Homework 4: PyGame + OpenGL — Textured Floor, Lighting, and Transparency

Files Provided

- main.py
- floor.jpg

Part A — Setup and Run (Student must demonstrate it runs)

Task A1. Create a new folder and place the following inside:

- main.py
- chess.png

Task A2. Install dependencies (use a virtual environment if you know how):

- pygame
- PyOpenGL
- Pillow (PIL)

Task A3. Run the program and verify you can do all of the following:

- See a chess-textured floor.
- See two spheres: one opaque and one transparent.
- Use mouse controls to orbit, pan, and zoom (based on the code controls).
- Use keyboard controls to move the light source (W/A/S/D/Q/E or as implemented).

Deliverable A. Submit:

- One screenshot showing both spheres and the floor.
- One short screen recording (10–20 seconds) demonstrating camera control and moving the light.

Part B — Understanding Questions (Answer in your own words)

Write concise answers (3–8 sentences each). Include code references (function names and key OpenGL calls).

1) Libraries and Roles

- What does PyGame do in this project, and what does PyOpenGL do?
- Why do we need Pillow (PIL) here instead of loading the image directly with OpenGL?

2) Rendering Pipeline Basics (Fixed-Function)

- Explain the difference between GL_PROJECTION and GL_MODELVIEW.
- Where does gluPerspective() belong, and why?

3) Textures

- In load_texture(), explain the purpose of: glTexImage2D, GL_TEXTURE_WRAP_S/T = GL_REPEAT, and glTexParameteri(... GL_LINEAR).
- Why do we flip the image vertically (Image.FLIP_TOP_BOTTOM)?

4) Lighting

- What does GL_LIGHT0 contribute to the scene?
- Explain why we call glLightfv(GL_LIGHT0, GL_POSITION, light_pos) every frame (not only once).

5) Floor Visibility

- Why does enabling GL_CULL_FACE often cause a single-quad floor to disappear at some camera angles?
- How does the provided code prevent that?

6) Transparency

- Explain why transparent objects are typically drawn after opaque objects.
- What does glDepthMask(GL_FALSE) do, and why is it used for the transparent sphere?

Deliverable B. **Submit**: your answers as HW04_answers.pdf or HW04_answers.docx.

Part C — Challenge Homework

Goal: Extend the demo into a small interactive scene editor with correct rendering behavior from any camera angle.

Required Features (must implement all)

C1) Object System (Scene Graph Lite)

Implement classes (or equivalent structures):

- SceneObject (base)
- SphereObject(SceneObject)
- Optional bonus: PlaneObject or BoxObject

Each object must store:

- Position (x, y, z)
- Scale or radius
- Color (r, g, b, a)
- Material properties (shininess and specular strength)
- A transparent flag derived from alpha (a < 1.0)

Requirement: All scene objects must be stored in a list and rendered each frame.

C2) Picking / Selection with Mouse (Core Difficulty)

When the user clicks (or modifier + click):

- Convert the mouse position into a ray in world space.
- Test intersection against all spheres.
- Select the closest hit sphere.

Minimum math required:

- Construct a view ray using projection and model-view matrices (or equivalent math).
- Ray–sphere intersection test.

Deliverable: The selected object must show a visible cue (e.g., highlight color, emissive effect, or wireframe overlay).

C3) Transform Controls (Move Selected Object)

After selecting a sphere, implement movement keys:

- I / K: move +Z / -Z
- J / L: move -X / +X
- U / O: move +Y / -Y

Requirement: Implement step-size control (fine vs coarse) using the Shift key.

C4) Correct Transparency Rendering (Sorting)

Implement correct rendering order:

- Render all opaque objects first.
- Render all transparent objects after that, sorted back-to-front by distance to the camera.

Requirement: Sorting must update dynamically as the camera moves.

C5) Lighting Control UI

Add a mode switch:

- Press TAB to toggle between Camera Control mode and Light Control mode.
- In Light Control mode: arrow keys move light in X/Z; PageUp/PageDown move light in Y.
- Display current light position on-screen (PyGame 2D text overlay is acceptable).

C6) Save/Load Scene (File I/O)

Implement saving and loading the scene as JSON:

- Store list of spheres: position, radius, RGBA, shininess, specular.
- Ctrl+S saves to scene.json.
- Ctrl+L loads from scene.json.

Constraints

- Must remain PyGame + PyOpenGL fixed pipeline (no shaders).
- Must keep the textured floor.
- Must support at least 10 spheres smoothly.
- Code must be organized into multiple files: main.py, scene.py, objects.py, picking.py, io_scene.py.

Deliverables

Submit a zip file named HW_OpenGL_04_<student_id>.zip containing:

- Source code (multiple files as specified).
- README.md with: (1) how to run, (2) controls, (3) short explanation of picking math and transparency sorting.
- One short video (10–20 seconds) demonstrating: add/select objects, move them, correct transparency overlap, and save/load.

Suggested Grading Rubric

- Part A (Run + evidence): 10%
- Part B (Concept questions): 20%
- Part C — Object system: 15%
- Part C — Picking / selection: 25%
- Part C — Transform controls: 10%
- Part C — Transparency sorting: 10%
- Part C — Light mode + display: 10%
- Part C — Save/Load: 10%

Note: Deductions apply for missing features, incorrect math, incorrect draw order for transparency, or unstable controls.

Submission Form:
https://forms.gle/AbJTRp2MsKYq1YJM8