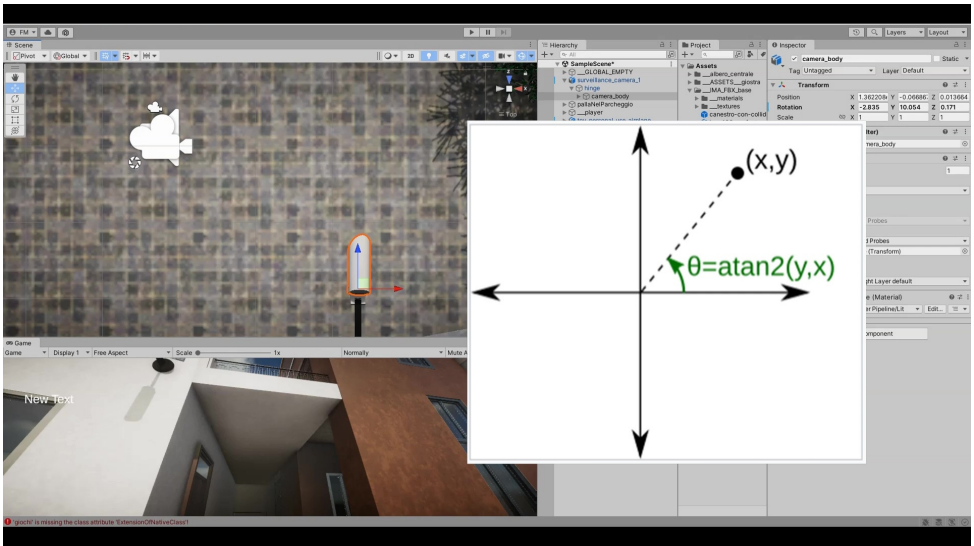


This PDF is available for free; feel free to share it!

[Subscribe to my YT channel](#) & turn on notifications to get updates on new videos!



Hello everyone!

This is the second of two tutorials on creating a "**Look At**" script in Unity 2022 to make an object point towards another one.

Specifically, in the first part we analyzed the problem and implemented a solution that involved **an instant rotation of the object**; in this tutorial, however, we will create a slower and more gradual rotation, using **Quaternions** and **Slerp**.

In this tutorial, I will assume you are familiar with concepts such as **Time.deltaTime** and the **Quaternion.Slerp** method, as I have covered them in other tutorials; if you are not familiar with these topics, I recommend checking out those tutorials first.



In the previous tutorial, we discussed the **global and local axes** of the camera, both to identify the axis that indicates its orientation and to indicate the axis around which to perform rotations.

We then reduced the problem to the two-dimensional plane and calculated **the rotation angle** around which to rotate the 3D model of the Surveillance Camera.

In our script, this angle is called ***deltaAngle*** and is measured in **degrees**.

To use ***Slerp***, we need three things:

- the quaternion that indicates the initial orientation;
- the quaternion that indicates the target orientation to be reached;
- a method to indicate, at each frame of the game's execution, at what point we are (in a range from 0 to 1) in transitioning from the initial orientation to the final one.

The first point is the simplest: **the rotation property** of transform, in fact, expresses the **object's orientation in quaternions**, so for this field we will use ***transform.rotation***.

The second point is not that complicated either: given an initial orientation expressed in quaternions, the quaternion that expresses **the target orientation** obtained **by rotating the initial one** is given by **the product** of the initial orientation and the rotation to be performed.

The rotation to be performed, in our script, is ***deltaAngle***, which is expressed in degrees, though: we need it to be expressed as a Quaternion...

Fortunately, the ***Euler*** function of the ***Quaternion*** class is just what we need: it takes three parameters (i.e., the rotation angles around the X, Y, and Z axes) and returns a quaternion that represents this transformation.

As mentioned earlier, to obtain the target quaternion we need to **multiply** the initial one by the rotation one, so in Update I can write:

```
Quaternion targetRotation = (transform.rotation * (Quaternion.Euler(0f,  
0f, deltaAngle))));
```

Right after, I can update the object's orientation by assigning the quaternion calculated by ***Quaternion.Slerp*** to ***transform.rotation***, with:

```
transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation,  
evaluationTime);
```

Obviously, now we need to define ***evaluationTime***: a float value, between 0 and 1, that represents the point at which we are in transitioning from the initial orientation to the final one.

First, I define and initialize this variable by writing:

```
private float evaluationTime;
```

at the beginning of the script class and:

```
evaluationTime = 0f;
```

inside the ***Start*** method.

```
public class LookAtObject : MonoBehaviour
{
    [SerializeField] private Transform objectA; // Inspector
    private float evaluationTime;

    // Start is called before the first frame update
    // Unity Message | 0 references
    void Start()
    {
        evaluationTime = 0f;
    }
}
```

The ***evaluationTime*** variable must then be **increased at each frame**, which can be done by writing:

```
evaluationTime += Time.deltaTime;
```

right after the instruction containing ***Slerp***.

This is, of course, just an example; my advice is to **multiply *Time.deltaTime* by a variable**, whose value can be adjusted in real-time in the Inspector, until an optimal rotation speed is achieved. I discussed this topic in my tutorial on ***Time.deltaTime***.

However, in this way, at some point ***evaluationTime*** will reach 1.0, as it is incremented at each game frame without a reset option...

... but when should we reset it?

The question can be posed this way: **when are we ready to start a new rotation of the model?**

The answer is: **when the model is already pointing at the Player**, waiting for them to move.

Normally I would tell you that when the Surveillance Camera model points at the Player, and the Player remains still, then ***deltaAngle*** equals 0, because there is no rotation to perform; however, with float values, especially in case of conversions between different units of measure, it is better to maintain **a certain margin**, so we will say that **the animation is finished** (and a new one can start as soon as the Player moves) **when the absolute value of *deltaAngle* is less than 1f**.

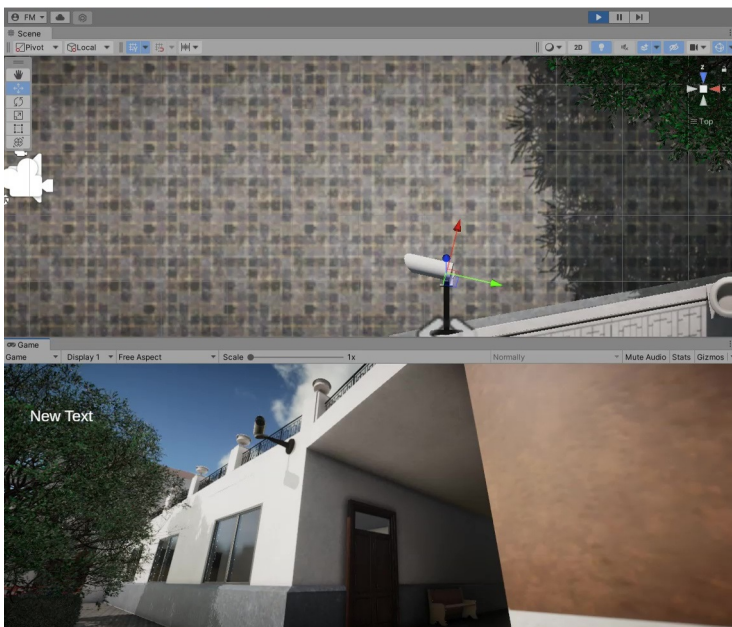
So, right before the creation of ***targetRotation***, inside the ***Update*** method, I write:
if(Mathf.Abs(deltaAngle) < 1f) evaluationTime = 0f;

then I put the following three instructions (i.e., the definition of the target quaternion, the orientation change with *Slerp*, and the increment of *evaluationTime*) inside the else block of this if statement.

UNITY - Look At script: pointing an object towards another; part 2: *Slerp*
www.francescomilanesi.com --- 2024 --- video: <https://youtu.be/jxFrrqT5FIk>

```
17 // Update is called once per frame
18 // Unity Message | 0 references
19 void Update()
20 {
21     float theta = Mathf.Atan2((objectA.transform.position.x - transform.position.x), (objectA.transform.position.y - transform.position.y));
22
23     float currentAngle = transform.eulerAngles.z;
24
25     float deltaAngle = Mathf.DeltaAngle(currentAngle, theta); // in degrees
26
27     // transform.Rotate(Vector3.forward, deltaAngle);
28
29     if (Mathf.Abs(deltaAngle) < 1f) evaluationTime = 0f;
30     else
31     {
32         Quaternion targetRotation = (transform.rotation * (Quaternion.Euler(0f, 0f, deltaAngle)));
33         transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, evaluationTime);
34         evaluationTime += Time.deltaTime;
35     }
36 }
```

I save the modified script, return to the Unity editor, and press **Play** to observe the result. In this case, the rotation speed is determined exclusively by *Time.deltaTime* and can be decreased or increased, as previously mentioned, by multiplying *Time.deltaTime* with a variable to be calibrated interactively.



So, to recap: in this tutorial, we have seen **how to convert the rotation angle** into Quaternions to calculate the Quaternion that represents the target orientation, and then **we used *Slerp*** to perform slower rotations, whose speed can also be

calibrated while keeping it independent of the execution platform, using *Time.deltaTime*.

We also defined a **reset condition for *Slerp*** so that the rotation restarts when the Player is stationary and ready to move again.

Ok, that's all for these two tutorials! See you soon!
