BSGLg, 85, 2025, 1-23 *Accepté le 2 février 2026*

INTERPOLATED REAL-TIME MOBILITY DATA IMPLEMENTATION IN DIGITAL TWINS

MISE EN ŒUVRE DE DONNÉES DE MOBILITÉ INTERPOLÉES EN TEMPS RÉEL DANS DES JUMEAUX NUMÉRIQUES

Susheel NATH, Valentin MARCHAND, Carl MÖRCH, Martin CANTER

Abstract
Digital Twins (DTs) are transforming how urban mobility is visualized, analyzed, and managed, yet current systems struggle to deliver real-time, smooth 3D representations due to sparse and irregular data from multiple providers. This study introduces a novel architecture for real-time Digital Twin visualization of multi-modal public transport by integrating heterogeneous data sources, including GTFS-RT feeds (De Lijn) and GeoJSON-based streams (STIB, SNCB). We address key technical challenges in animating incomplete mobility data by implementing a SLERP-based interpolation pipeline that ensures consistent and visually smooth motion. A Brussels-Capital Region case study demonstrates reduced dependency on update frequency, improved rendering smoothness, accurate geospatial conversion via Cesium, and readiness for scalable web deployment. Limitations and future directions, such as browser-based implementations with CesiumJS and Three.js, are also discussed.

Keywords
digital twin, interpolation, real-time data, public transport, GTFS-RT, GeoJSON, SLERP, simulation

Résumé
Les Digital Twins (DTs) transforment la manière dont les systèmes de mobilité urbaine sont visualisés, analysés et gérés. Cependant, les solutions actuelles peinent à offrir des représentations 3D fluides et en temps réel en raison de flux de données clairsemés et irréguliers provenant de multiples opérateurs. Cette étude présente une architecture innovante pour la visualisation en temps réel de transports publics multimodaux, intégrant des sources hétérogènes telles que les flux GTFS-RT (De Lijn) et les données GeoJSON (STIB, SNCB). Nous proposons un pipeline d'interpolation basé sur SLERP afin d'animer ces données de manière cohérente. Une étude de cas dans la Région de Bruxelles-Capitale démontre une meilleure fluidité visuelle, une moindre dépendance aux mises à jour, une conversion géospatiale précise via Cesium et une préparation pour un déploiement web évolutif. Les limites et perspectives, notamment via CesiumJS et Three.js, sont également discutées.

Mots-clés
jumeau numérique, interpolation, données en temps réel, transports publics, GTFS-RT, GeoJSON, SLERP, simulation

INTRODUCTION

Urban mobility has grown increasingly complex with the rise of multimodal transportation networks, growing urban populations, and increasing demands for real-time tracking, prediction, and optimization of transport services. Traditional traffic management systems often struggle to cope with the scale, variability, and interdependence of modern transportation networks. One promising solution to address these challenges is the use of Digital Twins.

A Digital Twin is a virtual representation of a physical system that is continuously updated with real-time data from its physical counterpart, enabling monitoring, simulation, analysis, and decision-making over time. It aids in improving, processing, and managing information at a physical and virtual level, and helps create improved control, optimization, and autonomy capabilities (Iliuţă *et al*., 2024; Clark *et al*., 2025).

Digital Twins have emerged as a powerful paradigm for modeling, simulating, and optimizing urban mobility systems by enabling continuous synchronization between physical vehicle transportation networks and their digital counterparts. These twins offer decision-makers, innovators, researchers, and citizens a shared

platform for monitoring infrastructure, forecasting disruptions, and enhancing decision-making through data-driven insights (Datta, 2016).

However, while static Digital Twins based on historic, non-real-time datasets are increasingly being deployed in city-scale infrastructure projects for traffic mobility understandings, the integration of dynamic, real-time data feeds, particularly from standardized sources like GTFS-RT (General Transit Feed Specification – Real-Time) and GeoJSON (Geographic JavaScript Object Notation) into interactive 3D environments remains a nascent and technically underdeveloped area (Kušić *et al*., 2022). This gap stems from several compounding challenges:

Firstly, GTFS-RT data is inherently sparse and inconsistent. GTFS-RT updates are often infrequent, and the level of detail and reliability varies significantly across transit public institutions and regions. This leads to difficulties in interpolating smooth and realistic vehicle movement trajectories and requires intelligent data handling strategies to fill gaps and minimize abrupt transitions in Digital Twin environments (Webb *et al*., 2025).

Secondly, GeoJSON feeds, while more structured, present their own challenges. GeoJSON is widely used for representing geographical features due to its human-readable format and compatibility with web services. However, when applied to real-time Digital Twin pipelines, several difficulties arise: parsing large, nested JSON structures introduces overhead; coordinate systems must often be transformed between WGS84 and local projections required for high-fidelity simulation; and metadata extraction is not standardized across providers, leading to heterogeneity in data quality (Butler *et al*., 2016).

Finally, real-time integration into high-fidelity 3D engines like Unity or Unreal, which presents a flexible means to integrate diverse datasets, introduces performance and architectural constraints. These platforms, while powerful, were not originally designed for large-scale, continuously updating, data-driven simulations. They often face issues such as rendering overhead, threading limitations, and resource intensive scene management, especially when handling hundreds or thousands of moving entities like vehicles (Crampen *et al*., 2024). Unity WebGL further amplifies these problems with its constrained memory model, lack of multithreading, and limitations in asynchronous data streaming.

To address these issues, we propose a pipeline for ingesting, parsing, interpolating, and visualizing heterogeneous data in urban mobility Digital Twins. The pipeline addresses the challenges inherent in GTFS-RT feeds, including irregular update intervals, inconsistent metadata across providers, and bandwidth constraints. It also accommodates both GTFS-RT streams (e.g., DeLijn) and GeoJSON feeds (e.g.,STIB and SNCB).

This work contributes to three areas:

A modular and multi-source architecture designed to efficiently ingest, process and harmonize heterogeneous real-time feeds from multiple transport agencies, addressing challenges such as inconsistent metadata, sparse updates, and variable data quality.

A visualization platform based on Unity that provides interactive 3D geospatial rendering of transit networks, enabling users to visualize vehicle locations and movements in real time within a geospatially accurate digital environment through precise coordinate transformations.

An extendable browser-compatible deployment that leverages web–based compatibility to deliver interactive, real-time visualizations directly in the browser, eliminating the need for platform-specific installations or plugins while maintaining performance and accessibility.

The objective behind this research is to address operational challenges when working with Urban Digital Twins and end-users. Whereas Digital Twin practitioners and domain experts are used to dealing with less interpretable representations, 3D visualizations are more accessible to non-experts, especially when dealing with complex city environments and multi-domain data. Achieving an efficient and smooth 3D visualization of GTFS-RT data through this novel approach will allow Digital Twins to have a larger reach for the decision makers and multiple stakeholders of this complex environment.

The paper is organized as follows: Section 2 provides a comprehensive review of the current state-of-the-art in the field, highlighting recent advances, ongoing challenges, and relevant approaches in the context of real-time urban mobility and Digital Twins. Section 3 outlines the proposed methodology, describing the overall approach, objectives, and the rationale behind key design decisions. In Section 4, the paper delves into the system architecture, implementation details, and the interpolation techniques employed, illustrating how real-time transit data is processed, visualized, and integrated into a 3D Digital Twin environment. Section 5 addresses the limitations of the current approach and explores potential directions for future research. Finally, Section 6 concludes the paper by summarizing the key findings, contributions, and implications.

I. STATE OF THE ART

Digital Twins (DTs) have emerged as transformative tools across various sectors, from architecture, engineering, and construction (AEC) to healthcare, manufacturing, and increasingly, urban mobility. Their strength lies in their ability to create virtual replicas of physical systems that are continuously synchronized with real-world data, enabling decision-makers, innovators, researchers and citizens alike to monitor, simulate, and optimize complex environments (Singh *et al*., 2022).

In the context of transportation, Digital Twins are being increasingly leveraged for traffic flow optimization, real-time monitoring, infrastructure planning and maintenance, public transit efficiency, and predictive modeling. Despite these advances, the application of Digital Twins in real-time, multimodal transit simulation, particularly those synchronized with live, city-scale datasets, remains fragmented, highlighting significant opportunities for further research and development (Irfan *et al*., 2024).

A. General Transit Feed Specification

The General Transit Feed Specification (GTFS) is a widely adopted standard that provides static transit data, such as stop locations, scheduled routes, and trip sequences. General Transit Feed Specification Real-Time (GTFS-RT), its real-time extension, delivers updates on vehicle positions, trip progress, movement delays, and arrival estimates. While valuable, GTFS-RT feeds often suffer from sparse, inconsistent intervals, and missing metadata, especially when sourced from multiple transit operators or public agencies (Newmark, 2024).

In addition, some agencies provide their mobility data via GeoJSON feeds. These datasets offer structured geometries and metadata but integrating them into 3D Digital Twins requires careful parsing, synchronization, and coordinating transformation to align with high-fidelity simulation environments.

In this context, static GTFS data serves as a static baseline, forming the backbone of scheduled operations. GTFS-RT streams then provide real-time observations of actual vehicle locations and their pertaining metadata. Together, they form a continuum that ensures both temporal continuity and analytical depth in Digital Twin environments.

Core challenges, however, are the integration, synchronization, and visualization of real-time GTFS-RT and GeoJSON data into dynamic, interactive 3D geospatial environments. While Digital Twins are conceptually equipped to mirror real-world transit systems, the irregular nature of real-time updates of GTFS-RT, due to latency, data quality issues, or limited infrastructure support, makes visual continuity and coherence difficult to achieve (Wong, 2025).

To address these gaps, interpolation and predictive modeling have become essential. Linear Interpolation (LERP) offers a simple and computationally efficient way to estimate positions between timestamps, but its linear assumptions often produces visually abrupt or mechanically unrealistic transitions when applied to curved paths, turning movements, or vehicle rotations. Spherical Linear Interpolation (SLERP), although originally defined for rotations on a sphere, provides smoother, constant-velocity interpolation on curved trajectories and therefore yields more natural, continuous motion for vehicle animation at the city scale. SLERP in retrospect produces more stable and realistic interpolated motion compared to LERP, especially for orientation changes and non-linear trajectories, hence offering an effective midpoint between simplicity and visual realism (Pei *et al*., 2019).

Beyond these geometric approaches, alternative models such as polynomial interpolation and spline-based methods can provide smoother trajectories but require denser data and exhibit instability with noisy inputs. More recently, data-driven predictors, including recurrent Neural Networks (NNs) and Long Short-Term Memory (LSTMs) have been used to model complex mobility patterns, handle missing data, and capture non-linear temporal dependencies in transit operations (Wang *et al*., 2024).

While these machine-learning approaches improve longer-term forecasting accuracy, they come with higher computational costs and require large number of historical datasets that are not always available. Considering the tradeoff between computational performance and data availability, SLERP offers a favourable balance, providing smoother and more geospatially consistent interpolations than simpler methods like LERP while remaining far less resource-intensive than machine-learning based predictors.

Further, for noisy data streams, Kalman Filters offer statistically optimal estimations of position and velocity, reducing jitter, and increasing confidence in motion trajectories. They predict the next state, update it with new measurements, and provide accurate estimates of position and velocity. This process reduces noise and inaccuracies, making motion trajectories more reliable (Pei *et al*., 2019).

While Kalman Filters are excellent for real-time smoothing and short-term trajectory estimation, they are limited for long-term prediction. Their assumptions of linear motion and Gaussian noise do not hold over extended periods in urban transit systems, where traffic, delays, and driver behaviour are highly variable. As a result, prediction errors grow quickly, making the filter unreliable beyond short horizons, leading to loss of accuracy as prediction intervals increase (Qiao *et al*., 2018).

Lastly, the utilization of OGC API conventions for Moving Features (OGC Publications, 2025) presents an emerging pathway for real-time mobility integration within Digital Twins and is highly compatible with interpolation pipelines. Future research may also incorporate machine learning-based predictors and hybrid Kalman neural network approaches to handle larger anomalies, missing data, or predict paths in the absence of sufficient data (Wang *et al*., 2025; Chib, 2024).

B. Visualization Challenges

After ingesting and harmonizing heterogeneous transit data, effective visualization becomes critical for easy decision-making, simulation, and stakeholder engagement. Native engines like Unity and Unreal Engine bring rendering power, interactivity, and usability; however, they often come at the cost of high computational demands, requiring dedicated GPUs, large memory allocations, and platform-specific builds. This limits the scalability and accessibility of Digital Twin applications, especially in public sector use cases, where deployments must span across devices and institutions with stakeholders of varying technical capabilities (Crampen *et al*., 2024).

Cesium, an open-source 3D geospatial engine, bridges the gap between GIS datasets and immersive environments. Through its Cesium for Unity SDK, developers can stream 3D terrain tilesets, BIM models, and high resolution imagery directly into Unity, integrating them with real-time simulation logic and interactive tools. Additionally, cross deployment framework CesiumJS, provides a lightweight, browser-based option capable of running on desktop and handheld devices, making it suitable for interoperable web-based 3D geospatial visualizations.

Although browser-native platforms offer high accessibility, Unity combined with Cesium was selected due to its superior support for high-fidelity simulation, physics, and real-time interaction, which are essential capabilities for Digital Twin prototyping multi-modal transit animation, and interpolation pipelines. Unity provides a mature ecosystem for custom tooling, C# based data ingestion, advanced rendering, and integration with GTFS-RT and GeoJSON workflows, which are difficult to achieve with browser-only technologies. Cesium for Unity further retains web interoperability through shared 3D Tiles and CesiumJS compatibility while enabling GPU accelerated visualization (Wurstle *et al*., 2022), allowing for web-based deployment that is easily accessible to users without performant hardware dependencies.

Together, these tools reduce the gap in feasibility, accuracy, interoperability, and performance optimization. Nevertheless, further work is needed to make Digital Twin systems intuitive, accessible, and optimally performant across a multitude of devices (Bernstetter *et al*., 2025; Keil *et al*., 2021). Bounded by native technical pipelines, alternatives are required to achieve true cross platform interoperability.

Despite existing advancements in Digital Twin development, challenges remain in harmonizing data quality, optimizing rendering performance, and ensuring consistent cross platform deployment (Bernstetter *et al*., 2025). This work addresses these gaps by combining multi-format data ingestion (GTFS-RT and GeoJSON), intelligent interpolation (SLERP), and advanced visualization within a unified architecture.

C. Data Standardization Challenges

Despite significant progress in geospatial technologies, the GIS community still lacks fully unified and widely adopted standards for representing, transmitting, and managing trajectory data. Existing mobility datasets, whether provided through GTFS-RT, proprietary APIs, or GeoJSON feeds, differ widely in structure, semantics, temporal granularity, and coordinate reference conventions. Such fragmentation complicates interoperability, limits reusability, and creates substantial challenges for integrating heterogeneous real-time feeds into Digital Twin environments (David *et al*., 2024).

In response to these interoperability inconsistencies, several OGC standards aim to harmonize geospatial representation, such as CityGML, which offers a structured semantic and geometric model of urban environments across Levels of Detail (LOD0–LOD4). While CityGML primarily addresses static 3D city models, its emphasis on hierarchical structure, semantic richness, and multi-resolution representation provides a strong foundation for interoperable urban data management. Emerging standards like OGC API – Moving Features and MF-JSON further extend this by defining consistent, browser-friendly formats for representing and streaming dynamic trajectories (OGC Publications, 2025).

Within this context, the proposed study aligns with ongoing standardization efforts by normalizing heterogeneous real-time feeds into a unified GeoJSON and CityGML based structure, and integrating interpolation methods that are directly compatible with OGC API Moving Features-compliant representations.

II. METHODOLOGY

This study introduces a modular, extensible, and performant pipeline for the ingestion, interpolation, and geospatial visualization of real-time public transport data using GTFS, GTFS-RT, and GeoJSON. The approach integrates standardized transit datasets with 3D rendering technologies, combining the geospatial capacities of Cesium's 3D tilesets, the real-time animation capabilities of Unity, and interpolation techniques optimized for sparse temporal datasets. The pipeline is designed for scalability, fault tolerance, and compliance with data governance policies, with a primary case study in the Brussels-Capital Region. It addresses challenges such as network latency, data sparsity, and multi-agency integration while ensuring smooth, visually coherent, accurate real-time visualizations for urban and intercity transit systems.

A. Data Integration Strategy

A central challenge is the integration of heterogeneous mobility datasets originating from multiple public transport agencies. In the Brussels case study, public transport operators provide data in different formats. Specifically, De Lijn exposes both static GTFS and GTFS-Realtime feeds, while STIB/MIVB, SNCB/NMBS and Infrabel primarily rely on GeoJSON based APIs for delivering static and dynamic information (Table 1) provides a summary of the agencies and their respective data formats.

It is worth noting that SNCB additionally provides GTFS-RT feeds; however, these do not include direct GPS positions, as estimated locations are reconstructed by the provider. Such differences in data granularity, frequency, and structure introduce challenges for Digital Twin implementation, requiring careful interpolation, coordinate transformation, and harmonization strategies to generate consistent, real-time representations of transit operations.

Agency	Static data	Real-time data	Formats used
DeLijn	GTFS	GTFS-RT	CSV (GTFS), Protobuf (RT)
STIB/MIVB	GeoJSON	GeoJSON	GeoJSON (stops, routes, vehicles)
SNCB/NMBS	GeoJSON	GeoJSON/ GTFS-RT	GeoJSON (vehicles)
Infrabel	GeoJSON	/	GeoJSON (segments, operational points)

Table 1. Summary of data formats for transportation agencies to study

The integration of GTFS and GTFS-RT datasets forms the backbone of the pipeline, merging static schedules with dynamic real-time updates to produce a unified view of transit operations. Static GTFS provides comprehensive data on stop locations, route geometries, trip schedules (departure and arrival times), and associated metadata. GTFS-RT complements this with live operational information, including vehicle positions (GPS coordinates with timestamps), trip progress (active trip segments, next stops, delays), and metadata such as vehicle identifiers and types. Synchronization between static and dynamic feeds is achieved through unique identifiers (trip IDs, route IDs, and vehicle IDs), ensuring consistent mapping across update cycles even when real-time feeds arrive late, contain partial information, or momentarily desync from scheduled data.

In contrast, GeoJSON feeds expose network segments, stops and real-time vehicles states directly as geographic features. To harmonize these heterogeneous sources, all feeds are unified into a structured representation compatible with Unity's data model. In practice, the system adopts GeoJSON as the canonical intermediate format, due to its geospatial structure and widespread interoperability. Unity does not define a proprietary JSON standard. The serialization language used by Unity is based on a specification from YAML. Instead, incoming GeoJSON files are deserialized into strongly typed C# classes using built-in *JsonUtility* and the *Json.NET* library for more complex structures. Each Feature is converted into a C# object following a unified schema which include the *geometry* and the *properties*. This schema harmonizes GeoJSON based feeds, GTFS-RT and static GTFS by encapsulating all resources into a consistent format. It helps with ingestion, interpolation and rendering. The resulting objects constitute the internal exchange format used throughout the Unity pipeline, ensuring consistency despite differences between agencies.

To optimize performance, the static GTFS datasets from DeLijn are cached locally. Since these static datasets do not change substantially over time (infrastructure information such as roads, railway lines, etc.), it only necessitates update on a monthly cycle which they are fetched on the subsequent access of the platform in a month's schedule, ensuring that cached data remains consistent with official updates. The GeoJSONs from STIB, Infrabel, and SNCB are called directly and modified if changes are detected on the supplier side.

A local check is performed to validate cached data. To maintain up-to-date information, a lightweight HTTPS request architecture has been designed to revalidate the cache monthly against the operator's API. In cases of structural changes, such as stop relocations due to roadworks, the system can force a cache update, ensuring that stop positions, route geometries, and related schedule adjustments remain consistent between planned and real-time feeds.

For GTFS-RT, data is fetched every 20 seconds and parsed via protocol buffer (protobuf) feeds (DeLijn), while GeoJSON (STIB, SNCB, Infrabel) are requested at similar intervals to maintain temporal consistency. Custom parsers handle deserialization and conversion into structured JSON objects compatible with Unity's data pipeline.

To handle discrepancies, such as missing trip IDs or inconsistent update frequencies, the system employs a reconciliation algorithm that uses fallback identifiers such as vehicle ID that is linked uniquely to the metadata. This ensures robust alignment even for agencies with varying data quality. Multi-agency data integration is supported through parallel ingestion pipelines,

each configured with agency-specific endpoints and schema mappings. For example, the system simultaneously processes STIB/MIVB's urban bus, metro and tram data alongside SNCB/NMBS's and DeLijn's intercity train and bus data respectively, maintaining separation of responsibilities.

B. System Architecture

The system architecture is organized into four independent layers: Ingestion, Processing, Visualization, and Interaction. Each layer is designed for modularity, scalability, and resilience, allowing the system to operate efficiently even under high vehicle counts or network constraints. The architecture incorporates data caching, asynchronous updates, error handling, and load balancing to minimize latency and ensure smooth, responsive performance in real-time scenarios.

1. Ingestion Layer

The Ingestion Layer is responsible for retrieving, parsing, and pre-processing heterogeneous data feeds from multiple public transport operators. It supports simultaneous connections to endpoints from STIB, SNCB, Infrabel, and De Lijn, ensuring that diverse urban mobility data can be ingested in real time. GTFS-RT streams are parsed from Protocol Buffer (protobuf) messages into structured JSON objects, while GeoJSON mobility feeds are ingested and converted into JSON in a similar format, enabling uniform downstream processing. Once ingested, all data is compartmentalized within Unity, where they are mapped to corresponding vehicles and stop metadata.

Additional intrinsics, such as interpolated positions and predicted paths, are computed downstream to enhance the fidelity of the Digital Twin. To manage asynchronous data streams and maintain reliability, the ingestion layer employs a rolling buffer, queuing updates for vehicles collectively to mitigate the effects of packet loss, network delays, or irregular update intervals. Furthermore, the layer monitors feed health and performs lightweight validation checks, flagging anomalies or missing data, which allows downstream processing to handle inconsistencies gracefully and ensures continuous, robust operation of the real-time system.

2. Processing Layer

Once ingested, the data transitions into the Processing Layer, where it undergoes interpolation, filtering, and validation to refine vehicle trajectories and ensure spatial consistency. A unified spherical linear interpolation (SLERP) pipeline is employed to preserve geospatial accuracy along great-circle paths, using quaternion mathematics to reduce cumulative trajectory errors. Unlike traditional linear interpolation, SLERP maintains smooth and realistic vehicle movements across curved paths, minimizing distortions that can arise over long distances or irregular routes. For static datasets, such as those provided by Infrabel representing railway line geometries, interpolation is not required, as these datasets primarily support infrastructure visualization rather than dynamic vehicle tracking.

In cases where positional data is sparse, delayed, or erratic, a Kalman filter is applied to enhance spatial continuity. This two-phase (predict–update) model uses vehicle speed, heading, and previous position estimates to smooth noisy measurements and predict interim locations, resulting in more coherent trajectories in real time. Additionally, the Processing Layer performs data harmonization, aligning timestamps, normalizing coordinate systems, and reconciling discrepancies across multiple feeds. This ensures that vehicles from different agencies reporting at different frequencies or with varying metadata completeness, are consistently represented within the Digital Twin.

As a result, these techniques allow the system to deliver high-fidelity, temporally and spatially coherent representations of urban mobility, forming a reliable foundation for downstream visualization and interaction.

3. Visualization Layer

The refined trajectories are then handed to the Visualization Layer, which renders them onto a geospatial canvas constructed in Unity using the Cesium for Unity SDK. High-fidelity 3D tilesets of the Brussels-Capital Region, streamed at *Level of Detail 2.2 (LOD2.2)* following CityGML conventions, provide accurate representations of terrain, buildings, and other infrastructure, enabling a realistic and immersive urban environment. The LOD2.2 CityGML

3D representation of buildings is derived from Paradigm's data repository, Datastore.Brussels (Ruhl, 2024). Paradigm is the Brussels-Capital region's central partner for digital transformation, responsible for managing and coordinating information-tech infrastructure, services, data, and ICT governance across regional and local authorities. In this case study's consideration, they are responsible for the accurate 3D data store of the infrastructure of Brussels-Capital Region.

Vehicles are instantiated as dynamic 3D *Prefabs* and animated along their interpolated trajectories, with positions updated in real time according to Unity's update cycle. This synchronization ensures smooth transitions, consistent playback at 60 frames per second, and visual coherence even under high vehicle density.

To enhance realism, additional visual elements such as route paths, stop markers, and metadata overlays are integrated, allowing users to contextualize vehicle movement within the broader transport network. The Visualization Layer also supports adaptive level-of-detail (LOD) management, dynamically adjusting rendering complexity based on camera distance and system performance to maintain responsiveness. Moreover, lighting, shadows, and material effects are leveraged to improve depth perception and spatial awareness, which is critical for both operational monitoring and scenario simulation. These features ensure that the Digital Twin not only reflects accurate real-time mobility data but also provides an intuitive, visually coherent, and interactive environment for users to analyze and explore urban transit dynamics.

4. Interaction Layer

Finally, the Interaction Layer governs all user-facing functionality, providing tools for exploration, analysis, and real-time monitoring of the Digital Twin. Built using Unity's UI Toolkit, it enables users to explore transit data by agency, vehicle type, or route, while also inspecting real-time metadata such as delays, travel durations, and vehicle status. Interactive dashboards and overlays allow stakeholders to visualize performance, track individual vehicles, or analyze aggregated traffic patterns across the city.

Camera navigation tools, including zoom, pan, orbit, and third-person perspectives, support intuitive exploration of the 3D environment, ensuring that users can examine both macro-scale traffic flows and micro-scale operational details. Contextual interactions, such as selecting a vehicle or stop, reveal additional information through pop-ups or linked panels, facilitating in-depth investigation and scenario analysis. The system transforms raw transit data into a dynamic, analytical, and user-centric interface, enabling planners, operators, and citizens alike to explore, interpret, and act on complex urban mobility information.

C. Synchronization and Animation

Synchronization between data updates and Unity's rendering loop is essential to achieve smooth real-time animations. Since real-world data arrives at unfixed intervals, the system cannot simply update vehicle positions directly at those intervals, as this would result in jerky or stepwise movements. To address this, Unity's *FixedUpdate()* function is employed. Unlike the normal *Update()* function that is tied to the device performance frame rate, *FixedUpdate()* runs at steady, consistent intervals, making it a reliable place to apply motion logic. This ensures that vehicle interpolation is calculated evenly over time, independent of rendering fluctuations, and prevents visual inconsistencies when frame rates vary.

To further refine movement, the system uses *AnimationCurve* keyframes to describe how vehicles transition between known data points. These curves act as mathematical timelines that define changes in position, rotation, and velocity across a specified duration. For example, instead of a vehicle instantly jumping from one location to the next, the AnimationCurve gradually moves it along the path, with smooth easing that can represent acceleration, deceleration, or curved trajectories. In practice, this allows Unity to fill in the gaps between real-time updates with continuous, natural motion.

Each vehicle is initialized (instantiated) as a Unity *GameObject Prefab*, which is essentially a container for all relevant data, visual and auditory components comprising that said element. The GameObject container also includes a 3D model

representing the vehicle type respectively (bus, tram, or train), along with attached scripts that manage its position, rotation, and movement behaviors. A dedicated interpolation buffer component is attached to each GameObject to store recent updates, enabling smooth transitions when new data is applied. Additionally, metadata components within the GameObject track properties such as vehicle ID, route, heading, and service status.

To optimize performance, further necessary due to the substantial number of vehicles and its comprising data to be represented in real-time, vehicle GameObjects are managed using an *object pooling system*. Instead of instantiating and destroying GameObjects each time a vehicle enters or leaves the scene, the object pool maintains a preallocated collection of reusable vehicle objects that is rather activated/enabled or deactivated/disabled respectively. When a new vehicle needs to be visualized, an inactive GameObject is retrieved from the pool, updated with the relevant model, metadata, and interpolation buffer, and activated. Once the vehicle is no longer visible or active, the GameObject is returned to the pool rather than being destroyed, minimizing costly memory allocations and garbage collection overhead. This approach ensures smooth runtime performance even when visualizing large fleets of buses, trams, and trains simultaneously, allowing the system to scale efficiently without dropping frames or introducing performance lags.

D. Geospatial Context Implementations and Optimizations

1. Level of Detail

The visualization layer leverages Cesium for Unity SDK to display 3D tilesets, maintaining geographic accuracy while benefiting from Unity's graphics and physics capabilities. Level of Detail (LOD) is employed to balance visual fidelity and performance. It is a technique that adjusts the geometric complexity of 3D models based on the viewer's distance or relevance, ensuring that nearby objects appear highly detailed while distant objects are simplified. For this study, we utilize LOD2.2 models, which capture essential urban features such as building heights, roof structures, and basic textures, providing a realistic yet computationally efficient representation of the city. In general, higher LODs, with intricate architectural details and textures, are reserved for close-up inspection, while lower LODs are used for faraway views, enabling smooth streaming and rendering of the large urban environment in real time. For this case study, we use LOD 2.2 representations of terrain and urban structures of the Brussels-Capital Region.

2. Geospatial coordinate conversion

A critical step in this integration is the conversion of geospatial coordinates (latitude, longitude, and altitude in WGS84) into Unity's local coordinate system, ensuring that real-world positions are faithfully represented within the 3D environment. This process relies on Cesium's georeferencing module, which transforms coordinates from WGS84 to Earth-Centered Earth-Fixed (ECEF) and subsequently into Unity's coordinate system. Such a two-step transformation is essential because Unity's Cartesian coordinate system cannot natively interpret geodetic coordinates without distortion.

To guarantee alignment between dynamic entities, such as vehicles, and the high-resolution 3D tilesets, all positional data undergoes this WGS84-to-ECEF conversion before being interpolated. This step not only preserves geodetic accuracy across large spatial scales but also ensures numerical stability during real-time rendering. The system avoids common pitfalls such as floating-point precision errors that emerge when rendering trajectories over wide areas, or visual drift when interpolated paths are computed directly in latitude–longitude space. Furthermore, integrating the conversion into the interpolation pipeline allows smoother transitions along great-circle paths, ensuring that vehicle movement remains realistic, even across long intercity routes.

3. Camera Culling

To further optimize performance, Cesium's runtime engine applies to camera-based culling in conjunction with level-of-detail management. Rather than rendering the entire urban dataset at once, the engine continuously evaluates the camera's position, orientation, and field of view to determine which tiles are relevant for display.

Tiles outside the visible frustum are excluded from rendering, while occluding elements (e.g., buildings hidden behind others) are deprioritized. This selective visibility ensures that GPU and memory resources are concentrated on the parts of the scene that directly contribute to the user's perspective.

The system enables efficient streaming of large-scale 3D environments, allowing navigation across the Brussels-Capital Region without overwhelming computational resources. This approach not only reduces overhead but also ensures smooth frame rates, even when handling dense urban geometries or transitioning rapidly between viewpoints. In practice, this means that users can pan, zoom, or orbit through the environment interactively, with Cesium dynamically managing which elements of the digital twin are prioritized for rendering in real time.

4. GPU Instancing

Vehicle animations are optimized using *GPU instancing*, a technique where identical meshes are rendered in a single draw call rather than being processed individually by the CPU. This greatly reduces the number of draw calls and minimizes CPU–GPU communication overhead, which is often the main performance bottleneck in large-scale simulations. Each vehicle instance carries its unique properties and metadata, allowing unique attributes such as position, orientation, speed, and even material properties to be applied per individual instance. As a result, thousands of unique vehicles can be animated simultaneously, with the GPU efficiently handling geometry replication while Unity's update cycle applies movement logic, ensuring both high rendering performance and visual diversity.

5. Terrain Collision Physics

Cesium's terrain collision system ensures that vehicles remain accurately anchored to the 3D environment by continuously sampling the underlying terrain and infrastructure height data. This means, for example, buses automatically follow road gradients, trains stay aligned with railway embankments, and trams remain fixed within their track corridors. The system dynamically adjusts vehicle altitude in real time, preventing common artifacts such as floating above the surface or sinking into the ground mesh. Through this tight coupling between vehicle positions and the digital terrain model of Brussels-Capital Region, vehicle movements appear both natural and geospatially precise, reflecting real elevation changes across roads, tracks, and urban landscapes.

E. Real-Time Data Handling and Network Resilience

1. Data Reliability

To ensure reliability despite network delays or inconsistent updates, the ingestion layer cross validates vehicle timestamps and trip progress. This ensures that only fresh and consistent data is applied to the Unity scene, maintaining smooth vehicle animations and accurate positioning. In cases where data is missing or corrupted, fallback heuristics estimate vehicle positions based on historical motion, preventing abrupt jumps or freezes in the visualization. These mechanisms are critical for maintaining continuous, real-time visual feedback even when transit agency feeds vary in quality or structure.

2. Data Cross-Agency Harmonization

Transit data comes from multiple agencies in different formats and update frequencies. For example, De Lijn provides GTFS-RT with GPS positions, while STIB/MIVB, SNCB/NMBS, and Infrabel use GeoJSON feeds, and SNCB's GTFS-RT lacks direct GPS. The system standardizes all feeds into a single JSON format, matching vehicle and trip IDs, filling missing metadata, and merging updates to maintain consistent vehicle paths.

To keep everything aligned, updates are buffered and timestamped. When data is missing or delayed, fallback rules estimate vehicle positions to prevent visual jumps. Coordinates are converted from WGS84 to Unity's local reference frame using Cesium's georeferencing, ensuring vehicles, stops, and infrastructure line up correctly. This creates a reliable, unified dataset for smooth real-time visualization.

3. Asynchronous Data Ingestion within Unity

Real-time feeds are fetched over low-latency HTTPS or HTTP (where secure) from transit

agency endpoints. Unity's core rendering system operates on a single-threaded loop, meaning all visual updates like vehicle animations, physics calculations, and user interactions happen sequentially on one main thread. If network operations were also to be performed directly on this thread, any delay in fetching or processing data could cause the entire simulation to freeze or stutter. To prevent this, all network operations are handled asynchronously using coroutines or task-based concurrency models, allowing the system to request and receive data in the background without blocking the main thread. Meanwhile, the main rendering loop continues to update animations and visuals smoothly. Once new feed data is available, it is safely passed back to the main thread, validated, and applied to the vehicle GameObjects. This separation of data ingestion from visual updates ensures that even if network conditions fluctuate such as delayed responses or intermittent outages, vehicle animations remain fluid and continuous.

F. Data Compliance, Security, and Minimal Resource Consumption

A foundational principle of the system is ensuring responsible and minimal consumption of publicly provided transit data in line with institutional constraints and data governance policies. GTFS-RT and GeoJSON endpoints exposed by public agencies are often rate-limited and intended for lightweight consumer applications. To prevent overloading these services, the ingestion layer employs fetching all vehicles from a given agency in a single request rather than pinging each vehicle individually. For example, STIB, SNCB, DeLijn, and Infrabel are each queried separately, resulting in four bulk ingestions that cover all vehicles for the respective provider. These aggregated datasets are then parsed within Unity, where they are converted into classes that individualize the data and metadata for each vehicle.

Cached static data is revalidated periodically rather than redundantly fetched, further reducing bandwidth usage. From a compliance perspective, all data interactions are performed over secure HTTPS channels and HTTP (where secure), and no personally identifiable information (PII) is stored or processed. The system adheres to the terms of use specified by each agency's open data policy and avoids persistent logging of sensitive metadata such as vehicle identifiers or routing heuristics beyond their session lifespan. Agency-specific APIs with additional authorization requirements (such as tokenized endpoints or API keys) are not exposed or hardcoded. This approach guarantees operational sustainability, institutional trust, and full alignment with EU GDPR and regional open data compliance standards.

G. Modular and Scalable System Design

A major methodological goal is the creation of scalable and extensible architecture. The system supports the integration of additional transportation agencies or cities with minimal reconfiguration. New GTFS, GTFS-RT, or GeoJSON sources can be incorporated by following open, standardized formats, and the ingestion and processing layers can parse and integrate these feeds with minimal manual intervention.

In addition to data scalability, the visualization layer is decoupled from specific rendering models or tile formats. This flexibility makes the platform suitable not only for transit visualization but also for broader digital twin applications, such as pedestrian simulation, emergency evacuation planning, or multimodal traffic integration.

Finally, the animation system is modular, allowing different motion models to be plugged in depending on transport mode. For example, a boat or ferry could follow wave-based motion models, while autonomous vehicle paths could use LiDAR-derived Lane data. This modularity ensures realistic simulation across diverse transport types without rewriting core animation logic.

III. IMPLEMENTATION

For this use case, Unity was selected as the primary development platform, leveraging the Cesium for Unity SDK to enable high-fidelity visualization of 3D geospatial terrain using tilesets sourced from Cesium Ion. This integration provides a foundation for rendering the Brussels-Capital Region's urban landscape, including LOD2.2 building geometries and terrain elevation data. The Cesium for Unity SDK facilitates real-time streaming of 3D tilesets, optimized with a tile cache to reduce network

latency and ensure smooth rendering at 60 FPS. The system supports dynamic LOD management, streaming high-resolution tiles near the camera and lower-resolution tiles (LOD1) at distances exceeding 1 km, reducing draw calls by 40% in dense urban areas. In addition, performance optimization mechanisms were integrated. Frustum culling is employed to avoid buildings and terrain falling outside the main camera view. Furthermore, the *CesiumGeoreference* component allows precise synchronization between Unity's local coordinate and global geospatial references. This mitigates floating point accuracy errors that typically occur when rendering large geographic datasets.

To represent public mobility in the Brussels-Capital Region, we integrated heterogeneous data feeds from four major public transport providers: STIB/MIVB (operating metro, tram, and bus services within Brussels-Capital Region), SNCB/NMBS (managing intercity and regional train services), Infrabel (providing railway infrastructure data, including track geometries and signal states), and De Lijn (handling bus and tram services across Flanders, with overlapping operations in Brussels-Capital Region). Importantly, these agencies expose their data through different formats: De Lijn provides GTFS and GTFS-RT streams, while STIB and SNCB publish GeoJSON-based feeds. Infrabel datasets, in contrast, are primarily static and used for structural railway representation.

These feeds deliver real-time vehicle positions (latitude, longitude, timestamp), trip updates (delays, current stop, next stop, travel time), and metadata (vehicle type, vehicle ID, route ID).

A normalization pipeline was implemented to harmonize these heterogeneous inputs. Agency specific identifiers are mapped to a unified schema using standardized JSON objects to ensure compatibility with Unity's data pipeline.

Static datasets are cross-referenced to validate real-time updates and achieve data alignment accuracy across providers.
The project infrastructure (Figure 1) follows a modular design. A dedicated script manager module system was developed in Unity to orchestrate the data ingestion and update pipeline for each different data provider. It follows similar architecture principles to that of MobilityTwin (Merten, Sakr, 2023, 2023), utilizing a *collector/harvestor* to obtain the data from different providers, and an *extractor/handler* to validate and structure the data into homogenous GeoJSON. The granular process is as follows:

Each provider has a dedicated *ScriptManager* class, implemented in C#, responsible for:

Fetching Data: The system asynchronously retrieves data feeds via HTTPS using a custom API client, specially designed to interface with Unity's *UnityWebRequest* framework. The API is queried at regular intervals of 20 seconds to ensure up-to-date information.

Decoding Data: Parsing incoming messages in Protobuf (for GTFS-RT) and JSON (for GeoJSON), converting them into structured GeoJSON objects compatible with C# facilitating their use.

Data Validation: Cross-referencing updates with GTFS or GeoJSON static data to detect anomalies. A failover mechanism discards stale updates (>60 seconds) and extrapolates positions using historical data, maintaining consistency.

(Figures 2 and 3) represent two detailed workflows for data acquisition and transformation done by the ApiClient for GeoJSON and GTFS-RT feeds. They illustrate how heterogeneous data sources (GeoJSON feeds and GTFS-RT feeds) are standardized into a unified GeoJSON structure that can be used by the Digital Twin engine.

(Figure 2) shows the workflow for GeoJSON data in red. A dedicated function, *FetchJsonDataThreaded*, initiates an HTTP request. If successful, the response is downloaded and deserialized in a thread separate from the main thread using the Json.NET library, which converts raw JSON into a structured representation in memory. The resulting GeoJSON structure is passed to the callback function for integration. This threaded design ensures that blocking I/O operations do not interfere with Unity's rendering loop, thus maintaining real-time responsiveness.

On the other hand, the blue section details the pipeline for GTFS-RT feeds, which require pre-

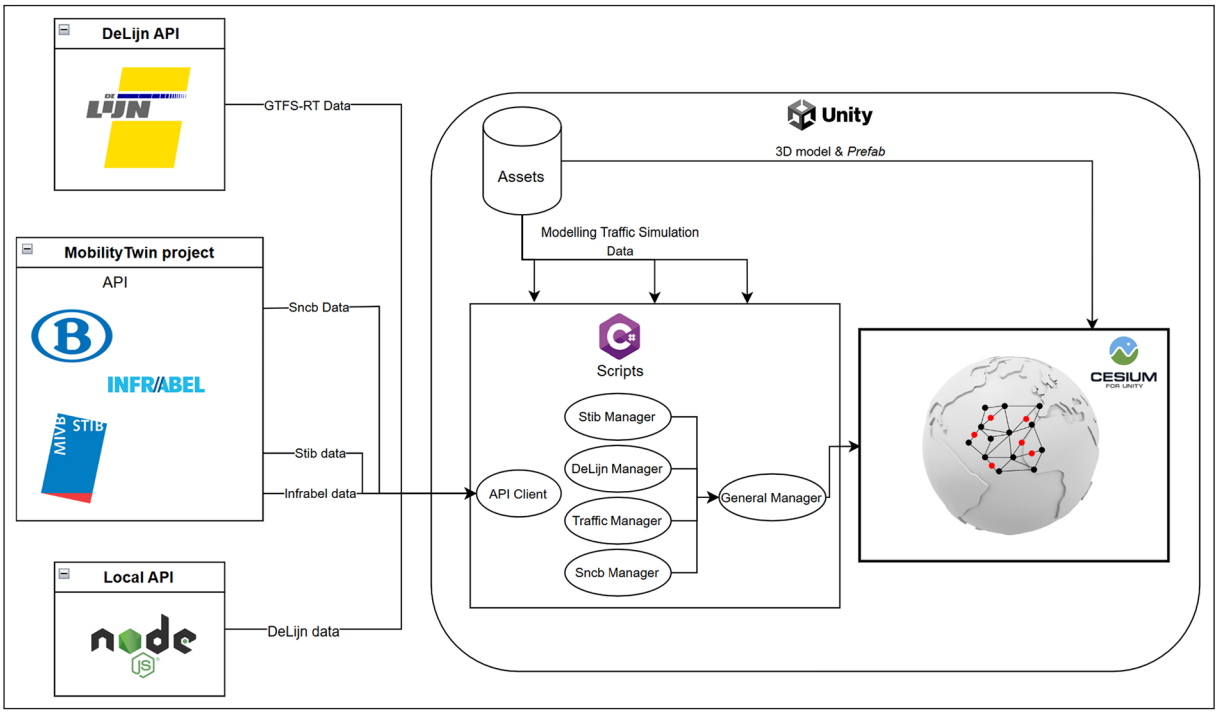

Figure 1. Overall project data flow data

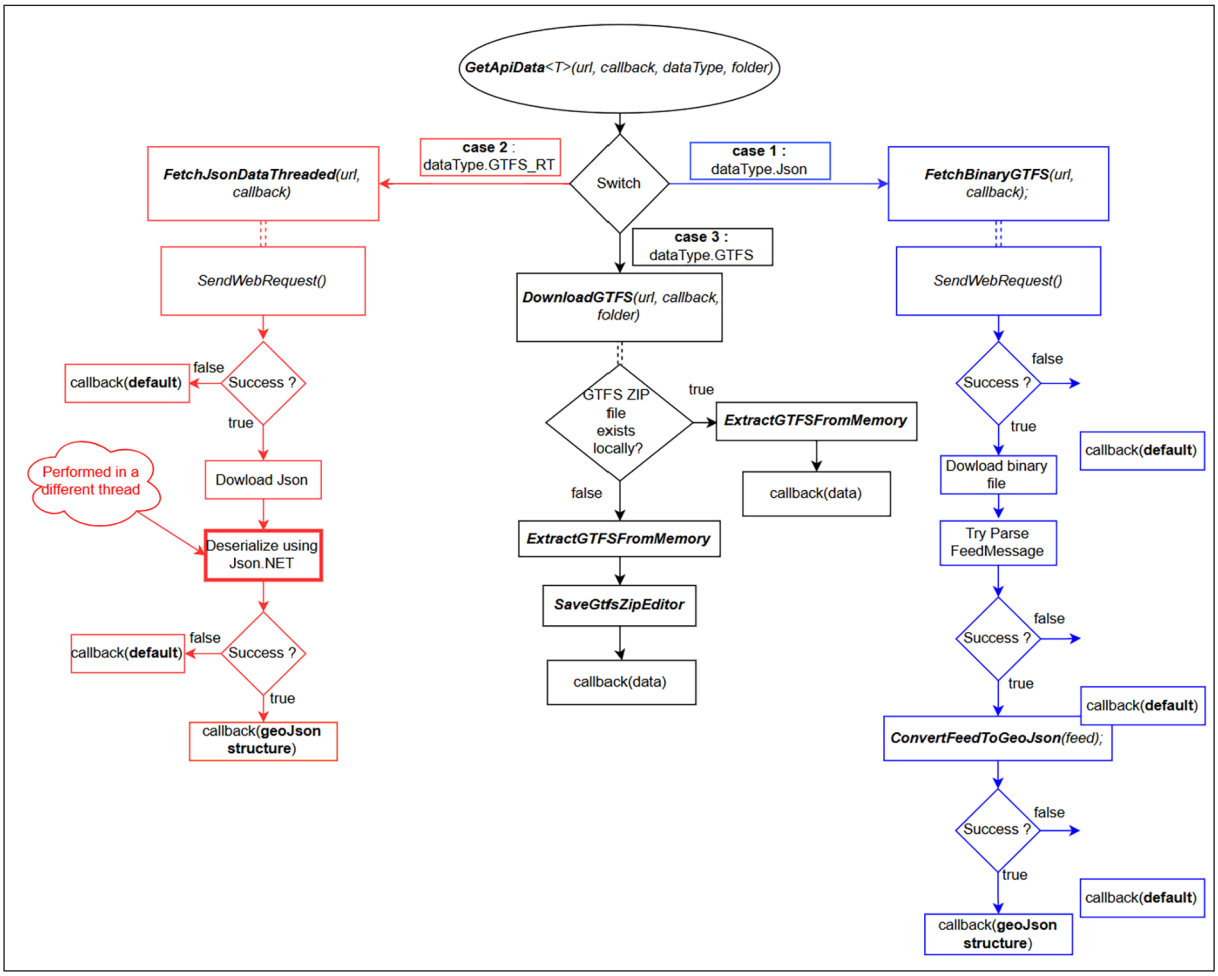

Figure 2. Ingesting different data sources done by the client API

processing because they are encoded in Google Protocol Buffers. The FetchBinaryGTFS entry point sends an HTTP request to retrieve the binary feed. The binary payload is then parsed into a FeedMessage object, a Protobuf schema defined for GTFS-RT. The feed is then passed to *ConvertFeedToGeoJson*, shown in (Figure 3). This conversion module iterates over each entity in the feed (vehicles, route updates, alerts) and constructs a corresponding GeoJSON feature. For entities with vehicle positions, a Point geometry with longitude/latitude coordinates is created. The metadata is converted into JSON objects and attached under the properties field. The result is a complete collection of GeoJSON features that reflect the real-time content of the GTFS-RT feed.

An essential choice in both workflows is the unification of outputs in the GeoJSON standard. This provides a common format for the subsequent steps of visualization, simulation, and/or merging with other geospatial layers. Although GeoJSON is less compact than Protobuf, its readability and broad support within the Unity ecosystem make it a well-suited intermediate format. The ApiClient therefore retrieves the raw data, analyzes and validates it, and then exports it in a consistent format ready for integration into the digital twin environment. This is furthermore in compliance to

In addition to GeoJSON and GTFS-RT, Api Client allows the acquisition of static GTFS datasets. In this dark workflow, the DownloadGTFS entry point checks whether a local copy of the GTFS ZIP archive is already available. If the file exists, it is extracted directly from memory. Otherwise, the archive is downloaded from the remote server, stored locally, and then unzipped. By supporting both static and real-time GTFS workflows, the system enables the integration of historical schedules with live operational data.

Each ScriptManager matches incoming data to corresponding vehicle GameObjects in the Unity scene. Upon receiving updates, the respective managers invoke interpolation routines, passing previous and current positions, timestamps, and orientation data.

All vehicle positions are initially expressed in World Geodetic System 1984 (WGS84)

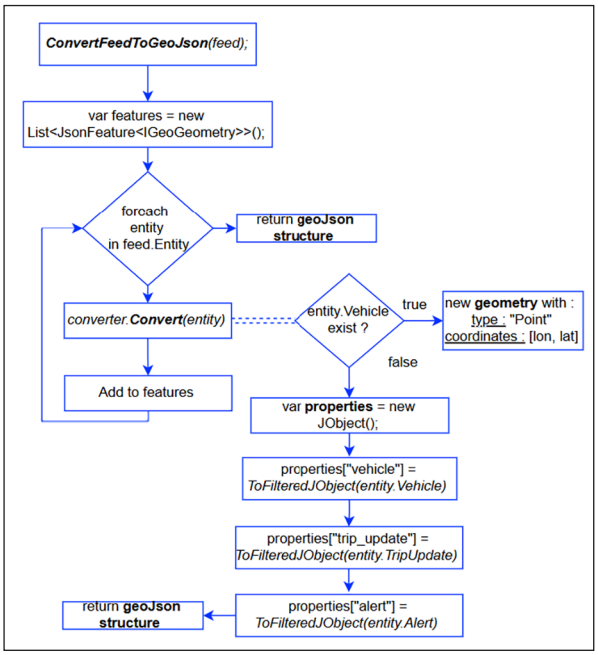

Figure 3. Conversion of GTFS-RT (Protobuf) feeds in a suitable JSON structure

coordinates, as provided by GeoJSON and GTFS-RT feeds. Cesium for Unity supplies internally an integrated geospatial transformation pipeline that converts coordinates into Earth-Centered, Earth Fixed (ECEF) coordinates. Once expressed in ECEF**,** the coordinates are transformed by the *CesiumForUnity.CesiumGeoreference* component into a local, meter-scaled Cartesian reference frame. This local frame is now compatible with Unity's rendering and physics subsystems, while mitigating floating-point precision degradation over large geographic extents.

Since altitude is often absent from GeoJSON datasets, vertical placement is done through terrain-based raycasting. A downward ray is cast from ECEF-derived position toward the terrain mesh at the intersection point, providing an accurate elevation relative to the 3D surface.

This WGS84 to ECEF to local frame pipeline ensures precise positioning of mobility assets across the Brussels-Capital Region. This maintains numerical stability and visual coherence within the virtual environment.

Since GeoJSON data often lacks altitude information, this method addresses this limitation by casting a terrain-based correction by raycasting against the 3D terrain mesh.

A core technical challenge arises from how Cesium for Unity manages geometry loading and interaction. As discussed earlier, Cesium for Unity includes a frustum culling mechanism designed to optimize real-time rendering. This system allows only those 3D tiles whose volume intersects with the frustum of the main camera to be loaded and kept in memory. While this behavior significantly improves performance, it does introduce limitations for applications requiring global and continuous coverage of the scene. In particular, the absence of *MeshCollider* generation for tiles outside the field of view prevents physical interaction with the entire modeled territory.

The problem lies in the absence of MeshColliders for tiles located outside the field of view. The MeshCollider, as a physical component, allows the engine to recognize the 3D geometry of a mesh. It therefore enables interactions such as collision detection via raycasting. Without MeshColliders, the geometry rendered by Cesium remains purely visual, which means that physical interactions are limited to tiles that are loaded and visible to the camera.

Cesium for Unity applies frustum-based streaming, loading only the tiles whose bounding volumes intersect the camera's view. While beneficial for performance, this behavior prevents the generation of MeshColliders for tiles outside the frustum. This limitation is incompatible with mobiliy simulations requiring continuous physical interaction across the entire Brussels-Capital domain.

To address this constraint, the pipeline implements a pre-generated global collision mesh prior to runtime. A custom scanning script incrementally repositions the camera across a predefined latitude-longitude grid covering the Brussels-Capital Region. At each step, Cesium loads the corresponding tiles and generates their *MeshColliders*. At the end, the script extracts, merges and stores these colliders into a single high-precision composite *MeshCollider* saved as a Unity Prefab. A descriptive diagram has been added to illustrate the complete scanning and mesh aggregation workflow in (Figure 5).

At runtime, this global Collider is instantiated independently of Cesium's streaming logic, ensuring persistent physics coverage even when tiles are not currently loaded or visible. This method preserves the advantages of Cesium's rendering while enabling physical consistent interactions throughout the entire study area.

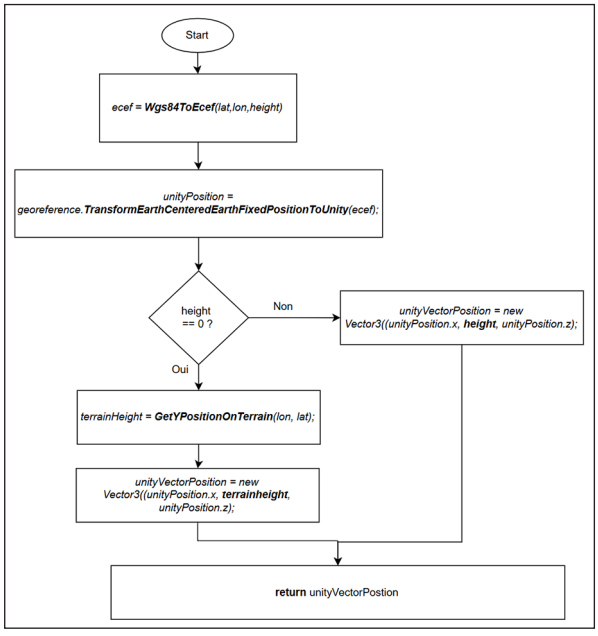

Figure 4. Conversion WGS84 to precise position in Unity space with Cesium for Unity

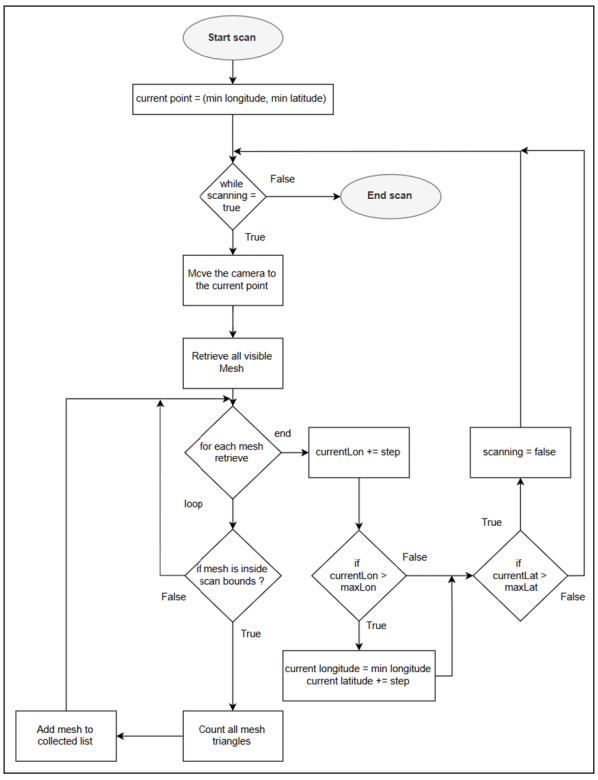

Figure 5. Workflow for Pre-Generating the Global Collision Mesh in Unity

The resulting collider, shown in (Figure 6), provides a stable foundation for real-time mobility simulations at city scale.

To address the 20-second update interval of GTFS-RT and GeoJSON feeds, which can cause visual jitter if used directly, the system employs an interpolation technique: Spherical Linear Interpolation (SLERP). SLERP is used for long-distance routes operated by SNCB/NMBS, STIB/MIVB, and De Lijn, and computes great-circle paths to preserve spherical accuracy over distances. It uses quaternions to interpolate between GPS coordinates, reducing trajectory errors compared to linear methods.

Vehicle updates from transit feeds sometimes arrive with inconsistent or sparse GPS metadata, which can cause sudden jumps, jitters, or unrealistic motion in vehicle positions. To mitigate this, Kalman Filters are applied to smooth trajectories by predicting interim locations based on previous speed, heading, and position. This two-phase predict–update filtering ensures spatial continuity even when updates are missing, delayed, or noisy, maintaining a stable and reliable real-time representation of vehicle movement.

Interpolation parameters are dynamically adjusted according to the transit agency and mode of transportation, allowing different motion behaviors for buses, trams, or trains while maintaining visual consistency. Spherical Linear Interpolation (SLERP) using quaternions is employed. This approach avoids cumulative errors that linear interpolation might introduce and produce smooth, realistic rotations and translations along great-circle paths.

Incoming data points are then matched to their corresponding vehicle GameObjects in Unity. The system passes previous and current positions, timestamps, and orientation data into the interpolation routines, generating precise movement updates in real time.

For animation, Unity's Timeline system is used in combination with custom scripts that dynamically generate keyframe animation curves for each vehicle. These curves integrate the interpolated positions and rotations, ensuring that vehicle

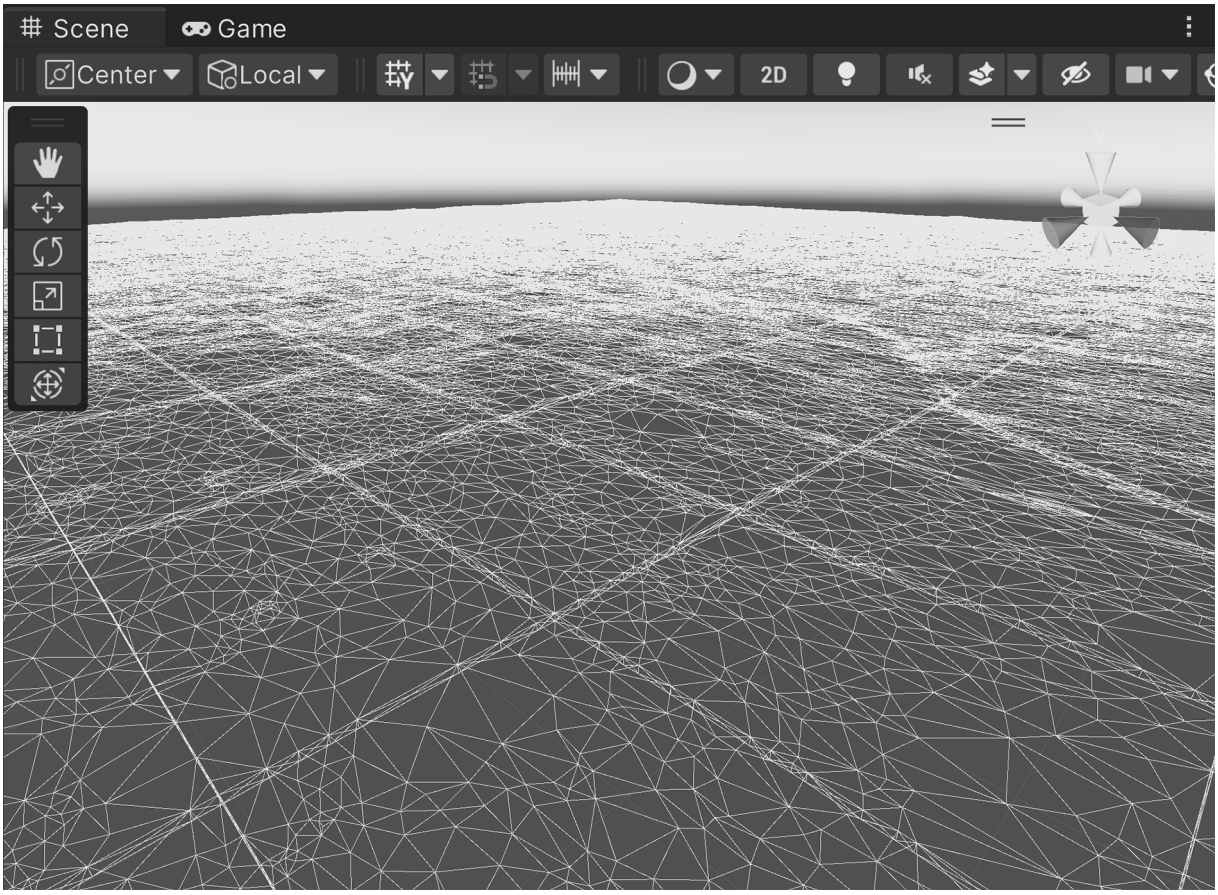

Figure 6. Brussels-Capital Region final mesh collider for physics-based interactions

motion is both temporally synchronized with live feed updates and visually smooth, providing a realistic representation of transit operations within the Digital Twin environment.

The result is an interactive and visually accurate simulation of the Brussels-Capital Region's public transit system, integrating real-time mobility data from multiple transport providers into a unified 3D visualization. Leveraging Unity's Event System and UI Toolkit, users can directly interact with vehicles rendered on the Cesium globe. Clicking on a vehicle reveals detailed contextual metadata, including its current route ID, next scheduled stop, latitude and longitude, and delay duration, enabling users to track vehicle performance and operational status in real time. In addition, the platform allows exploration by transport agencies, vehicle type, or time, making it possible to focus on specific routes or transit modes. Users can navigate the 3D environment with intuitive camera controls, such as zooming, panning, and orbiting, providing flexible perspectives from both city-wide overviews and close-up, street-level views. These capabilities transform raw mobility data into an immersive, analytical tool for researchers, planners, and stakeholders alike, supporting scenario analysis, operational monitoring, and informed decision-making.

To optimize performance and ensure responsible data usage, the system avoids pinging mobility APIs at excessively high frequencies, which could otherwise lead to bandwidth overload, degraded performance, or rate-limit violations. Instead, it fetches updates at the provider-recommended intervals (for example, GTFS-RT updates every 20 seconds) and leverages interpolated motion to generate smooth trajectories between received updates. This interpolation not only fills gaps in the data but also reduces the number of required network requests, lowering the load on both client devices and transit agency servers. Additionally, the system employs local caching of static datasets and asynchronous network operations, ensuring that even if feed updates are delayed or temporarily unavailable, vehicle animations remain continuous and visually fluid. This design provides a secure, efficient, and scalable platform for real-time transit visualization, balancing the needs for data accuracy, network efficiency, and user experience, while fully complying with public data usage policies and agency guidelines.

A. Context of work

This study was conducted through a collaborative effort between academic institutions and public administrations in the Brussels-Capital Region, centered on the development of Smart City concepts. The primary objective is to establish foundational components and practical methodologies by facilitating co-design processes between researchers and public sector stakeholders.

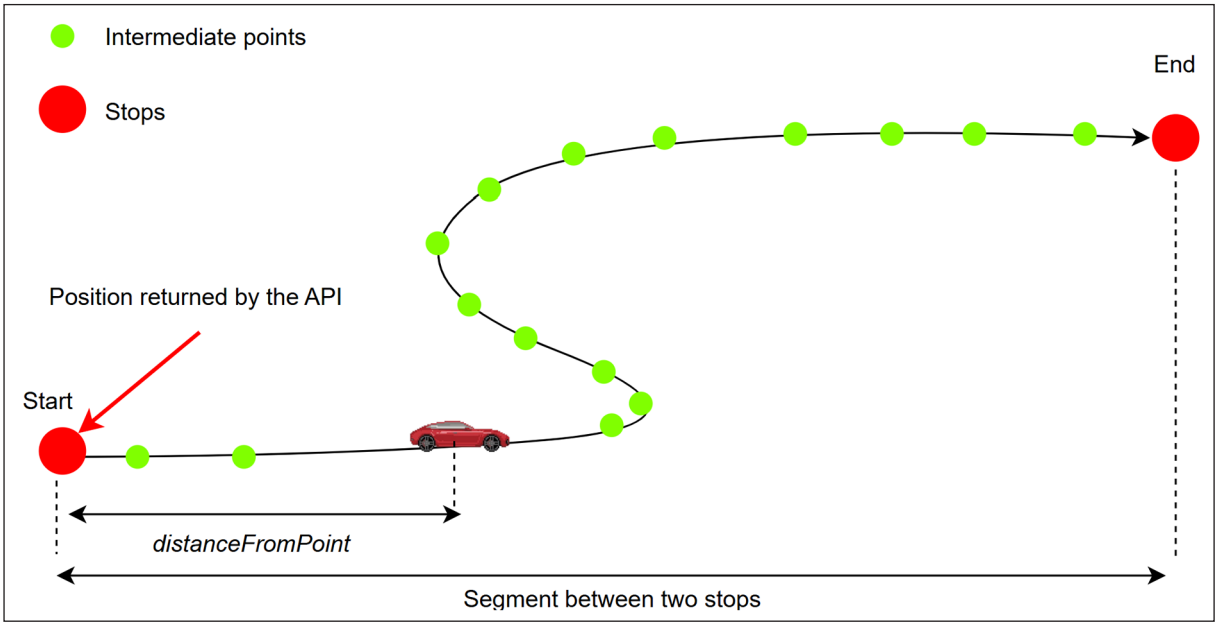

Figure 7. Diagram of vehicle position interpolation operations

Figure 8. Live set-up of a Digital Twin projected on the CAVE infrastructure at the FARI Institute. Photo from Thierry Geenen, 2025, Traffic Twin at FARI CAVE, Copyright FARI

These efforts result in the creation of proof-of-concept prototypes, which are subsequently integrated into the *CAVE infrastructure (Computer Augmented Virtual Environment)* (figure 8) for immersive testing, evaluation, and stakeholder demonstration.

The long-term strategy of the CAVE initiative is to develop a secure and interoperable data space that enables the sharing of both sensitive and non-sensitive data between public and private entities. This data space is intended to support research activities, improve data-driven decision-making, and foster collaboration across institutions. A key aspect of this framework is the inclusion of non-technical stakeholders, achieved through intuitive and accessible visualization tools. These capabilities are critical not only for stakeholder engagement but also for ensuring seamless integration with broader *Digital Twin ecosystems*. In this context, the real-time transit Digital Twin of the Brussels-Capital Region exemplifies how heterogeneous mobility data can be ingested, processed, and visualized within the CAVE. Through combining live GTFS-RT and GeoJSON feeds with interactive 3D visualizations in Unity and Cesium, the platform allows users, both technical and non-technical, to explore, analyze, and interact with public transit operations in an immersive, contextually rich environment.

B. Results and Discussion

Two aspects of the results of this work can be discussed. The first one considers the technical and quantitative performance of the system. The second one is a qualitative analysis of user feedback.

1. Quantitative results

Public transport traffic changes drastically over the course of the day, depending on the weekday and peak hour. The system is stress-tested during peak traffic time to ensure stability throughout the day, week, and year, for smooth interpolation and display in the CAVE environment. The number of elements in the Brussels Capital Region that is included in this work includes, for each public transport company, is provided in (Table 2). A visual representation through a screenshot of the digital twin is shown in Figure 8.

At peak hours on average, the system covers 7000 stops, and reaches over 1300 vehicles, with their associated metadata. It remains stable and perfectly able to display smooth movements through the interpolation, on top of the available buildings and ground routes of said vehicles.

	MIVB/ STIB	NMBS/ SNCB	DeLijn
No. of stops	3 982	743	2 535
Trams	199		
Metro	49		
Busses	410		376
Trains		278	

Table 2. Number of elements inside the Digital Twin

2. Qualitative results

The most important metric for qualitative analysis is user feedback during CAVE visits. Compared to static displays and maps previously available, or "jumping vehicles" situations without the interpolation, users and visitors of the Digital Twin of the CAVE at FARI react very positively to the display. Most requests relate to the direct usability of this implementation on their infrastructure, as it allows for a smooth exploration of the urban environment in real time, at any moment of the day. This feedback includes employees, both technical and non-technical, of the transport companies from which the data is collected, as well as public administration users familiar with the concept of a Digital Twin.

3. Discussion

The use of 3D visualization for mobility data offers an engaging and intuitive representation; however, its analytical value for mobility alone remains limited, as most transit and trajectory analyses can be effectively conducted in 2D. This limitation primarily affects technical experts who are accustomed to interpreting such systems within 2D environments. In the context of the Digital Twin presented in this work, most visitors have a different profile, usually wider and less familiar with such environments. For this larger public, a smoother and easier to understand representation through a 3D Urban Digital Twin framework allows to shift the focus of the discussions from an interpretation of the system to an analysis of the challenges of the city enhancing operational relevance.

By embedding mobility information within a realistic 3D urban context, the platform enables cross-domain analyses that incorporate other urban phenomena, such as air quality, noise exposure, solar access, and energy consumption. Emphasizing this integrative potential provides a strong rationale for adopting 3D environments in the proposed system.

IV. LIMITATIONS AND FUTURE SCOPE OF WORK

A. Data consistency

GTFS-RT and GeoJSON feeds, sourced from different municipalities and transport providers, exhibit significant variability in both data quality and update frequencies. Many feeds lack standardized or complete metadata, while others suffer from irregular update intervals or delayed transmissions due to operational constraints or technical limitations. These inconsistencies make real-time integration into Digital Twins particularly challenging, as smooth vehicle trajectories require reliable temporal and spatial continuity.

While the current implementation uses Spherical Linear Interpolation (SLERP) and coroutine-driven frame updates to smooth vehicle movement between sparse data points, these methods assume relatively consistent input quality. Future work will focus on developing adaptive, data-aware interpolation algorithms that dynamically respond to real-time metrics from transit feeds. These algorithms will detect anomalies such as missing, delayed, or erratic updates and adjust interpolation rates or trigger corrective measures using predictive modeling and historical patterns, including schedules, typical delays, and traffic conditions. Through generating plausible vehicle trajectories when live data is unreliable, this approach will ensure continuity of the simulation, maintaining both user experience and analytical validity even during data outages or disruptions.

B. Performance related

On the rendering front, native deployments using Cesium integrated with Unity deliver highly

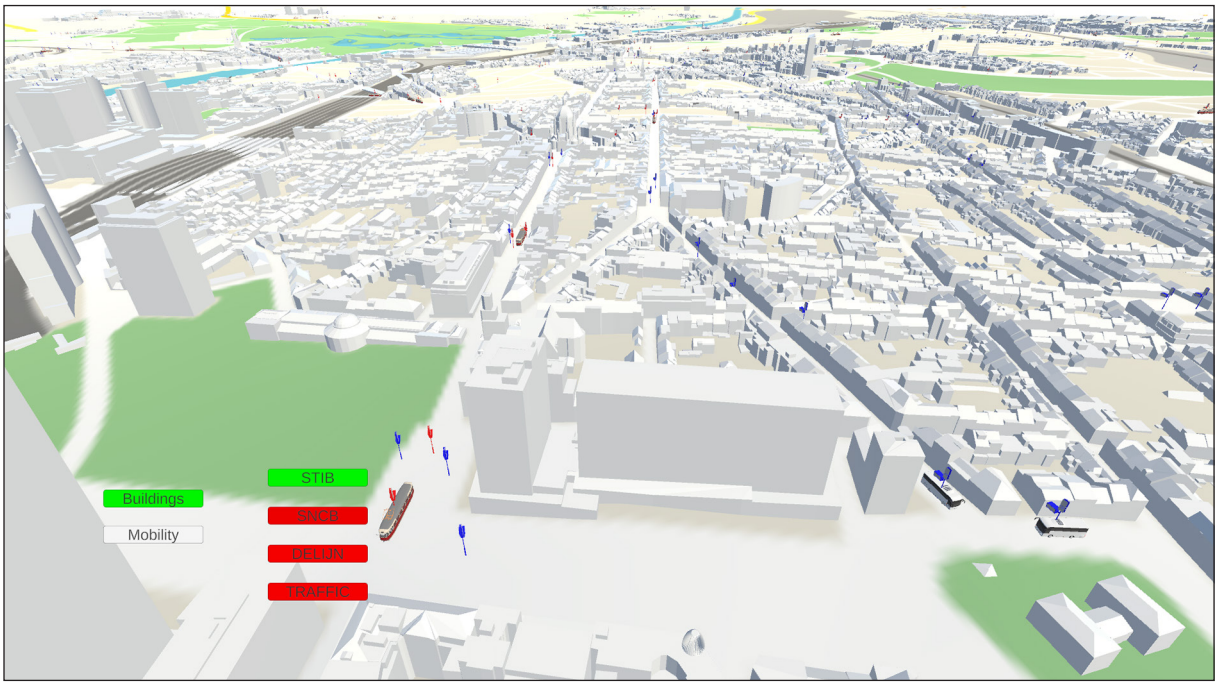

Figure 9. Screenshot of the Digital Twin showing the real-time position of public transport vehicles and their route, with LOD 2.2 buildings.

detailed and visually rich geospatial terrain representations, offering immersive realism essential for accurate digital twin applications. This level of fidelity is not only a technical achievement but also a practical necessity. Stakeholders require geographically accurate and visually coherent representations to monitor operation, coordinate responses, and communicate complex mobility dynamics to diverse audiences. Precise visualizations also provide a foundation for global analyses at city or regional scale where aggregated insights depend on the precise and consistent rendering.

However, this visual fidelity comes at the cost of significant hardware resource demands, including high GPU and CPU usage, which may limit accessibility for users with less powerful devices. While Unity WebGL builds improve accessibility by enabling browser-based interaction, they face technical challenges handling high-frequency data streams like GTFS-RT, especially when animating large amounts of vehicle agents in real time. WebGL's lack of multithreading and limited memory capacity constrains scalability and performance under such demanding loads.

To address these challenges, future developments will explore several optimization strategies. Implementing Level of Detail (LOD) techniques for terrain tiles will allow dynamic adjustment of visual complexity based on camera distance and scene importance, reducing rendering overhead without sacrificing critical detail. Additionally, cloud-based rendering and streaming approaches can offload the most computationally intensive tasks to powerful remote servers, delivering rendered frames with minimal local processing.

Beyond performance, these optimizations also create an entry point for subsequent AI models and simulation tools which rely on geospatially accurate to perform predictive analyses. By ensuring visual realism and computational efficiency, the framework supports also downstream analytical and decision-supports processes.

C. Platform related

Unity and Unity WebGL, while powerful for creating immersive 3D experiences, tend to be highly native platforms that often lock users and developers into specific software configurations and environments. This dependency can limit interoperability and accessibility, as applications built with Unity typically require particular runtime versions, browser support, or hardware capabilities, which may not be universally available. Unity WebGL attempts to bridge this gap by enabling browser deployment, but it still inherits many

constraints such as limited memory management, lack of multithreading, and performance bottlenecks when handling high-frequency real-time data streams. These factors restrict its usability in scenarios demanding broad platform compatibility and lightweight resource consumption.

To address the interoperability and accessibility challenges posed by native Unity and Unity WebGL deployments, a robust, non-native browser-based architecture is a must. The proposed strategy involves fully decoupling the simulation logic from the rendering layer. Real-time data ingestion, interpolation, and processing will be handled on the backend via a lightweight Node.js server, which communicates with the frontend through WebSockets. On the client side, CesiumJS will be used for high-performance geospatial terrain visualization, while Three.js will support dynamic 3D animation of transit vehicles and associated metadata layers. Adopting open web standards and modular technologies such as Vue.js for the UI layer will support easier integration with other digital dashboards and broader accessibility across devices without requiring end-user installations.

CONCLUSION

This study has developed and demonstrated a comprehensive framework for integrating heterogeneous real-time feeds into an interactive Digital Twin environment, leveraging Unity and the Cesium for Unity SDK. By harmonizing data from multiple transit providers in the Brussels-Capital Region, STIB/MIVB (metro, tram, bus via GeoJSON), SNCB/NMBS (intercity and regional trains via GeoJSON), Infrabel (railway infrastructure as static datasets), and De Lijn (Flanders buses and trams via GTFS/GTFS-RT), the framework delivers a unified, high-fidelity visualization of urban and regional mobility.

The system's modular architecture, comprising ingestion, processing, visualization, and interaction layers, ensures robust handling of real-time data streams, achieving smooth vehicle animations through adaptive asynchronous API pinging, parsing (Protobuf for GTFS-RT, JSON for GeoJSON), validation, and caching strategies. Static datasets are cross-referenced to validate updates, with stale or anomalous records discarded and positions extrapolated from historical trajectories.

Central to the framework's implementation is its interpolation and predictive modeling techniques. Spherical Linear Interpolation (SLERP) is applied consistently across both GTFS-RT and GeoJSON feeds, addressing the 20-second GTFS-RT update interval, ensuring spatially coherent trajectory on a geodetic sphere and reducing visual jitter caused by irregular update intervals. A Kalman filter enhances positional stability in noisy and infrequent datasets, maintaining continuity during data outages and delays.

The integration of Unity's Timeline system with Cesium's 3D tilesets provides a geospatially accurate Digital Twin of Brussels-Capital Region with positional accuracy. The interaction layer, built with Unity's UI Toolkit, empowers users to explore transit data through intuitive camera controls and metadata overlays.

While native deployments offer superior performance, the framework's WebGL-based browser implementation demonstrates significant potential for scalable, public-facing applications. This accessibility broadens the framework's applicability for urban planners, transit agencies, and the public, enabling real-time monitoring and analysis of complex transit systems through minimal hardware overheads.

The framework's contributions lie in its multi-source integration, modularity, scalability, and adaptability. Its decoupled architecture supports rapid integration of new agencies or data formats.

By optimizing data usage and prioritizing compliance, the system sets a benchmark for responsible consumption of public transit data, aligning with agency policies and data governance standards.

ACKNOWLEDGEMENTS

This work was made possible through the generous support of the Recovery and Resilience Facility (RRF), the European Regional Development Fund (ERDF), and the Brussels-Capital Region, whose funding enabled the research, development, and deployment of the Digital Twin framework presented in this study.

The authors gratefully acknowledge the collaboration of public transport agencies,

STIB/MIVB, SNCB/NMBS, Infrabel, and De Lijn, whose open data services and technical documentation provided the foundation for multi-modal transit integration within the Digital Twin environment.

BIBLIOGRAPHY

Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., & Schmidt, C. (2016). *The GeoJSON format* (RFC 7946). Internet Engineering Task Force. https://doi.org/10.17487/RFC7946

Chib, P. S., & Singh, P. (2024). *Pedestrian trajectory prediction with missing data: Datasets, imputation, and benchmarking* (arXiv:2411.00174). arXiv. https://doi.org/10.48550/arXiv.2411.00174

Clark, T., Kulkarni, V., & Barn, B. S. (2025). *An introduction to digital twins*. In Digital twins for simulation-based decision-making (pp.1–13). Springer. https://doi.org/10.1007/978-3-031-89654-5_1

Crampen, D., Yadav, Y., Ishar, S., Zlatanova, S., & Tian, W. (2024). Development of 3D UDTs for traffic monitoring in Unreal 5 game engine. The International Archives of the Photogrammetry, *Remote Sensing and Spatial Information Sciences*, XLVIII-4-2024, 131–138. https://doi.org/10.5194/isprs-archives-XLVIII-4-2024-131-2024

Datta, S. P. A. (2016). *Emergence of digital twins* (arXiv:1610.06467). arXiv. https://doi.org/10.48550/arXiv.1610.06467

David, I., Shao, G., Tilbury, D., Gomes, C., & Zarkhout, B. (2024). Interoperability of digital twins: Challenges, success factors, and future research directions. In Proceedings of the 12th International Symposium On Leveraging Applications of Formal Methods (ISoLA 2024). NIST. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=956427

Iliuţă, M.-E., Moisescu, M.-A., Pop, E., Ionita, A.-D., Caramihai, S.-I., & Mitulescu, T.-C. (2024). Digital Twin—A Review of the Evolution from Concept to Technology and Its Analytical Perspectives on Applications in Various Fields. *Applied Sciences*, 14(13), 5454. https://doi.org/10.3390/app14135454

Irfan, M. S., Dasgupta, S., & Rahman, M. (2024). Toward transportation digital twin systems for traffic safety and mobility: A review. *IEEE Internet of Things Journal*. https://doi.org/10.1109/JIOT.2024.3395186

Keil, J., Edler, D., Schmitt, T., & Dickmann, F. (2021). Creating immersive virtual environments based on open geospatial data and game engines. *KN Journal of Cartography and Geographic Information*, 71(1), 53–65. https://doi.org/10.1007/s42489-020-00069-6

Kušić, K., Schumann, R., & Ivanjko, E. (2022). A digital twin in transportation: Real-time synergy of traffic data streams and simulation for virtualizing motorway dynamics. *Advanced Engineering Informatics*, 54, 101858. https://doi.org/10.1016/j.aei.2022.101858

Merten, G. & Sakr, M. (2023) *AITwin/Components*, GitHub. https://github.com/AITwin/Components

Merten, G. & Sakr, M. (2023). *The Mobility Data Platform for Brussels*, Brussels Mobility Twin. https://mobilitytwin.brussels

Newmark, G. L. (2024). *Assessing GTFS accuracy*. Mineta Transportation Institute. https://doi.org/10.31979/mti.2024.2017

Open Geospatial Consortium (2025). OGC API – Moving Features Standard. OGC. https://www.ogc.org/standards/ogc-api-moving-features

Pei, Y., Biswas, S., Fussell, D. S., & Pingali, K. (2019). *An elementary introduction to Kalman filtering* (arXiv:1710.04055). https://doi.org/10.48550/arXiv.1710.04055

Qiao, S.-J., Han, N., Zhu, X.-W., Shu, H.-P., Zheng, J.-L., & Yuan, C.-A. (2018). A dynamic trajectory prediction algorithm based on Kalman filter. *Acta Electronica Sinica*, 46(2), 418–423. https://10.3969/j.issn.0372-2112.2018.02.022

Ruhl, T. (2024). *Datastore.brussels dataset: Brussels public geodata catalog*. https://datastore.brussels/web/data/dataset/e9ec 2aa4-cffd-11ee-bccc-00090ffe0001

Singh, M., Srivastava, R., Fuenmayor, E., Kuts, V., Qiao, Y., Murray, N., Devine, D. (2022). Applications of Digital Twin across Industries: A Review. *Applied Sciences*, 12(11), 5727. https://doi.org/10.3390/app12115727

Webb, L., Higgs, G., Langford, M., & Berry, R. (2025). Evaluating real-time and scheduled public transport data: Challenges and opportunities. *ISPRS International Journal of Geo-Information*, 14(7), Article 243. https://doi.org/10.3390/ijgi14070243

Wang, C., Zhang, J., Wang, J., & Wu, Y. (2025). Confidence-based fusion of AC-LSTM and Kalman filter for accurate space target trajectory prediction. *Aerospace*, 12(4), 347. https://doi.org/10.3390/aerospace12040347

Wang, J., Liu, K., & Li, H. (2024). LSTM-based graph attention network for vehicle trajectory prediction. *Computer Networks*, 235, 110477. https://doi.org/10.1016/j.comnet.2024.110477

Wong, J. (2025). *Algorithmic analysis of GTFS-RT vehicle position accuracy* (arXiv:2506.06479). arXiv. https://doi.org/10.48550/arXiv.2506.06479

Würstle, P., Padsala, R., Santhanavanich, T., & Coors, V. (2022). Viability testing of game engine usage for visualization of 3D geospatial data with OGC standards. ISPRS Annals of the Photogrammetry, *Remote Sensing and Spatial Information Sciences*, X-4/W2-2022, 281–288. https://doi.org/10.5194/isprs-annals-X-4-W2-2022-281-2022

Coordinates of authors:

Susheel NATH
FARI – AI for the Common Good Institute
Vrije Universiteit Brussel (VUB)
Susheel.nath@fari.brussels

Valentin MARCHAND
ISIB – Institut Superior Industriel de Bruxelles
Valentinmarchand02@gmail.com

Carl MÖRCH
FARI – AI for the Common Good Institute
Université Libre de Bruxelles (ULB)
Carl.morch@fari.brussels

Martin CANTER
FARI – AI for the Common Good Institute
Vrije Universiteit Brussel (VUB)
Martin.canter@fari.brussels