Lab2_assignment_report

OpenGL Code Walkthrough - 3D Object Drawing with Axes and Camera Movement

This code demonstrates how to render a scene with 3D objects using OpenGL, with an emphasis on creating an interactive environment featuring cube drawing, camera movement, dynamic rotation, and axis visualization.

1. Setting Up OpenGL and GLFW

- **Libraries Used**: The program includes several important libraries:
  - `glad`: To load OpenGL functions.
  - `GLFW`: For window creation and management.
  - `glm`: A mathematics library for matrix transformations.
  - Custom shader management (`shader.h`) and basic camera utilities (`basic_camera.h`).
- **Constants**:
  - `SCR_WIDTH` and `SCR_HEIGHT` define the window dimensions.
  - The **camera** and transformation parameters are declared globally to facilitate dynamic interaction.

2. Camera and Transformation Variables

- **Camera Setup**: A `BasicCamera` object is initialized to navigate the scene.
  - The camera is set to start at `(3.0f, 3.0f, 3.0f)` with a target of the origin `(0.0f, 0.0f, 0.0f)`.
  - It is adjustable through keyboard inputs (`WASD`, `R`, `E`) for movement and arrow keys for rotation.
- **Transformation Variables**:
  - **Translation** (`translate_X`, `translate_Y`, `translate_Z`): Controls object positioning.
  - **Rotation Angles** (`rotateAngle_X`, `rotateAngle_Y`, `rotateAngle_Z`): Handles rotation along the respective axes.
  - **Scaling Factors** (`scale_X`, `scale_Y`, `scale_Z`): Alters the size of objects.

3. GLFW Initialization

- The main function begins by initializing **GLFW** and setting the OpenGL version.

- A GLFW window is created using the `SCR_WIDTH` and `SCR_HEIGHT` values.

4. Shader Setup

- **Shader Program**:
  - Vertex and fragment shaders (`vertexShader.vs`, `fragmentShader.fs`) are compiled and linked into a shader program.
  - The shader program handles both the cube drawing and the axes drawing.

5. Cube Vertex Data

- **Vertex Array Object (VAO) and Vertex Buffer Object (VBO)**:
  - **Cube Vertices**: Defined for a cube, including both position and color attributes for each vertex.
  - **Indices**: Defined for indexing into the vertices to create cube faces.
  - **VAO/VBO Setup**: Configures position and color attributes using `glVertexAttribPointer`.

6. Render Loop

The main rendering loop runs while the window is open.

- **Clearing the Screen**: The screen color is set to `(0.2f, 0.3f, 0.3f, 1.0f)` and cleared on every frame.

6.1 Camera and Projection Setup

- **Projection Matrix**: A perspective projection is set using `glm::perspective` with the camera's zoom level.
- **View Matrix**: Calculated using the `BasicCamera`'s `createViewMatrix` method to reflect the current camera position and orientation.

6.2 Drawing Axes

- **Axis Drawing**:
  - Axes are drawn for better visualization of the scene.
  - The axes are drawn using lines with different colors (red, green, blue) to denote the X, Y, and Z axes respectively.

6.3 Drawing Objects

Several objects are drawn in the scene, including a floor, walls, a table, and a chair.

- **Drawing Function (`drawCube`)**:
  - The `drawCube` function encapsulates drawing operations for all the cubes.
  - **Transformations Applied**:
    - A parent transformation (`parentTrans`) is used for global transformations of all objects.

        - For each object, local transformations (position, rotation, scaling) are applied.
- **Cube Objects**:
    - **Table**: Consists of a rectangular tabletop and four cylindrical legs. The table surface has a wood-like color, while the legs have a metallic look.
    - **Chair**: The chair features a wooden seat with a fabric backrest. The legs are darker in color to differentiate from the seat.
    - **Walls and Floor**: Walls and floor are represented by flattened cubes with different scaling.

6.4 Dynamic Fan Animation

- **Ceiling Fan**:
    - A ceiling fan is dynamically rotated using an angle (`fanRotateAngle_Y`) which is continuously updated based on `deltaTime`.
    - The fan has three blades, each colored differently (red, green, blue), to simulate a rotating fan.

7. Input Handling (`processInput` Function)

- **Exit Window**: Pressing `ESC` closes the window.
- **Object Transformations**:
    - **Translation**: Use keys `I`, `K`, `J`, `L`, `O`, `P` to adjust X, Y, Z translations.
    - **Rotation**: Use `X`, `Y`, `Z` keys to rotate along the respective axes.
    - **Camera Movement**: The camera can be moved in different directions using `WASD`, `R`, and `E` keys.
    - **Camera Rotation**: The arrow keys (`UP`, `DOWN`, `LEFT`, `RIGHT`) are used to adjust the pitch and yaw of the camera. Keys `1` and `3` are used for roll adjustments.

8. Framebuffer Resize and Scroll Callback Functions

- **Framebuffer Resize**:
    - The `framebuffer_size_callback` function ensures that the viewport matches the new window size when resized.
- **Scroll Callback**:
    - The scroll wheel is used to zoom in or out by adjusting the camera's field of view.

9. Summary and Features

This project provides an interactive 3D environment created using OpenGL, showcasing multiple transformation techniques and dynamic camera movement.

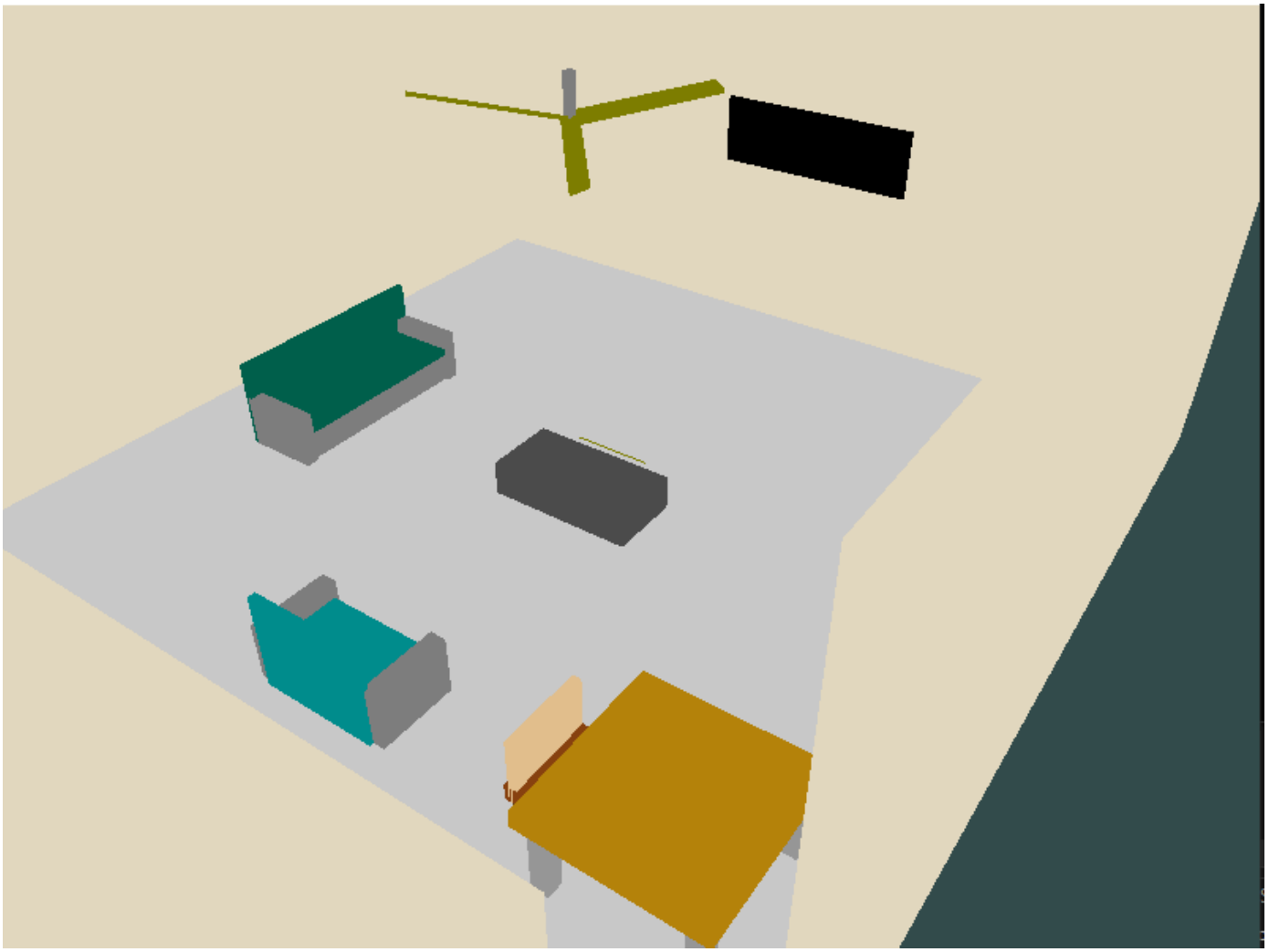
- **Objects Drawn**:
    1. **Floor and Walls**: Light-colored floor and beige walls create an enclosed room effect.

2. **Table and Chair**: Wood and metal textures differentiate parts of the furniture.
3. **Sofas**: Two different colored sofas are drawn to add realism.
4. **Ceiling Fan**: The fan rotates dynamically, adding a touch of realism to the scene.

- **User Interaction**:
  - The user can move, rotate, and scale objects, as well as navigate the scene with the camera.
  - The dynamic fan blades provide an animated aspect to the otherwise static environment.

Images

Below are some reference screenshots of the scene:



Conclusion

This code walkthrough demonstrates how to render complex 3D scenes using OpenGL with an emphasis on interactive camera movement and object transformations. By combining **glad**, **GLFW**, **glm**, and custom shaders, a comprehensive 3D scene is constructed. This project serves as a foundational step for building more sophisticated interactive 3D environments in OpenGL.