Dariusz SAWICKI[1], Łukasz IZDEBSKI[2], Agnieszka WOLSKA[3], Mariusz WISEŁKA[3]

Warsaw University of Technology (1), JRS Software, Warsaw (2), Central Institute for Labour Protection – National Research Institute (3)
ORCID: 1. 0000-0003-3990-0121; 2. 0000-0002-1998-4041; 3. 0000-0003-3912-605X; 4. 0000-0002-7145-6457

doi:10.15199/48.2024.12.61

Multiscreen Graphic Driver for Semi-Cave Virtual Reality in Vulkan Environment

Streszczenie. *Cave Automatic Virtual Environment (CAVE) to przykład instalacji multimedialnej umożliwiającej postrzeganie rzeczywistości wirtualnej (VR) w jej najlepszej postaci. Celem artykułu jest przedstawienie specjalistycznej instalacji tego typu (SEMI-CAVE), w której opracowano autorskie oprogramowanie do wyświetlania informacji z wykorzystaniem środowiska Vulkan. Zwrócono uwagę na korzyści, problemy i dobre praktyki programistyczne. Eksperymenty i badania przeprowadzone w SEMI-CAVE potwierdziły zalety przyjętego rozwiązania. (**Sterownik graficzny dla rzeczywistości wirtualnej typu CAVE w środowisku Vulkan**).*

Abstract. *Cave Automatic Virtual Environment (CAVE) is an example of a multimedia installation that allows perceiving virtual reality (VR) in its best form. The aim of the article is to present a specialized installation of this type (SEMI-CAVE), in which authors' own display software based on Vulkan environment was developed. The benefits, problems and good programming practices were also highlighted. Experiments and conducted research in SEMI-CAVE laboratory have confirmed the advantages of the proposed solution.*

Słowa kluczowe: środowisko wirtualne, zanurzenie w wirtualną rzeczywistość, wirtualna rzeczywistość typu CAVE
Keywords virtual environment, immersive VR, CAVE VR

Introduction

The virtual reality (VR) has become one of the most important achievements of computer science in recent years. In many areas of science, the use of VR techniques not only speeds up the study, but also facilitates them or sometimes simply allows conducting the research.

The considered realization – SEMI-CAVE – is an example of multimedia laboratory where VR is applied to specialized research. It is one of the laboratories which were built in the scope of project Tech-Safe-Bio [1], which was realized in recent years in Central Institute for Labour Protection - National Research Institute (CIOP-PIB). The state-of-the-art laboratories built in the Tech-Safe-Bio project are adjusted to conduct interdisciplinary research focused on, widely understood, protection of health and safety of workers. The SEMI-CAVE laboratory allows, among other things, to conduct research on the impact of the physical environment on employees. In this regard, virtual reality techniques and computer simulations offers practically un-limited research possibilities.

OpenGL (Open Graphics Library) is a programming tool for creating 2D and 3D graphics. It was created in the 90s of the last century and was popularized thanks to the work of Silicon Graphics Inc. Currently, OpenGL is open source software developed by the Khronos Group and is considered as the de facto industry standard. Vulkan is a similar tool for creating 3D graphics, developed since 2016 by the same specialists from the Khronos Group. Vulkan is a low-overhead tool, it is considered as a "thin graphic API" which gives the same possibilities as OpenGL but using less resource. Initially, Vulkan was treated as the next generation of OpenGL for computer game programming. However, both tools are currently being developed and used in a wide range of different applications. The most important advantage of Vulkan is that it allows the programmer to work at a lower level of programming and gives much better – full control over the graphics creation process.

The use of VR for a wide range of research requires precise control of the process of building a virtual world and effective display of virtual information. The need for full control of the graphics creation process in a specific SEMI-CAVE environment resulted in the use of Vulkan API. It is an advanced and complicated solution, with a non-obvious way of solving many programming tasks. It is probably the first time this tool has been used in Cave installations – the authors are not aware of any publications describing such examples. Experiments and conducted research in SEMI-CAVE laboratory have confirmed the advantages of Vulkan API and the suitability of this software for specialized applications.

The main aim of the article is to present how the Vulkan API was used to create display software in the SEMI-CAVE environment. To make the most of the hardware capabilities, we have developed own – proprietary solution of the display software using shader language programming. In the article, we also tried to include practical comments and advice resulting from our experiences.

Virtual Reality in CAVE System – The Short Survey

CAVE (Cave Automated Virtual Environment) is an implementation of virtual reality (VR) in which the participant (recipient of VR experiences) is in a small room. On its walls (often also on the floor and ceiling) an image of reality is projected into which the participant is immersed. The number of publications on CAVE and VR is very large. On the basis of review articles [2-4], several basic categories of technical solutions for CAVE systems can be distinguished. The first prototype of a CAVE type solution was built in 1992 [5]. It had 5 translucent walls (including the ceiling and the floor) on which the image was projected from the outside (rear projection). This installation was made on the basis of a cube with a side of about 2.10 m – not giving the participant too much free space. However, the entire installation took up much more space due to the external projectors and the supporting structure [5]. The need for a very large space to implement CAVE virtual reality with rear projection is the main disadvantage of such a solution. In later projects, six walls were used most often, but different constructions are also known. Single wall installation in the image resolutions of the range from 1024x768 [6] to 6000x4096 pixels [7], two walls [8], four walls [7] and five walls [9]. The most advanced and the most expanded solution today includes projecting onto 15 side walls plus the ceiling and the floor [10]. There were also solutions using: the sphere on the surface of which the projection is made [11] or solutions connecting the sphere and the cube [12].

Apart from the rear projection, the front projection (inside the CAVE object) onto the walls of the laboratory is also used [13]. This makes it possible to obtain a much larger CAVE area with a comparable size of the entire installation.

CAVE2 is a VR implementation in which the walls of the VR space are covered with LCD monitors [14]. The disadvantage of such a solution is the physical gaps between the screens at the point of their connection. This does not allow for full immersion into VR [15,16].

The presentation method of graphical information is the main problem of all CAVE solutions. Because of known difficulties, the special attention is paid on image stitching [17]. The visual properties related to the position of the observer and the quality of image stitching, are factors that influence the quality of immersion into VR very strongly [18].

Additional problems may arise in CAVE installations: users may experience severe and long-lasting visual disturbances. Sometimes they may feel isolated or confused and even panicked [8].

Planned research – expectations related to virtual reality

The aim of building the SEMI-CAVE installation was to use virtual reality to create different visual environments in workplaces. It was assumed that it is possible to study the influence of various parameters of the created environment on the broadly understood psychophysiology of vision (e.g. eye fatigue, psychomotor performance, visual performance, well-being, emotions, and human alertness). The study of human reactions to various visual stimuli and lighting parameters is important both for the safety of people at work as well as for their health and well-being, which indirectly translates into better work efficiency. Additional supplementation of the virtual space with real elements (desks, chairs, work tools, etc.) makes it possible to blur the boundary between the real and the virtual world, and thus definitely improves the immersion into VR. On the other hand, the necessity to use ancillary equipment (such as an eye tracker or EEG recording system) used in alertness level or psychomotor testing requires additional available space. Another type of research that has been considered in a virtual environment is research on discomfort glare from LED sources. This phenomenon is most often the result of the appearance of a light source in the field of view, the luminance of which is much higher than the surroundings luminance [19]. The use of VR with real elements is very convenient for glare research, as it gives the opportunity to simulate different lighting and working conditions, as well as various methods of assessing the phenomenon [20].

Such widely understood research tasks require a relatively large area of the room. This would be difficult to do in a typical very small CAVE installation. When designing SEMI-CAVE, a solution was selected that supports a relatively large space. This allowed us to meet the expectations related to the research plans.

SEMI-CAVE – realization

The SEMI-CAVE virtual reality installation was implemented in a room with dimensions of 8.6m x 4.3m x 6m. The projection is realized by projecting the image directly onto 4 walls of the room (front projection). The images are projected by 6 projectors: two for longer walls and one for shorter walls (Figure 1). Optical systems of the projectors, made in the short throw technology works with the use of perspective correction – Lens Shift and Keystone. It allows shadowless projection, when approaching a distance of about 1m to the wall, people with a height of 1.7m. The obtained distance of the "shadowless" approach (1m) is a compromise resulting from the use of projectors at a certain price. On the other hand, one may wonder if this is a sufficient distance. In the SEMI-CAVE installation, such conditions of shadow-free operation allow users to have an area of over 2m wide and over 6m long. It is a relatively large space that meets the design expectations. Thus, it is a space that would not be possible to achieve in a traditional, small CAVE installation space. The images are generated in our computer by the set of three high performance PNY professional graphics cards with NVIDIA graphic processor unit (GPU). Each card supports two images (two projectors). The specific conditions of SEMI-CAVE installation dictated the need of self-developed software (including appropriate drivers). Only such software gives us full control over the developed whole solution.

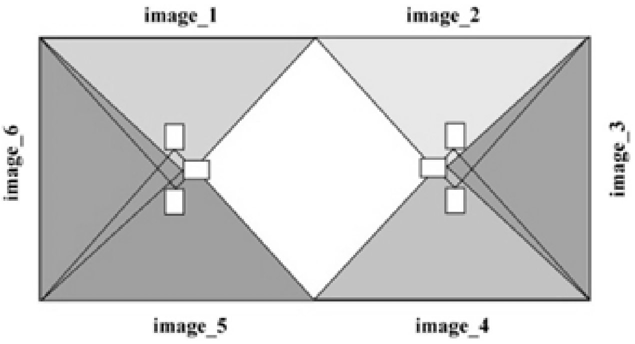

Fig. 1. The method of displaying images in the SEMI-CAVE installation (based on [21]). Connections of displayed images: on longer walls, images from adjacent projectors (images 1 and 2, images 4 and 5) overlap (50 pixels); in the corners of the room (pictures 2 and 3; 3 and 4; 5 and 6; 6 and 1) images are projected "to joint" (to the edge / image associated with the corner of the room).

The stitching problem in SEMI-CAVE virtual reality

The quality of the displayed image in CAVE installations is a key issue that determines the possibility of getting the best possible immersion into VR created inside the installation [22]. Particular attention is paid to the problems of stitching pictures [17].

The first important task is to correct the geometry so that the projected, adjacent images "fit together" and create one whole – one virtual world. Additionally, the problem is made worse by the passage of time – "aging" of the structure and micro-vibrations change (spoil) the display geometry. The geometry correction is, in fact, a properly selected transformation of a quadrilateral into a quadrilateral [23].

The second important task is to correct the color displayed by the projectors. The individual differences between the projectors used and their individual aging create small color changes. Unfortunately, such changes are registered by the sense of sight and have a negative impact on the quality of immersion into VR [24]. The measure of color mismatch is the difference in color coordinates [25].

Both corrections must be made while the information is displayed. This way effective correction of geometry and color for image stitching is a very important goal of the software we have developed for displaying information in the SEMI-CAVE environment.

OpenGL and VULKAN environment

The basic implementation problem in the SEMI-CAVE was the choice of the programming environment for software development. Although CAVE environments have been known for many years, the creators rarely present the software they have developed and used. Most often in publications we find only information about the spectacular possibilities of VR. Installation of SEMI-CAVE is a specific solution and the authors decided to develop their own display control, based on self-developed software. It was decided to use the Vulkan environment [26,27]. Vulkan API is a low-level, cross-platform application programming interface supporting the generation of 3D graphics. It is a tool developed by the Khronos Group. Vulkan API is mostly adopted (at the time of writing this article) in game industry, and well known in the game engines like Unreal Engine 4, Unity, CryEngine etc. Other industries slowly are adopting this API in their applications. However, many developers use documented experience from game programming in their applications, showing the great usefulness of Vulkan.

Vulkan API, in contrast to the previous generation of graphical APIs such as OpenGL, is intended and designed for more advanced and experienced graphics programmers. It provides, through its interfaces, more low-level access to the graphics card resources and the rendering pipeline. This can lead to a large number of potential pitfalls waiting for the programmer. Therefore, experience and strict adherence to the standard are very important. On the other hand, the experience of programmers from the Khronos Group meant that Vulkan API gives the possibility of convenient and effective programming at the application level, similar to OpenGL. It gives precise control over the communication of the 3D application with the window operating system (Windows). It allows, among other things, to gain direct control over the association of specific application windows with the display of information from selected graphics cards. This is particularly important in the case of multiscreen display implemented by several independent graphics cards – which is particularly important in the SEMI-CAVE installation.

Effective use of graphics card processors

Graphics cards used in the SEMI-CAVE installation have a very large application potential. However, their effective use is not an easy task. After analyzing the possibilities of programming these cards, it was decided that the entire image display system (display support and subsystems for geometry and color correction) would be fully implemented in shader language solution. It is worth emphasizing that this technique is very effectively supported by Vulkan API.

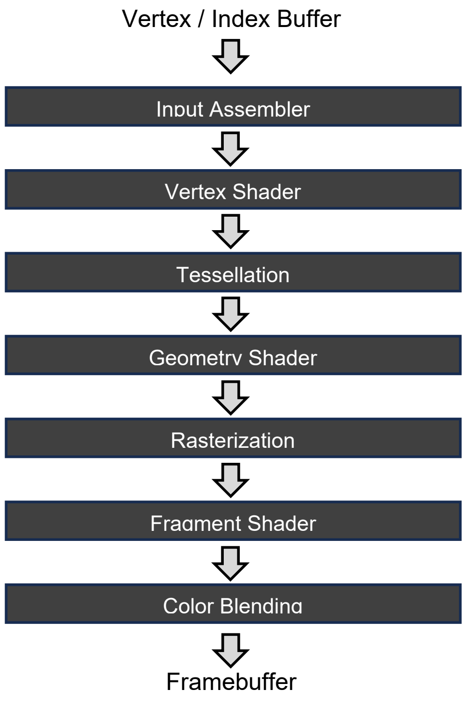

Fig. 2. Tasks of rendering pipeline described by modern shaders and implemented in hardware by GPU cores. On the base of https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics

Modern graphics cards (based on modern graphics processors) have a multi-core structure. The number of cores in the most powerful graphics cards exceeds 1000 (reaching even 10 000 for the latest products). Cores are highly optimized for generating 3D graphics and for general-purpose computing. All cores are clocked equally and are independent computing units. Performance studies show that the number of cores cannot be treated as a decisive parameter for the effectiveness of a graphics card (in a simple way), because efficiency is also strongly influenced by communication handling and memory performance. However, the greater the number of cores, the more parallel subprocesses can be run to handle graphics generation. The work of the cores is managed by the hardware task scheduler, whose task is to correctly divide the work into many processes and assign each of the processes to the appropriate GPU core.

Shader [28] is a relatively short program that is executed directly by the graphics card processor (and only by this processor). The shader is written in such a way as to use the hardware capabilities of the card, which is provided by the support of the parallel architecture of the processor cores, in the most effective way. A typical set of tasks in the rendering pipeline (supported by OpenGL and Vulkan) is shown in Figure 2. All operations have a hardware representation in the graphics card processor. On the other hand, because Vulkan is a continuation of OpenGL (by the same company), most of the procedures in them can be implemented interchangeably. This has been used effectively in the software for SEMI-CAVE.

It was assumed that the software would work in two modes.

- Working (standard) mode, when the images are displayed, and the normal use of a SEMI-CAVE installation is taking place. Both corrections must be made while the information is displayed.
- Editing mode, during which geometry and color can be corrected. In this mode, we can define the operating conditions of both corrections and see the consequences of their operation.

In the working mode, the application should be done in the most effective way, using the computer resources as little as possible.

Results – the shader driver developed for SEMI-CAVE installation

Vulkan API can offer comparable functionality to NVIDIA graphic cards with some additional logic to help with configuration multi GPU and multi-desktop environment. To achieve this, the XML configuration file was developed in which users enter data that describe the current system configuration (desktop and hardware mapping which can be found in NVIDIA Control Panel). This configuration file also contains information about the location of virtual cameras that are used for rendering SEMI-CAVE environment.

The graphical application for displaying information and also for stitching images in working mode has been implemented using GPU shaders. The stitching configuration is made using the XML configuration file and the final software has been prepared in the Visual Studio using the Vulkan environment [27,29].

To produce a smooth continuous transition of projected desktop images various techniques were used. All those techniques were using hardware features and for fixing no ideal position and orientation of projectors in-room images

The first step is the configuration of Vulkan’s projection of the virtual camera. As rendered images proportion and resolution are prepared too much projector’s resolution, so application is filling the front face of cull volume and model view projection matrix of the virtual camera is an identity matrix.

Desktop projection on walls was selected to receive the greatest common area. Subspace which was outside of the common area are organized by clipping planes. This mechanism realized by Vulkan API allows an application to provide information about additional clipping planes, against which rendered geometry will be clipped. Each user-specified clip plane (up to a minimum of 8 planes) can be turned on and off separately. For each of them, four plane equation coefficients can be provided using Vulkan API (Listing 1).

Listing 1.

```
gl_ClipDistance[0] = dot(pushConsts.cull_plane[0],gl_Position);
gl_ClipDistance[1] = dot(pushConsts.cull_plane[1],gl_Position);
gl_ClipDistance[2] = dot(pushConsts.cull_plane[2],gl_Position);
gl_ClipDistance[3] = dot(pushConsts.cull_plane[3],gl_Position);
```

Color correction is done on two adjacent areas, and the size of those left and right areas are defined by the user in the XML configuration file. The correction was implemented in a fragment shader. The rendered image was divided into the 3 vertical areas (Listing 2).

Listing 2.

```
vec3 mainColor = texture( texSampler, fragTexCoord ).xyz;
vec3 one = vec3(1.0,1.0,1.0);
float eqLineL = fragBlendLines.z;
float eqLineR = fragBlendLines.w;
float eqStepL = step(gl_FragCoord.x, eqLineL);
float eqStepR = step(gl_FragCoord.x, eqLineR);
float eqCenter = (1.0 - eqStepL)*(1.0 - eqStepR);
float smoothL = 1.0 - smoothstep(0.0, eqLineL, gl_FragCoord.x);
vec3 eqBlendL = one + smoothL*(fragColor.rgb - one);
float smoothR = 1.0 - smoothstep(0.0, eqLineR, gl_FragCoord.x);
vec3 eqBlendR = one + smoothR*(fragColor.rgb - one);
vec3 eqSumColor = vec3(eqCenter,eqCenter,eqCenter) +
                  (eqStepL*eqBlendL) + (eqStepR*eqBlendR);
vec3 finalColor = mainColor * eqSumColor;
```

Blending two adjacent and overlapped areas of projected desktop images was done also fragment shader. The rendered image was divided also into the 3 vertical areas (separate from color correction). Users can define the size of those areas in the XML configuration file. Blending can be turned on the left and right areas separately, and choose if linear, smooth interpolation is used to blend the current color of the rendered pixel with black color (Listing 3).

Listing 3.

```
float blendLineL = fragBlendLines.x;
float stepL = step(gl_FragCoord.x, blendLineL );
float blendLineR = fragBlendLines.y;
float stepR = step(gl_FragCoord.x, blendLineR );
float smoothstepL = smoothstep(0.0, blendLineL, gl_FragCoord.x);
float smoothStepR = smoothstep(0.0, blendLineR, gl_FragCoord.x);
float mSumBlend = (1.0-stepL)*(1.0-stepR) +
                  (stepL*smoothstepL)+(stepR*smoothStepR);
outColor = mFinalColor * mSumBlend;
```

On the basis of the procedure prepared for displaying information, an application carrying out appropriate tasks of correction has been prepared. The software has been implemented in Visual Studio using the OpenGL libraries [30], and works on the level of shader drivers [28]. Both modes (working mode and editing mode) use a common code associated with managing the display of windows in a Windows environment. All implemented operations within the appropriate shader have hardware representations in the graphics card processor. On the other hand, the division into appropriate procedures in OpenGL and Vulkan was designed to achieve the maximum performance of a given processor. Moreover, since Vulkan and OpenGL were designed by the same consortium, most of the procedures in them can be used interchangeably. This has been effectively applied in the SEMI-CAVE software.

In editing mode application is using OpenGL. This was made for faster iterations over developing stitching editor used in research. The same effects can be achieved with a Vulkan based editor, but time constraints have forced a well-known approach to be used.

According to the assumptions, the correction software meets the following requirements.

- Software support is implemented by means of a simple, intuitive interface enabling easy shape correction.
- The ability to visually check the correctness of the proposed geometry.
- In the correction of color and luminance, the software will support the measurements of proper parameters.
- The software will include a convenient and effective communication interface with the user.

We analyzed the image stitching problem and assumed that geometry correction would be performed first. Color correction will only be performed when all images are geometrically correct. Details of interface features of the prepared software for geometrical and luminance/color correction have been described in [21] and [31] respectively.

Vulkan is a next-generation open standard API that provides high-performance, cross-platform access to modern GPUs. Unlike, for example, OpenGL, it was designed with this in mind as the foundation for how modern graphics cards are built and how they are communicating with operating systems. Those guidelines provide a multi-threading-centric design to leverage modern multi-core CPUs and provide access to GPUs via multiple parallel command queues.

Communication between application and shaders are made by a feature which is available in Vulkan API called push constants. Push constants allow sending a small amount of data to any shader, in a very simple and effective way. Those data are stored in the Vulkan's Command Buffer itself.

An alternative approach will be using Vulkan's Uniform Buffers. To be able to access Uniform Buffer in the shader is accomplished through what is called in Vulkan API a descriptor. Usage of descriptors consists of three parts:

- Specify a descriptor layout during pipeline creation.
- Allocate a descriptor set from a descriptor pool.
- Bind the descriptor set during rendering.

The next step is tasks that involve Uniform Buffer itself, which can be described as:

- Creation of Uniform Buffer in specific memory type (depend on the size of data and usage).
- Copy data to the buffer memory.
- Depend on memory type explicit usage of memory barrier primitives.

Fortunately, data needed for control culling color correction and blending viewports used in Sami-Cave application is small enough that fit in Push Constant data limits (minimum supported limit is 128 bytes – defined by the standard [29]). This approach is easier to implement and does not result in a decrease in real-time performance.

The shader's code and communication are designed to be as thin as possible, which means that the overhead introduced by them on the CPU and GPU was as low as possible. Consequently, they could be used as an almost invisible intermediate layer between the hardware and future applications built on top of this approach.

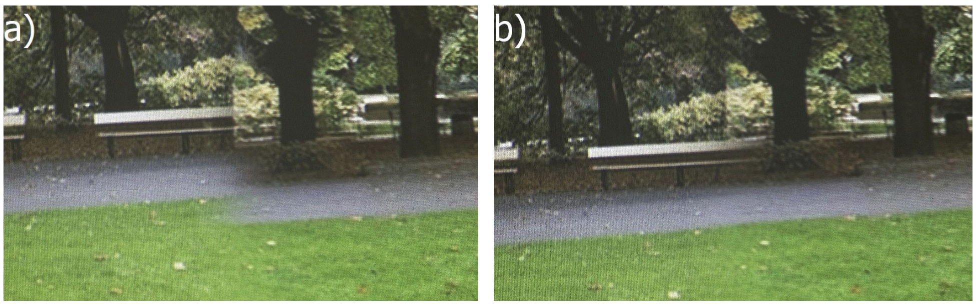

Fig. 3. The consequences of stitching with wrong geometry are visible on the example images of the Warsaw Saski Garden displayed in SEMI-CAVE. a) enlarger detail before correction, b) the same detail after correction of geometry.

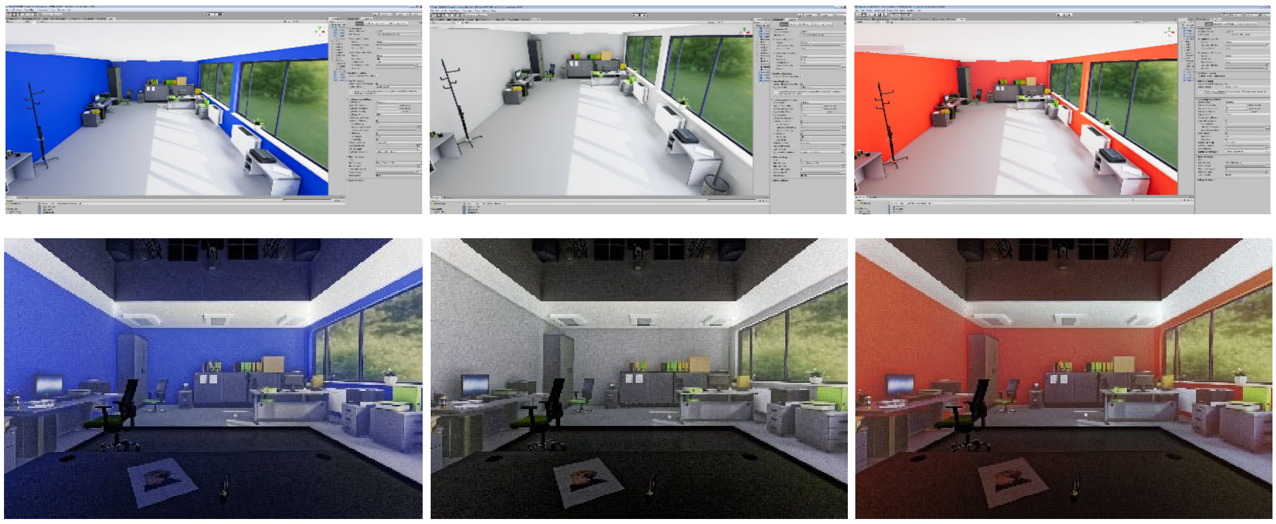

Fig. 4. Research scenarios for three colors of the environment: red, white and blue; upper row: computer visualization (graphics software), bottom row: SEMI-CAVE reconstruction (photos from realization).

Tests of the developed software

Initial tests for correct operation were carried out at the stage of application programming in the Visual Studio environment. Then, using the prepared images, the correctness was checked in the target SEMI-CAVE environment. Testing graphical applications has the advantage that potential errors are visible in the form of incorrect display of information. Final tests were performed on the target image sets in the SEMI-CAVE environment. Figure 3 shows an example of stitching using geometry correction. The correct display confirms the correctness of the geometry correction. It is worth emphasizing that on the one hand, testing the display in the SEMI-CAVE environment is convenient and relatively simple – "you can see the errors of the software". On the other hand, incorrectly displayed even small details (Figure 3.) significantly worsen the quality of the entire image displayed and make immersion in VR impossible.

The best test of the correct operation of the entire display software (i.e. a set of drivers, displaying and stitching images, correction of geometry and luminance / color) was to conduct real research in the SEMI-CAVE environment.

The first project was carried out to develop a method allowing the SEMI-CAVE installation to simulate selected working environments in order to study their impact on the psychophysiology of vision and the well-being of employees. In the experiments the lighting properties were examined in SEMI-CAVE environment. The pilot studies were carried out on a group of 46 people aged 19-29. This type of research, while ensuring a satisfactory immersion in a virtual environment, and in the same time while using additional measuring equipment (e.g. EEG) is only possible with the use of a SEMI-CAVE virtual environment.

The wide range of studies conducted requires a description in an independent article. The publication of the results of the conducted research is under preparation.

The 3D model of the office room was designed in Unity (Figure 4.). To increase the reality of the perceived impressions, the details of the virtual environment were taken care of – small pieces of equipment, rubbish on the floor, etc. The same real and virtual elements (e.g. chairs) were also added using the available space.

Additionally, after the completion of the research, discussions were held with all participants about immersion in virtual reality. Everyone rated the environment very highly. They paid particular attention to the use of a combination of real and virtual elements greatly enhancing the virtual impressions.

Benefits of the VULKAN environment application

Usage Vulkan API in graphics applications has very important and useful features. One of Vulkan's biggest advantages is that it supports a wide range of platforms. It means that application development can be easily ported to different computer platforms. Additionally, and this is also very important, the efficient operation of the Vulkan API can be implemented on a wide set of GPUs.

One word can best describe Vulkan API and this word is “explicit”. Developers have more explicit control over memory management and synchronization methods. Also integration with Platform Window System Interface is well designed and more explicit compared to OpenGL API. Modern API concepts like e.g. Command Buffers not only are based on construction of modern GPU but also:

- It reduced the complexity of Vulkan driver – this corresponds to minimalizing its overhead.
- It allows usage of multi-core CPUs in recording Command Buffers.
- It allows for better control and scaling multi GPU systems.

In Vulkan API are introduced a lot of mechanisms that let developers work as close to the hardware as the platform’s operating system allows. For example, Command Buffers are closely related to how the GPU hardware pipeline is implemented. Memory management (by introducing Memory Heaps and Memory Types) can be effectively implemented in a simple way by the programmer. Only he really know how his application will work and it is he who should can choose the best way how to organize memory allocation and deallocation.

VULKAN in an advanced graphical environment – some practical aspects

Because Vulkan API design roots are different than OpenGL (or other known techniques), programming graphical application using this API, requires more attention from the programmer. Very simple "Hello World" style graphical application consists of approximately 1500 lines of code. This is because Vulkan API introduces new concepts like Physical Device and Logical Device, Queues and Command Buffers, Synchronization Primitives, and last but not least resources that can be useful in memory allocation – Memory Types and Memory Heaps. And to be able to write graphics application developer need to learn all of these concepts and properly configure them. Vulkan API is designed around the idea of minimal driver overhead and one of the manifestations of that goal is that there is very limited error checking in the API by default. Working with Vulkan API drivers does not require runtime error checking, and developers must follow what is defined in the standard and available in the official examples in the repository. Therefore, it is recommended to use and turn on the application during the work and application Vulkan’s validation layers. Those layers (and not only layers library are included in LunarG’s Vulkan SDK [32]) are crucial in process of developing any Vulkan-based applications. It can help and make work easier when dealing with such low level and complex graphics API. It is worth mentioning that the popularity of the new API makes the availability of best practices and tutorials created by other developers are easily available on the web [33-35].

During development, it is good practice to register a debug callback by using functions that can be found in extension VK_EXT_debug_utils. This callback will receive driver’s calls which will contain various non-performance critical validation checks which it might perform. Also in this extension is available functionality that can attach debug with user names to various resources. Those names then are available in a callback and provide information about resources in e.g. debuggers’ tools like NVIDIA® Nsight™ Graphics [36] or RenderDoc [37].

Vulkan based application debuggers can help with developing Vulkan-based application by "capture" rendered frames and listed all recorded command buffer (with all its content like pipelines, descriptor sets, and draw commands) by showing them in a user-friendly way.

Another help to developers can be found in extension VK_EXT_debug_marker (or newer VK_EXT_debug_utils) and it is a functionality that can "annotate" your rendering regions and in this way improving the debugging and profiling experience when using debugging tools.

Something that is new for programmers (not only graphics programmers) is manual memory management which is deeply rooted in Vulkan API. In short words Vulkan API requires a lot of boilerplate code. There is an additional level of indirection (VkDeviceMemory is allocated separately from creating VkBuffer/VkImage) and they must be bound together. The application must also query the OS graphics driver for checking supported Memory Heaps and Memory Types, and recommended practice is to allocate bigger chunks of memory and assign parts to particular resources because there is a limit of allocation which the application can do in one run. To help with memory management best solution is using Vulkan® Memory Allocator [38]. This library is simple and easy to integrate API which is open source with MIT license and very popular with Vulkan’s developers.

Another thing which developer must learn is the proper usage of synchronization primitives (for not only performance), but this knowledge is shared with concurrent CPU programming knowledge. The basic recommendation is to minimize the use of barriers (a barrier may cause a GPU pipeline flush), make sure to always use the minimum set of resource usage flags, and use VK_IMAGE_LAYOUT_UNDEFINED when the previous content of the image is not needed.

Conclusions

In the paper we presented software solution for displaying information in specific SEMI-CAVE installation of VR. We decided to develop our own display control, based on self-developed software. The software has been built using Vulkan API. Vulkan API is widely used in the production of computer games; we applied Vulkan in VR for the research application.

We built our own display system using low level programming. The new display system supporting the used graphics cards has been developed. Our correction software provides the control of the geometry at the single pixel level, and also control over color and luminance of stitched parts. The conducted experiments and real research in the laboratory confirmed the correctness of the developed software as well as the correctness of the proposed solution of the SEMI-CAVE installation project. The participants of the study confirmed the correctness of immersion into VR.

Independent analyses show the advantages of Vulkan – low-level programming possibilities, full control over the entire image generation process and the high efficiency of graphics service [39]. Our experiments confirmed these advantages also in SEMI-CAVE – a specific and specialized CAVE environment.

On the other hand it is worth emphasizing that NVIDIA is prefers to support expensive graphics cards with specific, closed software, that is not more efficient but are compatible with the corresponding software standard. Closed and non-developmental solution for the programmer. We had to solve this problem – a special, fully controlled, shader driver has been developed, that allows displaying and synchronizing images in SEMI-CAVE environment. We have accomplished this through Vulkan API software and its properties.

ACKNOWLEDGEMENTS

This paper has been based on the results of a research task carried out within the scope of the fourth stage of the National Programme "Improvement of safety and working conditions" partly supported in 2017–2019 within the scope *of research and development --- by the Ministry of Science and Higher Education / National Centre for Research and Development. The Central Institute for Labour Protection -- National Research Institute is the Programme's main coordinator. Project of SEMI-CAVE is still developed.*

***Authors**: Prof. Dariusz J. Sawicki PhD. DSc., Warsaw University of Technology, Institute of Theory of Electrical Engineering, Measurement and Information Systems, Koszykowa 75, 00-662 Warsaw Poland, E-mail: dariusz.sawicki@pw.edu.pl. Łukasz Izdebski PhD. Eng., JRS Software Warsaw Poland, E-mail: lukasz.izdebski.jrs@gmail.com. Agnieszka Wolska PhD. DSc, prof. CIOP-PIB, Mariusz Wisełka MSc. Eng. Central Institute for Labour Protection - National Research Institute, Czerniakowska 16, 00-701 Warsaw Poland, E-mails: agwol@ciop.pl, marwi@ciop.pl.*

REFERENCES

[1] TECH-SAFE-BIO - The Centre for Research and Development on Work Processes and Safety Engineering, 2015. Available online: http://www.ciop.pl/CIOPPortalWAR/appmanager/ciop/en?_nfpb=true&_pageLabel=P33200114301448620711504.

[2] Kim M.J., Wang X., Love P.E.D., Li H., Kang S.C., Virtual reality for the built environment: a critical review of recent advances. *Journal of Information Technology in Construction*. 2013, 18, 279-305.

[3] Zhou N.N., Deng Y.L., Virtual reality: A state-of-the-art survey. *International Journal of Automation and Computing*. 2009, 6(4), 319-325.

[4] Muhanna M.A., Virtual reality and the CAVE: Taxonomy, interaction challenges and research directions. Journal of King Saud University – Computer and Information Sciences. 2015, 27(3), 344-361.

[5] Cruz-Neira C., Sandin D.J., DeFanti T.A., Kenyon R., Hart J.C., The CAVE: Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, 1992, 35(6), 64-72.

[6] Juarez A., Schonenberg B., Bartneck C., Implementing a Low-Cost CAVE System Using the CryEngine2. *Entertainment Computing*. 2010, 1(3-4), 157-164.

[7] CAVE Automatic Virtual Environment. Available online: http://www.visbox.com/products/cave/.

[8] Nichols S., Petel H., Health and safety implications of virtual reality: a review of empirical evidence. *Applied Ergonomics*. 2002, 33(3), 251-271.

[9] Sony 4K visualisation projectors drive Renault's virtual 'cave'. Available online: https://assets.pro.sony.eu/Web/ngp/pdf/sony-4k-visualisation-projectors-drive-renaults-virtual-cave.pdf.

[10] DeFanti T.A., Dawe G., Sandin D.J., Schulze J.P., Otto P., Girado J., Kuester F., Smarr L., Rao R., The StarCAVE, a third-generation CAVE and virtual reality OptIPortal. *Future Generation Computer Systems*. 2009, 25(2), 169-178.

[11] Fernandes K., Raja V., Eyre J. Cybersphere: The fully immersive spherical projection system. *Communications of the ACM*. 2003, 46(9), 141-146.

[12] Mazikowski A., Lebiedź J., Image Projection in Immersive 3D Visualization Laboratory. *Proc. of 18th International Con-ference on Knowledge-Based and Intelligent Information & Engineering Systems - KES2014. Procedia Computer Science* 2014, 35, 842-850.

[13] Jacobson J., Lewis M., Game engine virtual reality with CaveUT. *Computer.* 2005, 38(4), 79-82.

[14] Febretti A., Nishimoto A., Thigpen T., Talandis J., Long L., Pirtle J.D., Peterka T., Verlo A., Brown M., Plepys D., Sandin D., Renambot L., Johnson A., Leigh J., CAVE2: a hybrid reality environment for immersive simulation and infor-mation analysis. *Proc. SPIE 8649, The Engineering Reality of Virtual Reality*, 2013, 864903.

[15] Leigh J., Johnson A., Renambot L., DeFanti T., Brown M., Jeong B., Jagodic R., Krumbholz C., Svistula D., Hur H., Kooima R., Peterka T., Ge J., Falk C., Emerging from the CAVE: Collaboration in Ultra High Resolution Environments. *Proc. of First International Symposium on Universal Communication*, Kyoto, Japan, 2007, June 14-15.

[16] Krumbholz C., Leigh J., Johnson A., Renambot L., Kooima R., Lambda Table: High Resolution Tiled Display Table for Interacting with Large Visualizations. *Proc. of Workshop for Advanced Collaborative Environments (WACE),*. Redmond, Washington, USA 2005.

[17] Sajadi B., Majumder A., Auto-calibration of multi-projector CAVE-like immersive environments. *IEEE Trans. Visual Comput. Graphics*. 2011, 18(3), 381-393.

[18] CAVE VR course 2014. Available online: http://moodle.epfl.ch/pluginfile.php/1522455/mod_resource/content/7/Th_RB_NW_S3_CAVE_2017.pdf.

[19] Wolska A., Glare as a specific factor in the working environment. *Przegląd Elektrotechniczny*. 2013, 89(1), 142-144.

[20] Sawicki D., Wolska A., The unified semantic glare scale for GR and UGR indexes. *Proc. of the 2016 IEEE Lighting Conference of the Visegrad Countries (Lumen V4)*. 13-16.09.2016 Karpacz. DOI: 10.1109/LUMENV.2016.7745536

[21] Sawicki, D., Izdebski Ł., Wolska A., Wisełka M., Geometrical Picture Integration in Semi-Cave Virtual Reality. *Proc. of the International Conference on Computer-Human Interaction Research and Applications (CHIRA 2018)*. Seville, Spain, 19-21 September, 2018, 100-107.

[22] Slater M., A Note on Presence Terminology, 2003, Available online: https://www.researchgate.net/publication/242608507_A_Note_on_Presence_Terminology.

[23] Augustynowicz M., Sawicki D., Reconstruction of the relative coordinates of image using projective geometry. *Przegląd Elektrotechniczny*. 2016, 92(1), 208-211. doi:10.15199/48.2016.01.49

[24] Sawicki D., Wolska A., Wisełka M., Ordysinski Sz., Easing Function as a Tool of Color Correction for Display Stitching in Virtual Reality. *Proc. of the 20th International Conference on Image Analysis and Processing (ICIAP 2019). Lecture Notes in Computer Science*, 2019, 11752, 549-559. Springer. DOI: 10.1007/978-3-030-30645-8_50

[25] Mokrzycki W.S., Tatol M., Color difference ΔE : a survey. *Machine Graphics and Vision*, 2011, 20(4), 383-411.

[26] Khronos Vulkan Registry, Available online: https://www.khronos.org/registry/vulkan.

[27] Vulkan Tutorial, Available online: https://vulkan-tutorial.com/Introduction.

[28] Bailey, M., Cunningham S., *Graphics Shaders: Theory and Practice, Second Edition*. A K Peters/CRC Press 2011.

[29] Sellers G., Kessenich J., *Vulkan Programming Guide: The Official Guide to Learning Vulkan*. Addison-Wesley 2016.

[30] Sellers G., Wright R.S. Jr., Haemel N., *OpenGL Superbible: Comprehensive Tutorial and Reference (7th Edition)*. Addison-Wesley Professional 2015.

[31] Sawicki D., Izdebski Ł., Wolska A., Wisełka M., Luminance and color correction for display stitching in Semi-Cave virtual reality. *Proc. of the International Conference on Computer-Human Interaction Research and Applications (CHIRA 2019)*. Vienna, Austria, 20-21 September, 2019, 137-144.

[32] LunarG, Available online: https://www.lunarg.com/vulkan-sdk/.

[33] SaschaWillems / Vulkan. Available online: https://github.com/SaschaWillems/Vulkan .

[34] Intel. API without Secrets: Introduction to Vulkan* Part 1: The Beginning. Available online: https://software.intel.com/content/www/us/en/develop/articles/api-without-secrets-introduction-to-vulkan-part-1.html.

[35] Vulkan API samples, Available online: https://github.com/KhronosGroup/Vulkan-Samples/tree/master/samples#api-samples.

[36] NVIDIA Nsight Graphics, Available online: https://developer.nvidia.com/nsight-graphics.

[37] RenderDoc, Available online: https://renderdoc.org.

[38] Vulkan® Memory Allocator, Available online: https://gpuopen.com/vulkan-memory-allocator/.

[39] Lujan M., Baum M., Chen D., Zong Z., Evaluating the Performance and Energy Efficiency of OpenGL and Vulkan on a Graphics Rendering Server. *Proc. of International Conference on Computing, Networking and Communications (ICNC 2019)*. Vol.1, pp. 777-781.