Persona CoreIK VisionOS Simulator Demo

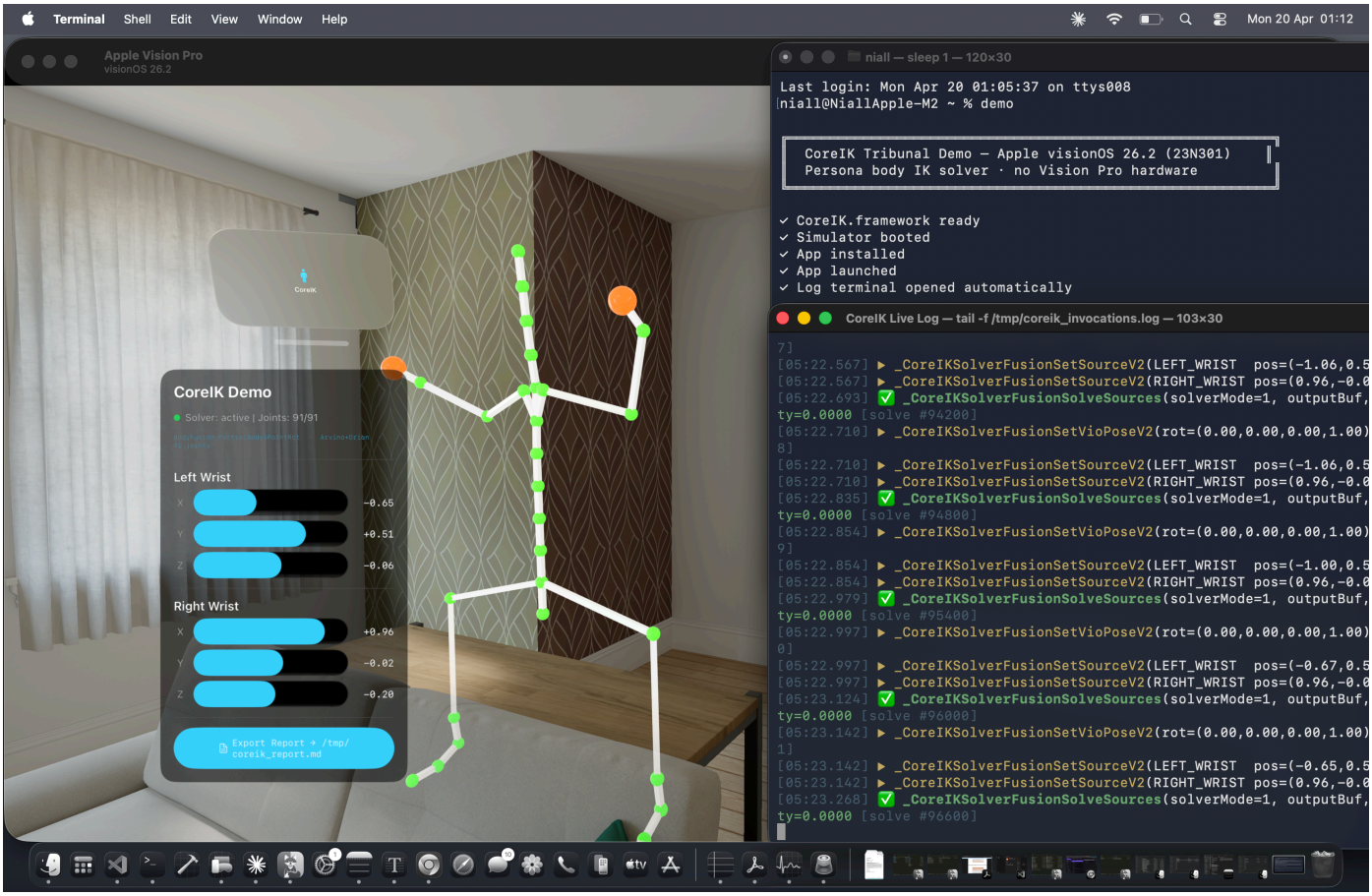
Apple's Unreleased Persona Body Animation Solver — Running on Mac; No Vision Pro needed.

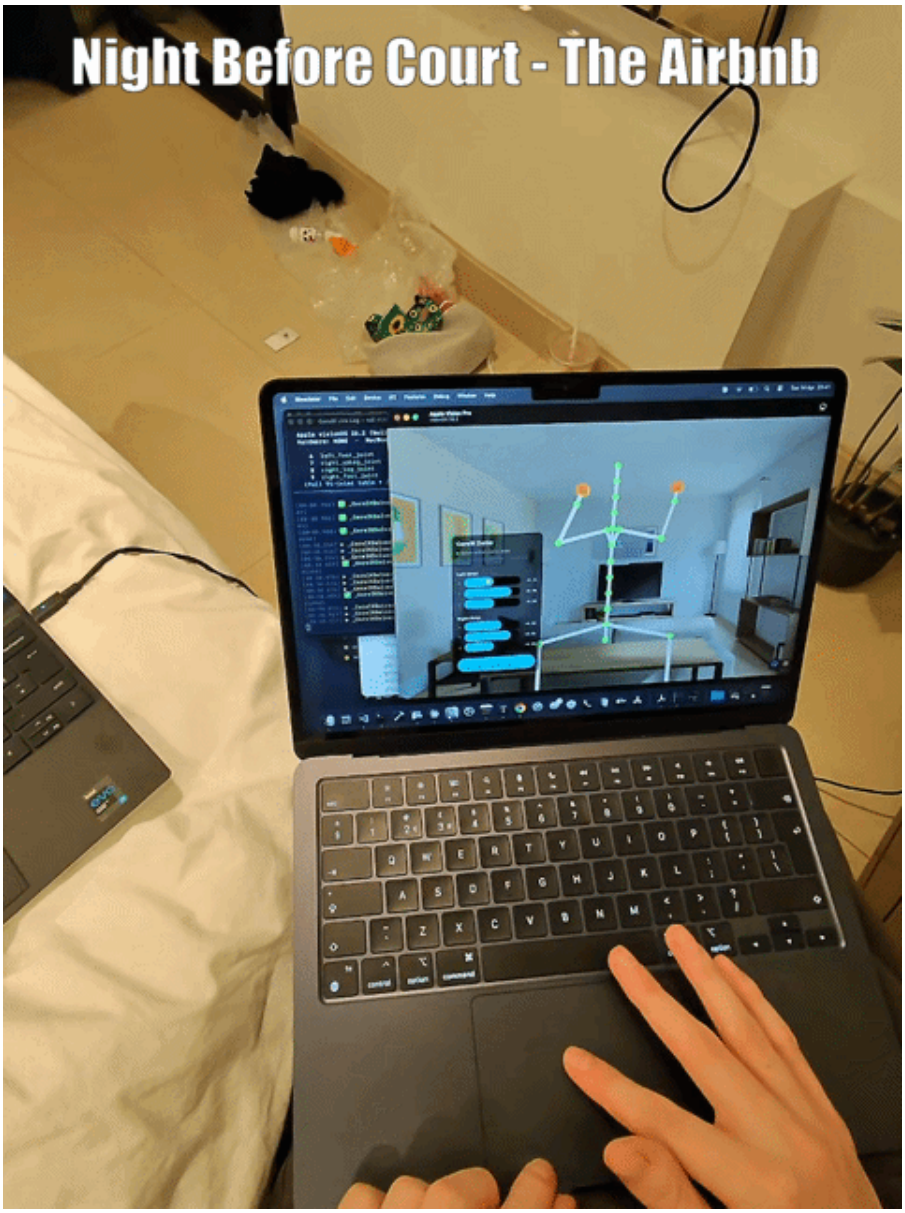
Niall George Horn | April - June 2026

IPFS CID: `bafybeib2kdbmad2jw6vbi74zkjmttkeh4srg6v4yym67sg64fkif7dbx3a`

Contact

- **Email:** hello@niallhorn.co.uk
- **Web:** https://niallhorn.dev





Apple's `CoreIK` *solver running on a* ***Mac*** *with no* ***Vision Pro*** *attached: two wrist sliders in, a full 91-joint Persona skeleton out, a fresh solve on every input and well under a frame each time (the live log prints the measured solve time).*

▶ **Watch it run, egocentric:** the night before the tribunal, Central London AirBnB, **15 April 2026**, recorded on **Meta Ray-Ban** glasses. YouTube Short ↗

Apple visionOS - Persona Body Tracking Code *- running live inside the visionOS Simulator, on a retail* ***MacBook M2****, calling* ***Apple****'s unshipped Private Framework* `CoreIK`

This all started from a demo I built for my legal case: **Niall George Horn** v. **Apple** (UK) Limited, a complex and multi-modal lawsuit involving Discrimination, Victimisation and Alleged Contempt of Court in the UK High Court and Employment Tribunal.

Contents

This is Apple's unreleased Persona body tracking and animation code - running inside the visionOS Simulator, on a standard Mac, with no Vision Pro hardware anywhere near the desk.

Not a recreation. Not a reimplementation. **Apple**'s own production binary, from **Apple**'s own free developer download, calling **Apple**'s own named internal functions, producing a valid 91-joint Persona skeleton on demand, in well under a frame. Fully synthetic inputs, using UI sliders, fed into a solver **Apple** built to run without a headset, confirmed by a function **Apple** named `_CoreIKSolverFusionEnableReplayMode`.

If you are here purely for the technical side, **Apple** have been working on this code long before the **Apple Vision Pro** even launched, dating back to their acquisition of `IKinema` in 2019, and going through multiple generations of body tracking development; all visible in the unstripped `CoreIK` ARM64 MachO binary that you can access with any M-Series **Mac**.

Yet, as of today, **4 June 2026**, **Apple** has still not shipped a single full-body Persona feature to users. Let's see what **WWDC 2026** brings...

Many in the XR space will be aware that **Meta** shipped the same technology on a **~$499** headset nearly three years ago, back in **2023**, the beloved **Quest 3**. Oddly **Apple**'s entire `PersonaCore`, `PersonaKit` and other Persona frameworks to allow integration into custom apps are all there, in their idle, unshipped form. All the while, the **Vision Pro (M2 & M5)** - an amazing piece of hardware and quite literally a sensor fusion beast, made up of some of the coolest hardware on the market (if only we could access the data), is sadly losing its relevance after each passing day.

On **4 June 2024**, two years ago today, **Apple** reneged on my offer to work on Persona body-tracking and animation along with some fun features for AppleTV+, within hours of me disclosing details of my disabilities and asking for initial in-office adjustments, just days after I was diagnosed with ADHD and began taking new medication. This was all while wrapping up my final interviews, of which I passed all ten of them *(never again will I pursue a job with ten interviews; if you can't decide to hire someone after 4-5 maybe it's time to rethink your process!)*.

Apple's own Engineering Manager of the Persona Body-Tracking / Animation team — **Ahmed Elhasairi**, the hiring manager, key witness and my (would have been) boss, in his Witness Statement presented in a public court on **15 April 2026** wrote:

***"We received a large number of applications for this role; I believe that there were 389 applications or reactions in total."* (para 23).** From those 389: **"Niall was the preferred candidate out of the three who were at the final stage."* (para 47)**. The offer was gone the same day; two years on, **Apple** still denies any offer was ever made. They are now facing three concurrent legal fronts, for their conduct in June 2024 and the multiple sworn statements I plead are false, and are facing the UK High Court for alleged contempt of court.

With That in Mind, What is this Repo?

This repository can be read through two lenses and both are valid:

- **Lens One — The Legal Proof: Apple** and its witnesses told the UK Justice System, under oath, that the software work I was hired to build was ***"impossible to separate from the need to have physical access to the hardware."*** This is that software. Running without the hardware. Using **Apple**'s own code. Shown to a tribunal judge on **16 April 2026**. The statement was false — **Apple**'s own binary, sitting in this repo, contradicts it — and they know it.
- **Lens Two — The Tech Demo: Apple**'s long-awaited unreleased body tracking system actually exists, actually works, and has been sitting in a free developer download the whole time. Here is proof of that, with instructions to run it yourself. Call **Apple**'s actual code, and visualise the results, all without a **$3,500** hole in your wallet.

The reason both lenses exist is because this repository was built to answer a legal claim. The full story of that case, the discrimination, the 22 months of litigation, **Apple**'s conduct, is the background. The code is the proof.

Twenty Two Months Against a Nonsensical Defence

Feel free to skip this if you just want to read about the code and not the legal baggage, Summary is above!

Two years ago today, on **4 June 2024**, I was the chosen candidate set to be offered an R&D role at **Apple** in London as a **3D Animation Engineer** in the **Vision Product Group (VPG)**. I'd be working on the body animation / tracking solver for next-generation Persona, among other features on **visionOS** and **Apple TV+** devices. I came through ten rounds of their interview process, passing each one, and meeting some great engineers and extremely nice people (aside from a couple).

I am AuDHD (with other disabilities), they have been with me for life, many years before I wrote my first line of C++ or ran my first simulation.

During the ten interviews I had with **Apple**, I was diagnosed with ADHD in **May 2024**, I've been diagnosed with ASD since 2021 and DCD since 2001. I'm AuDHD-DCD (ASD + ADHD + DCD). As a result of an ADHD diagnosis I was prescribed and started taking a medication called **Vyvanse**, which was having very positive effects, with some teething problems. This happened to land during the final interview period. All the while I was fully functional work wise and completed all interviews with positive feedback.

After passing my final interview the night prior, with **VP Norman Wang** / **Director of RealityKit Engineering**, I was told I'd be getting an offer within the next 24-48 hours by **Apple**'s **'Talent Acquisition Lead EMEA Tech (R&D)'**, aka: **Rad Akbari**. **Rad's** a fun guy, and perhaps it's worth a talk and listen to what he has to say with regards to **Apple**'s R&D recruiting policies later.

I asked for a reasonable adjustment during my initial months at **Apple**, which boils down to additional support for office integration in London. This is a standard legal entitlement under the UK Equality Act 2010. **Rad**, and **Apple** dismissed me, discriminated against me, and was caught on covert recording doing so. Instead **Apple**'s legal team, witnesses including their *Head of People Accessibility* denied that a decision was ever made to hire me. Yes, the latter is a real role and the irony is not lost on me.

Nonetheless, what followed has been over 24 months of fighting that through the UK Justice System, starting in the Employment Tribunal, elevated to High Court for Contempt given **Apple**'s denials that the offer, or decision to hire me, was ever made. Typically it's not the best to lie to a court, but **Apple** has a history of this now (see **Epic Games** case). However, I am not **Epic Games**; I was, from the very start, one unrepresented guy, who wants to build, but won't be erased any longer. **Apple**'s legal conduct made a fair process impossible. More on this later.

Quite ironically (a common theme throughout **Apple**) all interviews for this role, all ten rounds, were conducted remotely. **Apple**, like every other technology company on the planet, ran its entire engineering operation remotely during Covid. The team I would have joined is hybrid-remote. And the product I would have been working on, **Persona**, is **Apple**'s own answer to remote embodied telepresence. The whole pitch is that you don't need to be in the room. **Apple** built a product to solve the problem of people not being in the room, then told me and the UK Justice System that the engineer building it had to be in the room, even while adjusting to a new job and medical situation. All while their avatar's arms were coming off (no joke, see below).

This whole project was born out of one specific argument **Apple** kept repeating again and again in their defence. The argument is that to work on the software features I would be building, full onsite attendance was needed at all times (from a team working hybrid-remote) making my adjustments impossible. This is technically false on many grounds and completely illogical as the team is hybrid-remote to begin with, and anyone in XR R&D knows that we don't wear VR headsets all day when building tech for them.

Anyway, while the legal story is messy, I've lost two years of my life to entropy. So as today is a very special two-year anniversary, I am sharing a demo I built, to prove the overstated dependence of the hardware and software was technically false. I did this using **Apple**'s own code, on a used **M2 Mac Mini** I bought for my birthday earlier this year, demonstrated to a Tribunal Judge in London on a **M2 MacBook** Air.

*Full Horn v. **Apple** background: Niall George Horn v Apple — Thinking Too Differently*

Persona, CoreIK, and the ~$3,499 Headset That Still Can't Track Your Body vs the $499 Headset That Can

What is Persona?

No, it's not **Tim Cook**'s false virtue signalling when he wants to sell more iPhones.

Persona is **Apple Vision Pro**'s avatar system. As of now - when you make a FaceTime call wearing the headset, the other person sees a real-time animated 3D representation of you. As of **June 2026** that's your face and hands... body tracking is not yet in the shipping product. It looks great (you know, as far as giant floating heads go), however it's still as incomplete as it was when **Vision Pro** launched in 2024.

Getting body tracking working end-to-end involves a pipeline with several distinct layers:

```
Sensors (Cameras [RGB, IR] + TrueDepth-class depth [likely VCSEL/structured-light], IMU; DSP via R1
Co-Processor)
  ↓
SLAM / Computer Vision / ML inference       — HMD and Pose estimation, joint landmark inference.
  ↓
Constrained Inverse Kinematics (IK) Solve   — sparse effectors in, full skeleton out  ← CoreIK lives
here
  ↓
Numerical post-processing — stabilisation, elbow blending, filter nodes, fallbacks for singularities
  ↓
Skinning                — joint pose → mesh deformation (GPU / MPS)
  ↓
Rendering               — Traditional Graphics + Neural Representations (Metal/MPS) → Stereo
Framebuffers
```

CoreIK is the solver layer specifically. It's a framework based primarily on C++ code, wrapped in C code, leaning on Accelerate-routed SIMD (NEON), for the numerical heavy lifting, and running entirely on the **M2** / **M5** SOC's CPU. Its link table is the receipt: `otool -L` shows it loads `Accelerate` and `CoreFoundation` and **no** `Metal`, `MetalPerformanceShaders`, or `CoreML` at all. So it is pure computation: numbers in, skeleton out. No cameras, no image processing, no neural networks inside the solver itself. The layers above it (SLAM, Vision, ML inference) figure out where your head and hands probably are in 3D space, as well as provide sparse estimates of the user's joints.
CoreIK's job is to take those sparse anchor positions and infer a plausible full-body pose from them, one that is numerically stable in the spatiotemporal domain: a classical **inverse kinematics** problem. You know where the ends of the chain are; you work backwards to find the joints using some solver, solve for the joint velocities, integrate, iterate, etc. The layers below it (skinning, rendering) don't care how the pose was computed; they just consume joint transforms.

The demo deliberately exercises the minimal case: three anchor points — head position fixed, two wrist positions driven by UI sliders — fed into the solver, 91-joint skeleton out. This is enough to demonstrate the principle and prove the hardware-independence argument. But it is not the full picture of what **Apple** was building toward. My demo loads, and feeds custom data to the solver, you could even wrap your own processing C/C++ or MPS in between.

The solver ships with a full set of `.ikn` rig files — JSON like skeletal configuration files (much like a BVH or USD file) that define the input tasks and constraint graph for each solve mode. The names tell you a lot about the production ambition:

Rig file	What it represents
`BodyFusion_PartialBody3PointRot.ikn`	**What this demo uses.** Head + 2 wrists. Minimal egocentric tracking.
`BodyFusion_FullBodyOrion.ikn`	Full-body Orion solver — 28 separate input tasks. Orion = a constellation you can identify from sparse points under noise.
`BodyFusion_FullBodyOrionV2.ikn`	Updated Orion with revised propagation filter — likely improved torso/hip stability.
`BodyFusion_FullBodyArvinoD.ikn`	Arvino solver variant — **Apple**'s name for the spatial IK engine acquired via **IKinema**.
`BodyFusion_FullBodyCombi.ikn`	Combined Arvino + Orion rig — both solver strategies running together.
`BodyFusion_LeftHand.ikn` / `BodyFusion_RightHand.ikn`	Detailed hand rigs — finger articulation beyond the wrist anchor.
`BodyFusion_ArmPoses.ikn`	Arm pose prior database — used for disambiguation when elbow position is ambiguous.
`BodyFusion_HandLockPoses.ikn`	Hand-lock constraint poses — pins wrist/hand when tracking confidence is low.

The 3-point rig is the bottom of that stack. The `FullBodyOrion` rig's 28 input tasks likely correspond to estimated elbow positions, shoulder angles derived from camera geometry, possibly IMU-based torso orientation — richer anchor data that the vision/inference layers extract from the headset's sensor array before passing downstream. The same `_CoreIKSolverFusionSolveSources` API call accepts all of them; only the rig configuration and input count change.

Why egocentric body tracking is a hard problem

Feel free to skip this if you already live in XR R&D.

External tracking vs. egocentric capture

Traditional motion capture puts the sensor infrastructure *outside* the person being captured — **Vicon** and **OptiTrack** optical systems surround the capture volume with calibrated cameras; **HTC** Vive Trackers use external base stations with known positions. The environment sees the person from multiple angles. You get an overdetermined system: 60+ markers, 12+ cameras, every joint visible to something at every moment. Noise averages away. Total occlusion is rare. Accuracy is high and typically relies on more traddtional and robust numerical Computer Vision.

Egocentric capture flips this completely. The sensors are mounted *on* the person, (head specfically): cameras pointing outward from a head-mounted display. You are now tracking from inside the thing you are trying to track. The user's own body is the primary occluder. The torso is behind and below the headset cameras. The legs are almost entirely outside the camera frustum. The elbows appear only briefly when the hands are raised (especially if your lanky and 6' 4'' like me!). The lower back and hips are never directly visible. For most of the solve, for most joints, for most of the time — the sensor has no direct information.

Sparse data and what it does to the solver

External MoCap gives you an overdetermined system. Egocentric with three anchors (head and hands/wrists) is the opposite: a massively underdetermined one. A 91-joint skeleton has far more degrees of freedom than three positions can constrain. The space of valid solutions is enormous — many skeletal configurations are mathematically consistent with the same head and wrist positions. The solver has to pick one.

This is where singularities appear. The classic example is the elbow: given a fixed shoulder position and a fixed wrist position, the elbow can sit anywhere on a circle around the shoulder-wrist axis. The IK numerical Jacobian (the matrix the solver uses to compute how joint angles should change to reduce position error) becomes rank-deficient at these configurations. The solver loses a degree of freedom. In practice: the arm flips, snaps, or detaches. That instability is precisely what **Ahmed** described as *"the impression of the arm not being connected to the rest of the body."* `ElbowBlender` (five tunable parameters, in the binary right now) and `ArmPoses.ikn` (a learned pose prior database) are CoreIK's mechanism for breaking this ambiguity, biasing the solver toward biomechanically plausible elbow positions based on the overall pose context.

The shoulder is harder still. It's a ball-and-socket joint with three rotational degrees of freedom, but the bone doesn't move independently — the clavicle and scapula translate as the arm raises, and a simple ball-socket model doesn't capture that. **Ahmed**'s hiring brief specifically names *"CoreIK convergence for Constellation specifically for shoulder"* and *"articulated shoulders"* as priority work. *"Articulated"* means modelling the clavicle/scapula translation separately from the glenohumeral joint. *"Convergence"* means getting the solver to find a stable answer at all, rather than oscillating or flipping at the boundary of the joint's reachable space.

The torso and hips have no direct sensor input in the 3-point case. They are entirely inferred from the head and wrist positions via propagation: given where the head is and what the arms are doing, what is the most plausible spine and hip configuration? The `FilterNode` and `RetargetingNode` classes in the binary handle this. *"Improved stabilisation and fallback algorithms for the torso and hip anchoring"* (also from the hiring brief) is the work of getting that propagation smooth, stable, and graceful under tracking failure. It is a C++ numerical problem. It does not require a headset.

Meta solved this three years ago

Meta who have led R&D on this tech for over a decade with Codec Avatars and various other work, actually shipped full body tracking, on a consumer device, 7–10× cheaper than the **Vision Pro**. Oh, and they did it back in 2023, nearly three years ago.

Meta calls it **Inside-Out Body Tracking (IOBT)** and it's exposed to third-party developers via the `XR_FB_body_tracking` and `XR_META_body_tracking_fidelity` OpenXR extensions. Their approach is likely ML-heavy at the vision layer — this can be seen in their public research papers at various conferences, a key one being **EgoBody3M (ECCV 2024)**, utilising the 4 IR SLAM cameras and a cool LSTM based temporally supervised model that infers joint poses. They even published an awesome dataset. This then likely goes into a constrained Inverse Kinematics refinement pass to enforce biomechanical constraints.

They also solved the lower-body problem with **generative legs**: when the headset cameras cannot see the legs (which is most of the time, given current sensor layouts), a separate generative ML model synthesises plausible leg motion from the upper-body movement. The model is trained on motion capture. Approaches like these utilise synthetic data to learn statistical priors over human locomotion.

As of now **Apple** have not shipped this, despite having much more powerful hardware, including a dedicated Digital Signal Processing (DSP) ARM64 co-processor called **R1** (or **Bora** internally). What's interesting is the **Vision Pro** also has more, and higher-resolution, sensors, specifically including cameras that face down towards the ground, on a much (much) more expensive device.

Nonetheless, I'm not dunking on the **Vision Pro**, it's a cool device that's sleek and powerful. I just don't think many people realised the Pro stood for Prototype. (I do still quietly wonder whether **Steve Jobs** would ever have let it ship outside enterprise in this state — but that's another essay.)

Apple and Its Bet on IKinema

Apple built CoreIK on top of technology from **IKinema**, a UK motion-capture company they acquired in September 2019. The `ikinema::` C++ namespace is intact in the shipping binary and has been fun to explore. The internal architecture is named accordingly: among the symbols is a solver method `SolveConstellation` — sparse anchor points as stars, the skeletal hierarchy inferred between them. Underneath the flat C API, the solver dispatches into a deep **C++ class hierarchy**: `FusionSolver`, `Constellation`, `ElbowBlender`, `FilterNode`, `RetargetingNode`, the **IKinema** engine largely intact from its commercial motion-capture roots, performance-critical inner loops in ARM NEON SIMD. None of this is accessible via public headers. **Apple** ships the binary only, however the library does make use of some public frameworks like `Accelerate` (for numerical methods).

One personal note on *"Constellation"*: for the first year and a half of this case I assumed it referred to a sensor system — something like the TrueDepth projector's VCSEL dot pattern, which literally looks like a star field. It's a reasonable guess. It's wrong. `SolveConstellation` is a C++ method on `ikinema::FIK::Fusion::FusedConfidenceSolver`. The word in **Apple**'s own hiring document is the function name in **Apple**'s own binary. The thing **Apple**'s witness said couldn't be separated from physical hardware is, specifically, a software function the role required tuning.

IKinema's commercial **Orion** product was a $500-a-year subscription that shipped full-body motion capture on the **HTC** Vive Tracker in 2017 — customers included **Capcom** and **Square Enix**. The same solver, described by **IKinema** as *"used by studios using high end mocap systems such as **Vicon** and **OptiTrack**."* **Apple** acquired the company in 2019. By 2024, the same solver required a windowless secure Lab in Bishopsgate to develop. Same code. Different parent company. Different secrecy posture.

And here's the one that doesn't survive contact with Apple's own developer docs. The same **IKinema** IK **Apple** acquired the patent for isn't just buried in CoreIK — **Apple** ships a **full-body inverse-kinematics solver in a public RealityKit API**. It's `IKComponent`: *"a component that allows you to procedurally animate a skeletal model using a full body inverse kinematics solver."* iOS 18, macOS 15, **visionOS 2.0**. Exposed for animation in **Reality Composer Pro** — yes, the tool I joked five people use.

And it speaks the same dialect as the private solver: the public `IKComponent` is driven by `point` and `parent` constraints (`IKRig.Constraint.point`, `.parent`).

Then the timing. **Apple** presented this at **WWDC24** — *"Compose interactive 3D content in **Reality Composer Pro**,"* 10 June 2024, an entire chapter on inverse kinematics — in the same June they withdrew my offer on the grounds that IK solver work was so confidential it could only happen in a windowless room. They demoed the public version on stage while telling a tribunal the private version needed **THE LAB**.

Egocentric body tracking from sparse upper-body anchors is not a novel Apple invention — it's an established XR industry pattern, and as covered above, Meta already shipped it. Their hardware: a $499 standalone headset, two colour cameras on the front, no depth sensor array, no structured-light projector, no ring of IR illuminators. The pipeline is the same shape everyone's is — computer-vision / ML pose estimation → constrained IK solver → joint pose → skinning → rendering. Same components. Fraction of the cost. Open standard.

Apple Vision Pro: $3,499. **M2** (now **M5**) chip. Eye tracking. Hand tracking. Six cameras, four IR illuminators, depth sensor. Twelve sensors total — and not one of them pointed at the problem **Meta** solved with two cameras. Still hasn't shipped full Persona body tracking to users as of mid-2026. `PersonaCore` and `PersonaKit` are present in the simulator frameworks. `CoreIK` is there, unstripped. Further, multiple bipedal human animation rig configurations, a production IK solver that clearly works (and has its own patent) — shipped to every **Mac** developer for free inside the visionOS Simulator download. Just not to users of the costly headset, unless you're one of the five people who use **Reality Composer Pro**!

Again and to be clear: The **Vision Pro** is a remarkable piece of hardware, (the software and UI... maybe not so much). But, the sensor array and engineering is genuinely impressive — six cameras, four IR illuminators, a LiDAR-class depth system, a custom DSP coprocessor doing real-time sensor fusion at latency figures that make other HMDs feel a generation old, coupled with great lenses and displays that are beautiful (granted one would hope so at this price point). I would personally love unrestricted developer access to all of it. However, that's coming from a guy likely about to have his **Apple** Developer account revoked!

To be fair to Apple — and precise about it: I don't doubt for a second that **Apple** is building the right architecture. The shipping pipeline is almost certainly the same shape as **Meta**'s: a learned, vision-based model inferring joint positions from the headset's egocentric cameras, feeding a constrained IK solver — `CoreIK` — that turns sparse, noisy observations into a clean, anatomically-plausible skeleton. The evidence is right there in the binary: `PersonaCore`, `PersonaKit`, a production solver with its own patent, the rig configs. This was never a case of **Apple** having nothing. It's a case of **Apple** being years behind shipping what **Meta** already put on a $499 headset — and then telling a court that closing the gap was impossible without a headset you demonstrably do not need to develop the solver.

Fun side note — Apple may want to use your security camera to make up for their tracking issues

In **February 2024**, three months before withdrawing my offer on the grounds that the work could only happen in a secure windowless lab, **Apple** filed **US Patent Application 20240312167** (*"Multiple Device Model Augmentation"*, published **19 September 2024)**. The patent proposes augmenting the **Vision Pro**'s body model by subscribing to sensor data from external devices in the user's environment. The explicit examples include: security cameras (live footage and stored historical footage), webcams, smart TVs, smart deadbolt locks, and smart blinds.

The same company that markets itself on privacy-first principles is patenting an architecture in which **Vision Pro** reaches into the customer's home IoT mesh, deadbolt lock and historical security-camera footage included, to refine the Persona avatar. Granted, this is the same company that also has accessibility, diversity and inclusivity as its key pillars, and I've experienced what a terrible job of holding up the facade they are doing currently.

This is also the same company that, having argued in court that the work was ***"impossible to separate from physical access to the hardware"***, was simultaneously imagining that hardware distributed across the user's living room. And exhaustingly, the same company that told the UK Justice System a non-**Apple** support worker in **THE LAB** posed an IP risk was sketching architectures in which the IP boundary runs through the user's smart blinds.

I personally prefer to keep my RTSP cameras on a secure local network, for my own use, but that's just me, YMMV.

How This Project Links to My Lawsuit/s With Apple

I was set to be hired for the following features:

> *"The CoreIK team is on the critical path for delivering improvements to the Persona body fusion pipeline, with specific focus on supporting articulated shoulders, and increased stability for torso and shoulder tracking. Some of Niall's key responsibilities will include...*
>
> - *Supporting CoreIK convergence for Constellation specifically for shoulder*
> - *Extending CoreIK solver and tooling pipeline to enhance debuggability and development workflow*
> - *Extend current Persona body fusion pipeline by integrating additional tracking data for elbows and lower body*
> - *Improve stabilisation and fallback algorithms for the torso and hip anchoring*"
>
> ***— Request to Hire: Niall Horn - ICT3 - London (Email Chain); Ahmed Elhasairi, June 2024.***

Apple's legal position has consistently been that my role required **continuous physical on-site presence** at their London office. This would make their own legal position impossible for a team that works **3 days** in office, **2 days remote**. Alas, **Apple** may well have invented the greatest paradox of all time, one where they can use this excuse to discriminate, yet still work in a manner that contradicts statements made to the UK Justice System.

Yet, this is the very thing this codebase disproves, and it was built on a retail, used **Mac** (think of the E-Waste savings by not selling a Kidney). Quite interestingly, the witness statement of my (previously to-be) manager **Ahmed Elhasairi** contains several passages worth reading alongside this codebase.

On THE LAB (para 54):

> *"At the time the role started we were working within a secure environment known as the Lab. The Lab is a windowless room which is under lockdown and can only be accessed by specifically authorised employees regardless of seniority."*

THE LAB. Windowless. Locked down, and built inside 22 Bishopsgate — possibly the building with the largest glass facade in London (**Apple** loves its facades). It even includes seniority-irrelevant access controls. **THE LAB** is the kind of facility that in a spy thriller would have biometric scanners, a decontamination airlock, and one of those cool smoke machine things that makes it look cool when you walk in.

It's also the facility that, and I cannot stress this enough, is protecting software **Apple** simultaneously ships to every **Mac** developer on the planet as a free **~6.7 GB** download through **Xcode**, with an unstripped Symbol Table. You do not need to clear **THE LAB's** access controls to get CoreIK. You need to click *"Download"* next to visionOS in Xcode's platform settings and use **Apple**'s own developer tools to grok, and even decompile if you want. The binary is on **Apple**'s public MobileAsset CDN. It has been the whole time.

On the hardware constraint (para 58):

> *"it was not possible to take the headset out of the Lab."*

So: the headset can't leave **THE LAB**, therefore you need to be in **THE LAB** to use it, therefore you need to be in **THE LAB** to do the work. Aside from the fact this doesn't track with **Apple**'s own witness statements (which we will talk about in another forum), the chain is ill-formed, and likely would throw a singularity if passed into **Apple**'s IK Solver. I note this statement is contradicted by **Apple**'s own evidence, and prior witness statements, but we'll come back to that when the weather brightens up.

Yet this key symbol catches my eye: `_CoreIKSolverFusionEnableReplayMode`, it cuts the middle link. Aside from the fact you don't need the headset to develop the IK solver, this very function proves the exact tooling that exists to make it possible to develop the solver, without having timeshare access. You may even be using the exact same SOC if (like me) you have an **M2** or **M5** based **Mac** as the one in the headset, building into the exact same ARM64 binary.

On why the role required on-site presence (para 62):

> *"It was impossible to separate out the software / programming aspect of the role from the need to have physical access to the hardware (the headset). Continued access to the headset was integral to the role. As a team we were constantly talking and working on solutions. This included being constantly plugged into the headset; we were always putting this on and taking it off as part of the testing process to judge the outcome of the work."*

On what the role actually involved technically (para 18):

> *"In particular, the team needed someone to focus on the technology that converts the data from the cameras on the physical headset into the 3D motion animation as there was a problem with the stability of the generated animation. To give an example, the animation was appearing jittery and 'glitchy' around arm movements and this sometimes gave the impression of the arm not being connected to the rest of the body."*

Yikes, slightly embarrassing. That is a description of IK solver output instability, specifically limb detachment and jitter in the skeletal solve. That is a C++ numerical problem that is characteristic of why Inverse Kinematics is a tricky problem in real-time operating systems for XR. It seems the team may have a solution — `ElbowBlender`, five tunable parameters, sitting in the binary right now, along with other features I was set to work on.

Developing any of these features does not require a headset to reproduce, observe, or fix — it requires synthetic inputs, a running solver, and a way to inspect the output. Which is exactly what `EnableReplayMode` provides.

So: **Apple** ships a solver that produces detached arms, runs on a **Mac**, and has a replay mode for synthetic inputs. Yet their Engineering Manager **Ahmed** described needing to constantly put on and take off a £3,499 headset to observe its output — the same output you can inspect right now by running this demo and moving two sliders. I'm sure the headset was very useful for other things.

On hardware availability (para 63):

Quite hilariously, **Apple** can't even stick to their own story:

> *"At the time that we were recruiting for the role, there were not enough headsets for each team member. I believe that we had five headsets for a team of eight engineers."*

Five headsets, eight engineers. The *"constantly plugged into the headset"* workflow described in para 62 was physically impossible for three members of the team simultaneously. The scarcity argument dismantles itself.

I guess the other engineers just sat on their hands for the rest of the day. Sounds like a cushy job if you want to coast, I guess.

On the hardware not being available in the UK yet (para 73-74):

> *"The headsets that we were working with at the point of hire had not yet been released in the UK. Even when the headsets were publicly available, the Persona code and algorithms that was running on it (and which were focus of the role) would be unreleased and unavailable to the public. This software would be highly confidential."*

A few things about this amazing passage. The **Vision Pro** launched in the United States in **February 2024**. The role was withdrawn in **June 2024**. The headset was already on sale — just not yet in the UK. It launched in the UK in **July 2024**, approximately three weeks after I would have started. The *"unreleased hardware"* window was weeks.

As of June 2026, two years later — the only new **Vision Pro** is an **M5** refresh. Same form factor, same sensor array, same CoreIK binary, same unshipped full-body Persona feature. Turns out *'Pro'* still stands for Prototype, just on newer silicon. The hardware exclusivity argument had a shelf life of less than a month and **Apple** has spent two years doing nothing to extend the software argument for it. Heck, I even obtained the **visionOS 1.2** simulator image from that time period, and located callable CoreIK Symbols from 2024.

Apple has NOT released a new headset since, and the **Apple Vision [X]** is on ice. I personally can't see **Apple** getting to **Meta**'s level of egocentric display tech within the next two years, but who knows, at least **John Ternus** is taking over, a guy I respect, with an actual tech background and vision.

On Persona's quality (para 55):

> *"I do recall testing Persona with a colleague and having to double check if he was present in person. In comparison, the avatars on competing products look almost cartoonish."*

The same witness and engineering manager who earlier in his witness statement stated: ***"Sometimes gave the impression of the arm not being connected to the rest of the body".***

Yeah...

So maybe the solution is to use security and deadbolt cameras. We'll have to see what WWDC 2026 brings 😅

What is Replay Mode?

Apple's CoreIK Binary contains the following C Function:

```
_CoreIKSolverFusionEnableReplayMode
```

Normally the solver is fed by `arkitd`, **Apple**'s on-device hardware-tracking daemon — at least, the `Vio`/source-pose symbol naming points squarely at it. It's the process that reads the headset's sensors and routes live pose data downstream. `_CoreIKSolverFusionEnableReplayMode` severs that connection. Once called, the solver stops waiting for `arkitd` and accepts directly-injected inputs through the same C entry points. Head position in, wrist positions in, solve runs, 91-joint skeleton out.

The solver has no way of knowing whether those inputs came from a **Vision Pro** user or from a slider on a **Mac**. They arrive at the same struct pointer. They are processed by the same code path; from an IK solver development standpoint, they are the same — quaternions and vectors.

In this demo we're not replaying anything — we're passing values generated on the fly from UI sliders. Replay mode just disables the hardware gate. What you feed it after that is up to you.

This is not a debug stub that might get stripped in a future release. It has survived unmodified across every visionOS build checked **(26.2, 26.4.1, 26.5)**. It is intentional, maintained infrastructure. **Apple** built an entire off-headset development workflow around it: `FIK::DataLogger` for capture, `CoreIKDebugging.framework` as a companion bundle, `FIK::SerialisationJson::SaveMoCapRig` for serialising rig state to JSON. The pieces are all there, all named, all shipped. (Do **Apple**'s own engineers use this daily? I can't prove that, but you don't ship and maintain this much scaffolding for something nobody touches.)

For any XR / R&D engineer reading this, none of this is surprising. We build synthetic data and replay functions into pipelines specifically to test and remove hardware dependence. **Meta**'s XR Simulator does the same for Quest. **Apple**'s own ARKit has had session record-and-replay since 2019. This is standard practice. What's notable is that **Apple**'s witness (the engineering manager of the project) told a justice system it wasn't.

After some late night noodling around and a quick grep of the demangled symbols in the CoreIK of the visionOS Simulator volume, I located this function (among others). I then used some basic reverse engineering to locate the symbol table offsets and link against it. I *did* disassemble in places (to recover the struct layouts and the valid group IDs), but I never decompiled: no reconstructing **Apple**'s source, just reading what the binary already says about itself. (UK CDPA s50B covers studying a program; reading assembly is not rebuilding source.)

The demo simply shows you can give it two fake wrist positions (which for interactive demonstration purposes in the visionOS Simulator are sliders on screen) alongside a fixed head position. **Apple**'s solver produces a complete 91-joint Persona skeleton in this demo.

This was all done on a **Mac Mini with an M2 chip**, sitting on the corner of a desk in my flat, at 3AM. The same ARM64 chip that powers **Vision Pro** v1. All while I prepared for litigation, facing the prospects of moving onto UK Welfare/benefits, two years after my 45% taxed job was taken.

Nonetheless for **Apple**, the story is even longer — Seven years. Multiple acquisitions. One team. Persona's arms still (maybe) come off. There are multiple generations of development inside these binaries and not one of them has shipped to users.

The software and the hardware are not inseparable — quite like limbs from the Persona's body, **Apple**'s own code separates them. Just like **Apple**, **Rad**, **Ahmed** and **Antony** were instrumental in separating me from my hard-earned, 10-interview offer at **Apple** once they knew about my disabilities in detail.

What This Proves (and What It Doesn't)

This is the important part to be precise about — what this code shows.

The demo does	**The demo does not do**
Run **Apple**'s CoreIK IK solver	Reproduce the full Persona pipeline
Call `_CoreIKSolverFusionEnableReplayMode`	Track a real human body
Produce a valid 91-joint skeletal pose	Produce a photorealistic avatar
Use **Apple**'s unmodified production binary	Use any camera or sensor input
Render a stick-figure skeleton in 3D	Render face, skin, hair, or clothing
Solve in well under a frame (timing logged)	Use neural networks or computer vision
Prove the solver component runs without hardware	**Claim the full role required no hardware access ever**

The full Persona system on a real **Vision Pro** involves cameras, neural hand models, sensor fusion, hardware-locked daemons, and a rendering stack. None of that is here — and critically, none of that was the job. **Apple**'s Persona team is world-wide, with different teams siloed in different locations.

The role I was hired for was **3D Animation Engineer**, specifically focused on the IK solver layer: elbow disambiguation, solver stability, convergence tuning for the Constellation pipeline and future animation features. That is CoreIK. That is what this demo runs. The camera pipeline, the hardware engineering and neural rendering belong to different teams, different roles, different locations. Despite **Apple**'s concerted attempts to make my role sound bigger than it was, I was not hired to work on the full Persona stack. I was hired to work on the component this demo exercises.

To be clear about what is and isn't being argued here.

Nobody is saying hardware access is never needed. End-to-end integration testing, sensor calibration, latency tuning against real camera input, validating the full pipeline on a physical device — all of that legitimately requires hardware. That is not in dispute. In fact, **Apple** has been known to dogfood (assuming you're one of the lucky 5 engineers to nab a prototype, and don't have any disabilities that inconvenience the company).

What is in dispute is whether *writing and iterating on the IK solver itself* requires constant physical headset access. It doesn't. The solver is a C++ numerical computation engine. You feed it numbers, it outputs a skeleton. Synthetic inputs are not a workaround — they are the standard way to develop, test, and debug a component like this in isolation. `EnableReplayMode` exists precisely because **Apple**'s own engineers do exactly this.

Again I note this para from **Ahmed Elhasairi's** own witness statement because it inadvertently answers the question. On hardware availability (para 63):

> *"At the time that we were recruiting for the role, there were not enough headsets for each team member. I believe that we had five headsets for a team of eight engineers."*

Five headsets, eight engineers. Three people on that team, at any given moment, did not have a headset. By the logic of para 62, that the work was inseparable from physical hardware access, three members of the team could not work at all. That is obviously not what was happening. The solver work continued. It had to. The only way it could is if solver development does not, in fact, require constant headset access.

This is not a claim about state-of-the-art body tracking. The point here is narrower and more specific than the body tracking comparison: **Apple** told a tribunal that software of this kind could not be separated from physical hardware presence. That is false, demonstrably, using **Apple**'s own code. The **Meta** comparison just makes it funnier, especially when you consider their consistent slander of **Meta** in the Witness Statement:

> **On Meta (para 56):**
>
> *"If the code and algorithms that we were working on were leaked this would be very damaging for Apple. Meta (a major competitor) is working on something very similar to Persona called Meta Avatar. At the time that the role was being recruited for Meta's technology was cartoonish looking avatars that users could manually adjust..."*

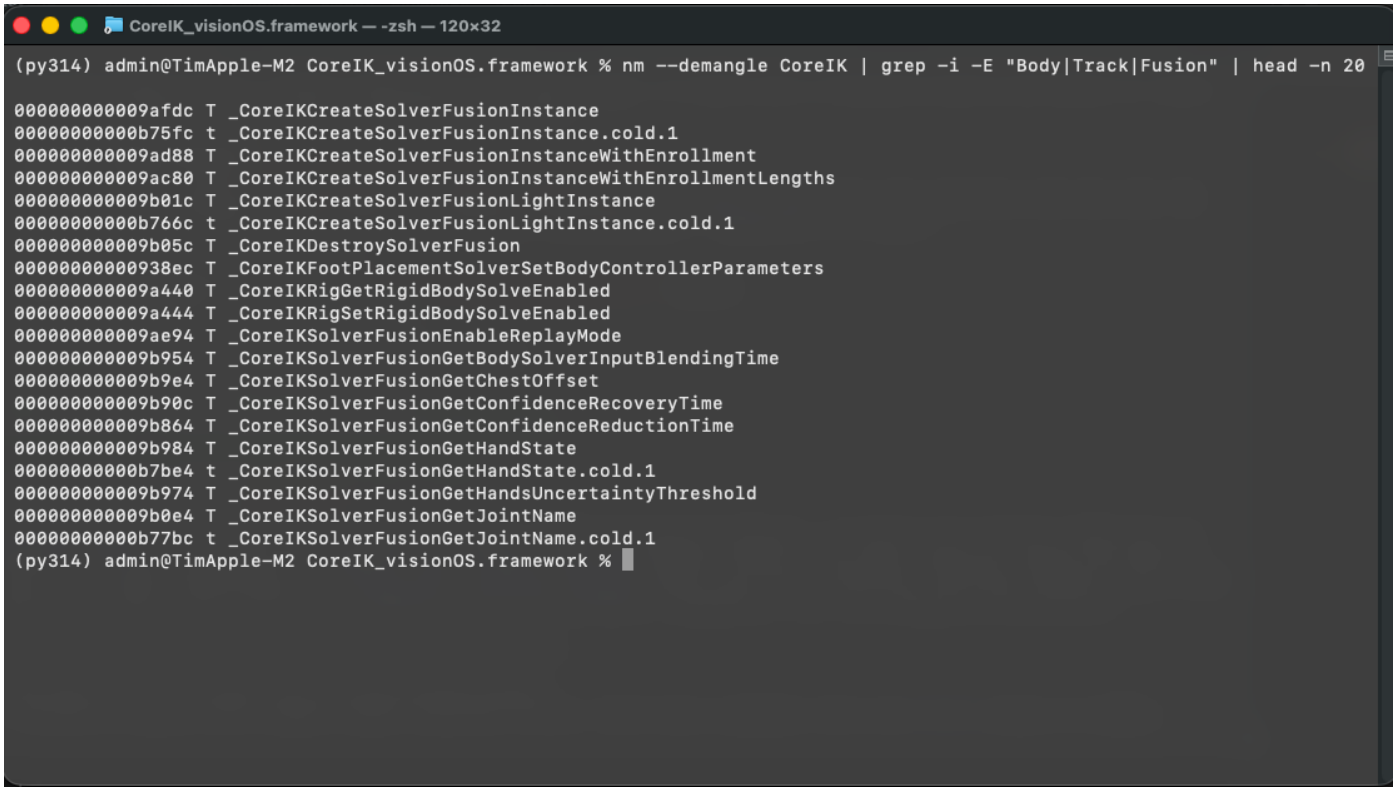
The *"cartoonish"* **Meta** avatars shipped — on a $499 headset, with working inside-out body tracking — nearly three years before this statement was signed. **Apple**'s photorealistic ones still can't keep an arm on. I don't think they want detached arms either, **Ahmed** :)

Two details worth noting for anyone digging deeper:

- **The head position is fixed** in this demo. It is not a slider, and not an oversight. In the active rig, the head joint is configured as the world-space parent anchor (`is_parent_constraint: true`) and the root is allowed to translate freely (`translate_root: true`). Move the head and the entire skeleton follows as a rigid body; that is the rig's intended egocentric VR behaviour, not a bug. (So yes: moving the head *translates* the whole skeleton but doesn't *reshape* it. That's the rigid-parent rig doing exactly what it should, not an input the solver quietly ignores.) Wiring the head to a slider is future work.
- **The production rigs suggest richer inputs exist upstream.** The full-body Orion rig bundled with the solver has 28 configured tasks — far beyond 3-point. The inputs to those tasks are wired at runtime by the C++ Fusion layer, not

stored in the rig files. What the production pipeline feeds those inputs is an open question; answering it would require dynamic analysis of the live Persona stack on a real device.

How This Project Came to Be: Finding the Functions

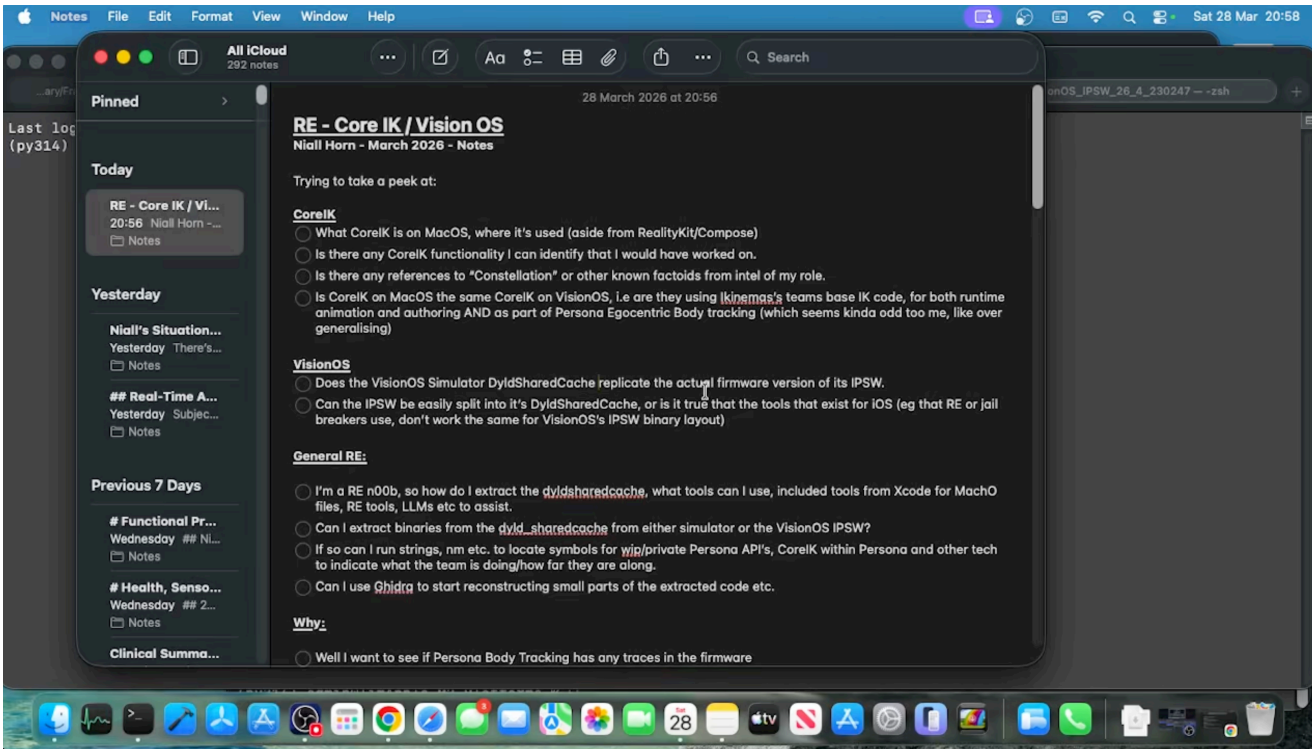


During the final days of finishing up my article on **Apple**'s conduct on its 50th Anniversary (**1st April 2026**), I had just recently spent the evening having a quick grok of both the internal and simulator image-based Private Frameworks that **Apple** ships. While it's common knowledge these private frameworks exist, and they are typically scanned for (and banned) in shipped apps, that was not my purpose. Instead I wanted to see if any of the code I would have been working on at **Apple** 2 years ago existed, and if so, how far along it was.

Full disclosure on how this got built. I was hired as a **C++ engineer** — not a Swift one, and definitely not a reverse-engineer. I'd never used Swift for any extended stretch, and I'm an RE n00b by any honest measure. With about **two weeks** before the tribunal hearing, I leaned on **Claude** (**Anthropic**'s LLM) to help me close those gaps: demangling and mapping symbols, working out the struct layouts, and wrestling Swift's `@convention(c)` function-pointer gymnastics into something that actually ran. None of that changes whether the result is *real* — every step below is reproducible by anyone with the free download and `nm` / `otool`, and I'd rather tell you AI helped me get there fast than pretend I memorised the Itanium C++ ABI in a fortnight. The credibility is in the reproducibility, not in me having done it unaided.

And while I'm owning the n00b part: this came together not just two weeks before the tribunal, but barely two weeks after I'd been at the **High Court** — running on fumes, **Vyvanse**, and spite. So my deeper read of *why* `CoreIK` does what it does — the call paths, the internal naming, the solver's guts — may well contain mistakes. They're mine, and I did my best to analyse honestly. The one thing that *isn't* up for debate is the part you can see with your own eyes: **Apple**'s binary loads, the solver runs, and a 91-joint skeleton appears on a **Mac** with no headset attached. That's not interpretation — that's the demo.

So if you spot an error in the analysis, I'd genuinely love to hear it. Open an issue, send a PR, correct my demangling. Heck — if you *work at* **Apple** and want to submit a pull request, hit me up. The offer stands.



*My notes from that evening, on my trusty TimAppleM2 — before I knew what any of it meant. A self admitted total RE n00b, out of the game for years thanks to **Apple**. I had no idea how much the `CoreIK` binary was about to give up. Please respect my amazing OPSEC game using **Apple** Notes!*

What I found was that the entire unstripped CoreIK binary, a direct mirror of the binary running on the **Vision Pro** hardware — was accessible from the Simulator's private framework. Not only that, but because the binaries were unstripped of their symbols, I could easily examine both the binary layout and the contents. This includes the ability to demangle the C++ functions, classes and globals throughout all the scoped namespaces, using **Apple**'s own shipped tools like `nm`, `dyldinfo` and other UNIX-derived tools that we use on a daily basis to examine what the Mach-O binary depends on and how it is utilised internally.

No decompilation (disassembly, yes; reconstructing source, no). No jailbreak. No leaked internal docs. No decrypted IPSW firmware either: purely the freely-distributed visionOS Simulator runtime DMG that **Apple** hands to every developer. Here is the actual process.

Step 1 — Find the binary

The visionOS Simulator runtime is a free download from Xcode → Settings → Platforms. Once installed, CoreSimulator mounts it as a disk volume at:

```
/Library/Developer/CoreSimulator/Volumes/xrOS_<BUILD>/
```

Inside that volume, CoreIK lives at:

```
.../RuntimeRoot/System/Library/PrivateFrameworks/CoreIK.framework/CoreIK
```

That's the production binary. Unstripped. Sitting on your **Mac**. Distributed by **Apple**.

Step 2 — Dump the symbol table

```
nm --demangle \
  /Library/Developer/CoreSimulator/Volumes/xrOS_23N301/\
Library/Developer/CoreSimulator/Profiles/Runtimes/\
xrOS\ 26.2.simruntime/Contents/Resources/RuntimeRoot/\
System/Library/PrivateFrameworks/CoreIK.framework/CoreIK \
  | grep -i "CoreIKSolverFusion" \
  | grep -v "ikinema"
```

Output (abbreviated):

```
000000000009ac80 T _CoreIKCreateSolverFusionInstanceWithEnrollmentLengths
000000000009ae94 T _CoreIKSolverFusionEnableReplayMode
000000000009ae14 T _CoreIKSolverFusionGetOutputPoseSizeV2
000000000009ae1c T _CoreIKSolverFusionGetJointNameV2
000000000009b144 T _CoreIKSolverFusionSetVioPoseV2
000000000009b2ac T _CoreIKSolverFusionSetSourcePoseV2
000000000009b3fc T _CoreIKSolverFusionSolveSources
000000000009b05c T _CoreIKDestroySolverFusion
```

`_CoreIKSolverFusionEnableReplayMode`. **Apple** named it. It's right there.

Step 3 — Why `dlsym()` doesn't work

A normal private framework you can at least call with `dlopen` + `dlsym`. Not here. CoreIK's symbols are in the `SYMTAB` (which is why `nm` sees them) but they are **absent from the `LC_DYLD_EXPORTS_TRIE`**. That's the section of the Mach-O that dyld actually uses at runtime for symbol lookup. `dlsym()` queries the exports trie, finds nothing, and returns `NULL`.

You can verify this yourself:

```
dyld_info -exports /path/to/CoreIK | grep CoreIKSolver
# → no output
```

Apple deliberately did not export these symbols for third-party use. They are internal function addresses that happen to be named in the symbol table for their own debugging tools. **Apple**'s own code that calls CoreIK was statically linked against it at build time — the addresses are baked in, no runtime lookup needed, no exports trie needed. `dlsym()` is a runtime mechanism and these symbols were never intended for it.

Step 4 — Offsets + ASLR slide

Because the addresses `nm` prints are static (pre-ASLR) byte offsets from the start of the binary, the approach is:

1. `dlopen()` the framework — this gets it mapped into our process
2. Find which dyld image index it landed at
3. Call `_dyld_get_image_vmaddr_slide(index)` — this is a public **Apple** API that returns the ASLR randomisation offset for that image
4. `runtime_address = nm_offset + aslr_slide`
5. Cast that address to a typed C function pointer and call it

In Swift:

```
// After dlopen() loads the framework, find its slide
var slide: Int = 0
for i in 0..<_dyld_image_count() {
    if let name = _dyld_get_image_name(i),
```

```
        String(cString: name).contains("CoreIK") {
         slide = _dyld_get_image_vmaddr_slide(i)
         break
    }
}

// Cast offset + slide to a callable function pointer
typealias EnableReplayFn = @convention(c) (UnsafeMutableRawPointer) -> Void
let fn = unsafeBitCast(
    UnsafeRawPointer(bitPattern: 0x0009ae94 + slide),
    to: EnableReplayFn.self
)
fn(solverInstance)
```

`deploy/derive_offsets.sh` automates the `nm` grep for each build and prints the values ready to paste. The offsets shift with each new simulator release; the symbol names have been stable across every version checked.

Step 5 — Working out what to feed it

The functions have no public headers. Working out the expected argument types was a combination of reading the rig files, reading the function names carefully, and a reasonable amount of trial and error.

The `.ikn` rig files were the first key source — they're plain JSON. Open any of them in a text editor and you get the full skeleton definition: 42 bones with rest-pose transforms, per-joint IK task goals with position and rotation weights, and crucially, `sources_` arrays left **empty** with a note that they are wired at C++ runtime. That told us the source inputs are runtime pose structs passed via the C API, not baked into the rig.

The function names did a lot of the work from there. The call sequence reconstructs itself from the names:

First, the supporting types I reconstructed (plain C, matching the demangled C++ views):

```
typedef struct {
    simd_float4 rotation;     // quaternion x,y,z,w  (identity = {0,0,0,1})
    simd_float4 translation;  // position  x,y,z,pad (pad ~1.0)
} FIKTransform;

typedef enum { BodyGroup_Head = 1, BodyGroup_LeftHand = 2, BodyGroup_RightHand = 3 } FIKBodyGroup; //
1-indexed
```

And the C entry points, with the prototypes I reconstructed:

Function	Prototype (reconstructed)	Notes
`_CoreIKCreateSolverFusionInstanceWithEnrollmentLengths`	`void* fn(int bodySourceType, float* boneLengths, uint8_t* enrollmentStates, size_t count)`	`bodySourceType=0` → internal type 1 (default); NULL/0 enrolment = defaults. (Yes, that's the real, full name. Someone on the naming committee earned their salary.)
`_CoreIKSolverFusionEnableReplayMode`	`void fn(void* solver)`	The *"no hardware, please"* switch. A direct branch to `CoreIKSolverFusion::enableReplayMode()`.
`_CoreIKSolverFusionSetVioPoseV2`	`void fn(void* solver, simd_float4 rotation, simd_float4 translation)`	Head pose. `VIO` = Visual-Inertial Odometry. The transform is passed by value in SIMD registers (32 bytes).
`_CoreIKSolverFusionSetSourcePoseV2`	`void fn(void* solver, int bodyGroup, FIKTransform* poses, float* confidences, int count, float scalar_conf, double ts)`	One call per tracked source (one per wrist). `bodyGroup` is 1-indexed (see `FIKBodyGroup`).
`_CoreIKSolverFusionSolveSources`	`double fn(void* solver, int format, FIKTransform* outPoses, float* outConf, int count)`	**The one to use.** Actually writes the output buffer. `format` is an `OutputFormat` enum (1-indexed; `1` = local-space output), *not* a *"solver mode"*. Returns a quality metric.
`_CoreIKSolverFusionSolve`	`double fn(void* solver, int format, FIKTransform* outPoses, float* outConf, int count, double timestamp)`	**Does NOT write output:** it routes to `solveWithTime` internally (sentinel test: 0 of 2912 bytes written). An easy trap; use `SolveSources`.
`_CoreIKSolverFusionGetOutputPoseSizeV2`	`int fn(void)`	Always returns 91. No solver arg.
`_CoreIKSolverFusionGetJointNameV2`	`const char* fn(int index)`	Static joint-name table, index 0–90. No solver arg.
`_CoreIKDestroySolverFusion`	`void fn(void* solver)`	Cleanup.

And here is the part that makes the whole *"no decompilation, just read the symbols"* point in two lines. The flat C entry points are thin wrappers over the real C++ methods, and the unstripped symbol table shows *both* (offsets here are from build `23O470`; they shift per build, the mapping doesn't):

```
000000000009a60c T _CoreIKSolverFusionSolveSources
00000000000a8174 t CoreIKSolverFusion::solveSources(FIK::Fusion::OutputFormat,
FIK::IKArrayView<FIK::Transform>, FIK::IKArrayView<float>)
```

The exported `T` C stub maps straight to the local `t` demangled C++ implementation. Note how the arguments line up: the C `int format` is the C++ `FIK::Fusion::OutputFormat`, and the C `outPoses` / `outConf` buffers are the C++ `IKArrayView<Transform>` / `IKArrayView<float>`. **Apple** named both; `nm` prints both.

The naming is precise enough that you can reconstruct the intended call sequence from the symbol table alone, before writing a single line of code against it.

Working out the *types* sounds like the hard part, but most of it was free, because the unstripped symbol table is C++, and the Itanium ABI encodes the full signature *into the mangled name*. So the derivation went cheapest-to-most-expensive, and the disassembler was a fallback, not the main tool:

1. **Demangled names gave the types directly.** `nm --demangle` on a C++ symbol like `_ZN3FIK16SerialisationJson11SaveMoCapRig...` decodes to `SaveMoCapRig(FIK::MoCapRig const&, std::string)`. The type identities *and* argument order fall straight out, no disassembly needed. Same for `setSourceData(BodyGroup, IKArrayView<Transform const>, IKArrayView<float const>, float, double)`. The simulator binary is unstripped, so **Apple**'s own symbol table just hands you the signatures.
2. **The flat C wrappers let us sidestep C++ layout entirely. Apple** ships `extern "C"` shims (`_CoreIKCreateSolverFusionInstanceWithEnrollmentLengths`, `_CoreIKSolverFusionSetSourcePoseV2`, …) that

take plain pointers/ints/floats. For most calls you never need the internal class layout, just the wrapper's argument list, readable from the demangled name plus the standard AArch64 convention (x0, x1… for ints/pointers, q0/q1 for SIMD).

3. **Accessor functions handed over the data tables.** The 91 joint names and the parent hierarchy didn't come from disassembly, they came from *calling* `GetJointNameV2(i)` and `GetOutputPoseSizeV2()` (literally `mov w0, #0x5b; ret`, i.e. 91). **Apple**'s own static tables enumerate the skeleton for you.
4. **The JSON path was the escape hatch.** Whatever `SaveMoCapRig` / `ExportRigAsJSONToFilePath` writes to disk *is* the struct contents, so for anything too painful to reverse you could dump it to JSON and read the layout off the output instead of decoding bytes.
5. **Disassembly was reserved for the two things the above couldn't give:**
    - `FIKTransform`'s byte layout: 32 bytes = `2 × simd_float4`, established by watching `SetVioPose` store `q0`, `q1` to the stack. Demangling told us the *type* was `Transform`; only the disassembly proved it's 32 bytes passed by value in SIMD registers, not the 64-byte 4×4 matrix an early guess had assumed.
    - The 1-indexing of `format`, from `convertFormat` doing `sub w0, w0, #1`.
6. **And the genuinely empirical bit was black-box, not disassembly at all:** that `SolveSources` writes the output buffer while `Solve` does not was found by writing a sentinel byte-pattern and counting what changed (0 of 2912 bytes for `Solve`).

So, honestly: the type *names and signatures* were free (demangling), the *data tables* were free (accessors), the JSON path was a fallback definition, and **disassembly was reserved for byte-level layout and ABI details only.** The whole thing was callable because we read the types off the C++ side these C shims delegate to. **Apple** shipped both halves, unstripped, in a free download.

Starting with the CLI and the LC_SUB_CLIENT trick

Before the visionOS Simulator app prototype existed (less than two weeks before the hearing started), the first proof-of-concept was a plain C command-line tool.

Getting that to work required one additional step: CoreIK has an `LC_SUB_CLIENT` load command in its Mach-O header, which tells dyld to only allow known, named clients to link against it. Any process not listed gets rejected at load time.

`LC_SUB_CLIENT` gives a named client access to a dylib, for example if you run:
`dyld_info -load_commands CoreIK | grep -iA 20 "LC_SUB_CLIENT"` you will see the following output, with the following fields `cmd, cmdsize, client`:

```
Load command #23
              cmd: LC_SUB_CLIENT
          cmdsize: 0x18
           client: "BodyPoseKit"
Load command #24
              cmd: LC_SUB_CLIENT
          cmdsize: 0x18
           client: "Enrollment"
Load command #25
              cmd: LC_SUB_CLIENT
          cmdsize: 0x28
           client: "AltruisticBodyPoseKit"
```

The fix was straightforward — a four-byte patch to the framework binary (a copy, never the original). The `LC_SUB_CLIENT` command type is `0x14`. Overwriting those four bytes with a value dyld doesn't recognise as a required command (`0xFF`) causes dyld to skip it silently. Unknown non-required load commands are ignored by design. I have no idea why this works, and I'm guessing **Apple** are aware of this…

One patch, the binary loads, the solver runs. No decompilation, no ROP, no kernel exploits. Just reading the Mach-O spec and writing four bytes.

That CLI is not in this repo, and neither is the Python script I wrote to do the patch as it modifies a copy of **Apple**'s binary which adds unnecessary legal surface area. Nonetheless it was the first time the solver ran outside of **Apple**'s ecosystem, and it's what confirmed the approach was sound before building the GUI demo.

The visionOS Simulator app sidesteps `LC_SUB_CLIENT` entirely — the simulator's dyld enforcement is relaxed compared to a real device, so `dlopen` just works. No patching needed. The only codesigning is an ad-hoc sign on the `/tmp` copy so dyld accepts it, which requires no developer account or **Apple** certificate.

A note on the Xcode project name

The Xcode project is still called `HelloVisionOS` — because that's what I named it when I started this thinking it was a quick experiment that probably wouldn't work. Turns out it did. Renaming it felt like bad luck at that point, and by the time it was running in front of a tribunal judge it definitely felt like bad luck to touch it. So `HelloVisionOS` it stays.

Granted, in light of **Apple**'s conduct I've had to say `GoodbyeVisionOS`, so maybe we can imagine the project being called that instead :)

Is This Stolen Code, or IP?

No.

The **Apple** software in this demo comes from the **visionOS Simulator runtime** — a free download **Apple** distributes to every **Mac** developer via their own software update servers (the same CDN they use for iOS updates).

You get it through Xcode → Settings → Platforms. The framework binary inside it is unstripped and fully readable with standard tools (`nm`, `lldb`, `dlopen`) that **Apple** ships with Xcode.

The CoreIK binary itself is not included. Neither are **Apple**'s rig files. `deploy/setup.sh` copies the binary, along with the three `.ikn` rig files it needs (the head + two-wrist rig, the rest pose, and the enrolment rig), from your own locally installed simulator runtime into `/tmp/CoreIK.framework/` at setup time. Nothing of **Apple**'s binary or rig data is committed to this repo; it all comes from the copy already sitting on your machine, and never leaves it.

A note on provenance: Simulator DMG, not decrypted firmware. Everything here was done using **Apple**'s freely-distributed visionOS *Simulator* runtime, an unencrypted disk image **Apple** serves to any developer through Xcode. **None of this used a decrypted IPSW, any device-firmware extraction, or any circumvention of Apple's encryption.** That distinction matters: the Simulator binary ships in the clear, from **Apple**, for exactly the kind of development this demonstrates. (I did separately analyse decrypted IPSWs back in early April 2026, and that turned up some genuinely interesting things, but none of it was needed here, and that's a story for a future, more dev-focused write-up on my tech blog.)

Legal basis for binary analysis: UK CDPA 1988 s50B (study of a computer program) and s50BA (decompilation for interoperability).

One last thing about this binary supposedly being protected by **THE LAB**: the proof we have full access to the *"super-secret"* binary is that we can even read the typos in the code. Namely: `kBoneLeftShoulerBlade` — missing the *'h'* in shoulder. Present in both the visionOS 26.2 simulator binary and the 26.4 device firmware. Survived every refactor, every code review, every OS release, and two corporate owners. A human fingerprint in code that is simultaneously classified Area-51-windowless-room and available on **Apple**'s public CDN. Real engineers wrote this under real deadline pressure. That's what **THE LAB** was protecting.

Honestly: if I wanted a production egocentric body tracking system for my own avatar pipeline, I would not be doing any of this. I would write my own solver, drive it with ML pose estimation from a camera (a fun project in its own right), explore numerical approaches to make up the gaps, and avoid ***"the impression of the arm not being connected to the rest of the body"*** as per **Apple**'s own witness statement of **Ahmed Elhasairi** about the state of Persona I was hired to fix back in **June**

2024. I prefer avatars that have the impression of fully connected arms — unless the user wanted, by design, to detach limbs to accurately represent their disability, rather than by numerical error.

The Swift App: A Thin Shell Around One Interesting File

I'll be honest: the app is not the point, and it shows. It's a deliberately minimal SwiftUI + RealityKit shell, thrown together to get **Apple**'s solver on screen in front of a judge — built fast, under deadline, by someone who will happily tell you he does not enjoy writing Swift. If you came for elegant Swift, close the tab. If you came to watch **Apple**'s own binary get called from outside **Apple**'s pipeline, there is really only one file that matters: `CoreIKBridge.swift`.

Everything else is glue:

File	What it does
`CoreIKBridge.swift`	**The whole point.** `dlopen`s the framework, resolves each function by its `nm` offset + ASLR slide, casts the result to a typed C function pointer, and calls **Apple**'s solver directly. Reads `/tmp/CoreIK.framework/BUILD` to pick the right offset table per visionOS build. This is the file to read.
`SkeletonRenderer.swift`	Procedural spheres (joints) and boxes (bones), emissive PBR so they glow against passthrough. Per frame it only nudges transforms — it never rebuilds geometry.
`ImmersiveView.swift`	The `RealityView` immersive space. Holds the skeleton plus a head-anchored control panel, and runs a ~60 fps loop that only re-solves when a slider actually moves (`needsBurst`).
`ControlPanel.swift`	The head-locked SwiftUI panel — the wrist sliders and the live CoreIK invocation counter.
`CoreIKEventLog.swift`	Logs every solver call to `/tmp/coreik_invocations.log` so `logs.sh` can stream it on the host.
`AppModel.swift`, `ContentView.swift`, `HelloVisionOSApp.swift`, `ToggleImmersiveSpaceButton.swift`	Standard SwiftUI plumbing — shared state, the launch window, the app entry point, and the button that opens/closes the immersive space.

No clever architecture, no dependency injection, no tests. It is a demonstration harness, not a product. The interesting code lives in `CoreIKBridge.swift`; the rest of the files exist to give those calls two sliders and somewhere to draw the result. That was always the deal — the app is the frame, **Apple**'s binary is the painting.

Running It Yourself

What you need

- macOS 15 (Sequoia) or later, **Apple** Silicon
- Xcode 26.x
- At least one visionOS Simulator runtime installed (Xcode → Settings → Platforms → visionOS, ~6.7 GB)

You do not need a **Vision Pro**. You do not need a paid **Apple** developer account ($99/year). A free **Apple** ID is sufficient — that is all Xcode requires to download the visionOS Simulator runtime. **Apple** distributes it via the same CDN they use for software updates. The CoreIK binary is sitting on the **Mac** of anyone who has ever done visionOS development, whether they paid **Apple** anything or not.

Scripts

Everything lives in `deploy/`. No IDE required, no sourcing required — just run the scripts directly.

Script	What it does
`./deploy/setup.sh`	Copies CoreIK.framework to `/tmp`, codesigns it. **Run after every reboot.**
`./deploy/build.sh`	Compiles the app for the visionOS Simulator (~60s first build)
`./deploy/launch.sh`	Boots the **Vision Pro** sim, clears stale state, installs and launches the app
`./deploy/logs.sh`	Streams the live CoreIK function call log from `/tmp/coreik_invocations.log`
`./deploy/verify.sh`	End-to-end check on the staged build — setup → build → launch, confirms the solver fires. Set `COREIK_VERSION` inline to target a version.
`./deploy/test_all_builds.sh`	Runs `verify.sh` across every supported build (26.2 / 26.4.1 / 26.5) and tallies pass/fail
`./deploy/derive_offsets.sh`	Derives symbol offsets for a new visionOS build via `nm`
`deploy/coreik_offsets/`	Pre-generated offset records for verified builds (26.2, 26.4.1, 26.5)

Setup and run

```
git clone <this-repo> Persona_CoreIKDemo
cd Persona_CoreIKDemo
./deploy/setup.sh     # run once per boot
./deploy/build.sh     # compile
./deploy/launch.sh    # boot sim + install + launch
./deploy/logs.sh      # second terminal — stream live log
```

Tap **Enter Immersive Space** in the Simulator. A 91-joint stick-figure skeleton appears. Use the wrist sliders to move it. Watch `logs.sh` stream **Apple**'s private function names in real time, every call counted, timestamped, logged to `/tmp/coreik_invocations.log`.

> **Two things you'll notice the first time:**
>
> - `launch.sh` **installs** the app into the booted simulator before launching it, so the first run takes a moment to appear in the Simulator window. That's expected.
> - The Simulator may pop a **"RealityLauncher quit unexpectedly"** dialog. That's a visionOS *Simulator* helper process (it hosts immersive scenes), **not this app**. Dismiss it; the demo keeps running and solving. It's a known sim quirk, nothing in this repo.

Re-run `setup.sh` after every reboot — `/tmp` is cleared on restart.

Where things live on disk

*Skip this if you're already comfortable with **Apple**'s developer toolchain.*

- **Xcode Command Line Tools binaries** (nm, lldb, etc.):

```
/Library/Developer/CommandLineTools/usr/bin/
```

- **Mounted simulator volumes** — CoreSimulator mounts each runtime here on demand:

```
/Library/Developer/CoreSimulator/Volumes/xrOS_<BUILD>/
```

- **CoreIK binary inside a mounted runtime:**

```
/Library/Developer/CoreSimulator/Volumes/xrOS_<BUILD>/
  Library/Developer/CoreSimulator/Profiles/Runtimes/
  xrOS <VERSION>.simruntime/Contents/Resources/RuntimeRoot/
  System/Library/PrivateFrameworks/CoreIK.framework/CoreIK
```

Yes, that path is genuinely that deep. I have to give a huge shout out to whoever decided putting a space in `xrOS <VERSION>.simruntime` — good job, maybe try underscores or dashes next time.

That space isn't just an aesthetic problem. `ikinema::getResourcePath` inside CoreIK calls `CFURLCopyPath` to resolve the path to its own rig files. `CFURLCopyPath` returns a URL-encoded string — so the space in `xrOS 26.2.simruntime` becomes `%20`, and the resulting path gets handed to `std::ifstream::open()`. The file at `xrOS%2026.2.simruntime/...` does not exist. The solver silently returns nil. No error, no crash, just a skeleton that never appears.

The fix: `OpenInterpose.m` replaces `CFURLCopyPath` with `CFURLCopyFileSystemPath` process-wide at load time, using `__DATA,__interpose` — a standard dyld mechanism **Apple** documents for exactly this kind of runtime substitution. `CFURLCopyFileSystemPath` returns a plain filesystem path with the space intact. The binary on disk is never touched. This is the same class of fix as the kind of low-level framework debugging the role was hired to do.

`RuntimeRoot/` is where it becomes a faithful mirror of a real visionOS device filesystem — everything under it maps 1:1 to `/` on an actual **Vision Pro**.

Working copy created by `setup.sh` — space-free path in `/tmp` as a belt-and-braces fix on top of the interpose:

```
/tmp/CoreIK.framework/CoreIK        ← the binary
/tmp/CoreIK.framework/BUILD         ← build tag (e.g. 23O470)
/tmp/CoreIK.framework/*.ikn         ← rig files
```

`/tmp` is cleared on every reboot — re-run `setup.sh` after restart.

Live log written by the running app:

```
/tmp/coreik_invocations.log
```

The simulator's `/tmp` maps directly to the host's `/private/tmp` — no container boundary, so `tail -f /tmp/coreik_invocations.log` on the host streams what the simulator app writes in real time.

xcrun

`xcrun` is a small wrapper **Apple** ships with Xcode Command Line Tools. It routes commands to the right toolchain without needing full paths. Everything in this repo runs from the terminal — you don't need to open Xcode IDE at all.

```
xcode-select -p   # confirm toolchain path
xcodebuild -version
```

visionOS version → build tag reference

The naming is not obvious. Inside CoreSimulator the OS is called `xrOS` (**Apple**'s internal name before the visionOS marketing rebrand). `setup.sh --list` will show what's installed on your machine.

Marketing name	Internal name	Build tag	Notes
visionOS 1.2	xrOS 1.2	`21O5565d`	Oldest on record here
visionOS 26.2	xrOS 26.2	`23N301`	Original tribunal demo target — verified ✅
visionOS 26.4.1	xrOS 26.4.1	`23O249a`	Verified April 2026 — all symbols identical ✅
visionOS 26.5	xrOS 26.5	`23O470`	Verified June 2026 — offsets identical to 26.4.1 ✅

The jump from 1.x to 26.x happened at WWDC 2025 when **Apple** aligned all OS version numbers with the calendar year (same as iOS 26, macOS Tahoe 26). There is no visionOS 2.x through 25.x.

Choosing a visionOS version — and where it's set

The version is chosen in exactly one place: `setup.sh`. Everything downstream reads it back from the `/tmp/CoreIK.framework/BUILD` tag that `setup.sh` writes — so you never pass a version to `build.sh`, `launch.sh`, or `verify.sh`. One source of truth, set once.

Three ways to tell `setup.sh` which runtime to stage (highest precedence first):

```
./deploy/setup.sh --version 26.5          # flag — explicit, one-off
COREIK_VERSION=26.5 ./deploy/setup.sh     # inline env — one-off, no export or sourcing needed
export COREIK_VERSION=26.5                # exported env — sticks for the shell session (add to
~/.zshrc)
./deploy/setup.sh                         # nothing set → the newest installed runtime wins
```

Script	Version-aware?	How it knows which build
`setup.sh`	**Yes — this is where you set it**	`--version` flag, or `COREIK_VERSION` env, else newest installed
`build.sh` · `launch.sh` · `verify.sh`	No	read the `BUILD` tag `setup.sh` staged to `/tmp`

So `verify.sh` takes no `--version` either — it checks whatever's staged (set `COREIK_VERSION` inline to target one), and `test_all_builds.sh` loops every supported build for you.

Supporting a brand-new build

When **Apple** ships a new visionOS release, function offsets shift. Stage it, then re-derive:

```
COREIK_VERSION=27.0 ./deploy/setup.sh                  # stage the new runtime
./deploy/derive_offsets.sh /tmp/CoreIK.framework/CoreIK
```

Paste the printed Swift block into `HelloVisionOS/CoreIKBridge.swift` (inside `kCoreIKOffsets`) and commit the generated `deploy/coreik_offsets/<BUILD>.txt`. Symbol names have been stable across every version checked; only the offsets move.

WWDC 2026

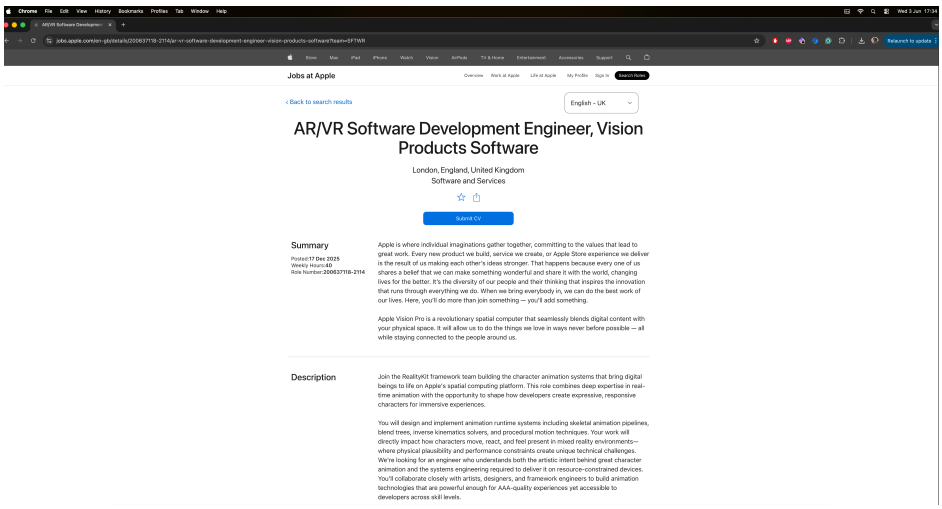
WWDC 2026 is days away. Quite ironically it reminds me that WWDC 2024 was occurring all while **Apple**'s discrimination and consequential gaslighting campaign was starting against me — where they announced **Apple Intelligence**... That was fun.

Maybe this year Siri will be able to actually follow commands and converge towards state-of-the-art NLP from 10 years ago!

More intrinsically, maybe visionOS 27 finally ships full Persona body tracking. Maybe you'll get to experience first-hand the jittery arms, the shoulder instability, and the limb detachment that **Ahmed** described under oath as requiring a secure windowless Lab (ahem **THE LAB**) to observe. If so, you're welcome. If not, well... I was hired to fix that. **Apple** had other ideas.

You Can Join This Awesome Team - Right Now

If you'd like to work on tech that **Meta** shipped 3 years ago, and you don't mind a bit of disability discrimination against neurodivergent individuals here and there, then good news - **Apple** is **STILL** hiring for the role on their team in London that has been online since **December 2025** (and was a repost of a prior role dating back to 2024) - [Apple London Animation AR/VR Engineer]



No, I'm not joking. Their reposted job advert is still online for the same team. ***[Accessed 03-06-2026]***

Either way, the solver runs. It ran this April in an AirBnB bedroom, the night before a tribunal. It runs now. It'll run on visionOS 27 too, same binary, same function names, same `EnableReplayMode`. Same typo.

Project Sunlight: Coming Soon

This is a technical artefact I built to prove the lies **Apple** was willing to put on the record — while already being sued for contempt of court. However, while it's nice to finally share this code and link it back to the ridiculous and contemptuous (as alleged) statements made to the UK Justice System by **Apple** and its witnesses, the technical side is (sadly) just one small part of the story, and this repository is one small piece of a much larger picture.

Both I and **Apple** know that.

I'm looking forward to sharing more about **Apple**'s promised apology that never materialised (in public), and **Apple**'s expressed *'regret'* behind closed doors once they believed they had flipped the table of litigation rules on me. Sadly for **Apple**, I'm not someone willing to sign away my rights to be human, with disabilities I and many others like me never chose, to give into a ruse that ultimately failed the moment I got back home after **Apple** attempted to force me into mediation behind closed doors, just months after they were threatening to bankrupt me for standing up.

If **Apple** or their overpriced legal team are reading this, and trying to figure out their next move, I say this:

Remember what I told you when you witnessed the consequences of what your 1990s litigation playbook has done to my life.

Look up towards the sky. The clouds are parting.

Sunlight is coming.

---

*Apple's `CoreIK.framework` binary and `.ikn` rig files remain the property of **Apple** Inc., used here under UK CDPA 1988 s50B / s50BA.*

No warranty or license for this code is offered.

References

- **Witness Statement of Ahmed Elhasairi, Engineering Manager** — Persona Animation Team, **Apple** (UK) Limited.
    - Submitted to London Central Employment Tribunal, heard publicly on 15 April 2026. *Mr N. Horn v. **Apple** (UK) Limited.* Quoted here by paragraph number; the full statement is part of the tribunal record.
- **Edge Case Existence Niall George Horn:** Apple at 50 — the case, the code, and what happened (my blog)
- **Apple Patent US20240312167A1:** Multiple Device Model Augmentation (2024) Patent Link
- **Meta Horizon Blog:** Inside-Out Body Tracking and Generative Legs — **Meta**'s overview of **Quest 3** IOBT, including the CV/IK pipeline and the separate generative-legs ML system for lower-body synthesis.
- **Khronos OpenXR Vendor Extension Spec:** `XR_FB_body_tracking` — the extension that exposed **Meta**'s body tracking joint output to third-parties (three-point tracking).
- **Khronos OpenXR Vendor Extension Spec:** `XR_META_body_tracking_fidelity` — the extension that allows apps to request full-body vs. three-point tracking on **Meta Quest 3**.
- **AvatarPoser — Jiang et al., ECCV 2022:** *Articulated Full-Body Pose Tracking from Sparse Motion Sensing* — the canonical academic approach to reconstructing a full-body pose from only head + hand (sparse) inputs. The exact egocentric, sparse-anchor IK problem CoreIK solves, established in the literature years ago.
- **EgoBody3M — ECCV 2024 (Meta):** *Egocentric Body Tracking on a VR Headset using a Diverse Dataset* — **Meta**'s real-image egocentric full-body tracking dataset + model (3M frames), running on-device from the SLAM cameras.
- **RealityKit `IKComponent` (Apple Developer Documentation):** `IKComponent` — **Apple**'s **public** full-body inverse-kinematics solver API (iOS 18 / macOS 15 / visionOS 2.0), the public face of the same **IKinema** lineage as CoreIK.
- **WWDC24 — Compose interactive 3D content in Reality Composer Pro:** Apple Developer, 10 June 2024 — **Apple**'s own session demoing RealityKit's *Full Body Inverse Kinematics* solver, in the same June my offer was withdrawn on the grounds that IK solver work could only happen in a windowless lab.
- **IKinema Orion:** Apple's 2019 acquisition of IKinema (AppleInsider) · Orion — full-body mocap on Vive Trackers, $500/yr (UploadVR) — the same IK middleware, before the windowless lab.
- **Epic Games v. Apple:** overview — in April 2025 Judge **Gonzalez Rogers** held **Apple** in civil contempt for *willfully* violating the 2021 injunction and referred it for possible *criminal* contempt, finding company executives had lied. (**Apple**'s *"history"* of misleading a court, cited above.)
- **UK CDPA 1988, s50B / s50BA:** legislation.gov.uk — the statutory basis for studying / decompiling a program for interoperability, cited as the legal basis for the binary analysis here.

IPFS Mirror

We all know **Apple** loves DMCAs almost as much as it loves spending hundreds of thousands of dollars of shareholder money on odd patent applications and defending lawsuits against disabled employees with overwhelming evidence against them. Nonetheless, given this involves **Apple**'s own code, distributed by themselves, and is highly embarrassing for them, it's possible **Apple** will take this down, and maybe someone will cave.

So as a single node in the system, pruned for incompatibility from **Apple**, just like with my article, this code is also hosted on the IPFS should my **GitLab** account magically go missing.

```
IPFS CID: bafybeib2kdbmad2jw6vbi74zkjmttkeh4srg6v4yym67sg64fkif7dbx3a
https://ipfs.edgecaseexistence.com/ipfs/bafybeib2kdbmad2jw6vbi74zkjmttkeh4srg6v4yym67sg64fkif7dbx3a
```

IPFS Public Gateway Download