iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.74992

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

Mobile VR Experiences Virtual House Tour

Kumaran G L[1], Kamalnath V[2], Mrs. Meena V[3]
[1,2]*Bachelor of Engineering in Computer Science and Engineering Adhiyamaan College of Engineering, ANNA University, Chennai*
[3]*M.E., Assistant Professor, Computer Science and Engineering Adhiyamaan College of Engineering, ANNA University, Chennai*

Abstract: ***The project "Mobile VR Experiences: Virtual House Tour" focuses on enhancing architectural visualization through an accessible, mobile-based Virtual Reality (VR) solution. Traditional visualization methods, such as blueprints and static renders, often fail to convey spatial scale, lighting, and atmosphere, while high-end VR systems remain costly. This project addresses these challenges by developing a realistic and immersive virtual house tour application using Unity integrated with the Google Cardboard SDK, optimized for standard smartphones. Detailed 3D models of structures and interiors were created in Blender, ensuring both aesthetic quality and architectural accuracy. The system provides intuitive navigation through gaze-based interaction and Bluetooth keyboard controls for user comfort. Performance testing confirmed low latency, high frame rates , and efficient resource utilization. The final application offers a cost-effective, high-performance visualization tool that democratizes immersive architectural experiences for designers, clients, and builders alike.***

Keywords: Mobile VR, Virtual House Tour, Architectural Visualization, Bluetooth Navigation, Cost-Effective Visualization, Google Cardboard SDK, Unity 3D, Blender, Unity Profiler.

I. INTRODUCTION

A. *Overview*

The field of architectural design and real estate visualization has historically relied upon blueprints, static rendered images, and physical scale models to convey the final appearance of a structure. While effective, these traditional methods often fail to transmit the true sense of scale, light, and atmosphere of a planned home. The advent of Virtual Reality (VR) offers a compelling solution, yet professional VR equipment remains costly and limits user access. This project, titled "Mobile VR Experiences: Virtual House Tour," seeks to democratize this visualization technology by implementing a robust, high-performance VR solution specifically tailored for readily available mobile devices. By focusing on mobile accessibility, the project ensures that sophisticated, immersive architectural visualization is no longer restricted by high hardware costs, allowing any individual with a smartphone to participate in the experience.

B. *Objective*

The main Objective of the project is

1) Develop a Highly Realistic and Immersive Mobile VR Application: The main goal was to create a Virtual House Tour application using the Unity platform that provides a sense of presence and realism comparable to high-end rendering solutions, all while running efficiently on mobile hardware.
2) Create High-Fidelity 3D Assets: To ensure architectural accuracy, a specific objective was set to generate all necessary three-dimensional models, including the detailed floor plans, structural elements, and realistic furniture models, using professional tools like Blender.
3) Deliver a Cost-Effective Visualization Alternative: Ultimately, the project aimed to provide a practical and cost-effective visualization tool that can serve as a superior alternative to expensive physical models or traditional high-fidelity rendering services for some applications.
4) Provide Flexible and Intuitive Navigation: The application was designed to accommodate different user preferences and physical limitations by implementing a navigation system: static control via Bluetooth keyboard input for comfort.

II. LITERATURE SURVEY

This chapter is dedicated to the comprehensive review of existing works and foundational technologies that are directly relevant to the development and implementation of the Mobile VR Virtual House Tour. The survey focuses on the successful application of these tools in previous projects and the proven methodologies for achieving a high-quality, accessible mobile VR experience.

1) *R. K. McMahan, "Exploring the Role of Presence in Virtual Environments," Journal of Architectural Computing, vol. 25, no. 3, pp. 45–52, 2021.*

McMahan (2021) emphasizes the critical role of *presence* in virtual environments, noting that user engagement and spatial perception significantly improve when a high sense of "being there" is achieved. In the context of a mobile VR home tour, this concept guides the development of immersive visual and interaction elements that replicate a realistic architectural walkthrough.

2) *J. Whyte, "Virtual Reality and the Built Environment," Architectural Engineering and Design Management, vol. 18, no. 4, pp. 312–326, 2022.*

Whyte (2022) discusses the transformative impact of virtual reality within the built environment, emphasizing its ability to enhance design visualization and stakeholder communication. This insight is directly applicable to mobile VR home tours, which aim to make architectural experiences more accessible and comprehensible for non-specialist users such as home buyers.

3) *S. A. Diniz, "Mobile Virtual Reality for Architecture: Democratizing Immersion," Procedia Computer Science, vol. 196, pp. 240–248, 2022.*

Diniz (2022) introduces the concept of "democratizing immersion" through mobile VR, emphasizing how portable technologies enable broader access to architectural visualization. This idea underpins the rationale for the Mobile VR Home Tour project, which seeks to deliver an affordable yet immersive experience to users exploring home environments.

4) *T. Jones and K. Lee, "Smartphone-Based VR Systems: Design and Performance," IEEE Access, vol. 9, pp. 77645–77658, 2021.*

Jones and Lee (2021) analyze smartphone-based VR systems, emphasizing the importance of optimizing performance and user comfort within the constraints of mobile hardware. Their research provides essential guidance for the technical design of the Mobile VR Home Tour, particularly in achieving stable frame rates and accurate motion tracking.

5) *Philipp Maruhn, VR Pedestrian Simulator Studies at Home: Comparing Google Cardboards to Simulators in the Lab and Reality, Technical University of Munich Research Paper, 2021.*

Maruhn (2021) examined user experiences with mobile VR systems such as Google Cardboard, finding that these low-cost solutions can approximate the immersive quality of laboratory simulators while remaining highly accessible for home use. This supports the viability of deploying the Mobile VR Home Tour as a realistic yet affordable alternative for spatial exploration.

6) *P. Sousa et al., "Optimization Techniques for Real-Time Mobile VR Rendering," International Conference on Virtual Systems and Multimedia, 2021.*

Sousa *et al.* (2021) explored several optimization techniques to enhance real-time rendering performance in mobile VR environments, including adaptive resolution scaling and efficient shader management. Their findings are directly applicable to the Mobile VR Home Tour project, where maintaining consistent frame rates is critical to preserving immersion and user comfort.

C. *Lin and H. Hsu, "Reducing Motion Sickness in VR Through Controlled Locomotion," Human–Computer Interaction Journal, vol. 37, no. 2, pp. 95–106, 2023.*

Lin and Hsu (2023) examined motion sickness mitigation in VR systems, highlighting that controlled locomotion methods—such as teleportation and adaptive movement—significantly reduce discomfort and enhance user engagement. Their research informs the design of the Mobile VR Home Tour's navigation system, ensuring smooth and comfortable exploration of virtual spaces.

D. *A. Patel and M. Kumar, "Gaze-Based Interaction in Mobile Virtual Environments," ACM Transactions on Interactive Systems, vol. 12, no. 1, pp. 1–18, 2022.*

Patel and Kumar (2022) examined gaze-based interaction methods for mobile virtual environments, emphasizing their potential to enhance accessibility and immersion without requiring external controllers. Their findings directly inform the Mobile VR Home Tour project, where gaze-based navigation simplifies user interaction and supports a more intuitive exploration of virtual spaces.

III. REVIEW AND ANALYSIS

The analysis of existing visualization methods reveals several limitations that this project addresses. Traditional architectural

visualization, whether through flat blueprints or rendered video walkthroughs, often fails to convey the spatial feel of a room. High-end VR solutions, while effective, require specialized computers and headsets, creating a significant barrier to entry for individual consumers. This project is grounded in the strategic choice of tools and technology. Unity was selected due to its robust support for cross-platform development, particularly its excellent compatibility with the mobile VR ecosystem, leveraging the Google Cardboard SDK and the Unity XR Interaction Toolkit. The decision to use Blender for general 3D assets. This combination ensures that the models are both aesthetically refined (Blender for texturing and furniture) and geometrically precise (load-bearing structures and accurate floor plans, like those shown in the provided floor plan screenshot). By focusing the entire pipeline on highly optimized, mobile-first asset creation, the system avoids the common pitfalls of complex desktop VR applications attempting to run on limited mobile resources. The core analysis concluded that Mobile VR is the most practical and accessible pathway to achieving a high- quality pre-construction visualization experience.

IV. SYSTEM DESIGN AND ARCHITECTURE

The Mobile VR Application is designed to deliver an immersive virtual reality experience on mobile devices, leveraging Android as the primary platform. The architecture integrates Unity for rendering and input handling, with a focus on Bluetooth-enabled input devices (e.g., keyboards or controllers) to enhance user interaction. This system emphasizes modularity, separating concerns such as user interface, 3D rendering, physics simulation, and hardware integration. The design ensures efficient performance on resource-constrained mobile hardware, utilizing sensors, Bluetooth (BT), and GPU for real-time VR processing.

A. High Level Architecture

The system follows a layered architecture, promoting separation of concerns and reusability. It is built on top of the Android operating system, incorporating Google's VR SDK for stereoscopic rendering and sensor fusion. Unity serves as the core game engine for handling 3D environments, physics, and inputs.

Key principles guiding the design:

1) Modularity: Components are decoupled to allow independent development and testing (e.g., input handling separate from rendering).
2) Performance Optimization: Real-time processing for VR requires low-latency handling of inputs, physics, and graphics, optimized for mobile GPUs.
3) Extensibility: Support for Bluetooth inputs enables integration with external devices, while the scene manager allows dynamic object and environment loading.
4) Platform Dependency: Relies on Android SDK for OS-level services and Google's VR SDK for head-tracking and stereo output.

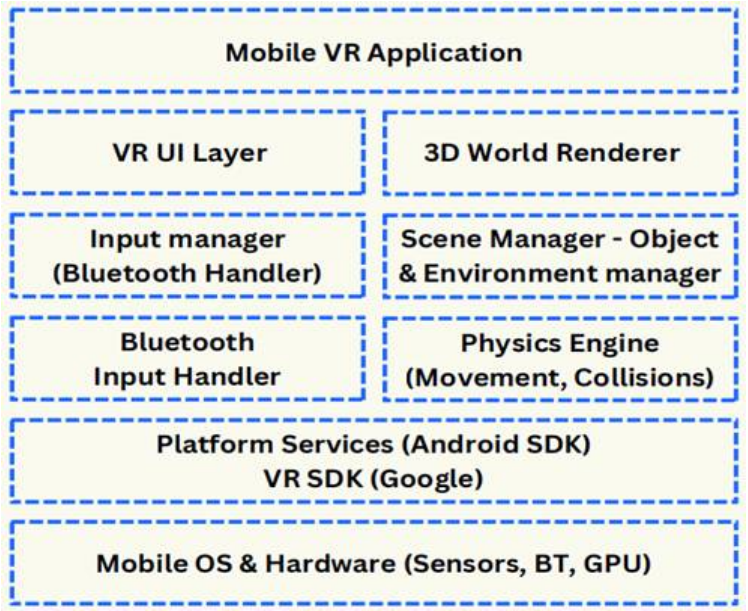


Fig 4.1: Core Module Architecture

a) Hardware & OS Layer (Mobile Device)
- Components: CPU, GPU, IMU sensors (accelerometer, gyroscope, magnetometer), Bluetooth radio, display/subsystem, battery, OS (Android).
- Responsibilities: provide raw sensor data, Bluetooth connectivity, graphics drivers, and native OS services.

b) Platform & VR SDK Layer
- Components: Android SDK (JNI where needed), VR SDK (Google VR / ARCore or Unity XR / OpenXR plugin).
- Responsibilities: abstraction of device and VR features (stereo projection, head pose), camera projection matrices, distortion correction, and lifecycle management.

c) Runtime & Middleware Layer (Unity Engine + Plugins)
- Components: Unity engine, XR plugin, native plugins (for advanced sensors or Bluetooth if native pairing required), physics engine (Unity PhysX or custom).
- Responsibilities: scene graph, rendering pipeline, update loop, physics simulation, native bridging.

d) System Services Layer
- Components: Input manager (Bluetooth handler), Scene Manager, Movement & Physics controller, UI manager, Audio manager.
- Responsibilities: interpret inputs, manage objects and scenes, apply physics and collision responses, and handle UI in VR.

e) Application Layer
- Components: game/experience logic, VR UI, interactions, content assets.
- Responsibilities: user interactions, gameplay or app flow, asset streaming, high-level orchestration.

B. Components and Interactions

The mobile VR system is composed of several modular components that work collaboratively to deliver immersive virtual experiences on a mobile device. Each component is responsible for specific tasks such as input handling, movement control, scene management, physics simulation, and rendering.

1) Bluetooth Input Handler

Purpose: Facilitates wireless communication between the mobile device and external input peripherals such as Bluetooth controllers or keyboards.

Responsibilities:
- Establishes and maintains Bluetooth connections.
- Receives raw key or control signals from the input device.
- Converts hardware-level input into standardized internal input events for use by higher-level modules.

Interactions:
- Communicates directly with the Mobile OS Bluetooth stack to receive device input.
- Sends processed input data to the Unity Input Handler for interpretation.
- Operates asynchronously to minimize latency during gameplay.

2) Unity Input Handler

Purpose: Acts as the intermediary between low-level hardware input and high-level application logic within the Unity engine.

Responsibilities:
- Translates raw Bluetooth or Android input into Unity-recognized events (e.g., movement, rotation, interaction).
- Maps input events to specific in-game actions through configurable bindings.
- Debounces noisy signals and applies smoothing for continuous inputs.

Interactions:
- Receives events from the Bluetooth Input Handler.
- Dispatches mapped actions to the Movement Controller, VR UI Layer, or Scene Manager.
- Works closely with the Unity event system to trigger interactive responses in VR.

3) Movement Controller

Purpose: Controls player or camera movement within the virtual environment based on user input.

Responsibilities:

- Converts action commands into spatial transformations (translation and rotation).
- Ensures motion adheres to environmental physics and collision boundaries.
- Provides different locomotion modes such as teleportation, smooth movement, and snap rotation.

Interactions:

- Receives action events from the Unity Input Handler.
- Updates the VR Camera Rig with new player coordinates and orientation.
- Relies on the Physics Engine for collision detection and movement constraints.

4) Scene Manager / Object Manager

Purpose: Manages all elements within the 3D scene, including environments, objects, and interactive entities.

Responsibilities:

- Loads and unloads virtual environments or levels dynamically.
- Handles object creation, destruction, and state management.
- Controls lighting, occlusion, and spatial organization of assets.

Interactions:

- Coordinates with the 3D World Renderer to render active scenes.
- Uses the Physics Engine for object interactions and environment dynamics.
- Responds to commands from the Movement Controller and UI Layer for environment transitions or updates.

5) Physics Engine

Purpose: Provides a realistic simulation of physical behaviours, enabling lifelike movement and object interactions in the VR environment.

Responsibilities:

- Handles rigid-body dynamics, collisions, and triggers.
- Simulates gravity, inertia, and force-based movements.
- Maintains real-time updates synchronized with the rendering loop.

Interactions:

- Works closely with the Scene Manager to apply physical rules to 3D objects.
- Updates positional data for the 3D Renderer to visualize correct movement.
- Informs the Movement Controller of collisions or movement restrictions.

6) 3D World Renderer & VR Camera Rig

Purpose: Generates and displays the stereoscopic visual output seen in the VR headset or viewer.

Responsibilities:

- Renders the virtual world for each eye to create depth perception.
- Manages shaders, lighting, and post-processing for visual realism.
- Integrates head-tracking data to update the user's viewpoint in real time.

Interactions:

- Receives camera updates from the Movement Controller and VR SDK.
- Fetches scene data from the Scene Manager for rendering.
- Sends the final stereo output to the VR display subsystem of the Android platform.

7) VR UI Layer

Purpose: Provides an interactive interface that allows users to engage with menus, HUDs, and control panels inside the VR environment.

Responsibilities:

- Renders 2D/3D UI elements at a comfortable visual distance.
- Supports gaze-based, gesture-based, or controller-based interaction.
- Manages context-sensitive UI updates (e.g., notifications, menus).

Interactions:

- Receives user selection and navigation events from the Unity Input Handler.
- Coordinates with the Scene Manager to trigger in-game events based on user actions.
- Utilizes rendering services from the 3D World Renderer for display.

8) Platform Services (Android SDK / VR SDK)

Purpose: Acts as the bridge between the Unity engine and the Android operating system, enabling access to hardware and VR functionalities.

Responsibilities:

- Provides APIs for Bluetooth, sensors, and graphics acceleration.
- Supplies VR SDK features like stereoscopic rendering, head tracking, and motion prediction.
- Manages lifecycle events (pause, resume, focus) and hardware resource allocation.

Interactions:

- Supports communication between the Unity Runtime and Mobile OS & Hardware.
- Feeds sensor and tracking data into the VR Camera Rig.
- Ensures optimized hardware utilization during rendering.

9) Mobile OS and Hardware

Purpose: Serves as the foundation for all operations by providing computational, sensory, and communication capabilities.

Responsibilities:

- Executes all application and rendering tasks using CPU and GPU.
- Provides IMU sensor data (gyroscope, accelerometer) for motion tracking.
- Handles Bluetooth connectivity and system-level power management.

Interactions:

- Delivers sensor and device input to the VR SDK and Bluetooth Handler.
- Executes low-level rendering commands from the 3D Renderer.
- Communicates final stereo frames to the display for the user's view.

10) Data Flow & Runtime Behavior

Device Boot & App Init

- Android loads the Unity app. VR SDK initializes XR subsystems and requests sensor access.
- Bluetooth subsystem starts scanning or reconnects to paired devices.

Input Handling

- Bluetooth device → Android Bluetooth stack → native or Unity plugin → Bluetooth Input Handler → Unity Input Handler.
- Optionally Android system keyboard/touch events follow the same path into Unity Input Handler.

Action Mapping

- Unity Input Handler maps events → action events (e.g., MOVE_FORWARD, TURN_LEFT).
- These events are dispatched to Movement Controller, UI Manager, or specific game objects.

Physics & Scene Update

- Movement Controller updates player transform and requests physics simulation.
- Physics Engine runs fixed-step updates, resolves collisions, triggers events (e.g., object pickups).

Rendering

- VR Camera Rig receives updated head pose from VR SDK + IMU fusion.

- Scene graph is culled and rendered per eye (or using single-pass stereo).
- Framebuffers processed and stereo output passed to device compositor or VR display.

Output

- Stereo frames are submitted to the VR display (headset's compositor or mobile display with lens correction).

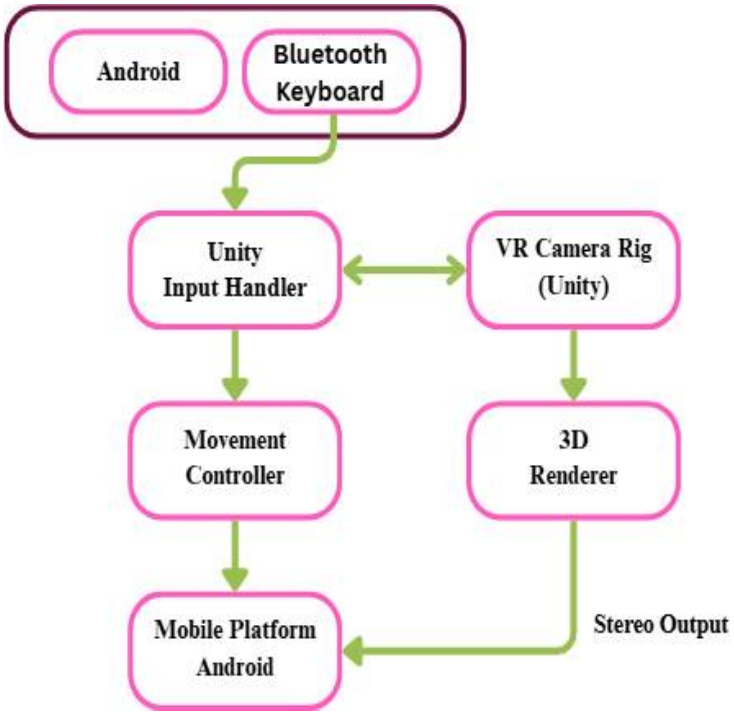


Fig 4.2: App Module Architecture

V. IMPLEMENTATION AND TECHNIQUES

A. Asset Creation and Integration Module

The implementation began with the creation of the virtual home itself. The exact dimensions and structural layout of the future house were modelled, resulting in precise floor plans and external geometry, as evidenced by the architectural drafts provided in the screenshots. Subsequently, Blender was used to add realism, creating detailed furniture, textures, and interior elements. A key technique in this module was asset optimization, where the polygon count and texture sizes of all models were carefully reduced to prevent strain on the mobile device's graphics capabilities. All models were then seamlessly integrated into the Unity project environment.

B. VR Setup and Camera Rig Module

This module focused on converting the standard 3D environment into an immersive Mobile VR experience. The Google Cardboard SDK was incorporated, which automatically handles the necessary transformations to render the dual, side-by-side images required for stereoscopic viewing. The system's camera rig was meticulously configured to receive real- time data from the mobile device's gyroscope, thus enabling highly accurate and low-latency head tracking. This ensures that as the user physically rotates their head, the virtual camera rotates simultaneously, maintaining the crucial sense of presence. The proper configuration of this rig was critical to prevent visual inconsistencies and VR-related discomfort.

C. Navigation Control Module

The navigation system was implemented to provide redundancy and user choice. The system utilizes Unity's input management to monitor input channels. For the Bluetooth keyboard input, standard WASD or directional key press events are captured and translated into continuous, smooth movement vectors within the virtual space. This input strategy ensures that the application is fully functional whether the user is standing or seated and utilizing a keyboard for control.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

VI. RESULTS AND PERFORMANCE EVALUATION

A. *Experimental Results*

Extensive testing was performed using Unity Profiler on target mobile devices to quantify the application's performance and stability, particularly in relation to the demanding requirements of a real-time VR application. The key findings related to responsiveness and fluidity are as follows:

1) Latency: The measured latency, which is the delay between a user input (such as head movement) and the resulting visual change on the screen, was consistently found to be less than 20 milliseconds (< 20ms). This low latency is vital for minimizing motion sickness and ensuring a believable, responsive virtual experience.

2) Frame Rate (FPS): The application maintained a highly stable frame rate, consistently operating at greater than 60 frames per second (> 60 FPS) throughout the tour. This high and stable frame rate ensures smooth visual updates, preventing choppiness and maintaining immersion.

3) CPU Utilization: Monitoring of the device's main processor load revealed that CPU utilization is efficient, with no significant spikes or unexpected performance drops occurring even during complex scene transitions or interaction events. This indicates that the multithreading and job scheduling techniques employed in Unity were effective in distributing the processing load optimally.

B. *Performance Insight*

Beyond fundamental metrics, deeper analysis using tools like the Unity Profiler Benchmarks provided detailed insights into resource management:

1) Memory Usage: The total App Requested Memory was confirmed to be well within the acceptable and optimal limits established for mobile VR applications. There were no instances of excessive memory leaks or sudden, detrimental memory spikes detected during long-term use, confirming efficient resource allocation across all graphics and native processing components.

2) Multithreading: The implementation demonstrated effective use of multiple processing threads, allowing the VR rendering pipeline to execute concurrently with input handling and scene logic, which is the primary reason for the stable frame rate and low latency achieved.

Fig 6.1: CPU Performance

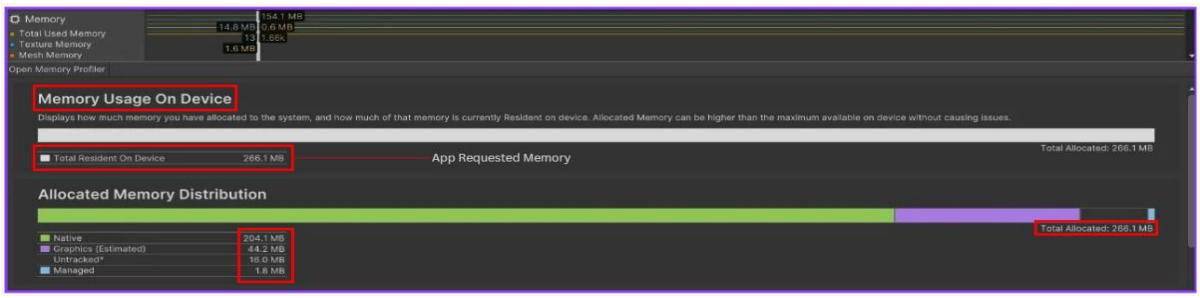


Fig 6.2: Optimized RAM Usage

C. Weak Spots Made Better (Optimization Cycle)

During initial testing, certain Weak Spots were identified, primarily related to asset loading times and initial frame rate stability when entering highly detailed rooms. These were addressed through a systematic optimization cycle and unity's native optimization options:

- Large texture files caused momentary stalls upon scene loading.
- All major textures were compressed and optimized to power-of-two resolutions, which significantly reduced the data size and improved load times.
- Certain complex 3D models with high geometric detail caused temporary frame rate drops.
- The geometry of these specific models, particularly decorative elements, was manually reviewed and reduced in complexity (lower polygon count) without noticeable loss of visual fidelity.

Result: This optimization cycle led to the observed excellent performance metrics, where the system now operates smoothly and maintains its low latency and high FPS threshold across the entire virtual house tour.

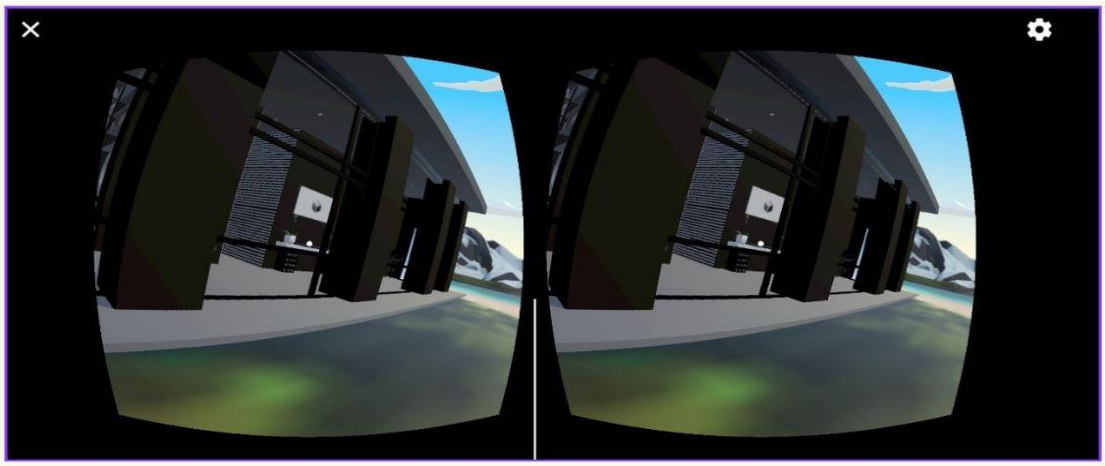


Fig 6.3 : Output

VII. CONCLUSION

The "Mobile VR Experiences: VIRTUAL HOUSE TOUR" project successfully achieved all of its established objectives. A highly realistic, immersive virtual tour application was developed using Unity, integrated with precision models from Blender. Crucially, the final application functions seamlessly as a cost-effective Mobile VR experience, maximizing user accessibility by relying on standard smartphones and the VR BOX.

The navigation system ensures a comfortable user experience for all individuals. Furthermore, rigorous performance evaluation confirmed the technical success of the system, demonstrating consistently low latency (< 20ms), high frame rate (> 60 FPS), and efficient memory and CPU utilization. This solution stands as a highly practical, ready-to-use tool for pre-construction architectural visualization, empowering clients to confidently experience their future home.

A. Source Code

1) PlayerMovementOld.cs

```
using System.Collections;
using System.Collections.Generic; using UnityEngine;

public class PlayerMovementOld : MonoBehaviour
{
    [Header("Movement Settings")] public float moveSpeed = 5f;

    void Update()
    {
        // Get horizontal (A/D, Left/Right Arrow) and vertical (W/S, Up/Down Arrow) input float horizontal =
```

```
        Input.GetAxis("Horizontal");
        float vertical = Input.GetAxis("Vertical");

        // Create a movement vector (XZ plane)
        Vector3 move = new Vector3(horizontal, 0, vertical);

        // Apply movement relative to world space
        transform.Translate(move * moveSpeed * Time.deltaTime, Space.World);
    }
}
```

2) CardboardStartup.cs:

```
using Google.XR.Cardboard; using UnityEngine;

/// <summary>
/// Initializes Cardboard XR Plugin.
/// </summary>
public class CardboardStartup : MonoBehaviour
{
    /// <summary>
    /// Start is called before the first frame update.
    /// </summary> public void Start()
    {
        // Configures the app to not shut down the screen and sets the brightness to maximum.

        // Brightness control is expected to work only in iOS, see:
        // https://docs.unity3d.com/ScriptReference/Screen-brightness.html. Screen.sleepTimeout = SleepTimeout.NeverSleep;
        Screen.brightness = 1.0f;

        // Checks if the device parameters are stored and scans them if not. if (!Api.HasDeviceParams())
        {
            Api.ScanDeviceParams();
        }
    }

    /// <summary>
    /// Update is called once per frame.
    /// </summary> public void Update()
    {
        if (Api.IsGearButtonPressed)
        {
            Api.ScanDeviceParams();
        }
        if (Api.IsCloseButtonPressed)
        {
            Application.Quit();
        }
        if (Api.IsTriggerHeldPressed)
        {
            Api.Recenter();
```

```
        }
        if (Api.HasNewDeviceParams())
        {
            Api.ReloadDeviceParams();
        }
        Api.UpdateScreenParams();
    }
}
```

3) GraphicsAPITextController.cs

```
using System.Collections;
using System.Collections.Generic; using UnityEngine;
using UnityEngine.Rendering;

/// <summary>
/// Initializes Graphics API Text Mesh according to the used graphics API.
/// </summary>
public class GraphicsAPITextController : MonoBehaviour
{
    /// <summary>
    /// Start is called before the first frame update.
    /// </summary> void Start()
    {
        TextMesh tm = gameObject.GetComponent(typeof(TextMesh)) as TextMesh; switch (SystemInfo.graphicsDeviceType)
        {
#if !UNITY_2023_1_OR_NEWER
            case GraphicsDeviceType.OpenGLES2: tm.text = "OpenGL ES 2";
                break;
#endif
case GraphicsDeviceType.OpenGLES3: tm.text = "OpenGL ES 3";
    break;
#if UNITY_IOS
            case GraphicsDeviceType.Metal: tm.text = "Metal";
                break;
#endif
#if UNITY_ANDROID
            case GraphicsDeviceType.Vulkan: tm.text = "Vulkan";
                break;
#endif

      }
    }
}

default:
    tm.text = "Unrecognized Graphics API"; break;
```

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

4) VrModeController.cs

```
using System.Collections; using Google.XR.Cardboard; using UnityEngine;
using UnityEngine.InputSystem;
using UnityEngine.InputSystem.Controls; using UnityEngine.InputSystem.Utilities; using UnityEngine.XR;
using UnityEngine.XR.Management;
using InputSystemTouchPhase = UnityEngine.InputSystem.TouchPhase;

/// <summary>
/// Turns VR mode on and off.
/// </summary>
public class VrModeController : MonoBehaviour
{
    // Field of view value to be used when the scene is not in VR mode. In case
    // XR isn't initialized on startup, this value could be taken from the main
    // camera and stored.
    private const float _defaultFieldOfView = 60.0f;

    // Main camera from the scene. private Camera _mainCamera;

    /// <summary>
    /// Gets a value indicating whether the screen has been touched this frame.
    /// </summary>
    private bool _isScreenTouched
    {
        get
        {
            TouchControl touch = GetFirstTouchIfExists(); return touch != null && touch.phase.ReadValue() ==
            InputSystemTouchPhase.Began;
        }
    }

    /// <summary>
    /// Gets a value indicating whether the VR mode is enabled.
    /// </summary>
    private bool _isVrModeEnabled
    {
        get
        {
            return XRGeneralSettings.Instance.Manager.isInitializationComplete;
        }
    }

    /// <summary>
    /// Start is called before the first frame update.
    /// </summary> public void Start()
    {
        // Saves the main camera from the scene.
        _mainCamera = Camera.main;
```

```
    // Configures the app to not shut down the screen and sets the brightness to maximum.
    // Brightness control is expected to work only in iOS, see:
    // https://docs.unity3d.com/ScriptReference/Screen-brightness.html. Screen.sleepTimeout = SleepTimeout.NeverSleep;
    Screen.brightness = 1.0f;

    // Checks if the device parameters are stored and scans them if not.
    // This is only required if the XR plugin is initialized on startup,
    // otherwise these API calls can be removed and just be used when the XR
    // plugin is started.
    if (!Api.HasDeviceParams())
    {
        Api.ScanDeviceParams();
    }
}

/// <summary>
/// Update is called once per frame.
/// </summary> public void Update()
{
    if (_isVrModeEnabled)
    {
        if (Api.IsCloseButtonPressed)
        {
            ExitVR();
        }
        if (Api.IsGearButtonPressed)
        {
            Api.ScanDeviceParams();
        }
        Api.UpdateScreenParams();
    }
    else
    {
        // TODO(b/171727815): Add a button to switch to VR mode. if (_isScreenTouched)
        {
            EnterVR();
        }
    }
}

/// <summary>
/// Checks if the screen has been touched during the current frame.
/// </summary>
/// <returns>
/// The first touch of the screen during the current frame. If the screen hasn't been touched,
/// returns null.
/// </returns>
private static TouchControl GetFirstTouchIfExists()
{
    Touchscreen touchScreen = Touchscreen.current; if (touchScreen == null)
```

```
    {
        return null;
    }

    if (!touchScreen.enabled)
    {
        InputSystem.EnableDevice(touchScreen);
    }
    ReadOnlyArray<TouchControl> touches = touchScreen.touches; if (touches.Count == 0)
    {
        return null;
    }
    return touches[0];
}
/// <summary>
/// Enters VR mode.
/// </summary>
private void EnterVR()
{
    StartCoroutine(StartXR());
    if (Api.HasNewDeviceParams())
    {
        Api.ReloadDeviceParams();
    }
}

/// <summary>
/// Exits VR mode.
/// </summary> private void ExitVR()
{
    StopXR();
}
private IEnumerator StartXR()
{
    Debug.Log("Initializing XR...");
    yield return XRGeneralSettings.Instance.Manager.InitializeLoader();

    if (XRGeneralSettings.Instance.Manager.activeLoader == null)
    {
        Debug.LogError("Initializing XR Failed.");
    }
    else
    {
        Debug.Log("XR initialized.");

        Debug.Log("Starting XR...");
        XRGeneralSettings.Instance.Manager.StartSubsystems(); Debug.Log("XR started.");
    }
}
private void StopXR()
```

```
    {
        Debug.Log("Stopping XR..."); XRGeneralSettings.Instance.Manager.StopSubsystems(); Debug.Log("XR stopped.");
        Debug.Log("Deinitializing XR..."); XRGeneralSettings.Instance.Manager.DeinitializeLoader(); Debug.Log("XR
        deinitialized.");

        _mainCamera.ResetAspect();
        _mainCamera.fieldOfView = _defaultFieldOfView;
    }
}
```

5) mainTemplate.gradle

```
apply plugin: 'com.android.library'
**APPLY_PLUGINS**

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'androidx.appcompat:appcompat:1.6.1' implementation 'com.google.android.gms:play-services-vision:20.1.3'
    implementation 'com.google.android.material:material:1.12.0' implementation 'com.google.protobuf:protobuf-javalite:3.19.4'
**DEPS**} android {
    namespace "com.unity3d.player"
    ndkPath "**NDKPATH**" compileSdkVersion **APIVERSION** buildToolsVersion '**BUILDTOOLS**'
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_11 targetCompatibility JavaVersion.VERSION_11
    }

    defaultConfig {
        minSdkVersion **MINSDKVERSION** targetSdkVersion **TARGETSDKVERSION** ndk {
            abiFilters **ABIFILTERS**
        }
        versionCode **VERSIONCODE** versionName '**VERSIONNAME**'
        consumerProguardFiles 'proguard-unity.txt'**USER_PROGUARD**
    }
    lintOptions { abortOnError false
    }

    aaptOptions {
        noCompress = **BUILTIN_NOCOMPRESS** + unityStreamingAssets.tokenize(', ') ignoreAssetsPattern =
        "!.svn:!.git:!.ds_store:!*.scc:!CVS:!thumbs.db:!picasa.ini:!*~"
    }**PACKAGING_OPTIONS**
}
**IL_CPP_BUILD_SETUP**
**SOURCE_BUILD_SETUP**
**EXTERNAL_SOURCES**
```

VIII. ACKNOWLEDGEMENT

It is one of the most efficient tasks in life to choose the appropriate words to express one’s gratitude to the beneficiaries. We are very much grateful to God who helped us all the way through the project and how molded us into what we are today.
We are grateful to our beloved Principal Dr. R. RADHAKRISHNAN, M.E., Ph.D., Adhiyamaan College of Engineering (An Autonomous Institution), Hosur for providing the opportunity to do this work in premises.

We acknowledge our heartfelt gratitude to Dr. G. FATHIMA, M.E., Ph.D., Professor and Head of the Department, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Hosur, for her guidance and valuable suggestions and encouragement throughout this project and made us to complete this project successfully.

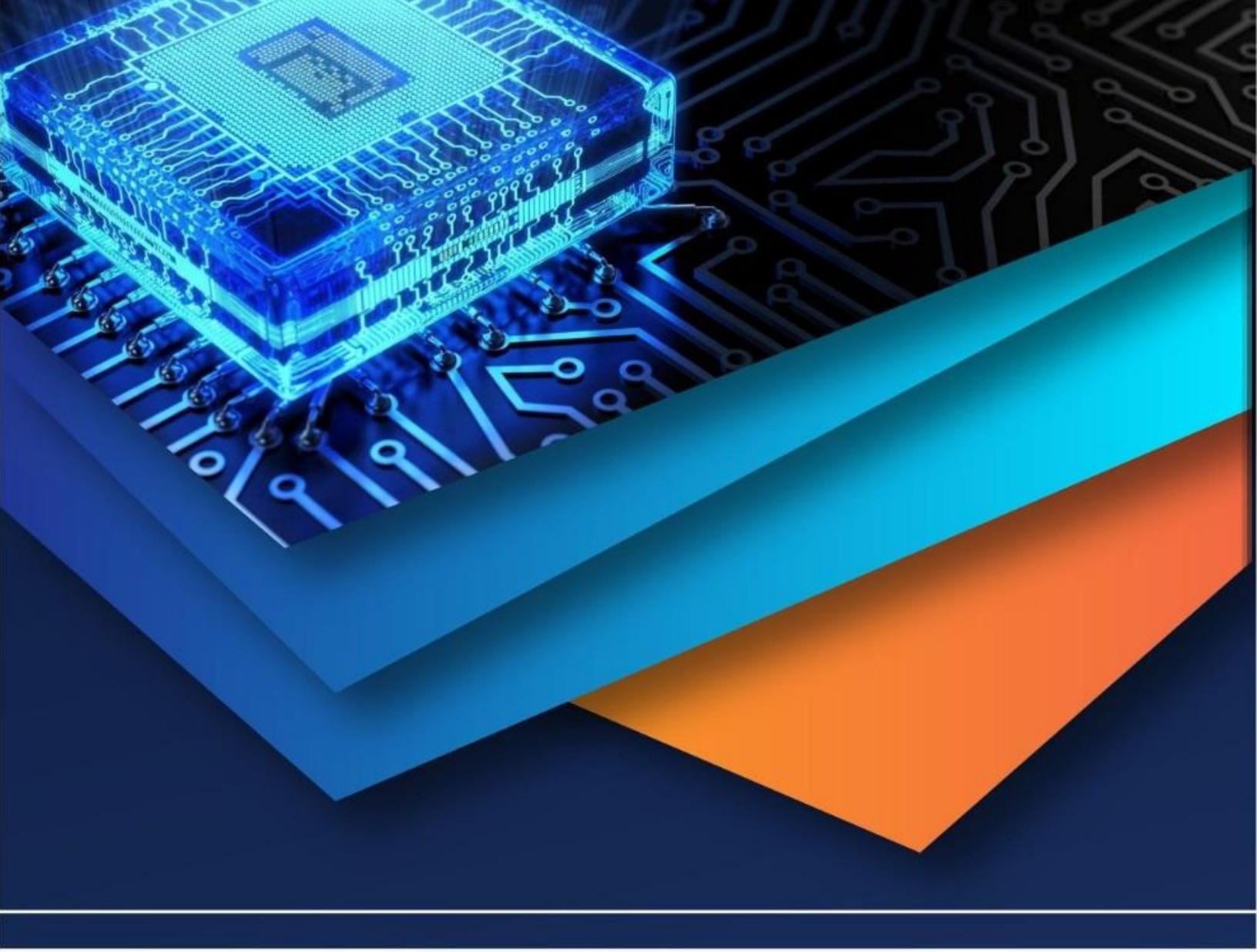
We are highly indebted to Mrs. V. MEENA, M.E., Supervisor, Assistant Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Hosur, whose immense support encouragement and valuable guidance were responsible to complete the project successfully.

We also extent our thanks to Project Coordinator and all Staff Members for their support in complete this project successfully.

Finally, we would like to thank to our parents, without their motivational and support would not have been possible for us to complete this project successfully.

REFERENCES

[1] R. K. McMahan, "Exploring the Role of Presence in Virtual Environments," Journal of Architectural Computing, vol. 25, no. 3, pp. 45–52, 2021.
[2] J. Whyte, "Virtual Reality and the Built Environment," Architectural Engineering and Design Management, vol. 18, no. 4, pp. 312–326, 2022.
[3] S. A. Diniz, "Mobile Virtual Reality for Architecture: Democratizing Immersion," Procedia Computer Science, vol. 196, pp. 240–248, 2022.
[4] T. Jones and K. Lee, "Smartphone-Based VR Systems: Design and Performance," IEEE Access, vol. 9, pp. 77645–77658, 2021.
[5] Philipp Maruhn, VR Pedestrian Simulator Studies at Home: Comparing Google Cardboards to Simulators in the Lab and Reality, Technical University of Munich Research Paper, 2021.
[6] P. Sousa et al., "Optimization Techniques for Real-Time Mobile VR Rendering," International Conference on Virtual Systems and Multimedia, 2021.
[7] C. Lin and H. Hsu, "Reducing Motion Sickness in VR Through Controlled Locomotion," Human–Computer Interaction Journal, vol. 37, no. 2, pp. 95–106, 2023.
[8] A. Patel and M. Kumar, "Gaze-Based Interaction in Mobile Virtual Environments," ACM Transactions on Interactive Systems, vol. 12, no. 1, pp. 1–18, 2022.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY