Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.11 n.55, p. 230-237. September/October, 2025.
DOI: https://doi.org/10.5935/jetia. v.11 n.55.2670

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

PERFORMANCE IMPROVEMENT IN REAL-TIME 3D RENDERING USING OPENGL FRAGMENT SHADER AND UNITY SURFACE SHADER

Eriq Muhammad Adams Jonemaro[1], Ivan Maulana Agusti[2], Herman Tolle[3], Muhammad Aminul Akbar[4] and Aryo Pinandito[5]

[1, 2, 3] Department of Informatics Engineering, Faculty of Computer Science, Brawijaya University, Malang, Indonesia.
[4, 5] Department of Information Systems, Faculty of Computer Science, Brawijaya University, Malang, Indonesia.

[1]https://orcid.org/0000-0001-8315-3210, [2]https://orcid.org/0009-0003-6453-3068, [3]https://orcid.org/0000-0002-1550-0513
[4]https://orcid.org/0000-0003-0794-7443, [5]0000-0002-8509-2383

Email: eriq.adams@ub.ac.id, ivanm63@student.ub.ac.id, emang@ub.ac.id, muhammad.aminul@ub.ac.id, aryo@ub.ac.id

ARTICLE INFO

Article History
Received: September 4, 2025
Revised: September 30, 2025
Accepted: October 7, 2025
Published: October 31, 2025

Keywords:
OpenGL fragment Shader,
Unity surface Shader,
Real-time 3D rendering,
GPU-based rendering,
Shader performance.

ABSTRACT

3D Objects with high geometric complexity are needed to create detailed visualization in video games. However, rendering 3D objects with high geometric complexity requires high computing resources, which can reduce the framerate. GPU-based rendering can make the rendering process efficient. This study uses comprehensive evaluation parameters to evaluate the performance of real-time 3D GPU-based rendering between Unity surface shader and OpenGL fragment shader, based on geometric complexity. The experiment was conducted by rendering various cube objects. The collected data, analyzed using statistical tests, showed that the OpenGL fragment shader had better overall performance in framerate, memory usage, and CPU usage, but more GPU Usage. Thus, this indicates that the OpenGL fragment shader relies more heavily on GPU processing power for rendering efficiency, making it better suited for desktop systems with dedicated GPUs, while Unity surface shaders may be more appropriate for systems with integrated graphics.

I. INTRODUCTION

The video game industry has experienced rapid development in recent years [1] [2]. Nowadays, the video game industry drives in advancement of three-dimensional graphics technology [3]. Real-time 3D object rendering technology plays a significant role in creating a realistic visual experience in the world of video games [4]. Increasingly significant improvements in graphics hardware technology allow rendering 3D worlds with increasingly detailed visuals [5].

The more visual detail rendered increases the complexity of the geometry, resulting in greater computing resource requirements and longer rendering times [6]. High rendering time will reduce framerate during rendering [7]. A high Framerate can create an immersive and seamless experience [8]. GPU-based rendering can result in higher framerates compared to CPU-based rendering [9]. It can be implemented using various platforms' game engines and graphics API.

Unity is a popular and de facto game engine used by game developers [10]. OpenGL is a low-level graphics API widely used in graphics, CAD, and Virtual Reality [11] applications and can run on multiple platforms [12]. GPU-based rendering in the Unity platform can be implemented using Unity compute shader and Unity surface shader. It can also be implemented using the OpenGL compute shader and the OpenGL fragment shader on the OpenGL platform. Unity compute shaders and OpenGL compute shaders are programs designed for general-purpose computing (GPGPU) that are not limited to graphics computing [13] [14]. In contrast, Unity surface shaders and OpenGL fragment shaders are designed explicitly for graphics computing.

While existing studies have examined performance differences between rendering platforms, understanding shader-specific performance characteristics remains underexplored. Studies comparing Unity and OpenGL have mainly focused on GPGPU implementations through compute shaders [15] [16] for specialized applications such as particle systems and deformable object simulations using framerate as the primary performance metric. However, these investigations do not address graphics-specific shader programs—Unity surface shaders and OpenGL fragment shaders—which game developers routinely employ for rendering complex

lighting scenarios [17] [18]. Furthermore, previous comparative studies typically measured only framerate without considering other critical performance metrics such as CPU usage, GPU usage, and memory consumption. This single performance metric approach prevents developers from understanding performance-resource trade-offs, such as whether a higher framerate comes at the cost of increased CPU overhead or memory consumption, which are critical for matching rendering technology to specific hardware constraints and performance requirements.

Therefore, this study investigates the real-time performance of 3D object rendering between Unity surface shader and OpenGL fragment shader based on geometry complexity, with comprehensive evaluation parameters including framerate, CPU usage, GPU usage, and memory usage. The results provide comprehensive guidelines for game developers to determine the platform and rendering technology that matches their hardware constraints and performance requirements.

II. LITERATURE REVIEW

II.1 GRAPHICS API PERFORMANCE COMPARISONS

GPU-based rendering is widely used to make the rendering process more efficient with computational parallelization that utilizes the advanced multi-core processing of the GPU. The large selection of GPU-based rendering technologies and platforms makes it difficult for game developers to choose appropriate platforms and technologies. Therefore, many studies investigate rendering performance in various technologies and platforms. Several studies investigating the performance of OpenGL in various specific development platforms, such as the one conducted by [19], show that GLSL performs better than WebGL in augmented reality applications. Meanwhile, OpenGL ES and OpenGL performance using mobile GPU and desktop GPU are explored by [20].

Furthermore, some studies evaluate the performance between OpenGL and other graphics APIs, such as [21], where OpenGL has more efficient energy consumption than Vulkan. However, it has lower performance than Vulkan using Nvidia's Jetson Nano. A similar study [22] showed that Vulkan has a higher framerate than OpenGL ES in big.LITTLE architecture. Additionally, based on studies conducted by [23], OpenGL is faster than DirectX, and Vulkan uses integrated graphics, while Vulkan is the fastest for complex models, while DirectX is the fastest for simple models using discrete graphics.

These API evaluations reveal important performance differences across platforms and hardware configurations. However, these studies evaluate native API implementations rather than game engine shader programs. The study that evaluates the performance characteristics of the Unity surface shaders and OpenGL fragment shaders—the graphics-specific shaders commonly used for Material rendering—across varying geometric complexity with comprehensive performance metrics remains underexplored.

II.2 UNITY AND OPENGL RENDERING PERFORMANCE

In addition to the OpenGL performance evaluation study with the graphics API, several studies evaluate the performance of OpenGL and Unity, such as [24], showing that LibGDX, which utilizes OpenGL, has lower CPU and memory usage than Unity. Several studies were also conducted to evaluate the performance of Unity compute shader, where Unity compute shader has higher performance than CPU-based rendering in the simulation of deformable body volume [25] and volumetric deformable object simulation [26]. In addition, Unity compute shader also has a higher framerate than OpenGL compute shader [16]. However, in deformable object simulation, Unity compute shader has a lower framerate than OpenGL compute shader [15].

While these studies provide valuable insights into Unity and OpenGL performance, critical limitations prevent an understanding of graphics rendering shader performance. Studies [15][16][25][26] use compute shaders for physics simulation rather than graphics shaders for material rendering, and measure combined simulation plus rendering workloads without isolating rendering performance characteristics. Study [24] tested 2D sprites with aggregate CPU and RAM metrics but no 3D shader analysis or GPU profiling. Across all studies, none examines Unity surface shaders versus OpenGL fragment shaders. They do not systematically vary geometric complexity to characterize shader performance, nor provide comprehensive CPU, GPU, and memory profiling simultaneously for rendering workloads. These gaps necessitate an investigation of Unity surface shader and OpenGL fragment shader performance across varying geometric complexity with multi-dimensional performance metrics.

III. METHODOLOGY

The research methodology flow in this study is shown in Figure 1. This study begins by formulating problems and research objectives, focusing on analyzing the performance of Unity surface Shader and OpenGL fragment Shader using multiple test parameters. Then proceed with the design of experiments and development of benchmark applications. The benchmark application evaluates Unity surface shader and OpenGL fragment shader performance. Performance data collection is done through the MSI Afterburner monitoring application. Finally, the data that has been collected will be analyzed using inferential statistics.

III.1 BENCHMARK APPLICATION DEVELOPMENT

The procedure for rendering cube objects with varying amounts will be implemented to simulate the rendering of 3D objects with various levels of geometrical complexities. Two benchmark applications are developed: one implemented using Unity surface shader and the other implemented using OpenGL fragment shader. The Unity surface shader benchmark application was developed using Unity 2022.3.22f1 with the C# programming language. Meanwhile, the OpenGL fragment shader benchmark application was developed using OpenGL 3.3 and the C++programming language. Implementation of the rendering process in both benchmark applications is made as identical as possible by providing cube dimension constraints using 1x1x1 units using the Phong-lighting model and shadow mapping. The flow of the rendering implementation procedure of both benchmark applications can be seen in Figures 2 and 3, respectively. Generally, the rendering process in both benchmark applications begins with creating cube objects by defining vertices, normals, and

UVs. Afterwards, the cube object's mesh data is sent to the GPU, which will later perform lighting calculation, and finally, the object is rendered on the screen.

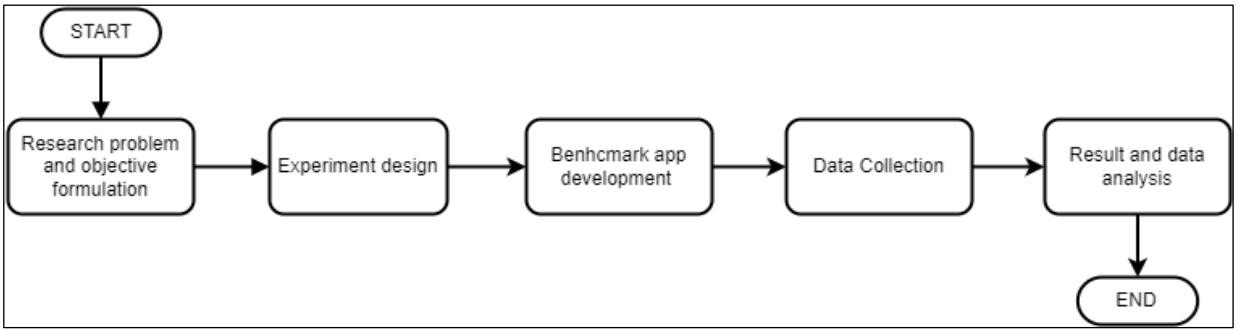

Figure 1: Research methodology flow.
Source: Authors, (2025).

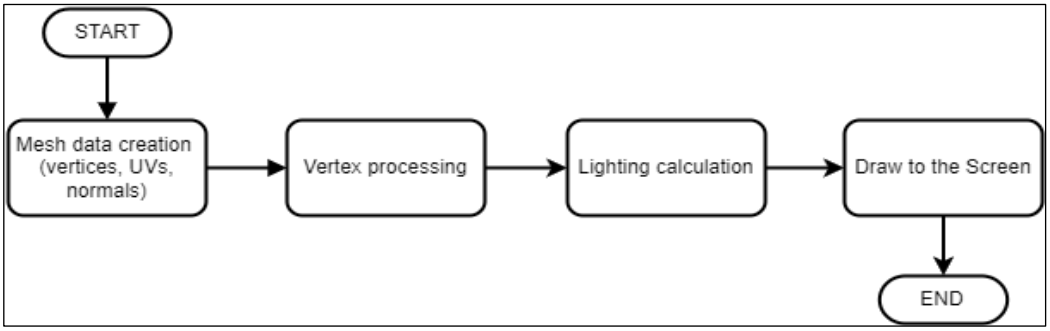

Figure 2: Rendering procedure using Unity surface shader.
Source: Authors, (2025).

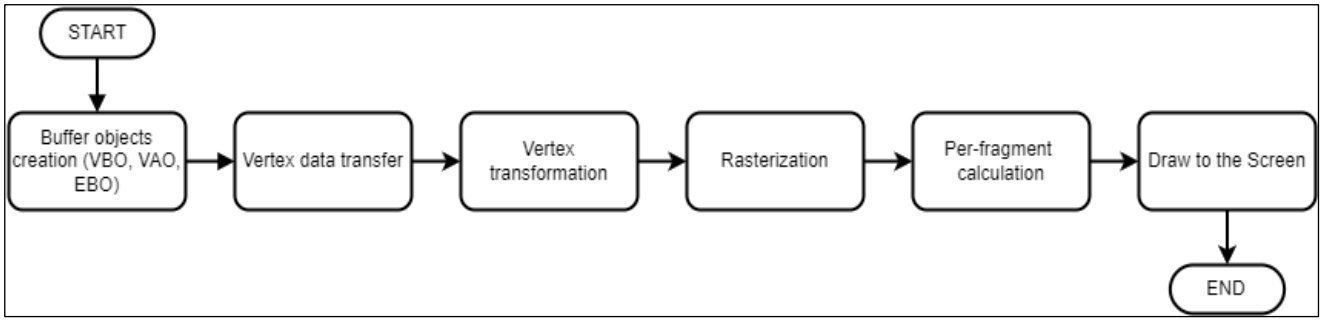

Figure 3: Rendering procedure using OpenGL fragment shader.
Source: Authors, (2025).

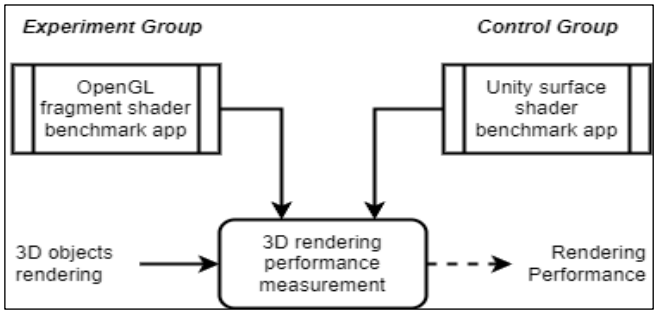

Figure 4: Experimental flow.
Source: Authors, (2025).

III.2 EXPERIMENT DESIGN

Figure 4 shows the experimental design of this study. 3D rendering performance will be compared using an OpenGL fragment shader and a Unity surface shader. Each shader is evaluated using a 3D cube object with a varying number of objects. This study used cube objects for evaluation, following established GPU benchmarking practices [27]. Using cubes for evaluation provide several advantages: (1) all cubes have the same shape and vertex count, ensuring consistent geometry; (2) complexity can be easily adjusted by adding more cubes (800 to 40,000); (3) cubes can be created programmatically without external 3D files, making the experiment reproducible on any system; and (4) both Unity and OpenGL can generate identical cubes. This controlled setup allows a clear comparison

of shader performance. The rendering performance in memory usage, CPU usage, GPU usage, and framerate is measured and analyzed statistically to compare shader performance in a desktop environment.

III.3 DATA COLLECTION

Data was collected by running two benchmark applications and the MSI Afterburner monitoring application. The tests involved varying the number of Cube objects from 800 to 40000, with two increments. Each test lasted for 1 minute on a Windows 11 64-bit operating system, using an Intel Core i5-12500H processor, NVIDIA RTX 3050 GPU, 1TB SSD, and 16GB of DDR4 RAM. The baseline resolution used for the tests was 1280x720 (HD). 720p was selected in this study for wide reproducibility in various hardware specifications, as aligned with this study [28]. Furthermore, this 720p is widely used in benchmark applications such as Unigine Superposition Benchmark [29].

III.4 DATA ANALYSIS

Log data obtained from the MSI Afterburner application will be processed and stored in spreadsheet format. The Shapiro-Wilk test is performed to determine whether the data is normally distributed. If the data is normally distributed, it will be used as a parametric test using the t-test, while the opposite will be a non-parametric test using the Mann Whitney U test. The t-test and the Mann-Whitney U test are used to determine if there is a significant difference between the performance of Unity surface shader and OpenGL fragment shader—all statistical analyses conducted using R Studio.

IV. RESULTS AND DISCUSSIONS

The experimental results from 50 trials for each shader show significant performance differences between the OpenGL fragment shader and the Unity surface shader in all evaluation parameters. A descriptive statistic of the performance measurements of the OpenGL fragment shader and the Unity surface shader is shown in Table 1. The normality test using the Shapiro-Wilk test shows that all performance data in all evaluation parameters are not in normal distribution (*p-value* <.05) (Tabel 2). Therefore, a non-parametric test with the Mann-Whitney U-test is used to determine significant differences in performance data in all evaluation parameters between the OpenGL fragment shader and the Unity surface shader. The results of the Mann-Whitney U-Test are shown in Table 3, where there are significant differences (*p-value* <.05) in all performance data from all evaluation parameters.

Based on the test results, OpenGL fragment shaders have an average frame rate 5.8 times higher than Unity surface shaders. In average memory usage, OpenGL fragment shaders are 4.4 times more efficient than Unity surface shaders. OpenGL fragment shaders also have an average CPU usage 1.6 times lower than Unity surface shaders. However, OpenGL fragment shaders have an average GPU usage 7.8 times higher than Unity surface shaders. The rendering comparison performance in framerate, CPU usage, GPU usage, and memory usage is shown in Figures 5, 6, 7, and 8, respectively.

Table 1: Descriptive statistics of OpenGL fragment shader and Unity surface shader performance measurements.

Shader	Parameter	Mean	SD	Min	Max	Median
Fragment Shader	CPU Usage (%)	13.06	1.74	11.00	18.00	13.00
Surface Shader	CPU Usage (%)	21.26	1.26	19.00	24.00	21.00
Fragment Shader	Framerate (FPS)	101.78	95.98	34.00	492.00	62.00
Surface Shader	Framerate (FPS)	17.46	29.57	4.00	181.00	7.00
Fragment Shader	GPU Usage (%)	31.12	5.82	26.00	52.00	27.50
Surface Shader	GPU Usage (%)	4.30	2.61	3.00	18.00	3.00
Fragment Shader	Memory Usage (MB)	110.00	1.07	107.00	111.00	110.00
Surface Shader	Memory Usage (MB)	485.20	146.83	237.00	715.00	492.00

Source: Authors, (2025).

Table 2: The Shapiro-Wilk test of performance data between the OpenGL fragment shader and the Unity surface shader.

Shader	Parameter	*p-value*
Fragment Shader	CPU Usage (%)	1.97×10^{-08}
Surface Shader	CPU Usage (%)	8.77×10^{-03}
Fragment Shader	Framerate (FPS)	4.00×10^{-09}
Surface Shader	Framerate (FPS)	4.40×10^{-12}
Fragment Shader	GPU Usage (%)	5.19×10^{-07}
Surface Shader	GPU Usage (%)	3.60×10^{-11}
Fragment Shader	Memory Usage (MB)	6.46×10^{-07}
Surface Shader	Memory Usage (MB)	1.56×10^{-02}

Source: Authors, (2025).

Table 3: Mann-Whitney U-Test results of OpenGL fragment shader and Unity surface shader performance measurements.

Parameter	*p-value*
CPU Usage (%)	2.52909×10^{-18}
GPU Usage (%)	2.00599×10^{-18}
Memory Usage (MB)	3.87951×10^{-18}
Framerate (FPS)	1.56527×10^{-14}

Source: Authors, (2025).

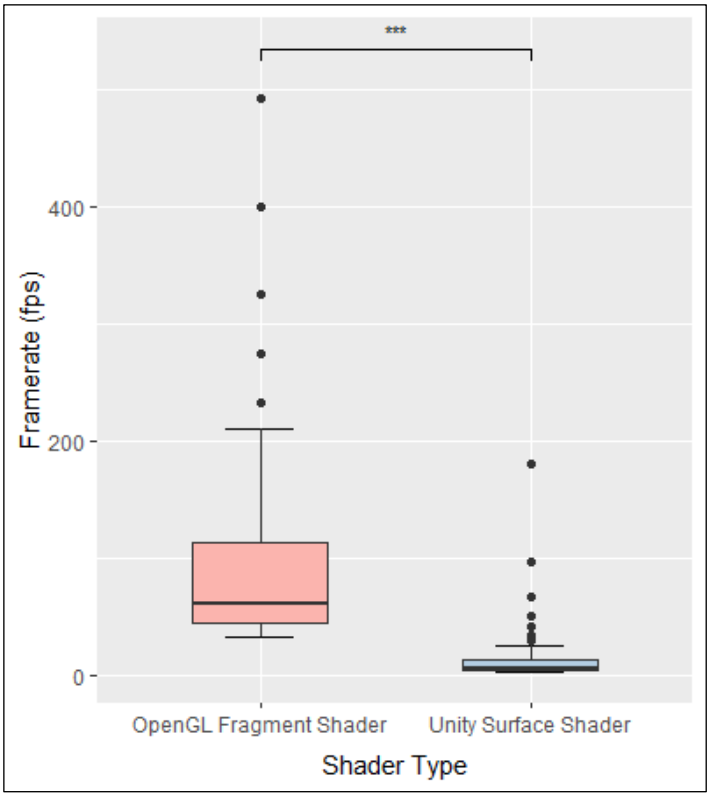

Figure 5. Comparison of the framerate in rendering between the OpenGL fragment shader and the Unity surface shader.
Source: Authors, (2025).

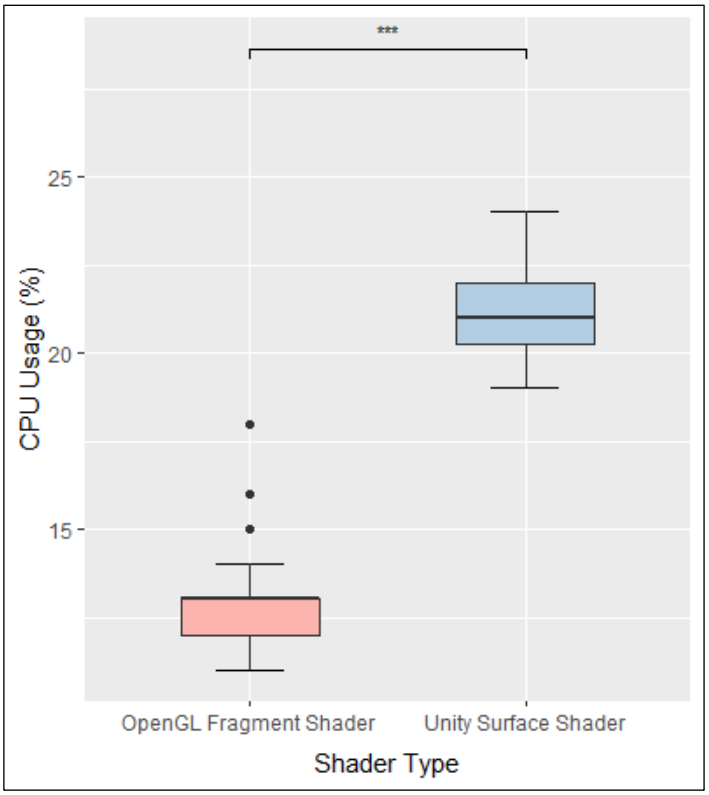

Figure 6: Comparison of CPU usage in rendering between the OpenGL fragment shader and the Unity surface shader.
Source: Authors, (2025).

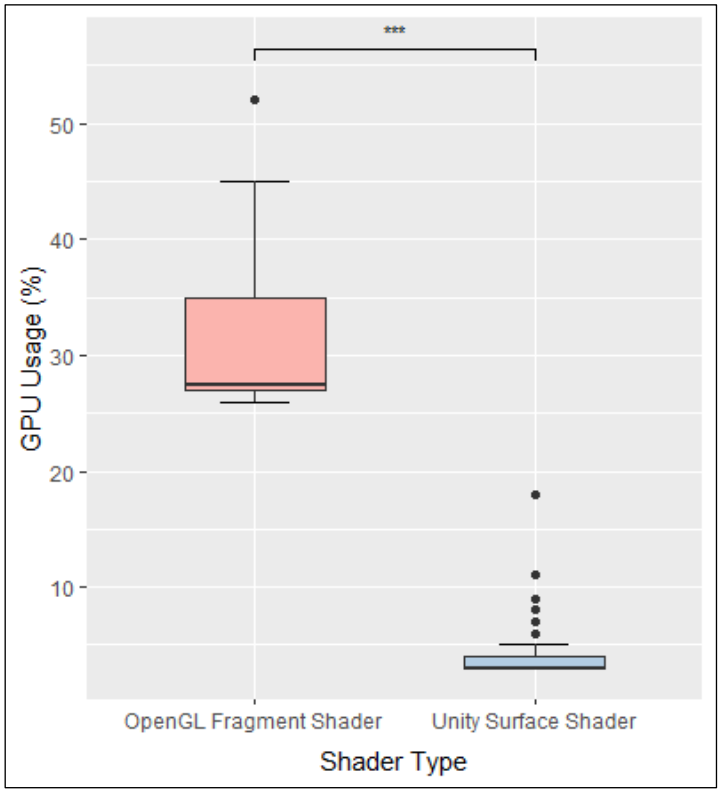

Figure 7: Comparison of GPU usage in rendering between the OpenGL fragment shader and the Unity surface shader.
Source: Authors, (2025).

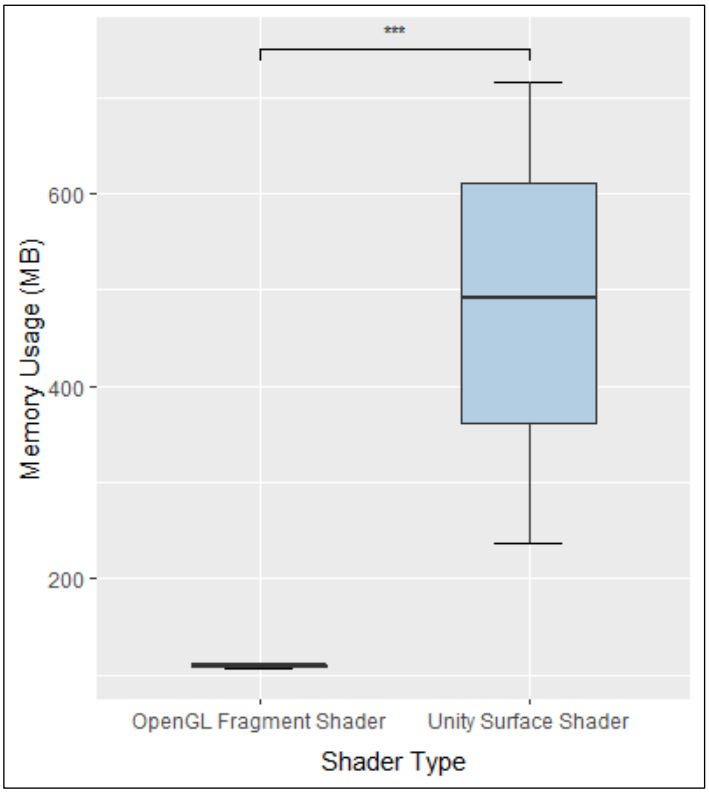

Figure 8: Comparison of memory usage in rendering between the OpenGL fragment shader and the Unity surface shader.
Source: Authors, (2025).

The higher GPU utilization of OpenGL fragment shaders leads to reduced CPU usage. Lower CPU and memory usage of OpenGL fragment shaders in this study aligns with the findings in [24]. Meanwhile, the higher frame rate of OpenGL fragment shaders in this study is also aligned with the experiments by [16]. However, in contrast to the findings obtained by [15], OpenGL compute shaders have lower frame rates than Unity compute shaders in particle simulations. These different findings indicate that performance characteristics depend highly on specific aspects of the rendering implementation.

The results suggest practical implications for platform selection. The higher GPU utilization in OpenGL fragment shaders indicates that they are better suited for desktop systems with dedicated GPUs. In contrast, Unity surface shaders are better suited for systems with integrated graphics. However, the higher GPU usage may impact power consumption, which is not measured in this study.

V. LIMITATIONS

Although this study provides valuable insights, this study has several limitations that must be addressed. First, cube objects used in evaluation may not fully represent real-world 3D models with various polygon counts and mesh complexities. Second, the experiment was conducted on a single desktop configuration, which can be extended to various configurations and platforms. Third, baseline 720p resolution may not scale linearly to higher and common used resolutions in modern applications such as 1080p or 4K. Additionally, this study focused on Phong lighting and shadow mapping, without evaluating advanced techniques such as physically-based rendering or global illumination. Framerate consistency, power consumption, particularly relevant for sustained workloads and mobile deployments, were not measured in the performance evaluation. Hence, addressing these limitations by evaluating real-world 3D models in various hardware configurations, platforms, and resolutions, while incorporating additional performance metrics, is needed. Thus, it can provide more comprehensive guidelines for shader technology selection.

VI. CONCLUSIONS

Based on the experimental results, OpenGL fragment shaders have better overall performance in framerate, lower memory usage, and lower CPU usage than Unity surface shaders. However, OpenGL fragment shaders required significantly more GPU usage, indicating that they rely more heavily on GPU processing power for optimal performance in their rendering process. Furthermore, OpenGL fragment shaders are particularly well-suited for desktop systems with dedicated GPUs and Unity surface shaders may be better suited for systems with integrated graphics. Thus, developers should take into account the capabilities of the target hardware, the performance requirements, and any specific resource limitations.Furthermore, these findings indicate that OpenGL fragment shaders can guide game developers in choosing the most appropriate rendering technology to build efficient games and graphics applications. However, further investigation must be extended using real-world 3D models across various hardware configurations, platforms, and resolutions. It should also include a wide range of performance metrics, such as frame consistency and power consumption, to provide more comprehensive guidelines for selecting shader technologies.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Eriq Muhammad Adams Jonemaro, Ivan Maulana Agusti, and Herman Tolle.
Methodology: Eriq Muhammad Adams Jonemaro, Ivan Maulana Agusti, and Herman Tolle.
Investigation: Eriq Muhammad Adams Jonemaro, Ivan Maulana Agusti.
Discussion of results: Eriq Muhammad Adams Jonemaro and Ivan Maulana Agusti.
Writing – Original Draft: Eriq Muhammad Adams Jonemaro and Ivan Maulana Agusti.
Writing – Review and Editing: Eriq Muhammad Adams Jonemaro and Aryo Pinandito.
Resources: Eriq Muhammad Adams Jonemaro, Ivan Maulana Agusti, and Muhammad Aminul Akbar.
Supervision: Aryo Pinandito.
Approval of the final text: Eriq Muhammad Adams Jonemaro, Ivan Maulana Agusti, Herman Tolle, Muhammad Aminul Akbar, and Aryo Pinandito.

VIII. ACKNOWLEDGMENTS

This research is part of the Scientific Article Publication Acceleration Program of the Faculty of Computer Science, Brawijaya University.

IX. REFERENCES

[1] H. Xiao, Y. Xia, and A. Baird, "Predicting digital product performance with team composition features derived from a graph network," *Decis. Support Syst.*, vol. 183, no. 114266, p. 114266, Aug. 2024.

[2] T. Kraemer, W. H. Weiger, and S. Heidenreich, "Do all stars shine the same? investigating the nonlinear effects of user and critic reviews on video game sales," *J. Bus. Res.*, vol. 188, no. 115034, p. 115034, Feb. 2025.

[3] Y. Lan, "Development and application of 3D modeling in game," *Academic Journal of Science and Technology*, vol. 7, no. 2, pp. 94–97, Sep. 2023.

[4] M. Casini, "Advanced digital design tools and methods," in *Construction 4.0*. Elsevier, 2022, pp. 263–334.

[5] R. M. Reffat, "A virtual platform for improving coordination and promoting cooperation on traffic safety," *Int. J. Transp. Sci. Technol.*, vol. 3, no. 1, pp. 43–62, Mar. 2014.

[6] B. Kuth, M. Oberberger, F. Kawala, S. Reitter, S. Michel, M. Chajdas, and Q. Meyer, "Real-time meshlet decompression," *Comput. Graph.*, no. 104292, p. 104292, Jul. 2025.

[7] N. Komodakis, C. Panagiotakis, and G. Tziritas, "3D visual reconstruction of large scale natural sites and their fauna," *Signal Process. Image Commun.*, vol. 20, no. 9-10, pp. 869–890, Oct. 2005.

[8] P. Wu, Y. Liu, H. Chen, X. Li, and H. Wang, "VR-empowered interior design: Enhancing efficiency and quality through immersive experiences," *Displays*, vol. 86, no. 102887, p. 102887, Jan. 2025.

[9] M. Yuan, L. Wang, and B. Ai, "Parallel terrain node processing and rendering: a GPU-accelerated approach," in *International Conference on Remote Sensing, Mapping, and Image Processing (RSMIP 2025)*, F. Tosti and R. Alvarez, Eds.SPIE, May 2025, p. 45.

[10] T. M. Moreira, J. G. de Faria, Jr, P. O. S. Vaz-de Melo, and G. Medeiros-Ribeiro, "Development and validation of an AI-Driven model for the la rance tidal barrage: A generalisable case study," *Appl. Energy*, vol. 332, no. 120506, p. 120506, Feb. 2023.

[11] Y. Liu, Z. Xie, and H. Liu, "Three-line structured light vision system for non-cooperative satellites in proximity operations," *Chin. J. Aeronaut.*, vol. 33, no. 5, pp. 1494–1504, May 2020.

[12] S. Liu, S. Tian, Z. Zhang, L. Liu, and T. Li, "Dynamic growth tomato inflorescence modeling with elastic mechanics data," *Smart Agric. Technol.*, vol. 11, no. 100884, p. 100884, Aug. 2025.

[13] D. Algis, B. Bramas, E. Darles, and L. Aveneau, "InteropUnityCUDA: A tool for interoperability between unity and CUDA," *Softw. Pract. Exp.*, vol. 55, no. 6, pp. 1127–1141, Jun. 2025.

[14] S. Park and N. Baek, "A shader-based ray tracing engine," *Appl. Sci. (Basel)*, vol. 11, no. 7, p. 3264, Apr. 2021.

[15] M. S. Kim, N.-J. Sung, Y.-J. Choi, and M. Hong, "Performance comparison of particle simulation using gpu between opengl and unity," *KIPS Transactions on Software and Data Engineering*, vol. 6, no. 10, pp. 479–486, 2017.

[16] M.-H. Choi, M. S. Kim, N.-J. Sung, Y.-J. Choi, and M. Hong, "Comparison in performance of parallel deformable object simulation between opengl and unity," *KIPS Trans. Softw. Data Eng*, vol. 6, pp. 479–486, 2017.

[17] A. Chrysanthakopoulou and K. Moustakas, "Real-time shader-based shadow and occlusion rendering in AR," in *2024 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, Mar. 2024.

[18] J. Zhu, Z. Wu, Q. Zhang, C. Liao, and Z. Huang, "WishGI: Lightweight static global illumination baking via spherical harmonics fitting," *ACM Trans. Graph.*, vol. 44, no. 4, pp. 1–12, Aug. 2025.

[19] M. Hamzaturrazak, E. M. A. Jonemaro, and A. Pinandito, "Performance analysis of 3D rendering method on web-based augmented reality application using WebGL and OpenGL shading language," in *Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology*. New York, NY, USA: ACM, Oct. 2023.

[20] J. McCaffrey, "Exploring mobile vs. desktop opengl performance," pp. 337–352, 2012.

[21] O. Ferraz, P. Menezes, V. Silva, and G. Falcao, "Benchmarking vulkan vs OpenGL rendering on Low-Power edge GPUs," in *2021 International Conference on Graphics and Interaction (ICGI)*. IEEE, Nov. 2021.

[22] M. Lujan, M. McCrary, B. W. Ford, and Z. Zong, "Vulkan vs OpenGL ES: Performance and energy efficiency comparison on the big.LITTLE architecture," in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE, Oct. 2021.

[23] M. Y. Protsyk and T. O. Korotyeyeva, "Analysis the efficiency of programming tools for 3d rendering in video game development," *Scientific Bulletin of UNFU*, vol. 35, no. 3, pp. 158–165, May 2025. [Online]. Available: https://nv.nltu.edu.ua/index.php/journal/article/view/2762

[24] P. Kosidło, K. Kowalczyk, and M. Badurowicz, "Comparison of capabilities of the unity environment and LibGDX in terms of computer game development," *jcsi*, vol. 21, pp. 324–329, Dec. 2021.

[25] H. Va, J. Ma, and M. Hong, "Comparative study of iteration and coefficient influence on volume preservation in position-based dynamics simulation," in *2023 IEEE 6th International Conference on Knowledge Innovation and Invention (ICKII)*. IEEE, Aug. 2023.

[26] H. Va, M.-H. Choi, and M. Hong, "Efficient simulation of volumetric deformable objects in Unity3D: GPU-accelerated Position-Based dynamics," *Electronics (Basel)*, vol. 12, no. 10, p. 2229, May 2023.

[27] F. Fossum, "Real-time rigid body interactions," Master's thesis, Institutt for datateknikk og informasjonsvitenskap, 2011.

[28] J.-H. Nah, Y. Suh, and Y. Lim, "L-bench: An android benchmark set for low-power mobile gpus," Computers Graphics, vol. 61, pp. 40–49, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009784931630111X

[29] D. Shergin, D. Vidiger, and A. Fofanova, "Superposition benchmark: innovative ssrtgi lighting in real time," in ACM SIGGRAPH 2017 Real Time Live!, ser. SIGGRAPH '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 25. [Online]. Available: https://doi.org/10.1145/3098333.3098335