

Documentation

Group/Game Name: Luminara

Brief description of implementation: Luminara is an escape puzzle game. The player moves in first-person, interacts with a button and two rotatable mirrors, and redirects a light beam into a sensor to open the door and win.

Core implementation areas:

- Scene composition: Cornell-like room, imported door model, mirrors, beam, sensor, HUD and win/lose overlays.
- Game logic: interaction, beam reflection, sensor detection, timer-based lose condition, replay/exit panels.
- Effects: environment mapping on mirror surfaces and a bloom/glow post-process style pass focused on beam highlights.

Features of the Game

- First-person movement and look controls.
- Interactive puzzle:
 - o toggle emitter button,
 - o rotate mirrors,
 - o route beam to sensor.
- Win condition when sensor is hit by the beam.
- Lose condition via countdown timer.
- 2D overlay UI:
 - o timer HUD,
 - o win/lose panels,
 - o replay/exit buttons.

Additional Libraries / Frameworks:

- Vulkan SDK (graphics API)
- GLFW (window/input)
- GLM (math)
- glslang/SPIR-V toolchain (shader compilation path in project setup)
- GCG Vulkan framework / launchpad (course framework; helper abstractions, DDS loading, pipeline helpers).
- INIReader (config parsing for window/camera/renderer settings).

Notes:

- No external physics engine is used.
- No sound middleware is used.

Gameplay:

Mandatory:

- 3D Geometry:
 - Non-trivial room + gameplay objects.
 - External model loading implemented (door OBJ via object loader).
- Playable:
 - Goal-oriented puzzle with interaction beyond camera movement.
- Min 60 FPS and Framerate Independence:
 - Framerate independence implemented (dt-based updates).
 - FPS shown in title bar each second.
 - Targeted at 60 FPS in normal viewpoints.
- Win/Lose Condition:
 - Win: beam reaches sensor.
 - Lose: timer reaches zero.
- Intuitive controls:
 - W/A/S/D + arrows for movement.
 - Single-event actions via callbacks (toggle keys, click interactions).
 - Toggle 'T' to hide/show the timer.
 - Toggle 'F' to enable/disable full screen mode.
 - Toggle 'R' to reset the game.
 - Toggle 'esc' to close the game.
 - Click on the button to turn on the light beam.
 - Click on the mirrors to rotate them
- Intuitive Camera:
 - Free first-person yaw/pitch camera with mouse look.
 - Hold the right mouse button to rotate the view.
- Illumination model:
 - Directional + point light, material parameters, normals, Phong-style shading.
- Textures:
 - Textured floor with UVs, mip chain creation and linear mip filtering.
- Moving Objects:
 - Beam segments evolve through reflections.
 - Mirrors rotate when clicked on.
 - Door opens on success.
- Documentation:
 - This document.

Optional:

- Heads-up Display:
 - Timer HUD as 2D overlay with toggle key 'T'.

Effects:

Texturing:

- Dynamic Environment Map:
 - Implemented as dynamic planar mirror reflections (live scene rendering) using:
 -
 -
 - Implementation outline:
 - A mirrored camera is computed per mirror each frame (mirror-space reflected view).
 - The scene is rendered to per-mirror offscreen color targets.
 - Mirror fragments are projected with the mirrored view-projection matrix and sampled from that dynamic render target.
 - Reflection is blended with Fresnel for angle-dependent reflectivity.
 - Mirror-side masking keeps one designated reflective side for gameplay readability.
 - References used:
 - LearnOpenGL Framebuffers / Planar Reflection
concepts: <https://learnopengl.com/Advanced-OpenGL/Framebuffers>
 - Project-specific extension:
 - Per-mirror mirrored-camera pass integrated into the gameplay render loop, so reflections update in real time with player movement and puzzle state.

Post Processing:

- Bloom/Glow:
 - Implemented as a post-style bloom composite pass:
 - Separate sampled render target is used as bloom input.
 - Bright-pass thresholding + weighted blur taps in fragment shader.
 - Red-dominance mask constrains bloom mostly to beam highlights.
 - Alpha-blended fullscreen composite over final image.
 - References used:
 - LearnOpenGL Bloom: <https://learnopengl.com/Advanced-Lighting/Bloom>
 - Khronos/Vulkan style multipass concepts from course examples and bloom references.
 - Project-specific extension:
 - Beam-focused bloom extraction (red-dominance filter) to avoid white wall/ceiling bleed artifacts.

Animation:

- Vertex Shader Animation:
 - Implemented in the vertex shader for beam rendering modes.
 - Used for the pulsation effect on the beam.
 - Implementation outline:

- Beam cylinder vertices are displaced radially in local space with a traveling sine wave.
- Time and axial position drive animation phase, producing motion along the beam direction.
- Normals are analytically re-derived after deformation to keep lighting stable.
- The resulting world position is then transformed with the standard model/view-projection path.
- References used:
 - GPU Gems style procedural vertex deformation concepts.
 - General sinusoidal deformation techniques for shader-based animation.
- Project-specific extension:
 - The deformation is restricted to beam modes only, keeping other scene geometry unaffected and ensuring puzzle readability.

Other special features:

- Win/lose overlay panels with replay/exit buttons.

Walk-through:

1. Start the game (`Luminara`).
2. Move with `W/A/S/D` (or arrow keys), rotate the view with the right button of the mouse.
3. Move closer to the emitter button and click in it to activate the beam.
4. Move closer to the mirrors and click on them to rotate them and redirect beam segments.
5. Route the beam into the sensor.
6. Door opens and win panel appears.
7. If timer reaches zero first, lose panel appears.
8. Use Replay to reset or Exit to close the game.

Where you can clearly see the effects:

- Environment Map:
 - Look at mirror surfaces from side viewpoints.
 - Reflection appears on one designated mirror side.
- Bloom/Glow:
 - Activate beam and observe bright red beam/glow halo contribution.
 - Effect is most visible around beam highlights and high-contrast beam regions.

For grading convenience:

- FPS is visible in window title for quick performance check.