FUSION OF POSE AND HEAD TRACKING DATA FOR IMMERSIVE MIXED-REALITY APPLICATION DEVELOPMENT

Katarzyna Czesak *Raúl Mohedano* *Pablo Carballeira* *Julián Cabrera* *Narciso García*

ABSTRACT

This work addresses the creation of a development framework where application developers can create, in a natural way, immersive physical activities where users experience a 3D first-person perception of full body control. The proposed framework is based on commercial motion sensors and a Head-Mounted Display (HMD), and a uses Unity 3D as a unifying environment where user pose, virtual scene and immersive visualization functions are coordinated. Our proposal is exemplified by the development of a toy application showing its practical use.

Index Terms— Virtual reality, Unity 3D, head-mounted device; body pose tracking, hand tracking.

1. INTRODUCTION

Immersive applications involving both physical interaction and virtual or augmented reality are already part our daily life. The available offer of such applications keeps increasing every day, and this tendency does not seem to cease in the following years, as they represent the starting point for a variety of consumer electronics applications, including mainly gaming but also training (i.e. simulators) and rehabilitation (i.e. supervised exercising).

Several reasons explain the increasing penetration of interactive immersive applications in the market. First, the latest advances in both human-machine interfaces and immersive visualization displays that have largely evolved in the last years, and are now achievable using simple commercial consumer electronic devices. As an example, pose estimation/tracking used to need mechanical sensors (as in traditional MoCap modeling) or purely visual cameras (either one or overlapping sets of them) plus a heavy, offline processing; in contrast, it can nowadays be performed online using commercial depth cameras (such as Microsoft Kinect [1]), which have become a robust *de facto* standard for such task. New capturing devices have been released along with SDKs allowing developers, to focus on application development, and not in the technical details of the underlying sensors.

An immersive experience of physical activities requires not only a scene visualization simulating realistically the user's point of view and the capture of the gestures designed to interact with the application, but also an accurate visual feedback of their body pose and movements. Commercial sensors such as Leap Motion [2] (short range, suitable for hand tracking), Creative Senz3D [3] (middle range, targeting gesture recognition and face tracking) or Kinect (larger range, suitable for body pose estimation) provide excellent capture results in their corresponding range, but cannot handle by themselves the full-body and detailed limb tracking that a first-person immersive application would require. To overcome such limitation, the information provided by different and complementary sensor types can be fused into one single 3D model, conforming the multiple resolution levels that detailed full-body pose would require. Such option, although possible, would combine sensors and displays from different vendors with no common SDKs for joint development: therefore, their combination would require ad hoc technical developments, limiting thus the interest of general app developers in it.

In order to address such limitation for app development, this work presents a development tool for the creation of immersive experiences in which the user can have the perception of full body control. The proposed tool fuses Kinect 2 and Leap Motion data for body pose estimation including detailed arm orientation and fingers, and Oculus Rift [4] for immersive first-person visualization: these choices are justified through this paper. This tool, created within the 2D/3D video game development environment Unity 3D [5], provides potential application developers with a straightforward asset to create accurate full-body animated avatar ready for its use in immersive applications.

2. OBJECTIVE AND TOOLS

This paper details the design of a development tool for the creation of immersive physical apps where users experience a 3D first-person perception of full body control, based on commercial motion sensors and a Head-Mounted Display (HMD). Unity 3D is used as the unifying framework where user pose, virtual scene and immersive visualization functions are coordinated, and where future developers can create in an easy and intuitive way different types of immersive applications implying body control.

The following subsections provide a brief analysis of the main available technologies that enable our tool, with special emphasis on those aspects driving our design choices.

2.1 Pose estimation sensors

Pose estimation sensors provide absolute or relative position and orientation of human users, either on their global body pose ([6][7]) or, more restricted but detailed, on certain parts of their bodies (e.g. head [9], hands [8] or fingertips [10]).

Several types of pose estimation sensors can be distinguished according to the technology they are based on, which conditions their scope and use. Systems based on magnetic sensors [6][8], infrared (IR, both passive and active sensors) [11] and optics (visual/depth cameras) [7] appear as the most capable technologies nowadays [12]. However, systems requiring users to wear physical components (such as magnetometers or IR emitters or passive reflectors) result in an unnatural experience, compared to the wearable-free pose estimation achieved by camera-based systems.

Subjective user experience indicates that realistic hand pose and movement play a crucial role in a natural perception of a first-person full body experience. Their relative importance might be rooted in their direct importance in terms of interaction with the surrounding (in our case, virtual) environment, based on actions such as lifting or pushing. Unfortunately, no existing wearable-free body pose estimation device achieves the resolution and accuracy required for this matter: therefore, specific hand tracking devices appear as an indispensable complement to full-body detectors for immersive perception.

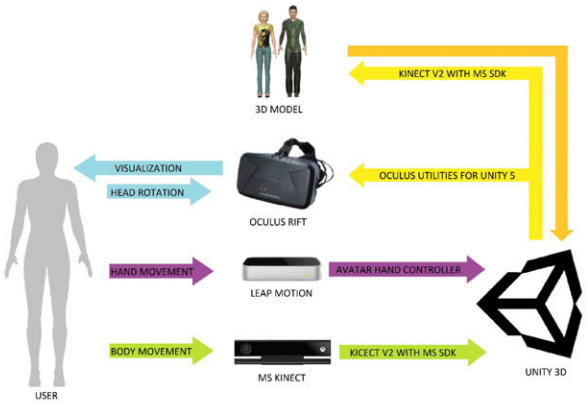

Fig. 1: Framework structure

The combination of full-body and hand trackers presents an additional restriction: interference. Most wearable-free systems rely on different usages of IR light, either in the form of structured light patterns, time-of-flight (ToF), or reflected intensity measurement. Therefore, not all existing devices are compatible, which restricts even more the joint choice for wearable-free body and hand tracker devices.

Our framework has taken all discussed aspects into consideration, along with their great market penetration, to choose the combination of Kinect 2 and Leap Motion for user full-body + hand positional info: the validity of this choice has been proved and evaluated by previous existing works on immersive experiences [13].

2.1.1 Body: Microsoft Kinect

Kinect is a motion sensing device designed by Microsoft for video game interaction and computer interfacing. It performs body pose tracking based on a RGB+D camera: its first version, of insufficient angular and depth resolution, was based on structured IR light; Kinect 2, the version used in this work, achieves a much finer resolution using a IR ToF camera.

Kinect 2 is provided with an SDK containing drivers and API, which allows accurate body pose inference, in the form of an animated skeleton, at distances between 0.5 m up to 4.5 m from the sensor, and scripts exemplifying application development. Inferred body skeletons includes two segments to describe hand (not finger) position and orientation: however, this information proves inaccurate at a medium range distance.

2.1.2 Hands: Leap Motion

Considering the importance of realistic, high resolution hand and finger tracking, a dedicated Leap Motion tracker has been included to this end.

Leap Motion controller is a small (7.5 × 2.5 × 1.2 cm) sensor using 3 high-powered infrared LEDs and two high resolution cameras for hand tracking. It does not use structured light or ToF, nor builds a proper depth map: instead, its SDK uses undisclosed algorithms based on stereo to track two hand skeletal models providing satisfactory performance even under partial occlusions.

2.2 Visualization: Head-mounted displays

Although they are not the only approach to immersive visualization (auto-stereoscopic/lightfield displays or, more recently, projector arrays have been also proposed to this end), HMDs appear as the most capable technology. HMDs, integrated into googles or a helmet, are able to accurately track user's head and simulate head orientation and/or motion parallax. HMDs are categorized depending on the number of displays into monocular (one for both eyes) or binocular [14]. They have been used for both virtual (VR) and augmented (AR) reality applications [15][16]: the number of currently existing commercial low cost HMDs, along with the increasing amount of 360VR content and games, ensures the life of this approach.

2.2.1 Oculus Rift

Oculus Rift is an HMD headset developed by Oculus VR that incorporates embedded inertial sensors hat monitor user's head orientation to adjust the displayed viewpoint accordingly.

Its second generation (referred to as Development Kit 2 or DK2), with a resolution of 960 × 1080 pixels (per eye) and 75 fps, has been chosen for the proposed framework due to its acceptance in the market and the vast existing support for its integration with multiple other devices and platforms. In addition to the head tracker sensors embedded in the helmet, it provides also an optional 3D positional tracking based on an external USB webcam and IR LEDs placed on the headset [17]. Although this external IR system can increase head pose accuracy, its use has been discarded in our framework as explained in Sec. 3.1.2.

2.3 Unity

Unity 3D is an integrated development environment by Unity Technologies. It allows the creation of 2D/3D games, visualizations and animations, and provides a dropdown interface enabling fast application development. In addition, Unity is cross-platform (aiming at PC, consoles, mobile devices and websites), and provides support for a large set of platforms.

Unity provides a series of advantages for app development that makes it perfect for the requirements of our work:

- Modular and rapid app development: Unity provides an environment, and 2D/3D modeling and animation tools for easy application development, providing also scripting support in UnityScript (similar to JavaScript) or C#. In addition, all elements in a Unity project, either imported from other source (3D models, audio files, image etc.) or created inside the project are represented in a form of an asset: these are elements that can be gathered in packages and reused, without any loss of functionality, in different projects. Additionally, Unity also provides its Asset Store to expedite application development. It is an online database with different resources, created by Unity developers and members of the community, that: can be directly imported into a project. Precisely, our proposal is based on assets from the Asset Store to drive the integration of Kinect2, Leap Motion and Oculus.
- Third-party 3D model support: Unity, which is not an advance 3D modeling tool itself, supports diverse file 3D modeling formats, and professional 3D model deigning tools such as 3DS Max, Maya or Blender have dedicated plug-ins to export models directly into a Unity scene.

3. DESCRIPTION OF THE FRAMEWORK

The developed framework, summarized in Fig. 1, comprises three key elements: 1) the integrated motion capture block, responsible for fusing the multi-level pose information provided by Kinect 2 and Leap Motion (and by Oculus VR) and making it available in Unity 3D, 2) the app development workspace, given by Unity 3D itself, and 3) the immersive visualization module, driven by and aimed for Oculus VR and assisted by additional readings provided by Kinect 2. First and third blocks, main subjects of the designed tool, are respectively detailed in Sec. 3.1 and 3.2; the second block, the developer's workspace, is directly provided by Unity 3D itself and therefore is not detailed further in this paper (although its role in our proposal is clarified by the app development guidelines provided in Sec. 4).

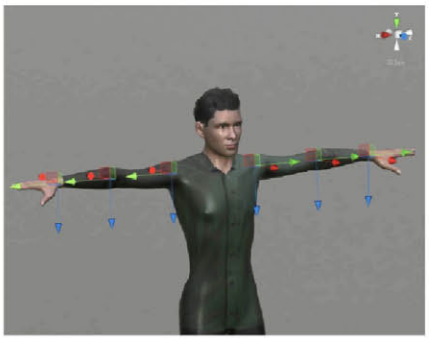

Fig. 2: Joint coordinate systems for the limbs

As stated previously, all the development tools are centralized in Unity 3D as a Unity project, where the different functional parts are included in the form of assets or combinations of them.

3.1 Integrated motion capture module

Assets from Unity Asset store have been used both for body and hand pose capture and handling. This information controls a human 3D model required by the developed tool; or, more precisely, a models' avatar, which is a definitions which inform Unity's animation system how to animate the transforms of a model.

3.1.1 3D model

Human 3D models might be created in an external modeling tool by any developer for each new app (see Sec. 4), as long as they fulfil certain requirements assumed by our design. First, the 3D mesh must be bounded to a skeleton rig consisting of bones and joints settled in a hierarchical structure: the parent (root) joint places the model in space and sets the global coordinate system for the model, and every subsequent joint will be connected to the root either directly, or through another joint. Second, joints belonging to the same limb (and arranged hierarchically as such) shall be referred to local coordinate systems sharing the same orientation (Fig. 2). Third, the rig should also have forward (FK) and inverse (IK) kinematics enabled. Fourth, avatar hand model must correspond with the Leap Motion hand model defined in its SDK.

Additionally, the model's rig shall be configured in Unity as "human" using its specific humanoid animation tab: this option is, usually performed automatically, make the existing skeleton match Unity's avatar bone structure. Finally, providing the model with a physical behavior (allowing it to be manipulated by different forces and interact with other game objects) requires the attachment of a Rigid Body component to it.

3.1.2 Body control

As mentioned and depicted in Fig. 3, tracking of different body parts is controlled by different devices: general body parts are controlled by Kinect 2; head position/orientation is captured complementarily by Kinect 2 plus Oculus VR inertial sensors; and hand pose is inferred by Leap Motion, and is referred to the absolute body position using the previously inferred head position.

- *Body*

Most body parts, and head and hands only partly, are controlled by Kinect 2. Kinect is represented by an empty object in Unity workspace, not having any visible feature but holding the necessary components and scripts. Sensor position and angle, required for a correct coordinate referencing of Kinect readings, must be set manually in the Unity inspector window.

The effective integration of Kinect's readings has been achieved by the scripts included in the *Kinect-v2-with-MS-SDK-v2.7.0* package. The two most important scripts are *KinectManager*,

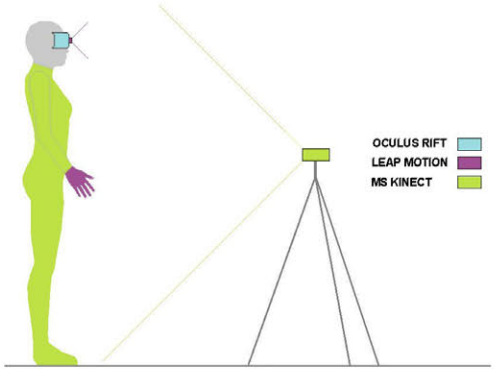

Fig. 3: Physical setting and body control division

which polls for sensor data (including color, depth, and body pose), and *AvatarController*. This later script transfers the positions and orientations of the user joints, read by *KinectManager* to the rigged skeleton linked to the human 3D model in an automatic manner. To this end, the script requires the manually assignation of joints and skeleton parts: this manual matching allows to disconnect model hands from Kinect 2 and to use Leap Motion as their controller instead.

- *Head*

Oculus DK2, used in this work, includes positional tracking to provide immersive visualization: however, those positioning capabilities also allow human 3D model positioning and animation. We explicitly use Oculus positional readings, made available within Unity through the corresponding official package. Such package contains multiple scripts and prefabs (prefabricated game object with attached components and scripts). Positional control has been achieved through the *OVRPlayerController* prefab.

Oculus DK2 performs orientation tracking using inertial sensors located within the headset, but adds an optional high-accuracy 3D tracking based on IR emitters included in the helmet plus an external IR camera. Unfortunately, the use of this absolute position tracker requires additional cable connections, resulting in a drastic limitation of user's range of displacement. Since such 3D position information can also be inferred from Kinect 2 readings (from the corresponding head joint), Oculus 3D positioning has been discarded for the framework.

- *Hands*

As discussed in Sec. 2.1, full body control feeling requires a finer tracking of hand and finger pose: our framework uses Leap Motion controller to that end. Official Leap Motion SDK offers Unity assets that integrate hand tracking by creating predefined visible hand models. Our framework requires hand readings to get transferred to (part of) a different avatar (the human 3D model partially animated by Kinect 2) instead: therefore, we have based hand control on the *Avatar Hand Controller* for Leap Motion available in the Asset Store.

Leap Motion sensor can provide forearm pose estimates basing on the read pose of its corresponding hand. This solution, however, is not based on direct readings: for this reason, Leap Motion forearm estimation has been discarded, using Kinect 2 readings instead. Hand positional readings are relative to head position, since Leap Motion controller is placed on the Oculus VR headset (facing front): thus, absolute hand/finger position is the product of the absolute head position, combination of Kinect 2 and Oculus VR, and their relative tracked pose (Fig. 3).

Hands are controlled using IK to automatically calculate the required joint transformations in order to orient the joints in the desired position. Avatar hand models are not recognized by default

by Leap Motion, so they need to be manually defined using the appropriate corresponding scripts *RiggedFinger* and *RiggedHand*. Both scripts include two vectors, finger pointing and palm facing, that have to be determined according to the rig definition.

3.2 Immersive visualization module

As indicated in Sec. 3.1.2, official Oculus VR SDK contains assets for the integration of the device in Unity. The *OVRPlayerController* prefab, used in our framework for controlling model head, is also used for 1st person scene visualization. Such prefab includes a physics capsule, a cross-hair component, and the *OVRCameraRig* prefab, which replaces regular camera in scene.

The mentioned *OVRCameraRig*, representing the binocular camera required by the Oculus headset, has been placed at the eye position of the 3D model but slightly shifted along z-axis in order to prevent situations where users can see through their human 3D model. In order to keep the camera at the proper height for all possible users, preset data from Oculus configuration is discarded and detected users' height is used instead.

4. IMMERSIVE APP DEVELOPMENT EXEMPLIFIED

In order to exemplify app development using the presented framework, a simple example interactive application was created. Such activity consists of a virtual room where user can interact by displacing, using hands and/or feet, floating cubes (Fig. 4).

Human 3D model was created and customized in Autodesk Character Generator online application. Although the downloaded models were ready-to-use and compatible with Unity, for better integration with the environment and implemented devices, additional rigging process was conducted using Adobe Mixamo Auto-Rigger tool.

Components used as scene background can be created using Unity, or downloaded from the Asset Store. Unity also allows the creation some primitive objects, such as cubes, spheres, capsules, or cylinders, directly in the scene. The room in the example contains cubes including interaction scripts, which are responsible for their movement and interaction with the user when collision with the corresponding cube and model colliders is detected. Colliders define the shape of the model for the purposes of physical collision: they are invisible and for primitive objects align with their shape. In the example app seven colliders were attached to the 3D model: one to the whole avatar to avoid permeation with the floor and six to hands, knees and feet, respectively.

5. FUTURE IMPROVEMENTS

The final version of the development platform meets the initial assumptions and can serve as a base for application development; however, there are still some refinements to be made. The basic problem is the large number of cables required for the system – both Leap Motion and Oculus need fixed connection.

Another limitation is the range of motion of the Leap Motion sensor, which loses track when hands are more than 35 cm away. Hands are not resetting to initial position, but stay at last remembered gesture. Also, due to division of control over different parts of upper limb, its rotation looks unnatural. Kinect, responsible for forearm control, is not detecting bone rotation while Leap motion identifies hand rotation. During such movement at the contact point (wrist) mesh is strained which does not look natural.

An extensive evaluation of the quality of experience needs to be performed to guide future improvements in the framework. Finally, although the framework is currently limited to the use of a defined set of hardware devices, it may be adapted to integrate multiple peripheral devices, from different vendors, by means of interface layers such as OSVR [18] or OpenVR [19].

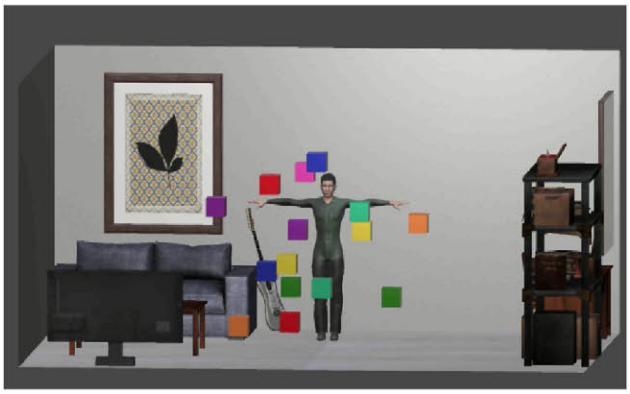

Fig. 4: Interactive app developed as an example

ACKNOWLEDGEMENT

This work was partly supported b by the Spanish Government under grant TEC2013-48453 (project MR-UHDTV) of the Ministerio de Economía y Competitividad.

REFERENCES

[1] E. Lachat, et al. "First experiences with Kinect v2 sensor for close range 3D modelling." *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 40.5 (2015).

[2] https://www.leapmotion.com/

[3] http://uk.creative.com/p/web-cameras/creative-senz3d

[4] https://www.oculus.com/

[5] https://unity3d.com/community

[6] http://www.priovr.com/

[7] http://vicovr.com/

[8] http://polhemus.com/micro-sensors/

[9] https://naturalpoint.com/trackir/

[10] http://www.intugine.com/

[11] http://www.delanengineering.com/

[12] G. Ligorio, A. M. Sabatini. "Extended Kalman filter-based methods for pose estimation using visual, inertial and magnetic sensors: Comparative analysis and performance evaluation." *Sensors* 13.2 (2013): 1919-1941.

[13] S. Greuter, S. Kenderdine, and J. Shaw. "Pure Land UNWIRED: New approaches to virtual reality for heritage at risk." In "Creative Technologies for Multidisciplinary Applications," Chapter 4, pp. 76-98. IGI Global, 2016.

[14] K. Kiyokawa. "Trends and vision of head mounted display in augmented reality." *International Symposium on Ubiquitous Virtual Reality*, 2012.

[15] O. Cakmakci, J. Rolland. "Head-worn displays: a review." *Display Technology, Journal of* 2.3 (2006): 199-216.

[16] M. Stengel, et al. "A non-obscuring eye tracking solution for wide field-of-view head-mounted displays." *Eurographics* 2014.

[17] B. A. Davis, K. Bryla, P. A. Benton. "*Oculus Rift in Action*." Manning Publications Company, 2015.

[18] http://osvr.github.io/whitepapers/introduction_to_osvr/

[19] https://github.com/ValveSoftware/openvr