Virtual observations

Howie, Scott; Gilardi, Marco

Published in:
Virtual Reality

DOI:
10.1007/s10055-020-00463-5

Published: 19/08/2020

Document Version
Publisher's PDF, also known as Version of record

Virtual Reality
https://doi.org/10.1007/s10055-020-00463-5

ORIGINAL ARTICLE

Virtual Observations: a software tool for contextual observation and assessment of user's actions in virtual reality

Scott Howie[1] · Marco Gilardi[1]

Received: 29 November 2019 / Accepted: 27 July 2020

Abstract
In this paper, we present 'Virtual Observation' (VO) a software tool for contextual observation and assessment of user's directly from within the virtual reality (VR) simulation framework. Unlike other recording systems, the VO system described in this paper focuses on recording and reconstructing VR user's positional, rotational and input data to recreate the same experience the user had with a VR simulation. Different from animation-based approaches, VO records user inputs and reconstructs the simulation from them and the user positional data. Moreover, the system allows the broadcast of this information to a remote machine enabling remote live observation of the simulation. Datasets recorded by the system can be shared by exporting them as XML files or, optionally, into a standalone online application, such as browser WebGL, allowing researchers, developers and educators to share and review a VR user simulation through a free-moving camera using a web browser. In this paper, the consistency of the data generated from the software by the client, server and reconstructed datasets acquired during real-time live observations was evaluated. We conclude that this Virtual Observation software offers detailed reconstruction of low-level information and visual information of user actions during simulations for both live and offline observations. We envision that our system will be of benefit for researchers, developers and educators that work with VR applications.

Keywords Virtual reality · Virtual Observation · Observing · Replaying · Reviewing · Simulations

1 Introduction

Direct observation of users in their context can offer insight on design challenges (Lazar et al. 2017) and circumvent the issue often encountered of users describing inaccurately what they did due to a lack of awareness or understanding of the task or system under study (Blomberg et al. 2007). The ability to replay and review user's sessions is an important tool for assessment (Lazar et al. 2017). However, when it comes to studying users in virtual reality (VR), performing contextual studies becomes challenging, as the user and the observers are positioned in two different contexts: the observer is in the real world, whilst the user is in the VR context. Observing users from inside the same VR simulation allows insight into their performance (Hanoun and Nahavandi 2018) and helps in determining the cause and effect relationships from user actions (Hanoun and Nahavandi 2018). Unlike desktop computers, VR equipment has six degrees of freedom (Tromp et al. 2003), and because of VR's unrestricted movement, usability issues are common for inexperienced users (Tromp et al. 2003).

Currently, to observe users using a VR simulation an observer needs to be physically present to observe the user's body movement in the real world, take notes on what the user does or video record and screen capture the user. Although rigorous, this type of observation is difficult, requires large data storage for the videos, can impact the simulation performance and offers limited insight of the users experience and interaction. The ideal solution would be to experience the exact perspective of users, but this approach has the drawback that it can induce motion sickness on the observer (Lopez et al. 2017). Observations can be conducted from mirrored perspectives of VR users on 2D screens, but these recordings cannot guarantee knowledge of the state and location of the input devices when the user

✉ Scott Howie
Scott.Howie@uws.ac.uk

Marco Gilardi
Marco.Gilardi@uws.ac.uk

[1] University of the West of Scotland, Paisley, UK

Published online: 19 August 2020

is not looking at them, limiting studies inside and outside laboratory conditions. Video recording users offer visual clarity of their movement in VR, but the point of view of the recording camera is fixed in position and user actions can be obscured from the camera point of view. Moreover, this approach requires the video data be paired and synced with simulation data, such as 2D-screen recordings of the VR perspective before it can be analysed.

This paper builds upon Howie and Gilardi's 'Virtual Observation' system described in Howie and Gilardi (2019). Howie and Gilardi (2019) system records and reconstructs virtual actions of users in a simulation, allowing researchers and developers to virtually observe users in a VR simulation. The term 'Virtual Observation' (VO) is used as observers are not observing the user directly, but instead they observe the input, movements and actions conducted by the user within the simulation. Collected input and movement data is used to recreate the user experience in the simulation with synced actions and approximate movement. In Howie and Gilardi (2019), VO was validated in a detailed simulation environment for fire safety which tasked the participant to correctly identify and extinguish fires in a VR training simulation (Howie and Gilardi 2019) (Fig. 1). VO was used for reconstructing users movement and actions for review *after-action*, meaning the observation was conducted after the user had completed the simulation (Howie and Gilardi 2019). The VO system in Howie and Gilardi (2019) can reconstruct VR simulations for a wide range of use cases (Wobbrock and Kientz 2016).

Building upon the initial version of VO (Howie and Gilardi 2019), we expanded the system functionalities to allow for remote live observation of users via WiFi from an external location in real time using Unity's default networking architecture, added configuration for full body tracking, optional hand tracking and an 'Full Simulation Capture' recording mode that can record and reconstruct external tracked object classes and variables. Moreover, controls such as play/pause and rewind/fast-forward were added to allow observers control over the reconstructed simulation.

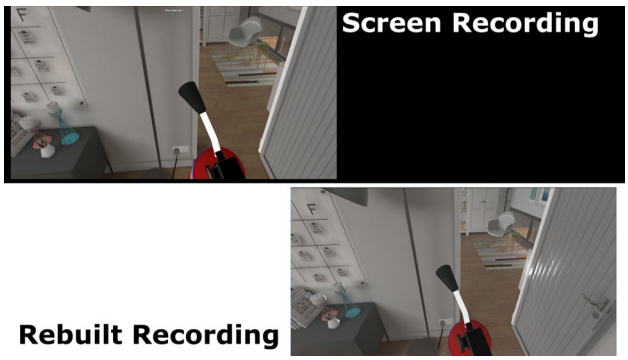

Fig. 1 Screenshot captured from video comparison showing real-time video capture of a simulation and rebuilt capture using Virtual Observation. The full video can be viewed here: https://www.youtube.com/watch?v=2YY-d7QMUVI . The Virtual Observation system can be included in any SteamVR compatible Unity project by adding the VO component to the default VR camera rig set-up and linking the input states of the developers/researchers interaction system (so the actions can be recorded and later replicated). Recorded data can be stored in XML or JSON file formats and supports all VR headsets and tracked controllers that are compatible with SteamVR and the Unity game engine

Although there are other popular platforms that facilitate the development of VR applications, such as the Unreal Engine, we chose to develop the system using Unity. This choice was made for convenience; Unity is one of the popular and widely used platforms for developing VR and the platform with which authors are most familiar. However, the principles underlying VO, i.e. recording user input and positional tracking data to reconstruct the simulation, can be transferred to other engines, and Unity should be seen throughout the paper as the mean that we chose to prove that a system such as VO can be developed.

This paper contributes to the field of virtual reality by presenting a new approach to recording VR simulations for training scenarios. The approach is implemented as a proof of concept of a versatile tool that will allow researchers, developers and educators that use VR, to observe, evaluate and share user actions and interactions within a VR simulation, either live or by reconstructing it via recorded data. Moreover, we demonstrate that:

1. data generated by VO can be reliably streamed to a remote machine for real-time live observation of a user VR session,
2. streamed VO data can be recorded in the observer's remote machine and that simulations reconstructed from this data are identical to the VO data recorded on the user machine during the user simulation, showing VO reliability in recording VR user sessions,
3. VO data can be used to share the VR user session after it has ended on both VR and non-VR platforms, such as a WebGL visualiser.

2 Definitions

The following terms will be used in this paper:

– *Motion key-frame* values are the position, rotation and scale of the tracked devices in the virtual simulation space. Motion key-frames are recorded through-out the duration of the simulation in sequential order, later being used to reproduce the movement of the VR user during reconstruction. We refer to motion key-frame data as *high-level information* which is acquired from real-

time values from the Unity transform component during a simulation.
- *Action key-frame* are frames in which input from one of the controller devices has been modified from a previous state, for example, a button on a tracked VR input device that changes state from idle to being pressed down would be recorded as an action key-frame, since a variable that derived from a VR input devices had been altered. We consider action key-frame data as *low-level information* since the data acquired is in a *raw data* format of values that correlate as one's, zero's or customised values, with input values acquired or derived from the VR controller button state changes. During reconstruction, the action key-frames are reconstructed to replicate the experience of a VR controller device being modified.
- The terms *Rewind* and *Fast-forward* are used to describe the functionality in the VO system that allow users to jump to a specific time frame, before or ahead of the current time position, and see the simulation from that point onward. The terms are used for a lack of better words to describe the functionality and should not be confused with being able to see the simulation replayed backwards or at accelerated speed.

3 Related work

The problem of finding a way to review VR simulations is not new, and systems similar to VO have been previously proposed in the literature. One of the earliest examples is Goldberg et al. (2003), which used early implementations of VR technology to replay and review army training simulations in an effort to measure users performance. Goldberg et al. (2003) after-action review system used event data collection processes to capture the events that took place during a simulation. Whilst this offers a guarantee that an event can be triggered repeatedly during the after-action review, the input of the VR user is not recorded. Therefore, any usability issues experienced by the VR user cannot be replicated and therefore prevent usability analyses to be conducted.

Greenhalgh et al. (2002) developed a technique of temporal links which enabled the recording of data in the MASSIVE-3 system (Greenhalgh et al. 2000), with the objective to link prior recordings to real-time VR environments. Greenhalgh et al. (2002) system operates by recording all changes made to the virtual environment at run time. The focus of Greenhalgh et al. (2002) work was integrating VR data for media use in film and television and incorporation of previous VR experiences into a real-time VR environment to create new content or review experiences. Using temporal links, Greenhalgh et al. (2002) method allowed recorded virtual environment can be replayed and embedded into a live virtual environment for purposes of extending the live virtual environments content or narrative structure. Although not stated, the implementation of temporal links (Greenhalgh et al. 2002) appears to be designed for desktop VR and not head-mounted display VR. Likewise, the description of the temporal links system (Greenhalgh et al. 2002) hints that the recording and reconstruction of the temporal links use animation-like recording of the virtual environment rather than raw data interpretation of the VR users as used in VO, which enables usability testing and repeatable interactions for development and bug fixing. In a use-case example for reviewing virtual experiences (Greenhalgh et al. 2002), the authors describe the ability to replay the virtual environment from any perspective but do not discuss any ability for interaction of user input or variable monitoring of recorded objects. This lack of clarity suggests the temporal links system is animation based which suffers from the same limitations as other animation reconstruction systems (Lopez et al. 2017), such as the lack of raw user-input data, which limits the understanding of usability issues from input devices.

Similar to Greenhalgh et al. (2002), Von Spiczak et al. (2007) expanded upon the OpenTracker framework (Reitmayr and Schmalstieg 2005) to create a multi-modal event processing system. A feature of Von Spiczak et al. (2007) approach is the ability to capture position, orientation and interactions of equipment from the multi-modal event data structure. This data structure can be used to replay, review and document VR interactions in a virtual environment in a serialised order. These recordings captured positions and orientations of the tracked objects along with other interactable object information within the virtual environment. When replayed, these simulations would reconstruct the event with a time-delay between the captured data points. Von Spiczak et al. (2007) show that their system is viable, but only discuss briefly the reconstruction of interactions in their paper, using the data reconstruction example as a use case for their multi-modal event processing system designed for the OpenTracker framework (Von Spiczak et al. 2007; Reitmayr and Schmalstieg 2005). Von Spiczak et al. (2007) make no mention of rewind or pause ability, observational potential, live broadcasting of data, portability of their system and functionality of the system beyond the ability to replay and log the VR interactions in real time. Both Greenhalgh et al. (2002) and Von Spiczak et al. (2007) implementations are designed as extensions for specific systems and are incompatible with the Unity platform, which relies on game engines. In comparison with VO, the discussed implementations (Greenhalgh et al. 2002; Von Spiczak et al. 2007) share some methodological similarities in approach for capturing and reconstructing actions; however, Von Spiczak et al.'s (2007) study lacks important functionalities for allowing effective observations of VR users, such as: control the perspective, time and playback of the reconstruction,

live observation, automatic data serialisation, recording and reconstructing objects and variable data beyond the VR interaction devices, lightweight portability of data and multi-user data capture.

More recently, Lopez et al. (2017) used a technique that creates and stores animation of movement and object manipulations conducted by the user. Although effective, Lopez et al. (2017) system is restricted at re-creating the motions of the simulation and does not store low-level information about the interaction. Because Lopez et al. (2017) system stores all data into animation files, the data can only be accessed using the Unity game engines animation system, preventing statistical data to be collected for state or positional analysis in external programs.

A system similar to VO was created by Jung et al. (2006). Although Jung et al. (2006) system is capable of creating reusable animations for animating 3D character models, it only records predetermined input states for grasping an identified object. Unlike our VO system, which aims to capture all states from VR input devices, such as the state of input buttons, including raw values of analog inputs. Jung et al. (2006) system focuses on storing the grasp events so they can later be imitated for manipulating virtual scenes with different grasping types. Like Lopez et al. (2017), Jung et al. (2006) pre-defined input and interaction objects limit the high-level information that can be gathered from observing the animations, with neither systems aiming to capture all states of the VR input devices. Both Lopez et al.'s (2017) and Jung et al.'s (2006) methods are limited as for in-depth analysis of user's actions in a simulation knowledge of high-level information (input and devices states) is necessary.

Alternative 'virtual' means of observation can be used to monitor participants in VR remotely through the use of cameras and remote screen sharing (Lazar et al. 2017), but these options cannot guarantee knowledge of the state or location of tracked devices. Even with these forms of observation available, for remote studies conducted 'in the wild', setting up software or hardware for observing remote users could be a challenge (Petrie et al. 2006). These observation methods also require the participant to have access to suitable recording hardware and accompanying software for remote observations to be possible.

Video and audio recordings are a common way for studies in laboratory and 'in the wild' to document experiments and procedures that can be then analysed qualitatively (FitzGerald 2012). These recordings aim to make it possible to analyse the events in the experiment after the experiment has been conducted using a format that is easy to share between collaborators and for demonstration purposes (FitzGerald 2012). Our VO system adopts (FitzGerald 2012) approach and innovates the medium to a digital data format designed for recording and reconstructing VR simulations that can be shared independently of a development system for browser-based collaboration. Parallel to transcription work that is often applied to audio and video recordings (FitzGerald 2012), VO allows for software detection of user input and movement, that in the future, could be paired with AI for analysis of participant actions.

Different from the systems discussed, VO focuses on the low-level acquisition and reconstruction of data in an easily controllable dataset that once implemented into an application automatically and non-invasively records the actions of a participant in a VR simulation, without any modification by the developer or end-user. In VO, the recording of user inputs depends exclusively on the input controllers SDK (OpenVR, Oculus SDK, Leap Motion or other third-party SDK) chosen for the VR simulation, allowing different types of controllers to be supported, as shown in Fig. 3 where hand tracking is used in place of the VIVE controllers, and new controllers can be easily added in the future to adapt to the fast evolving VR input modalities. Finally, rather than replaying a VR simulation as an animation (Lopez et al. 2017; Greenhalgh et al. 2002), VO reconstructs the experience using the low-level data of the VR users movement and input. Moreover, the system is designed to be lightweight, portable and user-friendly and it is developed using a modern game engine, namely Unity.

4 The Virtual Observation system functionalities and design

The VO system presented in this paper is a combination of unrestricted observation of users (Carranza et al. 2003) with an improved form of action capture (Lopez et al. 2017; Jung et al. 2006) and allows users to be observed from any perspective. Rather than storing information of the entire environment, the VO system records users movement and changes in low-level information data captures. The low-level information captured is configurable and as default supports all OpenVR hardware input devices, but can easily be extended to suit other input devices; for instance, Fig. 3 shows the integration of Leap Motion. A callback pattern allows VO to listen and dynamically record extended input, such as additional API input information. In contrast to Jung et al. (2006) and Lopez et al. (2017), which record the motion of the interacted virtual objects directly, the major contribution of the VO system is that only users inputs, device states and tracking information is recorded by the system. The VO system captures and records an unconstrained number of tracked objects and devices in addition to three tracked main VR devices: head-mounted display (HMD) and two controllers. By adding additional tracked objects (Fig. 2),

non-player controlled objects such as AI characters or non-deterministic physics-affected objects can be reliably

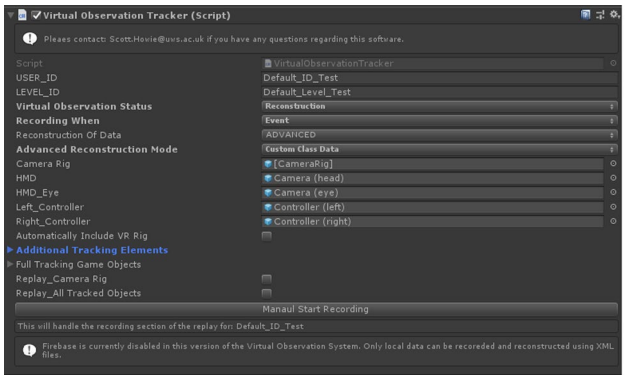

Fig. 2 The VO system is built for the Unity game engine. The component can be attached to any VR player object with options for HMD, HMD Eye and controllers the default options for tracking of movement. Additional tracking objects offer extension to the prerequisite tracking objects. The variable 'Full tracking Game Objects' is used for tracking a hierarchy of objects, this will track the parent object and all subsequent child objects

Fig. 3 Screenshot captured from a reconstructed simulation that used a leap motion tracking device to capture the movement of a user's hands. This demonstrates VO's ability to extend recording and reconstruction to a wide range of interaction devices. A video demonstration of the VO system can be viewed here: https://www.youtube.com/watch?v=IX1qjq1R134

tracked, ensuring that the reconstructed simulation is consistent with the simulation 'as-experienced' by the user. Unrestricted tracked objects and devices also allow for novel or unconventional objects to be tracked, expanding the scope of the tracking hardware to include additional interaction systems that rely on eye-tracking or hand-tracking (Fig. 3).

The proof of concept for the VO system was implemented as a plug-in for the Unity game engine and can operate in two modalities, 'Interaction Capture' and 'Full Simulation Capture'. The *Interaction Capture* mode records and reconstructs VR users position, rotation and actions within a simulation, isolating data captured to modifications made by a single participant. However, as the 'interaction capture' recording technique does not capture non-deterministic objects, an advanced method which can be toggled to automatically capture all gameobjects that contain non-deterministic factors, such as AI or physics impacted objects, was added to the system. The *'Full Simulation Capture' mode* extends upon the 'Interaction Capture' mode by including serialisable and custom-assigned variables of additional tracked objects, which are optional and can be used for the reconstruction of AI driven or physics gameobjects that are non-deterministic.

4.1 Interaction capture

To record a VR simulation in VO, positional and rotational data of tracked devices are motion key-frames that are recorded at a fixed frame rate interval of 10 milliseconds, which is the lowest value of the record and reconstruction functions being called during a VR simulation (see Howie and Gilardi 2019). Input action key-frame when a modified input state is detected is also recorded, along with all non-deterministic game data, such as modified objects with physics properties. The recording process stores the current transform of all tracked devices in the VR manager with the current input modifications. VR tracking configuration data, such the width and depth of the physical assigned VR area, is also stored to allow for tracking play-space and headset system information to be analysed. Tracking configuration data allows to contextualise the tracked movement of users relative to their configured tracking space in the real world during 'in the wild' studies.

4.2 Full simulation capture

The prototype of the VO system in Howie and Gilardi (2019) was originally intended for observation of pre-defined training simulations, which only required the knowledge of the VR user to operate (Howie and Gilardi 2019). In this paper, we present an extended version of VO that includes a 'Full Simulation Capture' method of recording external actions of other non-player controlled objects. Because some simulations rely on the knowledge of external human interaction factors to regenerate the simulation or AI characters, additional objects and their attached components can also be recorded and reconstructed to be in sync with the VR user. This recording process can automatically serialise the additional tracked component data of serialisable classes and user-defined data from custom or protected Unity components (i.e. Rigidbody or Colliders). This data can be recorded in sync with the VR user recording motion key-frame. This 'Full Simulation Capture' mode is less user-friendly than the 'Interaction Capture' mode because as it requires the

developer to specifically detail each component and variable that is required to be recorded and reconstructed during development. At present, 'Full Simulation Capture' mode uses customised classes to store the active status, tag and name of objects, accessible 'rigid body' and 'collider' variables along with fully serialisable custom classes of tracked objects.

4.3 Reconstruction system

In order to reconstruct the transform data (position and rotation), data-points are captured during the recording of the simulation at motion key-frames and action key-frames. Motion key-frame data points are captured every 10ms, which was found to be reliable for movement estimations, whilst action-frames are captured when an input state change is registered. During reconstruction, the system determines whether the reconstruction time matches a recorded action key-frame or motion key-frame (Fig. 4).

During an action key-frame, the input states of the tracked devices are reconstructed by rebuilding the data to a class structure readable by Unity and assigning them to the object that matches the tracker. In our case, this was a VR Input management class, which handled the input of the VR controllers that determined the action, values and states of our interaction system. When an action key-frame or motion key-frame event is called during the reconstruction process, all tracked objects are set to the position recorded during the recorded key-frame. The software modifies the tracked objects based on the current position of the tracked objects and next transforms location based on the next action key-frame or motion key-frame data point. This technique enables the motion of the participant between timed gaps in the data to be estimated from the previous and next action or motion key-frames. See Fig. 4 for a diagram of the reconstruction process.

To determine whether the simulation is recording or being reconstructed, conditional checks are used in the VO system along with individual information, thus determining whether a user has direct control over an object, i.e. the user is holding or interacting with the gameobject. These checks are primarily designed to let the system know that simulation data is expected to vary from recording data and prevents individual object recorded data to override the user-driven input actions. During reconstruction, this does not matter as the user-driven input actions are reconstructed before any other game data; therefore, any variable modification to held objects will remain consistent. As such, the checks are intended for aiding the debugging process of the simulation.

4.4 After-action review

To make it possible to review a simulation at a later date or time than the original simulation, an 'after-action review' (offline) process is implemented. Such system focuses on recording the data for reconstruction after the user has completed the simulation. Action events that are registered from a change in input state are recorded along with the current position of all tracked objects. Key-frames for action and motion are sequentially logged in order of time gap from the start of recording and stored immediately on a database or recorded locally to XML (once the simulation has finished). This data can be retrieved and used for a simulation to be reconstructed when desired by the developer or researcher.

To replicate the movement and actions of the input devices, rebuilt simulations use user datasets. User movement is smoothed between action key-frames when input is modified, and motion key-frames that capture the motion of the tracked devices. During instances of input changes, the position and rotation of the tracked devices is forcibly set to ensure that interaction is correctly mapped at the exact position and orientation recorded and is not affected by any delays or gaps in the animation smoothing process.

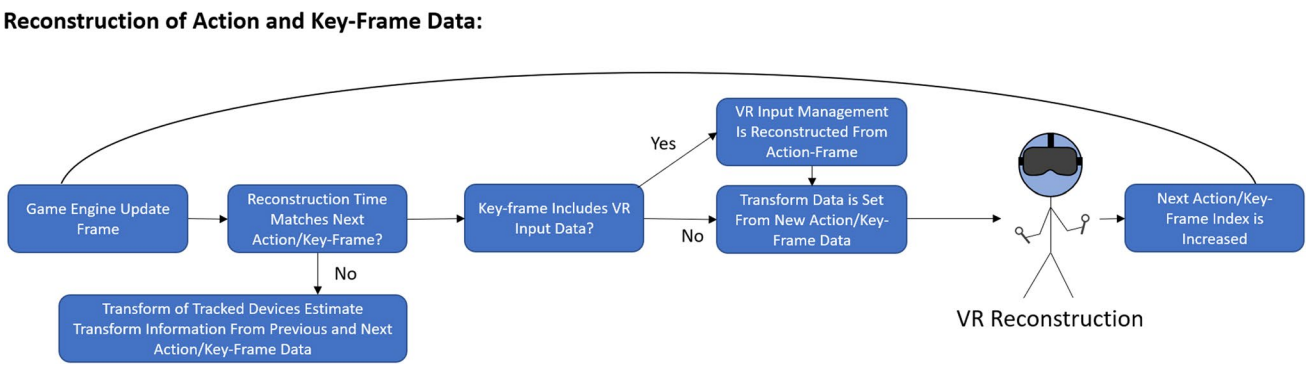

Fig. 4 The reconstruction process of the VO system, demonstrating how the reconstructed VR character transforms update based on the action or motion key-frame data. Transform data of tracked devices are recorded every 10ms as motion key-frames, whilst input management data and transform data of all tracked devices are recorded as action-frames when the a change of input state is detected

4.5 Live-action review

'Live-action review' is the process of virtually observing a user in a VR simulation in real time. This method is based on standard networking set-ups for multi-player games, in which a main user hosts the network-configured game acting as the server, with other players able connect to as clients, allowing data to be transferred continuously between client and server. The data structure and process of recording and reconstructing the data are the same as after-action review, but instead of storing it for later use, the data is streamed in real time from the client (user) to the server (observer). This process updates the simulation in real time by monitoring the actions and movement of the user from an external location. During live observation, actions and movement of users of the simulation who are clients on the server can be recorded locally, enabling data that is streamed to the server during the live observation to be stored using the same process as after-action review. Live-action review uses the Unity game engine networking suite to create the area for multiple clients or observers to be present within a simulation. Whilst only assessed with one client and one observer, in principle the software scales with Unity's multi-player functionality to support multiple members for both roles (simulation user or observer).

The VO system can be incorporated into existing infrastructures, requiring only one component class for data allocation. This approach allows the system to use Unity 2018's standard Networking Interface UNET for live transmission of data. Unity has since deprecated this Networking Interface, but the software will be compatible with Unity's replacement Networking architecture or any alternative that allows custom data transmission in real time. Live-action simulations use the networking data from the Unity network transform components to update the transforms of the tracked objects on the server (Fig. 5). These transforms have a send rate of 25 network updates per second, but locally saved movement recording of the tracked devices remains at 10 ms. When an input state change is registered on the client and sent to the server, all tracked objects on the server (including any other connected clients) are updated to their recorded position when the input state change was detected. This ensures that the transform of the objects during a received input action is consistent with their position on the client and is not affected by any lag or networking issues. Due to University firewall settings, we were only able to perform tests using a WiFi local area network (LAN); therefore, lag may be present when applied in different networks configurations depending on the connection of the server and client. Different network connections may cause potential issues over long distances or bandwidth limitations. However, we show in this paper that these issues are not caused by the VO system and are only dependant on the networking architecture used.

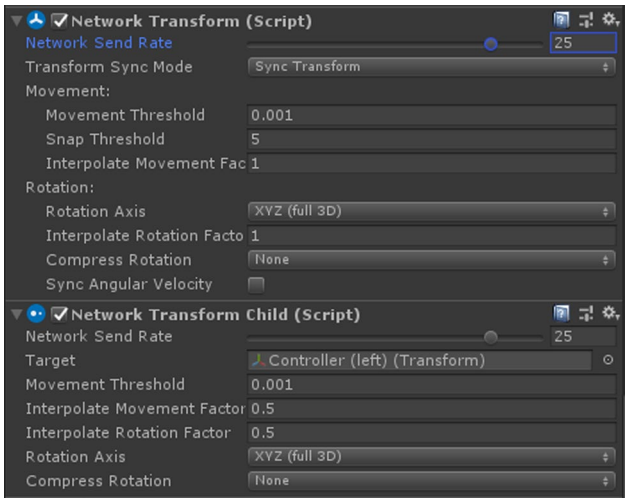

Fig. 5 Unity's networking configuration for sending packets of data between client and server in real time. Each player VR character is set as a local player authority on the network, with network transforms attached for the top hierarchical object, and child network transforms for all tracked child objects of the VR character. Data packets were sent for the tracked objects at a send rate network cycle of 25 updates per second

4.6 Reviewing observations

Using an inverse kinematic (IK) system, an estimated skeleton posture of the user can be generated using the rebuilt tracking points from the VR equipment (Fig. 8); these tracking points map the head, hand and other tracked VR locations of the user to an avatar for real time or after-action animation. To implement this, we used the FinalIK (Lang 2019) package for full-body tracking of the participants head, hands (controllers), torso and feet.

To observe users, the observer can either control a virtual 3D camera or use a separate HMD attached to the server computer (Fig. 6). Both forms of observation explore the reconstructed simulation in real time. To reproduce a user simulation, the rendering and update of the Unity game engine had to be mimicked to prevent positional data from being set or timed incorrectly from the original data timestamps. This required to modify the fixed update cycle of the Unity engine which we set to update every 10ms to match the recording interval.

Time-frame reproduction was improved with respect to Howie and Gilardi (2019) system, obtaining smoother transitions of action key-frame and motion key-frame capture points in after-action review reconstructions. One issue in Howie and Gilardi (2019) system was that a frame was skipped between the action and reproduction of the input

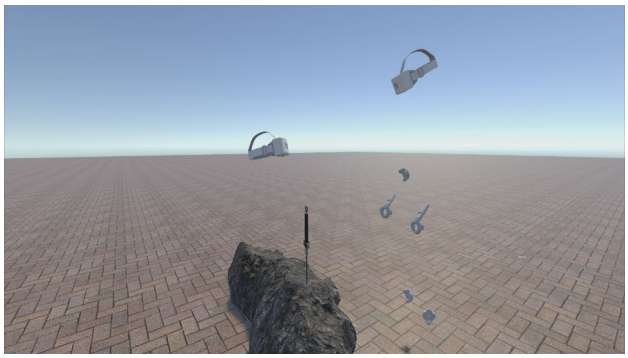

Fig. 6 Observers can observe users using a virtual 3D desktop camera or by equipping a VR HMD. Observations can take place live in real time with the user, or at a later stage for after-action observation. User datasets can be observed using the desktop system here: https://virtualobservationsystem.github.io/VirtualObservationSystem/

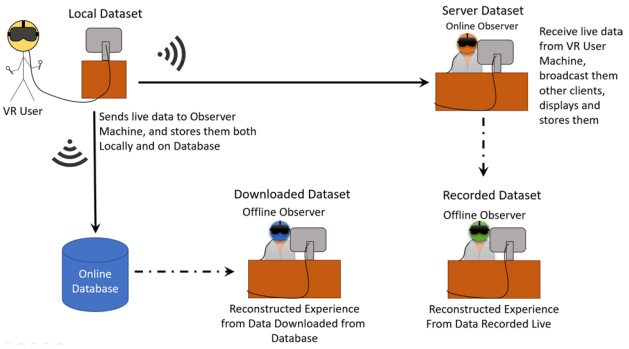

Fig. 7 The VO system has multiple methods of storing and retrieving data. For live observation, data is streamed to the server, updating the simulation in real time and storing the received information locally for offline observation later. For after-action review (offline observation), data is stored on an online database (Firebase) for retrieval by the server when necessary. A local XML copy can also be stored on the clients PC for recovery in local tests

which was caused by Unity's update pipeline. By using a modified order to the script execution pipeline, the skipped frame can be avoided with actions registered in the same action key-frame as the recorded input. To achieve this, the script execution order in Unity is changed so that the script that rebuilds the simulation is given priority over the input action scripts that determine the controller input actions; this ensures that the movement transforms and input state of the devices are exact during the time of an input state change. The execution of physics interactions for collisions is staggered during the reconstruction so the physics and events of objects replicate the expectations of the engine, with movement for physics collisions and triggers executed prior to input modifications.

The VO system is designed as 'drag and drop' component in Unity; this design choice was taken to simplify the integration of the system in VR simulations. The replication of movement can be achieved by adding the VO system to the top-hierarchy object of the player character and assigning the desired tracked objects (Fig. 2). Every VR player controlled character in a scene is individually recorded, independent of other users. This streamlines the process of storing and re-accessing data that is only applicable to a single-user performance and limits damage caused by bandwidth constraints or data corruption. For group simulations where an observed user interacts with other users in a single scenario, each individual dataset for all users can be loaded and reconstructed simultaneously, reproducing the movement and actions of all users within a single observation session. Recorded audio group communications can also be reproduced as VO records from the microphone input of each user if a microphone is attached to their VR headset and it is active. The audio can also be isolated for individual users, which is useful in scenarios where group communication makes it difficult to hear an individual user speaking.

Data can be acquired in several ways. For live observation of users, data streamed to the live observer can be recorded locally on the server. When data is received by the server during live observation, the reconstruction of the data is processed in real time and stored by replicating the data capture process that is being conducted on the client side. Alternatively, data can be acquired from the client by retrieving a locally stored XML file, avoiding the need for online database hosting. If a study is being conducted in a remote location, making the retrieval of data from the client computer impossible, an online database can be used to host the client data which can then later be retrieved by the observer when required. See Fig. 7 for a diagram of the data storage options.

4.7 Review functionalities

Review functionalities for observers allow to control the reconstructed simulation playback (play, pause, fast-forward and rewind) and observational position during and after-action review. These functionalities work independently of any external VR plug-ins or frameworks, allowing observations to be conducted on any Unity supported platform. In this paper, we show VO being used to observe a recorded VR simulation within a VR application, a desktop application, WebGL web browser hosted on a website and within the Unity game-engine itself and show the potential of VO being used for observational purposes in multiple fields.

Rewinding and Replaying a reconstructed simulation using the 'Full Simulation Capture' mode will result in repeated game events being re-simulated. During rewind instances, modified variable information will be restored back to previous values which will then be re-simulated when the reconstruction resumes in forward playmode.

Assuming both a fire extinguisher and fire were recorded along with the VR user, a reconstruction operating in rewind would reverse and the fire extinguisher capacity and the fire's health back to their previous values, along with all other recorded data recorded with the individual gameobjects. When resuming to play in forward mode, the simulation would reconstruct the actions of the user extinguishing the fire, since the variable information of the gameobjects was reset and the user-driven actions of using the fire extinguisher are consistent. This means in the fire training simulation demonstrated in Fig. 1, the fire could be extinguished and made re-active by the rewinding and re-simulating the experience.

Observing a training simulation within VR replicates real-world attributes of monitoring a trainee (see video in Fig. 9). Moreover, the option for observers to be invisible during the observation avoids users' feelings of discomfort caused by the physical presence of the observer. In VR, both the observer and VR user share the same context, both experiencing the presence and scale of the virtual environment. Unlike in the real world, observing in VR is non-intrusive, with the ability to monitor the training experience from any perspective without interfering with the VR training user's performance.

In desktop applications (Fig. 9), the observation is controlled using keyboard and mouse input and operates closer to the functionality of video playback, but with the functionality of altering the viewing perspective of the observation camera. This observational method has greater accessibility for observers since the application can be operated using standard computer hardware, with no additional hardware or programs beyond the simulation application.

Like the desktop implementation of VO, the WebGL browser functionality is an accessible and portable method of observing VR simulations (Fig. 6). WebGL applications can run on any web browser platform that supports Unity's WebGL platform. WebGL applications have the advantage of being portable to online websites that can run without any software being downloaded or installed. The portability and accessibility of VO enable VR simulations to be shared online, allowing the observer using the WebGL application to control their perspective of the observation.

Observing can be also done from within the Unity game engine (Fig. 3), enabling data analysis of the reconstruction. We envision this functionality particularly useful for developers, allowing them to identify bugs and usability issues during testing phases of VR applications. Moreover, developer can test their VR experiences on a user-recorded simulation so that, rather than having to frequently re-equip VR headset every time they need to test bug fixes, developers can monitor the reconstructed user actions live whilst focusing on the development output of the Unity game-engine log.

In all of these platforms, the core functionality of VO remains the same in each implementation, enabling playback to rewind, pause and play the reconstructed simulation that is controlled by the observer of the simulation.

4.8 Technical configuration

The live observation was incorporated into Unity's standard Networking configuration for Unity 2018. Unity's default networking handles the transform data of the VR tracked devices for real time (Fig. 5). To register input actions, the VO system monitors for changes in input state of the controllers. When an input change is detected, such as a button state from 'DOWN' to 'HELD', the VR input management class is converted to a JSON string and sent from the client to the server, which distributes this to all other clients connected in the simulation. On the server host, all data received from the connected clients is recorded and saved locally to an XML file once the simulation has elapsed.

5 Methodology

Building upon previous work (Howie and Gilardi 2019), we measure consistency between the data captured on the client computer, server and reconstructed simulation using data captured by the server, showing VO reliability in recording VR user sessions. We are currently only interested in demonstrating the potential for the software for use by researchers, educators and developers to aid VR simulation assessment and development rather than conducting a usability tests (Greenberg and Buxton 2008).

The assessment of consistent data only used the 'Interaction Capture' mode of VO because the environment did not feature any undefined configurations or unknown variable data. A controlled laboratory condition environment was used to validate consistency between client, server, and reconstructed datasets. The observer (one of the authors) hosted a multiplayer session of the simulation (server) located in a separate room next to the laboratory. A wireless local area network (LAN) was used to connect client to the server (observer) machine located in the adjoining room. Due to restrictions of the University firewall, connection between the client and server was achieved using a non-internet-connected router. It is unknown whether this had any negative or positive impact on networking performance or loss of data packets.

Participants for the test were equipped with a VR HMD, two VIVE controllers and three VIVE trackers (for a total of six tracking points) and were asked to start a standalone build of the application on the client machine. After participants entered the local IP address of the LAN server hosted by the observer, they joined the simulation session hosted by

the observer. The measured datasets consisted of the client who run the local version of the simulation (Client Dataset, Fig. 7), the server who hosted the multi-player simulation and observed the client in real time (Server Dataset), and reconstruction data generated from an after-action review simulation (Reconstructed Dataset) from the data captured by the server.

Once ethics approval was received by the university, students and staff from the university were recruited via email and word-of-mouth. Five participants volunteered and gave consent to take part. After consent, they were equipped with the VR headsets and allowed to start the session. During the experiment, they were asked to perform the same task repeatedly 35 times, picking up and dropping a sword in a virtual environment. This type of task was chosen as the system captures data only when an input state change is detected. Captured data is guaranteed at points of input state change in client, and, if the system is reliable, these data points should be replicated identically in the server and in the reconstruction, ensuring that the observed interactions and actions in the simulation are as close as they can be to the user interactions and actions. Each participant conducted the test in the laboratory room alone with the observer as the only other (invisible) character within the VR simulation. The observer could move in VR or using the desktop camera to observe the participant from any perspective.

To validate the consistency of the position and rotation data of each tracked VR object, local data captures were recorded on client and server and the server dataset was used to reconstruct the simulation and recapture the data. Local data captures were saved as XML files and were readable by the VO software to reconstruct the actions during a given time action key-frame.

As the client dataset stored inputs and devices states recorded directly from the hardware, it was used as the baseline to determine the consistency of the server dataset. The value d_i, obtained as the absolute difference of values of corresponding data linked to tracker i stored in the server s_i and in the client c_i datasets, as given in Eq. 1, was interpreted as a measure of data consistency between the two datasets; any difference between these datasets was attributed to the live broadcasting of the simulation via the LAN network.

$$d_i = |s_i - c_i|. \tag{1}$$

The difference v_i, computed between the server s_i and the reconstructed r_i datasets, as given in Eq. 2, is to be attributed to our reconstruction system.

$$v_i = |s_i - r_i|. \tag{2}$$

The datasets for the five participants containing the changes in inputs and states for each tracked device were used for the validation of the VO software, the client dataset had in total 4,869 entry points, whilst the server and reconstructed datasets had 4,866. Three packets were lost during the broadcasting between the client and the server; those data were removed from the client dataset during analysis.

6 Results

The difference d_i, as given in Eq. 1, between the client and server datasets and the difference v_i, as given in Eq. 2, between server and reconstructed datasets were analysed by identifying the maximum (worst) difference between the datasets for position and rotation. In the worst case, the client–server difference d_i was of the order of 10^{-7} for position and of the order of 10^{-3} for rotations (recorded as Euler angles), showing that the broadcasting introduced some errors in the data. Despite these small differences between client and server data, we consider the two datasets to be consistent as these small discrepancies are unlikely to be noticed by a human observer. The differences v_i between data in the server and reconstructed datasets were consistently zero for all data points, showing that the reconstruction system preserves the data used for the reconstruction.

A minor loss of packets was noticed when large data was streamed continuously over the network, and happened when a participant left their finger on the controller track-pad during the entirety of the simulation, generating small changes in input states for the track-pad. Considering the large amount of data sent, the loss was negligible in terms of data acquisition with only three instances out of 4,869 (0.0006%) packets lost for all participants data transmitted live. This is likely to be caused by excessive bandwidth used. During instances of packet loss, the system continued to operate using previous data received with follow-up data after the 10ms continuing the live reconstruction. Because the instances of packet loss were only noticed during continuous input state modification, threshold values can in future be used to prevent small changes from registering a state of input change. On a stable and reliable network connection, we do not anticipate that any issues will hinder the live observation for normal input usage once the threshold change value has been included. Because loss of streamed data could be a critical failure in the simulation process, networking features could be implemented to allow for lost packets to be resubmitted to keep the simulations in sync. Alternatively, local client data can be submitted for after-action review if networking issues prevent real-time observation. These limitations are caused by the network transmission of data and out of the scope of this paper.

The reconstruction of tracked VR object motions and input states allow for serial observation of user(s) actions. Rather than restricting analysis to after-action review, live observation allows for instantaneous visual clarification

to user's actions and their resulting performance in a VR simulation.

7 Discussion

The validation of the after-action system (Howie and Gilardi 2019) highlighted the potential uses of the VO system, user's actions and motions relative to their real-world posture (Fig. 8) and discovered cases of participants failing to interact with objects, using the wrong input command/button, as well as 'magical interaction' (Bowman et al. 2012), which removes the senses of weight and cumbersomeness from real-life equipment. Participants were also observed to frequently attempt to interact with objects in incorrect positions. During our live observation tests, we noticed similar findings regarding user inputs, but were able to address problems immediately by communicating to the participant using voice communication through an open doorway that connected both rooms that they were pressing the wrong button.

From a developer perspective, the ability to view and replay the input, actions and movement of users offers the chance to directly observe the process that lead to bugs or errors, as well as observe human behaviour during simulations. The problem with noticing issues during and after-action review is that the user has already completed the simulation by the time identifiable issues are found. If the user is observed live in real time, as allowed by the system in the paper, issues can easily be noticed and resolved without the user struggling and becoming frustrated by the lack of guidance caused by interaction or usability issues, which can be especially true for novice users (Tromp et al. 2003).

For simulations to be replicated as experienced by the user, the same build version of the application must be used during the live observation or after-action review reconstructions. If a different build version is used to reconstruct user data, it may result in the motions and actions of the reconstruction not replicating the same experience as the user. The need for consistent build versions only applies to situations where the changes from the reconstructed simulation build have a direct impact on the user interactive experience. For example, if a user picked up a box in a dataset (Build v1), but the box was no longer present in the updated version (Build v2), the actions and motions of the user would remain during reconstruction but no context would be available to understand what is happening from the observers perspective as the box is no longer exists. Therefore, backups of the published simulations should be archived to ensure when reviewing participant performance, the reconstructed simulations are not impacted by modified conditions.

In this study, we used the 'Interaction Capture' mode of VO for the recording and reconstruction of data. It is important to highlight, however, that any type of tracking

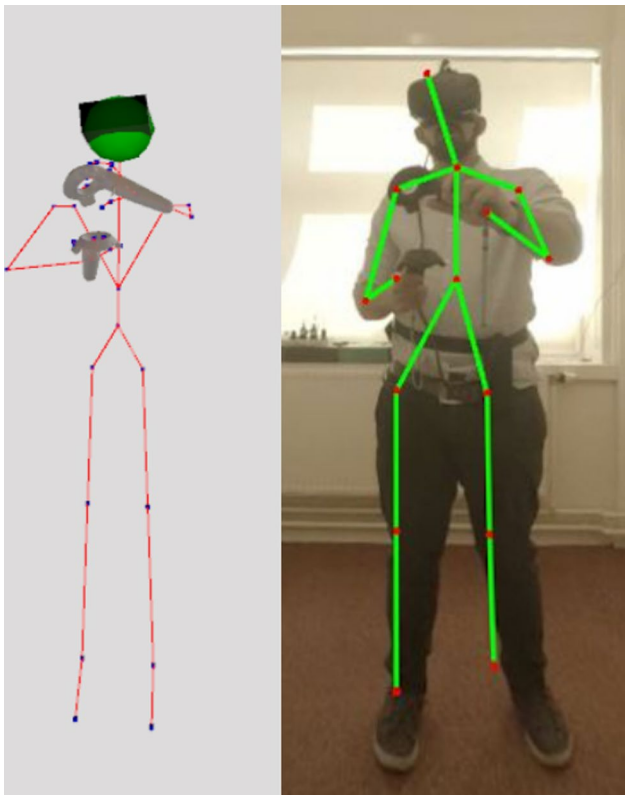

Fig. 8 Side-by-side comparison of an estimated skeleton posture obtained from the VO system (left) and a participants posture from GoPro video footage with OpenCV's pose estimation (right). The skeleton posture will deteriorate as and when the participant moves untracked joints of their body. In Howie and Gilardi's case, the lack of torso tracking caused the entire structure to rotate to match that of the HMD. Because feet and knee joints were untracked the IK system estimated the position of the knee joints relative to the ground and height of the HMD. Therefore, if the participant moved their feet from the default standing pose as seen above, the lower parts of the body would stay rigid

Fig. 9 VO was used in a commercial project for remote observation of a VR simulation during development giving the client the opportunity to monitor progress using VO WebGL and Desktop builds of the VR application. VO was later used to review trainee VR experiences, as seen here: https://www.dropbox.com/s/uaakovec3518cvx/VirtualObservationVRDemonstration.mp4?dl=0

device that supports the Unity object system for updating the transforms of the tracking device(s) will be supported by the system. This functionality aims to offer universal support to suit the needs of researchers and developers working with novel technology. To demonstrate the flexibility of the system, we used a Leap Motion hand tracking device to capture the object references our hands in Unity, whilst we interacted with a virtual car desktop screen to modify the car configuration. The movement of the hands whilst interacting with the virtual desktop screen was reconstructed, repeating the motions and button presses conducted during the recording (Fig. 3). This type of functionality can extend to other areas of tracking devices such as Face Tracking (Li et al. 2015; Olszewski et al. 2016) and Eye-Tracking (Meißner et al. 2017) which can offer greater insight into user experience. Jacob and Karn (2003) have stated that with eye-tracking, the scan path, gaze interest and fixation interest can indicate a user's intended target before they could 'actuate any other input device' (Jacob and Karn 2003, p. 589]. The inclusion of VO software will allow for these interaction devices to be saved and reconstructed, improving the data collection and analysis for studies. Using the VO software, it can streamline the process of assessing users, with reconstructed simulations able to accommodate several tracked devices and reconstruct them simultaneously with relative contextualisation to the other devices in the simulation.

In Howie and Gilardi (2019), the skeleton posture created using Unity's IK system was assessed, concluding that head and hands were tracked accurately with data provided from the VR equipment, but torso and feet tracking proved lacking since no real-world object was used to calibrate their position or orientation (Howie and Gilardi 2019). In this study, we rectified this issue by using three additional trackers, attached to both feet and the torso. Using all six tracking positions (HMD, two controllers, two feet trackers and torso tracker), we were able to achieve 'full-body' tracking of users in real-time observation and after-action review simulations. This tracking set-up could be used to monitor a user's location of core body points relative to the local tracked physical 'play-space' they have configured for their VR set-up.

Developers and researchers can use VO to analyse feasibility and usability of systems. Educators that adopt VR in their classrooms can use the data to check and assess how users have completed training procedures and verify they are correctly handling equipment or safety procedures. In Howie and Gilardi (2019), it was highlighted that participant data recorded of a fifteen-minute simulation using the VO system was always significantly smaller (around 80MB) than equal length video recordings using GoPro equipment (around 5.8GB). This continues to be true, and we demonstrated that the data can be transmitted on a network to obtain a real-time live simulation of what users are doing within a VR simulation.

To test the portability the VO system, we exported the project to a separate Unity development project which we used to create a WebGL application that replayed the data recorded during one of the validation sessions (see Fig. 6 for controllable observation demonstration). Because SteamVR does not support WebGL, VO bypasses these restrictions enabling the VR simulation to be reconstructed and observed in web browsers, which animation capture of simulation data for input modification would be unable to achieve due to the lack of SteamVR compatibility (see example in Fig. 6). The examples presented in this paper show the flexibility of VO for recording VR user simulations and reconstructing them on multiple platforms.

Datasets recorded from users with the VO system can be used to replicate the actions and motions of the user, simulating the use of VR equipment without relying on the equipment resources. The ability to share the user sessions allows external observers to analyse datasets without having access to the equipment, software and development area of the project and allowing researchers (after ethical approval and user consent) to share data with collaborators and reviewing panels, as well as allow other researchers validate research findings. Using our system, software houses can report bugs accurately to the relevant department and educators to document trainee assessments.

An issue we faced during this study was the variation of positional data of tracked objects during the action of picking up and dropping objects. We originally intended to compare the transform of each tracked device during the instance of the pickup or drop action being completed; however, we noticed slight inconsistencies in positional and rotational data between the datasets. After investigating the data recorded on the client PC and server PC, we discovered that the issue was caused by the minor modification of the controllers in real space during the fractional time difference of input state recorded and an action being captured. We anticipate this minor discrepancy in data was caused by the real-time updating of the render models which aims to keep the virtual controls consistent with their position in real world to satisfy the user experience. This minor discrepancy in pickup position does not affect the VO system as the action transform positions were consistent for both the client and server datasets. To keep action and input persistent during reconstruction the transform of the interaction devices were modified prior to the physics update of the game engine.

8 Limitations

Packet loss during the transmission of data could result in loss or corruption of simulation data as an action-frame would be skipped from the reconstructed simulation during

live observation. Like any set of data submitted over a wireless connection, there is potential for individual packets data to be lost. Unfortunately, this is a networking issue with the Unity game engine networking architecture and not our VO. We have provided potential solutions in Sect. 6, to mitigate this issue.

Recording of non-serialisable Unity components requires the developer to extend within the source-code the required variables, potentially making the 'Full Simulation Capture' mode too complex for non-technical users. We hope to improve this in the future, making the 'Full Simulation Capture' mode as user-friendly as the standard mode. One of the difficulties faced with the Full Simulation Capture version of the software was the inability to record data of the additional tracked objects dynamically. We were able to successfully record all data (variables) from objects and reconstruct them using system reflection. However, system reflection has an unavoidable impact on system performance and caused too much lag in the simulation during reconstruction.

9 Conclusion

This paper built on Howie and Gilardi's Virtual Observation system (Howie and Gilardi 2019) for observing the motions, inputs and actions of users in virtual reality simulations. VO was validated in Howie and Gilardi (2019) for after-action review of users performance in a detailed fire training simulation scenario (Fig. 1), proving that the VO system was capable of reconstructing detailed simulation from only the input and actions of a participant with no other tracking data required. In this paper, we have demonstrated the capability of the VO system for reconstructing simulations in real time with consistent datasets for live and after-action (offline) review of user sessions, capable of both full body tracking and hand tracking set-ups. We also demonstrated the ability for researchers, developers and educators to share VR simulation data to conventionally unsupported platforms (WebGL).

We conclude that these recordings are on par with Lopez et al. (2017), but offer greater clarity into high-level technical details of the simulation and the role played by the VR user (how the actions of the user affect the simulation procedure), offering context from cause and effect relationships of user's actions (Hanoun and Nahavandi 2018). With the inclusion of live observation, user performance can be immediately assessed with the potential for observers to have instant clarification to issues the user may be experiencing. For research purposes, the ability to generate visual data output to contextualise VR research can also be a key benefit for researchers and study validations. Given that screenshots and videos can often have limited contextual awareness, the ability for researchers to control the observation of user data themselves can help clarify authors' findings.

VO was incorporated into a commercial product to let clients review the functionality of a VR simulation through a WebGL visualiser without need for software to be downloaded or without the constraints of pre-recorded videos. Using the VO software, the client was able to view for themselves the interactions and protocols of a training simulation remotely from the development location without requiring VR equipment or additional software. Unfortunately, due to confidentiality agreements, we are not able to share any additional data or details at the moment (Fig. 9).

For researchers, the VO system offers greater scope for experiment complexity and reach, allowing for studies to be conducted in external locations ('in the wild') with data retrievable for analysis. For development of VR applications, this software can be of use to allow identification and replication of bugs with consistent repeatable simulations from a single capture dataset to ensure reliable analysis is conducted. After resolving a bug or issue experienced by the user, the same user dataset can be rerun to ensure that the issue captured originally is not repeated, indicating when the issue is resolved.

We conclude that VO of reconstructed simulations is a highly effective and versatile tool for researchers, educators and developers which is enhanced by the ability to observe simulations in real time. The streaming of data in real time during live observations also circumvents the need for data to be stored on external services, avoiding the issues faced by bandwidth usage experienced in Howie and Gilardi (2019).

For future work, it will be interesting to determine whether and how the VO system could be incorporated directly into an automated evaluation system that provides instant feedback (Hanoun and Nahavandi 2018) for situations where an observer is not available.

Acknowledgements The authors would like to thank Swagelok Scotland Ltd. for using our VO system to review simulation procedures, protocols and interactions during development of their training application and allowing the screenshot from their application to be included in this paper.

References

Blomberg J, Burrell M, Guest G (2007) The human-computer interaction handbook. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, chap An Ethnographic Approach to Design, pp 964–986. http://dl.acm.org/citation.cfm?id=772072.772133

Bowman DA, McMahan RP, Ragan ED (2012) Questioning naturalism in 3d user interfaces. Commun ACM 55(9):78–88

Carranza J, Theobalt C, Magnor MA, Seidel HP (2003) Free-viewpoint video of human actors. ACM TOG 22:569–577

FitzGerald E (2012) Analysing video and audio data: existing approaches and new innovations. Surface Learning Workshop 2012, Bristol, UK

Goldberg SL, Knerr BW, Grosse J (2003) Training dismounted combatants in virtual environments. Technical report, Army Research Development and Engineering Command Orlando FL Simulation

Greenberg S, Buxton B (2008) Usability evaluation considered harmful (some of the time). In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, pp 111–120

Greenhalgh C, Purbrick J, Snowdon D (2000) Inside massive-3: flexible support for data consistency and world structuring. In: Collaborative virtual environments: proceedings of the third international conference on collaborative virtual environments, vol 2000. CiteSeer, pp 119–127

Greenhalgh C, Flintham M, Purbrick J, Benford S (2002) Applications of temporal links: recording and replaying virtual environments. In: Proceedings IEEE virtual reality 2000. IEEE, pp 101–108

Hanoun S, Nahavandi S (2018) Current and future methodologies of after action review in simulation-based training. In: Proceeding of the 2018 annual IEEE international systems conference (SysCon). IEEE, pp 1–6

Howie SR, Gilardi M (2019) Virtual observation of virtual reality simulations. In: Extended abstracts of the 2019 CHI conference on human factors in computing systems, association for computing machinery, New York, NY, USA, CHI EA'19, pp 1–6. https://doi.org/10.1145/3290607.3312836

Jacob RJ, Karn KS (2003) Eye tracking in human–computer interaction and usability research: ready to deliver the promises. In: The mind's eye, Elsevier, pp 573–605

Jung B, Amor HB, Heumer G, Weber M (2006) From motion capture to action capture: a review of imitation learning techniques and their application to VR-based character animation. In: Proceedings of the ACM symposium on Virtual reality software and technology. ACM, pp 145–154

Lang P (2019) Final IK. https://assetstore.unity.com/packages/tools/animation/final-ik-14290. Accessed 04 Apr 2019

Lazar J, Feng JH, Hochheiser H (2017) Research methods in human–computer interaction. Morgan Kaufmann, Burlington

Li H, Trutoiu L, Olszewski K, Wei L, Trutna T, Hsieh PL, Nicholls A, Ma C (2015) Facial performance sensing head-mounted display. ACM ToG 34(4):47

Lopez T, Dumas O, Danieau F, Leroy B, Mollet N, Vial JF (2017) A playback tool for reviewing VR experiences. In: Proceedings of the 23rd ACM symposium on virtual reality software and technology. ACM, p 83

Meißner M, Pfeiffer J, Pfeiffer T, Oppewal H (2017) Combining virtual reality and mobile eye tracking to provide a naturalistic experimental environment for shopper research. J Bus Res 100:445–458

Olszewski K, Lim JJ, Saito S, Li H (2016) High-fidelity facial and speech animation for VR HMDs. ACM TOG 35(6):221

Petrie H, Hamilton F, King N, Pavan P (2006) Remote usability evaluations with disabled people. In: Proceedings of the SIGCHI conference on human factors in computing systems. ACM, pp 1133–1141

Reitmayr G, Schmalstieg D (2005) Opentracker: a flexible software design for three-dimensional interaction. Virtual Real 9(1):79–92

Tromp JG, Steed A, Wilson JR (2003) Systematic usability evaluation and design issues for collaborative virtual environments. Presence Teleoper Virtual Environ 12(3):241–267

Von Spiczak J, Samset E, DiMaio S, Reitmayr G, Schmalstieg D, Burghart C, Kikinis R (2007) Multimodal event streams for virtual reality. In: Multimedia computing and networking 2007, international society for optics and photonics, vol 6504, p 65040M

Wobbrock JO, Kientz JA (2016) Research contributions in human–computer interaction. Interactions 23(3):38–44