MODULE GAME DEVELOPMENT & MIXED REALITY 17,5 ECTS

BACHELOR FALLTERM

Game Development 1

5 ECTS

Course Content:

C++ according to current standards (object orientation, polymorphism, memory management, templates, STL, etc.); Memory areas (stack, heap, etc.); Runtime/memory debugging; Modern development environments (e.g., Visual Studio); Static/Dynamic Libraries; Basic game engine architectures (hierarchical, component-based); Design Patterns in game development (Singleton, Observer, Factory, Composite/Aggregate, etc.); Input/output methods, Graphical User Interface, and Accessibility; Event systems and callbacks; Basic artificial intelligence in games (basic Steering Behavior, Decision Trees, State Machines); Pathfinding (Dijkstra, A*); Physics simulation of rigid 2D bodies; Practical exercises and implementation of a simple game engine (C++).

Learning Outcomes: Students will be able to:

- Implement programs in a development environment using the C++ programming language, analyze program flow structuredly with the help of software tools (runtime, memory), and estimate the runtime of data structures.
- Explain basic algorithms and design patterns in the context of game development in their own words and implement basic functionalities modularly using them (e.g., Artificial Intelligence, simple physics, Event System, Graphical User Interface).
- Independently develop simple games in C++ and create, implement, and systematically debug the necessary modular game engine architecture. They will be able to analyze and fix errors and use version control systems (e.g., Git) in their development process.

Computer Graphics 1

3 ECTS

Course Content:

Applications of linear algebra in computer graphics (dot product for shading, cross product for normal vectors, affine transformations, orthographic/perspective projection, etc.); Coordinate spaces and transformations (object, world, NDC, camera, homogeneous coordinates, etc.); Ray tracing; Rasterization; Render pipeline (e.g., OpenGL); Depth buffer (principle, z-fighting, etc.); Geometry definitions (polygon models, Constructive Solid Geometry, volume data, implicit geometries, etc.); Simple real-time reflection models (Lambert, Phong, Blinn-Phong, etc.); Basics of Texture Mapping (2D/3D textures, Bump/Normal Mapping); Alpha-blending; Basics of shader programming (Vertex Shader, Fragment Shader); Debugging methods; Practical exercises using a modern graphics interface (e.g., OpenGL/GLSL).

Learning Outcomes: Students will be able to:

- Explain requirements for real-time rendering techniques, the difference between Ray Tracing and Rasterization techniques, and the structure of modern render pipelines in their own words.
- Programmatically create simple 3D worlds (e.g., OpenGL) by defining textured polygon geometry, transforming it through mathematical operations, and safely converting between different coordinate spaces.
- Create simple shader programs (e.g., GLSL).
- Systematically debug simple graphics applications to analyze and fix errors.

Computer Vision

4 ECTS

Course Content:

Image Filtering (cross-correlation, convolution, Box-Filter, Mean/Median-Filter, Gauss-Filter, Sobel-Filter, etc.); Fourier Transformation and digital image processing; High-pass/Low-pass filters; Sampling Theorem; Aliasing; Edge detection (e.g., Canny Edge, Sobel); Feature Descriptors (e.g., Harris Corner, SIFT); Feature Matching; Camera calibration (Pinhole camera, intrinsic parameters, radial/tangential distortion, etc.); Pose estimation; Epipolar geometry; Depth from Stereo; Structure from Motion and SLAM; Practical exercises using computer vision libraries (e.g., OpenCV).

Learning Outcomes: Students will be able to:

- Use simple filter operations of digital image processing and explain the relationship to the frequency domain in their own words.
- Extract image features such as edges and corners and algorithmically detect them in different images using descriptors.
- Explain the mathematical model of projection in a pinhole camera in their own words, calibrate real cameras, and determine poses (position, rotation) relative to known markers in space.
- Explain basic concepts of visual tracking systems in their own words and assess their limitations, optimal working conditions, and areas of application.
- Explain the basic concepts for determining depth values of a scene and the continuous position determination of cameras through tracking in their own words.

Software Design Patterns

2.5 ECTS

Course Content:

Introduction to object-oriented software design; Basics of UML notation; Design Patterns: Creational Patterns (e.g., Factory, Singleton), Structural Patterns (Adapter, Composite, Decorator, Facade Proxy), Behavioral Patterns (Iterator, Mediator, Observer, State, Strategy) and application examples; Basics of Refactoring; Practical exercises applying Design Patterns.

Learning Outcomes: Students will be able to:

- Apply their knowledge of Design Patterns and their advantages/disadvantages to sub-problems to modularize software architecture.
- Optimize parts of software architectures using Design Patterns (Refactoring).
- Understand basic software architectures using UML and communicate in professional language.
- Perform improvement steps in the code without introducing new functionality (Refactoring).

Multimedia Project 2 (MMP2a)

3 ECTS

Course Content:

Web or game project conducted in teams of two students from the MultiMediaTechnology program; independent selection and execution of the project; independent preparation of the project; software development during the studio week; application of software project management; use of git for teamwork; conducting simple user tests; topics fall within the fields of Web & Mobile Development or Game Development & Mixed Reality.

Learning Outcomes: Students will be able to:

o Develop a project idea for a simple programming project within technical constraints as a team. They develop a concept with reference to the target audience.

o Plan and program a simple software project as a team. They carry out the implementation within a given timeframe.

- Use the version control system git to manage the source code as a team.
- Find and utilize existing software packages (e.g., libraries, packages) and integrate them into their own software project.
- Use an issue tracker to prioritize features and bugs. Based on this, they make decisions about which issues need to be addressed to create a functional project within the given time.
- Apply methods and tools for team and time management independently, individually adapted, and situation-specifically for their project.
- Understand the group dynamic aspects in teamwork and perceive them in their own team-work, question, analyze, and consciously shape them.
- Present the finished project to instructors and students and conduct user tests. They describe the project in text, images, and video on the portfolio website of the program.