UNIVERSITY OF MINNESOTA
Driven to Discover®

Game Engine Foundations

CSCI 5980: Game Engine Architecture

Evan Suma Rosenberg | CSCI 5980 | Spring 2026

Today

- The game development ecosystem (roles + organizations)
- What a **game** is (as a real-time simulation)
- What a **game engine** is (and why the boundary is blurry)
- How engine needs change across genres
- A quick survey of major engines
- High-level architecture: runtime engine + tools + asset pipeline

Game development is interdisciplinary

- Games are large software systems + large content production efforts
- Teams typically include:
  - programmers/engineers
  - artists + animators
  - designers + writers
  - producers + management
  - support staff (QA, IT, marketing, etc.)

Engineers / Programmers

- Build the engine, game code, and production tools
  - rendering, physics/collision, animation, audio
  - AI and gameplay systems
  - networking and online services
  - tools, pipelines, build systems
- Often split into:
  - runtime-facing work (frame budget, memory, platform)
  - tool-facing work (iteration speed, stability, UX)

Artists

- Create most of the "player-facing" content:
  - concept art, modeling, texturing, lighting
  - animation (hand-keyed, mocap, procedural)
  - VFX (particles), UI art, cinematics
- Increasing role of "technical artists":
  - shader graphs/materials
  - rigging constraints
  - tooling + automation to speed iteration

Designers

- Define *rules* and *player experience*
  - mechanics, goals, progression, balance
- Often specialized:
  - systems design (economy/abilities/AI tuning)
  - level design (layouts, pacing, encounters)
  - narrative design / writing
- Strong dependency on tools:
  - editors, scripting, debugging views, tuning sliders

Producers

- Own schedule, milestones, risk, and communication
- Coordinate across engineering/art/design
- Watch scope creep and integration risk
- Keep the build "green" and the team unblocked
- Some smaller studios don’t have producers at all

Support staff + organizations

- Support staff:
  - QA, IT, HR/admin, marketing, community, legal
  - Varies by studio size
- Publishers handle:
  - funding, marketing, manufacturing/distribution (historically)
- Studios:
  - independent, publisher-owned subsidiaries, or "first-party" (console makers)

What is a game?

- Broadly includes:
  - board/card/casino games, war games, children's play
- In entertainment computing, game often implies:
  - a virtual world + player control + audiovisual feedback
- One framing:
  - *fun* as learning and mastering patterns (increasing challenge)

Games as simulations

- Most video games are:
  - soft real-time
  - interactive
  - agent-based
  - temporal
- Practical consequence:
  - you run an approximate world model under deadlines
  - you choose simplifications aggressively

Real-time simulations

- Deadline examples:
  - display updates: >= 24 Hz (often 30/60 fps)
  - physics: may need ~120 Hz for stability
  - AI "thinking": often >= 1 Hz to avoid "stupid" behavior
  - audio buffer servicing: often very frequent to avoid glitches
- Real-time categories:
  - Soft: missed deadlines degrade experience, not catastrophic
  - Hard: missed deadline could mean severe injury or death of a human operator

Analytic vs numerical models

- Analytic (closed form) example:
  - falling rigid body under constant gravity
- Numerical model:
  - compute next state from current state and derivatives
  - run repeatedly over discrete timesteps
- Key idea:
  - unpredictable player input breaks global analytic modeling
  - games lean heavily on numerical iteration

The game loop as numerical simulation

- A main loop runs repeatedly
- Each iteration gives systems a chance to advance state:
  - input
  - AI / gameplay logic
  - physics
  - animation
  - audio
  - rendering
- You're continually computing state at $t + \Delta t$

What is a game engine?

- Term emerged mid-1990s (FPS era)
- Separation between:
  - reusable core systems (renderer, collision, audio, etc.)
  - game-specific content + rules
- Modding + reuse:
  - toolkits + scripting (e.g., Quake C)
  - engine licensing becomes a business model

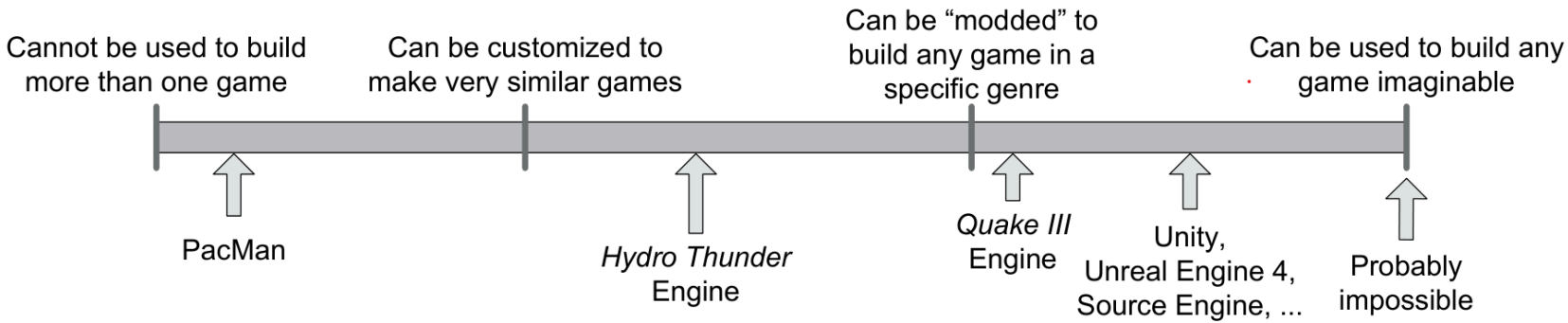
Engine vs game: blurry boundary

- Some engines separate cleanly; others barely do
- Example contrast:
  - renderer "knows how to draw an orc" (hard-coded)
  - "orc-ness is data" (materials, meshes, behaviors in data)
- Definitions shift during production as design solidifies

Data-driven architecture

- Data-driven designs improve reuse:
  - rules/behaviors/content in data, not code
- Hard-coded special cases reduce reuse:
  - difficult/impossible to build a different game later
- Practical takeaway:
  - "engine-ness" correlates with extensibility

Reusability is a gamut



- Not binary: engines fall along a spectrum
  - one-off game code (not reusable)
  - genre engines (build similar games)
  - general engines (broad reuse across genres)

Game engines aren’t one-size-fits-all

- The "hard parts" change by genre:
  - world representation + visibility
  - camera + locomotion constraints
  - AI scale + pathfinding needs
  - networking architecture
  - content workflows + tooling
  - performance bottlenecks

A game can always be made more impressive by fine-tuning the engine to the specific requirements and constraints of a particular game and/or hardware platform.

First-Person Shooters (FPS)

Figure 1.2. *Overwatch* by Blizzard Entertainment (Xbox One, PlayStation 4, Windows). (See Color Plate I.)

First-Person Shooters (FPS)

- Efficient rendering of large 3D virtual worlds
- A responsive camera control/aiming mechanic
- High-fidelity animations of the player’s virtual arms and weapons
- A forgiving player character motion and collision model, which often gives these games a "floaty" feel
- High-fidelity animations and artificial intelligence for the non-player characters (NPCs) = Small-scale online multiplayer capabilities

Platformers and Other Third-Person Games

Figure 1.3. *Jak II* by Naughty Dog (Jak, Daxter, Jak and Daxter, and Jak II © 2003, 2013/™ SIE. Created and developed by Naughty Dog, PlayStation 2.) (See Color Plate II.)

Platformers and Other Third-Person Games

- Character-centric traversal + animation fidelity
  - blending, IK, climb/vault, ledge logic
- Camera behavior is a major system
  - rotation typically controlled by human player
  - occlusion handling, framing, context sensitivity
- World interaction is broad:
  - puzzle-like environmental elements
  - triggers, scripted sequences, physics props

Platformers and Other Third-Person Games

Figure 1.5. *Tekken 3* by Namco (PlayStation). (See Color Plate IV.)

Fighting Games

- Typical characteristics
  - Two-player humanoid combat in ring/arena setting
  - Examples: Soul Calibur, Tekken 3
- Traditional technology focus
  - Rich set of fighting animations
  - Accurate hit detection
  - Complex user input system for button/joystick combinations
  - Crowds with relatively static backgrounds

Fighting Games

- Limited technical requirements historically
  - Small 3D world with camera centered on action
  - Little/no need for world subdivision or occlusion culling
  - Simple audio propagation models
- Modern fighting games advancements
  - High-definition character graphics
  - Realistic skin shaders with subsurface scattering and sweat effects
  - Photo-realistic lighting and particle effects
  - High-fidelity animations
  - Physics-based cloth and hair simulations

Racing Games

Figure 1.7. *Gran Turismo Sport* by Polyphony Digital (PlayStation 4). (See Color Plate VI.)

Racing Games

- Primary task: driving vehicles on tracks with multiple subgenres
  - Simulation racers ("sims"): realistic driving experience (Gran Turismo)
  - Arcade racers: over-the-top fun over realism (San Francisco Rush, Cruis'n USA, Hydro Thunder)
  - Street racing: tricked-out consumer vehicles (Need for Speed, Juiced)
  - Kart racing: popular characters in whacky vehicles (Mario Kart, Jak X, Freaky Flyers)
- Gameplay characteristics
  - Linear structure similar to older FPS games
  - Much faster travel speed than FPS
  - Long corridor-based or looped tracks with alternate routes and shortcuts

Racing Games

- Graphics focus
  - Detailed vehicles, tracks, and immediate surroundings
  - Kart racers also emphasize character rendering and animation
- Key technological features
  - Rendering tricks for distant backgrounds (2D cards for trees, hills, mountains)
  - Track divided into 2D regions ("sectors") for rendering optimization, visibility, AI pathfinding
  - Camera typically follows vehicle (third-person) or cockpit view (first-person)
  - Camera collision detection in tight spaces (tunnels, confined areas)

Strategy Games

- Genre defined by Dune II: The Building of a Dynasty (1992)
  - Examples: Warcraft, Command & Conquer, Age of Empires, Starcraft
  - Player deploys battle units strategically across large playing field
  - Typically displayed at oblique top-down viewing angle
- Subcategories include **turn-based** and **real-time strategy** (RTS)
- Viewing restrictions
  - Fixed or limited camera angles to see across large distances
  - Enables rendering engine optimizations

Strategy Games

Figure 1.8. *Age of Empires* by Ensemble Studios (Windows). (See Color Plate VII.)

Figure 1.9. *Total War: Warhammer 2* by Creative Assembly (Windows). (See Color Plate VIII.)

- Older games: grid-based world construction with orthographic projection (Age of Empires)
- Modern games: perspective projection and true 3D worlds, often with grid layouts for alignment (Total War: Warhammer 2)

Strategy Games

- Common technical characteristics
  - Low-resolution units to support large numbers on-screen simultaneously
  - Height-field terrain as primary canvas
  - Player can build structures on terrain in addition to deploying forces
  - User interaction via single-click and area-based unit selection plus menus/toolbars for commands

Massively Multiplayer Online Games (MMOG)

Figure 1.10. *World of Warcraft* by Blizzard Entertainment (Windows, MacOS). (See Color Plate IX.)

Massively Multiplayer Online Games (MMOG)

- Support thousands to hundreds of thousands of simultaneous players
  - Large, persistent virtual world (state persists beyond individual gameplay sessions)
  - Examples: Guild Wars 2, EverQuest, World of Warcraft, Star Wars Galaxies
- Subcategories
  - MMORPG (MMO role-playing games)
  - MMORTS (MMO real-time strategy)
  - MMOFPS (MMO first-person shooters)

Massively Multiplayer Online Games (MMOG)

- Server infrastructure
  - Powerful server battery maintains authoritative game world state
  - Manages user sign-in/out, inter-user chat, voice-over-IP (VoIP)
  - Handles billing and micro-transactions (primary revenue source)
- Technical characteristics and subgenres
  - Graphics fidelity lower than non-MMO counterparts due to huge world sizes and player counts
  - "Shared world" games (Destiny 2): on-the-fly matchmaking, limited player interactions, higher graphics fidelity
  - Battle royale subgenre (PUBG): ~100 players in "last man standing" survival gameplay, blurs line between regular multiplayer and MMO

Player-Authored Content

Figure 1.14. *Minecraft* by Markus "Notch" Persson / Mojang AB (Windows, MacOS, Xbox 360, PlayStation 3, PlayStation Vita, iOS). (See Color Plate XIII.)

Player-Authored Content

- Trend toward collaborative, player-created game experiences driven by social media
- LittleBigPlanet series
  - Technically puzzle platformers
  - Notable feature: players create, publish, and share custom game worlds
- Minecraft: most popular player-created content game
  - Simple cubic voxel-like construction with low-resolution textures
  - World elements: solid blocks or containers (torches, anvils, signs, fences)
  - Populated with player characters, animals (chickens, pigs), and "mobs" (villagers, zombies, creepers)

VR / AR / XR game engines

Superhot VR

VR / AR / XR game engines

- Stereo rendering + head tracking
- Very low latency requirements
- Input devices diversify:
  - tracked controllers, hand tracking, gaze
- Comfort constraints:
  - locomotion + motion sickness mitigation
- AR/MR adds:
  - world alignment, occlusion, sensing uncertainty

Game engine survey

- Real engines embody tradeoffs:
  - era, hardware, business model, team, genre assumptions
- Useful for:
  - vocabulary (common subsystems)
  - architectural patterns that repeatedly work
  - recognizing "why this engine feels like this"

Game engine survey

- Early licensed FPS lineage (id / Quake family)
- Large commercial general engines:
  - Unreal, Unity
- Widely used proprietary AAA engines:
  - Source, Frostbite, RAGE, CRYENGINE, etc.
- Open-source and niche engines exist (with tradeoffs)
  - Godot, Babylon.js, etc.

Quake Family of Engines

- Castle Wolfenstein 3D (1992) - first 3D FPS by id Software, pioneered new direction for gaming
- Quake family: Doom, Quake, Quake II, Quake III - share similar architecture
- Influenced major engines including Valve's Source engine (Half-Life)
- Source code freely available on GitHub - well-architected, written in C
- Educational value: Run under debugger with real game assets to analyze engine architecture

Unreal Engine

- Epic Games launched Unreal (1998)
  - major competitor to Quake
- Unreal Engine 2 (UE2) powered UT2004
  - widely used for mods, university projects, and commercial games
- Unreal Engine 4 (UE4) features industry-leading tools
  - graphical shader creation and Blueprints visual scripting
- Notable UE4 games: Rime, Genesis: Alpha One, Crackdown 3

Unreal Engine

- Known for extensive feature set and cohesive, easy-to-use tools
- Versatile platform for 3D first/third-person games and other genres
- Unreal Developer Network (UDN) provides documentation
  - some free, full access for licensees
- UE4 now accessible via low monthly subscription + profit sharing
  - viable for indie studios

Unity

- Powerful cross-platform game development and runtime engine
- Extensive platform support
  - mobile (iOS, Android), consoles (Xbox, PlayStation, Nintendo), desktop (Windows, Mac, Linux), VR systems, and TV boxes
- Primary design goals: ease of development and cross-platform deployment
- Integrated editor with real-time preview capabilities in editor or on target hardware

Unity

- Comprehensive toolset
  - performance analysis, asset conditioning pipeline, platform-specific optimization
- Scripting support
  - JavaScript, C#, or Boo
- Advanced features
  - animation retargeting system and networked multiplayer support
- Notable games: Deus Ex: The Fall, Hollow Knight, Cuphead

The Half-Life Source Engine

- Powers Half-Life 2, HL2: Episode One & Two, Team Fortress 2, and Portal (The Orange Box)
- High-quality engine rivaling Unreal Engine 4 in graphics and toolset
- Commercially offered under a "source available" license

DICE’s Frostbite

- Originated from Battlefield Bad Company development (2006)
- Most widely adopted engine within Electronic Arts (EA)
- Powers major EA franchises: Mass Effect, Battlefield, Need for Speed, Dragon Age, Star Wars Battlefront II
- Features:
  - FrostEd (unified asset creation tool)
  - Backend Services (tools pipeline)
- Proprietary (unavailable outside EA)

Rockstar Advanced Game Engine (RAGE)

- Drives Grand Theft Auto V and other Rockstar titles
- Developed by RAGE Technology Group (Rockstar San Diego division)
- Multi-platform support: PlayStation 4, Xbox One, PS3, Xbox 360, Wii, Windows, MacOS
- Notable games: GTA IV, Red Dead Redemption, Max Payne 3
- Proprietary engine for Rockstar’s internal studios

CRYENGINE

- Originally developed as tech demo for NVIDIA, evolved into Far Cry
- Powers multiple titles: Crysis, Codename Kingdoms, Ryse: Son of Rome, Everyone's Gone to the Rapture
- Latest version: CRYENGINE V
- Powerful asset-creation tools and feature-rich runtime engine with high-quality real-time graphics
- Multi-platform support: Xbox One, Xbox 360, PlayStation 4, PS3, Wii U, Linux, iOS, Android

Licensing an engine vs building in-house

- "Engine choice" is a systems decision
- Consider:
  - team expertise + schedule constraints
  - tooling needs + content workflow
  - platform targets + performance requirements
  - extensibility and long-term maintenance costs
- Middleware exists somewhere between the two extremes
  - buy subsystems (physics/audio/UI), integrate carefully

Open Source Engines

- Built by amateur and professional developers, provided online for free with available source code
- Licensing: typically GNU Public License (GPL) or Lesser GPL (LGPL)
  - GPL requires code sharing
  - LGPL allows proprietary use
- Large number available online - quality varies significantly

Object-Oriented Graphics Rendering Engine (OGRE)

- Well-architected, easy-to-learn 3D rendering engine
- Features
  - advanced lighting/shadows
  - skeletal animation
  - 2D overlay system for HUDs/GUIs
  - post-processing effects
- Not a full game engine but provides foundational components

Godot 4

- Free and open-source game engine under MIT license - fully permissive for commercial use
- Dual rendering backends: Vulkan-based Forward+ for high-end graphics, and mobile/web-optimized renderer
- Cross-platform deployment: Windows, macOS, Linux, iOS, Android, and web (HTML5)
- Comprehensive 2D and 3D capabilities in single engine
- Built-in scripting: GDScript (Python-like), C#, and C++ via GDExtension

Godot 4

- Node-based scene system for intuitive game object hierarchy and composition
- Integrated editor with visual shader editor, animation tools, and tilemap system
- Hardware support and interaction toolkit for VR/AR/XR
- Growing community and ecosystem - increasingly popular for indie game development
- Notable games: used in various indie titles gaining recognition in Steam and mobile markets

Other Notable Open Source Engines

- Panda3D: Python-based scripting engine for rapid prototyping of 3D games and virtual worlds
- Yake: game engine built on top of OGRE
- Crystal Space: extensible modular architecture
- Torque and Irrlicht: well-known open-source options
- Lumberyard: free cross-platform engine by Amazon
  - source available, based on CRYENGINE - not technically open-source

Open-source engine tradeoffs

- Pros:
  - inspect/modify source, avoid licensing cost
- Cons:
  - integration burden
  - smaller tooling ecosystems
  - support/maintenance becomes your job

2D engines for non-programmers

- Aimed at ease-of-use:
  - create games via GUI rather than programming
- Examples:
  - Multimedia Fusion 2
  - Game Salad Creator
  - MIT’s Scratch (also a learning tool for programming concepts)

Key Takeaways

- Game engines separate reusable systems from game-specific content
  - Boundary between "engine" and "game" is often blurry
  - Data-driven architectures improve reusability across projects
- Engine requirements vary dramatically by genre
  - FPS: efficient large-world rendering, responsive camera/aiming
  - Platformers: character animation fidelity, camera behavior systems
  - Racing: track optimization, distant background rendering tricks
  - Strategy: low-res units at scale, height-field terrain, grid-based layouts
  - MMO: server infrastructure, persistent worlds, lower graphics fidelity

Key Takeaways

- Modern engine landscape offers many options
  - Commercial general-purpose: Unreal, Unity (cross-platform, rich toolsets)
  - Proprietary AAA: Frostbite, RAGE, CRYENGINE (genre-optimized, internal use)
  - Open-source: Godot, OGRE (free, source access, integration burden)
- Engine choice is a systems decision
  - Consider team expertise, tooling needs, platform targets, performance requirements
  - Licensing vs building in-house vs middleware integration

Participation Exercise

- Choose a video game that wasn't mentioned in this lecture. Then, research what engine was used to develop it.
- Write a brief description of why the developers may have chosen that option (one paragraph is sufficient).
  - Was the engine licensed or developed in-house?
  - What tradeoffs do you think they might have considered?
- Submit your answer on Canvas within one week.

Questions?