**Graphical Processing Project**

**Name: Moga Eduard**

**Group: 30234**

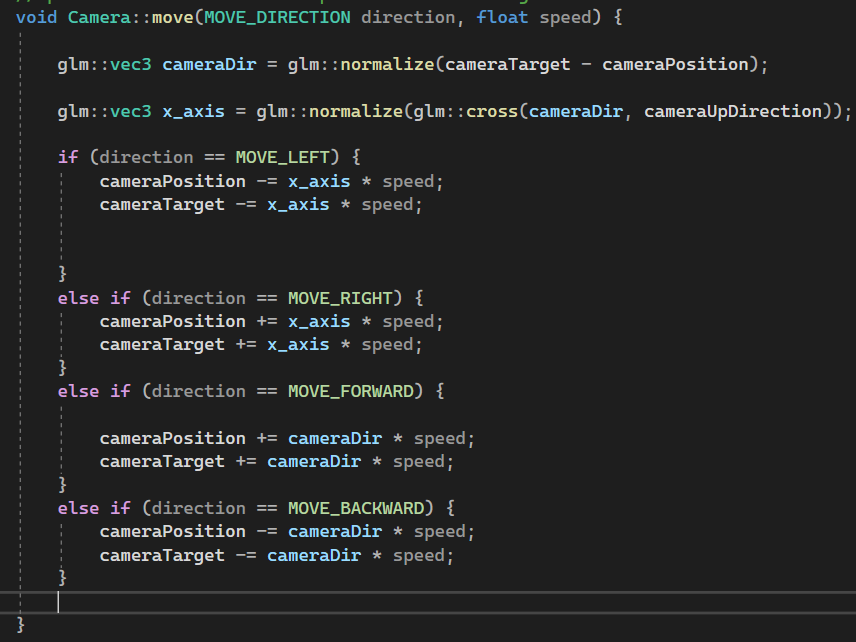
**Year: 2024-2025**

**Teacher: Nandra Cosmin**

*Functional details:*

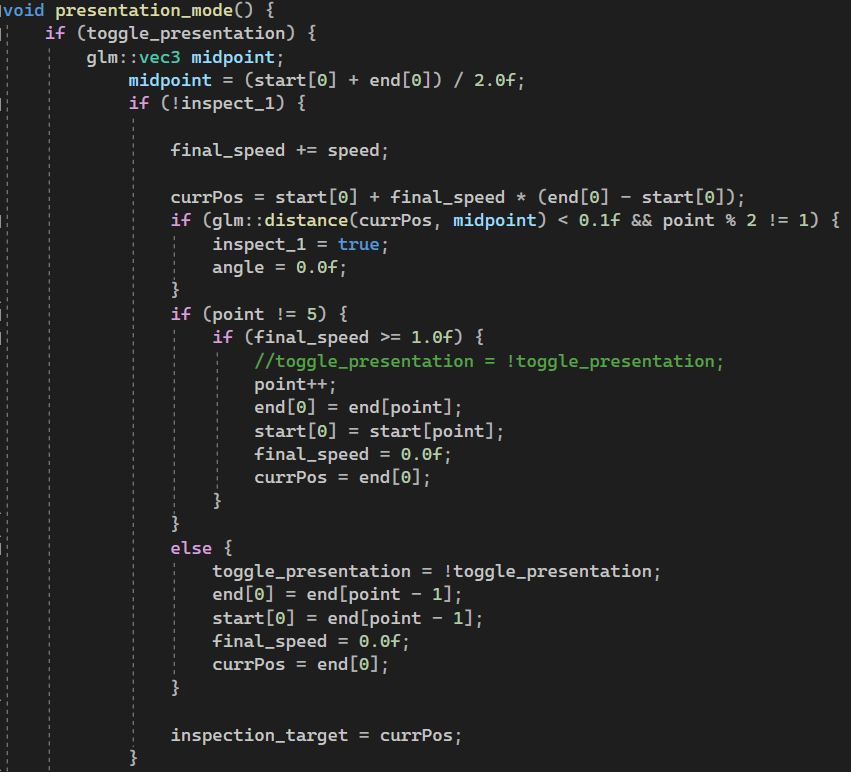
* A screen shot of a computer code

  Description automatically generated**Movement**: The camera system in this project is designed to provide dynamic control of the scene's view using a combination of translation and rotation mechanisms. The Camera class encapsulates core functionalities such as movement, rotation, and view matrix generation. The getViewMatrix function leverages the glm::lookAt method to create and return the view matrix, ensuring precise alignment of the camera's position, target, and up direction. The mouse input handling function, implemented as mouseCallback, integrates user interactions to dynamically adjust the camera's orientation. Using the GLFW library, mouse movements are tracked, with sensitivity adjustments to fine-tune the responsiveness of pitch and yaw calculations. This input is then used to update the camera's rotation parameters by invoking the rotate method of the Camera class.



A screen shot of a computer program

Description automatically generated

* A computer screen with text

  Description automatically generated**Presentation mode:** The presentation mode in this project introduces an automated camera animation that provides a cinematic transition through the 3D scene, activated by a user command. The animation is defined by a series of predefined waypoints start[5], end[5] and interpolates the camera's position between these points using a constant speed factor. The system calculates the midpoint between the current start and end positions, dynamically updating the camera's position (currPos) based on the interpolated vector. When the camera approaches the midpoint, it optionally enters an inspection mode, where it rotates around the current point to provide a comprehensive view of the environment. Transitioning between waypoints is seamless, ensuring a fluid and immersive visual presentation.
* **Light System (Directional light / Point light):** The directional light provides consistent illumination across the scene, simulating sunlight with a uniform direction. Using the computeLightComponents function, the shader calculates ambient, diffuse, and specular contributions based on the directional light's intensity and direction relative to the surface normals.Shadow mapping is integrated to enhance realism by accounting for occluded areas. A fog effect, modulated by the fragment's distance from the camera, adds depth to the scene and softens the transition between objects and the background. Texture mapping is employed to enrich material details, with dynamic blending between diffuse and specular maps. For the point lights, the computePointLight function supports up to three distinct light sources, each radiating in all directions with attenuation based on distance.

Difference in computing the point light is the use of lightPosWorld that is the light in the world space computed by ***lightPosWorld = model \* vec4(vPosition, 1.0f);***

A screenshot of a computer program

Description automatically generated**A computer screen shot of a black screen

Description automatically generated**

A screen shot of a computer code

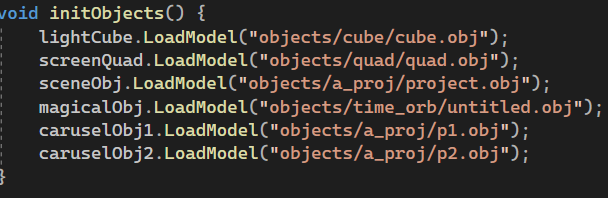
Description automatically generatedIn the project I implemented another shader like the first one for the change of the scenery, with different colors.



* **GL\_LINE / GL\_POINT:** Modes between line, point, and fill that showcase the wireframe and point of the objects used.
  + if (key == GLFW\_KEY\_2 && action == GLFW\_PRESS) {
  + toggle\_wireframe = !toggle\_wireframe;
  + toggle\_points = false;
  + if (!toggle\_wireframe) {
  + glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
  + }
  + else {
  + glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
  + }
  + }
  + if (key == GLFW\_KEY\_3 && action == GLFW\_PRESS) {
  + toggle\_points = !toggle\_points;
  + toggle\_wireframe = false;
  + if (!toggle\_points) {
  + glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
  + }
  + else {
  + glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT);
  + }
  + }
* **Textures:** The project includes a variety of textured 3D objects that add depth and realism to the scene. These objects are loaded using the LoadModel function, which prepares their geometry and associated textures for rendering.

Models, such as the screenQuad for post-processing effects and thematic objects like magicalObj and carousel parts (caruselObj1, caruselObj2), enrich the scene's narrative and sceneObj which is the entire static scene. Each model comes with its texture maps, including diffuse and specular layers, ensuring high-quality visual output during rendering.

The objects are initialized as obj from obj class (ex*: gps::Model3D sceneObj*;) . The draw function is called whenever we want to draw on the screen.



****

* **Shadows:** Shadows are calculated in the shader using the computeShadow function, which determines whether a fragment is in shadow by comparing its depth to the nearest depth recorded in the shadow map. The fragment's light-space coordinates are normalized and adjusted to the [0, 1] range for sampling from the shadow map texture.

The light-space transformation matrix is calculated using the computeLightSpaceTrMatrix function, which combines orthographic projection and a light-view matrix. This matrix aligns the light's perspective with the scene, capturing depth information for shadow mapping. The light-view matrix is created using glm::lookAt, positioning the light as the "camera" and orienting it toward the scene origin.

Depth information is stored in the shadow map during a dedicated rendering pass, which is then used in subsequent passes to determine shadowed regions.

A computer screen shot of white text

Description automatically generatedA screen shot of a computer code

Description automatically generatedA computer screen with text

Description automatically generated

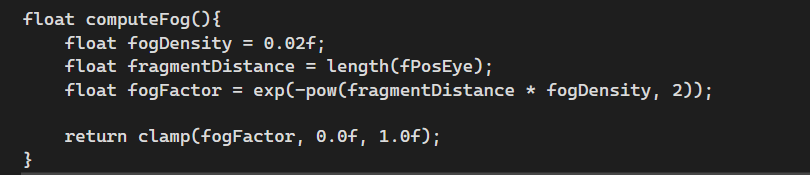
* **Animation:** To give a realistically look the objects should be separated from the scene object. In my case I made a rotating carousel which is made of 2 components. I separated the objects in blender and I set up the mass point in the origin in order to not translate the object to the origin in OpenGL.

The rotation is a simple operation in which I increment overtime the angle and afterwards I use a rotation matrix to rotate my model.

A computer screen with text on it

Description automatically generated

**Effects in the scene:**

1. **Fog:** The project integrates a fog effect to enhance depth perception and realism by simulating atmospheric scattering. The fog is computed in the shader using the computeFog function, which calculates the fog intensity based on the fragment's distance from the camera. This is achieved using an exponential formula where the fog density parameter (fogDensity) controls how quickly the fog intensifies with distance. The resulting fog factor is clamped between 0.0 and 1.0 to ensure proper blending, where 0.0 represents no fog and 1.0 represents complete fog coverage.

1. A screen shot of a computer program

   Description automatically generated**Transparent objects**: When rendering a transparent object, the activ\_tr variable is set to true, indicating that transparency is active. The shader is updated to reflect this by passing activ\_tr as a uniform to the fragment shader. In the rendering code, depth writing is disabled using glDepthMask(GL\_FALSE) to avoid conflicts with the depth buffer, ensuring that transparent objects do not overwrite objects behind them. Transparency is handled via OpenGL's blending functionality (glEnable(GL\_BLEND)) with the glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA) blend mode, which blends the object’s color based on its alpha value. After drawing the object, depth writing is re-enabled, and blending is disabled to restore the default state for subsequent non-transparent objects.

A screenshot of a computer program

Description automatically generated

1. **Fragment discarding:** To optimize the rendering of trees in the scene and achieve a more natural appearance, the project utilizes fragment discarding for tree models, which are represented as planes with textures. These planes were created in Blender and imported into the project with appropriate textures, including an alpha channel for transparency. During rendering, the fragment shader checks the alpha value of each pixel (stored in the colorFromTexture.a component). If the alpha value is below a certain threshold (in this case, 0.1), the fragment is discarded using the discard command.

A screen shot of a computer

Description automatically generated This ensures that any part of the texture that is nearly transparent or fully transparent is not drawn, effectively removing unnecessary fragments from the final image and leaving only the visible parts of the tree.

1. **Point light:** The point light system in your project is designed to handle multiple point lights that illuminate the scene. Each point light has a position and color, and the lighting calculations are based on the distance from the light source to the surfaces in the scene. The point light calculations are done in the fragment shader by iterating over all active point lights (in this case, 3 points light are defined).

A computer screen shot of a black screen

Description automatically generatedFor each point light, the shader computes the attenuation factor based on the distance between the light and the fragment. The attenuation is calculated using a combination of constant, linear, and quadratic terms, which simulate how light diminishes over distance. This attenuation factor is applied to the ambient, diffuse, and specular components of the lighting.

A screen shot of a computer program

Description automatically generated

1. **Rain effect:** The rain effect is implemented with a Rain structure that holds the x, y, and z coordinates for each raindrop's position, as well as the velocity of the drop. In the init\_rain() function, a specified number of raindrops (determined by the parameter nr) are generated with random positions and velocities. The x and z coordinates are randomly placed within a large range (5000), and the y coordinate is initialized between 20.0f and 70.0f, ensuring that the drops start above the scene. The velocity is also randomly assigned within a range to create variability in the falling speed.

In the update\_raindrops() function, the positions of all raindrops are updated by decreasing their y position based on their velocity, simulating the falling effect. If a raindrop goes below a threshold (y < -1.0f), it is reset to a new random position above the scene, giving the effect of continuous rain. The drops are then drawn in the draw\_raindrops() function.

The rain also has a sound that it. A black screen with white text

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. A screen shot of a computer program

   Description automatically generatedA computer screen with text on it

   Description automatically generated**Teleportation effect:** In this system, teleportation is implemented via a projectile (represented by a magical object) that acts as a beacon. When the player presses the N key, a projectile is fired, and when the player presses the 0 key, the player teleports to the last fired projectile.

A screen shot of a computer code

Description automatically generated

**Keybinds for actions:**

Basic Movement: WASD

Rotation of the light source: J L

Mouse movement

Shooting projectile: N

Teleport to the projectile: 0

Toggle rain (ON/OFF): R

Toggle point light (ON/OFF): P

Toggle second shader (new light): 1

Toggle wireframe (gl\_line) (ON/OFF): 2

Toggle pointframe (gl\_point) (ON/OFF): 3

Toggle presentation mode (ON/OFF): 4

*Specification of the scene:*

*The scene I’ve decided to make represents a post-apocalyptic world with a few desolate sceneries. The world is made of an almost empty establishment with a few survivors, anomalies and zombies and mutants.*

*A city in fog with buildings in the background

Description automatically generated*

*A building in a foggy field

Description automatically generated*

*A group of people walking on a road

Description automatically generated*

*Conclusion and future developments:*

For future developments I would like to implement the movement of the soldiers and the mutants. Implement more sounds for an eerie experience in the scene. Extends the establishment with an bigger city with more survivors and more ferocious monsters.

*Bibliography:*

* Cosmin Nandra Youtube videos
* <https://sketchfab.com/search?q=abandoned&type=models>
* <https://learnopengl.com/Getting-started/Camera>
* <https://stackoverflow.com/questions/8804880/use-playsound-in-c-opengl-to-play-sound-in-background>
* <https://stackoverflow.com/questions/1617370/how-to-use-alpha-transparency-in-opengl>