2.2. New software

2.2.1. *Introduction*

The first part of this chapter revealed the multitude of equipment needed to allow a virtual reality system to give the user the sensation of being in a virtual world and interacting with it. Similarly, augmented reality requires specific equipment to analyze the real world and allow the overlay of virtual objects. The software used to build a VR-AR application must therefore make it possible to optimally use all these devices and make them communicate with the digital simulation that processes the information received and computes the information to be restituted to the user. This device must therefore simultaneously manage a large number of functions, as shown in the interaction cycle. This cycle goes from the user's action until the perception of the result of this action (Figure 2.9).

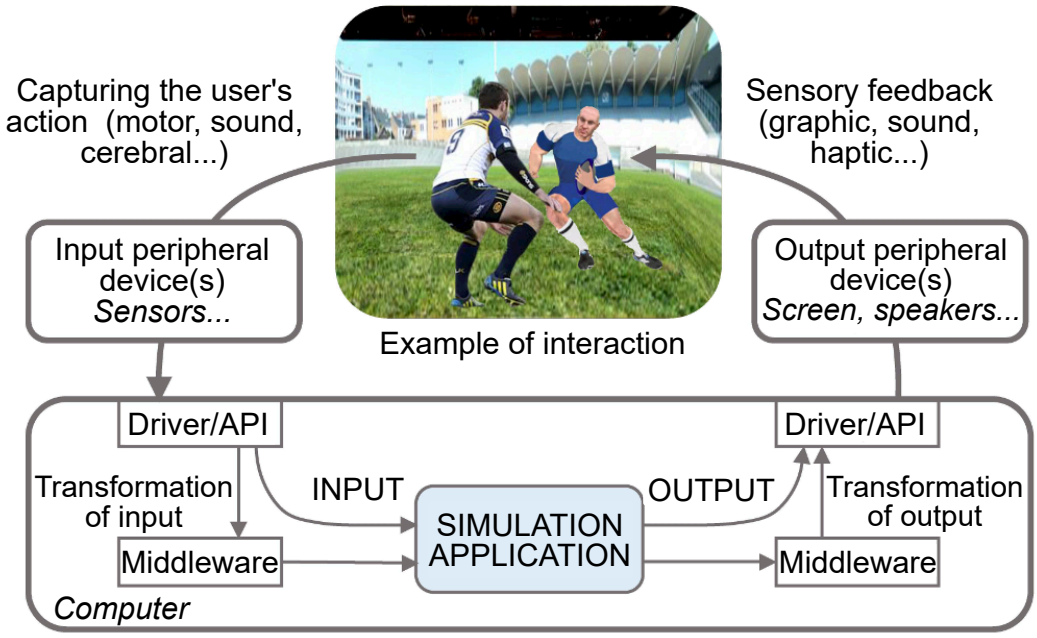

Figure 2.9. *Interaction cycle starting from the user's action until the perception of the result of this action. Developing a VR-AR application requires collecting data from the input devices, processing this information and deducing the sensory feedback to produce, then transmitting this information to the output devices*

Beyond the simulation of the 3D virtual world in which the user is immersed (in the case of VR) or which is superimposed on to the real world (in the case of AR), the application must be able to guarantee interaction between the user and this simulation. That is, it must be able to read the user's motor actions and supply the appropriate sensory information. Let us consider, for example, a VR-based sports training tool, the objective of which is to train a rugby defender to block an attacker who attempts to go around him with or without a body swerve (see Figure 2.10). In order to be successful, the application must provide a virtual adversary who reacts to the defender's real actions and adapts their attack. The first step for such an application consists of collecting the defender's movements using a motion capture device. The motor data is then transmitted to a computer through a driver that the application can consult via an interface called API (Application Programming Interface). The simulation then computes the virtual attacker's reaction, based on the defender's real action. This reaction by the attacker is translated through a modification in his animation, which is then transmitted to the output device via another API. The simulation must also manage other parameters such as change in the immersed subject's (user's) point of view, for example, due to the position of their head, if they are in a CAVE, or through their position/orientation, if they are using a headset (see section 3.2). The sensory feedback is then carried out by the output device or devices, here, for instance, only through a visual feedback in stereovision in the virtual environment that includes the attacker.

Figure 2.10. *Example of the interaction in virtual reality between (a) a real defender, fitted with a VR headset, and (b) a virtual attacker who may or may not use a body swerve to go around him*

Owing to the complexity of developing such VR-AR applications, it is common to use specific software. That is why many companies have specialized in developing solutions for a specific field. Just a few examples are XVR Simulation [XVR 17] for training in the field of safety and security, iris [IRI 17] for architecture, IC.IDO [ESI 17] for industrial prototyping, ParaView [PAR 17] for the analysis and visualization of complex data, FlowVR [FLO 17] for large-scale parallel simulation, and Augment [AUG 17] for managing and visualizing 3D content in AR. It is therefore possible to use this kind of "turnkey" application but, in this chapter, we will be discussing different approaches to creating a specific VR-AR application.

In accordance with the interaction cycle, the development of a VR-AR application can be divided into two parts. The first part, described in section 2.2.2, consists of developing digital simulations that process information obtained by the input devices and that compute the results to be furnished to the output devices. The second part, described in section 2.2.3, concerns the communication between this simulation and the input and output devices.

2.2.2. *Developing 3D applications*

A VR-AR application is based on use in a 3D world in which the user is immersed (in the case of VR) or which has been superimposed on the real world (in the case of AR). There are many ways in which this 3D environment can be managed and visualized, depending on factors such as cost, development time, flexibility or even ease of use. In this section, we will describe these different approaches, beginning with the most "basic" programming, all the way up to specific VR-AR tools.

2.2.2.1. *"Basic" graphic programming*

The most elementary approach consists of creating a 3D application by directly accessing the drivers and programming interfaces of the graphics cards of the equipment used. The drawback of this approach is that each application depends on the device. This difficulty may be overcome using programming interfaces such as OpenGL or DirectX, which make it possible to work outside of a particular type of equipment, rather than restricting oneself to a specific device. The main advantage of this approach is that it offers complete control over the entire creation process from the 3D

environment to how it is rendered graphically. It is thus possible to directly control the facets that make up the 3D objects, to create one's own scene graph, the hierarchic structure used to define relations and transformations between objects and notably manage animations or even propose new structures. This also allows us to control the graphic pipeline, the succession of steps required to go from calculating these facets until their final rendering, via the elimination of the hidden parts and application of textures and lighting. It is thus possible to choose during what step of the process this or that action must be carried out to optimize the application's performance.

This option therefore potentially offers the best performance as well as a very high flexibility. However, the work required is much more complex, as it requires the creation of the desired functionalities, the loading of the 3D environment (resulting from a modeler such as 3DS Max or Maya, for instance) and the recovery of data from the motion capture. The main drawback is the inability of such an approach to be portable.

2.2.2.2. *Graphic libraries*

In order to avoid creating all the required functionalities, libraries such as OpenSceneGraph [OPE 17b] offer a slightly greater control of 3D models, thanks mainly to how they manage the loading and saving of these models, animation methods used for the objects and also the control over lighting and shadow, camera placement, etc. These libraries make it possible to significantly speed up the creation of a 3D application. However, they still remain rather restricted to specialists.

In addition, some of these libraries may depend on operating systems such as Windows, Linux or Mac OSX. It is therefore difficult to develop solutions that will also work on mobile telephones or on video game consoles. Finally, in the context of VR-AR applications, one of the major problems is that they focus on the modeling, animation and rendering of 3D objects, but very rarely manage the associated VR-AR devices or the different sensors - all of which are, however, important elements in an interactive application. Apart from the cost of developing these interfaces with these peripheral devices (see section 2.2.3.1), it is, above all, the maintenance and evolution of these applications that pose a problem, given the extremely rapid developments in the field of VR-AR and the constant emergence of new peripheral equipment on the market.

2.2.2.3. *Video game engines*

With the aim of being more productive, the video game industry has, for many years, developed generic environments called “engines”, that are central to all their productions. These engines are now associated with highly powerful editors that make it very easy to create 3D applications. These editors (notably, via a graphic interface and without development) make it possible to manage the visual layout of a scene, the sound, the camera, the animation, etc. (see Figure 2.11). In addition, these engines work on different platforms; computers, mobile telephones or video game consoles. Hence, they are widely used to make games, not only on mobile telephones and video game consoles but also for online games.

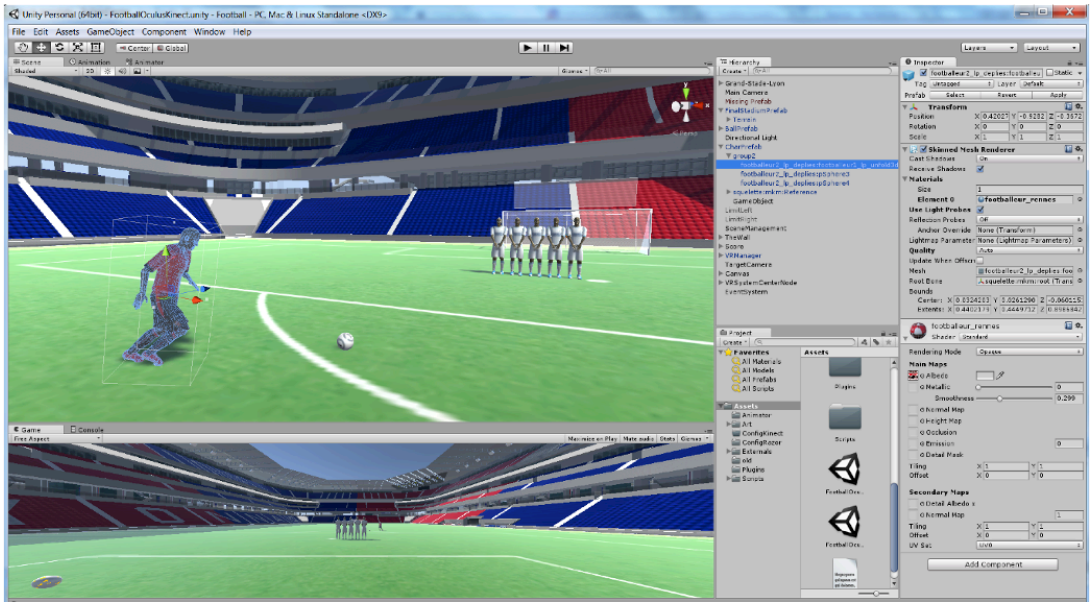

Figure 2.11. *Example of the graphic editor of the game engine Unity, which makes it possible to easily manage the visual layout of a scene, the sound, the camera, placement, etc. For a color version of this figure, see www.iste.co.uk/arnaldi/virtual.zip*

Finally, among the many existing engines, each with its own fame and ease of use, the best-known are: Unity [UNI 17], Unreal [UNR 17], Cry Engine [CRY 17], Ogre3D [OGR 17] and Irrlicht [IRR 17]. Apart from the ability to produce the same content on different platforms in very little time and, above all, using the same application, these engines offer a large number of functionalities that speed up the creation of these applications. These

include: managing all the graphic parameters related to the rendering of objects, lighting and camera placement. However, they also additionally permit the management of the physical simulation of objects (taking shocks into account, for example), spatialized sound to diffuse the sound environment by taking into account the sources of the sound, and the animation of complex structures such as virtual humans.

Finally, these tools have resulted in a large community of users who offer many additional resources such as tutorials to make them more accessible, as well as the development of scripts that can widen their functionalities, including within the field of VR-AR, as shown in the following section. Among the engines listed above, Unity has currently emerged as one of the major actors, due to its ease of use. It can thus be used by other communities such as the neurosciences, Sports and Physical Activities, medicine, etc.

2.2.3. *Managing peripheral devices*

After having developed the heart of the application, namely simulation, this must now be made to communicate with the user immersed in the experience, by exploiting the peripheral input devices which acquire motor information from the user and the output peripheral devices, which produce sensory feedback. Just as with the creation of 3D graphic simulations, it is possible to manage the interface with the peripheral devices at different levels, from direct control via a programming interface to the most high-level and most generic tools.

2.2.3.1. *Direct control of peripheral devices*

In order to allow an application to communicate with a peripheral driver, the constructor provides a programming interface that gives access to all the functions, making it possible to control this device or to exchange data with the device. Thus, the developer only needs to call upon these functions to allow an application to manage the peripheral devices. In practice, all equipment differ from one another and the programming interfaces may be very varied, including for peripheral devices that offer exactly the same functionalities. For example, depending on whether the peripheral device is connected through a USB port or through Bluetooth, the interface is likely to be different. Similarly, if you have two rotation sensors developed by two different manufacturers, it is

highly unlikely that they will have the same interfaces, at least for the function names.

Fortunately, certain norms have emerged resulting in standardized programming interfaces for classic peripheral devices (keyboards, mouse, joysticks, audio headsets or printers), which make it possible to access any keyboard or mouse without wondering about the manufacturer of the equipment. A change in brand does not prevent the application from working and, above all, requires no modification of its code. Unfortunately, there is no such standard, at present, for VR[13], leading the application developer to update their software for each new device and its associated interface. To avoid having to update these devices for each new equipment, the developer must construct an abstraction of the peripheral devices based on their functionality (a motion capture sensor, for example) and then create a new instance of this abstraction for each new equipment. In addition, the multiplication of the links between the application and the different interfaces increases the problems related to the management of different versions of these interfaces and the auto-detection of each equipment used. With the constant development of a large number of VR-AR tools, directly controlling these peripheral devices poses a large problem to app developers in terms of maintenance.

2.2.3.2. *Libraries for managing peripheral devices*

Libraries to manage peripheral devices were proposed in order to simplify communication with these devices. They offer abstractions that make it possible to address generic equipment offering a standardized interface rather than equipment of one particular brand. For motion sensors, for example, position and/or rotation information may be collected using the same functions, regardless of the technology used by the sensor. These libraries also offer a more or less simple means for the user to specify what peripheral device they are currently using, or even automatically detecting this when the application is launched. Finally, these libraries make it easy to configure peripheral devices by specifying initial data, for example, for display on the screens of a CAVE (see Figure 2.12) or even the initial positions of the joystick or headset.

13 In fact, such efforts at bringing in norms have been made in VR as well as AR, but these norms have not been applied.

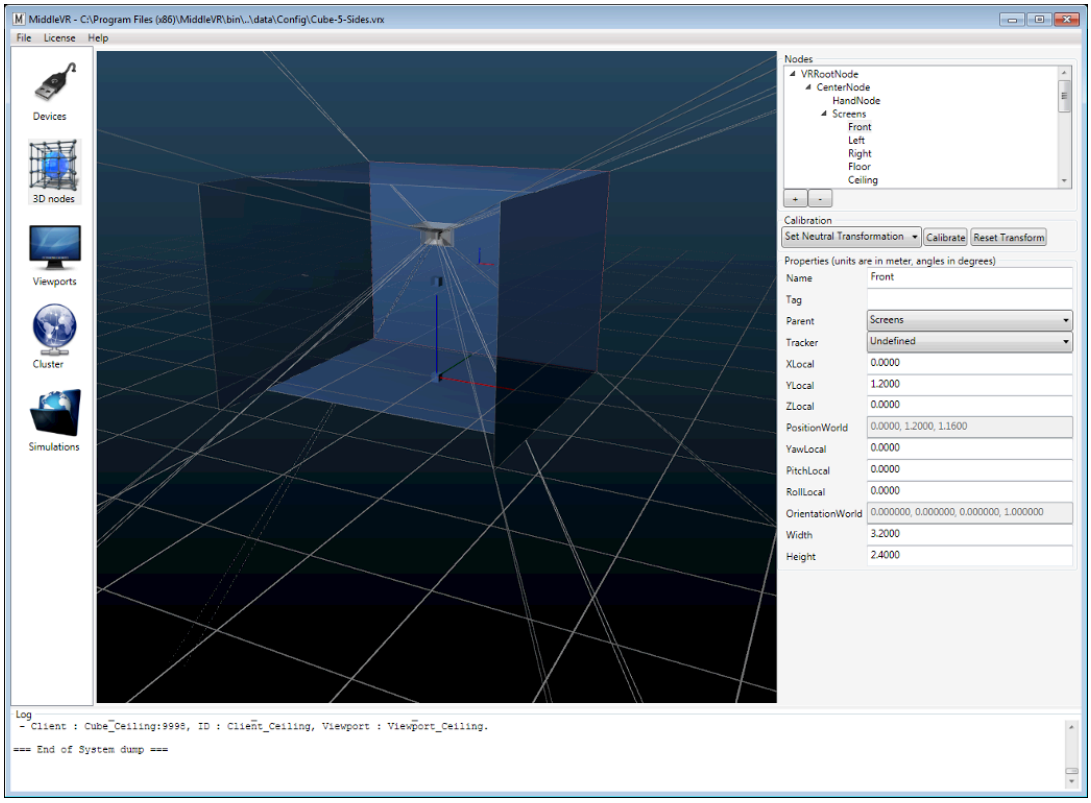

Figure 2.12. *Example for the configuration of a five-face peripheral visualization device, using MiddleVR. For a color version of this figure, see www.iste.co.uk/arnaldi/virtual.zip*

Apart from collecting data from the input peripheral devices, some of these libraries, such as VRPN (Virtual Reality Peripheral Network) [VRP 17] and trackd [TRA 17], are able to take into account the equipment that is connected to one or more computers through the network. This characteristic makes it possible for the developer to distance themselves from the chosen material architecture and communicate with its sensors, whether they are at a distance (through the network) or local (the same machine). Other libraries, such as CAVElib [CAV 17] are focused on the visual restitution of the simulation, with the management of the changes in point of view and stereovision on the varied projection configurations: from a simple screen to multi-screen and multi-machine systems such as the CAVE systems. Finally, some libraries propose managing all these different peripheral devices such as OSVR (Open-Source Virtual Reality for gaming) [OSV 17] or MiddleVR SDK [MID 17] and TechViz [TEC 17], which are libraries that are equipped

with a middleware, which is an external software positioned at the interface between the application and the equipment. In this case, its role is to provide a software interface to easily configure the different equipment that is to be used with the same software application.

As concerns AR, several libraries offer specific functionalities such as the evaluation of the position and orientation of the user in interactive time in real space. OpenCV [OPE 17a] makes it possible to acquire and process the images, from the detection of structures ranging from lines to complex motifs. All these functionalities make it possible to superimpose a 3D virtual object onto the real world observed by the user. The largest libraries are ARToolkit [ART 17], Vuforia [VUF 17] and Wikitude [WIK 17], which offer all the functionalities described above, managing mobile platforms and VR-AR headsets, and offer interfaces for the development tools (see section 2.2.4.2). Apple's ARkit will be launched in September 2017, providing the same functionalities to platforms based on iOS.

2.2.4. *Dedicated VR-AR software solutions*

Other higher-level software solutions propose integrating the management of 3D environments and peripheral devices in order to simplify the creation of the VR-AR application.

2.2.4.1. *Dedicated tools for the creation of VR-AR*

Some graphic tools can be used to simplify the process of creating VR-AR applications. For AR, for example, Wikitude offers developers an SDK on which a software solution can be constructed, which allows the recording of images to be recognized and then the content to be associated with these images without programming or even publishing the applications in one's own virtual shop.

The Eon Creator software, distributed by the company Eon Reality [EON 17], offers a similar VR-AR solution, which makes it possible to select 3D models, interact with these models and easily diffuse this content. It also offers a complete development environment to manage functionalities similar to those of video games, such as feedback and physical simulation. In addition to a similar development environment, the WorldViz software [WOR 17] makes it possible to manage peripherals with different

projections and/or multiple users, etc. It also offers configurations embedded in the equipment.

These tools dedicated to the creation of VR-AR applications remain small in number, however. Indeed, editors must dedicate a large number of resources to developing them. Restricting them only to VR-AR limits their use to a rather niche market today, which makes it difficult to make these solutions competitive with respect to a generic game engine reinforced with plugins.

2.2.4.2. *VR-AR plugins for video game engines*

Section 2.2.2.3 described the emergence of video game engines for the creation of 3D application content. Even though this software is not originally meant for VR-AR, their ease of use, their multiple functionalities and their openness to developing additional scripts ensure that they are relevant reference tools in these fields. Indeed, to compensate for the absence of management of the peripheral devices, which are one of the main components needed to use an interactive application, the developer community that uses these tools first created specific plugins based on the programming interfaces used by the constructors. With the large rise in numbers in this community, constructors now directly offer plugins to communicate with their new models as soon as these are launched and sometimes engines even integrate them as native tools as is the case with the Oculus and HTC Vive headsets and the Unity software.

Other companies offer more generic plugins used to manage peripheral devices, which are directly integrated into these motors. These plugins also extend existing functionalities with advanced management of stereoscopy, multi-computer synchronization in a cluster, force-feedback peripheral devices and managing multiple users in virtual reality. Some actors have developed generic libraries (see section 2.2.3.2) for output peripheral devices, such as getReal3D, or for all peripheral devices, such as MiddleVR for Unity or Techviz. Similarly, for AR, there are libraries such as Vuforia or Wikitude.

2.2.5. *Conclusion*

VR-AR applications are increasingly being developed with the help of video game engines such as Unity. Indeed, constructors now directly offer

plugins that communicate with their new peripheral devices and generic integrated solutions also exist, such as MiddleVR for Unity. This type of development makes it possible to implement solutions much more rapidly and at a lower cost, and to adapt them to the new peripheral devices without even recompiling the application. In addition, and most importantly, they make it possible to easily manage the addition of a new peripheral device, which is essential given the continuous development of new, low-cost VR-AR devices.

2.3. Bibliography

[ART 17] ARTOOLKIT, "ARToolkit", artoolkit.org, 2017.

[AUG 17] AUGMENT, "Augment", www.augment.com, 2017.

[AZU 97] AZUMA R.T., "A survey of augmented reality", *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, August 1997.

[CAV 17] CAVELIB, "CAVElib", www.mechdyne.com/software.aspx?name=CAVELib, 2017.

[CRU 92] CRUZ-NEIRA C., SANDIN D.J., DEFANTI T.A. *et al.*, "The CAVE: audio visual experience automatic virtual environment", *Communication ACM*, vol. 35, no. 6, pp. 64–72, ACM, June 1992.

[CRU 93] CRUZ-NEIRA C., SANDIN D.J., DEFANTI T.A., "Surround-screen projection-based virtual reality: the design and implementation of the CAVE", *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, pp. 135–142, 1993.

[CRY 17] CRY ENGINE, "Cry Engine", www.cryengine.com, 2017.

[EON 17] EON REALITY, "EON Reality", www.eonreality.com, 2017.

[ESI 17] ESI GROUP, "IC.IDO", www.esi-group.com, 2017.

[FLO 17] FLOWVR, "FlowVR", flowvr.sourceforge.net, 2017.

[FRE 14] FREY J., GERVAIS R., FLECK S. *et al.*, "Teegi: tangible EEG interface", *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST'14, New York, USA, ACM, pp. 301–308, 2014.

[FUC 05] FUCHS P., MOREAU G. (eds), *Le Traité de la Réalité Virtuelle*, Les Presses de l'Ecole des Mines, Paris, 2005.

[FUC 09] FUCHS P., MOREAU G., DONIKIAN S., *Le traité de la réalité virtuelle Volume 5 - Les humains virtuels*, Mathématique et informatique, Les Presses de l'Ecole des Mines, Paris, 2009.

[IRI 17] IRIS, "iris", irisvr.com, 2017.

[IRR 17] IRRLICHT, "Irrlicht", irrlicht.sourceforge.net, 2017.