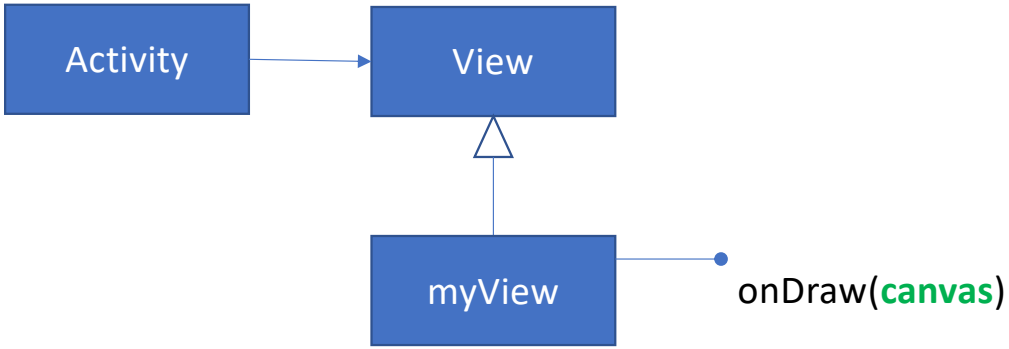
An Introduction to Android 2D graphics

Roberto Beraldi

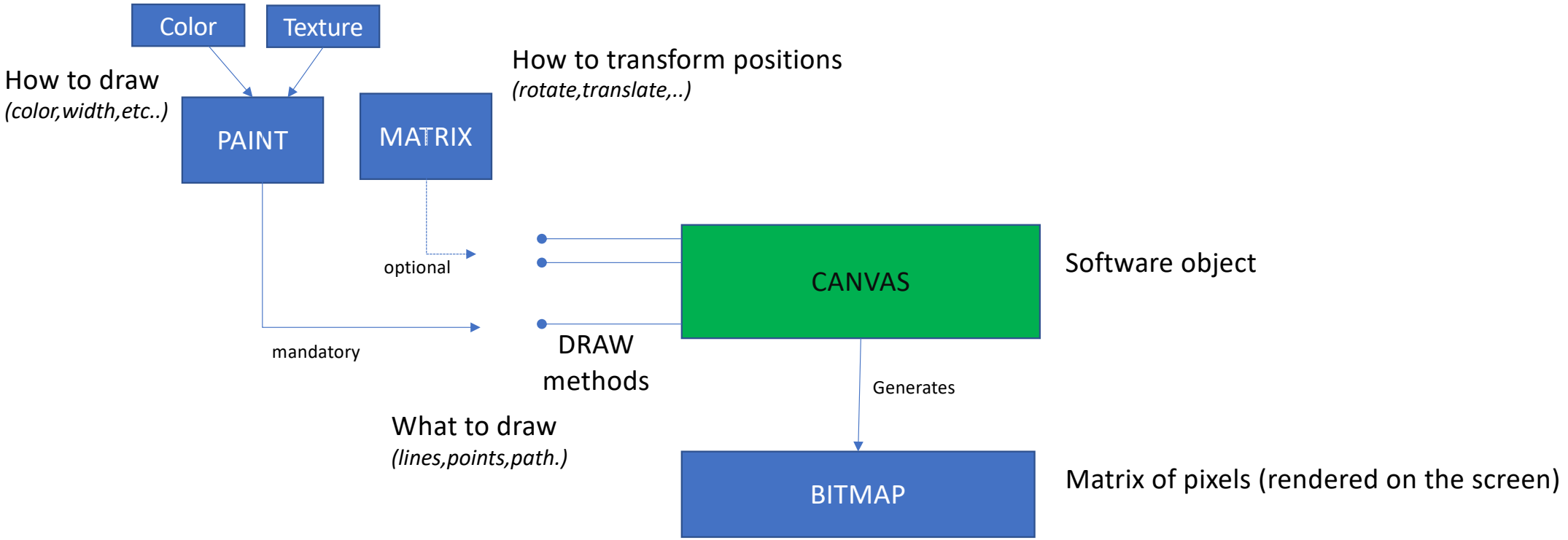
Introduction

- Till now views were defined by assembling predefined graphical elements (e.g., buttons,..)
- We can define the content of of view by specializing the View class



- The visual content is generated by programmatically (**onDraw** method)

2D graphics (android.graphics.*)



Canvas drawing primitives

- Set of functions that allow to draw 2D shapes
  - Elemental
  - Composite

- Draw methods are specified in the Device Coordinate Frame
- Draw methods use pixels as unit of measurement (expressed in float)

Device Coordinate Frame

- Viewport: surface available for drawing (depends on the device and its orientation)

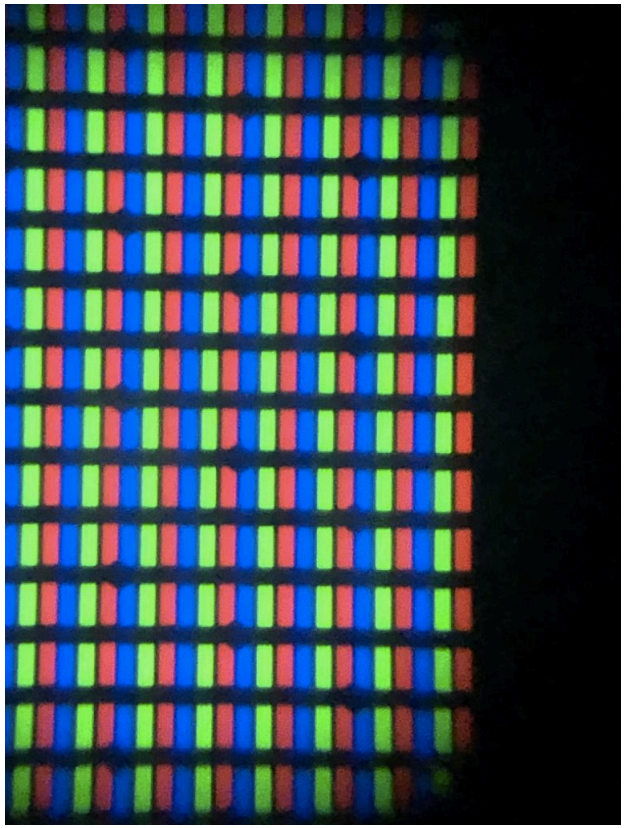
DEVICE	OPERATING SYSTEM	PHYSICAL SIZE "	PHYSICAL SIZE CM	WIDTH PX	HEIGHT PX	DEVICE WIDTH	PX PER INCH
PIXEL XL							
Google Pixel XL	Android	5.5	14.0	1440	2560	1440	534

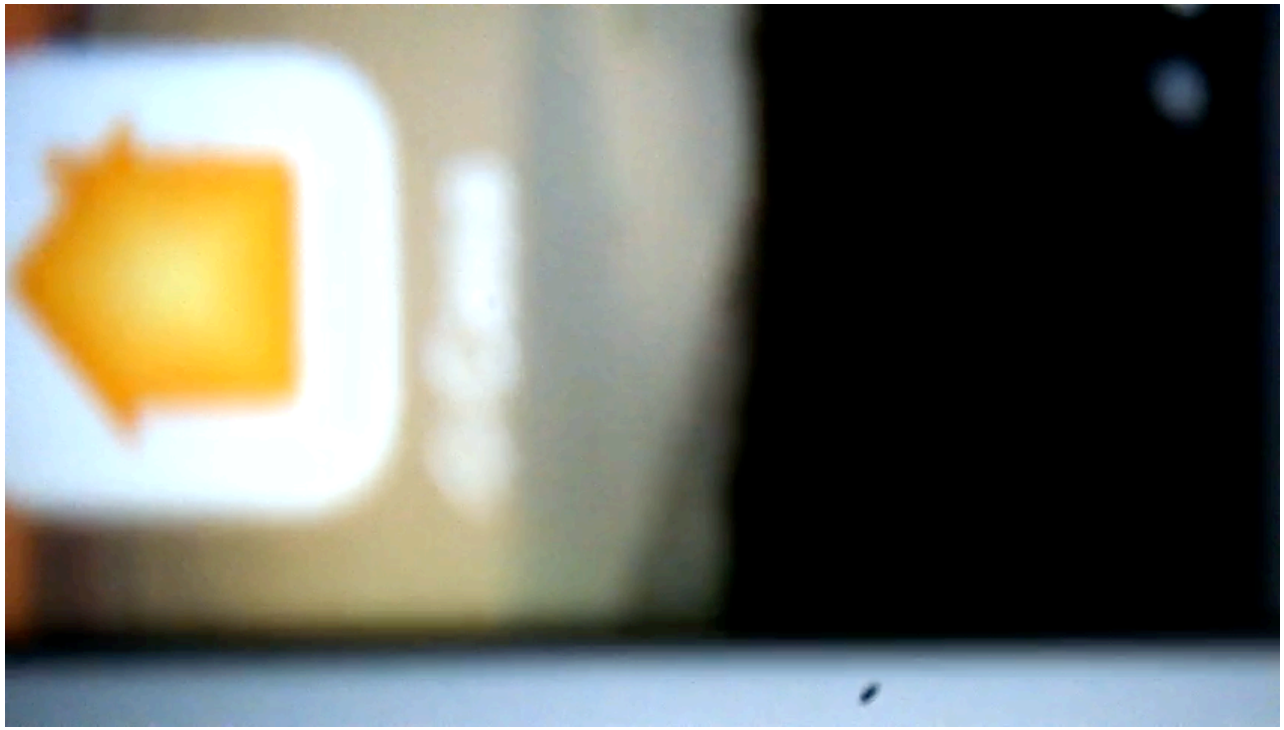
What is the physical dimension of a pixel

- To determine the physical dimension of a pixel, one need to know the density dpi (dots per inch, or pixels per inch)
- 1 px on a 160 dpi screen is roughly a square of edge x….
- 1 inch = 25.4 mm
- 160x = 25.4 → x=0.158 mm = 158μm

- On a 320DPI screen, a pixel is ½ thinner than on a 160DPI

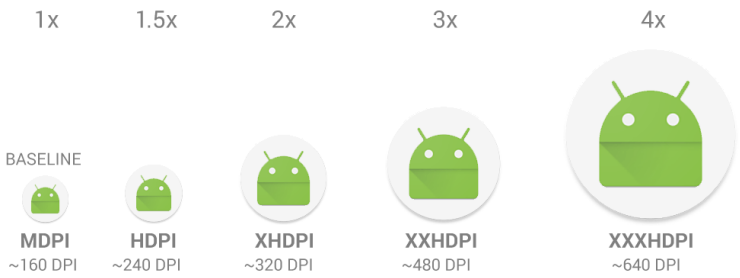
Pixel view 20x





Why DPI matters?

- Visual aspect: the higher the density the shorter the physical size of a same bitmap
- One need to use images with DPI dependent size
- Use dp (density-independent pixel) units



Why DPI matters?

- Imagine an app in which a scroll or fling gesture is recognized after the user's finger has moved by at least 16 pixels.
- On a baseline screen, a user's must move by 16 pixels / 160 dpi, which equals 1/10th of an inch (or 2.5 mm) before the gesture is recognized.
- On a device with a high-density display (240dpi), the user's must move by 16 pixels / 240 dpi, which equals 1/15th of an inch (or 1.7 mm).
- The distance is much shorter and the app thus appears more sensitive to the user.

Example taken from: https://developer.android.com/training/multiscreen/screendensitie

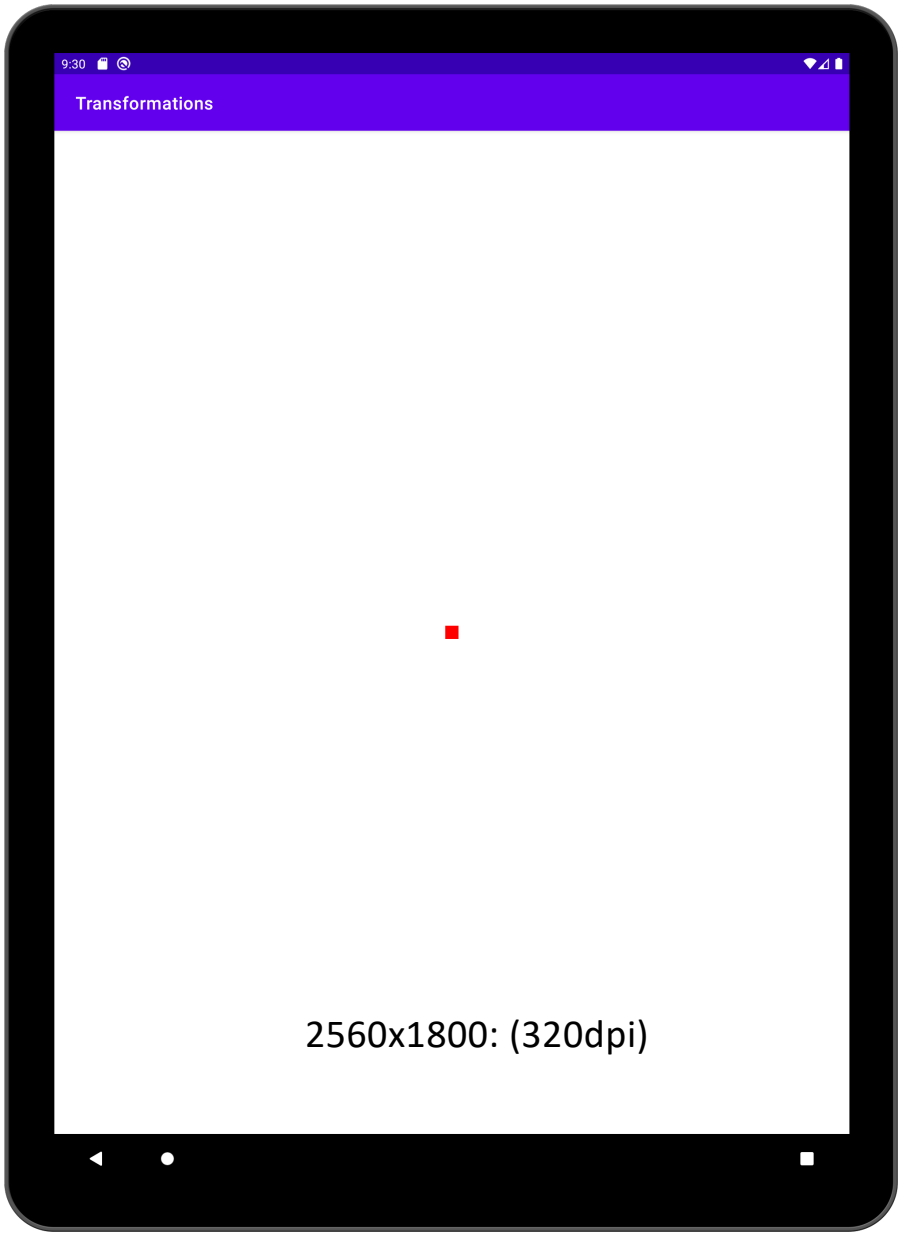
Drawing primitives

- Point (Points)
- Line (Lines)
- Path

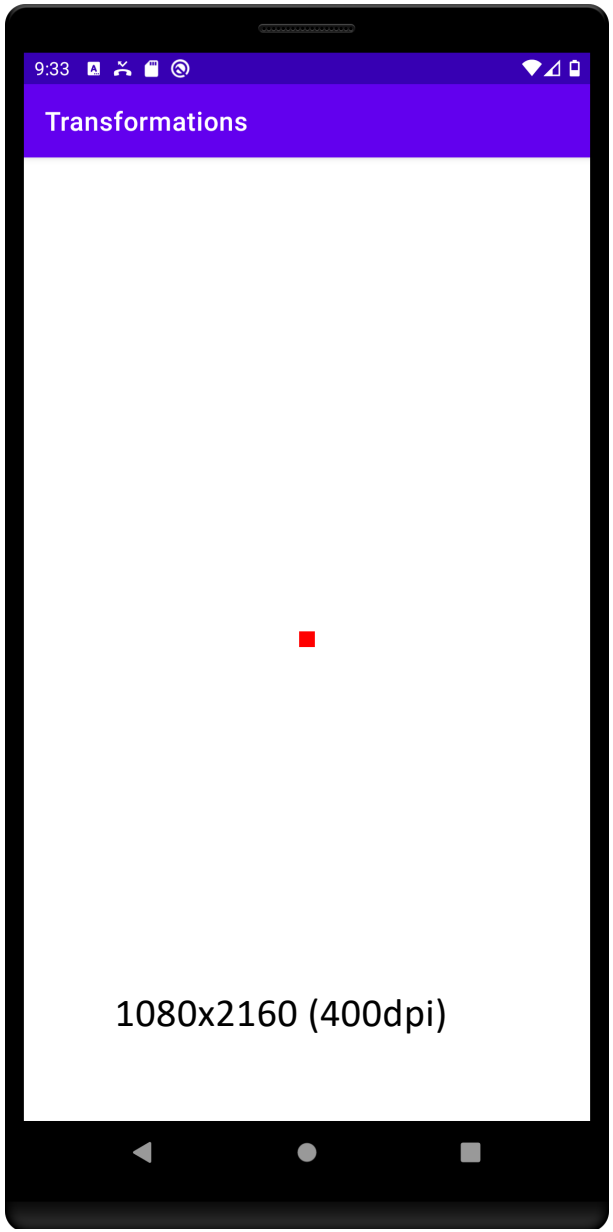
- Triangles
  - Connected triangle allows to approximate (almost) any shape

- Predefined shapes
  - Rect, Circle, Oval, Arc

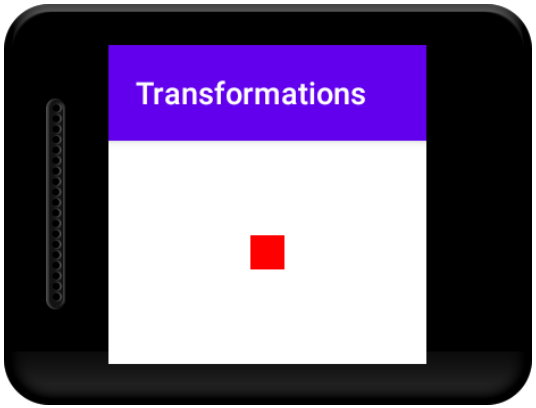
- Text



2560x1800: (320dpi)



1080x2160 (400dpi)

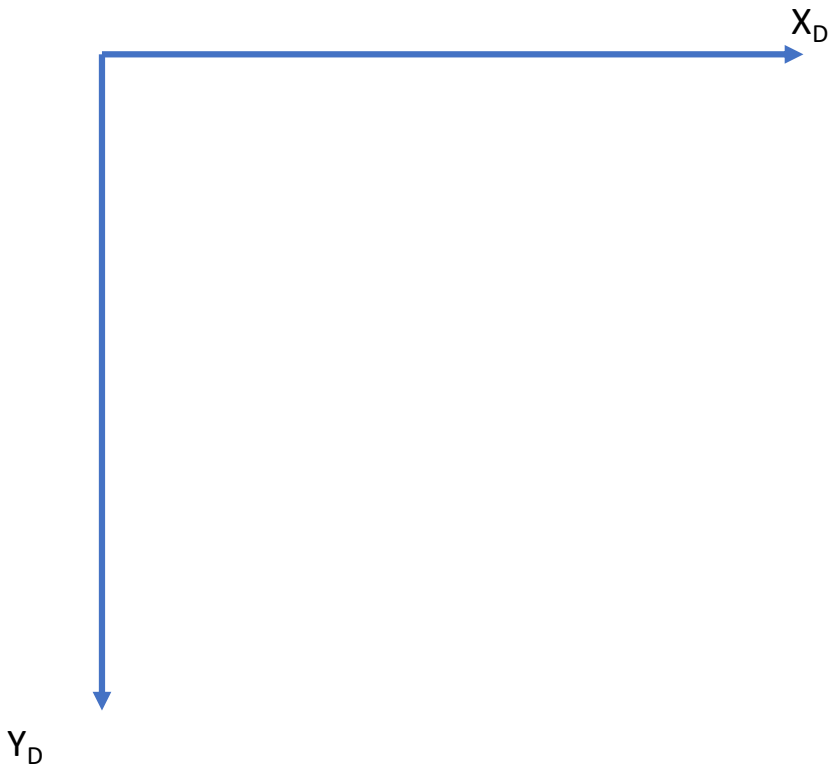


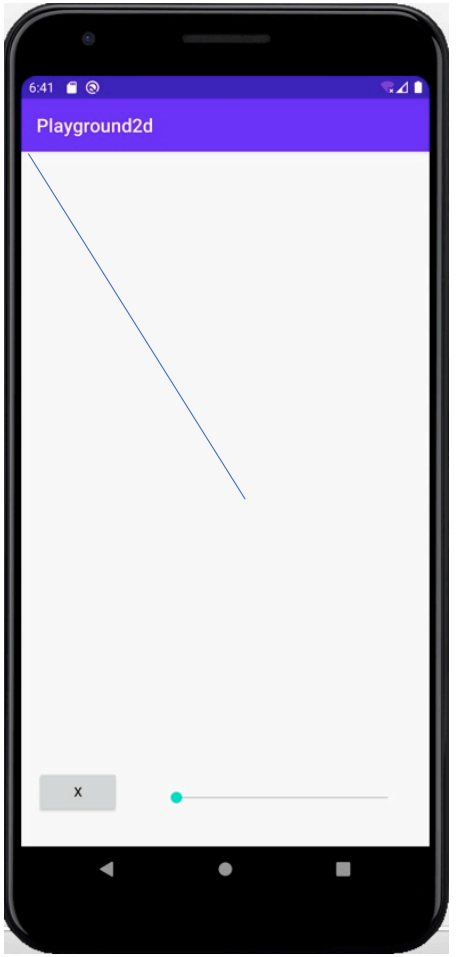
280x280 (240dpi)

```
val mPaint = Paint().apply {
style=Paint.Style.STROKE
color = Color.RED
strokeWidth=30f}
```

```
val cx = canvas.width/2f
val cy = canvas.height/2f
canvas.drawPoint(cx,cy,mPaint)
```

A simple example (draw a line)





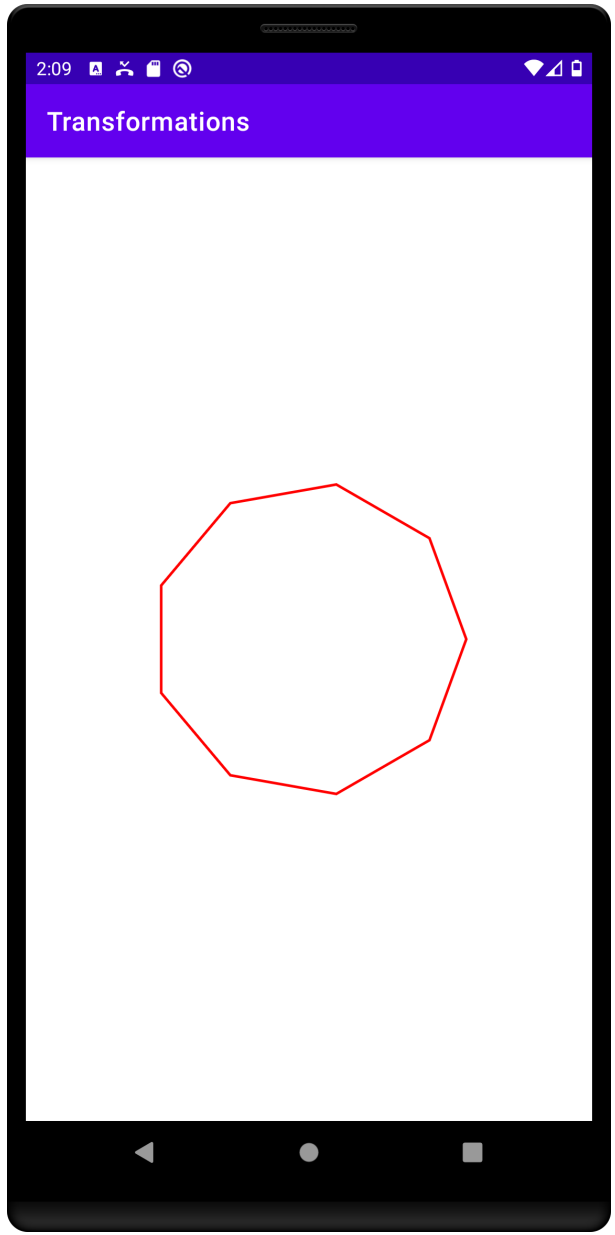

```
canvas.drawLine(0f,0f,canvas.width/2f,canvas.height/2f,mypaint)
```

drawPath

- Allows to define a path:
- Reset
- Starting point → MoveTo
- Move to another point → LineTo
- Close

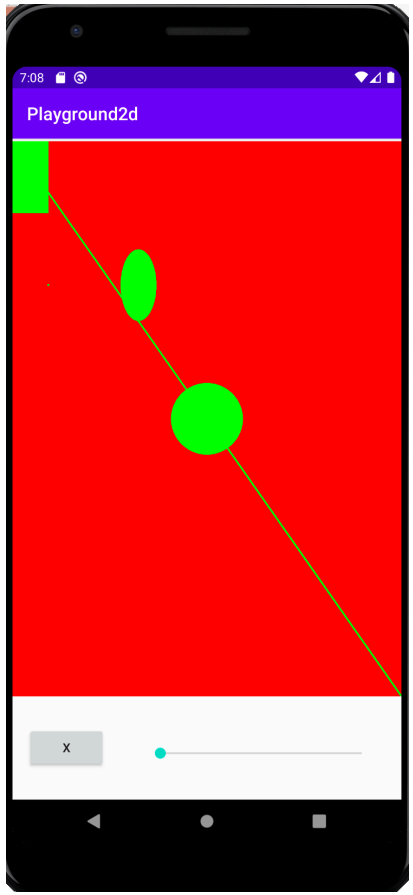
drawPath

```
val sides = 9
val rad = 300
val path = Path()
val angle = 2.0 * Math.PI / sides
path.moveTo(
        cx + (rad * Math.cos(0.0)).toFloat(),
        cy + (rad * Math.sin(0.0)).toFloat())
for (i in 1 until sides) {
    path.lineTo(
            cx + (rad * Math.cos(angle * i)).toFloat(),
            cy + (rad * Math.sin(angle * i)).toFloat())
}
path.close()
```



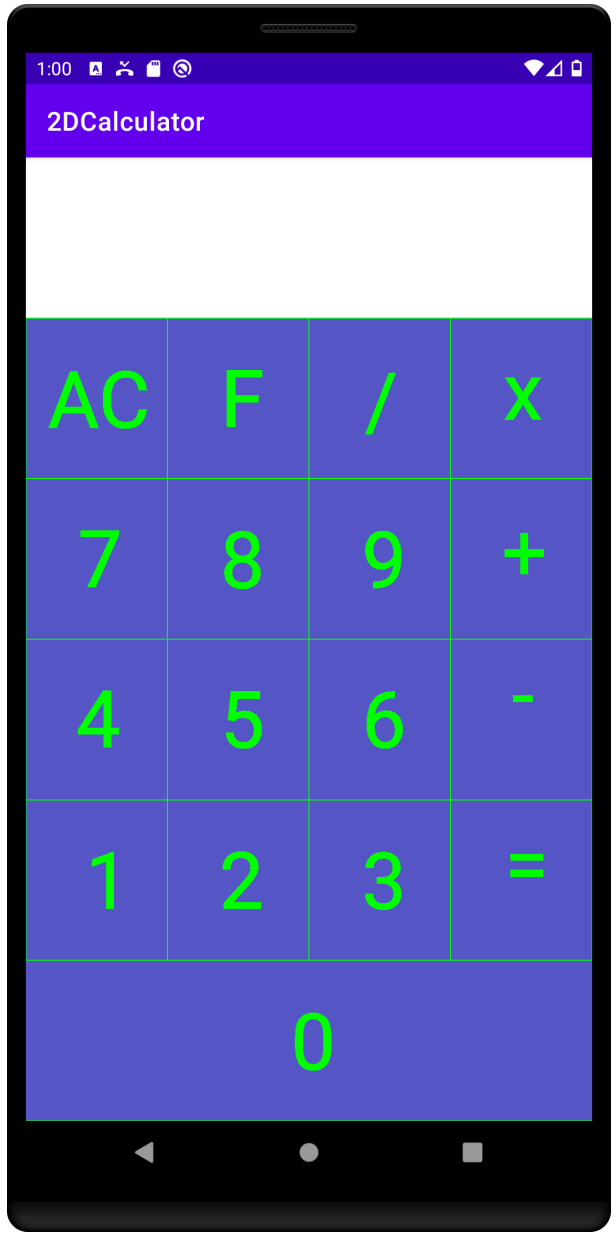
Other drawing examples

```
canvas?.drawCircle(w/2f,h/2f,100f,mypaint)
canvas?.drawOval(300f,300f,400f,500f,mypaint)
canvas?.drawRect(0f,0f,100f,200f,mypaint)
canvas?.drawLine(0f,0f,w.toFloat(),h.toFloat(),mypaint)
```



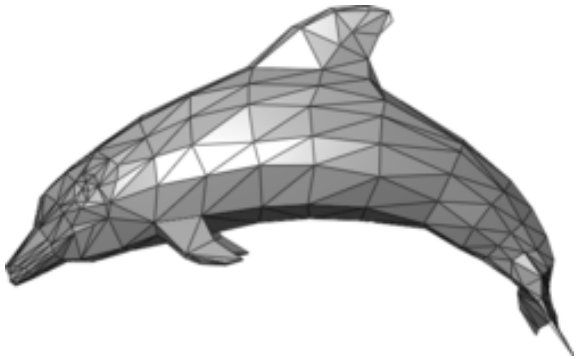
In this example, the viewport is the red area

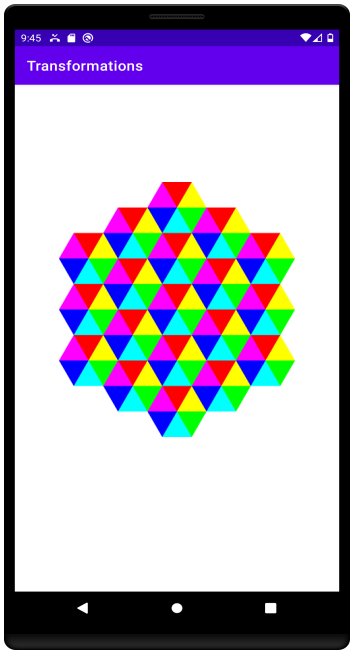
Example: 2Dcalculator

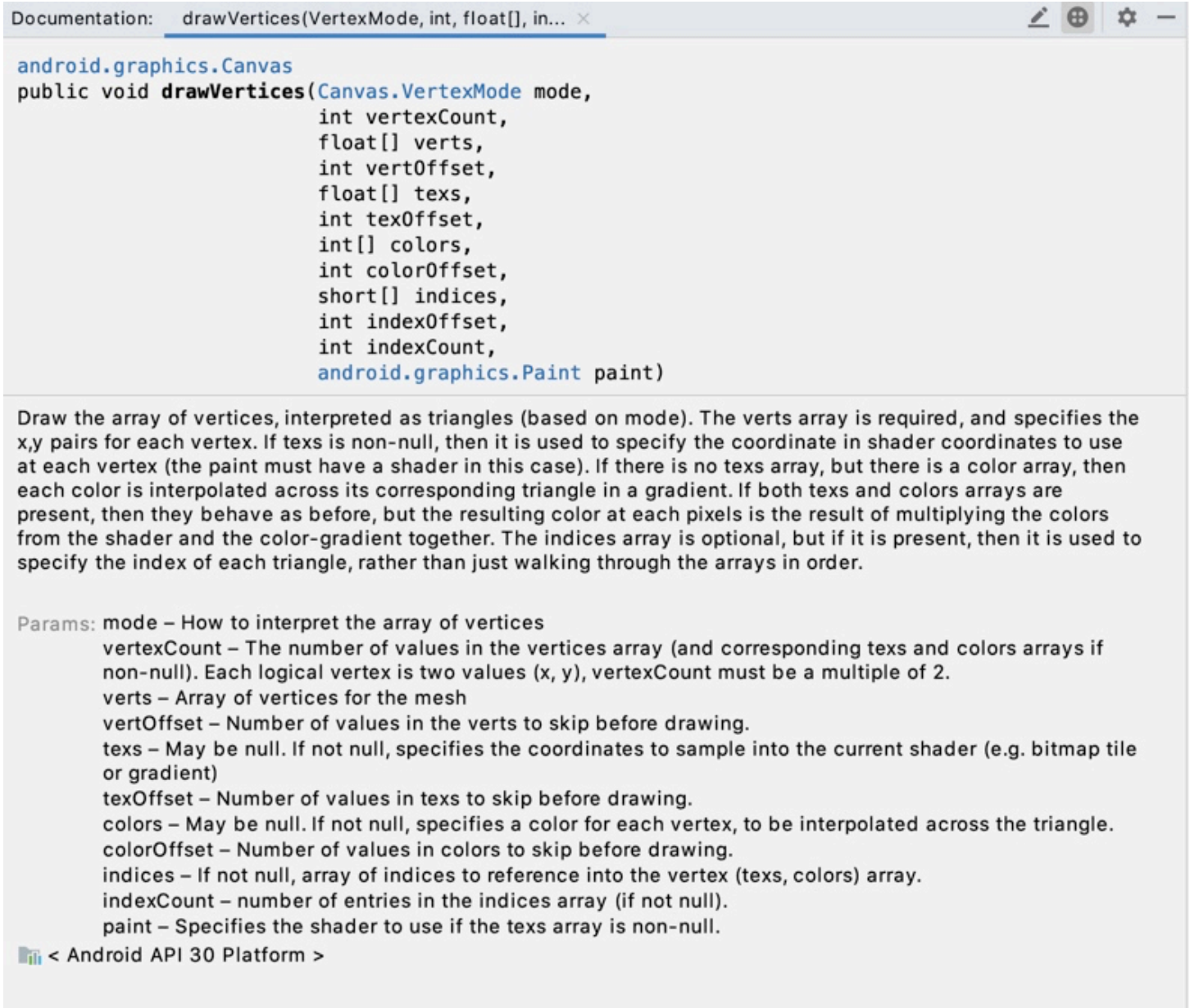


Drawing arbitrary shapes

- Triangle is the basic shape used to create a **mesh** that approximates an arbitrary shape
- Triangles allow efficient rendering, especially needed for real-time apps





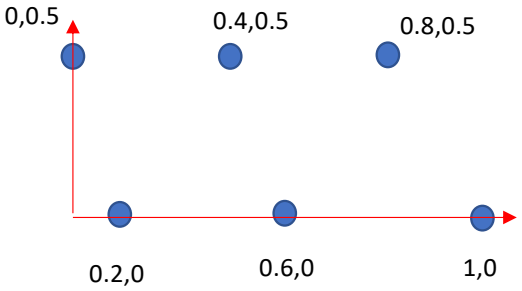

Documentation:
drawVertices(VertexMode, int, float[], in...
android.graphics.Canvas
public void drawVertices(Canvas.VertexMode mode,
int vertexCount,
float[] verts,
int vertOffset,
float[] texs,
int texOffset,
int[] colors,
int colorOffset,
short[] indices,
int indexOffset,
int indexCount,
android.graphics.Paint paint)
Draw the array of vertices, interpreted as triangles (based on mode). The verts array is required, and specifies the x,y pairs for each vertex. If texs is non-null, then it is used to specify the coordinate in shader coordinates to use at each vertex (the paint must have a shader in this case). If there is no texs array, but there is a color array, then each color is interpolated across its corresponding triangle in a gradient. If both texs and colors arrays are present, then they behave as before, but the resulting color at each pixels is the result of multiplying the colors from the shader and the color-gradient together. The indices array is optional, but if it is present, then it is used to specify the index of each triangle, rather than just walking through the arrays in order.
Params: mode – How to interpret the array of vertices
vertexCount – The number of values in the vertices array (and corresponding texs and colors arrays if non-null). Each logical vertex is two values (x, y), vertexCount must be a multiple of 2.
verts – Array of vertices for the mesh
vertOffset – Number of values in the verts to skip before drawing.
texs – May be null. If not null, specifies the coordinates to sample into the current shader (e.g. bitmap tile or gradient)
texOffset – Number of values in texs to skip before drawing.
colors – May be null. If not null, specifies a color for each vertex, to be interpolated across the triangle.
colorOffset – Number of values in colors to skip before drawing.
indices – If not null, array of indices to reference into the vertex (texs, colors) array.
indexCount – number of entries in the indices array (if not null).
paint – Specifies the shader to use if the texs array is non-null.
< Android API 30 Platform >

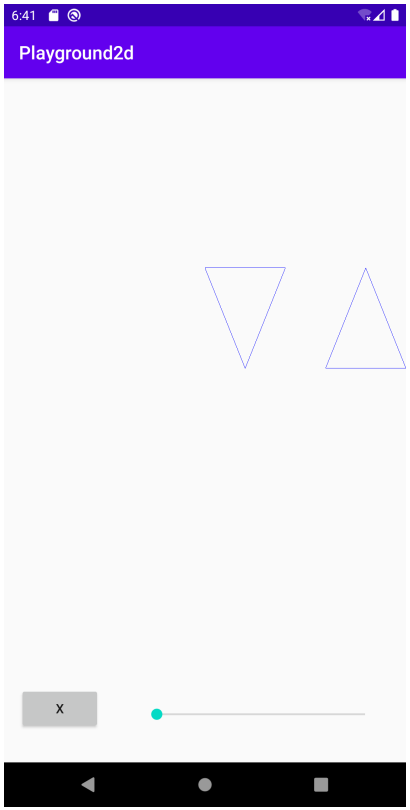
Drawing Mode of Triangles

- A triangle is defined by three vertexes
- Given a **stream** of vertexes there are three different modes to draw triangles from these vertexes:
- TRIANGLE
- STRIP
- FAN

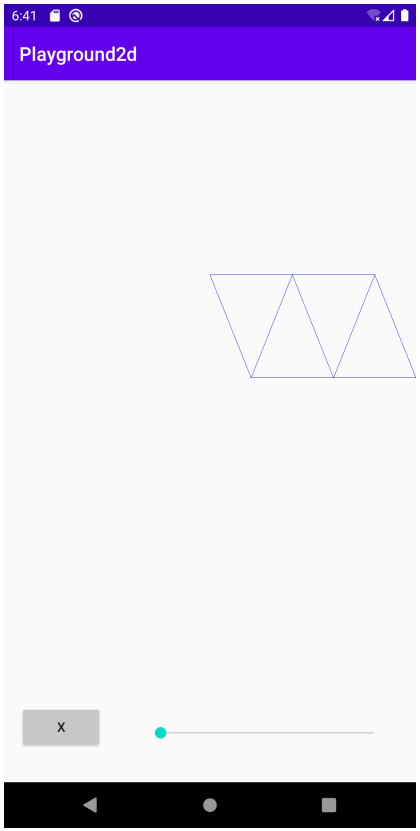
Modes: Triangle, Strip, Fan

- Triangle: Any 3 vertexes define a triangle
- Strip: Set of ‘continuous’ triangle
- Fan: Set of triangles sharing one vertex

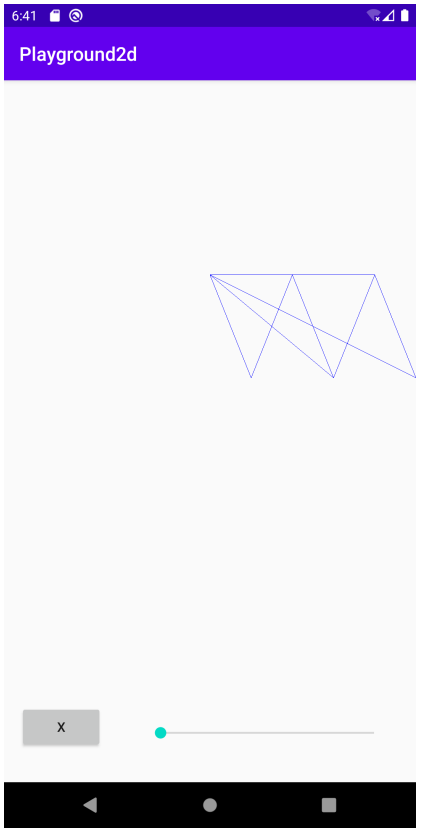




Triangle



Strip



Fan

Triangle fill

```
val vertex = floatArrayOf(
        200f,200f,
        300f,1000f,
        1000f,1000f)
```

Colour along an edge

$C_{02}(\alpha)=\alpha C_0+(1-\alpha)C_2$

Colour of a inner pixel

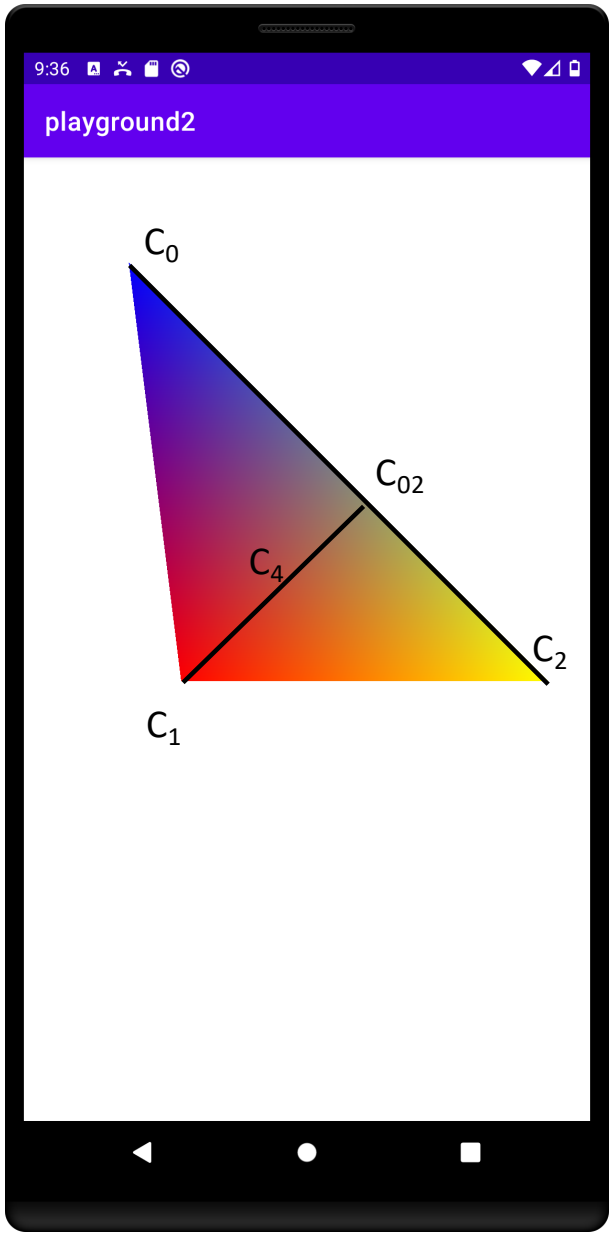
$C_4(\beta)=\beta C_1+(1-\beta)C_{02}$

```
val colors = intArrayOf(
        Color.BLUE,
        Color.RED,
        Color.YELLOW)
```

gradient interpolation

```
canvas.drawVertices(Canvas.VertexMode.TRIANGLES,vertex.size
,vertex,
        0,
        null, 0,
        colors, 0,
        null, 0,0,
        mPaint)
```

No texture

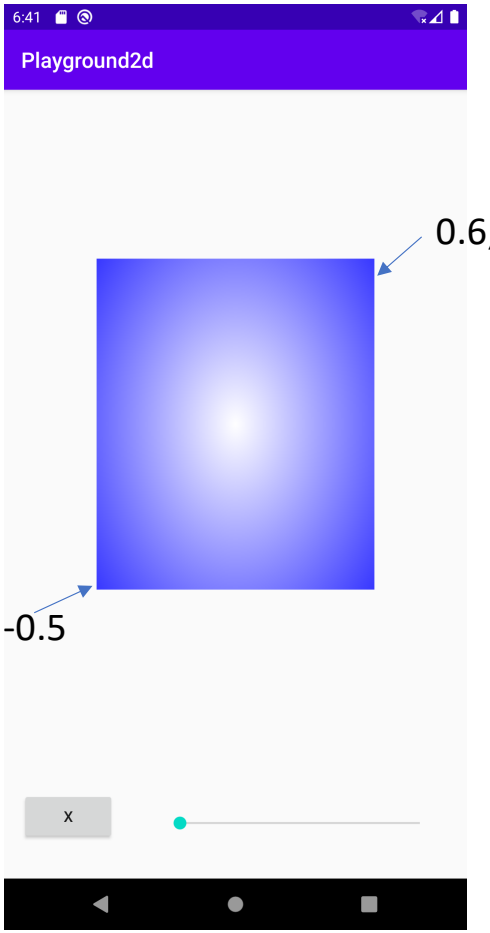


Shader

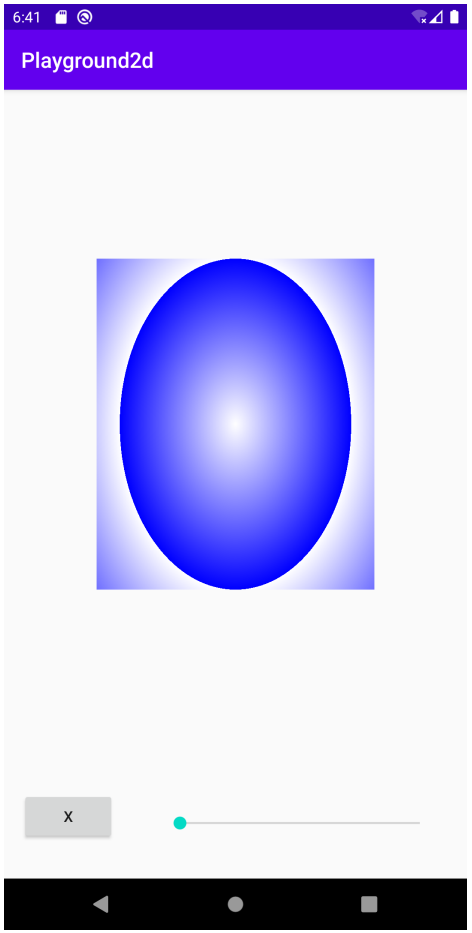
- Define the colours used to fill a shape
- More flexible than interpolation
- Define how colours change from one to another
  - Radial, Gradient
- Use a Local Transformation Matrix to modify the fill pattern, before applying shading
  - (see later)

private var S = *floatArrayOf*(-0.6f,-0.5f, 0.6f,0.5f)

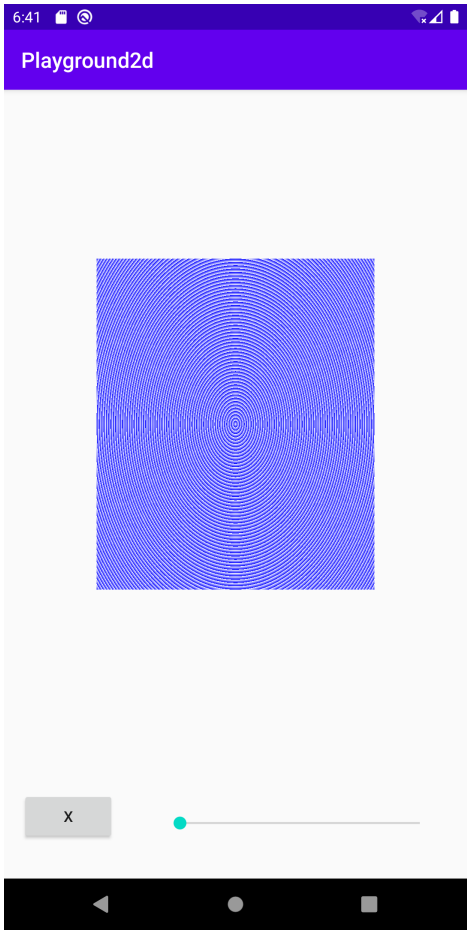
Example: Radial Gradient from white to blue



centre=0,radius=1



centre=0,radius=0.5

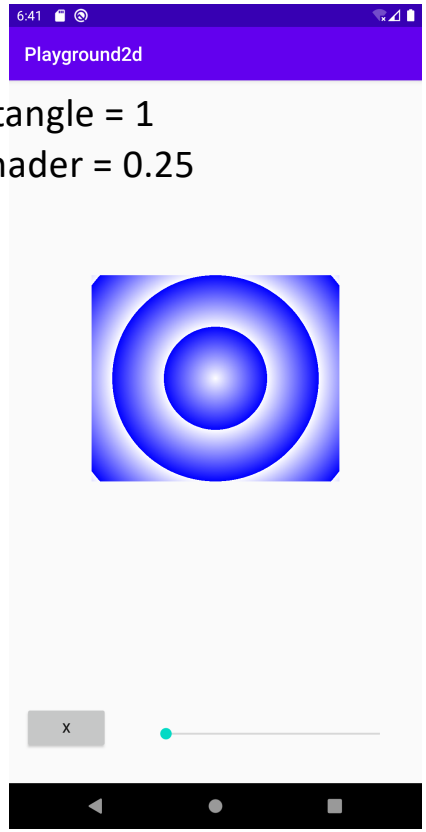


centre=0,radius=0.01

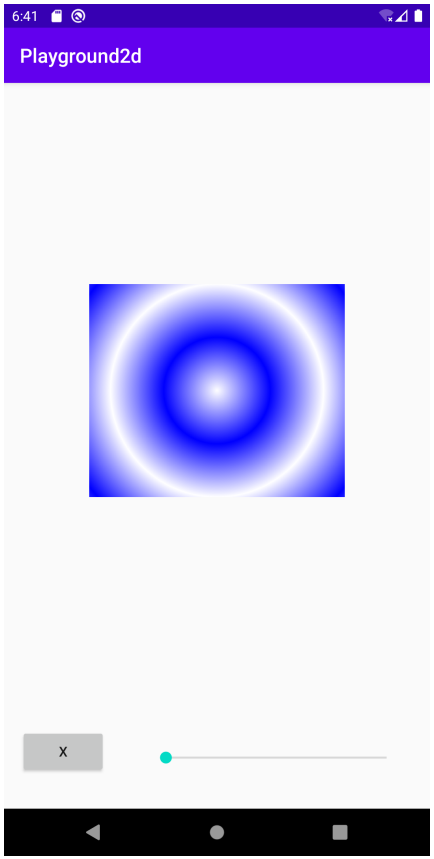
RadialGradient(0f,0f,0.25f,Color.*WHITE*,Color.*BLUE*,Shader.TileMode.**MIRROR**)

Example: Radial Gradient from white to blue

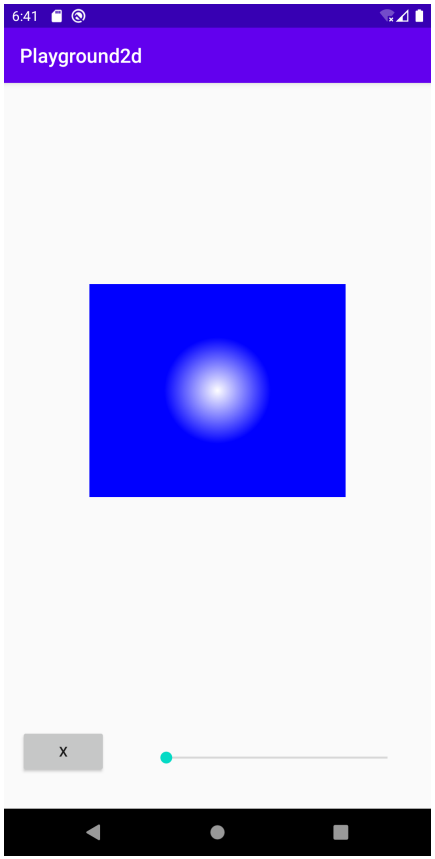
Edge rectangle = 1
Radius shader = 0.25



Repeat



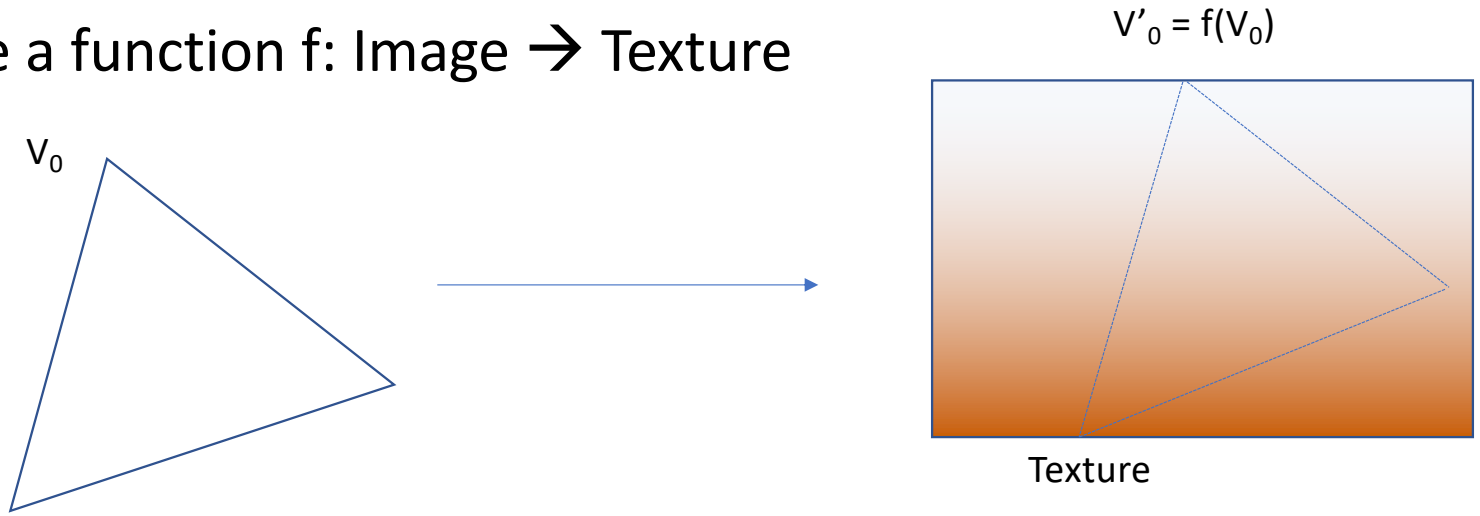
Mirror

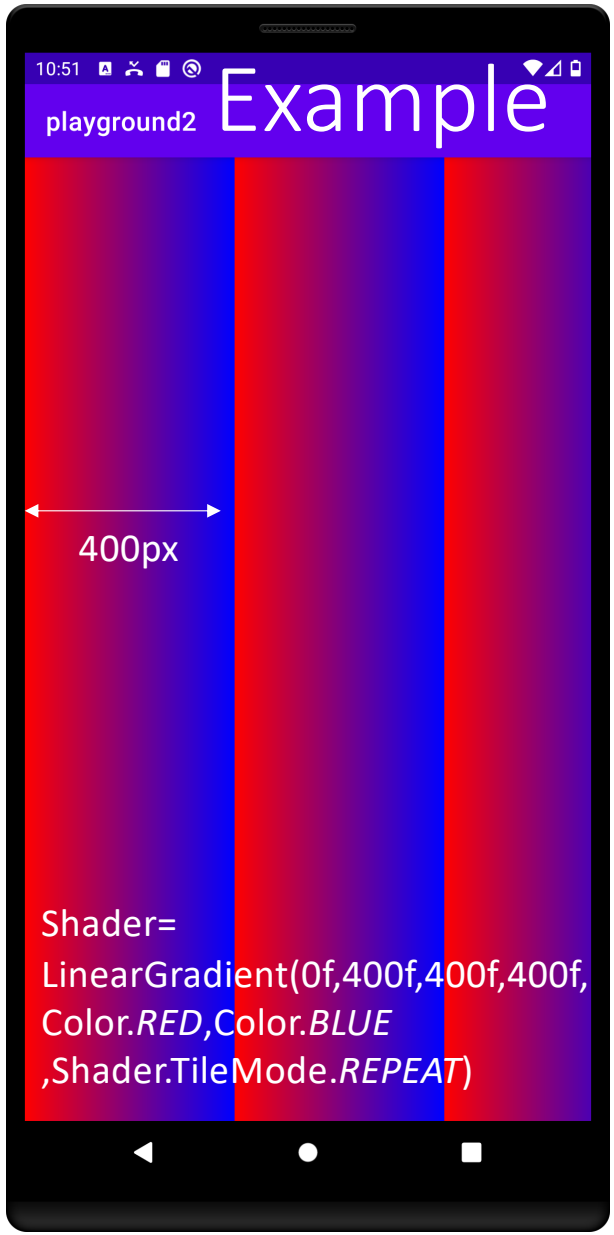


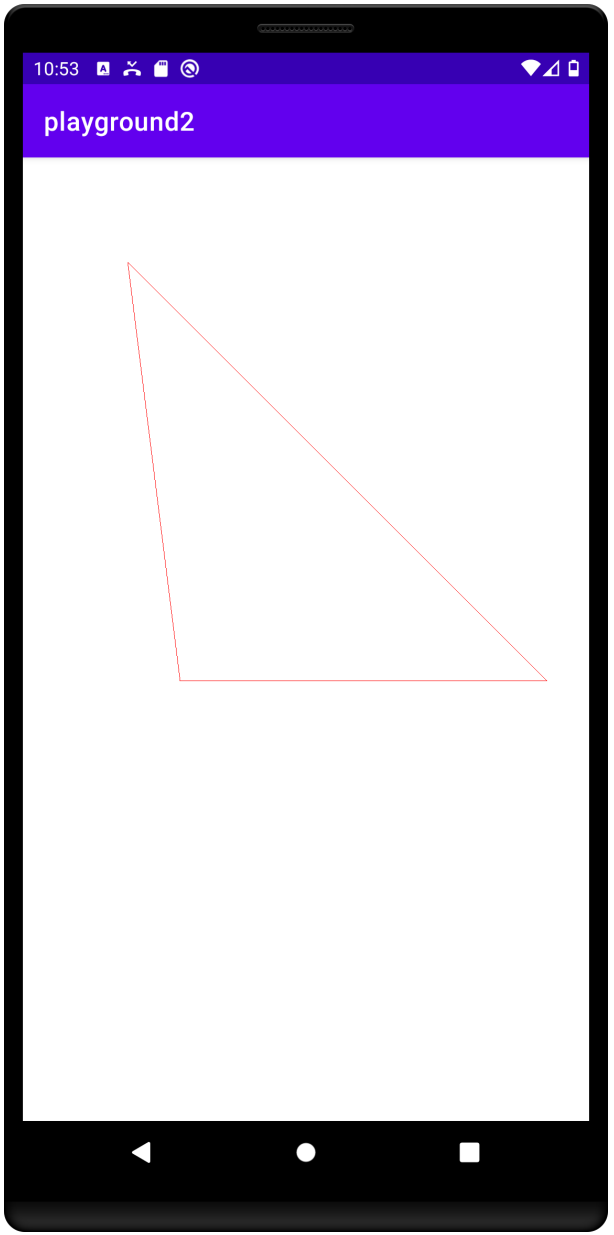
Clamp

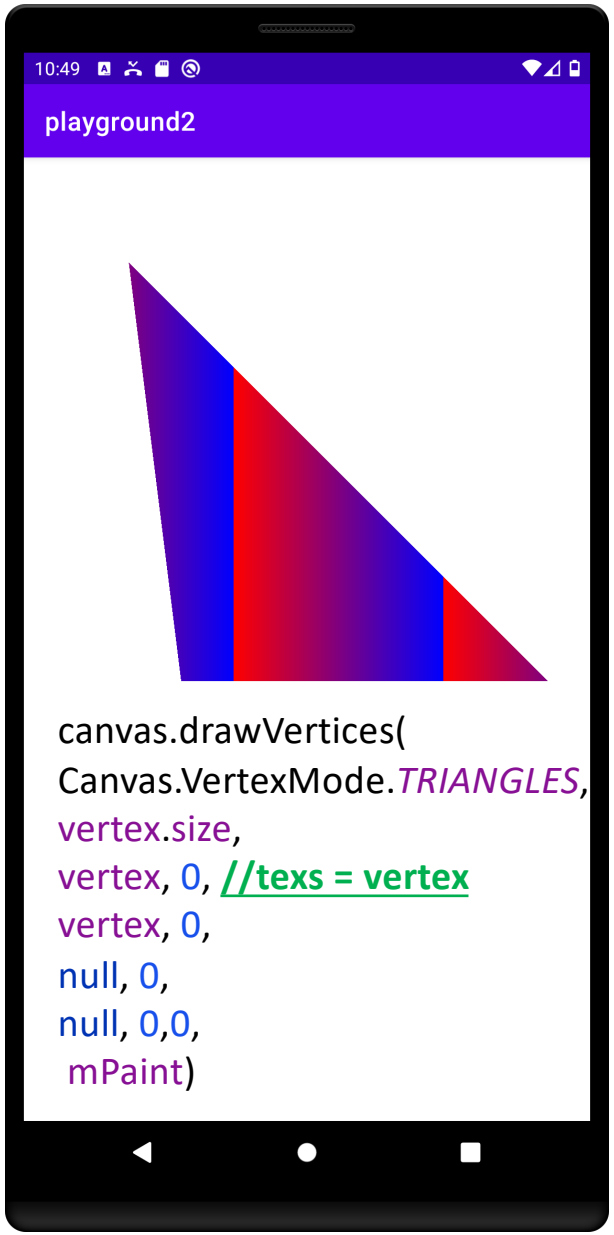
Texture mapping

- Texture allows to determine the colour of a pixel starting from an arbitrary texture image
- How to determine the coordinate of the texture tile associated to a pixel?
- Use a function f: Image → Texture




10:51
playground2
Example
400px
Shader=
LinearGradient(0f,400f,400f,400f,
Color.RED,Color.BLUE
,Shader.TileMode.REPEAT)


10:53
playground2


10:49
playground2
canvas.drawVertices(
Canvas.VertexMode.TRIANGLES,
vertex.size,
vertex, 0, //texs = vertex
vertex, 0,
null, 0,
null, 0,0,
mPaint)

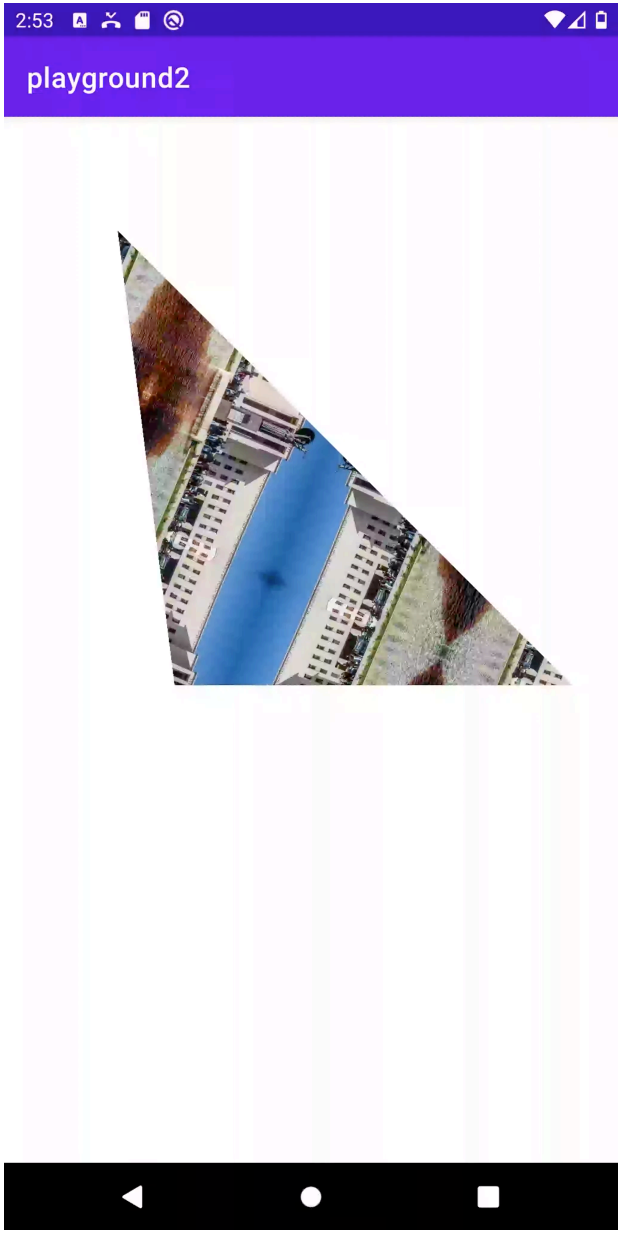
Bitmap texture

- Bitmap image as textures mimics a surface, a material, a pattern or even a picture
- Provide more sense of reality

Example

```
var sapienza =
BitmapFactory.decodeStream(getContext().assets.open("sapienza.jpg"))
```

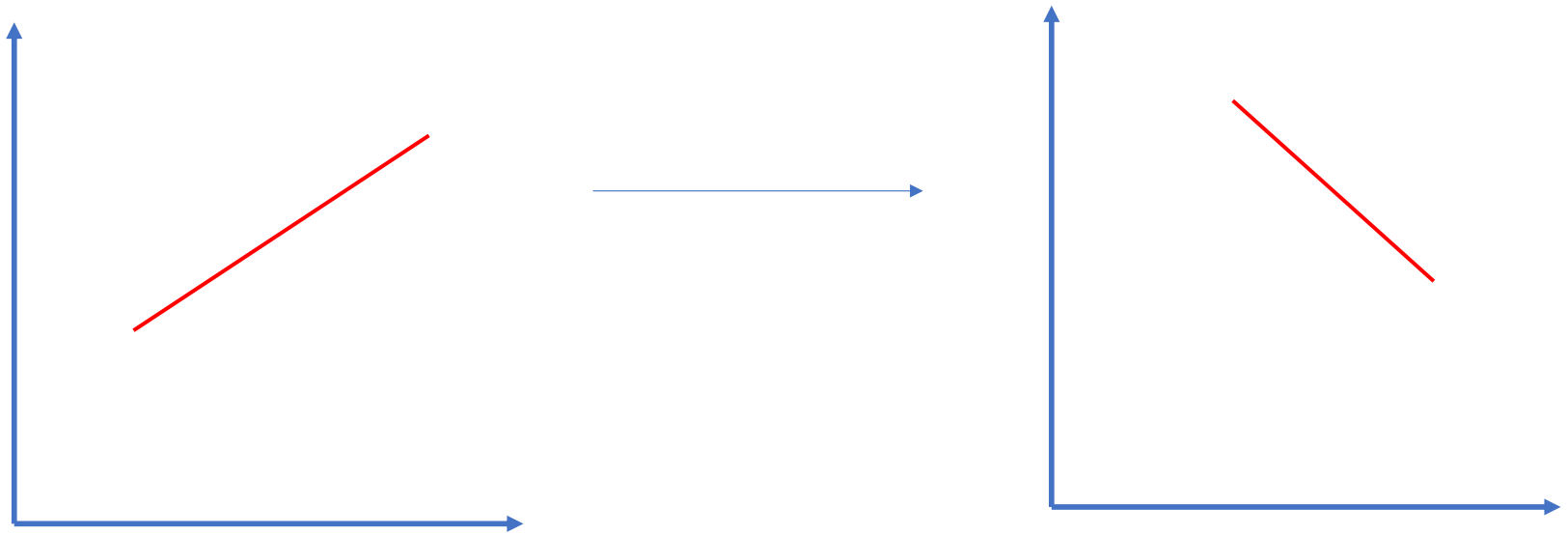
```
BitmapShader(sapienza,Shader.TileMode.REPEAT,Shader.TileMode.REPEAT)
```



Example

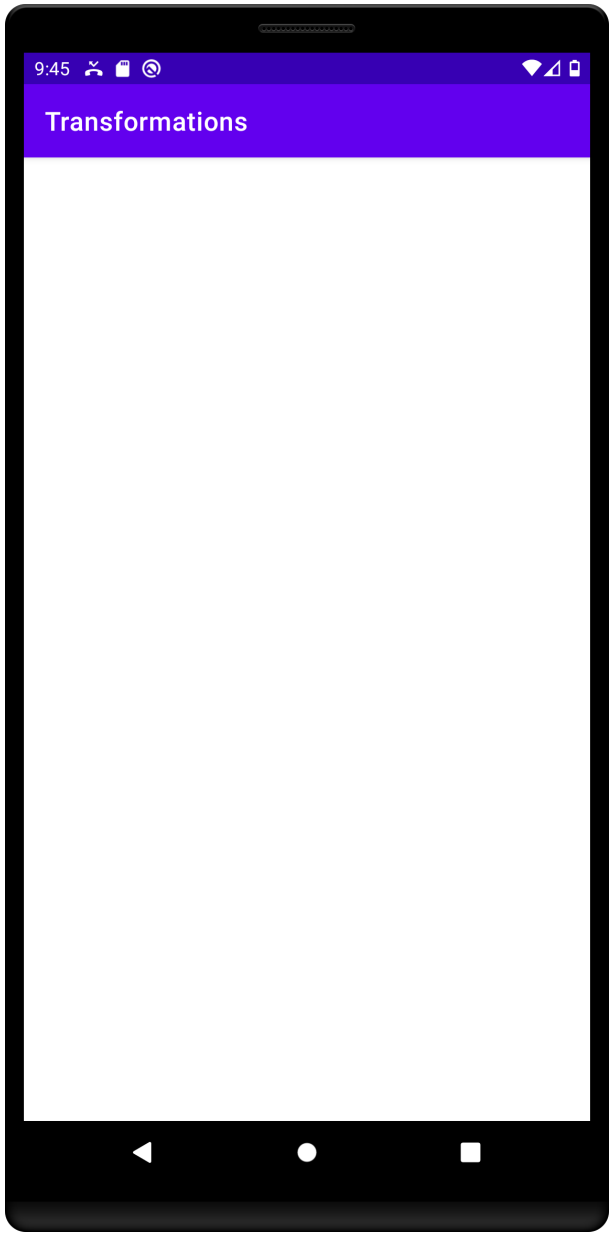
QUESTIONS?

Linear transformations



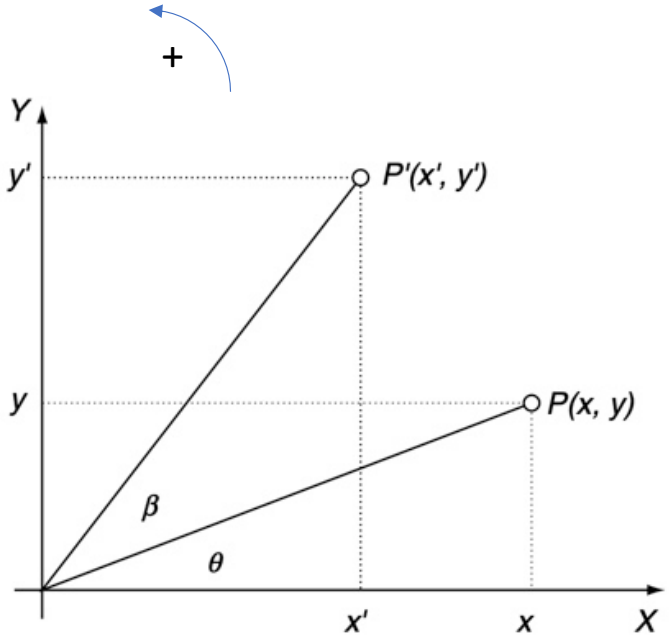
A linear transformation maps lines to lines
A linear transformation is expressed by a matrix

How to rotate a pixel?



2D Rotations

$x' = R\cos(\theta + \beta)$
$y' = R\sin(\theta + \beta)$



$x' = R(\cos(\theta)\cos(\beta) - \sin(\theta)\sin(\beta))$
$y' = R(\sin(\theta)\cos(\beta) + \cos(\theta)\sin(\beta))$

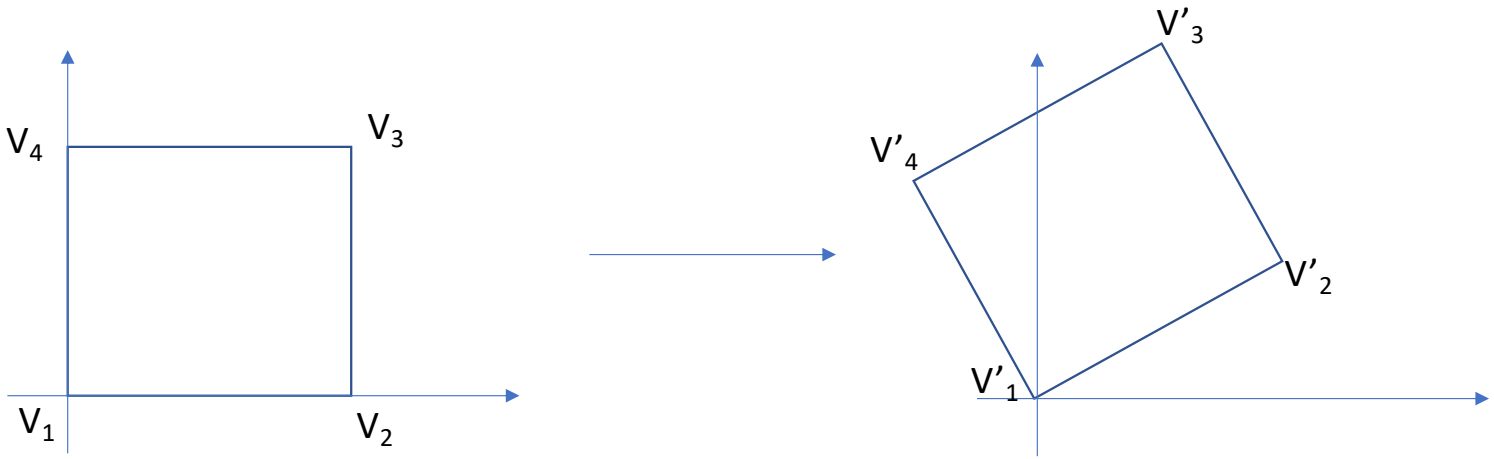
$x' = R\,(x/R\cos(\beta) - y/R\sin(\beta))$
$y' = R\,(y/R\cos(\beta) + x/R\sin(\beta))$

$x' = x\cos(\beta) - y\sin(\beta)$
$y' = x\sin(\beta) + y\cos(\beta)$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$P = \begin{bmatrix} x \\ y \end{bmatrix}$ $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ $T = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix}$ $P' = T\,P$

How to ‘rotate a square’?

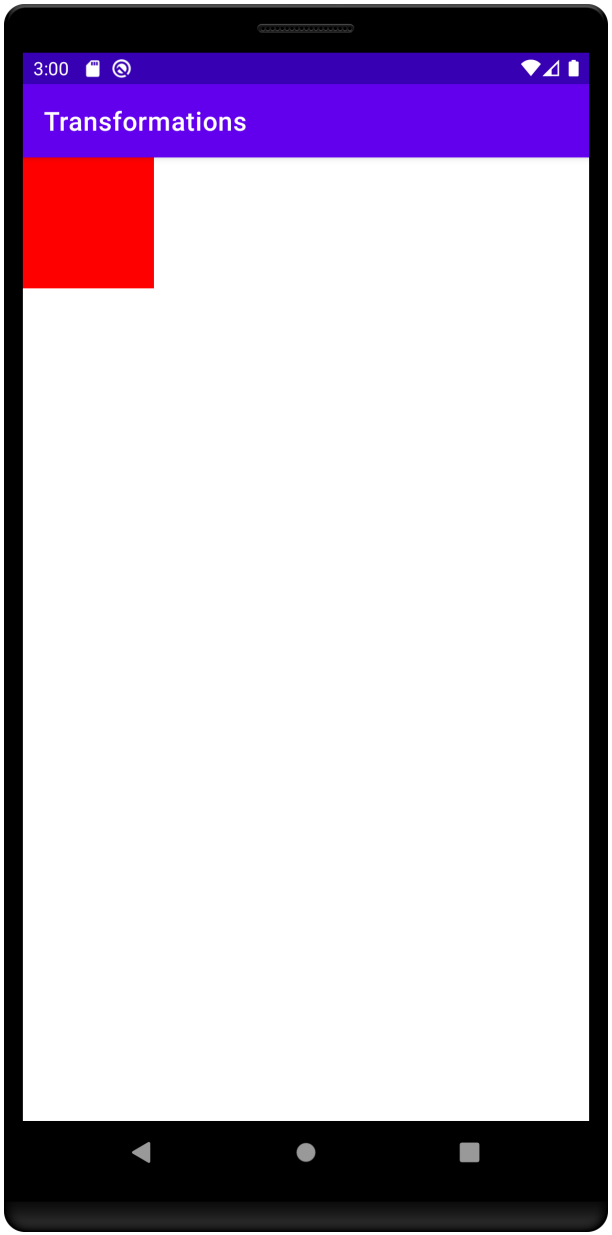


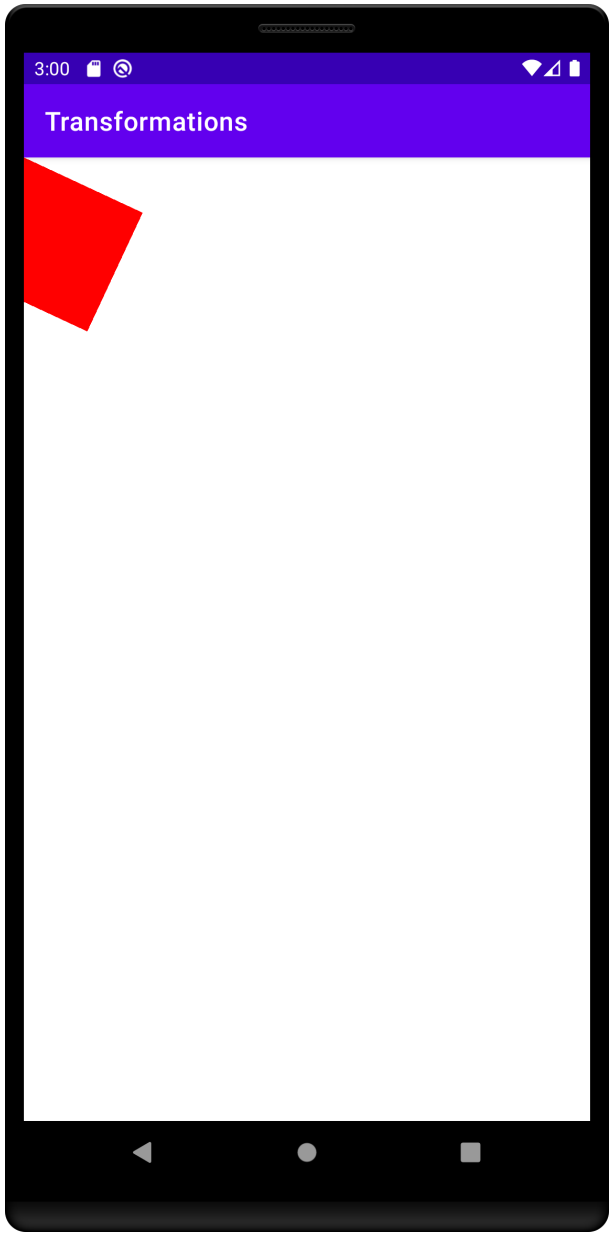
- Rotate the 4 vertexes then draw the lines among them
- $V_i' = T \times V_i$
- This is true for any polygon (since a line is mapped to a line)

Properties of Rotation

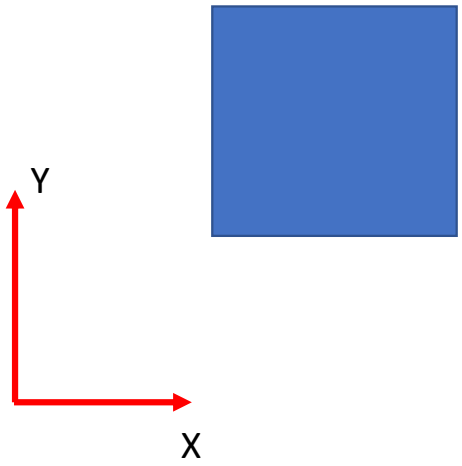
- Because T is linear, a segment line is mapped to another segment line
- T preserves the distance among points (it is in fact called a rigid body transformation)
  - Exercise: Proof that $x'^2+y'^2= x^2+y^2$
- T Preserves parallelism among lines

Example

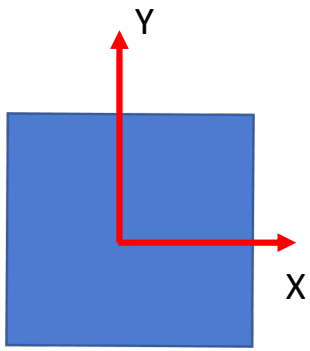




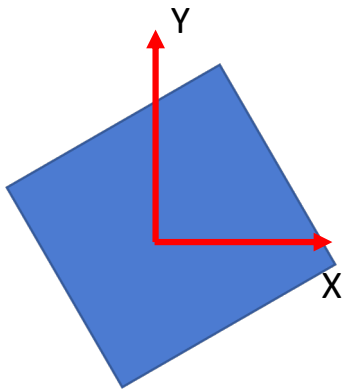
Rotation around a point (x_c,y_c)



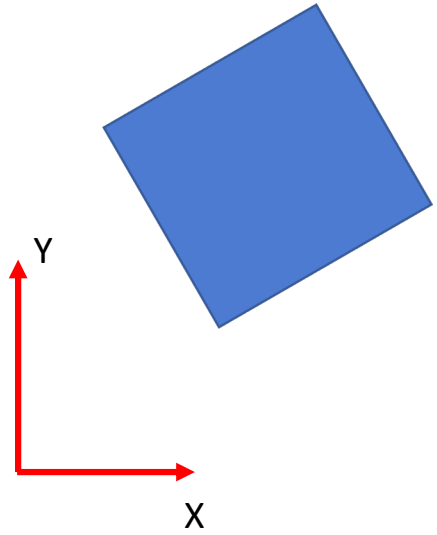
Initial position



Move to the origin
so that x_c,y_c → (0,0)



Rotate around (0,0)

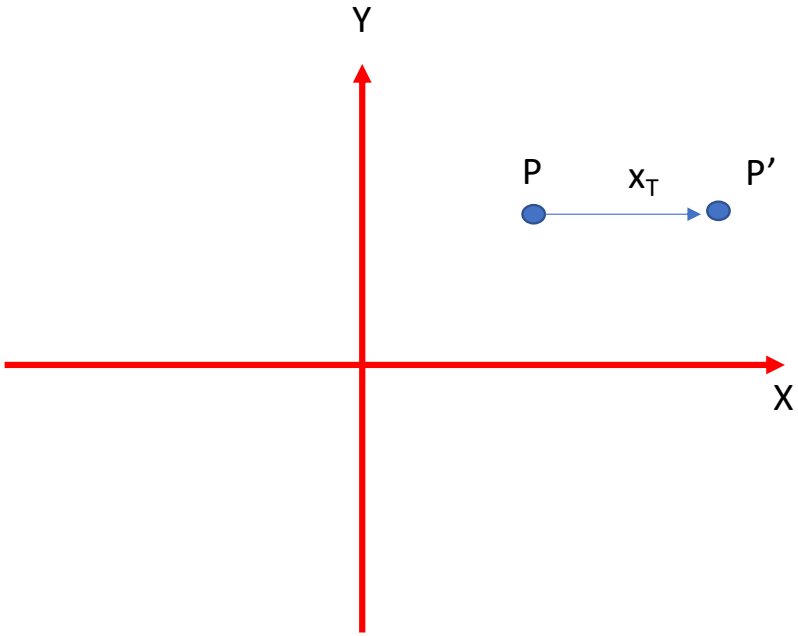


Move back (0,0) →x_c,y_c

Translation along X

- If we are restricted ourself to use a 2x2 matrix a translation cannot be expressed by a Matrix
- In fact, no coefficients can provide the correct result…

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$x' = x\ a_{11} + y\ a_{12}$$
$$y' = x\ a_{21} + y\ a_{22}$$

?

$$x' = x + x_T$$
$$y' = y$$

Augmenting the matrix

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$x' = x\ a_{11} + y\ a_{12} + w\ a_{13}$$
$$y' = x\ a_{21} + y\ a_{22} + w\ a_{23}$$

$$x' = x + x_T$$
$$y' = y$$

a_{11} = 1, a_{12} = 0, a_{13} = x_T and w=1

a_{21} = 0, a_{22} = 1, a_{23} = 0

a_{31}, a_{32}, a_{33} ?

A desirable property: two consecutive translations correspond to matrix multiplication

This is achieved by setting: a_{31}=0 $a_{32} = 0$ $a_{33} = 1$

Translation along X

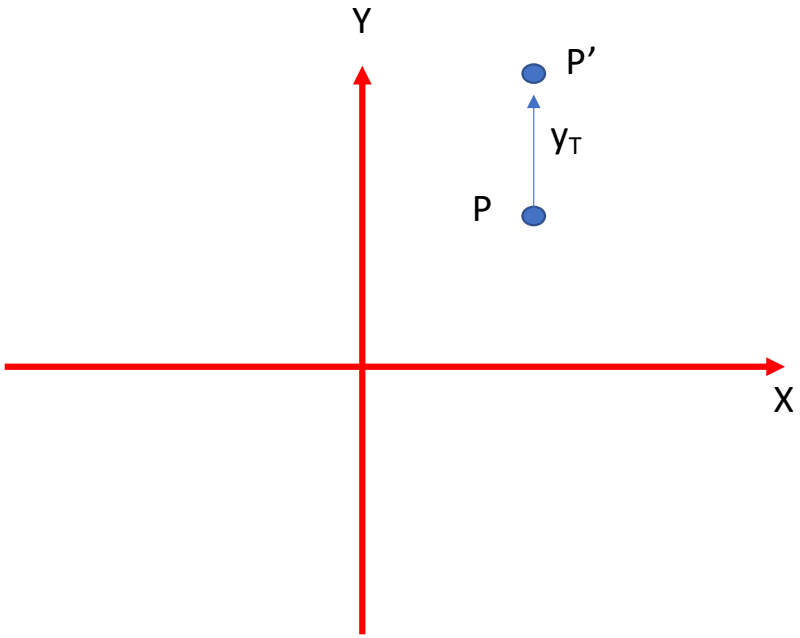
$$\text{Translation} = \begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x'_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_T + x'_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation along Y

$$T_Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix}$$

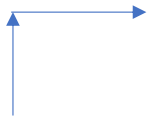
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x \\ y + y_T \\ w \end{bmatrix}$$



Matrix concatenation (aka multiplication)

- Translations alone are commutative: $T_X\ T_Y = T_Y\ T_X$
- The final position of a point subjects to a set of translations is the same for any order they have been applied

$$\begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix}$$



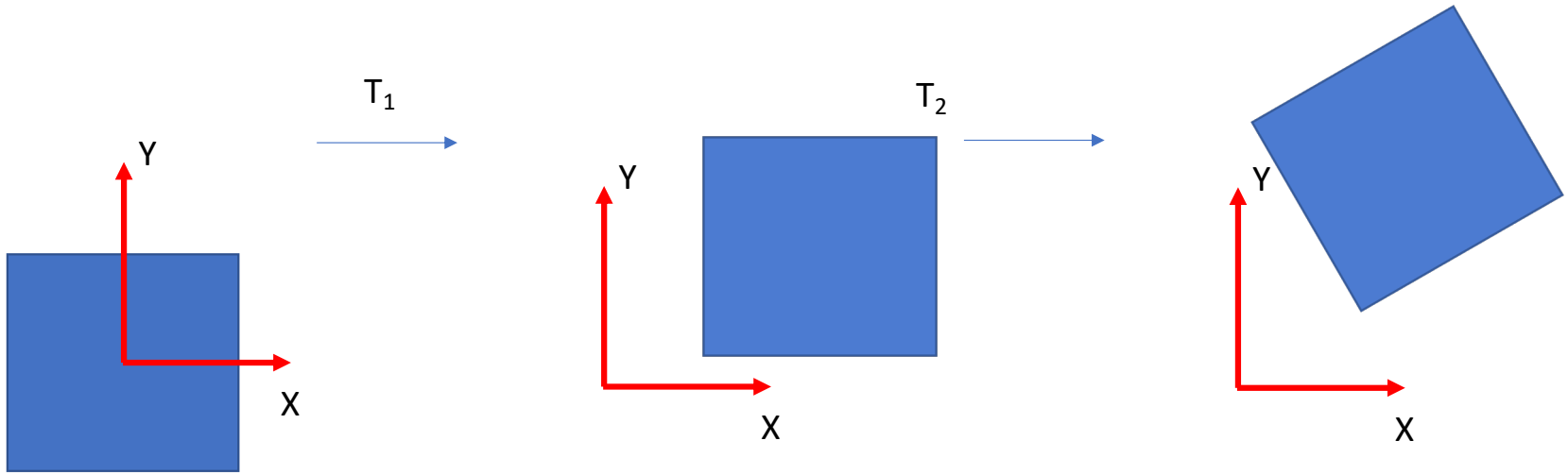
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_T \\ 0 & 1 & y_T \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of arbitrary movements

- Rotations and translations do not commute
- In general, matrix multiplication is in fact not commutative
- Matrix concatenation: pre and post concatenations

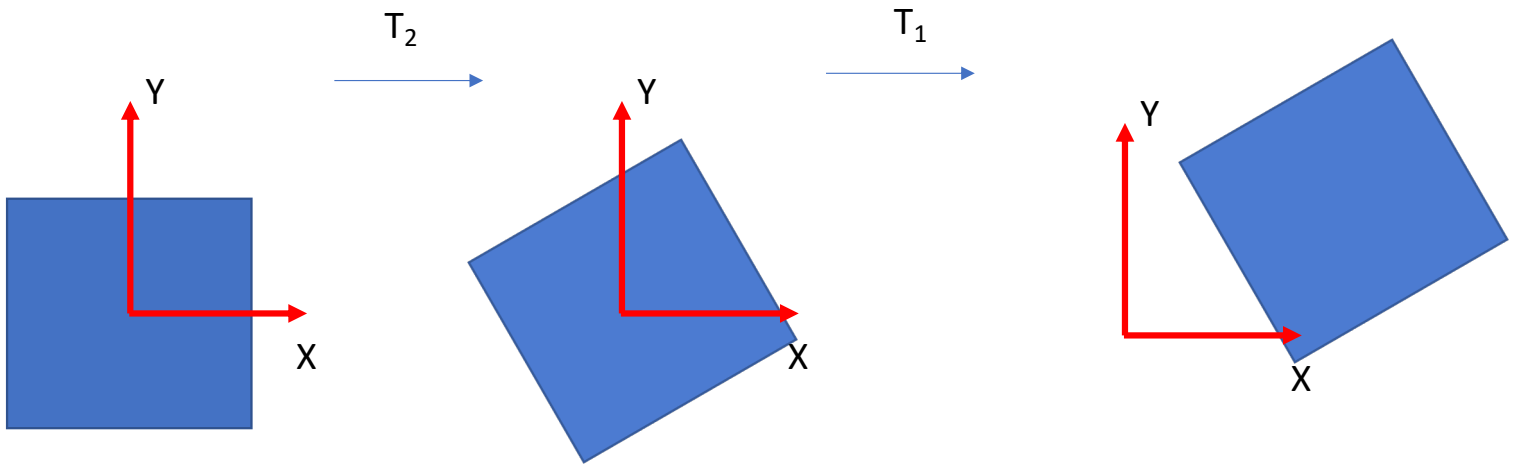
Post-concatenation (post-multiplication)

- $T_2\ T_1$ (T_1 post- concatenates T_2)
- Mnemonic rule for transformation: T_2 applied after (post) T_1
- Example: T_2 is a rotation and T_1 a translation



Pre-concatenation (pre-multiplication)

- T_1 T_2 (T_1 pre- concatenates T_2)
- Example: T_2 is a rotation and T_1 a translation



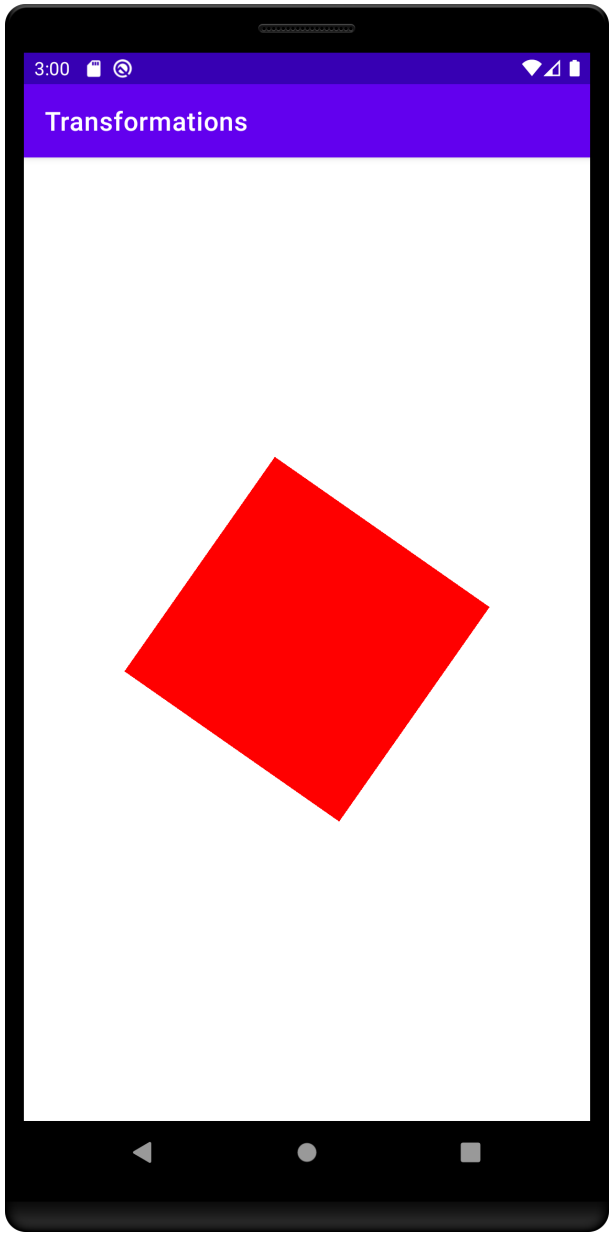
Matrix concatenation

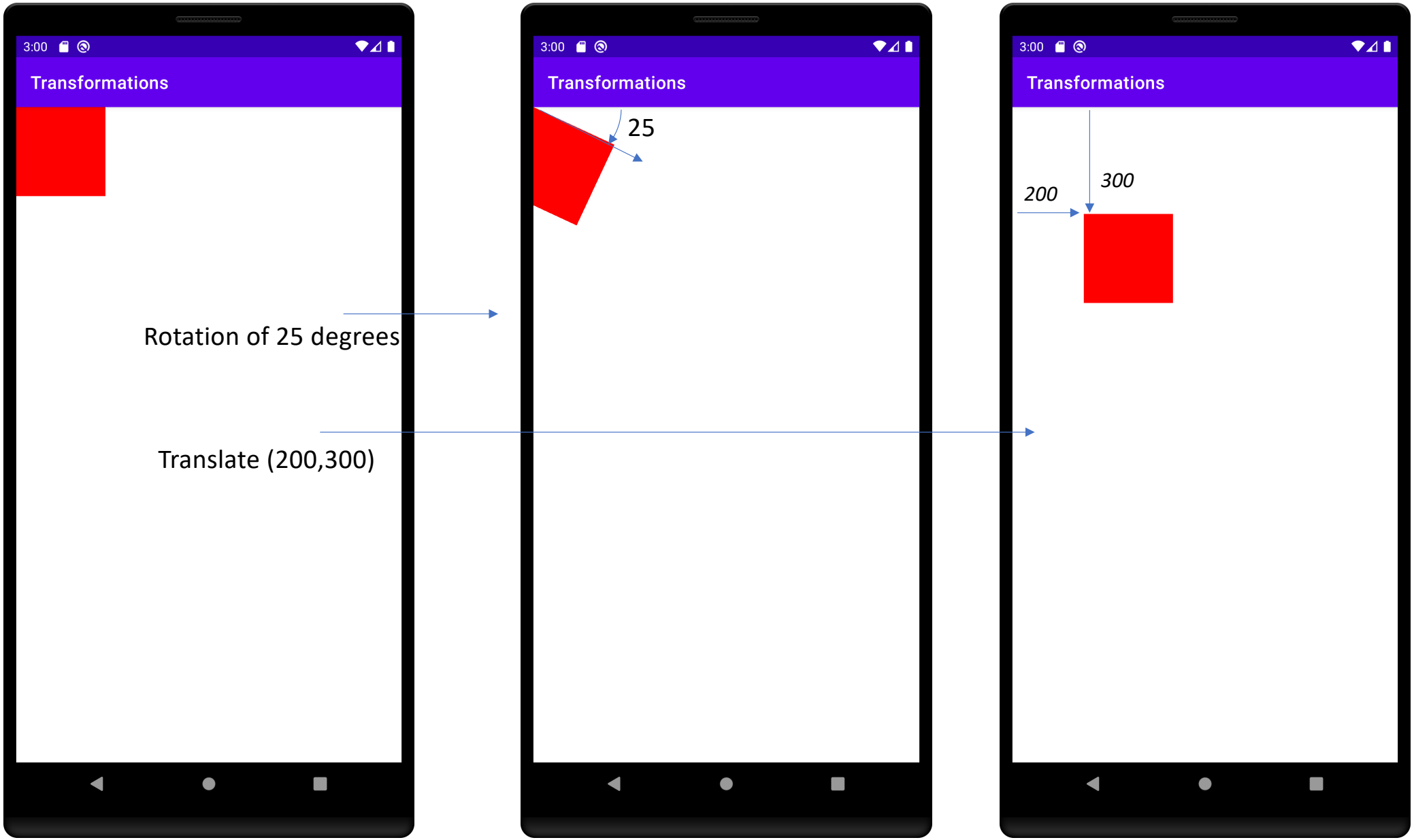
- BA = A.post(B)
- AB = A.pre(B)

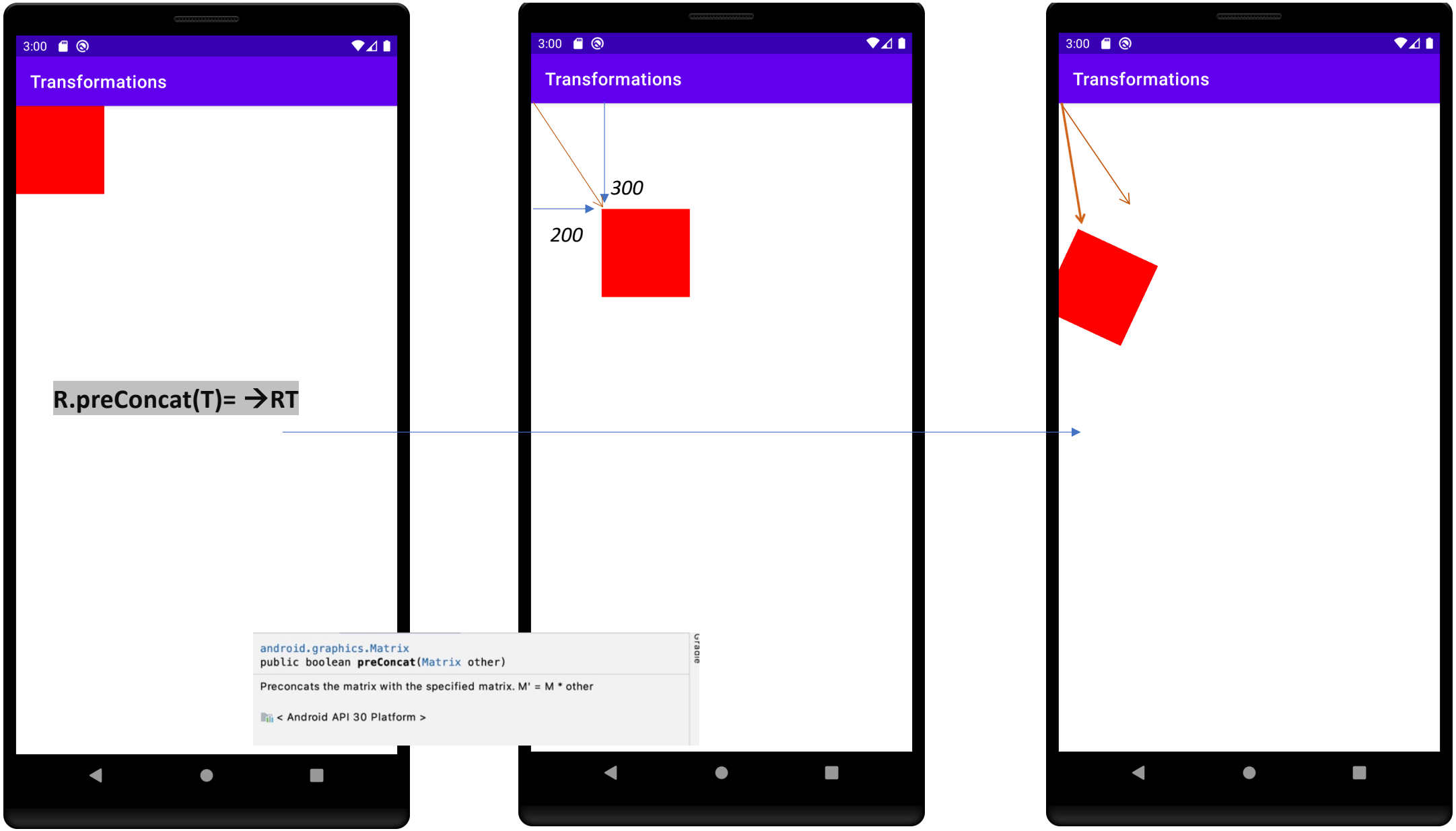
Rotation around an arbitrary point

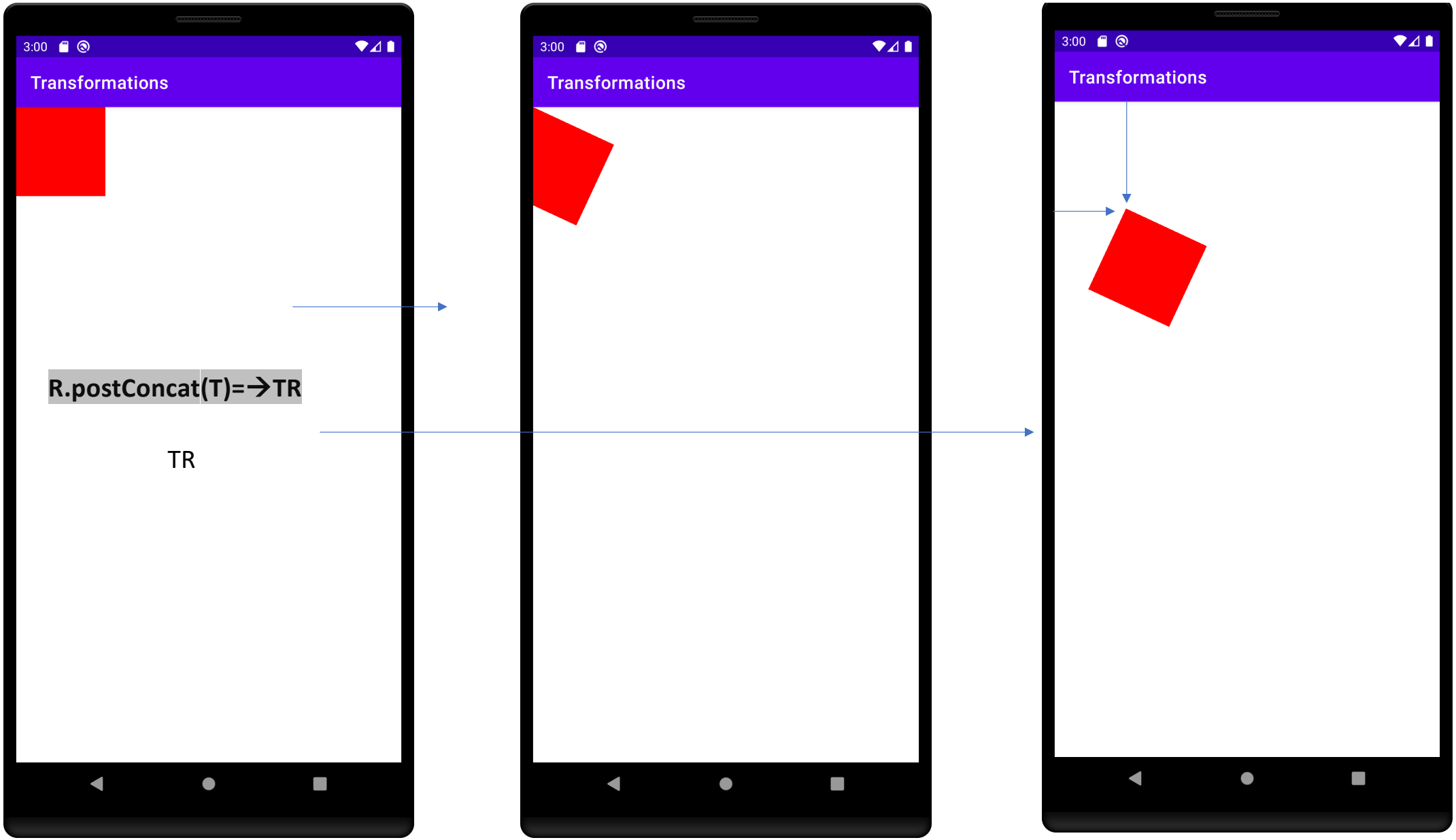
- $M = T_p \, R \, T^{-1}{}_p$

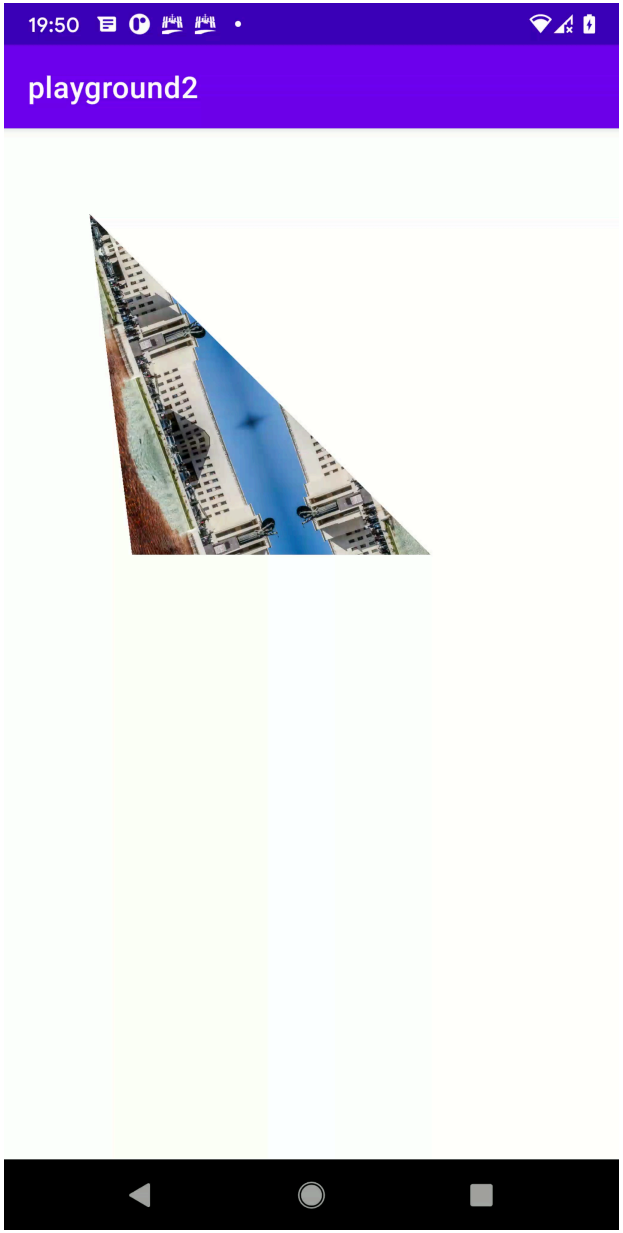
```
T.setTranslate(cx,cy)
M.setRotate(alpha)
M.postConcat(T)
T.setTranslate(-cx,-cy)
M.preConcat(T)
```




3:00
Transformations
Rotation of 25 degrees
Translate (200,300)
25
200
300


3:00
Transformations
300
200
R.preConcat(T)= →RT
android.graphics.Matrix
public boolean preConcat(Matrix other)
Preconcats the matrix with the specified matrix. M' = M * other
< Android API 30 Platform >


3:00
Transformations
R.postConcat(T)=→TR
TR
3:00
Transformations
3:00
Transformations




```
mPaint.shader.getLocalMatrix(localMatrix)
localMatrix.setTranslate(a,a)
localMatrix.postScale(0.5f,0.5f)
localMatrix.postRotate(0.1f*a)

mPaint.shader.setLocalMatrix(localMatrix)
a+=1f
invalidate()
```

Other 2D transformation types

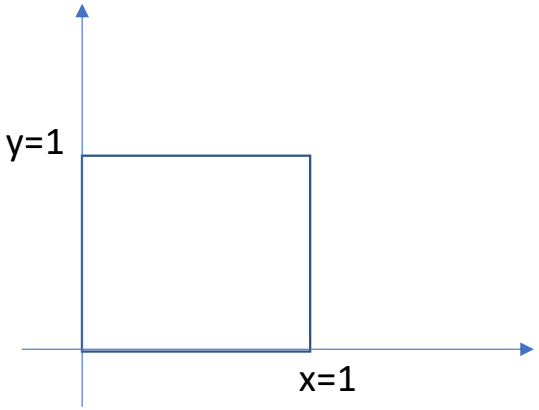
- Affine transformations
- Maps line to line
- Parallel lines to parallel lines
  - Do not preserve distances (

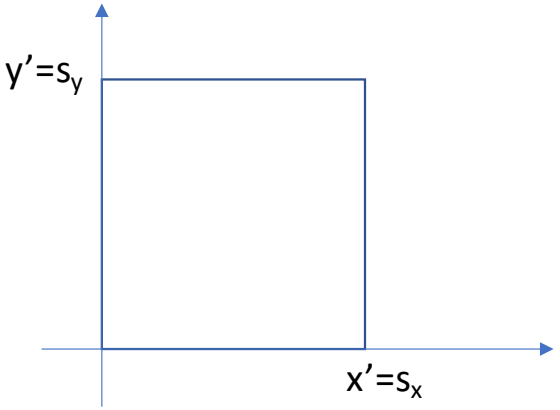
- **Rotation, Translations, Scaling, Skew**

Scaling

$(x,y) \rightarrow (s_x x, s_y y)$

$$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



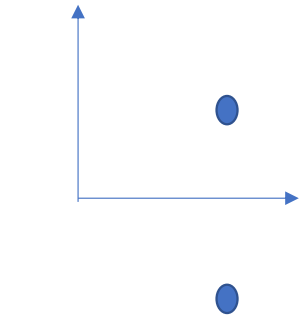


Reflection (scaling factor =-1)

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

Reflection around X axis

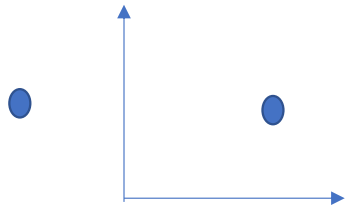
x’=x
y’=-y



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

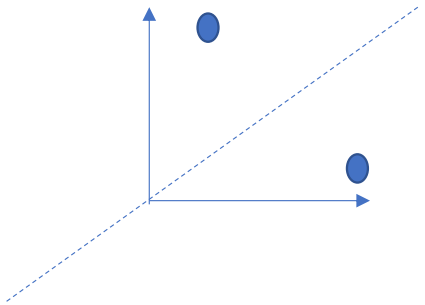
Reflection around Y axis

x’=-x
y’=y

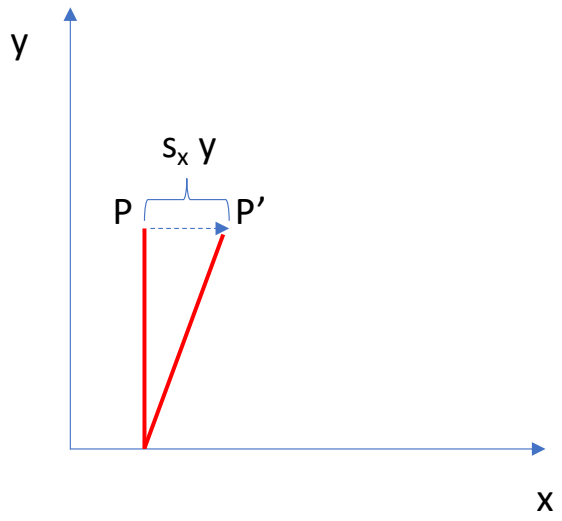


Exercise

- Find the reflection around the oblique line



2D Skew transformations

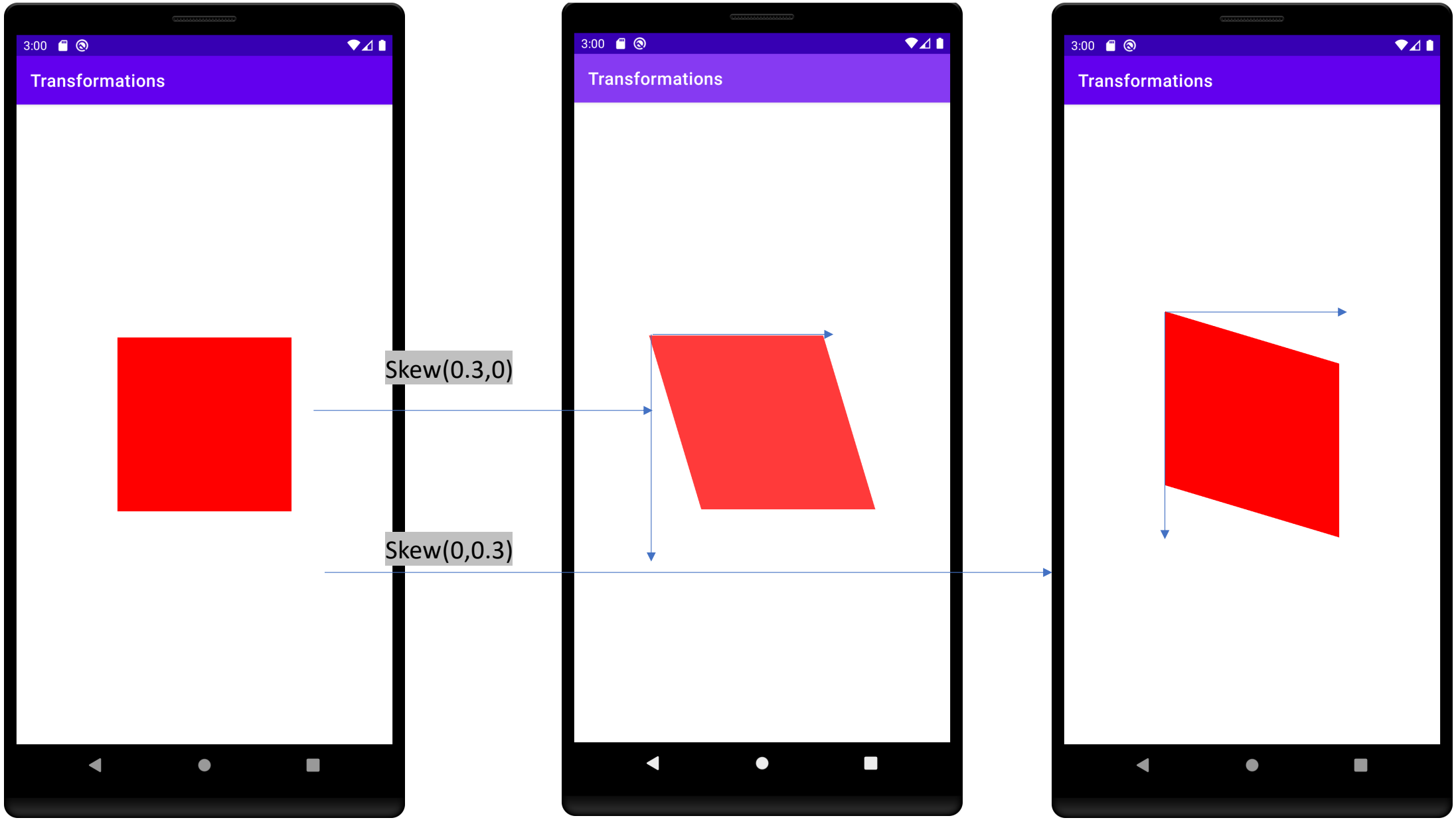


$x' = x + s_x y$

$y' = y$

The amount of skew depends on y

$$T = \begin{bmatrix} 1 & s_X \\ 0 & 1 \end{bmatrix}$$

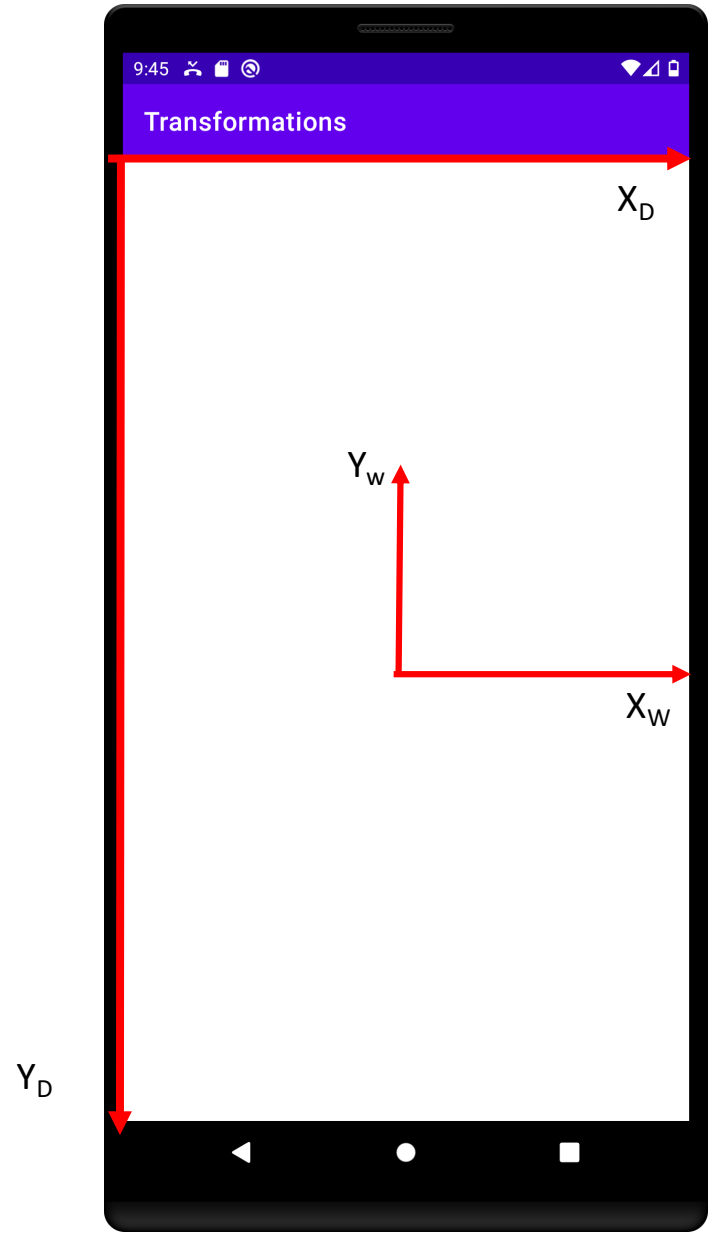

3:00
Transformations
Skew(0.3,0)
Skew(0,0.3)
3:00
Transformations
3:00
Transformations

Transformations in Android

- Android provides the class Matrix (is a proxy to C++ implementation)
- A Matrix can be applied to a single shape
- A Matrix can be applied to the whole canvas

Exercise

- Find the transformation that maps objects drawn in a world frame cantered at the centre of the screen (see figure) to the device frame



QUESTIONS?