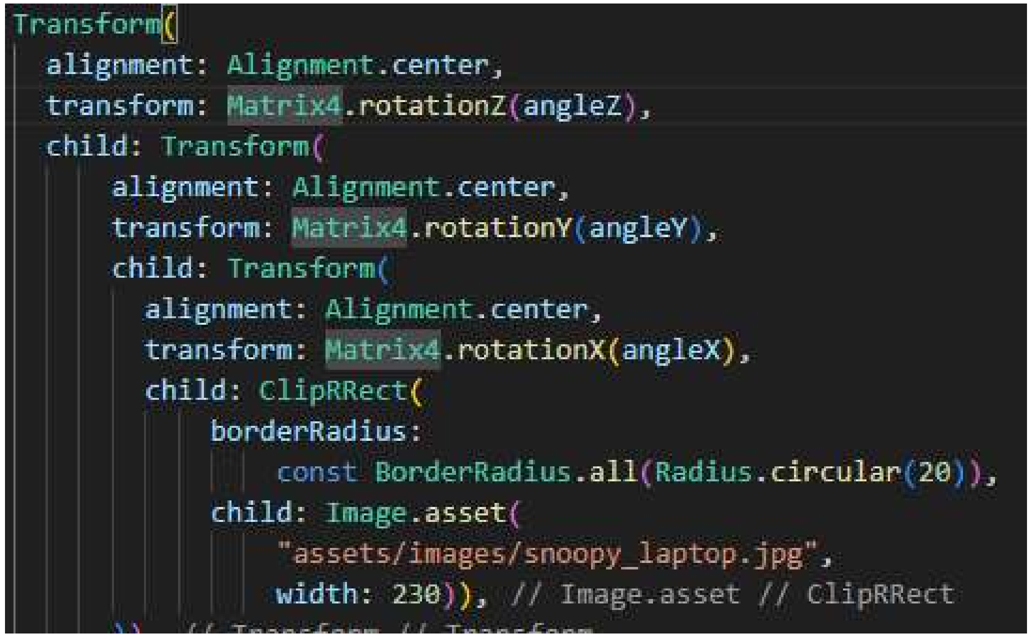
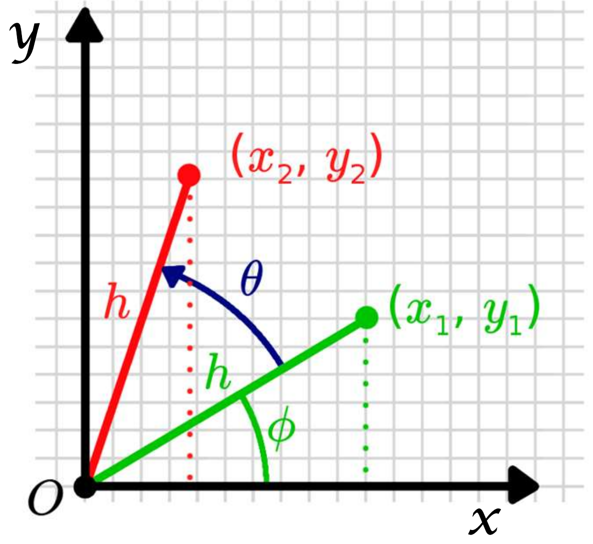
Why is a Matrix4 used in Transform widget ?



```
Transform(
 alignment: Alignment.center,
 transform: Matrix4.rotationZ(angleZ),
 child: Transform(
   alignment: Alignment.center,
   transform: Matrix4.rotationY(angleY),
   child: Transform(
    alignment: Alignment.center,
    transform: Matrix4.rotationX(angleX),
    child: ClipRRect(
      borderRadius:
        const BorderRadius.all(Radius.circular(20)),
      child: Image.asset(
        "assets/images/snoopy_laptop.jpg",
        width: 230)), // Image.asset // ClipRRect
   )), // Transform // Transform
```

Rotation in two dimensions



$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} h\cos(\phi) \\ h\sin(\phi) \end{bmatrix}$$

Additionstheoreme Trigonometrie

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} h\cos(\phi+\theta) \\ h\sin(\phi+\theta) \end{bmatrix}$$

$$= \begin{bmatrix} h\cos(\phi)\cos(\theta) - h\sin(\phi)\sin(\theta) \\ h\sin(\phi)\cos(\theta) + h\cos(\phi)\sin(\theta) \end{bmatrix}$$

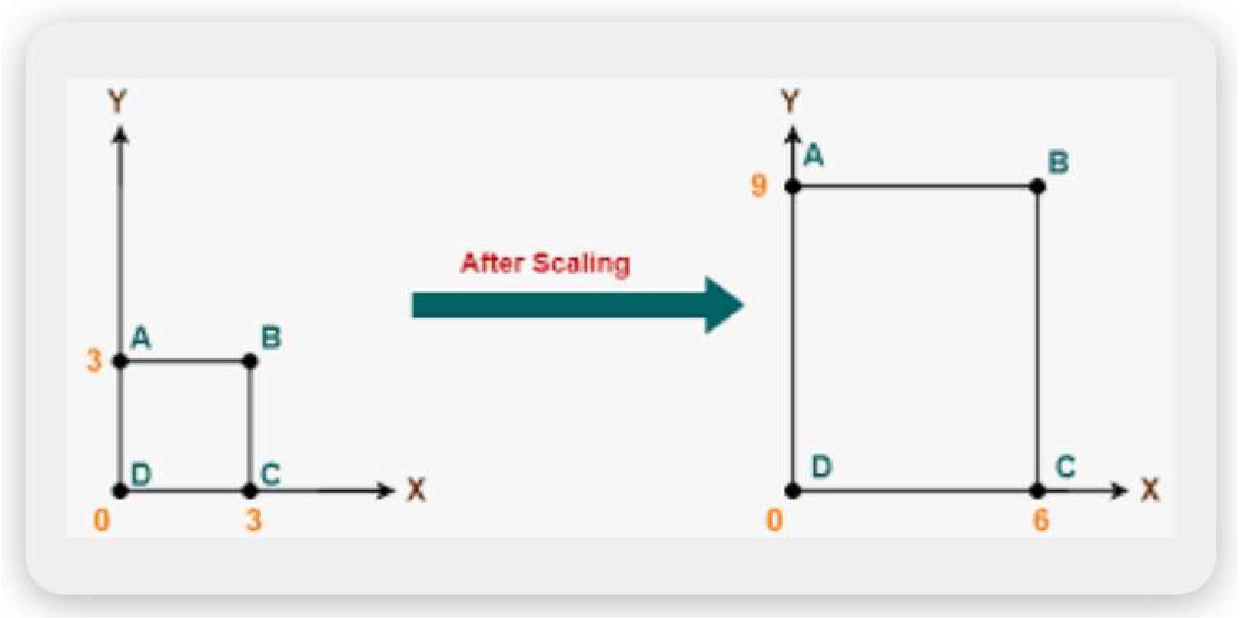
$$= \begin{bmatrix} x_1\cos(\theta) - y_1\sin(\theta) \\ x_1\sin(\theta) + y_1\cos(\theta) \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_{11}b_1 + a_{12}b_2 + \dots \ a_{1n}b_n \\ a_{21}b_1 + a_{22}b_2 + \dots \ a_{2n}b_n \\ \vdots \\ a_{m1}b_1 + a_{m2}b_2 + \dots \ a_{mn}b_n \end{bmatrix}$$

$$\boxed{\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}}$$

Copied from https://articulatedrobotics.xyz/tutorials/coordinate-transforms/rotation-matrices-2d/
and https://www.geeksforgeeks.org/parallel-matrix-vector-multiplication-in-numpy/

Scaling in two dimensions



x‘ = 2 * x   y‘ = 3 * y

In general:

x‘ = sx * x   y‘ = sy * y

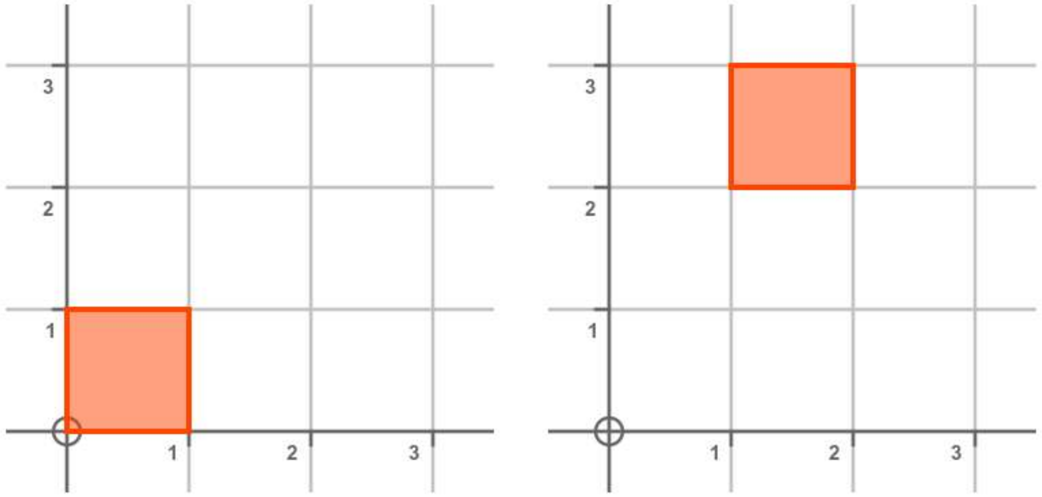
Matrix Representation of Scaling

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Copied from https://bcalabs.org/subject/2d-transformation-in-computer-graphics and https://www.gatevidyalay.com/scaling-in-computer-graphics-definition-examples/

Translation

Translating a shape means moving it to a different position, without changing its shape or orientation in any way. This diagram illustrates a translation by (1, 2). The square moves by 1 unit in the x-direction and 2 units in the y-direction:

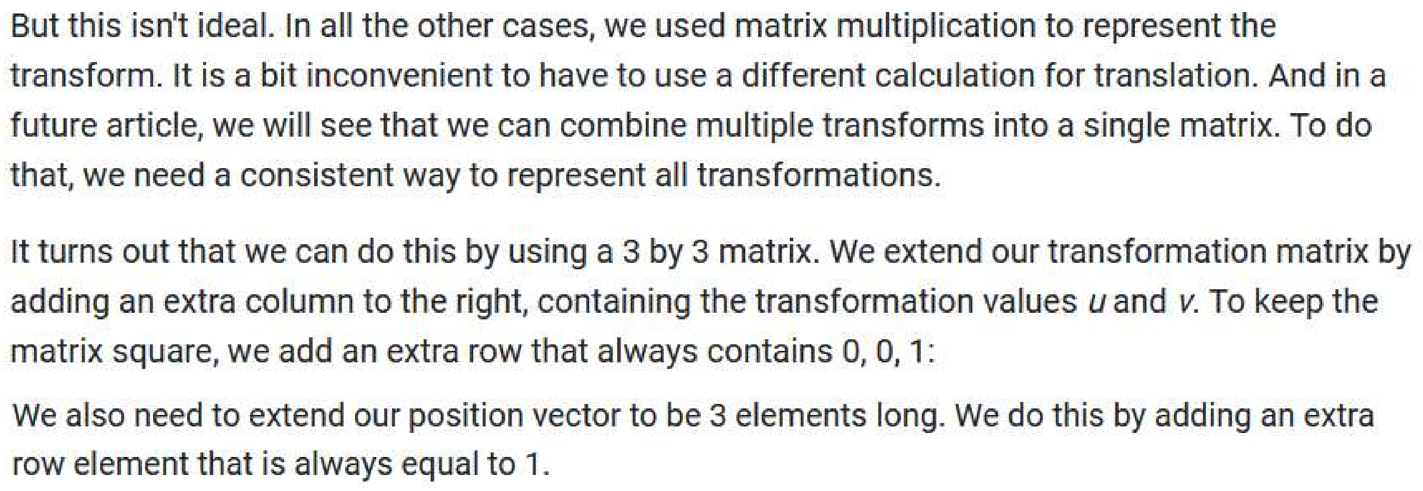


We can represent this by adding 2 vectors, the original position *(x, y)* and a displacement vector *(u, v)*:

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x + u \\ y + v \end{pmatrix}$$

Copied from https://bcalabs.org/subject/2d-transformation-in-computer-graphics

Translation using matrix multiplication



But this isn't ideal. In all the other cases, we used matrix multiplication to represent the transform. It is a bit inconvenient to have to use a different calculation for translation. And in a future article, we will see that we can combine multiple transforms into a single matrix. To do that, we need a consistent way to represent all transformations.

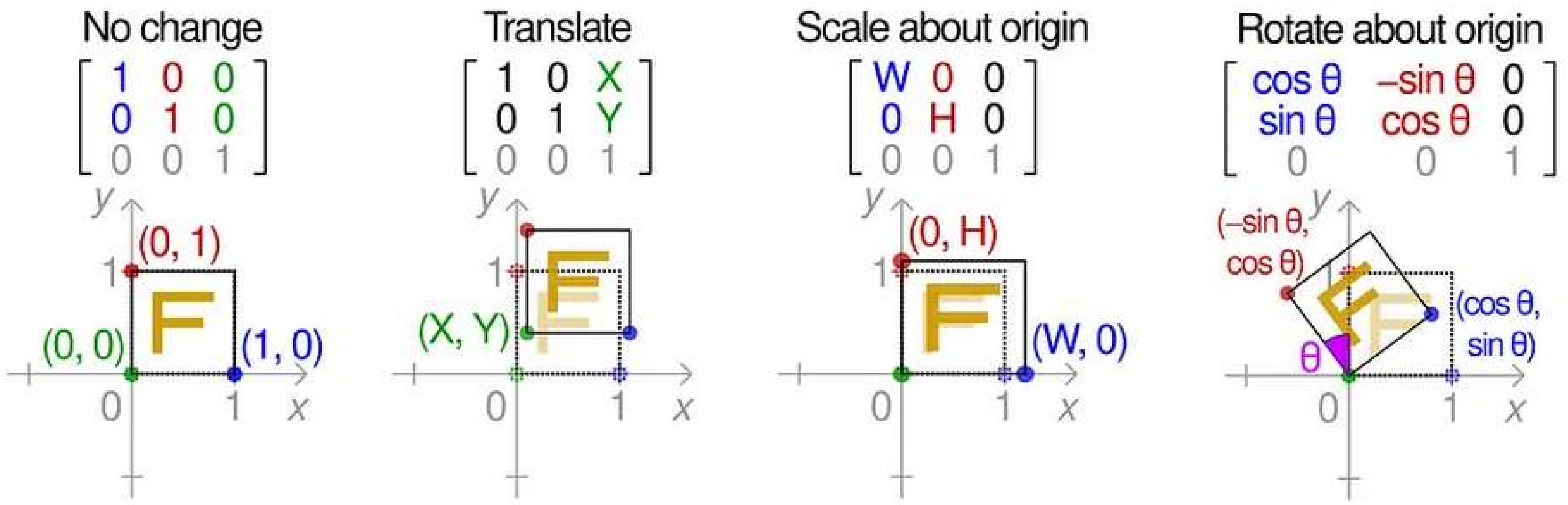
It turns out that we can do this by using a 3 by 3 matrix. We extend our transformation matrix by adding an extra column to the right, containing the transformation values *u* and *v*. To keep the matrix square, we add an extra row that always contains 0, 0, 1:

We also need to extend our position vector to be 3 elements long. We do this by adding an extra row element that is always equal to 1.

$$\begin{pmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+u \\ y+v \\ 1 \end{pmatrix}$$

Copied from https://bcalabs.org/subject/2d-transformation-in-computer-graphics

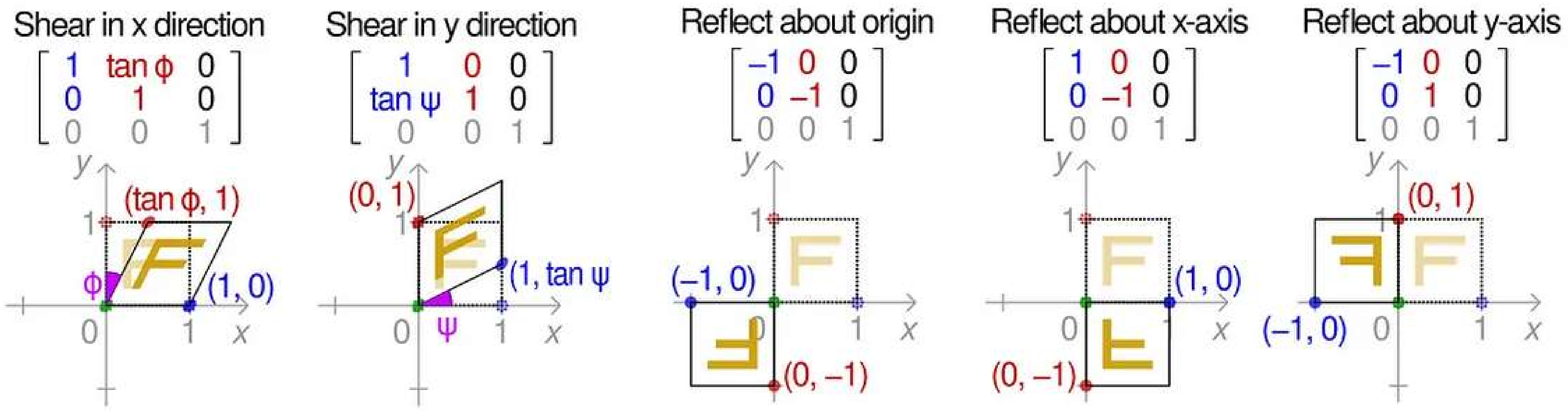
2D Transformations with 3x3 Matrices



No change
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(0, 0)
(1, 0)
Translate
$\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$
(X, Y)
Scale about origin
$\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, H)
(W, 0)
Rotate about origin
$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(−sin θ, cos θ)
(cos θ, sin θ)
θ

Copied from https://medium.com/@junfeng142857/affine-transformation-why-3d-matrix-for-a-2d-transformation-8922b08bce75

2D Transformations with 3x3 Matrices



Shear in x direction
$\begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(tan φ, 1)
φ
(1, 0)
Shear in y direction
$\begin{bmatrix} 1 & 0 & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(1, tan ψ
ψ
Reflect about origin
$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(–1, 0)
(0, –1)
Reflect about x-axis
$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(1, 0)
(0, –1)
Reflect about y-axis
$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
(0, 1)
(–1, 0)
x
y
0
1

Copied from https://medium.com/@junfeng142857/affine-transformation-why-3d-matrix-for-a-2d-transformation-8922b08bce75

3D Transformations with 4x4 Matrices

➢ **Translation Transformation-**

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_x \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

➢ **Scaling Transformation-**

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation about X-

$$\begin{aligned} y' &= y\cos\gamma - z\sin\gamma \\ z' &= y\sin\gamma + z\cos\gamma \\ x' &= x \end{aligned}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation about Y-

$$\begin{aligned} z' &= z\cos\beta - x\sin\beta \\ x' &= z\sin\beta + x\cos\beta \\ y' &= y \end{aligned}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotation about Z-

$$\begin{aligned} x' &= x\cos\alpha - y\sin\alpha \\ y' &= x\sin\alpha + y\cos\alpha \\ z' &= z \end{aligned}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

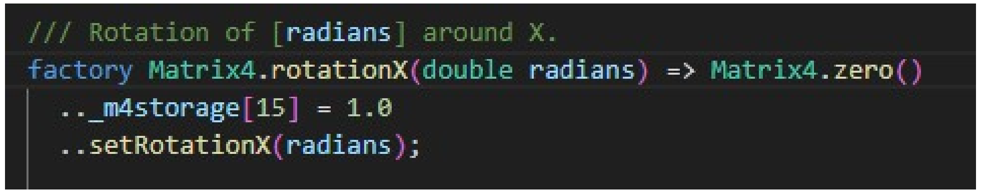
Copied from https://medium.com/@junfeng142857/affine-transformation-why-3d-matrix-for-a-2d-transformation-8922b08bce75

Implementation in Flutter

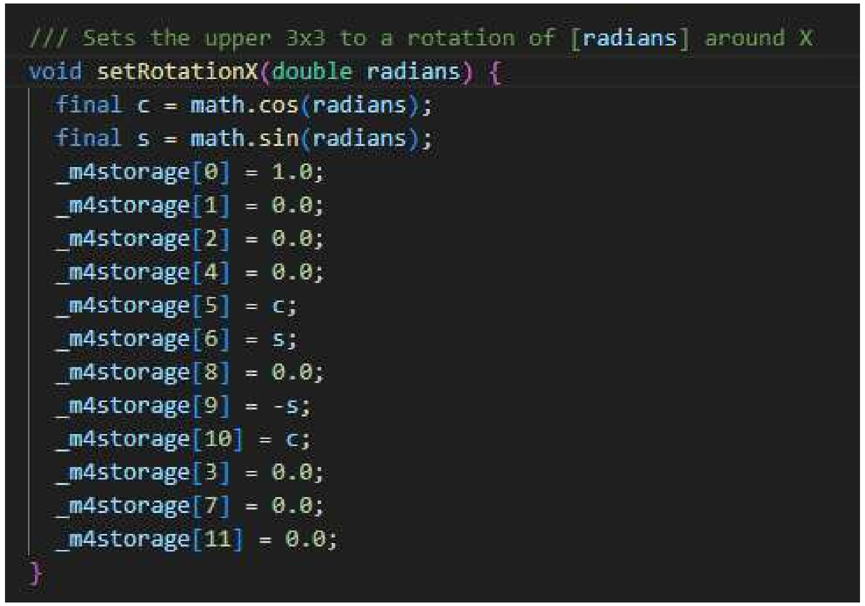
Rotation about X-

$$y' = y\cos\gamma - z\sin\gamma$$
$$z' = y\sin\gamma + z\cos\gamma$$
$$x' = x$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



```
/// Rotation of [radians] around X.
factory Matrix4.rotationX(double radians) => Matrix4.zero()
  .._m4storage[15] = 1.0
  ..setRotationX(radians);
```



```
/// Sets the upper 3x3 to a rotation of [radians] around X
void setRotationX(double radians) {
  final c = math.cos(radians);
  final s = math.sin(radians);
  _m4storage[0] = 1.0;
  _m4storage[1] = 0.0;
  _m4storage[2] = 0.0;
  _m4storage[4] = 0.0;
  _m4storage[5] = c;
  _m4storage[6] = s;
  _m4storage[8] = 0.0;
  _m4storage[9] = -s;
  _m4storage[10] = c;
  _m4storage[3] = 0.0;
  _m4storage[7] = 0.0;
  _m4storage[11] = 0.0;
}
```