ISSN: 2633-4828 Vol. 5, No. 3, September 2023

International Journal of Applied Engineering and Technology

A Graphics Device Driver Implementation Compliant with OpenGL SC 2.0.1 Standard Specification

Nakhoon Baek

School of Computer Science and Engineering, Kyungpook National University, Republic of Korea

Graduate School of Data Science, Kyungpook National University, Republic of Korea

Email: nbaek@knu.ac.kr

Date of Submission: 19th August 2023 Revised: 29th August 2023 Accepted: 5th September 2023

How to Cite: **Baek, N. (2023), A graphics device driver implementation compliant with OpenGL SC 2.0.1 standard specification, International Journal of Applied Engineering and Technology 5(3), pp. 143-149.**

Abstract **- Various 3D graphics outputs are increasingly required in the field of safety-critical applications including military, avionics, aerospace, and medical applications. In this paper, we present a graphics device driver implementation of OpenGL SC (Open Graphics Library – Safety Critical) 2.0.1, a 3D graphics standard specification that reflects the needs of the safety-critical field. OpenGL SC 2.0.1 was newly announced in 2019, and only a few limited number of implementation cases have been reported so far. In this paper, to increase its own portability, we designed to only use some functions of DRM (direct rendering manager) in a general Linux environment, while various test results were confirmed targeting specific GPUs, in our case, Intel HD630 GPU. The implementation results can be used to implement 3D graphics terminals in the military and aviation fields.**

Index Terms **- Graphics Device Driver, OpenGL SC 2.0.1, Safety-Critical Implementation.**

INTRODUCTION

Recently, almost all computers, tablets, smartphones, and embedded devices provide 3D graphics output as one of its fundamental features. Among the libraries for 3D graphics output, the *OpenGL* library [1,2,3,4,5,6] is currently the most widely used. They are used as standard 3D graphics output libraries in supercomputers, workstations, desktops, tablets, smartphones, as well as many embedded boards.

More precisely, with respect to the target devices, the OpenGL standard specifications can be classified into three different categories:

- **OpenGL** for general computers including desktops, workstations, and mainframes,
- **OpenGL ES** (embedded system) [7,8,9,10,11], specialized for mobile phones, tablets, and embedded devices, and
- **OpenGL SC** (Safety-Critical) [12,13,14,15], specialized for military and avionics devices.

Actually, all these standards are related to each other, as shown in Figure 1. Basically, the original OpenGL standards are developed first, and then, OpenGL ES standards are derived from them. As the next and the most improved step, the OpenGL SC standards are derived from OpenGL ES standards. According to these development processes, the target markets become more specific and narrower, even though the applied technologies become more refined.

Among them, our focus is the OpenGL SC standards, which are consistently used for safety-critical applications, including military, avionics, medical, and automobile-related applications. It was revised to the 1.0.1 standard [13] in 2009, after the original 1.0 standard [12] was announced in 2005, for enabling 3D graphics output in the military and avionics fields.

However, these OpenGL SC 1.0 and 1.1 standards were designed with the fixed function pipeline techniques, as shown in Figure 2, and thus, they had explicit limitations in that they could not accommodate the recent programmable graphics pipeline features.

In 2016, the Open GL~SC 2.0 standard [14] with the new programmable graphics pipeline features was announced. Based on the existing OpenGL ES 2.0 standard [9], this standard was refined through removing redundant elements which could cause non-Safety-Critical situations and adding several advanced extensions. The shader language, which is essential for the programmable pipeline, has also changed [16,17,18].

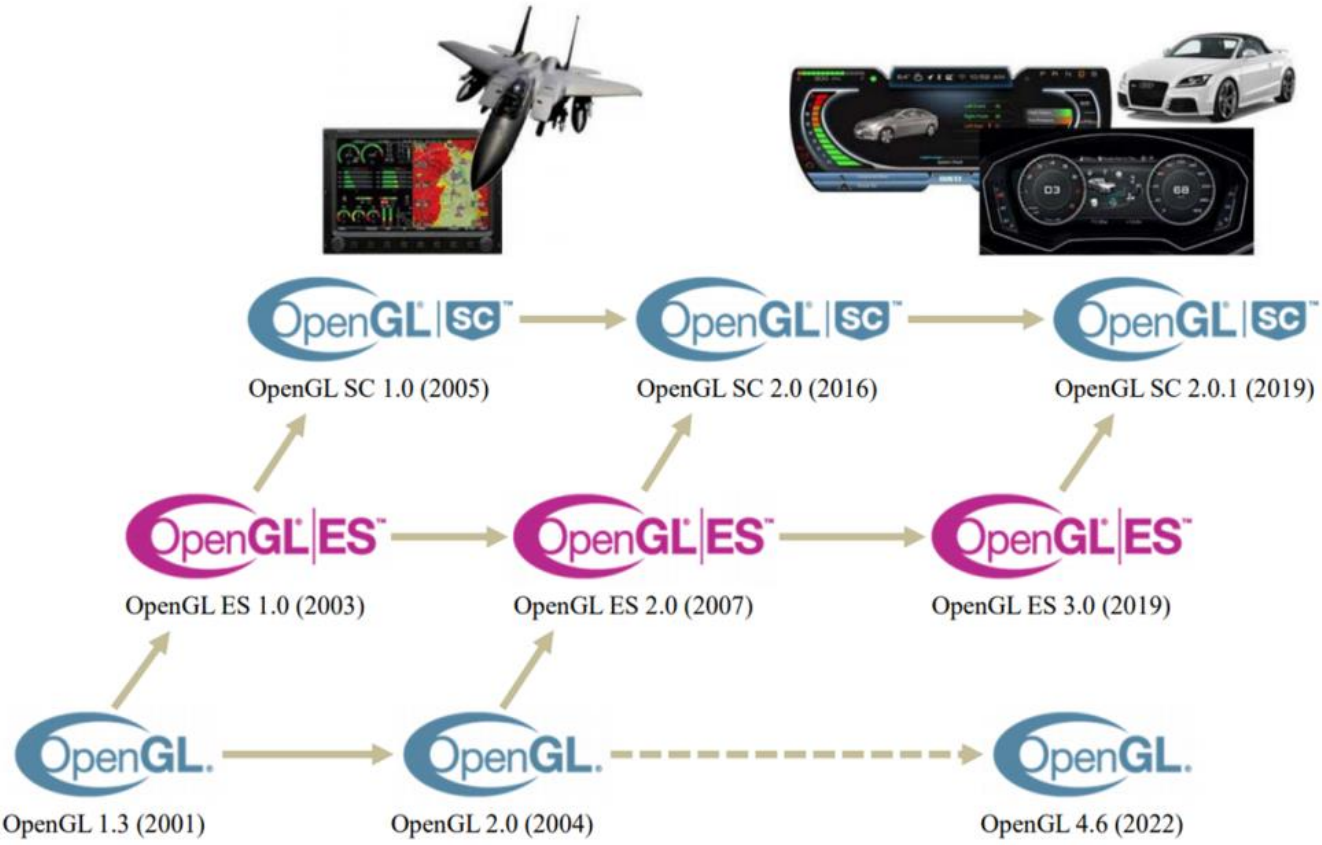

Figure 1. OpenGL SC was derived from OpenGL ES and OpenGL

Currently, the OpenGL SC 2.0 standard is replacing the existing 1.0 standard, mainly in the military and avionics application markets. The *Khronos Group*, which supervises the whole OpenGL standards, provides certification through the OpenGL SC 2.0 conformance test. There are a number of products that are currently undergoing the certification process, so products that have obtained certification are likely to come out soon.

For safety-critical products especially in the military and avionics markets, the OpenGL SC can be the most appropriate solution for their 3D graphics output. We already developed and commercialized OpenGL SC 1.0 and also basic components of OpenGL~SC 2.0 features [19,20,21].

Additionally, one more important thing to the OpenGL SC 2.0 is actually the introduction of the new modified OpenGL SC specification of version 2.0.1, in 2019 [15]. In this paper, we present a graphical device driver for the OpenGL SC 2.0.1 standard specification. Our implementation is designed to be used in Linux operating systems. As its actual target hardware, Intel HD530 3D graphics chip was selected, and it is directly driven using Linux *DRM* (direct rendering manager) features. Design steps and implementation details will be followed.

DESIGN AND IMPLEMENTATION

The 3D graphics standard of OpenGL SC 2.0 absolutely requires the use of the programmable graphics pipeline, as shown in Figure 3. Since the release of OpenGL 2.1 in 2006, this method has rapidly grown into one of the most important de facto industrial standards for 3D graphics.

According to the development of brand-new computer graphics hardware devices, the role of the graphics card (or most equivalently, GPU) continuously became more important. From the application developer's point of view, it is more efficient to access, control, and use these GPUs, directly. Therefore, in the case of OpenGL, the OpenGL Shader Language (GLSL) is provided as a new programming language, which directly controls the GPUs [2]. More precisely, modern GPUs provide all of the shader language hardware, its middleware environment, and even shader language compilers. These full features have already been applied since OpenGL 2.1 [3,4,6], OpenGL ES 2.0 [11,18], and OpenGL SC 2.0 [15].

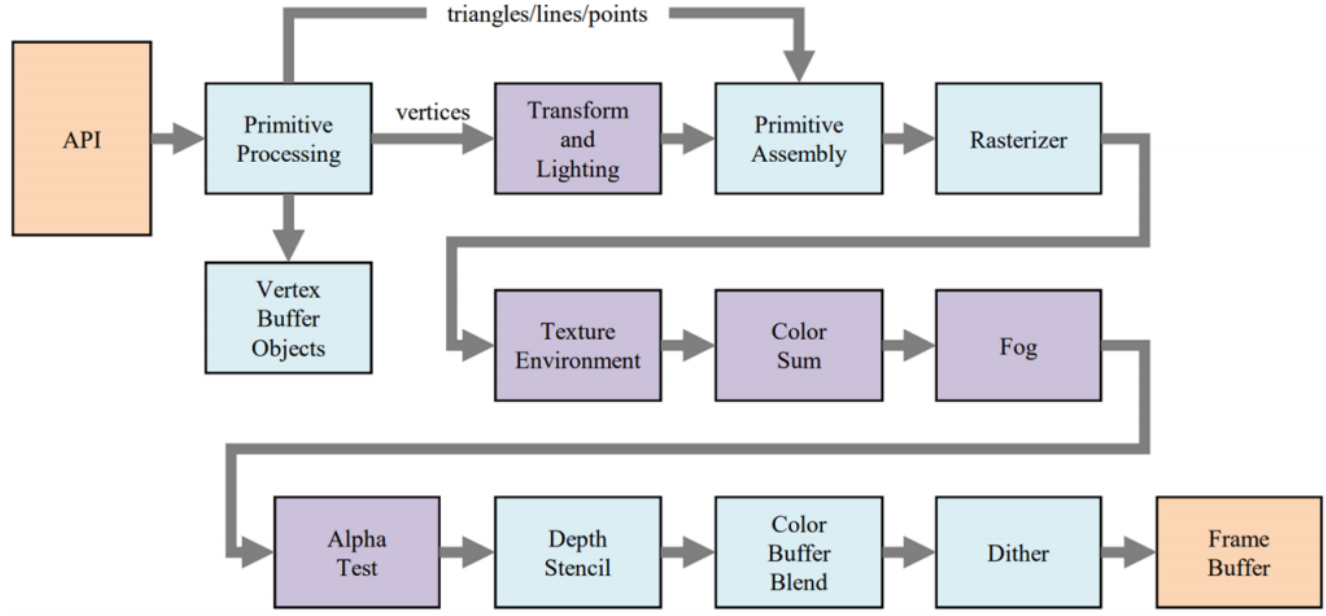

Figure 2. The fixed-function graphics pipeline used in OpenGL SC 1.0.

In the case of OpenGL SC, the latest version of OpenGL SC 2.0.1 [15] adds an important feature, off-line shader compiling. Previous shader compilers are designed to perform the on-line compiling, which means on-the-fly compiling of shader source codes, during its OpenGL application program execution. In this case, we should endure the time delays and security issues, due to the on-line compilation process. Through adding the off-line compiling features, most of these problems are solved, using pre-compiled execution files. This off-line compiling feature was added relatively recently, and only available in the latest version of OpenGL SC 2.0.1. In this paper, we have added this feature to accomplish our OpenGL SC 2.0.1 compliant implementation.

The entire pipeline was carried out by referring to the OpenGL ES 2.0 implementation [22], which have been developed by our team, and adding additional functions required by the OpenGL SC 2.0.1 standard specification. Technically, the most challenging part was providing an offline compiler for the OpenGL SC shader language and integrating it into the system.

For this purpose, in this paper, as shown in Figure 4 and 5, a separate off-line compiler file format was defined and modified to enable information exchange between the shader compiler and the OpenGL SC 2.0.1 graphics device driver.

In order to provide the off-line compilation features, we fully use the existing on-line shader compiler. In other words, after compiling with the existing shader compiler, the results are intercepted and stored as independent binary images. Technically, in these binary images, it is also necessary to store additional information, such as which variable a particular register is connected to.

This conceptual design was implemented in our previous paper [23], which presented our practical file formats, and showed that it worked well. However, in the previous paper, it had a limitation in that it was implemented as a prototype using the *Mesa* graphics system [24]. We improved this and succeeded in a step-by-step development, and finally independently separating them after sufficient verifications. Finally, it was newly designed as an independent module, targeting the Intel HD 530 chip, which is a specific graphics card model in the Linux system, and completed in the form of an independent device driver.

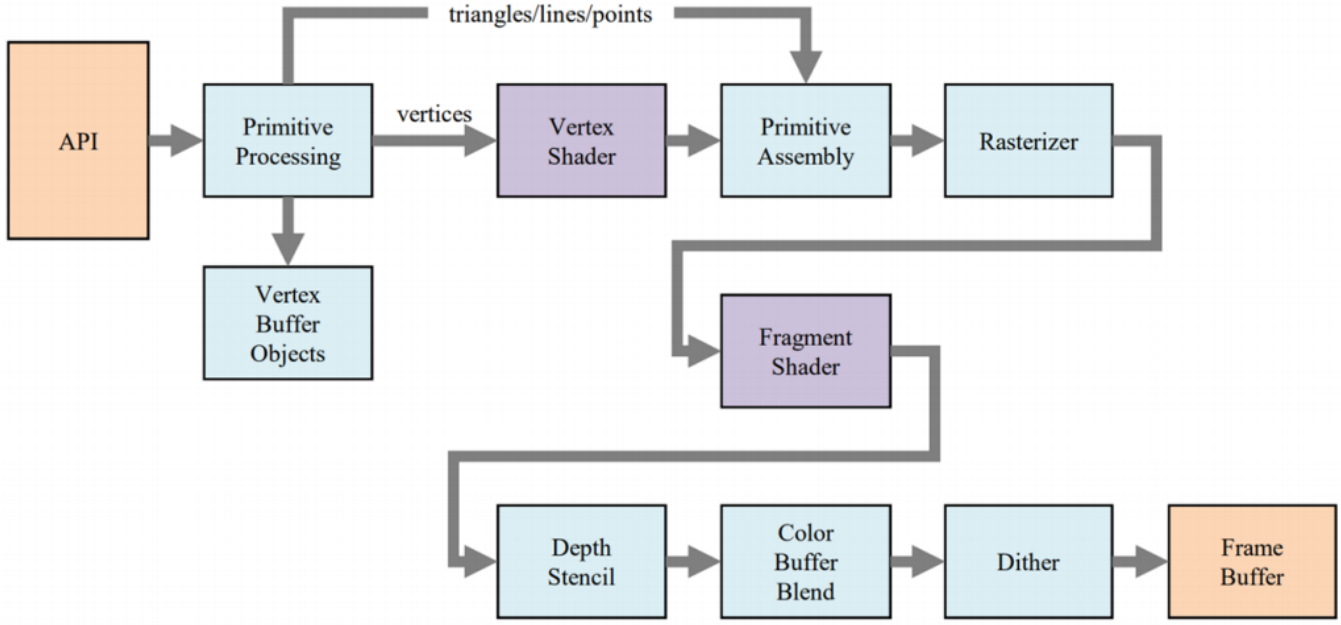

FIGURE 3. THE PROGRAMMABLE GRAPHICS PIPELINE USED IN OPENGL SC 2.0.1.

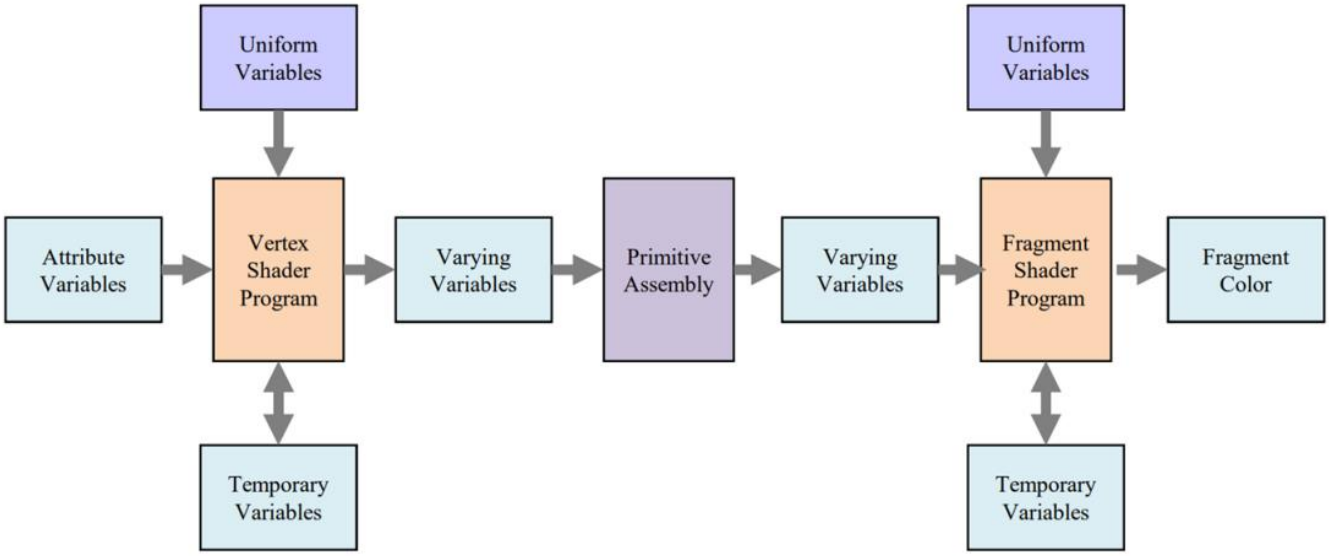

FIGURE 4. OUR DESIGN VIEW OF OPENGL SC 2.0.1 SHADER PROGRAM PROCESSING.

The most problematic point in the development of 3D graphics systems is that current graphics systems are working on the top of a 2D window system.

In embedded devices, it can be a major limitation that a large 2D graphics window system such as the *X window system* must be implemented first to provide 3D graphics functions.

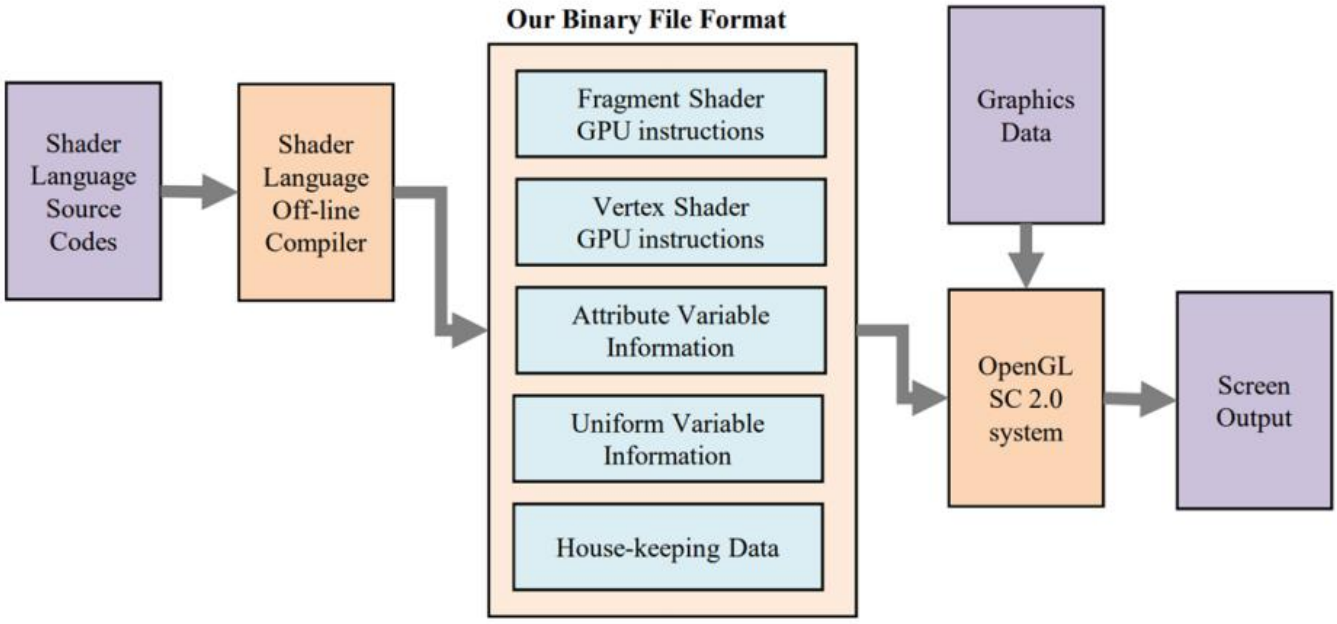

Figure 5. The overall architecture of our off-line shader compiler.

Our system can provide 3D graphics OpenGL SC functions directly without a window system by using DRM module [25]. It uses a DRM module built into the Linux kernel or provided as a separate library to switch to graphics mode and provides direct connections from the GPU chips to the screen output. The current system provides all API function interfaces of OpenGL SC 2.0.1, and the function modules operate on Linux-based devices that provide DRM (direct rendering manager) modules. Intel HD 530 chip was selected as the target graphics chip, and the GPU instructions of the corresponding chip were directly controlled.

We chose the Linux operating system for the convenience of development. In the Linux systems, kernel-level modules control the 3D graphics cards and GPUs. Since current 3D graphics cards are quite complex, it is difficult to control them with a single module. The core features include *direct rendering manager* (DRM) [25,26], *kernel mode setting* (KMS) [27], *graphics execution manager* (GEM) [28], and others.

First, the DRM module was introduced to directly access the frame-buffer memory. In the case of the classical 2D graphics systems, this frame-buffer access feature was sufficient to implement more complex graphics functions. However, currently this method alone cannot create a 3D graphics system, while this function is still needed.

Next, if you use DRM, KMS, GEM, and others, graphics output is possible even without the support of the underlying window systems.

In the case of OpenGL, it was confirmed that even 3D output is possible without a window system. The possibility of graphics output using actual DRM has been presented in the form of a prototype in our previous paper [23]. Finally, we were able to complete the graphic device driver using Linux kernel modules.

The finally developed system provides suitable outputs in various test programs. Figure 6 is examples of test screens, through combining various functions such as output primitives, textures, stencils, and blending, to finally show that all functions work well without any problems.

CONCLUSION

For 3D graphics output in safety-critical applications, including military and avionics application programs, the use of safety-critical libraries such as OpenGL SC is strongly required. In this paper, we show an example of a graphics driver implementation, which complies with the recently announced OpenGL SC 2.0.1 standard, through directly controlling Intel HD 530 graphics chip on a Linux operating system.

This method minimizes the software and hardware resources required for operation through using the kernel-level DRM module directly, instead of the general Windows-based software stack architectures. In addition, since it can be operated as long as the DRM interface is provided, portability to various target systems including embedded environments is enhanced. In near future, our system can be used for various embedded systems, even for safety-critical applications.

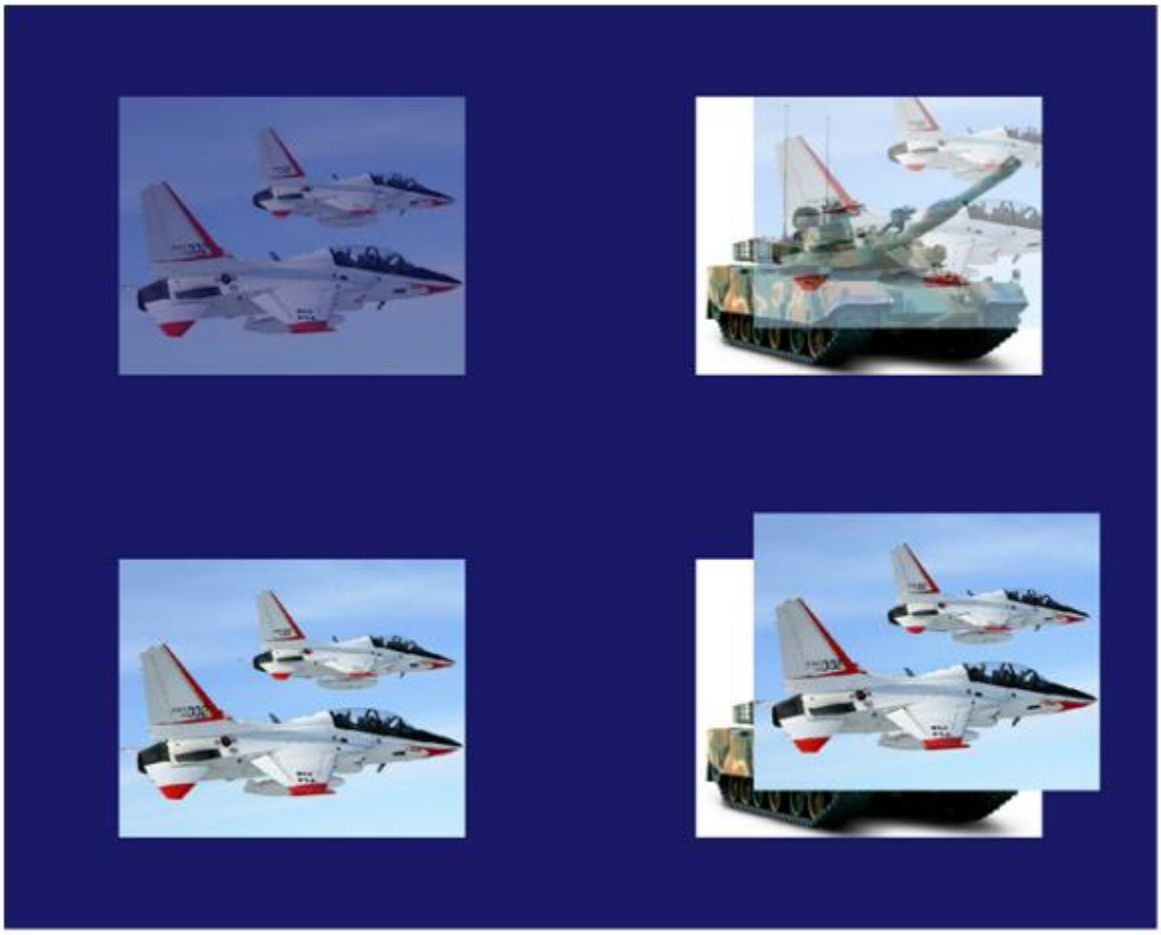

FIGURE 6. EXAMPLES OF SCREEN SHOTS FROM OUR IMPLEMENTATION.

ACKNOWLEDGMENT

This work has supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grand No.NRF-2019R1I1A3A01061310).

This study was supported by the BK21 FOUR project (AI-driven Convergence Software Education Research Program) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (4199990214394).

REFERENCES

[1] M. Segal and K. Akeley. The OpenGL Graphics System: A Specification, Version 1.3. Khronos Group, Aug. 2001.

[2] M. Segal and K. Akeley, The OpenGL Graphics System: A Specification, Version 2.1. Khronos Group, Sep. 2006.

[3] M. Segal and K. Akeley, The OpenGL Graphics System: A Specification, Version 4.6 (Core Profile). Khronos Group, May 2022.

[4] M. Segal and K. Akeley, The OpenGL Graphics System: A Specification, Version 4.6 (Compatibility Profile). Khronos Group, May 2022.

[5] J. Kessenich, The OpenGL Shading Language, Version 1.20. Khronos Group, Sep. 2006.

[6] J. Kessenich, D. Baldwin, and R. Rost, The OpenGL Shading Language, Version 4.60.7. Khronos Group, Jul. 2019.

[7] D. Blythe, OpenGL ES Common/Common-Lite Profile Specification, version 1.0.02. Khronos Group, Apr. 2008.

[8] A. Munshi and J. Leech, OpenGL ES Common/Common-Lite Profile Specification, version 1.1.12 (Full Specification). Khronos Group, Apr. 2008.

[9] A. Munshi and J. Leech, OpenGL ES Common Profile Specification, version 2.0.25 (Full Specification). Khronos Group, Nov. 2010.

[10] J. Leech and B. Lipchak, OpenGL ES version 3.0.6. Khronos Group, Nov. 2019.

[11] J. Leech, OpenGL ES version 3.2. Khronos Group, May 2022.

[12] C. Hall and C. Knaus, OpenGL ES: Safety-Critical Profile Specification, version 1.0. Khronos Group, Jun. 2005.

[13] B. Stockwell, OpenGL SC: Safety-Critical Profile Specification, version 1.0.1. Khronos Group, Mar. 2009.

[14] A. Fabius and S. Viggers, OpenGL SC Version 2.0.0 (Full Specification). Khronos Group, Apr. 2016.

[15] A. Fabius and S. Viggers, OpenGL SC Version 2.0.1 (Full Specification). Khronos Group, Jul. 2019.

[16] R. J. Simpson, The OpenGL ES Shading Language, Version 1.00, Revision 17. Khronos Group, May 2009.

[17] R. J. Simpson, The OpenGL ES Shading Language, Version 3.00, Revision 6. Khronos Group, Jan. 2016.

[18] R. J. Simpson and J. Kessenich, The OpenGL ES Shading Language, Version 3.20.6. Khronos Group, Jul. 2019.

[19] N. Baek and H. Lee, "OpenGL ES 1.1 implementation based on openGL," Multimedia Tools and Applications, vol. 57, no. 3, pp. 669–685, 2012.

[20] N. Baek, "OpenGL SC implementation on the openGL hardware," IEICE Tranactions on Information and Systems, vol. E95-D, no. 10, pp. 2598–2592, 2012.

[21] N. Baek, "Providing safety-critical 3D graphics features on single-board computers," BigDAS 2016, 2016.

[22] N. Baek, "Prototype implementation of the openGL ES 2.0 shading language off-line compiler," Cluster Computing, vol. 22, pp. 943–948, 2018.

[23] N. Baek and K. Kim, "Design and implementation of openGL SC 2.0 rendering pipeline," Cluster Computing, vol. 22, pp. 931–936, 2019.

[24] Mesa Team, The Mesa 3D Graphics Library, (retrieved in Sep 2023). [Online]. Available: http://www.mesa3d.org/

[25] R. E. Faith, The Direct Rendering Manager: Kernel Support for the Direct Rendering Infrastructure, 2020 (retrieved in Sep 2023). [Online]. Available: http://dri.sourceforge.net/doc/drm low level.html

[26] J. Fonseca, Direct Rendering Infrastructure: Architecture, 2005 (retrieved in Sep 2023). [Online]. Available: https://paginas.fe.up.pt/mei04010/dri-architecture.pdf

[27] Arch Linux, Kernel Mode Setting, 2018 (retrieved in Sep 2023). [Online]. Available: https://wiki.archlinux.org/index.php/kernel mode setting

[28] K. Packard and E. Anholt, The Graphics Execution Manager: Part of the Direct Rendering Manager, 2008 (retrieved in Sep 2023). [Online]. Available: https://lwn.net/Articles/283798/

AUTHOR INFORMATION

Nakhoon Baek, Professor, School of Computer Science and Engineering, Kyungpook National University, Daegu, Republic of Korea. E-mail: nbaek@knu.ac.kr.