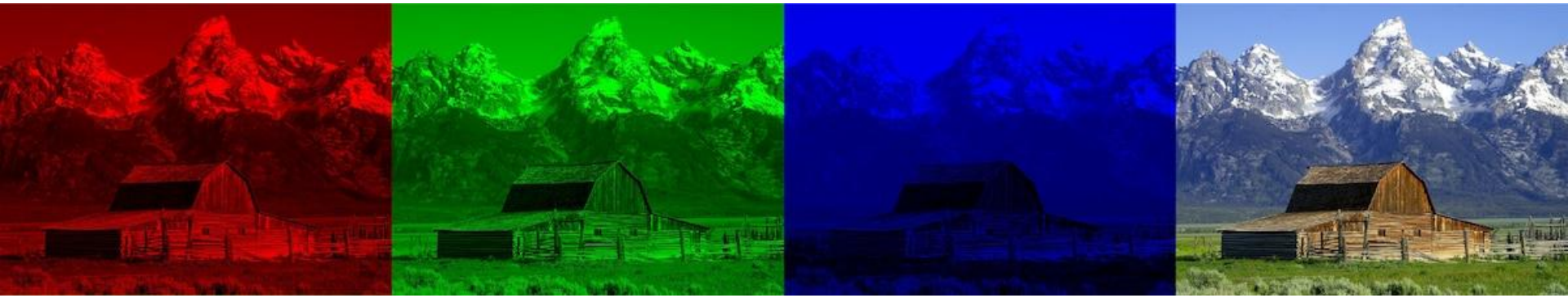
Electromagnetic energy spectrum

Colour Models (RGB Colour Model)

- RGB colour model uses the three primary colours: Red, Green and Blue - This model is an **additive** colour model - for screen displays
- Colours in this model are produced by adding light from different colour sources
- Applying each light channel, red, green and blue, to a full-colour photograph alters its colour appearance.

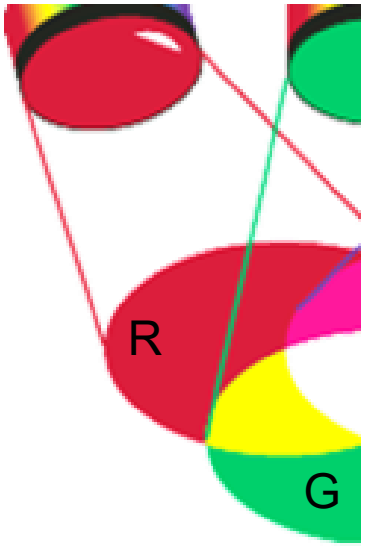
R - Red
G - Green
B - Blue



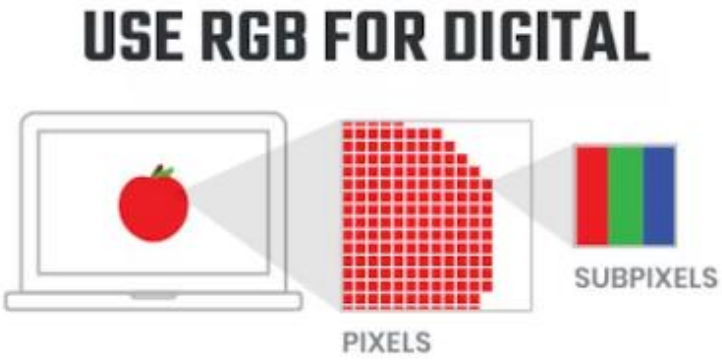


Colour Models
(RGB Colour Model)

- Examples:
  - adding all Red, Green, and Blue **lights** equally will give a neutral (or achromatic/colorless) colour like white or grey
  - adding Red and Green equally gives Yellow

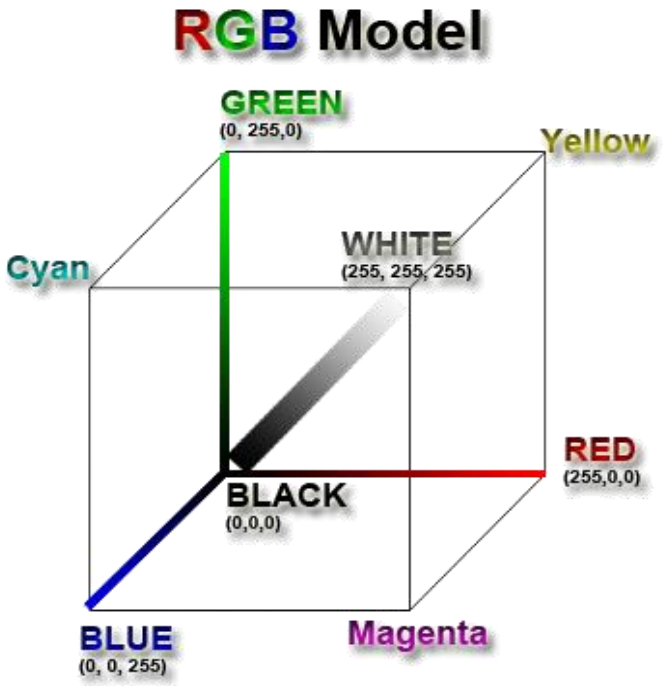


- This is the way our computers and televisions work, and is the standard model used in a colour monitor for TV or computers for screen displays



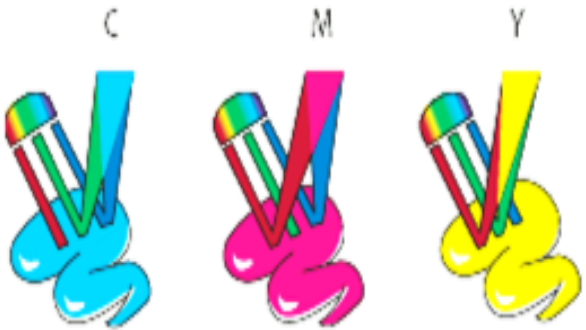
Colour Models
(RGB Colour Model)

- In computer, each colour can be represented with 256 shades (from 0 to 255, in 8-bits or 1-byte) for each RGB component
  - (0, 0, 0) is black; (255, 255, 255) is white
  - (255, 0, 0) is red
  - (0, 255, 0) is green
  - (0, 0, 255) is blue
  - (255, 255, 0) is yellow
  - (0, 255, 255) is cyan
  - (255, 0, 255) is magenta



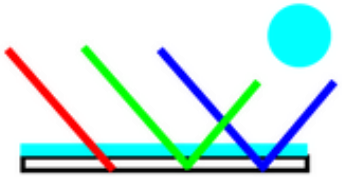
Colour Models (CMYK Colour Model)

- Additive colours are created by adding coloured light to black environment.
- While subtractive colours are created by completely or partially absorbing (or subtracting) some light wavelengths and reflecting others.
- CMY model uses 3 colour components: **Cyan, Magenta, and Yellow.**
- This model is a **subtractive** colour model
  - Each colour is formed by the **mixture of paints**, dyes, inks, and natural colorants which absorb some wavelengths of light and reflecting others
  - E.g. Subtraction of all three CMY inks equally from white yields grey or black

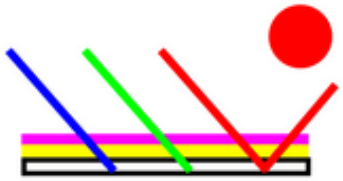


Colour Models
(CMYK Colour Model)

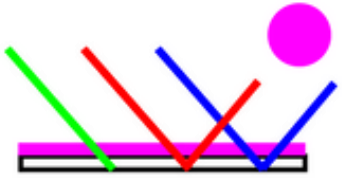
- Subtractive colours begin as white. As you add filters to the white light, such as ink, this white light takes on the appearance of colour.
- How primary CMYK colours mix to achieve the full range of hues:



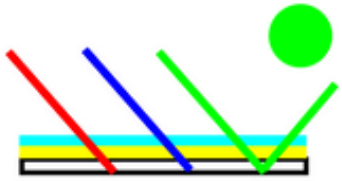
Ink Color: C
Absorbs: R
Reflects: G + B
Appears: C



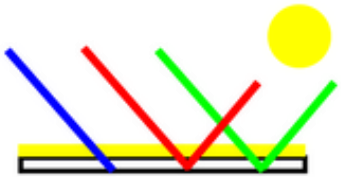
Ink Color: M + Y
Absorbs: G + B
Reflects: R
Appears: R



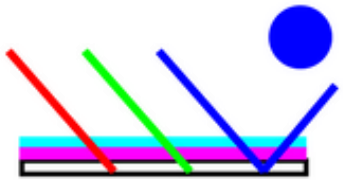
Ink Color: M
Absorbs: G
Reflects: R + B
Appears: M



Ink Color: C + Y
Absorbs: R + B
Reflects: G
Appears: G



Ink Color: Y
Absorbs: B
Reflects: R + G
Appears: Y

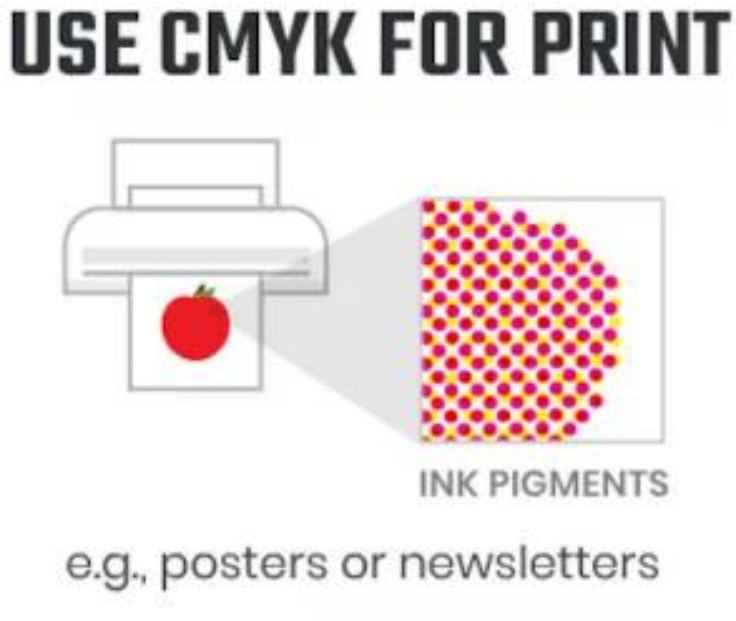


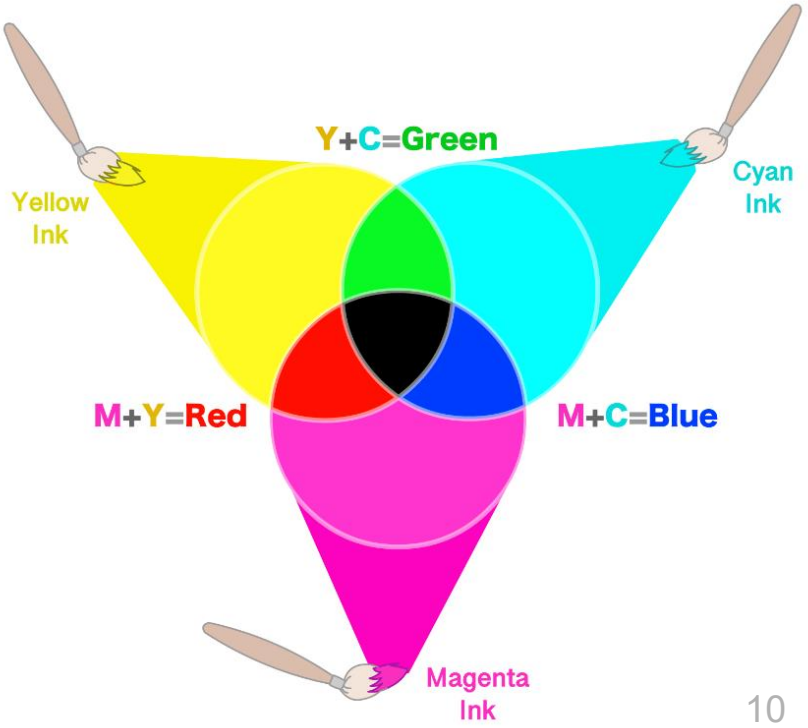
Ink Color: C + M
Absorbs: R + G
Reflects: B
Appears: B

Colour Models (CMYK Colour Model)

C - Cyan
M - Magenta
Y - Yellow

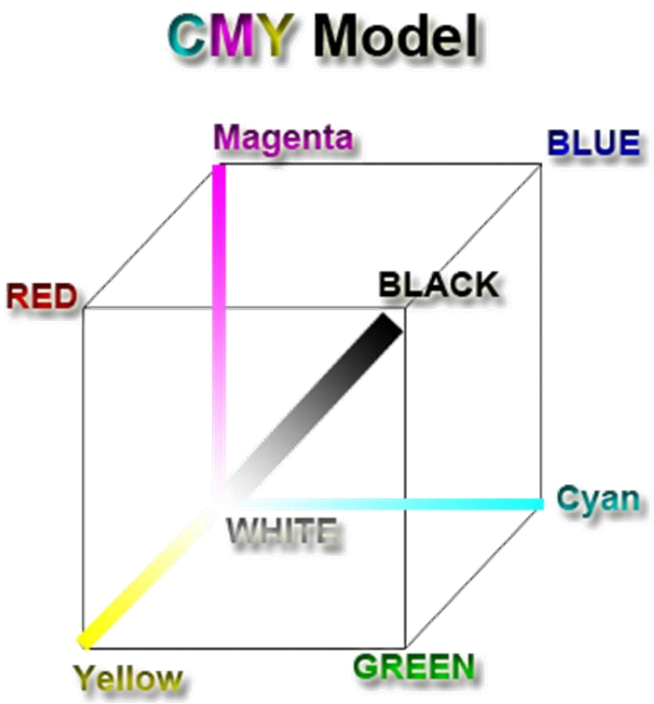
- CMY colour model could be used in colour painting or printer that colours are “subtracted” using the ink of Cyan, Magenta, and Yellow
- Black is often added (to form the CMYK model) because the “black” generated by mixing Cyan, Magenta and Yellow components is unsatisfactory, e.g. in colour printing





Colour Models
(HSV Colour Model)

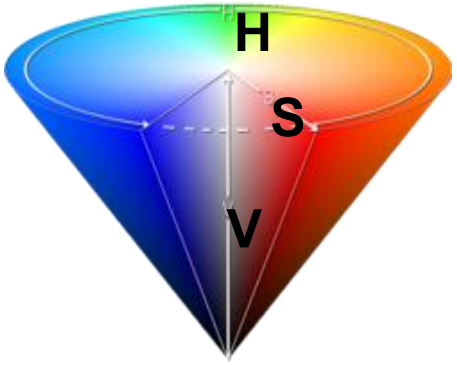
- An example of how applying each ink pigment - cyan, magenta, yellow and black - to a full-colour photograph alters its colour appearance.
- The CMYK model is mostly used for printing or other similar processes



Colour Models (HSV Colour Model)

Revisit

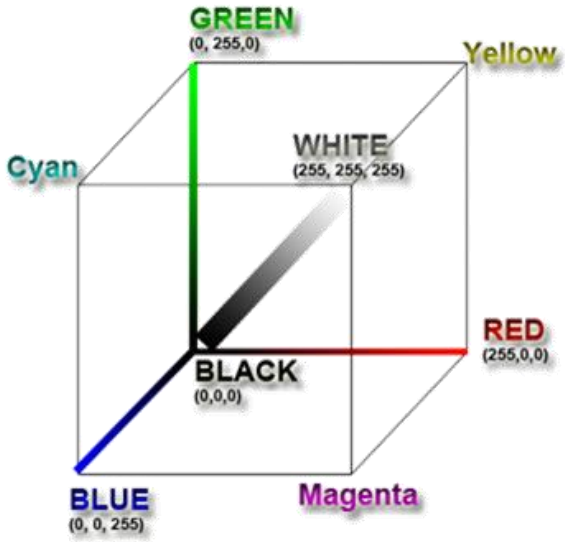
- HSV is a common colour model which could be regarded as another representation of an RGB colour model
- It attempts to describe perceptual colour characteristics more meaningfully than RGB colour model, while remaining computationally simple
- Colours into seven basic hues, each produced by a single wavelength – red, orange, yellow, green, blue, indigo, and violet.
- Saturation" refers to the purity of a hue. Saturation ranges from pure colour (100%) to gray (0%).



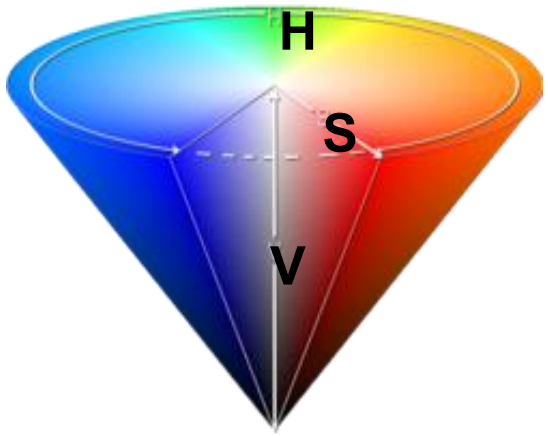
Colour Models (HSV Colour Model)

Revisit

- HSV model can be derived from the RGB cube
  - H: Hue (colour type)
  - S: Saturation (purity of colour)
  - V: Value (brightness of colour)
- HSV colour model can be illustrated with an inverted cone

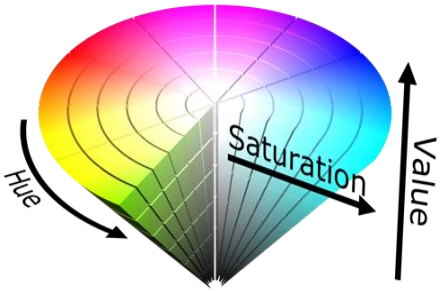


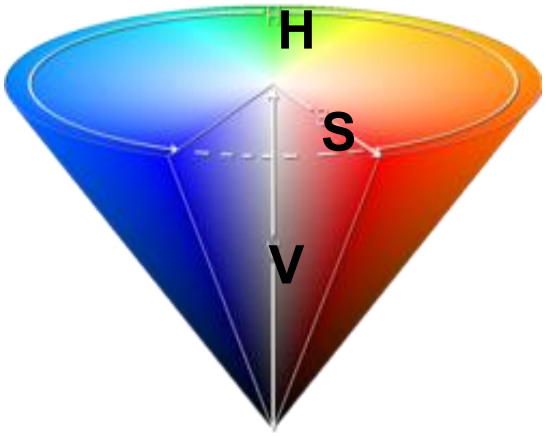
RGB Cube, the colour model



Colour Models
(HSV Colour Model)

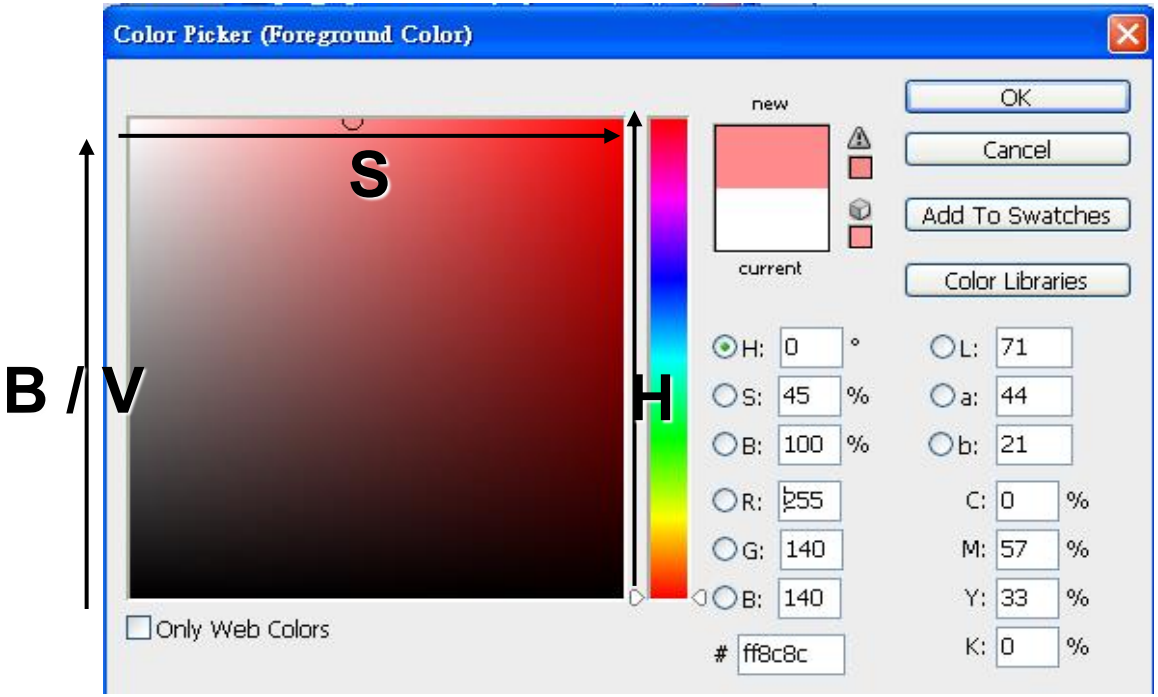
- A 360 degree describes the hue
  - Complementary colours are 180º apart.
  - E.g. Red→0º and Cyan→180º
- Saturation and value are measured as the horizontal and vertical axes
  - Neutral colours (white, grey or black) are obtained if saturation equals zero





Colour Models
(HSV Colour Model)

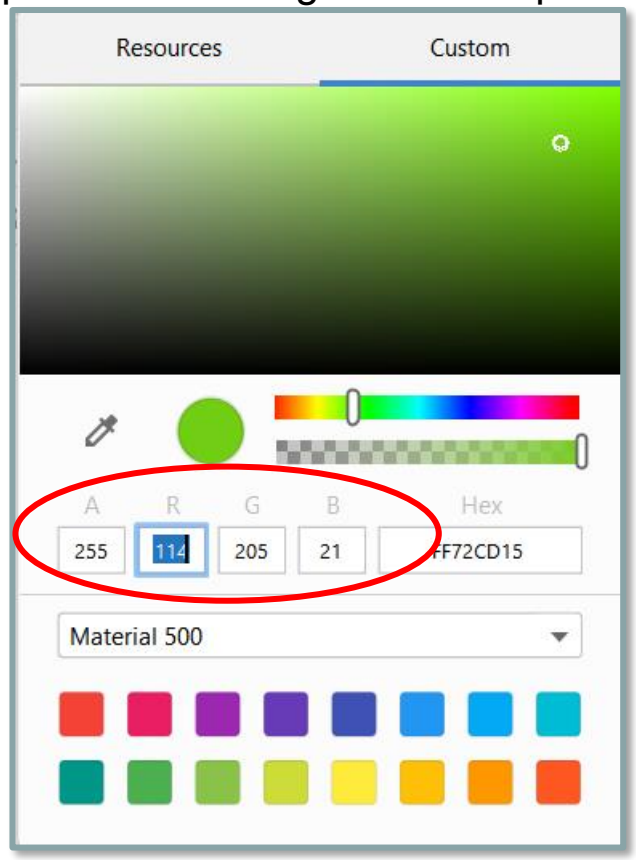
- HSV is also known as HSB, B for Brightness
- Many software provide tools to choose colour with this model, including Color Picker in Android Studio or Photoshop

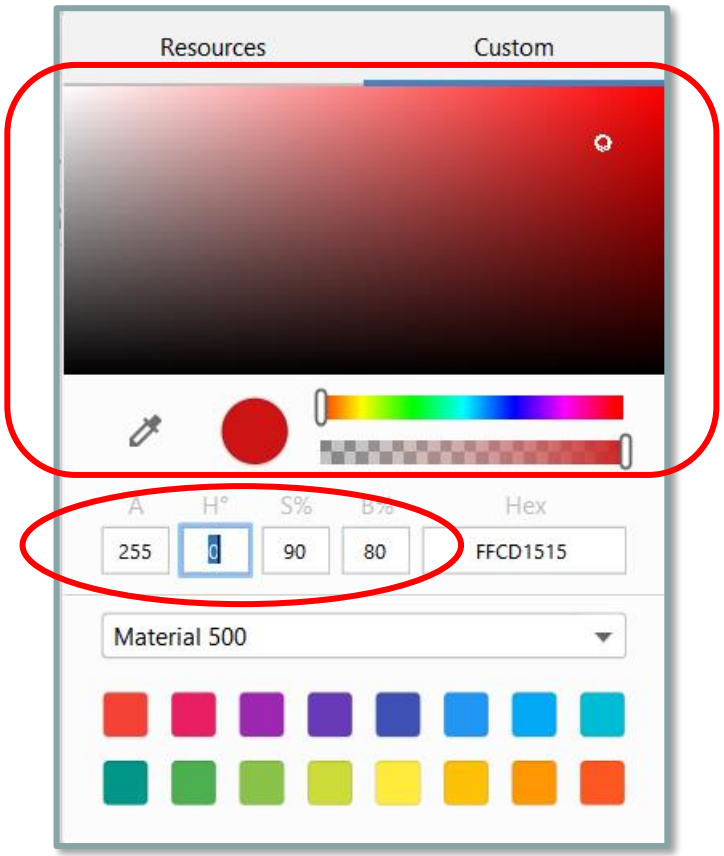


Colour Models
(HSV Colour Model)

- Color Picker in Android Studio
  - Left (RGB mode)
  - Right (HSV/HSB mode)

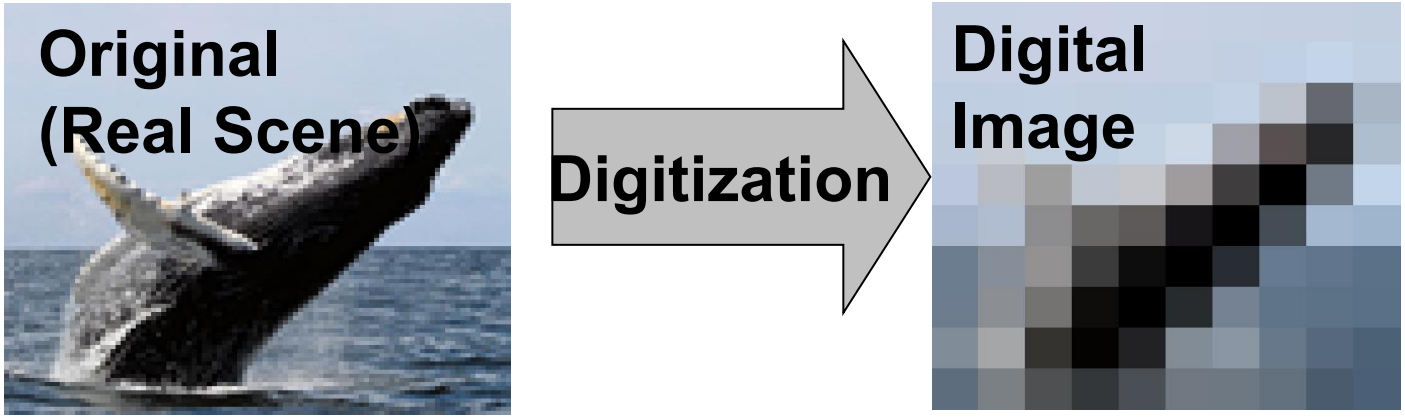
- (The alpha channel “A” (also called alpha planes) is a colour component that represents the degree of transparency)



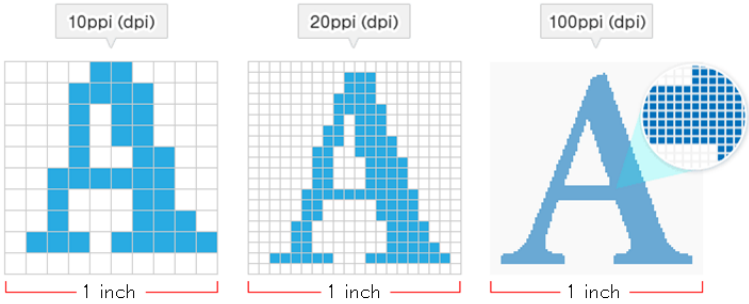


Digital Images

- Digital images, such as our digital photos, could be represented in a form of series of dots or pixels (picture elements) laid out in a 2D grid
- These pixel-based digital images are also called “raster” images or “bitmaps”
- Common major factors and features for representing digital images include:
  - Image Resolution
  - Colour Depth (or bit-depth, bit resolution)
  - Image Compression



Digital Images
(Image Resolution)



- Image Resolution
  - `Width x Height` dimension in pixels
  - Total number of pixels
    - E.g. Total number of pixels for a `640 x 480` image is `307200` pixels

- Pixel count refers to the number of pixels across the length and width of a digital image—that is, the image dimensions in pixels. Pixels, or "picture elements"

- 300 PPI is industry standard quality.

Digital Images
(Image Resolution)

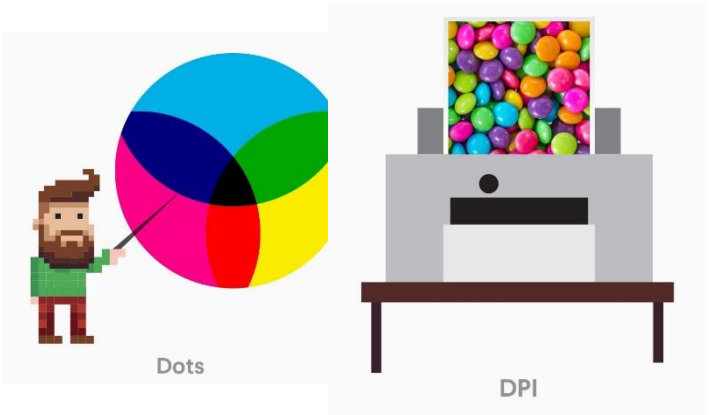
- **PPI, or pixels per inch**, refers both to the fixed number of pixels that a screen can display and the density of pixels within a digital image.
- Given the same visual content, higher image resolution gives much refined details
  - An image with a higher PPI tends to be higher quality because it has a greater pixel density (increases with file size)
  - PPI is most useful in preparing files for printing

A lower PPI resolution results in less detail and a pixelated image

A higher PPI resolution results in more detail and a sharper image

Digital Images
(Image Resolution)

- **DPI, or dots per inch**, refers to the resolution value of a physical printer. Printers reproduce an image by spitting out tiny dots, and the number of dots per inch affects the amount of detail and overall quality of the print.
- DPI uses the CMYK (cyan, magenta, yellow and key/black) colour model to control the amount of red, green, and blue light that is reflected from white paper.



Printer dots mix CMYK inks

300 DPI
High resolution - perfect for printing

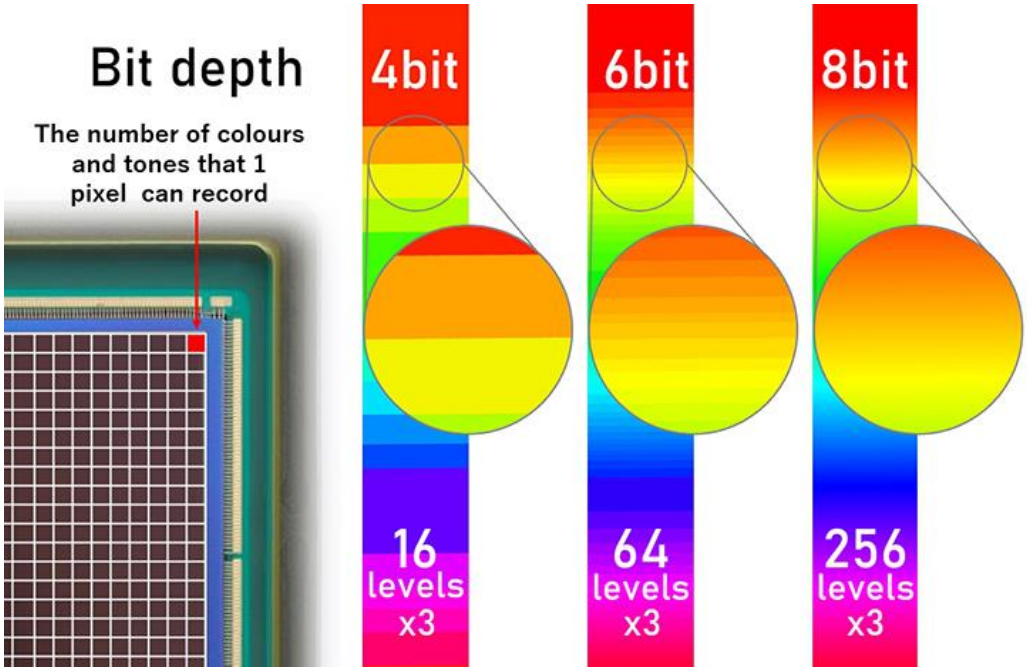
72 DPI
Low resolution - not suitable for printing

DPI describes the amount of detail in an image based on the concentration of printer dots

Digital Images (Colour Depth)

- Each pixel contains a specific colour or grayscale information, within a colour or grayscale range
- This refers to the number of bits (termed colour-depth) devoted to store colour or intensity information about a pixel
  - 1 bit (1) → 2^1 = 2 possible shades / colours
  - 8 bit (11010001) → 2^8 = 256 shades / colours

* Remark: 1 byte = 8 bits



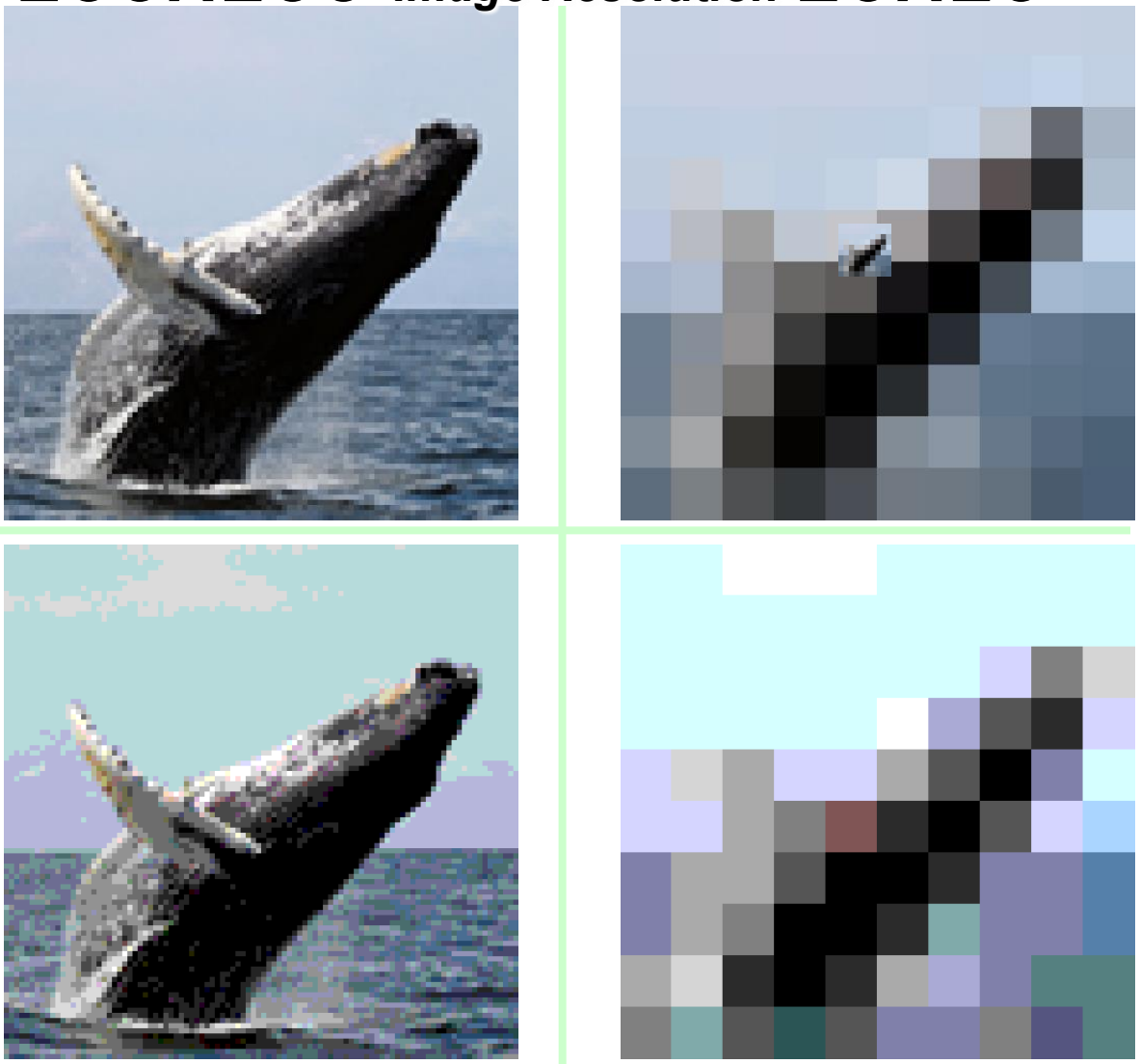
Digital Images
(Examples of Image Resolution and Colour Depth)

100x100 ***Image Resolution*** 10x10

24 bits (3x8)

Colour Depth

9 bits (3x3)



Digital Images
(Image Compression)

- A raw image is often compressed with image compression techniques (for reducing file sizes), and then written to memory as an image file
- There are different image compression techniques and formats, and they can be categorized as *lossless* and *lossy.*

Digital Images
(Image Compression)

- There are two methods of compression – lossy and lossless.
- *Lossless* compression can retain all captured information of the original raw image. (Reduces file size by removing unnecessary metadata.)
  - However, the compressed file size is comparatively large
- *Lossy* image compression loses image information in certain degree (Reduces file size by permanently removing some of the original data.)
  - Lossy image compression can achieve a higher compression ratio
    - e.g. A typical JPEG can compress from the original raw image to its one-tenth size easily, with a reasonable quality

Lossy Compression

- Lossy compression is typically used when a file can afford to lose some data, and/or if storage space is limited

- Here, an algorithm scans image files and reduces their size by discarding information considered less important or undetectable to the human eye.

- Using lossy methods therefore requires you to make a balanced judgement between:
  - Storage/delivery requirements
  - Loading times (e.g. on the web)
  - Image quality

The Difference between two compressions

	Lossy	Lossless
Pros	Small file sizes. Ideal for web use. Lots of tools, plugins and software support it.	No loss in quality. Slight decreases in file sizes.
Cons	Quality degrades due to higher rate of compression.	Compressed files are larger than lossy files.

	Lossy	Lossless
What it does	Permanently removes file data.	Restores and rebuilds compressed data.
When it's used	When file information loss is acceptable.	When file information loss is unacceptable.
Applications	Images, video, audio	Text, images, audio
File types	Images: JPEG Video: MPEG, AVC, HEVC Audio: MP3, AAC	Images: RAW, BMP, PNG General: ZIP Audio: WAV, FLAC

Image File Size

Image File Size (in bytes):
(Assume that there is no image compression)

$$\frac{\text{Number of Pixels in Image} \times \text{Color Resolution (in bits)}}{8}$$

	Image Size in Pixels	Screen Size	Color Resolution (in Bits)	Number of Available Colors	Approximated File Size (in Bytes)
1.	640 x 480	Full screen	8	256	307, 200
2.	320 x 240	Quarter screen	8	256	76, 800
3.	1024 x 768	Full screen	24	16.7 million	2, 359, 296

- To convert **bytes** to **megabytes (MB),** File size (in Bytes) is divided by 1,048,576 (1 MB = 1024 * 1024 bytes)

- Image file size (in MB) = 307,200 / 1,048,576
  = 0.29 MB (approx.)

Image File Size – Compression Ratio

$$\text{Compression Ratio} = \frac{\text{Uncompressed File Size}}{\text{Compressed File Size}}$$

- Example: When a 10 MB file is compressed to 2 MB, the compression ratio is (10 / 2) = 5.

Image File Size (in bytes):
(Assume that the compression ratio is CR)

$$\frac{\text{Number of Pixels in Image} \times \text{Color Resolution (in bits)}}{8 \times \text{CR}}$$

Example: From Case 1), File Image file size (in MB) when the CR is 5:

$$= \frac{640 \times 480 \times 8}{8 \times 5}$$

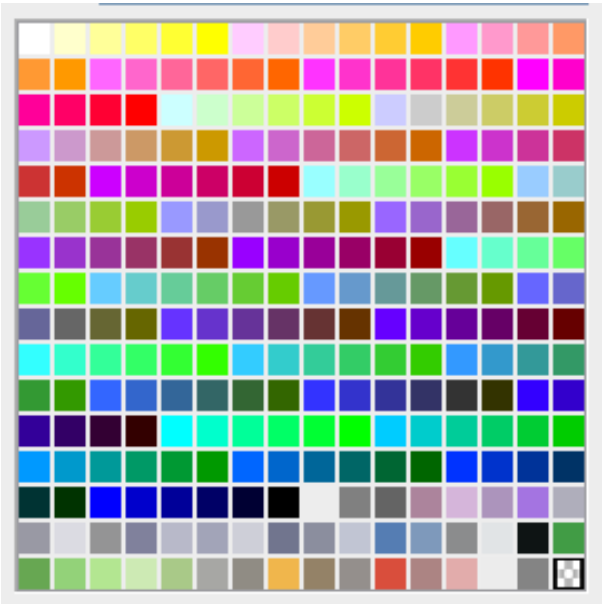
= 61440 Bytes
= 61440 / 1,048,576
= 0.06 MB (approx.)

Digital Images
(Image Transparency)

- In developing mobile apps with composing different layers of visual elements together, it is often required to represent and manipulate transparency data
- In order to composite layers of image elements, it is necessary to keep an associated transparent information for each element of the composition
- There are two common techniques for representing transparency:
  - Transparent Pixel (only binary transparency)
  - Alpha Channel (support partial transparency)

Digital Images
(Image Transparency)

- With ***Transparent Pixels***, digit images have binary transparency (pixels are either fully transparent or fully opaque)
- This transparent pixel is often associated with Indexed colour (through colour palette or colour map table)
- E.g. In GIF images, an index can be designated as a "transparent pixel". Any pixel assigned this index is treated as fully-transparent.



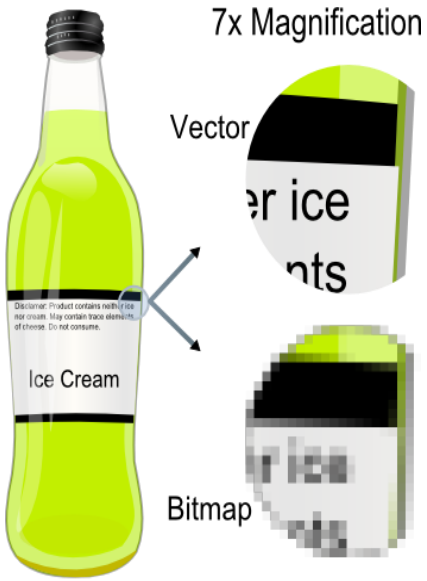
Digital Images
(Image Transparency)

- Colour information in digital images is generally contained in three colour channels: red, green, and blue
- In addition, an image can include an extra fourth channel, called an ***Alpha Channel***, that contains transparency information, e.g. in PNG files
- An alpha channel provides a way to store images and their transparency information in a single file without disturbing the original colour channels
- Together with original RBG color model, we may have an extended ARGB color model with the separate Alpha channel component.

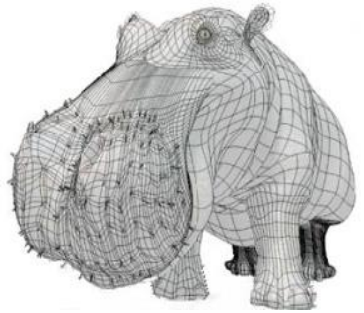
Digital Images
(Image Transparency)

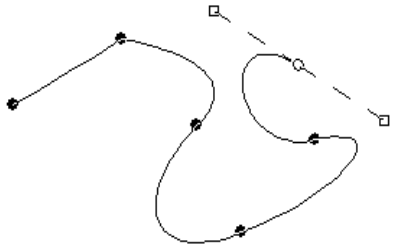
- Transparency data is stored in alpha channel, with a value between **`min`** and **`max`**, e.g. **`0.0`** to **`1.0`**, **`0%`** to **`100%`**, or **`0`** to **`255`** in 8-bits
- **`0.0`** (**`0%`** or **`0`** in 8-bits) indicates fully transparent; **`1.0`** (**`100%`** or **`255`** in 8-bits) indicates fully opaque
- **`0.5`** indicates a half transparency (**`50%`** or **`127`** in 8-bits)
- When presenting transparency with color in an alpha channel, white (value 255 in 8-bits) is often used to indicate fully opaque, black (value `0` in 8-bits) indicates fully-transparent, and shades of gray indicate partially transparent.

Digital Images (Raster vs. Vector Graphics)



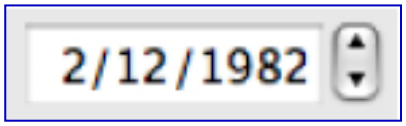
- Raster images / Graphics (pixel-based or bitmap images composed of pixels) suffer from a problem - resolution-dependent
  - Cannot retain the quality after rescaling (esp. enlarging)
- Vector objects in Vector Graphics are based on mathematical formulas.
  - They are resolution-independent (fully scalable), and could be retained in high-quality output at different resolutions
  - They could be easily rescaled and transformed with their mathematical representations
  - E.g. 3D geometric modeling used for industrial products, films and animations)





Developing Android App (Graphical User Interface)

- The Graphical User Interface (GUI) is everything that the user can see and interact with
- Android provides pre-built UI components such as structured layout objects and input controls that you can use to build your UI
  - Other modules for special interfaces such as dialogs, notifications and menus
- Visual elements and UI objects must typically be larger (to be an adequate target for touches) while at the same time make as efficient use of screen as possible
- E.g.: Date picker object which is from a class that appears on both mobile and desktop platforms. It is typically larger for finger interaction (better than direct data input (number)) on the screen.



Developing Android App (Graphical User Interface)

- Android runs on a variety of devices that offer ***different screen*** sizes and densities

- Related terms and issues to be considered when developing apps:
  - Screen Size:
    - Actual physical size, measured as the screen's diagonal
  - Screen Density:
    - Pixels within screen area; usually referred to as **`dpi`** (dots per inch)
  - Orientation:
    - Orientation of the screen from the user's point of view, either landscape or portrait
  - Resolution:
    - Total number of pixels on screen (width and height)
  - Density-independent pixel (dp):
    - virtual pixel unit used to express UI layout dimensions or position in a density-independent way
  - Scale-independent pixels (sp), in particular for text
    - Same size as dp, by default, but it resizes based on the user's preferred text size.

Android Views

- The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects.

- A View is a simple building block of a user interface.

- View objects are usually UI widgets or elements such as buttons or text fields and ViewGroup objects are invisible view containers that define how the child views are laid out, such as in a grid or a vertical list.

- Android provides an XML setting file that corresponds to the subclasses of View and ViewGroup so you can define your UI in XML using a hierarchy of UI elements.

Android Views

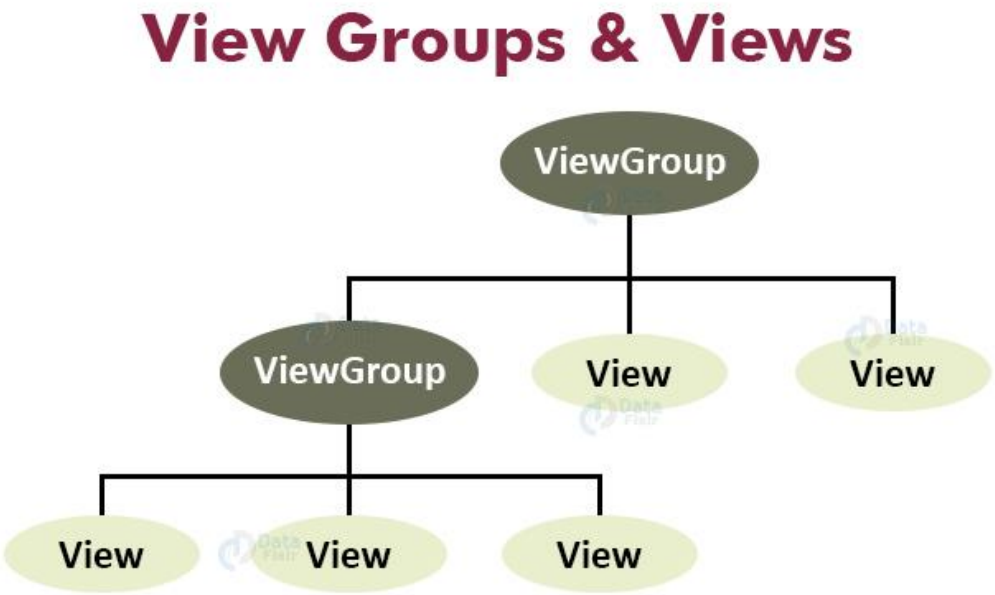
- A view can be easily implemented in an Application using the Java code. Its creation is more easy in the XML layout file of the project. (TextView, EditText, or even a button).
- It occupies the area on the screen in a rectangular area and is responsible for drawing and event handling.
- View is a superclass of all the graphical user interface components.

Types of Android Views

- Different types of views that are available in Android for the developers:
  - TextView
  - EditText
  - Button
  - Image Button
  - Date Picker
  - RadioButton
  - Image View

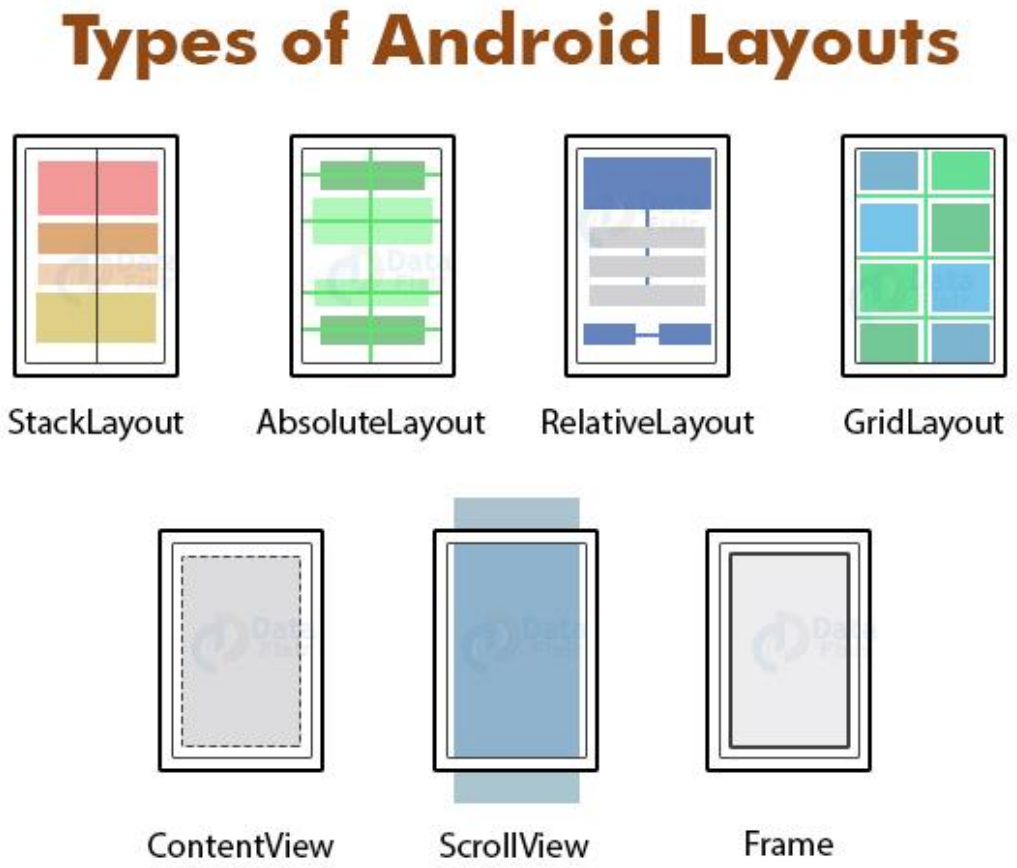
What is Android View Group?

- A View Group is a subclass of the View Class and can be considered as a superclass of Layouts.
- It provides an invisible container to hold the views or layouts. View Group instances and views work together as a container for Layouts.
- To understand in simpler words it can be understood as a special view that can hold other views that are often known as a child view.



Types of Layouts in Android

- Android Studio supports a variety of layout types, each serving different purposes and providing flexibility in UI design.
  - **Linear Layout**
  - **Relative Layout**
  - **Constraint Layout**
  - **Table Layout**
  - **Frame Layout**
  - **List View**
  - **Grid View**
  - Absolute Layout
  - **WebView**
  - ScrollView



The pictorial representations of different layouts

Types of Layouts in Android

- ConstraintLayout
  - ConstraintLayout is a more advanced and versatile layout that allows you to create large and complex layouts with a flat view hierarchy. It is similar to RelativeLayout but more powerful and efficient.
- LinearLayout
  - LinearLayout aligns all children in a single direction - vertically or horizontally. It is simple to use and ideal for creating straightforward, linear UIs.
- RelativeLayout
  - RelativeLayout positions its children relative to each other or the parent. This layout type provides a flexible way to place widgets in relation to each other.

Types of Layouts in Android

- FrameLayout
  - FrameLayout is designed to block out an area on the screen to display a single item. Multiple children can be added, but they will overlap each other.
- TableLayout
  - TableLayout arranges its children into rows and columns, similar to an HTML table. It is useful for creating grid-like layouts.
- GridLayout
  - GridLayout places its children in a rectangular grid. It is highly flexible and helps in creating UIs where the elements need to be aligned in a grid format.

1. Constraint Layout

- Constraint Layout is a layout used in Android view in which we set various constraints for various views with respect to screen or various views.
  - It enables us to build layouts without grouping multiple view groups.

```
android:layout_alignParentTop= “true”

android:layout_alignParentLeft= “true”
```

- If we write the above code, the element will get aligned on the top left of the parent container.
- Various constraints for a particular view in Constraint Layout are:

```
app:layout_constraintLeft_toLeftOf:

app:layout_constraintRight_toRightOf:

app:layout_constraintBottom_toBottomOf:

app:layout_constraintTop_toTopOf:
```

2. Linear Layout

- Linear layout places the elements in a linear manner. A Linear manner means one element per line. This layout creates various kinds of forms on Android. In this, arrangement of the elements is in a top to bottom manner.
- This can have two orientations:
  - Vertical Orientation – It is shown above where the widgets such as TextViews, and all in a vertical manner.
  - Horizontal Orientation – It is shown above where the widgets such as TextViews, and all in a horizontal manner.

3. Relative Layout

- This layout is for specifying the position of the elements in relation to the other elements that are present there.
- In the relative layout, alignment of the position of the elements to the parent container is possible. To define it in such a way, we write the following:

```
android:layout_alignParentTop= “true”

android:layout_alignParentLeft= “true”
```

- If we write the above code, the element will get aligned on the top left of the parent container.
- If we want to align it with some other element in the same container, it can be defined is as follows:

```
android:layout_alignLeft= “@+id/element_name”

android:layout_below= “@+id/element_name”
```

- This will align the element below the other element to its left.

Layout setting in a xml layout file

- A Layout is defined and the attributes for customizing a Layout are included.

```
<?xml version="1.0" encoding="utf-8"?>

  <LinearLayout
       android:id="@+id/layout2"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:layout_weight="1"
       android:background="#8ED3EB"
       android:gravity="center"
       android:orientation="vertical" >

       <TextView
           android:id="@+id/textView4"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_marginLeft="10dp"
           android:layout_marginTop="-40dp"
           android:fontFamily="@font/almendra_bold"
           android:text="This is a TextView" />

  </LinearLayout>
```

Layout Attribute Setting in an XML Layout File

- `android:id`: It uniquely identifies the Android Layout.
- `android:layout_height`: It sets the height of the layout.
- `android:layout_width`: It sets the width of the layout.
- `android:layout_gravity`: It sets the position of the child view.
- `android:layout_marginTop`: It sets the margin from the top of the layout.
- `android:layout_marginBottom`: It sets the margin from the bottom of the layout.
- `android:layout_marginLeft`: It sets the margin of the from the left of the layout.
- `android:layout_marginRight`: It sets the margin of the from the right of the layout.
- `android:layout_x`: It specifies the x coordinates of the layout.
- `android:layout_y`: It specifies the y coordinates of the layout.

Layout Setting in an XML Layout File

- Creating Layouts
  - Layouts in Android Studio can be created either by writing XML code or by using the graphical layout editor. A combination of both is used.
- XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. (Derived from Standard Generalized Markup Language(SGML).)
  - Developer is required to implement and define the tags in XML. XML tags define the data and used to store and organize data.In Android, the XML is used to implement UI-related data,
- XML Layouts
  - XML-based layouts are defined in XML files. These files reside in the res/layout directory and define the UI structure in a declarative manner.
- Graphical Layout Editor
  - The graphical layout editor in Android Studio allows you to drag and drop UI elements onto a visual design surface. This tool provides a convenient way to create and adjust layouts without writing XML code directly.

References

- Part of this slide set is prepared and extracted from the followings:
  - Baker, H. (2004) “Computer Graphics with OpenGL”, 3rd Edition, Prentice Hall
  - Bouweraerts, D (2005) “Introduction to Computer Graphics - Design Professional”, Course Technology
  - Coorough, C. & Shuiman, J. (2006) “Multimedia For The Web Revealed”, Thomson Learning.
  - Fling, B. (2009) *‘Mobile Design and Development: Practical techniques for creating mobile sites and web apps’*, O’Reilly Media, Inc.
  - Galer, M. & Horvat, L. (2005) “Digital imaging”, 3rd Edition, Boston : Focal Press
  - Hoober, S. and Berkman, E (2012) *‘Designing Mobile Interfaces: Patterns for Interaction Design’*, O’Reilly Media, Inc. (Website: http://4ourth.com/wiki)
  - Jones, M. and Marsden, G. (2006) ‘*Mobile Interaction Design*’, John Wiley & Sons Ltd.
  - Neil T. (2014) *Mobile Design Pattern Gallery (2ed)*, O’Reilly Media, Inc.
  - Nolan, G., Cinar, O. and Truxall, D. (2014) *‘Android Best Practices’*, Apress Media
  - Overmars M. (2011) *‘Designing Successful iPhone and Android Games with GameMaker’*: http://www.yoyogames.com/docs/designing_successful_iphone_games.pdf
  - Smith, K. (2005, ed.) “Handbook of visual communication : theory, methods, and media”, N.J. : Lawrence Erlbaum Associates Inc.
  - Tidwell, J. (2011) *‘Designing Interfaces’* (2ed), O’Reilly Media, Inc.
  - Apple Inc. (2013-10-22) ‘iOS Human Interface Guidelines’, https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf
  - Google Inc. ‘Website for Android Developers’, http://developer.android.com/
  - Google Inc. ‘Website for Android Sources’, https://source.android.com/
  - Apple Inc. ‘Website for Apple Developers’, Website: https://developer.apple.com/
  - Microsoft Corp. ‘Website for Windows Developers’, https://developer.microsoft.com/en-us/windows
  - Wikipedia Website: http://en.wikipedia.org

- This set of slides is for teaching purpose only