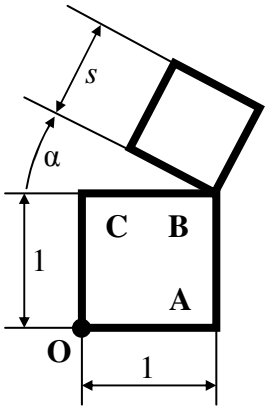
2IV60 – Exercise 2: Transformations and viewing

1. Given a square with width and height 1, with the lower left corner in the origin:



   a. Give a transformation matrix to transform this square to a square with width *s,* rotated over an angle α, such that the lower right corner coincides with the upper right corner of the original square (see figure).

      First we translate the lower right corner to the origin (**T**(−1,0)), next we scale (**S**(*s*, *s*)) and rotate (**R**(−α)). Following the figure, a negative angle has to be used. Finally, we translate the lower right corner (which is now in the origin) to the upper right corner with **T**(1,1). This all in global coordinates. The composite matrix **M** is hence given by: **M** = **T**(1,1) **R**(−α) **S**(*s*,*s*) **T**(−1,0).

   b. We consider a more generic version. Give a recipe for a transformation matrix for a rotated and scaled unit square **OABC**, such that a new corner **O**', **A**', **B**' or **C**' coincides with one of the original corners **O**, **A**, **B** of **C**.

      Define $\mathbf{P}_i$ as follows:
      $\mathbf{P}_0 = \mathbf{O} = (0,0)$;
      $\mathbf{P}_1 = \mathbf{A} = (1,0)$;
      $\mathbf{P}_2 = \mathbf{B} = (1,1)$;
      $\mathbf{P}_3 = \mathbf{C} = (0,1)$.
      The task is now to transform the square such that a point $\mathbf{P}_i$ is mapped to a point $\mathbf{P}_k$. In the answer for question a) two translations are used. The translation T(−1,0) determines around which point is rotated, the translation **T**(1,1) determines where this point ends up. In the new numbering of the points, the rotation is made around $\mathbf{P}_1$, the final translation is towards $\mathbf{P}_2$. If we generalize this, we get **M** = **T**($\mathbf{P}_i$) **R**(−α) **S**(*s*,*s*) **T**(−$\mathbf{P}_k$).

2. We consider 3D rotations.
   a. Give 3×3 rotation matrices for a rotation $\mathbf{R}_x(\alpha)$ around the *x*-as and a rotation $\mathbf{R}_z(\alpha)$ around the *z*-axis.

$$R_x(\alpha)=\begin{pmatrix}1 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha\\ 0 & \sin\alpha & \cos\alpha\end{pmatrix},\ \text{en}\ R_z(\alpha)=\begin{pmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{pmatrix}.$$

b. Calculate (explicitly) the composite rotation matrix for a rotation over 90 degrees ($\pi/2$ radians) around the x-axis followed by a rotation over 90 degrees over the (global) z-axis.

$$R_z(90)R_x(90)=\begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{pmatrix}=\begin{pmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{pmatrix}.$$

c. What is the effect of these two rotations? Elucidate your answer.

The x-axis is mapped to the y-axis, the y-axis to the z-axis, and the z-axis to the x-axis. In other words, the axes are cyclically permuted. We can interpret this effect as a counterclockwise rotation around a vector (1, 1, 1) over an angle of 120 degrees.

d. Do we get the same effect if we change the order of the rotations?

No. Matrix-multiplications do not commute. If we calculate the effect, we get:

$$R_x(90)R_z(90)=\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}=\begin{pmatrix}0 & -1 & 0\\ 0 & 0 & -1\\ 1 & 0 & 0\end{pmatrix}$$

and indeed, the effect is dramatically different. The x-axis is mapped to the z-axis, the y-axis to the negative x-axis, and the z-axis to the negative y-axis. This corresponds to a rotation around the vector (1, −1, 1). This last vector can be found by using the facts that (a) sequences of rotations lead to rotations; (b) a rotation leaves the rotation axis in tact. Hence, a rotation axis can be found by solving:

$$\begin{pmatrix}0 & -1 & 0\\ 0 & 0 & -1\\ 1 & 0 & 0\end{pmatrix}\begin{pmatrix}a\\ b\\ c\end{pmatrix}=\begin{pmatrix}a\\ b\\ c\end{pmatrix},\ \text{which gives}\ -b=a,-c=b,\ \text{and}\ a=c.$$

Setting c=1 (arbitrarily) gives the values for a and b.

3. We consider a distorted cube. In the original state the cube is described by $0\le x,y,z\le 1$. After distortion the following applies:
   - The lower face (in the XOY-plane) remains fixed;
   - The upper face is horizontal, but moves: the point (0,0,1) moves to (*a, b, c*) ;
   - all edges remain straight and all faces remain flat.

   a. Give a transformation matrix for this distortion.

   The origin remains fixed, hence there is no translation. Furthermore, also the x-axis (1, 0, 0) and the y-axis (0, 1, 0) do not change after transformation. However,

the new direction of the z-axis (0, 0, 1) is (a, b, c). Using this, we can immediately write down the transformation matrix:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Check that if we multiply the points (0, 0, 0, 1), (1, 0, 0, 1), (0, 1, 0, 1) and (0, 0, 1, 0) with $\mathbf{M}$ ($x' = \mathbf{M}x$, x' and x column vectors) we indeed get the desired result.

4. Given a digital map. It is desired to show a part of this map in a viewport on the screen. In the center of the viewport the point **C** (in map coordinates) has to be shown, the width of the part to be displayed is w (again in map coordinates). Obviously, distortion is not allowed. The viewport is specified in pixels. For the pixel coordinates it holds that the origin (the point (0,0)) is located in the upper left corner and the point (Nx−1, Ny−1) in the lower right corner. The upper left corner of the viewport has coordinates ($\mathbf{X}_{mi}$, $\mathbf{Y}_{mi}$), the lower right corner ($\mathbf{X}_{ma}$, $\mathbf{Y}_{ma}$).
   a. Give a transformation to map a point **Q** in map coordinates to a point **Q**' in pixel coordinates.

We first determine the height h of the window, by requiring that the aspect ratio of window and viewport are the same:

$$\frac{h}{w} = \frac{Yma - Ymi}{Xma - Xmi} \text{ hence } h = w\frac{Yma - Ymi}{Xma - Xmi}.$$

Using this we can find the lower left corner **P** of the window:

$\mathbf{P} = \mathbf{C} - (w/2, h/2)$.

We move the point **P** to the origin with $\mathbf{T}(-\mathbf{P})$. Next we scale the width to 1, using $\mathbf{S}(1/w, 1/w)$, and we scale to the width of the viewport with $\mathbf{S}(\mathbf{X}_{ma} - \mathbf{X}_{mi}, \mathbf{X}_{ma} - \mathbf{X}_{mi})$. We use the same scale factor for x and y, such that no distortion is introduced. To make sure that the y-axis has the proper direction, we reflect around the horizontal axis with $\mathbf{S}(1, -1)$. Finally we translate the origin to the lower left corner of the viewport with $\mathbf{T}(\mathbf{X}_{mi}, \mathbf{Y}_{ma})$. Taken together, with combined scaling, we get:

$$\mathbf{M} = \mathbf{T}(\mathbf{X}_{mi}, \mathbf{Y}_{ma})\, \mathbf{S}((\mathbf{X}_{ma} - \mathbf{X}_{mi})/w, -(\mathbf{X}_{ma} - \mathbf{X}_{mi}) / w)\, \mathbf{T}(-P).$$

   b. A user indicates a point **R**' (pixel coordinates). Give a transformation to determine the corresponding point R in map coordinates.

This transformation is the inverse of **M**. $\mathbf{R}' = \mathbf{M}\mathbf{R}$, hence $\mathbf{R} = \mathbf{M}^{-1}\mathbf{R}'$. We can determine the inverse using some generic procedure for **M**, but we can also derive it from the formula. It should hold that $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$. We achieve this by right multiplying **M** with a sequence of transformations that cancel the transformations of **M** from right to left. This leads to:

$$\mathbf{M}^{-1} = \mathbf{T}(\mathbf{P})\, \mathbf{S}(w/(\mathbf{X}_{ma} - \mathbf{X}_{mi}), -w/(\mathbf{X}_{ma} - \mathbf{X}_{mi}))\, \mathbf{T}(-\mathbf{X}_{mi}, -\mathbf{Y}_{ma}).$$

5. Given a 3D scene that is viewed with a virtual camera. The camera has position **P**, points in the direction **W**, and lines in the direction of a vector **V** are shown vertically on the screen. It may be assumed that **V** and **W** are unit vectors and that they are

orthogonal. In the following questions the effect of a given camera movement on **P**, **W** and **V** is asked for.

a. Give recipes to move the camera forward and backward, up and down, and to the left and to the right.

We define an auxiliary vector **U**=**W**$\times$**V**. We now have a complete axis frame for our virtual camera:: **U** points to the right, **V** up, and **W** in the direction of the screen. We can use these to perform the desired operations. It can also be done with generic transformation routines, but here it is easier to do it directly.
If the displacement is *a,* then we get

Forward and backward: $\mathbf{P}' = \mathbf{P} \pm a\mathbf{W}$;
Up and down: $\mathbf{P}' = \mathbf{P} \pm a\mathbf{V}$;
Right / left: $\mathbf{P}' = \mathbf{P} \pm a\mathbf{U}$.

The vectors **U** and **V** do not change, the direction remains the same.

b. Give recipes to rotate the camera: *pitch* (left/right), *yaw* (up/down), *roll* (rotation around center of viewing axis).

Here the vectors **V** and **W** change, and **P** does not change. Again, this can be done with generic routines, but it is easier to do it directly.
Two orthonormal 3D vectors **X** en **Y** can be rotated via:

```
procedure RotVecs(X, Y: T3Dvector; var X', Y': T3Dvector; α: real);
begin
    X' =  cos α X + sin α Y ;
    Y' = –sin α X + cos α Y
end;
```

With this procedure we can write down the rotations asked as:

```
pitch:    RotVecs(U, W, U', W', α);  // rotation around V
yaw:      RotVecs(V, W, V', W', α);  // rotation around U
roll:     RotVecs(U, V, U', V', α);  // rotation around W
```

c. Suppose that the camera is centered at a point **C**. Give again recipes for the rotation, such that the camera remains fixed on this point.

The answer for b gives the requested new orientation of the vectors. The new position of **P** can be determined by using the meaning of the vectors of the frame. The vector **W** points from **P** to **C**, and after rotating the camera we want that **W'** points from **P'** to **C**. We already know the vector **W'**, and we can use that to easily determine **P'**:

$$\mathbf{P}' = \mathbf{C} - \mathbf{W}' \, | \, \mathbf{P} - \mathbf{C} \, | \, .$$

In other words, we find the new position of the camera by starting in **C**, and going in the direction of –**W'** over the same distance $| \, \mathbf{P} - \mathbf{C} \, |$ as in the original situation.

For students attracted by challenges:

6. Consider question 3, where we looked at displaying a digital map. Suppose that a user has centered the map on a point **C** (for instance Eindhoven) with a width w (10 km). Next he indicates that he wants to center the map on a point **C**' (say, Amsterdam) with a width w' (20 km). An animation $\mathbf{C}(t), w(t),$ is desired, such that a smooth transition is showed.
    a. What are the requirements?

    We might want that (a) the animation is continuous (no jerky motions), that (b) the 'amount' of motion on the screen is constant and that (c) the animation is efficient: from **C** to **C'** in the shortest time.

    b. How to define such an animation?

    We can first zoom out, next translate the camera, and finally zoom in: move the camera up, side-ways, and down again. To get a constant change during zooming out, we change w with a constant factor: $w(t+\Delta t) = f\, w(t)$, with $f > 1$, until $w(t) = w_{MAX}$. If we change this difference equation into a differential equation ($dw/dt = fw$) and solve it, we get

    $$w(t) = \exp(f\, t).$$

    For w_{MAX} we use $\alpha|\, \mathbf{C}-\mathbf{C'}\, |$, such that the maximal zoom-out factor depends on the distance between **C** and **C'**. Note that a linear change of w ($w(t+\Delta t) = w(t) + \Delta w$) does not give a constant change: If w = 10 km a change of 1 km has much more impact than for w=100km. The translation however can be done in a linear way:

    $$\boldsymbol{C}(t+\Delta t\,) = \boldsymbol{C}(t) + s(\mathbf{C}-\mathbf{C'}).$$

    During zooming in we use a factor $1/f$. Finally we tune the factors α, s and f experimentally, such that visually a nice animation results.

    This solution can be improved. At the transition from zooming to translation a (little) shock occurs, and also, it is not the most efficient route. It is better to move and zoom in (and out) simultaneously, our virtual camera has to move over the map using a kind of curve. For a complete elaboration of this, see:

    Wijk, J.J. van, Wim A.A. Nuij. *Smooth and Efficient Zooming and Panning.* In: T. Munzner, S. North (eds.), Proceedings IEEE Symposium on Information Visualization (InfoVis'2003), IEEE Computer Society Press, October 2003, p. 15-22. (http://www.win.tue.nl/~vanwijk/zoompan.pdf).