Project 3 — Illumination and Shading

Students: **Ilia Taitsel (67258)**, **Oleksandra Kozlova (68739)**

December 1, 2025

Shaders and Variables

We use two GLSL programs:

- **Gouraud**: `shader1.vert + shader1.frag`,
- **Phong**: `shader2.vert + shader2.frag`.

Both share the same semantic uniforms and attributes; only the place where lighting is computed changes (vertex vs. fragment).

Gouraud Vertex Shader (`shader1.vert`)

Inputs (attributes)

- `in vec4 a_position` — vertex position in object space.
- `in vec4 a_normal` — vertex normal in object space.

Uniforms

- `const int MAX_LIGHTS` — maximum number of supported lights (8).
- `struct LightInfo {...}` — per-light data:
  - `vec3 ambient, diffuse, specular` — light intensities.
  - `vec4 position` — position or direction in (camera / world coordinates).
  - `int type` — 0: point, 1: directional, 2: spot.
  - `bool enabled` — whether this light is enabled.
  - `vec3 axis` — spotlight axis (camera / world coordinates).
  - `float cutoff` — cutoff parameter.
  - `float aperture` — aperture angle (specified in degrees).
- `struct MaterialInfo {...}` — object material:
  - `vec3 Ka, Kd, Ks` — ambient, diffuse, specular reflectance.
  - `float shininess` — shininess of the material.
- `uniform int u_n_lights` — actual number of lights in use.

- `uniform LightInfo u_lights[MAX_LIGHTS]` — array of lights.
- `uniform MaterialInfo u_material` — current object material.
- `uniform mat4 u_projection` — projection matrix.
- `uniform mat4 u_model_view` — model-view matrix.
- `uniform mat4 u_normals` — normal matrix.

Outputs

- `out vec4 v_color` — final vertex color after per-vertex Phong lighting, to be interpolated by the rasterizer.

Gouraud Fragment Shader (`shader1.frag`)

Inputs

- `in vec4 v_color` — interpolated vertex color.

Uniforms

- No extra uniforms.

Outputs

- `out vec4 color` — final fragment color, set directly from `v_color`.

Phong Vertex Shader (`shader2.vert`)

Inputs (attributes)

- `in vec4 a_position` — vertex position in object space.
- `in vec4 a_normal` — vertex normal in object space.

Uniforms

- `uniform mat4 u_projection` — projection matrix.
- `uniform mat4 u_model_view` — model-view matrix.
- `uniform mat4 u_normals` — normal matrix.

Outputs

- `out vec3 v_normal` — normal in camera space (later normalized in the fragment shader).
- `out vec3 v_posC` — position in camera space.
- `out vec3 v_viewer` — direction from fragment to camera (in camera space).

Phong Fragment Shader (`shader2.frag`)

Inputs

- `in vec3 v_normal` — interpolated normal (camera space).
- `in vec3 v_posC` — interpolated position (camera space).
- `in vec3 v_viewer` — interpolated camera direction.

Uniforms Same structures and uniforms as in `shader1.vert`:

- `const int MAX_LIGHTS;`
- `struct LightInfo { ambient, diffuse, specular, position, type, enabled, axis, cutoff, aperture };`
- `struct MaterialInfo { Ka, Kd, Ks, shininess };`
- `uniform int u_n_lights;`
- `uniform LightInfo u_lights[MAX_LIGHTS];`
- `uniform MaterialInfo u_material;`

Outputs

- `out vec4 color` — final fragment color after per-fragment Phong lighting with possible spotlight attenuation.

Scene Representation

We do *not* store the scene in an explicit JSON scene graph. Instead, object transforms are built procedurally using a matrix stack (`STACK`):

- the camera view matrix `mView` is loaded into the stack,
- each object (platform, cube, bunny, torus, cylinder) is drawn by:
  - `STACK.pushMatrix()`,
  - applying translations and scales (for placement in the four quadrants),
  - uploading `u_model_view` and `u_normals`,
  - issuing the corresponding `draw` call,
  - `STACK.popMatrix()`.

Light positions for world-space point/spot lights are also visualized as small spheres using the same stack mechanism.

Extra, Missing and Partial Functionalities

Extra / Beyond the Base Specification

- **Multiple lights with scalable GUI** Array-based light storage and the `buildLightsGUI()` function make adding new lights a matter of pushing a new object into the `lights` array.

- **World vs. camera light coordinates** Each light has a `coordinate_space` flag (world / camera). The JavaScript function `uploadLightInfo()` converts positions and axes from world to camera coordinates before sending them to the shaders.

- **Mouse look + WASD flying camera** The mouse (drag) rotates the camera around the target, and the keys `W/A/S/D` move the camera parallel to the ground plane, producing a simple "fly" navigation.

- **Camera reset and wheel-based zoom/dolly** A GUI button restores the camera to its default configuration; the mouse wheel changes the field of view or moves the camera along the view direction depending on modifier keys.

- **Light markers** World-space point/spot lights are rendered as small spheres to visualize their positions in the scene.

Partial or Deviating Implementations

- **Scene graph** No explicit JSON or variable-based scene graph is used; the hierarchy is implicit in the matrix stack operations (`STACK.pushMatrix()`, `STACK.popMatrix()`, translations and scales around each draw call). For this small, fixed scene, a procedural approach is more compact and easier to maintain.