Hole Controlled on a 3D Map: A Physics-Based Game

AUBERT Hippolyte Ferdinand Joseph | DE FREITAS MARTINS Alexandre Kevin
Department of Game Software
Keimyung University
alexandredfm.contact@gmail.com | hippolyte.aubert@epitech.eu
Github Repository

June 23, 2025

Ideas

The core idea of our project is to create a 3D physics-based game where the player controls a mobile hole on a map. The main mechanic is inspired by games where the player grows by consuming objects in the environment. As the hole swallows more objects, it increases in size, allowing the player to interact with larger and more challenging obstacles. The goal is to combine real-time physics, interactive gameplay, and dynamic object behavior in a fun and engaging way.

Features

- **Real-time control:** Move the hole using WASD keys.
- **Swallowing mechanic:** Objects are pulled in and removed when overlapping the hole.
- **Growth system:** The hole grows in size after consuming objects, unlocking new gameplay possibilities.
- **Physics simulation:** Objects have variable properties (mass, collision response) and interact realistically.
- **Timer and score:** Track player performance with a timer and score system.
- **UI controls:** Buttons for running, resetting, and toggling hitbox visibility.
- **Hitbox visualization:** Toggle hitbox rendering for debugging and understanding collisions.

Implementation

The game is implemented in C++ using OpenGL for 3D rendering and FLTK for the user interface. The physics engine is based on the Cyclone library, with custom extensions for collision detection and response. Game objects, including the hole and obstacles, are represented as 3D models with associated rigidbodies and bounding volumes.

Physics and Motion

The motion of each object is governed by Newton's second law:

$$\vec{F} = m\vec{a} \tag{1}$$

where $\vec{F}$ is the net force, m is mass, and $\vec{a}$ is acceleration. The position and velocity are updated each frame:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}\Delta t \tag{2}$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t}\Delta t \tag{3}$$

Collision Detection

We use Axis-Aligned Bounding Box (AABB) overlap for fast collision detection:

$$\text{overlap} = (x_{1,\max} > x_{2,\min}) \wedge (x_{1,\min} < x_{2,\max}) \tag{4}$$

and similarly for y and z axes. For more precise collisions, the Separating Axis Theorem (SAT) is used for oriented boxes.

Swallowing Mechanic

An object is swallowed if its center enters the hole's radius:

$$\|\vec{x}_{\text{object}} - \vec{x}_{\text{hole}}\| < r_{\text{hole}} \tag{5}$$

where r_{hole} is the current radius of the hole.

Growth Curve

The hole's radius increases as objects are swallowed:

$$r_{\text{hole,new}} = r_{\text{hole,old}} + k \cdot m_{\text{swallowed}} \tag{6}$$

where k is a growth constant and $m_{\text{swallowed}}$ is the mass of the object.

The main gameplay loop handles user input, updates physics, checks for collisions, and renders the scene. The UI provides buttons to start/stop the game, reset the scene, and toggle hitbox visibility, which is useful for debugging and visualizing the physics interactions.

The codebase is organized into modular components: physics, rendering, UI, and game logic are separated for maintainability. Assets such as models and textures are loaded at runtime, and the game state is managed to allow for smooth resets and restarts.

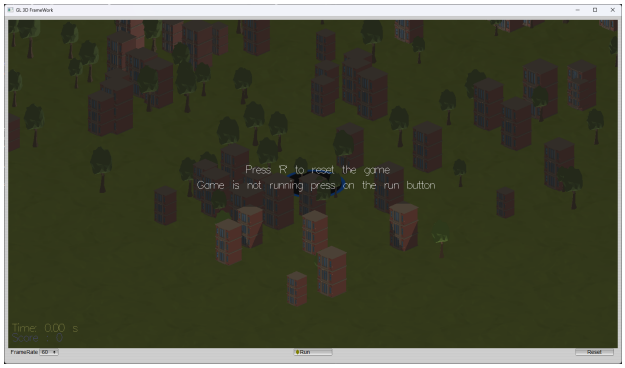
What We Learned

This term project provided hands-on experience with real-time physics simulation, collision detection, and game development. Key takeaways include:

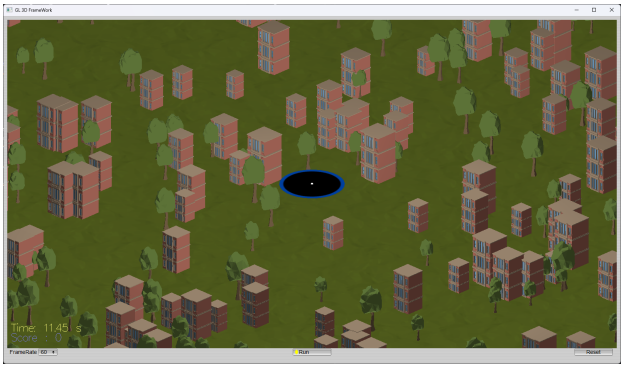
- Integrating a physics engine with a custom rendering pipeline, and translating mathematical models into efficient code.
- Managing object lifecycles and dynamic scene updates, including memory management and object pooling.
- Designing and balancing gameplay mechanics (growth, swallowing, scoring) using mathematical formulas and playtesting.
- Implementing and debugging UI features, such as the hitbox toggle, to aid development and visualization.
- Tuning physics parameters (mass, friction, restitution) for both realism and fun, and addressing stability issues (e.g., jitter, tunneling).
- Working with modular code and collaborating as a team, using version control and code reviews.
- Gaining practical skills in C++, OpenGL, and FLTK, and learning to debug complex interactions between subsystems.

We also learned the importance of visualization tools (like hitbox rendering) for debugging and understanding the behavior of the simulation, and how small changes in parameters can have large effects on gameplay and stability.

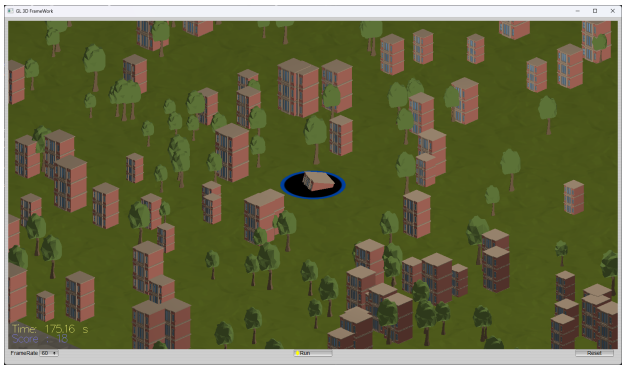
Overall, the project deepened our understanding of interactive 3D applications and the challenges of real-time simulation and user experience design.



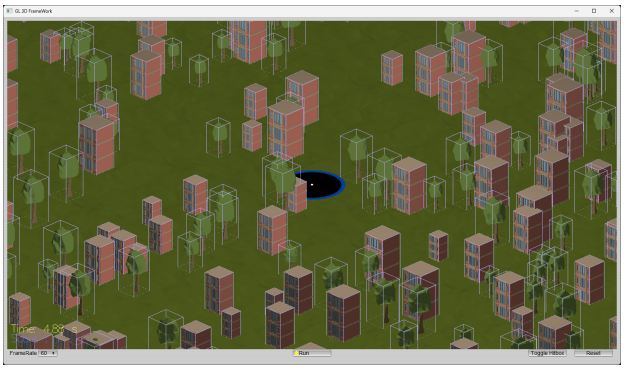
(a) First Launch



(b) Play Game



(c) Building Swallowed



(d) Hitbox Visualization

Figure 1: Game Screenshots