F1 Simulation: Mixed Reality Experience

Atit Kharel

Department of Computer Science,
University of Arkansas at Little Rock

Abstract

This report details the development of an immersive Formula 1 (F1) race simulation using Unity Engine and data from the OpenF1 API. The project leverages mixed reality (MR) technology to provide a unique, interactive experience, where users can observe and interact with F1 races through Head Mounted Displays (HMDs) and Cave Automatic Virtual Reality (CAVE) Setup. By utilizing data on tracks, cars, drivers, and race events, the simulation dynamically represents past races. The project also integrates user interactions through hand tracking and raycasting, enabling intuitive control of the simulation. This report discusses work flow of the simulation, a critial analysis of the current implementation, along with potential improvements and enhancements, to further enhance the simulation's realism and user experience. The project highlights the potential of merging real-world data with MR technology to create interactive simulations. These systems can revolutionize fields like sports analytics, education, and training by providing enhanced visualization and real-time analysis tools.

1 Introduction

Formula 1 is renowned for its cutting-edge technology and data-driven strategies, making it an ideal domain for simulation projects. F1 Simulation project leverages these attributes to create an immersive Mixed Reality (MR) experience that mirrors the complexity of real-world F1 races. The development was done for HMDs like the Oculus Quest 3 [1] and a CAVE Setup [2]. The Unity Engine was chosen as the platform for development due to its support for multiple headsets and its robust 3D rendering capabilities.

2 Technologies

2.1 Unity

Unity was used as the main development environment, providing the necessary tools for building 3D environments and integrating virtual objects into a mixed-reality experience. The engine's C# scripting capabilities were used to implement the simulation's logic and interactions.

2.2 OpenF1 API

The OpenF1 API [3] is an open-source platform that provides real-time or historical data on Formula 1 race events, drivers, teams, and tracks. These data were used to populate the simulation with accurate information to visualize the drivers, cars, and track.

2.3 Meta All-In-One XR Plugin

The Meta All-In-One XR is a plugin provided by Meta that enables developers to create immersive experiences in both virtual reality (VR) and mixed reality (MR). The plugin provides features like hand tracking and spatial mapping, which were integral to this project.

2.4 Oculus Quest 3

The Oculus Quest 3 is a standalone VR headset that provides a wireless and untethered experience. The device's capabilities in hand tracking and spatial mapping were utilized to test the Simulations, improving the development process.

2.5 CAVE

A Cave Automatic Virtual Environment (CAVE) is an immersive virtual reality system designed as a room-sized cube with projection screens on multiple walls, the floor, and occasionally the ceiling. The CAVE system utilized in this project features multiple synchronized displays on three walls and two projectors dedicated to floor projections. It supports stereoscopic 3D visualization, enabling a fully immersive experience when viewed with 3D glasses. Interaction and user tracking are facilitated through HTC Vive controllers and trackers, ensuring seamless engagement with the virtual environment.

3 Project Components

3.1 Data Collection and Processing

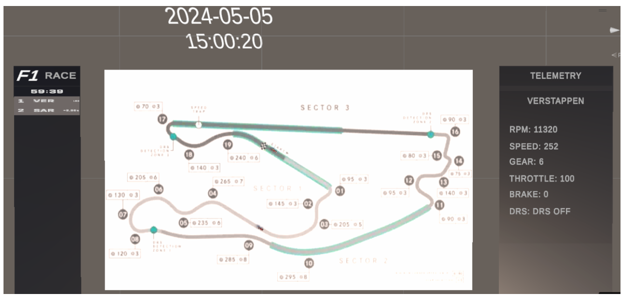
The project communicates with multiple endpoints of the OpenF1 API to fetch data on tracks, driver details, car telemetry, and their location on the track using coroutine in Unity. The fetched data is processed (stored and sorted ascendingly by date/time) for each data entry to ensure the simulation runs in chronological order. The data is then used to populate the simulation by spawning new car for each driver and updating their position and telemetry.

To update the data for simulation, a virtual countdown timer is used to simulate the data updates. We take the time difference between the current time and the time of the next data entry to update the simulation and interpolate the car position to avoid sudden jumps for inconsistent data.

Each driver can have around 22,000 data entries for each category depending on track size, with each data entry having their own timestamp. The data is fetched and processed in real-time to simulate which makes it usable even in live race scenarios not limiting to historical race simulations.

3.2 Environment and UI

The simulation starts with a 3d plane representing the track, with 3D models of cars on the track. Each car is labeled with the driver's acronym for easy identification. The user interface (UI) [Figure 1] has 3 main components: a plane with track's image, a leaderboard on the left (with position, name and interval from leader), and a driver telemetry panel on the right (with speed, gear, RPM, and throttle/brake input). The virtual clock for the simulation is displayed at the top of the UI, showing the current time in the simulation.



RPM: 11320
SPEED: 252
GEAR: 6
THROTTLE: 100
BRAKE: 0
DRS: DRS OFF

Figure 1: *Main UI of the F1 Simulation*

For controlling the simulation, the user can use a sim control UI [Figure 2], toggled by pressing 'X' button on the left controller. The Control UI allows the user to start, pause, reset, increse or decrease the simulation speed. Both controller and hand tracking can be used to interact with the UI.

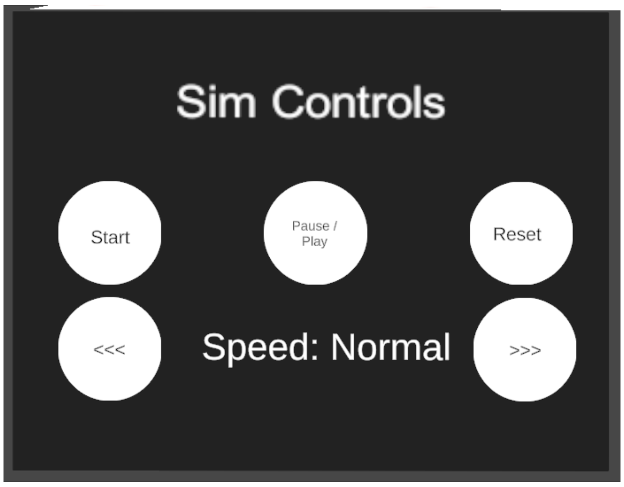


Figure 2: *Simulation Controller UI*

3.3 Car Simulation

To update the car's position in simulation, the system uses a method that receives a driver's number and a list of track data entries (containing time, position and telemetry data). The method then uses binary search to find the correct interval of data points that surround the current simulation time. These two points, referred to as `prevData` or $\mathbf{p}_{\text{prev}}$ and `nextData` or $\mathbf{p}_{\text{next}}$, are the closest time-stamped positions before and after the current time. If valid previous and next points are found, the car's position and rotation is updated by interpolating between these two data points. This results in smooth movement and rotation of the car between the two data points.

Position Interpolation Given two data points, the previous position $\mathbf{p}_{\text{prev}} = (x_{\text{prev}}, y_{\text{prev}})$ and the next position $\mathbf{p}_{\text{next}} = (x_{\text{next}}, y_{\text{next}})$, the interpolation factor t is calculated as:

$$t = \frac{\Delta t_{\text{simulated}}}{\Delta t_{\text{interval}}} = \frac{T_{\text{simulated}} - T_{\text{prev}}}{T_{\text{next}} - T_{\text{prev}}}$$

where $T_{\text{simulated}}$ is the current simulation time, and T_{prev} and T_{next} are the timestamps of the previous and next data points, respectively.

The interpolated position $\mathbf{p}(t)$ is given by:

$$\mathbf{p}(t) = (1-t)\mathbf{p}_{\text{prev}} + t\mathbf{p}_{\text{next}}$$

Expanding into components:

$$x(t) = (1-t)x_{\text{prev}} + tx_{\text{next}}, \quad y(t) = (1-t)y_{\text{prev}} + ty_{\text{next}}$$

Rotation Interpolation The car's angle θ between the previous and next positions is calculated as:

$$\theta = \arctan\left(\frac{\Delta x}{\Delta y}\right) = \arctan\left(\frac{x_{\text{next}} - x_{\text{prev}}}{y_{\text{next}} - y_{\text{prev}}}\right)$$

Note: Although elevation data (z-coordinate) is also available in the OpenF1 API, it was omitted in this implementation to simplify calculations and focus on core positional and rotational accuracy.

Implementation The interpolation is achieved in Unity by using the `Vector3.Lerp` function for position & `Mathf.Atan2` and `Quaternion.Slerp` for angle and rotation. [Listing 1] shows the C# code that implements the described method.

```
    private void InterpolatePosition(int
driverNumber, TrackData prevData,
TrackData nextData)
    {
        float timeBetween = (float)(DateTime.
Parse(nextData.date) - DateTime.Parse(
prevData.date)).TotalSeconds;
        float timeSincePrev = (float)(
simulatedTime - DateTime.Parse(prevData.
date)).TotalSeconds;
        float t = Mathf.Clamp01(timeSincePrev
 / timeBetween);

        // Interpolate position
        Vector3 prevPosition = new Vector3(
prevData.x, 0, prevData.y);
        Vector3 nextPosition = new Vector3(
nextData.x, 0, nextData.y);
        Vector3 interpolatedPosition =
Vector3.Lerp(prevPosition, nextPosition,
t);
        carModels[driverNumber].transform.
localPosition = interpolatedPosition;

        // Interpolate rotation
        float angle = Mathf.Atan2(
nextPosition.x - prevPosition.x,
nextPosition.z - prevPosition.z) * Mathf.
Rad2Deg;
        carModels[driverNumber].transform.
localRotation = Quaternion.Slerp(
carModels[driverNumber].transform.
localRotation, Quaternion.Euler(0, angle,
 0), t);
    }
```

Listing 1: *Interpolation of Car Position and Rotation*

3.4 Leaderboard and Telemetry

The leaderboard on the left is updated based on the data's timestamp, with the leaderboard showing the driver's position, name, and interval from the leader. The telemetry panel on the right displays the driver's speed, gear, RPM, and throttle/brake input which is updated similarly to the car's position as both the data is from the same data entry.

3.5 Interactions

For the sim control UI, the user needs controller input to toggle the UI and interact with the buttons. Other than that, the simulation supports both controller and hand tracking for interactions. The user is able to grab the main track UI and move or rotate it using the controller or hand tracking. Users can use two hands to grab the track and scale it to zoom in or out.

4 CAVE Integration

After the HMD setup, the integration of this project into a CAVE system allowed for testing and adaptation of the project for various environments and devices. This also allowed for a more realistic and engaging simulation experience. This effort involved refining user interfaces, improving interactions, and using unique visualization techniques to ensure a functional and immersive experience within the CAVE setup.

4.1 UI/UX

The project was progressively adapted to suit the unique characteristics of the CAVE system. The User Interface and Interactions were optimized and positioned for ease of use and spatial clarity utilizing all walls of the CAVE.

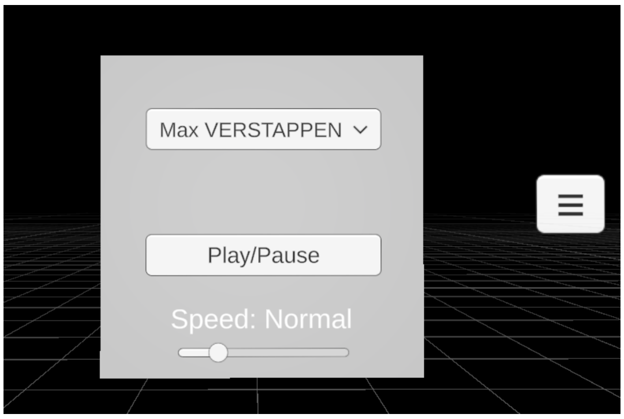
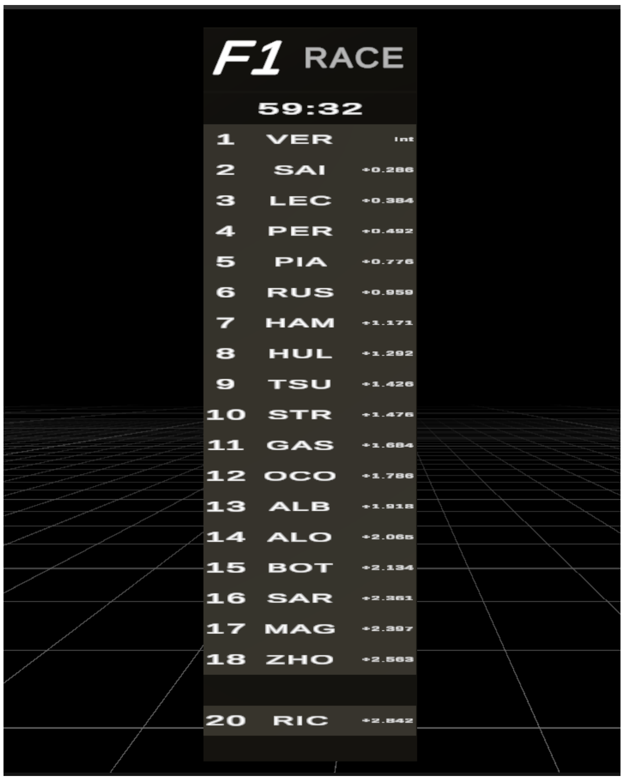


Figure 3: *Controls Menu on the right wall of the CAVE with toggle button.*

Right Wall: A **controls menu** [Fig. 3] is placed on the right wall, allowing users to start, pause, play, or change the speed of the simulation. Additional controls include a driver selection drop-down and a toggle button for showing or hiding the menu.

Left Wall: A live-updating **race leaderboard** [Fig. 4] is displayed on the left wall which also shows real-time gaps between the drivers.


F1 RACE

59:32

1	VER	Int
2	SAI	+0.286
3	LEC	+0.384
4	PER	+0.492
5	PIA	+0.776
6	RUS	+0.959
7	HAM	+1.171
8	HUL	+1.292
9	TSU	+1.426
10	STR	+1.476
11	GAS	+1.684
12	OCO	+1.786
13	ALB	+1.918
14	ALO	+2.065
15	BOT	+2.134
16	SAR	+2.361
17	MAG	+2.397
18	ZHO	+2.563
20	RIC	+2.842

Figure 4: *Leaderboard on the left wall of the CAVE.*

The spatial distribution of controls across walls improves accessibility and reduces clutter. For example, positioning the leaderboard on the left wall ensures the user can track rankings without interfering with driver selection on the right wall. Initial user tests indicated that spatially distributed controls made interaction more intuitive, with users particularly appreciating the ability to toggle menus on demand.

4.2 Focused Driver View

Another UI enhancement includes a *per-driver view* to focus on the car of the selected driver and a *driver details panel* positioned above the focused car always facing the user.

Per-Driver View: [Fig. 5] Selecting a driver from the *controls menu* displays their car beside the main track map, with animated wheels and real-time rotation.

Driver Details Panel: [Fig. 5] This panel includes essential driver details such as name, team, driver number, and headshot. Additionally, telemetry data like speed, RPM, throttle press percentage, brake press percentage, and more is displayed dynamically.

Custom Shader: A custom Unity shader was implemented to create a spotlight effect for the focused driver view. This shader occludes the surrounding

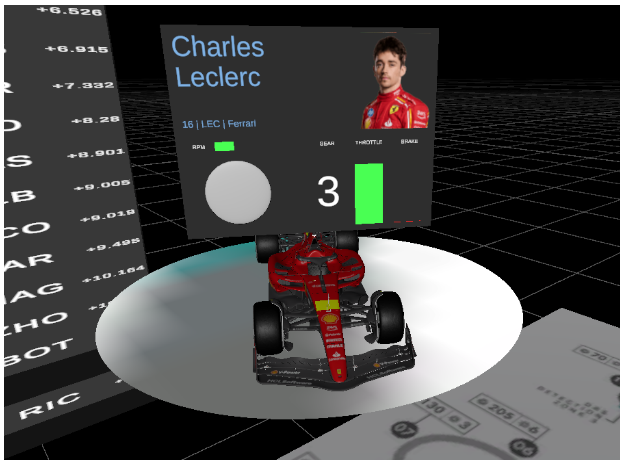


Figure 5: *Focused Driver View with a driver details panel above.*

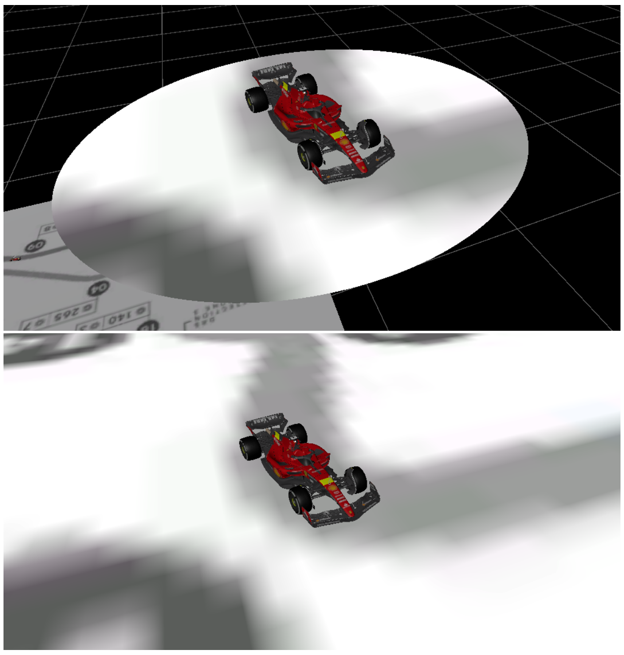


Figure 6: *Focused view comparison demonstrating the shader's spotlight effect on the track.*

map area while highlighting the car and the spherical area of the track around the car. The car remains stationary in the scene, while the track moves around it, simulating its high-speed motion. The car's speed data is used to ensure synchronized wheel rotation and track movement preserving the perception of the car's speed. Figure [6] illustrates the contrast between the focused view with and without the shader applied.

5 Critical Analysis

5.1 Data Precision

The simulation currently uses only two coordinates for interpolation, which can lead to inaccuracies in the car's position and rotation. For instance, during sharp corners, the current two-coordinate interpolation occasionally produces abrupt transitions, breaking immersion. Additionally, the data from the API is already not precise enough and is scaled, further contributing to these inaccuracies. To improve precision, the simulation can be enhanced to include elevation data and use more advanced interpolation techniques like Catmull-Rom splines or Bezier curves to accurately represent movements like overtaking, cornering, and braking.

5.2 Interaction Enhancements

The simulation's current interaction model is confusing and lacks intuitiveness. It can be enhanced by adding more features like gesture recognition, voice commands, or gaze-based interactions. These features can provide a more intuitive and immersive experience for the user, allowing them to control the simulation more naturally [4].

5.3 Enhanced Track Map Visualization

The current simulation utilizes a flat 2D texture of the track map applied to a plane surface. Upgrading this to a detailed 3D model of the track could greatly enhance the visual fidelity and overall immersion of the simulation. A 3D track model would allow users to perceive elevation changes, corners, and track features more realistically, creating a more engaging environment. Additionally, incorporating a 3D track would complement the *focused driver view*, making it more dynamic and immersive by visually integrating the car's movements with the terrain and track layout.

5.4 Environment Awareness

The simulation's environment awareness can be improved by integrating spatial mapping. This can help the simulation adapt to the user's physical environment, providing a more realistic and immersive experience. Spatial mapping can also be used to create more interactive elements in the simulation, such as placing the track in the user's physical space like a tabletop, floor or wall.

5.5 Performance Optimization

The simulation first loads all data and then processes it, which can lead to performance issues with large datasets. To optimize performance, the simulation can be modified to load and process data in chunks or on-demand, reducing the initial load time and memory usage.

5.6 Local Data Storage

The simulation currently fetches data from multiple OpenF1 API endpoints, which can lead to network latency and data synchronization issues. To address this, we can setup a local database to store the data and update it periodically to ensure the simulation runs smoothly and efficiently without relying on external APIs.

6 Improvements and Future Work

The F1 Simulation project has the potential for further improvements and enhancements to provide a more engaging and immersive experience for users. Some of the key areas for improvement include:

6.1 Multiplayer Support

Multiplayer support could enable collaborative analysis or competitive scenarios, requiring network synchronization to ensure seamless interactions across devices.

6.2 Data Visualization

The simulation can be enhanced to include advanced data visualization features, such as speed plots, lap time plots, and telemetry graphs. These visualizations can provide deeper insights for fans, teams, and analysts. FastF1 [5] is a good example of a tool similar to OpenF1 that provides such visualizations. It offers data in the form of extended Pandas DataFrames and integrates with Matplotlib for easy data visualization. Figures [7, 8, 9] show examples of speed, lap time and gear shifts plots for drivers taken from FastF1.

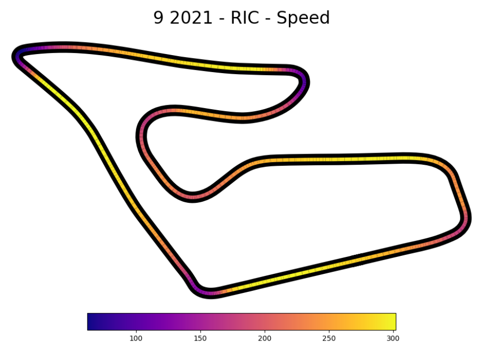


Figure 7: *Speed Plot of a Driver on Track*

6.3 Weather and Track Conditions

Taking weather data from the OpenF1 API, the simulation could be enhanced to include dynamic weather and track conditions, such as rain, fog, or changing

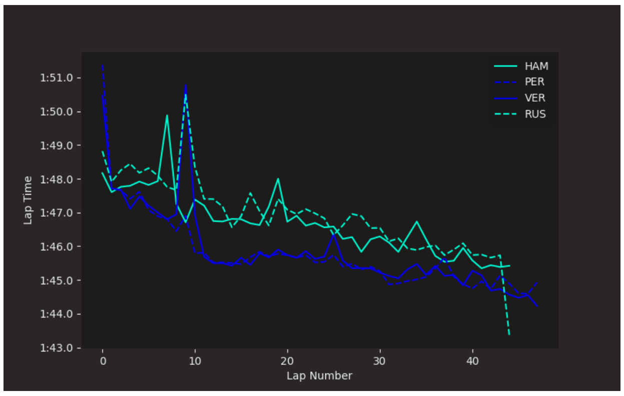


Figure 8: *Lap Time Plot of Four Drivers*

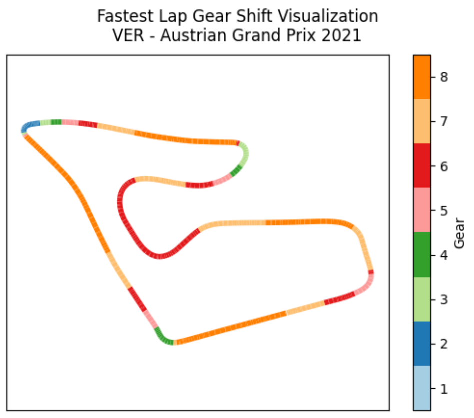


Figure 9: *Gear Shift Visualization for the Fastest Lap of a Driver*

track temperatures. This can add an extra layer of realism to the simulation.

7 Conclusion

The integration of real-world data with MR technology showcases the feasibility of creating interactive digital twins, opening avenues for realistic simulations in sports, training, and beyond. This approach could benefit industries such as automotive testing, disaster simulation training, and educational modules where real-world data enhances immersion and interactivity.

References

[1] Oculus. *Oculus Quest 3*. Meta. URL: `https://www.oculus.com/quest-3/`.

[2] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. "Surround-screen projection-based virtual reality: the design and implementation of the CAVE". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: Association for Computing Machinery, 1993, pp. 135–142. ISBN: 0897916018. DOI: 10.1145/166117.166134. URL: `https://doi.org/10.1145/166117.166134`.

[3] Bruno Godefroy. *OpenF1 API | Real-time and historical Formula 1 data*. OpenF1. URL: `https://openf1.org/`.

[4] Connor Daniel Flick et al. "Trade-offs in Augmented Reality User Interfaces for Controlling a Smart Environment". In: *Proceedings of the 2021 ACM Symposium on Spatial User Interaction*. SUI '21. Virtual Event, USA: Association for Computing Machinery, 2021. ISBN: 9781450390910. DOI: 10.1145/3485279.3485288. URL: `https://doi.org/10.1145/3485279.3485288`.

[5] theOehrly. *FastF1 Documentation*. URL: `https://docs.fastf1.dev/`.