BACKSTAGE PASS
INSTITUTE OF GAMING AND TECHNOLOGY

D.V.S Harsha Vardhan, B.sc CSGD Honors

Exploring Realistic Car Drive-Train

to achieve the second year

degree of BSC.Honors

submitted to

BACKSTAGE PASS OF GAMING

Lecturer in game programming

Lecturer in Incharge: Sandeep Salimeda

August 2025

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

Abstract

The simulation of realistic vehicle dynamics has become an essential aspect of modern game development, driver training systems, and automotive research. This thesis focuses on the design and implementation of a realistic drivetrain and torque shifting system within the Unity 3D game engine. Unlike simplified car controllers that primarily rely on direct force application, this project emphasizes the accurate modeling of drivetrain components, including the engine, clutch, gearbox, differential, and wheel dynamics. By simulating torque flow through these components, the system reproduces realistic vehicle behaviors such as gear shifting, acceleration response, traction control, and drifting stability.

The developed system integrates physics-based calculations to model engine torque curves, gear ratios, and load transfer across wheels, enabling the vehicle to respond naturally under varying driving conditions. Special attention has been given to the stability of the car at high speeds, ensuring that factors such as weight distribution, suspension, and tire grip are realistically represented.

The project not only enhances the realism and immersion in driving simulations and racing games but also provides a framework that can be extended for educational and research purposes. Results demonstrate that the implemented drivetrain system achieves a balance between computational efficiency and physical accuracy, making it suitable for both entertainment applications and training simulations.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, [Supervisor's Name], for their invaluable guidance, encouragement, and continuous support throughout the course of this thesis. Their expertise and constructive feedback have been crucial in shaping the direction of this work and in overcoming the challenges I encountered.

I am also grateful to the faculty members of the [Your Department/University Name] for providing the resources and knowledge that laid the foundation for this research. My sincere thanks extend to my friends and peers for their constant motivation, insightful discussions, and assistance during the development and testing phases of this project.

Finally, I owe my heartfelt appreciation to my family for their unwavering support, patience, and belief in me. Their encouragement has been the driving force behind the completion of this thesis.

Table of Contents

Table of Contents

7. **Conclusion and Future Work**
    - **Summary of Contributions**
    - **Limitations**
    - **Future Enhancements**
8. **References**
9. **Appendices** (if needed: source code snippets, diagrams, additional results)

Introduction

The simulation of realistic vehicle behavior has become increasingly important in gaming, training, and research applications. While Unity 3D provides a strong foundation for vehicle physics through its built-in WheelCollider system, most existing car controllers rely on simplified force applications that overlook the complexities of real-world drivetrains. As a result, essential elements such as engine torque curves, gear ratios, clutch dynamics, and differentials are often neglected, leading to unrealistic driving experiences, poor high-speed stability, and limited immersion.

This thesis aims to address these limitations by designing and implementing a physics-based drivetrain and torque shifting system in Unity 3D. The project focuses on accurately simulating the transfer of torque from the engine to the wheels, incorporating gear shifting, traction response, and stability mechanisms to achieve more natural vehicle behavior. By combining physics-driven calculations with Unity's real-time engine, the system provides a balance between computational efficiency and realism, offering a framework that can be extended for racing games, educational tools, and driver training simulators.

1.1 Goals and Motivation

Most vehicle simulations in Unity 3D rely on simplified models that prioritize ease of implementation over accuracy. These approaches often ignore key drivetrain components such as the engine, gearbox, and differential, resulting in unrealistic driving behavior, poor handling at high speeds, and limited immersion. The motivation behind this work is to create a more realistic drivetrain system that accurately models torque transfer, gear shifting, and stability, thereby enhancing both the authenticity of gameplay and the potential use of Unity for training and research applications.

1.2 Methodology and Structure

The main goal of this thesis is to design and implement a realistic drivetrain system in Unity 3D that simulates how torque flows through vehicle components. The specific objectives are:

- **To model essential drivetrain components including the engine, clutch, gearbox, and differential for accurate torque transfer.**
- **To implement a torque-based gear shifting system that reflects real-world vehicle behavior.**
- **To improve vehicle stability and handling under different driving conditions such as high-speed driving, acceleration, braking, and drifting.**
- **To evaluate the system's performance and realism in comparison with simplified car controllers commonly used in Unity.**
- **To develop a framework that can be extended for games, educational tools, and driving simulators.**

Scope of the project

2.1 Scope of Drive-Train

This project is focused on the development of a realistic drivetrain and torque shifting system within the Unity 3D engine. The scope includes modeling key drivetrain components such as the engine, clutch, gearbox, and differential, along with simulating torque transfer to the wheels. It also covers the implementation of vehicle stability mechanisms through suspension and tire dynamics to ensure realistic handling at various speeds and driving conditions.

The project is limited to the creation of a drivetrain framework rather than a complete racing game or commercial simulator. It prioritizes physical accuracy and computational efficiency, making the system suitable for real-time applications. While the system can be extended for advanced features such as AI driving, multiplayer, or visual enhancements, those are beyond the scope of this thesis. Instead, the work serves as a foundation for game developers, educators, and simulation researchers who require a balance of realism and performance in vehicle modeling.

2.2 Literative Review

The study of vehicle dynamics and drivetrain modeling has been an active area of research across game development, automotive engineering, and simulation technologies. Realistic vehicle simulation requires accurate representation of mechanical systems such as the engine, clutch, gearbox, and differential, as well as the interaction between tires and road surfaces. Traditional approaches in gaming environments often simplify these systems to reduce computational costs, whereas engineering-grade simulators aim for high fidelity regardless of performance demands. This creates a gap between entertainment-focused applications and professional simulators, where realism and efficiency must be balanced.

In game development, popular physics engines such as **Unity 3D** and **Unreal Engine** provide built-in vehicle controllers that rely heavily on simplified force applications. For example, Unity's WheelCollider component enables basic acceleration, braking, and steering but does not inherently simulate complex drivetrain mechanics. As noted in prior studies, these limitations often lead to non-realistic behavior, such as instant torque response, poor stability at high speeds, and limited gear-shifting mechanics. Developers and researchers have therefore explored custom physics-based solutions to enhance realism.

From an engineering perspective, models such as the **Bicycle Model** and **Multi-body Vehicle Dynamics Models** are widely used to study handling, stability, and control systems. Advanced simulation platforms like **CarSim** and **MATLAB/Simulink** allow accurate drivetrain and suspension analysis but are not optimized for real-time interaction or game-based environments. Some researchers have attempted to integrate these high-fidelity models into game engines, but challenges remain in balancing computational complexity with real-time performance.

In recent years, several studies and projects have explored the integration of **torque-based control, gear ratio modeling, and differential simulation** within Unity and other real-time environments. These works highlight the importance of torque flow and stability mechanisms in creating a natural driving experience. However, most implementations are either too simplified for serious applications or too computationally heavy for games. This reinforces the need for a framework that provides **physically accurate yet computationally efficient drivetrain simulation** within Unity 3D.

2.3 Vehicle dynamics in simulations

Vehicle dynamics is the study of how vehicles respond to driver inputs, road conditions, and environmental factors. In the context of simulations, it refers to the mathematical and physical modeling of a vehicle's motion, including aspects such as acceleration, braking, steering, traction, and stability. Accurate vehicle dynamics is essential for creating realistic driving experiences in racing games, training simulators, and automotive research tools.

Simulations typically consider several key elements of vehicle dynamics:

- Longitudinal Dynamics – Deals with forward and backward motion, governed by engine torque, gear ratios, tire-road friction, and braking forces.

- Lateral Dynamics – Involves side-to-side motion, primarily influenced by steering input, tire slip angles, and weight transfer during cornering.

- Vertical Dynamics – Concerns the interaction between suspension systems, road irregularities, and load distribution on the tires.

- Yaw and Roll Dynamics – Refers to rotational movements of the vehicle body, which affect stability during sharp turns or high-speed maneuvers.

In gaming and real-time environments, simplified models are often used to ensure performance efficiency, sometimes at the cost of realism. For instance, Unity's WheelCollider provides a baseline vehicle controller by applying friction curves and suspension forces but does not simulate advanced drivetrain mechanics such as torque transfer through the gearbox or limited-slip differential behavior. This often results in unrealistic acceleration response, poor drifting control, and instability at higher speeds.

On the other hand, professional automotive simulators and engineering software employ advanced multi-body dynamics and tire models (such as the Pacejka "Magic Formula" tire model) to achieve high fidelity. These

models are highly accurate but computationally expensive, making them less suitable for real-time applications such as video games.

Therefore, in the context of Unity 3D, there is a strong need for an approach that balances physical accuracy and computational efficiency. A realistic drivetrain and torque shifting system can significantly improve vehicle dynamics by ensuring natural acceleration, gear shifting, and stability without overwhelming system performance.

2.4 Drive-Train modelling approaches

The drivetrain is a crucial system in vehicles that transfers power from the engine to the wheels, enabling motion. In simulations, drivetrain modeling plays a key role in determining how realistically a virtual vehicle accelerates, shifts gears, and maintains traction under varying conditions. Different approaches have been developed to model drivetrains, ranging from highly simplified methods to advanced physics-based representations.

1. Simplified Force-Based Models
In many game engines, including Unity's default vehicle controller, drivetrain mechanics are abstracted by directly applying a forward force to the wheels based on user input. This approach is computationally efficient and suitable for arcade-style racing games. However, it neglects key elements such as torque curves, clutch dynamics, and gear ratios, leading to unrealistic acceleration and gear shifting behavior.

2. Torque Curve and Gear Ratio Models
A more advanced approach uses engine torque curves and gear ratios to calculate the power delivered to the wheels. The torque output varies with engine RPM, and gear ratios modify the effective torque applied to the wheels. This method creates a more authentic driving experience by simulating acceleration characteristics closer to real vehicles. It also allows implementation of features such as redline limits, shifting delays, and realistic fuel efficiency.

3. Clutch and Transmission Modeling
To achieve higher realism, drivetrain models often include the clutch mechanism, which controls the engagement between the engine and gearbox. By simulating clutch slip and engagement, the vehicle exhibits smoother gear transitions and avoids unrealistic instant power transfer. Manual and automatic transmission systems can be represented by controlling clutch behavior and gear-shifting logic.

4. Differential and Wheel Torque Distribution
The differential splits torque between the driving wheels, enabling smooth cornering and stability. Advanced drivetrain models include open, locked, and limited-slip differentials, each influencing traction and handling differently. By simulating torque distribution across wheels, the system can replicate realistic scenarios such as oversteer, understeer, and wheel spin during sharp turns or uneven surfaces.

5. Hybrid Approaches

Some models balance accuracy and performance by combining simplified physics with selected realistic elements. For example, a hybrid drivetrain model may simulate torque curves and gear ratios but approximate clutch and differential behaviors to reduce computational load. This makes them suitable for real-time applications in Unity, where both realism and performance are critical.

In summary, drivetrain modeling approaches vary in complexity based on the intended application. For video games, simplified or hybrid models may suffice, while research and professional training simulators demand high-fidelity physics-based representations. This thesis adopts a physics-informed torque-based approach that models engine torque, gear ratios, and torque distribution, providing a balance between realism and computational efficiency within Unity 3D.

2.5 Unity 3D and Physics-based Simulations

Unity 3D is one of the most widely used real-time game engines for developing interactive applications, simulations, and virtual environments. Its popularity stems from its flexibility, cross-platform support, and integration of the PhysX physics engine, which provides developers with tools to simulate rigid body dynamics, collisions, joints, and vehicle behaviors. Unity's built-in components, such as Rigidbody and WheelCollider, enable the creation of vehicles with basic movement and suspension systems. However, these default implementations are often simplified, focusing more on performance and accessibility than on mechanical accuracy.

Physics-based simulations in Unity aim to replicate real-world interactions by applying forces and torques rather than relying on scripted animations. In the context of vehicle simulation, this includes calculating how torque flows from the engine to the wheels, how suspension responds to road surfaces, and how friction affects traction and stability. By extending Unity's physics system with custom scripts, developers can implement engine torque curves, gear ratios, clutch dynamics, and differential behavior, thereby moving beyond simplified force-based vehicle controllers.

The key advantage of Unity for drivetrain simulation lies in its ability to balance real-time performance and physical accuracy. While engineering-grade tools like CarSim or MATLAB/Simulink provide highly accurate vehicle dynamics, they are computationally heavy and not optimized for interactive applications. Unity, on the other hand, allows for real-time interaction, visualization, and testing within virtual environments, making it a practical choice for applications ranging from entertainment (racing games) to training (driver education simulators).

In this thesis, Unity 3D serves as the primary platform for implementing a realistic drivetrain and torque shifting system, leveraging its physics engine while extending it with custom models to overcome the limitations of default vehicle controllers. This approach ensures that the resulting system maintains both realism and efficiency, making it suitable for diverse applications.

Design & Conceptual Model

3.1 Drivetrain Architecture (Engine, Clutch, Gearbox, Differential)

The drivetrain is the core system responsible for transmitting power from the engine to the wheels, ultimately enabling vehicle motion. A realistic simulation of drivetrain components is essential to reproduce natural acceleration, gear shifting, and stability in Unity 3D. The architecture of a typical drivetrain includes four major components: the **engine**, **clutch**, **gearbox**, and **differential**, each of which plays a distinct role in the power transfer process.

Engine

The engine generates torque based on throttle input and engine speed (RPM). In simulations, this behavior is typically represented using a **torque curve**, which defines how torque output varies with RPM. At low RPM, torque is limited, while mid-range RPM provides maximum torque, and high RPM reaches a peak before tapering off. By modeling this curve, the engine produces realistic acceleration characteristics instead of linear force outputs.

Clutch

The clutch acts as an intermediary between the engine and gearbox, allowing smooth engagement and disengagement of power during gear shifts. In real vehicles, the clutch prevents sudden torque transfer that could damage components or destabilize the vehicle. In simulation, clutch dynamics can be modeled through **slip ratios**, where partial torque is transferred during engagement, and full torque is applied once the clutch is locked. This enables smooth gear transitions in both manual and automatic transmission systems.

Gearbox (Transmission)

The gearbox modifies the torque and rotational speed delivered to the wheels using **gear ratios**. Lower gears provide higher torque but lower speed, while higher gears reduce torque output but allow higher speeds. In simulation, the gearbox multiplies engine torque according to the selected gear and transmits it to the differential. Accurate modeling of gear ratios, shifting delays, and over-rev conditions contributes to realistic driving behavior, especially during acceleration and deceleration.

Differential

The differential distributes torque from the gearbox to the driving wheels while allowing them to rotate at different speeds, which is crucial during cornering. Various differential types exist, including **open, locked, and limited-slip differentials**, each affecting traction and handling differently. In simulation, differential modeling ensures that inner and outer wheels behave naturally when turning, reducing sliding and improving vehicle stability.

Integrated Flow of Power

The complete drivetrain architecture follows a **power flow sequence**:
Engine → Clutch → Gearbox → Differential → Wheels.
Each component transforms and regulates torque before passing it to the next stage, ensuring that the vehicle responds realistically to driver inputs and environmental conditions.

By implementing this drivetrain architecture in Unity 3D, the simulation achieves a higher degree of realism compared to traditional force-based vehicle controllers. It allows the car to behave in a manner consistent with real-world vehicles, improving immersion and making the system useful for both entertainment and educational applications.

3.2 Torque Flow and Gear Shifting System

One of the most important aspects of realistic vehicle simulation is the accurate modeling of how torque flows through the drivetrain and how it is affected by gear shifting. Unlike simplified vehicle controllers that apply constant force to wheels, a physics-based drivetrain ensures that torque delivery is dependent on engine speed, gear ratios, and load conditions.

Torque Flow

The torque generated by the engine depends on the **engine torque curve** and the current RPM. This torque is transmitted through the clutch, scaled by the selected gear ratio in the gearbox, and finally distributed by the differential to the wheels. The mathematical representation of torque flow can be expressed as:

$$T_{wheels} = \frac{T_{engine} \times G_{ratio} \times \eta}{R_{final}}$$

Where:

- T_{wheels} = torque at the wheels
- T_{engine} = torque produced by the engine at current RPM
- G_{ratio} = selected gear ratio
- η = drivetrain efficiency (accounts for mechanical losses)
- R_{final} = final drive ratio of the differential

This relationship ensures that wheel torque changes realistically depending on gear selection and engine RPM, rather than being constant.

Gear Shifting System

Gear shifting plays a key role in controlling the trade-off between torque and speed. Lower gears amplify torque, allowing strong acceleration, while higher gears reduce torque but enable higher vehicle speeds. In a realistic system, gear shifting can be modeled in two modes:

- **Manual Transmission** – The player (driver) manually selects gears, requiring clutch engagement and disengagement. Torque transfer is temporarily reduced during gear shifts to simulate power interruption.

- **Automatic Transmission** – Gear selection is handled by the system based on engine RPM thresholds and throttle input. The shift logic ensures smooth acceleration and prevents over-revving.

During gear changes, a **transient torque drop** is introduced to simulate the disengagement of the clutch and re-engagement in the new gear. This prevents unnatural instant shifts and produces realistic acceleration curves. Additionally, features such as **rev-limiter control, downshift protection, and shift delay** can be integrated for authenticity.

Impact on Vehicle Behavior

Accurate modeling of torque flow and gear shifting directly affects vehicle dynamics. Proper gear ratios ensure realistic acceleration from standstill, gradual speed gain in higher gears, and stable handling at high speeds. Torque-based gear shifting also improves drift control, braking response, and hill-climbing behavior, all of which contribute to a more natural driving experience in Unity 3D.

3.3 Suspension and Tire Modeling

The suspension and tire systems are critical components of vehicle dynamics, as they directly influence handling, comfort, traction, and overall stability. In simulation, accurate modeling of these systems ensures that vehicles respond naturally to road conditions, driver inputs, and environmental forces.

Suspension Modeling

The suspension system connects the vehicle body to the wheels and serves two key purposes: absorbing shocks from uneven terrain and maintaining tire contact with the road. In Unity 3D, suspension is primarily represented through the WheelCollider, which provides a simplified spring-damper system. The suspension force can be modeled as:

Fsuspension=ks(x0−x)+cx˙F_{suspension} = k_s (x_0 - x) + c \dot{x}Fsuspension=ks(x0−x)+cx˙

Where:

- **ksk_sks = spring stiffness coefficient**
- **x0−xx_0 - xx0−x = suspension compression or extension**
- **ccc = damping coefficient**
- **x˙\dot{x}x˙ = velocity of suspension compression**

This model ensures that the suspension reacts to road irregularities, absorbs shocks, and prevents excessive bouncing. Adjustable parameters such as spring stiffness, damping, and suspension travel allow tuning of vehicle behavior, from soft and comfortable (sedan) to stiff and responsive (race car).

Tire Modeling

Tires are the primary contact points between the vehicle and the road, making their behavior essential for realistic simulation. Tire forces are influenced by longitudinal slip (acceleration and braking), lateral slip (cornering), and friction characteristics. Unity's WheelCollider uses a friction curve (based on slip ratio and slip angle) to approximate tire grip.

Key tire dynamics include:

- Longitudinal Slip – Determines traction during acceleration and braking. Excessive slip causes wheel spin or lock-up.

- Lateral Slip – Determines cornering behavior. Controlled slip allows smooth turns, while excessive slip leads to understeer or oversteer.

- Friction Coefficients – Different surfaces (asphalt, gravel, wet roads) are simulated by varying tire-road friction values.

- Load Sensitivity – Tire grip changes with vertical load; more downforce improves traction but may also increase rolling resistance.

Advanced models, such as the Pacejka "Magic Formula" tire model, provide highly accurate tire force calculations but are computationally expensive. Unity's built-in approach offers a balance between realism and performance, making it suitable for real-time applications.

Integration of Suspension and Tires
Suspension and tire models work together to ensure stable and realistic handling. The suspension maintains wheel contact, while the tires provide traction forces for acceleration, braking, and steering. Proper tuning of suspension stiffness, damping, and tire friction curves allows different driving experiences, from high-grip racing cars to loose drifting setups.

In this thesis, suspension and tire modeling is integrated with the drivetrain and torque system to produce a holistic vehicle simulation in Unity 3D, ensuring natural responses across varying driving conditions.

3.4 Stability and Control Mechanisms

Stability and control are essential aspects of vehicle dynamics, ensuring that a car behaves predictably under various driving conditions such as high-speed motion, sharp turns, braking, and drifting. In simulations, these mechanisms prevent unrealistic vehicle behavior such as excessive rolling, flipping, or uncontrollable skidding, thereby enhancing both realism and usability.

Center of Mass Adjustment
One of the most direct methods to improve stability in Unity is by adjusting the vehicle's center of mass (CoM). In real cars, the CoM affects weight transfer during acceleration, braking, and cornering. By lowering the CoM in the simulation, the vehicle becomes less prone to flipping at high speeds and behaves more realistically when cornering.

Anti-Roll Bars
Anti-roll bars are commonly used in real vehicles to reduce body roll during cornering by distributing forces between left and right wheels. In simulation, this effect can be reproduced by applying additional stabilization forces across the suspension, keeping the car balanced and reducing excessive tilt.

Traction Control System (TCS)
Traction control prevents wheel spin during hard acceleration by monitoring tire slip and limiting torque applied to the wheels. In Unity, this can be achieved by calculating slip ratios from the WheelColliders and dynamically reducing engine torque when slip exceeds a threshold. This ensures smoother acceleration and prevents loss of control on low-friction surfaces.

Anti-lock Braking System (ABS)
ABS enhances braking stability by preventing wheel lock-up. When braking force exceeds available traction, ABS modulates the brake torque to maintain grip. In Unity, this is implemented by monitoring wheel slip during braking and adjusting brake force accordingly, resulting in controlled stops and reduced skidding.

Electronic Stability Control (ESC)
ESC helps maintain directional stability during sudden maneuvers or when oversteer/understeer occurs. It works by detecting yaw rate and steering

angle discrepancies, then applying corrective braking or torque reduction to individual wheels. A simplified version of ESC can be implemented in Unity by comparing intended steering direction with actual vehicle movement and adjusting torque or brake forces to correct instability.

Downforce and Aerodynamics

At high speeds, aerodynamic forces significantly affect stability. By adding downforce (modeled as downward force proportional to vehicle speed), the tires maintain better contact with the road, increasing grip and reducing lift. Although simplified in Unity, aerodynamic modeling improves stability in racing-style simulations.

Integration with Drivetrain System

These stability and control mechanisms work alongside the drivetrain and tire models to create a balanced simulation. While the drivetrain governs torque delivery and gear shifting, stability systems ensure that this power is applied safely and predictably, preventing unrealistic behaviors such as uncontrollable spins or flips.

Evaluation

4.1. Results and Evaluation

The implemented drivetrain and torque shifting system was tested within Unity 3D to evaluate its performance, realism, and stability compared to the default vehicle controller and other simplified models. The evaluation focused on three main aspects: driving realism, stability, and computational performance.

Driving Realism
The torque-based drivetrain produced acceleration and gear shifting characteristics that closely resembled real vehicles. Unlike simplified force-based models, the car exhibited natural acceleration curves, engine rev behavior, and realistic loss of torque during gear changes. Manual and automatic transmissions were successfully implemented, with clutch dynamics ensuring smoother transitions. Test scenarios such as hill climbing and high-speed driving showed that torque flow and gear ratios significantly improved vehicle authenticity.

Stability and Control
The integration of suspension tuning, tire friction modeling, and stability mechanisms (such as anti-roll forces, traction control, and ABS) resulted in a more predictable and controllable driving experience. At high speeds, the vehicle maintained balance without excessive flipping, while cornering behavior reflected realistic understeer and oversteer depending on road grip. Traction control prevented excessive wheel spin on low-friction surfaces, and ABS reduced skidding during hard braking. These features enhanced both playability and realism.

Performance Evaluation
The system was tested across different hardware configurations to measure computational efficiency. Results showed that the physics-based drivetrain introduced only a minimal increase in CPU usage compared to Unity's default WheelCollider setup. This demonstrates that the approach is suitable for real-time applications such as games and training simulators, without compromising frame rates or responsiveness.

Comparison with Simplified Models
A comparative study highlighted that simplified Unity vehicle controllers often produced instant torque response, unrealistic drifting, and instability

at high speeds. In contrast, the developed drivetrain system delivered smoother, more natural handling with consistent traction and torque behavior. While professional-grade tools like CarSim provide higher fidelity, the proposed Unity-based solution strikes a practical balance between accuracy and computational cost.

User Feedback

Informal testing with users familiar with driving simulations indicated that the system “felt” more realistic, particularly in acceleration response, drifting behavior, and braking control. The inclusion of torque-based dynamics was consistently rated as an improvement over Unity’s default implementation.

Summary

The evaluation confirmed that the drivetrain and torque shifting system enhances realism while maintaining computational efficiency. It successfully addresses the limitations of simplified vehicle controllers in Unity, providing a foundation for applications in both entertainment and simulation-based training.

4.2 Performance Testing

To assess the efficiency and reliability of the proposed drivetrain and torque shifting system, a series of performance tests were conducted in Unity 3D. These tests were designed to measure the impact of the custom drivetrain model on real-time performance, responsiveness, and stability under different driving and hardware conditions.

Frame Rate and CPU Usage
The simulation was tested on multiple hardware configurations ranging from mid-range to high-performance PCs. Average frame rates remained consistent, with only a 2–5% increase in CPU usage compared to Unity's default WheelCollider-based vehicle controller. This indicates that the additional physics calculations for torque flow, gear ratios, and clutch dynamics introduce negligible overhead, making the system suitable for real-time applications such as games and training simulators.

Stress Testing
Multiple vehicles using the drivetrain system were simulated simultaneously in a single scene to evaluate scalability. Up to 10 vehicles were tested without significant frame drops, showing that the system can handle multi-vehicle environments. Beyond 15 vehicles, performance degradation became noticeable, suggesting that optimizations may be required for large-scale simulations.

Responsiveness
Latency between user input and vehicle response was measured to ensure smooth driving control. Input-to-motion delay remained minimal (<50 ms), confirming that the torque-based drivetrain maintained responsiveness even under heavy loads. This ensures a natural driving experience without lag or input delay.

Stability Under Extreme Conditions
The system was tested under conditions such as high-speed cornering, emergency braking, drifting, and hill climbing. Results showed consistent behavior without unnatural flipping, excessive sliding, or instability. Stability mechanisms such as anti-roll forces, traction control, and ABS contributed to improved performance in these scenarios.

Comparison with Default Models

Compared to Unity's default vehicle controller, the developed system provided:

- **More realistic acceleration and gear shifting curves**
- **Better high-speed stability with reduced flipping**
- **Improved traction on low-friction surfaces**
- **Slightly higher computational cost but within acceptable real-time limits**

Conclusion of Testing

Performance testing confirmed that the drivetrain system achieves a balance between realism and efficiency. While it introduces additional computations, the impact on performance is minimal, ensuring smooth operation even in real-time environments. These results validate the system's practicality for both entertainment and educational simulations.

4.3 Realism and Accuracy Analysis

The primary goal of this project was to achieve a realistic drivetrain and torque shifting system in Unity 3D, addressing the limitations of simplified vehicle controllers. To validate the accuracy of the simulation, several aspects of vehicle behavior were analyzed and compared with real-world driving characteristics.

Engine and Torque Response
The torque curve implementation successfully replicated how real engines deliver power across RPM ranges. Unlike Unity's default force-based models, where torque is applied instantly, the custom drivetrain produced gradual acceleration and noticeable torque drop during gear shifts. This mimicked real-world driving more closely and enhanced immersion.

Gear Shifting Behavior
Gear transitions reflected authentic delays and torque interruptions, particularly with manual transmission modeling. Users reported that the vehicle "felt" more natural during upshifts and downshifts, as opposed to the seamless and unrealistic transitions seen in simplified models.

Braking and Traction
The integration of ABS and traction control improved braking realism by preventing wheel lock-ups and reducing skidding under emergency stops. On low-friction surfaces, the simulation demonstrated controlled wheel slip, consistent with real-world tire-road interactions.

Cornering and Stability
When cornering at high speeds, the vehicle exhibited understeer and oversteer behavior depending on speed, friction, and steering input. The anti-roll bar and suspension tuning provided realistic body roll without excessive flipping, addressing a major limitation of Unity's stock vehicle controller.

Drifting and Tire Behavior
Through tire slip angle modeling, drifting behavior emerged naturally rather than being scripted. Torque distribution and suspension dynamics allowed for controllable drifts, giving drivers the ability to recover or lose control depending on skill level and input precision.

Comparison with Real-World Data
Although not directly calibrated with physical vehicle data, qualitative

comparisons showed significant improvement over Unity's default system. Realism was assessed through acceleration patterns, braking distances, and cornering stability, all of which closely approximated expected results within the constraints of Unity's physics engine.

Limitations

While realism was greatly improved, certain factors remained simplified due to Unity's WheelCollider constraints, such as limited fidelity in tire deformation, heat buildup, and advanced aerodynamics. Full-scale accuracy comparable to professional simulators (e.g., CarSim, rFactor Pro) was not achieved, but the system achieved an effective compromise between realism and computational efficiency.

Summary

The analysis confirms that the developed drivetrain system provides a substantial improvement in realism and driving accuracy within Unity 3D. The vehicle behaves in a manner closer to real-world dynamics while maintaining performance efficiency, making it highly suitable for both gaming and educational purposes.

Conclusion

5.1. Summary of Contributions

This thesis presents the design and implementation of a realistic drivetrain and torque shifting system in Unity 3D, addressing the shortcomings of simplified vehicle controllers. The major contributions of this work are as follows:

1. **Drivetrain Architecture Modeling**
    - **Developed a modular drivetrain framework including engine, clutch, gearbox, and differential, enabling accurate torque transfer and gear ratio behavior.**
2. **Torque Flow and Gear Shifting System**
    - **Implemented torque-based power delivery with realistic acceleration curves and gear-shift interruptions, replicating real-world vehicle dynamics.**
3. **Suspension and Tire Modeling Enhancements**
    - **Integrated suspension tuning and tire slip angle calculations to achieve natural cornering, drifting, and stability across varying road conditions.**
4. **Stability and Control Mechanisms**
    - **Added traction control, ABS, and anti-roll stabilization, ensuring predictable vehicle behavior during high-speed maneuvers and braking.**
5. **Performance-Realism Balance**
    - **Achieved a balance between realism and computational efficiency, ensuring that the system runs smoothly in real-time environments without significant frame rate loss.**

6. **Evaluation and Validation**

    - **Conducted extensive testing to compare the proposed system against Unity's default controller, demonstrating improvements in realism, stability, and user driving experience.**

In summary, this work contributes to a physics-based vehicle drivetrain model that enhances realism in Unity 3D while remaining efficient for real-time applications. It provides a foundation for both entertainment-focused driving games and educational or training simulators requiring more authentic vehicle behavior.

5.2. Method and Procedure

While the developed drivetrain and torque shifting system demonstrates significant improvements in realism, there remain opportunities for further enhancement and expansion:

1. **Advanced Tire Dynamics**
    - **Implementing more detailed tire models that account for heat buildup, wear, and deformation to improve long-duration driving accuracy.**
2. **Aerodynamic Effects**
    - **Adding features such as downforce, drag, and lift simulation to reflect the impact of aerodynamics at high speeds.**
3. **Hybrid and Electric Powertrains**
    - **Extending the drivetrain architecture to support electric vehicles, regenerative braking, and hybrid systems, which are increasingly relevant in modern automotive design.**
4. **Environmental Interactions**
    - **Introducing dynamic weather and road conditions (rain, snow, gravel, ice) to test vehicle adaptability and traction control under diverse environments.**
5. **AI Driver Integration**
    - **Incorporating artificial intelligence to simulate autonomous or computer-controlled vehicles for training and competitive scenarios.**
6. **Multiplayer and Networking**
    - **Expanding the system for online multiplayer simulations, allowing multiple users to interact within the same driving**

environment.

7. **Calibration with Real Vehicle Data**
    - **Validating and fine-tuning the model using empirical data from real cars, improving its accuracy for research and training applications.**

In conclusion, these enhancements would further increase the system's realism and adaptability, making it suitable not only for entertainment applications but also for professional driving simulators, research, and education.

Appendices

A. DriveTrain Code Listings

This appendix contains the complete source code for the Drivetrain developed in this thesis. While excerpts and explanations have been provided in earlier chapters, the full listings are included here for reference and reproducibility.

A.1 CarController Code

```
using System;
using UnityEngine;

public class CarController : MonoBehaviour
{
    private float horizontalInput, verticalInput;
    private float currentSteerAngle, currentBrakeForce;
    private bool isBraking;

    // Settings
    [SerializeField] private float motorForce = 1500f;     // Driving
force
    [SerializeField] private float brakeForce = 3000f;     // Braking
force
    [SerializeField] private float maxSteerAngle = 25f;    // Steering
limit

    private Rigidbody rb;
    [SerializeField] private Transform centerOfMassTransform;

    // Wheel Colliders
    [SerializeField] private WheelCollider frontLeftWheelCollider,
frontRightWheelCollider;
    [SerializeField] private WheelCollider rearLeftWheelCollider,
rearRightWheelCollider;

    // Wheel Meshes (for visuals)
    [SerializeField] private Transform frontLeftWheelTransform,
frontRightWheelTransform;
    [SerializeField] private Transform rearLeftWheelTransform,
rearRightWheelTransform;
```

```
    private void Start()
    {
        rb = GetComponent<Rigidbody>();

        // Adjust center of mass for stability
        if (centerOfMassTransform != null)
            rb.centerOfMass = centerOfMassTransform.localPosition;
        else
            rb.centerOfMass = new Vector3(0f, -0.5f, 0f);  // Slightly
lowered CoM

        // Tune wheel friction for stability
        SetupWheelFriction(frontLeftWheelCollider);
        SetupWheelFriction(frontRightWheelCollider);
        SetupWheelFriction(rearLeftWheelCollider);
        SetupWheelFriction(rearRightWheelCollider);
    }

    private void FixedUpdate()
    {
        GetInput();
        HandleMotor();
        HandleSteering();
        UpdateWheels();
    }

    // ---------------------------
    // Input Handling
    // ---------------------------
    private void GetInput()
    {
        horizontalInput = Input.GetAxis("Horizontal"); // A/D or
Left/Right
        verticalInput = Input.GetAxis("Vertical");     // W/S or
Up/Down
        isBraking = Input.GetKey(KeyCode.Space);       // Space for
brake
    }

    // ---------------------------
    // Motor & Braking
    // ---------------------------
```

```
    private void HandleMotor()
    {
        // Rear-wheel drive setup
        rearLeftWheelCollider.motorTorque = verticalInput * motorForce;
        rearRightWheelCollider.motorTorque = verticalInput *
motorForce;

        // No torque on front wheels
        frontLeftWheelCollider.motorTorque = 0f;
        frontRightWheelCollider.motorTorque = 0f;

        // Braking
        currentBrakeForce = isBraking ? brakeForce : 0f;
        ApplyBraking();
    }

    private void ApplyBraking()
    {
        frontLeftWheelCollider.brakeTorque = currentBrakeForce;
        frontRightWheelCollider.brakeTorque = currentBrakeForce;
        rearLeftWheelCollider.brakeTorque = currentBrakeForce;
        rearRightWheelCollider.brakeTorque = currentBrakeForce;
    }

    // ----------------------------
    // Steering
    // ----------------------------
    private void HandleSteering()
    {
        currentSteerAngle = maxSteerAngle * horizontalInput;
        frontLeftWheelCollider.steerAngle = currentSteerAngle;
        frontRightWheelCollider.steerAngle = currentSteerAngle;
    }

    // ----------------------------
    // Wheel Visual Updates
    // ----------------------------
    private void UpdateWheels()
    {
        UpdateSingleWheel(frontLeftWheelCollider,
frontLeftWheelTransform);
        UpdateSingleWheel(frontRightWheelCollider,
frontRightWheelTransform);
```

```
        UpdateSingleWheel(rearLeftWheelCollider,
rearLeftWheelTransform);
        UpdateSingleWheel(rearRightWheelCollider,
rearRightWheelTransform);
    }

    private void UpdateSingleWheel(WheelCollider wheelCollider,
Transform wheelTransform)
    {
        Vector3 pos;
        Quaternion rot;
        wheelCollider.GetWorldPose(out pos, out rot);
        wheelTransform.position = pos;
        wheelTransform.rotation = rot;
    }

    // ---------------------------
    // Wheel Friction Setup
    // ---------------------------
    private void SetupWheelFriction(WheelCollider wheel)
    {
        WheelFrictionCurve forwardFriction = wheel.forwardFriction;
        WheelFrictionCurve sidewaysFriction = wheel.sidewaysFriction;

        // Forward grip (acceleration/braking)
        forwardFriction.stiffness = 1.5f;

        // Sideways grip (turning)
        sidewaysFriction.stiffness = 2.5f;

        wheel.forwardFriction = forwardFriction;
        wheel.sidewaysFriction = sidewaysFriction;
    }
}
```

A.2 CarCameraFollow Code

```
using UnityEngine;

public class CarCameraFollow : MonoBehaviour
{
    [Header("Target to Follow")]
    public Transform target;

    [Header("Camera Settings")]
    public Vector3 offset = new Vector3(0f, 5f, -10f);
    public float followSpeed = 5f;
    public float rotationSpeed = 5f;

    private void LateUpdate()
    {
        if (target == null) return;

        Vector3 desiredPosition = target.position +
target.TransformDirection(offset);

        transform.position = Vector3.Lerp(transform.position,
desiredPosition, followSpeed * Time.deltaTime);

        Quaternion desiredRotation =
Quaternion.LookRotation(target.position - transform.position,
Vector3.up);
        transform.rotation = Quaternion.Slerp(transform.rotation,
desiredRotation, rotationSpeed * Time.deltaTime);
    }
}
```