**OpenGL Interactive Scene Implementation Report**

# Introduction

This report explains a 3D interactive scene made with OpenGL. I combined many techniques into one shader, including PBR lighting, texture mapping, normal mapping, and shadow maps. The scene has a torus model loaded from an OBJ file and a wooden crate made with code. I used a geometry shader to add hair to the torus. The hair falls with gravity and has a color gradient. This project was a technical challenge and helped me learn modern OpenGL rendering.

# 2. Scene Overview

The models are the same as in CW1. But now, I added hair to the torus. The whole scene uses PBR rendering. Shadows are also added.

# 3. Graphics Techniques

## 3.1 Combining Shaders

In CW1, I used one shader for each technique (texture mapping, normal mapping, etc.). But now, I needed to add shadows. If I added shadow code to every shader, it would be messy and hard to maintain. So, I worked hard to combine all shaders into one. I used uniform options to choose which technique to use. All techniques share the same shadow code, so shadows appear on both the crate and walls.

When combining shaders, some variables became unused. I used a dummy variable trick. I added their values to the alpha channel of the output color. Since alpha is already 1.0, this does not change the result. This stops OpenGL from removing unused uniforms.

## 3.2 Shadow Maps

I added FrameBuffer and DepthFrameBuffer classes to store shadow maps. I also made a 2dSpirit shader to show the shadow map on screen. This helps me check if shadows are working correctly.

I added shadow.vert and shadow.frag shaders to render shadows.

Used PCF (Percentage Closer Filtering) to make shadows softer.

Added bias to avoid the "Peter-panning" effect (shadows floating above surfaces).

In normalMap.frag, I passed the light space matrix and shadow map. Then, I mixed the shadow with the color to create the final effect.

## 3.3 Hair with Geometry Shader

I made a geometry shader to create hair.

First, I extended the torus’s normals to make hair strands.

Then, I changed the color at the start and end of each hair to make a gradient.

I split each hair into 20 segments. Each segment starts where the last one ended, with a small offset in the Y direction. This makes the hair bend downward, like gravity is pulling it.

## 3.4 PBR (Physically Based Rendering)

To use PBR, I made a function:

PBRMaterial ConvertToPBR(vec3 Kd, vec3 Ks, float shininess)

This converts the old uMaterial (Phong) to PBR material.

Used GGX for normal distribution and Smith for geometry shading.

Replaced Phong shading with PBR code from research papers.

Added gamma correction to make lighting look more natural.

# 4. Interactive Features

New controls:

* Right-click + drag: Move camera view.
* WASD: Move camera position.
* +/- keys: Change cubemap reflection strength.
* ESC: Quit.
* Left/Right arrows: Change light rotation speed.
* Spacebar: Toggle light rotation.
* M key: Toggle shadow map debug view.
* These work by changing shader uniforms and updating object positions.

# 5. Challenges & Learning

Main difficulties:

* Combining shaders: Much harder than using separate shaders.
* PBR coding: PBR code is long, and shaders can’t use breakpoints. I had to tweak values and check the visuals.
* Shadow debugging: If the shadow matrix is wrong, it’s hard to find the mistake. I had to test carefully.

These challenges helped me understand OpenGL rendering, matrix math, and shader programming better.

# 6. Future Improvements

The project works well, but I can make it better:

* Add more lights for better shadows.
* Tweak PBR settings for more realism.
* Let PBR use textures (like roughness maps).

This project is a good start, and I will keep learning advanced graphics techniques.