WHITEPAPER · OCTOBER 2026 · COMING SOON

Margo Engine

A WebGPU engine for the web. Core plus stage is 24,843 bytes gzipped. The catalog lists 107 packages (none published to npm yet), with one install that holds them all. Every number we publish names its source.

Scott Campbell · Margo Interactive

Sizes at commit 51d8786a; tests at 01836c27; other figures name their own commit or date

PDF

24,843 B
core + stage, min + gzip, Wasm kernel and WebGPU backend included; the build fails a change at 25,000
`entry-sizes.json` · `0e71a9024`

107
packages in the catalog, all reachable from `npm i margoengine` (none published to npm yet)
`docs/catalog.json`

6,498
Node tests in every `npm run verify`: 6,496 pass, 2 skipped, 0 fail
`results/test-count.json` · `01836c27`

153 of 161
feature rows with an equivalent Margo package; 4 partial, 4 missing
`docs/three-parity-matrix.md` · `bec8b597`

Where it stands today

- **Coming soon.** The packages are not on npm yet, and the source repository stays private while we finish the launch checklist.
- **WebGPU only, for now.** About 88% of browsers in use support WebGPU: 85.8% fully and 2.02% partly (caniuse.com/webgpu, StatCounter data for September 2026). A fuller WebGL2 compatibility package is in development as an optional download that WebGPU users never load. Today the basic WebGL2 package draws core meshes, and the React Canvas takes loadFallback and fallbackContent.
- **Few browsers verified.** Real-GPU results come from Edge and Chrome on Windows 11 (NVIDIA, Intel and AMD GPUs) and from Chromium on one Linux machine with an Intel GPU, where three suites have not finished (section 10). Firefox, Safari, macOS and phones are not verified yet.
- **No speed claims against other engines.** The same-scene comparison tool exists, but no quiet-machine run has been made. The numbers on this page are our own internal measurements, and each names its hardware.

0 Abstract

Margo Engine is a 3D engine for web applications, written in TypeScript. It has a Rust kernel compiled to WebAssembly and a WebGPU renderer. Core plus the high-level stage (`createStage`) measures 61,333 bytes minified and 24,843 bytes gzipped. The build fails at 25,000.

Everything else lives in optional, side-effect-free packages: glTF PBR, shadows, post-processing, physics, voxel worlds with multiplayer, a path tracer, exporters and React. A bundle contains only the packages it imports, and large runtimes load only when a file needs them.

This paper covers the design, what we have measured, how the engine was built, and the limits we know about. Each figure names the file it came from. Where we have no committed measurement, we say so. A companion paper covers Margo Voxel.

```
import {createStage} from 'margoengine';
const app = await createStage('#scene');
const orb = app.sphere({color: '#84d6bd', roughness: 0.25});
app.onTick(delta ⇒ {orb.rotation.z += delta;});
window.addEventListener('pagehide', () ⇒ app.dispose(), {once: true});
```

1 One install, kits and starters

`npm i margoengine` installs the whole engine. Its root is the stage, and every entry of every package is `margoengine/<package>`, one `export *` line each, so an app still bundles only what it imports. A test checks that the alias and the package bundle to the same bytes.

A kit is one import line with what an app of one kind usually needs:

Kit	For	Every name used, min + gzip
`margoengine/game`	models, animation, light and shadow, sky, physics, audio, input, effects	307,012 B
`margoengine/viz`	figures and GPU compute, points, lines, splats, terrain, labels and text	277,396 B
`margoengine/web`	a glTF model, studio light, orbit, HTML on top	155,058 B

source docs/guides/kits.md · bd8e5a69 · tests/umbrella.test.js re-measures on every run and fails on a 3% move

An app that uses part of a kit pays only for that part. `npm create margo` has five starters: web, game, viz, react and next. Each opens to a lit, animated scene with three things to try. Time from create to first frame under `vite` dev in Edge, with the packages linked (no install), one run each on the RTX A6000 PC:

Starter	What it shows	First frame
web	a studio scene with shadows and a turntable	788 ms
react	shapes in sun with soft shadows and bloom	835 ms
viz	a 300,000-point galaxy with bloom and labels	968 ms
game	a character who walks and runs, physics crates, orbs to collect	1,308 ms

source examples/package-benchmarks/results/starters-first-frame.json · tools/create-margo/test/browser-check.mjs · 8cc9e939 · the next starter is type-checked, not timed

2 Design decisions

1. **A small core that stays small.** New capability goes into optional packages through the extension hooks in section 4. CI rejects any change that pushes core + stage to 25,000 bytes gzip. Multiple views and scenes were the one approved exception: 867 bytes gzip with the kernel changes that came with them (section 4).
2. **Data-oriented transforms in Wasm.** Positions, rotations, hierarchy, world matrices, bounds and culling live in flat typed arrays in Wasm memory, computed by one Rust function per frame.
3. **WebGPU first, guaranteed limits only.** The device asks for no limit above WebGPU's guaranteed minimums, so a scene asks nothing of the GPU that a phone is allowed to refuse.
4. **Errors that name the fix.** Packages throw a `MargoError` with a code, a `TypeError` or a `RangeError`, and the message says how to get past it: `RangeError: Too many glTF nodes (100001 > 100000; raise the maxNodes option)`.
5. **No number without a source.** Every published figure has a row in a claims inventory with its evidence and status (section 9).

3 The kernel

`packages/kernel/src/lib.rs` is `no_std` Rust compiled to Wasm. Each entity is a slot in a structure of arrays: position, rotation and scale, parent, local and world bounding spheres, world matrix, joint limits, last frame's inputs and scratch space. JavaScript writes transforms into views of that memory; nothing crosses the boundary per entity.

The tick follows three rules:

- **Recompute only what changed.** An entity is recomputed when its transform, parent, bounds or hidden flag changed, along with its descendants. A static scene pays only for the input comparison, about 41 ns per entity on the test machine.
- **Publish nothing rather than something wrong.** A bad quaternion, a zero scale, a non-finite parent or a hierarchy deeper than 64 yields no matrix. World matrices are written only after every node validates.
- **No allocation.** The store starts at 1,024 slots and doubles on demand up to 2^{23} (8,388,608) outside the tick. A growth that cannot allocate throws and leaves the store as it was.

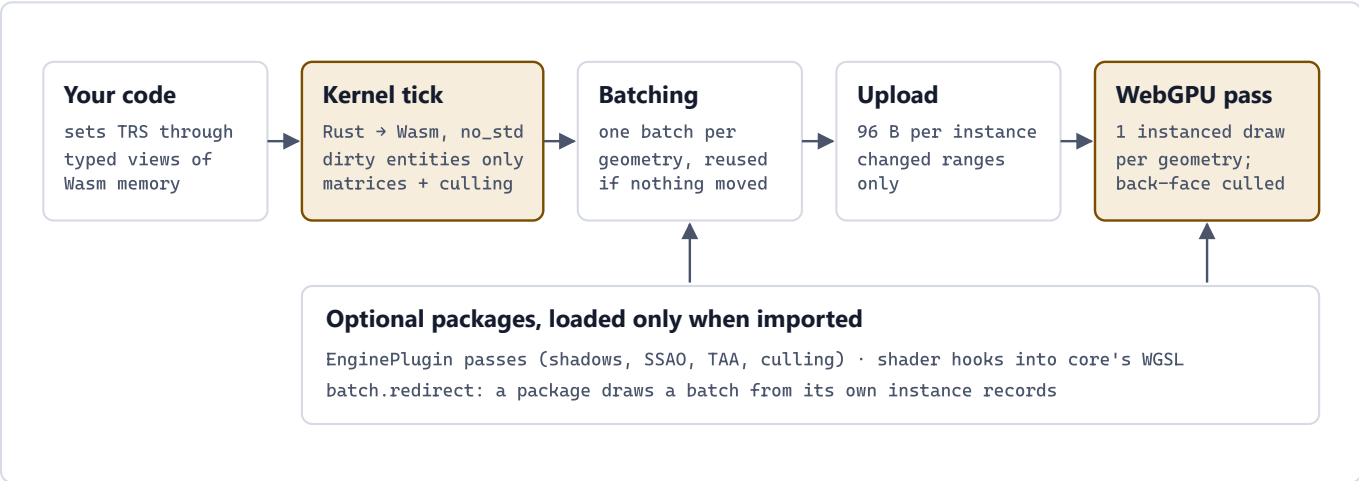
Node, the same Wasm the browser loads, a fake renderer, and an assembly grown from 1,024 slots. Quiet-gated (CPU at 8%), AC power:

Entities	Kernel memory	Tick, static	Tick, 1% moving	Full recompute
100,006	44.9 MB	3.3 ms	4.5 ms	14.4 ms
499,992	176.6 MB	20.2 ms	26.2 ms	74.9 ms
1,000,012	352.1 MB	41.5 ms	43.4 ms	138 ms

source examples/package-benchmarks/parts/results-scale-node.json · scale-node.mjs · Intel Core Ultra 7 258V

The cost is linear. A test checks that dirty propagation produces the same world matrices and visibility, bit for bit, as a fresh full recompute, through growth to 1,048,576 slots, 63-deep chains, moves, removals and slot reuse.

4 The renderer, views and hooks



One frame. The shaded steps are the two that do the heavy lifting.

Core groups visible meshes by geometry and draws each group with one instanced call, with per-instance material values. When nothing changed, it reuses last frame's instance lists and uploads only the matrices that moved. A group of single-sided, unmirrored instances draws with hardware back-face culling and no `discard`, which keeps early depth testing on. The renderer uses reversed float depth and GPU picking with integer ids, and it recovers from device loss.

Multiple views and scenes

One engine draws several views of one scene in a frame: split screen, the four-view layout, picture-in-picture, a minimap. It also draws separate scenes: a HUD with its own objects and lighting, an item preview in a menu, a portal rendered to a texture. Effects with history keep it per view, and every package is view-aware. This is in core, as an approved exception to the size freeze. It added 867 bytes gzip to core + stage, because a smaller

kernel paid for most of the 2.7 KB of new code. `npm run check:views` draws views on a real GPU and checks pixels, picks and validation errors in every merge.

source docs/multi-view-roadmap.md · 0ca52f2a · entry-sizes.json · 51d8786a

Extension points

Passes	`EnginePlugin` frame-graph passes. Shadows, post effects, TAA and GPU culling attach with `X.attach(engine, options)`.
Shader hooks	Packages patch core's WGSL at named points; shader-contract tests fail when a patch no longer matches the text it expects.
Batch redirect	A pass may set `batch.redirect = {instances, args}`; core then draws that batch from the package's records with `drawIndexedIndirect`. This lets GPU culling run on ordinary core meshes.
Raw state	Wasm memory and GPU objects are reachable only through `@margoengine/core/extension`, so the public API stays portable.

5 Size

Sizes are minified and gzipped at zlib defaults, per `exports` entry, with peers and other Margo packages external. Core + stage includes the embedded Wasm kernel and the WebGPU backend, and gives you a mesh, one directional light, orbit controls and the loader shell.

Entry	Raw	Gzip	Notes
`@margoengine/core`	53,470	22,225	Wasm kernel, WebGPU renderer, views and scenes
core + stage	61,333	24,843	the build-gated figure

source examples/package-benchmarks/results/entry-sizes.json · record 0e71a9024 · dirty: false · `npm run measure:size` on main at 38ce7fbaa, 2026-11-12

Every other entry and worker, and every lazily loaded file, is in the same record and on margoengine.com/sizes. A release test builds a page that calls `createStage` and loads a small glTF, records every request in headless Edge, and fails if the page fetches anything beyond the HTML, its chunks and the model (`examples/package-benchmarks/results/basic-scene-network.json`).

We do not call Margo the smallest engine. We publish our method and our numbers. A size comparison with other engines will be run with the same build settings from the folder where those engines live (`npm run bench -- size --engines=all`), and published with its method when it has been run.

6 What is in the box

Area	What exists today	Source
Rendering	glTF 2.0 metallic-roughness PBR, image-based lighting, cascaded shadows, clustered point, spot and area lights, order-independent transparency, LOD, GPU culling, text	package READMEs
Post	bloom, SSAO, GTAO, SSGI, SSR, motion blur, tone mapping, depth of field, TAA	packages/post, dof, tonemap, taa
Voxels	chunked worlds, meshing, lighting, streaming; multiplayer: one server process keeps 20 ticks per second for 900 players spread over a map or 350 crowded in one spot (AMD Ryzen 9 9950X)	docs/voxel-net-capacity-2026-10-21.md · voxel-net/bench/results.json
Path tracing	a progressive WebGPU path tracer for voxel models, through the engine camera	packages/voxel-pathtracer · voxel whitepaper
Export	glTF 2.0 / GLB, STL, OBJ with MTL, PLY, static USDZ and COLLADA 1.4.1	packages/exporters/README.md
Frameworks	React 18 and 19 components, and a Next.js starter	packages/react · tools/create-margo
Simulation	Rapier physics, navigation meshes, ECS, state machines, networking	package READMEs

Of the feature list this page counts, 153 of 161 rows have a Margo package, 4 are partial and 4 are missing. The API is Margo's own.

`source docs/three-parity-matrix.md · bec8b597`

7 Internal measurements

These are measurements of Margo against earlier Margo, on named hardware. They are not comparisons with other engines. The method: headless Edge with vsync and the frame cap off, WebGPU timestamp queries on every pass, a GPU lock so only one GPU job runs at a time, and a quiet-machine gate. Our benchmark tool, `npm run bench`, now runs every such measurement this way.

Large scenes

`FloatingOrigin` keeps a 10 mm part at 0.019 px of jitter up to 6,000 km from the origin, where float32 alone stops drawing it (`results/large-coordinates-gpu.json`).

8 How it was built

Margo is built by one person directing several AI coding systems through an external Agent Factory. Scott Campbell sets the goals and product rules. Factory owns routing, work isolation, review, commits, integration, publication, machine scheduling and durable history outside this repository. Nobody should trust the engine because of who or what wrote it. The checks below are what make it trustworthy, and section 10 shows what they caught in three days, including our own mistakes.

- **One external controller.** Factory owns every workflow and Git mutation. The engine repository contains code, documentation and non-committing quality tools, not a second queue, integrator, scheduler or stop-order system.
- **Parallel work, isolated.** Factory gives each coding task an isolated checkout and serializes integration to `main`. The repository does not create worktrees or branches in response to an ordinary edit.
- **Proportionate checks.** A build runs the affected package checks and relevant dependents. `npm run check:merge:full`, pinned proofs, stress tests and benchmarks remain explicit release or owner-requested work rather than automatic repetition after every small change.
- **Bounded hardware use.** GPU correctness tests use a generic shared lock with hard time limits; measurements run exclusively. Factory chooses a capable machine and records deferred coverage instead of treating an unavailable runtime as a code defect.
- **A regression test for every bug.** A fix lands with a test that fails without it. When a whole class of bug slips through, a check for the class is added, for example the gate on fixed waits in GPU checks and the reassociating-compiler stand-in in section 10.
- **Independent full validation.** Factory can schedule `check:merge:full` and pinned proofs on the combined tree for release evidence without embedding a fleet scheduler or publisher in Margo.
- **One rerun, only after a logged GPU driver reset.** A step that fails while the Windows event log shows a GPU driver reset runs once more. The reset and the rerun are printed and kept as a receipt in the check's result file, and a second failure fails the check. A failure without a logged reset is never retried.
- **Clean room.** No files from other repositories, under any licence. Code is written from public specifications, papers and descriptions. When a 2026-10-21 audit found areas close to other engines' or libraries' code, each was rewritten from a words-only specification by sessions that never saw the old code. Other engines are run only from their own folder for benchmarks. The few standard files every engine uses, such as sample models and prebuilt decoders, are listed with source and licence in `THIRD-PARTY-NOTICES.md`.
- **Gates that only ratchet.** `npm run quality` runs in every verify and in CI: entry sizes and heavy static imports, duplicated code, untested packages and entries, TSDoc and error conventions, and the clean-room checks. Each can only shrink unless a change records its reason. `npm run hygiene` scores eight more code gates per package.
- **Outside eyes.** Test runs on machines we do not develop on (an AMD mini PC, an Intel Linux server, a second laptop) and an outside audit found problems our own runs had missed. Their findings are listed in section 10 and in `docs/known-bugs.md`.
- **Honest reporting.** Sessions report measured numbers only, label anything measured on a busy machine as such, and say when a check could not run.

We describe the method because it is how the engine was made. AI does not make the engine better by itself. The evidence for the engine is the same as for any engine: the tests, the measurements and the code.

9 Keeping the numbers honest

Tests and checks

- **6,498 Node tests** in every `npm run verify` at d18f2596: 6,496 pass, 2 skipped, 0 fail, on Node 26.12 (`results/test-count.json`). An outside audit's run on a second laptop at 74a01810 passed 6,344 of 6,346, with 2 skipped.
- **49 GPU-call proofs.** A recording GPU logs every call, descriptor, shader and buffer write, and a proof compares a baseline commit with a candidate exactly. That checks claims such as "attaching nothing from this package leaves core's GPU work unchanged". Each proof is pinned to the commit that made its claim, and 20 run against the working tree in every merge.
- **Pixels on a real GPU.** The visual check renders 62 views of sample models and demos and compares them with references captured four times per GPU and browser. A view is never compared with references from another GPU (`tests/visual/profiles/`).
- **Every shader compiled.** `npm run check:wgsl` compiles every WGSL source the packages hold on the browser's WebGPU, with feature flags both on and off, including shaders no view draws.
- **Hostile files.** Every parser of untrusted input has a seeded fuzzer: glTF, Draco, KTX2, meshopt, HDR, EXR, FBX, OBJ, STL, PLY, 3MF, USD, IFC and network snapshots. Its findings were fixed with regression tests. Examples: a 314-byte PLY that allocated 336 MiB, a 3MF zip entry that inflated 1,000-fold, and IFC files that looped forever.
- **Security.** A review fixed model files whose URIs escaped their directory, a quadratic regular expression in the STL reader, and glTF keys that reached the object prototype. The standard WebGPU path and the other WebGL2 tiers need no `'unsafe-eval'`; the Wasm kernel needs `'wasm-unsafe-eval'`, and a strict policy on the page and every worker ran with zero violations in Edge (`docs/security.md`, `results/csp-check.json`). The optional WebGL2 CPU compute tier of `webgl2-compat` builds its shaders as JavaScript with `new Function`, so it needs `'unsafe-eval'` on the page and is refused (MargoError `unsupported`) without it (`packages/webgl2-compat/README.md`).

A claims inventory

`docs/whitepaper/claims.md` lists every claim the public docs make or imply, with its evidence and a status. A claim is PROVEN only when a committed script and a committed result support its exact wording. The audit that built it corrected or removed every claim that overstated its evidence.

The words we do not use, and why:

- "first" or "only": we have not surveyed every engine;
- "smallest": not without a published method and measurement;
- "drop-in replacement": the API is our own;

- "production-ready" or "runs everywhere": section 11 says why not yet;
- faster than any other engine: no quiet-machine comparison has been run.

10 What our checks caught, and what we got wrong

From 2026-10-26 to 10-28 the engine ran on more machines than before: an AMD Radeon 680M mini PC (Windows, Edge and Chrome), a small Ubuntu server with an Intel UHD GPU (Chromium on Mesa's Vulkan driver), a second laptop for an outside audit, and Node 22 beside Node 24 and 26. Most of what broke was in our maths, our tests or our machines, and several first explanations were wrong. Each item names its fix commit and regression test; the full entries, with every number's source and the rerun commands, are entries 37 to 44 of `docs/whitepaper/perf-history.md`.

GPU compilers rewrote our precise maths

On the Radeon 680M, `preciseLog1p(5.963922e-8)` was 8,383,741 units in the last place (ulp) off, against a bound of 2; on the Intel UHD with Mesa, 14 of the 17 precise functions were out of bound. WGSL allows a compiler to reassociate floating-point operations, and the compilers folded the error-free steps the functions depend on, for example `(1 + x) - 1` into `x`. A bitcast round trip does not stop that. The fix, `pmHide` in `packages/shared/precise-math.js`, rebuilds a value from its bits with integer operations, which no compiler can fold (`b6bea3e9`). All 17 are within bound on the three vendors. What we got wrong: our first stand-in for such a compiler only folded self-cancelling sums, so it missed Mesa factoring a sum by one of its own terms; the stand-in now does both, and the old code fails it on any GPU (`baa66526`). A dense sweep (573,720 inputs for sin, cos and tan, 298,646 for pow) is 0 over bound on Intel and NVIDIA; the pre-fix code on Intel had 1,791 sin results over bound. Guards: `tests/precise-math.test.js`, `packages/compute/tests/maths.test.js`, compute `test:browser:reassociate` and `test:browser:sweep`. Not measured yet: what the extra work costs.

A proof failed on two machines, and the engine was fine

The `math-consolidation` proof passed on Node 26 and failed on two machines with Node 22: 21 of 24 digests differed. ECMAScript leaves the rounding of `**` and the approximated `Math` functions to the engine; V8 in Node 26 uses the C library's `pow` and V8 in Node 22 its own, so `10 ** -5` differs. The proof now builds inputs from exact operations and fails if a taped call uses an approximated function (`ab27353f`, guard `tests/proofs-golden.test.js`, run in CI on Node 22.12 and 26).

Flaky tests had one shared cause

Four tests passed alone and failed under load, each because its timing depended on how busy the machine was: a heap probe that measured the shared test runner's heap (5.9 MB once; in its own process the same loop grows by at most 30 KB), a catalog test that read the system clipboard every process shares (both `6d08b0af`), a 300 ms server timeout whose clock started before the child process ran (`3a59aa55`), and a walk check that bounded the eye's movement per frame while the engine clamps a slow frame at 0.1 s (`7714e84e`; the guard `109ec352` walks at 1/60, 25, 40 and 100 ms frames and still fails a 0.4 m jump). The same class gave

two more: browser suites sharing one Vite cache (`7ab8cd13`) and a suite waiting silently for a fixed port (`6ddd733b`). What we got wrong: we first blamed time spent queueing for the GPU lock; measurement ruled that out.

Headless Edge picked the wrong GPU

After a reboot, headless Edge on the main PC used the integrated AMD GPU instead of the RTX A6000 whenever the Remote Desktop session was disconnected, and every visual check failed for lack of references for that GPU. Chromium uses DXGI adapter 0, and Windows lists first the adapter that carries the session's display. Power-preference flags and the Windows per-app setting had no measured effect. The launcher now looks up the hardware adapter with the most video memory at launch and passes its LUID (`85c96a55`, `scripts/lib/gpu-adapter.mjs`, guard `tests/gpu-adapter.test.js`). Still unexplained: before that reboot, checks got the A6000 with the session disconnected.

NVIDIA driver resets: cause not found

Nine `nvlddmkm` event 153 resets on 10-27 and 28 each took one page's WebGPU device during busy check runs, and the next page asking for a device was refused. We measured that event 153 is not the Windows watchdog: a deliberate 3 s dispatch gives a watchdog report (event 141) and no 153. The 141 reset is worse for correctness: WebGPU kept the device, and the cut-short work returned wrong results with no error. Our first theory, long path tracer dispatches, was wrong. We bounded them anyway (`49dca9dd`, at most 2^{20} paths per dispatch, same pixels), but controlled runs gave no reset, and kills, GPU process crashes, device churn and contention did not reproduce one. Eight of the nine came while the Remote Desktop session was connected. **The cause is not found.** What changed is the checks (`51d8786a`): a lost device fails at once, the message names the reset from the Windows event log, and a step that fails while a reset is logged runs once more, with the reset and the rerun kept as a receipt, and a run that passed only that way is marked as one (`6a8f7658`); a second failure fails the check. A watchdog reset that cuts work short while a check still passes is not caught this way.

Node 22 for end users

The packages must run on Node 22.12 or newer. On Node 22.12.0 the packed packages build, install into a clean project, type-check their README examples and pass the bundler matrix (`a4fca9be`); a scan of every shipped source file for 33 APIs that Node 22.12 lacks found no unguarded use. The unit tests of the 47 packages that run in Node pass on 22.12.0 and 22.22.1: 355 files, 5 of 5 runs, 2,866 tests on 22.12.0 (`86ed9107`). This found two real bugs: netcode's `openWebSocket` hung forever on a refused connection, because Node 22's WebSocket fires only `error` (`7f0dd465`), and data after the end of a compressed stream was accepted on Node 22 in the FBX, ZIP, SOG and voxel chunk readers (`0687e6ec`, guard `tests/trailing-data.test.js`). These runs were on Windows; the new CI jobs have not run on GitHub yet.

Linux on a real Intel GPU

On the Ubuntu server, 42 of 43 package GPU suites were started, one at a time, and 38 passed (main `b92998d6`). The text suite failed because the server has no colour emoji font and the test relies on one from

the system; that is open. The voxel, voxel path tracer and root browser suites were stopped by the heat watchdog at 82 C before they finished, so **they have no Linux result**. An earlier run there found a real engine bug: after an effect was removed, the voxel renderer drew nothing for a frame or two while pipelines recompiled (fixed in `baa66526`; removing an effect out of order still does it, and is open).

An outside audit

An outside audit of revision `74a01810` combined a source review, canary tests on a second laptop (temporary files, mock fetches, fake sockets and loopback servers only), a full verify, a short fuzz run over 36 targets and GPU checks. Verify passed 6,344 of 6,346 tests, with 2 skipped. It reported 20 findings, each marked reproduced, code confirmed or project documented, and it reproduced our known 4x MSAA blending variance on a second GPU (Intel Arc 140V: 4 frames of 450 differ at one pixel by up to 9 of 255). It states its own limits: a short fuzz run, local canaries only, no production test, and a passing verify does not show there are no bugs. Our own security review and fuzzers had not found these. Status on `main`: 16 are fixed, each with a regression test that fails without the fix; 1 is being fixed, and we describe it once its fix lands; 3 are limits the netcode room server does not have yet (netcode leaves authentication to the application), and proposals for them are pending. The fixed ones, by class:

- **Files outside their folder** (`4bc89574`): the margo-convert command line read a model's external files, deleted stale outputs and accepted an output path without checking that each stayed in its folder, so a crafted `.gltf` could pull in outside files, a crafted manifest could delete one, and `./file` or a link could overwrite the input. Every path is now checked against its folder, links included. Guard: `tests/margo-convert-paths.test.js`.
- **URL allowlist and redirects** (`c6e7623c`): an `allowedURLs` entry matched by string prefix, so `/assets` also allowed `/assets-other/`, and a redirect's final URL was never checked. Entries now match on a path boundary and every redirect is checked. Guard: `tests/security.test.js`.
- **OBJ material libraries** (`b96f5f40`): `loadOBJ` fetched any `mtllib` named in the file, from any host, with the caller's headers, and with no total budget. Libraries now stay in the OBJ's folder or the allowlist, headers go only to the OBJ's origin, and the libraries share a budget. Guard: `packages/loaders/tests/mtl.test.js`.
- **Network servers** (`158076f1`, `6bebf878`, `8c8b51a0`): a netcode room was joinable before its set-up finished and a failing room hook left the client hanging; control frames counted against the WebSocket message limit and close reasons were cut by characters instead of UTF-8 bytes; voxel-net dropped a hello sent in the same write as the upgrade and never retried a chunk whose generation failed. The voxel-net prototype also had no authentication, no cap on connections before the hello and no bound on edited chunks in memory; it now has all three, and accepts pages from its own host by default (`ddc5c5ff`). Guards: `packages/netcode/tests/rooms.test.js`, `transports.test.js`, `packages/voxel-net/tests/websocket.test.js`, `limits.test.js`.

```
source docs/whitepaper/perf-history.md entries 37 to 44 · docs/known-bugs.md · commit messages named above
```

11 Honest limits

- **WebGPU only today.** PBR, shadows, image-based lighting and most packages need WebGPU, which about 88% of browsers in use support (caniuse.com/webgpu, September 2026). A fuller WebGL2 compatibility package is in development as an optional download that WebGPU users never load; today WebGL2 draws basic meshes only.
- **Not yet verified** on Firefox, Safari, macOS, Android or iPhone. Real-GPU results come from Edge and Chrome on Windows 11 (an NVIDIA RTX A6000 desktop, Intel Arc 140V laptops and an AMD Radeon 680M mini PC) and from Chromium on one Ubuntu machine with an Intel UHD GPU, where the voxel, path tracer and root browser suites have not yet run to the end (section 10).
- **No editor yet.** Scenes are built in code or imported from files.
- **No comparison with other engines yet.** The tool runs the same scenes in other engines, interleaved on one machine, but the quiet-machine runs have not been made. Until they are, we make no speed claim against any other engine.
- **Next.js production builds** need Rapier installed today, because the React package's root reaches it. The React package is being split so they do not.
- **Design envelope.** 8,388,608 entities; 100,000 nodes per file by default; 16 textures per shader stage; float32 coordinates without `FloatingOrigin`. Every limit, with the error it raises and the package that takes you past it, is in `docs/limits.md`.

12 Coming soon

- The packages on npm, `npm create margo`, and the public repository.
- A WebGL2 fallback for the optional packages.
- Firefox, Safari and phone verification.
- Quiet-machine comparisons with other engines, published with their method and the places other engines win.
- Data visualization and GPU compute suites.
- An editor.

13 Reproduce

From a checkout (Node 24.18 or newer for the repository; the published packages run on Node 22.12 or newer). Each command writes or prints the figure it backs; GPU commands take the shared GPU lock.

```
npm ci
npm run build:packages
npm run measure:size          # core + stage gzip, fails at 25,000
npm run verify                # the 6,498 tests, quality gates, API reports,
current proofs
npm run bench                 # every benchmark suite (docs/benchmarking.md)
npm run proofs                # all 49 pinned GPU-call proofs
npm run fuzz -- --short       # every parser fuzzer, 4 s each
npm run check:merge           # verify + site, visual, views, WGSL and packed-
package checks (GPU)
```

Per-claim commands are in `docs/whitepaper/reproduce.md`.

Margo Engine's own code is Apache-2.0 licensed. Third-party code that ships inside packages (the Draco and Basis decoders) is listed with its licence in `THIRD-PARTY-NOTICES.md`.

This paper as a PDF Margo Voxel whitepaper Sizes Catalog Third-party notices

Built with precision, named with love. For Margie.