Assignment 4: Camera

Course: IGME 309 – Real Time Simulations for Games II

Golisano College of Computing and Information Sciences

School of Interactive Games and Media

Rochester Institute of Technology

Due: Check in MyCourses

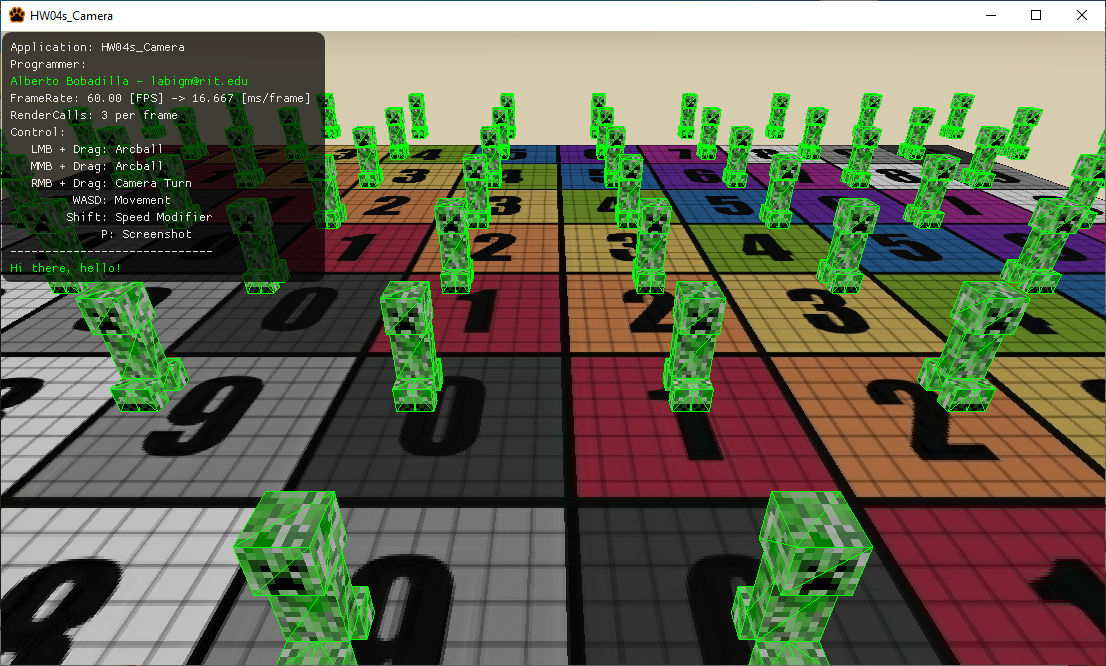
Deliverable: Unzipped Camera.cpp file (no full solution unless using a custom framework)

## Getting Started

Video description: <https://www.youtube.com/watch?v=HFa2S2mAqXo>

In the class repository, I’ve included a solution under the \_Binary folder.

The starter code will give you this out of the box:



## Objective

Implement a fully interactive 3D camera system that supports six degrees of movement and mouse-controlled view rotation using quaternions. The camera must move relative to its current orientation, not world axes, and function similarly to camera systems in Fist Person 3D games like Minecraft (similar to creative mode but you move in a view vector not maintaining height regardless of where you are looking at).

## Overview

Starter code is available in the course repository. You are not required to use this codebase, but if you do not, you must port the expected behavior into your own framework. The \_Binary folder includes a working demo for reference.  
  
The starter scene renders a scene with Creepers. The camera is initialized at the center, and movement input (W and S) is partially implemented. Your task is to complete full movement and view rotation functionality within the Camera class.

## Camera Features to Implement

## 1. Basic Movement – 30%

* Implement WASD and QE movement controls without rotating the camera:
* W/S: Move forward/backward
* A/D: Move left/right
* Q/E: Move vertically up/down

10% per movement pair. Keyboard input part is already implement, use only the methods in the class to process the displacement value

## 2. View Rotation – 20%

* Allow the camera to rotate based on right mouse button input:
* Implement horizontal and vertical view rotation using quaternions.
* Do not implement roll.
* Rotation input handling is already in place; only quaternion math is needed.

## 3. Orientation-Based Movement – 50%

* Camera movement must be relative to the view direction, not the world axes:
* Pressing W should always move the camera 'forward' in its current direction.
* Movement should be consistent regardless of camera orientation.
* This is standard behavior in most 3D applications.

## Rubric (100%)

|  |  |  |
| --- | --- | --- |
| Component | Weight | Notes |
| Basic Movement (WASD, QE) | 30% | Movement must be smooth and operate along world axes |
| Mouse-based View Rotation | 20% | Must use quaternions; horizontal and vertical only |
| Orientation-Based Movement | 50% | Movement is local to camera space, not global |

## Deductions

* -20%: Hardcoded behavior in the camera class
* -20%: Missing or insufficient code comments
* -30%: Not using quaternions / Gimbal Lock present
* -30%: Performing view calculations outside camera class

## Hints

* Implement simple movement first.
* Then implement rotation using quaternions.
* Lastly, rework movement to function in local (camera) space.
* Avoid gimbal lock by keeping your up vector and forward vector non-aligned.
* Clamping vertical angles to prevent “perfect up” views can help.
* Use glm::lookAt() carefully; ensure vectors are not colinear.

## Submission Instructions

* Submit only your project folder as a simple uncompressed file Camera.cpp.
* Must be under 20 KB if using starter code.
* If using a custom engine/framework, submit the entire solution.
* Push your solution to your repository with commit message: E04 Deliverable
* Upload the file to the dropbox A04 - Camera
* In the Dropbox comments, include the link to your Git repository.