CS 405 PROJECT2 REPORT

Zeynep Özeren

31941

Github Link: https://github.com/zeynepozeren1/CS405_Project2

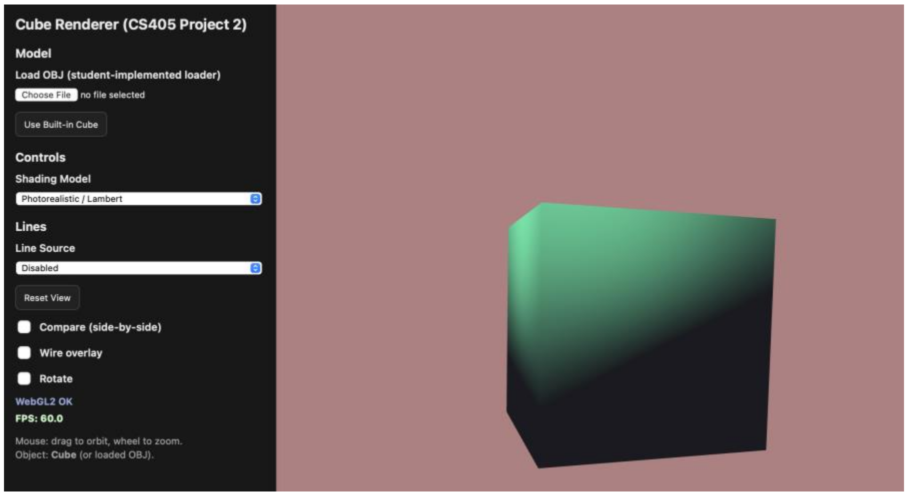
This project implements a WebGL2-based NPR renderer with a clean modular structure and an interactive UI for runtime switching between techniques. The user can select a shading model and line style while viewing the same scene in real time, and a side-by-side compare mode is available to evaluate stylistic differences under identical viewing conditions. To satisfy the "model loader" expectation, a student-implemented OBJ loader is included so the renderer can display either built-in geometry or externally loaded meshes. A live FPS counter is shown to verify real-time performance (typically $\geq$30 FPS).

For shading, Lambert diffuse is used as the photorealistic baseline, computed from max(N·L,0). Gooch shading replaces dark shadows with a perceptual cool-to-warm ramp by blending between user-selected cool (shadow) and warm (lit) colors based on N·L, improving form readability for technical illustration. Toon/cel shading quantizes continuous diffuse lighting into discrete tone bands using a 1D ramp (LUT) sampled by the lighting value, with the number of bands controlled by a "steps" parameter. For line rendering, multiple options are supported: an object-space silhouette outline using the flipped-hull approach (expanded backface pass), and screen-space edge detection using either depth discontinuities or normal differences via a post-process pass. As the required student-designed or significantly extended method, a procedural cross-hatching shader is implemented using tri-planar projection; hatch density increases in darker regions (lower N·L), and the user can adjust hatch density, line width, and strength from the UI.

The implementation is organized into small, focused modules. index.html defines the sidebar controls and canvas. src/main.js initializes WebGL, creates the renderer, loads a default mesh, attaches the OBJ file input, and runs the render loop while computing FPS. src/renderer.js contains the rendering pipeline, shader programs, uniform updates, optional multi-pass edge rendering, and the compare viewport logic. src/ui.js connects the HTML controls to renderer setters and toggles control visibility based on the active mode. src/objLoader.js parses OBJ text into indexed buffers (positions, normals, indices) and normalizes scale, while src/geometry.js provides built-in meshes and line indices for wire/edge rendering. Finally, src/gl.js handles WebGL2 context setup and validation, and src/utils.js provides compact matrix and math utilities to keep the rendering code clean.

How to run

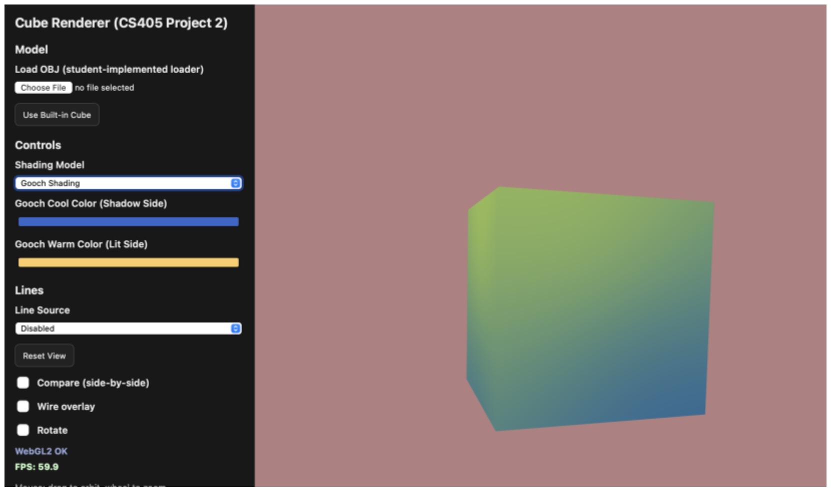
This project uses ES modules (`<script type="module">`), so it must be served from a local web server (it will not run via `file://`). In order to open we must use Open with Live Server.



no file selected
Shading Model: Photorealistic / Lambert
Line Source: Disabled
WebGL2 OK
FPS: 60.0
Mouse: drag to orbit, wheel to zoom.
Object: Cube (or loaded OBJ).

Figure 1: Initial state

The application starts in Lambert (photorealistic baseline) with lines disabled, showing the default model in the main viewport. The sidebar exposes runtime controls for shading model selection, line source, compare mode, and a student-implemented OBJ loader for loading external meshes. The FPS counter (≈60 FPS here) demonstrates real-time performance while interacting with the renderer.


Cube Renderer (CS405 Project 2)
Model
Load OBJ (student-implemented loader)
Choose File
Use Built-in Cube
Controls
Gooch Cool Color (Shadow Side)
Gooch Warm Color (Lit Side)
Lines
Reset View
Compare (side-by-side)
Wire overlay
Rotate

no file selected
Shading Model: Gooch Shading
Line Source: Disabled
WebGL2 OK
FPS: 59.9

Figure 2: Gooch shading

This figure shows Gooch shading, which replaces dark-to-bright shading with a cool-to-warm color transition driven by N·L. The cool (shadow) and warm (lit) tones are controlled interactively via the two color pickers, making it easy to tune the illustrative look at runtime. This demonstrates one of the required NPR shading techniques selectable from the UI.

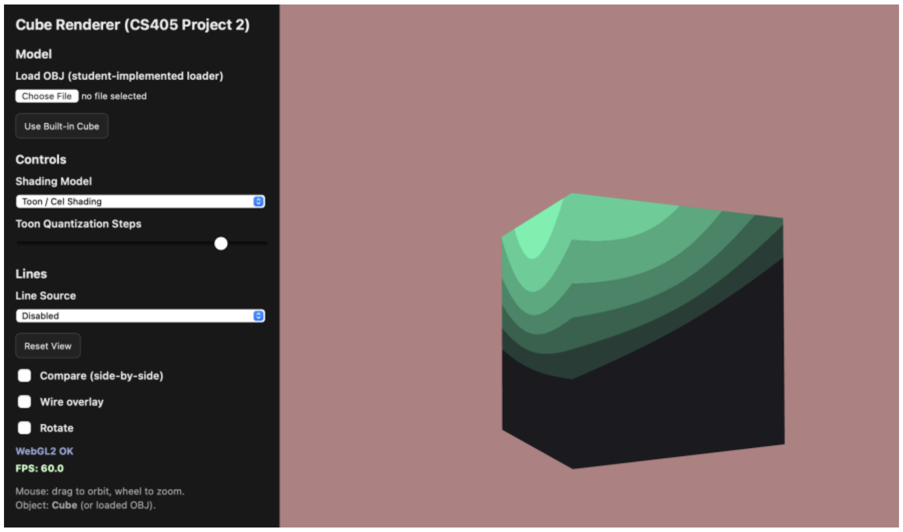

Cube Renderer (CS405 Project 2)
Model
Load OBJ (student-implemented loader)
Choose File no file selected
Use Built-in Cube
Controls
Shading Model
Toon / Cel Shading
Toon Quantization Steps
Lines
Line Source
Disabled
Reset View
Compare (side-by-side)
Wire overlay
Rotate
WebGL2 OK
FPS: 60.0
Mouse: drag to orbit, wheel to zoom.
Object: Cube (or loaded OBJ).

Figure 3: Toon / Cell Shading with high steps

Here the renderer applies toon shading by quantizing diffuse lighting into discrete tone bands using a ramp/LUT, producing a stylized cartoon-like appearance. With a high “steps” value, the surface shows many narrow bands, making the quantization effect clearly visible. This is another runtime-selectable NPR technique, and the FPS readout indicates the effect runs in real time.

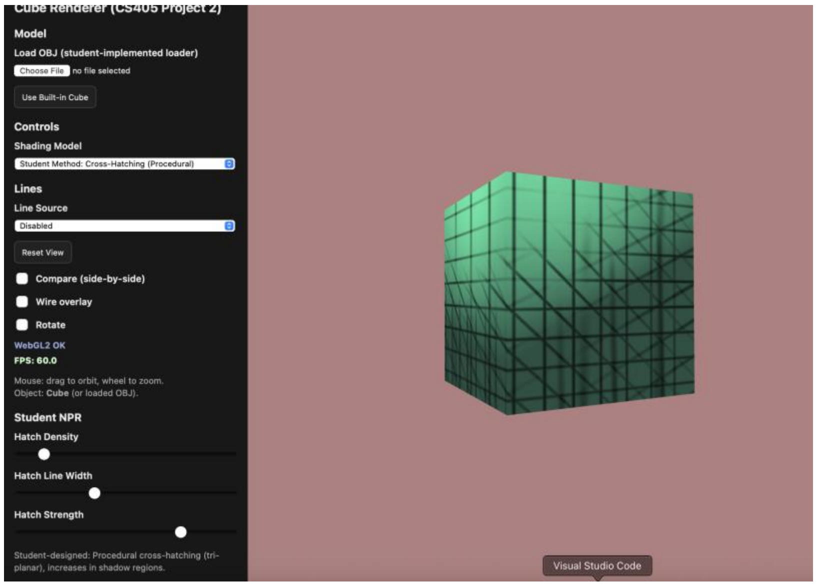
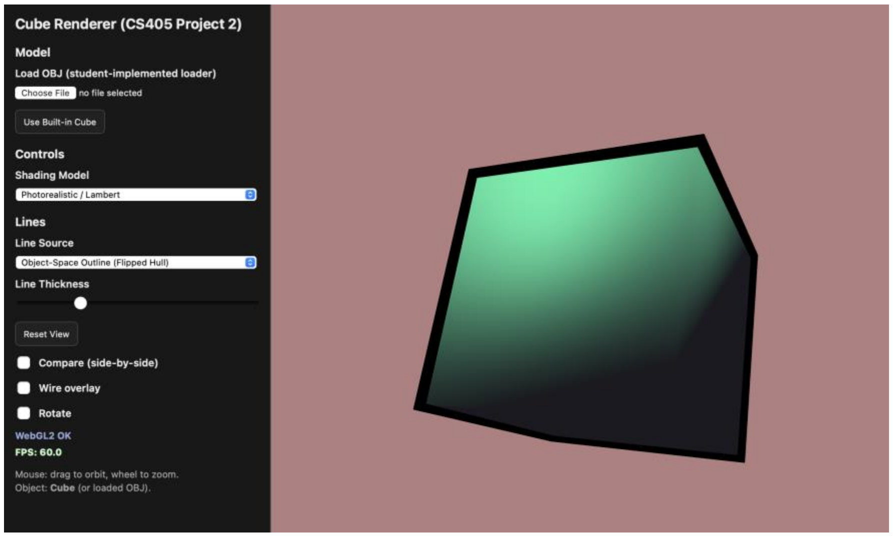

Cube Renderer (CS405 Project 2)
Model
Load OBJ (student-implemented loader)
Choose File no file selected
Use Built-in Cube
Controls
Shading Model
Student Method: Cross-Hatching (Procedural)
Lines
Line Source
Disabled
Reset View
Compare (side-by-side)
Wire overlay
Rotate
WebGL2 OK
FPS: 60.0
Mouse: drag to orbit, wheel to zoom.
Object: Cube (or loaded OBJ).
Student NPR
Hatch Density
Hatch Line Width
Hatch Strength
Student-designed: Procedural cross-hatching (tri-planar), increases in shadow regions.
Visual Studio Code

Figure 4: Student Method: Cross-Hatching

This figure shows the student-designed NPR technique: a procedural cross-hatching shader that represents tone using line patterns instead of smooth shading. The hatch pattern becomes visually denser in darker regions (lower N·L), which mimics traditional ink/etching style shading. The effect is fully interactive: hatch density, line width, and strength can be adjusted at runtime from the "Student NPR" controls, and it still runs in real time (FPS ≈ 60).

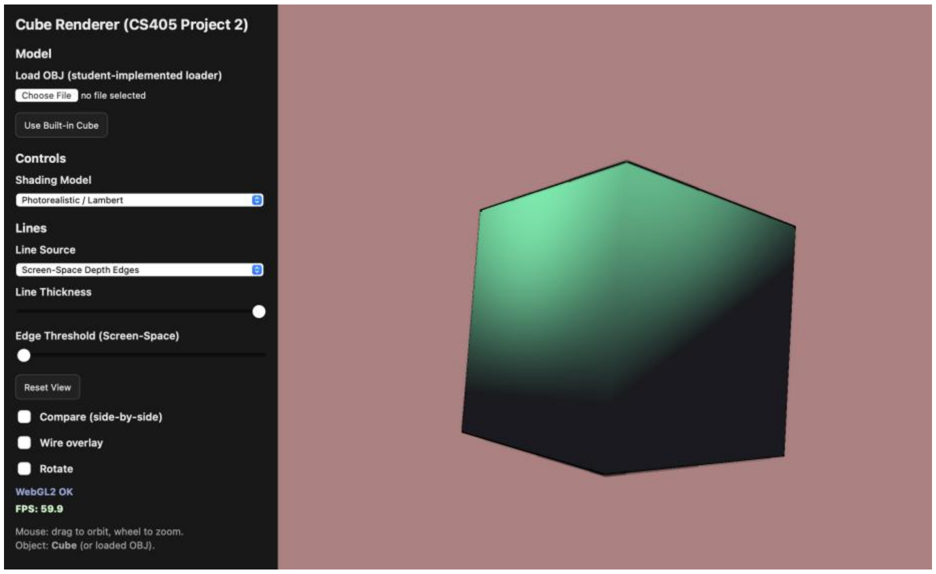

Cube Renderer (CS405 Project 2)
Model
Load OBJ (student-implemented loader)

Controls
Shading Model: Photorealistic / Lambert
Lines
Line Source: Object-Space Outline (Flipped Hull)
Line Thickness

WebGL2 OK
FPS: 60.0
Mouse: drag to orbit, wheel to zoom.
Object: Cube (or loaded OBJ).

Figure 5: Object-space outline with Line thickness slider

Here the renderer applies toon shading by quantizing diffuse lighting into discrete tone bands using a ramp/LUT, producing a stylized cartoon-like appearance. With a high "steps" value, the surface shows many narrow bands, making the quantization effect clearly visible. This is another runtime-selectable NPR technique, and the FPS readout indicates the effect runs in real time.

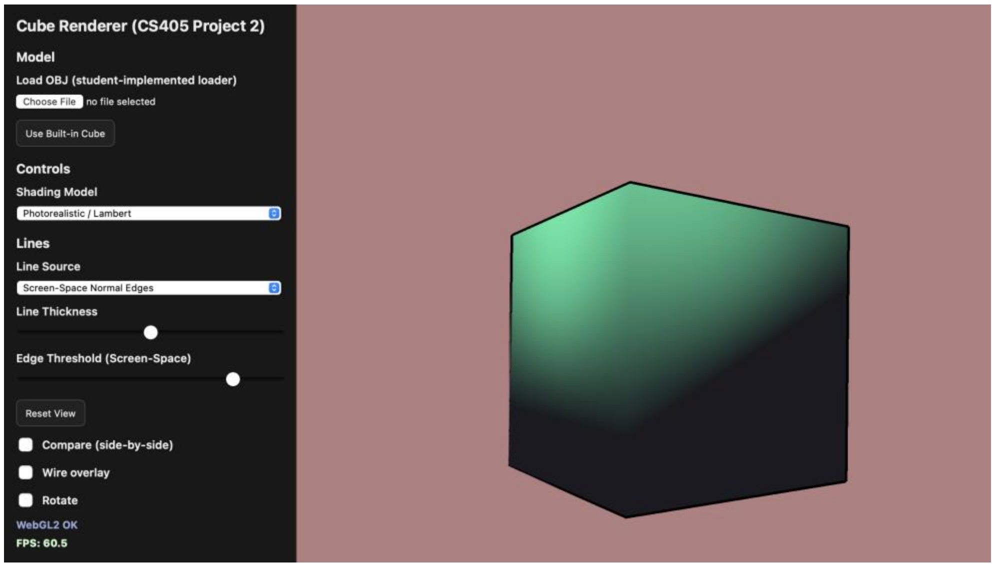

Cube Renderer (CS405 Project 2)
Model
Load OBJ (student-implemented loader)

Controls
Shading Model: Photorealistic / Lambert
Lines
Line Source: Screen-Space Depth Edges
Line Thickness
Edge Threshold (Screen-Space)

WebGL2 OK
FPS: 59.9
Mouse: drag to orbit, wheel to zoom.
Object: Cube (or loaded OBJ).

Figure 6: Screen-space Depth Edges

This figure demonstrates an object-space silhouette outline generated using the flipped-hull technique: the mesh is slightly expanded and rendered as a backface pass to create a bold black contour around the object. The outline thickness is controlled interactively with the “Line Thickness” slider, allowing quick tuning of the stylized inked look. Because it is computed in object space, the outline is stable and does not rely on post-processing.



no file selected
Shading Model: Photorealistic / Lambert
Line Source: Screen-Space Normal Edges
WebGL2 OK
FPS: 60.5

Figure 7: Screen-space Normal Edges

This figure shows screen-space edge detection based on **normal differences** rather than depth. The renderer first stores encoded normals (and depth) into an offscreen buffer, then a fullscreen pass compares neighboring normal samples; large changes indicate surface feature boundaries and are drawn as outlines. The “Line Thickness” and “Edge Threshold” controls adjust the width and sensitivity of the detected edges, making this method effective at emphasizing shape details even when depth changes are small.