VR MP2.2 Tips

September 13, 2018

1 Transforms Review

Transforms encode a conversion between coordinate frames. While their underlying mathematical implementation may vary, they will generally encode scale, translation (position), and rotation (orientation). We'll forego the discussion of scale for this assignment.

We will define $T_{b,a}$ as the transformation from coordinate frame a to coordinate frame b. This convention is in-line with the way that transforms are composed (applied). For example, if we have two transforms, $T_{b,a}$ and $T_{c,b}$, which represent the transitions between coordinate frame a to b and b to c respectively, then we can get the transition between coordinate frame a to c ($T_{c,a}$) as:

$T_{c,a} = T_{c,b} * T_{b,a}$, where "*" denotes a general transform-composition operator.

2 Transforms in Unity

While sometimes in class we've discussed the representation of Transforms as Homogeneous Transformation Matrices, Unity doesn't encode them this way - and most engines don't (Homogeneous Transformation Matrices are subject to Gimbal Lock).

In Unity, transforms contain a vector for translation and a quaternion for orientation. The transform-composition operation is slightly more complicated than a simple matrix multiplication, but you don't have to derive it for this MP (because we ask you to disable position and rotation tracking separately).

3 Tips

When VR is enabled in Unity, the Transform of the GameObject with the Camera component represents $T_{Room,HMD}$ - where the HMD is relative to its

tracking space. This value is controlled by Unity. Any attempts to write to it - before or during runtime - will be ignored. This means it's effectively a constant.

How do we move the player around in the world, then? If we make the GameObject with the Camera component the child of another GameObject ("parent"), any modifications to the parent's transform will affect the child in the following way:

$T_{world,child} = T_{world,parent} * T_{parent,child}$

In the context of moving the HMD, think of the parent object's transform as the transformation from the room's coordinate frame to the world coordinate frame. That is,

$T_{world,HMD} = T_{world,room} * T_{room,HMD}$

Because $T_{world,HMD}$ is a composition of transforms, it can't be modified directly. And in the context of this assignment, it's a constant anyway (when position tracking is disabled, you want the position encoded by $T_{world,HMD}$ to remain constant; when rotation tracking is disabled, you want the encoded rotation to remain constant).

The only value left to modify is $T_{world,room}$. The solution to this problem then becomes:

Modify $T_{world,room}$ in response to $T_{room,HMD}$ such that the position/rotation of the composition $T_{world,HMD}$ remains constant.

To clarify this further, when either tracking feature is disabled, the player should not snap to the origin, but rather remain as they were when they disabled tracking. For position tracking, they should not snap to world position (0, 0, 0); for orientation tracking, their orientation should not immediately become the identity quaternion.