Vol.2019-1 No.1
2019/01/12

Mobile phone affordance controlling rigged extended reality scenes

モバイル　フォン　アフォーダンスによる拡張現実シーンの操作

Loïck Walle [†] *and Michael Cohen* [†]

Abstract: In an XR (extended reality) space, the most important aspect is not the space itself but the interaction with it and what is possible to do in this space. Virtual reality (VR) is the best known XR instance by common people, largely due to its deployment in video games, simulators, and applications related to creative fields. However, even if its starts to be well known, people are still reluctant to use it, mainly due to its expensive hardware and difficulty to configure. Nevertheless, nowadays, most people own a very powerful device comparable to VR controller which can operate without external motion sensors and thereby make a VR set more affordable: a smartphone.

An ordinary smartphone has about as many sensors as any VR controller or video game console controller and, with an application installed on that smartphone or tablet, position data can be sent to another device continuously, in the same way that VR controllers do with their motion trackers. Moreover, by sending this data to a server continuously for redistribution to clients subscribed to multicast channels, it is possible to connect many users to a shared application, which is then no longer only for personal use but used by a social cohort. Thus, one or many users can move 3D objects in a virtual space or control real devices, simultaneously or separately with separate smartphones changing the virtual environment, not depending on one single inhabitant, but upon an entire cohort's will.

Keywords: XR, augmented virtuality, smartphone, sensors, Collaborative Virtual Environment (CVE), Unity controller.

1. Introduction

For many years now, almost everybody knows or at least has some notions about virtual reality. Whether changing our perception of reality with augmented reality (AR) or exploring an entirely virtual world with virtual reality (VR), people are eager to try this technology since it was publicized through many different media such as movies (eXistenZ, 1999), video games (Robinson: The Journey, 2016), and Japanese animation (Sword Art Online, 2009).

Extended reality (XR) is a generalization of AR, VR, and MR (mixed reality), and devices using XR have yet to become commonplace, because such devices are still expensive, complex to use, or just do not fulfill inflated expectations. However, thanks to equipment such as a VR headset, motion sensors, and controllers, it is possible to access many and various virtual spaces. For example, the most common uses of a VR set are video games which normally only require reflection, reactivity, and the use of both hands (keyboard and mouse). However, with virtual equipment, the human body can be used directly as an interactive element of a virtual environment, in a reduced space. Of course, virtual reality applications aren't limited to video games but can go a lot farther as, for example, in the form of a simulator in order to increase user knowledge and skills or, in a more therapeutic area, present users a situation which allows them to recover some ability or to cure some psychological disorder (such as fear of heights, spiders, flying, etc.)

Thus, XR can improve daily life in many different ways, but it is necessary to find cheaper and more affordable ways to deliver it. We devised a means to interact easily with virtual worlds while maintaining affordability by using a device featuring enough sensors to rival VR controllers and motion trackers. This device is a simple smartphone which can be turned into a controller for virtual space, and sparing users from having to buy specific controllers. A smartphone is, for most of us, something common we use everyday, but it's also a multisensor platform. Each smartphone features 16 to 24 different built-in sensors, including many microelectromechanical systems (MEMS) such as the pressure sensor used for tactile screen and also the inertial measurement unit (IMU), used for position tracking with gyro sensors and accelerometers. Thereby, smartphones are as performant as dedicated VR/consoles controllers.

The main aim of this project is to find a way to interact, via a simple smartphone or tablet, with an XR environment, in realtime with the possibility of involving many users simultaneously.

[†] Spatial Media Group, Computer Arts Lab; University of Aizu; Aizu-Wakamatsu, Fukushima 965-8580; Japan

The objective is not to create a particular video game but to create an interactive and social environment shared by a community. The second aim is to create a situation by which it becomes possible to trick user's perception, by projecting their image into a virtual avatar. This would not only amuse the user but also demonstrate that this experience can be use as much in virtual reality as mixed reality. Finally, a tertiary but not insignificant objective was to make virtual scenes as realistic and entertaining as possible, the aim being not only to satisfy user's expectations who use virtual reality mainly to entertain themselves but also to improve the feeling of virtual projection mentioned in the second aim.

2. Methodology

2.1. Deploy a server (data retrieval and multicast)

Creating a multicasting server is not difficult, since it only requires forwarding streamed data from a smartphone or a tablet. The server doesn't need to manipulate or change anything, just receive and distribute. But an important feature is the ability to multicast different information, such as smartphone rotation (yaw, roll, and pitch) or translation (surge, heave, and sway), on different channels at the same time. One or many different data streams from a smartphone can be sent to manipulate multiple objects in a virtual space.

With this requirement in mind, using a Collaborative Virtual Environment, or CVE, was the most convenient approach since this one is used for collaboration and interaction of many participants inside a shared virtual space by synchronizing distributed client. Moreover, a CVE server manages multiple channels simultaneously allowing multicast events, where the number of channels depends on the server's performance.

The principle of a CVE is simple: one computer acts as a server and redistributes all information concerning the virtual space. Other computers which host various clients receive data from the server, interact with the environment, then send an update request and what's changed to the server (Figure 1).

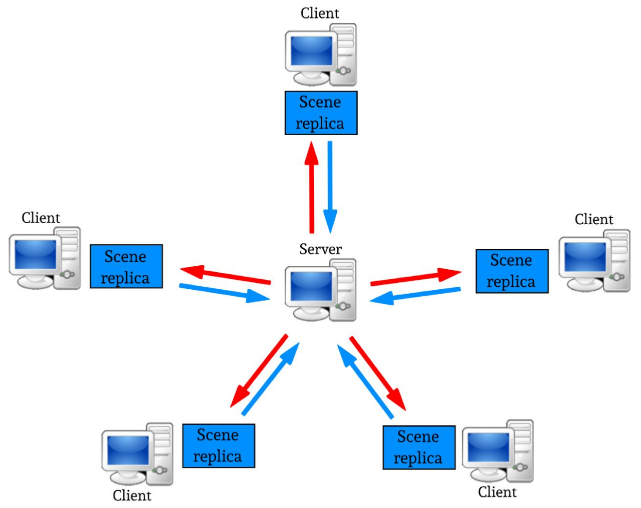


Figure 1: Collaborative Virtual Environment (CVE) client-server (star topology, a.k.a. hub and spokes) architecture

2.2. Deploy an application (data capture and transmission)

The server can receive data from any device, but we still need to enable such data stream sources. The most flexible way to do that is to make an application able to transmit continuously position data from devices to a server specified by its IP address. That's why the application "Twhirleds" was developed for both Google Android (using Android Studio) and for Apple iOS (using Xcode).

Functionalities of this mobile application include the following:

- It measures (rotation values) of the device by acting as a position sensor (Figure 2a).
- It allows selecting which information is transmitted ("throttling") in order to not overload a server with excessive data (Figure 2a).
- It allows adjusting the transmitted data (transmission rate, polarity, wrap-around, …) (Figure 2b).
- It allows, of course, entering a CVE server IP address, and also choosing on which channels the user wishes to transmit device data (Figure 2b).

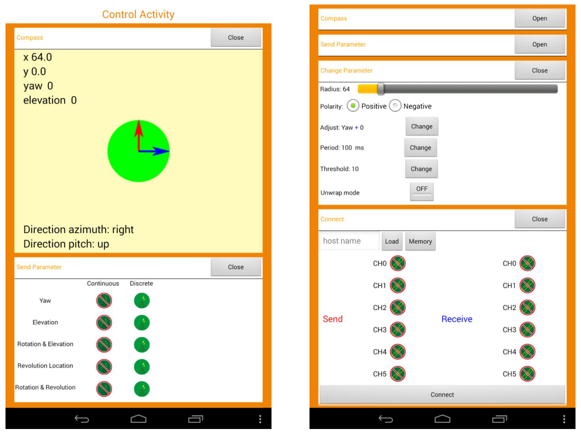


Figure 2: "Twhirleds" on Android screenshots: left (part a) and right (part b)

Both versions (Android and iOS) work basically the same way: a user selects which kind of data is to be transmitted, on which channels, and to what server. Once a connection is established with a CVE server, the device continuously sends its position data to specific sockets corresponding to each channel. These data are sent from wireless devices in the form of TCP sockets through the application using a cellular or Wi-Fi connection without requiring any application-level reply from the server.

Once a connection is established, data streams are received continuously by a CVE server (using a transcoding bridge if necessary) on its different channels. These streams can be retrieved by any kind of compliant device or software, using the IP address of the CVE server. For instance, a Unity program can join a session (through a bridge) to receive updates from smartphones. Overall data flow is represented by Figure 3.

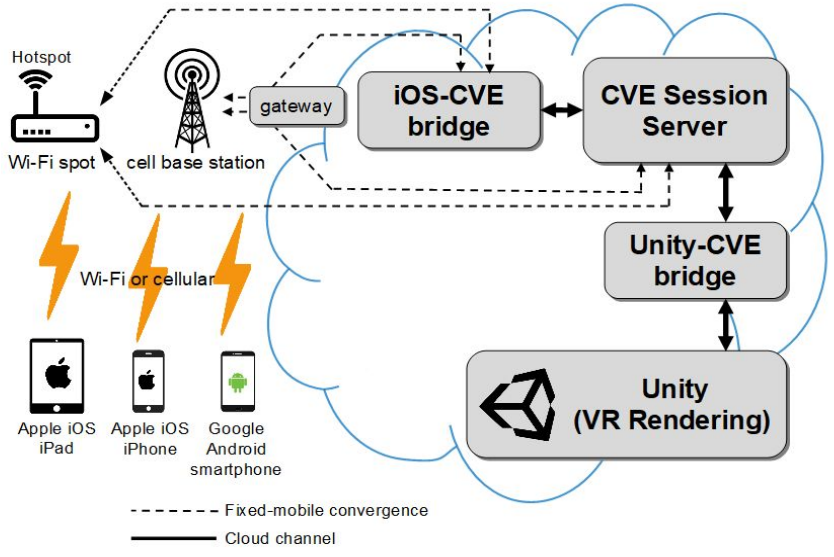


Figure 3: Network architecture

2.3. Configure Unity (data reception)

Inside Unity, a script (written in C#) receives information from channels shared by a CVE server through its transcoding bridge. This script handles communication with scripts attached as components to Unity scene objects, encoding and decoding from the FlatBuffers format, including UDP/TCP-based communication with a Unity-CVE bridge (transcoder). To add this script to a project, a developer must associate it with some scene object (even an otherwise empty one), specifying a CVE server IP address, server send port, and server receive port.

2.4. Create virtual environment (Blender)

Data transmission is now continuous and receivable on multiple independent channels in Unity. It only remains to realize the last step, which is to create a virtual environment where conditions are following:

- At least one scene must allow users to visually project themselves into a virtual space (realistic enough to allow the user to associate himself with a virtual character).
- At least one scene requires that users get involved collectively in a shared objective.
- All scenes must be entertaining for users or provide interesting experience.

Unity is not suitable for creation of complicated objects and characters. Therefore Blender (v. 2.79) was used. It is computer-aided design (CAD) modeling software which is free and open source. It allows creation of objects that might be needed, such as avatars, scenes, props, etc.

2.4.1. Create objects

As for all CAD software, every object starts by the creation of one or several "basic" shapes, such as cubes, spheres, cylinders, cones, etc… From these objects, it is possible to create objects more complex such as those necessary for the "mise en scène" of a project. Of course, the creation of very complex objects requires use of more advanced methods than simple object editing. Sometimes, it is required to use physics in order to recreate objects with a realistic aspect. Taking the example of the first scene, the scene theater curtain is one of these complex objects which has been animated. Although it is just a simple curtain, it is necessary to model collision with the floor and itself to create folds with natural seeming irregularity to look realistic, as shown in Figure 4.

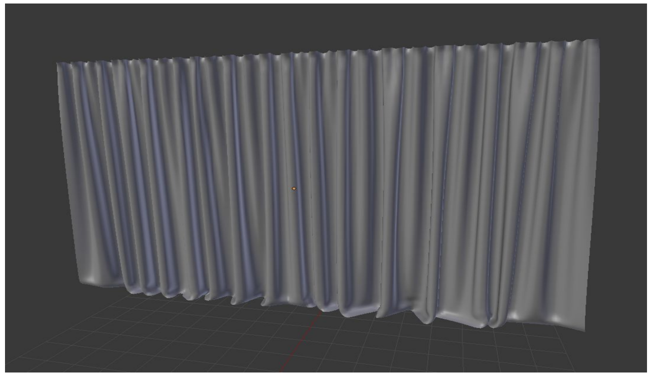


Figure 4: Scene 1, theater curtain (Blender screenshot)

2.4.2. Create a character

Even if it seems difficult to model objects with complex aspect using CAD software, objects created by the hand of man follow a certain logic and precision that is more easily achieved on 3D software. However, modeling living beings such as animals, plants, and, humans brings a whole other level of complexity that may require a lot of time, as shown in Figure 5 which has fewer than 9200 vertices and about the same number of faces.

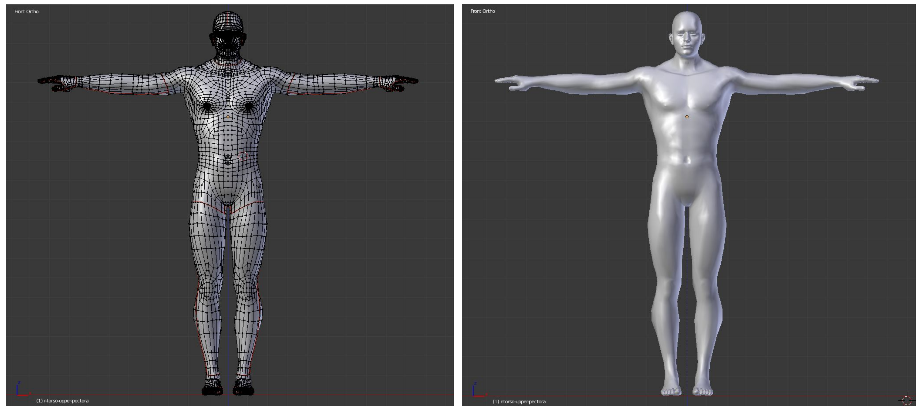


Figure. 5: Scene 1, character (Blender screenshots)

2.4.3. Objects texturing

Creating realistic objects involves textures. A texture is a 2D image representing a surface, offering the possibility to simulate appearance once applied to an object. More than a simple color, textures allow making an object with fewer faces (low-poly) as realistic as possible while maintaining high performance level. To be able to use these textures, it is required to unfold every 3D object onto a 2D "UV" plan or "Unwrap," as shown in Figure 6.

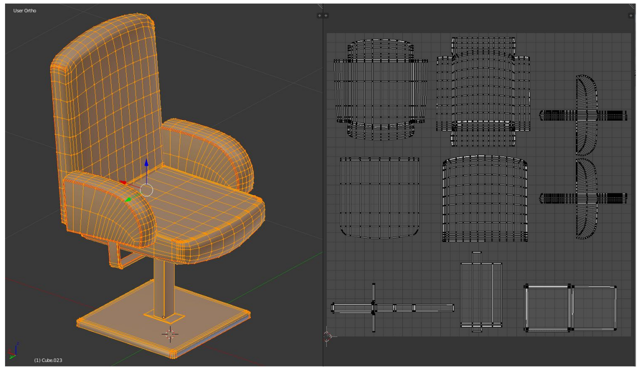


Figure 6: Theater chair in 3D perspective (left) and unwrapped (right) (Blender screenshots)

Once an object has been "unwrapped," it is possible to add a texture to it, and after putting and setting these texture on a 3D objects, it is possible to get a realistic appearance, as the main curtain of the theater shown in Figure 7. As soon as this step is realized and satisfying enough, these objects and textures can be imported and used in Unity.

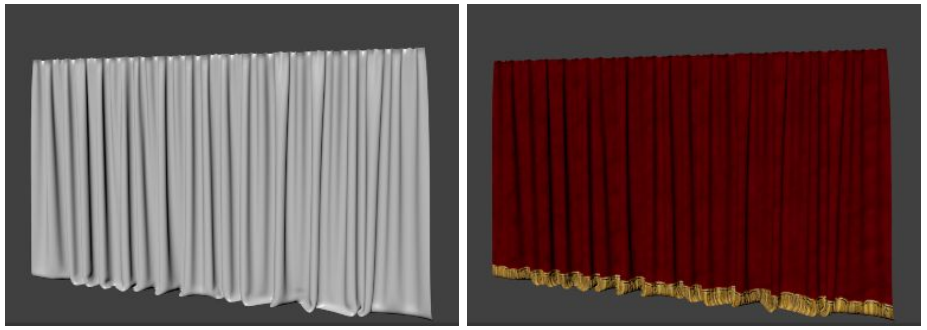


Figure 7: Theater curtain without texture (left) and with (right) (Blender screenshots)

2.4.4. Create avatar animation

Once all objects are modeled, textured and imported into Unity, the character and props must be animated in virtual space. However, instead of directly using the Unity animation editor for character animation, it was more suitable to continue to use Blender, mainly because it was easier and faster to create realistic animation. To realize this, it was required to create an armature which was rigged to the avatar to move it easily. An armature acts the same as a skeleton, except that it is this armature, and not muscles and tendons, which serves to move the body. Once created, this armature must be rigged to an object to move every vertex via corresponding bones. Once an armature has been rigged to an object, it is possible to control the object externally (Figure 8a).

However, even if it is possible to easily animate an avatar, it is still slow and not realist since, by default, the armature moves by following the "Forward Kinematics" method, a.k.a. FK. This method is the way to animate an object by explicitly manipulating each bone of an armature. FK has very high accuracy, which is convenient for machine or robot animation, but not for living beings, so it was important to change to the "Inverse Kinematics" method, a.k.a. IK. This method is hard to parameterize because the user must determine every constraint (limit) for each joint, but once it's done (Figure 8b), animations are easier to make, and look more real and natural.

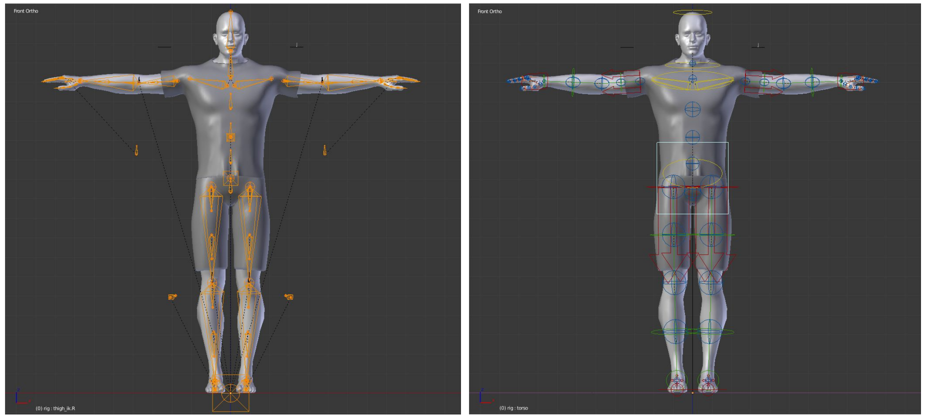


Figure 8: An avatar with FK armature at left (a) and with IK armature at right (b) (Blender screenshots)

Once an armature is finally ready, it is possible to create all the necessary animation for this project and to ensure that these animations are realistic. The animation of the avatar can be created by adjusting its armature, inserting a keyframe to the corresponding frame and, adjusting a bezier curve in a graph editor, repeating these operations until natural gesture is obtained. When an animation is composed, it is saved as an "action" to import into Unity. An animation is composed of several actions using the "action editor." Each of these actions defines movements expressed by the avatar in the virtual space.

2.5. Create virtual interaction (Unity)

2.5.1. Prepare the scene: "mise en scène"

Concerning scene preparation, it is needed to import objects realized in Blender. Since Blender and Unity are software collaborating with each other, it is possible to import objects easily by saving Blender files (.blend) directly into a Unity project file. One need only drag an object into the 3D space to see it appear. Once imported, a material must be created and assigned to the object. This material is required because it allows the designer to attribute colors or textures and also to set its reflection from lights. Finally, when all objects are correctly placed, lights must created, placed, and arranged to get effective shadows and lighting, which are an important aspect of satisfying immersion.

2.5.2. Synchronize smartphone and character animation: projecting affordance into mixed reality space

As mentioned in Section 2.3, Unity can be extended to continuously receive information from devices such as smartphones and tablets. Concerning synchronisation between simple objects and smartphone data, it is not difficult since the system just needs to correspond object coordinates and smartphone coordinates in realtime. However, synchronization of smartphone data and character animation is quite different, since it is a gesture which changes instead of an object.

Firstly, an "animator" must be assigned to a character to get an animation set to use during a scene. This animation set is called an "animator controller," itself defined by all possible actions that an avatar might use and in which order it will be able to use them. In case of Scene 1 "*poi*", the character uses an animator controller called "spinControl". Its actions are represented by Figure 9, which allow it to whirl a *poi* with left hand, switch hands, whirl with right hand, and switch hands again to initial position.

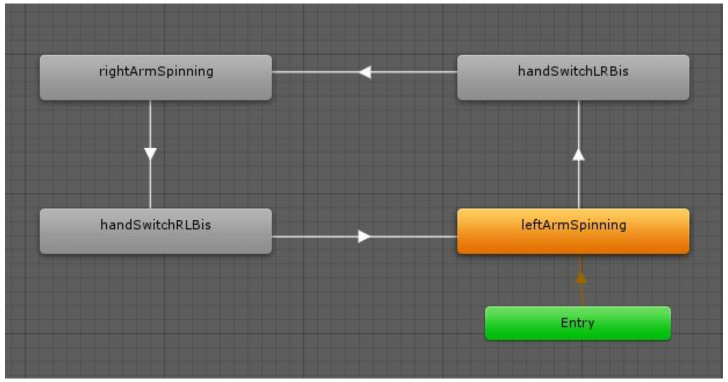


Figure 9: spinControl routine (Unity animator screenshot)

To play animations at the right moment, in order to make correspond them to coordinates, it is possible to play animations frame by frame. Thus, for "leftArmSpinning" and "rightArmSpinning" actions which require synchronization with the *poi* affordance (and so with server data) to be realistic and fluid, these animations have been realized across 360 frames (frame amount by animation doesn't really matter since it is more by convenience of work and explanation) and are played depending on given angles. For example, if a smartphone sends an angle of 270° then the animation will be played at frame 270. However this solution works only if the data sent are uniform and continuous (degree by degree), which is not the actual case.

It was therefore necessary to create smooth transition between two frames with a large gap to avoid a latency effect and loss of fluidity. This transition is realized in two parts: the first is to use smartphone coordinates for each received angle, which become a new arrival point as the previous position becomes the new depart point for the animation. Then, it is needed to adapt transition speed depending on the gap between these two points, so the bigger the gap the faster the transition (of course, transition speed should stay within human or natural limits). The second part is the fact to apply the transition on the shortest path between two points and not following clockwise or not.

3. Results

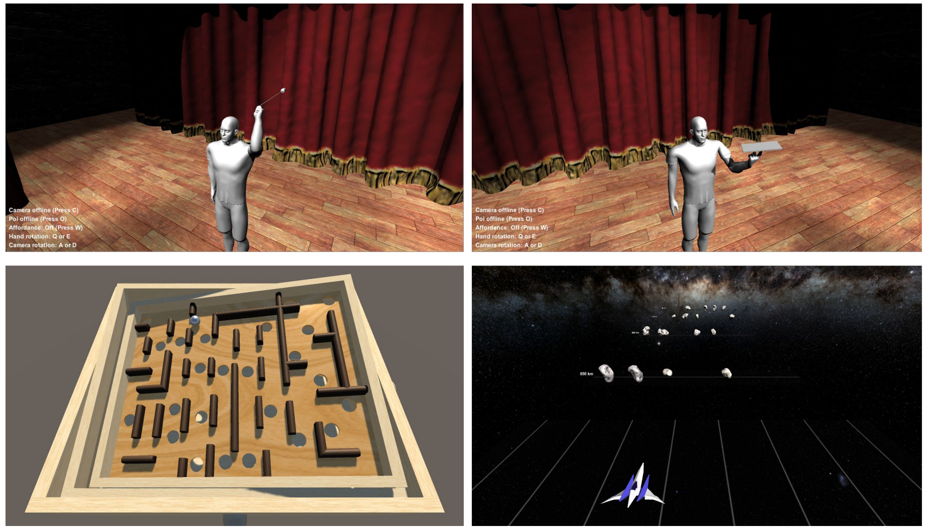
To summarize, it is useful to review the objectives of this project and to evaluate, point by point, if these objectives have been achieved or not. The main objective was to find a way to transmit data in realtime from smartphone or tablet integrated sensors to an XR environment and involving many user simultaneously.

For that, the first step was to set up a server able to continuously receive data but also to keep these data segregated across multiple independent channels. The second step was to transmit location data from smartphone or tablet to that server. The third step was to exploit data received by the server, to use it in virtual spaces. As explained previously, devices are able to transmit continuously their location data to one or many different channels maintained on a CVE server which are then used by clients. This information may also be received continuously into Unity through a CVE-Unity bridge to manipulate one or many 3D objects. Finally, the last step was the creation of situations allowing the user to visually project themselves into a virtual space, and to make this situation realistic and entertaining. Four scenes were created in Unity, controllable by one person or by a cohort via consumer devices.

This first and second scenes represent a way to visually project the user into XR environment by whirling a *poi* or padiddling a tablet. The user sends device information (yaw angle) to channel #1 controlling the character animation as well as the poi. By using an artificial poi (the smartphone is put inside a box itself attached to a tether), the user can get the impression of looking at himself as an avatar in 3D space (Figure 10a). Also, if data is sent to channel #2, the camera is manipulated, turning around the avatar to get different perspective. Moreover, the avatar is self-conscious and reacts depending of the camera's location (dorsal or frontal view), switching hands to maintain a good point of view. Finally, it is possible to activate an affordance which will totally change the user experience by moving the avatar depending not on the smartphone angle and the relative avatar location but depending on the camera, and so, of the user perception.

The aim of the third scene, marble maze, is to allow users to play cooperatively. It comprises several levels whose difficulty and gameplay evolves gradually. In the first level, a user sending data to channel #1 can manipulate the board in order to move the marble towards the goal, but if the ball falls into a hole then the game is over. The second level has the same objective but, a second user (connected to channel #2) is required to tilt the board (Figure 10c). For the third level, the board is circular and the second user manipulates labyrinth walls, so he is the only one able to open the way for the first user reach the goal. This scene involves a lot of physics simulation, especially collision and gravity.

For the last scene, users are involved in an space runner game. A user connected to channel #1 can move the spaceship from left to right to dodge approaching asteroids. The more the user dodges asteroids, the more points are earned for a final score (Figure 10d). During the game, many objects spawn to make the game adaptive faster or slower. A user connected to channel #2 can move the background to distract the first user.


Camera offline (Press C)
Poi offline (Press O)
Affordance: Off (Press W)
Hand rotation: Q or E
Camera rotation: A or D

Camera offline (Press C)
Poi offline (Press O)
Affordance: Off (Press W)
Hand rotation: Q or E
Camera rotation: A or D

Figure 10: Scene rendering (Scene 1: *poi*, top-left (a); Scene 2: padiddle, top-right (b); Scene 3: marble maze, bottom-left (c); Scene 4: space endless runner, bottom-right (d))

4. Discussion

As mentioned at the start of this paper, the use of tools related to virtual reality, and so to extended reality, has became a common conversation subject for not only for users specializing in informatics but also by common people. Nevertheless, the use of AR and VR tools, especially VR systems, have yet to be widespread through the world due to expensive hardware which is difficult to configure.

However, as has been demonstrated, it is possible to use an ordinary smartphone or tablet to transmit position data and to manipulate 3D objects in a virtual space at the same as any VR controller with smartphones. Nevertheless, this project has not been without any issues (even if some of them have been pretty interesting) or difficulties. Moreover, this project can go much further but was limited due to a lack of knowledge and time.

Although original objectives to realize this project have been fulfilled, the prototypes scenes are just a suggestion of future research. As shown, any smartphone or tablet can be used as an XR controller in any kind of virtual space. Such techniques can also be used to deploy remote control of real objects.

At home, by connecting to a domestic network, it is possible to modulate ambient lightning or loudspeaker intensity by simple gesture. One can control a TV by targeting a menu and press a button in the same way as the Wiimote controls a game menu, or closing or opening windows. Once applied, every smartphone or tablet can become a universal remote control for any device in the home.

In the video games field, it is possible to use a smartphone as controller to play to any game on console. Not only can one push buttons, but gesture expressed by the device can also affect the gameplay.

In robotics, a smartphone will be able to remotely control actuators or robotized arms as the same way as gloves with integrated sensors. Or one could take control of a vehicle and drive it as a remote-controlled car or as karts in a Mario kart game.

References

[1] Joseph J. LaViola, Ernst Kruijff, Jr., Ryan P. McMahan, Doug A. Bowman, and, Ivan Poupyrev, 3D user interfaces: theory and practice, Second edition, Part 3: Hardware Technologies For 3D User Interface, Chapter 6: 3D User Interface Input Hardware, Addison-Wesley, 2017, 978-013-403432-4.
[2] Michael Cohen, Rasika Ranaweera, Bektur Ryskeldiev, Tomohiro Oyama, and Aya Hashimoto. “Twhirleds”: Spun and whirled affordances controlling multimodal mobile-ambient environments with reality distortion and synchronized lighting to preserve intuitive alignment”. In: ScPA: Scientific Phone Apps and Mobile Devices 3.5 (2017). Ed. by David Philip Lane and Samuel Ken-En Gan, pp. 1–20. issn: 2364-4958. doi: 10.1186/s41070-017-0017-x.
[3] John M. Blain, “The Complete Guide to Blender Graphics, 3rd Edition, Computer Modeling & Animation”, CRC press, Taylor & Francis Group, 2016, 978-1498746458.
[4] Blender official website, https://www.blender.org/
[5] Unity official website, https://unity3d.com/
[6] Android Developers official website, https://developer.android.com/
[7] Zig Simulator official website, https://zig-project.com/
[8] WiFi Mouse official website, http://wifimouse.necta.us/
[9] Unified Remote official website, https://www.unifiedremote.com/
[10] Monect PC Control official website, https://www.monect.com/
[11] Merge VR official website, https://mergevr.com/
[12] Google Cardboard official website, https://vr.google.com/cardboard/
[13] eXistenZ (1999) [Movie], https://www.imdb.com/title/tt0120907/
[14] Robinson: The Journey (2016) [Computer video game], Crytek, http://www.robinsonthegame.com/
[15] Sword Art Online (2009) [Light novel / Japanese animation], wrote by Reki Kawahara, animated by A-1 Pictures.