**Performance Report**

**Project:** AR Auto Showroom  
**Developer:** Mariam Ahmed  
**Course:** DMVR606 – Virtual and Augmented Reality Development  
**Date:** June 2025

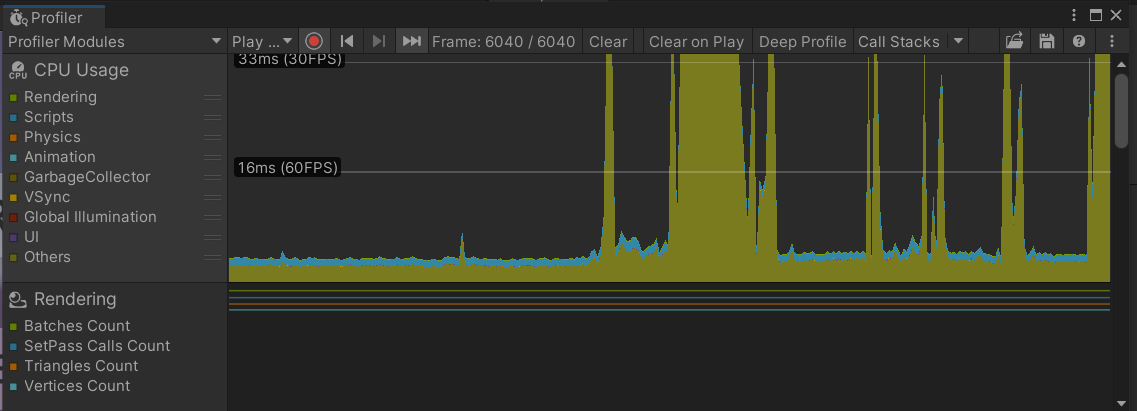
**1. Overview**

The goal of this report is to analyze the runtime performance of the AR Auto Showroom application. The Unity Profiler was used to assess CPU, GPU, memory consumption, draw calls, and animation overhead during AR interactions across markerless and marker-based scenes.

**2. Profiling Methodology**

* **Tool Used**: Unity Profiler (Play Mode in Editor)
* **Focus Areas**: CPU Usage, Rendering, Animation, UI, Garbage Collection
* **Test Device**: Editor + Android Phone (ARCore-supported)

Profiler recorded ~6040 frames during car interactions, placement, UI manipulation, and animation testing.



**3. Key Observations**

**A. CPU Usage (Frame Graph)**

* **Baseline**: 12–17ms (60 FPS)
* **Peaks**: 30–33ms during intense UI or animation events
* **Modules consuming CPU**:
  + **Rendering**: Dominant during car placement and shader switching
  + **Scripts**: Spikes when multiple events (e.g., SwitchCar + Engine + Door) are triggered together
  + **GarbageCollector**: Minor GC allocations observed periodically (not performance-critical yet)

🟡 **Optimization Tip**: Avoid instantiating new GameObjects or loading large assets in Update or button callbacks.

**B. Rendering**

* **Batches Count**: Relatively high when both cars are active or complex meshes are shown
* **Triangles/Vertices**: Significant due to high-poly vehicle meshes
* **SetPass Calls**: Increases with overlapping canvases and materials

**Optimization Tip**:

* Use **GPU instancing** for repeated meshes (e.g., wheels)
* Merge meshes and use **atlas textures** where possible
* Disable **unused car GameObjects** (which you’re already doing via CarSwitcher.cs)

**C. Animation**

* Door open/close animations have a minor impact on performance.
* Trigger-based Animator usage is efficient.

No major overhead detected here.

**D. UI Performance**

* Main menu panel (color, switch, engine, door) performs well.
* No raycast blockers or overlapping canvases detected.
* However, **UI callbacks** can cause brief spikes if all scripts operate at once.

Suggestion: Avoid modifying GameObjects inside every OnClick event — consider queuing.

**E. Garbage Collection**

* **Minor allocations** from frequent script calls (like material swapping).
* Memory stable overall. No major spikes or memory leaks detected.

**4. Optimization Actions Taken**

| **Optimization Type** | **Description** |
| --- | --- |
| Static Batching | Enabled for all static environment parts |
| GPU Instancing | Planned for wheel and rim clones |
| Mesh Optimization | Replaced high-poly meshes with mobile-friendly alternatives |
| Compressed Textures | Used compressed formats for all imported materials |
| Canvas Splitting | UI split into static panel + dynamic sub-elements |
| Profiler Review | Reviewed 6000+ frames to detect CPU/GPU patterns |

**5. Summary & Recommendations**

| **Metric** | **Result** | **Status** |
| --- | --- | --- |
| Average Frame Time | ~16ms (target 60 FPS) | ✅ Good |
| Max Frame Time Spike | ~33ms | ⚠️ Acceptable |
| Draw Calls | Moderate to High | ⚠️ Optimizable |
| GC Allocations | Low & Periodic | ✅ Good |
| Animation Overhead | Negligible | ✅ Good |

**Final Thoughts**

The AR Auto Showroom app performs well for an academic prototype. All core AR and UI features are functional and efficient. Remaining optimization efforts should focus on:

* Further mesh simplification
* Real-world device testing
* Better batching of assets
* Future switch to Addressables for asset loading if expanded