International Journal of Computer Science Engineering Techniques (IJCSE) Volume 10 Issue 3 ,May-June 2026

ISSN : 2455-135X

Image Processing Techniques in Real-Time Rendering: Shader-Based Visual Storytelling in Video Games

Akshay Ananth, E. Raghuvaran, S. Dhivya

Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Tamil Nadu, India

akshay.ananth754@gmail.com

Received: Revised: Accepted: Published:

Abstract - *This paper presents the design and implementation of a real-time Unreal Engine game prototype that uses image-processing shaders as visual storytelling tools. The work combines a modular gameplay architecture with non-photorealistic rendering techniques to create a playable third-person action experience. Core character systems, including a damageable interface, damage system component, and Gameplay Ability System support, are implemented in C++ and inherited through a common MainCharacterBase. Higher-level gameplay logic, including common enemy encounters, boss progression, hallway sections, triggers, spawners, and blockers, is implemented using Unreal Engine Blueprints for faster iteration. The visual module uses post-process materials applied through Post Process Volumes to implement cel shading, retro dithering, and toon outline effects. Levels are constructed using Fab assets and Unreal Engine lighting, with draw-distance optimization applied to reduce unnecessary rendering cost. The prototype demonstrates that shader-based stylization can function as more than a visual filter by contributing to atmosphere, level identity, and narrative expression while preserving interactive gameplay.*

Keywords - *game development, image processing, non-photorealistic rendering, real-time rendering, Unreal Engine*

International Journal of Computer Science Engineering Techniques (IJCSE)

1. Introduction

The video game industry increasingly depends on real-time rendering pipelines that must balance visual fidelity, interactivity, and hardware constraints. Modern engines provide advanced features for lighting, asset rendering, animation, scripting, and post-processing, but effective implementation still requires careful system design. The present work explores this problem through the development of a third-person Unreal Engine action game in which stylized image-processing shaders are used as part of the visual and narrative structure of the game.

The central idea of the project is that shaders can be treated not only as decorative effects but also as expressive systems. Instead of using purely photorealistic rendering, the prototype applies non-photorealistic rendering methods such as cel shading, dithering, and toon outlining to shape how the player perceives each level. These visual changes are integrated with gameplay progression, dialogue, user interface prompts, and level design so that the appearance of the world supports the intended mood and identity of the experience.

The project uses Unreal Engine as the development platform because it provides an integrated environment for C++, Blueprints, post-process materials, animation montages, enhanced input, lighting, and real-time testing. The implementation follows a hybrid approach: reusable character systems are developed in C++, while encounter logic and level-specific gameplay flow are implemented through Blueprints. This division allows the project to maintain a modular software structure while still supporting fast iteration during level design and testing.

2. Related Work

Non-photorealistic rendering has been studied as a way to communicate shape, structure, and meaning without depending on realistic lighting alone. Technical illustration and stylized rendering research show that abstraction, outlines, quantized lighting, and texture-based representation can improve clarity or create a specific artistic effect. These ideas are especially relevant to games because the player must constantly interpret characters, spaces, hazards, and narrative cues in real time.

Image-processing techniques such as filtering, edge detection, and dithering are also relevant to the project. When applied to a rendered frame, these techniques modify the final image rather than individual object materials. This makes them suitable for post-process effects in Unreal Engine. A single full-screen material can alter the appearance of the entire level, which reduces the need to create separate stylized materials for every object in the world.

ISSN : 2455-135X

Real-time stylized rendering work has also shown that cartoon-like visual styles, silhouettes, and simplified lighting can be used in interactive 3D environments without removing player control. This project builds on that idea by implementing shaders as full-screen post-process materials and connecting them to level identity and game storytelling.

3. Materials and Methods

The proposed system is organized into two primary modules: a gameplay module and a visual module. The gameplay module handles the player character, enemy artificial intelligence, damage events, abilities, encounter triggers, spawners, blockers, and level progression. The visual module handles post-process materials, rendering configuration, lighting, and the shader-based narrative presentation of the world. Figure 1 shows the high-level architecture of the system.

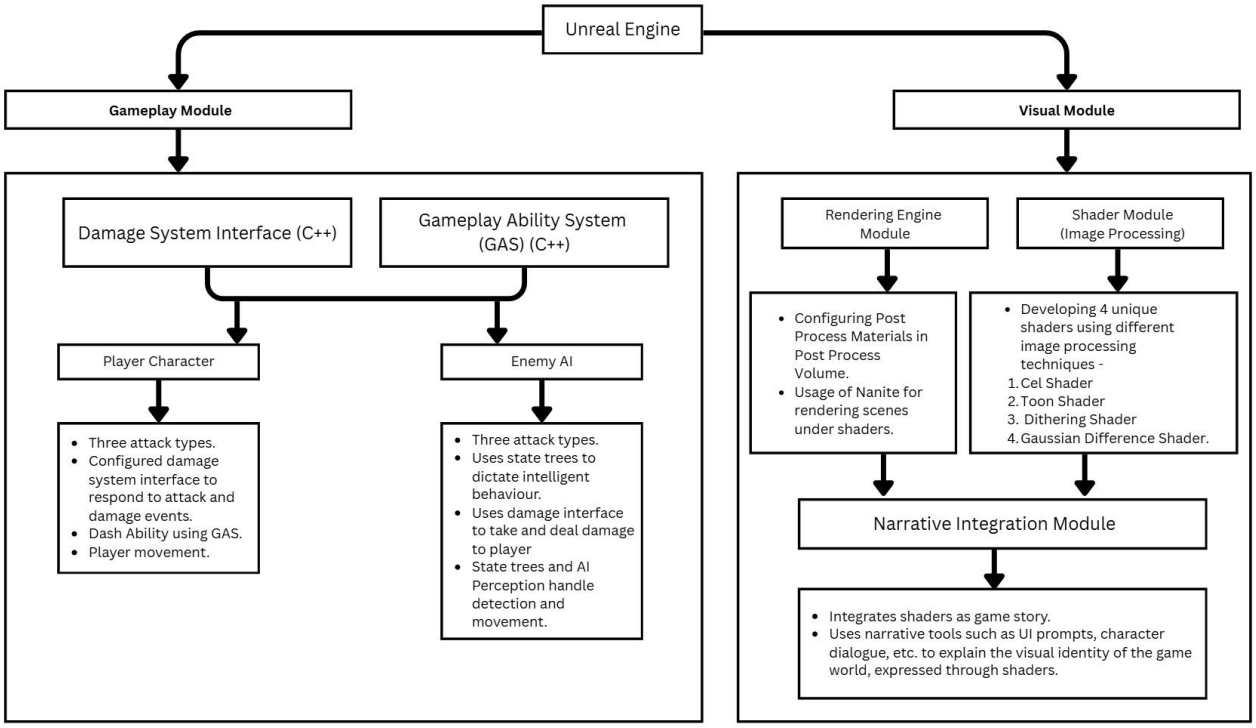

Fig. 1 High-level system architecture of the Unreal Engine prototype

3.1 Character Architecture

The character architecture is based on a common MainCharacterBase class. The player character and enemy characters inherit from this base class, allowing them to share the same damage and ability-related structure. The base class contains the damageable interface, the damage system component, and the required Gameplay Ability System configuration. This avoids duplication and ensures that all major characters can receive damage, respond to hits, heal, and trigger death events through a common pipeline.

The damageable interface defines functions such as Take Damage, On Damage Taken, On Death, and On Heal. The interface acts as a contract that identifies whether an actor can participate in combat interactions. When an attack trace hits an actor, the attacking system does not need to know whether the target is a player, common enemy, or boss. It only checks whether the target implements the interface and then calls the relevant damage function.

The damage system component stores and manages health-related values such as current health and starting or maximum health. When damage is applied, the component reduces health, checks whether the character is alive, and triggers the appropriate response. If the character survives, damage feedback such as hit reaction or UI update can be executed. If health reaches zero, the death event is called and the relevant gameplay controller can be notified.

The Gameplay Ability System is used in a focused manner. Rather than implementing the entire combat system through GAS, the project uses it primarily for the player dodge or dash ability. This gives the dash a structured ability framework while keeping the rest of the combat logic easier to implement and debug through interfaces, components, and Blueprints.

ISSN : 2455-135X

3.2 Blueprint Gameplay Logic

Gameplay progression is implemented entirely in Blueprints because encounter flow, trigger placement, spawner configuration, and level transitions require frequent iteration. The main Blueprint systems are common enemy spawning and progression, boss spawning and progression, and the hallway section system.

For common enemy encounters, three separate actors are used: a trigger, a spawner, and a progress blocker. The trigger detects when the player enters a combat space. Once activated, it calls one or more spawners to create enemy actors at selected points in the level. The progress blocker prevents the player from leaving the area until the encounter has been cleared. Each spawned enemy reports its death through the damage system, and the encounter checks whether all active enemies have been defeated. Once the condition is satisfied, the blocker is removed, often with a visual effect and sound cue.

Boss encounters follow a similar structure but include additional presentation elements. A boss trigger activates a boss spawner, displays the boss name and health bar, and plays spawn visual effects. Instead of opening a normal blocker when the boss dies, the system spawns the level transition gateway. This makes the boss encounter the final progression requirement for that section of the game.

The hallway implementation divides the game into sequential combat sections. Each section contains its own triggers, spawners, blockers, and references to the next section. This structure controls pacing by ensuring that only the current encounter is active. It also improves performance because enemies are spawned only when needed instead of being active throughout the entire level.

3.3 Shader Module

The shader module is implemented using Unreal Engine post-process materials. A post-process material is applied after the scene has been rendered, meaning it modifies the final image viewed by the camera. This is suitable for full-screen stylization because the same shader can affect the whole level without manually changing the material of every mesh.

The cel shader reduces smooth lighting transitions into discrete tonal bands. This creates a graphic non-photorealistic appearance and helps convert the 3D scene into a flatter, illustration-like image. The method supports readability and gives the level a distinct visual style.

The retro dithering shader applies a patterned threshold effect that simulates intermediate tones through pixel patterns. This produces a digital or retro visual identity and emphasizes the idea that the game world is being interpreted through an image-processing layer.

The toon outline shader detects strong differences in scene depth, surface normals, or object boundaries and darkens those edge regions. The resulting outlines make characters and environmental shapes more readable and contribute to a cartoon-like visual language. Together, the shaders support the project objective of using image processing as a narrative and stylistic tool.

3.4 World Design, Lighting, Animation and Input

The game levels were constructed in Unreal Engine using 3D assets from Fab. These assets were arranged manually to create hallways, combat spaces, transition zones, and boss arenas. The layout was designed around the gameplay systems so that the player could enter an area, trigger enemies, clear the encounter, and then progress to the next section.

Unreal Engine lighting was used to define the atmosphere of each scene. Lighting also affects how the post-process shaders appear, especially in cel-shaded and dithered scenes where tonal contrast is important. To reduce rendering cost, lights were optimized using draw-distance or attenuation settings so that they render only when the player is close enough. This prevents distant lights from consuming resources unnecessarily.

Animations were sourced from Unreal Engine template files and integrated through animation Blueprints, animation montages, and animation notifies. Montages are used for attacks, while notifies mark the timing of gameplay events such as enabling damage traces, playing effects, or closing an attack window. This ensures that damage is applied only during the active portion of the animation.

User input is handled using Unreal Engine Enhanced Input. Input actions were configured for movement, camera control, attacks, and dash. Keyboard, mouse, and controller mappings can call the same gameplay actions, allowing the player to interact with the game through different input devices without duplicating the underlying logic.

4. Results and Discussion

The completed prototype demonstrates a functional integration of gameplay systems and shader-based rendering. The C++ character framework provides a reusable foundation for player, enemy, and boss actors. Since damage logic is centralized through the interface and component, the same attack pipeline can be used across multiple character types. This reduces implementation duplication and supports future expansion.

The Blueprint encounter systems successfully organize the gameplay loop into clear stages: enter a section, trigger an encounter, defeat enemies, unlock progression, and advance to the next area. The boss system extends this structure by adding health bar presentation, visual effects, and a level gateway after defeat. The hallway system further improves pacing by dividing the level into smaller controlled combat spaces.

The shader module shows that full-screen post-process materials are an efficient way to apply stylized rendering across a level. The cel shader, dithering shader, and toon outline shader each create a different visual identity while

ISSN : 2455-135X

still allowing the game to remain interactive. These effects make the visual module central to the experience rather than a purely cosmetic layer.

Performance testing was conducted on a laptop with an AMD Ryzen 7 5800HS CPU, Nvidia RTX 3050 Laptop GPU with 4 GB VRAM, 16 GB RAM, and SSD storage. In-editor testing on medium settings produced generally playable performance around 60 FPS with occasional drops in complex scenes. Standalone testing at high render quality produced lower performance, with the peak observed at approximately 34 FPS and heavier scenes dropping significantly. This indicates that the prototype is playable under reduced or medium settings, but further optimization is required for consistent standalone performance on lower VRAM systems.

5. Conclusion

This paper presented a real-time Unreal Engine game prototype that combines modular gameplay systems with shader-based visual storytelling. The implementation uses C++ for reusable character systems and Blueprints for level-specific gameplay logic. The damageable interface, damage system component, and MainCharacterBase structure allow the player, enemies, and bosses to share a consistent combat pipeline. GAS is used specifically for the player dash ability, keeping the ability system focused and manageable.

The visual contribution of the project lies in the use of post-process materials for non-photorealistic rendering. Cel shading, retro dithering, and toon outlining are applied at the scene level through Post Process Volumes. These shaders provide distinct visual identities and demonstrate how image-processing techniques can contribute to mood, readability, and narrative expression in games.

The project also demonstrates the value of modular encounter systems. Triggers, spawners, blockers, boss health bars, and level gateways work together to create a complete gameplay loop. The result is a playable prototype that integrates gameplay, rendering, animation, input, and level design into a unified system.

6. Limitations and Future Work

The current implementation has several limitations. The Gameplay Ability System is used only for the player dash ability, while attacks and other combat events are handled through custom systems. Future versions could expand GAS to include attacks, cooldowns, buffs, debuffs, enemy skills, and boss abilities. The damage system could also be expanded with armour, stagger values, elemental damage, invincibility frames, and damage scaling.

The AI system can be improved with group coordination, tactical spacing, retreat logic, adaptive attack selection, and multi-phase boss behaviour. The world design currently depends on third-party assets from Fab, so future versions could include custom models, textures, environmental storytelling, puzzles, and more interactive level elements.

The shader system can also be extended. Future work may include Kuwahara filtering, Gaussian difference effects, glitch distortion, chromatic aberration, pixel sorting, or health-based visual degradation. These effects could be dynamically controlled during story events or boss phases to strengthen the relationship between gameplay and visual storytelling. Additional profiling, shader optimization, level streaming, asset LOD configuration, and Nanite tuning would also be required for a more polished standalone release.

Conflicts of Interest

The author declares that there is no conflict of interest regarding the publication of this paper.

Funding Statement

This research received no external funding.

Acknowledgments

The author expresses sincere gratitude to the Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, for academic support and guidance during the development of this project.

References

[1] T. He, Y. Zhong, P. Isenberg, and T. Isenberg, “Design Characterization for Black-and-White Textures in Visualization,” IEEE Transactions on Visualization and Computer Graphics, vol. 30, no. 1, pp. 1019–1029, Jan. 2024.

[2] A. Gooch, B. Gooch, P. Shirley, and E. Cohen, “A Non-Photorealistic Lighting Model for Automatic Technical Illustration,” Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, pp. 447–452, 1998.

[3] A. Hertzmann, “A Survey of Stroke-Based Rendering,” IEEE Computer Graphics and Applications, vol. 23, no. 4, pp. 70–81, Jul.–Aug. 2003.

[4] D. DeCarlo and A. Santella, “Stylization and Abstraction of Photographs,” ACM Transactions on Graphics, vol. 21, no. 3, pp. 769–776, 2002.

[5] A. Lake, C. Marshall, M. Harris, and M. Blackstein, “Stylized Rendering Techniques for Scalable Real-Time 3D Animation,” Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering, pp. 13–20, 2000.

[6] M. Pharr, W. Jakob, and G. Humphreys, Physically Based Rendering: From Theory to Implementation, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2016.

[7] B. Gooch and A. Gooch, Non-Photorealistic Rendering. Natick, MA, USA: A K Peters, 2001.

[8] T. Akenine-Möller, E. Haines, and N. Hoffman, Real-Time Rendering, 4th ed. Boca Raton, FL, USA: CRC Press, 2018.