B. K. Szabó: "**Using a virtual reality headset in the simulation of the control room of a nuclear power plant**", Proceedings of the 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS), Debrecen, Hungary, 16-18 May 2022, pp. 249 - 252, DOI: 10.1109/CITDS54976.2022.9914190.

# Using a virtual reality headset in the simulation of the control room of a nuclear power plant

B. Katalin Szabó
*Reactor Monitoring and Simulator Laboratory*
*Centre for Energy Research*
Budapest, Hungary
http://orcid.org/0000-0003-3010-2510

***Abstract*—Simulators for operator training have long been in use for nuclear power plants. These are traditionally full-scope simulators with an expensive physical replica of the control room. Recent advances in virtual reality headsets provide affordable means for presenting a stereoscopic view of a virtual model of the control room. While the commercially available headsets are still not perfect, they offer more realism than flat screens, showing a stereoscopic view. The paper provides some details on how a virtual reality headset has been utilized for viewing the virtual control room of the Paks Nuclear Power Plant, modeled with the Blender Game Engine, with an added autofocus feature based on a pragmatic method. Some aspects of the solution are outlined, and, partially based on the experiences gained in the project, current problems and future trends of virtual reality headsets are discussed.**

***Keywords—control room, simulator, virtual reality, headset, autofocus***

## I. INTRODUCTION

The full-scope replica simulator of the Paks Nuclear Power Plant, mainly for operator training, has been in use since 1988 (with several refurbishments). It contains a physical replica of the main control room which is a traditional control room containing a lot of pushbuttons and switches. Three operators and their supervisor work in the huge control room which has panels in multiple rows providing information and control, as can be seen in Fig. 1. A virtual 3D model of this control room was created a few years ago [1] and connected to the software system of the simulator. It uses the stand-alone version of the Blender Game Engine. The first implementation utilized flat screens. Development continued with the integration of a virtual reality (VR) headset. The user, wearing the headset, is seated at a table during the use of the application. The aim was to create a demo application using affordable hardware.

## II. HARDWARE, OPERATING SYSTEM, AND GAME ENGINE

At the start of the project, the two most promising and affordable head-mounted displays available were the Oculus Rift CV1 and the HTC Vive. The Oculus Rift CV1 was chosen, which has 1080×1200 resolution per eye, a 90 Hz refresh rate, and 110° field of view. The separation of the lenses is adjustable by a dial at the bottom of the device, in order to accommodate a wide range of interpupillary distances. The Rift has integrated headphones and rotational and positional tracking. In this project only its rotational tracking (gyroscope) is used. The touchless Leap Motion hand movement detector is used for controlling the navigation in the control room and manipulating the switches and buttons [2].

Our application has been created under the 64-bit Windows 10 operating system, in a PC with Intel i7 processor and NVIDIA GeForce GTX 1080 graphics card.

Fig. 1. The real control room (source: [1])

The virtual control room has been realized in the Blender Game Engine (BGE), a component of the Blender (*https://www.blender.org/*) suite, a free and open-source 3D creation suite (cf. [3,4]). The version used in the project is 2.76b. (The Blender Game Engine is no longer developed but the old versions are still available. Also, a successor, a fork called UPGE, at *https://upbge.org/*, is still being maintained. It would also be possible to switch to another game engine. Right now the most widely used game engine for similar applications is Unity, which is able to import models made wih Blender, but the scripts would have to be reworked.)

Similar applications modeling control rooms of nuclear power plants in virtual reality, found in the literature (all dated after this project started), are [5, 6, 7, 8, 9].

The application described in [5] uses the Unity game engine with a HTC Vive headset with Tobii Pro VR Integration eye tracking, and the HTC Vive controller. The other publications are all related to the Finnish Loviisa nuclear power plant and the APROS process simulation software. [6] reports using Unity, the Oculus Rift headset with rotational and positional tracking, Oculus remote controller, keyboard, and mouse. [7] and [8] list Unity, a Varjo headset and Valve Index controllers. [9] mentions Unity, the Oculus Rift and Varjo VR-1 headsets.

## III. INTEGRATING THE HEADSET WITH THE APPLICATION

For the open-source BGE, the open-source package O*penHMD* (*http://www.openhmd.net/*) has been selected for the project. O*penHMD* is able to drive various VR headsets, including the Oculus Rift, the picture appears in the headset

978-1-6654-9653-7/22/$31.00 ©2022 IEEE

in extended mode (treating the screen of the headset as a monitor) and the package is able to handle the gyroscope input of the headset as well.

For O*penHMD*, the open-source *HIDAPI* package (*https://github.com/signal11/hidapi*) is necessary. It could be built with *Microsoft Visual Studio Express 2013*, but O*penHMD* had to be built with *minGW-w64* (because the Visual Studio Express version used did not support the C99 standard). O*penHMD* was developed for Linux, no Windows-compatible version was available, so some modifications had to be made in the source code:

In *OpenhMD.cpp*, the line *#include <Windows.h>* replaced *#include <sys/time.h>*. The *sleep* function has been changed to

```
void OpenHMD::sleep(double seconds) {
    DWORD millisecs;
    millisecs = (DWORD)(1000 * seconds);
    Sleep(millisecs);
}
```

In the *print* function, the line *float f[len];*

had to be changed to *float *f = new float[len];*

For integrating with the BGE, the *python-rift* binding (*https://github.com/lubosz/python-rift*, related blog post: *https://lubosz.wordpress.com/2013/06/26/oculus-rift-support-in-blender-game-engine/*) has been utilized.

The *openhmd.pyd* library (a special DLL library) had to be built with the *setup.py* of *python-rift*. This library defines a Python class through which the functions of the *OpenHMD* library can be called. The building of this library required some tricks, as, to match the Python version (3.4) used by Blender 2.76b, it was necessary to use a specific Microsoft Visual Studio version, namely, Visual Studio 2010, and this development system was freely available only in a 32-bit version, while our development platform is 64-bit. This obstacle was overcome by the use of the 64-bit Windows 7.1 Software Development Kit.

For the camera in the Blender scene, the most important settings in Properties – Render were the following:

Stereo: Stereo (not the default None)

Stereo mode: Side by side (not the default Anaglyph)

Shading: Multitexture (not the default GLSL)

Framing: Letterbox (not the default Scale)

## IV. Implementing an autofocus feature

When the Oculus Rift was integrated into the project and tests were made with it, it became clear after a while that a fixed-focus camera is inadequate, as the operator sometimes has to view a faraway panel, and at other times he/she has to have a close look at e.g. a switch which he/she wants to manipulate. No matter what focus was fixed before the program run, it was not adequate for all cases.

The stereo camera in the BGE has been built on the basis of real cameras rather than human eyes. It provides two pictures for the two eyes, and certain parameters can be set in it, but the possibilities for runtime adjustments are few.

The eyeballs of the two human eyes are almost parallel when looking at a faraway object. In this case the two pictures that the left and the right eye get, respectively, are almost the same. However, when looking at a near object, the eye muscles do not only change the focus of the eye lens (this process is called accommodation) but the eyeballs also turn a bit inwards around their vertical axes (this automatism is called vergence), so the two pictures that the two eyes get can differ considerably. They can be roughly approximated as images partially shifted left and right (this approximation does not model properly the inward-turning movement of the eyes when looking near). A good overview of the subject of stereoscopic view in head-mounted displays, with explanatory drawings and a list of further literature can be found in [10].

In the BGE it is possible to adjust the focus of cameras, even programmatically during the program run, and it is also possible to set a camera parameter called “eye separation” (which is a somewhat misleading name, as this distance is not equivalent with human interpupillary distance but it is the distance of the two “subcameras” in the stereo camera). However, tests revealed that the product of the values of focus and eye separation is always 30, if you change one of the values, the other will automatically change accordingly. This seems to be the hard-coding of a “rule of thumb” mentioned e.g. by *http://paulbourke.net/stereographics/stereorender/*, even though its origin is not clear. So, in the BGE, you have only one parameter in your hands when you want to focus the camera.

In principle, Blender provides a possibility to specify an object the camera should focus on continuously, but in practice this does not work in the standalone BGE (at least not in the version used for the project).

There exist Blender addons which control the focus. However, neither the currently available version of *RealCamera* (*https://3d-wolf.com/products/camera/*), nor *Photographer-4* (*https://nullpk.com/photographer-4-blender-addon/*), nor *Autofocus Pro* (formerly: *Aperture*, *https://blendermarket.com/products/aperture*) support the BGE used in this project.

Nevertheless, with the use of raycasting [11], it is possible to implement an autofocus feature programmatically.

A solution should use computing resources as sparingly as possible, as this is a real-time application which fulfills several functions periodically: rendering a stereoscopic picture for the headset, handling rotational input from the headset and hand movement input from the Leap Motion, updating the 3D model accordingly, handling data input from the simulator database and animating display objects accordingly, sending data about user actions (such as pressing a button) to the simulator database. Python scripts can be run in the model, but this is interpreted code which runs more slowly than compiled code, so, when possible, complicated calculations should rather be

performed by the inherent, compiled code of the BGE and not by an interpreted script.

As this version of the application does not include eye tracking, it is assumed that the object of interest is at the middle of the screen (the user should position himself/herself so that whatever he/she wants to focus on should be in the middle of the screen).

In order to adapt the stereo picture so that the image of the object of interest gets in focus, we have to determine how far that object is. The BGE function *getScreenRay* can cast an invisible ray from the camera towards a point on the screen (specified with x and y coordinates within the [0,1] range). The maximal length of the ray may be specified. The return value is the first object hit or *None*. On the basis of the object's identity, the distance between (the base of) the object and the camera can be calculated. (A more correct distance calculation could use the exact hit point where the ray hits the object, but this simpler method proved adequate.)

To keep calculations to the minimum in the runtime application, it was decided that the optimal eye separation value as the function of object distance should be determined experimentally. For this, an experimental build was made from the application. In this version the user can change the position of the camera and aim it at any object in the control room by orienting the view (with head rotation) in the headset so that the object of interest gets to the middle of the screen. Then, by using raycasting (described above), the identity of the object and its distance from the camera can be determined. After the object has been targeted, the user can manually increase or decrease the value of the eye separation in small increments/decrements. When he/she judges that the stereo picture in the headset is satisfactory, he/she presses a key which records the eye separation value set, together with the distance of the object. From these data, an algorithm can be derived, determining an eye separation value as a function of distance. As computing resources should be used sparingly, the algorithm had to be simple. A few keyboard keys were used for the various operations. The user had to handle these keys blindly (note: the real runtime application does not require the use of the keyboard during the simulation run while the user wears the headset).

Separate keys were used for moving the camera nearer to or away from the control room panels. Another pair of keys were used to increase/decrease eye separation value for the renderer, by a small amount. Yet another key had to be pressed when the object to focus on was in the middle of the screen and pressed again when the stereo experience was adequate (after adjusting eye separation). Then it made a "snapshot": it logged camera position, the name of the object which was in "focus" in the middle of the screen and its distance from the camera, and it also recorded the eye separation value. A further key suspended raycasting (to prevent accidentally hitting other objects with accidental head movements during the eye separation adjustment attempts). Raycasting was automatically resumed when the key to make the snapshot was pressed.

The functions *bge.render.getEyeSeparation* and *bge.render.setEyeSeparation* were used for getting and setting eye separation, respectively.

Based on the measured results, a primitive algorithm has been constructed. It uses only a few comparisons and no mathematical calculations (not even interpolations): Depending on the range the distance value falls into, a corresponding eye separation value is selected from 10 distinct eye separation values.

According to tests made with the above algorithm in effect, focusing has become adequate, some numerical displays and texts on control panels are more clearly discernible when the autofocus works. (More formal tests are planned, in which users' opinion about the implementation of the autofocus feature will be asked in a questionnaire.) For the user it takes a little time to get used to setting the view so that the object of interest gets to the middle of the screen. (This would become unnecessary with the implementation of eye tracking.) It must be noted that with changing the eye separation value, the focus value automatically changes, which in turn automatically changes the field of view, so some zooming occurs. However, according to the first tests, this is tolerable.

## V. Headsets: problems and perspectives

Current headsets do not really have the potential for becoming daily wear yet. They suffer from a number of problems. These may deteriorate the user experience and might even cause discomfort and sickness. There are areas where improvement is possible.

### *A. Physical characteristics*

Physical characteristics such as size, weight, shape, the feel of the material, adjustability to heads of different shapes and sizes all influence whether a headset can be worn comfortably at length. It also matters how much it heats during use and how much ventilation it provides. Field of view can also be an issue.

Read more in [12] and

*https://www.jabil.com/blog/top-augmented-and-virtual-reality-challenges.html*

### *B. Resolution and graphical data communication throughput*

The resolution of the Oculus Rift used in our project is not really adequate, the control room looks a bit "pixelated". There are already better, affordable headsets available with a "somewhat" better resolution, the question is how much is enough. This also depends on the field of application: a technical application like ours does not demand as high a resolution as an artistic application using refined shapes and colors would.

Having higher resolution also means that considerably more graphical data has to be sent from the graphics card to the headset. The Oculus Rift CV1 is able to use the HDMI port of a PC, but headsets of higher resolution must use the Display port, because of the higher throughput demands (which are also affected by the refresh rate).

However, throughput demands can be reduced with a trick called foveated rendering: using eye tracking, you are able to tell where the user looks, and it is sufficient to display only that area in high resolution. while the surrounding area may be displayed in lower resolution. More info on foveated rendering: *https://www.cnet.com/tech/mobile/vr-games-can-look-amazing-with-this-game-changing-imaging-tech/* Varjo's solution optically combines two displays together: *https://varjo.com/blog/introducing-bionic-display-how-varjo-delivers-human-eye-resolution/*

It must be mentioned that throughput demands and the advances in wireless communication technologies (5G etc.) influence whether quality headsets must be tethered (connected to the computer with a cable) or could be wireless.

### C. Vergence-accommodation conflict

Two characteristics of the workings of the human eye, accommodation and vergence, have already been mentioned in the previous section. The development of these two biological mechanisms has taken millions of years. They operate in tune, and, unfortunately, current headsets cause a conflict between them. The essence of the problem is that the screens presenting the two images for the two eyes are at a fixed distance from the eyes, causing them to focus at this distance. However, a 3D object can be anywhere in the virtual space. In the case of near objects, the vergence mechanism, making the two eyeballs rotate around their vertical axes towards the object, comes into effect, while the focus still remains fixed. This mismatch may cause discomfort, nausea, "simulator sickness". There are attempts to resolve this problem (see [13, 14]), but a really good solution has failed to emerge so far.

### D. Drift

Sensors measuring rotation are not infinitely precise. Small errors accumulate and influence the measured values. Such error is called drift. E.g. in our project the *OpenHMD* package used does not perform any drift correction. When you put down the headset on a table but you leave the rotation detection on and rotate the view in accordance with the "measured" rotation, you will notice that the view will slowly rotate, even though the headset is completely motionless. Such drift has to be compensated, especially if the headset is used for more than a few minutes at a time. The SDK-s provided by the vendors usually – more or less – perform this compensation, but a good compensation is usually possible only when you are able to compare the measured values with values obtained from other tracking systems parallelly active. (For drift compensation for the Oculus Rift, see

*https://freecontent.manning.com/wp-content/uploads/2015/05/oculus-sdk-drift-correction-and-prediction.pdf*
*https://www.roadtovr.com/oculus-rift-sdk-updated-to-v0-2-1-initial-magnetometer-drift-correction-and-more/* .)

### E. Combining with sensors and providing stimuli for other senses

Most available headsets are already combined with at least rotational tracking sensors to track head rotation, but often also with positional tracking. Hand-held controllers and audio output (headphones) are also common. The commercial success of a headset may highly depend on how much its vendor can provide a solution as complete as required. In the future this may include haptic feedback as well. Eye tracking has already been mentioned above, it can be used not only for foveated rendering but also as a means of interaction with the virtual world.

## ACKNOWLEDGMENT

The author is a PhD student at the Faculty of Informatics at the University of Debrecen, Hungary, and is grateful to her supervisor Dr. Attila Gilányi for his encouragement and support.

## REFERENCES

[1] G. Házi, and J. Páles, "Virtual control room for the full-scope simulator of the Paks nuclear power plant," ["Virtuális vezénylő a paksi teljesléptékű szimulátorhoz," in Hungarian], Nukleon, December 2013. Available: *http://nuklearis.hu/sites/default/files/nukleon/6_4_146_Hazi.pdf*

[2] B. K. Szabó, "Interaction in an immersive virtual reality application." 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Naples, Italy, 2019, pp. 35–40.

[3] R. Hess, The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender, No Starch Press San Francisco, CA, USA, 2007

[4] J. M. Blain, The Complete Guide to Blender Graphics: Computer Modeling and Animation, 4th ed., A K Peters/CRC Press, 2017

[5] Z. Liu, Q. Zhao, L. Zhang, X. Zhang, J. Fan, Q. Wang, and P. Wu, "Quantitative Evaluation on the Effect of Experience Under Emergency Situations in NPP Main Control Room Based on Multimodal Data," Nuclear Technology, 207(4), 2021, pp. 575–581.

[6] J. D. Bergroth, H. M. Koskinen, and J. O. Laarni, "Use of immersive 3-D virtual reality environments in control room validations," Nuclear Technology, 202(2–3), 2018, pp. 278–289.

[7] J. Laarni, M. Liinasuo, S. Pakarinen, K. Lukander, T. Passi, V. Pitkänen and L. Salo, "Building cognitive readiness and resilience skills for situation assessment and diagnostic reasoning in a VR CR," International Conference on Human-Computer Interaction July 2020, pp. 77–84, Springer, Cham.

[8] J. Laarni, M. Liinasuo, S. Pakarinen, K. Lukander, and T. Passi, "Control Room Operators' Cognitive Strategies in Complex Troubleshooting," International Conference on Applied Human Factors and Ergonomics, July 2021, pp. 238–245, Springer, Cham.

[9] V. Pitkänen, "Utilization of virtual reality in Loviisa nuclear power plant," Master's Thesis, Lappeenranta-Lahti University of Technology, 2020

[10] F. Cutolo, and V. Ferrari, "The role of camera convergence in stereoscopic video see-through augmented reality displays," Int. J. Adv. Comput. Sci. Appl., vol. 9, no. 8, 2018, pp. 12–17.

[11] A. Z. Aktaş, and E. Orçun, "A survey of computer game development," The Journal of Defense Modeling and Simulation, 13(2), 2016, pp. 239–251.

[12] L. Kugler, "The state of virtual reality hardware," Communications of the ACM, 64(2), 2021, pp. 15–16.

[13] G. Kramida, "Resolving the Vergence-Accommodation Conflict in Head Mounted Displays," IEEE Transactions on Visualization and Computer Graphics, 22(7), 2016, pp. 1912–1931.

[14] R. Konrad, "What is the vergence-accommodation conflict and how do we fix it?," XRDS: Crossroads, The ACM Magazine for Students, 22(1), 2015, pp. 52–55.