A Platform for Collecting User Behaviour Data during Social VR Experiments Using Mozilla Hubs

Thomas Röggla
t.rooggla@cwi.nl
Centrum Wiskunde & Informatica
The Netherlands

David A. Shamma
aymans@acm.org
Toyota Research Institute
USA

Julie R. Williamson
julie.williamson@glasgow.ac.uk
University of Glasgow
United Kingdom

Irene Viola
i.viola@cwi.nl
Centrum Wiskunde & Informatica
The Netherlands

Silvia Rossi
s.rossi@cwi.nl
Centrum Wiskunde & Informatica
The Netherlands

Pablo Cesar
p.s.cesar@cwi.nl
Centrum Wiskunde & Informatica
TU Delft
The Netherlands

Figure 1: Workflow overview enabled by the proposed toolchain: from data gathering in Mozilla Hubs, collection and validation on a cloud machine to final data analysis.

ABSTRACT

In recent years, a large variety of online communication tools have emerged, including social Virtual Reality (VR) platforms for interacting in a virtual world with participants being represented as virtual avatars. Given their popularity, an active area of research focuses on improving the user experience in these virtual experiences. To enable experimentation at large scale on online platforms, it is however essential to collect behavioural data (e.g. movements and audio information). In this work, we present a toolchain that enables the running of experiments using a modified version of the social VR platform Mozilla Hubs. Specifically, our toolkit enables collection and tracking of user positions and movements at a central location, enabling fine-grained analysis of user behaviour during a social VR experience. The proposed tool is available at https://github.com/cwi-dis/mozillahubs-datalogger

KEYWORDS
Social VR, data collection, data visualisation, plugin, Mozilla Hubs

CCS CONCEPTS
• Human-centered computing → Visualization systems and tools; • Information systems → Web services.

This work is licensed under a Creative Commons Attribution International 4.0 License.

MMVE ’24, April 15–18, 2024, Bari, Italy
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0618-9/24/04
https://doi.org/10.1145/3652212.3652225

1 INTRODUCTION

Over the past few years, a new era for remote communication has begun thanks to technological advances and the introduction of novel online services in the realm of Virtual Reality (VR). VR goes beyond traditional remote communication technologies, putting the users at the center of the action and providing them with a sense of immersion and new interactive capabilities. Going one step further, social VR applications enable the virtual co-presence of multiple users within the same virtual environment, allowing body interactions similar to face-to-face communication and creating new possibilities not only for remote communication but also for collaboration and social presence, redefining the way individuals engage in virtual experiences [6, 17, 18, 23].

Social VR has been used by researchers and practitioners to enable collaborative work [4, 5, 7] and design experiences in areas such as health care [16], food [19], learning and training [1, 10, 15, 27], artistic design [22] and museum exploration [20]. A key aspect
of social VR is its ability to enable embodied interactions so that users can navigate the space and interact with one another using body language and non-verbal cues as well as verbal communication. Thus, the spatial dynamics of social interaction, such as proxemics, play a significant role in understanding interpersonal relationships and communication patterns within virtual environments. Physical displacement and proxemic interactions have been analyzed to investigate which social cues are the most influential and relevant to ensure presence and immersion [14, 25]. To do so, however, researchers are in need of social VR platforms that enable the accurate logging of behavioural data (e.g. body position and rotation, interaction modalities or audio information) to analyse user behaviour [21]. Platforms such as Ubiq [9] and VR2Gather [24] were developed to enable such logging and support researchers in running user studies in controlled environments. However, such frameworks require Unity knowledge to design and develop VR experiences, which might prove difficult for designers and researchers with low technical skills in this software. Moreover, their deployment over the network needs to be orchestrated by the researchers, posing further challenges. Current commercial platforms such as Mozilla Hubs [8], Spatial.io [12], Rec Room [11] and VRChat [13] offer easy design and deployment, but do not offer data logging. This is the research gap we aim to address in this paper.

In this work, we present a toolkit that enables the running of experiments using a modified version of the popular social VR platform Mozilla Hubs [8]. The main advantage of this experimental platform relies on its versatility in working with immersive devices, such as head mounted displays, but also traditional web browsers on computers or mobile devices. Specifically, we further extend the toolkit presented in [25] such that position and movement of head and hands can be collected and tracked at a central location, enabling fine-grained analysis of user behaviour during a social VR session (Figure 1). In the following sections, we describe the implementation of the platform and we give some examples of deployment to demonstrate how it can be used to foster research in the field.

2 PLATFORM

Figure 1 outlines the workflow enabled by the system presented in this paper. The first step is data acquisition, which is realised by a plugin for the popular Social VR platform Mozilla Hubs. Then, the gathered data is streamed via HTTP to a data collection component, an optimised purpose-designed web server that validates and stores the data in a compressed format. Finally, the collected data can be visualised and checked for validity before being more deeply analysed. In following, we further describe these components.

Data Acquisition. Mozilla Hubs completely runs within a browser environment and is based on open-source libraries: ThreeJS [3] for WebGL support and AFrame [2] for integration with VR headsets. Our toolchain consists of a client-side, which runs in the user’s browser and is responsible for collection and transmission of the data and a server-side, which validates the received data and stores it for further processing. Communication between the two is enabled through the HTTP protocol using POST requests. The client-side consists of a JavaScript module which is registered in the global scope of the A-Frame library. This gives the module access to all objects within the virtual world and is executed by A-Frame on every tick of its event loop. After system initialisation, the tick function is activated and runs on every frame update of the system. The module extracts a variety of information from the browser’s DOM tree and the virtual environment. In the current version of the client-side module, system-related metrics, such as timestamp and frame rate are collected from the browser environment while behavioural data is extracted from the DOM tree. A complete list of metrics collected by these two sources are given in Table 1a and 1b, respectively. All this data is updated and saved based on the frame rate of the user’s headset. To minimise the number of requests sent to the data collection server, the data is buffered and a POST request containing all the data as a JSON-formatted payload is only sent every 4000 ticks.

Data Collection. The data collection is done on a separate cloud server which is responsible for validating and storing all session data. As shown in Figure 2, the cloud server with a single HTTP POST endpoint is implemented using the Go programming language to achieve adequate performance and keep system load to a minimum. Running as a background process, the server listens for incoming POST requests on TCP port 6000. Further, all requests are handed to the server through a Nginx reverse proxy, which takes care of CORS policy validation to allow the Mozilla Hubs clients to communicate directly with the data collection server via AJAX.

Upon reception of a request on the right endpoint, a streaming JSON decoder is instantiated, ready to receive the payload body of the POST request. Once the entire payload is received and validated, the program checks the presence of all required fields. If all required fields are present, the decoded payload is converted to a comma-separated format and appended to a compressed CSV file via a streaming GZip compressor. The server also adds a UNIX timestamp to each record, which can be used to correct possible time drift and/or inaccuracies in the timestamps received from the clients. To prevent file corruption through concurrent access, the write operation is guarded by a mutex. After a successful write, the request handler returns a message with HTTP status 200 to the client; if the submitted data did not pass validation, the server returns an error with HTTP status 400; and if the data could not be written, an error with HTTP status 500 is returned. Through the use of GZip compression, the data collected in a session, which typically amounts to about 2 GB, can be compressed to about 500 MB, keeping storage space use to a minimum. Further, by using a streaming compressor, the file handle can be held onto without having to close and reopen for every request.
The deployment of the proposed toolchain involves a series of prerequisites. Chief among them is a private instance of Mozilla Hubs that can be achieved by using Mozilla’s official AWS CloudFormation recipe. This recipe will deploy all the needed services on a selected AWS account and start them. After configuration of a custom domain, email settings for login and the admin panel, the custom client including the data collection JavaScript submodule can be checked out from Github and deployed to the running instance directly from the command line, following the guides found in the Hubs documentation. The second prerequisite to complete the system is the server to collect and validate the data. The Go-based server can be checked out from Github and, after compilation of the sources to a self-contained binary, can be launched and will start listening for incoming HTTP requests on port 6000. If the server runs on a domain different from the Mozilla Hubs instance, the clients will refuse to send AJAX requests to the data collection server because it would violate the Cross-Origin Security Policy (CORS). To address this, we encourage putting the data collection server behind a reverse-proxy such as Nginx and configure it to allow requests from the domain name of the Hubs instance. This way, Nginx handles CORS negotiation and hands off authorised requests to the server. From this point onward, the logger is automatically enabled for any client that joins a room with the parameter ?log=true appended to the query string of its URL and will start streaming content to the data collection server. Arrival of data can be monitored on the standard output of the server.

Our proposed toolkit can be essential to investigate interactions in social VR. The system [25] on which we based our work has enabled investigation on digital proxemics, an emerging area focused on understanding the human use of space within virtual environments. Specifically, it has been used to analyse how people use space in a virtual academic workshop [25] and how personal displacement changes between VR and traditional desktop users [26]. In both cases, the system supported the gathering and collecting of data from participants allowing data visualisations as shown in Figure 3 and enabling new behavioural analysis. The use of our novel toolkit will enable the augmentation of collected data, paving the way to new investigations such as the impact of the design of virtual environments and how interaction unfolds in social VR.

It should be noted, however, that Mozilla recently announced the sunset of their AWS deployment recipe, complicating the use of their private instances. As alternative, Mozilla started offering a new professional plan, which outsources the hosting and management completely to Mozilla, while still allowing to deploy custom clients. This makes use of Hubs easier, as the management of the infrastructure is completely taken care of, albeit slightly changing the deployment process of our toolchain.

### 4 CONCLUSION

This work described an easy-to-deploy tool for collecting and storing behavioural data from the popular social VR tool Mozilla Hubs. Our solution can be integrated into a running instance of Hubs to gather metrics from participants within a browser environment. Collected data is stored off-site using an optimised purpose-built web server in a compressed format, making it possible to store substantial amounts of data without placing too much load on the host system.

---

**Table 1: User metrics collected by the toolchain**

<table>
<thead>
<tr>
<th>User metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>Device’s UNIX timestamp</td>
</tr>
<tr>
<td>fps</td>
<td>Current frame rate</td>
</tr>
<tr>
<td>uuid</td>
<td>Random UUID</td>
</tr>
<tr>
<td>user_agent</td>
<td>Device user agent</td>
</tr>
<tr>
<td>isBrowser</td>
<td>Device type</td>
</tr>
<tr>
<td>isLandscape</td>
<td>Device orientation</td>
</tr>
<tr>
<td>isWebXRAvailable</td>
<td>VR availability</td>
</tr>
<tr>
<td>avatarID</td>
<td>Avatar ID</td>
</tr>
<tr>
<td>isHeadsetConnected</td>
<td>Headset connection status</td>
</tr>
<tr>
<td>isRecording</td>
<td>Recording status</td>
</tr>
<tr>
<td>pathname</td>
<td>Current URL</td>
</tr>
<tr>
<td>urlQuery</td>
<td>Query section of the URL</td>
</tr>
<tr>
<td>isLoaded</td>
<td>Has user finished loading</td>
</tr>
<tr>
<td>isEntered</td>
<td>Has user joined room</td>
</tr>
<tr>
<td>isFlying</td>
<td>Is user flying</td>
</tr>
<tr>
<td>isVisible</td>
<td>Is user visible</td>
</tr>
<tr>
<td>isSpeaking</td>
<td>Is user speaking</td>
</tr>
<tr>
<td>isMuted</td>
<td>Is user muted</td>
</tr>
<tr>
<td>volume</td>
<td>Current user volume</td>
</tr>
<tr>
<td>rigPos{X, Y, Z}</td>
<td>Avatar position</td>
</tr>
<tr>
<td>rigDir{X, Y, Z}</td>
<td>Avatar direction</td>
</tr>
<tr>
<td>rigQuat{X, Y, Z, W}</td>
<td>Avatar quaternion rotation</td>
</tr>
<tr>
<td>povPos{X, Y, Z}</td>
<td>POV position</td>
</tr>
<tr>
<td>povDir{X, Y, Z}</td>
<td>POV direction</td>
</tr>
<tr>
<td>povQuat{X, Y, Z, W}</td>
<td>POV quaternion rotation</td>
</tr>
</tbody>
</table>

---

**Figure 3: Use case of instrumented Mozilla Hubs in an academic workshop** [25].

Data Visualisation. The presented toolchain also offers the possibility to perform a quick sanity check on the collected data by taking the data stream and replaying it using a crudely rendered model of a head and hands, placed into a 3D environment. This allows researchers to quickly assess the viability of the collected data before analysing it. From this point onward, the data that consists of a CSV format with a column for all the properties in Table 1, indexed and sorted by timestamp. The file can be decompressed and analysed using conventional data analysis tools. For reference, Figure 3 shows an example of a visualisation generated from results obtained using the toolchain during a previous study.
ACKNOWLEDGMENTS
This work was in part supported through the European Commis-
sion Horizon Europe program under grant 101070109 TRANSMIXR (https://transmizr.eu/).

REFERENCES