

# **Computer Graphics**

## **Project 1**

Alexandra Madeira - 2221926

Pedro Ferreira - 2212613

# Camera

## Constructor

- Initializes the camera's position, orientation (Yaw and Pitch), and default settings like movement speed, mouse sensitivity, and zoom level.
- Calls `updateCameraVectors` to compute the initial orientation.

## GetViewMatrix

- Uses the `glm::lookAt` function to compute the view matrix, determining how the scene is viewed from the camera's perspective.
- `glm::lookAt` takes the camera's position, its target (Position + Front), and the Up vector.

## ProcessKeyboard

- Adjusts the camera's position based on input direction and movement speed.
- Supports diagonal movement by normalizing the resultant vector if multiple keys are pressed.

## ProcessMouseMovement

- Adjusts the Yaw and Pitch angles based on mouse input.
- Optionally constrains Pitch to prevent flipping the camera upside down, limiting it to  $[-89^\circ, 89^\circ]$ .
- Updates the camera's orientation vectors by calling `updateCameraVectors`.

## ProcessMouseScroll

- Changes the Zoom value based on scroll input.
- Constrains the zoom level between a minimum ( $1^\circ$ ) and maximum ( $45^\circ$ ).

## updateCameraVectors

- Calculates the Front vector using trigonometric functions and Euler angles (Yaw, Pitch).
- Updates the Right and Up vectors using the cross product:
  - Right is perpendicular to Front and WorldUp.
  - Up is perpendicular to Right and Front.

## toggleYLock

- Locks the camera's Y-axis to allow realistic first-person movement.
- Can be toggled using the `E` key.
- Both locked and unlocked movement modes are supported due to the map's verticality.

# Character

## Constructor

- Initializes the character's position, team, and rotation offset.
- Assigns a default rotation based on the team:
  - CT: 90°
  - T: -90°
- Applies an optional rotOffset for customization.

## getModelMatrix

- Generates the model matrix for rendering the character's position, orientation, and scale.
- Constructs the matrix using GLM functions:
  - Translation: glm::translate.
  - Rotation: glm::rotate around the Y-axis.
  - Scaling: glm::scale to 2.5% of the original size.
- Returns the final transformation matrix.

# Crosshair

## Constructor

- Initializes window dimensions and crosshair color.
- Calls setupBuffers to prepare the geometry for rendering.

## Destructor

- Frees GPU resources by deleting the VAO and VBO.

## setupBuffers

- Configures the VAO and VBO to define crosshair geometry.
- Creates horizontal and vertical lines centered at (0, 0, 0).
- Uses interleaved data for positions and texture coordinates.

## render

- Renders the crosshair as a 2D overlay at the center of the screen.
- Temporarily disables depth testing to ensure visibility.
- Uses an orthographic projection (glm::ortho) based on window dimensions.

# OBJLoader

## Constructor

- Initializes containers for vertices, UVs, normals, indices, and materials.

## loadOBJ(const std::string& path)

- Reads .obj files to load geometry and material data.
- Processes lines for vertex positions (v), UVs (vt), normals (vn), and faces (f).

## loadMTL(const std::string& mtlPath)

- Reads .mtl files to load material definitions and associated textures.

## loadTexture(const std::string& textureFilename)

- Loads textures into OpenGL, generates mipmaps, and sets filtering/wrapping parameters.

## Getter Methods

- getVertices, getUVs, getMaterials, getIndices: Return parsed geometry data.

# Renderer

## Constructor and Destructor

- Initializes rendering buffers and frees resources during cleanup.

## initialize(const OBJLoader& objLoader)

- Sets up VAOs, VBOs, and EBOs for models loaded by OBJLoader.

## render(const ShaderProgram& shaderProgram, const OBJLoader& objLoader)

- Renders models using configured buffers and materials.

# Shader

## Constructor: `ShaderProgram`

- Loads, compiles, and links vertex and fragment shaders.
- Performs error checking for shader compilation and linking.

## Destructor

- Deletes shader programs to free GPU resources.

## Uniform Setting Functions

- Provide methods to pass data (e.g., transformations) to the GPU.

# Skybox

## Constructor and Destructor

- Sets up resources for rendering a cube map and frees them during cleanup.

## `initialize(const std::vector<std::string>& faces)`

- Loads textures for the skybox and sets up cube geometry.

## `render(const ShaderProgram& shader)`

- Renders the skybox cube.

# WindowManager

## Constructor and Destructor

- Manages SDL2 window creation and OpenGL context initialization.

## initialize()

- Sets up SDL2, OpenGL attributes, and V-Sync.
- Initializes GLAD for OpenGL function loading.

## handleEvents(bool& running)

- Processes SDL events such as quitting or keypresses.

## swapBuffers()

- Swaps front and back buffers for double-buffered rendering.