Volume 01, Issue 02, December 2025 : 12-21 Received: June 15, 2025. Revised: December 26, 2025

International Journal of Electrical and Intelligent Engineering

3D Object Movement Transformation Using FPS and TPS Camera View Modes in OpenGL

Maulina Safitri [1]*, Rama Yusuf Mahendra[2], Rasyeedah Binti Mohd Othman[3], Riffani Fathia Annisa[4]

[1,4] Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia
[2] Institut Teknologi Sepuluh Nopember Surabaya, Indonesia
[3] Universiti Teknologi PETRONAS, Malaysia
***** Correspondong author's Email: maulinasafitri@uin-malang.ac.id

Abstract: Object transformation in three-dimensional space is a fundamental component in the development of interactive and realistic 3D modeling systems, particularly for control-based visual simulations and gaming applications. This study investigates the use of two camera viewpoint modes First-Person Shooter (FPS) and Third-Person Shooter (TPS) in a 3D object movement simulation implemented using OpenGL. The system is developed in Python using the Pygame library and applies basic object transformations, including translation, rotation, and scaling, based on homogeneous coordinates. Both camera modes are evaluated within the same simulation environment consisting of a car object, boundary walls, and obstacles. Experimental results show that the TPS camera mode provides better navigation performance by reducing collision frequency and offering a broader view of the environment, while the FPS camera mode delivers a more immersive experience with limited spatial visibility. Comparative graphs of navigation completion time and collision count highlight clear performance differences between the two camera modes. These results indicate that camera viewpoint selection significantly affects navigation efficiency and user experience in 3D visualization systems. The proposed simulation can serve as a foundation for visual-based control systems, virtual training environments, and educational applications involving spatial navigation.

Keywords: 3D Modeling, OpenGL, Object Transformation, Interactive Simulation.

1. Introduction

Over the past few decades, three-dimensional (3D) object modeling has become a core component in various application domains, including digital games, virtual simulations, digital architecture, and vision-based control systems. Previous studies have shown that 3D modeling enables realistic, interactive, and dynamic representations of objects and environments, making it a fundamental element in the development of intelligent visual systems [1] [2].

A crucial aspect of 3D modeling is object transformation, particularly displacement transformation, which allows objects to move and change orientation dynamically within three-dimensional space. Such transformations are essential not only for visual realism but also for vision-based control systems that require accurate spatial representation and real-time object interaction [3].

Several prior studies have investigated the role of camera perspective in interactive 3D environments. Emmrich et al. [1] and Diego et al. [4] analyzed first-person and third-person perspectives in gaming and virtual reality contexts, focusing mainly on user immersion, comfort, and interaction techniques. Other works have concentrated on rendering efficiency and visualization performance using OpenGL or WebGL frameworks, emphasizing graphical optimization rather than navigation behavior or control effectiveness [2] [5]. These studies demonstrate the importance of viewpoint design in 3D systems but generally treat camera perspective as a design choice without quantitatively evaluating its impact on navigation performance.

In addition, established literature on OpenGL-based graphics systems provides strong theoretical and practical foundations for object transformation and 3D visualization, particularly in experimental and educational simulation environments [6] [7]. However, existing studies rarely integrate a direct comparison of First-Person Shooter (FPS) and Third-Person Shooter (TPS) camera view modes within the same object transformation framework while measuring navigation efficiency using objective performance metrics.

To address this research gap, this study implements and compares FPS and TPS camera view modes in a 3D object movement simulation using OpenGL. In this research, FPS and TPS are explicitly defined as camera viewpoint modes rather than object

Volume 01, Issue 02, December 2025 : 12-21 Received: June 15, 2025. Revised: December 26, 2025

International Journal of Electrical and Intelligent Engineering

transformation methods. The system is developed using Python, Pygame, and OpenGL, and both camera modes are evaluated under identical environmental conditions based on navigation completion time and collision frequency. The objective of this study is to analyze how camera viewpoint selection influences navigation effectiveness and user experience in interactive 3D simulations, thereby contributing to the design of more effective visual-based control systems, virtual training platforms, and educational applications.

2. Method

2.1 System Flow Diagram

The system follows a sequential processing flow. The simulation begins with the initialization of the Pygame and OpenGL environments to configure the display window and graphical settings. Next, the 3D scene is constructed by defining the primary objects, including a car, boundary walls, and obstacles.

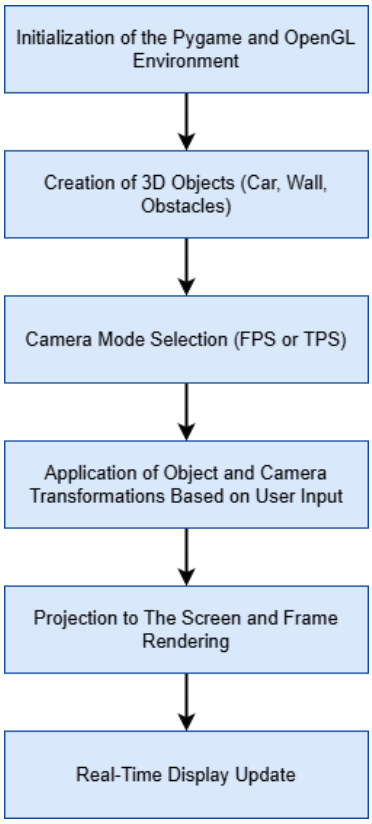

Figure 1. System flow diagram of the 3D object movement simulation using FPS and TPS camera view modes.

The system then selects the camera view mode, either First-Person Shooter (FPS) or Third-Person Shooter (TPS), which determines the user's viewpoint within the simulation environment [8]. Based on user input, object transformations (translation, rotation, and scaling) and camera transformations are applied accordingly. The transformed scene is subsequently projected onto the screen and rendered as visual frames. Finally, the display is updated in real time to ensure that all user interactions and system changes are immediately reflected.

2.2 System Architecture and Development Environment

The system is designed and implemented using the Python programming language, with the Pygame library handling display management and user input, and OpenGL serving as the primary API for 3D graphical visualization [9]. This combination enables the development of a flexible and modular simulation system capable of real-time interaction and rendering.

The system architecture is organized into three main transformation stages: Object Transformation, View (Camera) Transformation, and Projection Transformation. Object Transformation manages the position, orientation, and scale of 3D objects using transformation matrices. View Transformation controls the camera configuration, including the implementation of FPS and TPS camera view modes. Projection Transformation maps the 3D scene onto the 2D display plane for visualization, all of which are implemented through matrix-based transformations in OpenGL.

2.3 Mathematical Representation of 3D Transformations

Each object in a 3D scene is typically defined within its own coordinate system, known as *model space* (also referred to as *local space* or *object space*). When assembling multiple objects into a single scene, their vertices must be transformed from local space into *world space*, which serves as a common reference frame for all objects. This process is referred to as a *world transformation* [10].

Volume 01, Issue 02, December 2025 : 12-21 Received: June 15, 2025. Revised: December 26, 2025

International Journal of Electrical and Intelligent Engineering

The world transformation consists of a sequence of scaling (adjusting the object's size to match world dimensions), rotation (orienting the object axes), and translation (moving the object from the origin to its position in world space). Rotation and scaling are classified as linear transformations because they preserve vector addition and scalar multiplication by definition [6]. Linear transformations combined with translation constitute what is known as an affine transformation [6].

In affine transformations, straight lines remain straight and the ratios of distances between points are preserved. In OpenGL, a vertex **V** located at position *(x, y, z)* is represented as a 3×1 column vector:

$$V = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (1)$$

2.3.1 Scaling

Scaling is a transformation that modifies the size of an object relative to the coordinate axes in three-dimensional space. In 3D graphics, scaling is applied independently along the x, y, and z axes and can be represented using a diagonal scaling matrix. This transformation preserves the object's shape proportions when uniform scaling is applied and is commonly used in object resizing operations in OpenGL-based rendering systems [6] [9].

The scaling transformation matrix **S** is defined as:

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & sz \end{bmatrix} \quad (2)$$

where Sx, Sy, and Sz denote the scaling factors along the x, y, and z axes, respectively. The transformed vertex $\mathbf{V'}$ is obtained through matrix multiplication $\mathbf{V'}$ =S **V** [6].

2.3.2 Rotation

Rotation in three-dimensional space is performed about a specific axis, in contrast to two-dimensional rotation, which occurs around a rotation center. A 3D rotation around the x, y, or z axis by an angle θ\thetaθ, measured counterclockwise following the right-hand rule, can be represented using standard rotation matrices [6] [9].

Rotation around the z-axis is defined as:

$$Rz(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Rotation around the x-axis is defined as:

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix} \quad (4)$$

Rotation around the z-axis is defined as:

$$Ry(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \quad (5)$$

The rotation angles about the x, y, and z axes are commonly referred to as Euler angles, which can be combined to represent an object's arbitrary orientation in 3D space. The resulting transformation is known as an Euler rotation and is widely used in real-time graphics applications [6].

2.3.3 Translation

Translation is a transformation that shifts an object's position in space without altering its orientation or scale. Unlike scaling and rotation, translation is not a linear transformation; however, it can be modeled using vector addition [6]. A translation by a displacement vector $\boldsymbol{d} = [dx, dy, dz]^T$ can be expressed as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} + \begin{bmatrix} x + dx \\ y + dy \\ z + dz \end{bmatrix} \quad (6)$$

In OpenGL, translation is implemented using four-component homogeneous coordinates, where a vertex is represented as (x,y,z,1). Using homogeneous coordinates allows translation to be expressed as a matrix multiplication [6][9]. The translation matrix T(**d**) is defined as:

Volume 01, Issue 02, December 2025 : 12-21 Received: June 15, 2025. Revised: December 26, 2025

International Journal of Electrical and Intelligent Engineering

$$T(\boldsymbol{d}) = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 1 & 0 & 0 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

where $\boldsymbol{d} = [dx, dy, dz]^T$ is the translation vector. The transformed vertex $\mathbf{V}'$ is obtained as:

$$\boldsymbol{V}' = T(\boldsymbol{d})\boldsymbol{V} = \begin{bmatrix} x + dx \\ y + dy \\ z + dz \\ 1 \end{bmatrix} \quad (8)$$

If the homogeneous coordinate $w \neq 1$, the corresponding Cartesian coordinates are obtained by normalization (x/w, y/w, z/w). When $w = 0$, the vector represents a direction rather than a point [6].

2.4 Transformation Implementation within the System

2.4.1 Object transformation

Object transformations are applied to modify the position, orientation, and scale of 3D objects within the simulation environment. In this system, object transformations are implemented using standard OpenGL transformation functions, including `glTranslatef()`, `glRotatef()`, and `glScalef()`, which correspond to translation, rotation, and scaling operations, respectively [6][9]. These transformations are based on matrix operations described in Section 2.3 and are applied in the object's local coordinate system before being mapped to world space.

The transformation parameters are updated dynamically based on system logic and user input, and the resulting transformation matrices are passed to the rendering pipeline to ensure correct spatial positioning of objects during real-time visualization [6].

2.4.2 View Transformation

View transformation is responsible for controlling the camera position and orientation within the 3D simulation environment. In this study, view transformation is used to implement two camera view modes: First-Person Shooter (FPS) and Third-Person Shooter (TPS). Camera movement and orientation are controlled through translation and rotation operations using OpenGL functions such as `glTranslatef()` and `glRotatef()`, while user input is handled via the Pygame library [6] [11].

In FPS mode, the camera is positioned at the object's location, simulating a first-person perspective in which camera motion directly follows object movement. In TPS mode, the camera is positioned at an offset relative to the object, typically behind and above it, providing a broader view of the surrounding environment. The rendered scene is updated in real time using `pygame.display.flip()`, enabling responsive interaction and smooth navigation [11].

2.4.3 Projection Transformation

Projection transformation maps the three-dimensional scene onto the two-dimensional display plane. In this system, perspective projection is applied to simulate depth perception consistent with human visual experience. Perspective projection is implemented using OpenGL projection matrices, allowing objects farther from the camera to appear smaller on the screen [6][9].

Although both FPS and TPS modes utilize the same projection model, the perceived visual output differs due to variations in camera position and orientation. The FPS mode emphasizes immersion by aligning the camera with the object's forward direction, whereas the TPS mode enhances situational awareness by maintaining a wider viewing angle of the environment. This separation ensures that the observed performance differences between FPS and TPS modes are attributed to camera viewpoint configuration rather than differences in object transformation or projection algorithms [6].

2.5 Experimental Setup

The experiment involved 14 participants, all of whom were undergraduate students from STEM-related disciplines. Most participants had prior experience playing 3D games, with more than half classified as intermediate-level players. This background ensured that the participants were familiar with 3D navigation mechanics and camera-based interaction.

Each participant was required to complete a navigation task using both camera modes: First-Person Shooter (FPS) and Third-Person Shooter (TPS). The task consisted of navigating a car object

International Journal of Electrical and Intelligent Engineering

through a virtual 3D environment while avoiding collisions with obstacles and boundary walls. For each trial, participants were instructed to reach the designated endpoint as quickly as possible without intentionally colliding with any obstacles.

The experimental conditions were not identical across trials; the arena layout, initial position, and obstacle configuration differed between runs. However, the camera parameters, control scheme, and transformation mechanisms were kept constant for both FPS and TPS modes to ensure a fair comparison between the two viewpoints.

During each trial, the system automatically recorded performance metrics, including the navigation completion time (in seconds) and the number of collisions encountered. These metrics were logged internally by the system using timestamp-based measurement and collision counters. The values reported in the Results section represent the average performance across all participants for each camera mode.

3. Results and Discussion

This chapter presents the results obtained from the 3D simulation experiments using FPS and TPS view modes. The experiments were conducted with 14 participants, each of whom performed the navigation task using both view modes. For each trial, the system automatically recorded the completion time and the number of collisions. The results are presented through system visualizations, movement simulations, collision detection outputs, and quantitative performance graphs across the following subchapters.

3.1 3D Object Design and Simulation Environment

The simulation environment consists of several 3D elements, including a car as the primary object, walls as arena boundaries, and obstacles serving as navigation barriers. These objects are constructed using basic OpenGL transformations, namely `glPushMatrix()`, `glTranslatef()`, `glRotatef()`, and `glScalef()`, within the world coordinate system. The geometric structure of the car is simplified using basic shapes such as cubes and rectangular prisms, while the obstacles are statically positioned to facilitate collision detection tests.

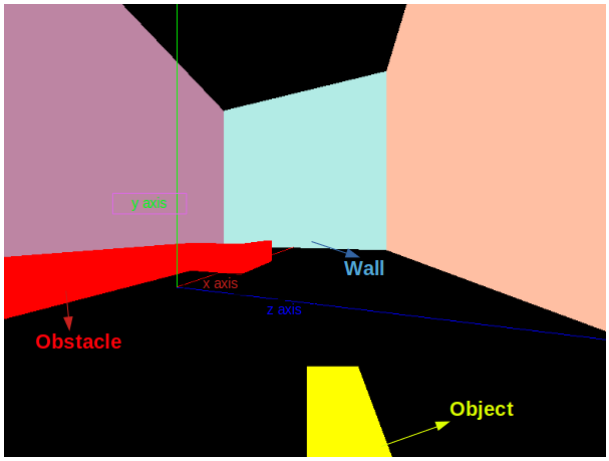

Figure 2. 3D simulation environment showing the car object, boundary walls, and obstacles.

Fig. 2 illustrates the overall 3D simulation environment, including the car object, arena boundaries, and obstacle layout used in the experiment.

3.2 FPS and TPS View Modes

3.2.1 FPS Mode

In FPS mode, the camera is positioned inside the car object, providing a first-person perspective. This setup offers an immersive experience, simulating the sensation of being inside the vehicle and directly controlling its movement direction. However, the main challenge lies in the limited field of view, as surrounding objects remain unseen unless the view direction is manually changed, as illustrated in Fig. 2.

The transformation implementation involves rotation and translation based on user input, as follows:

```
if view.mode == "FPS":
    glRotatef(90, 0, 1, 0)
    glRotatef(-angle, 0, 1, 0)
    glTranslatef(-position.x, -10, -position.z)
```

User input (forward, backward, turn) is captured via `pygame.key.get_pressed()` events and recalculated into motion direction vectors. The movement response appears smooth due to the

International Journal of Electrical and Intelligent Engineering

combination of position updates and dynamic camera angle adjustments.

3.2.2 TPS Mode

In TPS mode, the camera is positioned above and slightly behind the car object. The camera is configured to provide a comprehensive view of the entire arena and surrounding obstacles, granting users broader environmental awareness and control. This mode is commonly employed in strategy-based or navigation-focused games as illustrated in Fig. 3.

The transformation implementation also involves rotation and translation based on user input, as follows:

```
elif view.mode == "TPS":
    glTranslatef(0, 0, -35)
    glRotatef(90, 1, 0, 0)
```

Since the camera is not attached to the object's coordinate system, users can more easily avoid obstacles and plan movement trajectories. This difference in visibility directly impacts the effectiveness of navigation.

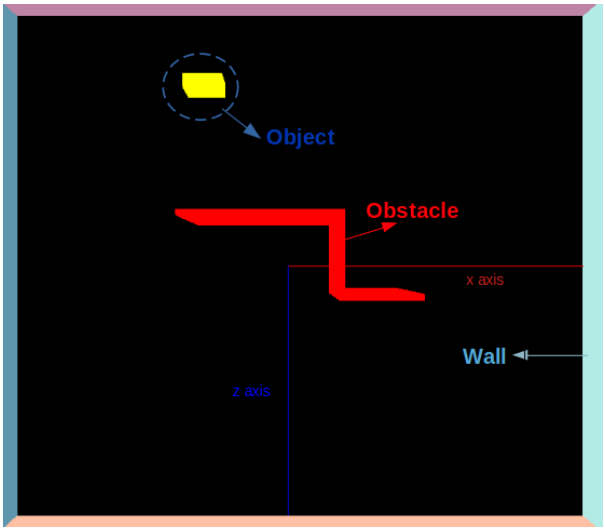

Figure 3. TPS camera view showing the observer's perspective, visualizing the entire track and surrounding objects.

3.3 Vehicle Movement

The simulation supports four primary movements: forward, backward, left turn, and right turn. A combination of rotation and translation generates dynamic and realistic animation of vehicle motion. The direction of movement is calculated using Euler angles, where the directional vector is determined by trigonometric functions `math.cos()` and `math.sin()` based on the current orientation angle.

Linear vehicle movements, including forward and backward motion, are illustrated in Figure 4 and Fig. 5, respectively. These movements represent straight-line translation along the vehicle's forward axis without rotational changes.

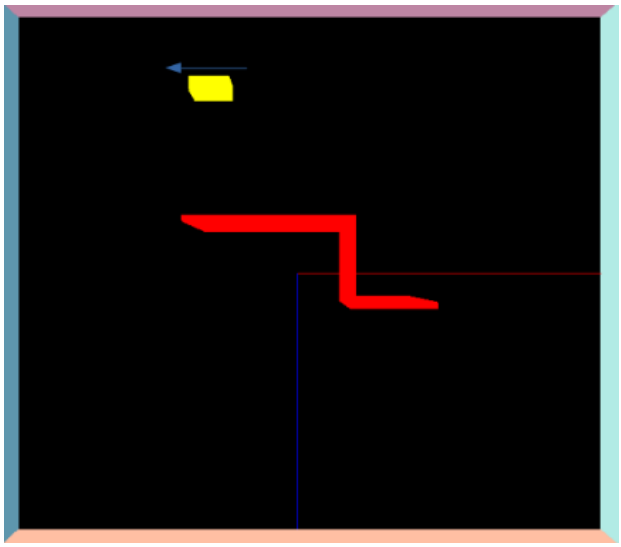

Figure 4. Linear vehicle movement in the forward direction.

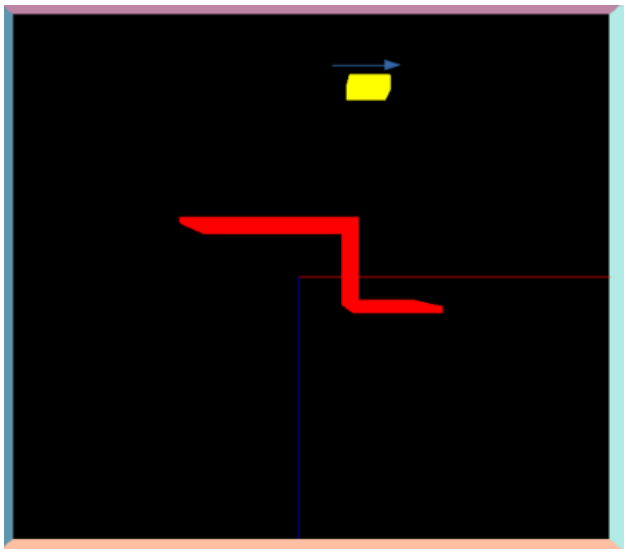

Figure 5. Linear vehicle movement in the backward direction.

Turning motions are only permitted when the vehicle is in motion (either forward or backward), in accordance with realistic physical simulation

International Journal of Electrical and Intelligent Engineering

principles. The combined movements involving translation and rotation are shown in Fig. 6–9, which depict directional combinations such as forward–turn right, forward–turn left, backward–turn left, and backward–turn right.

The smoothness of the animation is maintained through frame-by-frame display refreshing using `pygame.display.flip()`, in combination with `glClear()` to reset the rendering buffer before each frame update.

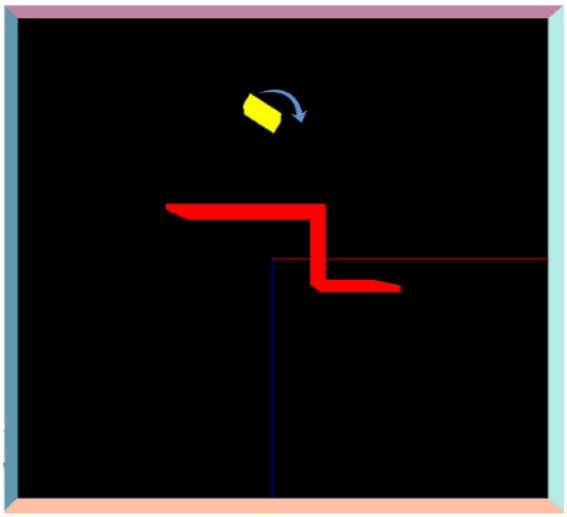

Figure 6. Vehicle movement with directional combination: forward and turn right.

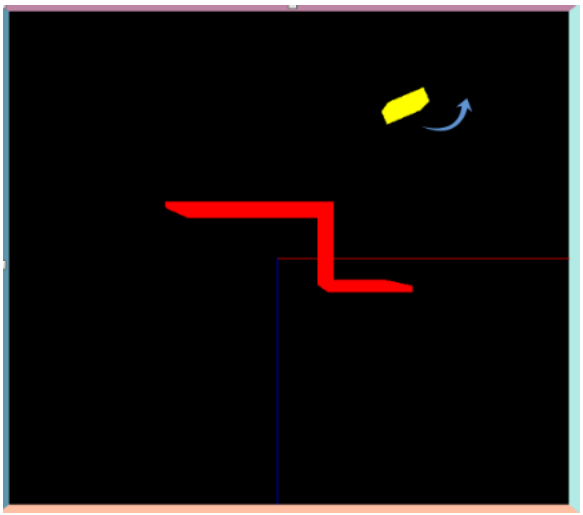

Figure 7. Vehicle movement with directional combination: forward and turn left.

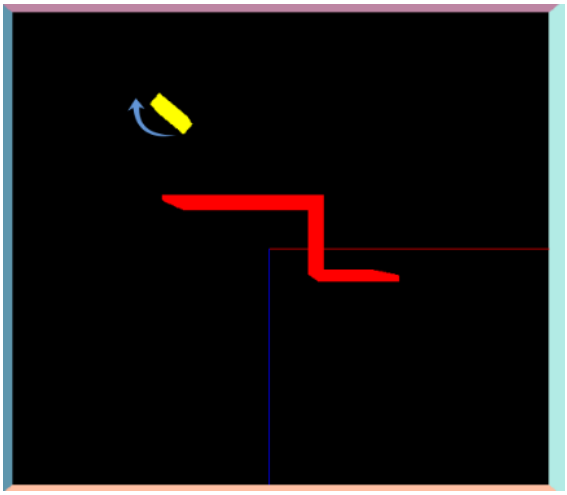

Figure 8. Vehicle movement with directional combination: backward and turn left.

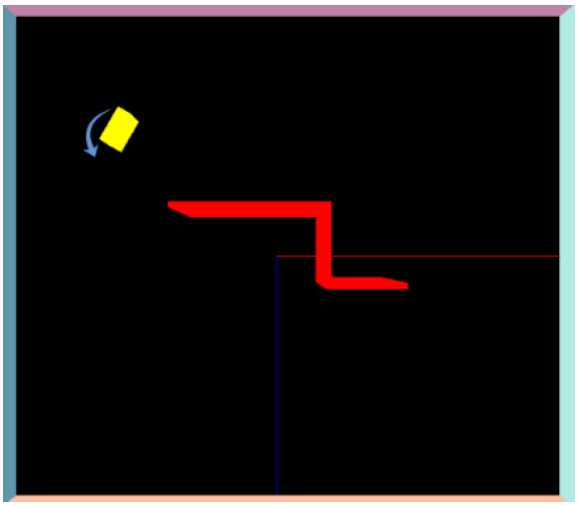

Figure 9. Vehicle movement with directional combination: backward and turn right.

3.4 Collison Detection

The simulation incorporates a collision detection system between the vehicle and both arena boundaries and internal obstacles. The approach used is bounding box testing, which compares the vehicle's position coordinates with the obstacle boundaries and applies position correction in case of penetration.

For obstacle detection, the system calculates the minimum distance between the car and the barrier using absolute differences, namely `abs(position.x_car-position.x_obs)` and `abs(position.z_car-position.z_obs)`. If the computed distance falls below a predefined threshold (e.g., 10 pixels), a collision is assumed to have

Volume 01, Issue 02, December 2025 : 12-21 Received: June 15, 2025. Revised: December 26, 2025

International Journal of Electrical and Intelligent Engineering

occurred, and the vehicle movement is restricted to prevent further penetration.

Fig. 10 illustrates a collision event between the vehicle and the arena wall, while Fig. 11 shows a collision between the vehicle and an internal obstacle.

Figure 10. Visualization of vehicle collision with the arena boundary wall.

Figure 11. Visualization of vehicle collision with an internal obstacle.

3.5 Quantitative Evaluation and Performance Graph

The quantitative results shown in this section represent the average performance across all 14 participants for both FPS and TPS modes. To evaluate the performance of the two implemented approaches, a simple quantitative assessment was conducted based on two parameters:

1. Average time (in seconds) required by users to complete the navigation path
2. Number of collisions occurring during navigation

The quantitative comparison between FPS and TPS modes is summarized in Table 1, which reports the average completion time and average number of collisions for each mode.

Table 1. Average completion time and collision count for FPS and TPS navigation modes.

Mode	Average Time in Seconds	Average Number of Collisions
FPS	28.2	3.4
TPS	25.6	1.8

To further visualize the performance differences, Fig. 12 presents a comparison graph illustrating navigation efficiency based on the two evaluated metrics.

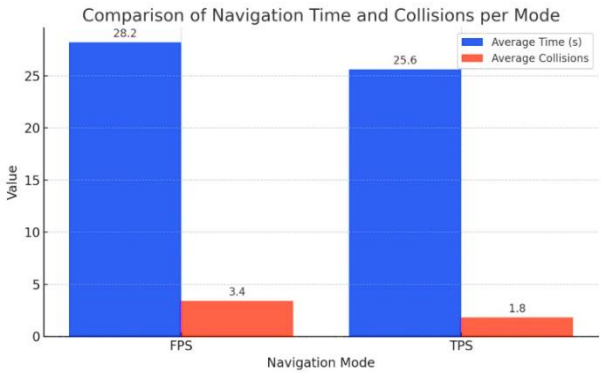

Figure 12. Performance comparison graph illustrating navigation efficiency between FPS and TPS modes based on average completion time and collision frequency.

As shown in Fig. 11, users in FPS mode required an average of 28.2 seconds to complete the navigation task and experienced approximately 3.4 collisions. This result reflects the immersive nature of the first-person perspective, where limited field of view may reduce spatial awareness and increase the difficulty of obstacle avoidance. In contrast, TPS mode achieved a lower average completion time of 25.6 seconds and a reduced collision count of 1.8. This indicates that the third-person perspective provides improved environmental visibility, allowing users to plan movement paths more effectively and avoid obstacles. Overall, the results suggest that TPS mode offers better navigation efficiency, while FPS mode emphasizes immersion with slightly reduced control accuracy.

International Journal of Electrical and Intelligent Engineering

3.6 Analysis

Based on the experimental observations, the TPS mode provides more stable navigation control and results in fewer collisions, as users benefit from a broader field of view of the simulated environment. In contrast, the FPS mode offers a more immersive experience but limits the user's ability to anticipate objects outside the direct line of sight.

Each camera mode presents its own advantages depending on the application context. For action-oriented or high-speed gaming applications, the FPS mode is generally preferred due to its immersive characteristics. However, for training simulations or strategy-based navigation tasks that require accuracy and situational awareness, the TPS mode proves to be more effective.

4. Conclusion

This This study successfully implemented two 3D viewpoint approaches using OpenGL, namely the First-Person Shooter (FPS) and Third-Person Shooter (TPS) techniques. Both approaches were applied to enable interactive and realistic object displacement transformations in three-dimensional space. The experimental results revealed that:

1. TPS mode offers advantages in visual navigation due to its wider perspective, reducing collisions and enhancing the user's ability to understand the spatial context of the environment.
2. FPS mode delivers a more immersive and realistic user experience but suffers from limited visibility, which may increase the likelihood of collisions or navigational errors.

The transformations applied translation, rotation, and scaling were successfully integrated using homogeneous transformation matrices. The system, implemented using Python, Pygame, and OpenGL, demonstrated flexibility and effectiveness in supporting 3D modeling and navigation simulations.

This system shows potential for use in visual-based navigation simulations of autonomous robots or vehicles, serving as a training platform prior to real-world deployment**.** For future research, the system can be extended by incorporating adaptive or hybrid camera modes that dynamically switch between FPS and TPS based on navigation context. Additionally, more complex environments, larger participant groups, and additional performance metrics such as user workload or path optimality can be explored to further evaluate navigation effectiveness. The integration of intelligent control algorithms or machine learning-based navigation strategies also represents a promising direction for future development.

References

[1] K. Emmrich, A. Krekhov and S. Cmentowski, "Streaming vr games to the broad audience: A comparison of the first-person and third-person perspectives," pp. 1-14, 2021.

[2] M. A. AboArab, V. T. Potsika and D. I. Fotiadis, "DECODE-3DViz: Efficient WebGL-Based High-Fidelity Visualization of Large-Scale Images using Level of Detail and Data Chunk Streaming," *Journal of Imaging Informatics in Medicine*, pp. 1-19, 2025.

[3] Sharma, Ajay, Patel, R. Kumar and P. , "Computer vision-based smart monitoring and control system for crop," Springer, 2024, pp. 65-82.

[4] M. Diego, L. H.-N. and W. , "An in-depth exploration of the effect of 2d/3d views and controller types on first person shooter games in virtual reality," *IEEE International Symposium on Mixed and Augmented Reality (ISMAR*, pp. 712-724, 2020.

[5] P. Y. &. M. M. V. Timokhin, "Hybrid Visualization with Vulkan-OpenGL: Technology and Methods of Implementation in Virtual Environment Systems," *Scientific Visualization*, vol. 15(3), p. 7–17, 2023.

[6] S. Guha, Computer Graphics Through OpenGL®: From Theory to Experiments, CRC Press Taylor & Francis Group, 2023.

[7] M. Adnani and A. Z. Falani, "Implementasi Open Gl Untuk Pembuatan Objek 3d," *JOURNAL ZETROEM*, vol. 3, pp. 1-6, 2021.

International Journal of Electrical and Intelligent Engineering

[8] Ahmed, S. Nabeel, Khaliq, Ayesha and Irfan, "Unreal Engine's Realistic War First-Third Person Shooting Game: Fallen Heroes," *INTERNATIONAL JOURNAL OF SPECIAL EDUCATION*, vol. 37, 2022.

[9] Kosarevsky, Sergey, Latypov and Viktor, 3D Graphics Rendering Cookbook: A comprehensive guide to exploring rendering algorithms in modern OpenGL and Vulkan, Packt Publishing Ltd, 2021.

[10] white, C. Mallcolm, Fang, Hongjian and Nakata, "PyKonal: a Python package for solving the eikonal equation in spherical and Cartesian coordinates using the fast marching method," *Seismological Research Letters-Seismological Society of America,* pp. 2378-2389, 2020.

[11] M. Naufal, T. Wiyuna, A. D. Bintarum and A. F. Burhanudin, "Desain Simulasi Gerak Parabola Sebagai Pemanfaatan Pembelajaran Fisika SMA Kelas X Menggunakan Pygame," *Mitra Pilar: Jurnal Pendidikan, Inovasi, dan Terapan Teknologi*, vol. 1 No.1, pp. 155-170, 2022.