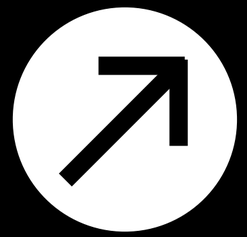
Satrajit Ghosh

Final Project Presentation

2024

LAG Engine



The Game Engine of the Future

Introduction

This project presents the foundational work for a fully custom-built 2D and 3D game engine designed to support realistic physics simulation and advanced interactive systems., The engine features core mechanics such as constraint-based cloth simulation, particle dynamics, and object interaction—implemented entirely from first principles without relying on prebuilt engines . The system operates across both 2D and 3D contexts, establishing a shared physics layer that enables modular expansion. The project serves as the groundwork for a broader research trajectory and is intended to evolve into a full-fledged Master's Thesis. Future directions include parallelization of the physics system for high-performance simulation and the integration of agentic AI components to enable autonomous interactions within dynamic environments. By emphasizing control, extensibility, and a dual-dimensional architecture, this work lays the technical foundation for scalable, research-driven development in simulation and game design.

Research

Building a game engine from scratch is a complex task that involves rendering, physics, and scene management. To guide development, foundational resources like *Game Engine Architecture* [1], the *Quake* source code [2], and TheCherno's tutorials [3] were consulted. Community insights from r/gamedev [4] also helped refine lightweight UI and modular design choices.

References

[1] https://www.gameenginebook.com/
[2] https://github.com/id-Software/Quake/tree/master/WinQuake
[3] https://www.youtube.com/@TheCherno/videos
[4] https://www.reddit.com/r/gamedev/comments/122wegt/lightweight_c_gui_libraryframework_for_games/

Features

Dual-Dimension Physics Core

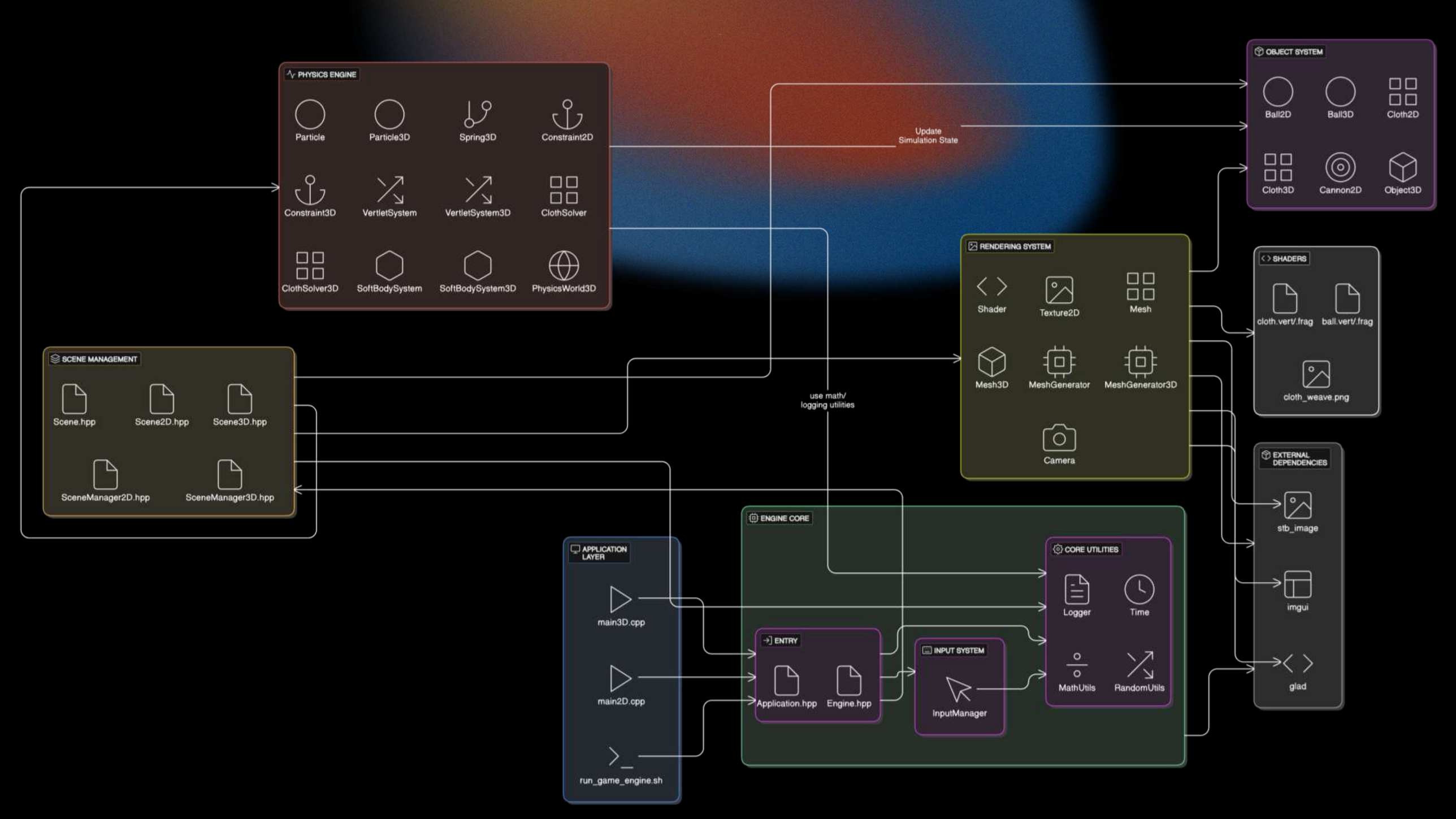
Seamless support for both 2D and 3D simulations using a unified Verlet-based physics system

Realistic Cloth with Dynamic Tearing

Custom-built cloth simulation with real-time deformation and tear propagation driven by constraint stress.by real-time stress thresholds for visual believability.

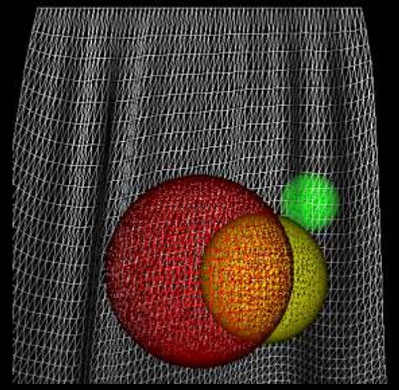
Engineered for AI and Parallel Scalability

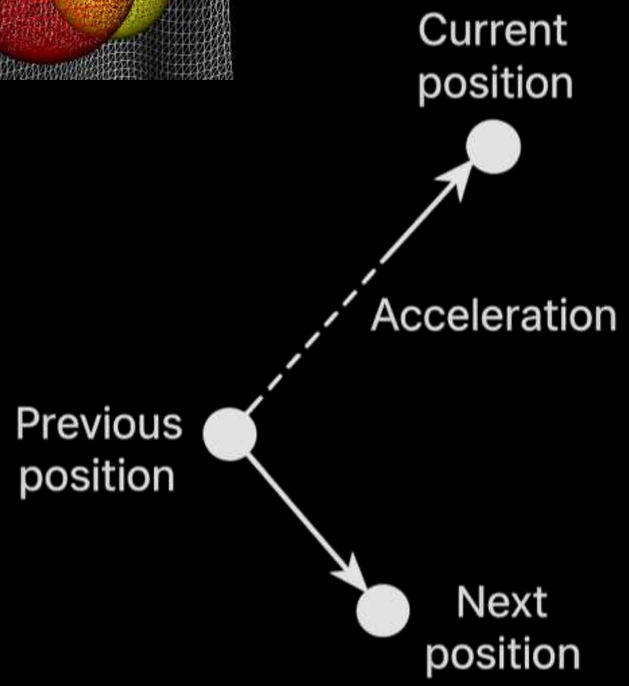
Modular architecture designed for agentic AI integration and future parallel execution.


PHYSICS ENGINE
Particle
Particle3D
Spring3D
Constraint2D
Constraint3D
VertletSystem
VertletSystem3D
ClothSolver
ClothSolver3D
SoftBodySystem
SoftBodySystem3D
PhysicsWorld3D
OBJECT SYSTEM
Ball2D
Ball3D
Cloth2D
Cloth3D
Cannon2D
Object3D
Update
Simulation State
RENDERING SYSTEM
Shader
Texture2D
Mesh
Mesh3D
MeshGenerator
MeshGenerator3D
Camera
SHADERS
cloth.vert/.frag
ball.vert/.frag
cloth_weave.png
SCENE MANAGEMENT
Scene.hpp
Scene2D.hpp
Scene3D.hpp
SceneManager2D.hpp
SceneManager3D.hpp
use math/
logging utilities
EXTERNAL
DEPENDENCIES
stb_image
imgui
glad
ENGINE CORE
APPLICATION
LAYER
main3D.cpp
main2D.cpp
run_game_engine.sh
CORE UTILITIES
Logger
Time
MathUtils
RandomUtils
ENTRY
Application.hpp
Engine.hpp
INPUT SYSTEM
InputManager

Vertlet Integration

- **Verlet Integration** is a numerical method used to simulate motion without explicitly tracking velocity.
- It estimates a particle’s next position using its current and previous positions along with acceleration.
- This method offers superior stability over Euler integration, especially for systems with constraints like cloth or soft bodies.
- It's ideal for real-time physics where small errors in velocity accumulation can destabilize simulations.

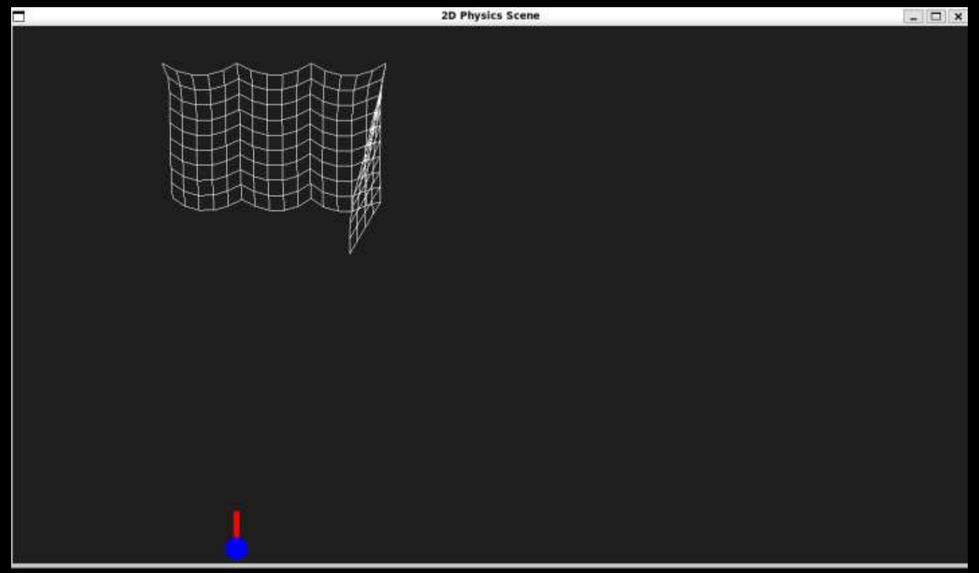




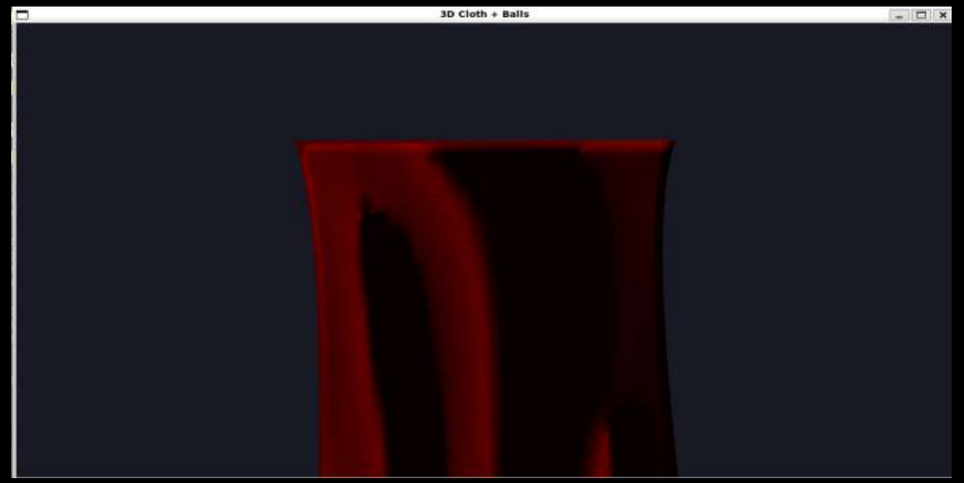
$$x(t+\Delta t) = x(t) + (x(t) - x(t-\Delta t)) + a * \Delta t^2$$

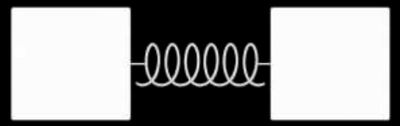
Cloth Sim Setup

Cloth Simulation in 2D
- Implemented in the XY plane using Cloth2D, Particle, and Constraint2D classes.
- Uses Verlet integration for physics stability and smooth motion.
- Includes structural, shear, and optional bending constraints.
- Supports tearing by breaking constraints under stress.
- Collides with Ball2D objects using simple distance checks and response forces.
- Rendered as lines or mesh using basic 2D OpenGL drawing routines.



Cloth Simulation in 3D
- Implemented in the XZ plane using Cloth3D, Particle3D, and Spring3D.
- Applies structural, shear, and bending springs for realistic fabric behavior.
- Uses the same Verlet physics system adapted for 3D vectors.
- Integrates with PhysicsWorld3D for simulation and time management.
- Includes sphere-cloth collision with Ball3D, with realistic momentum exchange.
- Dynamically updates a triangle mesh with normals for GPU rendering.





Ball Generation and Interaction in the Game Engine

Physics and Interaction

•Balls are created dynamically in both 2D and 3D worlds using engine-specific constructors (Ball, Ball3D).

•Each ball has mass, velocity, and collision bounds integrated into the Verlet physics system.

•In 2D, collision resolution is simple and cloth reacts to contact with applied tearing force.

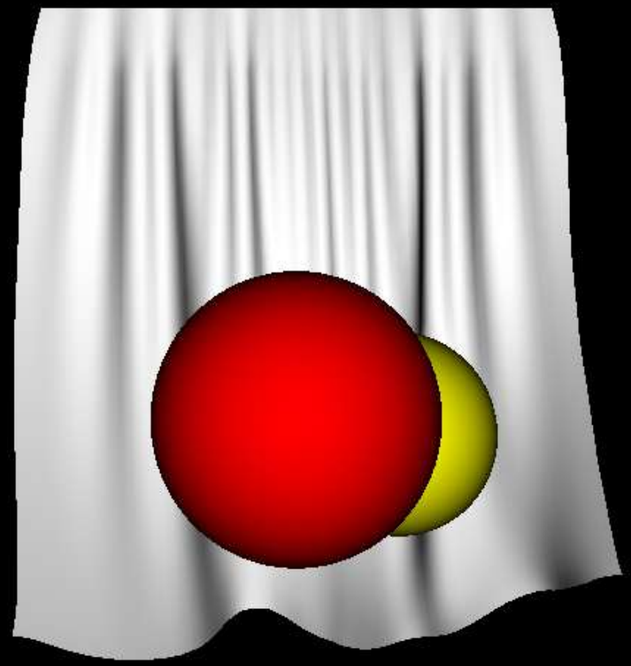
•In 3D, the cloth-ball collision involves sphere-ray tests and normal-based force redistribution.

Rendering

•Balls are rendered using custom shaders (ball.vert, ball.frag) with lighting and material parameters.

•In 2D, circles are visualized via flat shaders and mesh generators.

•In 3D, spheres are generated using UV-sphere triangulation and drawn with smooth lighting and depth handling.



Shaders

- **Modular Shader Management**: A custom Shader class handles compilation, linking, and uniform updates, streamlining GPU communication.

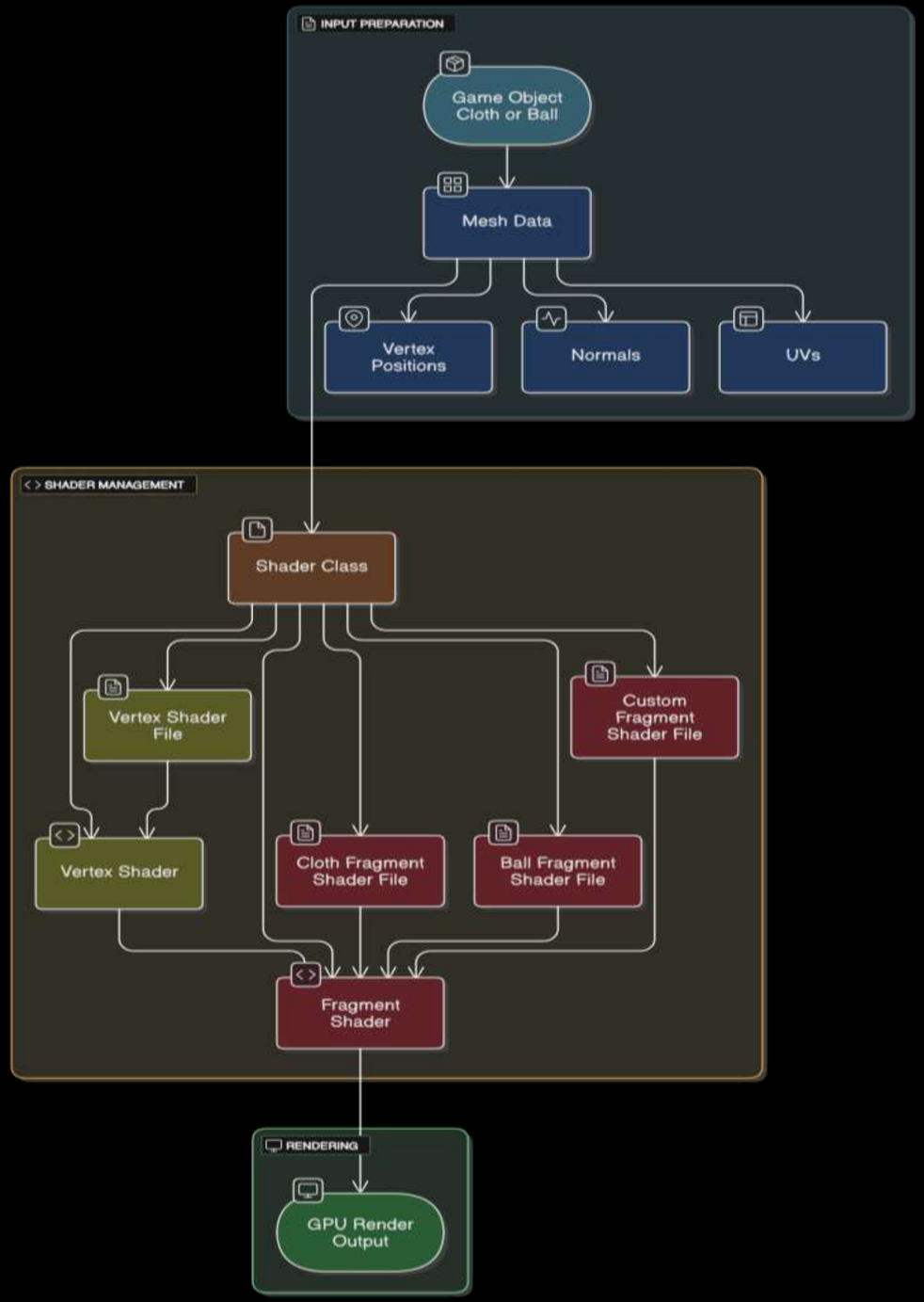
- **Realistic Lighting**: Shaders support advanced effects like roughness, rim lighting, ambient occlusion, and procedural fabric patterns for cloth.

- **Unified Material System**: Materials and lights are applied through structured methods (setMaterial, setLight) for consistent rendering.

- **Efficient Binding**: Utility functions simplify setting uniforms (e.g., matrices, vectors, floats) dynamically at runtime.

- **Error Handling**: An internal logger tracks shader load and compile-time errors for easier debugging.

- **Flexible Rendering**: The system supports both procedural shading (for balls) and texture-based shading (for cloth).



Current Temporary UI

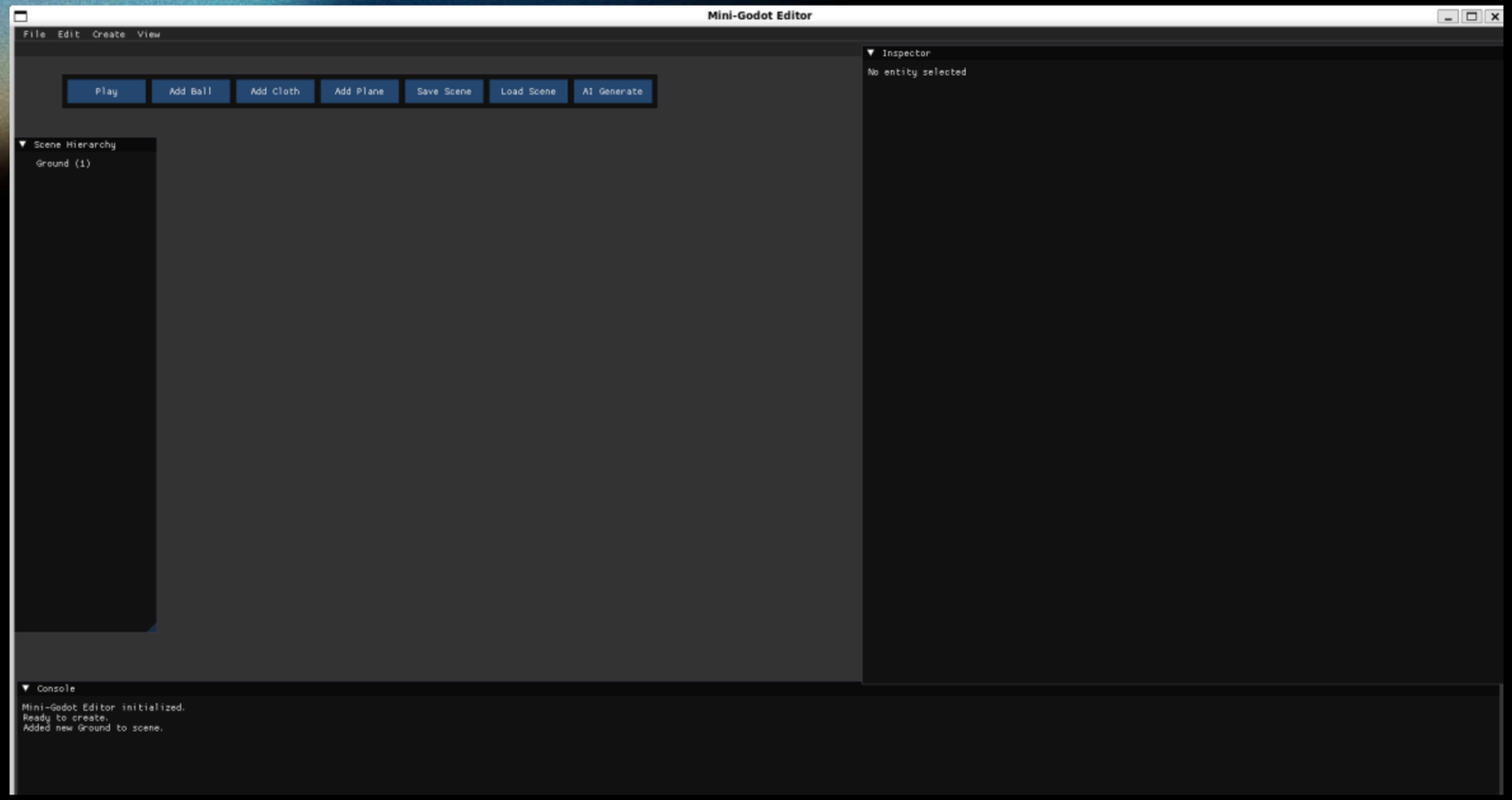
•The current UI is implemented using ImGui, chosen for its ease of integration and real-time flexibility during development.

•It provides temporary controls for adjusting cloth physics parameters, toggling ball spawning, and visual debugging (e.g., wireframes, normals).

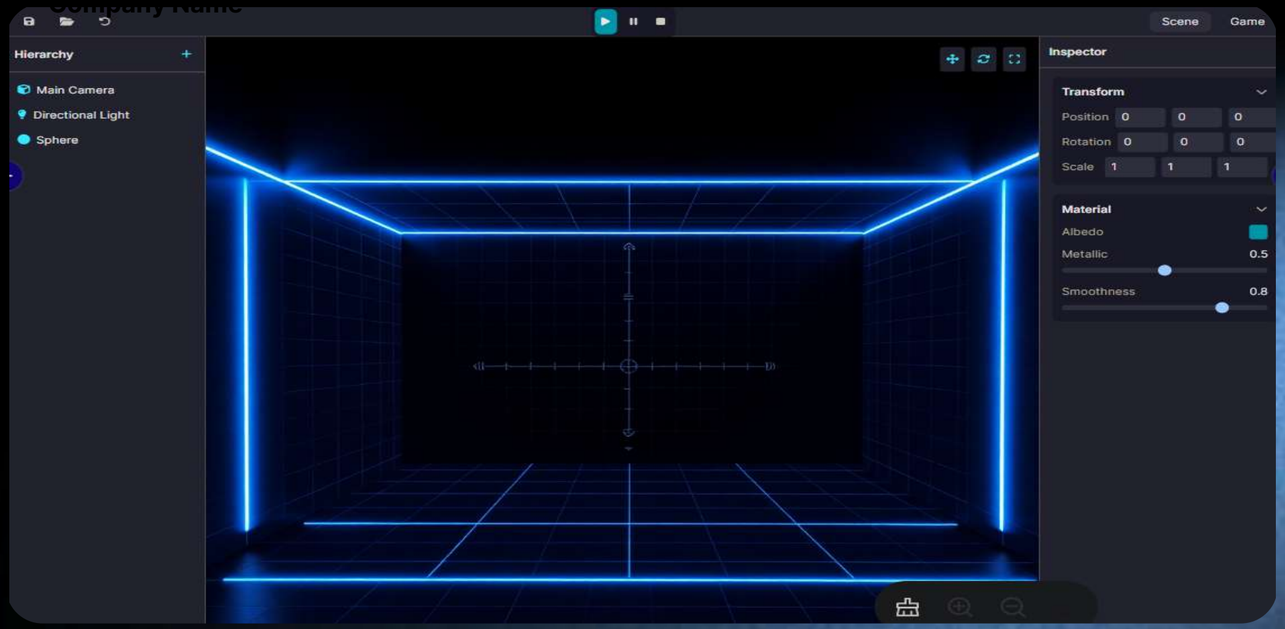
•UI elements are defined in the main render loop, allowing immediate feedback during simulation testing.

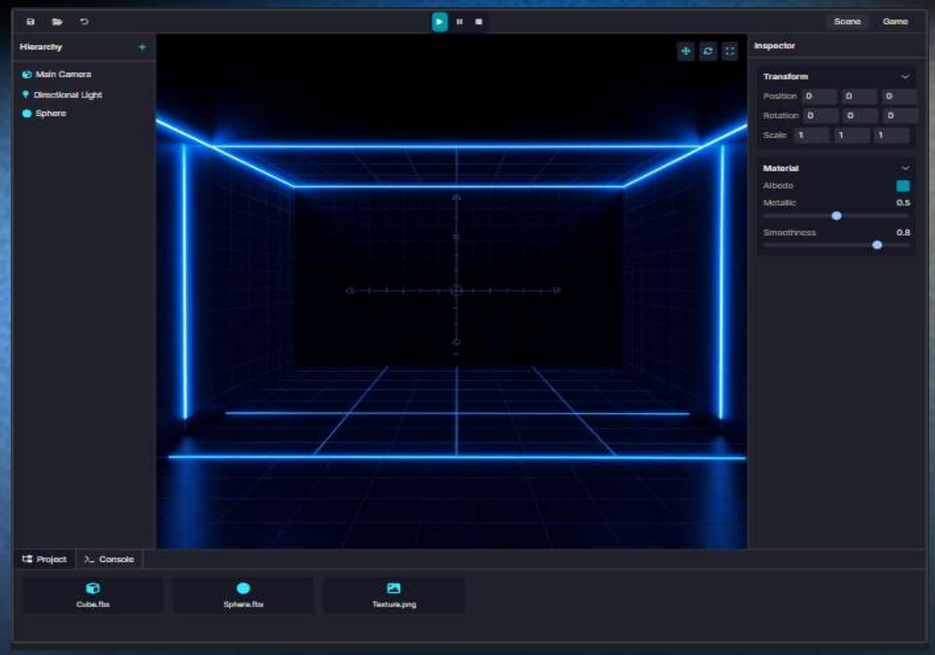
•While effective for prototyping, it lacks custom styling and layout flexibility suitable for a full game engine interface.

•Future upgrades include replacing ImGui with a custom dockable GUI system or integrating Dear ImGui + ImGuizmo for 3D transform editing.
•Long-term plans involve building a node-based editor and panel system similar to commercial engines like Unity or Godot.

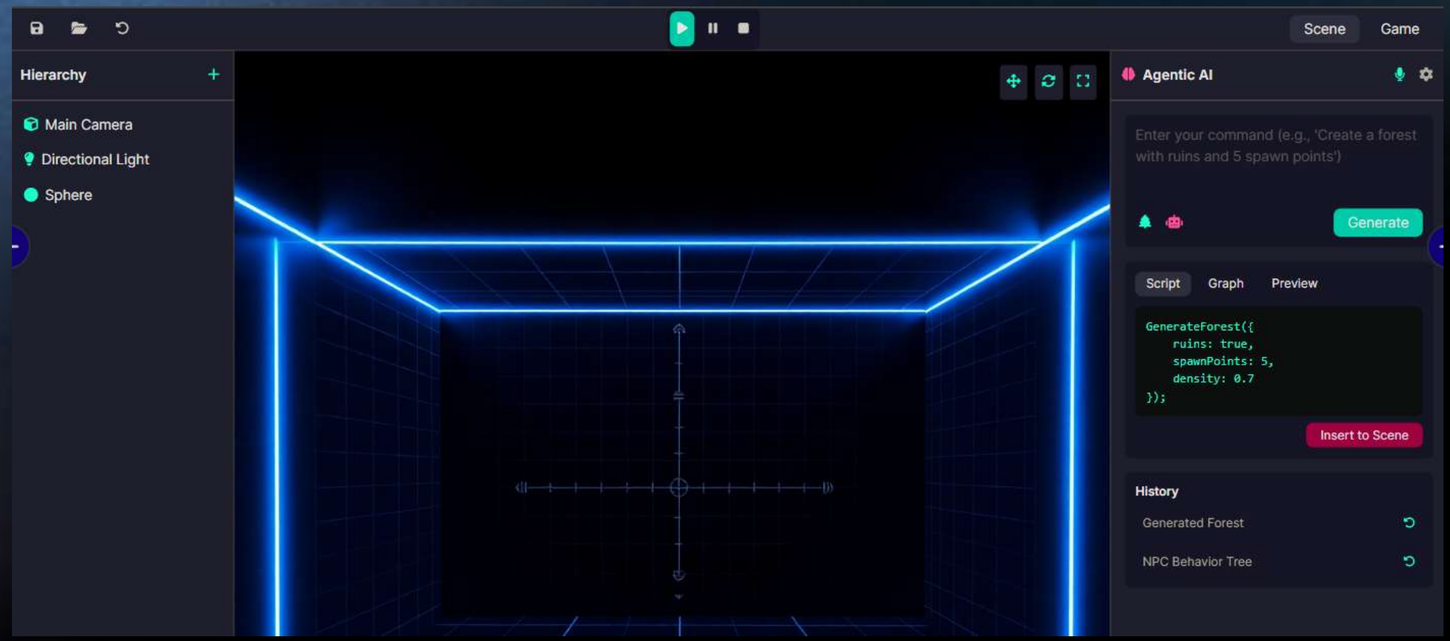


Current UI made using Imgui

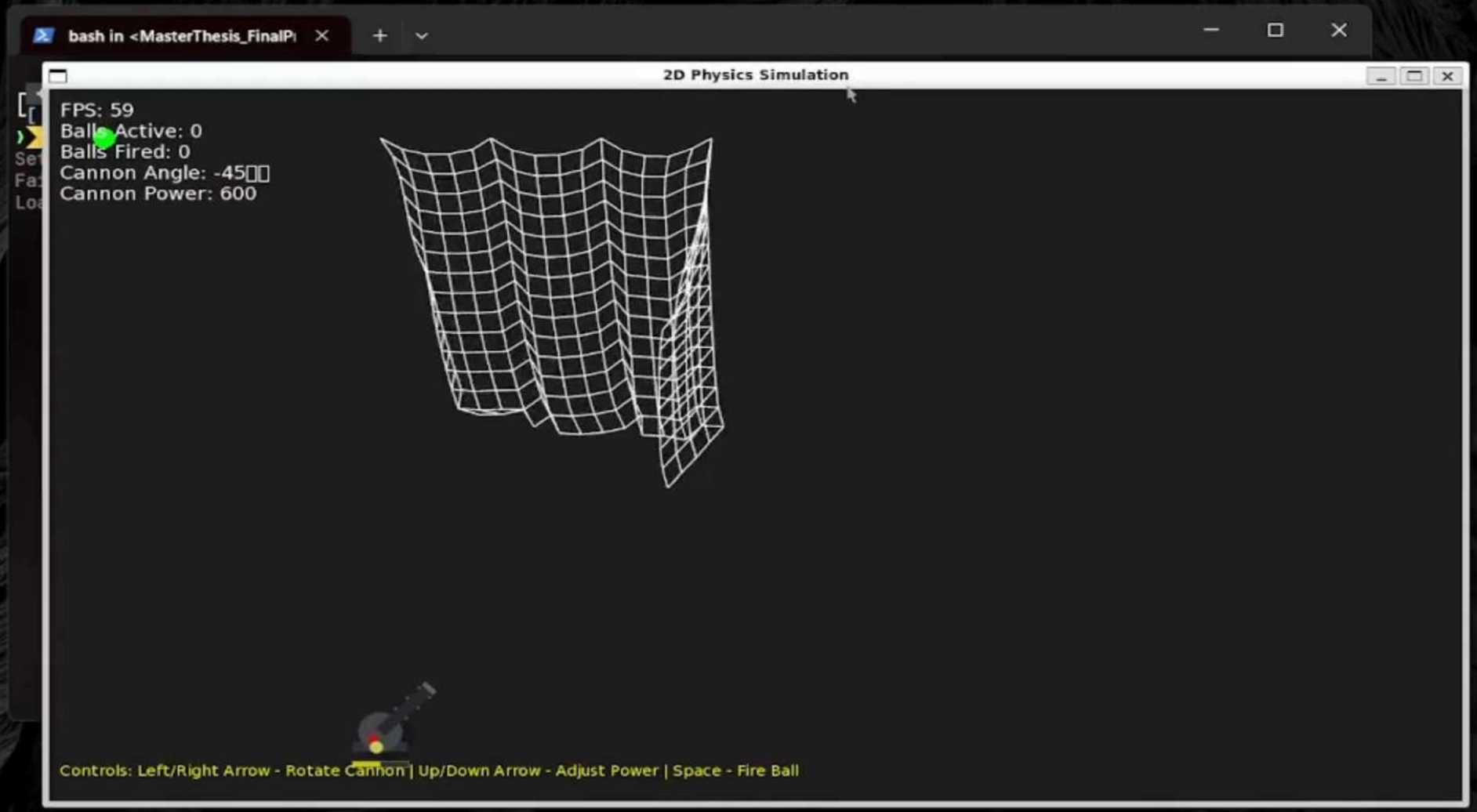




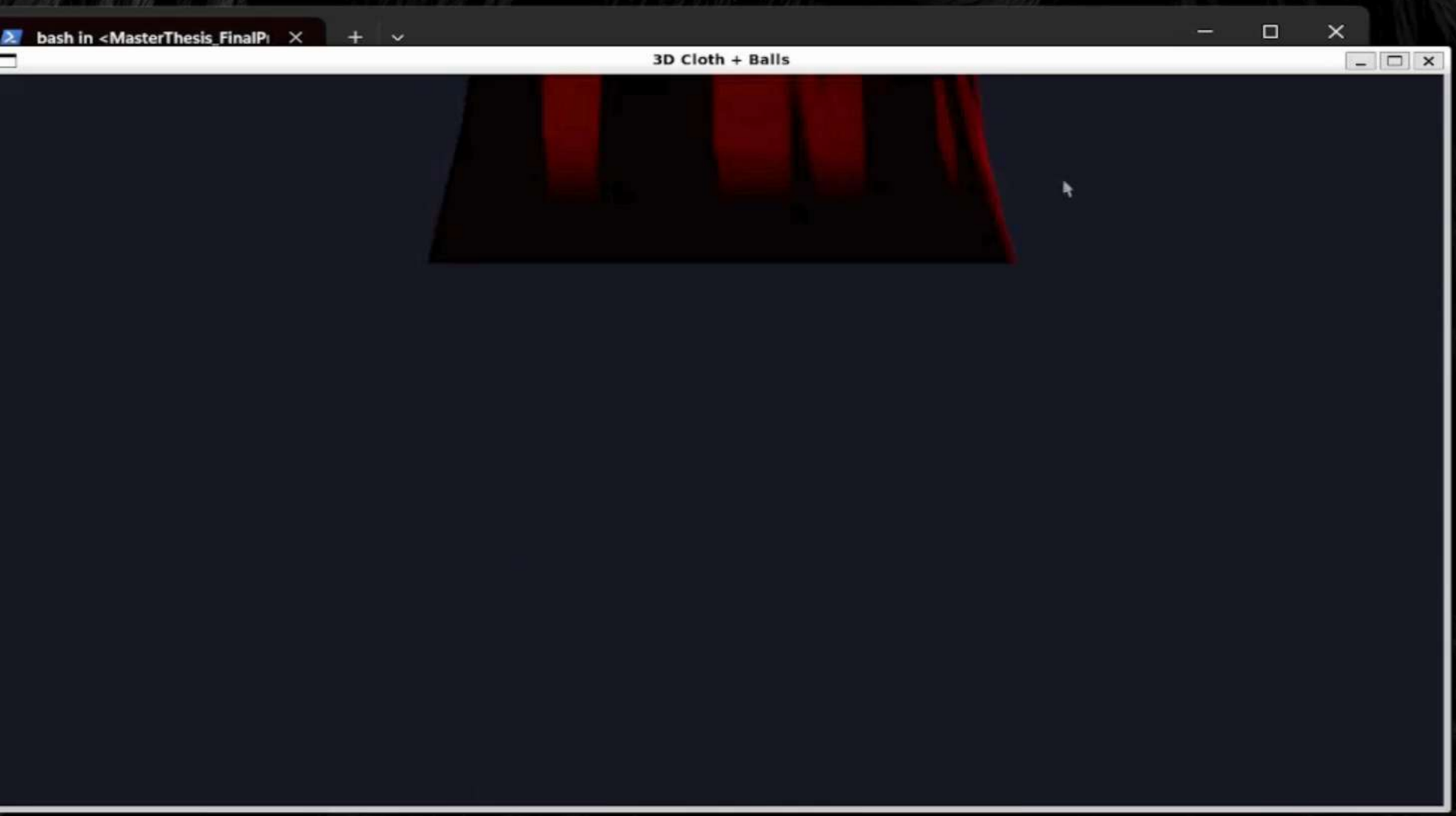
Proposed UI/UX



Demo



2D cloth physics demonstration rendered with wireframe visualization, illustrating spring-based constraint resolution and projectile impact.



Real-time simulation of 3D cloth

Non-Modular Cloth Simulation