1. Introduction & Scene Concept

This project developed an interactive 3D scene to demonstrate a comprehensive understanding of real-time graphics rendering. The scene features diverse .obj models (couch, aircraft, guitar, ground plane) chosen to test rendering capabilities such as texturing, Phong lighting, basic animation, and Screen-Space Ambient Occlusion (SSAO). The application targets WebGL 2.0 for modern browsers, achieving real-time performance.

2. Technical Design Summary

The environment was built using WebGL 2.0 and the glMatrix JavaScript library, contained within a single HTML5 document (FinalProject.html) with embedded GLSL shaders.

Asset Management: External .obj files for the couch, aircraft, and guitar were loaded via a custom asynchronous parser handling positions, normals, and texture coordinates. Diffuse textures (JPG/JPEG) were applied, with mipmapping for power-of-two textures and CLAMP_TO_EDGE for others.

Rendering Pipeline & SSAO: A multi-pass approach was used for SSAO:

1. G-Buffer Pass: Scene geometry was rendered to an FBO, outputting view-space normals (RGB) and packed view-space depth (Alpha) to a texture.
2. SSAO Calculation Pass: Using the G-Buffer, a full-screen quad shader reconstructed view-space positions. It then sampled a 64-sample hemispherical kernel in tangent space (oriented by the surface normal and a noise texture) to calculate an occlusion factor for each pixel.
3. SSAO Blur Pass: A 3x3 box blur was applied to the raw SSAO texture to reduce noise.
4. Main Lighting & Composition Pass: The scene was rendered to the canvas. Phong lighting (ambient, diffuse, specular) was calculated, with diffuse color from textures. The blurred SSAO factor modulated the ambient light component. This pass also handled toggling for wireframe or SSAO debug views.

Transformations & Interactivity: Standard MVP transformations were managed with glMatrix. The scene includes an interactive mouse-orbiting/zooming camera, time-based rotation for the aircraft, and UI buttons for rendering mode selection.

3. Shader Logic Explanation

Several GLSL shader programs drive the rendering:

- G-Buffer Shaders (gbuffer-vertex-shader, gbuffer-fragment-shader): The vertex shader transforms geometry to view space. The fragment shader packs view-space normals (RGB) and depth (A) into the G-Buffer texture.
- SSAO Calculation Shaders (ssao-vertex-shader, ssao-fragment-shader): The fragment shader reconstructs view position from G-Buffer depth. It then uses a TBN matrix (from surface normal and a noise texture) to orient a 64-sample kernel. It compares sample depths against G-Buffer depths to determine occlusion.
- SSAO Blur Shaders (ssao-blur-vertex-shader, ssao-blur-fragment-shader): These apply a 3x3 box blur to the SSAO output texture to soften the effect.
- Main Lighting Shaders (vertex-shader, fragment-shader): The vertex shader performs MVP transformations. The fragment shader implements Phong lighting, samples diffuse textures, and applies the blurred SSAO factor to the ambient light term. It also handles conditional rendering for wireframe and SSAO debug modes.

4. What You Learned & Challenges Faced

This project provided invaluable experience in advanced real-time rendering.

Key Insights & Learning Experiences: The most practical insight was gained during SSAO debugging and tuning. Witnessing how significantly parameters like sampling RADIUS (in ssao-fragment-shader) impacted visual quality and artifacts like ghosting underscored that algorithm implementation is only part of the challenge; parameter understanding and effective debugging (e.g., outputting intermediate values as colors) are crucial. The project directly applied concepts from course modules on Transformations (Weeks 3 & 4), Lighting/Shading (Week 6), Texturing (Week 7), and Advanced Rendering (Week 9). Achieving 60 FPS with multi-pass SSAO, confirmed via Chrome's FPS meter, was encouraging, demonstrating the viability of advanced effects with careful implementation.

Challenges Encountered and Solutions:

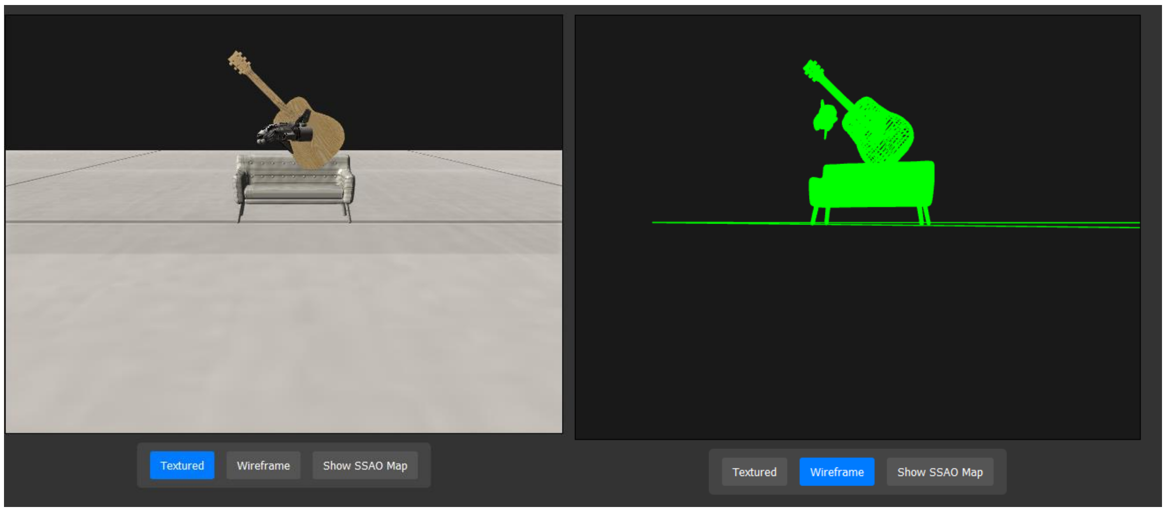
1. OBJ Model Integration: Significant scaling (e.g., couch by 0.001, aircraft by 0.1) and rotational adjustments were needed to integrate diverse models cohesively.
2. SSAO Ghosting Artifacts: An initial SSAO ghosting issue, where occlusion seemed to linger incorrectly (e.g., under the moved aircraft), was resolved by substantially reducing the SSAO RADIUS in ssao-fragment-shader (to around 0.15-0.3), localizing the sampling.
3. G-Buffer Accuracy: Ensuring correct packing and reconstruction of normals and depth in the G-Buffer (gbuffer-fragment-shader) and SSAO (ssao-fragment-shader) passes required meticulous debugging.

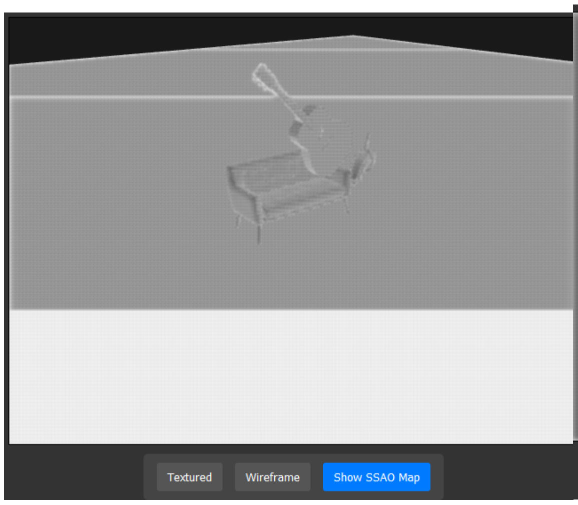
4. Complex Shader Debugging: Managing multiple interdependent shaders was challenging. Visually outputting intermediate shader values as colors was an indispensable debugging technique.

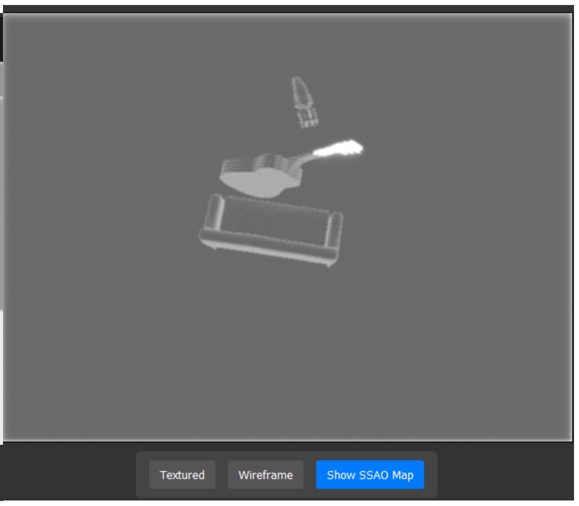
5. Future Improvements

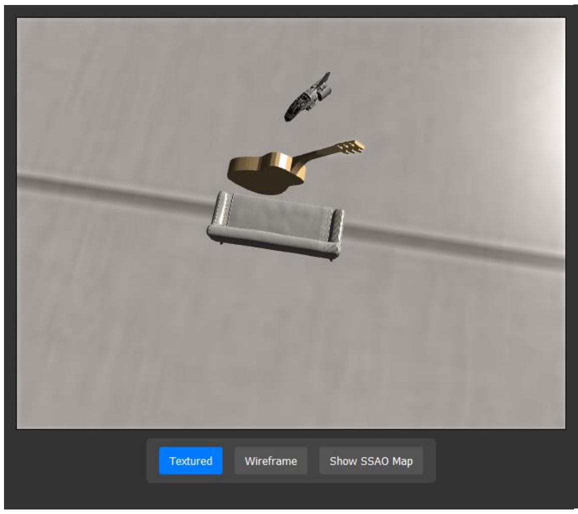
Given more time, implementing shadow mapping for direct shadows from the primary light source would be the next logical step to further enhance scene realism, complementing the ambient occlusion.

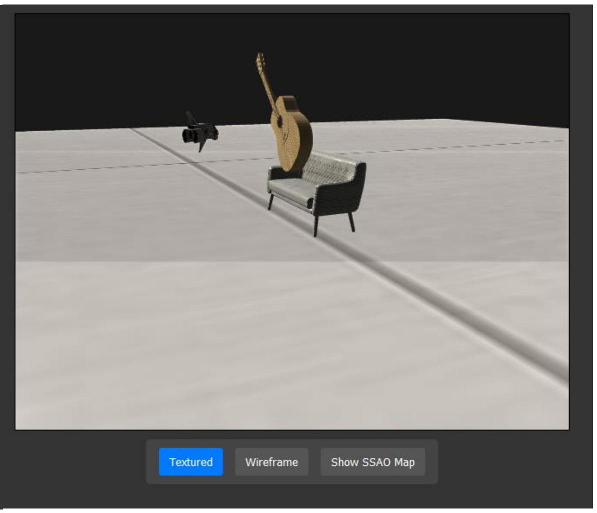
Screenshots:








Textured
Wireframe
Show SSAO Map


Textured
Wireframe
Show SSAO Map