

CSE 4208: Computer Graphics Laboratory

3D Treasure Hunt

By,

Farhatun Shama

Roll:1907033

Date of Submission: 22-01-2025



Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

Objectives

- To design and develop a 3D interactive treasure hunt game using OpenGL 3.3.
- To implement object interaction mechanisms for locating hidden keys in a 3D environment.
- To create dynamic shooting mechanics.
- To integrate lighting, textures, and blending for enhanced realism.
- To demonstrate the application of transformations and animations in creating complex gameplay scenarios.

Introduction

The 3D Treasure Hunt Game is an OpenGL-based project designed to create an immersive and interactive gaming experience. The game comprises three distinct rooms with unique challenges. In the first room, players explore a treasure-filled environment and interact with objects to uncover a hidden key required to unlock the next room. The second room features a shooting challenge where players use spherical discs to hit static cylindrical targets, testing their aim and precision. The final room raises the difficulty as cylindrical targets move dynamically along the X-axis while simultaneously rotating around their own axes, challenging the players' reflexes and strategic thinking. This project combines object interaction, shooting mechanics, and advanced graphical effects, showcasing the practical application of modern OpenGL techniques in 3D game development.

Tools Used

- OpenGL 3.3
- GLFW (Graphics Library Framework)
- GLM (OpenGL Mathematics Library)
- Visual Studio

Methodology

Object Selection on Mouse Click: Object selection is based on ray-object intersection. When the user clicks, a ray is cast from the camera through the mouse cursor into the scene. The intersection of this ray with the bounding spheres of objects determines the selection.

Screen Space to Normalized Device Coordinates System (NDCS): The screen space mouse coordinates are converted to Normalized Device Coordinates (NDC). Below is the equation.

$$x_{\text{NDC}} = \frac{2x_{\text{SCS}}}{\text{width}} - 1, \quad y_{\text{NDC}} = 1 - \frac{2y_{\text{SCS}}}{\text{height}}$$

NDCS to World Space Transformation: The NDCS coordinates are transformed into world space in two steps:

- I. From NDCS to View Space (VCS)
- II. From View Space to World Space

Transformations on Object Selection: When an object is selected, its position is transformed to search for the key.

Car Implementation: The car is designed to move back and forth along a straight path within a predefined range. Its position is updated based on a speed parameter and the elapsed time (deltaTime). The direction of movement is tracked using a Boolean flag (movingForward) that toggles whenever the car reaches the limits of its range. The car's wheels are programmed to rotate proportionally to the distance traveled, ensuring realistic motion. This is achieved by incrementing the wheel's rotation angle using a formula that relates the traveled distance to the wheel's circumference. For rendering, the car body is represented as a scaled cube, and the wheels are rendered as cylinders positioned relative to the car body. Each wheel is translated, rotated to simulate movement, and scaled for appropriate size. The car's motion and wheel rotations are updated continuously using delta time, ensuring smooth animation.

Ghost Implementation: The ghost's animation consists of two primary motions: floating and rotation. The floating motion is achieved by oscillating the ghost's vertical position using a sine function, creating a smooth up-and-down movement. The amplitude and speed of the sine wave control the range and frequency of the float. At the topmost point of its motion, the ghost transitions into a rotation phase, during which its rotation angle increases incrementally at a defined speed. Once the rotation completes a full circle (360 degrees), the ghost resumes floating. The ghost is composed of multiple parts, such as a curved object for the main body and a cone for the tail, which are dynamically transformed based on the animation state. For rendering, the ghost's base position is translated, and additional transformations (floating and rotation) are applied in sequence to achieve the desired effects. The animations for floating and rotation are updated per frame using delta time to ensure smooth transitions.

Motion in Room 2 (Arrow and Target)

1. **Arrow Motion:** The arrow follows a specific direction for hitting target.
2. **Target Motion:** Targets oscillate horizontally till hit or timeout.

Key Points for the Game Implementation

Object Selection on Mouse Click

- Capture mouse click and calculate a ray from the camera through the clicked screen position.
- Transform the ray into world coordinates.
- For each object in the scene:
 - Transform the object's bounding sphere to world coordinates.
 - Perform ray-sphere intersection using the ray and transformed sphere.
 - If the intersection is valid:
 - Highlight or select the object.
 - Perform actions based on the object type (e.g., lift the object, check if it has a key).

Arrow Logic

- On mouse click for shooting:
 - Calculate the arrow's direction based on the camera and mouse position.
 - Set the arrow's flying state to true and position it at the camera's location.
- Update arrow position:
 - If the arrow is flying, move it along its direction with a fixed speed.
 - If the arrow moves out of bounds, reset its position and state.
- Check for arrow collisions with targets:
 - If the arrow hits a target, mark the target as hit, stop the arrow, and update the score.

Car Animation

- Initialize car parameters, including speed, position, and motion range.
- Update car position:
 - Move the car back and forth within a defined range.
 - Rotate its wheels proportionally to the distance traveled.
- Render the car:
 - Apply transformations for the car body and wheels based on the current position and wheel rotation.

Ghost Animation

- Initialize ghost animation parameters, including amplitude, speed, and rotation.

- Update ghost position:
 - Float the ghost vertically using a sine wave function.
 - At the peak of the motion, start rotating the ghost.
 - Stop rotation after completing a full cycle.

Target Logic

- Initialize targets with positions, textures, and motion parameters.
- Update target positions:
 - Move targets in a sine wave pattern if dynamic.
- Render targets with applied textures and transformations.
- Check if all targets are hit to progress the game state.

Setup

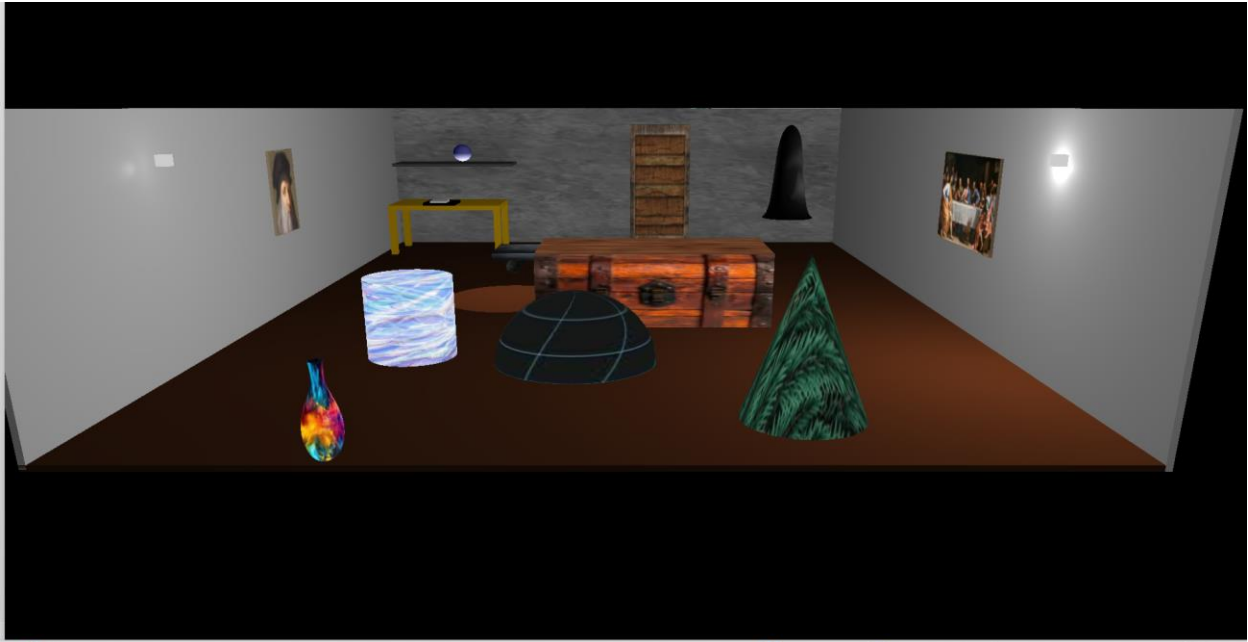


Figure 1: Initial Room

Figure 1 demonstrates the initial view. Figure 2 shows the same room from an upper view. The Figure 3 shows the shooting space (both static and dynamic).



Figure 2: Initial Room from top



Figure 3: Room 2 and 3

Input Handling

The `processInput` function handles user input for the 3D environment, allowing interaction with the game through keyboard controls. Key functionalities include:

1. Camera Movement:

- **W, A, S, D keys:** Move the camera forward, backward, left, and right.
- **H, F, T, G, Q, E keys:** Adjust the camera's position in 3D space along the X, Y, and Z axes.
- **R key:** Resets the camera to its initial position.

2. Lighting Controls:

- **Directional Light (Key 1):** Toggles the state of the directional light (on/off).
- **Point Lights (Keys 2 and 3):** Independently toggles two point lights.
- **Spotlight (Key 4):** Toggles the spotlight.
- **Ambient Lighting (Key 5):** Enables or disables ambient lighting for all light sources.
- **Diffuse Lighting (Key 6):** Toggles diffuse lighting for all light sources.
- **Specular Lighting (Key 7):** Toggles specular lighting for all light sources.

3. Debouncing Mechanism:

- A state variable is used for each key to avoid repeated actions while a key is held down, ensuring smooth toggling functionality.
- 'O' key to open the door.

4. Window Control:

- **Escape key:** Closes the window and terminates the program.

Discussion

The developed game was implemented using OpenGL to showcase various interactive 3D graphics techniques. Object selection was enabled through ray casting, where mouse input was transformed into world coordinates to identify intersections with objects, ensuring accurate interactions. However, minor mismatches were observed during object selection, which were attributed to pseudo-depth inconsistencies caused by approximations in mapping normalized device coordinates (NDC) to world coordinates. Different coordinate transformations were employed, including screen space to NDC and subsequently to world coordinate space (WCS), facilitated by transformation matrices derived from projection and view transformations. Interactive elements such as a floating ghost and a moving car were programmed to exhibit realistic motion through periodic oscillations and directional movement, demonstrating the use of trigonometric functions and time-based updates. Objects were made to respond dynamically to user actions, such as being lifted upon selection and triggering transitions to the next room upon key retrieval. The second room featured arrow shooting mechanics, enabling straight trajectories calculated based on mouse-click directions. The game environment, integrating object transformations and realistic motion, was successfully designed to be intuitive and engaging, reflecting effective use of graphics programming techniques.

Conclusion

The interactive 3D game successfully demonstrated advanced graphics programming techniques, including object selection, coordinate transformations, and realistic motion. The implementation showcased user engagement through dynamic interactions and environmental transitions. Despite minor limitations like pseudo-depth mismatches, the project achieved its objectives, providing a comprehensive learning experience in computer graphics.

References

- Lab Slides