Game Engine Development and Architecture

Grayson von Goetz and Matthew Lee

Fall 2024

Course Overview

This course focuses on the fundamentals of game engine architecture. It explores the core systems that power modern games and provides insight into how these systems work together to create interactive and immersive experiences.

Topics Covered

1. **Introduction to Game Engines**
    - Overview of game engine components, including rendering, physics, input handling, and audio.
    - Key differences between game engines and standalone games.
    - The importance of data-driven architecture for reusability and scalability.
2. **Rendering Systems**
    - Study of 2D and 3D rendering pipelines.
    - Optimizations like frustum culling, occlusion culling, and spatial partitioning.
    - Introduction to shaders, lighting models, and material systems.
3. **Physics and Collision Detection**
    - Core concepts of rigid body dynamics, particle systems, and collision handling.
    - Integration of physics libraries and creating custom physics engines.
    - Techniques for resolving collisions and maintaining real-time performance.
4. **Entity-Component-System (ECS) Architecture**
    - Understanding ECS as a modular and scalable framework for game objects.
    - Separation of concerns: entities (game objects), components (data), and systems (logic).
    - Implementing ECS to improve performance and reusability.

5. **Input and Event Handling**
    - Mapping hardware inputs (keyboard, mouse, controller) to in-game actions.
    - Designing custom input systems for flexibility and extensibility.
6. **Audio Systems**
    - Basics of sound processing and integration into games.
    - 2D and 3D audio spatialization for immersive soundscapes.
7. **Tools and Asset Management**
    - Building asset pipelines for models, textures, and animations.
    - Optimization strategies for loading, managing, and utilizing assets efficiently.
8. **Networking and Multiplayer**
    - Architectures for real-time multiplayer games (client-server and peer-to-peer).
    - Synchronizing game state across clients and managing latency.
9. **Scripting Systems**
    - Embedding scripting languages (e.g., Lua, Python) to define game logic dynamically.
    - Benefits of scripting for rapid prototyping and customization.
10. **Parallelism and Optimization**
    - Techniques for parallel processing and multithreading.
    - Profiling and debugging tools for identifying performance bottlenecks.

Learning Outcomes

- Develop a solid understanding of the key components of a game engine.
- Gain hands-on experience designing and implementing modular systems.
- Evaluate and compare game engines for different genres and use cases.
- Learn optimization techniques to ensure smooth and responsive gameplay.