Final Practice

Project Documentation

Xavier Canals Hernáez - 15/01/2025

Index

General Description

This project is a graphical implementation in C++ that uses **OpenGL** to render 3D graphics, aiming to demonstrate the use of technologies such as **SDL2**, **GLAD**, and **SOIL2** for window management, OpenGL context handling, and texture loading. The project's goal is to illustrate fundamental concepts of graphics programming, such as shader management, textures, lighting, 3D meshes, and advanced visual effects like post-processing.

The project is structured modularly, with each class handling a specific aspect of the rendering process. The classes provide a solid foundation for experimenting and expanding with more graphical effects and optimization techniques.

Project Architecture

The project architecture is composed of different modules that manage specific aspects of graphic rendering and user interface interaction. The main modules are as follows:

Class Window

Responsibility: manages the creation and handling of the SDL window, the initialization of the OpenGL context, and the swapping of window buffers.

Dependencies: SDL2, GLAD.

Key Methods:

- **Window:** constructor that initializes the window and the OpenGL context.
- **swapBuffers:** swaps the window buffers to display the rendered content on screen.

Class Texture

Responsibility: handles 2D textures and cubemaps. It loads textures from image files and associates them with OpenGL objects.

Dependencies: SOIL2, GLAD.

Key Methods:

- **loadImage:** loads an image from a file into a ColorBuffer object.
- **createTexture2D:** creates a 2D texture from a loaded image.
- **createTextureCubeMap:** creates a cubemap (3D texture) from six images.

Class Shader

Responsibility: manages the loading and compilation of shaders (both vertex and fragment shaders) used in the OpenGL rendering pipeline.

Dependencies: GLAD.

Key Methods:

- **use:** activates the current shader for use in rendering.
- **compileShaders:** compiles the vertex and fragment shaders from the provided source code and links them into a shader program.

Class Terrain

Responsibility: represents a 3D terrain that can be rendered. It is responsible for generating vertex coordinates and corresponding texture coordinates, and for applying a shader to draw it on screen.

Dependencies: GLAD, GLM, Shader, Texture.

Key Methods:

- **render:** renders the terrain using the defined shader and textures.
- **resize:** adjusts the camera projection when the window is resized.

Class Lighting

Responsibility: manages the light sources in the scene. It allows adding directional, point, or area lights and controlling their characteristics like color, intensity, and position.

Dependencies: GLAD, GLM, Shader.

Key Methods:

- **Lighting:** default constructor for the Lighting class. Initializes the light properties with default values.
- **configureLight:** configures the lighting parameters in the given shader program by setting uniform values.

Class MeshLoader

Responsibility: loads 3D models (meshes) from files, such as **OBJ** or **FBX** formats, and creates the corresponding vertex and texture buffers for OpenGL rendering.

Dependencies: GLAD, Assimp.

Key Methods:

- **Render:** renders the mesh with the specified transformations and camera.
- **loadMesh:** loads the mesh from a file and sets up the vertex buffers.

Class Postprocess

Responsibility: applies visual effects on the scene after it has been rendered, such as blur, light effects, or post-processing using shaders.

Dependencies: GLAD, GLM, Shader.

Key Methods:

- **buildFramebuffer:** initializes the framebuffer and associated textures.
- **renderFramebuffer:** renders the post-processed image to the screen.

Class Skybox

Responsibility: represents a spherical or cubical sky that is rendered as the background of the scene. A cubemap texture is used to create a distant sky or landscape effect.

Dependencies: GLAD, Texture.

Key Methods:

- **Skybox:** constructor that initializes the skybox by loading a cube map texture.
- **render:** renders the skybox using the provided camera.

Class Scene

Responsibility: manages the organization of 3D objects in the scene. It handles the management of various elements like lights, cameras, and meshes, and coordinates their rendering.

Dependencies: Lighting, MeshLoader, Camera, Texture.

Key Methods:

- **update:** updates the scene (handles camera movement and object updates).
- **render:** renders the scene's objects (models, terrain, skybox, etc.).

Technical notes

Shaders

- Shaders are loaded and compiled within the Shader class. It also handles the assignment of uniform variables, such as transformation matrices.
- The project uses a basic vertex shader and fragment shader, though they can be extended for more advanced visual effects.

3D Mesh Loading

- The MeshLoader class allows loading 3D models from external files and converting them into meshes that can be rendered in the scene.
- It is possible to load simple 3D models (complex scene objects are still not possible) into the scene that will be texturized with their original albedo texture (for the moment the program is not able to apply multiple textures).

Terrain Rendering

- The terrain is generated from a mesh of vertices and texture coordinates are assigned to each vertex. The fragment shader then applies the texture to simulate a 3D terrain.
- It works fine but there is a problem when rendering it with triangles. There is a commented code line that renders it using a *line strip* which shows the terrain correctly, but for the moment using triangles shows the terrain not so good due to a problem with the *index vector*.

Camera

- It is possible to control the camera to navigate the scene.
- Mouse is used to control Camera direction while WASD keys are used to move it.

Texture Management

- The project uses SOIL2 to load textures from image files. Textures are loaded as either 2D maps or cubemaps, depending on the texture type required.
- Textures are assigned to OpenGL objects using the Texture class, and they are managed using the OpenGL-generated texture identifiers.

Transparency

- It is possible to apply transparency to meshes by applying a float between 0 and 1 to the last attribute of the mesh's constructor. The float will determine the level of opacity of the mesh with 1 being the value set to a completely opaque mesh.
- There is a bug that, even if a mesh is transparent and allows the user to see through it, when it is between the skybox and the user's camera, it becomes completely invisible, probably due to a rendering error in the skybox itself.

Lighting and shadows

- The Lighting class allows managing various light sources in the scene, such as directional and point lights. Lights affect how objects are illuminated in the scene.
- For the moment lighting has been programmed but there are still difficulties. Lighting affects the color of non-textured objects but in the same way, as if the normals of the vertex were not obtained correctly.

Skybox

- A skybox has been added to the scene. It consists of a cube texturized by 6 different .png, one for each side of the cube.

Post-Processing Effects

- The PostProcess class is used to apply effects on the final image after the scene has been rendered, allowing techniques like blur or color correction.
- Despite the code having been implemented, there are problems when applying them to the scene.

Additional Considerations

1. **Compatibility:** the project is designed to be compatible with systems that support OpenGL 3.3 or higher. The OpenGL features used are standard and should work on most modern platforms.

2. **Optimization:** although the project has been designed to be simple and demonstrate basic concepts of graphic programming while seeking clarity and optimization, there is still room for improvement.

3. **Extensions:** This project can easily be extended to include more graphical features, such as advanced lighting, shadows, perspective cameras and more complex visual effects.

Conclusion

This project provides a solid foundation for understanding how to interact with OpenGL using C++, as well as offering a practical example of how to create and render a 3D terrain with textures. Through the provided classes, advanced graphics concepts such as shader management and efficient resource handling in a 3D context can be explored.