Project Report: Gravity Box

A 2.5D Platformer Game in C++ and OpenGL

Course: Computer Graphics & Multimedia Lab

Course Code: CSE-358

Submitted To:

Most. Thasin Sabira

Lecturer, Dept. of CSE,

Audity Ghosh,

Lecturer, Dept. of CSE

Submitted By:

Name	**ID**	**Role**
Razibul Hasan Badhon	0432310005101083	Team Leader / Core Architecture
Tasneem Jahan Purnota	0432310005101089	Level Design / VFX
Mostofa Kamal Raz	0432310005101079	Game Logic / Physics

Date of Submission: November 12, 2025

1. Introduction

"Gravity Box" is a simple but challenging 2.5D (2D gameplay in a 3D world) platformer game. We built it using C++ and the OpenGL graphics library. The main goal for the player is to collect all the green targets in a level while avoiding the red hazards. The core mechanic of our game is the ability to flip gravity with the press of a button, allowing the player to walk on the ceiling. This report explains the game's features, the libraries we used, the basic code structure, and shows our final result.

2. Main Features

Our game includes the following features:

- **Gravity Flipping:** The main feature. The player can press the SPACE bar to instantly flip the game's gravity, pulling them from the floor to the ceiling or vice-versa.
- **2.5D Gameplay:** The game looks 3D, but all movement and physics happen on a 2D plane (the X and Y axes). This makes the game easy to control but visually interesting.
- **Player Controls:** The player moves left and right using the A and D keys.
- **Collision Detection:** The player properly collides with the walls, floor, and ceiling.
- **Targets and Hazards:** The goal is to collect all green "Targets" If the player touches a red "Hazard," the level resets.
- **Dynamic Levels:** The game can support multiple levels. When all targets are collected, the game automatically loads the next level, which has more targets.
- **Scoring System:** The player's score increases for every target they collect and gets a bonus for completing a level.
- **Particle Effects:** We added simple particle explosions for when the player flips gravity, collects a target, or hits a hazard. This makes the game feel more responsive and "juicy."

3. Libraries and Tools

We used several key libraries to build our game:

- **OpenGL (Open Graphics Library):** This is the main API (Application Programming Interface) we used to draw all the 3D shapes, lines, and graphics to the screen.
- **GLFW (Graphics Library Framework):** We used this library to create the game window, make an OpenGL context, and get all the keyboard input from the player (like A, D, and SPACE).
- **GLAD:** This is a helper library that loads all the modern OpenGL functions for us so we can use them.
- **GLM (OpenGL Mathematics):** OpenGL doesn't do math. We used this library to handle all our vector and matrix math, which is needed to move, rotate, and position objects in 3D space.
- **Standard C++ Libraries:** We used standard libraries like <vector> (to hold lists of targets and particles) and <cmath> (for math calculations).

4. Simple Pseudocode

Here is a simple overview of how our code is structured.

The Main Game Loop

This is the "heartbeat" of our game. It runs over and over, 60 times per second.

```
START Game
  Initialize_Window_and_OpenGL()
  Load_Shaders_and_Models()
  Spawn_Level(1)

  WHILE ( a_player_has_not_closed_the_game )
    // 1. Check Input
    Process_Player_Input()

    // 2. Update Game Logic
    Update_Game_Physics()
    Check_for_Collisions()
    Update_Particles()

    // 3. Draw Everything
    Clear_the_Screen()
    Draw_the_Cube_Border()
    Draw_the_Player()
    Draw_all_Targets()
    Draw_all_Hazards()
    Draw_all_Particles()

    // 4. Show the new frame
    Swap_Buffers()
  END WHILE

END Game
```

Game Logic (How things update)

```
FUNCTION Update_Game_Physics()
    // Apply gravity to the player's velocity
    player.velocity = player.velocity + gravity

    // Move the player based on their velocity
    player.position = player.position + player.velocity

    // Check for wall collisions
    IF (player_hits_a_wall)
        stop_player_at_wall
    END IF

    // Check for hazard collisions
    IF (player_hits_a_hazard)
        Create_Explosion()
        Reset_the_Level()
    END IF

    // Check for target collisions
    IF (player_hits_a_target)
        Collect_the_Target()
        Add_to_Score()
        Create_Explosion()
    END IF

    // Check for win condition
    IF (all_targets_are_collected)
        Go_to_Next_Level()
    END IF
END FUNCTION
```

Input Handling (How we read keys)

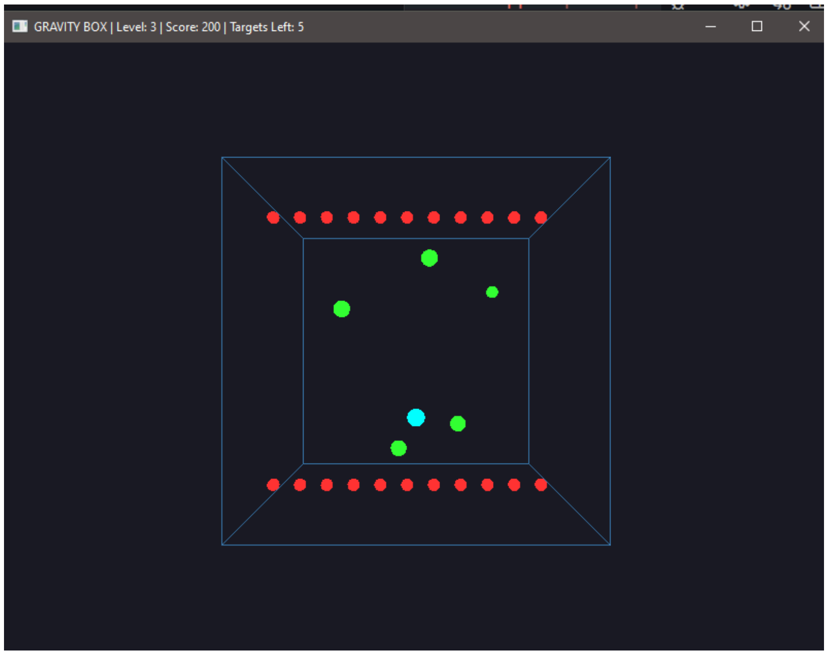
```
FUNCTION Process_Player_Input()
    IF (A_key_is_pressed)
        player.velocity.x = -speed
    ELSE IF (D_key_is_pressed)
        player.velocity.x = +speed
    ELSE
        player.velocity.x = 0 // Stop moving
    END IF

    IF (SPACE_key_was_just_pressed)
        gravity.y = gravity.y * -1 // Flip gravity
        Create_Explosion()
    END IF
END FUNCTION
```

4.1 Code Changes Summary

During the development of Gravity Box, we modified and extended the base OpenGL template code to implement our game's mechanics and visual effects. The following summarizes the key changes:

- Window Title
  - Updated the window creation line to display "Rotating Yellow Rectangle" as the title, reflecting our test object.
- Vertex Data
  - Replaced the single-triangle setup with six vertices forming two triangles.
  - This allowed us to render a complete rectangle without using an Element Buffer Object (EBO).
- Transformations (Movement)
  - Introduced translation logic using glm::translate.
  - The rectangle's position now oscillates smoothly along both the X and Y axes based on sin(timeValue).
- Uniforms for Transform
  - Added a uniform matrix (transform) in the vertex shader.
  - Passed transformation data from the CPU to the GPU using glUniformMatrix4fv.
- Color Animation
  - Implemented dynamic color changes by animating the green channel intensity with fabs(sin(timeValue)).
  - This produces a fade effect: rectangle transitions between black → green → black.
- Draw Call
  - Updated the draw function to glDrawArrays(GL_TRIANGLES, 0, 6) to render six vertices (two triangles) instead of three.

5. Game Output (Screenshot)



Here a screenshots of "Gravity Box" in action.

This image shows the basic level, with the blue player, green targets, and red hazards on the top and bottom.

Here, the player has pressed SPACE. You can see the gravity-flip particle effect and the player is now on the ceiling.

6. GitHub Repository

Our complete source code, including all libraries and the final .cpp file, is available at our public GitHub repository.

Link: Gravity Box - Game

7. Conclusion

This project was a great experience for our team. We learned how a game engine works from the ground up, starting from a blank window. We learned how to manage a game loop, handle player input, implement physics, and use shaders to draw objects. The biggest challenge was understanding the 3D math (matrices) and fixing collision bugs. In the future, we could add more features like moving hazards, different types of levels, and sound effects.