UNIVERSITY OF FLORIDA

Digital Worlds Institute — College of the Arts

DIG4930

GAME ENGINE DEVELOPMENT

Design, build, and deploy a 3D game engine from the ground up

---

Screenshot from a game made in this class with a student-built game engine.

Course Syllabus

Instructor

Dr. Angelos Barmpoutis

Professor of Digital Arts and Sciences

CONTENTS

A WORD FROM THE INSTRUCTOR

Hello students! I'm delighted to have you in this course and excited to begin this journey with you. This course asks students to build their own game engine rather than treat Unity or Unreal as a black box. Over the semester, you'll design and build your own 3D game engine from the ground up and then use it to create and deploy a game across multiple platforms, including mobile devices and extended reality systems.

Along the way, we'll explore the core building blocks of modern game engines: how 3D objects are represented and transformed, how scenes are structured and animated, how physics and rendering systems work, and how shaders, input controllers, and resource management come together to create interactive experiences. By the end of the course, you'll not only understand how game engines work, you'll have built one yourself.

I'm looking forward to seeing what you create. Welcome, and enjoy the process!

— Dr. Angelos Barmpoutis

COURSE DESCRIPTION

In this course, students learn how to develop their own 3D game engine from scratch and use it to deploy a game in various gaming environments such as phones, tablets, and extended reality headsets. Topics covered include 3D object encoding, 3D transformations, hierarchical scene structuring, the animation loop, physics engines, rendering with shaders, input controller event handling, output vibration (haptic) triggering, game resource management, and deployment.

By the end of this course, students will be able to:

- Understand the core components of a 3D game engine
- Develop custom game engines
- Deploy fully functional games on mobile and extended reality devices using the engines they created

REQUIRED SOFTWARE & TOOLS

This course uses two primary development environments across the semester. Students should install each tool as it is introduced in the corresponding module.

Unity During this course, we reference an existing game engine so that we can better understand its complexity and structure, and attempt to replicate some of its components. Unity is a popular 3D game engine that offers a free license for instructional purposes. Used in Modules 1–4 for procedural mesh generation, OBJ/glTF parsing, and 3D transformations.

Android Studio We need a development environment that allows us to build our own game engine and test it across a range of gaming platforms, including phones, tablets, and extended reality devices. Android Studio is a free, professional IDE that natively builds applications for Android-based devices, including Meta Quest headsets and many others. Used from Module 5 onward to build a custom native Android game engine, including GLSL shaders, OpenXR input, shadow mapping, physics, and networking.

Meta OpenXR SDK (optional) Students who want to build assignments specifically for Meta Quest headsets should download the latest SDK from Meta and use it within Android Studio: github.com/meta-quest/Meta-OpenXR-SDK

Additional libraries Bullet Physics / jBullet, OkHttp WebSockets, and Google MediaPipe are introduced in their respective modules.

TEXTBOOK

Required Textbook This class does not have a required textbook; the instructor will provide notes covering each module's content.

Supplemental Textbook Although not used as a course textbook, students who want a supplemental resource outside the class material are encouraged to consider: Pouhela, F. (2024). *3D Game Engine Development*. Independently published. ISBN 9798878081344.

LEARNING ACTIVITIES & ASSIGNMENTS

There are weekly programming assignments in which students practice the content covered in that week's module. Several of these skills are also combined into a complete game engine that students gradually build across the semester. Toward the end of the semester, students use their own game engine to develop a small game of their choosing.

Throughout the semester, students also participate in blog-like discussion boards with their classmates on topics related to the various components of game engines.

The Semester-Long Project Arc

Beyond the weekly exercises, a single project threads through the second half of the course: students propose their game engine project in Module 7, submit a progress update in each subsequent module (Modules 8–12), and present and submit the finished engine and game in Module 13. This staged structure gives students recurring, low-stakes checkpoints and regular feedback well before the final deadline, rather than a single high-stakes deliverable at the end of the term.

GRADING

Grades are primarily composed of weekly coding assignments, with the remainder allocated to attendance and participation activities such as discussion posts and quizzes.

- Weekly coding assignments — 90%
- Attendance & participation (discussion posts, quizzes) — 10%

Grading Scale

Letter Grade	Percentage
A	94–100%
A–	90–93%
B+	87–89%
B	83–86%
B–	80–82%
C+	77–79%
C	73–76%
C–	70–72%
D+	67–69%
D	63–66%
D–	60–62%
E	59% and below

Additional information on grading policies is available on the university's Syllabus Policies page.

STUDENT FEEDBACK SURVEYS

Each semester, students complete two anonymous surveys to give the instructor feedback on elements of the course such as instructor presence, lectures, and assignment quality. Students are graded for participation in these surveys, not for their answers.

Mid-Semester Course Survey Takes place during Week 6 and acts as a prerequisite for Module 6 — it must be completed before moving forward in the course.

End-of-Semester Course Survey Takes place during Week 14.

COURSE ACCESS & TECHNICAL NOTES

- To gain access to the content in this course, the Course Orientation Module (including its assignments) must be completed in its entirety.
- Students have 16 weeks from the day of enrollment to complete this course.
- Disable your pop-up blocker to avoid content being blocked in the learning management system.
- Students new to the LMS should review the platform’s quick-start guide and student help resources.

COURSE SCHEDULE OVERVIEW

The course is organized into an orientation week followed by thirteen modules, progressing from foundational concepts (geometry, transformations, and file formats) through rendering and shading, to input, physics, networking, and applied case studies, culminating in a final project and presentations.

Week	Module	Topic	Deliverable
0	**Orientation**	Welcome, syllabus, LMS structure, and required software	*Course Orientation Quiz*
1	**Module 1**	Introduction to Game Engines	*Exploring the Components of a Game Engine*
2	**Module 2**	3D Geometric Representation	*Object Maker class*
3	**Module 3**	3D Object File Formats	*OBJ Parser*
4	**Module 4**	3D Transformations and Camera Systems	*Scene Hierarchy implementation*
5	**Module 5**	Introduction to Shaders	*GLSL shader assignment*
6	**Module 6**	Advanced Shaders	*GLSL shader edits + Shader Techniques presentation*
7	**Module 7**	Shadow Rendering	*Shadow mapping assignment + Progress Update 5*
8	**Module 8**	Input Systems: Controllers, Haptics, and Touch Interfaces	*OpenXR input assignment + Final Project Proposal*
9	**Module 9**	Physics Engines	*Physics assignment + Progress Update*
10	**Module 10**	Networking for Multi-Player Gaming	*Networking assignment + Progress Update 2*
11	**Module 11**	Special Topic: Building Endless Runner Games	*Endless runner assignment + Progress Update 3*
12	**Module 12**	Natural User Interfaces and Real-Time Body Tracking	*Body-tracking assignment + Progress Update 4*
13	**Module 13**	Final Presentations	*Final Presentation + Final Project Submission*

COURSE POLICIES

Attendance Policy

Students may only participate in classes if they are registered officially or approved to audit with evidence of having paid audit fees. Students are responsible for satisfying all academic objectives as defined by the instructor, and absences count from the first class meeting.

Excused and Unexcused Absences

Acceptable reasons for absence or failure to engage in class include illness; Title IX–related situations; serious accidents or emergencies affecting the student, roommates, or family; special curricular requirements (e.g., judging trips, field trips, professional conferences); military obligation; severe weather; religious holidays; participation in official university activities; and court-imposed legal obligations. Other reasons may be approved at the instructor's discretion.

For planned absences, students should inform the instructor as early as possible before the class; for unplanned absences due to accidents or emergencies, students should contact the instructor as soon as conditions permit. Students are permitted a reasonable amount of time to make up missed material or activities.

Students who do not participate in at least one of the first two class meetings, without contacting the department to indicate intent, may be dropped from the course. Excessive absences, after due warning, may result in a failing grade.

Religious Holidays Guidelines

Students should inform the instructor, in advance, of religious observances that will conflict with class attendance, tests, or other class activities; the instructor is then obligated to accommodate that observance. Students shall be permitted a reasonable amount of time to make up missed material and shall not be penalized for religious-observance absences. A student who believes they have been unreasonably denied this accommodation may seek redress through the student grievance procedure.

Absence Due to Illness

Students absent due to illness should contact the instructor, if feasible, as early as possible before the missed class or activity, and will be permitted a reasonable amount of time to make up the missed material. Students should contact their college by the applicable deadline to drop a course for medical reasons, or petition the Dean of Students Office.

Twelve-Day Rule

Students participating in university-sponsored athletic or scholarly activities may be absent up to 12 scholastic days per semester without penalty, provided the student or advisor notifies the instructor as early as possible. Instructors must be flexible in rescheduling exams, assignments, and other required activities and must not penalize the student. Students previously warned in writing about the impact of absences on their performance should not incur additional absences, even within the 12-day allowance.

Late & Make-Up Work Policy

Work submitted late always receives partial credit, so turning in late work is preferable to not submitting an assignment at all, which results in a zero for that task.

Lateness	Penalty
Up to 12 hours late	20% penalty of maximum possible points (e.g., 100 → 80)
12–48 hours late	30% penalty of maximum possible points (e.g., 100 → 70)
After 48 hours	50% penalty of maximum possible points (e.g., 100 → 50)

Exceptions: documented medical issues (with an official, dated and signed letter from a health care provider), or other issues discussed with and approved by the instructor ahead of time.

Academic Integrity

Students are expected to uphold the UF Honor Code in all coursework. Programming assignments are individual work unless explicitly designated as collaborative; students may discuss concepts and debugging strategies with peers, but the code submitted must be the student's own. Reusing publicly available code (e.g., engine tutorials, Stack Overflow snippets) is permitted only with a clear, in-code citation of the source — undisclosed copying is treated as a violation of academic integrity.

Use of Generative AI Tools

Students may use AI coding assistants (e.g., GitHub Copilot, ChatGPT, Claude) for tasks such as debugging, boilerplate generation, and exploring alternative approaches, provided the student can explain, defend, and modify every line of submitted code. Submitting AI-generated code the student does not understand — or that was not meaningfully adapted to the assignment — does not meet the learning objectives of the course and may be treated as an academic integrity violation. When in doubt, disclose AI assistance in a short code comment describing what was used and how.

Accommodations for Students with Disabilities

Students requesting classroom accommodations must first register with the university's Disability Resource Center, which will provide documentation to the student to share with the instructor when requesting accommodation. Students are encouraged to share their accommodation letter and discuss specific needs with the instructor as early in the semester as possible.

Diversity, Access & Inclusion

This course welcomes students of all backgrounds, identities, and levels of prior technical experience. Game engine development draws on mathematics, art, and software engineering alike, and students bring different strengths to each. Respectful, constructive communication is expected in all discussion boards, group activities, and peer feedback; disrespectful or exclusionary conduct will not be tolerated.

University Policies & Resources

Additional information about grading policies, support for students with disabilities, course evaluations, the Honor Code, and other course policies and campus resources can be found on the university's Syllabus Policies page.

MODULE-BY-MODULE BREAKDOWN

ORIENTATION

Welcome to the Course

The orientation introduces the structure of the course, its policies, and the software students will need before technical content begins. Students meet the instructor and their peers, review the syllabus and learning-management-system layout, and download the required software.

Learning Objectives

- Identify the key components of the syllabus, including the objectives of the class, assignments, and other requirements and policies
- Identify the structure of the class in the learning management system and how to find resources provided within that structure
- Discover the required software for this class and download it to your computer

Activities

- Attend the live orientation lecture
- Meet your peers and introduce yourself to the class through the “Hi! Introduce Yourself” discussion
- Participate in the discussion: What Is a Game Engine?
- Complete the Course Orientation Quiz
- (Optional) Check out the Course General Questions Forum — post questions, answer peers, or share related resources

MODULE 1

Introduction to Game Engines

This module introduces the concept of a game engine as a complex software system that supports the creation of interactive digital experiences. Students explore what distinguishes a game engine from a game itself, examine the major components common to modern engines (rendering, input, physics, asset management, and scripting), and analyze how these components work together — setting the conceptual foundation for building an engine later in the course.

Learning Objectives

- Define what a game engine is and explain its role in game and interactive application development
- Identify and describe the core components of a modern game engine
- Distinguish between a game engine and the games built on top of it
- Analyze an existing game engine and outline its major subsystems
- Discuss how the scale and complexity of game engines have evolved over time

Activities

- Attend the live lectures
- Download and install Unity

- Complete the assignment: Exploring the Components of a Game Engine
- Participate in the discussion: How many people are needed to develop a game engine?

MODULE 2

3D Geometric Representation

This module introduces triangular meshes as the dominant data representation for 3D surfaces in graphics, simulation, and interactive applications. Students learn how complex surfaces are discretized into vertices and indexed triangles, and how per-vertex attributes such as normals and UV texture coordinates are defined and used for shading and texturing, alongside optimization considerations such as vertex/face counts, memory budgets, and face culling.

Learning Objectives

- Explain how 3D surfaces are represented using triangular meshes and indexed geometry
- Construct and interpret triangle index buffers and vertex buffers
- Define and manage per-vertex attributes, including positions, normals, and UV texture coordinates
- Analyze the memory footprint of mesh data in terms of vertex, face, and byte counts
- Evaluate optimization techniques such as face culling, vertex reuse, and attribute packing

Activities

- Attend the live lectures
- Use the template code: MeshMaker.cs
- Complete the programming assignment: Develop an Object Maker class (procedurally generate planes, boxes, pyramids, ellipsoids, and cylinders in Unity without using Unity's primitive shapes)

MODULE 3

3D Object File Formats

This module introduces common 3D object file formats used in graphics, visualization, and real-time rendering pipelines, with emphasis on the differences between text-based and binary formats and a detailed examination of OBJ and modern GPU-oriented formats such as glTF.

Learning Objectives

- Identify common 3D object file formats and describe their intended use cases
- Distinguish between text-based and binary 3D file formats
- Interpret the structure and syntax of text-based formats such as OBJ
- Explain how glTF organizes data using scenes, nodes, meshes, accessors, bufferViews, and buffers
- Trace how vertex and index data are mapped from file storage to GPU memory
- Select an appropriate 3D file format for a given application or pipeline

Activities

- Attend the live lectures
- Explore sample 3D models using a plain text editor

- Participate in the discussion: Which 3D file format do you like better?
- Consult the code templates: OBJParser.cs and OBJGameObject.cs
- Complete the programming assignment: Develop an OBJ Parser to load textured 3D objects (vertices, normals, uv, triangles and material properties) and render them in Unity.

MODULE 4

3D Transformations and Camera Systems

This module introduces the mathematical and conceptual foundations required to position and view objects in a 3D game engine — how objects move from local space into world, camera, and screen space using model, view, and projection matrices, including hierarchical transformations and camera systems.

Learning Objectives

- Explain the differences between local, world, view, and projection spaces
- Construct and combine translation, rotation, and scaling transformations
- Build model matrices from position, rotation, and scale parameters
- Understand and implement hierarchical (parent–child) transformations
- Explain the role of the camera and compute a view matrix
- Implement a solar-system animation using a custom camera and transformation system

Activities

- Attend the live lectures
- Use the template code for Custom Node Hierarchy: CustomNode.cs, and the example SolarSystem.cs
- Complete the programming assignment: Build a set of object-oriented classes that implement Scene Hierarchy and create an animated Solar system scene in Unity (without using Unity's hierarchy).

MODULE 5

Introduction to Shaders

The course transitions to Android Studio to begin building a custom game engine framework, focusing on programmable graphics through OpenGL shaders. Students are introduced to GLSL — how vertex and fragment shaders operate, how data flows through the rendering pipeline, and how shaders control visual output.

Learning Objectives

- Configure and use Android Studio for game engine development
- Set up the foundation of a custom Android-based game engine project
- Write and compile basic GLSL vertex and fragment shaders
- Understand shader inputs, outputs, and common data structures used in GLSL
- Implement common shader operations such as transformations, coloring, and lighting basics

Activities

- Attend the live lectures
- Install the latest version of Android Studio

- Create a Pixel 6 virtual device (API 32 “Sv2,” Android 12L) in the Device Manager
- Explore the project: GameEngineDev.zip
- Consult the Android reference documentation for Matrix and GLES30
- Complete the programming assignment: write and compile a basic GLSL vertex/fragment shader pair and apply it in the custom Android engine

MODULE 6

Advanced Shaders

This module builds on introductory shader concepts by exploring widely used real-time rendering techniques implemented in GLSL, such as the Phong shading model, normal maps, and MatCap shaders, and how texture-based and lighting-based effects can be integrated into a game engine.

Learning Objectives

- Implement textured shaders to apply image-based surface detail in GLSL
- Explain and implement the Phong shading model for realistic lighting
- Use normal maps to simulate fine surface detail without increasing geometry complexity
- Implement MatCap shaders for efficient material appearance rendering
- Analyze the visual and performance trade-offs between different shading techniques

Activities

- Attend the live lectures
- Consult the code: GameEngineDev2.zip
- Complete the assignment: Develop advanced GLSL Shaders (including matcap, normal map, and multiple light source shaders).
- Create a presentation: Explain Shader Techniques

MODULE 7

Shadow Rendering

Students extend their custom game engine to implement real-time shadow rendering using shadow mapping. A dedicated shadow map shader is introduced and integrated into the existing pipeline, with attention to common artifacts (aliasing, acne, peter-panning), performance optimization, and parameter tuning.

Learning Objectives

- Explain the principles of shadow mapping and how depth information generates real-time shadows
- Implement a shadow map rendering pass using programmable shaders in a mobile OpenGL ES pipeline
- Integrate shadow mapping into existing vertex and fragment shaders to support dynamic lighting
- Identify and address common shadow mapping artifacts
- Optimize shadow rendering performance for mobile devices and tune shadow parameters

Activities

- Attend the live lecture

- Experiment with the code from class: GameEngineDev_shadows.zip
- Complete the programming assignment: implement a shadow-map rendering pass and integrate it into the existing shaders
- Submit Final Project Progress Update 5 (final update before presentations)
- Prepare for final presentations

MODULE 8

Input Systems: Controllers, Haptics, and Touch Interfaces

This module introduces input handling in immersive applications using the OpenXR standard with the Meta OpenXR SDK for native Android development, including controller-based input (buttons, analog triggers, thumbsticks, action bindings, and haptic feedback), and touchscreen-based input as a fallback and comparative pathway for non-VR Android devices.

Learning Objectives

- Explain the architecture and purpose of the OpenXR input system
- Set up and configure the Meta OpenXR SDK within Android Studio for native development
- Describe the action-based input model used in OpenXR (actions, action sets, bindings, interaction profiles)
- Implement touchscreen-based input handling on Android (gesture detection, touch events, multi-touch)
- Compare immersive controller input with mobile touchscreen interaction models

Activities

- Attend the live lectures
- Read the OpenXR specification and the Meta OpenXR Mobile SDK documentation
- Download the Meta OpenXR SDK and the Java for Quest project; open and compile the XrCompositor_NativeActivity sample in Android Studio
- Complete the programming assignment: implement OpenXR controller input handling (buttons, triggers, thumbsticks) with a touchscreen input fallback
- Submit the Final Project Proposal

MODULE 9

Physics Engines

This module introduces the fundamental concepts of physics engines used to simulate realistic motion and interaction between objects — rigid bodies, collision shapes (spheres, boxes, capsules, planes), and key physical parameters (mass, restitution, friction) — demonstrated with the Bullet Physics Engine and the jBullet library in Android Studio.

Learning Objectives

- Describe rigid body dynamics and how they are used in real-time physics simulation
- Identify common collision shapes used in physics engines
- Distinguish between dynamic and kinematic objects in a physics simulation
- Interpret mass, restitution, and friction and explain how they affect object behavior

- Integrate and use the Bullet physics engine library within an Android Studio project

Activities

- Attend the live lectures
- Review the Bullet physics engine and explore its source code
- Modify the code from class: GameEngineDev_physics.zip
- Complete the programming assignment: build a rigid-body physics scene with Bullet, combining collider shapes and tuning mass, restitution, and friction
- Submit the Final Project Progress Update

MODULE 10

Networking for Multi-Player Gaming

This module introduces the fundamentals of networking in game engine development, including client–server architectures, data synchronization, and communication protocols, with hands-on implementation of networking features using WebSockets to support responsive multiplayer gaming.

Learning Objectives

- Explain the core principles of computer networking as they apply to game engines
- Implement basic network communication using WebSockets
- Design and implement synchronization strategies for game state
- Develop simple multiplayer features such as player movement and interaction over a network
- Evaluate trade-offs between performance, consistency, and scalability in networked games

Activities

- Attend the live lectures
- Consult the official WebSocket protocol (RFC 6455) and the OkHttp WebSocket API
- Explore the code: GameEngineDev_networking.zip and Game Engine Dev_multiplayer_with_VR.zip
- Complete the programming assignment: implement WebSocket-based state synchronization for player position and interaction
- Submit Final Project Progress Update 2

MODULE 11

Special Topic: Building Endless Runner Games

This module explores the design and implementation of endless runner games, using the engine built throughout the semester to support procedurally generated environments, continuous forward motion, obstacle spawning, and performance optimization — using Unity's Trash Dash as a case study alongside the class's own Android-based engine.

Learning Objectives

- Design and implement continuous forward movement systems for player-controlled characters
- Develop procedural content generation techniques for dynamically creating terrain and obstacles

- Implement object pooling to efficiently manage frequently spawned and destroyed entities
- Analyze performance bottlenecks specific to endless runner architectures and propose solutions
- Adapt core engine systems to support genre-specific gameplay constraints

Activities

- Attend the live lecture
- Explore Unity's Trash Dash endless runner sample, especially TrackManager.cs and TrackSegment.cs
- Run the J4Q_Phone/EndlessUniverse example in Android Studio and modify Level.java and LevelSegment.java
- Complete the programming assignment: implement procedural track-segment spawning and object pooling in the custom engine
- Submit Final Project Progress Update 3

MODULE 12

Natural User Interfaces and Real-Time Body Tracking

Students explore natural user interfaces built on real-time body tracking, using AI-based monocular video pose estimation (e.g., Google's MediaPipe) to extract skeletal keypoints and convert them into joint rotations via inverse kinematics that drive real-time avatar animation, working with the in-house UPose Unity package for tracking and retargeting.

Learning Objectives

- Use AI-based frameworks such as MediaPipe to extract 2D/3D body keypoints from video input
- Convert tracked keypoints into joint rotations using inverse kinematics (IK)
- Map computed joint angles to a digital avatar to enable real-time character animation
- Evaluate limitations and sources of error in vision-based body tracking
- Use and extend the UPose Unity package for real-time body tracking and avatar control

Activities

- Attend the live lecture
- Experiment with the UPose code repository and import the UPose Unity package
- Complete the programming assignment: map MediaPipe pose keypoints to avatar joint rotations using inverse kinematics
- Submit Final Project Progress Update 4

MODULE 13

Final Presentations

In this final module, students present their work and discuss what they developed, the challenges they faced, and the lessons they learned, exploring the unique game engine components each project explored through peer interaction.

Learning Objectives

- Clearly present and communicate the design, architecture, and functionality of a game engine project to a technical audience
- Critically reflect on the development process, identifying key challenges, limitations, and solutions
- Articulate technical decisions, including trade-offs between performance, usability, and scalability
- Analyze and evaluate peer projects and provide constructive, actionable feedback
- Reflect on growth as a developer and identify areas for continued learning

Activities

- Attend the live presentations
- Present the final project
- Submit the Final Project Submission

ABOUT THE INSTRUCTOR

Dr. Angelos Barmpoutis is a Professor of Digital Arts and Sciences and coordinator of research and technology at the Digital Worlds Institute, University of Florida College of the Arts, with affiliate faculty appointments in the Department of Computer & Information Science and Engineering and the Center for the Arts, Migration, and Entrepreneurship.

Since 2010, as the Institute's inaugural tenure-track faculty member, he has helped build an interdisciplinary academic unit bridging the arts with other disciplines — developing curricula that now serve more than 500 undergraduates a year, alongside a graduate program in immersive technologies. His research treats computational systems not as production tools but as creative collaborators: systems that expand artistic expression, advance computer vision and AI methods, and generate real societal impact, from the concert hall to the clinic.

He directs the Digital Innovation and Public Experience Design lab in the College of the Arts, was named UF Research Foundation Professor (2020–2023) and Teacher of the Year (2017), and currently chairs the Interactive and Emerging Media curricular area at UF.

Contact Information & Office Hours

Email angelos@digitalworlds.ufl.edu

Office Hours As announced in the course's LMS page or by appointment

Phone +1 (352) 294-2042

Office Lab E109, CSE Building, University of Florida

Mail University of Florida, PO Box 115800, Gainesville, FL 32611, USA

Website angelosbarmpoutis.com