Pennsylvania Turnpike Simulator: An Application of ThreeJS

Jeremiah Giordani

Abstract ThreeJS is a powerful framework for creating 3D graphics and interactive applications in the browser using WebGL. This project demonstrates the use of ThreeJS to develop a dynamic and engaging racing game-style application called "Pennsylvania Turnpike Simulator". The game simulates a driving experience on the Pennsylvania Turnpike, dynamically loading road chunks and NPC vehicles, to minimize resource overhead and ensure smooth gameplay. Key features include real-time collision detection, health monitoring, and responsive controls for an immersive user experience. Additionally, the game incorporates a finish line mechanism, checkpoints, and popups for both success and failure scenarios, showcasing how ThreeJS can be leveraged for interactive and visually compelling applications. This writeup details the design, implementation, and challenges faced during the development of the simulator.

Introduction

The goal of this project was to create an engaging racing game-style application using the ThreeJS framework, demonstrating its capabilities for dynamic 3D graphics and interactive simulations in the browser. This section outlines the objectives of the Pennsylvania Turnpike Simulator, discusses related work in driving simulator applications, and details the approach taken.

Goal

The primary goal of this project was to develop an interactive racing-style game that leverages the ThreeJS framework to create a dynamic and visually engaging simulation of driving on the Pennsylvania Turnpike. The game focuses on providing a smooth and responsive experience, incorporating features such as real-time collision detection, NPC interactions, and dynamically loaded road segments to maintain performance. This application is primarily designed for entertainment, appealing to users who enjoy browser-based games with a realistic yet accessible driving mechanic.

Related Work

Driving simulation has been a significant area of research, with applications spanning vehicle design, driver behavior studies, and autonomous systems development. Existing literature provides a comprehensive overview of the components and applications of driving simulators. For example, [2] reviews driving simulator components, such as vehicle dynamics models, motion systems, and virtual environments, and examines how these elements interact with human perceptual systems to create a realistic driving experience. The study also highlights state-of-the-art vehicle simulators and their applications in industry and academia. Similarly, [5] explores the characteristics, functionality, and limitations of mid-level driving simulators, which use animated computer graphics and interactive controls to study driver behavior and highway research. This paper also addresses simulator sickness issues and surveys contemporary driving simulators, making it relevant to discussions of dynamic simulations like ours.

Several works have focused on creating platforms tailored for specific use cases in driving simulation. For instance, CARLA, introduced by [1], is an open-source simulator designed to support autonomous driving research. CARLA provides a flexible platform for training and validating autonomous systems, offering assets like urban layouts and vehicles, and supports the specification of environmental conditions and sensor suites. Similarly, [3] discusses the construction of a low-cost, PC-based driving simulator capable of simulating five degrees of freedom (DOF) motion. This simulator incorporates vehicle dynamics models and driver operation signals, demonstrating how specialized hardware and software solutions can achieve accurate motion simulations. Additionally, [4] emphasizes the increasing use of JavaScript for web-based applications, highlighting the importance of leveraging dynamic features for interactive simulations. These studies provide foundational insights into the design and implementation of dynamic driving simulations, which informed the development of the Pennsylvania Turnpike Simulator.

Approach

The development of the *Pennsylvania Turnpike Simulator* leveraged the COS 426 project template as a foundation, allowing for a structured and efficient setup for working with ThreeJS and browser-based 3D graphics. To create a visually engaging environment, we incorporated 3D models sourced from poly.pizza, which provided high-quality assets for vehicles and road elements. **Outside of these foundational resources, all code and design were developed entirely from scratch**, ensuring originality in implementation and gameplay features.

This approach should work well in the context of interactive, browser-based simulations because the ThreeJS framework is optimized for rendering 3D graphics in real-time, and the COS 426 template is tailored for streamlined integration of features like web deployment, scenic rendering, and event-driven interactions.

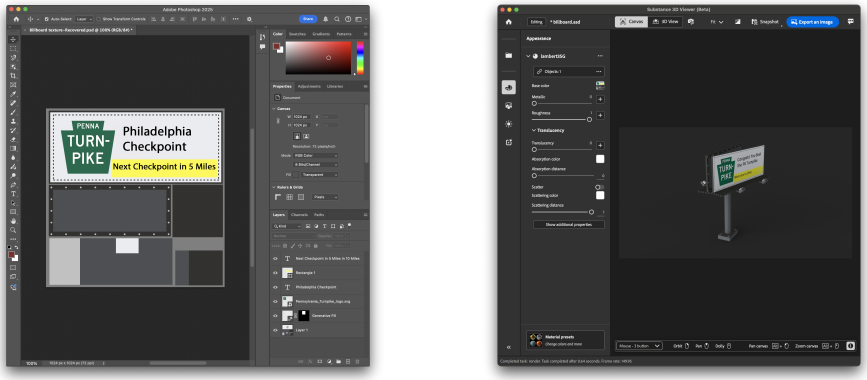
Methodology

The Pennsylvania Turnpike Simulator required implementing several key pieces to execute the approach effectively. The core functionality of the game revolves around four main components: objects, scene utilities, latency management, and the overall scene structure. For each piece, we explored practical implementations to achieve smooth gameplay, balanced visuals, and efficient resource usage. The implementations chosen aimed to prioritize performance and interactivity while maintaining a high level of originality and customization.

Objects

The game features a variety of objects, including basic objects like the road and NPC vehicles, as well as interactive elements such as checkpoints. All 3D models were sourced from an online resource, poly.pizza, which provided high-quality assets that could be integrated seamlessly into the game. However, the checkpoints required additional customization to reflect Pennsylvania Turnpike-specific content.

For the checkpoints, we began by downloading textures and editing them in Photoshop to overlay custom text and designs that matched the theme of the simulator. The modified textures were then applied to the original models using Adobe Substance Viewer, allowing us to visualize and export the updated assets with high fidelity. Figure 1 shows the process of creating these custom models.



(a) *Editing the Texture*

(b) *Creating the Model*

FIGURE 1. *Designing the Checkpoint Models*

Scene Utils

The Pennsylvania Turnpike Simulator required a robust set of utility functions to manage the dynamic and interactive aspects of the game. These utilities included collision detection, player controls, a health bar, road management, and a status display. Figure 2 shows the gameplay interface.

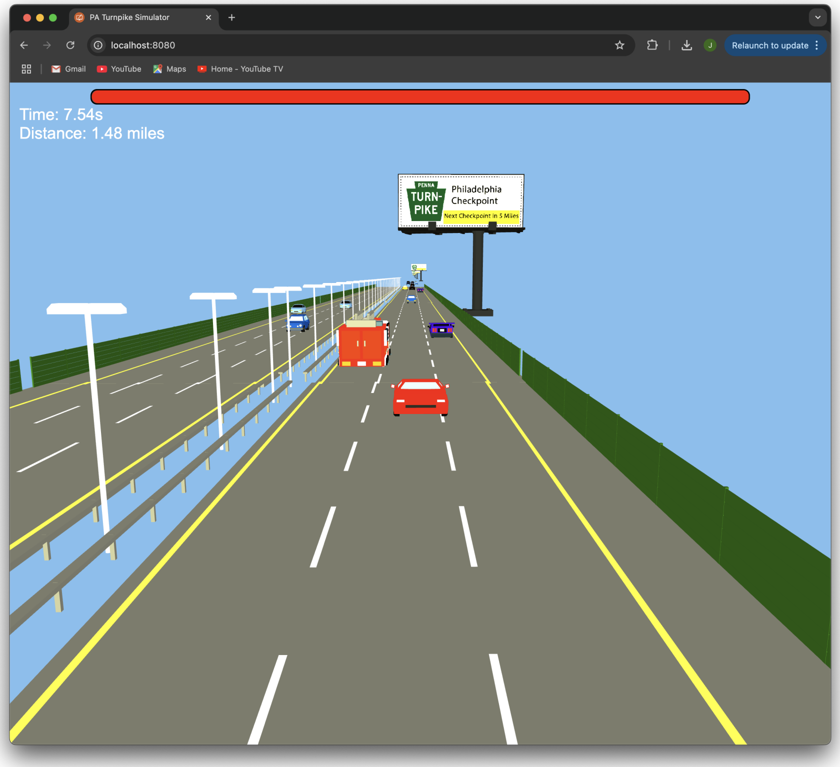


FIGURE 2. *Gameplay*

Collisions:

Collision detection was a critical component for ensuring interactivity with NPC vehicles. We implemented bounding boxes for all moving objects, updating their positions dynamically to detect overlaps. This approach allowed for real-time feedback, such as reducing the player's health or adjusting vehicle speeds during collisions, enhancing the gameplay experience.

Controls:

Player controls were implemented using event listeners to capture keyboard input. This allowed the player to accelerate, decelerate, and steer their car using the 'W', 'A',

’S’, and ’D’ keys. The controls were designed to feel responsive and natural while accounting for factors like friction and deceleration when keys were released.

Health Bar:

The health bar was dynamically rendered on the screen, updating in real time as the player’s health changed due to collisions or other gameplay events. This visual element provided immediate feedback on the player’s status, adding tension and urgency to the gameplay.

Road Management:

The road was dynamically loaded in chunks to minimize latency and manage GPU/CPU usage effectively. As the player progressed, new chunks were preloaded ahead of time, while older chunks were removed, ensuring smooth performance without overloading resources.

Status Display:

A status display at the top of the screen showed the elapsed time and distance traveled in real-time. This feature kept the player informed of their progress and was updated every frame to maintain accuracy.

Together, these scene utilities provided the core functionality required for an interactive, engaging, and smooth gaming experience. They were designed to work seamlessly with the dynamic nature of the scene and the game’s overall objectives.

Minimizing Latency

Efficiently managing resource usage and minimizing latency was critical for ensuring smooth gameplay in the Pennsylvania Turnpike Simulator. Since the road and associated elements dynamically extended as the player progressed, two main approaches were considered: preloading chunks or loading them in real-time. Loading all chunks at once was not viable due to the high resource demand, which would have overwhelmed the GPU and CPU, causing severe latency and performance issues. Similarly, loading chunks in real-time as the player approached each section resulted in noticeable delays and stutters in gameplay.

To address these challenges, we implemented a dynamic loading system that preloads chunks of the road and NPCs ahead of the player. This system ensures a seamless experience by keeping the road segments within the player’s render distance loaded while offloading older chunks to free up resources. Preloading strikes a balance between performance and visual smoothness, eliminating the stuttering that might occur with real-time loading. The final implementation includes mechanisms for caching model templates to avoid redundant file loading and dynamically managing NPCs in both forward and opposing directions. An outline of the code that handles road model loading is provided in Listing 1 The real-time removal of old segments further ensures efficient resource usage without compromising the immersive gameplay experience.

Listing 1. *Reusing Loaded Model Template*

```
    async loadModelTemplate() {
        return new Promise((resolve, reject) => {
            this.loader.load(
                MODEL,
                (gltf) => {
                    this.modelTemplate = gltf.scene;
                    console.log("Model template loaded");
                    resolve();
                }
                ...
            );
        });
    }

    addSegment(chunkNumber) {
        return new Promise((resolve, reject) => {
            if (!this.modelTemplate) {
                reject(new Error("Model template not loaded"));
                return;
            }

            ...

            // Clone the cached model for the new chunk
            const roadModel = this.modelTemplate.clone();

            ...

            // Add the cloned model to the group and track it
            this.add(roadModel);
            this.segments.push({ chunk: chunkNumber, model: roadModel
                });

            ...

            resolve();
        });
    }
```

Scene

The scene structure was built around a class called SeedScene, which acted as the central manager for all elements in the game. This class initialized and controlled the core components, including the road, player, checkpoints, NPCs, and utilities such as collision detection and status display. The scene dynamically updated these elements, ensuring that objects were positioned and managed according to the player's progress.

This modular design allowed for an efficient and flexible structure, ensuring that the scene was responsive to the player's actions while maintaining performance. Together with the utilities and dynamic loading system, the scene provided the foundation for the game's engaging and immersive experience.

Results

The success of the Pennsylvania Turnpike Simulator was measured based on its performance, interactivity, and overall user experience. Key metrics included the frame rate, responsiveness of controls, and smoothness of transitions between road chunks. I employed the help from three peers to play the game and provide feedback on its functionality, performance, and overall user experience. They tested various aspects of the game, including the responsiveness of the controls, the smoothness of transitions between road chunks, and the effectiveness of collision detection. Their feedback highlighted what was working well and identified areas for improvement. Based on their observations, I adjusted the difficulty of the game to increase as the user progresses through the course, ensuring a more engaging and challenging gameplay experience.

The feedback from my peers also confirmed that key features such as the dynamic road loading, NPC interactions, and checkpoint mechanics worked as intended and contributed to a seamless and immersive experience.

Discussion

The approach taken for the Pennsylvania Turnpike Simulator proved to be highly promising, particularly in demonstrating the power of ThreeJS for dynamic and interactive browser-based simulations. The integration of features like responsive controls, real-time collision detection, and adaptive difficulty based on player progression enhanced the gameplay experience significantly. This approach is effective for similar projects where resource constraints and dynamic updates are crucial, making it a strong foundation for future development.

However, there are opportunities to explore variants of this approach to further improve the game. Follow-up work could focus on expanding the game's mechanics, such as adding multiplayer capabilities or more complex AI behaviors for NPCs. Through this project, I learned the importance of balancing performance with interactivity, the value of iterative feedback in refining gameplay, and the challenges of integrating multiple dynamic systems in a real-time application. This experience has laid a solid foundation for tackling more advanced projects in the future.

Ethical Concerns

One ethical concern associated with the Pennsylvania Turnpike Simulator is the potential glorification of reckless driving behaviors. As a game designed to simulate high-speed driving, it could unintentionally normalize unsafe driving practices, such as speeding or weaving between cars, especially for impressionable players. According to the ACM Code of Ethics, computing professionals should avoid harm (*"Avoid harm"* principle), which includes minimizing risks to public safety. To mitigate this concern, the game could incorporate educational elements, such as reminders about safe driving practices or penalties for reckless behaviors. These features would encourage responsible driving and align with the ethical principle of contributing positively to society.

Another potential issue is the competitive nature of the game, which could incentivize players to focus solely on achieving high scores or fast times, disregarding in-game consequences like health depletion or collisions. This aspect could indirectly promote a mindset that prioritizes results over safety, conflicting with the ACM guideline to be fair and not deceptive (*"Be fair and take action not to discriminate"*). To address this, the game could reward cautious and strategic driving by including alternative scoring mechanisms, such as bonuses for avoiding collisions or maintaining steady speeds. Additionally, integrating messaging that emphasizes skillful and considerate driving over raw speed would reinforce positive behaviors.

Conclusion

The Pennsylvania Turnpike Simulator effectively achieved its primary goal of creating an engaging and interactive racing-style game using ThreeJS. The dynamic road loading, real-time collision detection, and adaptive gameplay mechanics provided a smooth and immersive experience that was well-received during testing. Feedback from peers highlighted the strengths of the game, such as its responsiveness and visual appeal, while also guiding improvements like the adjustment of difficulty as players progressed. These successes demonstrate that the project fulfilled its intended objectives.

Future steps for the project include revisiting NPC behavior to make it more intelligent and realistic, as well as expanding the variety of gameplay elements, such as procedurally generated road features and environmental effects. Another key area for improvement is further optimizing performance, particularly when scaling the game for larger or more complex environments. Addressing these issues would enhance the game's longevity and replayability, solidifying its potential as a robust and entertaining application.

References

[1] Lucas Bruck, Bruce Haycock, and Ali Emadi. A Review of Driving Simulation Technology and Applications. In: *IEEE Open Journal of Vehicular Technology* 2 (2021), pp. 1–16. DOI: 10.1109/OJVT.2020.3036582.

[2] Alexey Dosovitskiy et al. CARLA: An Open Urban Driving Simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 1–16. Available at <https://proceedings.mlr.press/v78/dosovitskiy17a.html>.

[3] A.R.W. Huang and Chihsiuh Chen. A low-cost driving simulator for full vehicle dynamics simulation. In: *IEEE Transactions on Vehicular Technology* 52.1 (2003), pp. 162–172. DOI: 10.1109/TVT.2002.807157.

[4] Gregor Richards et al. An analysis of the dynamic behavior of JavaScript programs. In: *SIGPLAN Not.* 45.6 (June 2010), pp. 1–12. DOI: 10.1145/1809028.1806598. Available at <https://doi.org/10.1145/1809028.1806598>.

[5] David H. Weir and Allen J. Clark. A Survey of Mid-Level Driving Simulators. In: *SAE Transactions* 104 (1995), pp. 86–106. Available at <http://www.jstor.org/stable/44473207> (visited on 11/12/2024).