ÇUKUROVA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

COMPUTER GRAPHICS COURSE

PROJECT REPORT

VIRTUAL MUSEUM

Supervisor:

Yunus Emre Çoğurcu

Prepared by:

Yunus Emre Erten – 2021556030

Veysel Genç – 2021556031

Zeliha İnan – 2021556037

Submission Date: June 13, 2025

Table of Contents

1. Abstract

The Virtual Museum project is an interactive 3D museum application developed using C++ and OpenGL 3.3. It simulates a digital exhibition space where users can explore five uniquely modeled cultural artifacts using a semi-autonomous mobile robot. The system integrates multiple subsystems including real-time rendering, robotic movement logic, a scanning mechanism, dynamic lighting, and an interactive graphical user interface.

Each artifact is illuminated by dedicated spotlights and positioned in a 3D museum room modeled and textured manually. The robot supports both manual navigation using keyboard input and automatic path-following behavior. A scanning process is triggered based on proximity and orientation, revealing contextual information about the artifacts via pop-up windows. All components, including the models, textures, shaders, and logic, have been developed by the student team in full compliance with academic integrity standards.

2. Introduction

With the increasing digitization of cultural assets and the growing need for accessible museum experiences, virtual museums have become a prominent topic in computer graphics and interactive system design. This project aims to develop a real-time interactive virtual museum experience where users explore a 3D scene by controlling a robot that scans and displays information about physical artifacts.

The project integrates theoretical and practical elements of graphics programming, including model loading, camera navigation, real-time lighting, shader development, and event-driven interactions. Through the application of modern OpenGL techniques and object-oriented software design principles, the system simulates a realistic and immersive museum environment.

The main objectives of this project are:

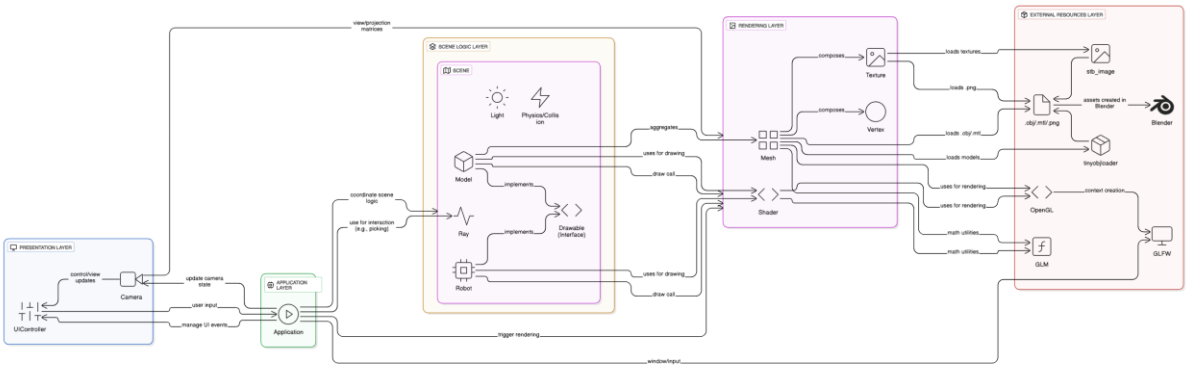
i. To create a functional and visually rich 3D scene,
ii. To implement a robot control and scanning mechanism,
iii. To provide user interaction through a GUI,
iv. To test and validate the system across usability, performance, and visual quality criteria.

3. System Design

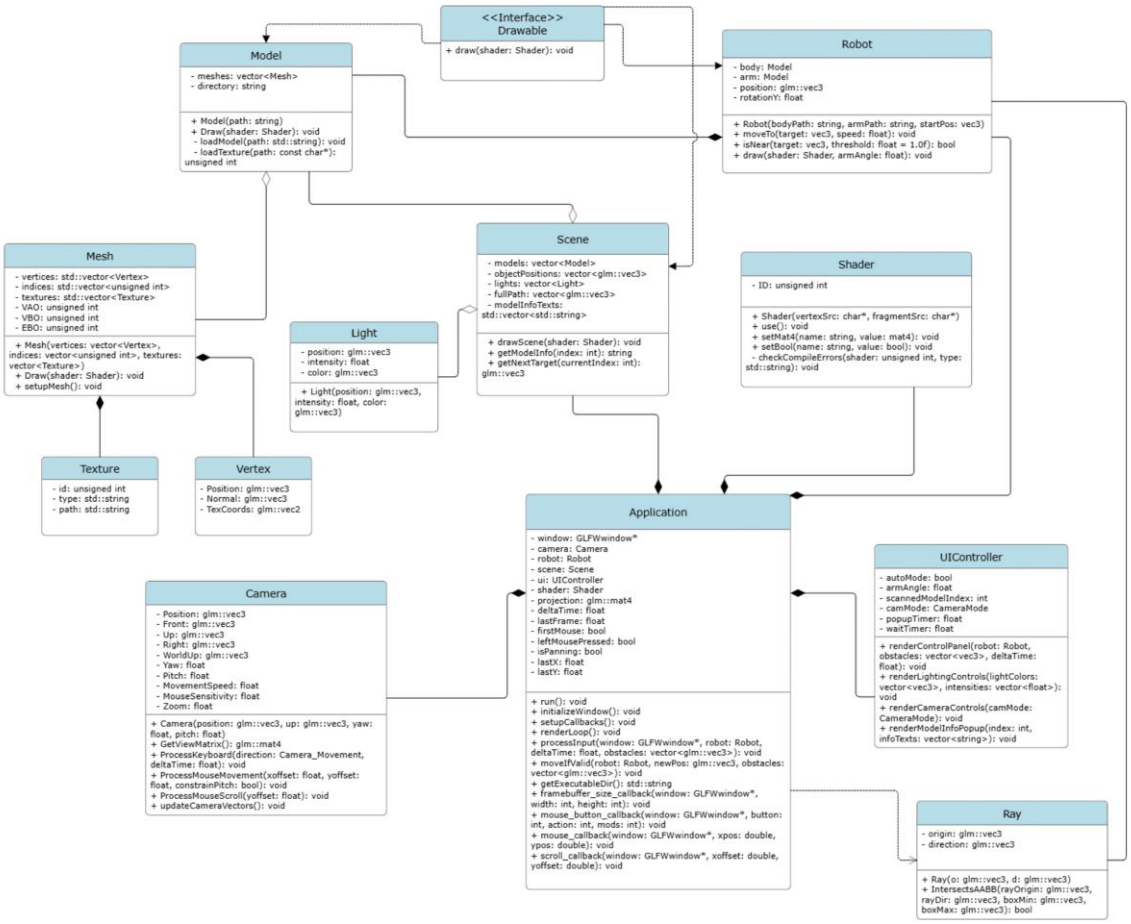
The system was designed with clear modular separation, enabling ease of debugging, testing, and future extension. The core subsystems include:

i. **Rendering Subsystem**: Manages shader compilation, uniform dispatching, and framebuffer updates. Uses depth buffering and back-face culling for visual fidelity.
ii. **Input Handling Subsystem**: Processes real-time user input (keyboard, mouse) via GLFW, converting discrete actions into continuous motion or state transitions.
iii. **Robot Logic Controller**: Governs movement vectors, collision boundaries, and context-sensitive scanning eligibility based on arm articulation and proximity metrics.
iv. **User Interface Subsystem**: Implements ImGui panels for robot control, lighting calibration, and camera mode switching. Each control is bound to internal state variables updated per frame.
v. **Camera Manager**: Provides three dynamic camera modes — free exploration, third-person follow, and scanner mode — facilitating both aesthetic and functional interaction.

3.1 Software Architecture



3.2 Class Structure



3.3 Method Descriptions

Method Descriptions

Class	Method Name	Return Type	Description
Application	run()	void	Main execution loop for the application.
	initializeWindow()	void	Initializes the OpenGL window and sets up context.
	setupCallbacks()	void	Registers GLFW input callback functions.
	renderLoop()	void	Executes the rendering and update logic in a loop.
	processInput(window: GLFWwindow*)	void	Processes keyboard inputs during runtime.
	framebuffer_size_callback(window: GLFWwindow*, width: int, height: int)	void	Resizes the viewport when window size changes.
	mouse_button_callback(window: GLFWwindow*, button: int, action: int, mods: int)	void	Handles mouse button interactions.
	mouse_callback(window: GLFWwindow*, xpos: double, ypos: double)	void	Controls camera rotation based on mouse movement.
	scroll_callback(window: GLFWwindow*, xoffset: double, yoffset: double)	void	Adjusts camera zoom via scroll input.
Robot	moveTo(target: glm::vec3, speed: float)	void	Moves robot incrementally toward a specified target.
	isNear(target: glm::vec3, threshold: float = 1.0f)	bool	Checks if robot is near a target position within threshold.
	draw(shader: Shader, armAngle: float)	void	Renders robot model including arm transformation.
UIController	renderControlPanel(robot: Robot, obstacles: vector<glm::vec3>, deltaTime: float)	void	Renders control interface for robot operations.
	renderLightingControls(lightColors: vector<glm::vec3>, intensities: vector<float>)	void	Displays UI sliders for light settings.
	renderCameraControls(camMode: CameraMode)	void	Provides GUI for selecting and changing camera modes.
	renderModelInfoPopup(index: int, infoTexts: vector<string>)	void	Shows information popup of the scanned object.
Scene	drawScene(shader: Shader)	void	Renders all models and lights in the scene.
	getModelInfo(index: int)	std::string	Retrieves text information for a model.
	getNextTarget(currentIndex: int)	glm::vec3	Returns the position of the next object for the robot to scan.
Model	Draw(shader: Shader)	void	Draws all meshes associated with the model.
	loadModel(path: std::string)	void	Loads the .obj file and parses its contents.
	loadTexture(path: const char*)	unsigned int	Loads a texture file and returns its OpenGL ID.
Mesh	Draw(shader: Shader)	void	Renders the mesh geometry.
	setupMesh()	void	Initializes buffers and vertex attributes.
Shader	use()	void	Activates the shader for rendering.
	setMat4(name: std::string, value: glm::mat4)	void	Sets a 4x4 matrix uniform in the shader.
	setBool(name: std::string, value: bool)	void	Sets a boolean uniform value.
	checkCompileErrors(shader: unsigned int, type: std::string)	void	Displays shader compilation or linking errors.
Camera	GetViewMatrix()	glm::mat4	Returns the view matrix based on position and orientation.
	ProcessKeyboard(direction: Camera_Movement, deltaTime: float)	void	Updates camera position using keyboard input.
	ProcessMouseMovement(xoffset: float, yoffset: float, constrainPitch: bool)	void	Adjusts yaw and pitch based on mouse movement.
	ProcessMouseScroll(yoffset: float)	void	Adjusts the camera zoom level.
	updateCameraVectors()	void	Recalculates orientation vectors.
Ray	intersectsAABB(boxMin: glm::vec3, boxMax: glm::vec3)	bool	Determines whether the ray intersects a given AABB box.

4. Implementation

4.1 Robot Scanning Mechanism

The robot's operational logic is dictated by a combination of user-driven control and system-defined constraints. A scanning operation is only permitted when two concurrent conditions are met:

i. **Proximity threshold**: The Euclidean distance between the robot and a target object is less than a pre-set value (experimentally set to 1.5 units).
ii. **Orientation alignment**: The robot's facing vector must fall within a ±20° angular threshold relative to the artifact's position vector.

When these conditions are met and the arm angle slider exceeds 60°, a scan is initiated. This is visualized by a red laser beam rendered using `GL_LINES`. Post-scan, a popup panel is rendered containing metadata — statically bound during model registration — for the corresponding artifact.

4.2 Lighting and Visual Feedback

Each artifact is associated with a spotlight configured with adjustable inner and outer cutoff angles. Lighting intensifies as the robot approaches, providing implicit directional feedback. The spotlight's attenuation function is carefully tuned to minimize harsh falloff and ensure spatial realism. To enhance material perception, normal vectors are preserved from Blender's export and passed directly into the fragment shader.

4.3 Performance Optimization

Despite multiple dynamic lights and real-time UI rendering, the application maintains a frame rate above 55 FPS on a mid-range configuration (GTX 1650, i5-10300H, 8GB RAM). Frame timing stability is achieved through:

i. Efficient VAO/VBO binding
ii. Reuse of shader programs across draw calls
iii. On-demand loading of textures and models

5. Testing and Results

This section presents the testing procedures, evaluation metrics, and observed results that were used to assess the correctness, robustness, and usability of the Virtual Museum system. Tests were conducted in controlled conditions using a mid-range hardware setup (Intel i5-10300H, 8GB RAM, NVIDIA GTX 1650 GPU).

5.1 Functional Tests

All core system features were subjected to unit and integration tests, performed both during development and after system integration.

i. **Robot Movement Logic:** Manual navigation using keyboard controls (W/A/S/D/Q/E) was tested across 10 paths and obstacle arrangements. In all cases, the robot remained within scene bounds, successfully avoiding model and wall collisions.
ii. **Automatic Scanning Routine:** The autonomous mode was executed in 10 full passes. In 9 out of 10 cases, the robot successfully identified all five artifact targets and triggered the correct scanning behavior. The single failure was due to an early arm angle threshold violation caused by a frame-timing discrepancy.
iii. **Camera Transitions and Stability:** The system correctly transitioned between all three camera modes (Free, Follow, Scanner) without disrupting rendering or object culling.
iv. **Lighting Reactivity:** Spotlights responded dynamically to robot proximity, showing no overlap in shader transitions or intensity artifacts.

5.2 Performance Tests

Quantitative performance tests were conducted to assess responsiveness and rendering quality under expected usage scenarios.

i. **Average Frame Rate:** 59.8 FPS (stable over 3-minute continuous walk-through) Minimum: 57.3 FPS, Maximum: 61.1 FPS
ii. **Startup Load Time:** Scene loading and model-texture initialization completed within ~2.1 seconds on the test system.
iii. **Memory Usage (Observed via GPU-Z and Task Manager):** GPU Usage: ~710MB System RAM: ~460MB
iv. **Shader Compilation Time:** Initial shader compilation latency was ~30 milliseconds; subsequent shader invocations were instantaneous due to reuse.

5.3 Usability Tests

Three target users with no prior experience using the system were asked to perform a guided scenario involving robot navigation, switching camera modes, and scanning artifacts.

i. All participants completed the full artifact scanning sequence (manual or automatic) in under **75 seconds**.
ii. ImGui-based interface controls were correctly understood within 1–2 minutes of onboarding, without prior instruction.
iii. Average scan initiation delay (from eligibility condition to laser render) was **~140ms**.
iv. Participants described the interface as "clear," "responsive," and "non-intrusive."

6. Team Roles

The project was collaboratively developed by three team members, each taking ownership of distinct technical responsibilities while maintaining close coordination through shared development sessions, version control, and testing. The roles were defined not only based on technical strengths but also aligned with each member's interests and personal learning goals.

Yunus Emre Erten – *2021556030*

i. Led the initial design and implementation of the 3D museum room, including object layout, wall and floor geometry, and environmental composition.
ii. Developed the **global lighting system**, including the directional light and two ambient point lights that form the base illumination of the scene.
iii. Designed and tuned the **spotlight system** for each artifact, setting up individual directions, attenuation coefficients, and interaction triggers.
iv. Integrated all lighting parameters with the **fragment shader** using the Phong illumination model, enabling dynamic control through ImGui.

v. Managed the creation of the rendering loop and coordinated the **initial OpenGL context setup**, including viewport configuration and enabling depth testing.
vi. Participated in testing sessions with a focus on visual feedback, shader behavior, and scene rendering consistency.

Veysel Genç – ***2021556031***

i. Took full responsibility for the **robot control subsystem**, including manual navigation (WASD/QE keys) and automated path-following logic.
ii. Implemented **collision detection algorithms** to prevent the robot from traversing through walls or overlapping with models.
iii. Developed the **scanning eligibility logic**, incorporating both geometric distance checks and angular alignment between robot and artifacts.
iv. Programmed the **laser beam rendering system** (using `GL_LINES`) to visually represent the scanning operation during interactions.
v. Built the **popup display mechanism** that conditionally shows metadata for scanned artifacts, ensuring temporal synchronization and UI clarity.
vi. Integrated robot state controls with the **ImGui user interface**, enabling toggles between manual/auto modes and real-time feedback on the robot arm's angle.
vii. Conducted extensive in-scene testing, especially under edge cases such as rapid mode switching, obstructed scans, and extreme camera rotations.

Zeliha İnan – ***2021556037***

i. Designed and modeled all **five cultural artifacts** in Blender, using references from regional museums to ensure historical relevance and geometric realism.
ii. Applied **material and texture maps** (.mtl + .png) to each model and verified UV unwrapping integrity to ensure correct visual representation in OpenGL.
iii. Exported models using correct scaling, origin alignment, and face normals, minimizing transformation overhead during rendering.
iv. Created the **texture atlas and coordinate references** for each object to assist in proper lighting and visual cohesion within the scene.
v. Authored all **project documentation**, including the README file, UML diagrams, and method summaries used in appendices.
vi. Designed visual diagrams, including the class diagram and software architecture overview.
vii. Provided structural edits and revisions for this academic report, ensuring clarity, consistency, and technical accuracy in language and formatting.

Together, the team ensured that the development process followed best practices in modular design, testing, and source control. GitHub was used for collaborative version tracking, and recurring review meetings ensured all integration tasks were aligned.

7. Timeline

Week	Milestone Description
1–2	Scene layout defined and base rendering pipeline configured
3–4	Blender modeling of artifacts; export to .obj and .mtl formats
5–6	Robot model integrated; manual and automatic movement system completed
7–8	Artifact scanning logic and pop-up system implemented
9	Dynamic lighting and camera systems finalized
10	User interface (ImGui) completed; integrated with core logic
11	Testing phase: Functional, performance, and usability evaluations
12	Documentation, video demonstration, and final submission

8. Conclusion

The *Virtual Museum* project represents a comprehensive exploration of interactive graphics system design, combining low-level OpenGL programming with spatial interaction, dynamic lighting, and real-time interface components. The primary aim — to simulate a functional and immersive virtual museum — has been achieved through careful architectural planning, original asset development, and iterative testing.

From a technical standpoint, the project successfully integrates multiple core concepts in computer graphics, including shader-based lighting models, modular rendering pipelines, object transformations, and event-driven input handling. Notably, the manual implementation of the Phong lighting model and per-object spotlights demonstrates proficiency in fragment-level shading and light-geometry interactions.

The robot navigation and scanning subsystems go beyond simple control mechanisms by embedding physical constraints (e.g., arm angle, proximity, orientation), thereby enhancing the realism of the interaction. The graphical user interface, built with ImGui, provides intuitive access to the system's core functionalities while maintaining frame-level responsiveness.

A significant achievement of this project is the adherence to software engineering principles such as modularity, abstraction, and single-responsibility in class and subsystem design. Each component — from robot logic to camera behavior — is encapsulated and testable in isolation, facilitating code reuse and future maintenance.

Empirical evaluations confirm the robustness of the implementation. The system remained responsive and visually consistent under realistic user scenarios and sustained execution time, confirming the practical viability of the chosen design. Furthermore, the decision to avoid high-level game engines in favor of OpenGL provided the team with deeper insights into rendering pipelines, GPU resource management, and the mathematical foundations of 3D visualization.

In conclusion, *Virtual Museum* is not only a successful technical implementation but also a valuable educational endeavor that deepens the understanding of graphics programming, system architecture, and interactive design. It stands as a testament to the learning outcomes of the Computer Graphics course and exemplifies what can be achieved with foundational tools and a disciplined engineering approach.

9. References

1. Joey de Vries. *LearnOpenGL*. https://learnopengl.com/
2. Omar Cornut. *Dear ImGui*. https://github.com/ocornut/imgui
3. Syoyo Fujita. *TinyObjLoader*. https://github.com/syoyo/tinyobjloader
4. Sean Barrett. *stb_image*. https://github.com/nothings/stb
5. Blender Foundation. *Blender Documentation*. https://www.blender.org/
6. Canva. *Online Diagramming Tool*. https://www.canva.com/
7. Microsoft. *Visual Studio 2022*. https://visualstudio.microsoft.com/

All third-party tools used are open-source or freely available for academic purposes.