

Liverpool John Moores University

Title: CONSOLE GAME DEVELOPMENT
Status: Definitive
Code: **7119COMP** (121342)
Version Start Date: 01-08-2021

Owning School/Faculty: Computer Science and Mathematics
Teaching School/Faculty: Computer Science and Mathematics

Team	Leader
Chris Carter	Y
Sud Sudirman	

Academic Level: FHEQ7 **Credit Value:** 20 **Total Delivered Hours:** 57
Total Learning Hours: 200 **Private Study:** 143

Delivery Options

Course typically offered: Semester 2

Component	Contact Hours
Lecture	22
Practical	33

Grading Basis: 50 %

Assessment Details

Category	Short Description	Description	Weighting (%)	Exam Duration
Artefacts	AS1	Developing a Game Application using a Console Development Kit.	50	
Exam	AS2	Examination.	50	2

Aims

To provide an overview of the hardware, systems architecture and ecosystem of a modern games console and its associated development kit.

To explain the processing models, processor taxonomies and parallel and threaded computation models that exist on modern games consoles.

To understand the low-level processes and optimise the memory layout and

*processing tasks of the core game subsystems on both CPU and GPU.
To adapt programmatic game and real-time rendering algorithms and tailor them to a console environment.*

Learning Outcomes

After completing the module the student should be able to:

- 1 Produce console game applications using game SDKs.
- 2 Use low-level APIs, middleware, and game engines targetting a given console ecosystem in order to solve game software development problems on a games console.
- 3 Critically evaluate the hardware architecture and processing models employed in modern game consoles.
- 4 Explain the theories of parallel processing, GPU processing and memory management strategies in relation to a modern console architecture.

Learning Outcomes of Assessments

The assessment item list is assessed via the learning outcomes listed:

Console Game Application Examination	1	2
	3	4

Outline Syllabus

*Evolution of game console hardware and architecture.
Types of game platforms.
Game platform languages.
Game platform constraints
Task-based decomposition
Using Game Engines and Middleware targetting Game Consoles.
Using Close-To-Metal Console SDKs and APIs
Using Console Profiling and Debugging Tools.
Game Console Platform Support and OS Ecosystems.
Assembly/machine language programming; Instruction formats; Addressing modes; I/O and interrupts.
Shared memory multiprocessors/multicore organization.
Introduction to SIMD vs. MIMD and the Flynn Taxonomy.
Review of physical memory and memory management hardware.
Memory hierarchy: importance of temporal and spatial locality.
Main memory organization and operations.
Buses: bus protocols, arbitration, direct-memory access (DMA).
Instruction pipelining.
Introduction to instruction-level parallelism (ILP).
Example SIMD and MIMD instruction sets and architectures.
Symmetric multiprocessing (SMP).
Shared multiprocessor memory systems and memory consistency.*

Multiprocessor cache coherence.
Multicore architectures and hardware support for synchronization.
Performance figures of merit.
Workloads and representative benchmarks, and methods of collecting and analyzing performance figures of merit.
CPI (Cycles per Instruction) equation as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
Amdahl's Law.
Managing Latencies: memory vs. disk latencies.
Caches and the effects of spatial and temporal locality on performance in processors and systems.
Introduction into the processor memory hierarchy and the formula for average memory access time.
Superscalar architecture.
Vector processors and GPUs.
Hardware support for multithreading.
Memory/disk management requirements in a real-time environment.
Why system performance needs to be evaluated and what is to be evaluated.
Implementation strategies such as threads.
Data-parallel decomposition.
Shared Memory.
Maintaining spatial locality.
Sequential vs. parallel processing.
Parallel programming vs. concurrent programming.
Request parallelism vs. Task parallelism.
Re-Defining Algorithms for Games using Console Architecture Awareness.

Learning Activities

Lectures – to deliver the theoretical concepts on console game development with demonstration.

Practical – Tutor-led practical session with hands-on work in the computer laboratory.

Further exercises – additional exercises for students to work on in their own time.

Directed learning – provides additional reading to enable practical work to be completed.

Learning materials can be accessed digitally via University Virtual Learning Environment (VLE).

Notes

This module will explain the key architectural and system-level design choices and CPU/GPU processing models used in Triple-A game console design and software development.

Students will be exposed to the development kits of a modern game console and will be learning to use the low-level libraries, middleware and game engines of a console

in order to build and deploy a console game. Students will learn to collaborate as a development team within simulated industry conditions. The module will cover both algorithms and technologies specific to console development and key principles of processor taxonomies, multi-processing programming, stream and vector programming and memory management techniques that are required to target a constrained architecture that a game console provides.