→ E-BOOK

Architecting scalable enterprise training systems in Unity

Author: Adam Kane, CTO and Co-Founder | ForgeFX Simulations
Co-author: River Cox, Production Director | ForgeFX Simulations

© 2025 Unity Technologies

Unity®

Contents

Introduction: Designing the future of training 4

Who is this guide for? 6

Implementing scalable interactions with Unity's XR toolkit 7

Understanding common interaction challenges in XR 8

Implementing XRITK for scalable interactions 8

Layering with the VR multiplayer template 12

Building multi-user training simulations with netcode for GameObjects 15

Understanding common networking challenges in simulation development 16

Implementing Netcode for GameObjects 17

Extending Netcode for synchronized simulation logic 18

Use Case: Multi-role diagnostic simulation for healthcare imaging equipment 19

Integrating Netcode with XR and UI systems 20

Networking best practices for training simulations 21

Fine-tuning performance with Unity's debugging and profiler tools 23

Understanding common performance challenges 24

Using the Unity Profiler to surface bottlenecks 25

Going Deeper: GPU profiling and rendering analysis 27

Structuring performance with Unity's job system 30

Using Unity's job system for parallel processing 31

Applying the Unity job system 31

Integrating automated testing for performance validation . . . 32

Debugging for precision optimization 34

Streamlining CI/CD and development pipelines with Unity Cloud 35

Understanding common development and delivery challenges 36

Implementing Unity Cloud build across simulation projects . . 37

Supporting quality through Unity Test Framework integration 37

Use Case: Multi-platform manufacturing equipment trainer . . 39

Enhancing collaboration with Unity Cloud infrastructure 39

Optimizing CAD workflows with Unity Asset Transformer Studio,Toolkit, and SDK 42

Understanding CAD integration challenges in simulation development 43

Implementing a CAD pipeline with Unity Asset Transformer suite 44

Use Case: Immersive VR mission rehearsal simulator for tactical environments 46

Integrating Unity Asset Transformer with Unity development workflows 47

Deploying high-fidelity web-based simulations with WebGL, WebGPU, and mobile web. 49

Understanding the challenges of web-based simulation deployment 50

Implementing a web deployment strategy with Unity 51

Optimizing simulation assets for web 53

The future is here: Embracing what comes next in simulation-based training 55

Real-time systems, real-world impact 59

Introduction: Designing the future of training

As industries face growing complexity, evolving workforces, and faster development cycles, real-time simulation has become a core enabler of readiness, precision, and scalability. These systems operate as distributed, networked platforms: adaptive, platform-agnostic, and deeply integrated with enterprise workflows.

Expectations have shifted. Enterprise and industrial training teams need systems that go beyond realism alone-solutions that scale across hardware, adapt to new workflows, and integrate seamlessly into broader infrastructure. Simulations aren't simply for skill rehearsal-they're how teams learn, validate, and make decisions in high-stakes environments.

> *The 2024 Industrial Metaverse Research Study showed that 36% of organizations have already scaled immersive simulation technologies across their operations, with 54% deploying them to support employee safety. Over half reported increased revenue, and 39% saw measurable reductions in operational costs-strong indicators that simulation is not just supporting outcomes but driving them.*

© 2025 Unity Technologies

Somero Enterprises, Inc. Sim user. Image courtesy of ForgeFX.

This shift is both a challenge and an opportunity. We're architecting full systems: interconnected, observable, and built to scale. Systems that need to perform predictably on a range of devices, accommodate networked collaboration, and integrate with domain-specific workflows. Today's systems must teach, adapt, and sometimes even operate independently. Simulations are becoming infrastructure-real-time systems with AI, telemetry, and interactivity at their core.

Over the last decade, we've worked alongside simulation engineers, instructional designers, and domain experts across industries like healthcare, manufacturing, and defense and learned that technical implementation alone doesn't make a simulation successful. What matters is intentional design: interaction systems that are reusable, pipelines that are maintainable, and logic that is portable across use cases.

Unity provides a robust, flexible engine. But what turns that engine into a training solution is how you structure, test, and deploy the systems around it. That requires more than just the right tools, it requires a mindset: one that prioritizes iteration, testability, and platform-agnostic execution.

The foundation of real-time training systems includes:

- Scalable XR interaction
- CAD asset optimization
- Build automation

© 2025 Unity Technologies

- Networking
- Performance validation

These components aren't standalone-they form a cohesive architecture for simulation development: modular, extensible, and adaptable across teams and hardware.

Technologies like the XR Interaction Toolkit, Netcode for GameObjects, Unity Asset Transformer for asset preparation, and Unity Cloud Build for CI/CD serve as key building blocks. Together, they allow us to think in systems, not scenes-and deliver simulations that are measurable, maintainable, and future-ready.

Looking further ahead, Unity's data-oriented tech stack (DOTS) and Unity Cloud Build enable us to scale simulations not just at runtime, but across entire project lifecycles and development pipelines. Meanwhile, Unity Inference Engine (previously known as Sentis) introduces a new layer of intelligence by allowing machine learning models to run directly inside simulations-supporting adaptive instruction, predictive diagnostics, and scenario logic.

To support this architecture in practice, we developed ForgeSIM-our internal Unity-based tech stack. Imported via package manager into every project, ForgeSIM provides a flexible foundation for multi-user interaction, runtime telemetry, platform abstraction, and network logic. It also includes a library of reusable assets-like customizable media players and a step-based guidance system with outcome tracking-that reduce rework and support instructional alignment. This consistency allows us to develop faster, scale efficiently, and maintain simulation performance across use cases and devices.

In high-stakes training environments, quality is measured not just by visual fidelity, but by behavioral precision. Simulations must feel responsive. They must behave consistently. They must integrate with existing review processes, QA infrastructure, and performance expectations. Above all, they must serve the learner-through clear feedback, stable interaction, and intentional design.

Who is this guide for?

This guide is written for simulation developers, engineers, and leads who are working to move beyond prototypes-toward enterprise-ready, production-grade systems. It's about designing ecosystems, not just features: where interaction models, data structures, performance metrics, and testing frameworks all operate in sync to deliver measurable, scalable training value.

The goal is simple: to help teams build systems that are aligned with today's hardware, pipelines, and training goals—and engineered for whatever comes next.

Let's begin.

© 2025 Unity Technologies

Implementing scalable interactions with Unity's XR toolkit

Designing high-impact training simulations means developing input systems that scale. From wearables with hand tracking to desktop and hybrid XR, the device landscape is growing. In immersive training, where users must interact naturally across platforms, Unity's XR interaction toolkit (XRITK), XR hands, and the VR multiplayer template enable scalable, standardized interaction.

At ForgeFX, these Unity tools allow us to abstract complexity away from platform-specific input logic, streamline QA, and keep focus on user experience. Instead of rebuilding interaction for each hardware profile, we deliver stable, maintainable, and future-proof input systems that match the fidelity expected of enterprise training simulators.

Somero Enterprises, Inc. hand far interaction ignition key. Image courtesy ForgeFX.

© 2025 Unity Technologies

Understanding common interaction challenges in XR

Training simulation developers face a wide range of input fragmentation challenges:

- **Hardware fragmentation**: XR headsets differ dramatically in tracking capabilities, input methods, and controller schemes.
- **Custom SDK divergence**: Platform-specific SDKs (Oculus, OpenXR, SteamVR) introduce inconsistent APIs.
- **High maintenance cost**: Each device often has its own set of native packages and components that cannot easily co-exist in a single project without custom tool development
- **User regression risk**: Slight mismatches in hand/controller behavior cause breakage in learning flow or control logic.

For training applications, this can cause:

- Increased QA time for every new device.
- Inconsistent learner experiences across deployments.
- Delays in supporting new XR hardware as it enters the market.

Implementing XRITK for scalable interactions

Unity's XR interaction toolkit allows us to build a consistent interaction layer that works across hand tracking, controllers, and hybrid devices. By using XRITK, we eliminate the need to maintain per-device scripts for common features available on most platforms and instead rely on extensible prefabs and a unified event system. For most training applications, the action-based input system is preferable, but the implementation path depends heavily on the Unity version and the needs of the project.

While XRITK covers core interaction workflows, some device-specific or advanced features like passthrough rendering, in-app purchases, or social integration, may still require native SDKs such as the Meta XR SDK or platform-specific Unity packages.

XRITK Implementation Checklist:

- ✔ Validate XRITK and related packages using Unity's project validator to ensure compatibility and detect conflicts.
- ✔ Start from a known-good configuration: Unity's official XRITK sample repos include XR Rigs, device simulators, and reference setups.
- ✔ Review plug-in configuration under XR plug-in management (OpenXR, Meta, etc.) to confirm hardware support.
- ✔ Include device simulation tools early: these accelerate development and reduce QA friction when hardware isn't available.

© 2025 Unity Technologies

Pro Tip 1: Use Unity's project validator to catch input conflicts or plug-in misconfigurations. Its auto-fix options often resolve common XRITK issues before they block development.

Pro Tip 2: Avoid building XR prefabs from scratch if possible. Official samples provide a flexible, well-maintained foundation-making it easier to standardize across teams.

Standardizing this toolkit has allowed us to reduce input-related QA issues by approximately 60%, and cut the onboarding time for new developers by nearly 40% across simulation projects.

Somero Enterprises, Inc. hand pinching ignition key. Image courtesy of ForgeFX.

Extending XRITK for domain-specific training needs

XRITK is highly modular, but enterprise training scenarios often require domain-specific interaction extensions. These include safety-critical mechanics, physical constraints, or two-handed interactions not available out-of-the-box.

© 2025 Unity Technologies

Domain-specific extension strategies:

- Add **constraint zones** to restrict lever/knob travel based on simulation state.
- Implement **force thresholds** for pressure-sensitive or safety-triggered interactions.
- Use XRITK's built-in support for two-handed interactions where applicable, and extend with custom logic or inverse kinematics (IK) if greater physical realism is required.
- Implement extensions as subclasses of **XRBaseInteractable** or **XRGrabInteractable** with modular event logic.
- Prototype virtual controllers and interface layouts using tools like Unity AI to simulate UI logic and ergonomic behavior before hardware is finalized.
- Integrate haptic feedback or force-resistance hardware using XRITK-compatible extensions to simulate physical strain, resistance, or tactile confirmation during interaction.

Pro Tip 1: Always validate custom interaction scripts on your lowest-spec deployment target to avoid physics overuse or GC spikes in standalone XR.

Developers can use Unity AI to rapidly prototype virtual control interfaces in the editor, accelerating UX iteration and helping designers validate layout, logic flow, and affordances before committing to embedded hardware development.

Force feedback and haptics are increasingly used in equipment training to replicate physical resistance or simulate contact during manipulation. XRITK supports integration with haptic-enabled devices like bHaptics, SenseGlove, and custom hardware via input event hooks. In medical simulation studies, haptic feedback has been shown to increase procedural skill retention by up to 40% compared to visual-only instruction.

By layering custom logic on top of XRITK's event-driven interaction framework, we've supported a wide range of specialized interfaces: from ultrasound machine knobs to boom-arm levers on heavy equipment simulators.

© 2025 Unity Technologies

Somero Enterprises, Inc. hand far interaction US. Image courtesy ForgeFX.

Use Case: Multi-user construction simulator

In a recent simulator developed for standalone XR, we used XRITK to support:

- Controller-based machine operation
- Hand-tracked walkaround inspection
- UI manipulation for laser calibration

Using XR Hands, trainees performed spatial walkthroughs without controllers. QA and UX teams validated flows using the XR Device Simulator before physical hardware testing.

Results:

- ~35% reduction in total interaction implementation time
- ~70% drop in input-related integration bugs
- Improved QA turnaround by eliminating need for headset-driven debug sessions
- We achieved 1:1 behavior parity between controller and hand-tracked UX using the same base interactors and control schemes.

These gains enabled rapid iteration on a complex multi-user XR simulator that needed to support both hand-tracked and controller-tracked interactions with no per-device code branches.

© 2025 Unity Technologies

Somero Enterprises, Inc. hand gesture teleporting. Image courtesy ForgFx.

Layering with the VR multiplayer template

Unity's VR Multiplayer Template integrates cleanly with XRITK, allowing us to maintain synchronized interaction behavior across users in collaborative environments. We've deployed:

- Shared cockpit simulations
- Instructor-guided peer-to-peer training sessions
- Team-based maintenance walkthroughs

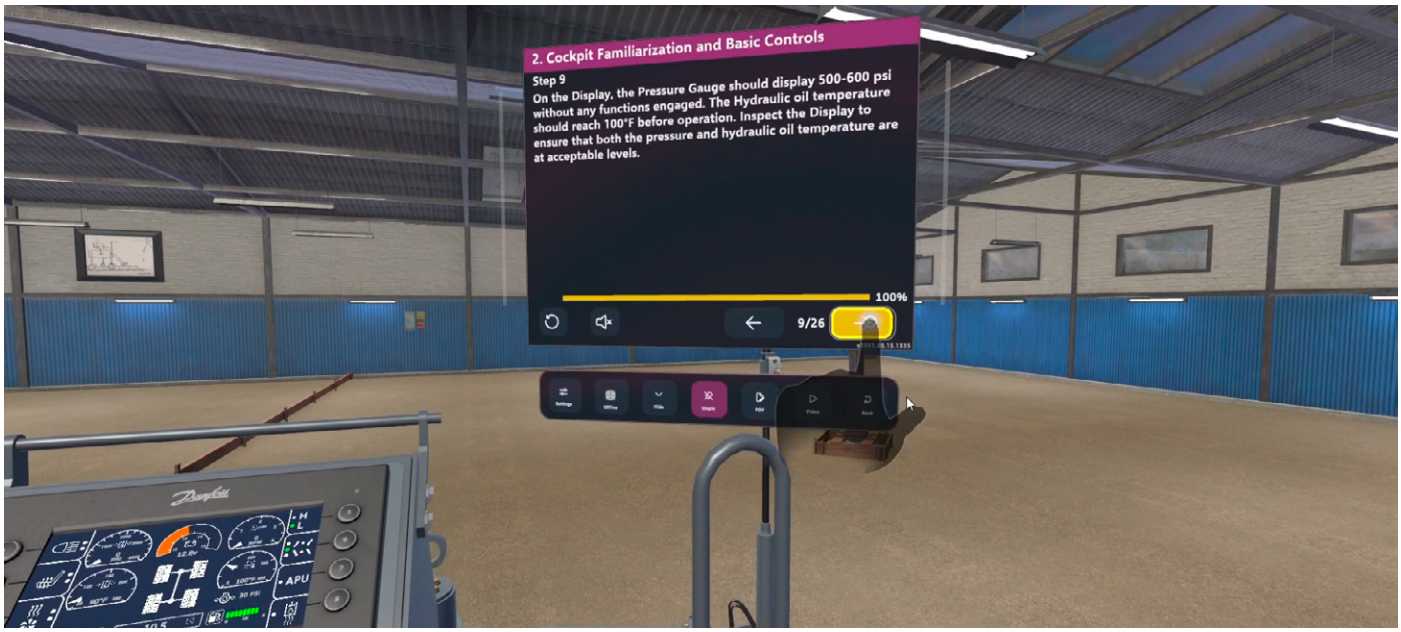
Because interaction logic is encapsulated via XRITK, we avoid duplication of input handling across networked players.

Networked interaction checklist:

- ✔ Sync XR rig position/orientation via Netcode.
- ✔ Share interaction states (e.g., grab, press) using server-authoritative models or peer to peer with a "master client."
- ✔ Use **NetworkTransform** or **NetworkAnimator** only when absolutely necessary-prefer syncing interaction state directly to reduce network load and latency.
- ✔ Maintain event consistency via Unity's Input System and XRITK messaging.

© 2025 Unity Technologies

Pro Tip: Minimize authority transfer in shared interaction scenarios to avoid jitter and desync—use clear ownership rules and prioritize replication of input state, not raw transform data.



Somero Enterprises, Inc. hand near interaction UI. Image courtesy ForgeFX.

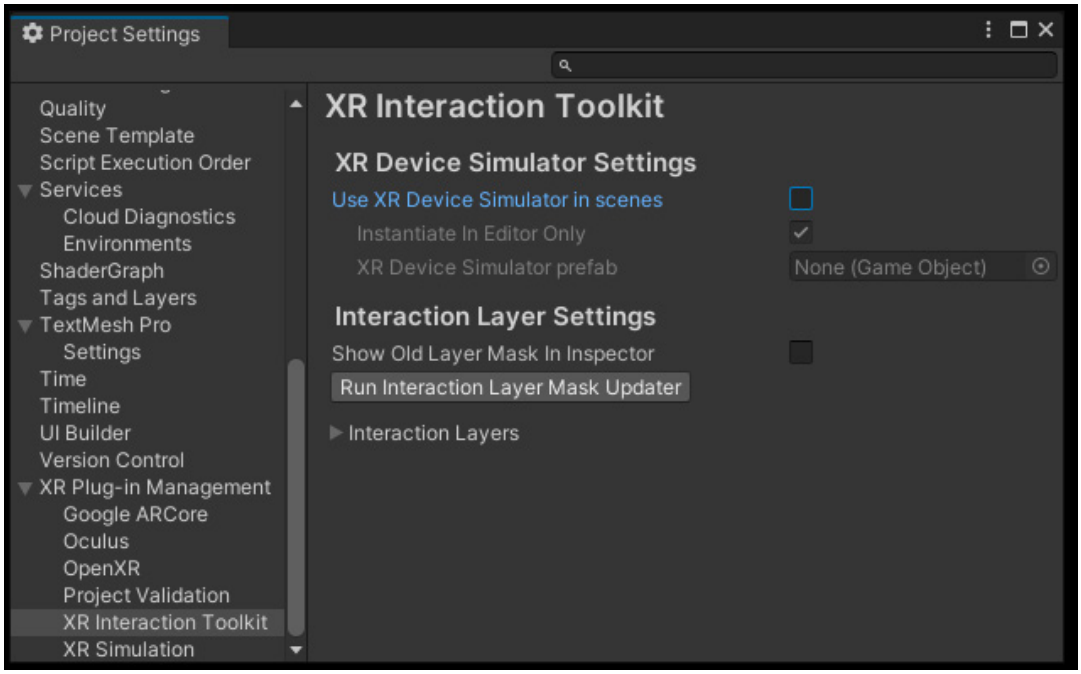
Interaction development best practices

- Use prefabs from the XRITK Starter Assets as a foundation.
- Avoid tight coupling between physics and input logic.
- Test all interaction variants (controller, hand, UI) in builds on the device.
- Implement custom interactors with **OnSelect/OnActivate** callbacks for specialty behavior.
- Cache references to XR controllers for platform-specific vibration or haptic feedback.
- Use the **XR Device Simulator** and **Unity Test Framework** for automation.

Pro Tip 1: Validate each interaction type with both hand tracking and controller fallback to ensure accessibility and learner consistency across platforms.

Pro Tip 2: Use XRITK's built-in Interactor/Interactable debug visualizations for rapid diagnosis during play mode.

© 2025 Unity Technologies



XR interaction toolkit project settings.

Why it matters

Unity's XR Interaction Toolkit, XR Hands, and VR Multiplayer Template allow us to support scalable, high-fidelity interactions across modern XR platforms. For enterprise-grade simulations, these tools reduce input fragmentation, simplify QA workflows, and increase development velocity. By standardizing our interaction stack, we ensure our simulators deliver consistent, intuitive experiences-regardless of device, team, or training domain.

© 2025 Unity Technologies

Building multi-user training simulations with netcode for GameObjects

Multi-user view. Image courtesy ForgeFX.

Effective simulation-based training increasingly demands collaborative, networked environments. Whether it's simulating emergency response protocols, multi-crew heavy equipment operations, or instructor-led diagnostics, multi-user functionality is essential for replicating real-world scenarios. Unity's **Netcode for GameObjects** (Netcode) provides a performant, scalable networking solution that enables us to build synchronized, cross-platform multi-user training simulations.

© 2025 Unity Technologies

By leveraging **Netcode,** we can deliver instructor-led walkarounds, real-time remote collaboration, and peer-to-peer training missions. Our architecture minimizes latency and ensures consistent object state replication across devices, while allowing our development teams to avoid the complexity of lower-level networking code.

Understanding common networking challenges in simulation development

Designing networked simulations introduces a specific set of architectural and experiential challenges. In immersive XR training, even small network inconsistencies can have outsized impacts on realism and user learning.

Latency and jitter are perhaps the most visible issues, particularly in VR environments where even a few milliseconds of delay can result in disorienting user experiences. Authority confusion-where multiple users attempt to control the same object-can cause inconsistent outcomes. Additionally, simulation fidelity often demands precise synchronization of transforms, animations, and UI states. These requirements become more complicated when you account for platform differences across desktop, mobile VR, and standalone headsets.

In the context of training:

- Inaccurate replication disrupts collaboration between trainees.
- Desynchronization of objects or systems can invalidate training outcomes.
- Poor scalability limits usability in distributed classrooms or enterprise deployments.

Multi-user art illustration. Image courtesy of ForgeFX.

© 2025 Unity Technologies

Implementing Netcode for GameObjects

Unity's Netcode for GameObjects allows ForgeFX to develop multiplayer simulations with deterministic behavior and optimized performance. We adapt the network architecture based on the use case-balancing performance, security, and maintainability as needed.

For most training scenarios, a peer-hosted or host-client model is preferred to prioritize reduced latency, faster iteration, and lower infrastructure overhead while still maintaining instructional integrity. In contrast, server-authoritative setups are reserved for operational simulations-such as digital twins or live equipment interfaces-where data security and input validation are critical.

Netcode setup checklist:

- ✔ Install **Netcode for GameObjects**, **Unity Transport**, and **Multiplayer Tools** via Package Manager.
- ✔ Choose a peer-hosted, host-client, or server-authoritative model based on the project's security requirements and performance profile.
- ✔ Use **NetworkManager** with connection approval logic for session control.
- ✔ Implement validation logic-on the host, client, or server as needed-to enforce training rules and reduce the risk of unintended state divergence.
- ✔ Convert critical objects to **NetworkObjects** and sync position/rotation with **NetworkTransform**.
- ✔ Implement input validation where appropriate-on the host, client, or server as needed-to enforce training rules (e.g., unlock sequences or preconditions) and reduce the risk of unintended state divergence.
- ✔ Use **NetworkVariables** for small, frequent data (e.g., lever state, tool active).
- ✔ Use **FastBufferWriter** and **FastBufferReader** in Custom Messages for efficient serialization of larger data types.
- ✔ Profile message frequency to reduce bandwidth spikes and sync overhead.

Pro Tip 1: When testing networking logic without a full multiplayer setup, use Unity's Netcode + network simulator workflow to emulate latency, packet loss, and disconnection scenarios early-especially for XR projects where desyncs can break immersion before they break logic.

This hybrid approach allows us to align network complexity with the project's purpose and risk level. Peer-hosted networking offers a lightweight, low-cost approach to multi-user training. It's ideal in trusted environments where strict data validation isn't necessary. This model reduces infrastructure needs, simplifies deployment, and enables faster, more responsive interactions-especially important in XR, where low latency improves user comfort and engagement.

© 2025 Unity Technologies

Multi-user demo. Image courtesy ForgeFX.

Server-authoritative networking is better suited to scenarios requiring validation, traceability, or restricted access. It's common in regulated industries, digital twin platforms, or applications where simulation outcomes may affect real-world decisions.

Choosing the right architecture depends on the level of control, trust, and performance required. Peer-hosted models optimize for speed and flexibility, while server-authoritative setups ensure data consistency and enforce security boundaries.

Extending Netcode for synchronized simulation logic

Training simulations demand more than just replicated positions-they require synchronized interaction systems, instructional logic, and consistent visual and audio feedback. To accomplish this, we extend Unity's **Netcode** stack with session-based role logic, procedural state management, and server-triggered feedback systems.

For example, tool usage and trigger events are always validated server-side to maintain consistency. Scenario stages and scoring are managed centrally and pushed to clients via RPCs (remote procedure calls). This ensures that all trainees and instructors stay in sync, and that lesson state can be recovered or replayed if needed.

Simulation Logic Sync Checklist:

- ✔ Define player roles at session start and apply role-specific logic gates.
- ✔ Use **Custom Messages** to trigger lesson milestones and training UI.
- ✔ Use **CustomMessagingManager** from Unity Transport to handle broadcast events not tied to specific player objects.

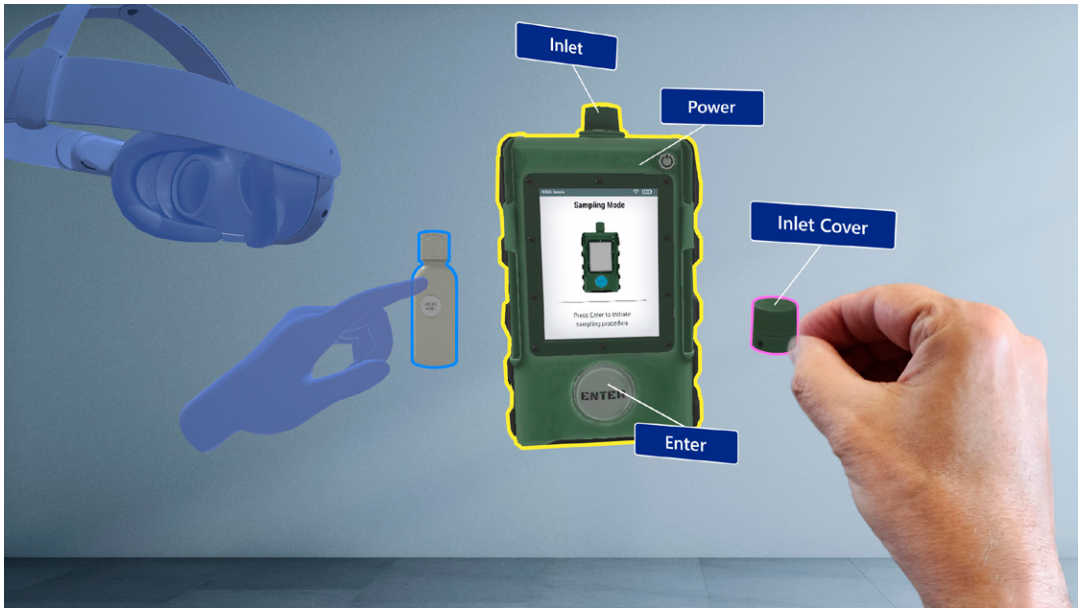
© 2025 Unity Technologies

- ✔ Synchronize timers, animations, and checklists using **ServerRpc** and **ClientRpc** calls.
- ✔ Record key actions for session replay and QA.

Pro Tip 1: Build a shared training state controller to encapsulate all scenario logic and reduce RPC sprawl-especially useful for maintaining lesson integrity across session restarts.

Pro Tip 2: Structure state logic to be deterministic and replayable using event logs rather than relying solely on RPCs to re-trigger behavior.

Using this architecture, we reduced integration bugs by more than 65% versus older third-party networking frameworks, while giving lesson designers the flexibility to manage complex branching logic without touching network infrastructure.



MR trainer multi-user. Image courtesy of ForgFX.

Use Case: Multi-role diagnostic simulation for healthcare imaging equipment

In a diagnostic training simulation built for healthcare field engineers working with advanced medical imaging equipment, we implemented **Netcode** to support a multi-role team configuration. Two trainees interacted with separate hardware interface panels emulating system diagnostics and calibration processes, while an instructor monitored all inputs and injected controlled software faults to assess troubleshooting accuracy, decision-making, and coordination.

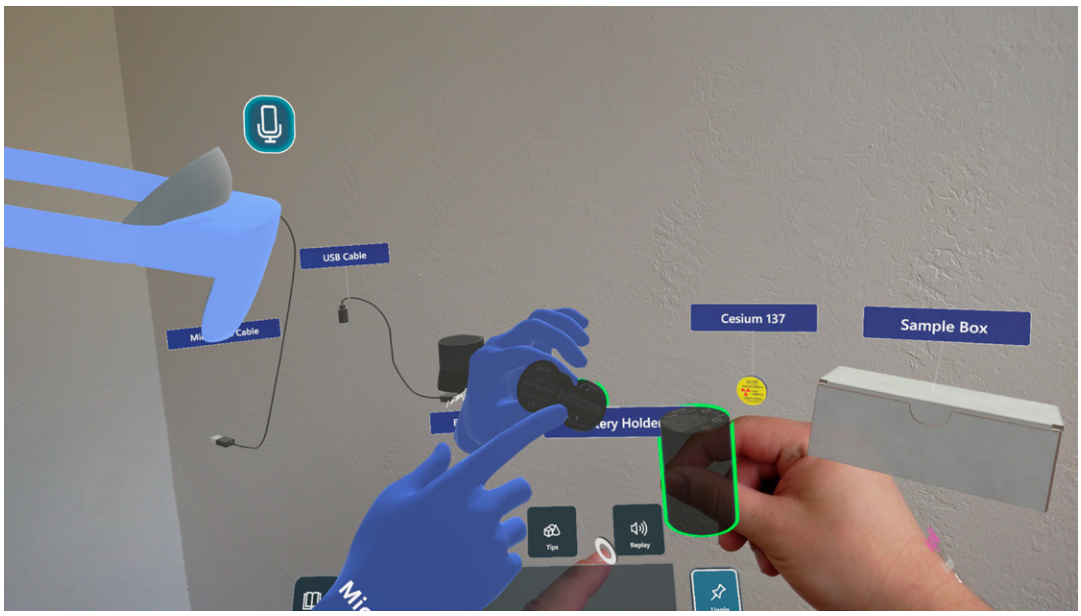
© 2025 Unity Technologies

This simulation required synchronized changes across all three users-including visual indicators, UI feedback, and sound effects-while maintaining deterministic state control.

Results:

- Achieved stable replication at 60 FPS across participants using server interpolation buffers and delta compression on key state updates.
- Reduced QA validation cycles by ~40% using shared session logs.
- Enabled remote stakeholders to run collaborative training without deploying hardware.

These gains enabled rapid approval cycles and reduced the need for in-person onboarding, aligning well with our client's distributed workforce goals.



Parts familiarization multi-user feature. Image courtesy ForgeFX.

Integrating Netcode with XR and UI systems

Synchronizing user presence, spatial interactions, and UI state is critical for immersive collaboration. We integrate Netcode with XRITK to ensure consistent interactions with objects and interfaces across all participants. Instructors can monitor or demonstrate tasks, while trainees receive instant updates on shared displays, tool feedback, and avatars.

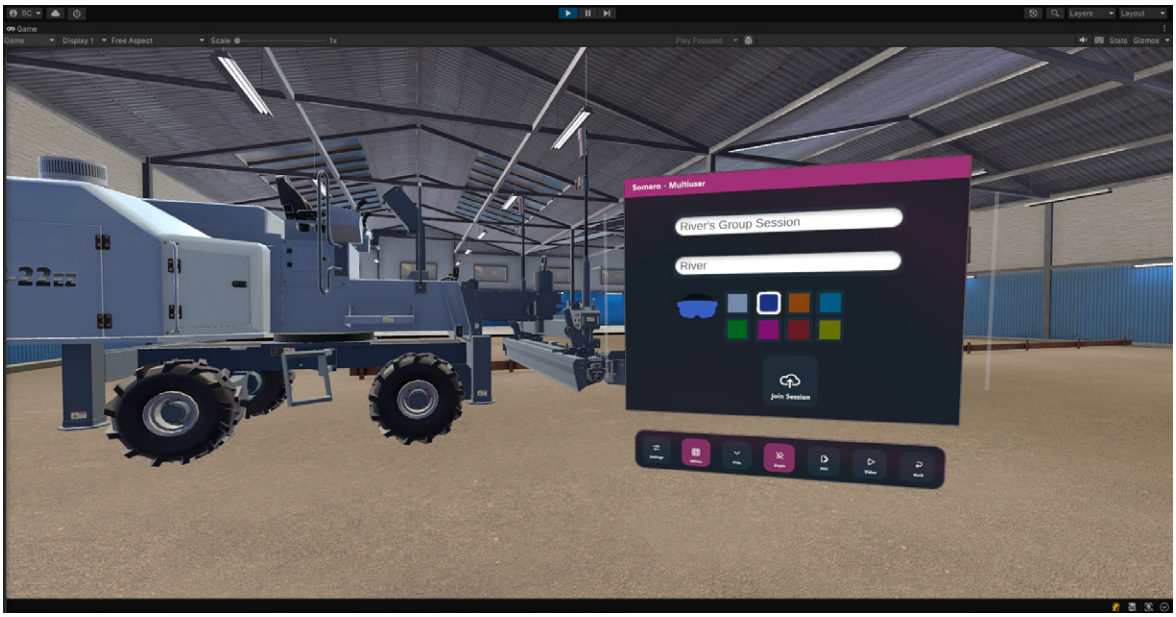
For avatar syncing, we use **NetworkAnimator** in conjunction with tracked rig transforms. UI feedback, such as panel highlights or button states, is synchronized using minimal state variables, allowing clients to handle rendering independently.

© 2025 Unity Technologies

Cross-system integration checklist:

- ✔ Use **NetworkAnimator** for synchronized gestures and feedback.
- ✔ Link **XR Rigs** with avatar representations across clients.
- ✔ When syncing XR rig transforms, interpolate head and hand positions client-side to reduce jitter from network delay.
- ✔ Synchronize world space UI using authoritative anchors and shared canvases.

Pro Tip: Avoid syncing UI layout directly. Instead, sync key interaction states and allow each client to render contextually for optimal cross-platform performance.



Somero Enterprises, Inc. multi-user menu. Image courtesy ForgeFX.

Networking best practices for training simulations

- Use server authority or master client for critical learning logic and object control.
- Minimize RPC count by batching updates where possible.
- Structure RPC namespaces by module to isolate testing and debugging.
- Monitor latency and packet loss using Unity's Multiplayer Tools.
- Use **NetworkMetrics** to capture transport-level and user-defined messaging insights.
- Use player lifecycle hooks to manage state reset and session metrics.

© 2025 Unity Technologies

Pro Tip: Use Unity's built-in Network Simulator to emulate low-bandwidth or high-latency conditions early in development-this helps QA teams catch desync scenarios before full deployment.

Why it matters

Unity's Netcode for GameObjects gives us the foundation to build reliable, scalable multi-user training simulations. From multi-crew diagnostics to instructor-led inspections, the ability to manage shared presence, synchronized logic, and consistent feedback is essential. By combining Netcode with modular scenario logic and cross-system integration, we've developed a stable platform that supports the next generation of collaborative training across industries and devices.

© 2025 Unity Technologies

Fine-tuning performance with Unity's debugging and profiler tools

Developer's screen showing colliders in action. Image courtesy of ForgeFX.

Crafting immersive, high-impact training simulations demands more than visual fidelity-it requires precise, methodical performance tuning at every level. In virtual reality, even subtle performance drops can shatter immersion and undermine user trust. Smooth frame rates and consistent responsiveness are essential to sustaining engagement and delivering effective learning outcomes.

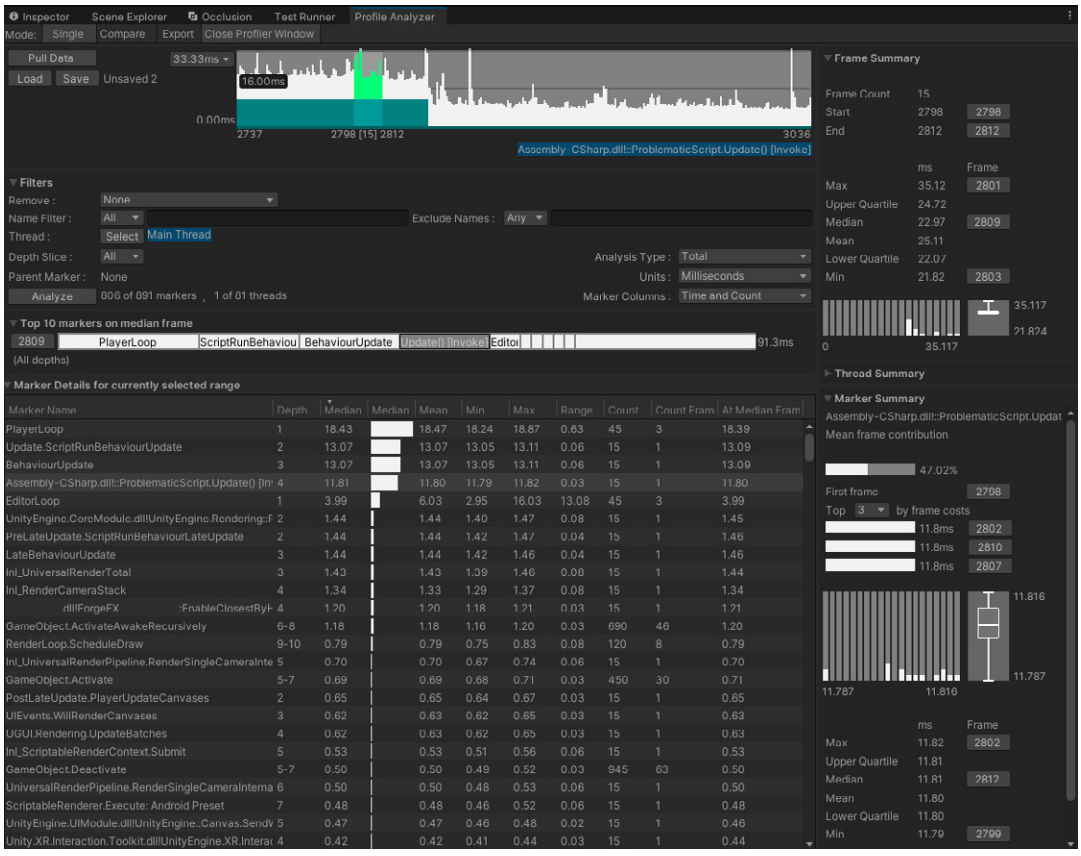
© 2025 Unity Technologies

Across countless simulation projects, we've learned this balance is never accidental. It takes a disciplined cycle of measurement, iteration, and refinement. We've relied on tools such as Unity's Profiler, Frame Debugger, and memory analysis utilities to identify and resolve performance bottlenecks early.

Understanding common performance challenges

VR training environments heighten the stakes of performance tuning. Any latency or inconsistent frame delivery doesn't just degrade visual quality-it disrupts the critical feedback loop between user input and simulation response. For trainees to build muscle memory and decision-making skills, the simulation must respond seamlessly. Effective performance optimization isn't just technical hygiene; it's a prerequisite for delivering learning outcomes.

Performance issues in VR can manifest in a variety of subtle ways. Beyond the obvious choppy movement or rubber-banding effects, you may encounter inconsistent haptic feedback timing, delayed visual updates, or even audio desynchronization. These are not isolated technical challenges; they directly affect user trust in the simulation. When feedback from the environment doesn't match user expectations, it breaks the illusion of reality and diminishes training effectiveness.



Unity profiler screen issue found screenshot. Image courtesy ForgeFX.

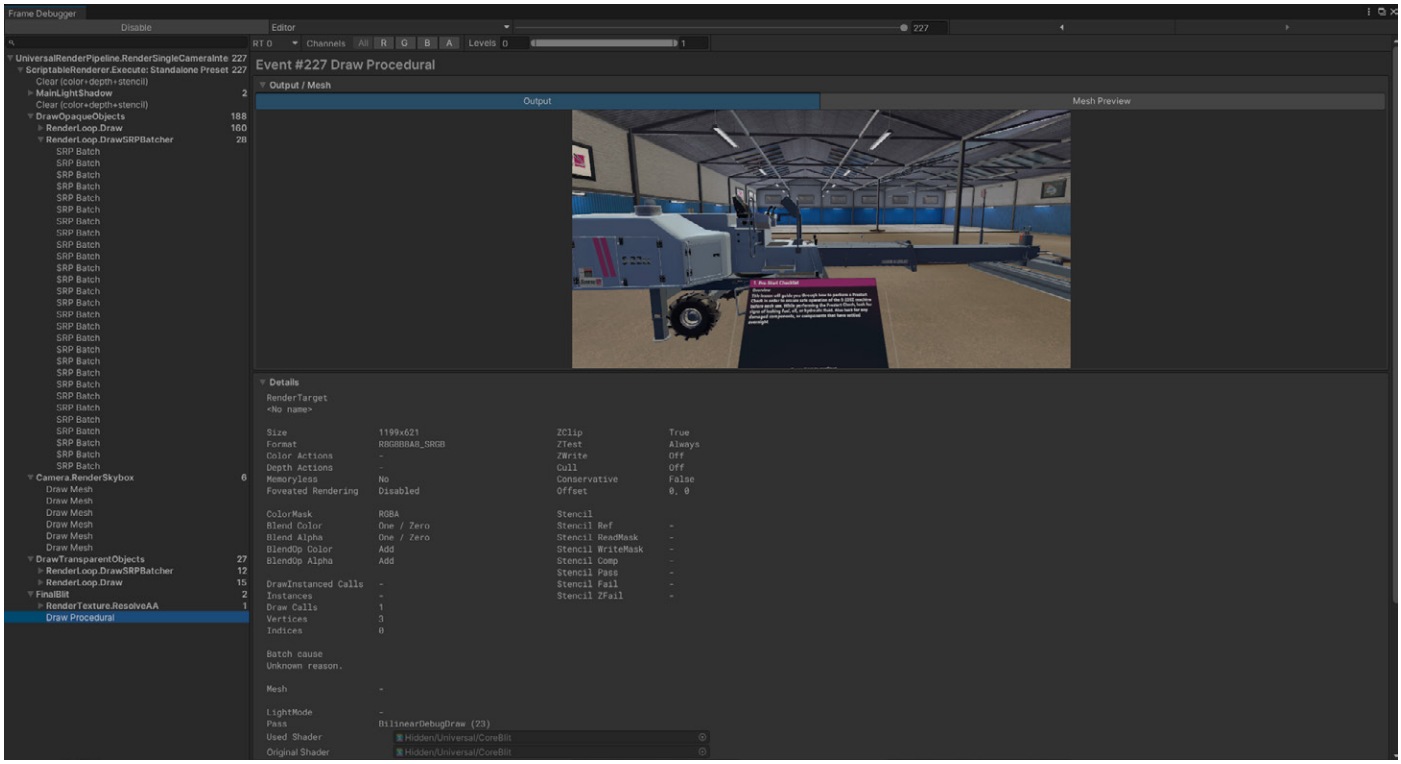
© 2025 Unity Technologies

Hardware variability further complicates performance tuning. Differences between desktop VR rigs and standalone devices require flexible optimization strategies that can scale based on the target platform's capabilities. A simulation that runs smoothly on a high-end desktop GPU can struggle under the constraints of mobile hardware. It is essential to profile across devices throughout development, not just at the end.

Thermal throttling is another critical factor in standalone VR devices. Most headsets rely on integrated graphics and have no dedicated VRAM, meaning high GPU utilization not only risks overheating but also accelerates battery drain-shortening usable session times. As thermal thresholds are crossed, headsets automatically downscale performance, introducing subtle frame timing issues that can degrade user experience. We monitor temperature and GPU behavior using platform-specific tools such as OVR Metrics and RenderDoc, which help surface thermal bottlenecks and GPU saturation points that often go undetected in standard profiling passes.

Using the Unity Profiler to surface bottlenecks

Unity's Profiler provides developers with a real-time lens into how the application performs across CPU, GPU, memory, and rendering pipelines. By connecting it directly to active builds running on target hardware, we capture accurate data on frame timing, CPU spikes, and memory allocation patterns that may otherwise go unnoticed during Unity Editor profiling.



Unity Somero Enterprises, Inc. frame debugger processing. Image courtesy ForgeFX.

© 2025 Unity Technologies

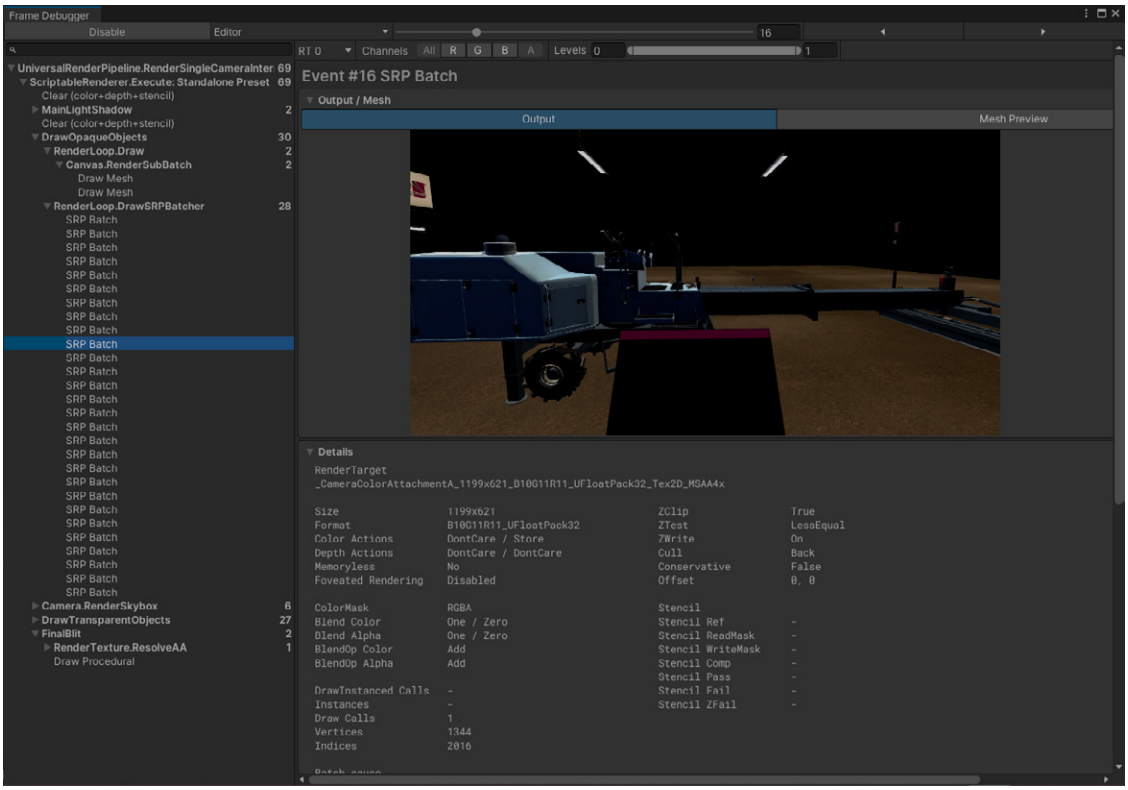
Performance profiling checklist:

- **CPU performance**

  - ✔ Monitor high CPU usage, particularly in **Rendering** or **Scripts** sections.
  - ✔ Watch for excessive Garbage Collection (GC Alloc) values. Repeated allocations may signal avoidable runtime instantiations-use object pooling, preloading, and pooled memory arrays to minimize spikes.
  - ✔ Apply **object pooling** techniques to minimize runtime instantiations and reduce garbage collection spikes.

- **GPU performance**

  - ✔ Enable the **GPU Profiler** for in-depth insights beyond standard CPU-bound metrics.
  - ✔ On standalone VR devices, use the relevant external tools like **Meta Quest Developer Hub** or **Android GPU Inspector** to capture detailed GPU metrics.
  - ✔ Activate **GPU Frame Timing** in Unity to detect rendering stalls affecting frame rate stability.



Unity Somero Enterprises, Inc. frame debugger building view. Image courtesy ForgeFX.

© 2025 Unity Technologies

Pro Tip: Profile during active, complex simulation scenes to capture real-world usage patterns rather than relying solely on simplified test environments.

Profiling early—and continuously—helps us detect performance regressions before they take root. We typically start by reviewing CPU and GPU time per frame, then drill down into specific areas such as script execution and rendering overhead. Whether it's physics calculations overloading the main thread, excessive garbage collection, or shader complexity straining the GPU, the Profiler acts as an early warning system that guides our optimization efforts.

An often overlooked advantage of early profiling is uncovering performance issues tied to specific user flows. For example, complex machinery training scenarios where users rapidly switch between interactive panels and heavy physics environments can reveal hidden spikes. Capturing these moments in profiling sessions ensures no surprises in final builds.

Going Deeper: GPU profiling and rendering analysis

Efficient rendering is only part of the picture. Main thread congestion is a common bottleneck in simulation projects, especially those involving complex physics interactions. To address this, we begin by ensuring the rendering workload is as lean as possible.

Unity's default Profiler provides an excellent foundation, but deeper GPU insights often require external tools. For standalone VR devices, we rely on solutions like Meta Quest Developer Hub and Android GPU Inspector to expose fine-grained GPU performance metrics. Enabling GPU frame timing in Unity further reveals subtle rendering stalls that may otherwise escape notice.

Rendering optimization checklist:

- **Draw call reduction:**
  - ✔ Use **static and dynamic batching** where applicable.
  - ✔ Minimize **material variants** to maintain batch efficiency.
  - ✔ Enable **GPU instancing** for objects that share materials.
- **Shader optimization:**
  - ✔ Simplify shader complexity to reduce GPU load.
  - ✔ Minimize **shader variant usage** to prevent batching breaks.

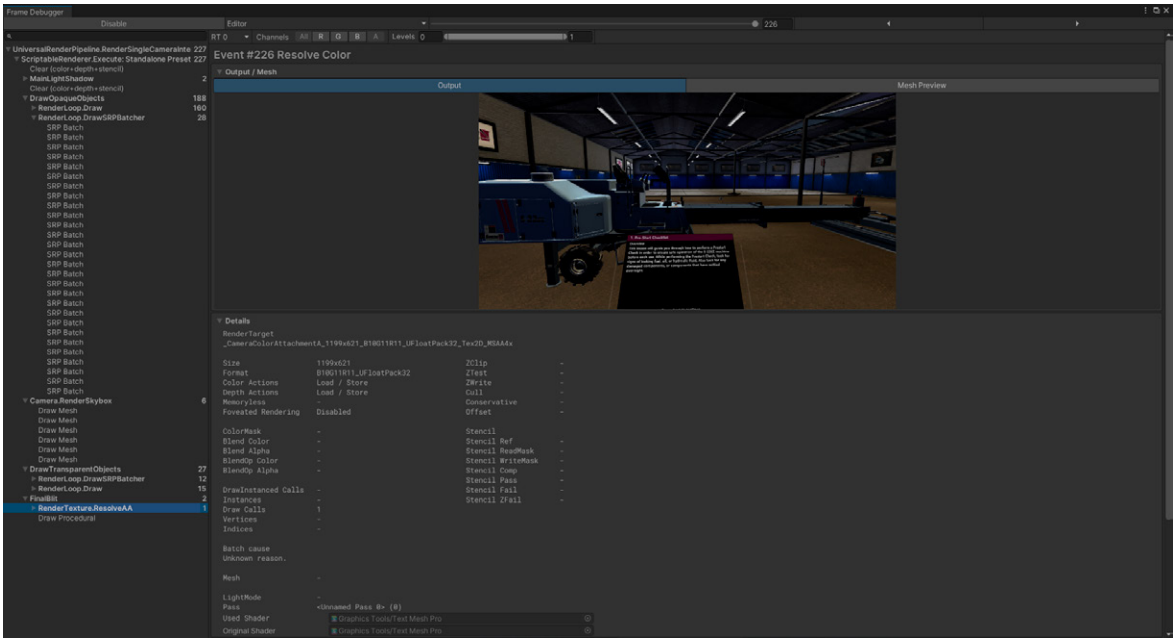
Pro Tip: Before making shader changes, profile the most complex scene first to measure the true impact of shader optimizations on frame rate and GPU load.

© 2025 Unity Technologies

Advanced texture and material optimization:

- Use texture atlases to reduce draw calls by consolidating multiple object textures into a single material.
- Apply texture compression based on target platform requirements (e.g., ASTC for Android, DXT1/5 for Windows).
- Use channel mapping to pack multiple grayscale maps (e.g., metallic, roughness, AO) into RGBA channels of a single texture; requires custom shaders.
- Always verify draw call count on target hardware-batching may behave differently in Editor vs. device builds.

Pro Tip: Unity's Frame Debugger is invaluable for investigating batching behavior-but always validates GPU performance on-device, especially for standalone VR targets, where driver-level behavior and thermal limits affect batching outcomes.



Unity Somero Enterprises, Inc. frame debugger processing view. Image courtesy ForgeFX.

In practice, these optimizations have yielded substantial improvements. In one instance, batching improvements alone reduced draw calls by 40%, with corresponding gains in frame delivery consistency.

© 2025 Unity Technologies

Lighting decisions also play a critical role in GPU performance. Overly complex real-time lighting setups, especially in industrial simulations with reflective materials, can overwhelm mobile GPUs. Whenever feasible, baking lighting for static objects can drastically reduce runtime costs, freeing up resources for dynamic elements that truly require real-time computation.

It's also worth evaluating your texture resolutions. While high-resolution textures improve visual fidelity, they place additional strain on the GPU and memory bandwidth.

Pro Tip: Mipmapping and texture atlasing can help reduce sampling costs and keep texture lookups efficient.

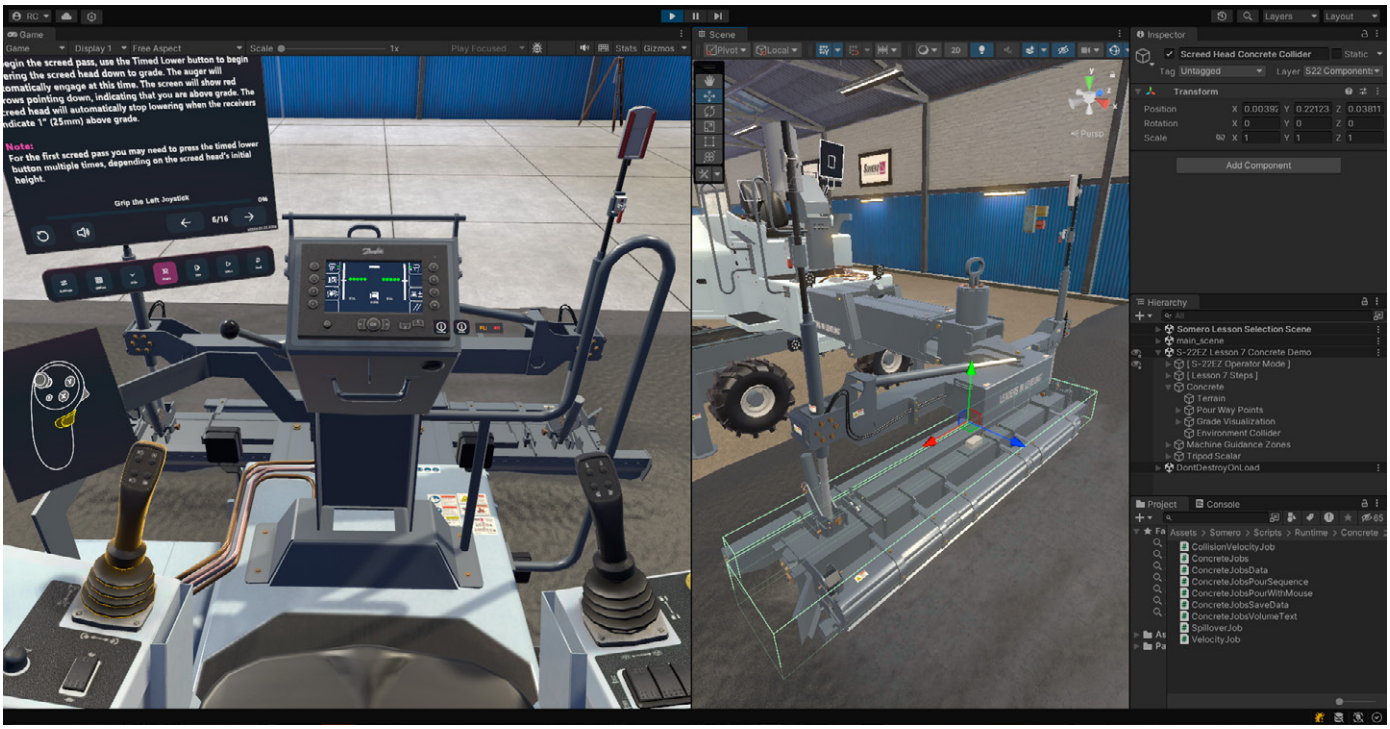
Why it matters

Performance tuning is not a one-time effort. It's an integrated process-profiling early, testing often, and validating continuously. With Unity's suite of profiling and debugging tools, we move from reactive optimization to proactive design, shaping systems that meet the demands of both hardware and learners.

Ultimately, performance is the foundation of presence. It's what turns a simulation from a visual demonstration into a believable, embodied experience. When systems run smoothly, users stop thinking about the technology-and start thinking about the task at hand. That's when training becomes transformation.

© 2025 Unity Technologies

Structuring performance with Unity's job system



Somero Enterprises, Inc. screeding in action view. Image courtesy ForgeFX.

Once rendering performance has been optimized and GPU overhead brought under control, the next frontier in simulation performance is the main thread. Unity's job system provides

© 2025 Unity Technologies

a structured, scalable way to distribute CPU-intensive work-such as physics calculations, procedural logic, or environmental updates-across multiple cores. In complex, data-driven training systems, this can dramatically improve responsiveness and free the main thread for critical simulation logic.

Using Unity's job system for parallel processing

Main thread congestion is a frequent bottleneck in real-time simulations, particularly in scenarios involving procedural environments, physics systems, or large volumes of dynamic data. Unity's Job System offers a structured way to offload these computational tasks to other CPU cores-improving throughput without sacrificing determinism.

Unity's multithreaded architecture allows simulation developers to schedule compute-heavy work-such as environmental updates, AI state evaluation, or multi-agent systems-without blocking critical simulation logic. The result is a more responsive system that scales across devices and remains performant under load.

Applying the Unity job system

Using **IJob** and **IJobParallelFor**, we structure compute-heavy operations into parallel jobs while managing data through **NativeArray** to avoid race conditions. The **Burst Compiler** enhances this further, translating C# jobs into highly optimized machine code that reduces CPU load significantly.

Job System best practices:

- **Structure jobs for independence.** Keep dependencies on the main thread minimal to avoid delays.
- **Use Native Containers.** Leverage **NativeArray** or **NativeList** for memory management without race conditions.
- **Apply the Burst Compiler.** Compile jobs into optimized native code for faster execution.
- **Schedule large tasks with IJobParallelFor.** Distribute workloads like position updates or force calculations across cores.
- **AI Behavior Optimization.** Offload heavy tasks like sensor checks and data processing to jobs, while keeping real-time decision logic on the main thread to maintain responsiveness.
- **Use job-based terrain sampling and heightmap manipulation** for fluid-like materials where dynamic surface updates are required.

Pro Tip1: When simulating soft or deformable materials like concrete, clamp terrain height values and apply smoothing jobs to prevent unnatural spikes during abrupt interactions.

© 2025 Unity Technologies

Pro Tip 2: Test jobs incrementally. Start with smaller workloads and gradually scale complexity, verifying thread safety and performance gains at each stage.

One practical example is in handling physics-intensive simulations where fluid dynamics or material flow needs real-time responsiveness. By offloading pre-calculation of material displacement to jobs and only committing final transformations back to the main thread, we preserved fluid motion fidelity without taxing the CPU.

In practice, we use Unity's job system extensively to simulate fluid-like material behavior in construction training scenarios. One job might analyze terrain heightmap changes and spill excess "concrete" to adjacent regions, preserving performance even when sudden forces (like a virtual dump truck collision) displace large amounts of material. Other jobs track collider overlap and dynamically update roughness/smoothness values via alpha blending on terrain textures. These secondary thread calculations help preserve the illusion of fluid interaction-without stalling the main thread or causing observable frame drops.

Understanding job scheduling and execution order is also key to unlocking peak performance. Take the time to visualize the dependencies between jobs and main thread tasks to eliminate unnecessary sync points and minimize thread stalls.

From experience, the key to success with multi-threaded processing is maintaining clear boundaries between the job system and the main thread. Jobs should operate independently wherever possible, passing only essential data back once computations are complete. This approach preserves thread safety and prevents unnecessary delays.

To maintain consistent frame delivery, we balance multi-threaded compute with thermal constraints by offloading physics logic while minimizing unnecessary GPU load. We also monitor GPU usage in real time using tools like **OVR Metrics** and **RenderDoc**, helping us identify spikes that may lead to throttling or overheating.

Integrating automated testing for performance validation

Optimization doesn't end with manual profiling. Integrating Unity's **Performance Test** package into our CI/CD pipeline enables automated benchmarking of frame times, CPU usage, and overall performance consistency across builds.

© 2025 Unity Technologies

Somero Enterprises, Inc. screeding alternative view. Image courtesy ForgeFX.

Performance testing workflow:

- **Establish fixed test scenes** with predetermined object layouts and interactions.
- **Automate benchmarks** using Unity's **PerformanceTest** package.
- **Include stress tests** to validate performance under high-load conditions.
- **Monitor regressions** automatically after each commit via CI/CD integration.

Pro Tip: Maintain fixed testing conditions-such as specific scenes and interaction flows-to ensure consistent benchmark comparisons across test iterations.

Furthermore, make it a practice to establish baseline performance metrics early in development. This provides a reference point for evaluating the impact of new features or optimizations, ensuring consistent quality throughout the development cycle.

Automation not only increases coverage but also enables your team to respond to regressions rapidly. Setting up notifications for performance dips helps ensure any regressions are addressed in the same development cycle, preventing small issues from compounding over time.

For VR applications, consistency is paramount. Fixed environments ensure reliable performance comparisons across iterations. These automated tests help us catch regressions early, often within hours of a new commit, keeping our performance targets on track.

© 2025 Unity Technologies

Debugging for precision optimization

Unity's debugging tools complement our profiling efforts by offering fine-grained diagnostics. The **Frame Debugger** lets us step through individual rendering passes to identify overdraw, redundant operations, or shader inefficiencies. The **Memory Profiler** highlights allocation spikes and persistent memory usage that could lead to performance degradation. For physics-heavy projects, Unity's **Physics Debug Visualization** pinpoints costly collision checks and rigidbody interactions.

Debugging techniques:

- Use the **Frame Debugger** during play mode to inspect draw calls frame by frame.
- Optimize transparency and apply **Occlusion Culling** to reduce overdraw.
- Fine-tune physics layers and **Fixed Timestep** intervals (e.g., from 0.02 to 0.016) to balance CPU load and simulation accuracy.
- Apply **Physics.IgnoreCollision** to eliminate unnecessary collision checks.
- Simplify **Rigidbody** complexity to reduce physics computation overhead.
- Leverage the **Memory Profiler** to track GC allocations and persistent memory usage.

Pro Tip: When debugging, always recreate performance issues in a controlled environment first. This isolates variables and ensures fixes address root causes rather than symptoms.

Incremental improvements in these areas can deliver substantial cumulative gains. In a heavy-equipment training simulator, these combined optimizations led to a 35% increase in frame rate stability and a 25% reduction in CPU load—tangible improvements that directly enhanced trainee immersion and operational realism.

Why it matters

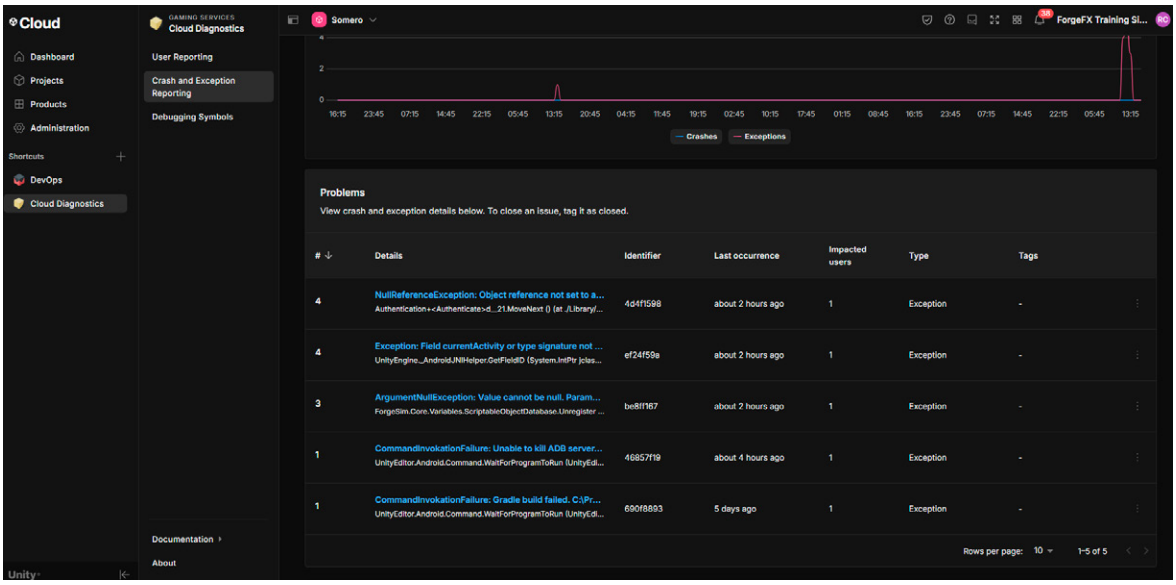
Performance optimization is an ongoing process. It's an iterative cycle of profiling, analysis, and fine-tuning that persists throughout the development lifecycle. By embedding Unity's performance tools into our workflows: from early-stage profiling to automated regression testing, we build simulations that not only meet technical standards but genuinely elevate the training experience.

Ultimately, performance is more than a technical metric. It's the invisible thread that preserves immersion, sustains engagement, and ensures every moment in the simulation translates into effective, memorable learning.

Consider experimenting with the Vulkan graphics API on supported devices such as Meta Quest. Vulkan can offer improved multi-threading, better GPU utilization, and lower driver overhead compared to OpenGL ES, especially beneficial in complex or visually intensive scenes.

© 2025 Unity Technologies

Streamlining CI/CD and development pipelines with Unity Cloud



Unity Cloud diagnostics report.

Modern training simulation projects demand rapid iteration, stable deployment pipelines, and seamless collaboration across teams. Unity Cloud provides an integrated platform that addresses these needs-enabling version control, automated builds, test orchestration, performance monitoring, and asset organization through a unified interface.

© 2024 Unity Technologies

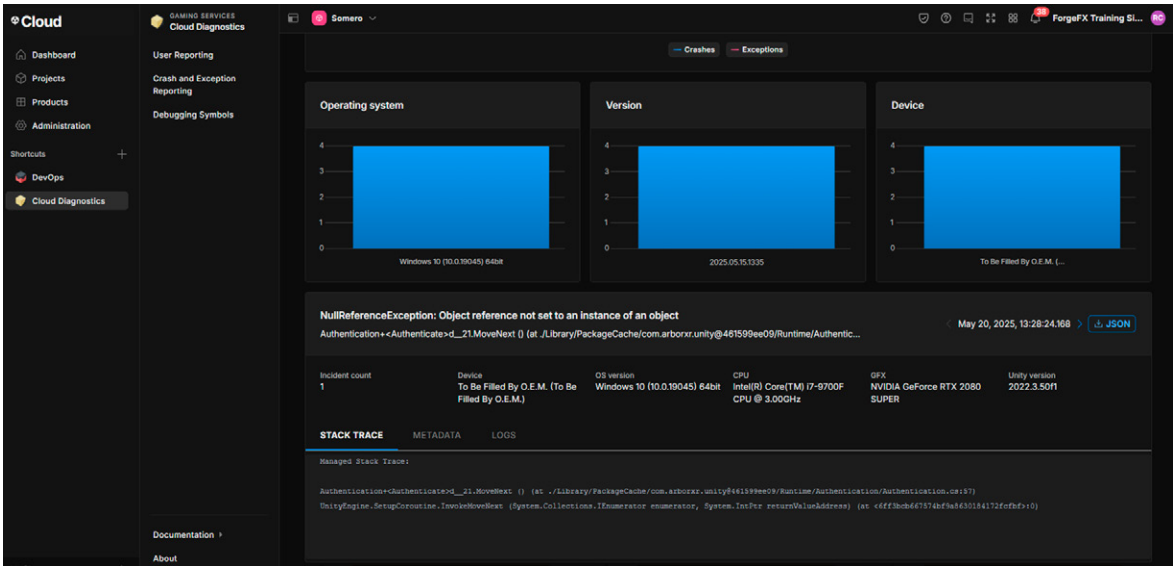
At ForgeFX, Unity Cloud Build plays a central role in our CI/CD strategy, helping us maintain high quality across multi-platform XR and desktop builds. While we do not use Unity Asset Manager or Version Control directly, our workflows are fully aligned with Unity Cloud's deployment infrastructure. Unity DevOps tools such as **Cloud Diagnostics** and **Build History** give our non-developer stakeholders actionable visibility without requiring direct access to the project source. Paired with the Unity Test Framework, Unity Cloud Build empowers us to detect regressions early and ship confidently.

Understanding common development and delivery challenges

Building and maintaining enterprise-grade training simulations means managing complexity across platforms, assets, and teams. Without cloud-based automation and observability, developers face persistent friction:

- Manual build processes introduce risk and delay.
- Version drift across platforms leads to inconsistent outputs.
- Test coverage gaps cause regressions to be discovered late in the cycle.
- Disconnected tools fragment developer visibility.

For training simulations, these issues scale quickly. A late-breaking UI regression or an asset reference failure can invalidate a release milestone, stall QA, or block stakeholder review.



Unity Cloud diagnostics report.

Implementing Unity Cloud build across simulation projects

Unity Cloud Build allows us to automate and parallelize builds across target platforms. This accelerates delivery, reduces developer overhead, and improves validation speed. Projects that target standalone VR headsets, tablets, and desktop simultaneously benefit from centralized configuration and remote orchestration.

Unity Cloud Build Setup Checklist:

- ✔ Connect the project repository to Unity Cloud via the Cloud Dashboard.
- ✔ Create build targets for each platform (Windows, Android/Quest, WebGL).
- ✔ Configure build scripting symbols and scene lists per platform.
- ✔ Schedule nightly or post-merge builds for high-priority branches.
- ✔ Set up build notifications via Slack or email for transparency.
- ✔ If you're using GitHub or GitLab, configure webhook integrations to trigger builds on merge, tag, or commit events.

Pro Tip 1: Use pre-export build validation scripts to catch errors like missing references or unresolved dependencies before a cloud build consumes compute resources.

Pro Tip 2: Use [Category("FeatureName")] attributes to organize large test libraries and enable targeted execution per feature or build stage.

This workflow has reduced integration time during final QA windows by over 40%, especially for simulation builds requiring multiple asset bundles or dynamic lesson content.

Supporting quality through Unity Test Framework integration

Although a key part of ForgeFX's QA workflow focuses on manual UX validation and device-specific QA, we also integrate Unity's Test Framework into our Cloud Build pipeline for repeatable automated testing against most pull requests and merges in key branches. Unit and integration tests validate critical systems such as lesson progression, simulation state machines, and UI logic, and ensure that critical issues are detected at the earliest possible moment when it is easiest to resolve or roll back changes.

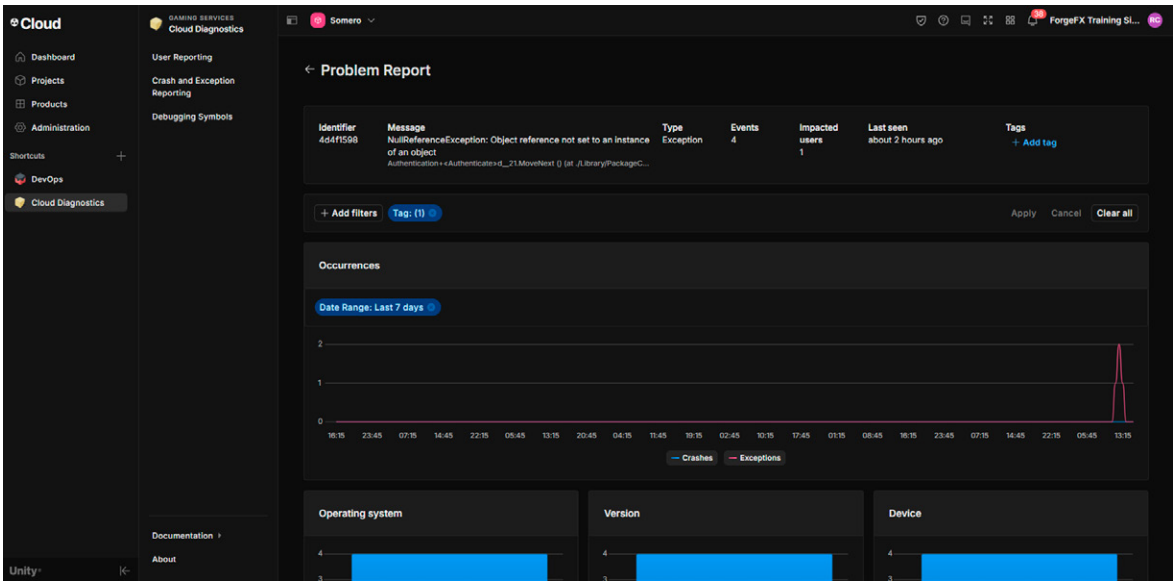
© 2024 Unity Technologies

Simulation Testing Checklist:

- ✔ Include EditMode tests for component integrity and prefab validity to check for missing references or incorrect / game breaking configurations.
- ✔ Use PlayMode tests to validate lesson flow, scoring, and user feedback.
- ✔ Log errors and warnings when key issues are detected in runtime code – play mode tests can be triggered to fail when these errors are logged to allow for an easy increase in test coverage
- ✔ Run test suites in local builds and enable Test Runner integration in Unity Cloud.
- ✔ Track failed test cases by simulation module and prioritize resolution based on regression risk.
- ✔ Integrate automated unit test results into Slack or email to ensure test results are seen and acted on promptly

Pro Tip: Maintain per-module test assemblies (e.g., Tests.UI, Tests.Lesson, Tests.Input) to improve clarity and allow partial reruns during iterative feature testing.

This approach has reduced QA-reported functional bugs by more than 50% in simulation projects using structured lesson progression or dynamic role gating.



Unity Cloud diagnostics specific issue.

© 2024 Unity Technologies

Use Case: Multi-platform manufacturing equipment trainer

In a manufacturing training simulator developed for both standalone VR and Windows desktop, Unity Cloud Build enabled simultaneous platform builds from a single repository. This ensured consistent versioning across form factors while allowing stakeholders to preview in their preferred environment.

We configured platform-specific build settings, preloaded test scenarios, and validation test scripts for:

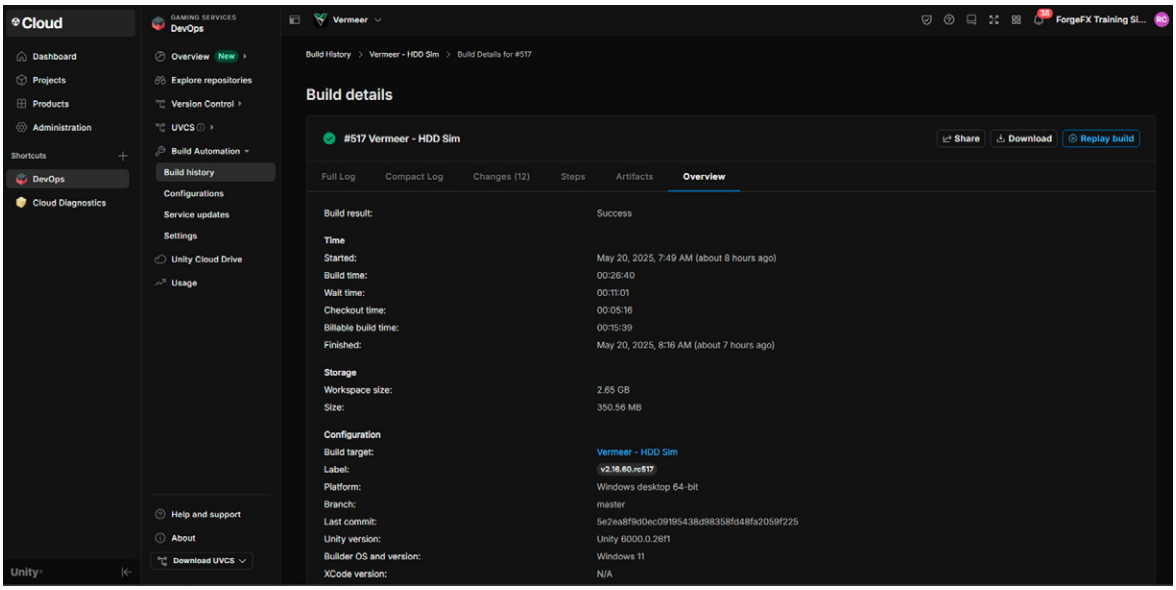
- Input mapping verification per hardware platform
- Lesson content availability and asset reference checks
- Scene load completion metrics and memory usage diagnostics

Results:

- Improved build delivery time by 60%
- Reduced test deployment errors from 3–4 per week to near zero
- Enabled client feedback on training content within 24 hours of code freeze

This pipeline became the foundation for a shared staging environment used by internal teams, QA, and external reviewers.

Enhancing collaboration with Unity Cloud infrastructure



Vermeer Corporation Unity Cloud dashboard specific build.

© 2024 Unity Technologies

In addition to build automation and testing, Unity Cloud provides shared infrastructure that improves traceability and team alignment. While ForgeFX uses external source control (e.g., Git), Unity Cloud's dashboard integrations allow our producers and QA leads to monitor build progress, triage issues, and trigger rebuilds without interrupting engineering workflows.

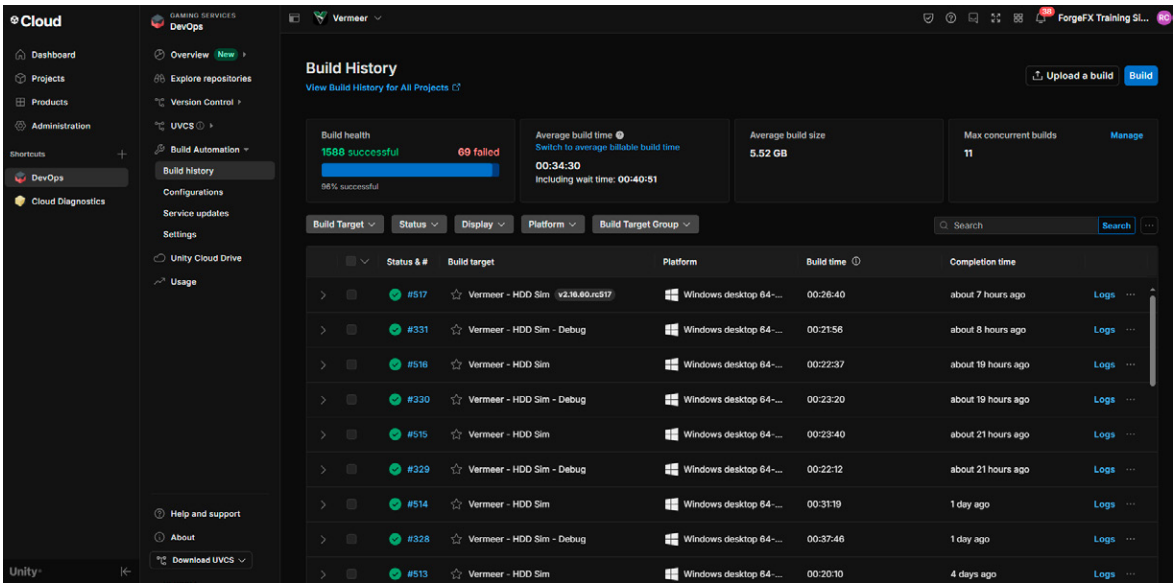
Build artifacts are retained in a centralized location with platform and time-tagged metadata, simplifying release candidate approval and historical traceability.

Team Workflow Integration Checklist:

- ✔ Automate posting of build results and changelogs to team Slack channels.
- ✔ Maintain internal wiki with build target status and validation metrics.
- ✔ Use build labels to tag stable milestones and QA-verified versions.
- ✔ Track build duration, pass/fail history, and output package sizes over time.
- ✔ Review recent changes average build times or build sizes change unexpectedly

Pro Tip 1: Encourage producers and QA leads to use Unity Cloud's build history view as a diagnostic timeline for performance, stability, and asset bloat regressions.

Pro Tip 2: Most issues in multi-platform simulations stem from subtle prefab serialization or addressables conflicts—frequent history reviews help isolate these regressions early.



Vermeer Corporation Unity Cloud dashboard.

© 2024 Unity Technologies

Cloud build best practices for simulation projects

- Maintain small, modular build targets to reduce runtime load.
- Include platform-specific scenes only when needed.
- Monitor cloud build duration and optimize for long-term stability.
- Validate prefabs, serialized assets, and addressables with each commit.
- Regularly review test logs to spot trends in unstable systems.

Pro Tip 1: Use Unity Cloud's distributed caching to accelerate frequent builds-particularly useful in projects with large geometry imports or multi-gigabyte asset bundles.

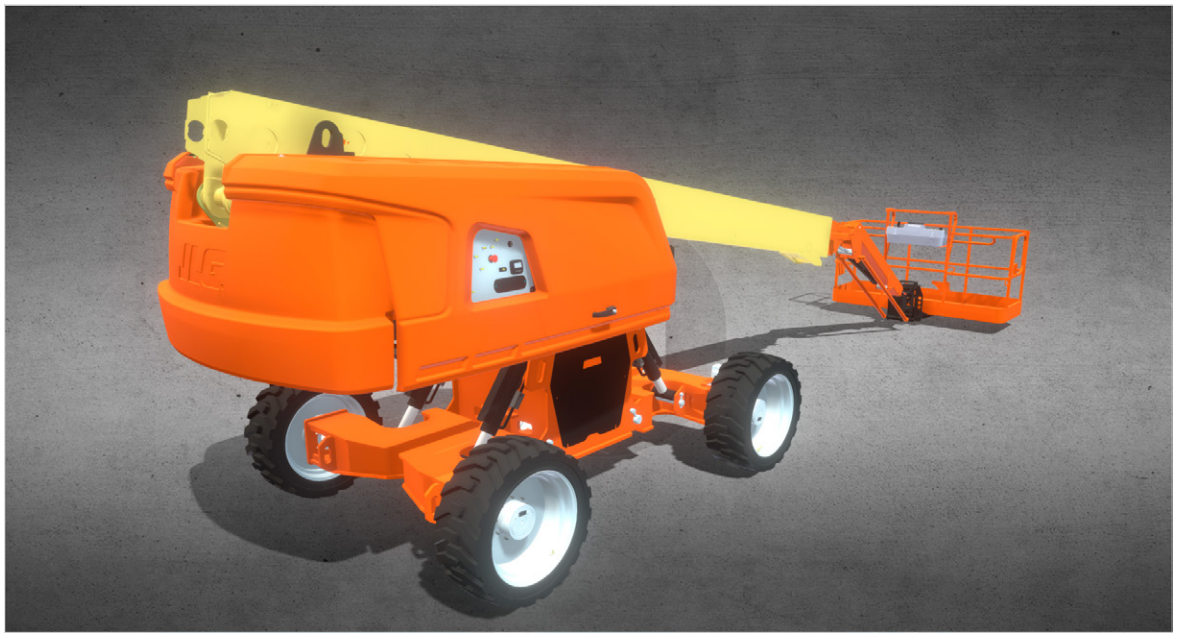
Pro Tip 2: Also enable incremental asset import in Unity project settings to reduce pre-build time across developers and cloud agents.

Why it matters

Unity Cloud Build offers ForgeFX a dependable, transparent pipeline for simulation development across platforms and teams. When combined with targeted test coverage, modular configuration, and clean build automation practices, Unity Cloud becomes a core asset in reducing cycle time and ensuring build consistency. For organizations developing complex, multi-platform training simulations, it provides a proven path to high-quality delivery at scale.

© 2024 Unity Technologies

Optimizing CAD workflows with Unity Asset Transformer Studio,Toolkit, and SDK



JLG model. Image courtesy ForgeFX.

Training simulations often begin long before a line of code is written. For industries that rely on highly engineered equipment, such as aerospace, automotive, and heavy machinery-simulation, fidelity depends on accurate, optimized geometry. That's why Unity Asset Transformer Studio, Unity Asset Transformer Plugin, and Unity Asset Transformer SDK have become indispensable to ForgeFX's asset pipeline. These tools allow us to transform complex CAD models into performant, real-time training assets tailored for Unity.

© 2024 Unity Technologies

The Unity Asset Transformer product suite provides an essential bridge between engineering and simulation. It enables us to preserve part hierarchies, material mappings, and design intent while stripping unnecessary detail, reducing polygon count, and enforcing naming standards required for interactivity. It also ensures compatibility with Unity's coordinate system, which differs from that of most CAD sources (Z-up vs Y-up). By building a robust CAD pipeline around Unity Asset Transformer, we ensure simulation accuracy without compromising runtime performance.

Understanding CAD integration challenges in simulation development

Unlike game development, simulation-based training often begins with source geometry from mechanical or industrial CAD systems. These models present multiple technical challenges:

- Unoptimized geometry with millions of triangles and high complexity
- Inconsistent part naming, nested hierarchies, or hidden dependencies
- Incompatible formats for Unity's import system
- Misaligned materials, scale issues, or redundant data

For real-time training, these problems affect both performance and usability. Excessive triangle counts lead to poor frame rates, particularly on standalone XR hardware. Poorly structured assets complicate scripting and interactivity. And mismatched materials introduce visual noise that undermines realism.

Vermeer Corporation model. Image courtesy ForgeFX.

© 2024 Unity Technologies

Implementing a CAD pipeline with Unity Asset Transformer suite

ForgeFX uses Unity Asset Transformer Studio to preprocess high-fidelity CAD models and then imports them via Unity Asset Transformer Toolkit into Unity for staging, interaction scripting, and optimization. For automation-heavy workflows or batch imports, we rely on Unity Asset Transformer SDK to enforce standardized geometry treatment across asset types.

Unity Asset Transformer asset pipeline checklist:

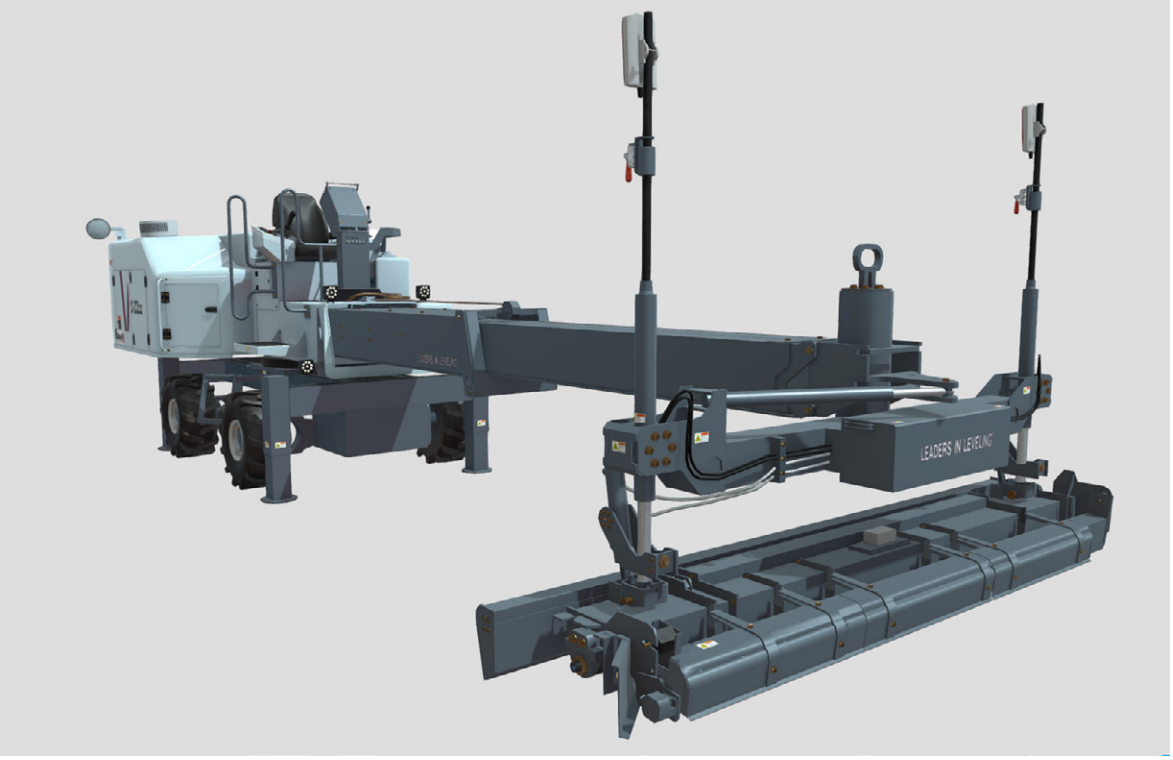
- ✔ Import STEP, IGES, or JT files into **Unity Asset Transformer Studio** for decimation and part cleanup
- ✔ Apply tessellation presets matched to target platform (e.g., mobile VR, PC)
- ✔ Optimize hierarchy by flattening or merging assemblies where applicable
- ✔ Reassign or re-map materials and verify UV integrity
- ✔ Export to FBX or Unity-ready format using export rules
- ✔ Import into Unity using **Unity Asset Transformer Toolkit** to preserve hierarchy and metadata

Pro Tip 1: When decimating assemblies with moving parts, lock pivot points and component origins in Unity Asset Transformer Studio (specifically its simplification and pivot management modules) before simplification—this ensures correct behavior during runtime animation or IK-based interaction.

Pro Tip 2: Use the scene tree search and rule-based selection in Unity Asset Transformer Automate (or Unity Asset Transformer Studio's rule-based processing module, depending on deployment) to batch-process redundant features (like threads, fillets, or internal volumes) that are invisible during simulation but costly at runtime.

This workflow allows us to reduce asset complexity by 80–90% while preserving functional visual fidelity. It's particularly critical for simulations that require runtime physics, high frame rate, or XR hand interaction.

© 2024 Unity Technologies



Somero Enterprises, Inc. Enterprises, Inc. model. Image courtesy ForgeFX.

Extending Unity Asset Transformer SDK for Automation and Consistency

In complex simulation projects with dozens of parts and recurring product variants, manual import and cleanup quickly becomes unscalable. We use Unity Asset Transformer SDK to programmatically automate asset ingestion, naming enforcement, and export workflows.

Typical SDK scripts handle:

- Tessellation parameter assignment based on part metadata
- Layer filtering (e.g., remove construction geometry, fasteners)
- Merging low-priority parts into single meshes
- Applying metadata tags for Unity scripts (e.g., interactable, highlightable)

CAD optimization automation checklist:

- ✔ Define material presets and naming conventions
- ✔ Automate transform origin resets per part
- ✔ Batch export assemblies to Unity-ready FBX using shared configuration
- ✔ Generate QA reports with triangle counts, missing UVs, and hierarchy checks

© 2024 Unity Technologies

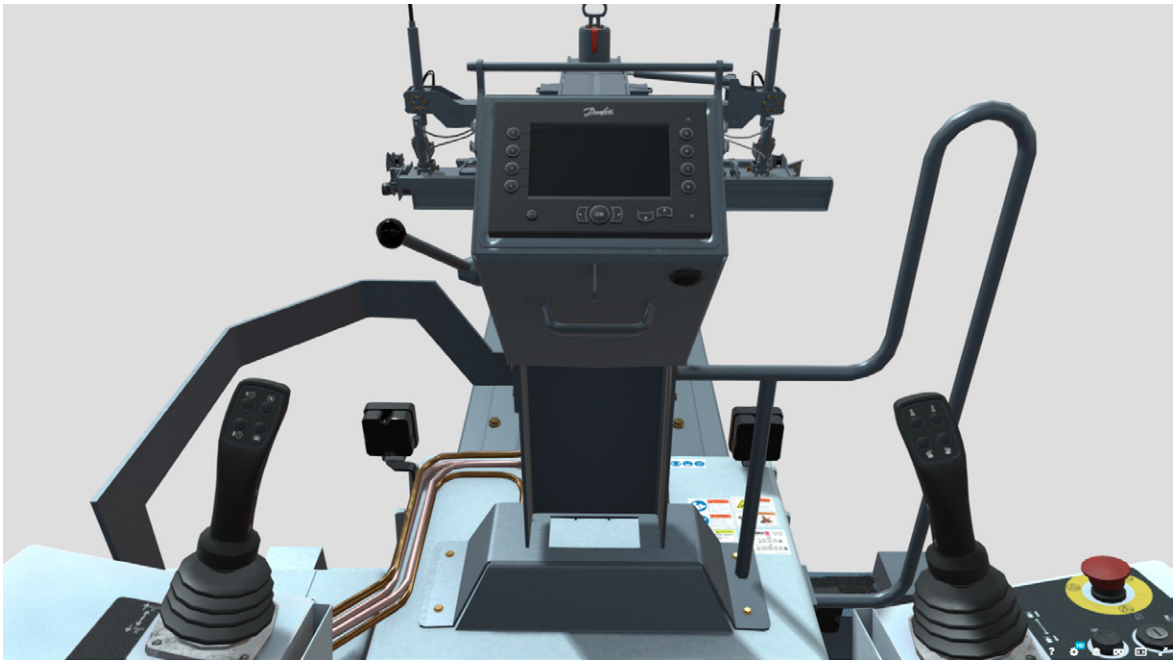
Pro Tip 1: Use Unity Asset Transformer SDK to generate a manifest of simulation-relevant parts (e.g., controls, sensors, access points)—this improves communication between 3D artists and Unity developers, reducing rework and mislabeling.

Pro Tip 2: The Unity Asset Transformer SDK runs in a Python-based environment, enabling integration into broader DCC or PLM systems if enterprise pipelines require cross-tool orchestration.

This approach ensures consistency across asset contributors and project phases, reducing friction during asset integration and feature development.

Use Case: Immersive VR mission rehearsal simulator for tactical environments

In one VR training simulation developed for tactical readiness and mission rehearsal, we imported and optimized high-complexity CAD models representing realistic laboratory environments and specialized handheld detection equipment. Using Unity Asset Transformer Studio, we reduced a multi-million triangle environment, derived from LiDAR scans and photogrammetry, to runtime-ready geometry without losing fidelity in critical interactive zones. Assemblies were flattened and cleaned to support AI-guided procedural workflows, and assets such as chemical detection tools were tagged for interactive scripting in Unity.



Somero Enterprises, Inc. model alternate angle. Image courtesy ForgeFX.

© 2024 Unity Technologies

Results:

- Reduced geometry preparation time by over 70%
- Achieved smooth runtime at 72 FPS on standalone VR headsets
- Enabled real-time AI-guided interactions within high-fidelity simulation spaces through clean, optimized asset structure

Without the Unity Asset Transformer Studio Toolkit, the scope and performance goals would have required weeks of manual decimation and reprocessing.

Integrating Unity Asset Transformer with Unity development workflows

Once Unity Asset Transformer-processed assets are in Unity, our teams script interactivity, attach lesson logic, and apply runtime materials. The clean hierarchy and reduced triangle count accelerate everything from UI highlight callouts to collision-based interactions.

We also apply Unity's LOD tools and baked lighting systems more efficiently due to predictable geometry structure and scale.

Unity Integration Checklist:

- ✔ Validate pivot alignment for all animated or interactable parts
- ✔ Use standardized naming to link Unity scripts to target meshes
- ✔ Group similar parts under shared prefabs for multi-role training reuse
- ✔ Apply LOD groups where polygon reduction allows performance gain
- ✔ Leverage addressables for dynamic asset loading by scenario or lesson stage
- ✔ Decouple behavior logic from raw mesh hierarchies by mirroring key structures in clean logic prefabs: this protects gameplay components from breakage during CAD updates.

Pro Tip 1: Avoid attaching logic directly to imported mesh objects. Instead, create clean prefabs that mirror the CAD hierarchy and hold all scripts and triggers—so model updates never break core behavior.

Pro Tip 2: Maintain a shared prefab library of Unity Asset Transformer-processed assemblies across simulation projects-this improves asset reuse and reduces onboarding time for new developers.

© 2024 Unity Technologies

Pro Tip 3: Use Unity's Mesh Compression and Static Batching for final optimization after import, especially when LOD switching isn't sufficient for performance targets.

CAD pipeline best practices for simulation projects

- Build asset ingestion rules around project scope (e.g., high vs low fidelity)
- Work with SMEs to define which parts require interactivity
- Always validate mesh normals and pivot placement before Unity export
- Coordinate early with developers to match material naming with shader use
- Maintain audit logs of imported vs used parts to reduce simulation bloat

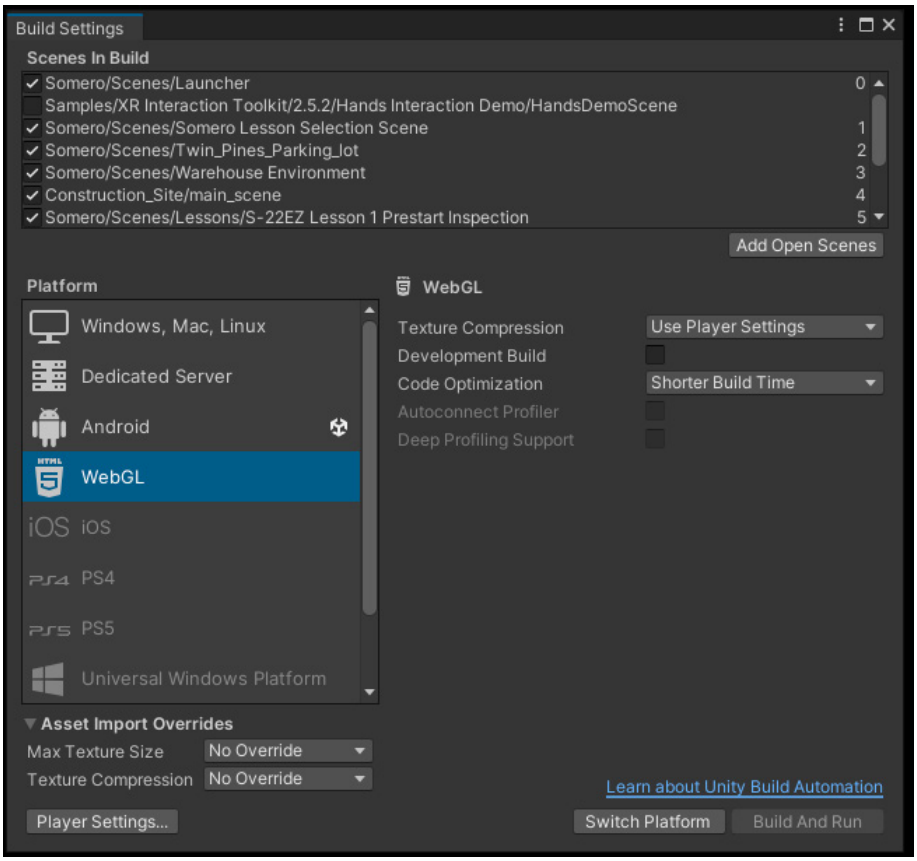
Pro Tip: Don't optimize everything, segment assets by training value. Parts irrelevant to interaction can be grouped or replaced with impostors to preserve budget for simulation-critical geometry.

Why it matters

Unity Asset Transformer tools are essential for bringing real-world CAD models into high-performance Unity-based simulations. From decimation to automation, they enable ForgeFX to deliver accurate, responsive, and visually coherent training environments that meet the demands of XR hardware and enterprise expectations. In training simulation projects, Unity Asset Transformer is the critical link between engineering precision and simulation scalability; transforming massive industrial datasets into immersive, instructional assets ready for real-time deployment.

© 2024 Unity Technologies

Deploying high-fidelity web-based simulations with WebGL, WebGPU, and mobile web



Somero Enterprises, Inc. Unity build settings WebGL installed.

© 2024 Unity Technologies

Web-based deployment has become a cornerstone of accessible training and review solutions; especially when device variability, installation barriers, and stakeholder availability are concerns. Unity's support for WebGL, WebGPU (via experimental backend), and mobile browser platforms allows developers to deliver high-performance simulations without requiring native application installation.

Web-based builds follow a "build once, deploy anywhere" model, eliminating the need for separate platform-specific distributions. This streamlines development, reduces maintenance overhead, and makes wide deployment more cost-effective across a diverse range of devices.

These technologies extend simulation access to tablets, laptops, and phones, lowering the barrier for product stakeholders, remote trainees, and field users who require a lightweight yet interactive experience. Web-based delivery accelerates client feedback, distributes review builds, and even runs light-touch training simulations directly from the browser.

Understanding the challenges of web-based simulation deployment

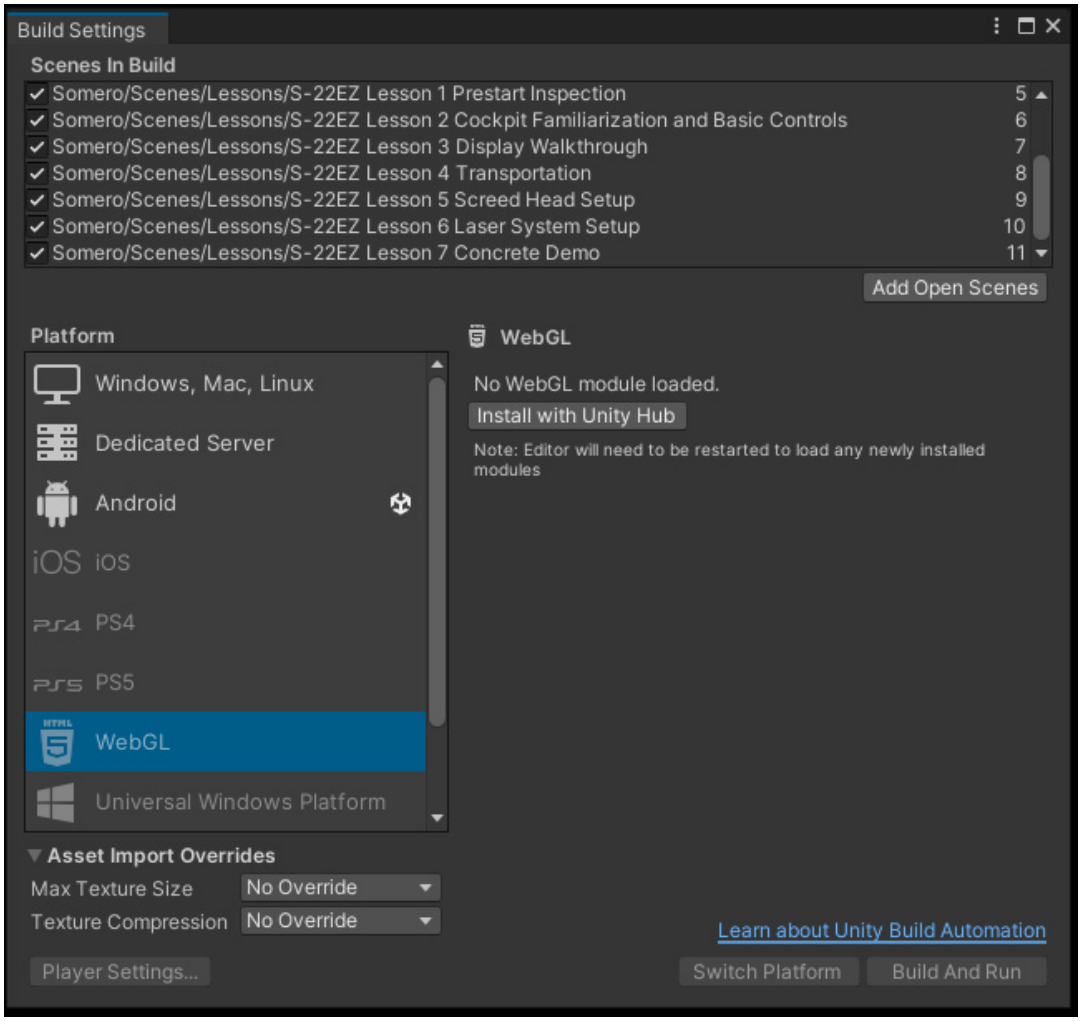
Web deployment introduces its own constraints compared to native builds. These include:

- GPU and memory limitations on mobile browsers
- Load time sensitivity due to large asset bundles
- Limited access to multithreaded features (e.g., no Unity Jobs support in WebGL)
- Shader model differences that affect rendering quality

While Unity's Burst Compiler and Job System are powerful for native builds, these are not currently supported in WebGL. Developers must rely on single-threaded or coroutine-based solutions in browser targets.

Without proper optimization and cross-platform testing, browser builds can suffer from degraded visual fidelity, sluggish interactivity, and platform-specific bugs-compromising the user experience and diluting the value of the simulation. Developers should test across multiple devices and browsers to catch rendering issues early-for instance, those related to shader compatibility and GPU behavior.

© 2024 Unity Technologies



Somero Enterprises, Inc. Unity build settings WebGLnot installed.

Implementing a web deployment strategy with Unity

We recommend structuring Unity projects to support WebGL builds from the start. This will help avoid runtime APIs and pipeline features that may be incompatible with Web platforms. When possible, use the WebGPU backend for improved performance, especially for larger model reviews or dynamic content.

Web deployment setup checklist:

- ✔ Enable WebGL or WebGPU build target in Unity's Build Settings
- ✔ Avoid using incompatible features like compute shaders or threading
- ✔ Compress assets using Brotli or Gzip to reduce load times
- ✔ Use **Addressables** to stream large content bundles incrementally

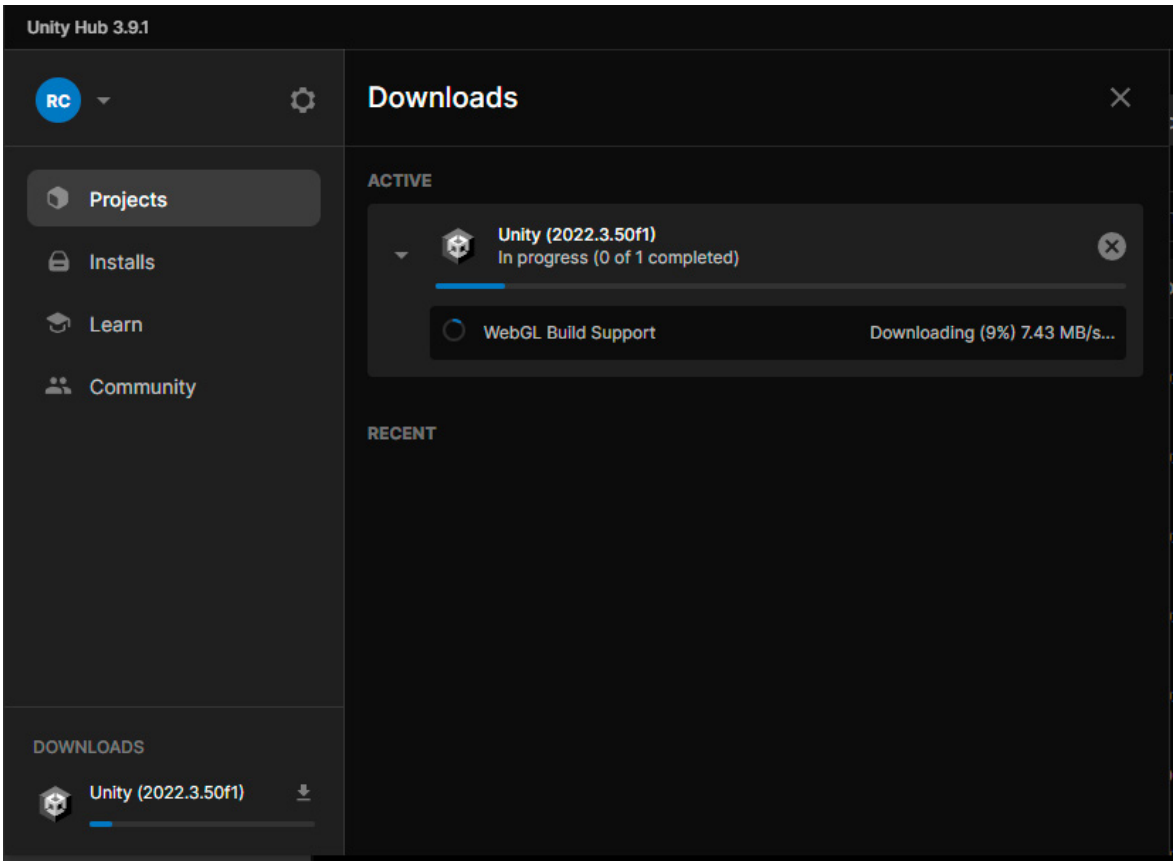
© 2024 Unity Technologies

- ✔ Implement custom loading UI to manage perception during boot-up
- ✔ Profile WebGL builds using browser-based developer tools
- ✔ Configure LOD settings to downscale complex meshes for browser deployment

Pro Tip 1: Use Unity's WebAssembly Memory growth setting cautiously— initial heap size must be balanced with browser limits; too large will crash mobile, while too small leads to out of memory errors under load.

Pro Tip 2: Unity allows per-platform control over LOD targets (e.g., Windows vs. WebGL). For WebGL builds, consider forcing LOD1 as the highest quality level to maintain performance while preserving model clarity. This enables LOD0 to remain available for high-end platforms without affecting browser experience.

By keeping platform limitations in mind, you ensure that web builds maintain responsive UI, acceptable frame rates, and device-appropriate rendering quality.



Unity hub downloading WebGL support.

© 2024 Unity Technologies

Optimizing simulation assets for web

Web builds depend on aggressive optimization without compromising clarity. Pre-process simulation content to fit within memory and GPU constraints typical of browser-based devices.

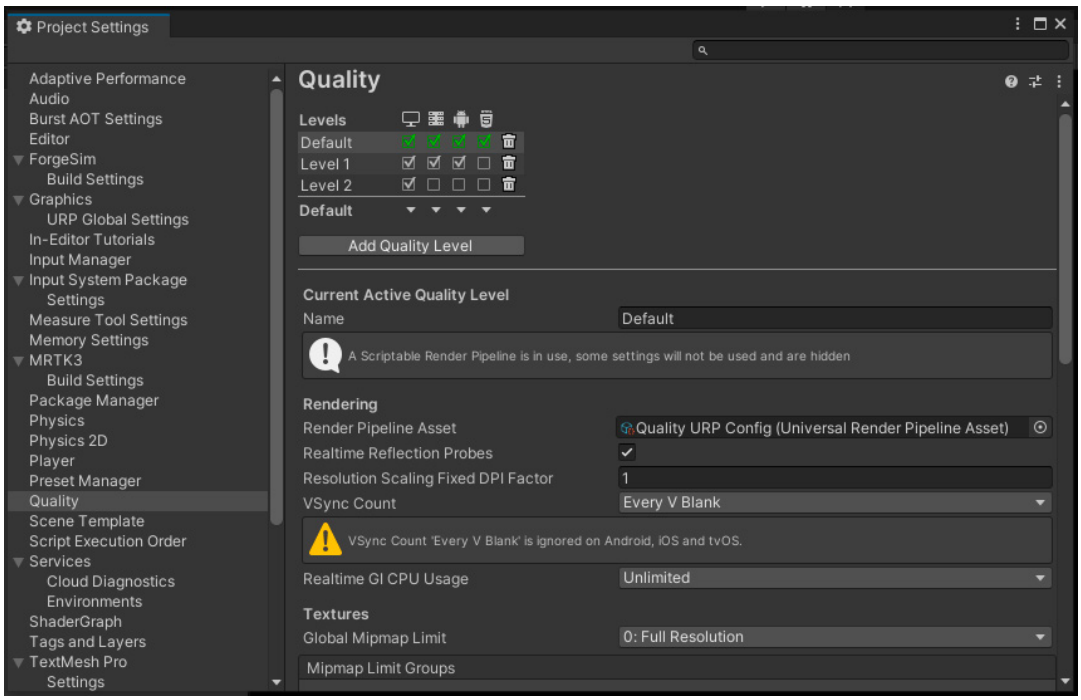
This includes:

- Reducing texture sizes and using compressed formats (e.g., ASTC, DXT)
- Mesh simplification and LOD creation for all major assemblies
- Baking lighting where possible, avoiding real-time GI
- Streamlining UI logic to minimize event overhead

Web asset optimization checklist:

- ✔ Use TextureImporter settings for max-size scaling per asset type
- ✔ Apply Unity's Mesh Simplifier or third-party tools for geometry reduction
- ✔ Group interactables logically to minimize draw calls
- ✔ Test memory allocation across Chrome, Safari, and Firefox
- ✔ Monitor asset bundle sizes and serve over CDN for edge-caching

Pro Tip: For mobile web targets, reduce draw call complexity below 1,000 and avoid alpha-blended or screen-space transparent UI unless essential—it's one of the most expensive operations on mobile GPUs.



Unity quality settings for WebGL android and windows. Image courtesy ForgeFX.

© 2024 Unity Technologies

Best practices for web-based training and review builds

- Avoid monolithic scenes; break content into additive, asynchronous loads
- Use asset streaming to support large environments
- Minimize scene hierarchy depth to reduce DOM-like traversal costs
- Include fallback controls for touchscreen-only devices
- Use native browser console logging for remote QA support

Pro Tip 1: Encourage users to bookmark the simulation URL and return asynchronously: WebGL enables session continuity across disconnected training and review cycles.

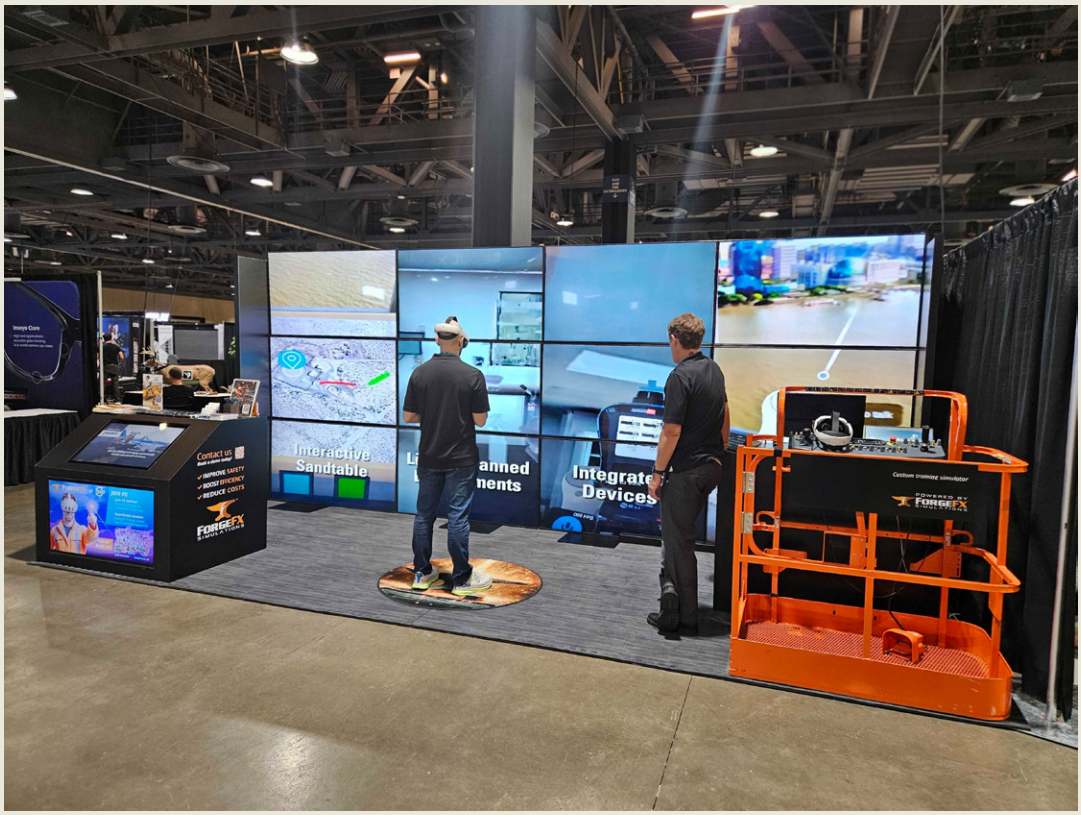
Pro Tip 2: Use Application.persistentDataPath carefully on WebGL—it maps to IndexedDB, which has quota limits and security caveats across browsers.

Why it matters

WebGL, WebGPU, and mobile browser deployment give you the ability to share simulation content quickly, broadly, and with minimal friction. In scenarios where hardware access is uneven or time is limited, these technologies enable fast iteration, live collaboration, and remote review. Browser-based deployment extends the reach and impact of simulation assets beyond engineering and training teams-empowering entire organizations to evaluate, learn from, and iterate on simulation-based solutions in real time.

© 2024 Unity Technologies

The future is here: Embracing what comes next in simulation-based training



Users at the ForgeFX demo booth. Image courtesy ForgeFX.

© 2024 Unity Technologies

Back when real-time engines were still novelties in the enterprise world, simulation was a controlled environment-a sandbox that mimicked procedures, mirrored interfaces, and gave users a rehearsal space for tasks they'd eventually perform in real world settings. That's still vital. But what's emerging now is different. Today, simulations are becoming participants. They observe. They adapt. They inform. And increasingly, they interact with live systems and real data.

We're entering a phase where the training environment isn't a separate world-it's layered directly over reality, with real-time inputs, responsive logic, and adaptive intelligence. The technologies enabling this aren't theoretical. They're operational: real-time digital twins, XR-based teleoperation interfaces, and AI-driven instruction using machine learning models embedded inside Unity. Across industries, these tools are already transforming simulation into a live decision-support layer-whether it's in aerospace maintenance tracking, smart factory analytics, or immersive remote inspection of industrial systems .

A digital twin, properly architected, is more than a visualization. It's a behavioral model, reacting to the same inputs as its real-world counterpart and producing data for forecasting, diagnostics, or guided maintenance. It gives context to training-real or simulated-because it reflects not just what *could* happen, but what *is* happening. For example, Siemens has implemented digital twins in its manufacturing processes, achieving a 30% reduction in maintenance costs and improved decision-making accuracy.

VR training mode. Image courtesy ForgeFX.

Unity's Data-Oriented Technology Stack (DOTS) supports this level of fidelity by handling concurrent data streams at scale with low overhead-making it possible for digital twins to ingest real-time sensor inputs, simulate component behavior, and deliver performance feedback without frame drops. With ECS and the Burst compiler, developers can simulate thousands of mechanical or procedural systems in parallel while ensuring real-time

© 2024 Unity Technologies

performance. This is critical in environments that demand deterministic behavior and frame-locked training loops.

Teleoperation, meanwhile, introduces a new modality to simulation: action. Through XR, users can now control physical machinery from afar. When designed with haptic cues, failovers, and real-time sensor feedback, these systems transcend their training origins and become operational tools, allowing operators to perform mission-critical tasks from safe, remote vantage points.

Unity's XR Interaction Toolkit and OpenXR framework support enable developers to prototype and deploy teleoperation systems across headsets and controllers, while XR Hands extends this fidelity to natural input for zero-latency interactions. This allows remote operations to feel grounded and intuitive-even across heterogeneous hardware. In nuclear facilities, teams using Jackal UGVs for remote inspection have reported a reduction in human exposure times by up to 40%, showing measurable gains in both safety and operational uptime.

And then there's artificial intelligence-not just as ambient logic, but as an embodied guide. Machine learning can model expertise, interpret performance trends, and tailor instruction in real time. Whether through an intelligent virtual instructor or a dynamic scenario that reshapes itself based on user input, AI becomes not just a feature, but an instructional partner.

Unity Inference Engine (previously called Sentis) makes this real. By enabling on-device inference for models like pose detection, classification, voice intent, or predictive logic, the Inference Engine reduces dependency on cloud inference and avoids latency bottlenecks-crucial for mobile, XR, or disconnected deployments. Unity's own samples demonstrate how these models can operate within fixed time steps, powering gesture recognition, predictive analytics, or real-time scenario feedback-capabilities that are especially relevant in safety training, gesture-based control systems, and fault diagnosis simulations.

Where Unity Inference Engine optimizes runtime intelligence, Unity AI accelerates creation. With tools for AI-assisted code generation, behavior scripting, UI generation, and 3D layout, Unity AI streamlines the prototyping phase enabling faster iteration and reduced design-to-deployment time. For simulation development teams managing tight feedback loops, Unity AI can cut implementation effort significantly-allowing more focus on fidelity and functionality.

These technologies open the door to powerful new paradigms-but they demand a foundation that supports integration, adaptability, and scale. That starts with Unity's modularity, but extends to how simulation teams design systems.

Best practices for forward-facing simulation teams

- Architect simulation logic as services, not scenes, to support real-time data input and modular behaviors.
- Use lightweight data structures to decouple real-time telemetry from interactive logic.
- Design XR interaction patterns with redundancy for fallback input and platform portability.

© 2024 Unity Technologies

- Treat AI systems as assistants, not authorities-crafting scenarios that evolve with the learner.
- Integrate observability from day one-log key simulation states and user inputs for review.
- Use Unity's Cloud Diagnostics to track crashes, anomalies, and performance regressions across test and deployment pipelines.

Recommended tech directions for simulation teams

- **Adopt Unity Inference Engine** for embedded AI instruction, predictive maintenance, and adaptive feedback systems.
- **Prototype Digital Twins** with real-time data ingest using **DOTS and ECS** for behavioral realism.
- **Deploy XR teleoperation interfaces** using **XRITK, XR Hands**, and **OpenXR** to ensure hardware scalability.
- **Incorporate Unity Analytics and Cloud Diagnostics** to measure simulation performance in the field.
- **Design for resilience** by enabling on-device inference, offline persistence, and modular runtime logic.
- **Explore Unity Render Streaming or WebRTC** for low latency mirroring and collaborative QA sessions.

Pro Tip: If you want a simulation to behave like a live system, start by teaching it to listen. Reactive design begins with observability. Use telemetry, session logs, and real-time feedback to capture what's working, what's not, and what's next. Pair Unity's diagnostics with structured event logging and WebRTC-based mirroring to replicate sessions and reduce remote QA cycles by up to 30% in field-testing scenarios.

Networked, persistent workspaces are also evolving. With Unity's Netcode and XRITK stack, simulations can support synchronous collaboration, where learners, instructors, and SMEs can train, observe, and annotate together inside the same shared digital environment. These collaborative workspaces can support role-based interaction, team-based procedures, and even asynchronous review through persistent simulation states, shared annotations, and replay systems, mirroring real-world operational coordination.

These emerging technologies don't just elevate what simulations *do*-they redefine what simulations *are*. Not just training platforms, but decision-support tools. Not just instruction, but interaction. This is where simulation stops being software and starts becoming infrastructure.

It's not a question of whether these technologies will be adopted. It's a question of whether we will build for them-modularly, openly, and with a mindset that every feature might one day power a live system.

© 2024 Unity Technologies

We are no longer simulating readiness. We are building systems that *are* ready. With technologies like Unity's Inference Engine (previously called Sentis) for runtime AI, DOTS for high-throughput simulation logic, Unity AI for AI-assisted content and behavior authoring, and XR foundations that scale across hardware, Unity becomes more than a 3D engine. It becomes the operating system of simulation itself-powering creation, interaction, and intelligence across the training lifecycle.

ForgeFX simulations construction training simulators. Image courtesy ForgeFX.

Real-time systems, real-world impact

Simulation has moved to the center of how modern organizations design, operate, and transfer knowledge. It's becoming the connective tissue between how systems are built, operated, and understood. As technology, environments, and expectations evolve, simulation gives us a way to explore complexity safely, test decisions early, and translate knowledge into capability-at scale.

Everything in this eBook points to one idea: Unity is more than an engine-it's an ecosystem for building highly intentional, technically rigorous training systems. And it's the discipline with which we use its tools that turns potential into capability.

When we talk about XR interaction, asset pipelines, cloud workflows, or multiplayer logic, we're really talking about structure. About choices made upstream that preserve clarity and reduce friction. About systems that can be handed off, extended, versioned, and scaled-because they were architected to be sustainable from the start.

Technologies like AI-guided instruction, digital twins, and XR teleoperation aren't valuable because they're new-they're valuable because they respond directly to enterprise training

© 2024 Unity Technologies

needs: contextual feedback, remote accessibility, continuous validation, and measurable learning outcomes. Unity's expanding ecosystem-Inference Engine for embedded AI, DOTS for high-performance logic, XRITK for natural interaction, and Unity Cloud for CI/CD-makes this transformation not just possible, but repeatable. Unity gives us the practical foundation to make those responses real—and to integrate them meaningfully into how organizations train, review, and operate. And the demand for that foundation is accelerating.

In 2024, 62% of enterprises reported increasing investment in immersive systems, with a 30% rise in companies allocating $10M+ annually-clear evidence that scalable simulation is fast becoming the backbone of enterprise training.

As simulation developers, our responsibility is to meet those needs with systems that behave predictably, perform reliably, and teach effectively. Whether you're building for VR, desktop, tablet, or browser-whether your users are operators, technicians, instructors, or analysts-the mission is the same: to make training more accessible, more precise, and more responsive to the realities it prepares people for.

Simulation doesn't just convey information-it shapes behavior. It turns repetition into mastery and feedback into actionable insight.

If there's one takeaway from all the examples, workflows, and implementation strategies shared here, it's this: excellence in simulation isn't about what you add. It's about what you enable.

Unity gives us the engine. We build the infrastructure. We shape the future.

© 2024 Unity Technologies


Unity®