Games Development

Chapter Three - Games Engines

Ahmed A. Qazzaz

Nov. 2024

Based on

Collection of Source online and books

references at last slide

Game Engines

- Introduction and Definition
- Core Components of Game Engine
- Types of Game Engines
- How Game Engines Work
- Popular Game Engines
- Criteria for Choosing a Game Engine
- Mobile-First Feature in Game Engine
- Building a Game Without a Game Engine
- Trends and Future of Game Engines

Introduction and Definition

- Definition of Game Engines
- History of Game Engines
- Importance of Game Engines

Introduction

Definition of Game Engine

- A software framework that provides tools and systems for game development.
- Separates core software components (e.g., rendering, physics) from game-specific assets and rules.
- Enables developers to reuse key components for different games.
- Differentiated by being extensible and adaptable for various game genres.
- Example: General-purpose engines like Unity or Unreal allow broader use, while specialized engines are fine-tuned for specific genres.

Introduction

History of Game Engine

- Emerged in the mid-1990s, with first-person shooters like Doom by id Software.
- Doom introduced modular architecture, separating core software from assets and rules.
- Led to the creation of "modding communities" with toolkits for modifying existing games.
- Late 1990s saw engines like Quake III Arena and Unreal, designed for reusability via scripting languages (e.g., Quake C).
- Today, engines are key in game development, enabling extensive reuse and licensing as a revenue model.

Introduction

Importance of Game Engine

- Simplifies development by providing pre-built systems (e.g., rendering, physics, scripting).
- Enables rapid prototyping and iteration.
- Reduces costs and time compared to building custom systems from scratch.
- Facilitates cross-platform development (e.g., PC, console, mobile).
- Supports the gaming ecosystem, including modding, community growth, and tool standardization.

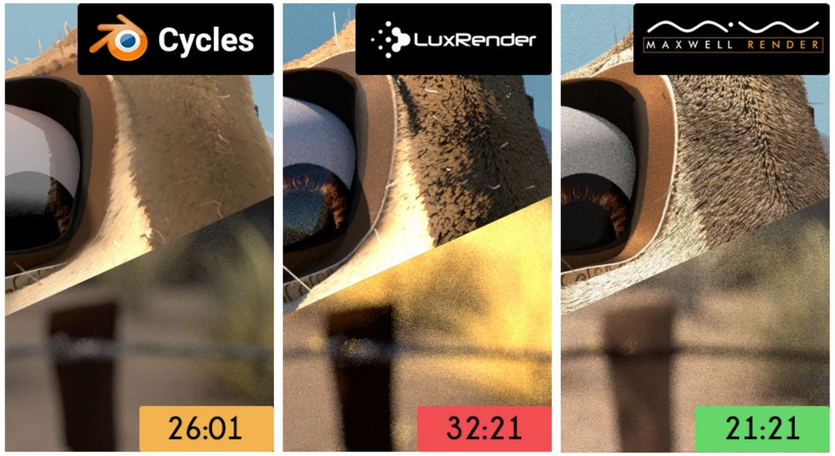
Core Components

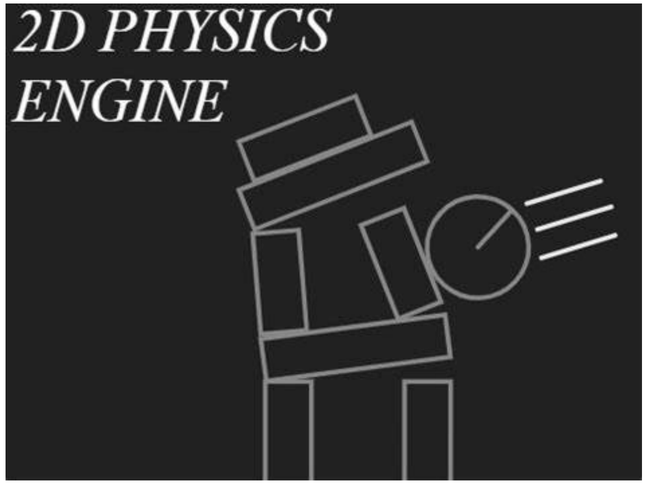
- Rendering Engine
- Physics Engine
- Audio Engine
- Input Handling
- Scripting System
- Artificial Intelligence
- Networking
- Resource Management
- Battery Optimization
- Mobile Features
- Platform Independent
- OS Utilities

Core Components of a Game Engine

PART I

- **Rendering Engine**
  - Responsible for rendering 2D and 3D graphics.
  - Converts game data into visual representations.
  - Supports shaders, lighting, and textures.
  - Examples: OpenGL, DirectX, Vulkan.
- **Physics Engine**
  - Simulates real-world physics (gravity, collision, friction).
  - Ensures objects interact realistically.
  - Used for rigid bodies, soft bodies, and particle systems.
  - Examples: NVIDIA PhysX, Havok, Bullet.

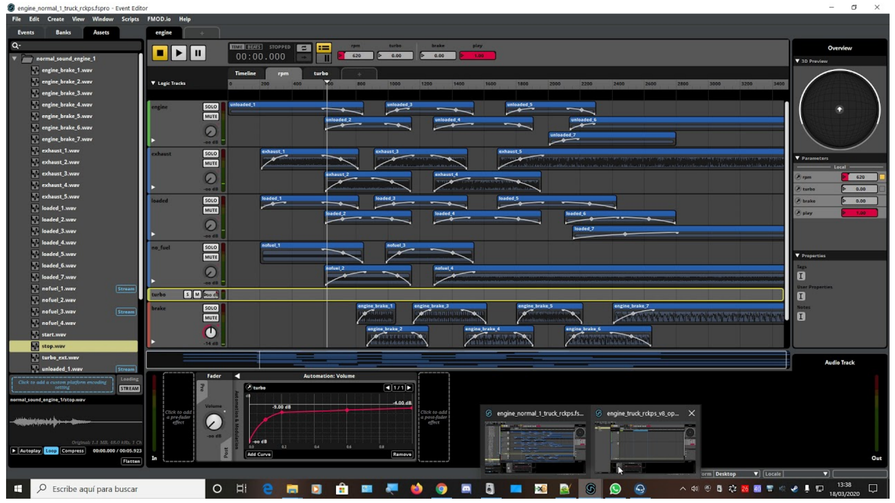




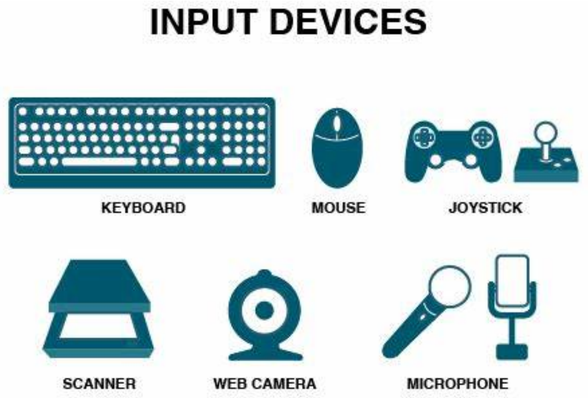
Core Components of a Game Engine

PART II

- **Audio Engine**
  - Handles sound effects, background music, and voice-overs.
  - Supports 2D and 3D audio spatialization.
  - Includes mixing, playback, and volume control.
  - Examples: FMOD, OpenAL, Wwise.



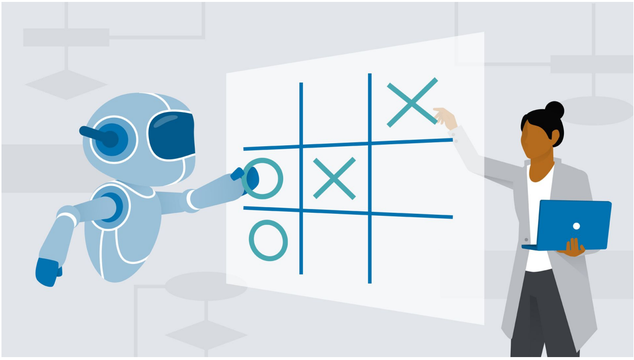
- **Input Handling**
  - Captures player inputs (keyboard, mouse, gamepad, touch).
  - Handles gestures, accelerometers, and gyroscopes for mobile devices.
  - Interfaces with game logic to react to user commands



Core Components of a Game Engine

PART III

- **Scripting System**
  - Provides a flexible way to program game logic.
  - Commonly uses Lua, Python, C#, or custom scripting languages.
  - Allows non-programmers to modify gameplay easily.
  - Enables rapid prototyping and iteration.
- **Artificial Intelligence (AI)**
  - Powers NPC behavior, decision-making, and pathfinding.
  - Algorithms include A* pathfinding, finite state machines, and neural networks.
  - Enables dynamic, adaptive game worlds.



Core Components of a Game Engine

PART IV

- **Networking**
  - Provides support for multiplayer gaming.
  - Includes features like matchmaking, syncing game states, and real-time data transfer.
  - Common protocols: TCP/IP, UDP.
- **Resource Management**
  - Optimizes memory and CPU usage by managing assets (textures, sounds, etc.).
  - Implements streaming for large worlds to load content dynamically.
  - Ensures efficient handling of limited hardware resources.

Core Components of a Game Engine

PART V

- **Battery Optimization**
  - Mobile engines prioritize minimizing power consumption to extend device battery life.
  - Reducing frame rates when not necessary.
  - Using adaptive quality settings based on hardware.
  - Managing background tasks efficiently.
  - Dynamic resolution scaling.
  - Reducing draw calls and using batching for rendering.
  - Optimized physics and AI updates.

Core Components of a Game Engine

PART VI

- **Mobile-Specific Features**
  - Built-in support for push notifications to engage users outside the app.
  - **In-App Purchases (IAP) systems for monetization:**
    - Integration with Google Play and Apple Store APIs.
    - Handling secure transactions and subscriptions.
  - **Ads Integration:**
    - Supports various ad networks (e.g., AdMob, Unity Ads).
    - Seamless integration of rewarded videos, banners, and interstitial ads.
  - **Other tools for mobile:**
    - Gesture recognition.
    - Support for sensors like accelerometers and gyroscopes.

Core Components of a Game Engine

Platform Independent Layer

- **Platform Independent Layer**
  - Abstracts platform-specific features to provide a consistent interface for the game engine.
  - Handles cross-platform rendering (e.g., using APIs like OpenGL, Vulkan, or DirectX).
  - Provides unified input handling for different devices (e.g., controllers, touchscreens).
  - Standardizes file system access and resource loading.
  - Example: A game engine uses this layer to allow the same game to run on Windows, macOS, Linux, iOS, and Android without changes to core game logic.

Core Components of a Game Engine

OS Utility Layer

- **OS Utility Layer**
  - Interfaces directly with the operating system to access hardware and system-level features.
  - Manages low-level system calls and resources.
  - Accesses OS-specific utilities like timers, threads, and networking APIs.
  - Handles tasks such as memory allocation, file I/O, and audio drivers.
  - Example: On Windows, this layer might use Win32 APIs, while on macOS, it would interface with Core Foundation or Metal.