Group Assignment 1

Deadline: **February 18th, 2021**

In this assignment, you will continue working on the planetary system that you created. This time, you will use affine transformations for the relative movement of the celestial bodies. Then you will work with the relative positions and coordinate systems of several space ships in various scenarios.
Note: Make sure to always set the world center mode to the camera!

PREPARATION

- Download and print the templates for image targets *spaceship_set* and *earth* from the Group Assignments page on Blackboard.

Transformations and coordinate systems

TASKS

a) **4pt** Adapt your earlier approach, to work *without* using the Unity GameObject hierarchy: Make the earth rotate around the sun **on an ellipse** by updating its position in a script. Likewise, make the moon rotate around the earth by directly adjusting its position. You will need to use transformation matrices to calculate the new positions of earth and moon, respectively.
Note: To create the transformation matrices, you may only use the `Matrix4x4` struct and the operations it offers.

- Assign your game object positions to the translation part of your resulting transformation matrix.
- For this task, you may use Unity without Vuforia, i.e., you do not need to attach this system of objects to an image target or add an AR camera to the scene.

Explain: In the hierarchy-based implementation, the planets rotate around their own axis, and that rotation affects other planets in the hierarchy. By using transformation matrices the planets no longer rotate around their own axis. Why do they behave differently?

Explain: Does the order of transformation matrices matter? Why?

b) **1pt** Make all celestial bodies (Earth, Sun, Moon) rotate around their own local vertical axis, independent of their position, by adjusting their `transform.rotation` value.

c) **3pt** Create a new scene with the space shuttle and the landing strip image targets. You will need to use Vuforia again for this part.
Attach a small patch (e.g., Quad) to the space shuttle image target and assign a new material to it, which uses the *Unlit* → *Color* shader. In a script, set the color of this material at runtime to visualize whether the space shuttle is aligned with the landing strip. The color should go from red (at least one axis is orthogonal) to green (all axes are completely aligned).

- You can retrieve the basis vectors of your GameObjects `transform.right`, `transform.up` and `transform.forward`. Note that these vectors are normalized.
- You can use the following line of code to set the color of your quad at runtime:
  `yourGameObject.GetComponent<Renderer>().material.color = new Color(yourRedValue, yourGreenValue, yourBlueValue);`
  You can interpolate between colors using `Color.Lerp`.

d) **3pt** Attach a GameObject to the "nose" of the space shuttle. Transform the `localPosition` of this object from the local coordinate system of the space shuttle object to that of the landing strip. To do this, create a `Matrix4x4` (based on the GameObjects' model matrices) that first transforms the `localPosition` of the nose to world coordinates, and then to the local coordinate system of the landing strip. In the Unity scene, display the resulting position as text.

Explain: Imagine having to transform very many GameObject positions, or points, instead of just one. Which of the matrix transformations above must be done for each single point, and which can be done only once per frame?

e) **BONUS 1pt** Detect whether the space shuttle is approaching to land, i.e., its nose is located just above and next to the image target of the landing strip and pointing towards the landing strip. Use the nose position in local xyz-coordinates of the landing strip that you just computed, and define box-shaped boundaries for the landing strip. Check how far away x and z coordinates of the nose are from the origin of the local coordinate system and whether y is within an upper and lower threshold. Define any maximum height (max.y value) of the box that you find appropriate. In the Unity scene, display different text depending on whether the shuttle is on the correct course for landing (e.g., "Please approach runway" and "Prepare for landing").

f) **4pt** Create another scene with the Millennium Falcon and shuttle. Assign a button or key stroke to shoot a laser from the Falcon along its viewing direction (i.e., forward). Visualize the ray

in the scene, and check whether it hits the enemy ship. Visually indicate a hit, e.g., with an explosion (you can check out particle effects for this).

- Use the `drawLine` function in the auxiliary material to draw the laser in your scene.
- To detect laser hits, attach a box collider to the shuttle (make the bounding box a bit bigger to compensate for inaccuracies). Then use `Physics.RayCast` to cast a ray that returns a 'hit' when it intersects with a collider.
- Use this figure as an orientation for tasks f, g and h:

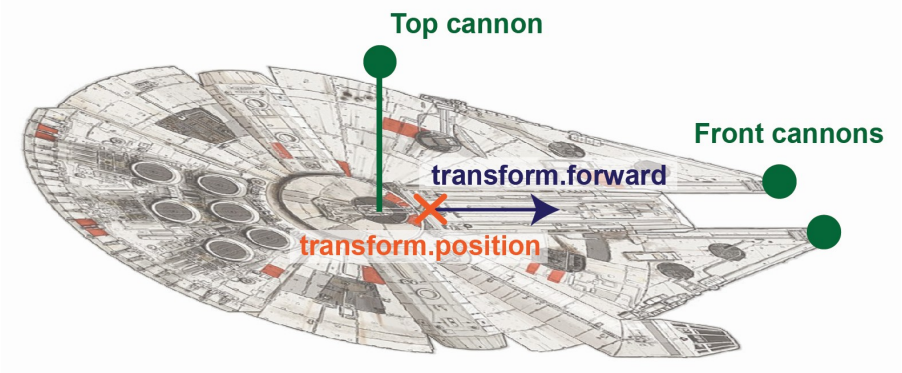

Figure 1: Local positions of cannons on the Millennium Falcon.

g) **BONUS 2pt** Shoot two lasers from both front cannons of the Falcon (see Figure 1). Instead of creating child GameObjects for your cannons and accessing their `tranform.position`, define their local positions in relation to the Falcon. Then use the Falcon's position and rotation in Euler angles to compute the positions of both cannons in world coordinates. You must use `Matrix4x4` to achieve this. Reuse the convenience functions for affine transformations from Assignment 1 (a).

- To retrieve the rotation of the Falcon ImageTarget in Euler angles you can use the following line of code: `yourGameObject.transform.rotation.eulerAngles;`

h) **BONUS 2pt** Shoot a laser from the top cannon (see Figure 1). Define the local position and rotation of the top canon in code, instead of using child GameObjects. The top cannon should be 2.5cm above the spaceship. Define yaw and pitch variables to adjust the angle of the top cannon and adjust the direction of the laser accordingly (i.e., apply the rotation). You may use only `Matrix4x4` multiplications to compute position and rotation of the top cannon in world coordinates. Again, use the convenience functions for affine transformations that you implemented in Assignment 1 (a). Try changing the order in which the translation and rotation matrices are concatenated. How does this affect the result?

Min / Total = **7.0pt / 20.0pt**