frontiers | Frontiers in Virtual Reality

TYPE Technology and Code
PUBLISHED 15 September 2022
DOI 10.3389/frvir.2022.937191

QuickVR: A standard library for virtual embodiment in unity

Ramon Oliva[1], Alejandro Beacco[1], Xavi Navarro[1] and Mel Slater[1,2]*

[1]Event Lab, Faculty of Psychology, University of Barcelona, Barcelona, Spain, [2]Institute of Neurosciences of the University of Barcelona, Barcelona, Spain

Check for updates

OPEN ACCESS

EDITED BY
Nadia Magnenat Thalmann,
Université de Genève, Switzerland

REVIEWED BY
Henrique Galvan Debarba,
Aalborg University Copenhagen,
Denmark
Jérémy Lacoche,
Orange (France), France

*CORRESPONDENCE
Mel Slater,
melslater@ub.edu

SPECIALTY SECTION
This article was submitted to Technologies for VR, a section of the journal Frontiers in Virtual Reality

RECEIVED 05 May 2022
ACCEPTED 19 August 2022
PUBLISHED 15 September 2022

CITATION
Oliva R, Beacco A, Navarro X and Slater M (2022), QuickVR: A standard library for virtual embodiment in unity. *Front. Virtual Real.* 3:937191. doi: 10.3389/frvir.2022.937191

In the last few years the field of Virtual Reality (VR) has experienced significant growth through the introduction of low-cost VR devices to the mass market. However, VR has been used for many years by researchers since it has proven to be a powerful tool across a vast array of research fields and applications. The key aspect of any VR experience is that it is completely immersive, which means that the virtual world totally surrounds the participant. Some game engines such as Unity already support VR out of the box and an application can be configured for VR in a matter of minutes. However, there is still the lack of a standard and easy to use tool in order to embody participants into a virtual human character that responds synchronously to their movements with corresponding virtual body movements. In this paper we introduce QuickVR, a library based on Unity which not only offers embodiment in a virtual character, but also provides a series of high level features that are necessary in any VR application, helping to dramatically reduce the production time. Our tool is easy to use by coding novices, but also easy extensible and customizable by more experienced programmers.

KEYWORDS

virtual reality, virtual environments, virtual reality software, virtual reality systems, embodiment, body ownership, avatar tracking, unity

1 Introduction

Virtual Reality (VR) is a well-established technology that has been used for many years in many research and application fields. However during the last decade we have witnessed VR becoming a popular platform for the game and entertainment industry. Reduction in cost but also the development of lightweight, high resolution and relatively simple to use Head Mounted Displays (HMDs) and other VR devices are moving this technology from being something exclusive to industry and specialized VR labs to an affordable system by the average consumer, certainly less expensive than many smartphones. There is currently a wide variety of VR devices in the market that cover different needs, from systems that are connected (physically wired or not) to a powerful VR-ready computer offering the most demanding and visually pleasant VR experiences, to other fully portable but less powerful solutions.

When the participant puts on the HMD, the real physical world is no longer seen (and maybe not heard) and the participant is completely surrounded by a 3D computer generated Virtual Environment (VE) that does not necessarily follows the same rules as physical reality. For example, we can modify the illusion of gravity in the virtual world, how the different physical forces interact, or create situations and effects that are simply not possible in the real world. We can obtain the head position and head gaze direction from the HMD in real time to dynamically update the point of view of the participant inside the VE, so the participant has the feeling of being immersed in the simulation. The VR system with six degrees of freedom head-tracking and at least one hand affords sensorimotor contingencies for perception, at least in the visual field, approximately the same as perception in physical reality. This typically leads to the illusion of being in the virtual world, usually referred to as *presence* (Sanchez-Vives and Slater (2005); Sheridan (1992); Slater (2009)).

But in VR we can go a step further as we can also replace the participant's real body by a *virtual body* (or 'avatar'), a synthetic humanoid representation of the participant in the VE. We use the term *Virtual Embodiment* (or simply *Embodiment*) to describe the process involving the necessary VR hardware and software to that end. When we use the data obtained by the different VR devices attached to the participant's body to make the avatar move synchronously with the participant's real movements, the *Body Ownership Illusion* (BOI) and *Agency* may arise (Kilteni et al., 2012). The BOI is when participants have the illusion that a virtual body or body part is their own and that it is the source of their sensations (Tsakiris, 2010). By *Agency* we mean that the person recognizes themselves as the cause of the actions and movements of the virtual body. Hence, the participant can see the VE from the first-person perspective (1PP) of the virtual body but also interact with virtual objects that only exist in that alternative reality, integrating the participant as being part of the simulation instead of a mere observer.

The illusions of presence, BOI and agency make VR a very powerful tool to carry out different experiments, for example, in the fields of neuroscience and psychology in a secure and controlled environment. VR has been shown to be an effective way to address different fears and phobias such as fear of public speaking (Slater et al., 2006) and fear of heights (Emmelkamp et al. (2002); Freeman et al. (2018)) and its positive effects may persist in the long term (Coelho et al., 2006). However, the real potential of VR is that we are able to put participants in situations that are simply not possible in the real world. For example (Peck et al. (2013); Banakou et al. (2016)), show that VR can be used to reduce implicit racial bias by embodying light skinned participants in a dark skinned avatar. Also, the avatar we are embodied in may change how we perceive the VE as demonstrated in (Banakou et al., 2013), where results show that participants tend to overestimate the size of the objects when they are embodied as a child. In (Seinfeld et al., 2018), male gender violence offenders were embodied in a female virtual body and they experienced verbal aggression from a virtual male from the 1PP of the woman. VR has been used also for pain experiments and to study how changing some properties of our virtual body may affect our own pain perception (Martini et al. (2015); Matamala-Gomez et al. (2019); Matamala-Gomez et al. (2020)). An extreme use of VR can be found in (Barberia et al., 2018), where participants experienced a complete life cycle with sessions every day over several days, culminating in a near death experience.

Game Engines such as *Unity* (Unity Technologies, 2021b) and *Unreal* (Epic Games, 2021) are widely used for general game development on multiple platforms. They offer a consistent solution to produce rich VEs with high quality graphics, physics, animations, particles, sound, networking and in short, a complete set of tools needed in any commercial game. Additionally, we can extend its capabilities with many plugins and assets or create our own to enrich our applications. Those engines are being used also by academic VR research groups due to the fact that they support most of the current VR devices either natively or by installing the corresponding plugin, so we can see our environment in many VR displays with almost no extra effort.

In order to obtain a complete immersive VR experience with full body tracking, we may need to combine the input of different tracking systems. In (Spanlang et al., 2014) the technical infrastructure needed to construct a VR lab for embodiment is discussed. They describe the hardware that they used but also a generic modular solution at the software level is presented for studying BOIs, combining the input coming from distinct VR devices. The library we present in this paper applies and extends such concepts, offering a fully functional implementation in *Unity* which potentially works on any hardware natively supported by this game engine.

There exist some packages and tools that extend the capabilities of *Unity's* VR built-in system, such as the *XR Interaction Toolkit* (Unity Technologies, 2021c) and *VRTK* (Extend Reality Ltd, 2021) by adding other functionalities such as teleporting and interacting with the environment and the elements of the user interface. However just a few tools address the problem of embodying in a virtual avatar. An example is *AvatarGo* (Ponton et al., 2022). However it currently only works with *SteamVR* compatible devices such as *HTC Vive*. Instead, *QuickVR* supports any device out of the box that implements the *XR Unity Framework* (Unity Technologies, 2021d).

Some other tools for creating VR experiments in *Unity* have been recently proposed (Watson et al. (2019); Brookes et al. (2019); Bebko and Troje (2020)) that simplify the process of writing the logic of the experiment and variable manipulation. This is often a time consuming and repetitive task that is common to most VR experiments. However, they do not offer a built-in solution to produce BOIs. To the best of our knowledge, our library is the first toolset in *Unity* that offers a cross platform solution for applications requiring embodiment

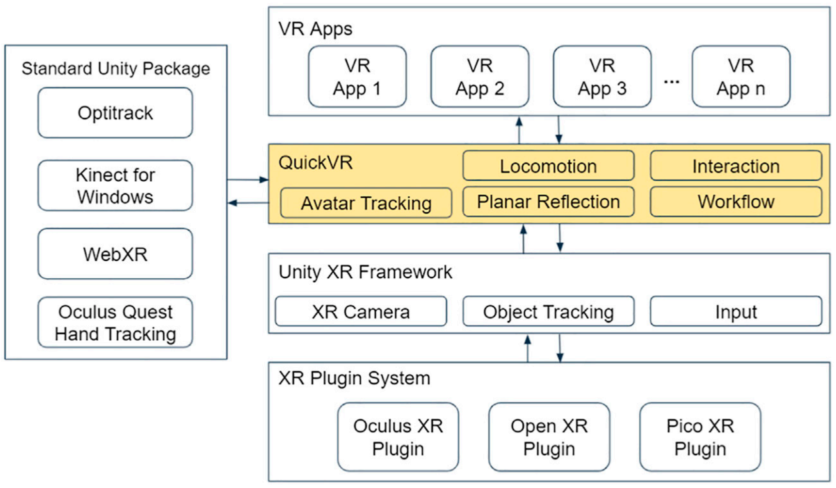


FIGURE 1
QuickVR is built on top of Unity's XR Framework, thus natively supporting any XR device implementing that interface. QuickVR can be also extended to support other XR devices and features provided through Standard Unity Packages. Our library is focused on supporting high level features commonly found in any VR application, whilst the Unity XR Framework covers the lower level details that are common in any XR system.

in VR, but also proposes a methodology that simplifies the logic coding and other helpful utilities useful in any VR application.

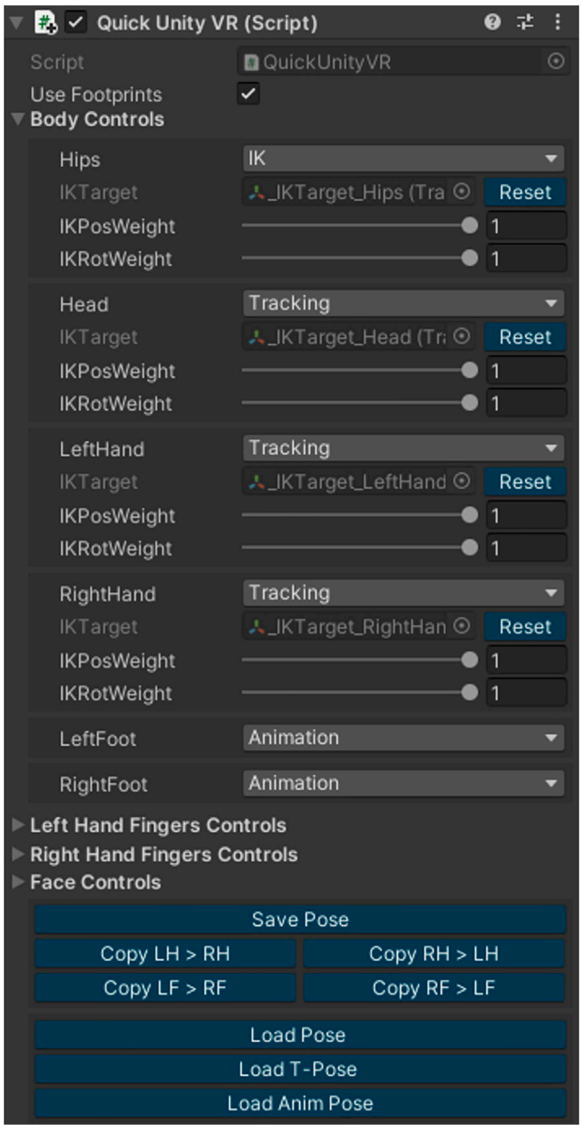
2 Overview

QuickVR is the result of the technical research and development in Virtual Reality that we have carried out over many years, and offers a series of tools to quickly develop your own VR applications with Unity. The main target is to be able to reuse as much code as possible from our past applications. Therefore, QuickVR is not something fixed or closed, but it keeps constantly evolving responding to the new requirements of the applications that we develop and the new features that Unity introduces. It is created also taking usability into consideration, so it is straightforward for anyone to start using it with a basic knowledge of Unity, as well as being easy to extend and add new functionalities and customize the existing ones.

As depicted in Figure 1 QuickVR is built on top of the Unity XR (VR/AR/MR) Framework, which is an abstraction that exposes the common functionalities of all the XR devices natively supported by Unity. It does not need to know about each specific implementation of a specific XR provider, but it acts directly on the common framework. The Unity XR Framework is a relatively new addition in the engine (present as a stable feature since version 2019.3), and nowadays it is the normative way for supporting XR devices so XR providers are advised to develop their plugins using the Unity XR Framework standard. This way, we only need to install the corresponding plugin for the desired XR device and it will be automatically recognized and usable by Unity out of the box and by extension, by QuickVR. So the core principle of Unity engine "Build once, deploy anywhere" is kept by design.

However, other XR providers do not supply such a plugin but they provide a Standard Unity Package, either because they are old devices that appeared before the introduction of the Unity XR Framework, as in the case of Kinect for Windows, or for some other technical reasons. There are also some specific features for some devices that are not supported yet through the Unity XR Framework. This is the case for the Oculus (now Meta) Quest family, where the HMD and the controllers are supported through the Oculus XR Plugin while the Hand Tracking feature is currently supported using a Standard Unity Package also provided by Oculus. In any case, QuickVR is designed in a way that can be easily extended to support such devices and features.

Whilst the Unity XR Framework is more focused on low level functionality such as stereo rendering, device tracking and input management, QuickVR is focused on higher level features that are also necessary in our developments, with body tracking the most important one, but also additional capabilities such as planar reflections, logic workflow management, locomotion or interaction with the environment. So the different applications that are built using QuickVR, have access to all those features already implemented. Without this library, those features would need to be implemented from scratch on every new application, which would increase the production time dramatically. With this approach, we can have a prototype of a new VR application in a matter of hours to days, depending on its complexity.



Quick Unity VR (Script)

Script: QuickUnityVR
Use Footprints: ✓
Body Controls
- Hips: IK
  - IKTarget: _IKTarget_Hips (Tra — Reset
  - IKPosWeight: 1
  - IKRotWeight: 1
- Head: Tracking
  - IKTarget: _IKTarget_Head (Tr — Reset
  - IKPosWeight: 1
  - IKRotWeight: 1
- LeftHand: Tracking
  - IKTarget: _IKTarget_LeftHand — Reset
  - IKPosWeight: 1
  - IKRotWeight: 1
- RightHand: Tracking
  - IKTarget: _IKTarget_RightHan — Reset
  - IKPosWeight: 1
  - IKRotWeight: 1
- LeftFoot: Animation
- RightFoot: Animation

Left Hand Fingers Controls
Right Hand Fingers Controls
Face Controls

FIGURE 2
The *QuickUnityVR* component is used to achieve embodiment in an avatar. We can specify how we apply the tracking data to each body part using the exposed controls, which are classified into different categories. In this example, the upper body is driven by the *Tracking* data, the hips are driven by the *IK* and the feet are driven by the *Animation*.

3 Using QuickVR

QuickVR is publicly available and free for non-profit and research projects.[1] The main page of the GitHub repository contains the source code and the instructions to install it into your Unity project. It is installed as any other Unity package. On the *Wiki* section you will find the instructions needed to prepare your project depending on your target device (*Meta Quest*, *Pico Neo 2/3*, *OpenXR*, . . .). Next there are a series of tutorials of increasing difficulty that progressively introduce the main features of the library and how to use them through practical examples. Finally there is an in depth description of such features and how to enable other advanced features for specific devices, like *Hand Tracking* for *Meta Quest* and *Eye Tracking* for the *Pico Neo Eye* devices.

The main feature provided by *QuickVR* is to be embodied in a virtual humanoid body, that we refer to as the *Avatar*, moving it synchronously (or asynchronously if required) with the movements of the real body and perceiving the scene from the point of view of this virtual surrogate body. Acquiring this is straightforward, we just need to add the component *QuickUnityVR* (see Figure 2) to an avatar that has been previously configured as a *Humanoid*, so that it has two legs, two arms and a head. Any *Humanoid* avatar is compatible with *QuickVR*, no matter the source (*Mixamo*, *Fuse*, *Character Creator*, *Rocketbox* . . .).

When the application starts, the system needs to be calibrated, i.e., we need to obtain some data in order to correctly map the real body pose of the participants to their avatars. In our case, we only require the participants to look forward (their forward direction in the real world) and the system is ready to go. We assume that the participant is embodied in an avatar of similar height and proportions. Embodiment into avatars with different shapes and proportions is discussed in section Update Copy Pose. Then a custom *Inverse Kinematics* algorithm is used to compute the pose of each defined kinematic chain. We can modify the update mode of each kinematic chain by using the controllers exposed by the *QuickUnityVR* component:

- *Tracking*: This is the default update mode. The kinematic chain is updated with the tracking data provided by the VR system being used. If the target VR system does not provide tracking data for that body part, this mode behaves as in *IK* mode.
- *IK*: The IK target is driven by the application. The developer manually sets the target to the desired position and orientation.
- *Animation*: The pose of that kinematic chain is driven completely by the animation, if any.

In both *Tracking* and *IK* modes, we can determine the weight of the IK system on that bone by tweaking the *IKPosWeight* and *IKRotWeight* attributes, which defines the influence of the IK target to determine the position and rotation respectively of that specific kinematic chain. Those values are in the range [0,1], where 0 is completely driven by the animation and 1 is completely driven by the IK system. The fingers for the left

1 https://github.com/eventlab-projects/com.quickvr.quickbase

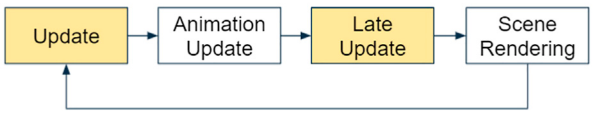


FIGURE 3
Unity's game loop. The different scripts can be registered to the *Update* or *LateUpdate* blocks depending on whether we want to execute that logic before or after the *Animation Update* block has been processed. The *Scene Rendering* block is the last step in the game loop and renders the scene from the defined virtual camera.

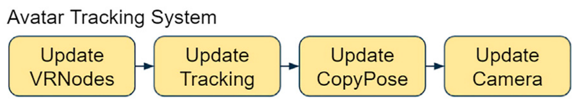


FIGURE 4
The *Avatar Tracking System* is divided into the following steps: *Update VRNodes*, *Update Tracking*, *Update CopyPose* and *Update Camera*. Before and after each one of these steps, an event is triggered where one can register in order to execute custom logic before or after any of such steps.

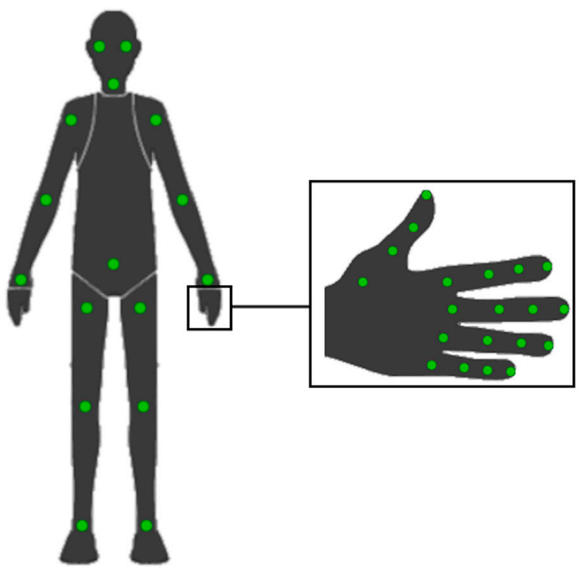


FIGURE 5
The green dots represents the available tracked points (*VR Nodes*) in a human body in *QuickVR*. There are tracking points for the main body parts: head, hips, wrists, elbows, shoulders, knees and feet. In the case of the hands, we have up to four tracking points on each finger. The points effectively tracked will depend on the VR systems that we are using.

and right hand can be controlled the same way using the corresponding finger control exposed in *Left Hand Fingers Controls* and *Right Hand Fingers Controls* and we will be able to move the fingers of the avatar with our own fingers if the VR devices supports finger tracking. Finally, The controllers for the eyes are found in the section *Face Controls* and we can manage the gaze direction and eye blinking of our virtual counterpart using such controls.

Our IK solution implements a basic two bone IK solver that is applied to each kinematic chain in the same order that can be seen on Figure 2. This guarantees that the bone parenting is respected as the final bone position of a specific chain will depend on the position of the previous bones in the *Humanoid* skeleton. We have decided to implement our own IK solver because we need a very fine control on when the IK step is executed.

4 Implementation

In Unity the logic update loop is distributed into two main parts, the *Update* and the *LateUpdate*. When creating a new Unity logic script, one can define an *Update* and/or a *LateUpdate* function that will be automatically called on the corresponding logic block at each frame. In between these two blocks there is the *Animation Update* where basically the animation graph is processed and the resulting animation is applied to the animated characters. Finally, Unity proceeds to render the virtual scene in the *Scene Rendering* block. This loop is repeated every frame until the application finishes (see Figure 3). In the following sections we explain in detail the main systems of *QuickVR* and how they are distributed inside the Unity logic loop.

4.1 Avatar tracking system

This is the core system of *QuickVR*, which covers the embodiment into a virtual humanoid avatar. The *Avatar Tracking System* occurs in the *LateUpdate* block, after the animation is processed and it is subdivided into different steps as depicted in Figure 4. Before and after the execution of each one of these steps, an event is triggered where you can register to execute your own code. For example, the *UpdateVRNodes* will trigger the *OnPreUpdateVRNodes* event before its execution and the *OnPostUpdateVRNodes* once it is finished.

4.1.1 Update VR nodes

The *VR Nodes* are all the possible tracked points in a human body exposed by *QuickVR* as depicted in Figure 5. In this step, we read the tracking data from the VR system and apply it to the corresponding *VR Node*. On the most common VR devices, we have an HMD and a pair of hand-held controllers, so the position and rotation of the *VR Node* in the head is obtained from the HMD, whilst the data of the *VR Nodes* on the wrists is obtained from the controllers. However, the number of tracked *VR Nodes* as well as the data we can obtain on each will depend on the VR devices being used. For example, if we are using *Meta Quest* we can track the fingers. With *Pico Neo two Eye* we can feed our system with eye tracking data such as the gaze direction and blinking information. In the case of the *VIVE*, we can attach a *VIVE Tracker* to any part of the body of the participant to obtain the position and rotation of that specific body point. Finally, it is possible also to combine different VR systems. For example, we can use a generic HMD combined with *OptiTrack* to get an immersive experience with high precision full body tracking, or use the *Kinect* system instead for a more portable and economic solution although the body tracking quality will not be so high.

4.1.2 Update Tracking

In this step we apply the tracking data acquired in the previous stage to an avatar. Its body pose is estimated by using a custom Inverse Kinematics (IK) system which adapts to the specific *VR Nodes* that are effectively tracked by the VR devices currently being used. More specifically, the upper body is estimated using the data on the head and wrists nodes. The lower body pose is estimated by the feet nodes if such data is available on the VR system. Otherwise, the position and rotation of the feet are driven by the application. The position of the virtual elbows and knees can be either retrieved from the corresponding nodes if those body points are tracked, otherwise they will be estimated in a postprocessing stage by the IK avoiding weird and impossible positions. This way, our VR library offers a flexible solution that works on any VR platform, ranging from mobile to desktop, and adapts to the different number of tracked nodes exposed by each VR platform.

Note that the same build can be used for the same platform with different numbers of tracked devices. As an example, consider the case A where an *HTC Vive* is used for tracking the head and wrists and two extra trackers are attached to the feet. On the other hand, case B only uses the *HTC Vive*. In both cases, we use the same build, the only difference is that in case A the feet pose will be extracted from the tracking data provided by the trackers on the feet, whilst in case B the feet pose will be estimated by applying an animation that takes into account the displacement of the HMD. The available tracking devices are automatically detected, so this is transparent to the programmer.

4.1.3 Update Copy Pose

In many of our applications we are required to be embodied in different avatars throughout the virtual experience. The most immediate solution would be to add a *QuickUnityVR* component to each of the virtual characters in which we need to be embodied. However, this approach requires a calibration of the system each time we switch from one character to another, as each character may differ in size and proportions. Moreover, the character may differ considerably in size with respect to the real participant, which can deal to visual artifacts when applying the IK system directly to that character. Thus we require only one calibration of the system, when the application starts, and to be able to embody the participant in any kind of virtual *Humanoid*, independently of its size.

In this situation, we define two types of avatar, the *Master Avatar* and the *Target Avatar*. The *Master Avatar* is a standard avatar ideally with a similar size as the real participant. The more accurate the match between the size of this avatar and the real participant, the better the result will be when applying the IK system. Therefore, the *Master Avatar* is the one we attach the *QuickUnityVR* component to which receives the tracking data from the different VR devices. The *Target Avatar* copies the full body pose from the *Master Avatar* after the tracking data has been applied, basically by transforming the rotations of each bone from the *Master Avatar* coordinates system to the *Target Avatar* one. This way, the *Target Avatar* can have an arbitrary size and proportions, contrary to the *Master Avatar*. In Figure 6 we can see that the body pose of the real participant is correctly mapped to distinct avatars with different body sizes and proportions.

Of course this solution is not perfect and there is room for improvement. For example, if participants are embodied in an avatar with a size or shape very different to how they really look and they try to touch the virtual body, visual artifacts may arise in the form of joint intersections or endpoints on the IK chain not reaching the contact point. In order to mitigate this, we are considering the introduction of an extra IK step as a post process after the pose is copied to the *Target Avatar*, to account for self intersections as well as intersections with the environment, and correct the position of the joints accordingly.

Also the mapping quality depends on how good the match is between the real participant and the *Master Avatar* as stated before. But even if the *Master Avatar* is not exactly the same size as the participant (by default we use a standard avatar as the master of about 1.7 m tall), it works pretty well in the vast majority of cases as we can extract from the empirical observation of our results in the body ownership illusion related questions that we have used in many studies, some of these referenced in section Results. For the most accurate mapping, we would need to retrieve more data during the calibration process such as forcing the participant to be in a specific position at the start (like the T-Pose). However, we consider that this can only be guaranteed under laboratory conditions, with the supervision of an experimenter. If the participants run the application at home without supervision (as we are currently doing in some experiments), we cannot guarantee that they will do the calibration process correctly.

4.1.4 Update Camera

The final step of the loop simply sets the virtual camera to match the point of view of the *Target Avatar*, i.e., the camera is set

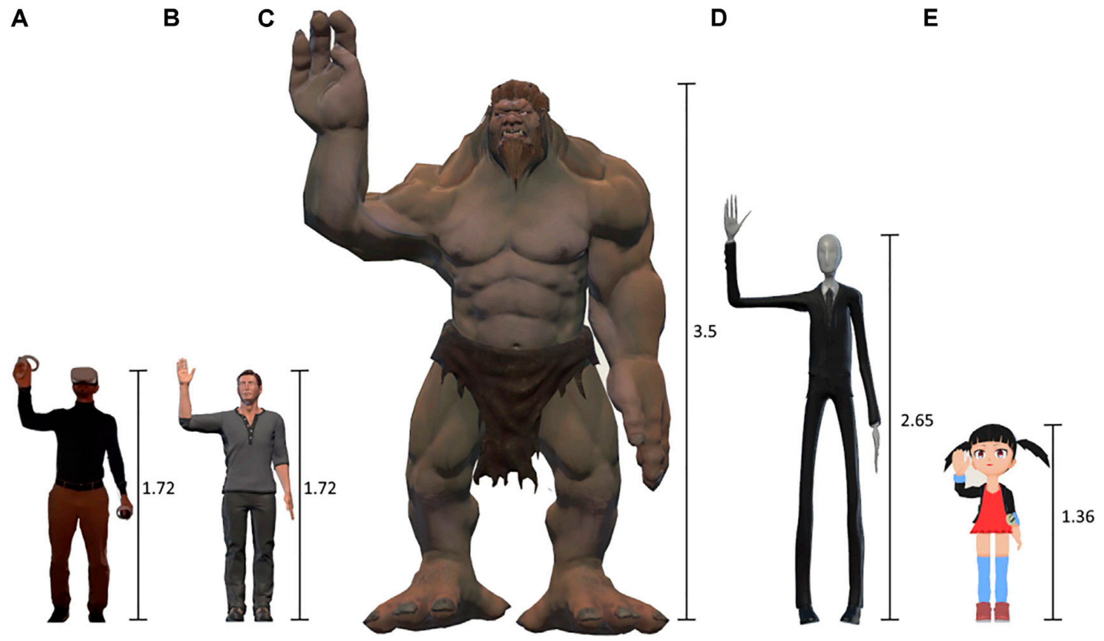


FIGURE 6
The real participant **(A)** and the *Master Avatar* **(B)**. The pose of **(A)** is applied to **(B)** and then it is copied to different *Target Avatars*: A giant **(C)**, a skinny and very tall humanoid figure **(D)** and a cartoon-like girl **(E)**. The vertical axis next to each figure represents the height in meters.

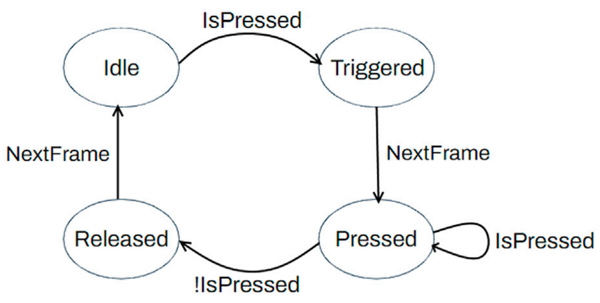


FIGURE 7
The possible states of a *Virtual Button*: *Idle* is the initial state. In the first frame that the button is pressed, it goes to the *Triggered* state. The button is in *Pressed* state during all the subsequent frames that it is pressed. The state *Released* is produced the first frame that the button is not pressed anymore, and on the next frame, it returns to the *Idle* state.

in the midpoint of the eyes and matches the head orientation. This process is delayed as much as possible in the update loop in order to get the best match between the virtual positioning of the camera and the physical participant's head pose.

4.2 InputManager system

The *InputSystem* by Unity already accounts for the most common input devices such as keyboard, mouse, gamepads and those XR controllers implementing the XR Framework such as the *Oculus/Meta* or *VIVE* controllers. The *InputManager* from QuickVR works on top of that and extends its capabilities. It defines a set of *Virtual Axes* and *Virtual Buttons*. A *Virtual Axis* is an input type that returns a value in the range [−1,1], whereas a *Virtual Button* is an input type that can have four possible states: *Idle*, *Triggered*, *Pressed* or *Released*. Figure 7 shows the state graph of a *Virtual Button*.

We can define any number of *Virtual Axes/Buttons* and then for each available *InputManager* implementation, we just need to map each *Virtual Axis/Button* to a specific axis/button exposed by that implementation. QuickVR offers an *InputManager* implementation for the most common types of input, but you can also easily create your own implementations to account for hand gestures, body postures, eye gaze movement and in short, anything that you consider that can be translated to a *Virtual Axis/Button* and use it to control the input of the application. When we are writing the logic of our application and we want to access to the input on a script, we just need to access using the corresponding accessor function using the *Virtual Axis/Button* name as follows:

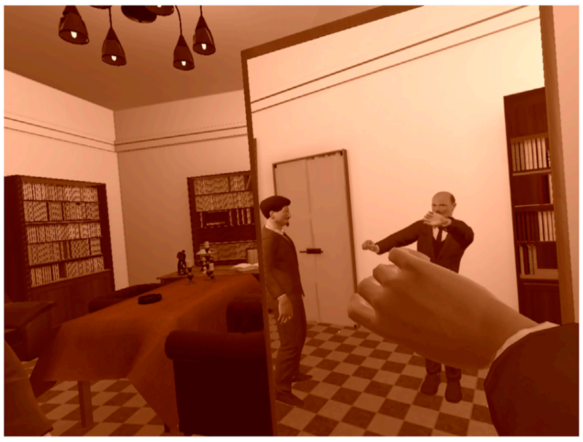


FIGURE 8
In many of our experiments, there is an initial stage where the participant is instructed to do some exercises in front of a virtual mirror to see how the virtual avatar moves as his real body. This greatly improves the BOI. Figure from (Slater et al., 2018).

- *InputManager.GetAxis(virtualAxis)*: Returns the real value in the range [-1,1] of any input device axis mapped to *virtualAxis*.
- *InputManager.GetButton(virtualButton)*: Returns true if any input device button mapped to *virtualButton* is on the *Triggered* or *Pressed* states; false otherwise.
- *InputManager.GetButtonDown(virtualButton)*: Returns true if any input device button mapped to *virtualButton* is on the *Triggered* state; false otherwise.
- *InputManager.GetButtonUp(virtualButton)*: Returns true if any input device button mapped to *virtualButton* is on the *Released* state; false otherwise.

The state of the *Input Manager* is updated inside the *Update* logic block in the Unity game loop, and it occurs before any other *QuickVR System* is executed. This way we guarantee that the input state is consistent thorough the same frame.

4.3 Mirror system

In our research we find that having virtual mirrors in the scene are either necessary (because we want participants to see what they look like) or helpful for the BOI (see Figure 8). *QuickVR* has a built-in system for virtual mirrors on planar surfaces. A mirror is defined by a 3D quad mesh that represents the reflection plane, whose equation is represented by the normal vector and the world space position of the quad. An additional virtual camera is created that takes the position and orientation of the participant's camera reflected by the plane equation. A render to texture is done from the point of view of the reflected camera, and this texture is then applied to the mirror's mesh. This process is carried out right after the *Update Camera* step and before the scene is rendered.

It has to be taken into consideration that having virtual mirrors in the scene is a costly process, since each mirror implies an extra render of the scene. This is specially important in VR experiences since internally each camera render is subdivided into a render for each eye position for stereo. Therefore, having a single mirror in VR implies that the scene is rendered 4 times (2 times from the point of view of the *Target Avatar* and two additional times from the reflected camera position).

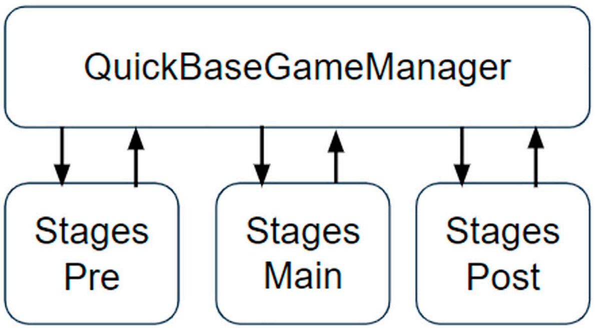


FIGURE 9
The workflow of a QuickVR application is subdivided into the following components: The main of the application (*QuickBaseGameManager*) and three logic blocks called *StagesPre*, *StagesMain* and *StagesPost*.

4.4 Workflow system

The QuickVR library offers a tool to easily create and manipulate the workflow of an application. The entry point, the 'main' of the application is defined by the *GameObject* containing the *QuickBaseGameManager* component. Its purpose is to initialize and configure the application and it automatically manages the logic flow between the different groups of *Stages*. A *Stage* (named *QuickStage* following the *QuickVR* naming convention) can be seen as each one of the steps into which we can subdivide the logic of our application. Each one of these steps can be arbitrarily complex and it is the developer's decision and responsibility to subdivide the logic into the appropriate number of *QuickStages*. Those logic pieces are grouped together into three different main blocks, as depicted in Figure 9:

- The *StagesPre* contains the logic that has to be executed when the application starts, and before the main logic of the application. Here we typically want to calibrate the existing VR devices or make sure that the HMD is correctly adjusted by displaying some text on the screen, so we make sure that the participant will see the scene clearly.

- The *StagesMain* contains the main logic of the scene, what the application has to do.
- Finally, the *StagesPost* contains the logic that we want to execute just before closing the application. For example, here we want to save some data to the disk or to the cloud that we have acquired during the execution of the application.

Also there is an event triggered before and after the execution of these three main blocks. For example, if we want to do an action just after the *StagesPre* are finished, but before the *StagesMain*, we can do so by registering to the corresponding event. There are some *QuickStages* already defined in *QuickVR* that can be used out of the box and offers different possibilities commonly used in many of our virtual experiences, like fading in and out the scene or additively loading scenes in background. But there are also other types of *QuickStages* that allow the creation of conditional blocks or loops. The *QuickStage* system is highly customizable, and we can inherit from any of the existing stages and override its behavior to adapt it to our specific needs, or create completely new stages by inheriting directly from *QuickStageBase* which is the base stage from which any other stage inherits from. Finally, in the end those stages are simply components contained in *GameObjects* in the scene's hierarchy. This means that we can reorder the logic by simply moving the corresponding *GameObject* in the hierarchy, or enable or disable that specific stage by simply enabling or disabling its *GameObject*.

There are three main functions that we have to take into consideration when creating a new *QuickStage*:

- *Init*: This function is called when the stage starts its execution. You want to set here any logic that is executed one time, when the stage starts.
- *CoUpdate*: This is the main coroutine that you want to override and here is where you usually want to set the logic of this specific stage. When *CoUpdate* ends, the next function *Finish* is automatically called.
- *Finish*: This function is called when the stage finishes and it basically ends the current stage and passes the flow control to the next stage in the hierarchy.

4.5 Locomotion

In some scenarios we require the participant to move through and explore the virtual world. We define the *Tracking Area* as the free space that the participant has in the real world surroundings to freely move without colliding with any physical obstacle. When the virtual world is completely enclosed in the *Tracking Area*, this becomes a trivial problem as we simply use the tracking data from the HMD (or an extra tracker attached to participant's hips for better results if this option is available in the VR systems that we are using) to compute the root displacement and orientation and apply it to the virtual avatar. We refer to this method as *Direct Locomotion*. However, the virtual world tends to be much larger than the physical *Tracking Area*. For this case, *QuickVR* has several built-in solutions, such as *Teleport* and *Walk in Place*.

5 Results

In this section we briefly describe some of the experiments produced in our lab using *QuickVR*. While all use most of the introduced systems to some degree, it is interesting to review them to illustrate how the library has been evolving and changing as new requirements for our experiments arose.

In (Bourdin et al., 2017) we studied two methods for producing an out-of-body experience (OBE) in VR. In both conditions, participants experienced the same initial stage where they had to do some exercises in front of a virtual mirror with their Virtual Body (VB), which moved synchronously with the movements of the real body using the *Avatar Tracking System*. We attached some vibration actuators to the wrists and ankles of the participants that vibrated when some virtual objects touched that part of the virtual body, thus improving the BOI. Then the viewpoint was lifted up and behind the VB while the VB remained in place. In one condition participants had no further connection with the VB, while in the other condition visuomotor and visuotactile synchrony continued. This was the first experiment implemented using *QuickVR* and the fundamentals were introduced here.

The Island (Barberia et al., 2018) is the most complex experiment that we have ever developed. Here we used the potential of VR to study the effects of virtual mortality in relation to life-attitudes. Participants were embodied in a human-like body in a virtual beautiful island along with two remotely located companions. They could explore the world and carry out tasks together that required the collaboration of the three of them to complete the task. The virtual body aged over time and each participant witnessed the death of the two companions and then her own death, which included some of the features of near death experiences (OBE, life review, the tunnel leading to white light). Results showed that this experience had a positive effect on life attitudes of the participants with respect to a waiting list control group, but also an interesting result was that participants were able to transmit the knowledge of how the world worked and how to solve the tasks from generation to generation, even though they could not talk to each other and they learnt by imitation from the older participants. Apart from the core *QuickVR* systems, we developed some specific locomotion metaphors using the full body to move through the environment, suitable for that magical world.

In (Slater et al., 2018) for the centenary of the 1917 Russian Revolution, we reconstructed a famous photograph in VR showing Lenin giving a speech to Red Army recruits in the

Sverdlov Square of Moscow, 1920. Depending on the experimental condition, we could see the scene from three different perspectives: embodied as Lenin, or a crowd member of the Red Army, or from a disembodied floating third person perspective. Here we introduced the *Copy Pose System* as in some conditions we required to be embodied in more than one avatar through the execution of the experiment, but we wanted to calibrate the application only once, at the beginning of the experience. The initial version was executed using an *Oculus Rift* or a *HTC VIVE*. In the latter case, additional trackers could be added to the feet and hips to obtain lower body tracking and get the root displacement of the participant thorough the scene. A recent version for *Quest* has been developed, which makes it fully portable and we can use the finger tracking feature of such device.

In (Neyret et al., 2020) 60 male participants experienced a VR scenario of sexual harassment (SH) of a woman by a group of males in a bar. Participants were equally divided into three different groups: they were only embodied as one of the males committing the aggression, others were also embodied as the woman and suffered the aggression from that 1PP and finally a control group only experienced the empty bar, not the SH. One week later, they took the role of the teacher in a VR version of Milgram's obedience experiment. Participants were encouraged by a group of three other virtual males to give electroshocks to a female learner. Results showed that those who had been embodied as the woman in the previous experience gave about half the number of shocks of those from the first group, and the control group was in between. For the first part of the experiment, we used the potential of QuickVR to work with multiple VR devices. The SH was pre-recorded using *Perception Neuron* for body tracking and the recorded animations were refined in a postprocess stage. Participants experienced both situations in the *HTC VIVE*.

6 Discussion

In this paper we have introduced *QuickVR*, a library for VR embodiment using the Unity engine. The core library is subdivided into different modules or systems, each one tackling a specific problem which includes not only virtual embodiment in VR, but also other high level features such as planar reflections, workflow management or locomotion. Such modules have been added and expanded in response to the requirements that we have had when developing a new experiment as well as the changes and updates that Unity has released during those years. Therefore, this is not a closed API, but a dynamic set of tools that, thanks to its modular design, makes it easy to use for anyone with a basic knowledge of Unity, but also experienced developers can easily add their own modules and extend the existing ones.

The way it is designed makes *QuickVR* a multiplatform tool. It works out of the box with most of the current VR devices that are available in the market. We have successfully developed applications for mobile platforms like *GearVR*, *Meta Quest 1 & two* and *Pico Neo two Eye*. On the desktop side, we have worked with several versions of *Oculus* including the *Oculus DK2* and *Oculus Rift* as well as the *HTC VIVE and VIVE PRO*. Although they are not all the VR devices available on the market, they are the most common and also the way the library is constructed makes it available for any VR device implementing the Unity XR Plugin interface without doing anything else apart from installing the corresponding plugin. We also have additional support for other less common devices that do come as a standard Unity package, such as *Kinect*, *OptiTrack* and *Perception Neuron* and any VR developer can add support for a specific device by implementing an exposed interface by *QuickVR*.

The library is a well proven tool as it has been used and matured for many years internally in our lab to produce our VR applications. Although the library has been developed in a VR research lab, we have developed a general solution that is usable for any type of VR application, especially those requiring embodiment in an avatar. We have recently released it to the public free for use for non-commercial projects. Although *QuickVR* is developed in Unity, the main concepts explained here can be extrapolated to other game engines, thus serving as a starting point for developers looking to construct a similar solution on other engines.

Although the presented work is not something fixed but is continuously evolving as Unity introduces new features, new VR devices are released on the market and we find new use cases for our studies, we consider that we have reached a point where the current structure is quite solid and extensible, and we do not expect that the core architecture and concepts introduced here will radically change on the future. However, for the sake of clarity, the specific revision that this paper is referring to is shown in the footnote[2].

As for future work, we are planing to introduce an extra IK step after the pose is copied to the target avatar, to account for self intersections with the body and other virtual elements in the environment. Moreover, we want to explore other IK solver algorithms (Caserman et al., 2019) as well as the introduction of existing IK tools, such as the well known *FinalIK* (Root Motion, 2021) or the *Unity Animation Rigging* package (Unity Technologies, 2021a). Those systems would bring even better quality to the default built in IK solver, and the user would have the possibility to choose the desired IK solver to apply.

We are also planning to introduce a more accurate calibration step (Caserman et al. (2019); Ponton et al. (2022)), which implies taking some body measurements and so the user needs to keep a specific pose on the calibration stage. However, this calibration process will not

2 Revision: 5d444b904e12f21eee4b833cc1d6869e1bc98c60

replace the current one, but will be completely optional for those applications and users where the default calibration does not produce a good enough result and a more accurate calibration is needed.

Finally, we want also to address the problem where feet tracking is not provided by the VR systems. In that case, we just apply an animation that takes into account the displacement of the HMD. So we have a blend tree with several walking animations, from walking very slow to running, it takes the speed of the HMD on that frame and outputs a walking animation that matches that speed. This solution is straightforward and easy to implement, but obviously it has the limitation that the position of the feet of the avatar are not going to match necessarily the position of the feet of the participant.

Data availability statement

The GitHub repository for QuickVR can be found here: https://github.com/eventlab-projects/com.quickvr.quickbase.

Author contributions

RO implemented the software and wrote the first draft of the paper. AB and XN contributed to the implementation of the software and the writing of the paper. MS obtained the funding and contributed to the writing of the paper.

Funding

This research is supported by the European Research Council Advanced grant Moments in Time in Immersive Virtual Environments (MoTIVE) grant number 742989.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Banakou, D., Groten, R., and Slater, M. (2013). Illusory ownership of a virtual child body causes overestimation of object sizes and implicit attitude changes. *Proc. Natl. Acad. Sci. U. S. A.* 110, 12846–12851. doi:10.1073/pnas.1306779110

Banakou, D., Hanumanthu, P. D., and Slater, M. (2016). Virtual embodiment of white people in a black virtual body leads to a sustained reduction in their implicit racial bias. *Front. Hum. Neurosci.* 10, 601. doi:10.3389/fnhum.2016.00601

Barberia, I., Oliva, R., Bourdin, P., and Slater, M. (2018). Virtual mortality and near-death experience after a prolonged exposure in a shared virtual reality may lead to positive life-attitude changes. *PLOS ONE* 13, e0203358. doi:10.1371/journal.pone.0203358

Bebko, A. O., and Troje, N. F. (2020). Bmltux: Design and control of experiments in virtual reality and beyond. doi:10.31234/osf.io/arvkf

Bourdin, P., Barberia, I., Oliva, R., and Slater, M. (2017). A virtual out-of-body experience reduces fear of death. *PLOS ONE* 12, e0169343. doi:10.1371/journal.pone.0169343

Brookes, J., Warburton, M., Alghadier, M., Mon-Williams, M., and Mushtaq, F. (2019). Studying human behavior with virtual reality: The unity experiment framework. *Behav. Res. Methods* 52, 455–463. doi:10.3758/s13428-019-01242-0

Caserman, P., Achenbach, P., and Gobel, S. (2019). "Analysis of inverse kinematics solutions for full-body reconstruction in virtual reality," in *2019 IEEE 7th international conference on serious games and applications for health* (SeGAH) *(IEEE).* doi:10.1109/segah.2019.8882429

Coelho, C. M., Santos, J. A., Silvério, J., and Silva, C. F. (2006). Virtual reality and acrophobia: One-year follow-up and case study. *CyberPsychology Behav.* 9, 336–341. doi:10.1089/cpb.2006.9.336

Emmelkamp, P., Krijn, M., Hulsbosch, A., de Vries, S., Schuemie, M., and van der Mast, C. (2002). Virtual reality treatment versus exposure *in vivo*: a comparative evaluation in acrophobia. *Behav. Res. Ther.* 40, 509–516. doi:10.1016/s0005-7967(01)00023-7

Epic Games (2021). *Unreal engine*. Available at: https://www.unrealengine.com/ (accessed July 9, 2021).

Extend Reality Ltd (2021). *Vrtk*. Available at: https://assetstore.unity.com/packages/tools/utilities/vrtk-v4-tilia-package-importer-214936 (accessed July 9, 2021).

Freeman, D., Haselton, P., Freeman, J., Spanlang, B., Kishore, S., Albery, E., et al. (2018). Automated psychological therapy using immersive virtual reality for treatment of fear of heights: a single-blind, parallel-group, randomised controlled trial. *Lancet Psychiatry* 5, 625–632. doi:10.1016/s2215-0366(18)30226-8

Kilteni, K., Groten, R., and Slater, M. (2012). The sense of embodiment in virtual reality. *Presence Teleoperators Virtual Environ.* 21, 373–387. doi:10.1162/pres_a_00124

Martini, M., Kilteni, K., Maselli, A., and Sanchez-Vives, M. V. (2015). The body fades away: Investigating the effects of transparency of an embodied virtual body on pain threshold and body ownership. *Sci. Rep.* 5, 13948. doi:10.1038/srep13948

Matamala-Gomez, M., Gonzalez, A. M. D., Slater, M., and Sanchez-Vives, M. V. (2019). Decreasing pain ratings in chronic arm pain through changing a virtual body: Different strategies for different pain types. *J. Pain* 20, 685–697. doi:10.1016/j.jpain.2018.12.001

Matamala-Gomez, M., Nierula, B., Donegan, T., Slater, M., and Sanchez-Vives, M. V. (2020). Manipulating the perceived shape and color of a virtual limb can modulate pain responses. *J. Clin. Med.* 9, 291. doi:10.3390/jcm9020291

Neyret, S., Navarro, X., Beacco, A., Oliva, R., Bourdin, P., Valenzuela, J., et al. (2020). An embodied perspective as a victim of sexual harassment in virtual reality reduces action conformity in a later milgram obedience scenario. *Sci. Rep.* 10, 6207. doi:10.1038/s41598-020-62932-w

Peck, T. C., Seinfeld, S., Aglioti, S. M., and Slater, M. (2013). Putting yourself in the skin of a black avatar reduces implicit racial bias. *Conscious. Cognition* 22, 779–787. doi:10.1016/j.concog.2013.04.016

Ponton, J. L., Monclús, E., and Pelechano, N. (2022). "AvatarGo: Plug and play self-avatars for VR," in *Eurographics 2022 - short papers*. Editors N. Pelechano and D. Vanderhaeghe (Geneva, Switzerland: The Eurographics Association). doi:10.2312/egs.20221037

Root Motion (2021). Finalik. Available at: http://root-motion.com/(accessed July 9, 2021).

Sanchez-Vives, M. V., and Slater, M. (2005). From presence to consciousness through virtual reality. *Nat. Rev. Neurosci.* 6, 332–339. doi:10.1038/nrn1651

Seinfeld, S., Arroyo-Palacios, J., Iruretagoyena, G., Hortensius, R., Zapata, L. E., Borland, D., et al. (2018). Offenders become the victim in virtual reality: Impact of changing perspective in domestic violence. *Sci. Rep.* 8, 2692. doi:10.1038/s41598-018-19987-7

Sheridan, T. B. (1992). Musings on telepresence and virtual presence. *Presence. (Camb).* 1, 120–126. doi:10.1162/pres.1992.1.1.120

Slater, M., Navarro, X., Valenzuela, J., Oliva, R., Beacco, A., Thorn, J., et al. (2018). Virtually being lenin enhances presence and engagement in a scene from the russian revolution. *Front. Robot. AI* 5, 91. doi:10.3389/frobt.2018.00091

Slater, M., Pertaub, D.-P., Barker, C., and Clark, D. M. (2006). An experimental study on fear of public speaking using a virtual environment. *CyberPsychology Behav.* 9, 627–633. doi:10.1089/cpb.2006.9.627

Slater, M. (2009). Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Phil. Trans. R. Soc. B* 364, 3549–3557. doi:10.1098/rstb.2009.0138

Spanlang, B., Normand, J.-M., Borland, D., Kilteni, K., Giannopoulos, E., Pomés, A., et al. (2014). How to build an embodiment lab: Achieving body representation illusions in virtual reality. *Front. Robot. AI* 1, 9. doi:10.3389/frobt.2014.00009

Tsakiris, M. (2010). My body in the brain: A neurocognitive model of body-ownership. *Neuropsychologia* 48, 703–712. doi:10.1016/j.neuropsychologia.2009.09.034

Unity Technologies (2021a). Animation rigging package. Available at: https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.2/manual/index.html (accessed July 9, 2021).

Unity Technologies (2021b). *Unity*. Available at: https://unity.com (accessed July 9, 2021).

Unity Technologies (2021c). *Xr interaction toolkit*. Available at: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html (accessed July 9, 2021).

Unity Technologies (2021d). Xr plug-in framework. Available at: https://docs.unity3d.com/Manual/XRPluginArchitecture.html (accessed July 9, 2021).

Watson, M. R., Benjamin, V., Christopher, T., Asif, H., and Thilo, W. (2019). Use: An integrative suite for temporally-precise psychophysical experiments in virtual environments for human, nonhuman, and artificially intelligent agents. bioRxiv. doi:10.1101/434944