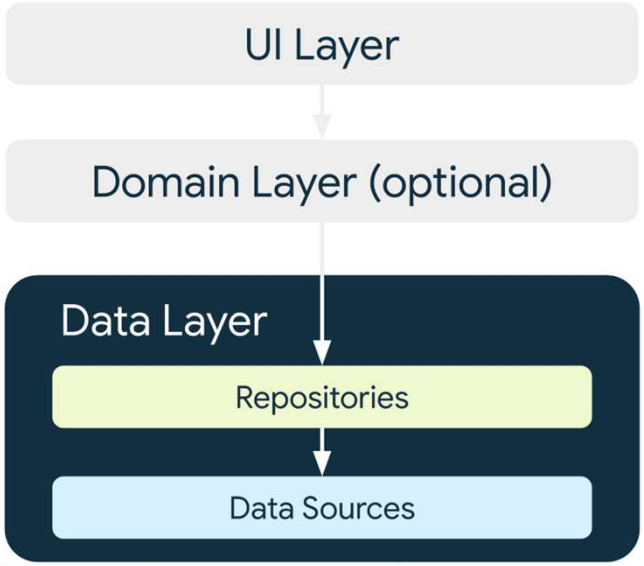
DATA LAYER AND DATA SOURCES (PART 1)

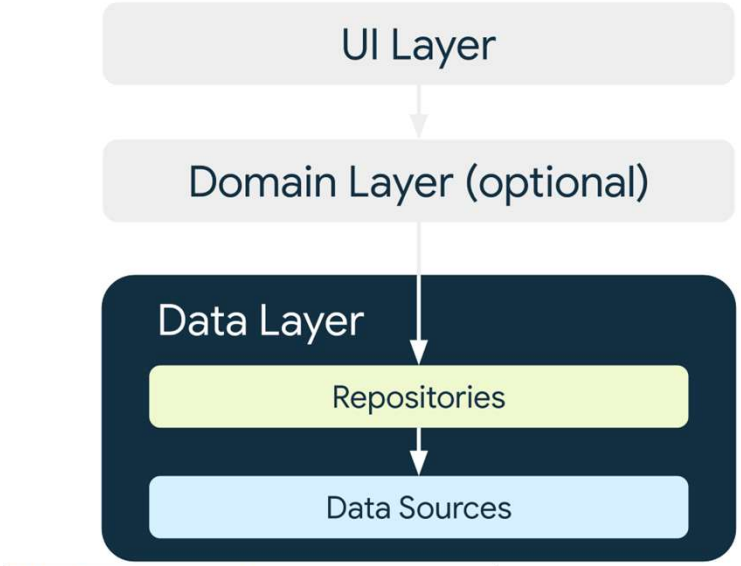
INTRODUCTION




```
interface UserRepository {
    suspend fun getUser(id: String): User
    suspend fun saveUser(user: User)
}
```

- In the UDF architecture, the Data Layer manages all data sources (local or remote), i.e. how data is obtained, stored, updated, etc.
- The connection with the domain layer is provided through a repository, which implements an interface defining all operations on data.
- The implementation of the interface is data-source specific. And can be modified without affecting any other part of the software
- The driving rule of its implementation is the offline-first principle, meaning the app should continue to work even without a network connection.

INTRODUCTION



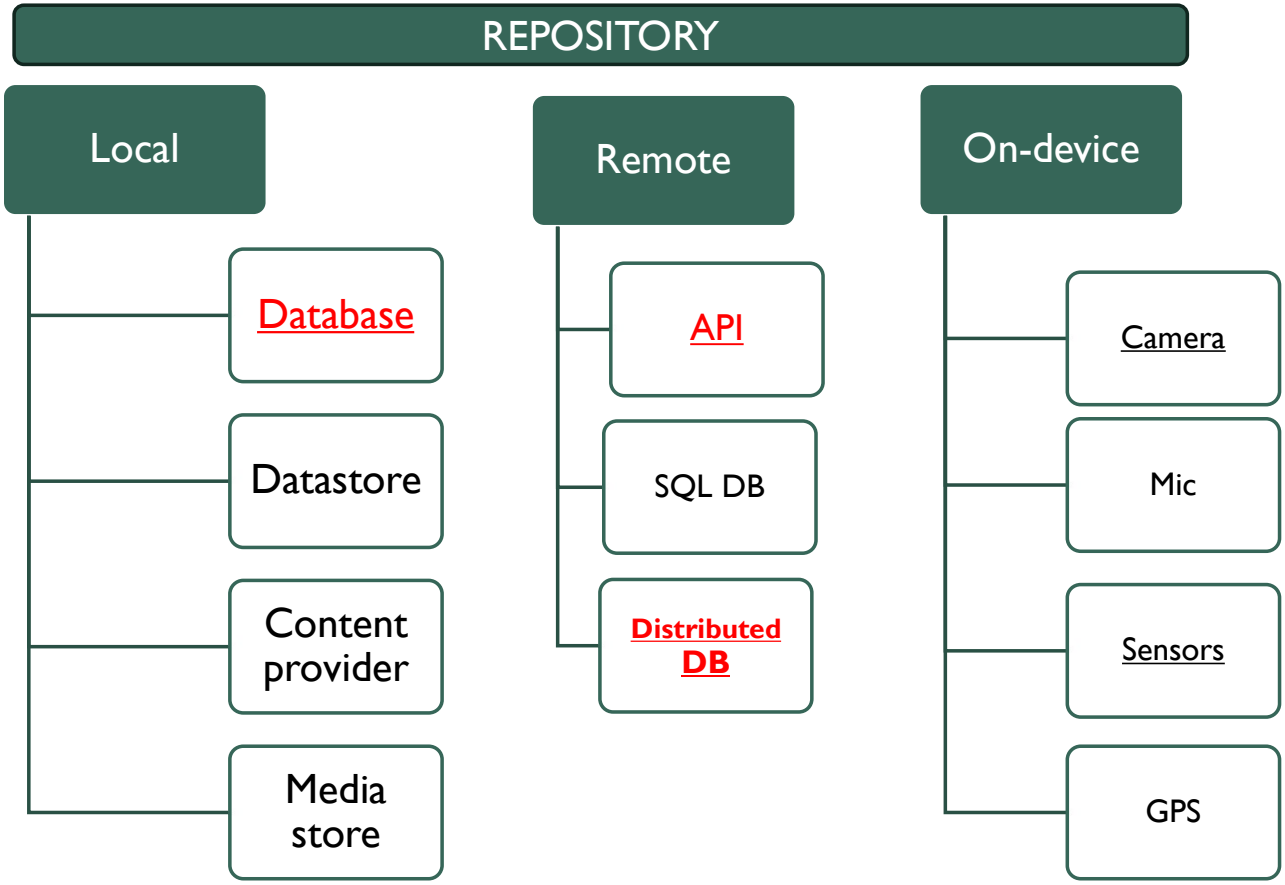
- The driving rule of its implementation is the offline-first principle, meaning the app should continue to work even without a network connection
- For example, cache data locally and keep it consistent with the remote source when the connection becomes available.

```
interface UserRepository {
    suspend fun getUser(id: String): User
    suspend fun saveUser(user: User)
}
```

```
class UserRepositoryImpl(
    private val remote: UserRemoteDataSource,
    private val local: UserLocalDataSource
) : UserRepository {
```

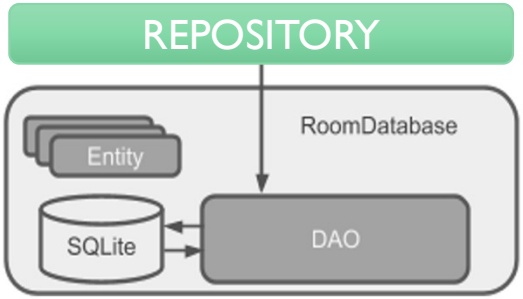
```
override suspend fun getUser(id: String): User {
    val cached = local.getUser(id)
    return cached ?: remote.getUser(id).also { local.saveUser(it) }
}
```

TAXONOMY OF DATA SOURCES



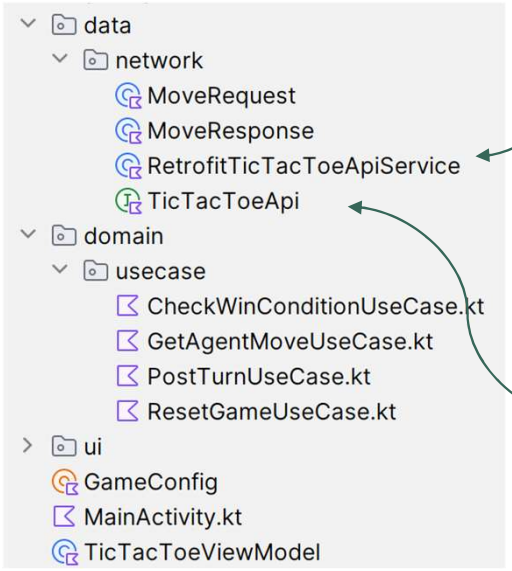
LOCAL DATA BASE

- Persistent
- structured data (SQL,document-based, key-value)
- on-demand access to query, read and write
- Low access rate
- ACID semantic consistency (SQL) or eventual consistency (noSQL)
- Example: Room / SQLite (Room is a library recommended to interact with the DB based on annotations)




```
@Dao
interface UserDao {
    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)

    @Query("SELECT * FROM user")
    fun getAll(): List<User>
}
```

EXAMPLE OF REMOTE DATA SOURCE

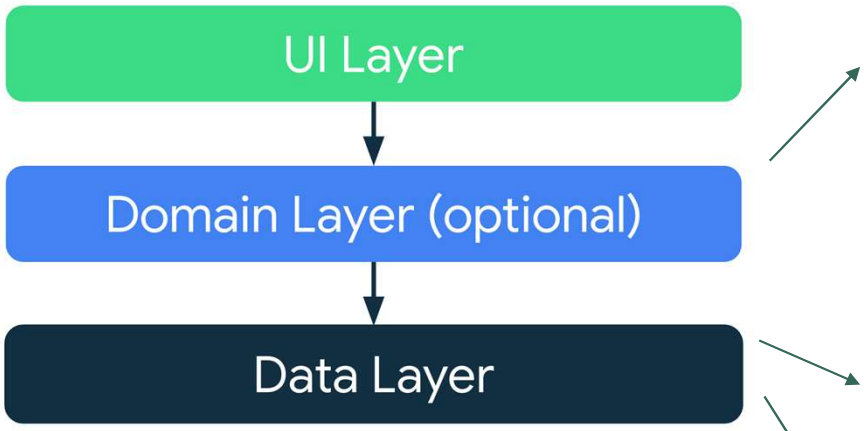



```
private val api = retrofit.create(TicTacToeApi::class.java)  3 Usages

override suspend fun getMove(board: Board): Pair<Int, Int>? {  1 Usage
    return try {
        val response = api.getMove(MoveRequest(board))
        Pair( first = response.move[0],  second = response.move[1])
    } catch (e: Exception) {
        Log.e( tag = "ApiService",  msg = "Error calling getMove API",  tr = e)
        null
    }
}
```

```
interface TicTacToeApi {  1 Usage
    @POST( value = "move")  1 Usage
    suspend fun getMove(@Body moveRequest: MoveRequest): MoveResponse

    @POST( value = "turn")  1 Usage
    suspend fun postTurn(@Body moveRequest: MoveRequest)

    @POST( value = "reset")  1 Usage
    suspend fun reset()
}
```

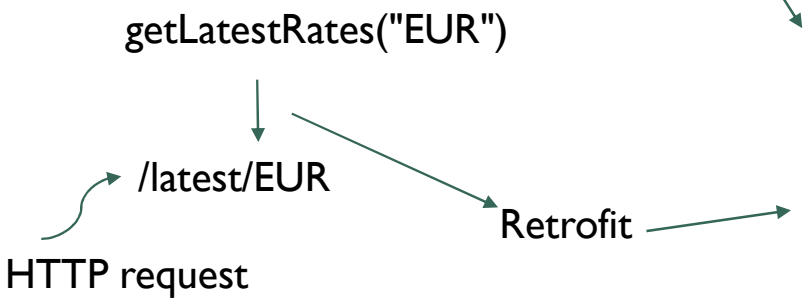
Annotations avoid the implementations which is delegated to Retrofit

ANOTHER OF REMOTE ACCESS VIA API (DEMO)




```
interface ExchangeRateRepository {
    suspend fun getRates(base: String): List<Rate>
}
```

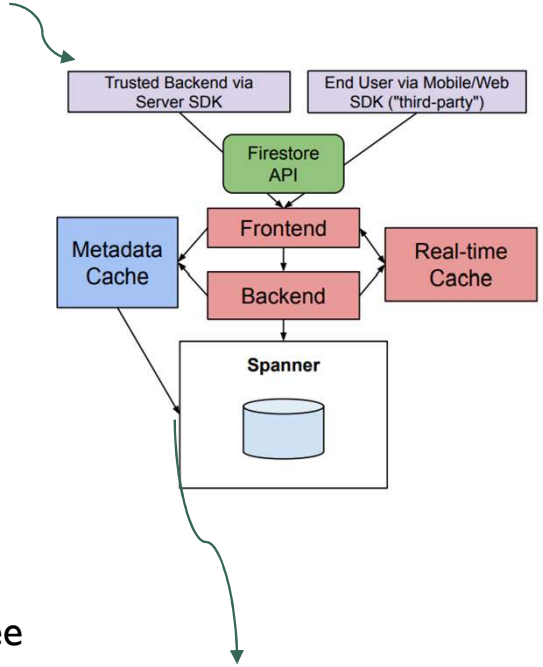
```
class ExchangeRateRepositoryImpl(
    private val apiService: ApiService
) : ExchangeRateRepository {
    override suspend fun getRates(base: String): List<Rate> {
        val response = apiService.getLatestRates(base)
        return response.conversionRates.map { (currency,
rate) ->
            Rate(currency, rate)
        }
    }
}
```




```
interface ApiService {
    @GET("latest")
    suspend fun getLatestRates(@Query("base") base:
String): RatesResponse
}
```

REMOTE DATA SOURCE (FIRESTORE)

"trusted" clients (es. Cloud Functions).



- Firestore is scalable, multi-regional real-time NoSQL database by Google
  - real-time data synchronization, automatic scaling and offline support
  - Manage off-line and strong consistency
  - it ensures Consistency Partition tolerance (CP), but not Availability (it can respond with an exception)
- Simplified integration via SDK

```
dependencies {
    implementation(platform(libs.firebase.bom))
    implementation(libs.firebase.firestore)
}
```

For details see

Firestore: The NoSQL Serverless Database for the Application Developer, Ram Kesavan and David Gay and Daniel Thevessen and Jimit Shah and C. Mohan, IEEE 39th International Conference on Data Engineering (ICDE), 2023

REMOTE DATA SOURCE (FIRESTORE)

```
val db = Firebase.firestore

val user = mapOf(
    "name" to "Fred",
    "age" to 34,
    "role" to "Professor"
)

db.collection("users")
    .document("fred123")
    .set(user)
    .addOnSuccessListener {
        Log.d("FIRESTORE", "Document successfully written!")
    }
    .addOnFailureListener {
        Log.e("FIRESTORE", "Error writing document: ${it.message}"
    }
```

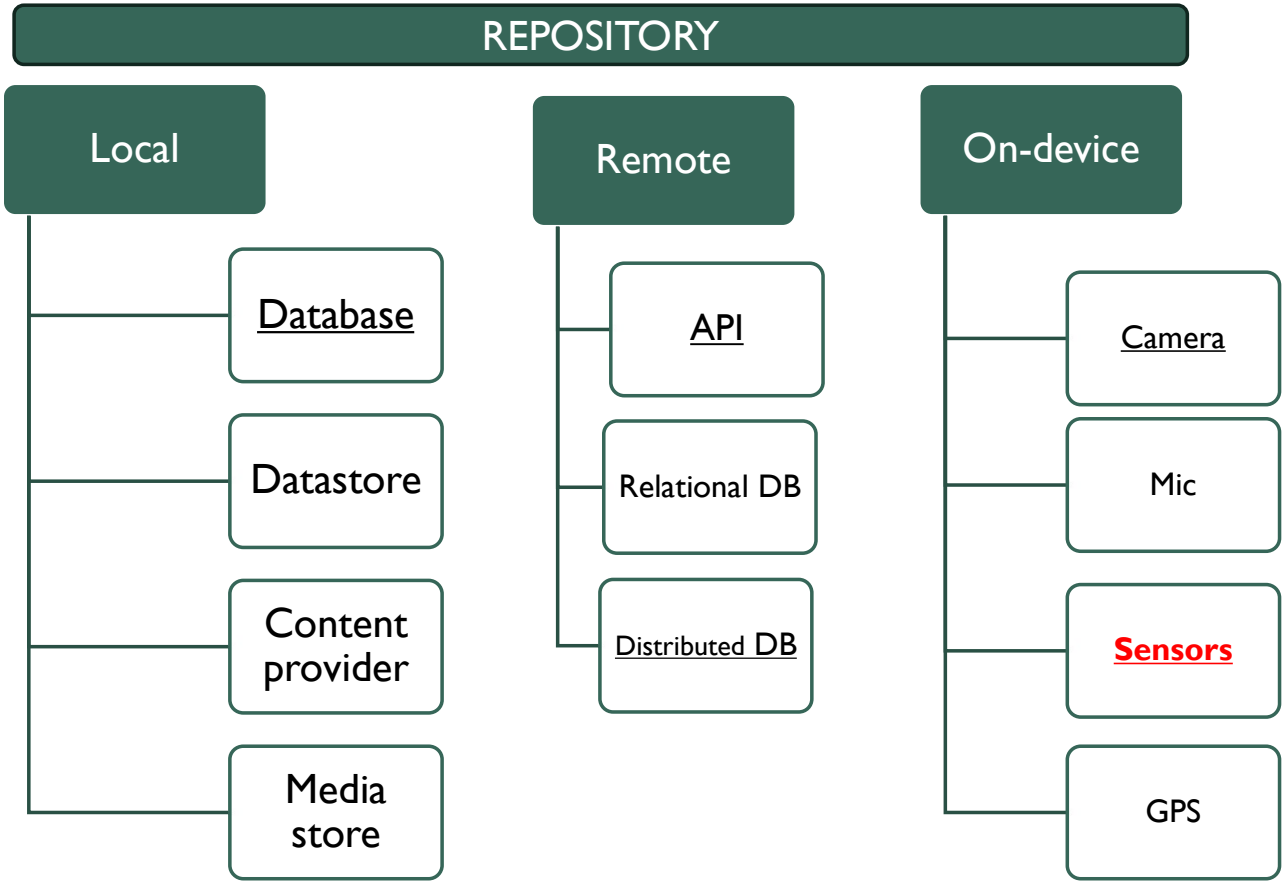
REMOTE DATA SOURCE (FIRESTORE)

```
fun sendBoard(gameId: String, newBoard: List<List<String>>) {
    db.collection("games")
        .document(gameId)
        .update("board", newBoard)
}
```

```
fun observeBoard(
    gameId: String,
    viewModel: TTTViewModel
) {
    db.collection("games")
        .document(gameId)
        .addSnapshotListener { snapshot, error ->

            // extract board (may be null)
            val newBoard = snapshot?.get("board") as? List<List<String>>

            // update ViewModel anyway (ViewModel decides what to do)
            viewModel.updateBoard(newBoard ?: listOf(listOf(), listOf(), listOf()))
        }
}
```

AN EXAMPLE OF APP WITH CLOUD SUPPORT

TAXONOMY OF DATA SOURCES



Besides classical data sources such as databases, smartphones also expose many other peculiar sources, coming from their rich set of sensors, the camera, and the GPS.

SENSORS AS DATA SOUCES

- Smartphones include many physical sensors that generate data used as inputs by applications, either in raw form or after some processing — for example, in a compass app.
- In general, sensors provide awareness of the surrounding environment, such as the device's orientation, the proximity of nearby objects, the amount of ambient light, and more.

SENSOR DATA CARACTERIZATION

- Sensors output pure numeric measurements:
- No semantics, no interpretation. Only physical measurements.

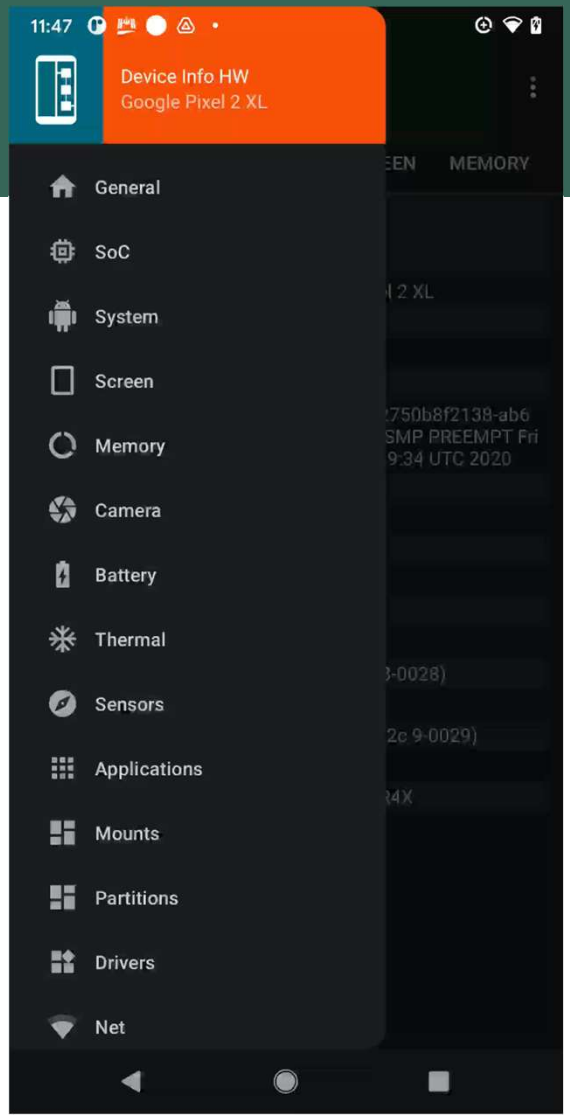
- Accelerometer → 3 floats (x, y, z) in m/s^2
- Gyroscope → 3 floats (angular velocity) in rad/s
- Magnetometer → 3 floats (μT)
- Barometer → 1 float (hPa)
- Light sensor → 1 float (lux)
- GPS → latitude, longitude, altitude, timestamp

- Sensors push data continuously:
- Accelerometer: 50–400 Hz
- Gyroscope: 50–400 Hz
- GPS: ~1 Hz
- Light: 5–10 Hz

- random noise
- temperature noise
- gyroscope drift
- magnetometer distortion
- Often requires filtering: e.g. low-pass

In adroid, sensors accessed via the SensorManager (next lessons)

A QUICK TOUR ON SENSORS



SENSORS IN ANDROID

- **Hardware sensors** that correspond to physical chips, usually are Micro-Electro-Mechanical Systems (MEMS), composed of two parts:
  - Mechanical part that exploits some physical law
  - Electrical part: used to transduce the mechanical quantity into an electrical readable value (e.g., acceleration → volts)

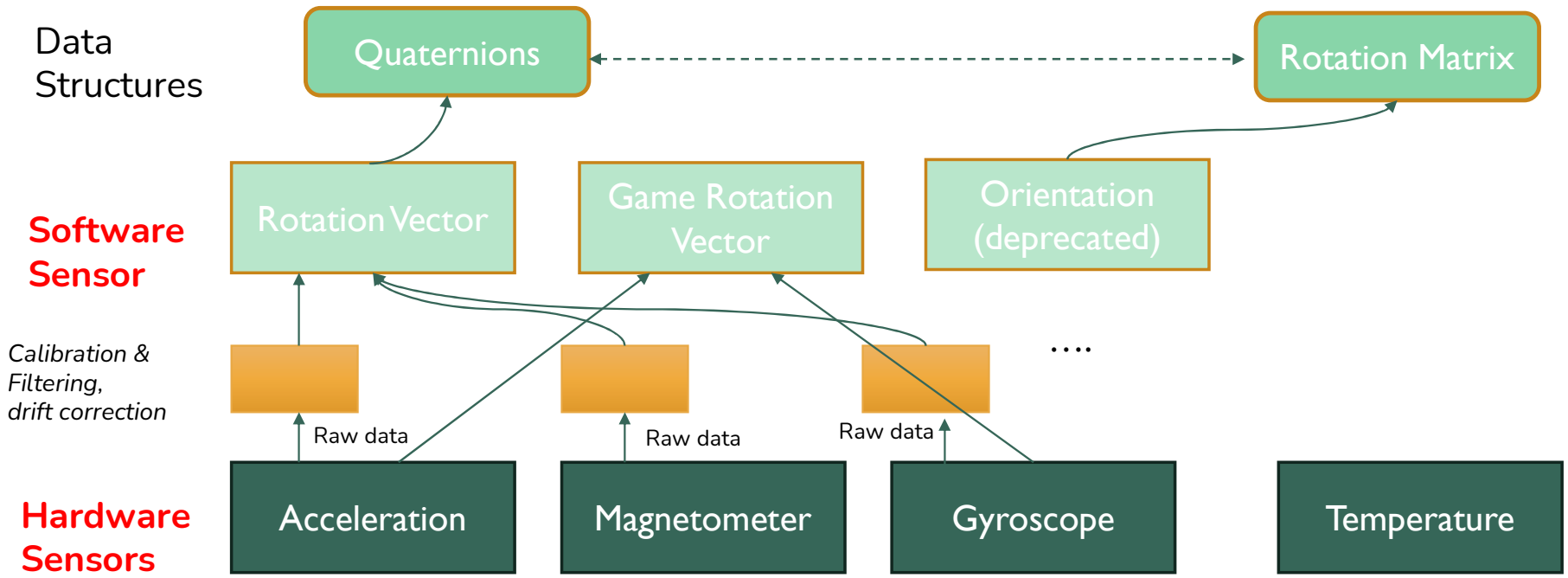
- **Software sensors** they merge (**sensor fusion**) many data from hardware sensors to provide a more abstracted valuable information (like the orientation, as described later)

SENSORS FOR ORIENTATION

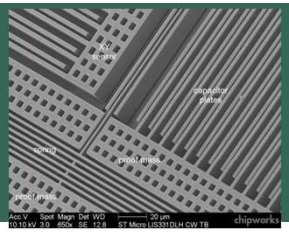
Calibration: determine α and β

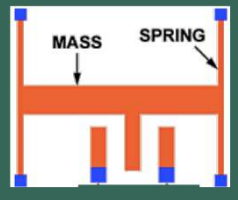
$y = F(x)$

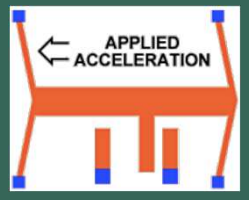
SENSOR

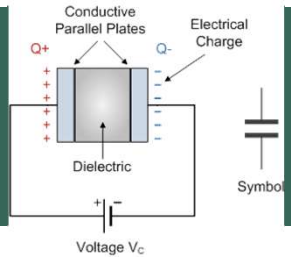


EXAMPLE





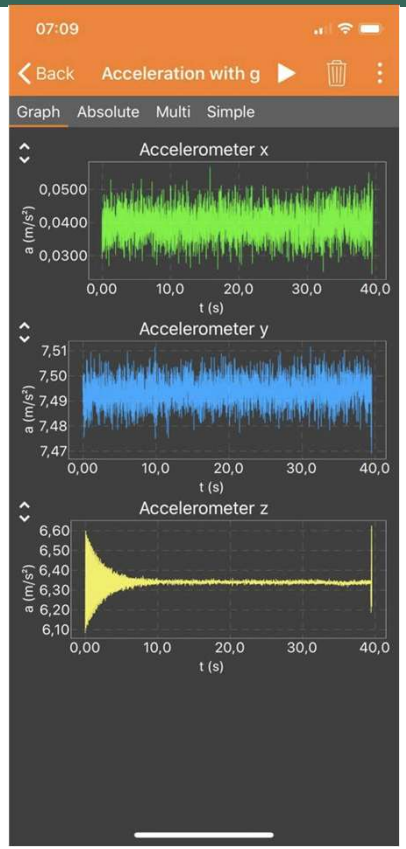




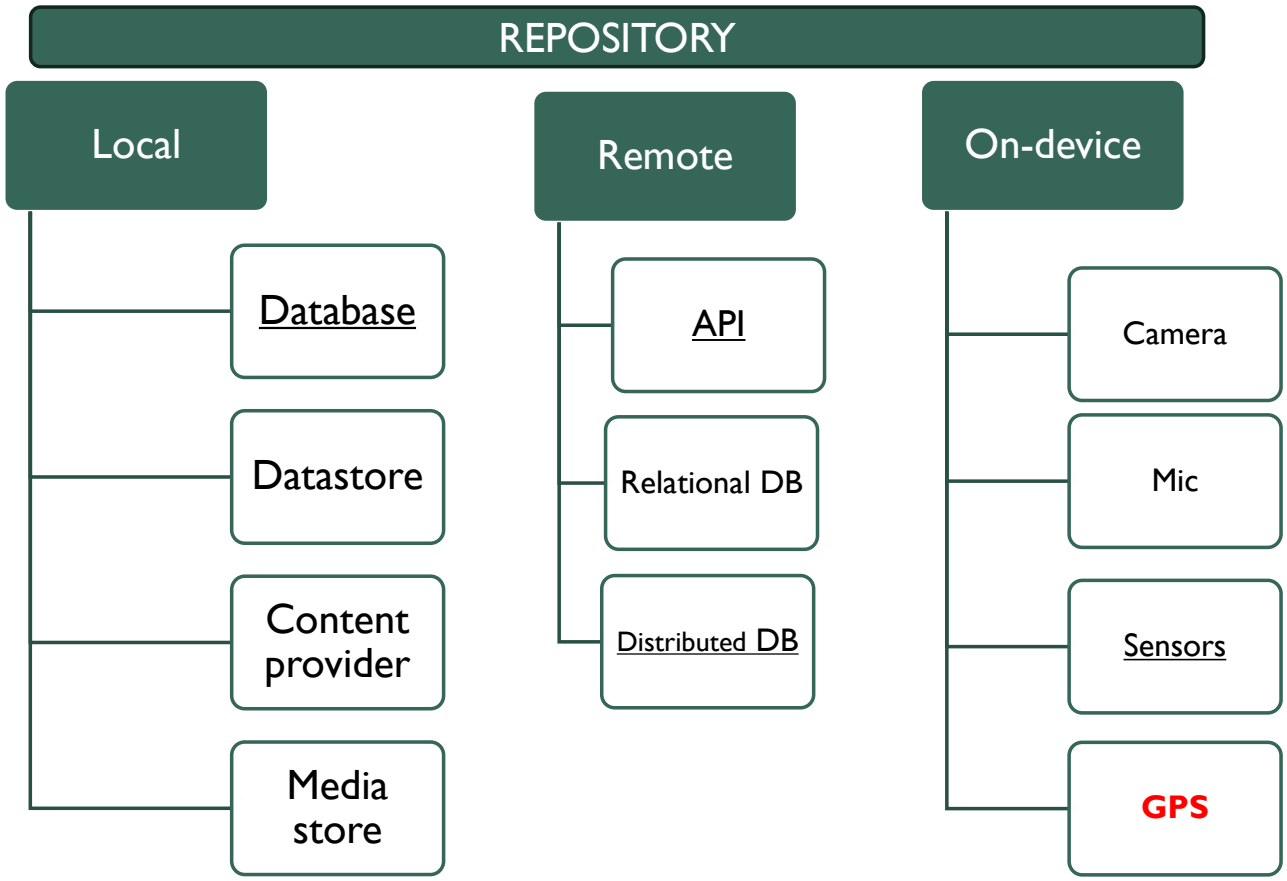
- LSM6DSM is a well-known 6-axis IMU (accelerometer + gyroscope).
- 156.9 m/s² ≈ 16 g
- Resolution 0.0024 m/s² ≈ 0.00024 g
  - This means the sensor can detect *very fine variations*.
  - It is extremely sensitive — it can detect accelerations about 4000 times smaller than gravity.
  - detect the vibration of your hand while you hold the phone perfectly still.
  - the micro-tremor of your muscles
  - the vibration coming from steps in the same room
  - the tiny shaking from someone typing
  - the slight tilt of the phone by less than 0.02°

EXAMPLE

- These data sources may also become a source of privacy leakage and security risks.
- The measurements are extremely precise, allowing very small vibrations caused by typing to be detected and analyzed
- For example, by analyzing the phone's accelerations, it is possible to infer which characters the user is typing on the screen.
- This is a trace of a smartphone when toughing the screen (phyphox)

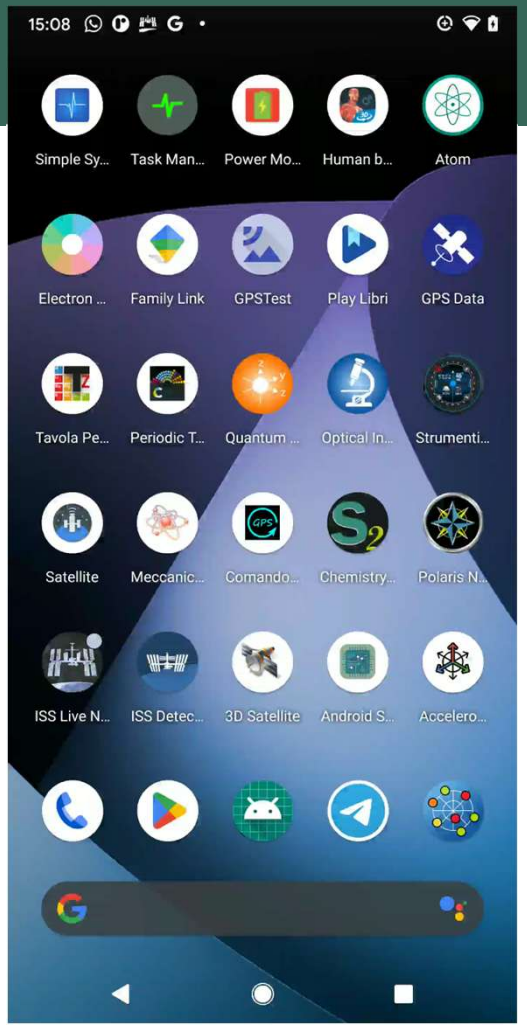


GPS


REPOSITORY
Local
Remote
On-device
Database
Datastore
Content provider
Media store
API
Relational DB
Distributed DB
Camera
Mic
Sensors
GPS

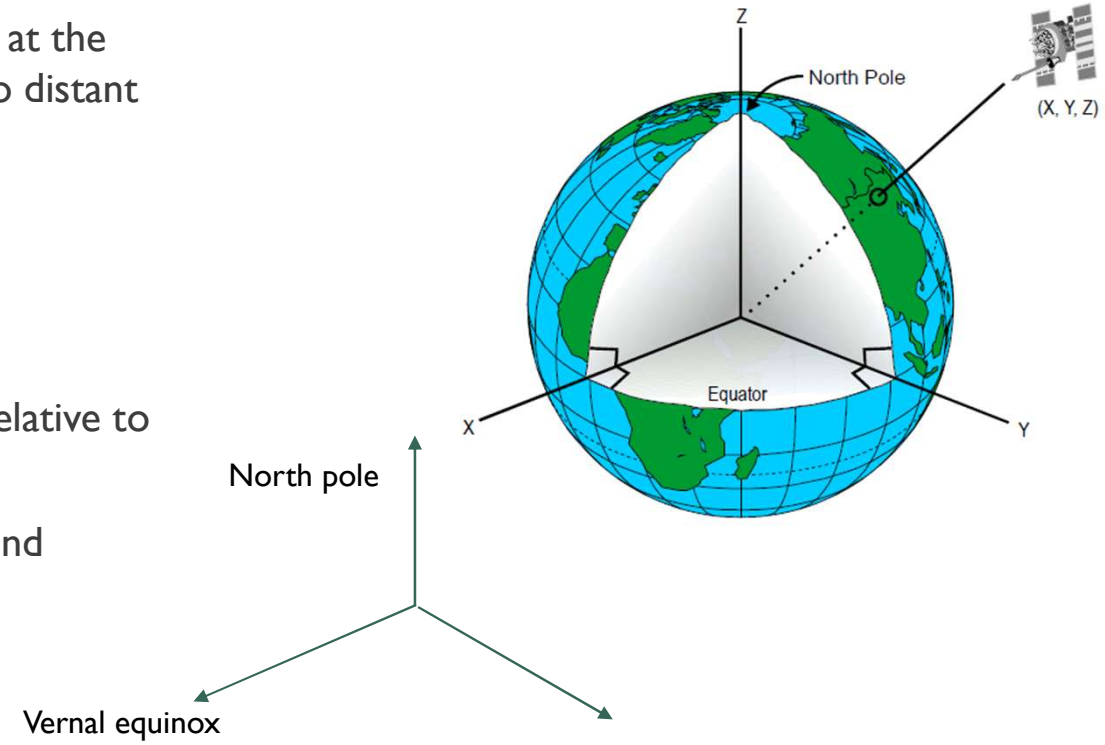
GPS

- The video shows the launch of a GPS test app, the satellite acquisition phase ("In View"), the subsequent docking ("In Use"), and the stabilization of the GNSS fix.
- Eventually the phone receives signals from 23 total satellites and uses about 7–10 satellites for a stable 3D fix.
- There are many satellite constellations used to estimate the position on the Earth
- GPS (United States)
- Galielo (Europe)
- GLONASS (Russia)
- BeiDou (Cina)



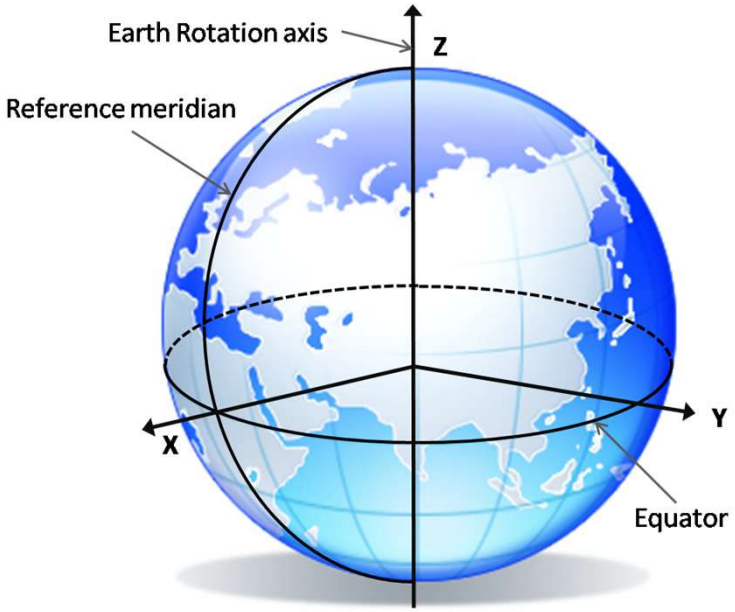
POSITION ESTIMATION: THE EARTH-CENTERED (ECI) FRAME

- **ECI** is a non-rotating, inertial reference frame centered at the Earth's center of mass, with axes X, Y, Z. Fixed relative to distant stars (does not rotate with the Earth).
  - The Z-axis pointing through the North Pole.
  - The X and Y axes lying in the equatorial plane.
  - Y pointing towards the vernal equinox.
- A GPS satellite represented with coordinates (X, Y, Z) relative to the Earth's center.
- GPS satellite orbits are defined and maintained (by ground stations) in the ECI frame.



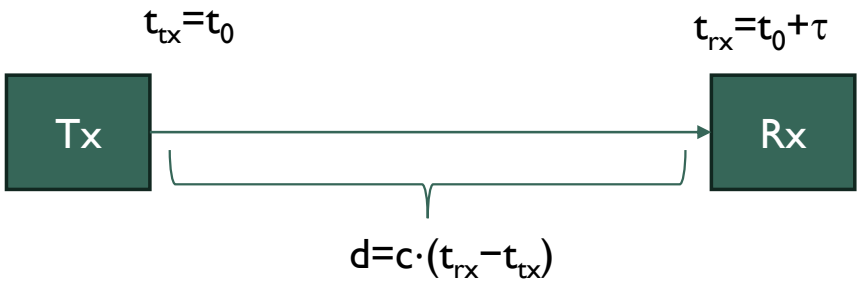
ECEF REFERENCE FRAME

- **ECEF** (Earth-Centered Earth-Fixed) is a geocentric Cartesian coordinate system that co-rotates with the Earth.
- The X–Y plane coincides with the equatorial plane, and the Z-axis, orthogonal to this plane, points toward the North Pole.
- The X-axis passes through a chosen reference meridian (Greenwich).
- A satellite broadcasts its orbital parameters (ephemeris) approx every 2 hours. The receiver computes the satellite's position 'now' from the ephemeris, and computes its distance from the satellite (pseudorange) using the difference between the sending time and the receiving time
- The position of the receiver is computed in the ECEF (WSG84) and converted in latitude-longitude



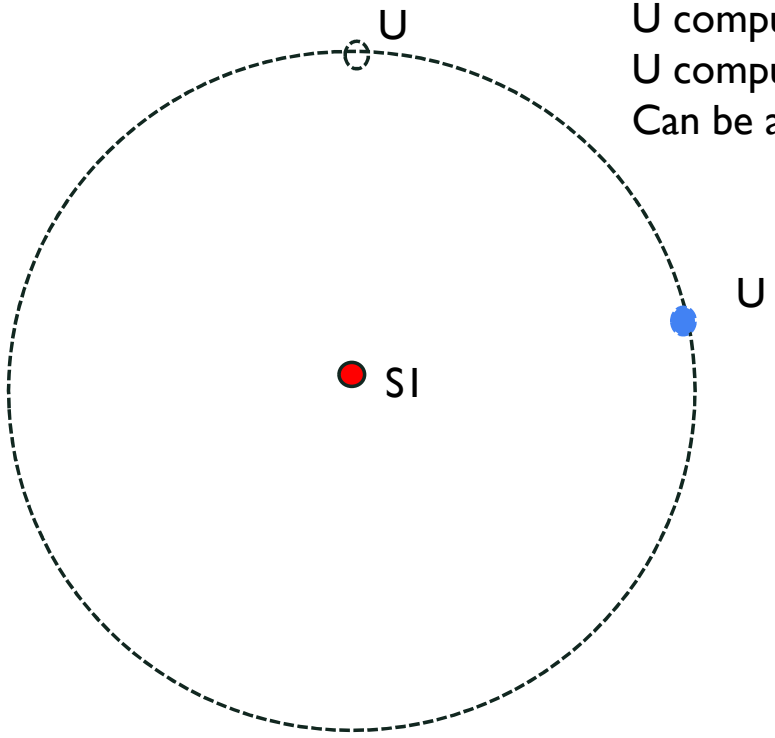
http://gnss.be/systems_tutorial.php

PSEUDORANGE AND LOCALIZATION



- If clocks at the sender and receiver sides are synchronized, then given the Time of Flight (τ), it is possible to compute the distance (range) among receiver and transmitter (the signal propagates at light speed c)
- Satellites use atomic clocks that are accurate
- Smartphones use a chipper clock hw that drifts compared to the true clock
- The drift is estimated

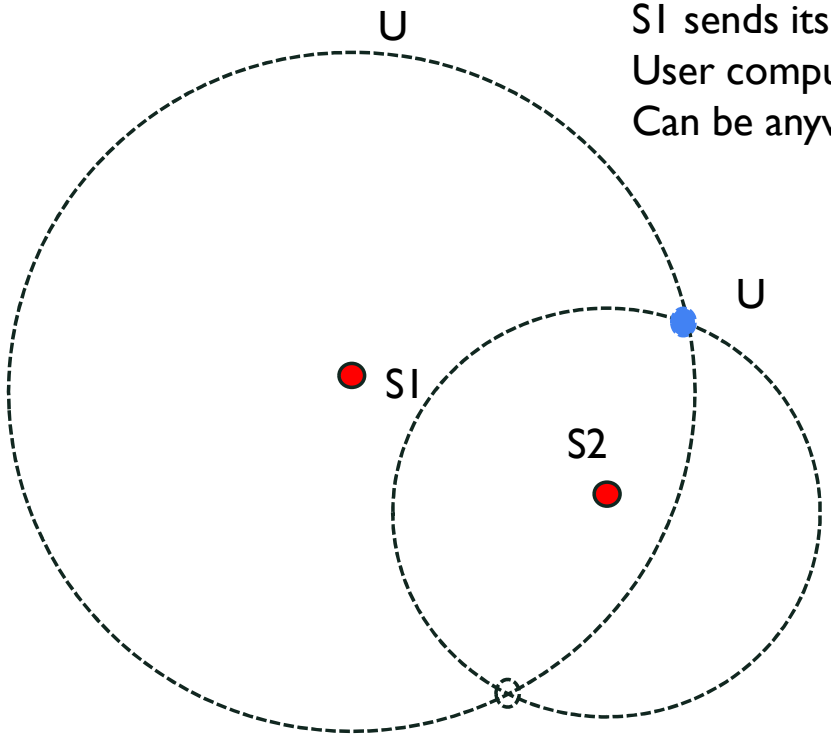
HOW TO COMPUTE THE POSITION



U computes the position of S1 (x1,y1)
U computes its distance d from S1
Can be anywhere in a circle C1

$d=c\cdot(t_{rx}-t_{tx})$

HOW TO COMPUTE THE POSITION

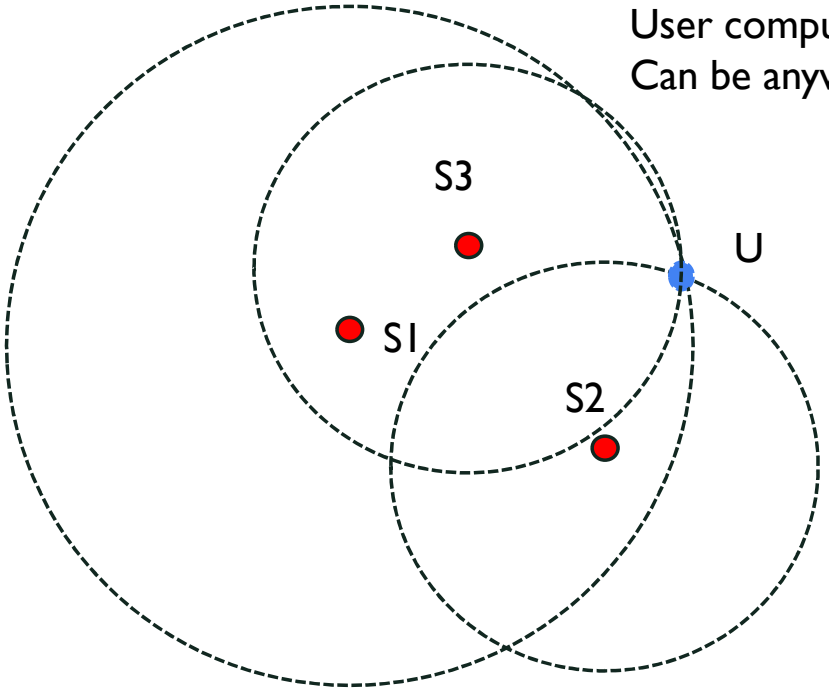


S1 sends its position (x1,y1)
User computes its distance from S1
Can be anywhere in a circle C1

U computes (x2,y2)
And the distance from S2
Can be anywhere in a circle C2

C1 and C2 intersects at two points
One additional reference is needed

HOW TO COMPUTE THE POSITION



S1 sends its position (x1,y1)
User computes its distance from S1
Can be anywhere in a circle C1

S2 sends its position (x2,y2)
User computes its distance from S2
Can be anywhere in a circle C2

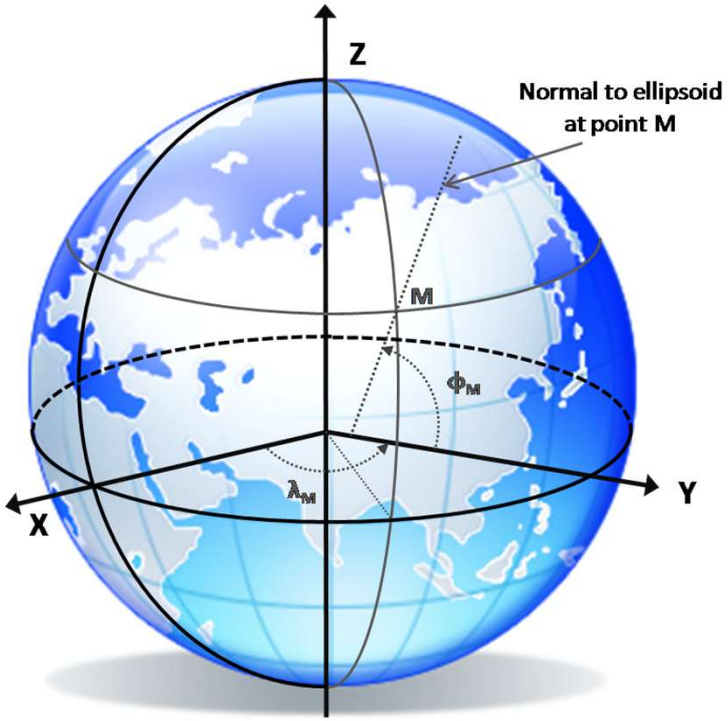
C1 and C2 intersects at two points
One additional reference is needed

If times are perfectly synchronized 3 distances are enough
If not, an additional measure is required.

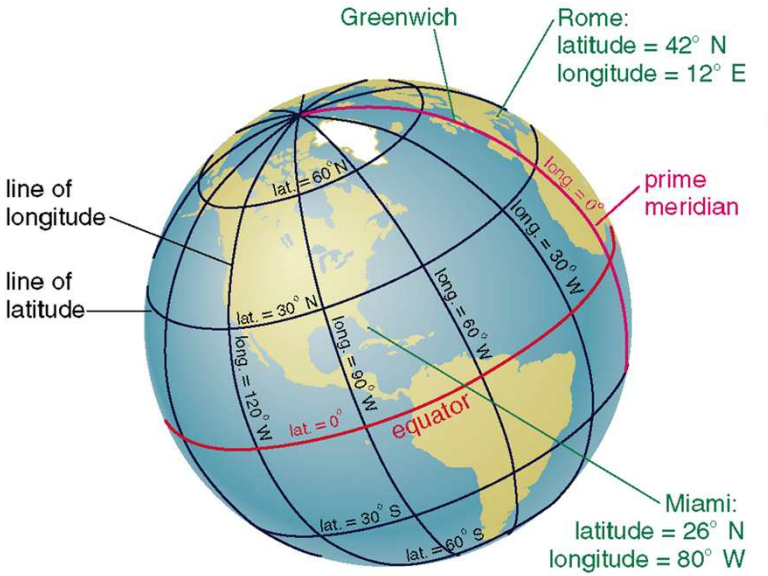
Today many other sources improve the precision, like wi-fi assisted, cellular networks and other on-device sensors (Location Fusion)

GEOGRAPHICAL LATITUDE AND LONGITUDE

- The position M is defined by only two spherical coordinates (instead of x,y,z)
- The geographical **latitude** of a point on Earth's surface (Φ_M) is the angle between the equatorial plane and the straight line that passes through that point and through (or close to) the centre of the Earth.
- The **longitude** (λ_M) of a point on Earth's surface is the angle east or west of reference prime meridian (Greenwich) to another meridian that passes through that point.

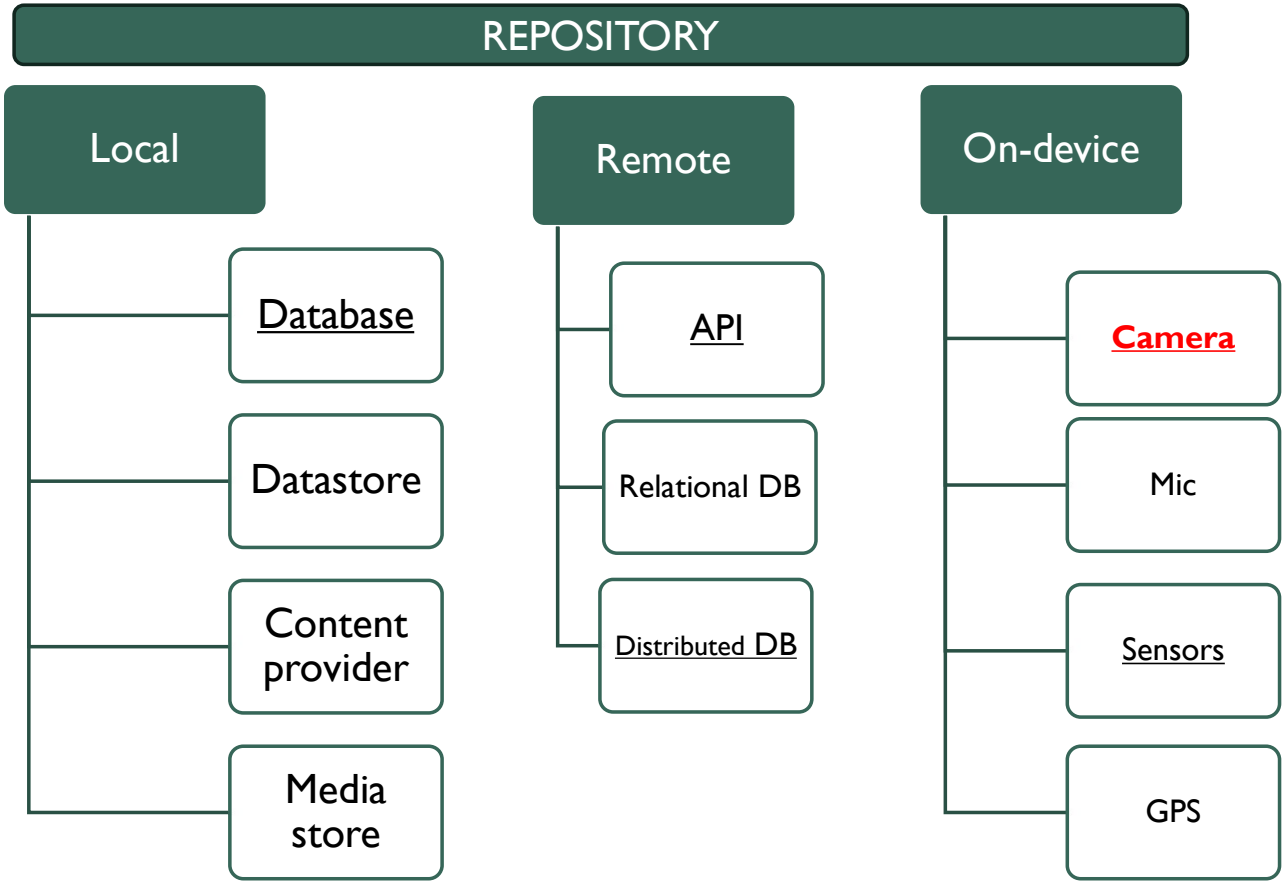


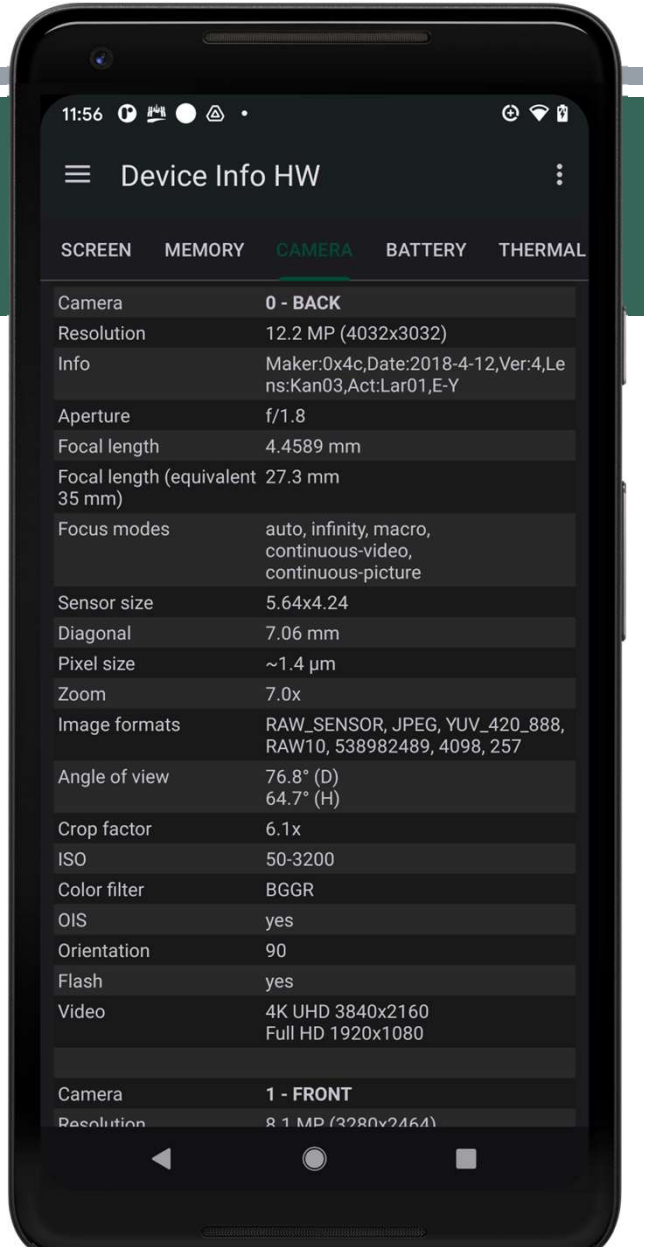
GEOGRAPHICAL LATITUDE AND LONGITUDE



- The accuracy of the position is in the range of 3-10 m (horizontally, lat-lon) and 25 m vertically
- The number of satellites of the fix is also important
- The accuracy increases over time (better clock drift estimation, better ionospheric model)

CAMERA AS DATA SOURCE





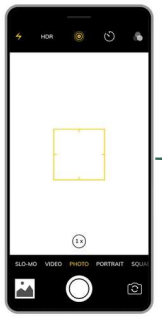
CAMERA AS DATA SOURCE

- The camera is a powerful data source, providing rich visual information that can be processed for tasks such as object detection, face recognition, barcode scanning, or scene understanding.
- Images can be acquired directly from the camera through the CameraX library, which provides lifecycle-aware APIs, dedicated use-cases (Preview, ImageCapture, ImageAnalysis), and background threads for real-time processing.
- A simpler alternative is to obtain an image from the device gallery using Intents, which allows the user to pick an existing photo without interacting with the camera pipeline.

SIMPLE FLOW WITH INTENTS

```
<uses-permission
android:name="android.permission.CAMERA"/>
```

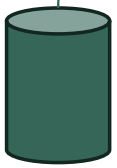
```
val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
startActivityForResult(intent, REQUEST_IMAGE_CAPTURE)
```



Camera

Intent

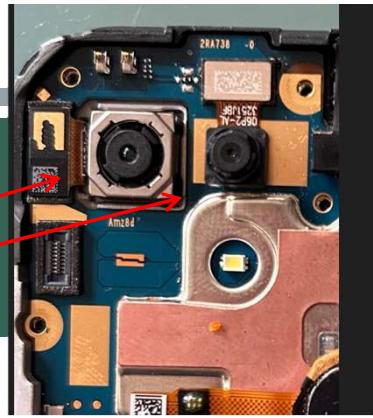
Gallery



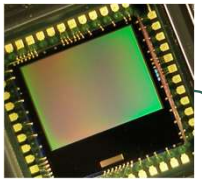
Bitmap → Mat → openCV

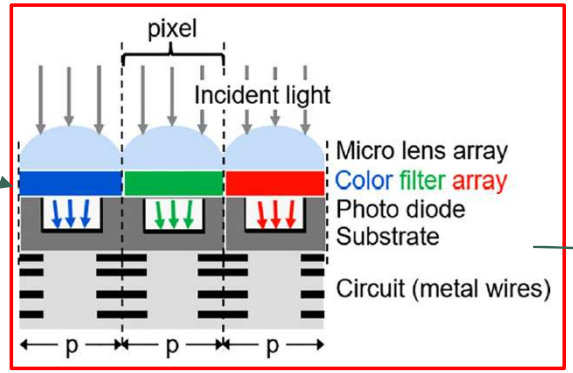
openCV → Mat → Bitmap → ImagaView

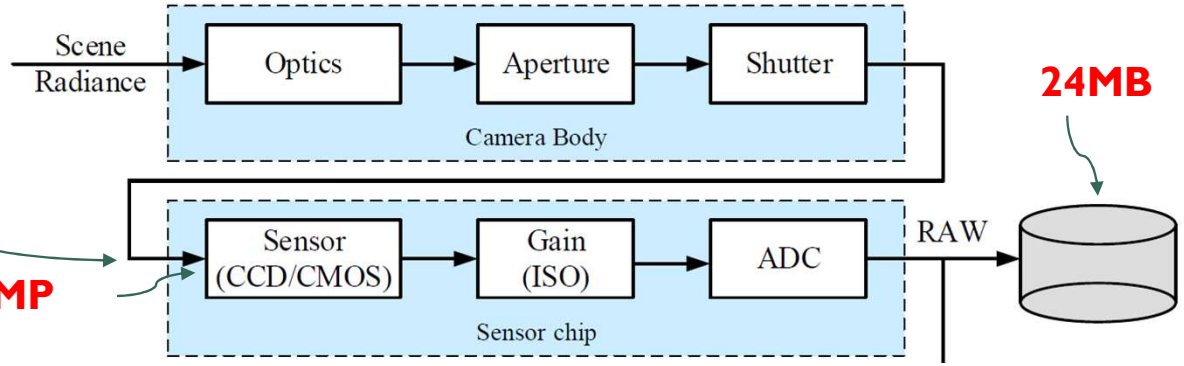
MORE ON CAMERA



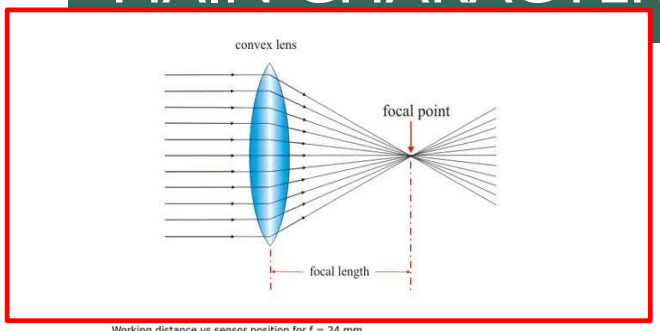
- An image is generated via a pipeline that includes the camera body and a sensor chip
- The camera body is composed by a system of lens, an aperture (fixed, e.g. f/2.4) and shutter (variable exposition time)
- The sensor chip is an array of pixels (picture elements) composed by a photodiode of some μm^2, a color filter and micro lens conveying lights towards the photodiode
- The sensor cell is a 'monochromatic' photodiode, it just measures the light intensity (luminance), i.e. the number of photons absorbed during an amount of time which hit the surface of the sensor
- The measure is amplified and discretized (ADC), using pixel-depth of 12-14 bits (aligned to 2 bytes): For example, 12MP sensor generates 12MPx2B=24MB

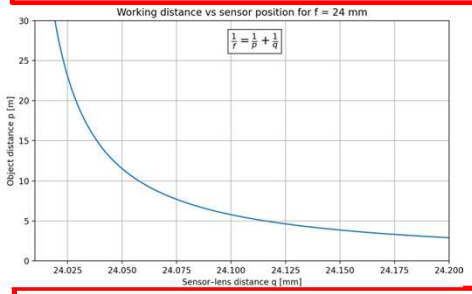


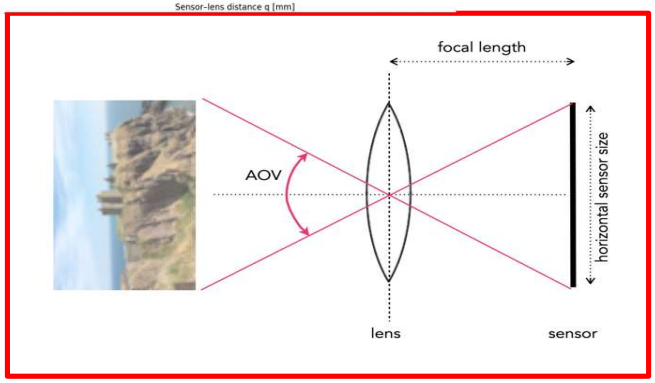




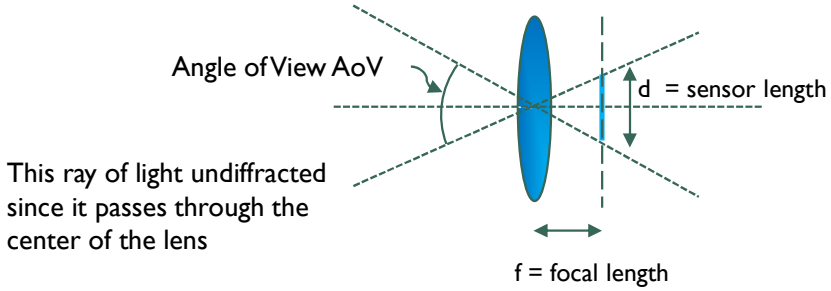
MAIN CHARACTERISTICS OF A CAMERA







- The system of lens in the camera body acts like a single convex lens with a focal length f.
- This is the distance from the lens to the point where rays coming from infinity are focused (focal point).
  - Working distance is the distance from the lens to the real object when the object is in focus.
- The focal length determines the Angle of View (AOV or AoV) which is the angle under which the ray of lights hit the sensor
- Assuming a sensor length d positioned at distance f, the AOV can be computed from a simple math:

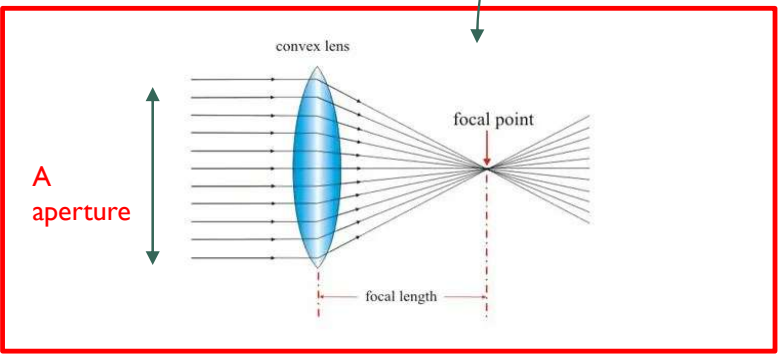


$d/2 = \sin(AOV/2)$
$f=\cos(AOV/2)$

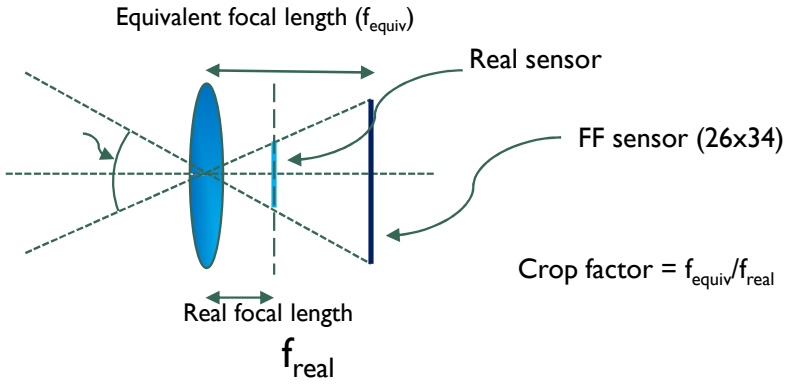
$\tan(AOV/2)=d/2f$

$AOV = 2 \cdot \arctan(d/2f)$

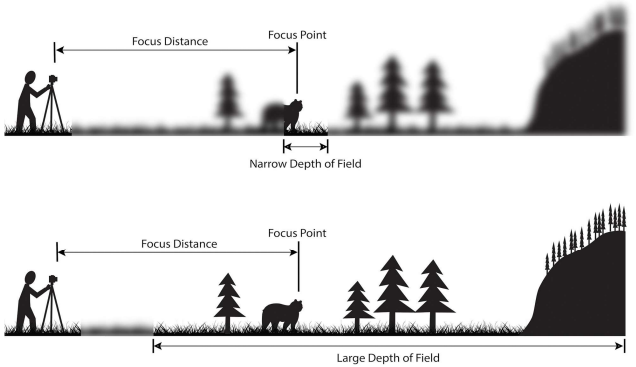
MAIN CHARACTERISTICS OF A CAMERA



- The aperture represents the diameter of the diaphragm A compared to the focal length, expressed as f numbers. For example, if the f number is "f/1.6", then A=f/1.6.
  - The specifications of a camera provide the 'equivalent' focal length, which is the focal length that a lens in front of a Full Frame (FF) sensor (35 mm) must have in order to have the same Angle of View of the real lens.
- Crop factor: The **crop factor** is the ratio between the diagonal of a reference *full-frame* sensor (36 × 24 mm) and the diagonal of another image sensor. It quantifies how much smaller the sensor is relative to full frame. Approximated by the ratio of the focal lengths. Use the real focal length to compute A



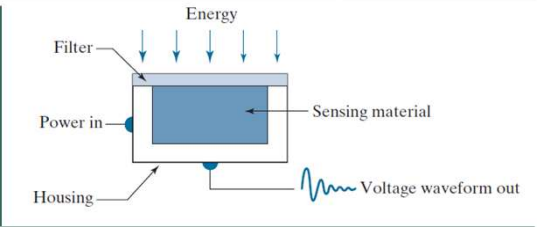
NUMBER OF CAMERAS



- The focal length determines the AOV (Angle of View) and, together with the aperture, it determines the DOF (Depth of Field), which is the portion of space that appears in focus ($DOF \propto N/F^2$)
- Since smartphone lenses cannot be changed (unlike in classical cameras), smartphones are equipped with more than one camera.
- For example — iPhone 14 Pro:
  - Main camera: 48 MP, f/1.78, 24 mm equivalent focal length
  - Ultra-wide: 12 MP, f/2.2, AOV ≈ 120°
  - Front camera: 12 MP, f/1.9
  - TrueDepth camera: 12 MP, f/1.9, 23 mm equivalent, with SL 3D sensor for depth and biometric recognition

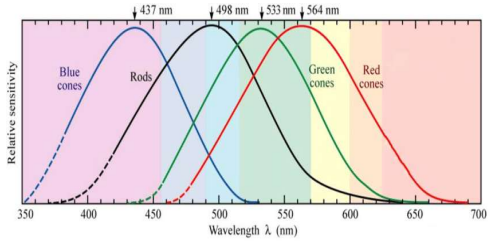
- **Face ID** is not performed using the standard camera. It uses IR light to reconstruct a 3D mesh of the face, and the matching code runs inside a separate trusted/secure zone.

COLOR PERCEPTION

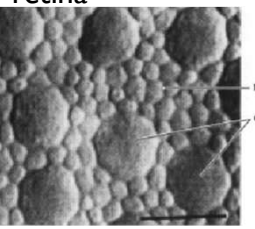


- Color perception results from the combined response of three types of cones: S-cones (blue), M-cones (green,) L-cones (red). The perceived color depends on the relative stimulation of S, M, and L. Rods operate in low-light (scotopic) vision and do not detect color.

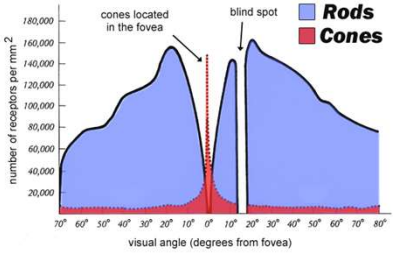
response



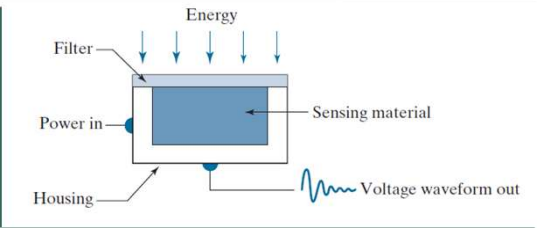
retina



distribution

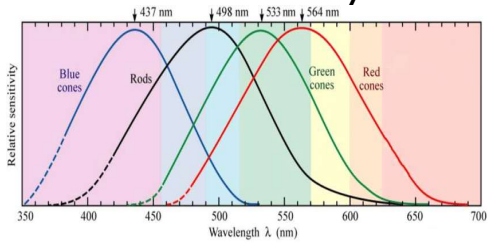


COLOR PERCEPTION

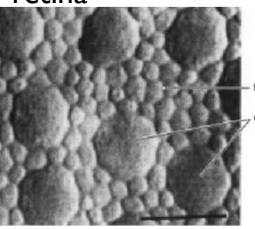


- Color perception results from the combined response of three types of cones: S-cones (blue), M-cones (green,) L-cones (red). The perceived color depends on the relative stimulation of S, M, and L. Rods operate in low-light (scotopic) vision and do not detect color.
- The human eye's sensitivity is higher around 500–600 nm (green-yellow region), which is due to the higher sensitivity of the L (long) and M (medium) cone cells in the retina.
- This also means that the perception of brightness (luminosity) is higher around the same wavelength range.
- For this reason, the color filter array placed in front of the sensor pixels follows a **Bayer RGGB pattern**, where green filters are twice as many as red or blue ones — reflecting the human eye's higher sensitivity to green light.

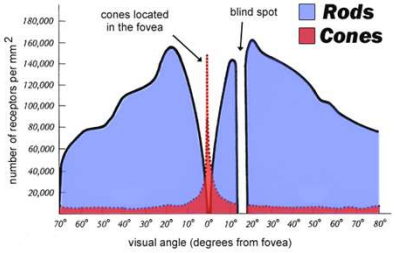
response

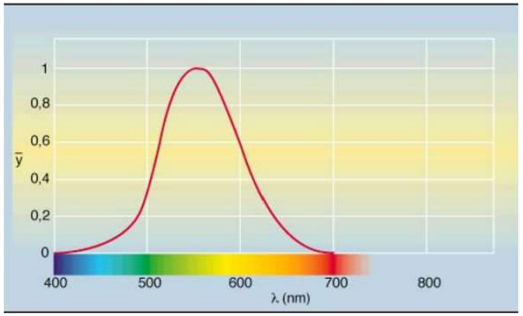


retina

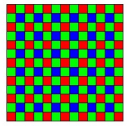


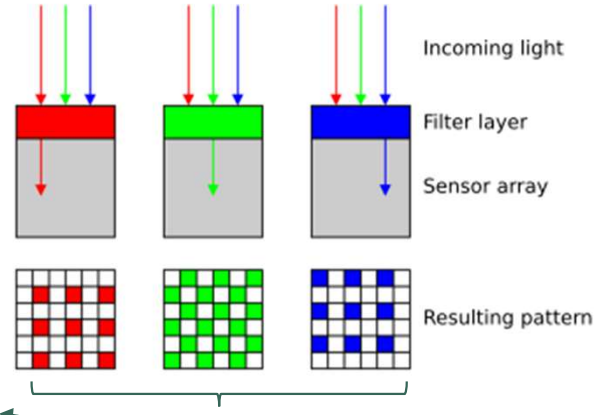
distribution





Normalized luminosity response

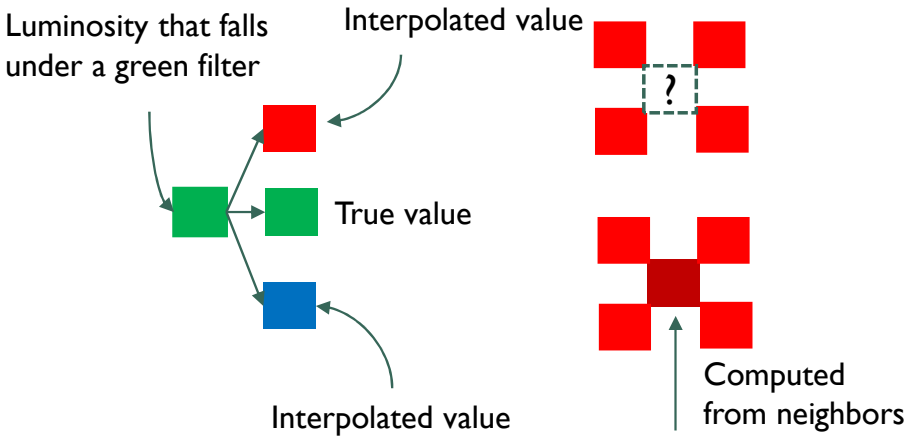


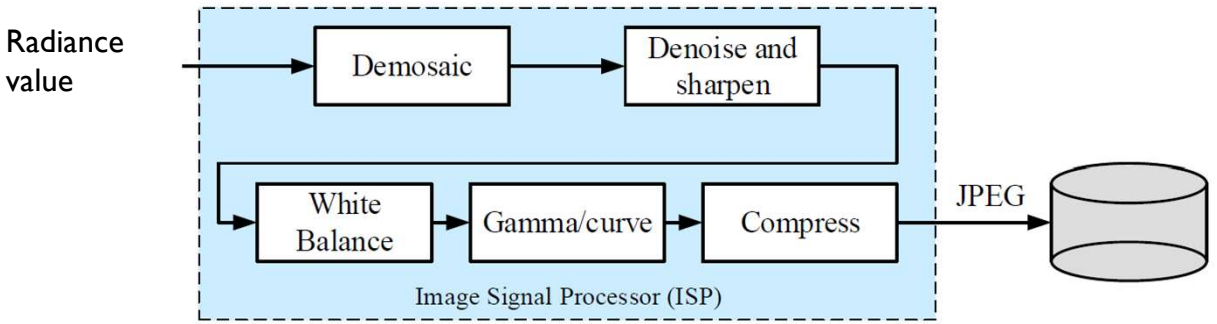


Filter with Bayer's pattern

CHARACTERIZING THE DATA

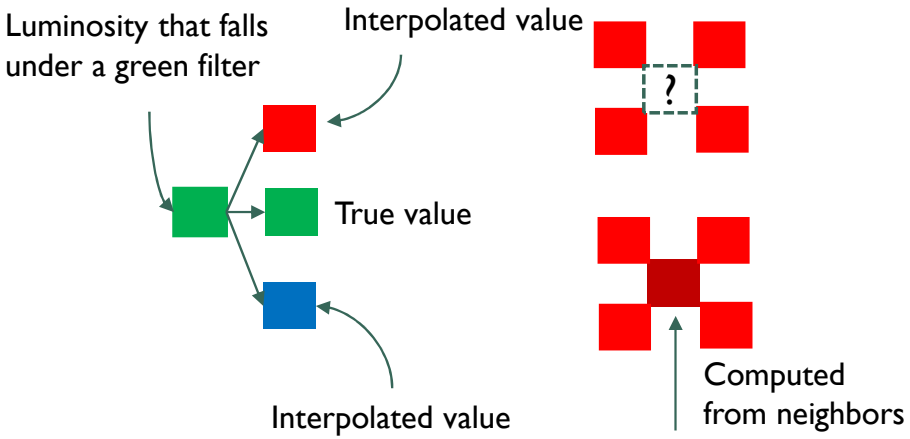
- The previous two stages in the pipeline converted the radiance (mW/cm^2) hitting the pixels into a matrix (sensor) of digital numbers (pixel).
- Due to the filter, each pixel in the sensor records the radiance for one color (R, G, or B).
- The missing colors are computed by interpolating neighboring pixels (Demosaic)

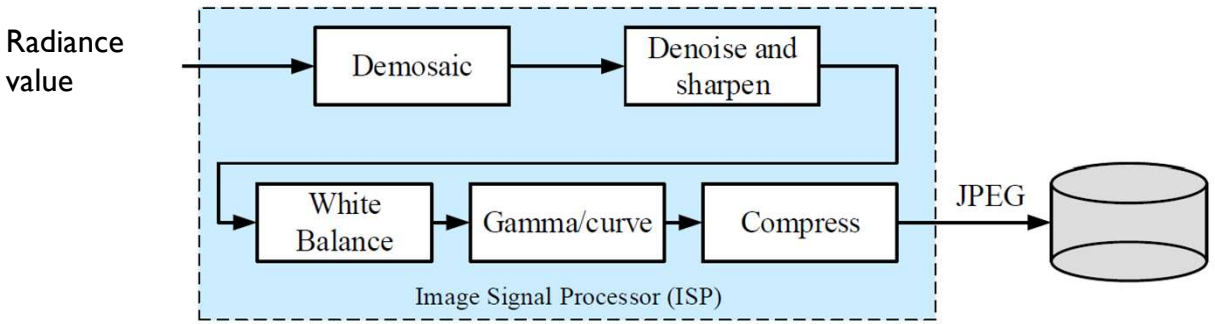




CHARACTERIZING THE DATA

- After this processing one single pixel has three values RGB, one true and the other interpolated
- Because the Bayer pattern allocates twice as many green pixels as red or blue, the green channel carries more spatial information and contributes the most to luminance detail (better spatial resolution).

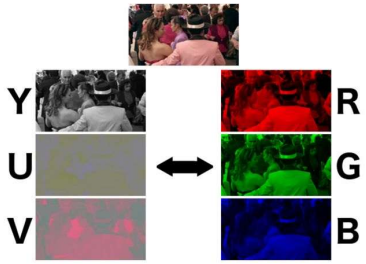


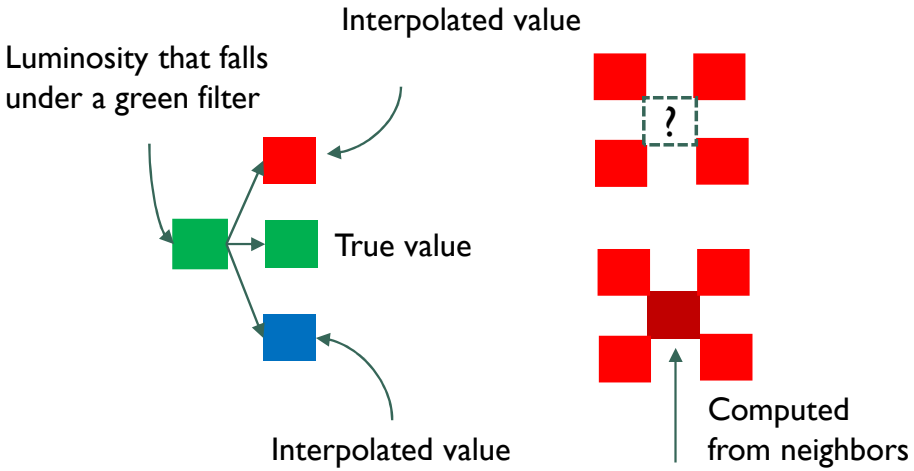


The images taken from a smartphone have several applications (see V. Blahnik and O. Schindelbeck, "Smartphone imaging technology and its Application" - Adv. Opt. Techn. 2021; 10(3): 145–232)

CHARACTERIZING THE DATA

- Other processing include white Balance: corrects color bias caused by different lighting conditions to make whites and grays appear neutral.
- Gamma correction: assigns higher weight to low (dark) values ($y=x^{1/\gamma}, \gamma=2.2$) because the human eye is more sensitive to differences in dark regions than in bright regions.
- Compression: transforms the representation of color from RGB to luminance + chrominance, such as YUV or YCbCr, which is more suitable for efficient encoding





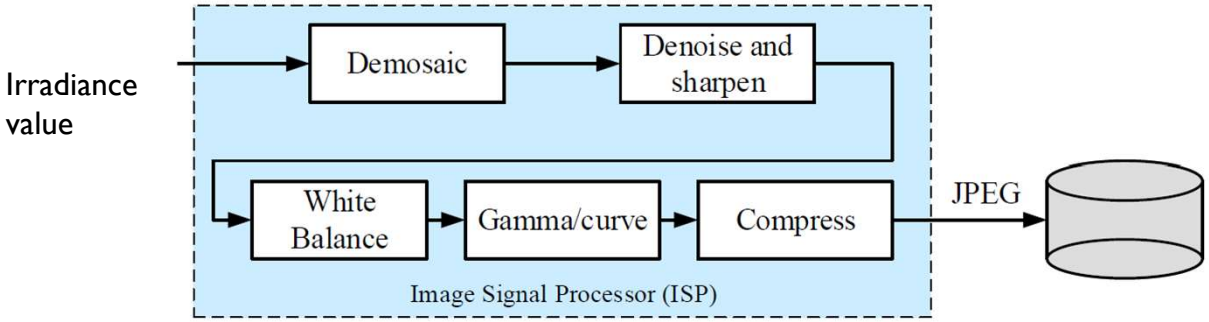
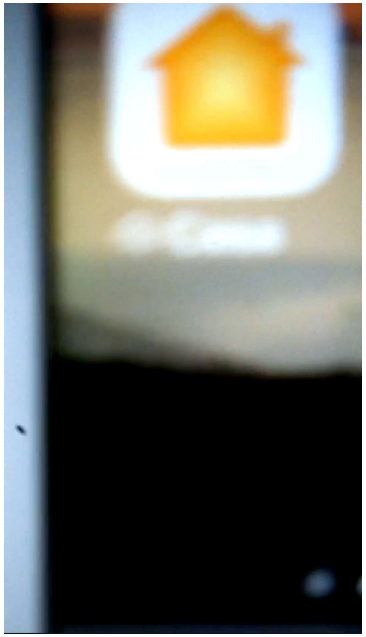


IMAGE REPRODUCTION

- An RGB image is rendered on the screen by illuminating 3 sub-pixels such that the perceived color matches the 'true' color (provides the same perception that the original color does)
- PPI (Pixels Per Inch) indicates the pixel density of a display, i.e., how many pixels are packed into one inch of screen space — higher values imply sharper perceived detail (in principle). (440 PPI = 440 pixels in 2.54 cm)



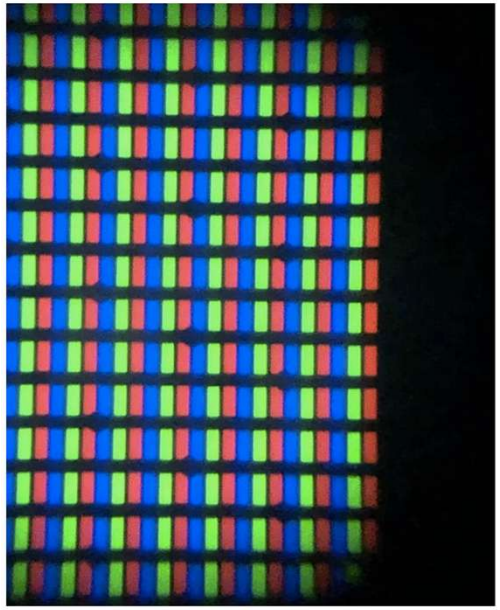
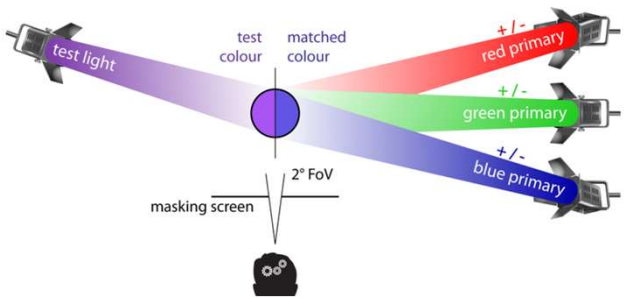
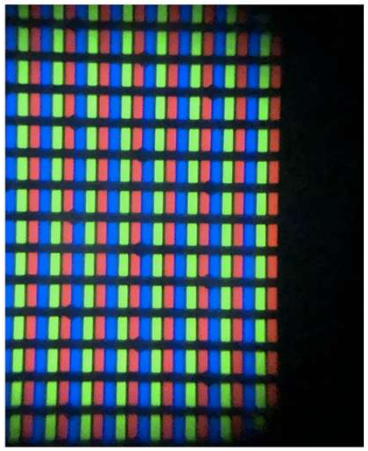
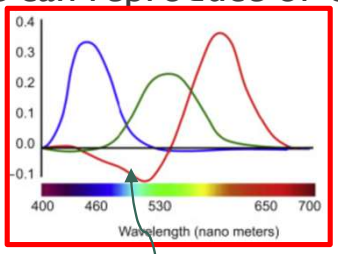


IMAGE REPRODUCTION

- The RGB image on the screen does not reproduce the exact spectral composition of the original light.
- Instead, it illuminates the R, G, B sub-pixels at the right intensities so that the combination appears visually like the "true" color.
- Digital displays are essentially doing metameric color matching.
- The gamut is the range of colors that a device or color space can reproduce or capture.

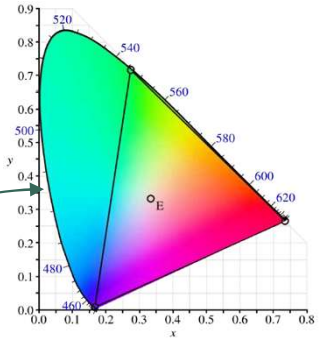






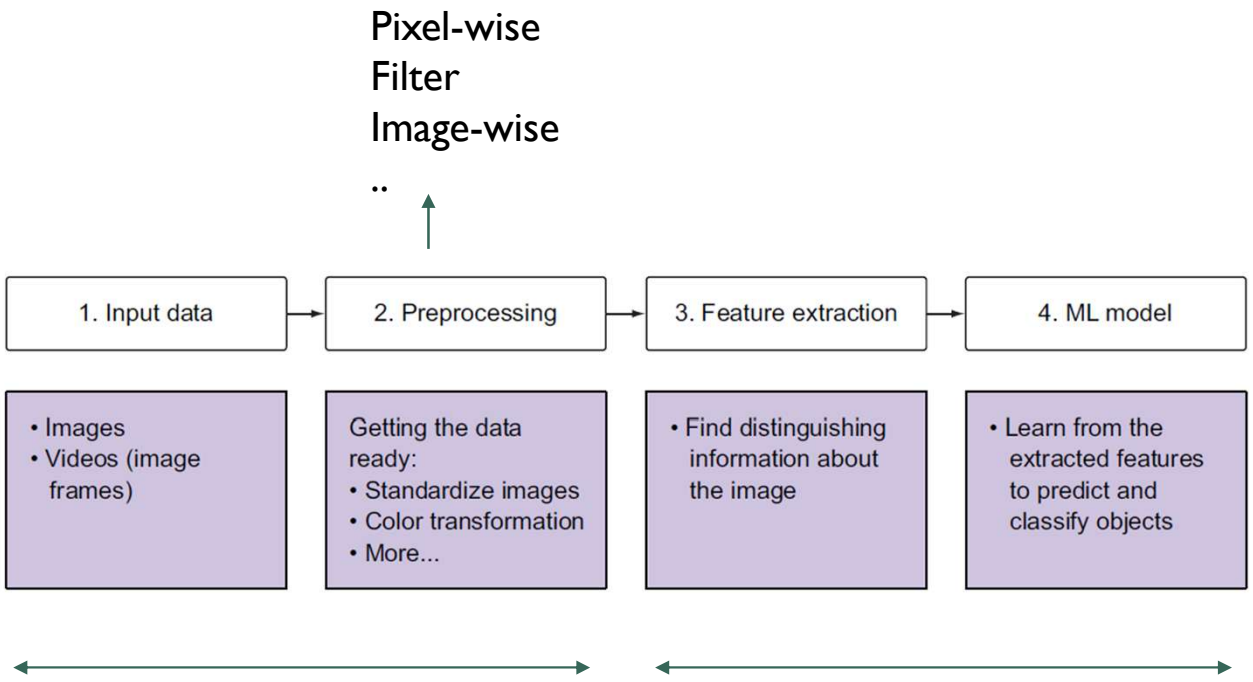
these colors cannot be reproduced with R+G+B

RGB encoding doesn't span the complete perceivable gamut



WHAT TO DO WITH AN IMAGE ?

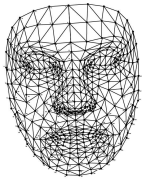
- core. **Core functionality**
- imgproc. **Image Processing**
- features2d. **2D Features Framework**
- objdetect. **Object Detection**
- dnn. **Deep Neural Network module**
- ml. **Machine Learning**
- …



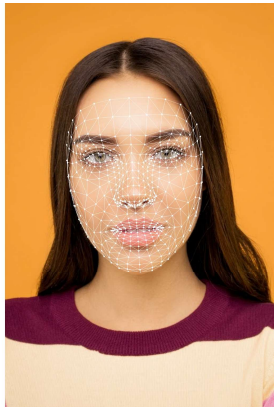
CameraX use cases
some simple techniques from openCV

Use predefined models from MLKit

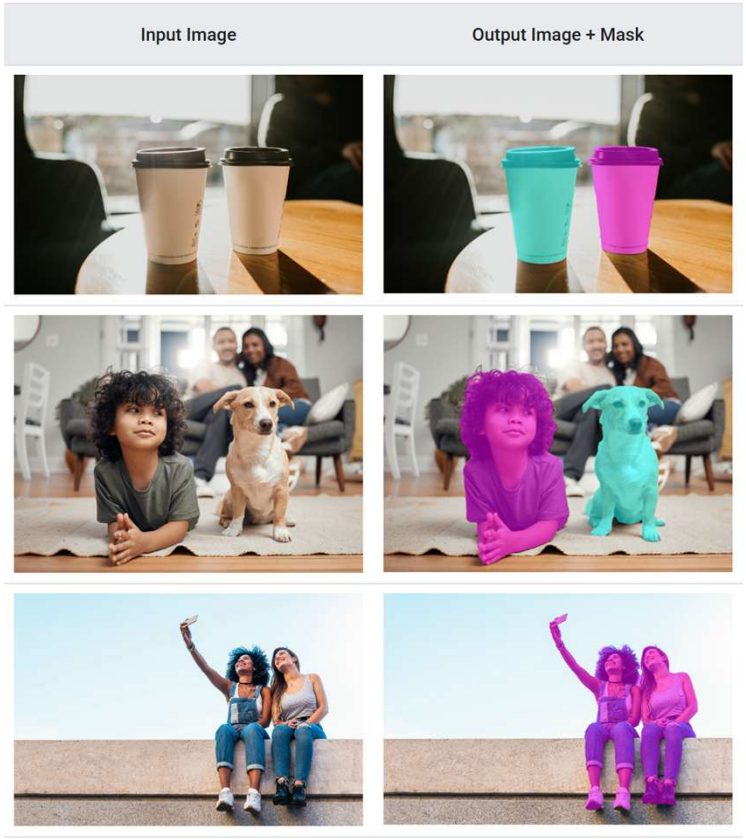
FACE MESH DETECTION



468 points



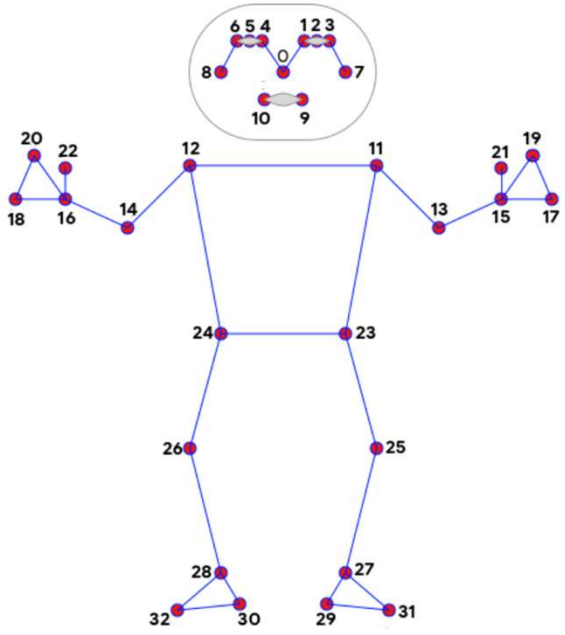
MLKIT (VISION API)



SUBJECT SEGMENTATION

MLKIT (VISION API)

- Pose estimation



Landmarks

0. Nose
1. Left_eye_inner
2. Left_eye
3. Left_eye_outer
4. Right_eye_inner
5. Right_eye
6. Right_eye_outer
7. Left_ear
8. Right_ear
9. Left_mouth
10. Right_mouth
11. Left_shoulder
12. Right_shoulder
13. Left_elbow
14. Right_elbow
15. Left_wrist
16. Right_wrist
17. Left_pinky
18. Right_pinky
19. Left_index
20. Right_index
21. Left_thumb
22. Right_thumb
23. Left_hip
24. Right_hip
25. Left_knee
26. Right_knee
27. Left_ankle
28. Right_ankle
29. Left_heel
30. Right_heel
31. Left_foot_index
32. Right_foot_index

WHAT ELSE TO DO WITH AN IMAGE ?

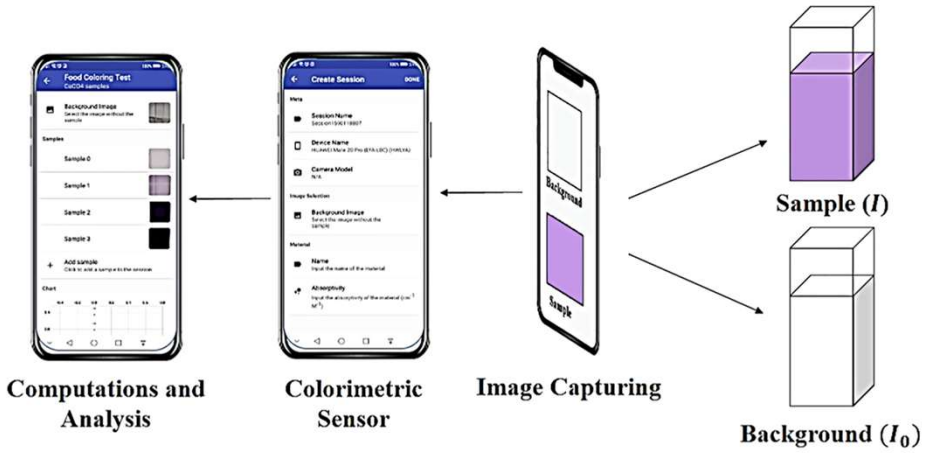


Fig. 2. Schematic illustration of the imaging mechanism showing both background and sample image

- Colorimetry (measures and quantifies color in an objective, numerical way) to estimate concentration of glucose, triglycerides, and urea.
- The concentration of a substance is determined by how much light of a specific wavelength is absorbed by the solution (Beer–Lambert law).

$$A = \log_{10}(I_0/I)$$

T. Alawsi, G. Proietti Mattia, Z. Al-Bawi, R. Beraldi, "Smartphone-based colorimetric sensor application for measuring biochemical material concentration", Sensing and Bio-Sensing Research, Volume 32,2021

EXAMPLE OF CAMERA2 CONTROL

- Very low-level: can configure almost everything manually.
- Focus (AF): AUTO, CONTINUOUS_PICTURE, CONTINUOUS_VIDEO, MANUAL.
- Exposure (AE): AUTO, locked, compensation values.
- White Balance (AWB): AUTO, daylight, cloudy, fluorescent, incandescent.
- Flash: AUTO, ON, OFF, TORCH.
- Color correction: matrices and transforms for advanced image tuning.
- control image post-processing: Noise reduction & edge enhancement
- JPEG orientation: rotate output based on device orientation.

CAMERA2 FLOW

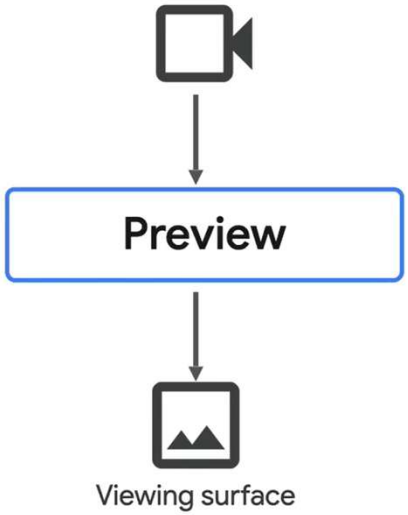
- CameraManager : enumerate cameras, get CameraCharacteristics.
- CameraDevice : open the camera hardware.
- CaptureRequest : define parameters (AF, AE, WB, ISO, flash, color transforms).
- CaptureReply : returns the parameters used.
- CameraCaptureSession : manage single or repeating requests (preview, capture, video).

```
val captureBuilder =
camera.createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE)
captureBuilder.addTarget(imageReader.surface)

// configure:
captureBuilder.set(CaptureRequest.JPEG_ORIENTATION, 90)
captureBuilder.set(CaptureRequest.CONTROL_AF_MODE,
    CaptureRequest.CONTROL_AF_MODE_AUTO)
session.capture(captureBuilder.build(), captureCallback, null)
```

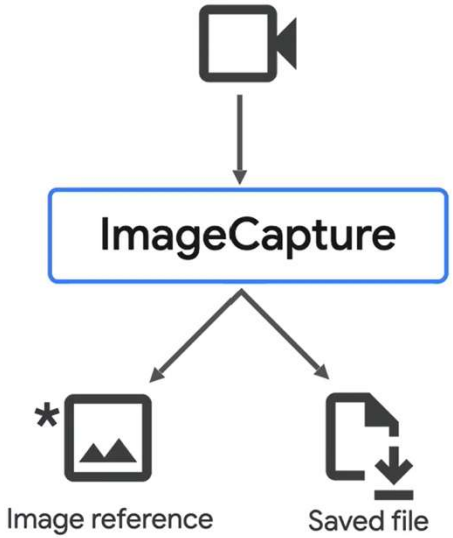
CAMERAX

- Jetpack library on top of Camera2
- Simplifies coding, thread and lifecycle management, allows preview, photo, and analysis in a simple pipeline
- Suitable for real-time frame processing and CV (OpenCV, MLKit,tensorflow)
  - Default color space YUV_420_888
  - Pixel-wise operations, filtering, object detection, classification, barcode scanning, etc..
- Image processing occurs on a separate thread (not on the UI main thread)
- Decide the backpressure strategy, i.e. what to do with frames produced at higher rate than processing capability (store in a finite queue or drop)
- Ready to use Use Case for
- Image Analysis
- Preview
- Video capture
- Image Analysis

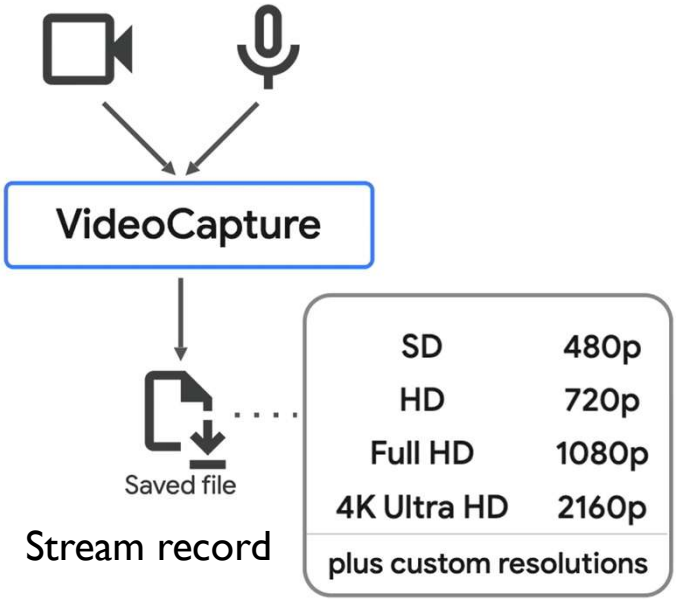
USE CASES



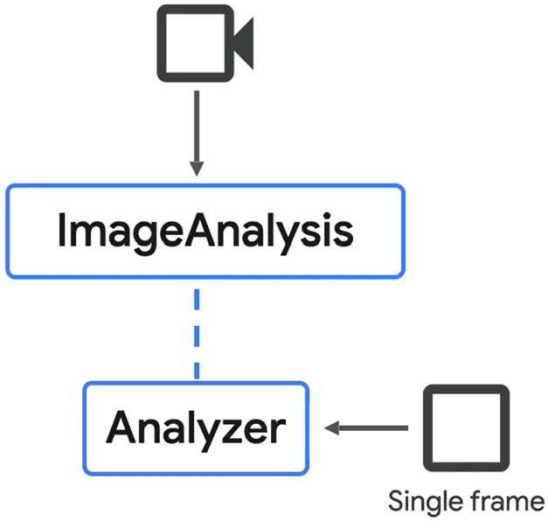
For example, in a video call to see how image looks line



High quality photos, Saved in a file or as a in memory reference



Stream record



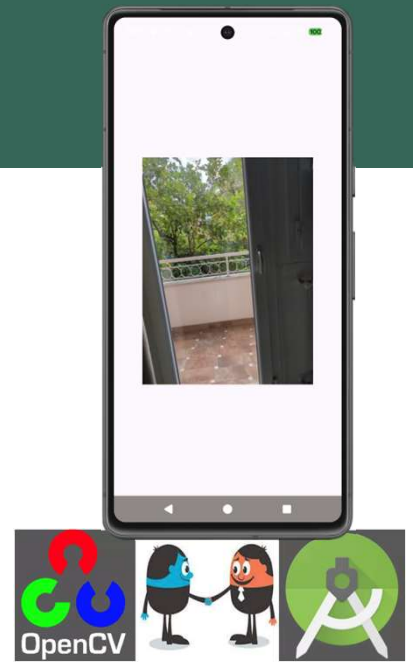
Per frame analysis with OpenCV,ML Kit..

EXAMPLE

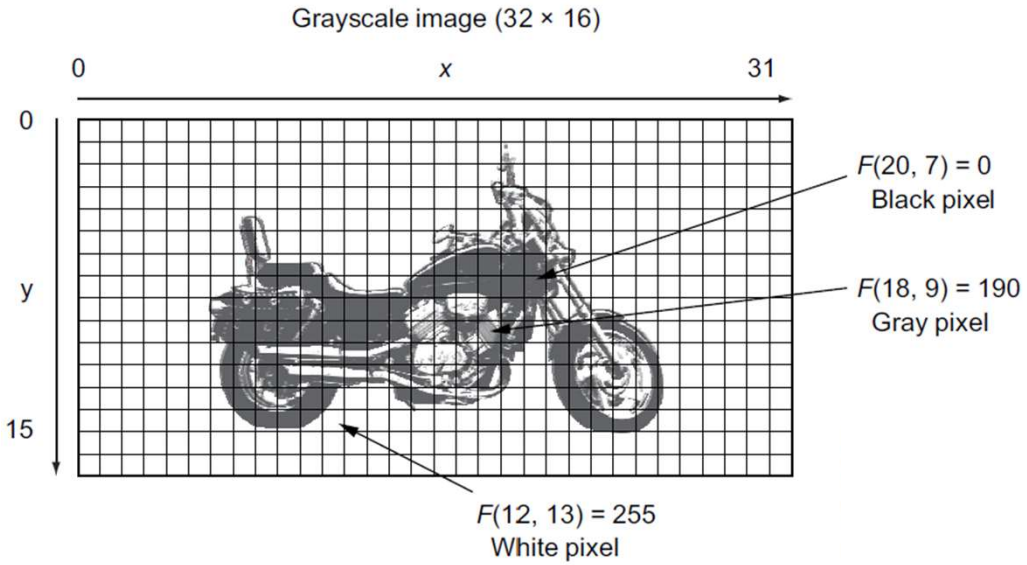
```
val imageAnalysis = ImageAnalysis.Builder()
    .setTargetResolution(Size(1280, 720))
    .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
    .build()

imageAnalysis.setAnalyzer(Executors.newSingleThreadExecutor()) { image ->
    // Analysis algorithm
    image.close() // unlock the image
}
```

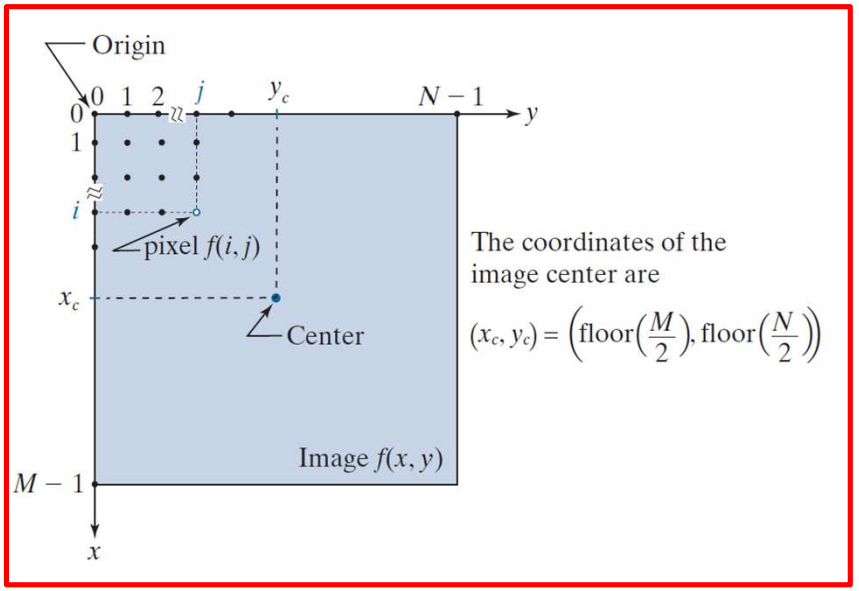
EXAMPLE

```
// analyzer
LaunchedEffect(Unit) {
    cameraController.setImageAnalysisAnalyzer(
        Executors.newSingleThreadExecutor()
    ) { imageProxy ->

        val mat = Mat()
        val mbi = imageProxy.toBitmap()

        bitmapToMat(imageProxy.toBitmap(),mat)
        Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGBA2GRAY)
        matToBitmap(mat,mbi)
        processedBitmap = mbi


        mat.release()
        imageProxy.close()
    }
}
```

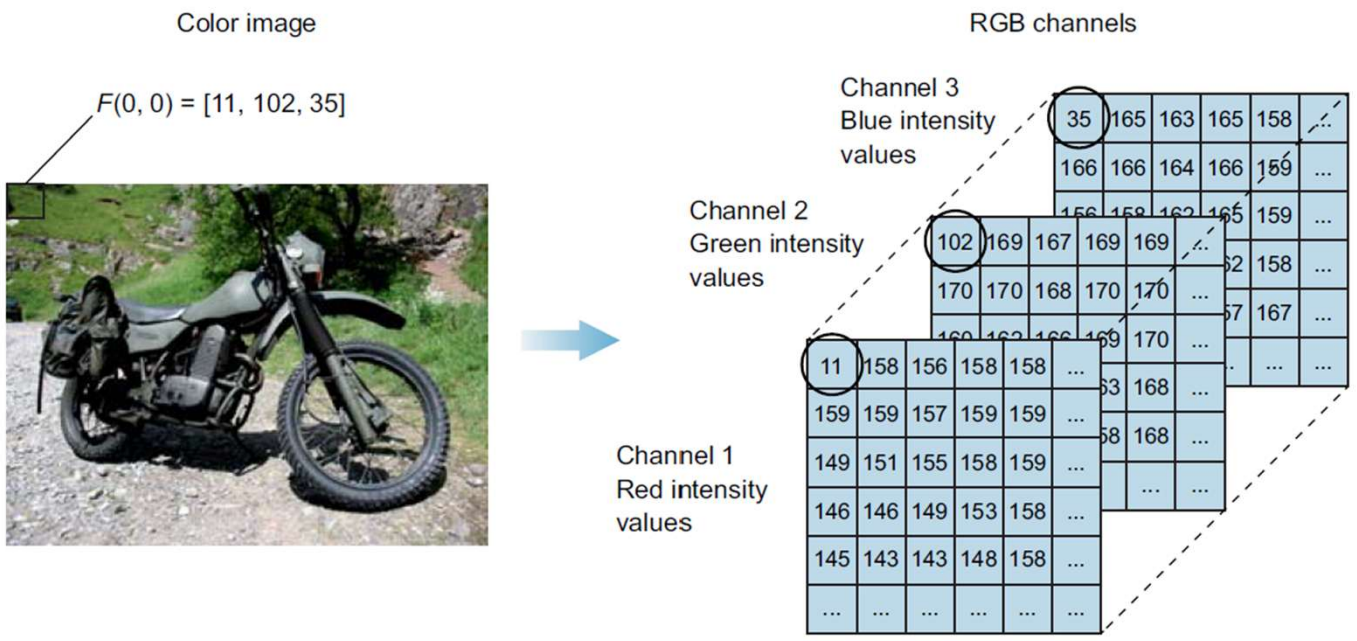


COORDINATE SYSTEM



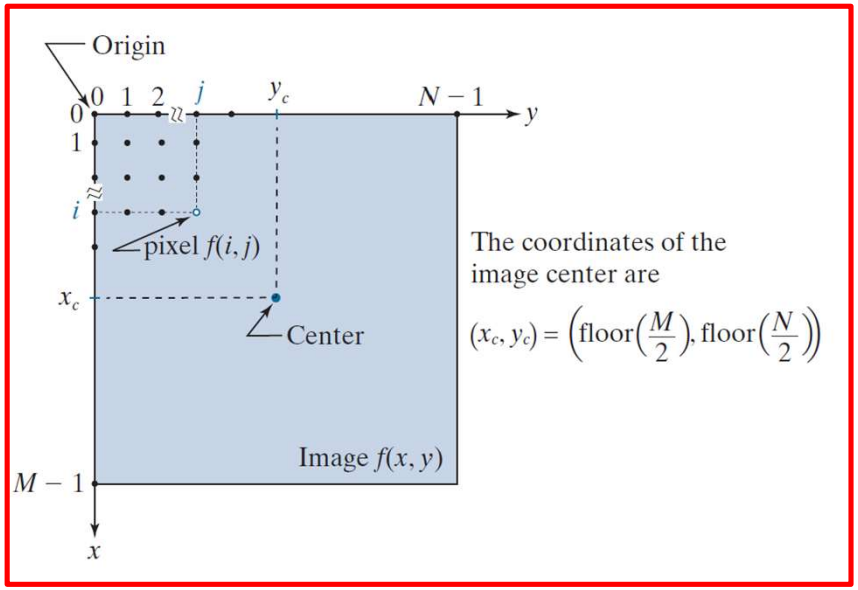


COLORED IMAGES



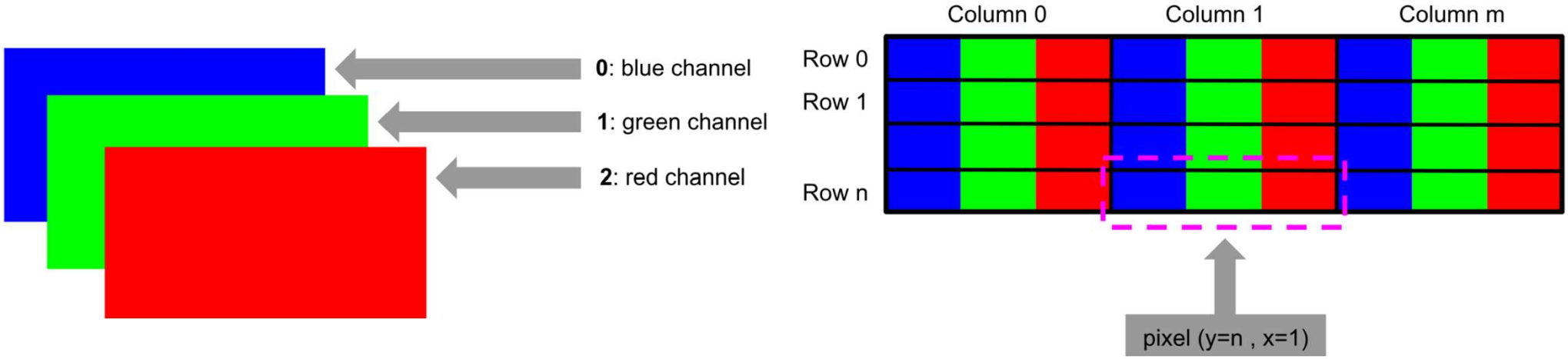
SIMPLE IMAGE PROCESSING (PIXEL-WISE)

- Arithmetic operations [saturation may occur]
- pixel'=pixel+k (brightening, darkening)
- pixel'=α pixel (simple contrast stretching)
- pixel'=α $pixel_1$+ β $pixel_2$ (sum of two images)
- pixel'=$\alpha pixel_1$+ (1-α)$pixel_2$ (linear blending)
- pixel'=|$pixel_1$-$pixel_2$| (simple motion detection)
- pixel' = 255 –pixel (inversion)
- Binary threshold
- pixel' = pixel > T : 255:0
- Logical operations (on binary images)
- pixel'=pixel & mask
- pixel'=pixel | mask
- pixel'=not pixel



Application	Transformation
Darken the image.	`G(x, y) = 0.5 * F(x, y)`
Brighten the image.	`G(x, y) = 2 * F(x, y)`
Move an object down 150 pixels.	`G(x, y) = F(x, y + 150)`
Remove the gray in an image to transform the image into black and white.	`G(x, y) = { 0 if F(x, y) < 130, 255 otherwise }`

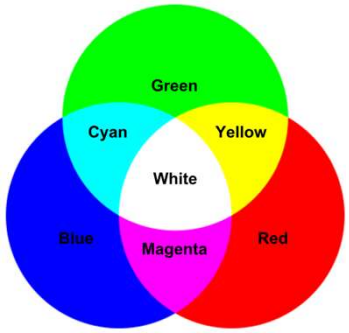
BGR ORDER IN OPENCV

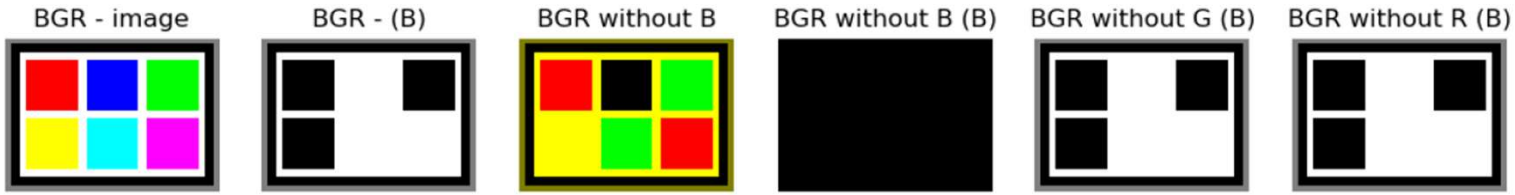


SPLITTING CHANNELS

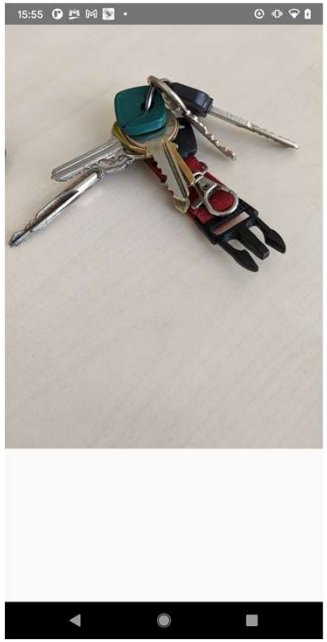
Image after the Red (R) and Green (G) channels are set to 0
Y=R+G=0
M=B+R=255
C=B+G=255

Without the Blue Channel:
Y→G
M→R

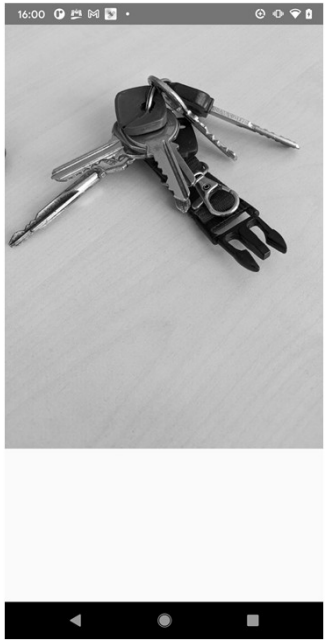




COLOUR SPACE TRANSFORMATIONS



Original

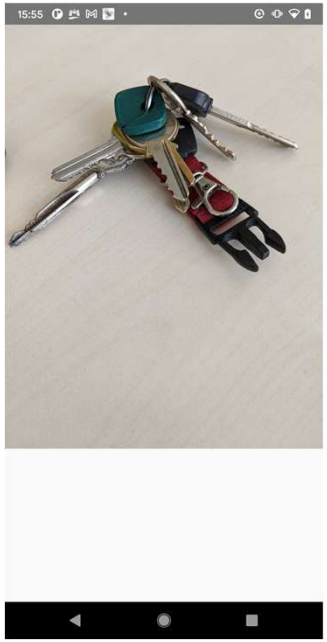


Grayscale

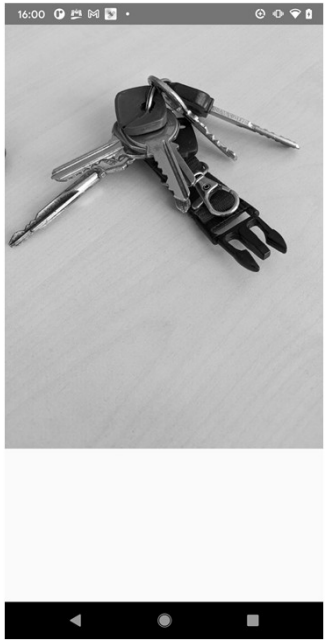
$$Y=0.299R+0.587G+0.114B$$

We convert images to grayscale because the luminance channel Y (the luminance) carries most of the structural information (texture, patterns..).
In addition, it simplifies the data (1 channel instead of 3), reduces noise from color components

COLOUR SPACE TRANSFORMATIONS



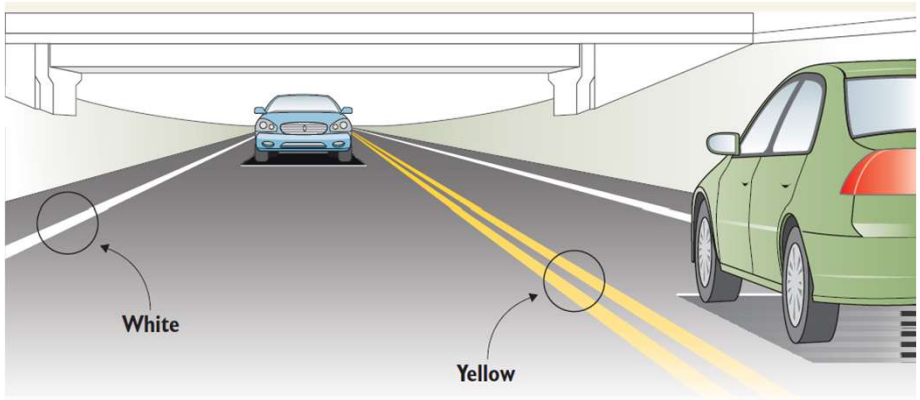
Original



Grayscale

$$Y=0.299R+0.587G+0.114B$$

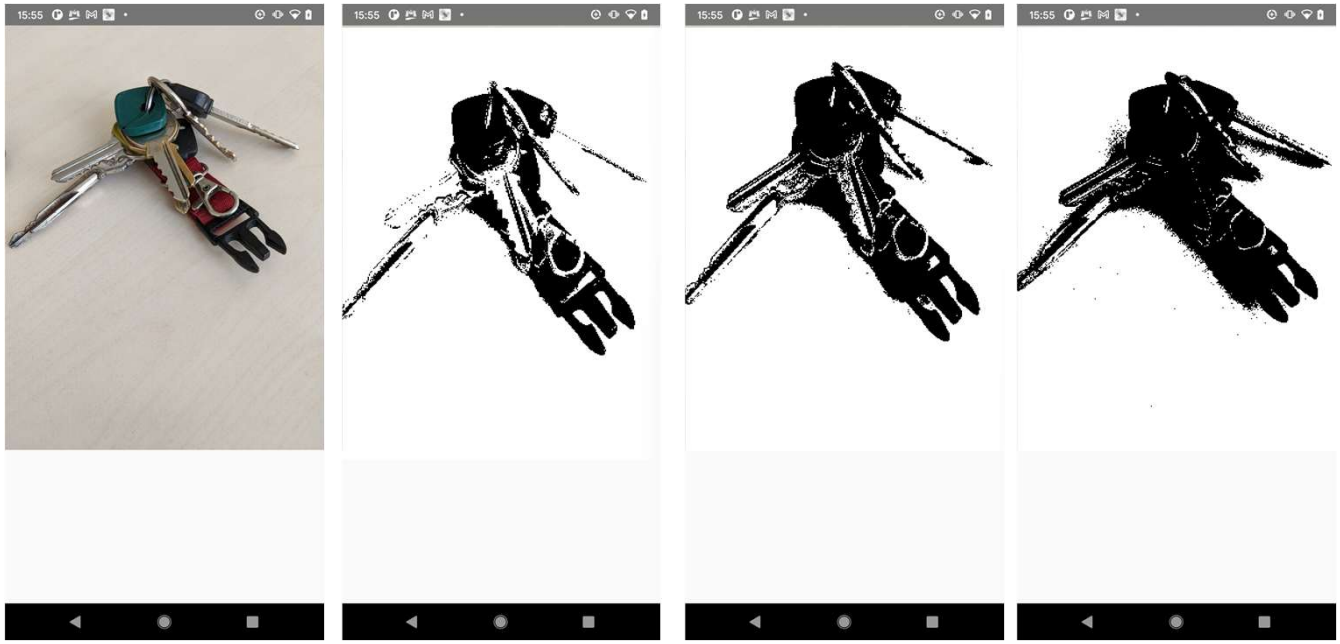
We convert images to grayscale because the luminance channel Y (the luminance) carries most of the structural information (texture, patterns..).
In addition, it simplifies the data (1 channel instead of 3), reduces noise from color components
Sometimes we may lose information when using grayscale



TRESHOLDING

- **Thresholding produces a binary image**, where each pixel is either **black or white**.
  It is widely used in image processing algorithms.
- For example, in classical **OCR**, thresholding is applied so that **text becomes white and the background becomes black**, making character extraction easier.

$$I_{bin}(x,y) = \begin{cases} 1 & \text{if } I(x,y) > T \\ 0 & \text{otherwise} \end{cases}$$



96 128 160

ADAPTIVE THRESHOLD

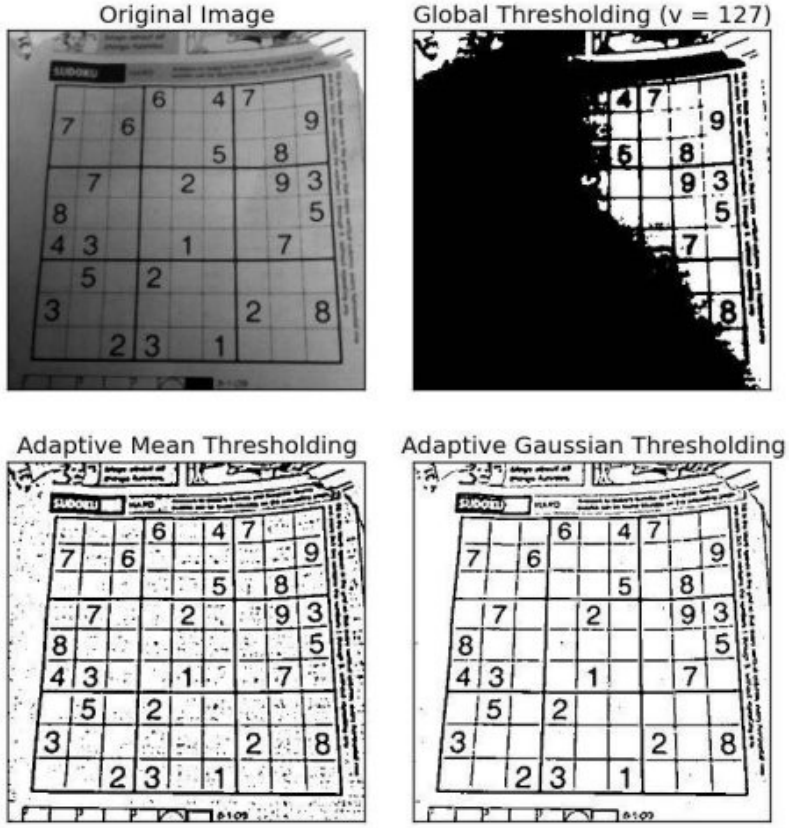
- Adaptive Thresholding uses a threshold map, i.e., a different threshold value for each pixel.
  The threshold at position (x, y) is computed as the average (or Gaussian-weighted average) of the neighboring pixels.

$$T(x,y) = \frac{1}{N^2} \sum I(x+i, y+j)$$

- This threshold map is then applied to the original image to produce the final binary output.

- The threshold is also the result of a convolution (next slides) of the image F with kernel h.

$$h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad G = F * h$$



CROMA KEYING

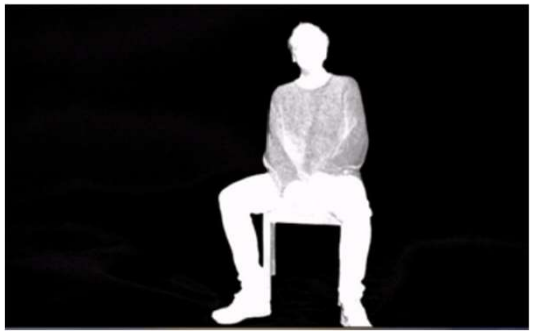
F(p)

$$M(p) = \begin{cases} 0, & \text{if } d(p) < T \quad \text{(pixel similar to key color} \rightarrow \text{remove)} \\ 1, & \text{otherwise} \quad \text{(pixel different from key color} \rightarrow \text{keep)} \end{cases}$$

$$d(p) = \sqrt{(R_p - R_K)^2 \; + \; (G_p - G_K)^2 \; + \; (B_p - B_K)^2}$$

Chroma keying builds a (binary) mask by measuring how close each pixel is to the key color.

$$F_{\text{out}}(p) \;=\; M(p)\,F(p) \;+\; \big(1 - M(p)\big)\,B(p)$$



Bad threshold T

Good threshold T

M(p)

IMAGE MATTING AND COMPOSITING

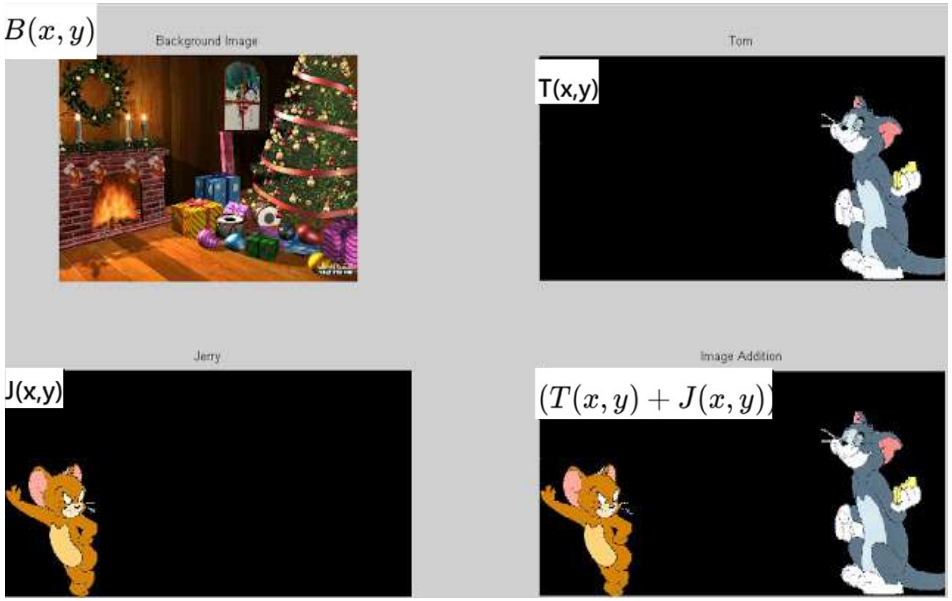
- Chroma keying is a simple form of matting, where pixels similar to a known background color are removed.
- Image matting and composition generalize this idea by estimating a continuous alpha matte and combining foreground (b) and background through alpha blending (c), allowing accurate extraction of fine details such as hair, fur, and transparent regions. (a) original, (d) final
- Images have a fourth channel, (BGRA). The alpha channel expresses opacity.

(a) (b) (c) (d)

$$I = \alpha F + (1 - \alpha) B$$

Chuang, Y.-Y., Curless, B., Salesin, D. H., and Szeliski, R. (2001). A Bayesian approach to digital matting. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pp. 264–271.

IMAGE COMPOSITION

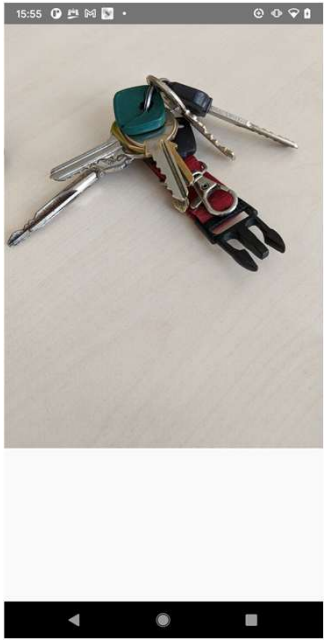


$$M(x,y) = \begin{cases} 1 & \text{Tom or Jerry} \\ 0 & \text{Background} \end{cases}$$

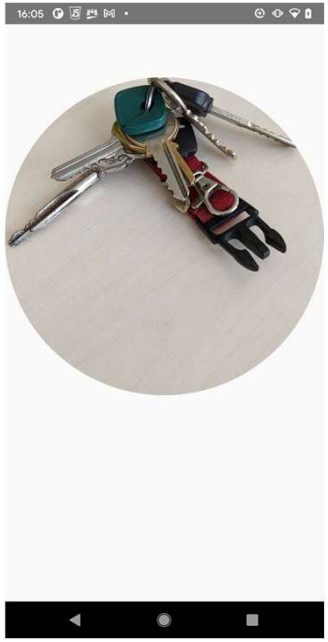
$$\boxed{Final = B(1-M) + (T+J)M}$$

https://www.imageeprocessing.com/2011/05/image-arithmetic.html

EXAMPLE OF LOGICAL OPERATIONS



Original

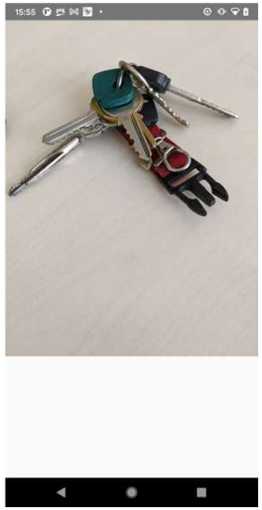


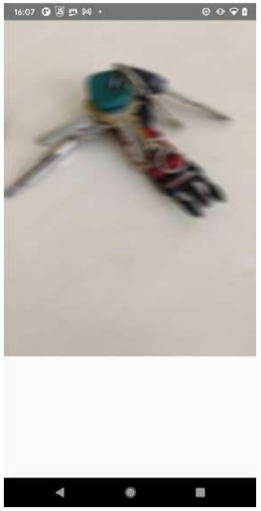
And

NEIGHBORS PROCESSING

- Output depends on surrounding pixels.
- Example: median filter, Sobel edge detection
- Convolutions: specialized neighborhood processing with a kernel moving over the image.
- Example: blur uses a kernel like adaptive mean threshold

Original





Blurred

$$g(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(x-i,\ y-j)\ h(i,j)$$

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

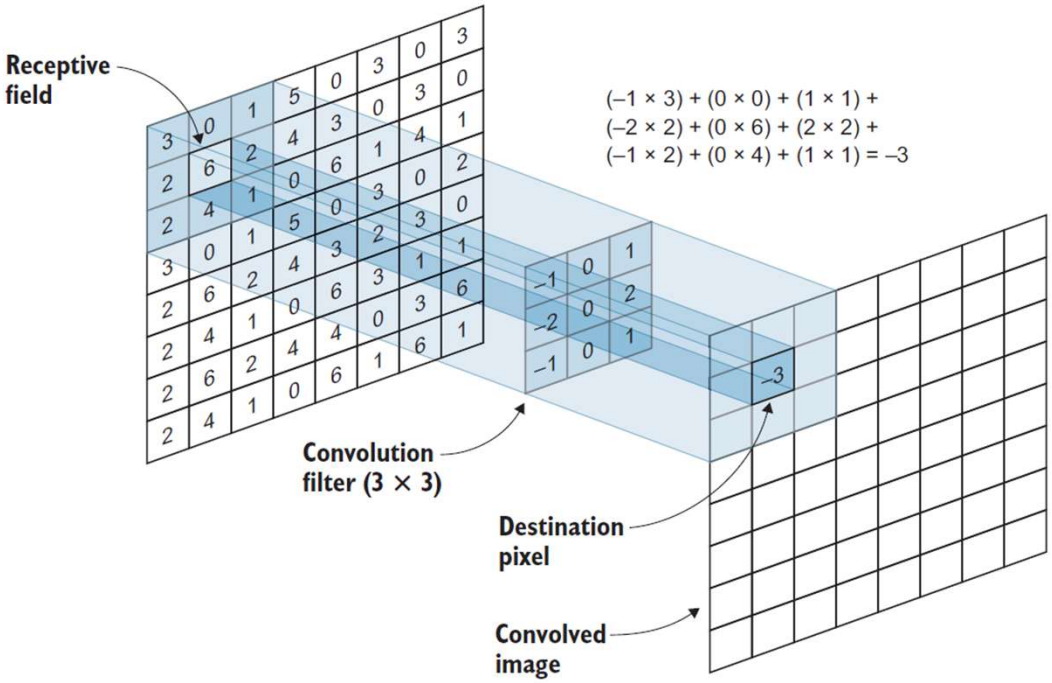
$h(x,y)$

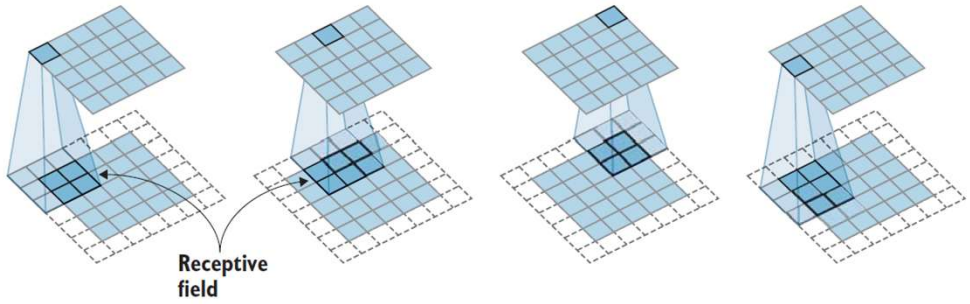
=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

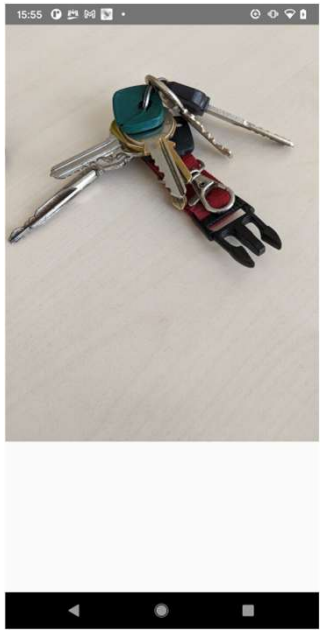
$g(x,y)$

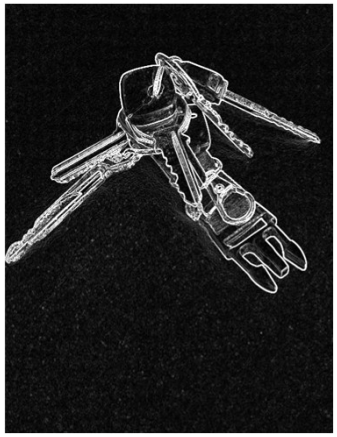
CONVOLUTION CONCEPTS



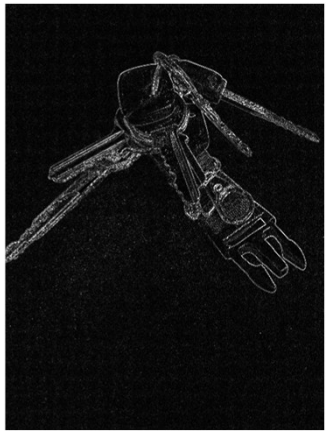


EDGE DETECTOR

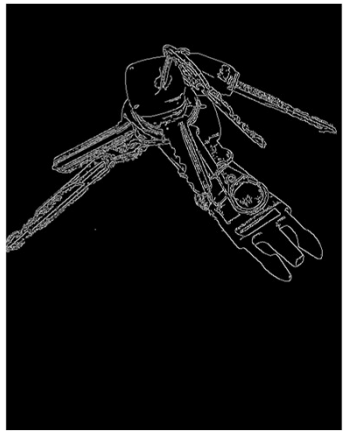




Sobel



Laplacian



Canny

ksize=3

cv2.Canny(gray, 100, 200)

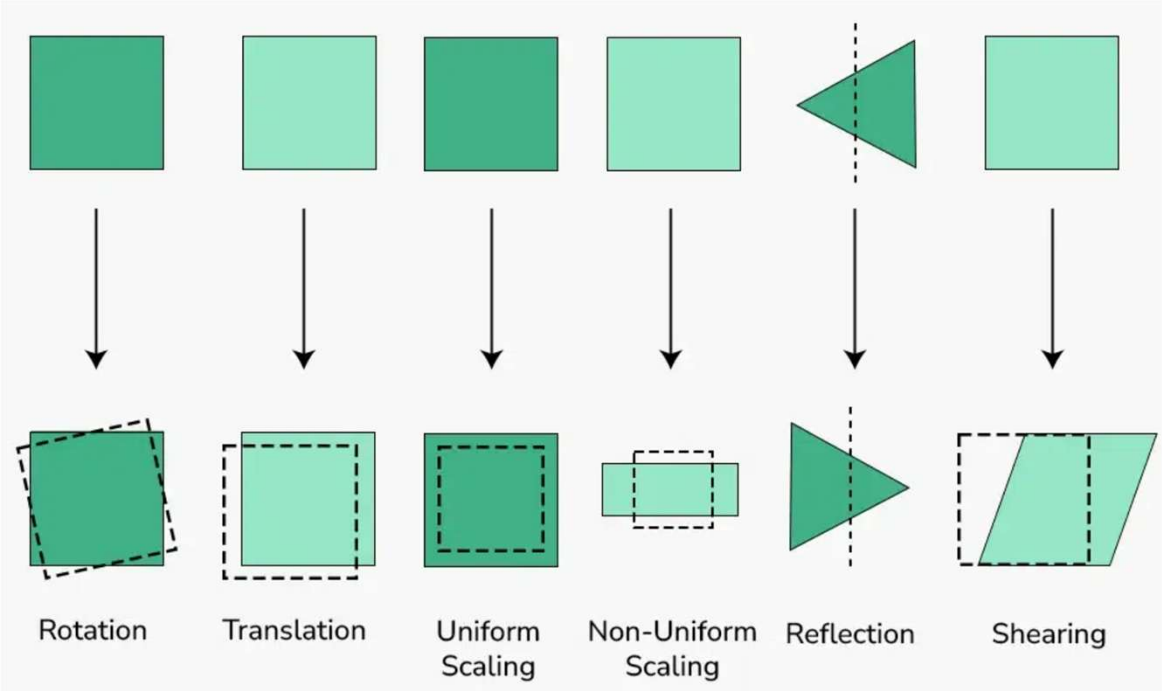
IMAGE-WISE PROCESSING

- Operations that use information from the entire image, not just individual pixels or local neighborhoods.
- Goal is to just global properties of the image, such as overall brightness, contrast, or tone distribution.
  - Histogram equalization: redistributes pixel intensities to make the histogram more uniform, enhancing contrast.
- Key point: unlike pixel-wise or neighborhood operations, each pixel's new value may depend on the statistics of the whole image.

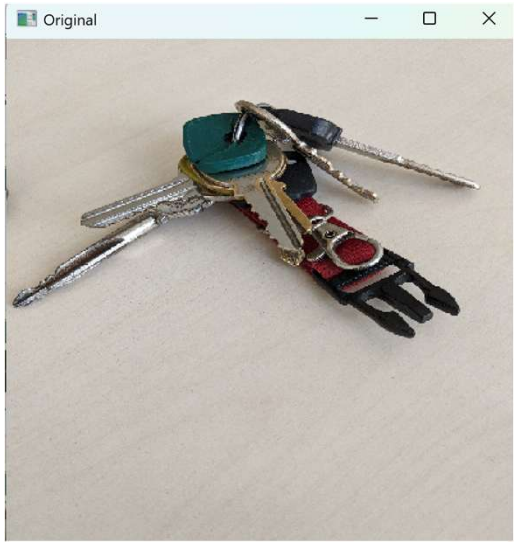
histogram equalization

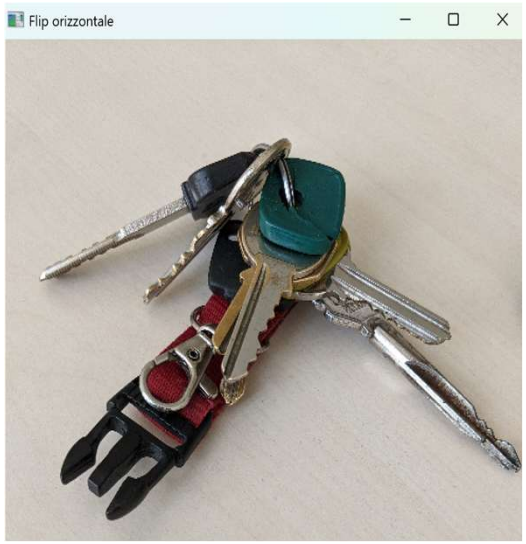
constrast Stretching

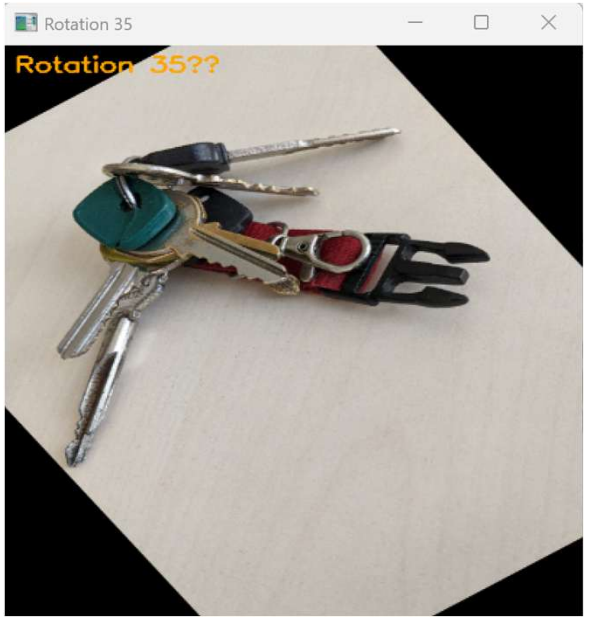
IMAGE GEOMETRIC TRANSFORMATION



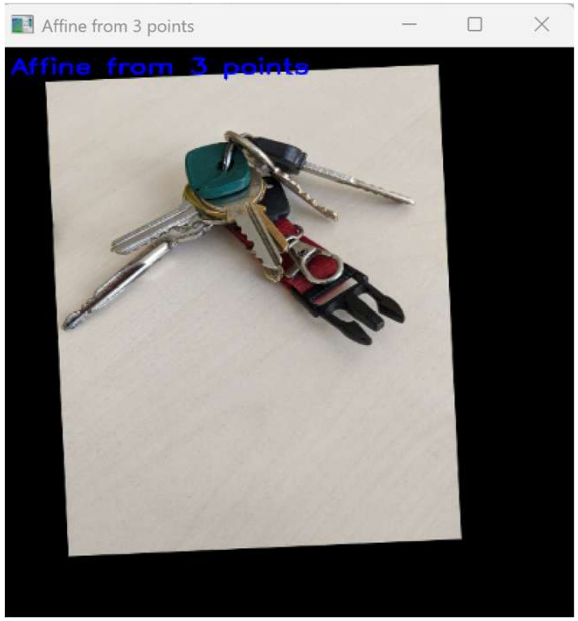
SOME GEOMETRIC TRANSFORMATIONS

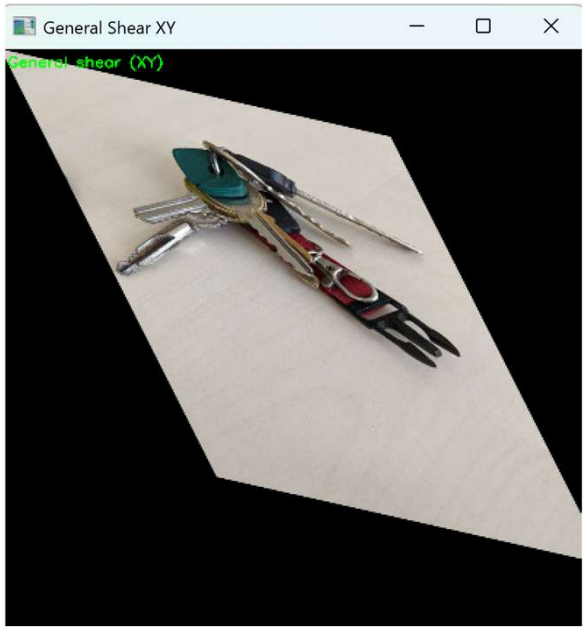


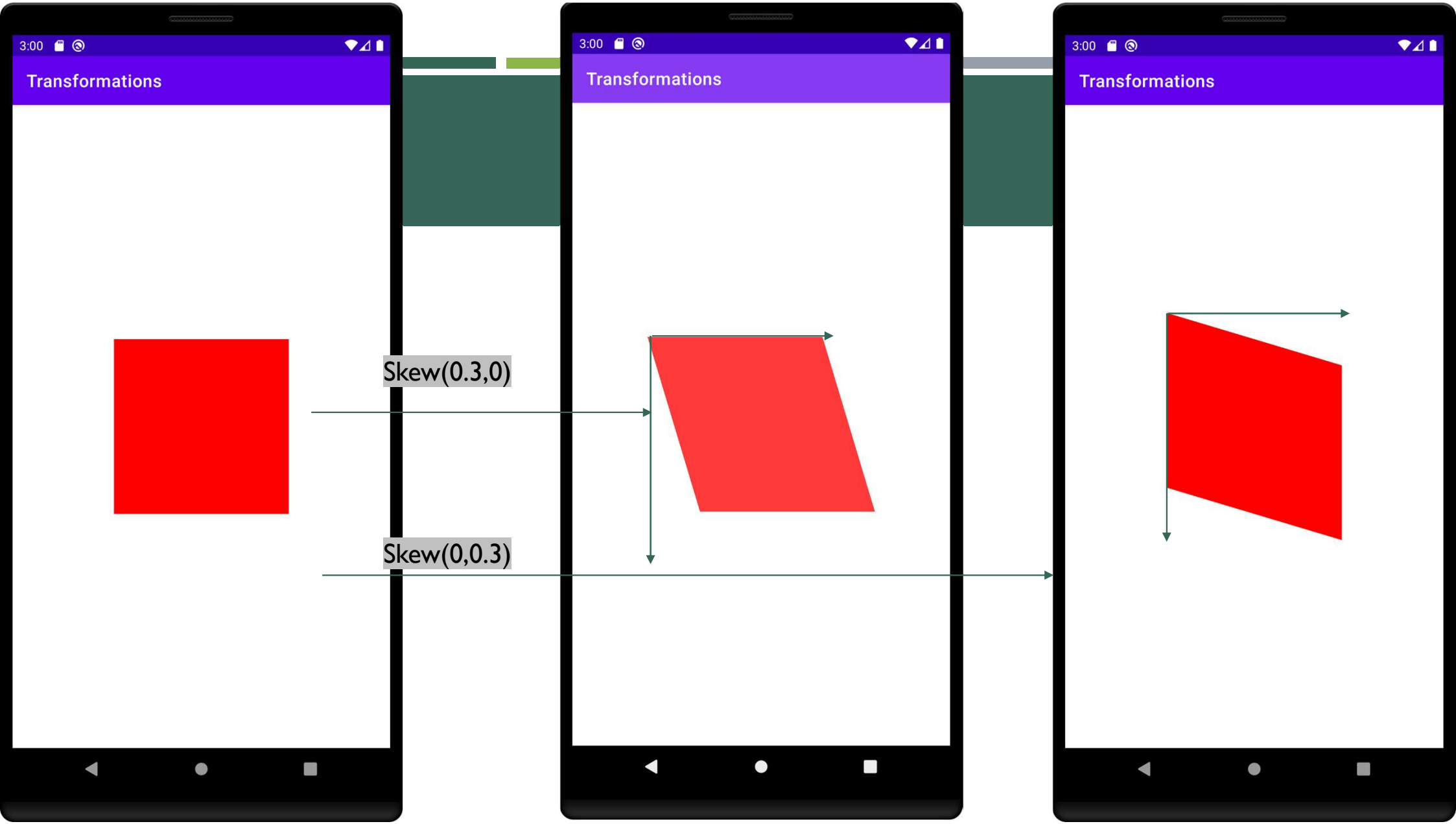




GEOMETRIC TRANSFORMATIONS






3:00
Transformations
Skew(0.3,0)
Skew(0,0.3)
3:00
Transformations
3:00
Transformations