Unreal Fest 2024 Seoul

Mobile in Unreal Engine in 5.4, 5.5 and beyond

Jack Porter
Lead, Engine Mobile Platforms
Epic Games

About me

Hi, I’m Jack Porter!

- Currently the Unreal Engine Lead for Mobile Platforms
- 25 years experience working on all versions of Unreal Engine, from UE 1 to UE 5
- Previously worked on UE Landscape, rendering, networking, core systems, UI and many others
- Lived in Korea for 20 years
- Rejoined Epic when we formed Epic Games Korea in 2009

Intro

Why does Epic care about mobile?

UE Mobile development roadmap

Android setup experience

Mobile & platform preview

Memory analysis tools

On-device iteration

Memory savings

Mobile renderer additions

Runtime PSO caching

Lumen

Nanite


Mobile
Content Iteration

Content iteration

While developing mobile or cross-platform games, content creators need to be able to see how their contents will look on device

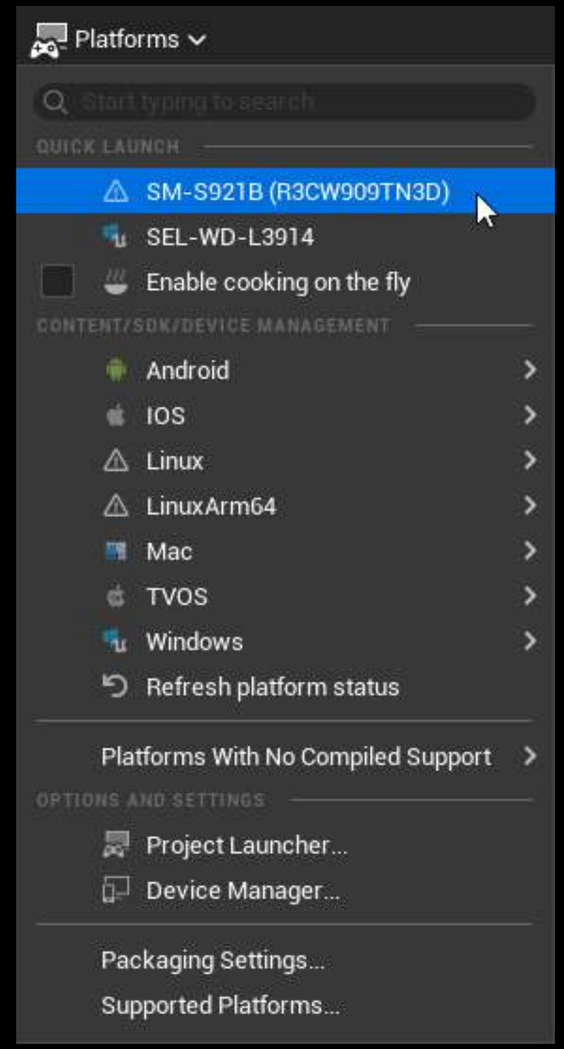
Option 1: Iterating directly on device

✓ Relatively quick for simple games using Quick Launch, or even to quickly package and install your game

✕ Not practical for large games due to slow cook process and time to install build and contents

✕ Each developer needs access to devices of each platform

→ **On-device iteration improvements currently under heavy development, ETA 5.6+**

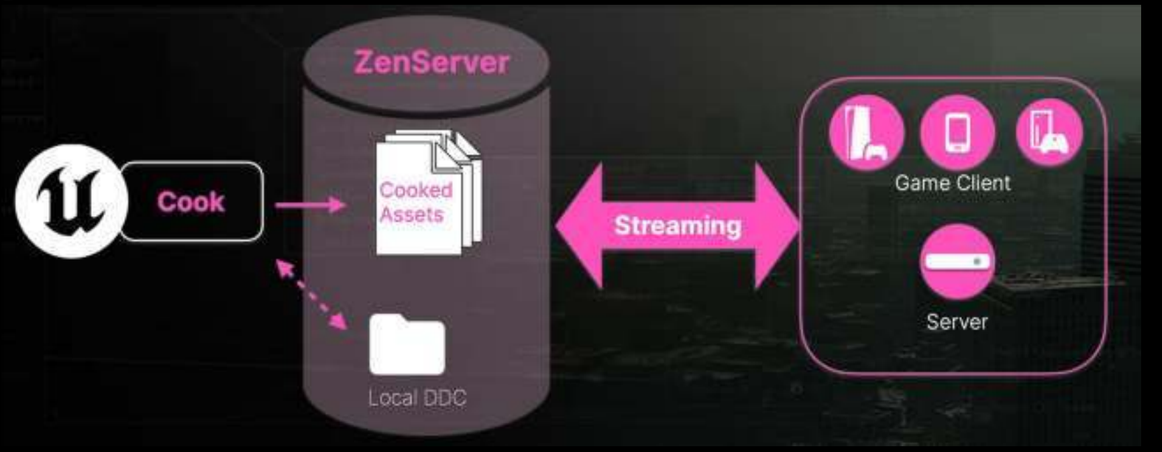


On-device iteration

New efficient workflow to iterate directly on target platforms under development.

ZenServer, the new local & shared DDC system, will allow:

- incremental local cooking of content based on a shared, build-server cook
- directly stream cooked data to mobile device over USB 3.0 cable or network

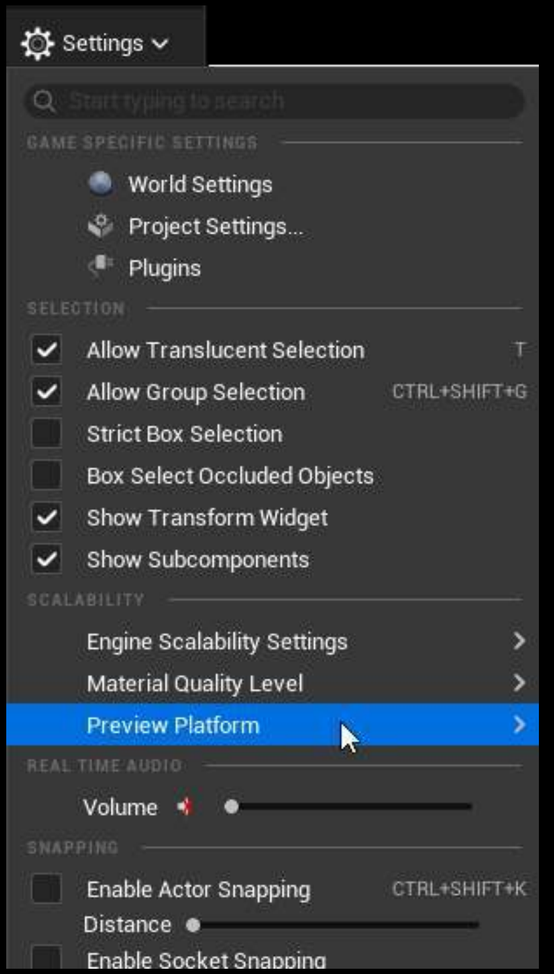


Early experimental features available in **UE 5.5**

→ **Upcoming presentation at Unreal Fest Seattle, Oct. 2024:**
"How to Improve Developer Efficiency in UE 5.5 and What's Coming Next"

Content iteration - Mobile Preview

Fast, accurate platform preview is a viable alternative for creating content for cross-platform games.



Option 2: Iterating with Editor Preview Platform

- ✓ Convenient way to test content for multiple platforms
- ✓ Uses the same shaders used on device, and applies the same scalability and device profile cvars
- ✓ All developers can test content for all platforms without their own devices and without leaving the editor
- ✗ Not all device shader limitations can be accurately emulated on PC

→ Currently making significant improvements, ETA 5.5+

Mobile Preview

Recent Improvements

< 5.3 - Only a single mobile preview shader platform & shaders

- ✕ Could not support different rendering settings for different devices
  - Mobile Forward vs Deferred
  - Shadow and lighting setups

5.3 - Added individual custom shader platforms for each real target platform

- ✓ Supports unique settings for each platform

5.4 - Usability and accuracy improvements

- ✓ Used to check budgets during our internal title development

Mobile Preview

Improvements coming UE 5.5 and beyond...

5.5 - Android Device Profile & Device-specific Preview using JSON

5.5 - Half-precision PC emulation

5.6+ - Future improvements

- Applying device aspect ratio and safe zone / notches in editor
- Preview of TextureLODGroup settings
- Simultaneous preview of multiple platforms

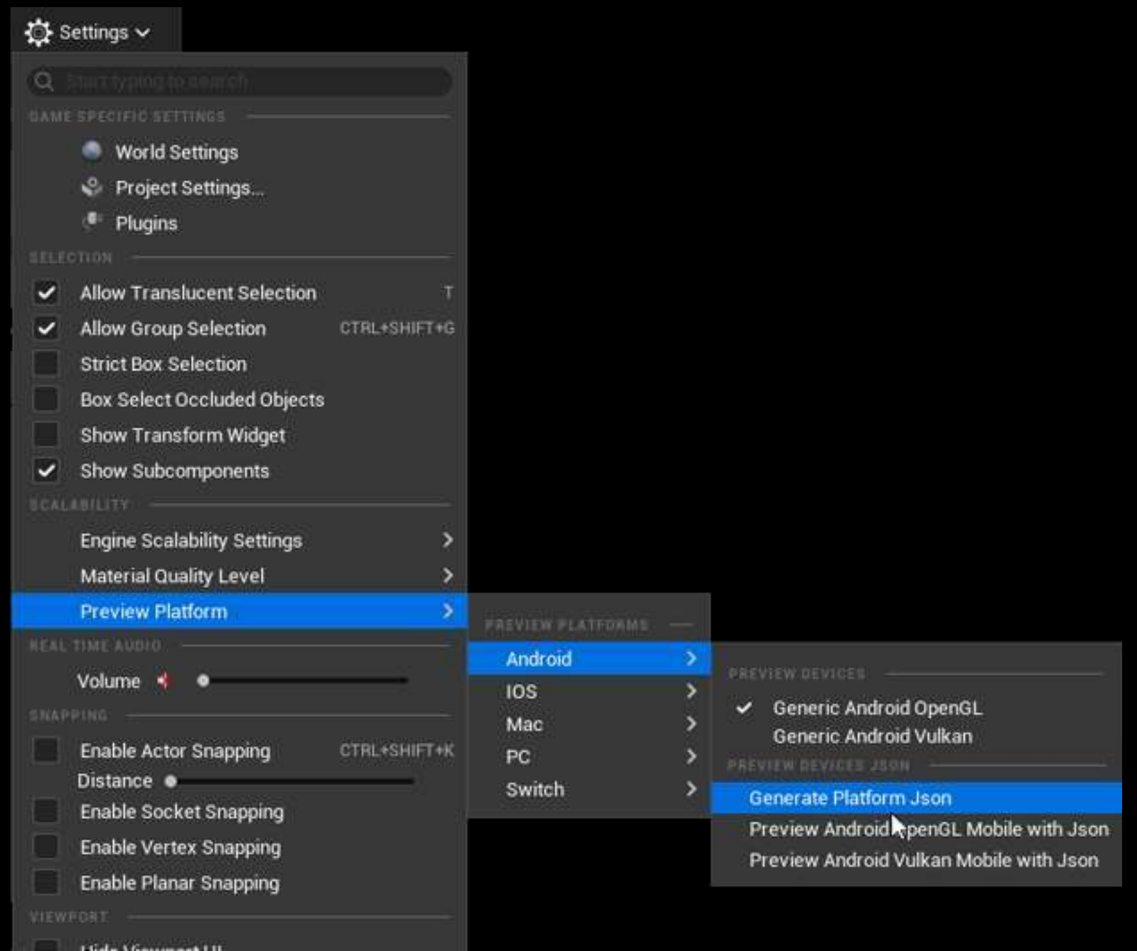
Mobile Preview - Device Json

Often you want to preview a specific device's settings, such as your representative high-spec Android phone.

Generate a Json file for any attached Android device

Preview as that device at any time

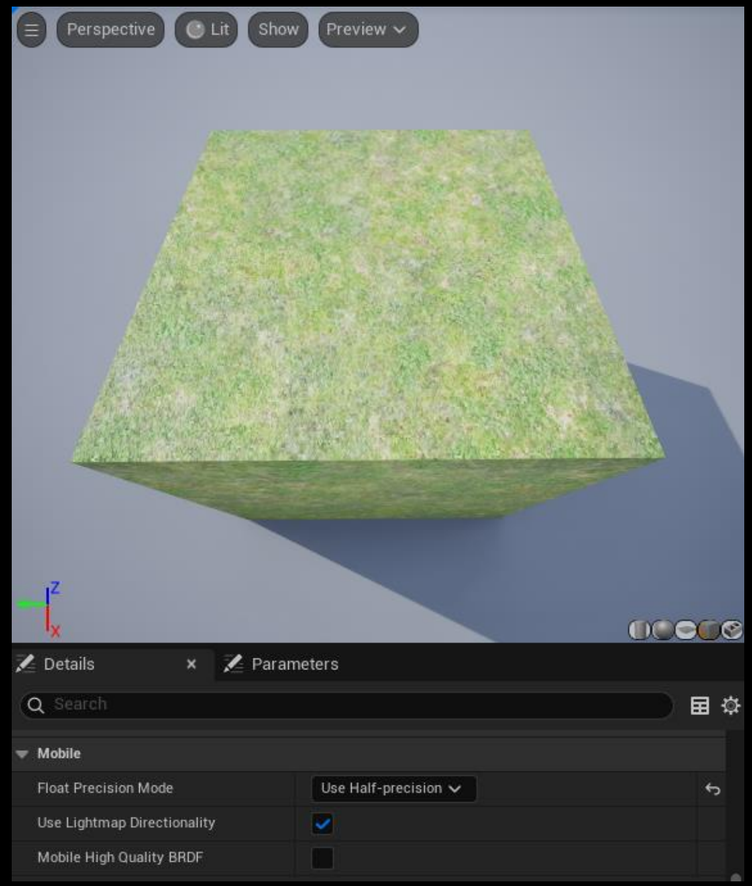
- Evaluates device profile rules as it will be on device
- Scalability, console variables and other settings applied



Mobile Preview - Half precision

Mobile hardware is optimized for half precision shader math, but until recently it was not available on desktop

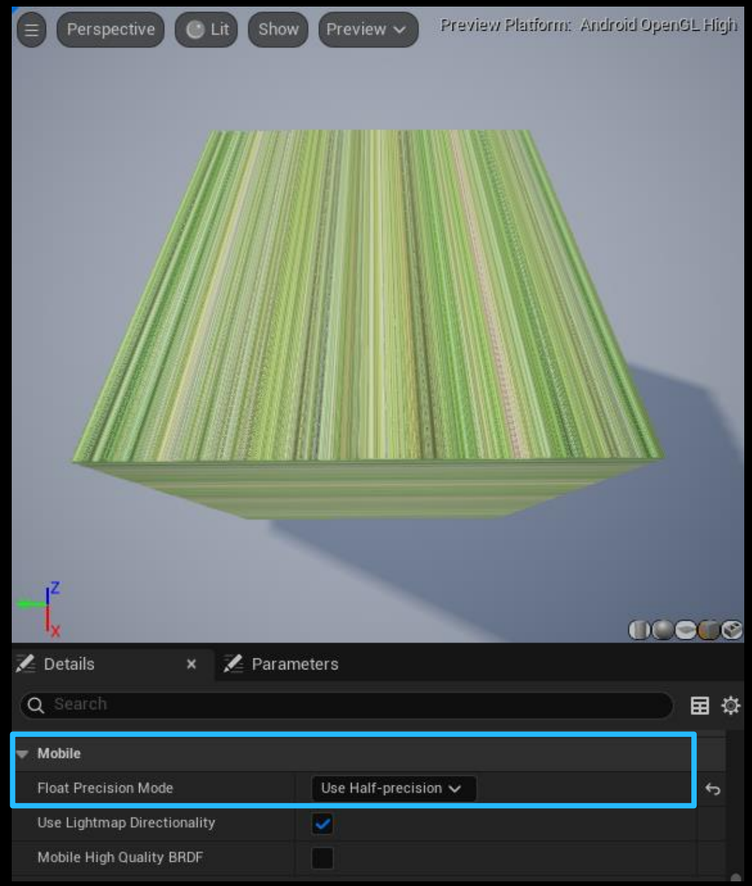
- Half precision floating point (FP16) types are generated in mobile pixel shaders for OpenGL ES / Vulkan for Android or Metal for iOS
  - Can override using “Float Precision Mode” for Project or Material
- FP16 has a range of ±65,504 which can overflow
  - World space calculations
  - Landscape texturing
  - UVs scaling
- Very common source of bugs on mobile, does not reproduce on Mobile Preview in 5.4



Mobile Preview - Half precision

Mobile hardware is optimized for half precision shader math, but until recently it was not available on desktop

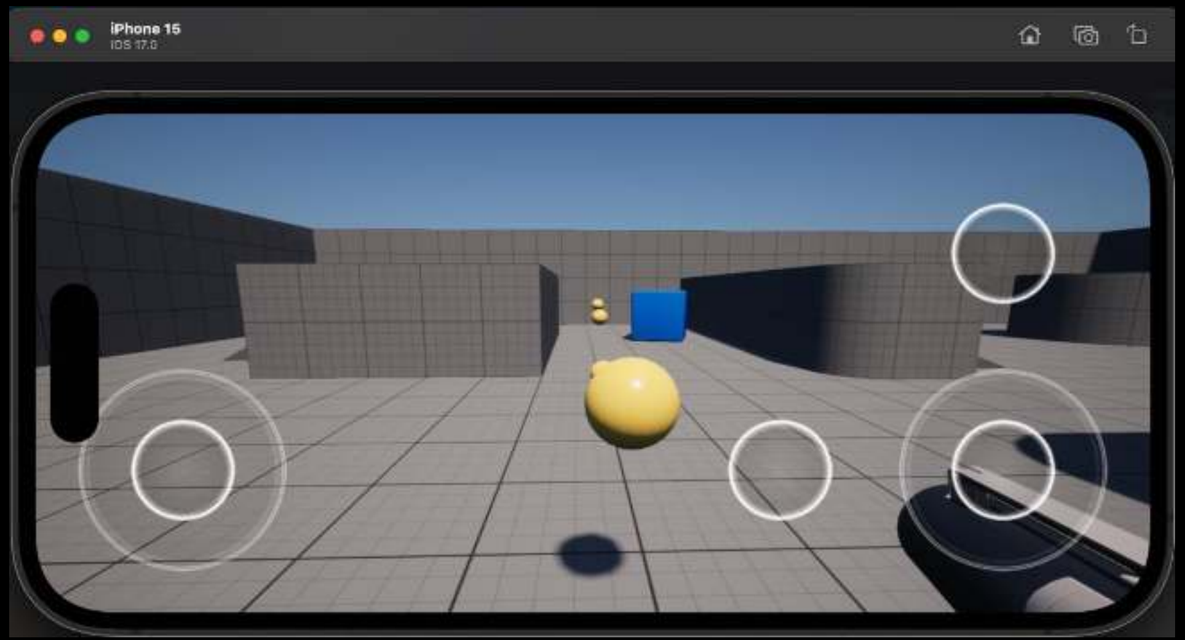
- In 5.5, Mobile preview on PC now generates half-precision shaders that run on modern desktop GPUs
- Follows project or material settings
- Behavior not identical to mobile GPUs but many content bugs now appear in mobile preview

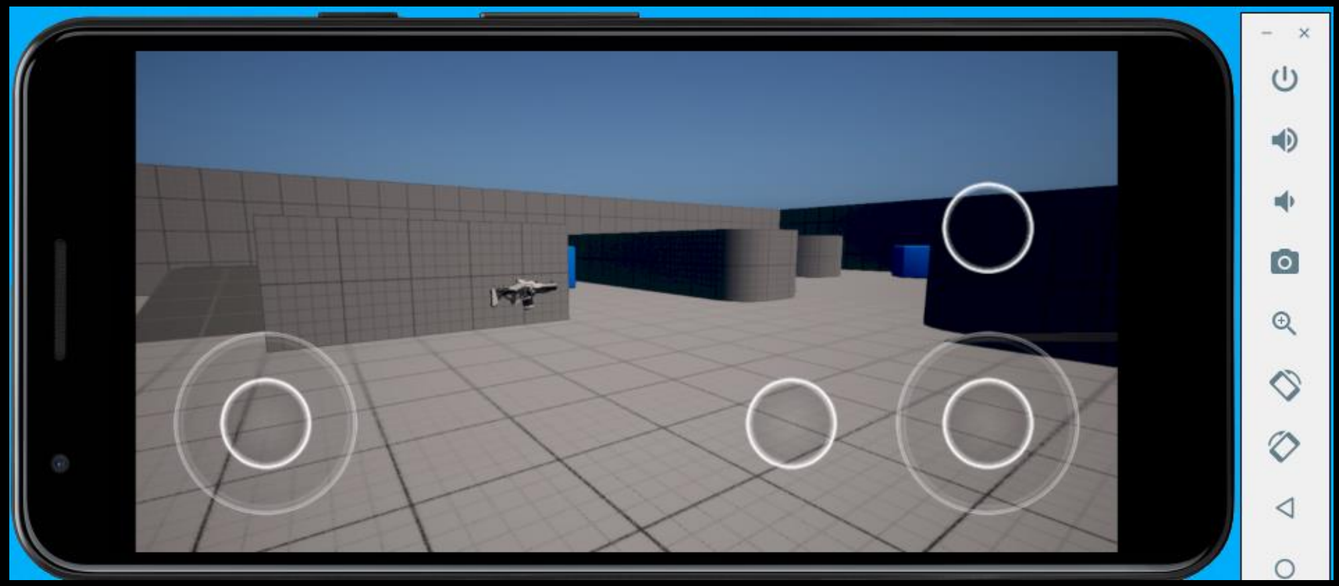


Android Emulator / iOS Simulator?

Do the emulators simplify previewing your game contents?

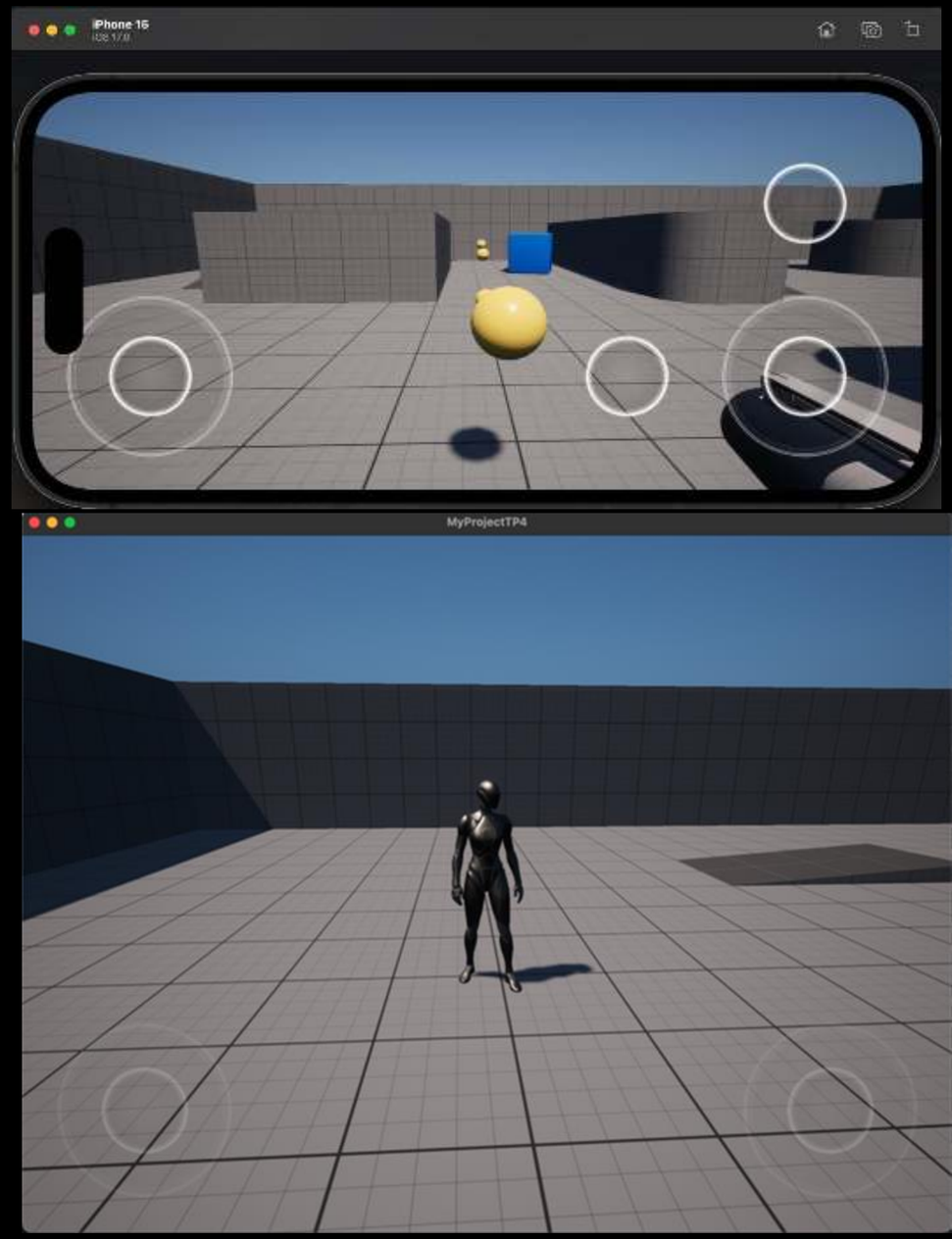
- The Xcode 15+ iOS Simulator supported in Unreal Engine 5.4
- MacOS can also natively launch iPhone/iPad apps on Apple silicon
- The Android Emulator from the Android 14 SDK can run UE 5.4 apps on Windows, Mac and Linux





iOS Simulator

Supported since 5.4



iOS Simulator

- ✓ Emulates the resolution any Apple device
- ✗ App and 3rd party libraries must be recompiled
- ✗ Only supported on Mac
- ✗ Limited GPU emulation (~A9)
- ✓ Works with Xcode debugger

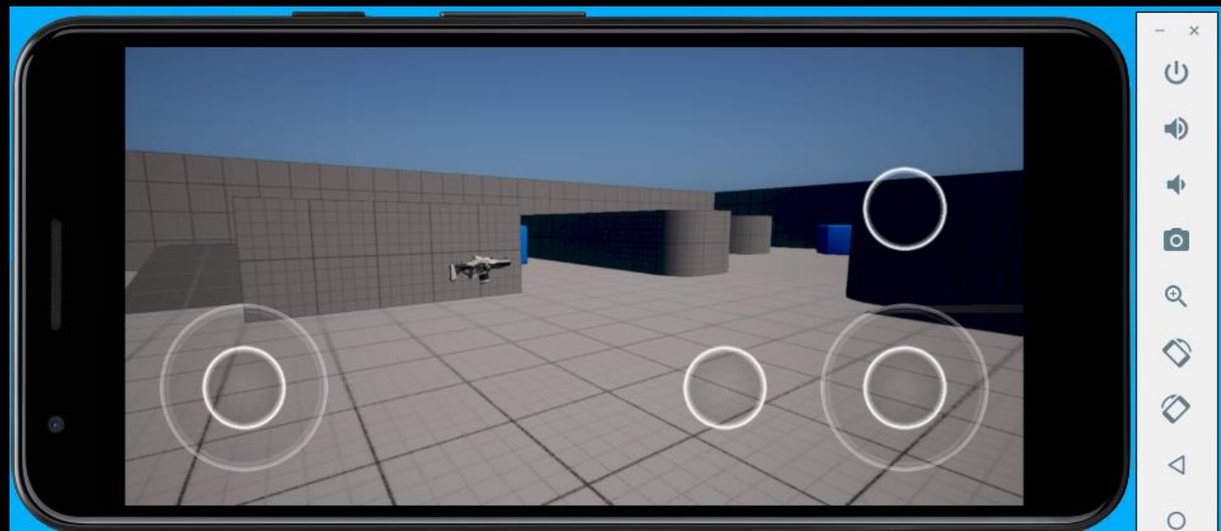
Native Mac iOS app support

- ✓ Works with unmodified iOS App
- ✓ Uses native Mac GPU
- ✗ Works only on Apple Silicon Macs
- ✗ Intended for end users and not developers
- ✗ Does not support debugger

Android Emulator

Supported since 5.4

- ✓ Can run an unmodified UE5 .apk binary and provides good arm64 CPU emulation on Intel CPUs (requires Android 14 emulator image for UE5)
- ✓ Arm64 apk runs natively on Arm CPUs such as Mac
- ✗ GPU is not emulated and passes straight to the host
- ✗ OpenGL ES emulation is only ES 3.0 and cannot run UE5
- ✓ But Vulkan works fine even on Mac
- ✗ CPU emulation requires hardware virtualization which can be an issue in some environments
- ✗ Debugger only works when app's CPU architecture matches host CPU



Mobile Memory

Memory

The biggest issue facing AAA / Online games shipping on iOS and Android

Compared to other platforms, mobile platforms have extra challenges:

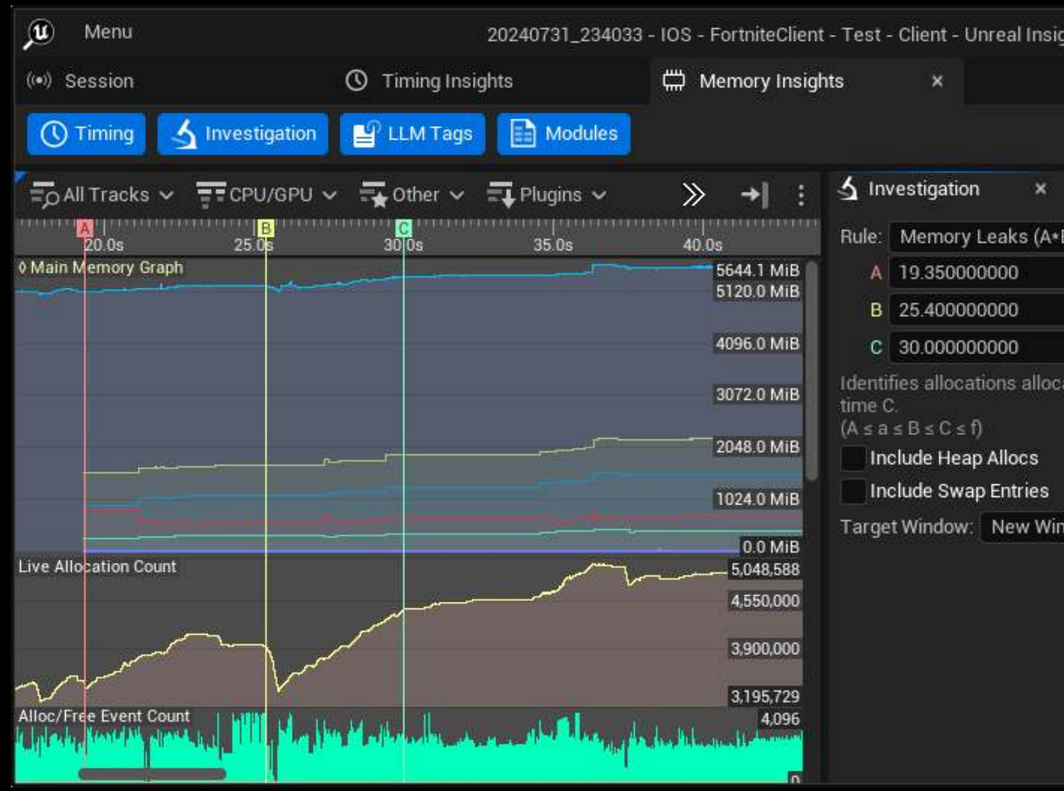
- Unlike PC, we have virtual memory but no swap file
  - ✗ If we ask for memory and there is none available, our game is killed (“OOM”)
  - ✓ Data can be “memory mapped”
  - ✗ But no “swap” to a large swap file on flash storage
  - ✗ On Android it’s common to compress memory pages and swap them to a RAM drive, “zRAM” reducing memory available for runtime use
- Unlike consoles, there is no fixed amount of memory available
  - ✗ Amount varies by device, by OS version and zRAM settings determined by vendor
  - ✗ On Android, background processes can take memory away from your game

-> Memory analysis tools on mobile are very important

Memory Insights

LLM trace and Memory Insights are the best tools we have for identifying unexpected memory allocations

- Memory Insights provides a stack trace at each allocation and deallocation
- ✓ Find cause of unexpected allocations
- ✓ Find memory leaks (eg between levels) using markers
  - Insights will show call stacks for allocations made in the first level that are still allocated in the second level



Mobile support:

5.4 - Android support added

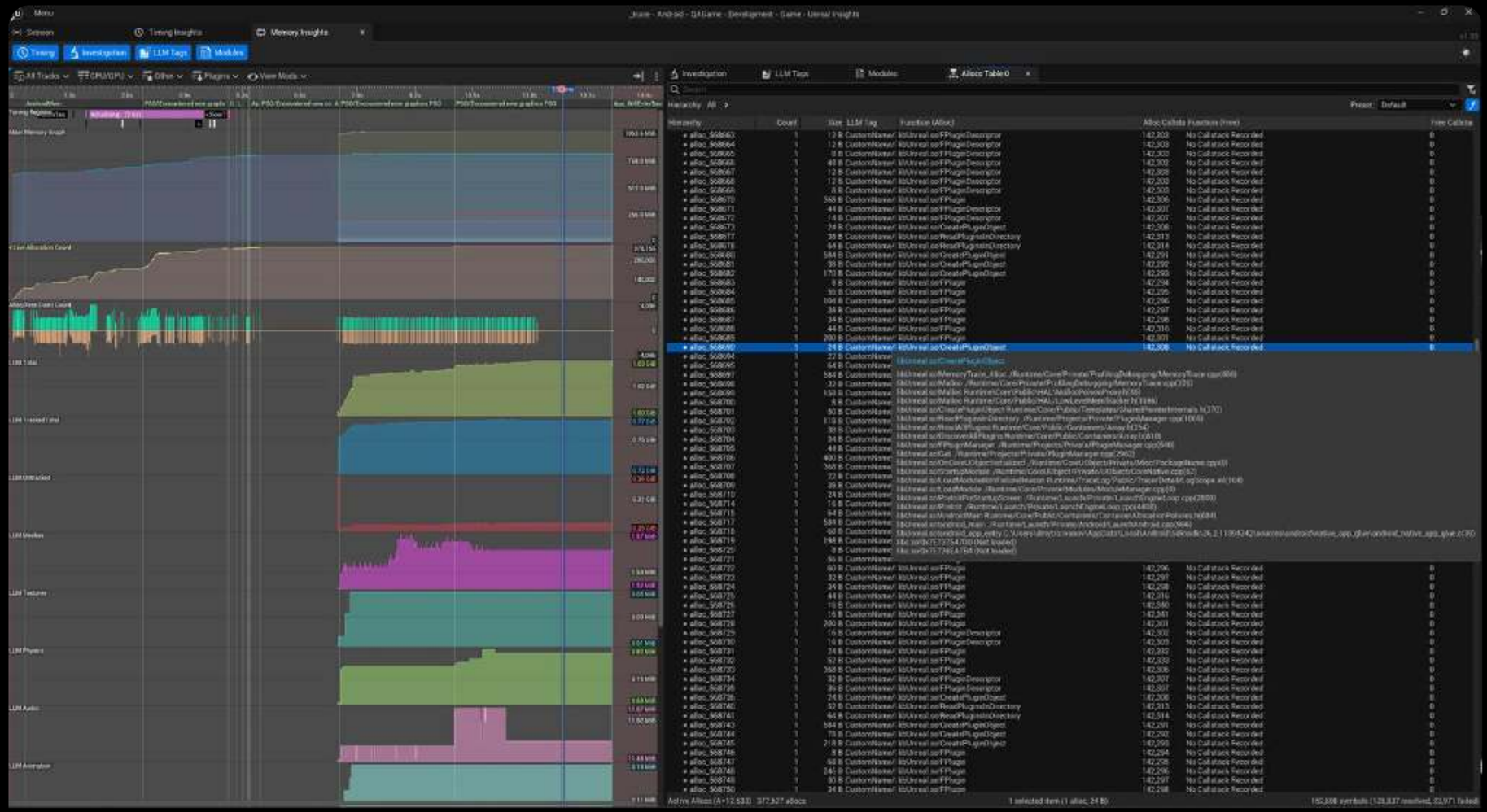
5.5 - iOS support added

Android native heap tracing added

Memory Insights on Android

New in UE 5.4

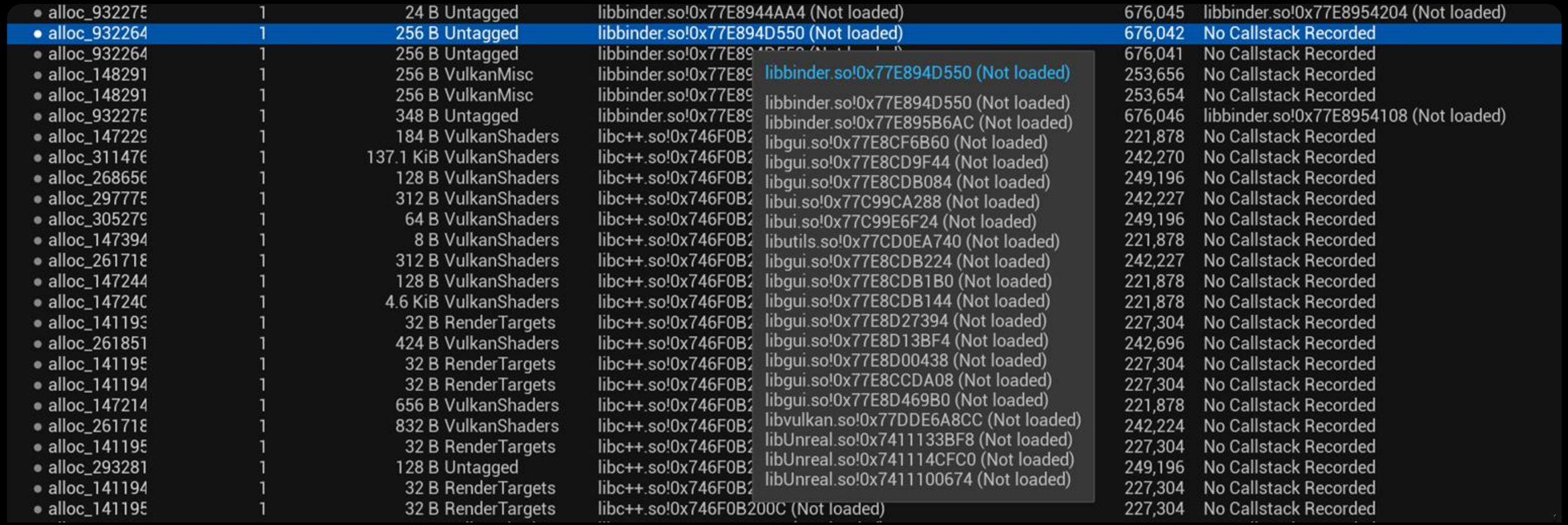
- Low impact on game performance with the help of new fast callstack walker



Memory Insights on Android - Native heap

5.5 - Native heap (scudo) memory tracking allows you to identify when memory allocated outside the game, eg by the GPU driver

- Possible to capture allocs outside of UE that go via libc.so
- Pass `-ScudoMemoryTracing` to UBT when building the .apk

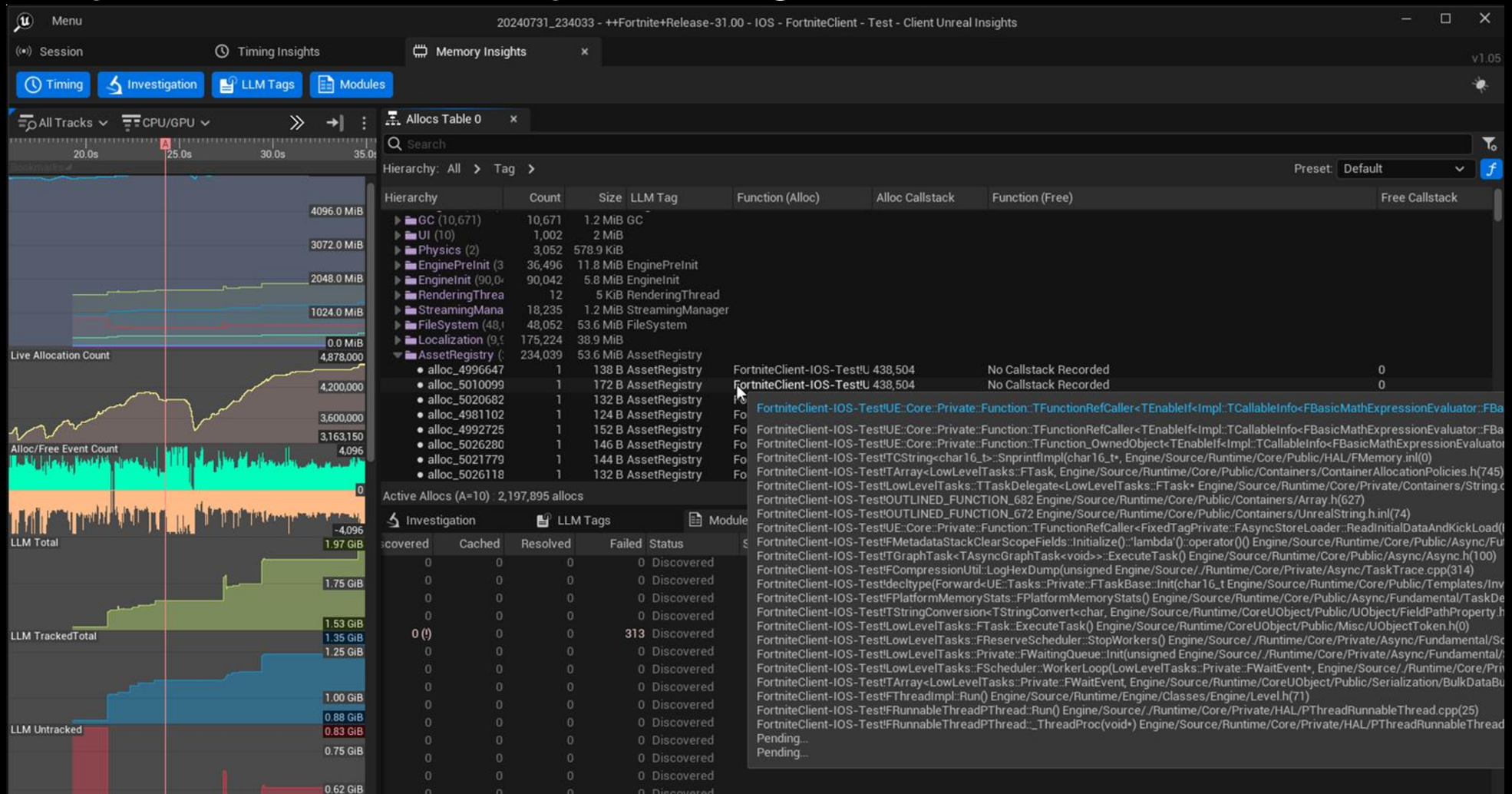


-> More native Android memory tools coming in 5.6, including zRAM analysis tools

Memory Insights on iOS

Coming in UE 5.5

- Uses Breakpad format “.psym” symbol file generated from the .dSYM
- Allows symbolication and analysis in Insights on Windows or Mac



Recent memory savings

Early in Fortnite iOS development, we were unable run on 4GB iOS devices. Memory Insights and LLM used to identify and mitigate.

Some issues were content:

- Assets unexpectedly loaded
- Draw distances, texture/mesh streaming pool sizes, LODs adjustments etc

However the many of the problems were the growth of our game's overall contents, finding Unreal Engine scalability limits on mobile:

- More assets overall -> bigger Asset Registry
- More Pak files and content -> more Pak file indices
- More unique materials -> bigger shader libraries
- More UObjects for assets and game systems -> bigger Name table and UObject lists
- More game text -> bigger localization tables

Recent memory savings

5.5 includes memory savings identified during Fortnite iOS development that benefit all UE mobile games

- MallocBinned3 enabled for iOS and Android and settings tuned for mobile
- Memory mapping for iOS Metal shader library metadata
- Engine PSO cache savings
- Added support for RAD Audio Codec for iOS and Android
- Many, many small savings

Estimated total engine-side savings **with our content** was at least **250 MB**.

Upcoming memory savings

For 5.5, we believe we have found most of the easy memory wins

For 5.6+ - Further memory savings planned

- Will require some work in some engine systems to make them more scalable, memory-mappable or otherwise more efficient.

Some significant offenders are:

- Asset Registry
- Localization strings
- FName table

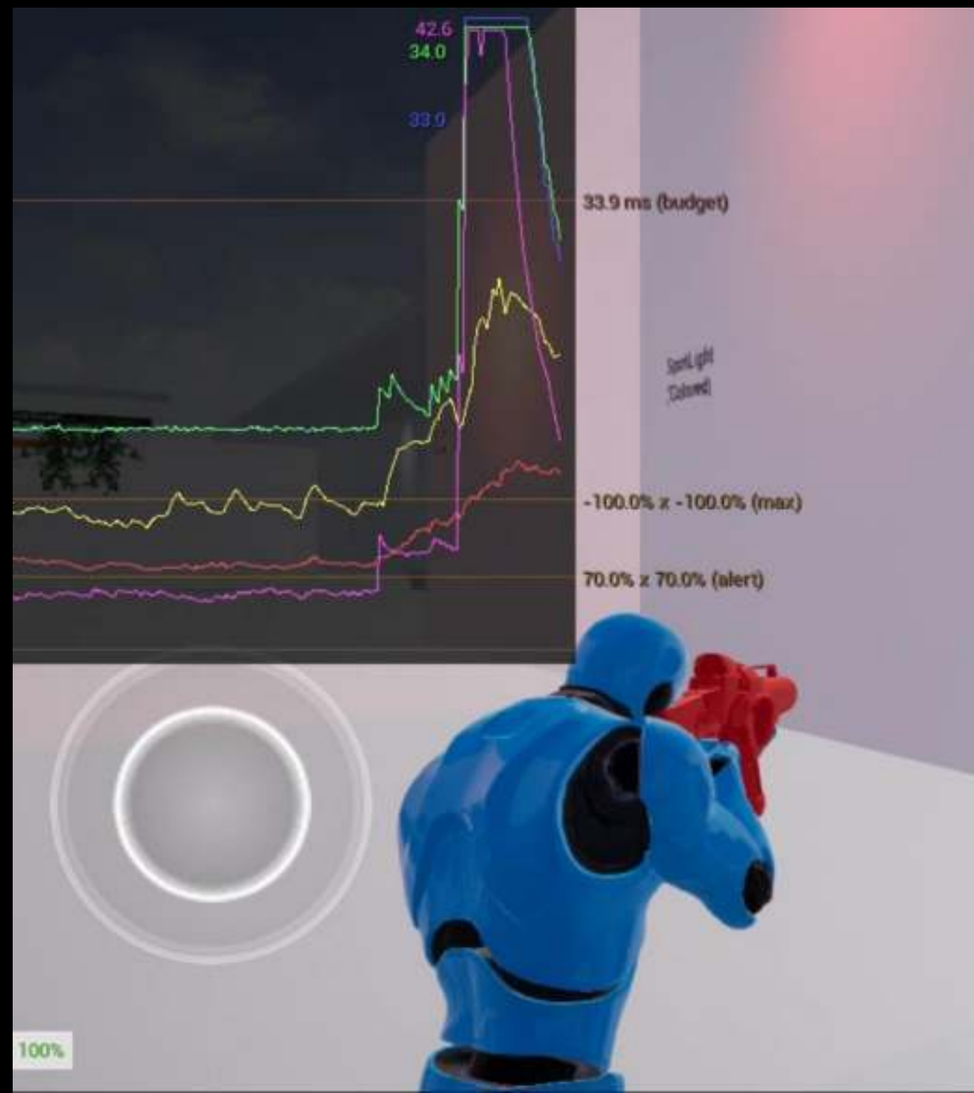

Mobile
Rendering & Performance

PSO caching

A common problem with UE on mobile is PSO hitching

A hitch occurs when a shader pipeline (PSO) is needed for rendering but it has not been compiled by the device yet

- Previous solution was offline PSO gathering
  - PSOs encountered during play testing were collected and packaged with the game
  - At first game launch, the collected PSOs were compiled
  - ✕ Time-consuming process required after each content update
  - ✕ Any PSO not encountered during play testing would still cause a hitch



Runtime PSO caching improvements

Runtime PSO caching predicts the shader pipelines that will be needed as content is **loaded** and compiles them before it rendered

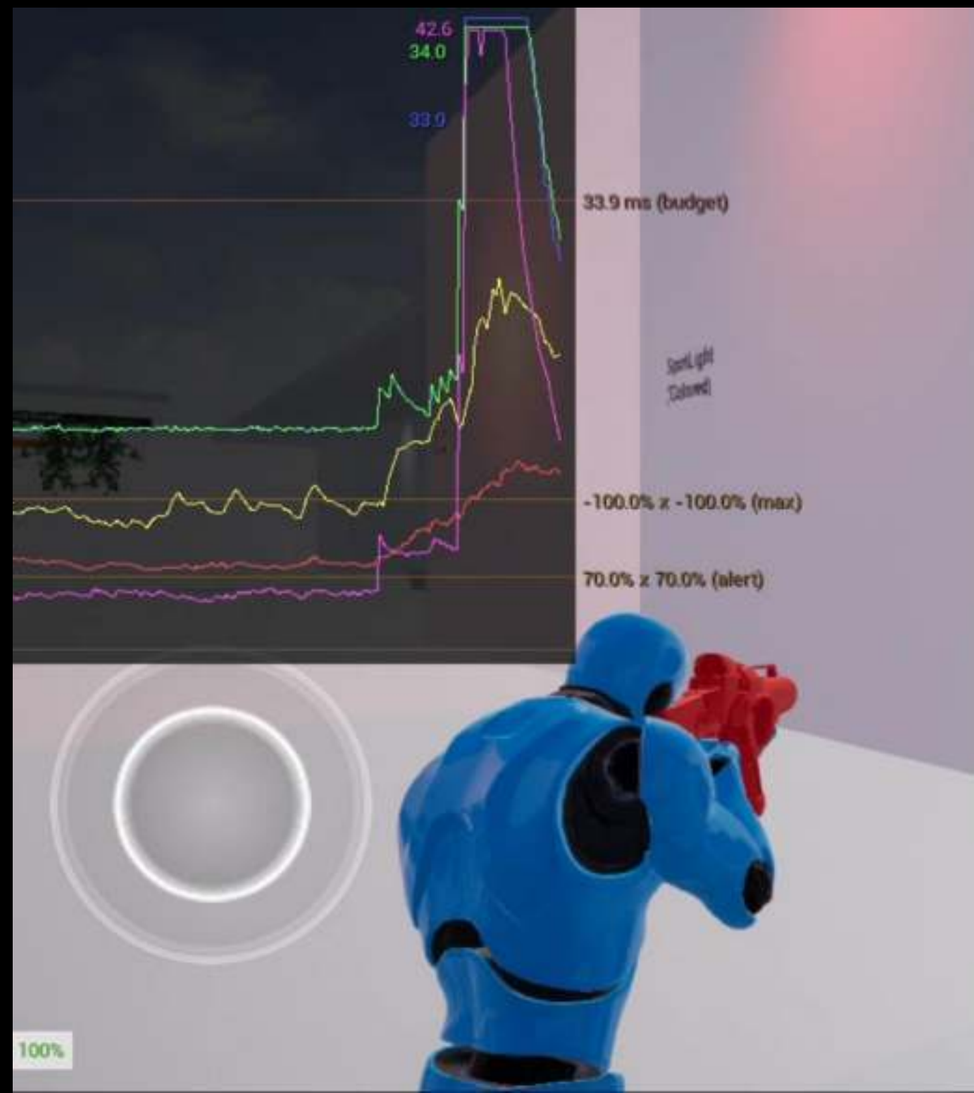
5.5 - Enabled by default for mobile

- Works with all lighting setups
- Reduced shader permutations

- Offline PSO gathering no longer necessary for hitch-free mobile games

- Android background PSO caching improvements

->See Dmitriy Dyomin's session at 1:30pm in this room for the details



iOS Windows cooking

iOS cooking improvements

5.5 - Better support for Windows Metal toolchain

- Previously we only supported generating inline Metal shaders
- Windows Metal toolchain now generates Platform Shader Libraries for iOS
- It is now possible to cook contents for your entire game on Windows, reducing the costs

→ **A Mac is only required to compile the C++ for iOS**

Toolchain from Apple at https://developer.apple.com/metal/tools/

Mobile Renderer improvements

Dedicated mobile renderer that supports forward and deferred modes

5.4 - Optimizations for dynamic lighting to support for many local lights

5.4 - Improvements in cached draw commands on mobile reducing CPU cost for object updates

->See Dmitriy Dyomin’s session at 1:30pm in this room for the details

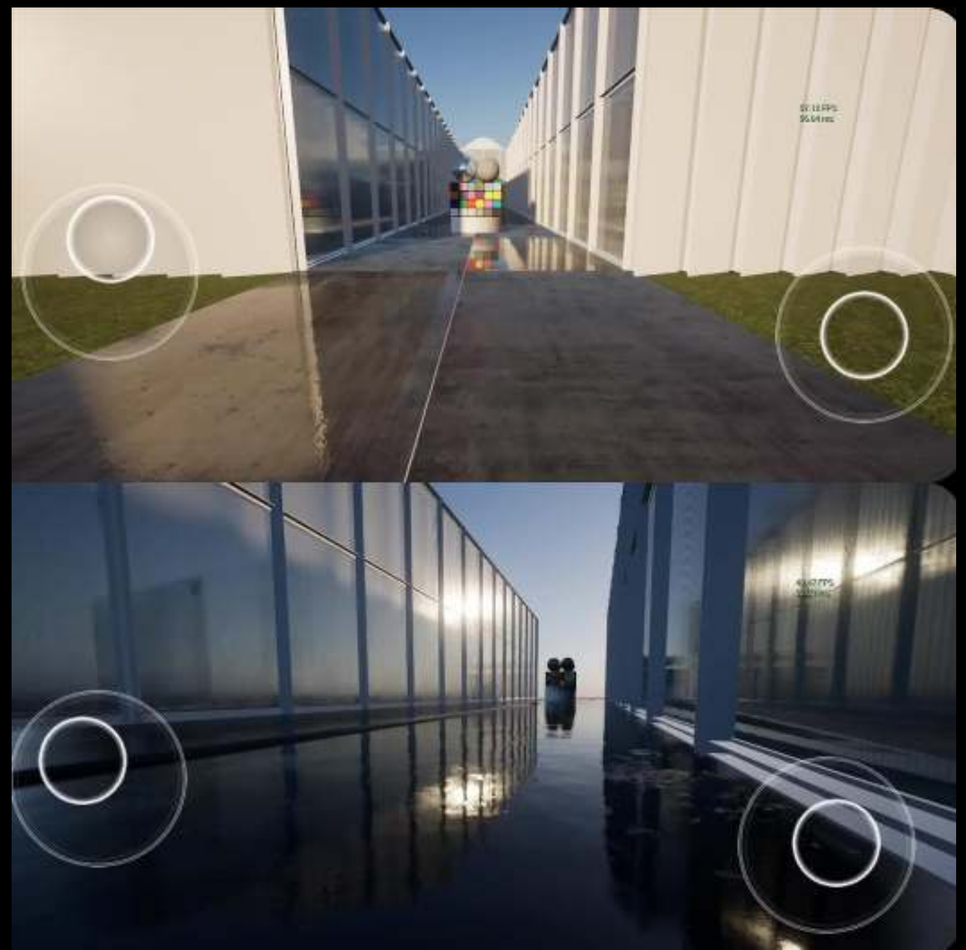
Mobile Renderer improvements

Features in the works...

5.5 - Mobile forward renderer is adding support for more rendering features

- D-buffer decals
- Rect lights
- Capsule shadows
- Point light shadows

5.5 - Added screen-space reflections



Desktop renderer

Nanite and Lumen on Android and iOS?

UE 5.4 - added experimental Raytracing support for Lumen on Android

- The latest Xclipse, Adreno and Mali-Immortalis mobile GPUs with Ray Tracing are supported

UE 5.6+ - iOS support is planned but requires an update of Metal Shader Converter

Desktop renderer

What about Nanite?

Regular Nanite on mobile is still several GPU generations away

- Mobile GPUs are beginning to support for 64-bit atomics needed for software rasterization pipeline
- But compute shader performance on mobile is still limiting factor
  - Nanite has recently moved to a Compute Shader pipeline to evalue materials

What we probably need is a more mobile-optimized solution to bring the continuous LOD and draw call benefits to mobile developers

→ **Maybe a Nanite path using hardware rasterization only is best for mobile, needs further research**

Conclusion

Returning to this roadmap...

Android setup experience

Mobile & platform preview

Memory analysis tools

On-device iteration

Memory savings

Mobile renderer

Runtime PSO caching

Lumen

Nanite

Thanks!

Any questions about these topics or other Unreal Engine mobile topics?