JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 10, pp. 165-173, October 2024
https://doi.org/10.9708/jksci.2024.29.10.165

Towards Next Generation Game Development: A Comprehensive Analysis of Game Engines Technologies

Soo Kyun Kim*, Iqbal Muhamad Ali**, Min Woo Ha***

*Professor, Dept. of Computer Engineering, Jeju National University, Jeju, Korea
**Student, Dept. of Computer Engineering, Jeju National University, Jeju, Korea
***Associate professor, College of Pharmacy, Jeju National University, Jeju, Korea

[Abstract]

Game engines are essential tools in game development, speeding up processes and simplifying the integration of various modules like physics, graphics, animations, and AI. This study provides a comprehensive overview of modern game engine technologies, including advanced rendering techniques, graphics APIs, physics simulations, AI integration, audio systems, networking, VR/AR, and development tools. It highlights recent advancements such as real-time ray tracing, physically based rendering, machine learning for content generation and intelligent NPCs, cloud gaming, and novel input methods like brain-computer interfaces. The paper also explores future directions, including enhanced cross-platform support and new technologies that will drive the evolution of game engines. This analysis serves as a valuable resource for developers, researchers, and industry professionals.

▸ **Key words**: Real-time Ray Tracing, Physically Based Rendering, Temporal Anti-Aliasing, Virtual Reality (VR) and Augmented Reality Workload, Game Engine Architecture

[요 약]

게임 엔진은 물리, 그래픽, 애니메이션, AI와 같은 다양한 모듈의 통합을 간소화하고 개발 속도를 높이는 데 필수적인 도구이다. 본 연구는 현대 게임 엔진 기술에 대한 포괄적인 개요를 제공하며, 고급 렌더링 기술, 그래픽 API, 물리 시뮬레이션, AI 통합, 오디오 시스템, 네트워킹, VR/AR 및 개발 도구를 다룬다. 또한, 실시간 레이 트레이싱, 물리 기반 렌더링, 콘텐츠 생성 및 지능형 NPC를 위한 머신러닝, 클라우드 게임, 뇌-컴퓨터 인터페이스와 같은 새로운 입력 방법 등 최신 발전 사항에 대해 분석한다. 본 논문은 게임 엔진의 발전을 이끌 미래의 방향성, 특히 플랫폼 간 지원 향상 및 신기술 도입에 대해 분석하고, 분석 결과는 개발자, 연구자 및 업계 전문가에게 유용한 자료로 활용될 수 있다.

▸ **주제어**: 실시간 레이 트레이싱, 물리 기반 렌더링 (PBR), 템포럴 안티앨리어싱 (TAA), 가상 현실 (VR) 및 증강 현실 (AR), 게임 엔진 아키텍처

• First Author: Soo Kyun Kim, Corresponding Author: Min Woo Ha
*Soo Kyun Kim (kimsk@jejunu.ac.kr), Dept. of Computer Engineering, Jeju National University
**Iqbal Muhamad Ali (aliiqbal073@gmail.com), Dept. of Computer Engineering, Jeju National University
***Min Woo Ha (minuha@jejunu.ac.kr), College of Pharmacy, Jeju National University
• Received: 2024. 07. 22, Revised: 2024. 09. 19, Accepted: 2024. 09. 25.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

Game engines are foundational software frameworks that power video game development across platforms like PCs, gaming consoles, mobile devices, and emerging technologies such as virtual and augmented reality (VR/AR). These toolkits include reusable components, tools, libraries, and APIs, enabling developers to efficiently create interactive gaming experiences. Initially, custom software was created for each game, but as complexity grew, the need for reusable frameworks led to the development of game engines. These engines allow developers to focus on game-specific logic while leveraging a robust codebase for common tasks like graphics rendering, physics simulation, audio processing, scripting, input handling, and resource management.

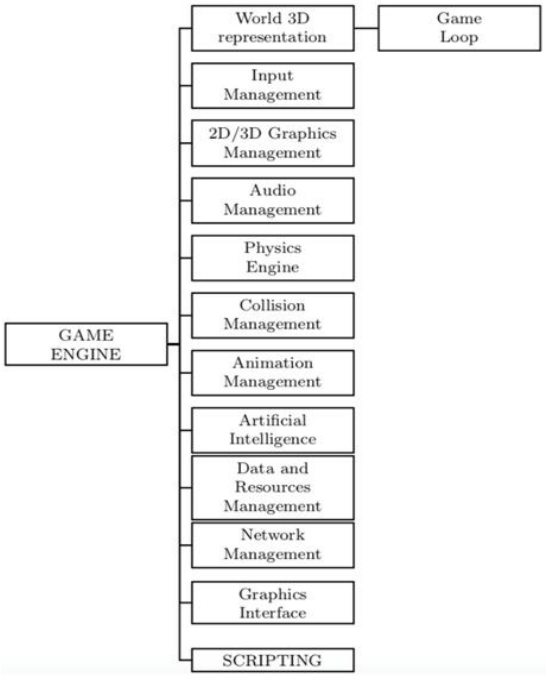


Fig. 1. Overview of Game Engine

Figure 1 presents a generalized overview of game engine architecture, illustrating the core components and systems that enable game development. Over the years, game engines have evolved significantly, incorporating advancements in hardware, programming languages, and development methodologies. Modern game engines employ advanced rendering techniques like real-time ray tracing and physically based rendering (PBR), and use cutting-edge graphics APIs like DirectX 12 and Vulkan for better performance. Physics simulations provide realistic representations of collisions, forces, and movements.

Table 1 provides a summary of various game engines, comparing them based on their licensing models, programming languages, and extensibility. This table helps illustrate the wide variety of engines available to developers, emphasizing the trade-offs in features and capabilities among commercial and open-source options

Table 1. Summary of Game Engines

No	Game Engine	License	Environment Extensibility	Extensions Programming Language
1	Torque Game Engine	Commercial	Very Good	Torque Script, C++
2	Doom Engine 3	Commercial	Very Good	Scripting Language, C++
3	Unreal Engine 2	Commercial	Very Good	Scripting Language, C++
4	EasyWay	Open Source	Poor	Java
5	Sauerbraten (Cube 2)	Open Source	Good	Scripting Language, C++
6	Simulus	Open Source	Poor	C++
7	Simple Direct Engine	Open Source	Poor	C#
8	Ghost Engine	Open Source	Poor	C++
9	Raydium	Open Source	Poor	C
10	Drome	Open Source	Good	C++

AI and machine learning integration enable intelligent NPCs, procedural content generation, and player behavior modeling. Advanced audio systems enhance the auditory experience, and networking capabilities support various architectures for cross-platform multiplayer

experiences. Modern engines support VR/AR natively, with features like motion tracking and spatial mapping enhancing realism.

Game engines also offer powerful development tools, such as visual scripting interfaces, asset management systems, and level editing tools, streamlining the development process and making it accessible to non-programmers. This paper provides an overview of game engine types and technologies, exploring state-of-the-art techniques and future directions, highlighting the cutting-edge technologies shaping the future of game development.

Several studies have explored modern game engines and their evolution, focusing on individual features such as rendering techniques, physics simulations, or platform support. For example, prior research by Vohera et al. [8] and Maggiorini et al.[9] delved into the architecture and capabilities of game engines, providing valuable insights into their technical frameworks. However, these studies often focus on either high-level overviews or specific aspects of engines like graphics rendering or physics without offering a comprehensive feature comparison across the latest versions of multiple engines

This paper provides a detailed comparative analysis of the latest versions of leading game engines, including Unity, Unreal Engine, and CryEngine. Different from the pror work, our research integrates a thorough feature-by-feature comparison across multiple engines, focusing on advancements in rendering, physics, AI, and VR/AR support.

II. Preliminaries

Using the integrated development environment (IDE) Unity, users can design models, create 3D animations, and develop video games. Compatible with both macOS and Windows, Unity supports game development for various platforms, including desktop systems (Windows, macOS, Linux), mobile devices (iOS, Android, Windows Phone), and consoles. Users select the appropriate Software Development Kit (SDK) for the target platform [1]. Unity supports scripting in C#, JavaScript, and Boo, and is developed using C++ and C#. Key features include Nvidia's PhysX physics engine, the Mechanism animation system, a built-in terrain editor, and MLAPI for multiplayer networking [2]. Unity employs MonoDevelop for integrated code editing and supports assets from Blender, 3ds Max, Cinema 4D, and Adobe Photoshop. Notable games developed with Unity include Assassin's Creed: Identity, Temple Run Trilogy, Call of Duty Mobile, and Escape Plan.

Table 2. Feature Comparison of Latest Version of Game Engines

Features	Unity (2023)	Unreal Engine 5 (2023)	Cry Engine 5.7 (2023)
Supported Platforms	PC, Console Mobile Web VR/AR	PC, Console Mobile Web VR/AR	PC, Console VR/AR
Scripting Languages	C#, JavaScript, Boo	C++	C++, Lua
Rendering	Real-time Ray Tracing, PBR	Real-time Ray Tracing, PBR	Physically Based Shading, CryLighting
Physics Engine	Nvidia PhysX	Chaos Physics	CryPhysics
VR/AR Support	Yes	Yes	Yes
Licensing Model	Freemium (Paid tiers)	Free with royalty	Free with royalty

1.1 Unreal Engine

Developed by Epic Games, Unreal Engine (UE) debuted in 1998, with the latest iteration, version 4.20, released in 2018. Created in C++, UE supports PC, mobile, console, and VR platforms [3]. It uses object-oriented programming and reusable libraries to streamline game development, focusing on design and gameplay rather than technical aspects. UE employs Disney's Physically Based Rendering for visual realism and advanced algorithms for

graphics. It achieves high frame rates and realism using Level of Detail (LoD) techniques [4]. Notable games developed with Unreal Engine include Dragon Ball Fighter Z, Ark: Survival Evolved, Street Fighter V, Borderlands 3, and Fortnite.

1.2 GameMaker

Originally created as Animo in 1991 and later rebranded by YOYO Games, GameMaker is popular for creating 2D cross-platform games. Using C# and C++, it is beginner-friendly and supports platforms like Android, iOS, PlayStation, Xbox, Windows Desktop, HTML5, Windows UWP, and Nintendo Switch [5]. GameMaker 2, released in 2017, offers licenses from $39 to $799. Notable games include Katana Zero, Hyper Light Drifter, Shovel Knight: Treasure Trove, Hotline Miami, and Undertale.

1.3 CryEngine

Developed by Crytek, CryEngine powers the Far Cry series. Using Lua and C++ for scripting, it supports Windows, Linux, PSVR, Xbox One, and PlayStation 4 [6]. CryEngine efficiently manages cross-platform changes and provides source code access for extensive customization [7]. Renowned for its graphics and performance capabilities, notable games using CryEngine include Ryse: Son of Rome, Prey, Kingdom Come: Deliverance, and the Crysis series.

In addition to the detaile discussion of various game engines, Table [2] provides a feature comparison of the latest versions of the engines, that includes Unity, Unreal Engine and CryEngine. This comparison focuses on key aspects such as supported platforms, scripting languages, rendering capabilities, physics engines, and VR/AR support.The table aims to highlight the differences in capabilities and features, offering insights into how these engines meet diverse development needs in modern game development.

III. The Proposed Scheme

Numerous studies [8, 9, 10] have explored various game engines. Our research proposes a universal framework for a game engine, encompassing several key components illustrated in Figure 2. As described by Gregory in his work on game engine architecture [11], the modular structure of a game engine ensures flexibility in developing for various platforms

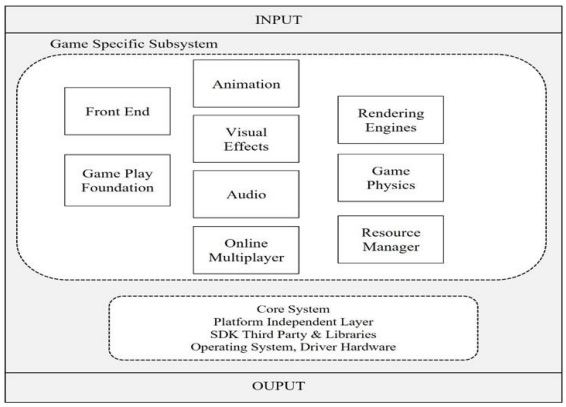


Fig. 2. A Generalized overview of Game Engine Architecture

The framework supports various platforms, including Xbox, PlayStation, Nintendo, PCs running Microsoft Windows and Linux, as well as iOS and Android devices [12]. The Target Hardware component identifies the specific platform for game deployment. Third-party software provides reusable components, reducing development time and costs, and allowing developers to focus on impactful game features. A Platform Independence Layer (PIL) ensures uniform functionality across diverse systems, shielding the engine's mechanics from platform-specific data.

At the core of a game engine are fundamental systems such as asynchronous file I/O, mathematical libraries, memory management, and string hashing. The Resource Manager provides a standardized interface to access game assets like textures, models, fonts, and other data. Rendering, whether 2D or 3D, focuses on the rapid display of visual elements. Visual effects such as particle

systems, dynamic shadows, and light mapping enhance the game's visual appeal.

Most games feature 2D graphics within a 3D environment, including GUIs, in-game menus, and HUDs. Technologies like On-Game Cinematics or Full Motion Video enhance the front-end experience. Physics and collision engines simulate physical interactions, ensuring objects react appropriately to forces and collisions. Animations, whether 2D using sprites or 3D using skeletal systems, are crucial for realistic movement.

The audio system is vital for immersion, generating sounds and music in response to game events. Networking capabilities are essential for multiplayer games, supporting formats like split-screen, networked, single-screen, and massively multiplayer online games (MMOG). Gameplay involves core actions and rules within the virtual environment [13], including player objectives, character skills, and object capabilities. Scripting systems in some engines facilitate the rapid development of these gameplay elements. Developers and designers work together to enhance the game with dynamic player characters, multiple camera systems, AI-controlled NPCs, weaponry, vehicles, and other subsystems.

IV. Current State of the Art Techniques for Game Engines

1. Rendering Techniques

1.1 Real-time ray tracing

Game engines are increasingly adopting real-time ray tracing technologies, such as NVIDIA RTX and AMD Ray Tracing, to enhance visual fidelity with realistic lighting, shadows, and reflections [14]. Real-time ray tracing simulates light behavior as it interacts with virtual objects, enabling accurate rendering of effects like global illumination, reflections, refractions, and soft shadows. This technology revolutionizes photorealistic graphics by eliminating the need for pre-computed lighting solutions and allowing dynamic lighting changes in real-time.

1.2 Physically based rendering (PBR)

PBR techniques simulate light and materials accurately, resulting in realistic visuals. PBR [15] uses properties like albedo, roughness, and metallic values to determine surface interactions with light, ensuring material consistency across various lighting conditions. This enhances visual coherence and allows for efficient, realistic texture mapping, reducing the need for manual adjustments.

1.3 Temporal anti-aliasing (TAA)

Advanced anti-aliasing techniques like TAA improve image quality by reducing aliasing artifacts and enhancing overall visual clarity. Aliasing refers to jagged edges in rendered graphics [16]. TAA leverages information from multiple frames to smooth these artifacts, resulting in cleaner images. This technique is particularly effective in games with dynamic camera movements, providing consistent anti-aliasing quality.

2. Graphics APIs

2.1 Low-level APIs

Modern game engines support low-level graphics APIs like DirectX 12 and Vulkan, which provide better performance, increased parallelism, and lower CPU overhead compared to their predecessors. These low-level APIs offer more direct control over graphics hardware, enabling developers to optimize resource usage and achieve higher frame rates. Additionally, they support features like explicit multi-GPU rendering, asynchronous compute, and advanced scheduling techniques, which can further enhance performance on modern hardware.

2.2 Cross-platform compatibility

Game engines are designed to support multiple graphics APIs, ensuring cross-platform compatibility, and enabling developers to target a

wide range of hardware and platforms. This includes support for APIs like DirectX, Vulkan, Metal (for Apple devices), and OpenGL, allowing game developers to build and deploy their games across various operating systems and devices without significant code changes.

3. Physics and Dynamics

3.1 Rigid body dynamics

Game engines use advanced physics engines to simulate rigid body dynamics, enabling realistic representation of object collisions, forces, and movements. These engines calculate interactions between rigid bodies, considering factors such as mass, inertia, friction, and restitution. This allows for accurate simulations of physical phenomena like collisions and explosions, enhancing the immersion of virtual environments.

3.2 Soft body dynamics

Some game engines also support soft body dynamics, allowing for realistic deformation of materials like cloth, fluids, and deformable objects. Techniques such as finite element methods, mass-spring systems, or position-based dynamics model the behavior of these materials. Soft body dynamics are crucial for realistic representations of character clothing, hair, and other flexible objects, adding realism to virtual environments.

4. Artificial Intelligence (AI) and Machine Learning

4.1 Pathfinding and navigation

Game engines use advanced pathfinding algorithms and navigation meshes to enable realistic and efficient movement of non-player characters (NPCs) and agents. Pathfinding algorithms, such as A* or Dijkstra's algorithm [17], calculate optimal paths for NPCs in complex environments, while navigation meshes represent navigable areas within the game world. These techniques ensure natural movement and obstacle avoidance, enhancing the game's believability.

Additionally, game engines are integrating machine learning capabilities to develop more intelligent and adaptive AI systems for NPCs, procedural content generation, and player behavior modeling [18].

5. Audio and Sound

5.1 Spatial audio

Game engines support advanced spatial audio technologies, such as 3D positional audio and audio occlusion, to create immersive auditory experiences. Spatial audio positions sound sources accurately in a 3D environment, simulating effects like distance attenuation, Doppler shift, and occlusion. These features enhance the sense of presence and immersion in games. Many game engines integrate with audio middleware solutions like FMOD, Wwise, and Fabric, which offer tools for sound design, music implementation, and audio asset management, including real-time parameter control, audio mixing, and environmental audio effects [19]. This integration streamlines audio workflows within the game engine.

6. Networking and Multiplayer

6.1 Client-server architectures

Game engines support various networking architectures, including client-server models, peer-to-peer networking, and hybrid approaches, enabling multiplayer experiences across different platforms. Client-server architectures, commonly used for large-scale multiplayer games, rely on a dedicated server to manage the game state and synchronize data between clients [20]. Peer-to-peer networking allows direct client communication, reducing the need for a central server. Hybrid approaches combine both architectures for scalable and robust multiplayer solutions.

6.2 Dedicated server support

Game engines often provide dedicated server solutions for efficient management and hosting of multiplayer sessions. Dedicated servers ensure consistent performance, reliable data synchronization, and centralized management of game logic and rules. They also support features like

matchmaking, player rankings, and leaderboards, enhancing the multiplayer experience.

7. Virtual and Augmented Reality (VR/AR)

7.1 AR integration

Many game engines support virtual and augmented reality platforms, such as Oculus Rift, HTC Vive, and ARKit/ARCore, facilitating immersive VR and AR experiences. This includes support for VR/AR rendering techniques, input handling, and performance optimizations.

7.2 VR/AR-specific features

Game engines implement features like motion tracking, stereoscopic rendering, and spatial mapping to enhance VR and AR applications. Motion tracking ensures accurate user movement tracking for natural interactions. Stereoscopic rendering creates depth perception by generating separate images for each eye, providing a realistic 3D experience. Spatial mapping allows AR applications to integrate virtual objects into the real world, enabling interactions within physical spaces.

8. Editor and Development Tools

8.1 Visual scripting

Game engines often provide visual scripting tools, allowing developers to create and modify game logic using a visual, node-based interface, reducing the barrier to entry for non-programmers. Visual scripting environments enable developers to construct game behaviors, interactions, and event handling through a graphical representation of nodes and connections, abstracting away the complexities of traditional coding. This approach can significantly streamline the development process and make game programming more accessible to a broader range of developers.

Table 3. Advantages and Disadvantages of Advance Game Deveolpment Technologies

Technology	Advantages	Disadvantages
Real-time Ray Tracing	- Highly realistic lighting,shadows, and reflections Dynamic lighting changes in real-time	- High computational cost Requires modern GPUs for optimal performance
Physically Based Rendering (PBR)	Consistent material appearance across lighting conditions Realistic texture mapping	- Requires careful calibration of material properties - Can be resource-intensive for complex scenes
Artificial Intelligence (AI)	- Enables intelligent NPCs and dynamic content generation - Enhances player immersion	- Complex to implement - Can increase development time and computational load
Virtual/Augmented Reality (VR/AR)	- Immersive experience - Expands possibilities for user interaction	- Requires specialized hardware (e.g., VR headsets) - Can cause motion sickness in some users
Machine Learning for NPCs and Content	More personalized gaming experiences - Can automate procedural content generation	- Requires large datasets and computational power - Difficult to predict behavior in complex scenarios

Table 3 summarizes the advantages and disadvantages of several advanced technologies shaping the future of game development.

8.2 Asset management

Game engines offer robust asset management systems for organizing, importing, and processing game assets such as 3D models, textures, animations, and audio files. These systems include tools for asset versioning, dependency tracking, and automated processing, ensuring efficient and consistent asset handling. They support industry-standard file formats and seamless integration with various content creation tools.

8.3. Level editing and world-building

Game engines provide powerful level editors and world-building tools for creating game environments, placing objects, and defining gameplay mechanics. These tools feature real-time previewing, terrain sculpting, object placement, lighting setup, and scripting integration, allowing developers to visualize and iterate on their designs effectively.

V. Conclusions

In conclusion, the game engine industry has made significant strides by integrating cutting-edge technologies that enhance game development and deliver immersive experiences. Recent advancements include improvements in rendering, AI, cloud-based technologies, and VR/AR capabilities. Looking forward, future developments are expected to focus on real-time ray tracing, machine learning integration, cloud solutions, and cross-platform development. Innovations such as enhanced VR/AR experiences, novel input methods, and continuous technological evolution will continue to transform interactive entertainment, allowing developers to create more sophisticated gaming experiences.

This paper provides a valuable comparative analysis of modern game engines, offering developers and researchers insights into the most suitable tools for their needs. By highlighting current trends such as real-time ray tracing, AI, and VR/AR, it sheds light on the future direction of game development. However, the study is limited by its reliance on existing documentation, which may not fully capture proprietary features, and the rapidly changing landscape of game engine technologies. Future research could focus on detailed case studies or performance benchmarks to further enrich this analysis.

ACKNOWLEDGEMENT

"This work was supported by the 2024 education, research and student guidance grant funded by Jeju National University"

REFERENCES

[1] Singh S, Kaur A. Game Development using Unity Game Engine. In2022 3rd International Conference on Computing,Analytics and Networks (ICAN) 2022 Nov 18 (pp. 1-6). IEEE . DOI: 10.1109/ICAN56228.2022.10007155

[2] Creighton RH. Unity 3D game development by example: A Seat-of-your-pants manual for building fun, groovy little games quickly. Packt Publishing Ltd; 2010 Sep 24. https://dl.acm.org/doi/abs/10.5555/1941162

[3] Ahamed S, Das A, Tanjib SM, Eity QN. Study of an application development environment based on unity game engine. AIRCC's International Journal of Computer Science and Information Technology. 2020 Feb 1:43-62. DOI: 10.5121/ijcsit.2020.12103

[4] Torres-Ferreyros CM, Festini-Wendorff MA, Shiguihara-Juárez PN. Developing a videogame using unreal engine based on a four stages methodology. In2016 IEEE ANDESCON 2016 Oct 19 (pp. 1-4). IEEE. 10.1109/ANDESCON.2016.7836249

[5] Christopoulou E, Xinogalos S. Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. International Journal of Serious Games. 2017 Dec 1;4(4). DOI: https://doi.org/10.17083/ijsg.v4i4.194

[6] Johnson C. Learning Basic Programming Concepts with Game Maker. International Journal of Computer Science Education in Schools. 2017 May;1(2):n2. DOI: 10.21585/ijcses.v1i2.5

[7] Cossu SM. Game Development With Game Maker Studio 2. London: Apress Media LLC. 2019. https://dl.acm.org/doi/abs/10.5555/3378992

[8] Vohera, C., Chheda, H., Chouhan, D., Desai, A. and Jain, V., 2021, July. Game engine architecture and comparative study of different game engines. In 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE. DOI: 10.1109/ICCCNT51525.2021.9579618

[9] Maggiorini D, Ripamonti LA, Zanon E, Bujari A, Palazzi CE. SMASH: A distributed game engine architecture. In2016 IEEE Symposium on Computers and Communication (ISCC) 2016 Jun 27 (pp. 196-201). IEEE. DOI: 10.1109/ISCC.2016.7543739

[10] Zarrad A. Game engine solutions. Simulation and Gaming. 2018 Feb 14:75-87. https://dl.acm.org/doi/abs/10.5555/2621961

[11] Gregory J. Game engine architecture. AK Peters/CRC Press; 2018 Jul 20. https://dl.acm.org/doi/abs/10.5555/2621961

[12] Wang, L., Shi, X. and Liu, Y., 2023. Foveated rendering: A state-of-the-art survey. Computational Visual Media, 9(2), pp.195-228. DOI: doi.org/10.1007/s41095-022-0306-4

[13] Westera W, Van der Vegt W, Bahreini K, Dascalu M, Van Lankveld G. Software components for serious game development. In10th European Conf. on Games Based Learning 2016 Oct 6 (pp. 765-772). Reading UK, Paisley, Scotland.

[14] Deng Y, Ni Y, Li Z, Mu S, Zhang W. Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques. ACM Computing Surveys (CSUR). 2017 Aug 30;50(4):1-41. DOI: https://doi.org/10.1145/3104067

[15] Shao K, Tang Z, Zhu Y, Li N, Zhao D. A survey of deep reinforcement learning in video games. arXiv preprint arXiv:1912.10944. 2019 Dec 23 DOI 10.48550/arXiv.1912.10944

[16] Joshi A, Sharma M, Al Zubi J. Artificial Intelligence in Games: Transforming the Gaming Skills. InDeep Learning in Gaming and Animations 2021 Dec 7 (pp. 103-122). CRC Press. eBook ISBN9781003231530.

[17] Westera W, Prada R, Mascarenhas S, Santos PA, Dias J, Guimarães M, Georgiadis K, Nyamsuren E, Bahreini K, Yumak Z, Christyowidiasmoro C. Artificial intelligence moving serious gaming: Presenting reusable game AI components. Education and Information Technologies. 2020 Jan;25:351-80. DOI https://doi.org/10.1007/s10639-019-09968-2

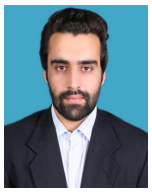
[18] Roberts P, editor. Game AI Uncovered: Volume One. CRC Press; 2024 Feb 23. DOI: 10.1201/9781003323549

[19] Schijven MP, Kikkawa T. Is there any (artificial) intelligence in gaming? Simulation & Gaming. 2022 Aug;53(4):315-6. DOI: https://doi.org/10.1177/10468781221101685

[20] Andrejkovics Z. Exploring the Potential Implications of Artificial Intelligence in Esports and related Video Games. International Journal of eSports Studies. 2023 Jun;1:1-8. DOI : 10.59924/IJeSS.2023.1.1

Authors

Soo Kyun Kim received Ph.D. in Computer Science & Engineering Department of Korea University, Seoul, Korea, in 2006. He joined the Telecommunication R&D center at Samsung Electronics Co., Ltd., from 2006 Game Engineering at Paichai University until 2020. He is now a professor at the Department of Computer Engineering at Jeju National University, Korea.



Iqbal Muhamad Ali is currently pursuing a PhD in Computer Engineering at Jeju National University, South Korea. He holds a Masters in Computer Science from COMSATS University Islamabad, Pakistan, and has lectured at both COMSATS University Islamabad and Hazara University, Mansehra. His research focuses on Machine Learning, Digital Image Processing, and Computer Vision.

Min Woo Ha received Ph.D. in Pharmaceutical Synthetic Chemistry from Seoul National University, College of Pharmacy in Korea. She is an associate professor at the College of Pharmacy, Jeju National University, Korea. As a member of PSK (The Pharmaceutical Society of Korea), KSOS (The Korean Society of Organic Synthesis), and KSIEC (The Korean Society of Industrial and Engineering Chemistry), she is expanding her research scope into the fields of pharmaceuticals and engineering chemistry through the design and preparation of potential organic materials.