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 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). 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 showcases the potential of integrating real-world data with MR technology to create engaging and immersive experiences for users, which can also be leveraged to enhance data visualization and analysis across various fields.

1 Introduction

The objective of the F1 Simulation project is to integrate real-world data from the OpenF1 API [1] into a mixed reality (MR) environment on HMDs like the Oculus Quest 3 [2]. The aim is to simulate a real-time Formula 1 race, allowing users to interact with the race virtually and view it in an MR setting. 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 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 offers features such as hand tracking, spatial mapping, hand tracking, which were utilized in the project.

2.4 Oculus Quest 3

The Oculus Quest 3 is a standalone VR headset that provides a wireless and untethered experience. The device was used to test the MR simulation, using the device's hand tracking and spatial mapping capabilities to enhance the development process.

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 accendingly 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 senarios 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 also has a name tag with the driver's acronym for 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.

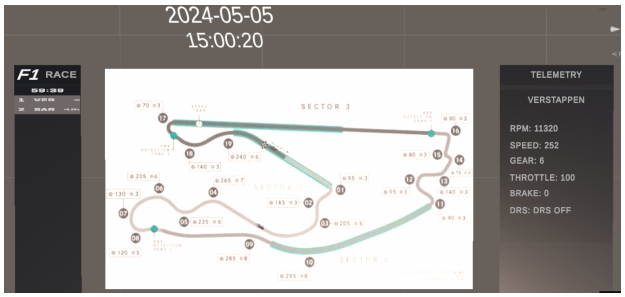

Figure 1: *Main UI of the F1 Simulation*

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: only two coordinates are used for interpolation as the car's elevation data is not considered in this project, but it is provided in the OpenF1 API data.

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.

```
1    private void InterpolatePosition(int
     driverNumber, TrackData prevData,
     TrackData nextData)
2    {
3        float timeBetween = (float)(DateTime.
     Parse(nextData.date) - DateTime.Parse(
     prevData.date)).TotalSeconds;
4        float timeSincePrev = (float)(
     simulatedTime - DateTime.Parse(prevData.
     date)).TotalSeconds;
5        float t = Mathf.Clamp01(timeSincePrev
      / timeBetween);
6
7        // Interpolate position
8        Vector3 prevPosition = new Vector3(
     prevData.x, 0, prevData.y);
```

```
9         Vector3 nextPosition = new Vector3(
  nextData.x, 0, nextData.y);
10        Vector3 interpolatedPosition =
  Vector3.Lerp(prevPosition, nextPosition,
  t);
11        carModels[driverNumber].transform.
  localPosition = interpolatedPosition;
12
13        // Interpolate rotation
14        float angle = Mathf.Atan2(
  nextPosition.x - prevPosition.x,
  nextPosition.z - prevPosition.z) * Mathf.
  Rad2Deg;
15        carModels[driverNumber].transform.
  localRotation = Quaternion.Slerp(
  carModels[driverNumber].transform.
  localRotation, Quaternion.Euler(0, angle,
   0), t);
16    }
```

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 Critical Analysis

4.1 Data Precision

The simulation currently uses only two coordinates for interpolation, which can lead to inaccuracies in the car's position and rotation. 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.

4.2 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 [3]

4.3 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.

4.4 Performance Optimization

The simulation currently loads all the data first 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.

4.5 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.

5 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:

5.1 Multiplayer Support

Adding multiplayer support to the simulation can enhance the user experience by allowing multiple users to interact with the simulation simultaneously.

5.2 Real-time Data Updates

The simulation can be enhanced to support real-time data updates, allowing users to view live Formula 1 races not limited to historical races.

5.3 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 [4] 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 [3, 4, 5] show examples of speed, lap time and gear shifts plots for drivers taken from FastF1.

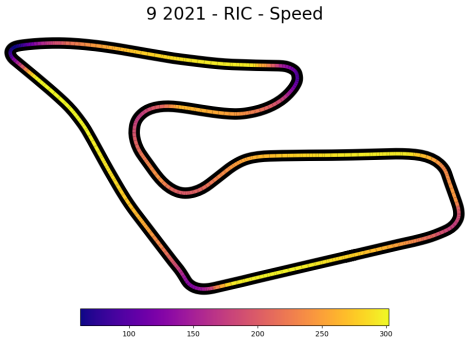

Figure 3: *Speed Plot of a Driver on Track*

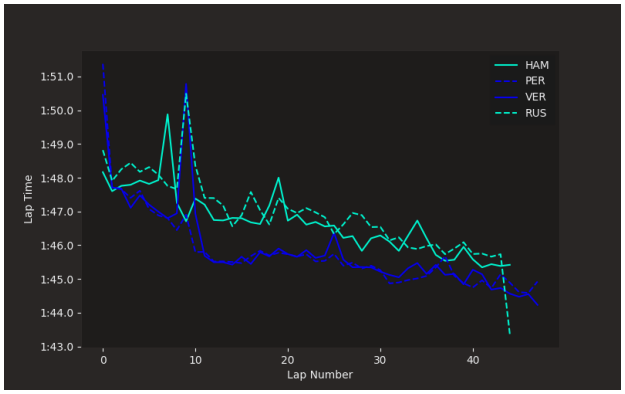

Figure 4: *Lap Time Plot of Four Drivers*

5.4 Weather and Track Conditions

Taking weather data from the OpenF1 API, the simulation can be enhanced to include dynamic weather and track conditions, such as rain, fog, or changing track temperatures. This can add an extra layer of realism and challenge to the simulation.

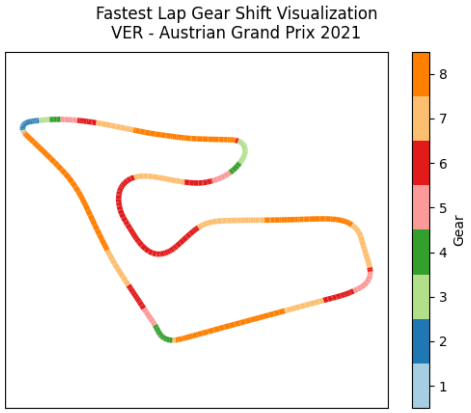

Figure 5: *Gear Shift Visualization for the Fastest Lap of a Driver*

6 Conclusion

The F1 Simulation project demonstrates the potential of integrating real-world data with mixed reality technology to create immersive and interactive experiences for users. By leveraging the OpenF1 API and Unity engine, the project simulates Formula 1 races introducing the concept of a digital twin, where the virtual simulation mirrors real-world race events and conditions. By utilizing this concept, the project not only provides an engaging user experience but also opens up possibilities for advanced data visualization and analysis, making it a potentially valuable tool for both entertainment and professional applications.

References

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

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

[3] 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.

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