Computers & Graphics 106 (2022) 66–76

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on WEB3D 2021 Selected

Designing, testing and adapting navigation techniques for the immersive web

Ahmed Kamal, Carlos Andujar *

ViRVIG, Universitat Politècnica de Catalunya - BarcelonaTech, Jordi Girona 1-3, Barcelona, 08034, Spain

ARTICLE INFO

Article history:
Received 1 March 2022
Received in revised form 22 May 2022
Accepted 25 May 2022
Available online 2 June 2022

Keywords:
Virtual Reality
WebXR
VR navigation
3D interaction techniques
Immersive Web

ABSTRACT

One of the most essential interactions in Virtual Reality (VR) is the user's ability to move around and explore the virtual environment. The design of the navigation technique plays a crucial role in the user experience since it determines key usability aspects. VR devices allow for an immersive exploration of 3D worlds, but navigation in VR is challenging for many users, due to potential usability issues related to specific VR controllers, user skills, and motion sickness. Although hundreds of interaction techniques have been proposed for this task, VR navigation still poses a high entry barrier for many users. In this paper we argue that adapting the navigation technique to its context of use can lead to substantial improvements in navigation usability and accessibility. The context of use includes the type of scene, the available physical space, as well as the profile of the user. We present a test platform to facilitate the design and fine-tuning of interaction techniques for 3D navigation. We focus on mainstream VR devices (headsets and controllers) and support the most common navigation metaphors (walking, flying, teleportation). The key idea is to let developers specify, at runtime, the exact mapping between user actions and locomotion changes, for any of the supported metaphors. Such mappings are described by a collection of parameters (e.g. maximum speed) whose values can be adjusted interactively through a GUI, or be provided by user-defined code which can be edited at runtime. Feedback obtained from developers suggests that this approach can be used to quickly adapt the navigation techniques to various people including persons with no previous 3D navigation skills, elderly people, and people with disabilities, as well as to the type, size and semantics of the virtual environment.

© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

The affordability of mainstream VR devices, including shell (phone-based) VR headsets and standalone devices, has substantially increased the amount of users who can try VR experiences at home [1]. Current VR experiences cover a broad range of user needs, including entertainment, education, virtual tourism, physical activity and social interaction. Besides VR games from major development companies, the recent adoption of WebXR standards in desktop, mobile and VR web browsers greatly facilitates the creation of VR experiences beyond entertainment, for example to disseminate Cultural Heritage (CH). Thanks to the advances in laser scanning and photogrammetry techniques, the 3D digitization of objects, buildings and sites has become a common practice [2,3] in order to document, preserve, study and disseminate CH. As a consequence, there is an increasing number of publicly-available CH 3D models, some of which can be explored in VR, either through 3D model sharing platforms (e.g. SketchFab) or through specific websites from the content creators, which often rely on open-source frameworks for the creation of interactive web presentations [4–6].

The web is arguably the ideal platform for the dissemination of 3D models. Web-based applications benefit from one of the essential features of the web: since it is open and based on well-defined standards, anyone can publish content which can be accessed by anyone [7]. The web is by far the most widely available platform, and web browsers exist for desktop computers, tablets, smartphones and AR/VR headsets. Furthermore, distributing content updates is easier for the web [8] and thus the web is an excellent medium for dynamic 3D content. Additionally, modern web-based VR experiences require no software installation, thus also reaching users with security concerns.

WebXR is a collection of open standards to render 3D scenes in web browsers running on AR/VR hardware. The *WebXR Device API* provides a platform-independent interface to access VR and AR devices [7]. This API allows web developers to get input from interaction devices and VR controllers, and render the 3D scene to the chosen device at the appropriate frame rate. WebXR has been used successfully to create immersive cultural heritage experiences [9,10], digital books [8], visual analytics [11,12], and training applications [13]. WebXR has also been adopted by

* Corresponding author.
E-mail address: andujar@cs.upc.edu (C. Andujar).

https://doi.org/10.1016/j.cag.2022.05.015
0097-8493/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

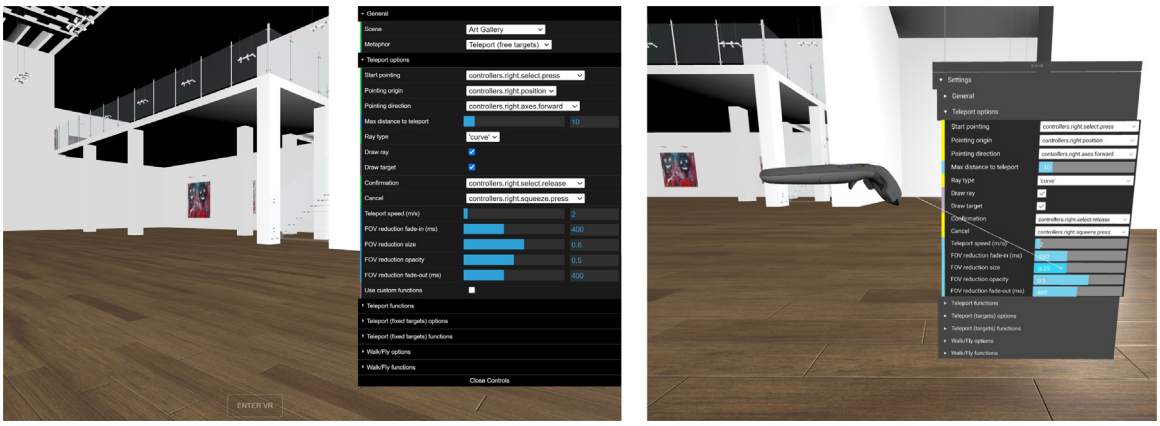

Fig. 1. Our framework characterizes a navigation technique in terms of a base interaction metaphor (in these examples, teleportation) and a collection of user-adjustable parameters. Parameter values can be adjusted in runtime through GUIs. Left: Teleportation options on the non-VR GUI, as shown on a desktop browser. Right: The same options on the VR GUI, available both in VR headsets and on desktop browsers running the WebXR API Emulator. Art Gallery model by 3DTwins licensed under CC-BY-4.0.

major 3D model sharing platforms (e.g. SketchFab) and it is supported by popular WebGL frameworks such as A-frame, ThreeJS, BabylonJS and Godot. In this paper we focus on navigation for web-based VR experiences based on the WebXR open standard.

Navigation is one of the major user tasks in 3D virtual environments, and interaction techniques for 3D navigation have received substantial attention [1]. A recent study has identified and characterized over one hundred navigation techniques for VR [14]. The reason for such an increasing variety of navigation techniques is twofold. On the one hand, AR/VR software and hardware technologies are in constant evolution, opening new opportunities for 3D interaction. On the other hand, current interaction techniques for 3D navigation have some limitations. For example, many works have pointed out the inherent trade-off between simulation sickness and accessibility [14]. As a result, no universal technique for 3D navigation exists that works well in all situations. The suitability of a certain interaction technique depends on many factors including the type of application (e.g. game vs. virtual tourism), the virtual environment (e.g. museum vs. archaeological site), the target VR platform (e.g. Google Cardboard vs. HTC Vive), the user profile (e.g. gamer vs. art historian), and the physical environment (sitting VR vs. large tracking room).

As a consequence, there is an increasing need for more exploratory studies on VR navigation techniques, as well as for mechanisms to facilitate the adaptation of existing techniques to those individuals for which VR navigation might imply a high entry barrier, such as people with disabilities or no prior 3D/VR experience.

In this paper we aim to facilitate the design, testing and fine-tuning of interaction techniques following the major navigation metaphors for VR (walking, flying and teleportation). At the core of these interaction techniques is the mapping of user input onto virtual camera changes, although other components also play an important role for the user experience, such as techniques for alleviating motion sickness [15].

We describe a web platform (using JavaScript, WebXR and ThreeJS) for defining and evaluating different variants of major navigation metaphors (Fig. 1). The platform has been designed to support easy experimentation on interaction techniques for navigation, where developers can easily create and test new mappings. Each of the supported navigation metaphors (e.g. walking) is defined by a collection of parameters (e.g. start trigger and maximum speed). The platform allows users and developers to modify the values of these parameters, either as values that can be modified through a GUI (e.g. a drop down menu to select which controller button initiates the locomotion, or a slider to choose the maximum speed) or as user-provided functions (JavaScript snippets) also editable through the GUI.

This paper is an extended and significantly revised version of a previous conference paper [16]. Compared to [16], we provide a more in-depth discussion of the presented technique (including examples of JavaScript snippets). Furthermore, we have added and tested a new GUI which is available in VR mode, and thus all parameter changes can be done directly from the VR headset; our previous prototype required most parameters to be changed on a desktop PC using the WebXR API emulator. Additionally, we discuss the collected feedback from some developers who tested our approach and provided valuable insights about its benefits and limitations. Their feedback suggests that our tool, by providing fast iteration cycles for customizing navigation to particular users and scenarios, can substantially improve the fitting of the navigation technique to its context of use. We believe this is a step towards more inclusive VR applications. Many of the navigation techniques available in games and other applications assume certain user capabilities that are not universally present. Using appropriate custom mappings facilitates lowering the VR entry barrier for these users.

2. Previous work

2.1. VR navigation

The essential user tasks in 3D virtual environments are selection, manipulation and navigation. Although all three tasks influence to some extent user performance, physical effort, user comfort, and other aspects determining the quality of the user experience, navigation plays an essential role in motion sickness. A proper design of the navigation technique could be the difference between enjoying the sense of presence in another world, and suffering discomfort, disorientation, and nausea [1].

The most natural way to travel in VR is real walking. Many current VR headsets support room-size tracking and therefore real walking is an excellent option for small-scale navigation. However, a natural walking interface is clearly not sufficient. VR navigation techniques must allow users to navigate virtual environments that extend far beyond the size of the physical space, to cover large virtual distances with small physical effort, and to inspect virtual environments at different heights and perspectives [1]. These limitations can be overcomed by using virtual

locomotion techniques, which can be classified into walking-based, steering-based, selection-based and manipulation-based techniques [1,17].

Walking-based techniques mimic the natural way of moving around. These techniques include real-walking as well other techniques involving repetitive motor actions such as head-bobbing and arm swinging [1]. Depending on the resemblance to the human gait cycle, walking-based techniques are classified as full gait, partial gait, and gait negation techniques [1,17]. Redirection techniques [18–20] attempt to keep the user within the limits of the physical space while being able to travel in a larger virtual environment. Partial gait techniques such as walking-in-place (WIP) have attracted much attention since its origins [21]. Finally, gait negation techniques aim to provide the full gait cycle while keeping the user stationary, at the expense of dedicated mechanical devices such as treadmills [1].

Steering-based techniques provide a continuous control of direction [17]. Virtual walking and flying metaphors are examples of steering-based techniques. The virtual direction can be controlled using the user's body (e.g. gaze, hand, body leaning) or through physical steering props [17]. Steering techniques are very popular in 2D/3D videogames (through the use of joysticks and game controllers), but they are less popular in VR games since they can trigger motion sickness.

Selection-based techniques greatly simplify navigation by letting users focus on where to get to instead of how to get to a specific location [17]. A widely known selection-based technique is teleportation [22], which moves the virtual camera to the chosen destination either instantly [23] or gradually [24,25]. Instant teleportation is very popular in the VR industry as it helps reduce the incidence of VR sickness [26,27].

Manipulation-based techniques achieve navigation by allowing users to manipulate their position, orientation and size, using gestures that either control the virtual camera (e.g. Eyeball-in-hand [28], World-in-Miniature [29]) or the virtual world (e.g. Scene-in-hand [28]). Actually, any manipulation technique can be used to control the virtual camera by *grabbing the air* gestures [17].

Despite a number of interaction metaphors and variants have been proposed in the literature, researchers agree that there is not a unique "best" technique for all scenarios, users and tasks.

2.2. Motion sickness and accessibility

A major issue with VR headsets is that users often experience adverse symptoms such as nausea, dizziness, and eye strain [17,30]. These symptoms are often referred to as motion sickness or simulator sickness. Despite the technology improvements regarding head tracking and end-to-end latency, a recent study with high-end VR headsets has shown that they still cause a greater level of sickness compared to desktop displays [31]. A number of theories have been proposed to explain simulator sickness [17]. Two well-known theories are the cue-conflict theory [32] and the postural stability theory [33]. Visually induced motion sickness (VIMS) has been linked to the sensation of illusory self-motion (vection), and although there is no direct causality between VIMS and vection, strong VIMS is rare without perceived self-motion [33]. Common methods for alleviating motion sickness are FOV reduction, brightness reduction, and independent visual background [15,30]. As stated above, some techniques such as instant teleportation lack optical flow and thus are unlikely to trigger motion sickness.

Furthermore, some works have pointed out that there is an inherent trade-off between accessibility and simulation sickness (see [14] for a review). In other words, if the navigation technique mimics real-world movement, it is likely to minimize motion sickness but at the expense of making the application less accessible to people with limited mobility. Conversely, if the navigation technique departs from real-world movement and embraces a more abstract mapping, for example requiring the user to just point or look at the intended target, we improve accessibility but we also increase the discrepancies between visual and vestibular stimuli which might result in user discomfort and motion sickness.

So we can conclude that no universal technique for 3D navigation exists that works well in all situations. The suitability of a certain interaction technique depends on many factors including the virtual environment, target platform, and user profile.

2.3. Inclusive VR

VR is an excellent medium to access new worlds and experiences, and as such it is attracting interest from many people. However VR applications are often not designed to be inclusive to all users. Besides motion sickness issues, most VR applications require users to perform some physical movements which could be an important barrier. People with functional diversity might not be able to experience VR without assistance. Some recent works address the problem of inclusive VR design, as well as adapting VR applications for people with disabilities, using either physical hardware-based solutions or conceptual tools [34]. Mott et al. [35] analyze alternative sensors to get user inputs from users with movement limitations, including motion, eye gaze, and audio sensors. Brain computer interfaces [36] are becoming a promising tool to provide VR control for people with serious motor function problems. Despite these encouraging works, many users are being left out from the vast majority of VR experiences in the immersive web [34].

2.4. Evaluation testbeds for VR navigation

A large number of systematic reviews and evaluation testbeds have been proposed for VR locomotion [1,37–39]. A recent example can be found in [40], where the authors propose a methodology for evaluating navigation techniques. Different metrics concerning users' performance and experience are discussed, including accuracy, input sensitivity, responsiveness, level of control, error-proneness, presence, motion sickness, physical effort, comfort, mental workload, cognitive demand, ease of use, intuitiveness, appropriateness, learnability and enjoyability [40]. These metrics can be assessed during the execution of a number of navigation tasks in a set of representative scenarios. Instead, in this paper we focus exclusively on how to facilitate the design and fine tuning of navigation techniques with immediate iteration cycles. We thus put the emphasis on adapting/tweaking navigation techniques to a particular context of use, rather than extensively evaluating a given technique on more general setups.

3. Our approach

3.1. Adapting the navigation to the context of use

We propose a tool to facilitate the design, testing and fine tuning of navigation techniques. The tool is based on the following premises: (a) adapting the navigation technique to a specific scenario can improve the user experience; for example, a narrow cave and a train station might require completely different navigation techniques; (b) adapting the technique to the intended primary task (search for a target, explore a virtual scene, or maneuver obstacles [17]) and scene semantics can also improve usability; for example, users exploring a CH site with mural paintings at different heights are likely to wish to zoom into the

Fig. 2. Left: Teleportation example. Once the target location is confirmed, the viewpoint moves to it. Right: Teleportation example with predefined nodes (shown here as yellow columns). Richard's Art Gallery - Audio Tour by shinymagic licensed under CC-BY-4.0. Art Gallery model by 3DTwins licensed under CC-BY-4.0.

paintings); (c) adapting the technique to specific user profiles (e.g. art historians) and functional varieties can make the application more inclusive; (d) fine tuning the different parameters and design options that define a navigation technique can make a large difference; very often the devil is in the details, and a poor choice for some parameter (e.g. size of predefined teleportation nodes) can make a navigation technique usable in some specific scenarios but unusable for the rest; (e) all the adaptations above are practical if they can be implemented and tested in an easy way, with nearly immediate iteration cycles, complete control of the mappings, and arbitrary runtime modifications.

3.2. Flexibility when adapting navigation techniques

A particular navigation technique must fully determine the mapping between certain user inputs and the corresponding locomotion changes. One way to approach the customization of navigation techniques is to view them as the combination of a certain base navigation metaphor (e.g. teleportation) along with a collection of parameters (e.g. how the pointing direction is defined) specific to that metaphor. Following this approach, we can distinguish three different levels at which users and designers can customize a navigation technique.

The first option is parameter-level adaptation. The designer chooses a specific base navigation metaphor (e.g. walking through continuous steering) and simply modifies predefined parameters, such as the maximum speed (e.g. 2 m/s). The designer can also specify non-numerical choices, such as which button must be used to start navigation, but available choices are predefined.

A more advanced option is snippet-based adaptation. Again, the designer chooses one of the predefined metaphors (e.g. teleportation), but this time parameter values can be provided also through user-defined functions (e.g. a function computing the speed in terms of the context). These code snippets allow designers to define an arbitrary mapping between user input and viewpoint changes, with the only limits imposed by the selected base metaphor.

A third level of adaptation, not directly supported by our prototype (that is, not doable directly from the GUI in real time) is user-defined metaphor. At this level, designers would be able to implement completely new locomotion metaphors (e.g. mimicking swimming), beyond those already supported by the framework.

Our prototype does support the first two levels of adaptation. Since the first level requires no programming skills, modifications can be done by the developer while testing a specific scenario, as well as by the end user at runtime. Unfortunately, this level is insufficient, for example, to get complete control on the mapping between user actions and locomotion changes. For example, one designer might want to automatically modify the speed whenever the user gets close to specific parts of the scene, or to adapt the speed to the height above the ground. We thus also support the second level through custom functions (Section 3.6).

3.3. Base navigation metaphors

Our prototype supports three major navigation metaphors: free teleportation (any target destination can be selected), teleportation nodes (target destinations are constrained to predefined nodes) and flying/walking (steering-based navigation). Most commercial games for VR headsets provide navigation techniques in one of these three categories. We describe below the basic behavior of these metaphors; parameters for customizing these techniques are described in Section 3.5.

In *free-target teleportation*, the user points at any target location and the application automatically moves the viewpoint to the target location [22–25]. Target locations are constrained to intersect geometry identified as floor (using e.g. a separate mesh for the floor, or ignoring surfaces whose normal vector deviates substantially from the vertical direction). At the destination, the height of the viewpoint is automatically adjusted so that the virtual floor matches the physical floor. See Fig. 2-left.

The *teleportation nodes* metaphor is similar to free-target teleportation, but target locations are limited to a predefined collection of nodes (Fig. 2-right). Teleportation nodes are suitable for scenes featuring distinctive points of interest. Some shell (phone-based) devices often include no buttons and thus are very limited in terms of navigation. Teleportation nodes can be used also on these devices since the predefined targets can be selected through a reticle cursor.

The *Walking/flying* metaphor allows users to walk (grounded navigation) or fly through the virtual environment, using a steering-based technique [17,26]. Any flying technique can be converted into a walking technique by either limiting the camera motion to the horizontal plane or by recomputing the camera height so that the distance to the physical floor matches that of the virtual floor. The later option requires ray casting but allows users to step over low obstacles.

3.4. GUIs for parameter configuration

A key issue for fast experimentation is to let users (and interaction designers) modify the behavior of the base techniques through a GUI exposing the different parameters that define a

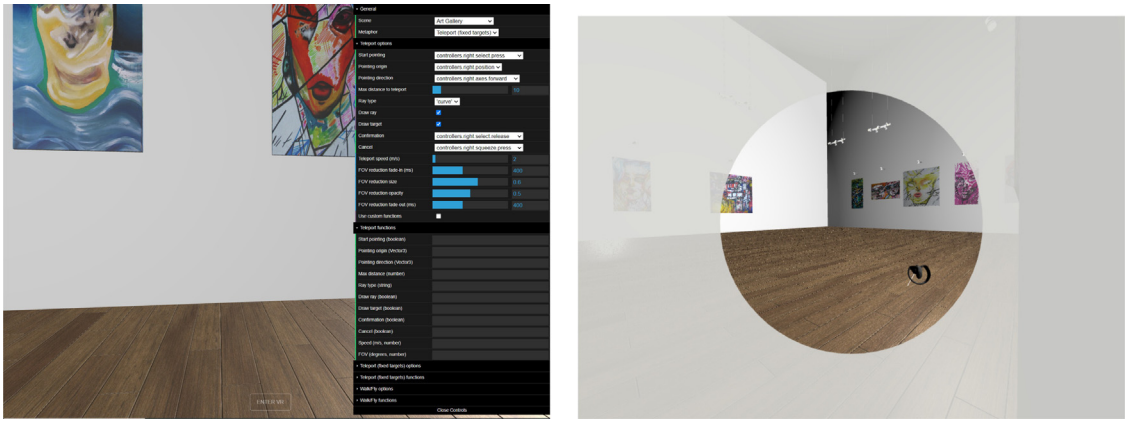

Fig. 3. The panel on the margin shows the 2D GUI for configuring teleportation techniques. Right: FOV reduction example. Art Gallery model by 3DTwins licensed under CC-BY-4.0.

specific incarnation of the navigation metaphor. Our prototype includes two different GUIs. The *2D GUI* is the preferred option when the application is not in VR mode. On a desktop PC, the 2D GUI can be operated with mouse and keyboard. Conversely, the *VR GUI* widgets are actually 3D geometry that is rendered together with the 3D scene, and thus it is also accessible when using a VR headset (as well as on a browser running on a desktop PC, through the WebXR API Emulator). Both GUIs show the same menu options and parameters (Fig. 1), effectively showing two different views of the same set of settings. All the discussion below assumes the use of the GUIs to customize the navigation technique.

3.5. Adapting the base metaphors through predefined options

3.5.1. Free teleportation

Users and designers can modify the behavior of the base teleportation technique through the GUI. The parameters that can be adjusted (Fig. 4) were decided taking into account well-established analyses on the different subtasks involved in selection-based techniques [17,41]. Fig. 3 shows the 2D GUI of our current prototype for teleportation customization. We now briefly describe the different parameters and their predefined options. The first group is concerned with the target selection part, which is based on classic virtual pointing with a selection ray.

- Start pointing: this indicates which user input initiates the selection (and causes the selection ray and/or selection target to be shown). Available choices include specific buttons (*select button*, *squeeze button*) of the left/right controllers, as well as an "always active" option. We also distinguish between press and release events. In combination with the *Confirmation* and *Cancel* triggers below, the parameters can be adjusted to require the user to hold the chosen button while pointing, or allow users to confirm/cancel the teleportation operation with another button.
- Pointing origin: origin of the selection ray. Options include the position of the head and those of the available controllers.
- Pointing direction: this option specifies how to compute the base pointing direction. Available choices include the controllers' main directions (forward, up, right...), and those from the user's head orientation.
- Max distance to teleport: we allow the user to limit the maximum distance for selectable target positions. This is especially suitable for curved rays (Fig. 2).

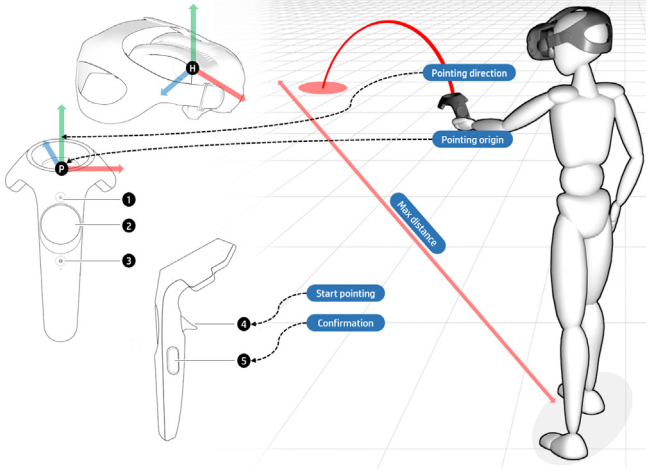

Fig. 4. Subset of the parameters that can be adjusted for teleportation. The dotted arrows show one possible mapping.

- Ray type: whether to use a straight line segment or a curved selection ray. Curved rays provide a more equally-spaced mapping between ray orientations and target distances.
- Draw ray: enable or disable the rendering of the selection ray.
- Draw target: enable or disable the rendering of the target location.
- Confirmation: which user input confirms the target selection and thus initiates the teleportation. Available options are similar to those for *start pointing* above. The teleportation is canceled if the target position is not valid (too close to an obstacle, or not over a surface recognized as floor).
- Cancel: which user input manually aborts the teleportation operation. Available options are similar to those for *start pointing* above.

The second group of options refer to the actual teleportation part, as well as FOV reduction to reduce motion sickness. See the accompanying video for a demonstration of specific options.

- Teleport speed (m/s): camera speed towards the target, or 0 for instant teleportation.
- FOV reduction fade-in: time (usually a few milliseconds) to gradually reduce the FOV.
- FOV reduction size: size of the circular portion of the FOV that is not affected by FOV reduction.

- FOV reduction opacity: opacity of the black region outside the reduced FOV.
- FOV reduction fade-out: time to gradually restore the FOV to its normal value.

3.5.2. Teleportation nodes

Similarly, our GUIs include options for teleportation with predefined nodes. The major difference is that predefined nodes are rendered as 3D proxies (e.g. yellow columns in Fig. 2). Users must point at these proxies, so the target selection part is essentially a typical 3D object selection task. The options that can be configured are the same as above, but without the *Ray type* option. The scale of the 3D objects used as proxies is also adjustable.

3.5.3. Walking/flying

For the sake of brevity, we only describe walking/flying options related to viewpoint changes. FOV reduction techniques are similar to those in previous metaphors.

- Start trigger: which user input (e.g. a button press) initiates the movement.
- Stop trigger: which user input (e.g. a button release) terminates the movement.
- Motion direction: how to compute the forward direction. Available choices include the controllers' forward direction, the user's head.
- Allow flying: this option determines whether the user is allowed to fly. If not enabled, the direction above is always projected onto the horizontal plane.
- Maximum speed and acceleration: when the user initiates the navigation, the initial speed is 0. The application updates the speed every frame (using the delta time from the previous frame) according to the chosen acceleration, until the speed reaches the maximum speed.

For those options specifying the binding with a particular input data source of event, the GUI drop-down menus can be configured to show either an easy-to-read description (e.g. "Left Controller - Select button is Pressed") or a string with the actual JavaScript expression that would reproduce this behavior (e.g. "controllers.left.select.press"). The later option simplifies writing custom code snippets, which are discussed below.

3.6. Adapting the base metaphors through custom functions

3.6.1. Motivation

Although the options above allow users to replicate many navigation techniques available for mainstream VR headsets, this adaptation is still limited to a collection of predefined choices. The most innovative part of this work is the idea of allowing experimenters to quickly define programmatically their own mappings between user actions (inputs) and navigation control (outputs). The user is free to provide Javascript (JS) code snippets for a collection of metaphor-specific functions defining all aspects of the navigation technique. There is a default implementation for all these functions (matching the value of its associated GUI parameter), so users can implement only a subset of them. These functions were inspired by Computer Graphics *shaders*: special pieces of code that run at specific stages of the graphics pipeline, and that can be customized to achieve different effects. In the same sense that shaders have a well defined task and execution framework, our custom functions also have a very clear purpose, for example, computing the maximum camera speed depending on the application state and the user input.

One can argue that developers and UX designers can always write these functions just as part of their JS modules. However, our application allows these functions to be *written directly on specific GUI components* (Fig. 5). Furthermore, the functions can be edited at any time and are evaluated at runtime. For example, we can navigate to a specific room, and test different mappings immediately without reloading the application. This method makes testing iteration cycles much shorter and thus allows for a more comprehensive testing of different mappings in varied situations.

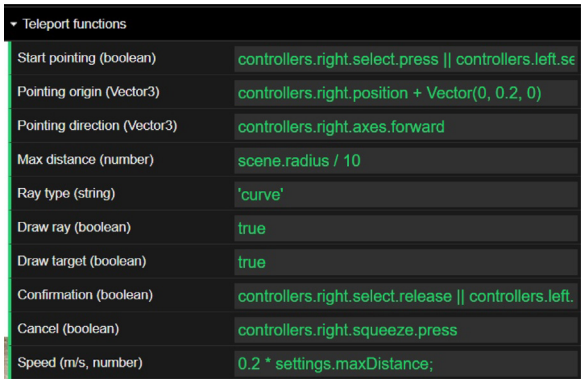

Fig. 5. 2D GUI with some code snippets for teleportation.

Table 1
Subset of the variables we expose for user-defined functions, with some useful expressions.

Variable	Description
scene	ThreeJS scene
camera	ThreeJS user camera
camera.position	Camera position
camera.axes.forward	World-space forward direction of the camera
controllers.right	Right VR controller
controllers.right.position	World-space position of the controller
controllers.right.axes.forward	World-space forward direction
delta	Delta time between two frames

3.6.2. Exposed variables

In order to facilitate writing these custom functions, we expose all relevant variables. Table 1 shows a subset of the exposed variables and sample expressions. The GUI drop-down menus (Fig. 3-left) show further examples.

3.6.3. Teleportation custom functions

We now describe a subset of the custom functions experimenters might provide to define precisely the behavior of teleportation. We indicate the expected JS return type within parentheses. Unless otherwise stated, the functions below are invoked automatically every frame. All functions have a matching parameter in the GUI. Entering code for a function overrides the value of its associated parameter, and leaving the function blank causes the application to use the GUI parameter.

- *Start pointing (boolean)*: target indication will be initiated as soon as this function returns true.
- *Pointing direction (Vector3)*: function providing the 3D vector to be used as pointing direction.
- *Confirmation (boolean)*: teleportation target is confirmed as soon as this function returns true.
- *Cancel (boolean)*: teleportation is canceled if this function returns true.
- *Speed* (m/s, *number*): function providing the speed for gradual teleportation.

See the complete list (including FOV reduction behavior) in the 2D GUI components of Figs. 3 and 5. Notice that custom functions match the breakdown for the different configurable options. We

found this feature to be essential to fully determine the behavior of the navigation technique when some few custom functions are provided: for each parameter (e.g. teleport speed), the application checks if the associated custom function is non-empty. If so, the application evaluates the function and uses its return value. Otherwise, the parameter takes the value from the corresponding GUI widget.

We considered alternative factorizations of the different subtasks involved in teleportation. For example, we could require the user to provide a single *target* function returning the selected target position, instead of asking separately for the target selection subtasks. Our factorization provides multiple advantages. On the one hand, it matches that of the GUI-adjustable options, and thus options are easier to understand and remember. On the other hand, the proposed factorization facilitates code reusability, and simplifies development since empty custom functions take the default value shown on the GUI. For example, the experimenter might chose the provide a custom function just for the pointing direction while reusing the default behavior for the rest.

Custom functions for the Teleportation Nodes metaphor are essentially the same as above.

3.6.4. Walking/flying

Here we describe the most relevant user-defined functions for walking/flying. Each frame the camera is displaced along the forward direction.

- *Start trigger (boolean)*: camera movement will start as soon as this function returns true.
- *Stop trigger (boolean)*: camera movement will stop as soon as this function returns true.
- *Motion direction (Vector3)*: function providing the 3D vector to be used as forward direction.
- *Max speed (number)*: function providing the maximum speed.
- *Acceleration (number)*: function providing the acceleration.

Functions returning the speed and acceleration values (as well as all other custom functions) might use exposed variables (Table 1) such as the delta time, scene properties (e.g. bounding box diagonal), camera properties (e.g. location, distance to floor), head/controller properties (e.g. orientation), as well as all settings shown in the GUI.

4. Implementation details

Our prototype was implemented using HTML and JS along with a collection of JS modules. The most relevant modules we used are Three.JS (a framework for WebGL that also supports WebXR), datGUI.JS (for the 2D GUI), datGUIVR.JS (for the VR GUI) and Tween.JS (for animation effects).

The evaluation of the code snippets is done using the JS eval() function with the user-provided code inserted into a predefined code template. Listing 1 shows how our internal code handles the evaluation of the snippet that determines the maximum speed (omitting exception handling for clarity). Simple expressions (e.g. controllers.left.position.x) are evaluated directly, whereas code including a *return* statement is treated as the body of an arbitrary anonymous function (which can include variable declarations, as well as conditional and iterative sentences).

Listing 2 shows a few self-explanatory examples of JS snippets that can be used for customization. In these examples the return statement has been included for clarity, though it can be omitted as the snippets consist of a single expression.

```
// if the snippet is empty, return the GUI value;
// otherwise evaluate it and return the new value
function UpdateValue(snippet, value) {
 if (!snippet) return value;
 if (!snippet.includes("return"))
  return eval(snippet);
 else
  return eval('(function() {' + snippet + '}())');
}

// use example
speed = UpdateValue(speedFunction, speed);
```

Listing 1: Internal code we use in our prototype for evaluating a JS code snippet. We omitted exception handling for clarity.

```
/// Adjusting the maximum speed
// Example 1: constant 1 m/s
return 1;
// Example 2: speed depends on camera height
return 1 + (0.5 * camera.position.y);
// Example 3: speed depends on controller orientation
return 1.5 * controllers.right.rotation.x;

/// Defining a direction (for pointing, walk/fly...)
// Example 1: camera-based
return camera.forward;
// Example 2: controller-based
return controllers.right.axes.forward;
// Example 3: controller-based + gravity
return controllers.right.axes.forward + Vector3(0,-0.1,0);

/// Triggers (to start teleportation, start walking...)
// Example 1: button press
return controllers.right.select.press;
// Example 2: button press (alt)
return controllers.right.squezze.press;
// Example 3: controller orientation threshold
return controllers.right.rotation.x < 0.5;
```

Listing 2: Simple examples of JS snippets that can entered through the GUI.

Although our tool focuses on 6-DoF devices, we also provide specific methods for 3-DoF devices such as cardboard-like devices. We support triggers based on fixing the state (position, orientation) of a VR device (headset, controllers). For example, this allow users to trigger teleportation when looking at a target for some time. For this task, we provide a function with the following prototype:

```
stable(expression, time, threshold)
```

that returns true if and only if the expression has not varied beyond the given threshold for the specified time period. The function behaves differently according to the type of the expression, in order to support numerical values as well as point, vectors, quaternions and Boolean expressions. For example, if the expression "x" evaluates to a number, the call

```
stable("x", t, e)
```

will check if $\max\{|x_i - \bar{x}|\} < \varepsilon$, where the maximum and mean are computed over the values x_i resulting from the evaluation of the expression "x" at all the previous frames covering the given period t. Every time the function returns true, data from the previous frames is reset so that further calls to the function with the same expression will return false until a new complete time period is available.

By default, Vector3 objects are interpreted as points and thus the threshold is assumed to indicate the maximum Euclidean

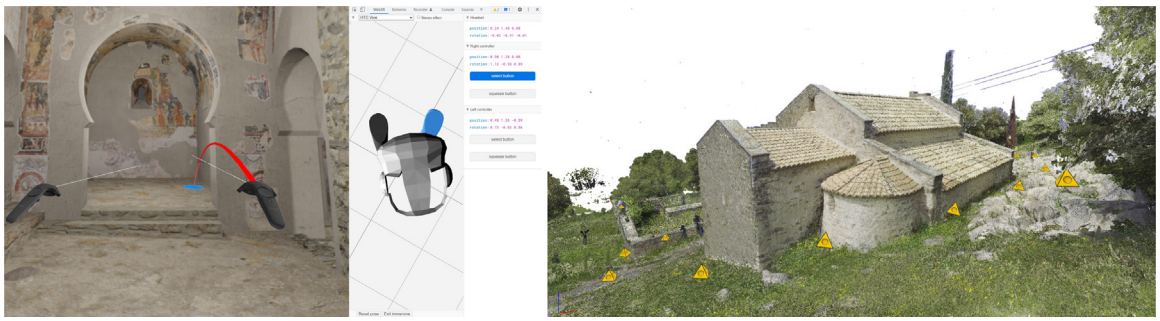

Fig. 6. St. Quirze de Pedret model. The image on the left also shows the WebXR API emulator.

distance. For unit-length vectors and quaternions, the threshold represents the maximum angular deviation. Finally, Boolean expressions do not require a threshold.

For example, the following snippets can be used as triggers for any option requiring a Boolean value:

```
// Ex. 1: check is position has changed below 10cm for 2s
stable("controllers.right.position", 2, 0.1)
// Ex. 2: check if orientation has changed below 0.35rad
stable("camera.rotation", 2, 0.35)
```

Notice that the expression must be passed as a string. This is because our implementation of `stable` requires storing the expressions being tracked, which are evaluated every frame.

Our current tool does not provide scene preparation features, which are out of the scope of this work. Therefore users are expected to prepare the scene, and define a suitable starting location for the camera. Since ThreeJS ray-scene intersections are not optimized for large meshes, users can provide also a simplified version of the scene to be used for ray casting and collision detection. Our implementation does perform collision detection (both for target selection during teleportation, and for avoiding traversing walls during walking/flying) but the collision response is currently not configurable (i.e. if enabled, collisions keep the camera position from being updated, but no sliding or other types of collision responses are implemented).

5. Use cases

In this section we discuss a couple of use cases that benefited from the short iteration cycles supported by our tool.

5.1. A Medieval Church with mural paintings

The goal was to provide intuitive navigation for art historians to explore the mural paintings of *St. Quirze de Pedret* using a standalone VR headset (Fig. 6). We used Google Chrome running on a desktop PC for testing different navigation techniques, and an Oculus Quest for fine tuning. While using the desktop PC, we used the *WebXR API Emulator*, a browser extension that enables developers to test WebXR content without using a real XR device by emulating mainstream XR devices and their controllers (Fig. 6). The tool allowed us to discard continuous steering techniques due to motion sickness issues. We thus opted to default to a free teleportation technique using the hand-held controllers for selecting the target position. Parameters such as maximum target distance and gradual teleportation speed were also fine tuned to the reduced size of the Church.

5.2. St. Quirze de Pedret surroundings and architecture

We wanted to enable art historians and the general public to explore the architecture of *St. Quirze de Pedret*, this time emphasizing on the outdoor model. This time free teleportation was tricky because the Church is on the edge of a long and jagged hill-mountain at the foot of the Pyrenees (Fig. 6). On the other hand, the scan locations where readily available, so we decided to adopt teleportation using as predefined nodes the scan locations, which also guarantee high-quality viewpoints of the model.

6. Evaluation

6.1. Purpose

We conducted a user study with web and computer graphics developers to obtain feedback about the practical utility of our approach to adapt the navigation technique to specific usages. All participants used a desktop PC and had access to a standalone VR headset. All test models (we used our own models and some CC-BY-licensed models from SketchFab) were included in the code base and thus readily available to be loaded in our framework. We mostly used cultural heritage models. The scenes were already "curated" in the sense that they were readily usable with teleportation and walk/fly metaphors (e.g. a mesh representing the floor was already defined to facilitate teleportation target selection).

6.2. Training

We asked participants to complete a short training session (about 20 min) consisting in watching a video describing the usage of the tool, reading a short documentation, and using the GUI of the application to complete simple exercises involving changing some parameters or writing minimalist JS code for the custom functions.

6.3. Task

First, participants had to choose one of the test scenes (available in a drop down menu). Then, they were asked to spend about 30 min in tweaking two different navigation mappings: one for their own preference ("expert" user) and another one for first-time VR users. Participants were instructed to use the application on a desktop browser, although they were invited to test it on the VR headset.

6.4. Participants

We recruited participants with 1+ year experience in either 3D graphics or JavaScript programming. Five participants (all male, aged 22–55) participated in the study.

Table 2
Questionnaire results (first round).

ID	Description	Mean
Q1	On a desktop PC, the GUI is useful for adjusting parameters	6.6
Q2	On a desktop PC, the GUI is useful for writing snippets	5.2
Q3	On VR mode, the GUI is useful for adjusting parameters	7.0
Q4	On VR mode, the GUI is useful for writing snippets	3.4
Q5	Adapting the technique by tweaking parameters was easy	6.6
Q6	Adapting the technique by writing snippets was easy	5.0
Q7	Overall the tool is useful	6.6

6.5. Questionnaires

In a first round, participants had to a complete a questionnaire after the task. The questionnaire included specific questions (see below) as well as open-ended questions about pros/cons and suggestions. After we collected all answers, we grouped and summarized the comments from the open questions. In a second round, we asked participants to read the comments from the other participants, and rate their agreement or disagreement with such comments.

6.6. Results

Table 2 shows the mean scores for the different questions on a 7-point Likert scale, with 1 point representing "strongly disagree" and 7 meaning "strongly agree"). Notice that overall the developers rated very positively the usefulness of the tool (Q7, mean score 6.6). Concerning the GUIs, both the 2D GUI and the VR GUI were found to be useful for adjusting parameters (Q1, mean score 6.6; Q3, mean score 7.0). Scores were less positive for writing custom functions (Q2, Q4), especially when using the VR GUI (Q4, mean score 3.4). This score was expected since this mode uses a virtual keyboard and the VR controllers, which is far more cumbersome than using a physical keyboard for writing text. Regarding the easiness of the tool for adapting the base navigation metaphors, users were again slightly more positive about tweaking parameters (Q5, mean score 6.6) than writing code snippets (Q6, mean score 5.0).

Table 3 shows the mean scores for the second-round questionnaire, whose questions were directly derived from the comments/suggestions made by the participants in the first round. Comments C1–C5 refer to custom functions. The fact the our prototype uses a non-specialized, single-line text input area limits the complexity of user-provided code to simple expressions (C1, C3). We plan to extend our prototype so that code snippets can be replaced by an URL with the actual code (C2). This would allow developers to write more complex snippets in an external editor, leaving the GUI editor for tweaking the simple ones. Comment C4 refers to the possibility of modifying from the GUI those variables declared in the snippets. This feature is available for example in Unity3D, which automatically exposes on the editor the public members of C# classes. We also plan to incorporate a similar feature (parsing the JS code) as it would minimize typing code with new values in favor of adjusting these values on the GUI. Finally, developers agreed that fully benefiting from custom functions requires some training (C5). These functions use concepts regular developers might not be familiar with (such as coordinate spaces and Three.JS classes).

Comments C6–C8 refer to the limitations of tweaking a VR navigation technique from a desktop browser running a WebXR API emulator. We agree that evaluating some usability aspects (e.g. motion sickness) definitely requires using the VR headset (C6, C7). Notice though that we allow parameter adjustment also in VR mode (Q3 in Table 2). Comment C8 refers to the way users can emulate the movement of the VR headset and controllers using a 2D mouse with the WebXR API emulator. As shown in the accompanying video, a single 1-DoF translation/rotation often requires multiple mouse clicks and displacements. Obviously, the 2D mouse cannot compete with the simultaneous 6-DoFs of the VR devices. This further stresses the importance of a VR version of the GUI, accessible from the VR headset. Finally, Comment C9 suggests a way to define multiple mappings (e.g. on a desktop browser), and switch between them using the GUI in VR mode. We also plan to add this feature.

7. Conclusions and future work

In this paper we have discussed the benefits of a web-based tool for quickly designing, testing and comparing navigation techniques. Our hypothesis is that, if testing iteration cycles are very short, designers can fine tune the default parameters of the chosen navigation technique to the specificity of the 3D environment, the user task, the target users and other contextual issues. Our tool supports the design of inclusive VR experiences in the sense that fast testing allows designers to offer a catalog of preset navigation techniques to cover the largest possible audience, and because the VR GUI allows end users to further customize navigation options to suit their needs. The feedback from developers suggests that the main strength of our approach is the possibility to tweak the parameter values from the GUIs (especially from the VR headset). Custom functions were also positively evaluated, though their use is mostly limited to compact expressions typed on a desktop browser. Even limited to compact code, since code snippets only address a very specific task, a simple expression often suffices to customize the navigation technique. Our tool is expandable and additional navigation metaphors (beyond walking, flying and teleportation) can be added. However, each new metaphor requires a suitable task decomposition leading to a proper set of configurable options, which might not be trivial for some metaphors, such as the World-in-Miniature [29], due to its rendering (e.g. occlusion handling) and manipulation requirements. Other types of extensions are easier to incorporate, such as modifying or customizing the proxies used to highlight teleportation nodes.

As future work, we plan to add the improvements suggested by the study participants, especially the exposure of variables declared in the code snippets so that they can be edited at runtime in the VR GUI. We also plan to conduct a user study where developers and final users (e.g. art historians) would be requested to jointly customize the navigation techniques to specific Cultural Heritage models.

We are working on making the configurable navigation module as agnostic as possible of its HTML/ThreeJS environment, so that it could be deployed in already-existing WebXR applications as seamlessly as possible. A major difficulty however is the interrelationship between user inputs for navigation and user inputs for other major user tasks in VR (such as object selection and manipulation). Therefore the tool is intended to simplify the development and customization of the navigation metaphors, rather than to provide a ready-to-use component requiring no additional JS programming.

We would like to explore the use of a node-based editor to define and fine-tune the navigation metaphors. Besides potential benefits in terms of ease of use, a node-based solution will facilitate the automatic generation of a schematic illustration of the metaphor (for example, a neutral avatar with arrow/text overlays indicating which inputs are used, and for which task). In our current version of the tool, snippets are not parsed by ourselves but just evaluated using JS eval() function, which hinders the potential extraction of the data we could use to graphically illustrate the mapping between user input and navigation. The addition of

Table 3
Questionnaire results (second round).

ID	Description	Mean
C1	The text widget is single-line so custom functions should be small.	6.4
C2	Besides actual code, the GUI should accept a link to the code.	6.4
C3	The function editor should support syntax highlighting and auto-completion.	5.6
C4	Variables declared in custom code should be exposed in the GUI.	6.4
C5	Fully benefiting from custom functions requires more extensive training.	6.0
C6	The desktop GUI is useful for initial values, but fine tuning requires the headset.	6.4
C7	The WebXR emulator speeds up testing but you need the headset to evaluate usability.	4.6
C8	Moving the controllers with the WebXR emulator is more complex than with physical devices.	7.0
C9	The VR GUI should allow switching between different mappings.	5.6

a node editor would involve important challenges though. Visual programming makes it easier to understand and customize the logic of the function for non-programmers, but it usually hinders reusability as copying parts of the node graph is not as easy as copying parts of the source code. Another advantage of our code snippets is that they are more compact and often can be shown in a single line of text (even as options of the GUI's dropdown menus). Another issue is that, although there are multiple JS node editors for WebGL, we are not aware of any ready-to-use node editor compatible with WebXR. Nevertheless, we plan to explore this possibility in the future.

CRediT authorship contribution statement

Ahmed Kamal: Conceptualization, Programming, Writing. **Carlos Andujar:** Conceptualization, Programming, Validation, Writing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been funded by MCIN/AEI/10.13039/501100011033/FEDER "A way to make Europe". Pedret model partially funded by EU Horizon 2020, JPICH Conservation, Protection and Use initiative (JPICH-0127) and the Spanish Agencia Estatal de Investigación, grant PCI2020-111979 *Enhancement of Heritage Experiences: the Middle Ages; Digital Layered Models of Architecture and Mural Paintings over Time (EHEM).* Art Gallery model by 3DTwins licensed under CC-BY-4.0. Richard's Art Gallery - Audio Tour by shinymagic licensed under CC-BY-4.0.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.05.015.

References

[1] Al Zayer M, MacNeilage P, Folmer E. Virtual locomotion: A survey. IEEE Trans Vis Comput Graphics 2020;26(6):2315–34.
[2] Gomes L, Bellon ORP, Silva L. 3D reconstruction methods for digital preservation of cultural heritage: A survey. Pattern Recognit Lett 2014;50:3–14.
[3] Pintore G, Mura C, Ganovelli F, Fuentes-Perez L, Pajarola R, Gobbetti E. State-of-the-art in automatic 3D reconstruction of structured indoor environments. Comput Graph Forum 2020;39(2):667–99.
[4] Potenziani M, Callieri M, Dellepiane M, Corsini M, Ponchio F, Scopigno R. 3DHOP: 3D heritage online presenter. Comput Graph 2015;52:129–41.
[5] Potenziani M, Callieri M, Scopigno R. Developing and maintaining a web 3D viewer for the CH community: an evaluation of the 3Dhop framework. In: Sablatnig R, Wimmer M, editors. Eurographics workshop on graphics and cultural heritage. The Eurographics Association; 2018.
[6] Schütz M. Potree: Rendering large point clouds in web browsers. [Bachelor's thesis], Vienna, Austria: Vienna University of Technology; 2016.
[7] MacIntyre B, Smith TF. Thoughts on the future of webxr and the immersive web. In: 2018 IEEE international symposium on mixed and augmented reality adjunct. IEEE; 2018, p. 338–42.
[8] Engberg M, Bolter JD, MacIntyre B. RealityMedia: an experimental digital book in WebXR. In: 2018 IEEE international symposium on mixed and augmented reality adjunct. IEEE; 2018, p. 324–7.
[9] Luigini A, Fanini B, Basso A, Basso D. Heritage education through serious games. A web-based proposal for primary schools to cope with distance learning. VITRUVIO-Int J Archit Technol Sustain 2020;5(2):73–85.
[10] Fanini B, Ferdani D, Demetrescu E. Temporal lensing: An interactive and scalable technique for Web3D/WebXR applications in cultural heritage. Heritage 2021;4(2):710–24.
[11] Butcher PW, John NW, Ritsos PD. Vria: A web-based framework for creating immersive analytics experiences. IEEE Trans Vis Comput Graphics 2020;27(7):3213–25.
[12] Fanini B, Cinque L. Encoding immersive sessions for online, interactive VR analytics. Virtual Real 2020;24(3):423–38.
[13] Ribeiro R, Ramos Ja, Safadinho D, Reis A, Rabadão C, Barroso Ja, et al. Web AR solution for UAV pilot training and usability testing. Sensors 2021;21(4):1456.
[14] Di Luca M, Seifi H, Egan S, Gonzalez-Franco M. Locomotion vault: The extra mile in analyzing VR locomotion techniques. In: Proceedings of the 2021 CHI conference on human factors in computing systems. 2021.
[15] Fernandes AS, Feiner SK. Combating VR sickness through subtle dynamic field-of-view modification. In: 2016 IEEE symposium on 3D user interfaces. IEEE; 2016, p. 201–10.
[16] Kamal A, Andujar C. A platform for developing and fine tuning adaptive 3D navigation techniques for the immersive web. In: The 26th international conference on 3D web technology. 2021.
[17] LaViola Jr JJ, Kruijff E, McMahan RP, Bowman D, Poupyrev IP. 3D user interfaces: theory and practice. Addison-Wesley Professional; 2017.
[18] Razzaque S, Swapp D, Slater M, Whitton MC, Steed A. Redirected walking in place. In: EGVE. vol. 2, 2002, p. 123–30.
[19] Suma EA, Bruder G, Steinicke F, Krum DM, Bolas M. A taxonomy for deploying redirection techniques in immersive virtual environments. In: 2012 IEEE virtual reality workshops. IEEE; 2012, p. 43–6.
[20] Steinicke F, Bruder G, Kohli L, Jerald J, Hinrichs K. Taxonomy and implementation of redirection techniques for ubiquitous passive haptic feedback. In: 2008 International conference on cyberworlds. IEEE; 2008, p. 217–23.
[21] Slater M, Steed A, Usoh M. The virtual treadmill: A naturalistic metaphor for navigation in immersive virtual environments. In: Virtual environments' 95. Springer; 1995, p. 135–48.
[22] Schroeder R, Huxor A, Smith A. Activeworlds: geography and social interaction in virtual reality. Futures 2001;33(7):569–87.
[23] Bozgeyikli E, Raij A, Katkoori S, Dubey R. Point & teleport locomotion technique for virtual reality. In: Proceedings of the 2016 annual symposium on computer-human interaction in play. 2016, p. 205–16.
[24] Mackinlay JD, Card SK, Robertson GG. Rapid controlled movement through a virtual 3D workspace. In: Proceedings of the 17th annual conference on computer graphics and interactive techniques. 1990, p. 171–6.
[25] Bhandari J, MacNeilage PR, Folmer E. Teleportation without spatial disorientation using optical flow cues. In: Graphics interface. 2018, p. 162–7.
[26] Christou CG, Aristidou P. Steering versus teleport locomotion for head mounted displays. In: International conference on augmented reality, virtual reality and computer graphics. Springer; 2017, p. 431–46.
[27] Coomer N, Bullard S, Clinton W, Williams-Sanders B. Evaluating the effects of four VR locomotion methods: joystick, arm-cycling, point-tugging, and teleporting. In: Proceedings of the 15th ACM symposium on applied perception. 2018, p. 1–8.

[28] Ware C, Osborne S. Exploration and virtual camera control in virtual three dimensional environments. In: Proceedings of the 1990 symposium on interactive 3D graphics. Association for Computing Machinery; 1990, p. 175–83.
[29] Stoakley R, Conway MJ, Pausch R. Virtual reality on a WIM: Interactive worlds in miniature. In: Proceedings of the SIGCHI conference on human factors in computing systems. USA: ACM Press/Addison-Wesley Publishing Co.; 1995, p. 265–72.
[30] Lim K, Lee J, Won K, Kala N, Lee T. A novel method for VR sickness reduction based on dynamic field of view processing. Virtual Real 2021;25(2):331–40.
[31] Yildirim C. Don't make me sick: investigating the incidence of cybersickness in commercial virtual reality headsets. Virtual Real 2019;1–9.
[32] Kolasinski EM. Simulator sickness in virtual environments. vol. 1027, US Army Research Institute for the Behavioral and Social Sciences; 1995.
[33] Keshavarz B, Riecke BE, Hettinger LJ, Campos JL. Vection and visually induced motion sickness: how are they related? Front Psychol 2015;6:472.
[34] Dombrowski M, Smith PA, Manero A, Sparkman J. Designing inclusive virtual reality experiences. In: International Conference on Human-Computer Interaction. Springer; 2019, p. 33–43.
[35] Mott M, Cutrell E, Franco MG, Holz C, Ofek E, Stoakley R, et al. Accessible by design: An opportunity for virtual reality. In: 2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct. IEEE; 2019, p. 451–4.
[36] Coogan CG, He B. Brain-computer interface control in a virtual reality environment and applications for the internet of things. IEEE Access 2018;6:10840–9.
[37] Nilsson NC, Serafin S, Steinicke F, Nordahl R. Natural walking in virtual reality: A review. Comput Entertain (CIE) 2018;16(2):1–22.
[38] Boletsis C. The new era of virtual reality locomotion: A systematic literature review of techniques and a proposed typology. Multimodal Technol Interact 2017;1(4):24.
[39] Cardoso JC, Perrotta A. A survey of real locomotion techniques for immersive virtual reality applications on head-mounted displays. Comput Graph 2019;85:55–73.
[40] Cannavò A, Calandra D, Pratticò FG, Gatteschi V, Lamberti F. An evaluation testbed for locomotion in virtual reality. IEEE Trans Vis Comput Graphics 2020;27(3):1871–89.
[41] Argelaguet F, Andujar C. A survey of 3D object selection techniques for virtual environments. Comput Graph 2013;37(3):121–36.