

Writing a Game Once and Running It Everywhere: Where Exact Replay Ends and Tolerance Begins

David Charlot, Open Interface Engineering

Research study · OpenIE research.openie.dev

<https://research.openie.dev/papers/game-engines/> · PDF <https://research.openie.dev/pdfs/game-engines.pdf>

Measurement. Research paper, not a journal final. Energy figures from the software reference are OpCounter analytical estimates, not board power. We have not synthesized or metered an FPGA board. No fabricated citations.

Can a game be written once as a checked state machine and projected into several engines and languages? For rules and simulation, yes, under strict arithmetic. For pixels across graphics hardware, no: only agreement within a measured tolerance is reachable.

Abstract

This study asks whether a game can be written once as a checked state machine and then projected into several engines and languages. The sources answer: yes for game rules and simulation, under strict numeric discipline; no for bit-exact pixels across graphics hardware, where only tolerance-based agreement is achievable.

In plain words

1. The rules of a game (who moves, what collides, who wins) can be stated once and replayed exactly, provided the arithmetic is pinned down.
2. The pictures a game draws cannot be made identical on every graphics card. They can be checked to stay within a measured error.
3. Today's game description formats each cover one slice (shaders, scenes, or rules). Almost none checks the seams between slices, and that seam is where this design concentrates its effort.

 Diagram of the proposed split. A box at the top holds the game Form P equals S , E , δ , D , λ , with typed interface schemas and a strict checker. Arrows lead to four projections: simulation (Rust, WebAssembly, XState, SCXML, Python) and scripting (Rhai, Lua, Luau) on the left in the exact zone, judged by a per-tick state hash; scene (BSN-like, glTF, OpenUSD) and rendering (Slang, WGSL, WESL) on the right in the tolerance zone, judged by per-pixel error. A dashed vertical line marks the determinism boundary. A band at the bottom lists host effects that enter only as logged events: input, clock, audio, network, asset input and output, GPU draws, cosmetic physics.

Figure 1. Proposed split. One Form feeds four projections. Simulation and scripting sit in the exact zone, judged by per-tick state hashes. Scene and rendering sit in the tolerance zone, judged by per-pixel error. Host effects enter only as logged events. [Open full size](#) · [PNG](#)

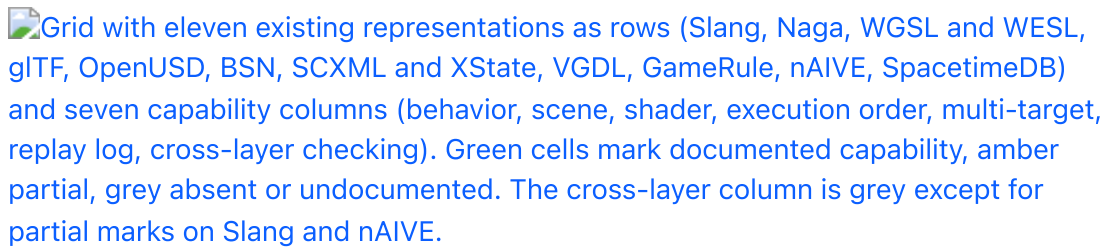
Grid with eleven existing representations as rows (Slang, Naga, WGSL and WESL, glTF, OpenUSD, BSN, SCXML and XState, VGD, GameRule, nAIVE, SpacetimeDB) and seven capability columns (behavior, scene, shader, execution order, multi-target, replay log, cross-layer checking). Green cells mark documented capability, amber partial, grey absent or undocumented. The cross-layer column is grey except for partial marks on Slang and nAIVE.

Figure 2. Coverage of existing representations, scored only from the documents read for this study. Green: documented capability. Amber: partial. Grey: absent or undocumented. The rightmost column, cross-layer checking, is nearly empty. [Open full size](#) · [PNG](#)

1. The idea under test

The [Reality Code Engine \(RCE\)](#) treats a program as a state machine called a Form, written $P = (S, E, \delta, D, \lambda)$:

- S : the set of states;
- E : the set of events (inputs);
- $\delta : S \times E \rightarrow S$: the transition function;
- D : an append-only log of every step taken;
- λ : the output map from state to observable output.

A strict checker either accepts a Form or names the rule it breaks. Emitters then project the accepted Form into Rust, Python, XState (a JavaScript and TypeScript statechart library), SCXML (State Chart XML, a World Wide Web Consortium Recommendation), and WebAssembly (Wasm). Because every step lands in D , any run can be replayed.

A game engine is a much larger machine. It runs a fixed-step simulation, a scripting layer, a scene description, and a rendering pipeline on a graphics processing unit (GPU). This study asks which of those layers can live inside a Form, which must stay outside as host effects, and where exactness has to give way to tolerance.

The idea has a long pedigree. The Stanford [Game Description Language \(GDL\) specification](#) covers “finite, discrete, deterministic multi-player games of complete information” and observes that the natural underlying model of such games “is a finite state machine.” What is new in a game engine is not the state machine but everything around it: floating-point physics, scripts, scenes, and GPUs.

2. What the landscape shows

2.1 Game logic as data already works, on one backend

GameRule ([Multimedia Systems 32:564, 2026](#)) expresses behavior as per-actor *when-do* rules over 14 create, read, update and delete actions and 6 conditions. Its deterministic contract is concrete: one central scheduler runs from a single FixedUpdate in declaration order, the last write wins, and spawns and deletes are deferred to the end of the traversal. The authors justify the contract by quoting Unity’s manual, which says script execution order “isn’t guaranteed across builds, machines, or Unity versions.” Across three Unity sample games (Haunted Jaunt, Survival Shooter, Tanks), behavior code shrank from 572 to 107 lines (–81.3%).

The price is measured. Rule interpretation costs about $c \approx 65 \mu s$ per actor per step. With a step budget $T = 20$ ms, the actor ceiling is

$$N_{\max} = \frac{T}{c} = \frac{20 \text{ ms}}{65 \mu s} \approx 308,$$

which agrees with the paper’s observation that the budget is exceeded between 200 and 350 actors (23.872 ms at 350). Only a Unity backend exists, so portability across engines is argued but not demonstrated.

2.2 Order of execution can be made deterministic by construction

The entity component system (ECS) pattern stores game state as components attached to entity identifiers and runs behavior as systems over them. Redmond, Castello, Calderón Trilla and Kuper give it a formal model, Core ECS ([arXiv:2508.15264](#)). They prove that a schedule in which every concurrent pair of mutations commutes is deterministic, and that two sub-schedules writing disjoint sets of component labels may run in parallel without changing the result. Surveying Bevy ECS, EnTT, Flecs, Specs and apecs, they find that none allows attaching a new component concurrently with another mutation; such structural changes are deferred and applied serially. Tasnim and Zhao ([SAC 2026, doi:10.1145/3748522.3779910](#)) add an operational semantics and a type system for archetype ECS that tracks read and write sets per system, and report on a Tower Defense simulation that a parallel archetype layout reached 109.54 frames per second against 36.13 for an object-oriented baseline. GameRule’s deferred spawn and delete is the same discipline, reached independently.

2.3 Compiling is not the same as being right

The Mage benchmark ([arXiv:2605.07342](#)) ran 858 generation attempts over 26 Unity mechanic patterns. Without an intermediate representation (IR), 43% of outputs passed at runtime, yet mechanism F1 (agreement between requested and implemented mechanics) was only about 0.12. Conditioning on an IR dropped runtime pass to 14% to 21% but lifted mechanism F1 to 0.82 to 1.00. Compile rate was anti-correlated with functional correctness: the missing-type error CS0246 appeared 469 times under IR conditioning and zero times without it. A behavior-only IR

and a full-scene IR were statistically indistinguishable (McNemar $p = 1.0$). The lesson for RCE: a checker that only proves “it compiles” measures the wrong quantity. Interface checks must be behavioral.

2.4 Public benchmarks with replayable traces exist

GameCraft-Bench ([arXiv:2606.17861](#); [repository](#)) offers 140 Godot 4 two-dimensional tasks in 15 families. Submissions carry replayable input traces. The strongest agent in the paper scored 41.46%, and the repository later reports 65.70%. It does not cover Unity, Unreal, 3D, multiplayer, or large physics scenes. [Summer Engine](#) publishes 17 open-source game templates (rally, platformer, tower defense, metroidvania, survivors, first-person shooter, LAN multiplayer, voxel, and others). The General Video Game AI framework and its Video Game Description Language (VGDL) ([arXiv:1802.10363](#)) supply 120 single-player and 60 two-player games defined by sprite, interaction, termination, and level-mapping sets.

2.5 Convergence across industry and research

Several efforts are converging on the same shape: declarative scene and pipeline files, a sandboxed scripting layer, a modern shader language, and an event log.

- [nAIVE](#) is built in Rust on wgpu with an ECS (hecs), YAML scenes, materials and render pipelines, Lua 5.4 with per-entity sandboxing, Slang shaders cross-compiled to the WebGPU Shading Language (WGSL), Rapier physics, an event bus with YAML schema validation, and a headless test runner. Its [site](#) shows a 92-line YAML file declaring a five-pass deferred pipeline. Its README also describes GPU compute simulation driving 50,000+ entities, a choice that places that simulation on the tolerance side of the boundary drawn in Section 3.
- Capcom describes REX (“RE neXt Engine”) as a staged evolution of RE ENGINE involving about 200 engineers ([Capcom IR](#)). Press coverage ([PC Gamer](#)) reports six programs, among them Re:Dox, which translates different types of data into a common set of rules, and RE:Flows, which converts game mechanic ideas into a standardised programming language; it also reports that more than 2,000 people use RE Engine.
- Bevy’s BSN proposal ([discussion 14437](#)) defines scenes as composable Patch, EntityPatch, and DynamicScene values with a `bsn!` macro and an asset format, while leaving event handlers unresolved.
- [SpacetimeDB](#) runs game logic as reducers that execute as atomic, consistent, isolated, durable (ACID) transactions over tables and replays a commit log, which is the RCE log D under another name.
- Executable Ontologies ([doi:10.48550/arXiv.2601.07964](#)) model game behavior in the Boldsea Semantic Language (BSL) over an append-only, immutable event graph. The paper recommends a hybrid in which physics, rendering and animation keep traditional high-frequency approaches while quests and decisions run on the event graph, joined through aggregated events. That is the same layer split proposed here.

3. The physics of divergence: why exactness is binary

Floating-point addition is not associative. On parallel hardware the order of a reduction varies run to run. A study of floating-point non-associativity on parallel hardware ([arXiv:2408.05148](https://arxiv.org/abs/2408.05148)) defines the scalar variability

$$V_s = 1 - \left| \frac{f_{nd}}{f_d} \right|,$$

where f_d is a deterministic reduction and f_{nd} a nondeterministic one. Permuting a sum of 10^6 normally distributed double-precision values changed the result by up to about 4.26×10^{-13} , and the maximum $|V_s|$ grew as $\sqrt{n}$. A deterministic tree reduction cost about 0.2% relative to atomic accumulation on an NVIDIA V100 and up to 7.8% on a GH200. In 1,000 graph-network training runs, every run produced distinct weights.

A game turns such tiny differences into visible ones. Rigid-body contact and many-body dynamics are typically chaotic. Under a local Lyapunov exponent Λ , a perturbation grows as

$$\delta(t) = \delta_0 e^{\Lambda t},$$

so the time until it reaches a gameplay-visible size Δ is

$$t^* = \frac{1}{\Lambda} \ln \frac{\Delta}{\delta_0}.$$

Illustrative model, not a measurement: take $\delta_0 \approx 6 \times 10^{-8}$ (one unit in the last place of single precision, relative), $\Delta = 1$ (an object on the other side of a wall), and an assumed $\Lambda = 2 \text{ s}^{-1}$. Then $t^* \approx 0.5 \ln(1.7 \times 10^7) \approx 8.3 \text{ s}$. The logarithm is the key: shrinking δ_0 a millionfold buys only about 6.9 more seconds. Approximate agreement therefore cannot rescue simulation. Either the state is bitwise identical or it eventually diverges.

Practice agrees. [Gaffer On Games](#) records that cross-machine determinism requires the same compiler and instruction set, that Battlezone 2 hit AMD versus Intel differences in transcendental functions, and that MotoGP replays could not be shared between Xbox and PC. It cites Markstein's advice to forbid fused multiply-add contraction and reassociation. The [Rapier guide](#) shows a working path: cross-platform determinism holds only with its enhanced-determinism feature (incompatible with `simd8`), on IEEE 754-2008 targets (Wasm qualifies), using software transcendentals.

For rendering, the [WGSL specification](#) states that "WGSL sometimes permits several possible behaviors for a given feature. This is a portability hazard." Its floating-point section goes further: "WGSL does not specify a rounding mode"; "An implementation may reassociate operations"; it "may fuse operations if the transformed expression is at least as accurate as the original"; subnormal inputs and outputs of many operations "may be flushed to zero"; and the `fma` built-in "may expand to an ordinary multiply ... and an add." Pixels are thus specified to differ across implementations. The [SPIR-V specification](#) offers a narrower tool: its `NoContraction` decoration

marks an arithmetic instruction that “cannot be combined with another instruction to form a single operation,” and such operations “are not allowed to reassociate.” That is a per-instruction guard on one back end, not a portable guarantee for a whole shader.

4. The IR table

Representation	Layer	Strongest property (documented)	What it lacks for a Form
Slang	Shaders	Module IR; back ends for D3D12, Vulkan, Metal, D3D11, OpenGL, CUDA, WebGPU, CPU; capability system	No game state, no events, no exact-semantics guarantee for floats
Naga	Shaders	Validated IR; WGSL, SPIR-V and GLSL in; SPIR-V, WGSL, MSL, HLSL and GLSL out	Same; no link to a scene schema
WGSL / WESL	Shaders	Uniformity analysis; WESL adds imports, conditional compilation, packages, <code>wgsl-test</code>	Permits reassociation, fusion and flush-to-zero; no host-side types
SPIR-V	Shader binary IR	NoContraction decoration forbids fusion and reassociation per instruction	Per-instruction only; no game semantics
MLIR	Compiler infrastructure	Multi-level dialects for lowering	Not a game model; semantics left to each dialect
glTF 2.0 (ISO/IEC 12113:2022)	Assets and scene	Interchange standard across tools	No behavior, no execution order
OpenUSD	Scene	Composition arcs, namespace paths	Self-described “Not a rigging system”; no simulation semantics
BSN	Scene	Typed patches over components, versioned imports	Event handlers unresolved; Bevy only
SCXML / XState	Behavior	Microstep and macrostep semantics, document-order priority, serial parallel regions	Data model is platform-specific; no numeric discipline, no scene or shader
GDL	Rules	Logic description of finite, discrete, deterministic games	No continuous physics, no rendering
VGDL	Rules	Compact 2D grid games, large public corpus	2D grid only; no rendering or physics fidelity
Core ECS	Execution order	Theorem giving a condition for deterministic concurrent schedules	A formal model, not an engine format; no scene or shader

Representation	Layer	Strongest property (documented)	What it lacks for a Form
GameRule	Rules	Deterministic scheduler contract, measured code savings	Unity only; about 65 μ s per actor; no formal checker
nAIVE	Whole engine	YAML scenes and pipelines, Lua, Slang, event schema validation	No published formal semantics or cross-layer type check
SpacetimeDB	Server logic	Reducers as transactions, commit-log replay	Server side only; no client rendering or scene

5. Proposed architecture

5.1 Layer split

1. **Simulation Form (exact).** S holds fixed-point or strict IEEE 754 values, with fused multiply-add and reassociation disabled and transcendentals implemented in software. E holds tick events and logged inputs. δ uses SCXML macrostep semantics: per tick, process events in queue order, resolve conflicts by document order, run parallel regions serially. Collections iterate in a declared order, following GameRule’s contract (declaration order, last write wins, structural changes deferred to the end of the tick). Where parallelism is wanted, the checker admits only schedules that satisfy the Core ECS disjointness condition.
2. **Scripting projection (exact).** Designer scripts compile to guards and actions over S , emitted as [Rhai](#) (sandboxed, with a “Don’t Panic” guarantee, no-std and Wasm support), Lua, or [Luau](#) (sandboxed by default, with gradual types). Scripts may only call δ through declared actions. They cannot read clocks, random sources, or GPU results directly.
3. **Scene projection (tolerance downstream).** λ maps S to a BSN-like patch tree whose component schema is part of the Form. glTF and OpenUSD serve as export targets for assets.
4. **Rendering projection (tolerance).** Materials and passes are authored in Slang and emitted to WGSL, and through Slang or Naga to other back ends. Rendering is a pure function of $\lambda(s)$. No GPU result ever flows back into S .

5.2 Inside the Form versus host effects

Inside: state, transitions, rule order, fixed-point physics for gameplay-relevant bodies, the random generator with its seed in S , interface schemas. Outside, as logged events: device input, wall-clock time, network packets, audio output, asset input and output, GPU draws, and any third-party physics used for cosmetic effects. Replay feeds D back through δ rather than touching devices. Storage cost is modest. At a tick rate $f_{\text{tick}} = 60$ Hz and $b = 16$ bytes of input per tick, the log grows at $f_{\text{tick}} \cdot b = 960$ B/s, about 3.5 MB per hour.

5.3 Determinism boundary

- **Exact zone criterion:** for targets A and B and every tick t , $H(s_t^A) = H(s_t^B)$, where H is a cryptographic hash over a canonical serialization of S .
- **Tolerance zone criterion:** for rendered frames I^A and I^B of the same s_t , the fraction of pixels with per-channel error above ε must not exceed $p_{\max}$. The design fixes $(\varepsilon, p_{\max})$ per material class from calibration runs, not from first principles. *Opinion:* a perceptual metric is preferable to raw error for final judgment, but this study cites none because none was read in full.
- Rollback networking in the style of [GGPO](#), which relies on input prediction and speculative execution, is valid only on the exact side of this boundary.

5.4 Cross-layer interface checking

The checker validates four typed edges before any emission: (a) script to simulation: every action names a declared transition with matching argument types; (b) simulation to scene: the output of λ matches the component schema; (c) scene to rendering: every material binding matches a Slang or WGSL resource of compatible type, group, and binding index; (d) event coverage: every event a script listens for is producible by some transition or host source. Mage’s finding (Section 2.3) is the reason edge checks are judged by behavior traces, not by compile success.

6. Falsifier-first test plan

Each hypothesis is listed with its kill criterion. If the criterion is met, the claim is withdrawn.

ID	Hypothesis	Test corpus	Measure	Kill criterion
H1	Simulation projections are bitwise identical	VGDL games from GVGA!; GameCraft-Bench traces	$H(s_t)$ across Rust native, Rust to Wasm, Python, XState for 10^5 ticks	Any hash mismatch at any tick on any pair
H2	Strict mode survives a hardware change	Same, on x86-64 and ARM64 hosts	Hash equality	Any mismatch
H3	Without strict mode, divergence follows $t^* \propto \ln(\Delta/\delta_0)$	Physics-heavy Summer Engine templates re-expressed as Forms	Divergence time against injected δ_0	Slope of t^* against $\ln(1/\delta_0)$ not distinguishable from zero
H4	Form-encoded rules cut code as GameRule did	Haunted Jaunt, Survival Shooter, Tanks behaviors	Lines of behavior code	Reduction under 50% (GameRule reached 81.3%)
H5	Form runtime exceeds the GameRule ceiling	Synthetic actor scaling	Step time against N	Step time over 20 ms below $N = 350$
H6	Interface checks catch real defects	Mage’s 26 patterns; injected binding and	Detection rate; mechanism F1	Detection below 90% of injected faults, or F1 no

ID	Hypothesis	Test corpus	Measure	Kill criterion
		schema faults		better than the no-IR 0.12
H7	Rendering stays within tolerance	Same scene on three GPU vendors via WGSL	Pixel error fraction	Above $p_{\max}$ on any calibrated material class
H8	Replay is complete	One-hour recorded sessions	Final hash reproduced from D alone	Any replay that needs state not in D
H9	Benchmark parity	GameCraft-Bench 140 tasks via a Form-to-Godot emitter	Pass rate against native Godot submissions	Lower pass rate on behavior-identical inputs

7. Infeasibility flags

1. **Cross-GPU bit-exact rendering: infeasible.** WGSL leaves the rounding mode unspecified and permits reassociation, fusion and flush-to-zero; parallel reduction order changes results (Section 3).
2. **Cross-platform exact physics with stock engine settings: infeasible.** Rapier requires a restricted feature set; Gaffer documents vendor differences in transcendentals.
3. **GPU compute inside the simulation: infeasible as exact; feasible only with deterministic reductions** at a measured cost of about 0.2% to 7.8% on the cited hardware, and still subject to compiler contraction unless guarded per instruction (SPIR-V NoContraction) on every target.
4. **Engine-portable behavior: not demonstrated in any source read.** GameRule is Unity only; nAIVE and BSN are single-engine.
5. **Interpreted rules at large actor counts: constrained.** The GameRule measurement implies a ceiling near 300 actors at 20 ms; a compiled projection is required beyond that.
6. **Proprietary engines as exact targets: unverified.** Unity's manual, as quoted by GameRule, disclaims ordering guarantees; the Unity and Unreal template catalogs could not be read (their pages render client-side), and no Unreal execution-order source was read.

8. Sources and limits

Every source above was read in full at the passage cited, except: the GameCraft-Bench paper (abstract and repository README read), the RCE about page (summary page read), and the Capcom press story (one article). Sources consulted but not cited as evidence: the [Ludii portal](#), which describes games as structured sets of ludemes; the Game Grammar hobby repository; and the [PlayCanvas](#), [Babylon.js](#) and [three.js](#) READMEs, which describe rendering libraries rather than game description formats.

Not read, paywalled or otherwise unavailable: T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning," IEEE CIG 2013, doi:10.1109/CIG.2013.6633610; J. Gregory, *Game Engine Architecture*, doi:10.1201/9781315267845. M. Ebner et al., "Towards a Video Game Description Language," doi:10.4230/DFU.Vol6.12191.85, is open access but was not read for this study.

The Lyapunov example uses an assumed exponent. No engine code was written and no hypothesis has been run.