|  |
| --- |
| MHI625659 -24-B-GLAS |
| Coursework |
| Games Programming 3 |

|  |
| --- |
| Module Leader: Bryan Young (Bryan.Young@gcu.ac.uk) |

*Session 2024 - 2025*

***Plagiarism***

*Attention is drawn to the University regulations on plagiarism. Whilst discussion of the coursework between students is encouraged, the actual work has to be undertaken individually. Collusion may result in a zero mark being recorded for the coursework for all concerned and may result in further action being taken.*

# Scenario

This coursework challenges students to apply their understanding of C++, GLSL, and OpenGL to develop a structured, optimised, and playable game inspired by Asteroids. The focus is on rendering, buffer management, and physics, ensuring a performant and scalable design.

Guidelines:

* The game must be developed in C++ using OpenGL and GLSL for rendering.
* Your project must be delivered in a folder with a full Visual Studio solution format.
* **Makefiles will not be accepted**—use a structured solution and project setup.
* A video demonstration of your game in action is required with your submission. This is essential in case of compilation issues due to hardcoded paths or other system-specific dependencies.

Your final submission should showcase a well-structured, efficient, and playable implementation, with marks awarded for both technical execution and game design polish.

The marking scheme is as follows:

## Marking Scheme

Marks for this coursework will be awarded on the following basis:

|  |  |
| --- | --- |
|  | Mark Up To |
| Core Code Functionality (30 Marks) - This section evaluates how well students apply key programming techniques covered in class. Full marks will be awarded for correct, efficient, and well-structured implementations. |  |
| Interleaved Buffers  **Description**: Efficient use of interleaved vertex buffers to store and process vertex attributes.  **Considerations**: The correct use of interleaved buffers for efficient vertex attribute storage and retrieval will be assessed, i.e. proper structuring of buffer data (position, normals, UVs packed into a single VBO). Correctly setting up and binding VAOs to access interleaved attributes, avoiding redundant data storage across multiple buffers. | 5 |
| Uniform Buffer Objects  **Description**: Use of UBOs to store and update transformation matrices efficiently.  **Considerations**: Storing Model, View, and Projection matrices in a UBO. Minimising redundant updates by only modifying data when necessary. Correctly binding UBOs across multiple shaders. | 5 |
| Wrapper Classes for Core Functionality  **Description**: Creation of clear, reusable, and well-encapsulated wrapper classes for OpenGL objects (e.g. Displays and Context).  **Considerations**: This component assesses the correct use of Resource Acquisition Is Initialisation (RAII) principles for managing OpenGL objects, ensuring automatic resource cleanup and preventing memory leaks. Abstracting OpenGL calls to improve code readability and maintainability. | 5 |
| Management Classes  **Description**: This component evaluates the efficiency and clarity of game component management through well-structured and well-abstracted management classes. These classes should handle core game elements such as transforms, shaders, and game objects in a way that enhances modularity, scalability, and performance.  **Considerations**: Each manager should only handle one category of objects (e.g., TransformManager should only manage transforms, ShaderManager should only handle shaders).  Use hashmaps (std::unordered\_map) for efficient lookup of shaders, transforms, and game objects by name or ID.Avoid unnecessary duplication—game objects should store pointers to shared meshes and shaders, rather than redundant copies. Transform data should be kept in a contiguous structure where possible to improve cache efficiency. | 5 |
| Physics Engine  **Description**: This component evaluates the implementation of a DLL based physics system that accurately models movement, acceleration, and basic collision detection. The physics engine should ensure that objects maintain momentum when movement keys are released, and acceleration should feel smooth and natural, rather than abrupt or unrealistic.  **Considerations**: The physics engine should directly modify object transforms rather than applying movement logic inside the game loop.  Velocity and acceleration should be handled as independent components, ensuring smooth motion rather than frame-dependent updates.  Players and asteroids should not snap to new positions but should exhibit continuous motion. When rotation changes, the ship should not immediately move in the new direction, but instead combine its current velocity with the new thrust vector.  Collision Detection:  A Broad Phase vs. Narrow Phase approach should be used to optimize collision detection efficiency.  Broad Phase: Use Axis-Aligned Bounding Boxes (AABB) to quickly filter out objects that are not close enough to collide.  Narrow Phase: Perform a more detailed collision check only between objects that passed the broad-phase test.  For asteroids, use bounding spheres or AABBs instead of per-triangle collisions for performance reasons. | 10 |
| Extension Material  **Description**: This section encourages students to push beyond the core requirements by introducing additional features that enhance rendering, gameplay, physics, or technical depth. Marks will be awarded based on complexity, originality, and how well the additions integrate into the overall project.  **The suggestions below are examples of possible extensions; students are encouraged to explore their own ideas. Any additional material implemented will be appropriately graded based on technical complexity and integration into the overall project. Creativity is encouraged, and students are welcome to implement original features not listed here.**  **Advanced Rendering & Shader Techniques:** Post-Processing & Framebuffers: Implementing Framebuffer Objects (FBOs) for effects such as bloom, motion blur, screen-space reflections, or deferred rendering. Expanding on previously covered concepts (e.g., implementing a fully deferred renderer instead of a simple FBO) will receive higher marks than direct reapplications of past work.  Bespoke UBO Layouts & Advanced Shader Structures: Creating custom UBO layouts to handle additional data such as light sources, material properties, or skeletal animation matrices will demonstrate an advanced understanding of shader management.  Optimised Rendering Pipelines: Implementing instanced rendering or reducing draw calls by batching similar objects can significantly improve performance.  **Advanced Physics & Mechanics:** Expanded Physics System: Enhancing the physics engine beyond the basic implementation, including features such as collision response (elasticity, friction), rotation physics, gravity wells, or object destruction.  Tag-Based ECS (Entity Component System) Structure: Instead of manually managing game objects, implementing a tag-based ECS for dynamic game object categorization and processing.  AI Behaviours & Game Logic: Implementing enemy AI with movement patterns, decision-making, or procedural generation.  **Performance Optimisation & Data Handling:** Memory Management Enhancements: Implementing object pooling, efficient data layouts, or reducing memory fragmentation will earn marks in this category.  Parallel Processing or Multi-Threading Considerations: Exploring multi-threading for physics calculations or asynchronous data streaming can greatly improve performance.  Reduced CPU-GPU Synchronisation Overhead: Optimisations such as persistent mapped buffers, bindless textures, or minimising unnecessary state changes. | 45 |
| Game Feel and Playability  **Description**: This section assesses the fluidity, responsiveness, and overall experience of the game. A technically competent implementation should still feel engaging and polished.  **The suggestions below are examples of possible extensions; students are encouraged to explore their own ideas. Any additional material implemented will be appropriately graded based on technical complexity and integration into the overall project. Creativity is encouraged, and students are welcome to implement original features not listed here.**  **Control Responsiveness:** Movement should feel smooth, responsive, and intuitive. Acceleration, deceleration, and turn rates must be handled in a way that enhances player control without feeling too stiff or too loose.  **Visual and Audio Feedback:** The game should provide clear visual and/or audio feedback for player actions, collisions, and interactions. This may include screen shake, sound effects, particle effects, or animations.  **Overall Playability and Engagement:** The final product should feel like a cohesive, playable game rather than a disconnected set of mechanics. The design should make the gameplay loop enjoyable and encourage continued play. | 25 |
| **-Total** | **100** |
|  |  |

**Note:**

Marks for each component listed above will be awarded based on the complexity and quality of the implementation and final product.

Any online resources, such as art assets and tutorials, must be properly credited.

While you are permitted to learn from online or offline tutorials, your coursework should extend beyond the tutorial content to achieve maximum marks. The submitted work should not simply be the direct result of following a tutorial.

## Submission

Submission of this coursework should be made electronically via **GCULearn** no later than **17:00 on 01/05/2025**.

Late submissions will not be accepted without a valid and documented reason.

# Plagiarism

Attention is drawn to the university regulations on plagiarism. Whilst discussion of the coursework between students is encouraged, the actual work has to be undertaken individually. Collusion may result in a zero mark being recorded for the coursework for all concerned and may result in further action being taken.

University regulations, codes & policies: <http://www.gcal.ac.uk/student/about/regulations/index.html>

Plagiarism and cheating: <http://www.gcal.ac.uk/student/coursework/regulations/plagiarism.html>