Amanda Clement / 40136259

COMP 371: Project Report

April 4th, 2023

Project Report: Real-time Rendering of Heterogeneous Fog

Description

The motivation behind my project was to find a middle-ground between homogeneous fog, which lacks a sense of realism due to its density being constant over the entire scene, and volumetric fog, which appears more realistic but has high performance costs. Heterogeneous fog, on the other hand, has a more realistic appearance than homogeneous fog as it accounts for variable fog density, while being less expensive to render than volumetric fog. As such, my goal was to render heterogeneous fog using OpenGL.

Approach

The heterogeneous fog model augments exponential fog by integrating Perlin turbulence into the equation, which involves the layering of Perlin noise octaves to achieve a greater amount of detail and shape than Perlin noise alone. The resulting fog equation was then used to blend the color of the fog with the colors in the scene. Finally, GUI elements and simple camera controls were added to allow the user to explore the real-time fog rendering.

Implementation: Meeting the Objectives and Challenges Faced

Objective 1: Implement Perlin noise, generating noise octave textures with it

The first stage of the project was to write a Perlin noise algorithm, based on Ken Perlin's *Improved Perlin Noise* implementation [1]. Although the implementation of the noise went as planned, the usage of the noise to generate textures introduced some complications due the visible tiling effect across the texture's surface. Once I had identified that the issue was related to a poor choice of cell size, I was able to fix the texture generation algorithm by experimenting with different sizes to prevent visible repetition.

Objective 2: Implement Perlin turbulence by layering the Perlin noise octaves

Once the Perlin noise algorithm was set up to generate noise textures composed of layered noise octaves, the next step was to use them to create Perlin turbulence using the following equation:

$$\textit{Perlin turbulence}(x) = \sum_{i=0}^{N-1} \frac{noise_i(x)}{2^i} \quad [2]$$

After trying out different combinations of octaves, the best results came from using four Perlin noise textures composed of 4, 8, 16 and 32 octaves respectively. Deciding how many layers and which octave counts to use was an iterative process as it was affected by multiple factors in the next stages such as the cube sizes and positions within the scene, texture size, fog density, etc.

Objective 3: Compute the fog factor by implementing Perlin turbulence into the exponential fog equation

The fog factor was computed using the following fog equation:

$$f = e^{-(d*g*turbulence(x))^{\wedge}2} \quad [2]$$

Where *f* is the fog coefficient, *d* is the distance between the considered point and the camera or viewer, and *g* is the fog density coefficient. Although the equation is not very complex, the fact that it works with distance, density and turbulence made it difficult to find the right balance between the textures and scene components to get the desired fog effect (i.e. experimenting with different Perlin octaves, choosing an appropriate fog density range, placing the camera at a reasonable distance from the geometry in the scene).

Objective 4: Create an OpenGL scene and blend the fog color with the scene colors

Most of the obstacles I encountered during the development of my project occurred in this stage. The challenge was figuring out what the best method was to apply the fog to the geometry in the scene. My first attempt was to layer the noise textures on top of a base texture, and apply the resulting texture to each surface of the geometry. The opacity of each noise texture would be determined by the Perlin turbulence formula. However this idea would only work if the base texture is black, or some other solid color if the Perlin texture is generated in shades of that color rather than grayscale. To work around this issue, I tried to modify the Perlin noise texture generation function to include an alpha channel so that any parts of the texture that are not white or light gray would be transparent, so that when it gets applied over the base texture of the geometry, those areas would show the base texture instead of black. Although this approach may have been feasible, I did not manage to get it working.

Finally, the approach I ended up taking was to use the fog factor to blend the background color (which could instead be a texture) with the fog color in the fragment shader. I followed a similar approach as proposed in Dorota Zdrojewska's research paper *Real time Rendering of Heterogeneous Fog Based on the Graphics Hardware Acceleration* [2]. However, in my implementation, I use the position of the vertices relative to the camera as the basis for my noise texture coordinate calculations. After much experimentation, this gave the most realistic result as it makes it so that the fog appears to be applied across the entire scene, rather than to each geometry surface individually. When the final texture is applied to each surface, we get visible texture seams along the cube edges. Another attempt I took was to try mapping the texture over the entire cube as one entity, but this resulted in a loss of definition in the cube edges, making it look 2-dimensional even when rotated. In either case, those attempts made the effect look artificial as the fog texture was applied to each cube individually, and not shared across the entire scene so there was discontinuity in the pattern. Using the position of the vertices relative to the camera in the calculations gave the best output.

Objective 5: Add features allowing user to interact with the fog and the scene

The last stage was to add user controls. I only added one camera control allowing the user to press the arrow keys to move the camera forwards, backwards and from side to side. I used the *imgui* library for the UI elements [3]. This stage mostly involved deciding which values to let the user have influence over and what the range of values should be.

Additional Challenges

Another challenge faced during the development of this project was to find a balance between all of the components that make up or interact with the fog. My initial strategy was to follow my objectives in a sequential order, but this approach became problematic when I tried to use the fog in the OpenGL scene. Although the components worked nicely independently, they looked odd in combination. This was especially true when adding the textures and animating the fog. A large part of the project development was spent experimenting with different combinations of values to get the desired fog effect.

Analysis of Results & Further Improvements

I was able to meet my objects and render heterogeneous fog, but can see areas for improvement. For instance, the animation of the fog is too repetitive and predictable, which may be noticeable to the viewer. A more sophisticated animation could make the fog appear more realistic. Also, although it was not specifically outlined as an objective of mine, the fog could be better demonstrated on an uneven surface like a terrain. By virtue of the cube being a single color and its surfaces being completely flat, the fog appears more artificial. However, for the purpose of the demonstration, applying the fog to a single-colored cube allows us to truly observe and analyze the result.

Resources

[1] Perlin, K. (2002, February 16). Improved Noise reference implementation. NYU Media Research Lab. Retrieved April 3, 2023, from https://mrl.cs.nyu.edu/~perlin/noise/

[2] Zdrojewska, D. Real Time Rendering of Heterogeneous Fog Based on the Graphics Hardware Acceleration. Retrieved February 7, 2023, from https://old.cescg.org/CESCG-2004/papers/34_ZdrojewskaDorota.pdf

[3] ocornut. OCORNUT/IMGUI: DEAR IMGUI: Bloat-free graphical user interface for C++ with minimal dependencies. GitHub. Retrieved March 20, 2023, from https://github.com/ocornut/imgui

[4] Learn OpenGL. Retrieved March 18, 2023, from https://learnopengl.com/

[5] Adrian's soapbox. Understanding Perlin Noise. (2014, August 9). Retrieved February 16, 2023, from https://adrianb.io/2014/08/09/perlinnoise.html