



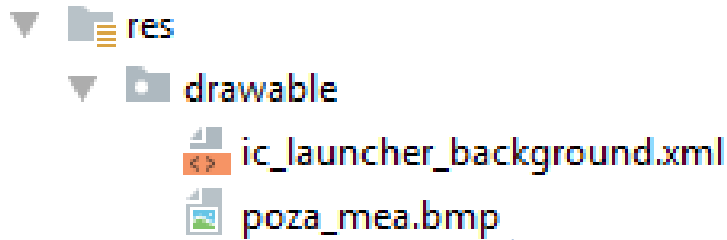
**Animation**



# How to display an image on screen

## Using only layout file

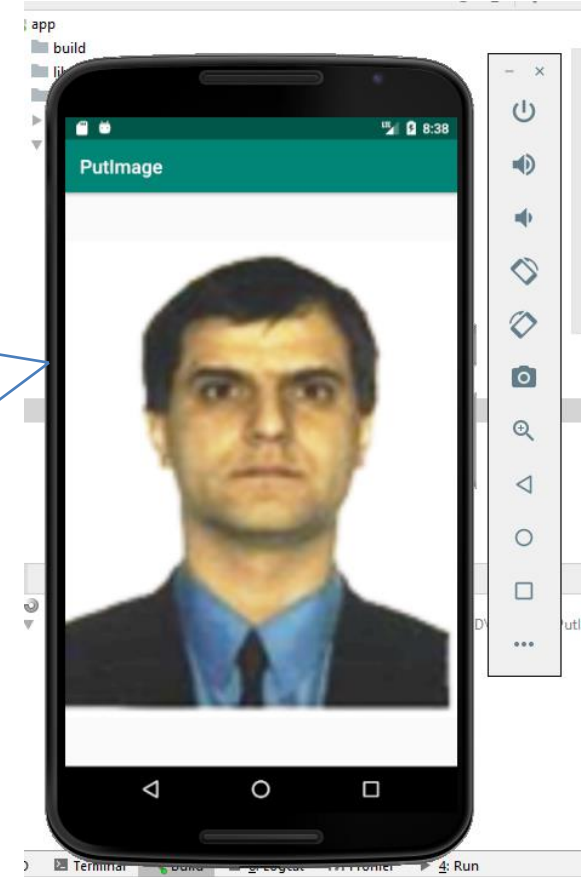
- Create a new project
- Put the picture you want to display in drawable subfolder



- Modify activity\_main.xml:

a)

```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:srcCompat="@drawable/poza_mea"  
    android:layout_centerVertical="true"  
    android:layout_centerHorizontal="true"/>
```



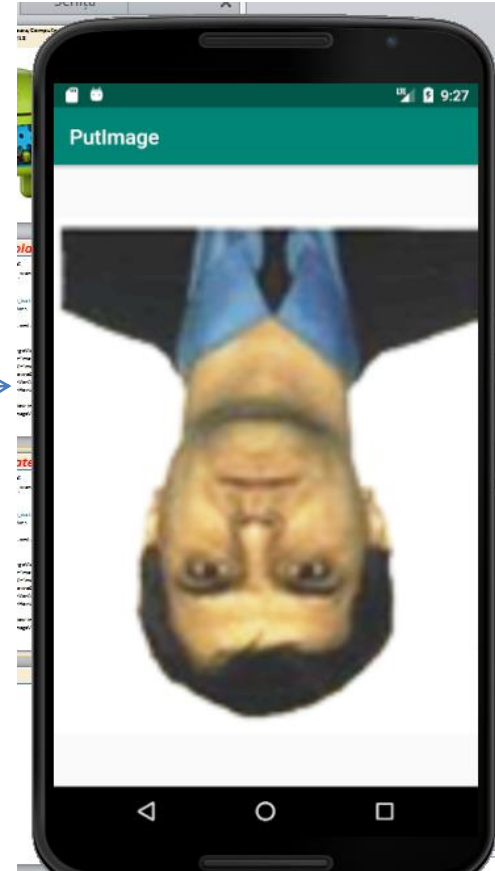
- b) Drag and drop ImageView in activity\_main.xml file,  
then double click on ImageView and choose the location of your picture.

*NOTE MainActivity: nothing to do*

# How to rotate an image

Modify activity\_main.xml:

```
<ImageView  
    android:id="@+id/imageView"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:srcCompat="@drawable/poza_mea"  
    android:rotation="180"  
    android:layout_centerVertical="true"  
    android:layout_centerHorizontal="true"/>
```

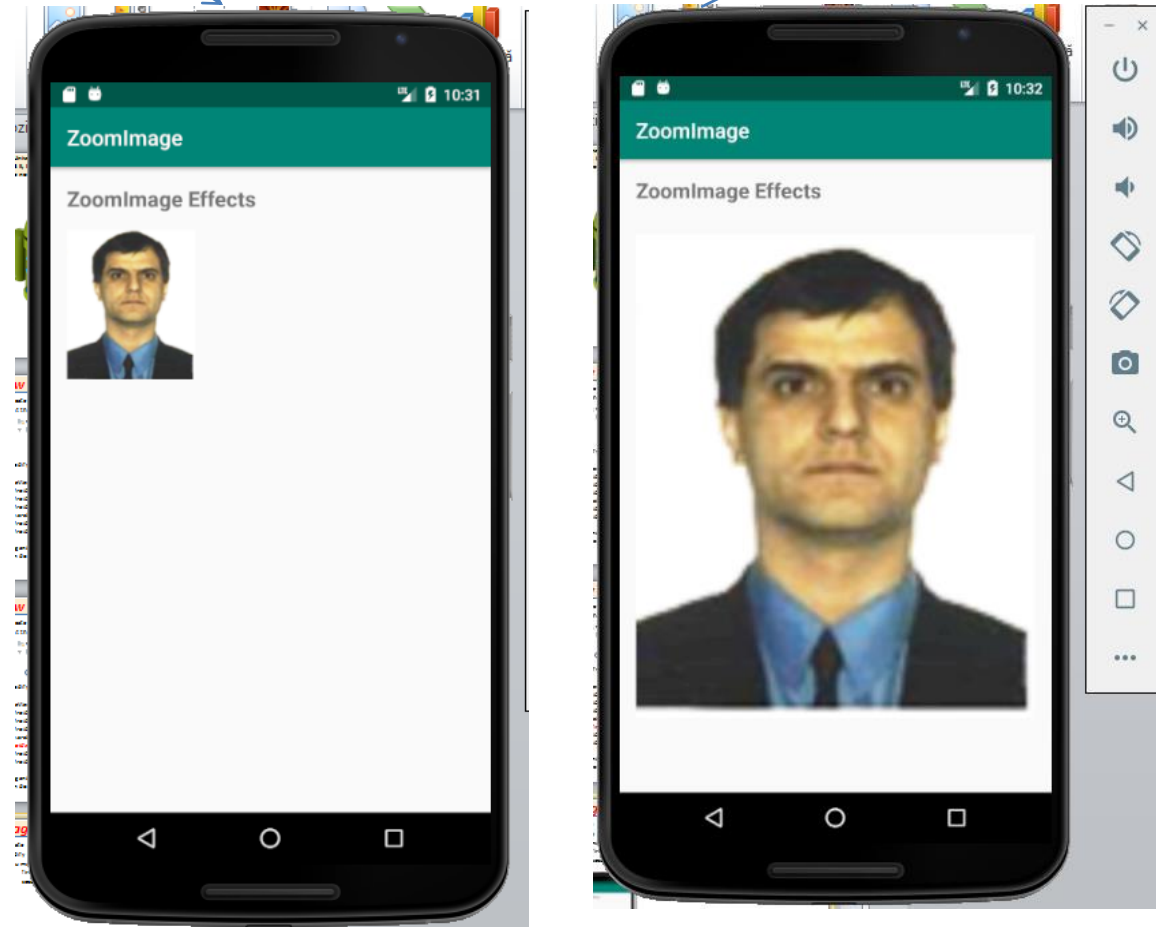


# Image Zoom – steps-

1. Create a new empty project and put your working image on *drawable* subfolder
2. Modify the layout
  - You must use *FrameLayout* style, because we have two different frames:
    - first: the original image
    - second: zooming

3. Modify MainActivity  
(click on image -> transition)

4. Rebuild and run app



# Image Zoom -activity\_main-

You must use *FrameLayout* style, because we have two different frames:

- first: the original image
- second: bigger image

*FrameLayout* is designed to block out an area on the screen to display a single item.

```
activity_main.xml x MainActivity.java x
1 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:id="@+id/container"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:layout_margin="16dp">
6     <LinearLayout
7         android:layout_width="match_parent"
8         android:layout_height="wrap_content"
9         android:orientation="vertical">
10
11         <TextView
12             android:layout_width="wrap_content"
13             android:layout_height="wrap_content"
14             android:layout_marginBottom="16dp"
15             android:text="ZoomImage Effects"
16             android:textSize="20sp"
17             android:textStyle="bold" />
18
19         <ImageView
20             android:id="@+id/imageview_image"
21             android:layout_width="120dp"
22             android:layout_height="140dp"
23             android:layout_marginRight="1dp"
24             android:scaleType="centerCrop"
25             android:src="@drawable/poza_mea" />
26     </LinearLayout>
27     <ImageView
28         android:id="@+id/expanded_image"
29         android:layout_width="match_parent"
30         android:layout_height="match_parent"
31         android:visibility="invisible" />
32 </FrameLayout>
```

This view is invisible, but it still takes up space for layout purposes.

# Image Zoom –MainActivity- libraries used

Android includes different animation APIs depending on what type of animation you want (animate a bitmap, physics-based motion)

*Animator* is a superclass for classes which provide basic support for animations

*AnimatorSet* provide a class used to play Animatorobjects in a specified order.

*ObjectAnimator* provides support for animating properties on target objects

*DecelerateInterpolator* is a public class used to change the speed of animation (first quickly and then decelerates)

*GraphicsPoint* holds two integer coordinates

*GraphicsRect* holds four integer coordinates for a rectangle. The rectangle is represented by the coordinates of its 4 edges (left top, right bottom).

*ImageView* is used to display image resources (bitmaps, etc)

```
import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.view.animation.DecelerateInterpolator;
import android.graphics.Point;
import android.graphics.Rect;
import android.os.Build;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ImageView;
```

# ***Image Zoom –MainActivity-***

**1. You must create a class from superclass Animator:**

```
private Animator mCurrentAnimator;
```

**2. Declare a variable used to set the animation duration:**

```
private int mAnimationDuration;
```

**To set this, you have two options:**

```
mAnimationDuration = getResources().getInteger( android.R.integer.config_longAnimTime);
```

or

```
mAnimationDuration = getResources().getInteger( android.R.integer.config_mediumAnimTime);
```

or

```
mAnimationDuration = getResources().getInteger(android.R.integer.config_shortAnimTime);
```

NOTE: on average, the ratio between these values is 500-400-200 ms, when default value is 300

# Image Zoom –MainActivity-

Two rectangles are used to fit the image on initial state and on final state:

```
final Rect startBounds = new Rect();
```

```
final Rect finalBounds = new Rect();
```

Four integer that represent coordinates of its two corners: left top, right bottom

---

A variable called *Offset* (Point type, i.e. 2 integers for x and y coordinates) is used to compute image moving:

```
final Point Offset = new Point();
```

---

We need to compute the *Scale*, a ratio between *startBounds* and *finalBounds* :

a)  $Scale = (\mathbf{float}) \text{startBounds.width()} / \text{finalBounds.width}();$

b)  $Scale = (\mathbf{float}) \text{startBounds.height()} / \text{finalBounds.height}();$

?

*Width* is the rectangle's width.

Note: method does not check for a valid rectangle (i.e. left <= right) so the result may be negative!!!!!!!!!!!!!!

a) When  $\text{finalBounds.width()} / \text{finalBounds.height()} > (\mathbf{float}) \text{startBounds.width()} / \text{startBounds.height}()$   
i.e. the final picture is greater than the initial picture



# *Image Zoom –MainActivity-*

After these, AnimatorSet class is instantiated:

```
AnimatorSet set = new AnimatorSet();
```

and after, the new image (with zoom) is put on screen on new coordinates using a given duration

Of course, this process must have a *start()* and an *end()*

Because in our app a button was used to start the zoom process, a method *OnClick()* is necessary

# *Using* android.graphics.Matrix

Library *android.graphics.Matrix* is a class that can be used to process images in android.

In fact, this class holds a 3x3 matrix for transforming coordinates of an image object

There are many methods that can be used to animate images: rotate, scale, screw, translate

It's not complicated to make animations using these methods.

See <https://developer.android.com/reference/kotlin/android/graphics/Matrix>

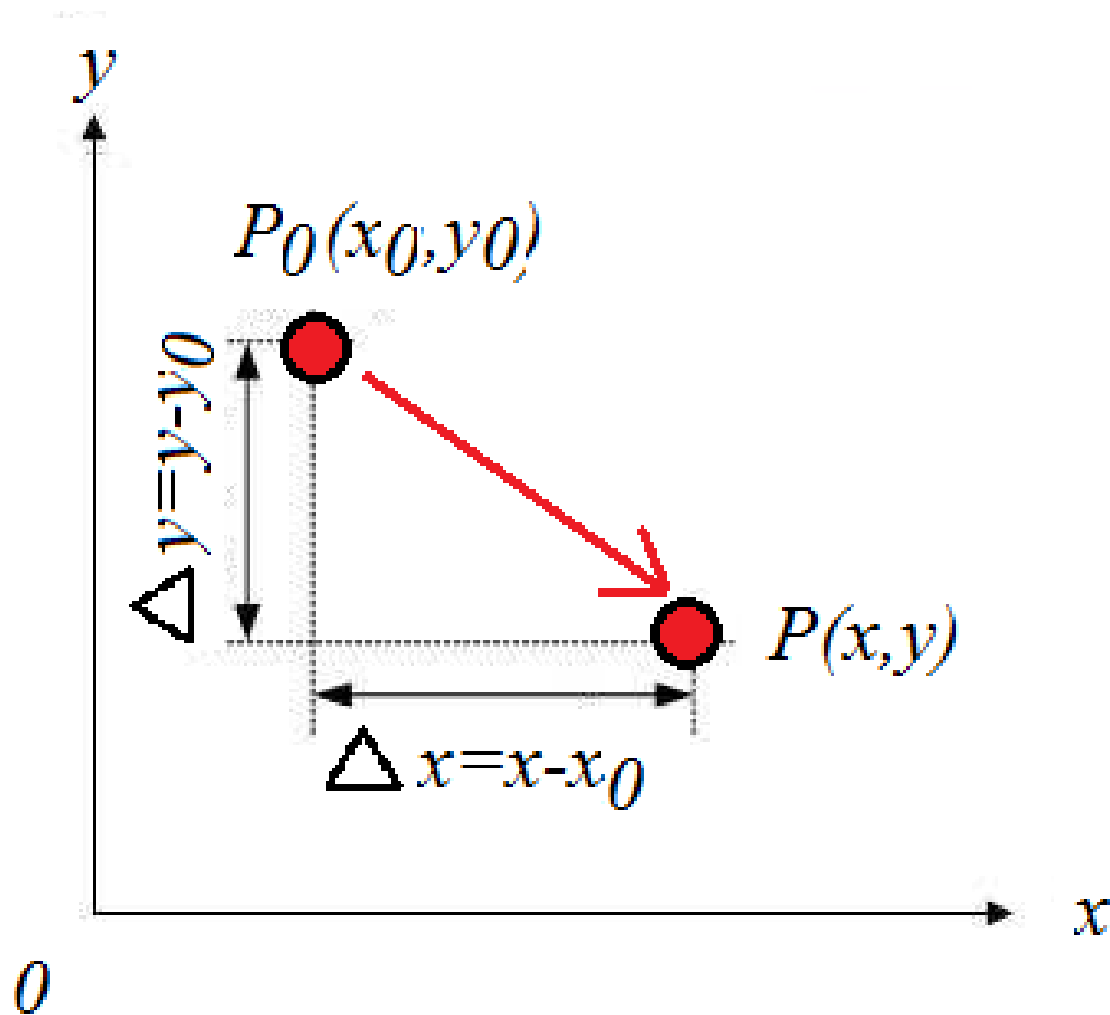
From this library point of view, an image is composed of pixels

To animate an image consist in transforming each pixel of the image

# Image Translation

What a translation is?

Move an image from  $P_0$  to  $P$ , without changing the image (size, shape...)



# Image Translation – activity\_main.xml

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:layout_editor_absoluteY="10dp"
    tools:layout_editor_absoluteX="10dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/buttonTranslateImage"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Translation" />
    </LinearLayout>
    <ImageView
        android:id="@+id/imageViewOriginal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minWidth="200sp"
        android:src="@drawable/poza_mea" />
    <ImageView
        android:id="@+id/imageViewMatrix"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minWidth="200sp" />
</LinearLayout>
```

id constraint constraintLayout

# *Image Translation – MainActivity-*

```
import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

import java.util.Random;
```

# Image Translation – MainActivity-

```
<ImageView  
    android:id="@+id/imageViewOriginal"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:minWidth="200sp"  
    android:src="@drawable/poza_mea" />
```

```
final ImageView imageViewOriginal = (ImageView) findViewById(R.id.imageViewOriginal);  
BitmapDrawable originalBitmapDrawable = (BitmapDrawable) imageViewOriginal.getDrawable();  
final Bitmap originalBitmap = originalBitmapDrawable.getBitmap();  
final int originalImageWidth = originalBitmap.getWidth();  
final int originalImageHeight = originalBitmap.getHeight();  
final Config originalImageConfig = originalBitmap.getConfig();
```

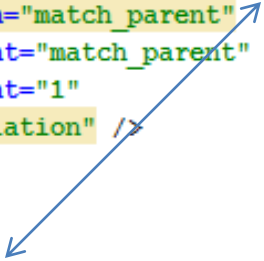
wraps a bitmap

?

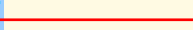
If bitmap's internal config is in one of the public formats, getConfig return that config, otherwise return null.

# Image Translation – MainActivity -

```
<Button
    android:id="@+id/buttonTranslationImage"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:text="Translation" />
```



```
Button buttonTranslation = (Button) findViewById(R.id.buttonTranslationImage);
buttonTranslation.setOnClickListener(new View.OnClickListener() {
```

```
    public void onClick(View view) {  The action after button was pressed
```

```
        // Generate random translation values (for x and y).
```

```
        Random random = new Random();
```

```
        int xTranslation = random.nextInt( bound: 1000);
```

```
        int yTranslation = random.nextInt( bound: 2000);
```

```
        this.translationImage(xTranslation, yTranslation);
```

```
    }
```



It's a user method

# ***Image Translation*** – Use Matrix class to process images

**1. Create a new Bitmap object** based on the original image's width and height. `Bitmap translateBitmap = Bitmap.createBitmap(originalImageWith + xTranslate, originalImageHeight + yTranslate, originalImageConfig);`

**2. Create a Canvas object** based on above Bitmap object. `Canvas translateCanvas = new Canvas(translateBitmap);`

**3. Create Matrix object**, and use it's method to set transformation info. `Matrix translateMatrix = new Matrix(); // Set x y translate value. translateMatrix.setTranslate(xTranslate, yTranslate);`

**4. Draw original bitmap image** to the newly created Canvas using Matrix effect. So the created Bitmap in step 1 will has the new image effect. `translateCanvas.drawBitmap(originalBitmap, translateMatrix, new Paint());`

**5. Set the newly created Bitmap to an ImageView** to show the transformation effect. `imageViewOriginal.setImageBitmap(translateBitmap);`



# Image Translation – MainActivity-

```
private void translationImage(int xTranslation, int yTranslation)
{
    //calculate the size of the image after the transformation.
    Bitmap translationBitmap = Bitmap.createBitmap( width: originalImageWith + xTranslation,
        height: originalImageHeight + yTranslation, originalImageConfig);

    Canvas translationCanvas = new Canvas(translationBitmap);

    Matrix translationMatrix = new Matrix();

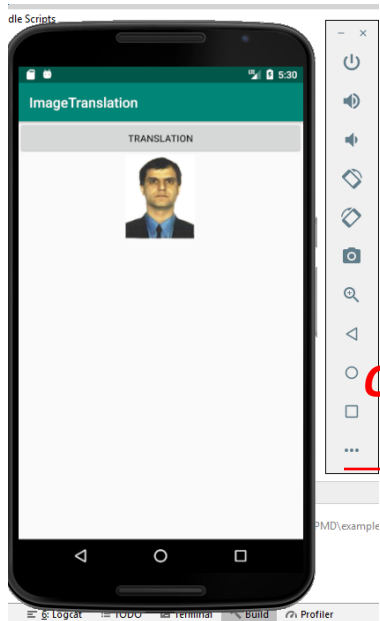
    // Set x and y translation values.
    translationMatrix.setTranslate(xTranslation, yTranslation);

    Paint paint = new Paint();
    translationCanvas.drawBitmap(originalBitmap, translationMatrix, paint);
    imageViewOriginal.setImageBitmap(translationBitmap);
}
```

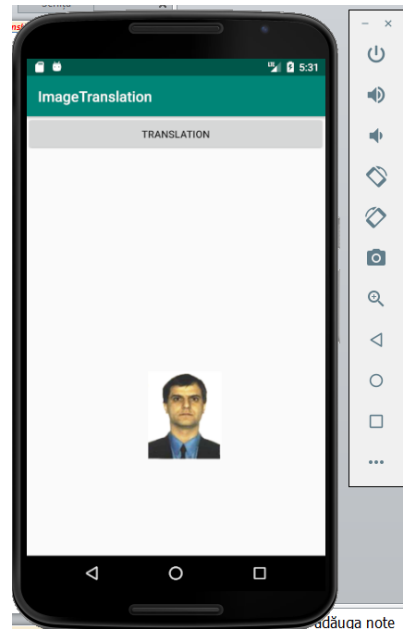
That is all. Build and run the app

# Image Translation – Final app-

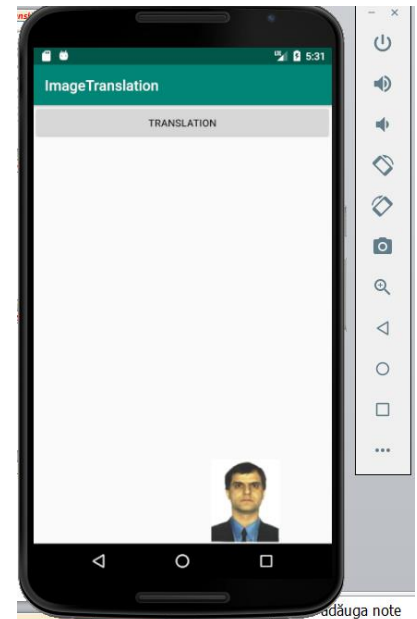
That is all. Build and run the app



*OnClick()*



*OnClick()*



# Image Rotation - what a rotation is-

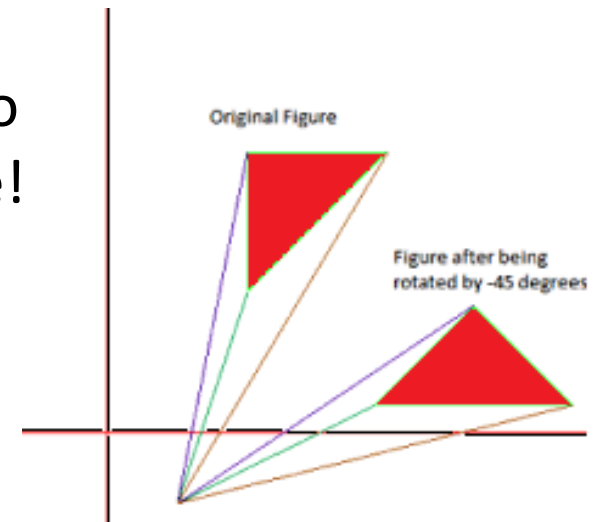
Rotation comes from geometry.

There are 2D (in plane) and 3D (in space) rotation

A 2D rotation is a circular movement of an object (in a 2D plane) around a center (or point) of rotation. For this we have one rotation axis

A three-dimensional object can be rotated around an infinite number of imaginary lines called *rotation axes*

NOTE: In the following we will refer to 2D rotation  
After a rotation the size and shape are the same!



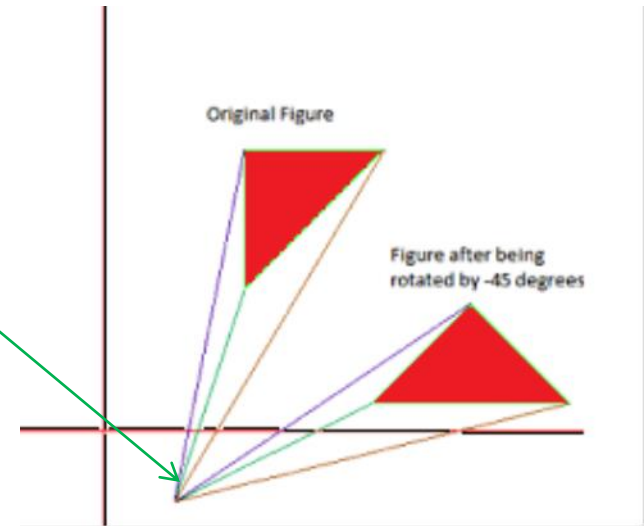
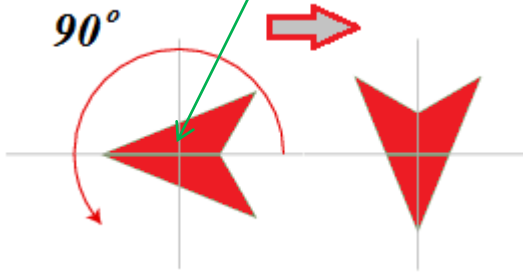
# Image Rotation - what a rotation is-

In case of a 2D rotation there are two possible situations:

- a) we know rotation angle and rotation axis and we want to obtain/compute the final position
- b) we know the axis and the position and we want to know the angle

Note: the axis of rotation is perpendicular to the plane (image), so, the most important are the coordinates of point where axis intersects the plan.

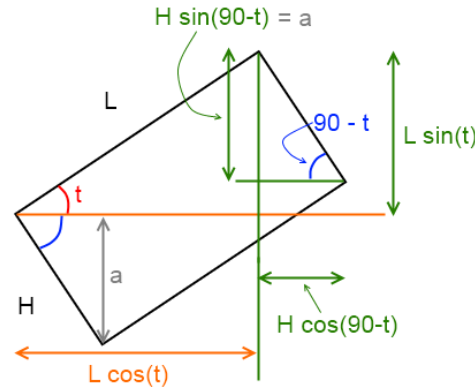
This point can be inside or outside related to image



In our project, we will be in case a) and the intersection point is inside

# Image Rotation -!-

More math is necessary to make a 2D rotation of an image and is not easy to implement all



But... it is very easy if we use Matrix library in AS where we have a friendly method called *setRotate*

And there is not very hard, because the original image (bitmap) is wrapped in a matrix

So, let use this library to rotate a 2D object, where rotation axis intersects image on its central point:  $x/2$  and  $y/2$

# Image Rotation – activity\_main.xml

The layout is similar like in previous example (Translation app).

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.imagerotation.MainActivity">
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        tools:layout_editor_absoluteY="8dp"
        tools:layout_editor_absoluteX="8dp">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
            <Button
                android:id="@+id/buttonRotateImage"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Rotate" />
        </LinearLayout>
        <ImageView
            android:id="@+id/imageViewOriginal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minWidth="200sp"
            android:src="@drawable/poza_mea" />
        <ImageView
            android:id="@+id/imageViewMatrix"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minWidth="200sp" />
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

# *Image Rotation – MainActivity -*

The most important used method :

**public void setRotate (float degrees, float px, float py)**

Set the matrix to rotate by the specified number of degrees, with a pivot point at (px, py). The pivot point is the coordinate that should remain unchanged by the specified transformation.

Another form of setRotate is

**public void setRotate (float degrees)**

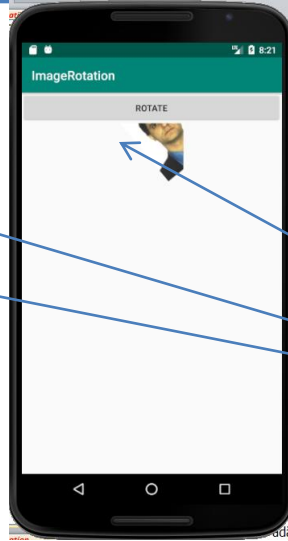
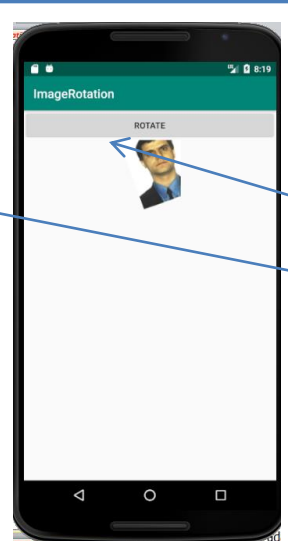
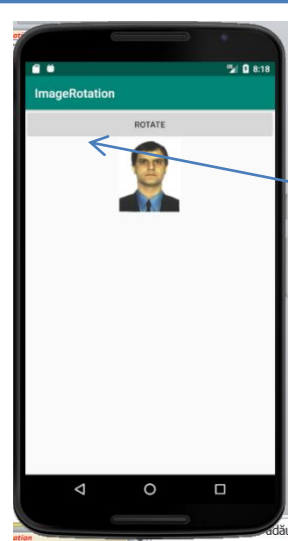
set the matrix to rotate about (0,0) by the specified number of degrees.

(0,0) / (px,py) point refers the left-upper corner of our image

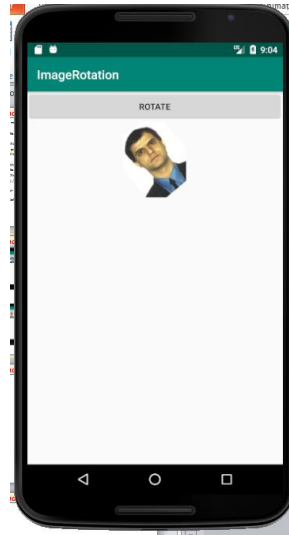
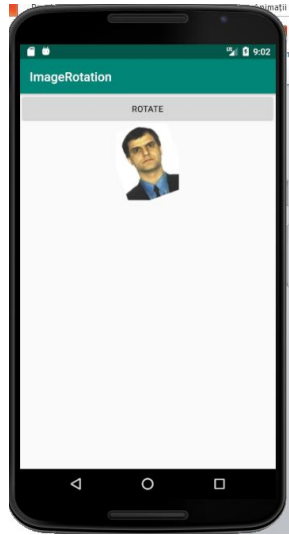
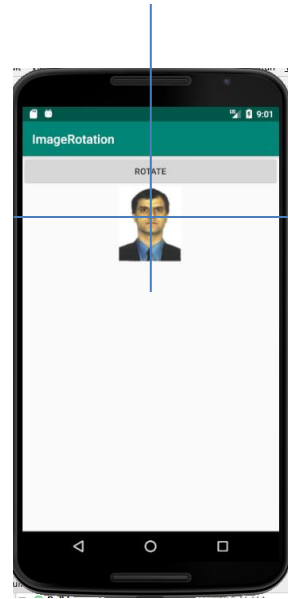
In our MainActivity.java this function is:

```
setRotate(rotateDegree, originalBitmap.getWidth()/2, originalBitmap.getHeight()/2);
```

# Image Rotation



setrotate(degree)  
left-top corner (0,0)



setrotate(deg, pxWidth/2, pyHeight/2)  
(central point)



# Scale Image

The image is reduced to the information that can be carried by the smaller image. There are many algorithms to do this: *nearest-neighbor interpolation, Lanczos resampling, Fourier-transform method, vectorization, neural networks methods,...*

*In Matrix class we have two dedicated methods for this operation:*

- **setScale(float scaleX, float scaleY)** : Scale image, scaleX and scaleY are the scaling ratio in X and Y direction.
- **setScale(float scaleX, float scaleY, float x, float y)** : Similar with setScale(float scaleX, float scaleY), but axis is (x, y).

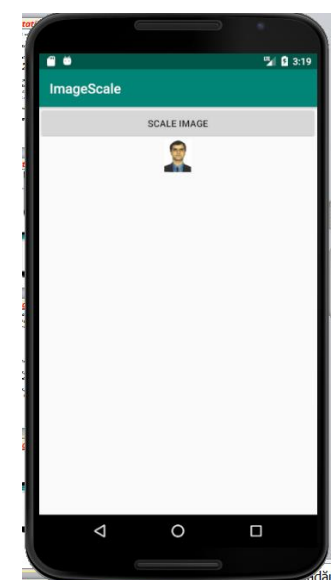
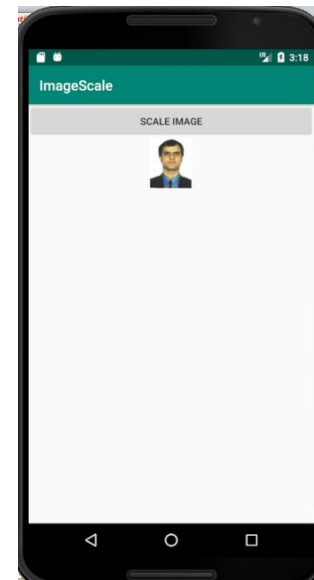
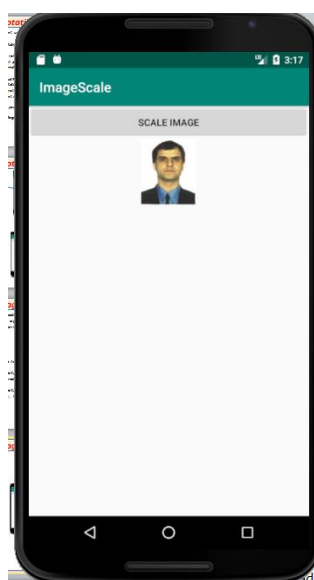
**Scaling** defines the size of the image. You can define two values — one for the x-axis and the other for the y-axis. But with scale you can also set a pivot point.

The **pivot** point defines which point will be unchanged by the transformation. By default it is at 0, 0 — the top-left point — meaning the image will stretch to the right and bottom, leaving the top-left unchanged (**setScale(0.5f, 0.5f)**)

If you want to scale the image from the centre, you can set the pivot to the centre of the image. (**setScale(0.5f, 0.5f, dWidth / 2f, dHeight / 2f)**)

Negative scale values-> mirror the image around an axis (or two).

# Scale Image

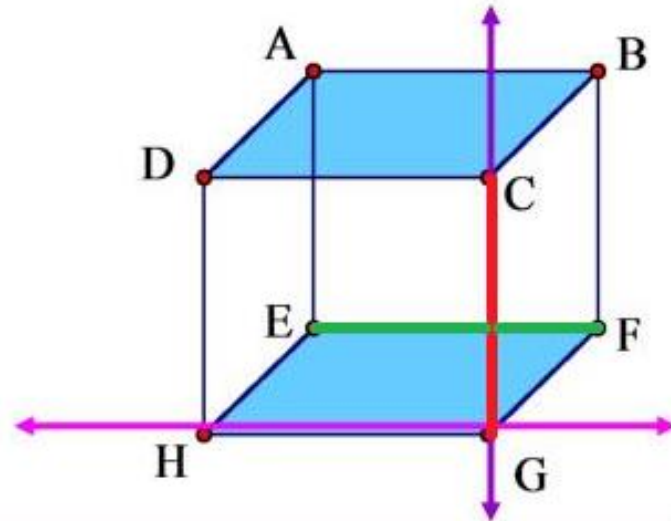


# Skew Image

## Skew Lines and Parallel Planes

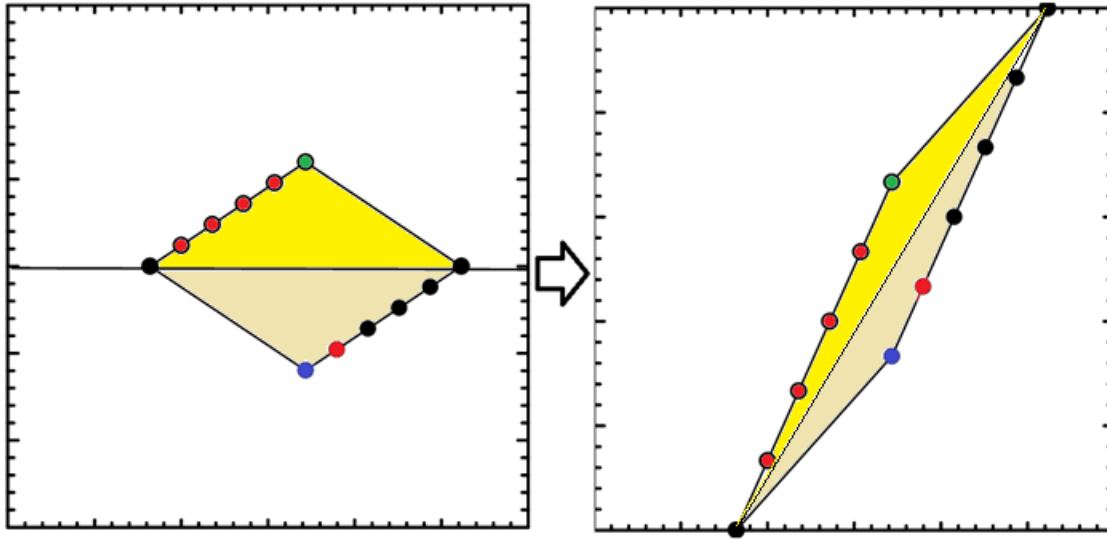
- Definition: Two lines are **skew** if they do not intersect and are not in the same plane (not coplanar). *Ex:*  $\overline{CG}$  and  $\overline{EF}$
- All planes are either parallel or intersecting.
- **Parallel planes are two planes that do not intersect.**

*Ex:* Plane ABC and Plane EFG



Two lines are skew lines if they **do not lie in the same plane**. Skew lines **never intersect**.

# Skew Image



No transformation

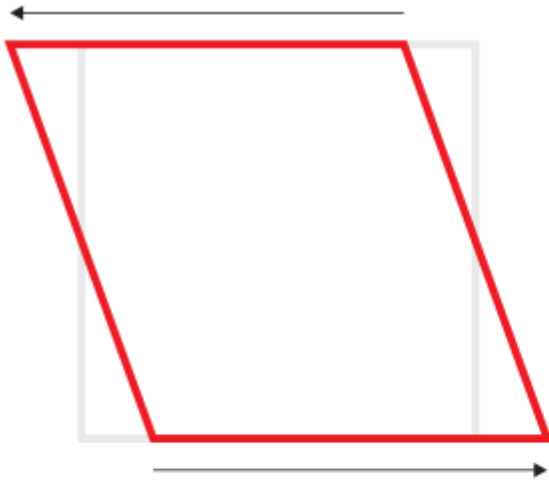


transform: skewY(30deg);

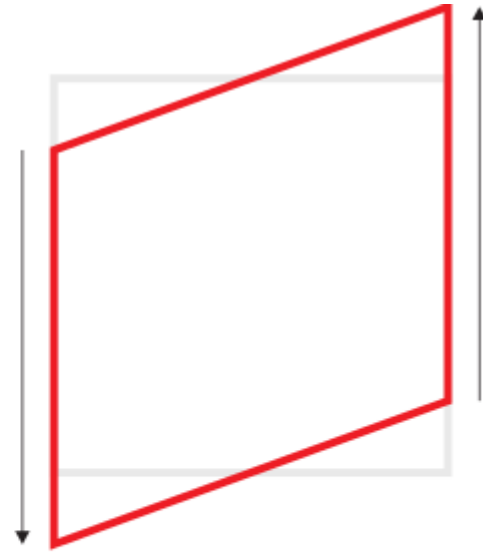


transform: skewY(-30deg);

# *Skew Image*



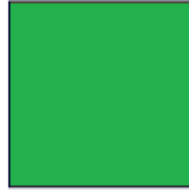
horizontal  
skew  $20^\circ$



vertical  
skew  $20^\circ$

# Skew Image

width = 100  
height = 100



$$\text{scaleX} = (1/\cos(\text{degree})) * 100$$

skew degree = 30



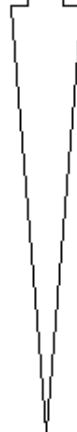
scaleX = 115.47

skew degree = 45



scaleX = 141.42

vertical skew



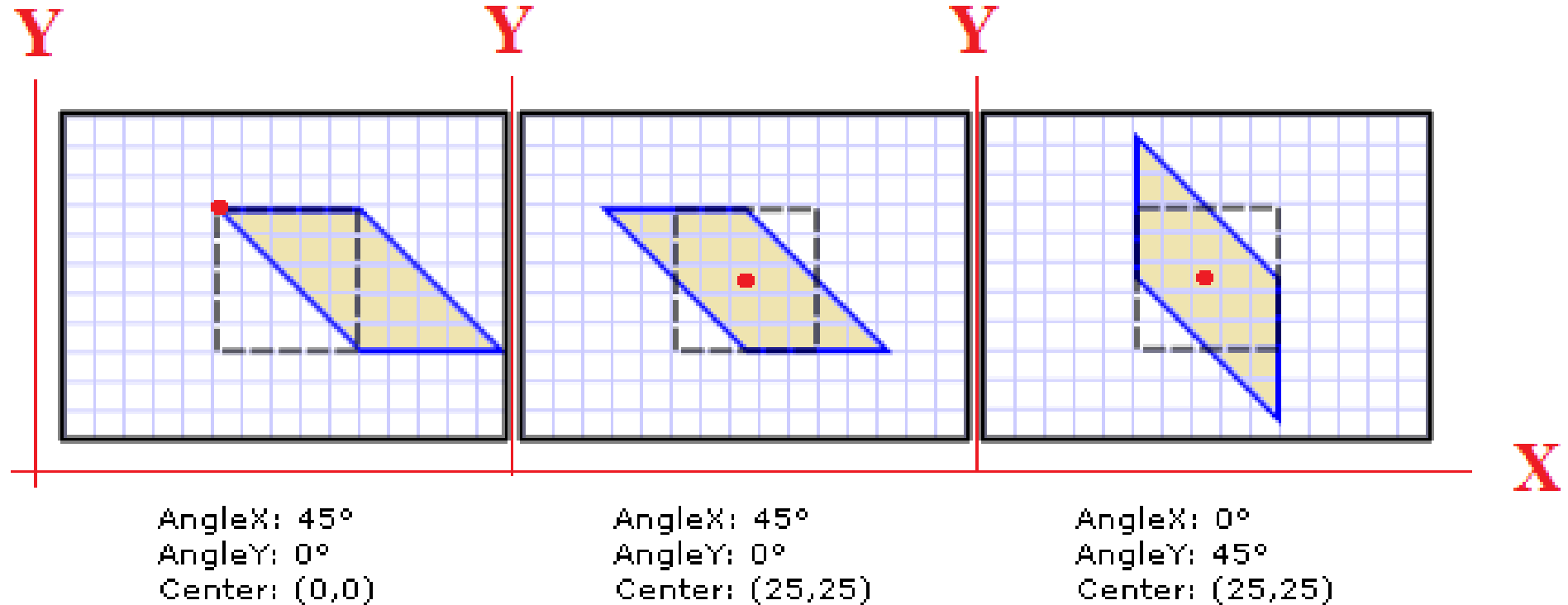
skew degree = - 30



skew degree = - 45

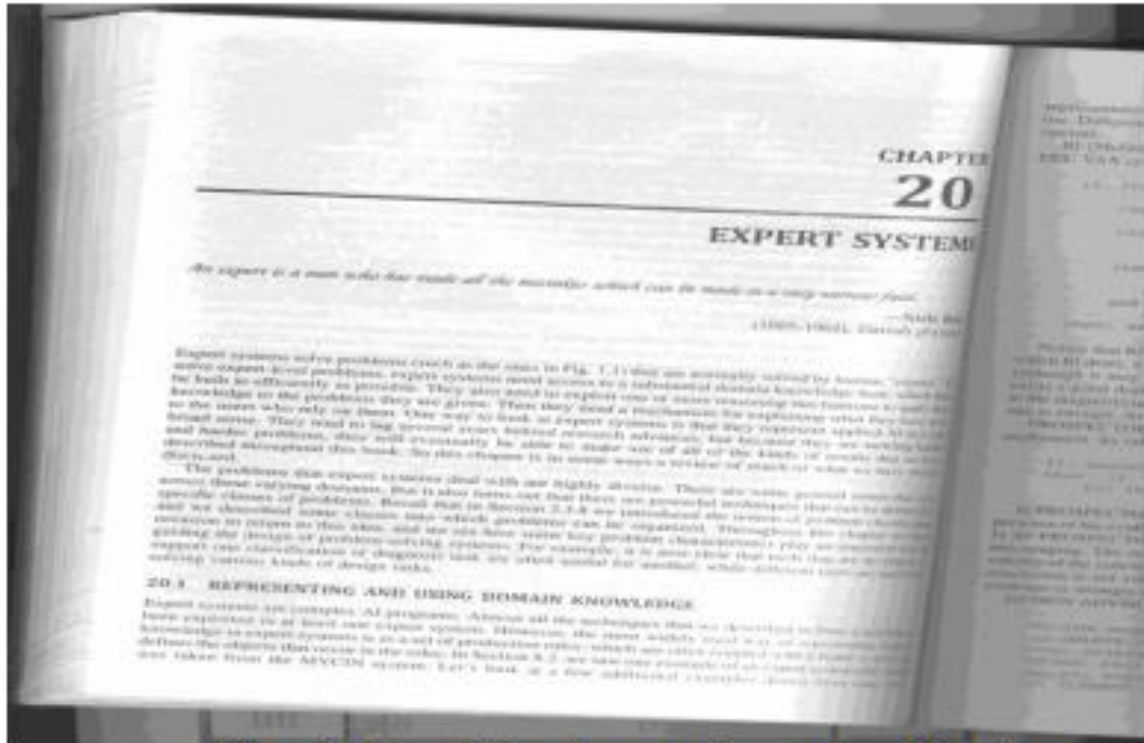


# Skew Image



# Skew Image - an unwanted effect

The image obtained after scanning an opened book page usually suffers from various scanning artifacts. One such major artifact is the Skew defect. This defect reduces the quality of the scanned images and cause many problems to the process of document image analysis. It is difficult to understand such documents by the Optical Character Recognizer (OCR).



Skew defect resulting after scanning an opened book page.



# Skew Image - an unwanted effect

How can we solve this problem?

1) **Hough Transform:** the process includes capturing the images with a camera, detecting the skew angle and applying Skew correction algorithm:



Skewed vehicle number plates (left) and Skew corrected vehicle number plates by using Hough Transform (right)

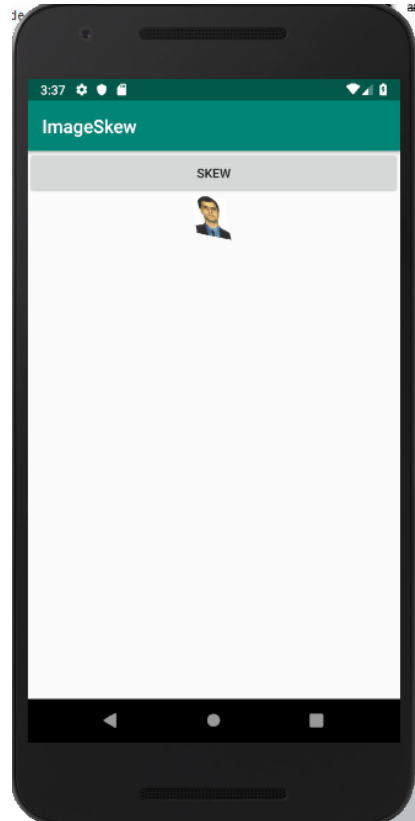
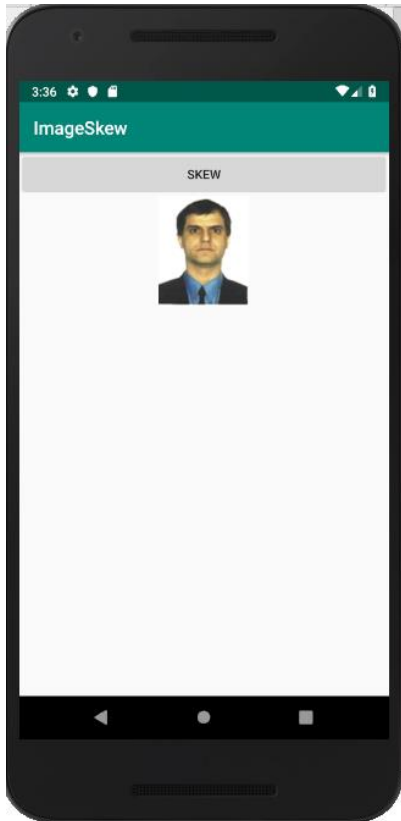
- 2) Method of extreme points
- 3) Radon transform
- 4) Principal component analysis (PCA)

# Skew Image

## In Matrix class:

**setSkew(float skewX, float skewY)** : Skew image, skewX and skewY are the skew ratio in X and Y direction.

**setSkew(float skewX, float skewY, float x, float y)** : Similar with setSkew(float skewX, float skewY), but axis is (x, y).



# Skew Image

Example:

```
setSkew(1f, 0f, dWidth / 2f, dHeight / 2f)
```

This will skew the image across the x-axis (and around the centre point) by 1, which is the width of the image, resulting in a 45 degree tilt of the image.

