Design and Development of a Low-Poly First-Person Psychological Horror Game Using Unreal Engine 5

R. Bavana Mercy[1], U. SasthaRuban[2], S. Saravana Kumar[3], S. Denis Kabilan[4]
[1]*Assistant Professor, Dept. of Computer Science Engineering, Mohamed Sathak Engineering College*
[2,3,4]*Student, Dept. of Computer Science Engineering, Mohamed Sathak Engineering College*

***Abstract*— This paper presents the design and development of a low-poly first-person psychological horror game using Unreal Engine 5 and Blender. Unlike conventional games that rely on high-end graphical realism, this project focuses on system-level design, event-driven architecture, and performance optimization. A retro 1990s-inspired visual style is achieved by implementing a custom rendering pipeline that includes vertex snapping, color banding, and reduced texture resolution. The system integrates modular player controls, inverse kinematics for character animation, and real-time interaction using Blueprint scripting. The developed game demonstrates how constrained visual design combined with efficient system architecture can produce an immersive and interactive experience. The results highlight stable performance, efficient resource usage, and effective gameplay mechanics.**

***Index Terms*— Unreal Engine, Game Development, Low-Poly Rendering, Event-Driven Architecture, Inverse Kinematics, Blueprint Scripting.**

I. INTRODUCTION

Modern game engines are designed to produce highly realistic visuals [1] using advanced rendering techniques. However, such realism often abstracts the underlying system mechanics that drive game logic and performance. This project takes a different approach by intentionally adopting a retro low-poly visual style inspired by 1990s gaming hardware.

By limiting graphical complexity, the focus shifts toward core system design, event-driven programming, and performance optimization. The developed game is a first-person psychological horror experience where progression is driven by player interaction rather than passive storytelling.

The project aims to bridge the gap between classic hardware constraints and modern game engine capabilities by implementing custom rendering techniques and efficient system architecture. This approach allows for a deeper understanding of real-time interactive systems while delivering an immersive gameplay experience.

II. LITERATURE REVIEW

Before developing the system, an analysis was conducted on early 3D rendering techniques and modern game engine architectures

A. Historical Hardware Limitations

Early gaming consoles from the mid-1990s had strict hardware limitations [2] limitations, including low-precision vertex processing and limited color palettes. These limitations often caused graphical artifacts such as vertex jittering. However, such imperfections contributed to the unique atmosphere of early psychological horror games [6].

B. Modern Game Engine Architecture

Today, engines like Unreal Engine calculate physics, lighting, and rendering with extreme precision [3] The challenge in using modern engines for a retro game is that their default settings (like Ray Tracing and Global Illumination) automatically "fix" the visual errors we actually want to keep. Our research focused on how to safely downgrade the rendering pipeline using DirectX 11 and custom code to bring back those classic visual constraints without breaking the engine's core performance.

III. SYSTEM DESIGN AND METHODOLOGY

To build the game, we divided the project into several independent modules. This modular approach makes the code cleaner [9] and easier to debug.

A. System Workflow

The system follows a modular workflow beginning with player input, followed by interaction processing, event triggering, and rendering output. This structured flow ensures efficient handling of gameplay logic and system resources.

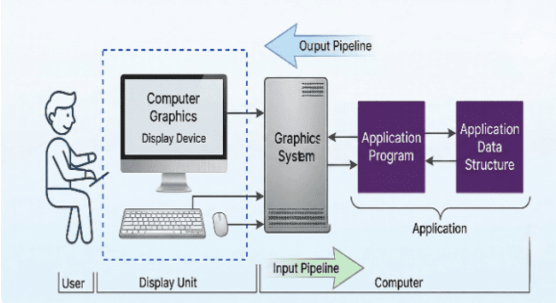

Fig. 1. System Workflow of the Proposed Game

B. Rendering Pipeline and Visual Aesthetics

To recreate the retro visual style, advanced rendering features were disabled [8] and replaced with custom material-based techniques. A vertex snapping method was implemented to simulate low-precision rendering, where vertex positions are forced to align with a virtual grid.

The mathematical representation is given as [7]:

$$NewPosition = \left\lfloor \frac{OriginalPosition}{GridSize} \right\rfloor \times GridSize$$

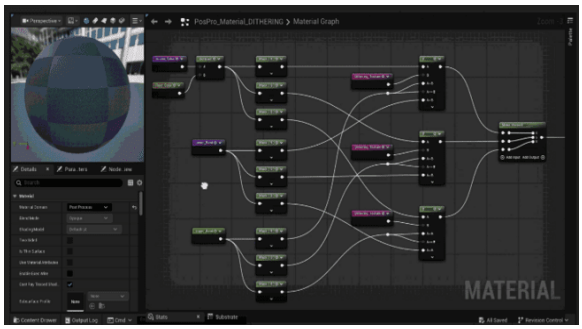

Fig. 2. Custom Rendering Pipeline with Vertex Snapping)

C. Asset Engineering in Blender

All 3D models were built with a strict polygon budget. For example, our main character's head was kept under 300 polygons. Instead of adding 3D details like wrinkles or facial features, we painted those details onto very small textures (128x128 pixels). We spent significant time mapping the 2D textures perfectly to the 3D models to ensure the pixels remained perfectly square and didn't stretch.

TABLE 1 RENDERING AND ASSET CONSTRAINTS

Rendering And Asset Optimization Constraints		
System Component	*Standard Modern Engine Default*	*Custom PS1 Implementation*
Character Mesh	> 10,000 Polygons	< 300 Polygons
Texture Resolution	2048 x 2048 pixels (2K)	128 x 128 pixels
Lighting Pipeline	Global Illumination / Ray Tracing	Disabled (Flat / Baked Lighting)
Geometry Rendering	High-Precision Floating Point	Real-Time Vertex Snapping (Grid)
Interaction Logic	Continuous "Event Tick"	Event-Driven (Line Trace)

D. Character Rigging and Inverse Kinematics

Animation requires a control system called a rig. Instead of animating every single bone manually (Forward Kinematics), we used a system called Inverse Kinematics (IK) [4] . By setting up IK constraints and pole targets in Blender we only need to move a single target like the character's hand and the math automatically calculates how the elbow and shoulder should bend.

We also kept the control bones completely separate from the bones that actually move the 3D mesh, ensuring our animation logic remained organized

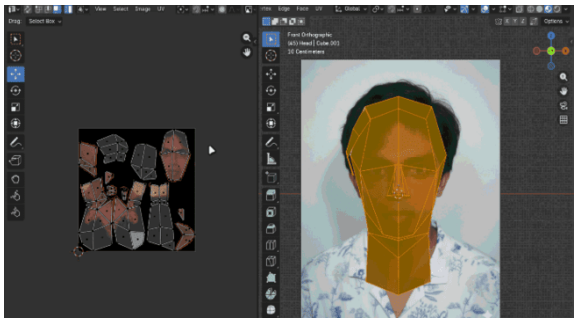

Fig.3. UV mapping low-poly character model

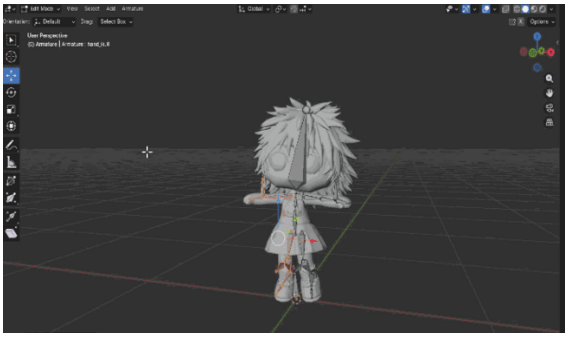

Fig.4. Skeletal hierarchy and Inverse Kinematics (IK) control rig engineered for the low-poly character model

IV. IMPLEMENTATION DETAILS

The programming of the game was done using Unreal Engine's Blueprint visual scripting system, focusing heavily on math and event-driven logic.

A. System Architecture

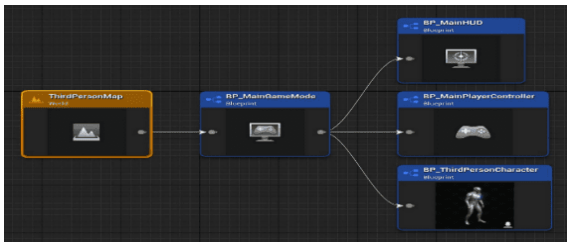

Fig. 5. Modular System Architecture

B. Event-Driven Architecture

The system uses an event-driven architecture [5] to improve performance. Instead of continuous execution using frame updates, interactions are triggered only when necessary. A raycasting mechanism detects player interactions and activates corresponding system responses.

C. Environmental Mathematics

Vector mathematics is used to create dynamic interactions. For example, door mechanics are implemented using dot product calculations to determine player orientation and control movement direction dynamically.

D. Player Control Systems

A state-based control system was implemented to manage player actions such as walking, sprinting, crouching, and interaction [10]. This prevents conflicting actions and ensures smooth gameplay transitions.

V. RESULTS AND DISCUSSION

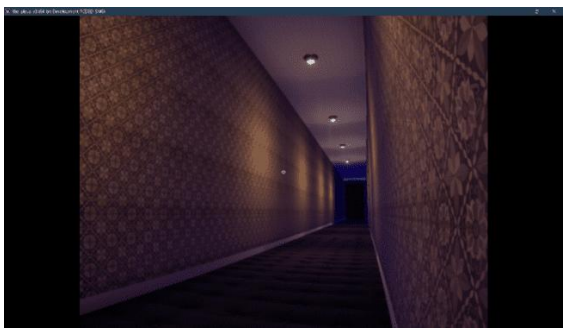

Fig. 6. Gameplay Environment and Interaction

The developed system demonstrates stable performance and smooth gameplay. The use of event-driven architecture reduces computational overhead, while the custom rendering pipeline successfully recreates a retro visual style.

User experience evaluation indicates that the combination of lighting, sound, and interaction effectively creates a psychological horror atmosphere.

VI. CONCLUSION

This project demonstrates the successful development of a low-poly first-person psychological horror game using modern game development tools. By focusing on system-level optimization and modular design, the project provides insights into real-time interactive system development.

The results confirm that immersive experiences can be achieved through efficient system design rather than reliance on high-end graphics. Future improvements may include advanced AI behavior and expanded gameplay features.

VII. ACKNOWLEDGMENT

We would like to express our sincere gratitude to our supervisor, Mrs. R. Bavana Mercy, for her continuous

guidance, support, and valuable feedback throughout the development of this project.

REFERENCES

[1] Epic Games, "Unreal Engine 5 Documentation: Materials and Post Process Effects," Epic Games Developer Network, 2023. [Online]. Available: https://docs.unrealengine.com.
[2] Blender Foundation, "Blender 4.0 Reference Manual: Armatures and InverseKinematics,"2023.[Online].Available: https://docs.blender.org.
[3] Sony Computer Entertainment, "PlayStation Hardware Specifications and Rendering Architecture," SCE Technical Documentation, 1995.
[4] J. Gregory, *Game Engine Architecture*, 3rd ed. Boca Raton, FL: CRC Press, 2018, pp. 115-142.
[5] R. Nystrom, *Game Programming Patterns*. Genever Benning, 2014, pp. 65-85.
[6] T. Perron, *Silent Hill: The Terror Engine*. Ann Arbor, MI: University of Michigan Press, 2012.
[7] F. D. Dunn and I. Parberry, *3D Math Primer for Graphics and Game Development*, 2nd ed. Boca Raton, FL: CRC Press, 2011.
[8] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*, 4th ed. Boca Raton, FL: CRC Press, 2018.
[9] B. McShaffry and D. Graham, *Game Coding Complete*, 4th ed. Boston, MA: Course Technology PTR, 2012.
[10] I. Millington, *AI for Games*, 3rd ed. Boca Raton, FL: CRC Press, 2019, pp. 315-330.