Drift: Immersive

Immersive, VR-ready cockpit driving experience built on Unity 6000.2.2f1 with XR Interaction Toolkit and OpenXR, developed by Georgi Tsvetanski.

Project Overview: The Core Simulation

My goal was to craft a concise yet high-fidelity driving simulation that seamlessly integrates physics, VR, intuitive UI feedback, and engaging interaction design.

- First-person cockpit perspective for maximum immersion
- VR head tracking via XR Rig with decoupled camera rotation
- Rigidbody-based physics with force-driven acceleration and steering
- Interactive steering wheel with Animator IK hand synchronization
- Real-time speedometer with normalized speed calculation

Why This Project Matters: Objectives & Passions

Real-time Simulation Design

Deepening expertise in complex real-time simulation environments.

VR Camera & Input Mastery

Gaining proficiency in VR camera systems and advanced input handling.

Modular Unity Systems

Building robust, reusable, and scalable modular systems within Unity.

Physics & Interaction Scripting

Enhancing skills in physics and intricate interaction scripting.

Portfolio-Ready Content

Developing high-quality, immersive content suitable for professional portfolios.

This project was a synthesis of my core passions: high-performance vehicles, virtual reality, and immersive interaction.

Concept & Planning: Blueprint for Immersion

Concept Summary

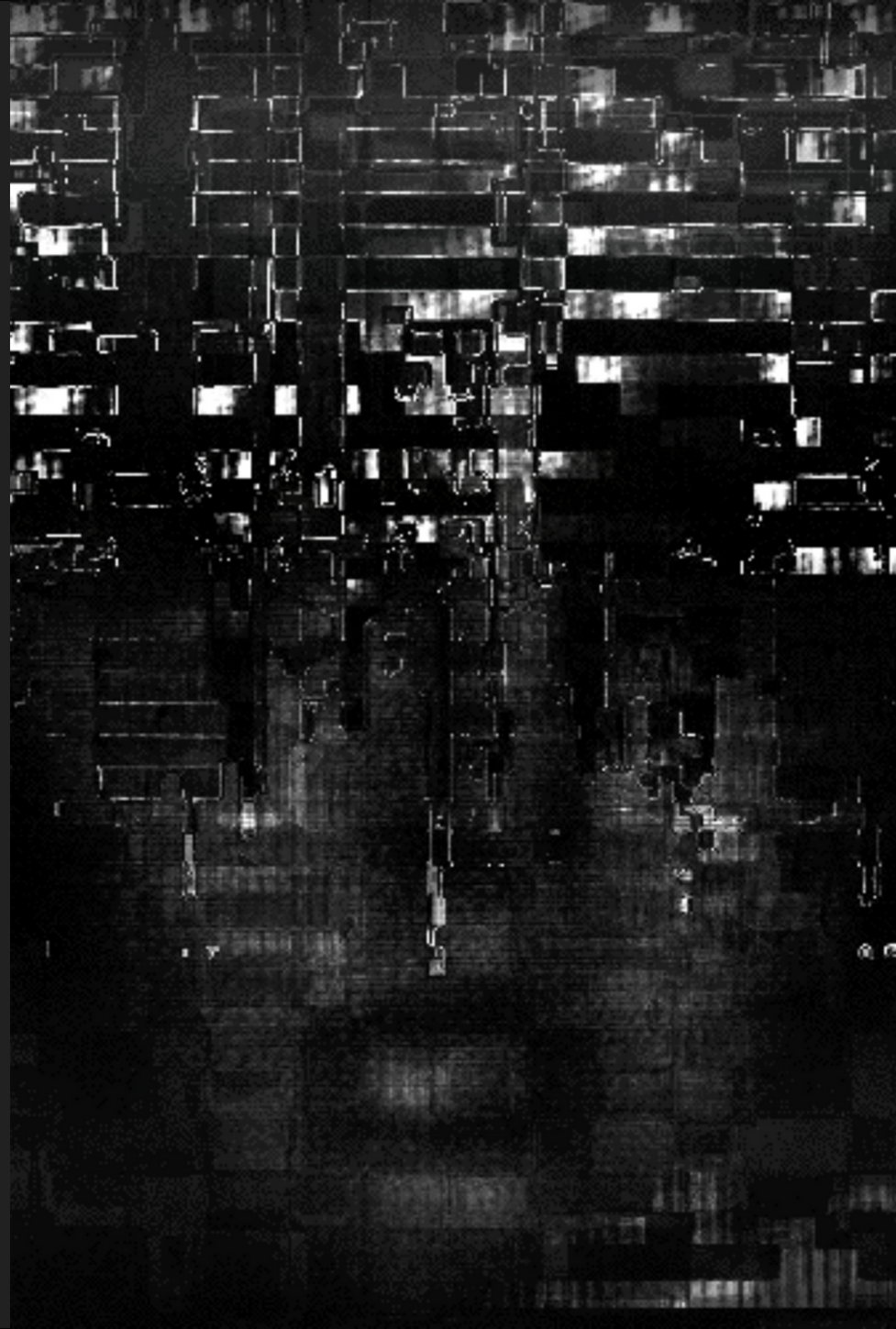
Gameplay Loop: Accelerate → Steer → Drift (optional) → Brake → Repeat with responsive feedback.

- Player is fully integrated into the car's cockpit environment.
- Core mechanics: acceleration, braking, precise steering, and dynamic drifting.
- Steering wheel and driver's hands require accurate, synchronized animation.
- Interior instruments must provide real-time, responsive feedback.

Planning Tools Utilized

- Comprehensive Game Design Document (GDD) as a graded deliverable
- Detailed Code Quality Report as a graded deliverable
- Structured weekly milestones
- Agile test-driven development (TDD) approach

My planning prioritized modularity: foundational movement, then nuanced drifting, intricate cockpit animation, responsive UI instruments, and finally, seamless VR integration. Modularity was essential for VR—decoupled systems allowed independent tuning of physics, camera, and input without cascading bugs.



Movement & Drift Systems: Crafting the Driving Feel

Realistic Car Movement

- Precisely tuned acceleration and deceleration curves.
- Responsive steering with advanced rotation smoothing for natural control.
- Rigidbody-based physics engine for authentic, real-world vehicle behavior.
- Rigidbody.AddForce() applies acceleration curves; steering uses Quaternion.Lerp for smooth rotation damping.

Dynamic Drift Behavior

- Dedicated "drift key" to momentarily reduce tire grip for controlled slides.
- Increased steering multiplier during drift to enhance slip angle and maneuverability.
- Smooth, gradual restoration of grip for seamless recovery from drifts.
- Drift key reduces drag coefficient and multiplies steering input by 1.5x; grip restoration uses Time.deltaTime for frame-independent recovery.

These intertwined systems were critical in defining the visceral driving sensation. Mastering the drift required meticulous calibration of traction, steering input, and the smooth transition back to stability.

Cockpit Systems: Bringing the Interior to Life

Initial Challenges

- Steering wheel disappeared due to local vs. world rotation inheritance—parent transform rotations were overriding local Euler angles.
- Driver's hands remaining static and misaligned with the wheel's rotation.

Innovative Solutions

- Completely rebuilt wheel rotation logic using only local Euler angles, eliminating inheritance issues.
- Implemented Inverse Kinematics (IK) targets directly on the steering wheel.
- Animator IK targets were parented directly to the steering wheel; hand bones follow via Animator.SetIKPosition() and Animator.SetIKRotation(), ensuring wrist alignment matches wheel rotation in real-time.

These enhancements made the cockpit feel truly alive. The wheel rotates with precision, and the driver’s hands follow its motion realistically via the IK system, boosting immersion.

VR Camera & Interaction: Optimizing for Immersion

Camera Jitter Resolution

Eliminated distracting camera jitter that occurred during car rotations by processing camera updates in LateUpdate(). The XR camera must not inherit vehicle rotation directly; instead, camera updates occur in LateUpdate() after physics, with rotation applied only to the car body, not the head rig.

Ghosting & Sensitivity Tuning

Addressed ghosting issues when steering with a controller and significantly reduced VR sensitivity for comfort. Controller input is offset in XR camera space rather than world space, preventing rotation lag.

Smoothed Transitions

Implemented smooth positional and rotational transitions to prevent motion sickness and enhance comfort.

FOV & Clipping Adjustments

Fine-tuned Field of View (FOV) and clipping planes for an optimal and comfortable VR viewing experience.

VR integration demanded the most meticulous tuning. Decoupling the car's rotation from the head tracking proved crucial, resolving visual jitter and significantly elevating the sense of presence.

Challenges vs Solutions: Technical Debugging

Problem

- Steering wheel disappearing during rotation
- Speedometer needle maxing out at idle
- VR camera jitter during vehicle turns

Diagnosis

- Local vs. world rotation inheritance conflict
- Speed value not normalized to gauge range
- Camera inheriting vehicle rotation directly

Solution

- Rebuilt rotation using local Euler angles only; parented IK targets to wheel
- Implemented normalizedSpeed = currentSpeed / maxSpeed formula with clamping
- Decoupled camera from vehicle; applied rotation in XR camera space via LateUpdate()

Interior Instruments: The Responsive Speedometer

Original Issue Identification

The speedometer needle erroneously maxed out, even when the vehicle was stationary, indicating an inaccurate display.

Root Cause Analysis

The raw speed value was not being correctly normalized to the speedometer's rotational range, leading to incorrect readings.

Final Implementation

- Calculated normalizedSpeed = currentSpeed / maxSpeed to map velocity to gauge range (0–1).
- Needle rotation = normalizedSpeed × maxRotation (e.g., 270°), then clamped to [0°, 270°].
- Smoothing via Vector3.Lerp reduces jitter for realistic needle motion.

This seemingly small detail profoundly enhanced immersion, transforming the cockpit from a static environment into a fully functional and believable space.

Final Results: A Showcase of Completed Features

Fully Functional Cockpit

Experience driving from a meticulously rendered, interactive cockpit perspective.

Steering Wheel & IK Hand Animation

Enjoy fluid, realistic hand animations synchronized with the steering wheel's movement.

Drift Mechanics & Speed Feedback

Engage dynamic drift mechanics complemented by accurate, real-time speed feedback.

VR-Ready Camera System

A fully optimized camera system designed for comfortable and immersive virtual reality experiences.

Reflection & Conclusion: Key Learnings

VR Update Loops

VR demands distinct update loop management compared to desktop environments.

IK System Nuances

Inverse Kinematics systems, though sensitive, offer immense power for realistic animation.

Modular Scripting

Clean, modular scripts are invaluable, saving countless hours in debugging.

Iterative Physics Tuning

Achieving optimal physics behavior is an iterative process of continuous refinement.

Documentation Benefits

Thorough documentation significantly boosts development speed and clarity.

XR Constraints & Offset Transforms

Understanding unscaled time, camera rig hierarchies, and transform offsets proved critical for stable VR.

Debugging Large Prefabs

IK rigs and complex hierarchies required systematic isolation and frame-by-frame analysis to identify inheritance and animation conflicts.

This project profoundly deepened my understanding of VR, advanced interaction design, and robust simulation architecture within Unity 6000.2.2f1. Debugging large prefabs with IK rigs taught me the importance of systematic isolation and frame-by-frame analysis. It elegantly synthesized all the key concepts explored this semester.