08 NOVEMBER 5, 2026 3 PP

From a phone scan to a film, through OrbLang

Felipe Arce · Collin Rijock · Tim Child

Collin Rijock built OrbLang, a typed language that compiles physical sites into simulator scenes. Felipe Arce built the infrastructure around it: capture from phone scans, remote GPU rendering, a content-addressed project catalog, and the tools we present films and live simulations with.

ORB ORBLANG SIMULATION ISAAC-SIM MUJOCO DIGITAL-TWIN INFRASTRUCTURE BROAD-DAYLIGHT OPEN-SOURCE



A copy room in six formats

On October 12 the Orb's first version went up: nine commits, 1,801 lines of Python and 16 tests. At 10:22 that night we merged pull request #1 on top of it. A phone scan of our office copy room came out as MuJoCo, Isaac Sim, GLB, USDZ, STEP and Minecraft files, with a Mac app to drive it.

The Orb's README calls it "real world as code." You describe parts, robots and sites in OrbLang, an HCL-style language whose numbers carry units, so `1 m + 1 kg` fails to compile. The compiler lowers the files into a world graph, one JSON document in SI units. `orb plan` shows what will change and `orb apply` hands the graph to seams, small programs that each write one simulator's files. Collin Rijock wrote that

first version and designed the language, the world graph, plan and apply, and the first two seams. Felipe Arce built the infrastructure around them: capture, five more seams, remote GPU rendering, storage, releases and the tools we present with.

Per the repo's walkthrough, a phone scan becomes an Isaac Sim render in about 15 minutes. Our first film made through the Orb, 60 seconds through a warehouse, a factory line and a solar field install, was built in one night: a rough cut at 3:45 a.m. and the final at 6:17 a.m. on October 14. Five revisions followed within 37 hours, the last 119 seconds long. Between October 13 and 18 the Orb shipped 23 tagged releases, and 13 projects on one laptop are linked to its shared catalog.

Capture is a compiler front end

Hand-written OrbLang works for one robot, not for a warehouse. That afternoon the goal became a pipeline that squares up capture data into a typed program that holds all of it and can be reversed. The world graph was the right middle, so capture became a front end to the OrbLang compiler.

`orb capture` takes a scan, a video or photos, levels and squares the room, and detects floors, walls and surfaces as typed boxes. The first decision was the target. We chose typed elements plus meshes, written as USD and GLB, over CAD as the core. Simulators consume meshes and labels, a scan is already a mesh, and fitting CAD surfaces to a scan loses detail. STEP stays an export.

The constraint we set was that nothing is lost, so there are two layers. The `.orb` file holds the squared-up room. The raw scan sits beside it under `blobs/sha256/`, and every element's `source` names that hash, the scanned area behind it and a confidence. Numbers are rounded to millimetres, so a rescan only changes the lines that moved. Exports carry the sources in their metadata, and `orb import` turns them back into the same program. A consistency test compiles a project through six seams and checks that every element appears once in each, within 2 mm.

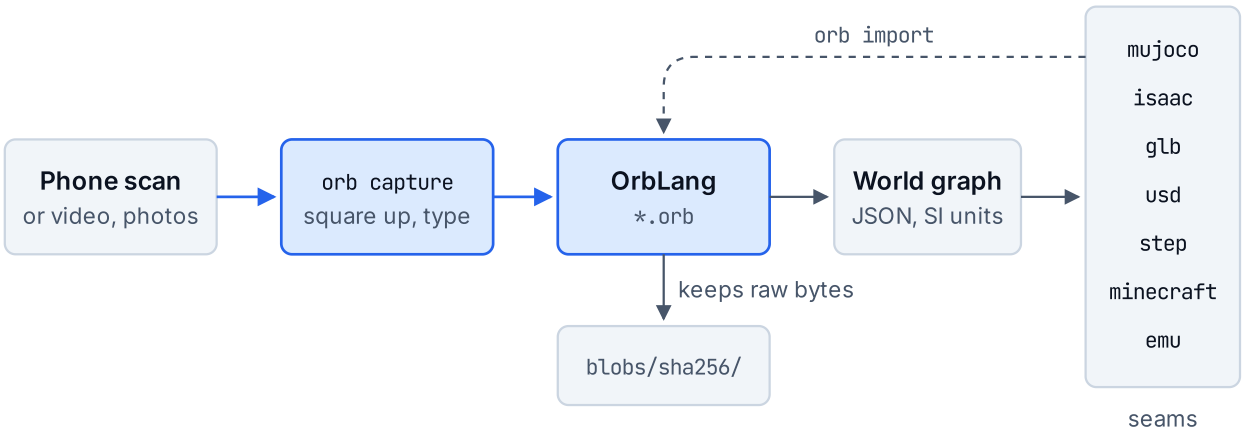


Figure 1. Capture is a front end to Collin's compiler (highlighted path). The program holds the squared-up room, the raw scan stays beside it by hash, and exports carry their sources so `orb import` rebuilds the same program.

Rendering somewhere else

Isaac Sim needs an NVIDIA GPU and our laptops are Macs, so `orb run isaac` is a remote job. It rsyncs the build to a GPU box in AWS, takes a `flock` so two people's renders queue instead of colliding, runs Isaac Sim headless and pulls the MP4 back. A render takes about 45 seconds on an idle box.

From the office, runs sometimes failed. A run opened about eight SSH connections in a minute, and something on the office network dropped bursts of new ones, so a good render could die on the final pull. We stopped debugging the network and changed the client. Every ssh and rsync in a run now rides one multiplexed connection (`ControlMaster`, kept for the next run), and only steps that are safe to repeat retry. That shipped as 0.2.4 on October 13.

Films use the same path, once OrbLang 0.3 added timelines and camera shots. A solar field of 83,334 instanced units rendered at about one second per frame while each prototype was a tree of about 30 USD prims. Flattening each prototype to one mesh brought it to 0.35 seconds.

Storage, cuts and live chapters

Projects are gigabytes, so we kept them out of git. Orb Studio Cloud is a catalog: each revision is a list of paths, SHA-256 hashes and sizes, and the bytes go straight between the laptop and our internal file store. The catalog never touches a byte, so it stays small.

Orb Cut sits on top. A cut is a shared timeline of clips and chapters. A present link plays the published render in any browser, and the presenter moves every open screen at once. A chapter can also be bound to the simulator: Isaac renders it as the presenter moves, and a pump streams the frames to the cloud over one long chunked HTTP POST. Only the newest frame waits, so a slow link drops frames instead of falling behind.

Release 0.8 made media download once per computer, cached by hash and renamed into place only after the hash matches. A big file adds HTTP Range streams only while each new one makes it faster, which took a cold pull of a published render from 18 seconds to 4.75.

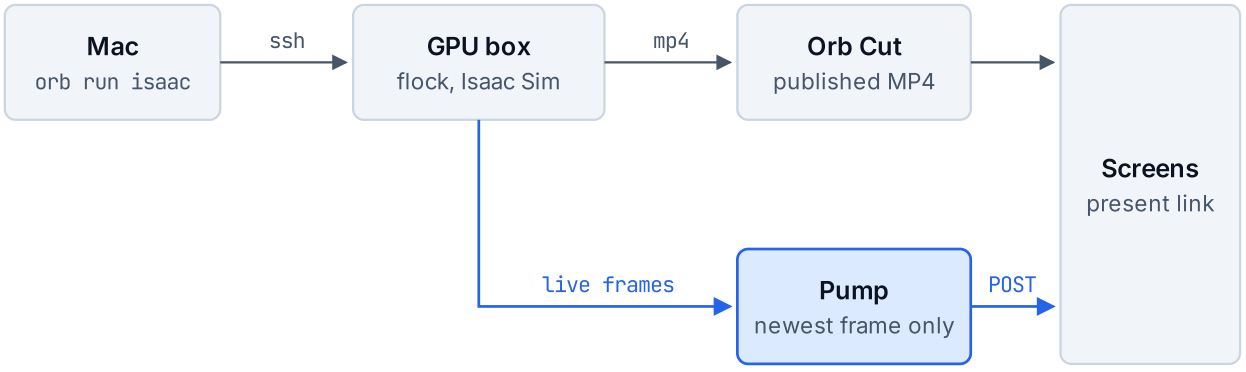


Figure 2. The render path. Films render on a shared GPU box and play from Orb Cut. A live chapter (highlighted) swaps the MP4 for frames Isaac renders while the presenter moves through it, streamed over one chunked HTTP POST.

Measure the plates

Two picture bugs taught us to measure. Slow aerial shots popped once every 250 frames, x264's default keyframe interval: between keyframes the encoder skips sub-pixel motion, then the keyframe snaps it into place. Frame-to-frame change hit 2.5 times its running median there, which ffmpeg's duplicate-frame filter missed. Every

Orb video writer now uses `keyint=infinite:no-fast-pskip=1:bframes=0`. Later the ground boiled in a film that a mean-brightness check called steady: Isaac Sim's default DLSS upscales from half resolution. We now render 4K with DLAA, scale down, and measure flicker per tile on every plate.

What it changed, and what is next

A room is now a 15-minute scan, a twin is a revision in a shared catalog, and our first film took one night. The same files feed MuJoCo for fast checks and Isaac Sim for pictures, so a site is modeled once instead of once per simulator.

Next is the piece we designed on the first day and haven't built: drift. `orb plan` should compare a fresh scan with the program and mark each element ok, drifted, missing, unexplained or unverified, where no data never counts as ok. Per-element sources are there for it. The hard part is matching elements across two scans of a place that really changed.

Orb and OrbLang are being open-sourced under Broad Daylight, Exowatt's open-source initiative, so other teams can model their own sites.