Detailed Analysis of Fluid Simulation Code

Neo

February 21, 2025

1 Introduction

This document explains the coding aspects of a C++ OpenGL fluid simulation program. It focuses on the rendering pipeline, shader implementations, mesh generation, and additional features such as frame recording and input management.

2 Code Structure Overview

The program is structured as a C++ application that uses several libraries:

- **GLFW** for window creation, input handling, and context management.
- **GLEW** to manage OpenGL extensions.
- **GLM** for vector and matrix operations.
- **stb_image_write** for saving rendered frames as PNG files.

The main functionality includes a water simulation that updates a mesh using sinusoidal wave functions and renders it with custom shaders. The code is modular, with distinct sections for shader compilation, mesh generation, simulation updates, rendering, and frame recording.

3 Shader Compilation and Linking

Two main functions manage shader creation:

- `compileshader`: Compiles a given shader (vertex or fragment) from source code. It performs error checking and outputs any compilation issues.
- `createshaderprogram`: Links the compiled vertex and fragment shaders into a complete shader program, and then deletes the individual shaders.

This process ensures that the shader program is valid before it is used for rendering.

4 Vertex Shader Analysis

The vertex shader performs standard transformation tasks:

- It transforms vertex positions from object space to world space using the model matrix.
- It applies view and projection matrices to convert the world-space coordinates into clip space.
- It transforms vertex normals using the upper-left 3x3 portion of the model matrix, ensuring that lighting calculations are correct.
- It passes the transformed world-space position and normal to the fragment shader.

This shader is essential for setting up the geometry for proper lighting and projection.

5 Fragment Shader Analysis and Toon Shading

The fragment shader implements a Lambertian diffuse lighting model combined with a toon (cel) shading effect:

- **Lambert Lighting:** The shader computes a diffuse component by taking the dot product of the normalized surface normal and the direction vector from the fragment to the light source. This yields a basic Lambertian reflection.
- **Toon Shading:** To create a stylized, cartoon-like look, the shader quantizes the diffuse (Lambert) value. It does this by multiplying the diffuse value by a uniform step count, applying the `floor` function, and then normalizing the result. This produces distinct lighting bands rather than a smooth gradient.
- **Color Interpolation:** The quantized light level is used to interpolate between a defined dark color and light color, determining the final fragment color.

The combination of these techniques results in a rendering effect that is both simple and visually distinctive.

6 Water Simulation and Mesh Generation

The water simulation is encapsulated in a custom class that:

- Builds a mesh representing a water volume with both top and bottom surfaces, as well as side faces.

- Uses two sinusoidal functions to update the heights of the top surface vertices, simulating wave motion.
- Offsets the bottom surface by a constant thickness relative to the top surface.
- Computes normals for the top and bottom surfaces using central differences. These normals are crucial for the lighting calculations performed in the fragment shader.

This approach allows for a dynamic water surface that updates in real time.

7 Frame Recording and Post-Processing

The code includes functionality for recording the simulation:

- When triggered (by pressing the S key), the simulation begins recording frames.
- Each frame is captured from the OpenGL framebuffer, flipped vertically to correct the coordinate origin, and saved as a PNG file.
- After a preset duration (16 seconds), the program uses an external `ffmpeg` command to compile the saved frames into a video file.

This feature provides a simple way to capture and review the simulation output, by for example combining all the images to a GIF.

8 Input and Window Management

Input and window management are handled using GLFW:

- A key callback function listens for the Escape key (to exit the simulation) and the S key (to initiate frame recording).
- A framebuffer size callback updates the OpenGL viewport and aspect ratio whenever the window is resized.

These functions ensure that the application remains responsive and correctly scaled across different window sizes.

9 Conclusion

The fluid simulation code integrates several standard OpenGL techniques to produce a dynamic water simulation. A vertex shader handles geometric transformations and normal recalculations, while a fragment shader applies Lambertian diffuse lighting with a toon shading effect. Additionally, the program features modular shader compilation, dynamic mesh updates, and a built-in frame recording mechanism. The overall structure emphasizes clarity and separation of concerns, making the code both functional and adaptable.

Tl;dr

Tl;dr: The code is a C++ OpenGL fluid simulation that builds and updates a water mesh. It uses a vertex shader for standard transformations and normal computations, and a fragment shader that implements Lambert lighting with toon shading. It also features frame recording and responsive input and window management.