PostGIS-Unity integration for scalable 2D/3D geospatial visualisation and querying

Mitko Aleksandrov[1], Sisi Zlatanova[2], Angela Schwering[3]

[1] Institute for Artificial Intelligence Research and Development of Serbia, Fruškogorska 1, 21000 Novi Sad, Serbia – mitko.aleksandrov@ivi.ac.rs
[2] Geospatial Research Innovation Development (GRID), School of Built Environment, UNSW Sydney, NSW 2052, Australia – s.zlatanova@unsw.edu.au
[3] Institute for Geoinformatics, University of Münster, Heisenbergstraße 2, 48149 Münster, Germany – schwering@uni-muenster.de

Keywords: PostGIS, Unity, 3D city models, LLMs, digital twins

Abstract

3D city models and interactive geospatial scenes have become increasingly used not only for visualisation, but also for simulation, spatial analysis, urban planning, and decision support. At the same time, game engines such as Unity are prominent platforms used for the development of geospatial applications, offering effective 3D rendering and interaction. However, existing workflows involving 3D geospatial data still depend on format conversions and dedicated server infrastructures for streaming spatial data, making database-centred scene management unnecessarily complex. This paper presents a PostGIS and Unity integration that explores a more direct communication between both solutions. The application supports (i) direct PostGIS-to-Unity visualisation of 3D spatial data without requiring a dedicated server, (ii) storage and retrieval of georeferenced 3D models and materials for scene loading, (iii) tile-based loading and metadata retrieval for scalable interactive environments, and (iv) query-driven scene rendering, including LLM support for spatial query generation assistance. Rather than treating the database only as a storage solution, the proposed approach positions PostGIS as an active runtime component for visualisation, querying, and update. The work is relevant for digital twins, urban visualisation, and exploratory geospatial applications where tight coupling between spatial databases, interactive 3D rendering, and AI-assisted querying is desirable.

1. Introduction

In recent years, 3D city models have moved well beyond their early role as visual products. They are now used across a wide range of domains, including environmental simulation, urban planning, navigation, infrastructure management, and decision-making support. Biljecki et al. (2015) pointed out that 3D city models already span across dozens of use cases and in more than one hundred application contexts, which makes them not just representational assets, but operational components of urban information systems. This raises expectations for tools that can connect storage, analysis, and interactive visualisation more directly.

At the same time, game engines have become increasingly relevant in geospatial context and research. Unity, in particular, offers a compelling setup for interactive 3D geovisualisation because it combines real-time rendering, flexible user interaction, and a mature development ecosystem. The current literature highlights several issues though: geospatial data often arrive through fragmented pipelines, georeferencing remains a nontrivial issue, and database integration is frequently indirect rather than native (Buyuksalih et al., 2017; Laksono and Aditya, 2019). Recent Unity-based geospatial workflow suggests requirement for explicit alignment between real-world coordinates and Unity's internal coordinate space, often via reference control points (Subramaniyam et al., 2024).

In practice, large 3D geospatial datasets are often converted into 3D Tiles or related structures and then served through dedicated server infrastructure. This has clear advantages for streaming and interoperability, and it is now a dominant pattern in the ecosystem. However, it also introduces another layer between source data and a rendering environment that needs to be run all the time. For workflows that need live database querying, repeated retrieval, and an instant visualisation, this separation can become limiting. This is a concussion drawn from the current emphasis on 3D Tiles standards, conversion tools, and server-based engine integration in recent literature (Marnat et al., 2022).

Most commonly PostGIS data are visualised in QGIS, when open-source solutions are considered. However, for 3D data the workflow is less seamless as for 2D data, lacking some core capabilities such as metadata exploration, georeferencing, scene editing, and no easy SQL to visualisation integration. While QGIS includes a 3D Map View and emerging support for 3D tiles-based content, current support remains partial, and the overall workflow is oriented more toward 3D viewing than towards database-centred scene management and query-driven interaction (Rantanen et al., 2023).

This paper investigates an alternative path through direct PostGIS integration to Unity. The core idea is to treat the spatial database not only as a storage backend, but also as a runtime source for 2D and 3D scene content, query execution, metadata retrieval, and georeferenced model persistence. The work further extends this architecture with Large Language Model (LLM)-assisted query generation through Model Context Protocol (MCP), allowing natural-language interaction and support for spatial SQL assistance and query-driven rendering. In this sense, the proposed solution sits also at the intersection between spatial databases, visualisation, and LLMs.

The contribution of the paper is therefore twofold. Technically, it presents a Unity solution that enables direct visualisation from PostGIS, georeferenced roundtripping of 3D content management, and tile-based loading for scalable interactive scenes. Conceptually, it argues for a database-centred alternative to purely server-centred or file-conversion-centred workflows, especially useful for research prototypes, digital twins, and exploratory geospatial applications that require tighter control over query logic and data management.

2. Background

PostGIS is one of the most widely used spatial extensions for relational databases, extending PostgreSQL with native support for spatial storage, indexing, and processing. Its importance in geospatial systems comes not only from support for standard vector and raster operations, but also from its ability to combine spatial functions with SQL querying, transaction handling, and

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026

data management. Regarding 3D, PostGIS supports geometry types and operations that go beyond GIS. The system includes support for POLYHEDRALSURFACE, TINs, and advanced 3D processing functions. These capabilities are relevant for Unity integration because they open a path for storage, retrieval, and processing directly from database. Database-centred management of semantic 3D city models is not new. Systems such as 3DCityDB have shown the value of using PostGIS as a backend for large, structured urban datasets compliant with CityGML (Yao et al., 2018). The importance of structured metadata in 3D city models is well established, since metadata support rapid understanding of dataset characteristics and help determine fitness for use (Labetski et al., 2018). What remains less developed is the direct bridge from that database layer into game-engines.

The use of game engines in geospatial applications has grown over the last two decades because they offer qualities that traditional GIS software often lacks: high-performance immersive rendering, custom interaction design, and integration with simulations and sensor data. Unity-based studies already framed the engine as a viable platform for 3D city modelling and visualisation, while more recent work has positioned game engines as useful environments for urban digital twins and city-wide data integration (Buyuksalih et al., 2017; Rantanen et al., 2023). Laksono and Aditya (2019) noted that game engines have limited native support for geospatial data and emphasized the difficulty of maintaining georeferenced models. Rantanen et al. (2023) likewise highlighted georeferencing as a recurring issue when integrating city-wide datasets in a game engine.

Over the past a few years, several plugins have been developed mainly focusing on geospatial data visualisation, including *Cesium for Unity*, *ArcGIS Maps SDK for Unity*, *Mapbox Unity SDK, and pg2b3dm* (Table 1). However, they lack support for concrete geospatial data importing and exporting options, as well as complex querying and data management.

Tool	Main purpose	Comparison to our work
ArcGIS Maps SDK for Unity	Unity access to ArcGIS maps, 3D content, XR/geospatial scenes	Not providing PostGIS database connection
Mapbox Maps SDK for Unity	Mapbox maps/location services in Unity	Focuses on Mapbox services, not PostGIS querying/export
Cesium for Unity	Standards-based streaming of optimised 3D geospatial content	Not providing PostGIS database connection
pg2b3dm	Create 3D tiles from PostGIS data	Aims at static PostGIS data visualisation, but not management

Table 1. Comparison of available Unity-based geospatial tools

Tile-based spatial organization is a common strategy for scalable geospatial data distribution, for example, through quadtree-based management of large-capacity datasets (Lee et al., 2020). In addition, 3D Tiles has become one of the central standards for streaming massive heterogeneous datasets. The OGC standard defines a hierarchical tileset structure for renderable 3D buildings, point clouds, BIM/CAD data, and photogrammetry mesh. Recent research around tools such as Py3DTilers reflect capabilities such as spatial data conversion, level-of-detail (LOD) generation, and server-compatible packaging with a goal of making large urban datasets viewable in real time (Marnat et al., 2022). Similarly, many current 3D geospatial and digital twin workflows emphasize application-level integration, viewer-side delivery, and standards-based transformation pipelines rather than direct runtime interaction with a spatial database (Jeddoub et al., 2023; Shaji et al., 2025). While such approaches are well suited to interoperability and scalable streaming, they can also function as an unnecessary intermediate step in workflows where the application needs to communicate directly with a spatial database for querying, filtering, metadata retrieval, or bidirectional editing.

A recent direction in interactive geospatial systems is the use of LLMs to lower the barrier to querying spatial and non-spatial data. Recent work on both spatial databases and GIS workflows suggests that natural language can translate user questions into executable spatial queries or analysis steps, especially for non-expert users (Liu et al., 2025; Pierdicca et al., 2025). At the same time, the main challenge is not language generation alone, but grounding the model in the actual database schema, domain semantics, and available executable operations, which is also a central issue in current text-to-SQL research. In this context, the MCP is relevant as a structured interoperability layer between LLMs and external tools (Hou et al., 2025). For the present work, MCP serve as the architectural bridge between an LLM and a PostGIS-backed Unity workflow: instead of treating query generation as an isolated prompt-engineering feature, the solution can expose database-aware operations through a structured tool layer, allowing the LLM to help formulate or refine queries while execution remains grounded in PostGIS and the Unity application logic.

3. System Design and Capabilities

Since the idea is to achieve an integration between Unity and PostGIS, it is important to carefully evaluate the capabilities of both systems to utilise the best from both.

PostGIS ecosystem provides powerful spatial data management, indexing, and advanced geospatial analysis directly within the database, making it highly efficient for handling large-scale and complex spatial queries. However, it lacks native interactive visualisation capabilities, particularly for 3D data, and some processing functions like ST_Tesselate are not as quick as they should be.

Unity, on the other hand, excels at real-time 3D visualisation, and user interactivity, as well as allows the communication with third party libraries for data processing and interaction. Its limitation is that it is not designed for spatial data storage and spatial retrieval, requiring external systems like PostGIS to efficiently manage and query geospatial data.

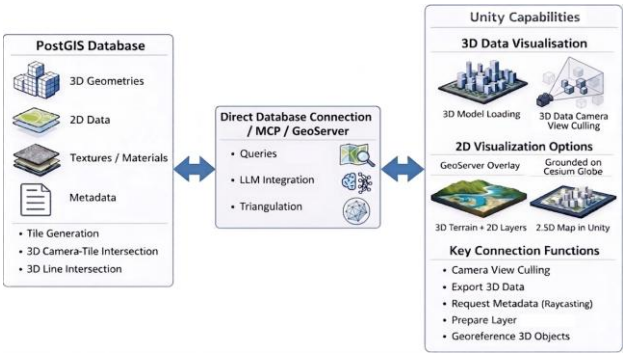

Figure 1. Architecture and functional capabilities of the Unity–PostGIS integration

The Figure 1 presents a conceptual workflow for integrating spatial data between a PostGIS database and a Unity-based application through a direct database connection and an MCP API layer. PostGIS acts as the central repository for multiple

asset types, including 3D geometries, 2D spatial data, textures or materials, and metadata. In addition, the database supports backend spatial processing tasks such as 3D camera–tile intersection, tile-based data retrieval, and 3D line intersection for metadata delivery. This means that PostGIS is not only used for storage, but also for preparing and filtering spatial content so that only the relevant data can be sent to the rendering system.

The middle layer, which is integrated into Unity for easier user usage, represents the communication interface between PostGIS and Unity interface. Through a direct database connection using *NetTopologySuite*, *ModelContextProtocol Core* and this *Triangulator*[1] library, Unity can request spatial queries, retrieve metadata, and support LLM-assisted interaction with the data. As a result, the connection layer supports both efficient data access and export, as well as more advanced system support such as intelligent query construction and explanation, as well as semantic interpretation of stored data.

Open-source *Cesium for Unity* plugin is used as a base. For 2D data, texture images are overlayed on top of 3D terrains, while for 3D we propose PostGIS-based tile structure that is used for visualisation in Unity. In addition, the application supports georeferencing of 3D objects, and object metadata retrieval relying on raycasting in Unity. Unity allows dynamic rendering of 2D and 3D data, while performing dynamic content culling based on camera view, visualising only visible tiles to a user.

3.1 Database design and data retrieval

To efficiently fetch data from PostGIS, a tile-based retrieval is implemented. Tiling enables loaded data in segments without waiting for the entire layer to load in memory first, as well as to update only a portion of the scene if a user moves. The original dataset is preserved as the main source layer, while all axillary structures (i.e., layers + indexes) are placed in a separate schema. For each layer, tile layer is created in the schema, and there is a layer representing the current camera view in Unity.

The tile layer stores a unique tile_id, a tile_extent capturing its extent, and an extent_3d geometry storing the 3D extent of the original data. This layer is created on the fly, the first time the user requests this layer. A 2D tile is stored in the same coordinate reference system as the original dataset so that features can be associated with tiles directly in 2D space, while the 3D extent is stored in the Cesium-based coordinate reference system (CRS), which is 3D Cartesian, metric-based, geocentric CRS - EPSG:4978. The reason for this is to match the Unity camera CRS when performing 3D intersection. In this way, 3D visibility is evaluated between the camera geometry and data extents in a tile. The tile_id is used only for Unity-side caching and tracking of already retrieved tiles.

To avoid duplicate retrieval of the same geometries in multiple tiles, a point indexing is applied to original geometry rather than creating 2D box index for each geometry. Therefore, ST_PointOnSurface function is used instead of ST_Centroid, guaranteeing that the representative point lies inside the original geometry.

The indexing strategy is split this way: a B-tree primary key is created on tile_id, a GiST index on tile_extent, and an n-D GiST index on extent_3d; the original dataset contains a point-based spatial index derived from the original geometry.

This keeps the retrieval logic duplicate-free, effectively fetches tiles with data that is visible by firstly performing 3D intersection between the Unity camera view and the 3D data extents per tile followed by intersection between tiles and points' indexes representing the original data, as well as preserves the possibility of removing the entire helper schema independently of the original data.

3.2 Database interaction

The interaction with PostGIS focuses on data preparation for effective rendering and storage, as well as to allow intelligent spatial data querying using LLMs.

3.2.1 Geometry data preparation. The role of *NetTopologySuite* library is to support the execution of queries and the handling of returned data within the Unity environment. However, the library is primarily limited to 2D geometry types that directly correspond to standard PostGIS representations, such as Point, LineString, and Polygon.

It is important to indicate that overlaying 2D geometry data directly from PostGIS on a 3D terrain in Unity is not recommended, since Cesium for Unity plugin relies on Hierarchical level-of-detail (HLOD) system to render a 3D terrain, which renders higher quality terrain parts closer to camera, and lower quality ones further away from it. As a result, even when snapping 2D data on a 3D terrain, the terrain starts updating with lower or higher quality terrain parts by moving in the scene, and some 2D geometry parts easily become invisible by going below the 3D terrain. Therefore, transferring 2D vector data to images/textures is recommended. In the given context, three options are evaluated, including GeoServer WMS image overlays, direct retrieval of GeoJSON, which is rasterised in Unity, as well as raster generation in PostGIS. In all three cases, texture images are shown on top of a 3D terrain.

For 3D data *Triangulator* library is used, triangulating PostGIS geometries in order to convert vector-based representations into triangle-based mesh structures suitable for rendering 3D objects in Unity. This step is particularly important because Unity relies on triangles for isualization, while PostGIS stores data in analytical geometric formats that are not directly compatible with real-time rendering pipelines. The library enables the generation of triangle meshes from polygonal inputs, and further processing within the Unity environment. In the proposed workflow, triangulation can be applied either on-the-fly after retrieving geometries from PostGIS or as a pre-processing step, where triangulated meshes are stored as pre-prepared layers to improve runtime performance. It is important to mention that relying on PostGIS's ST_Tesselate function is avoided to triangulate polyhedralsurfaces, because it is slower in order of a magnitude compared to the Triangulator library.

Coordinate precision in Unity is handled by separating global geospatial positioning from local mesh coordinates. Source geometries remain in their declared PostGIS CRS for storage and querying and are transformed to WGS84 longitude and latitude when transferred to the visualisation pipeline, while the Z coordinate is treated as ellipsoidal height.

3.2.2 PostGIS querying using LLMs. When it comes to querying of data, a SQL window is created allowing users to type and execute SQL queries, as well as visualise results (Figure 2). For each query user can pick colouring style, a prefab as a geometry replacement, scale, to ground on the terrain, and create tiles for smooth visualisation.

[1] https://github.com/bertt/triangulator (accessed on 04/07/2026)

https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026

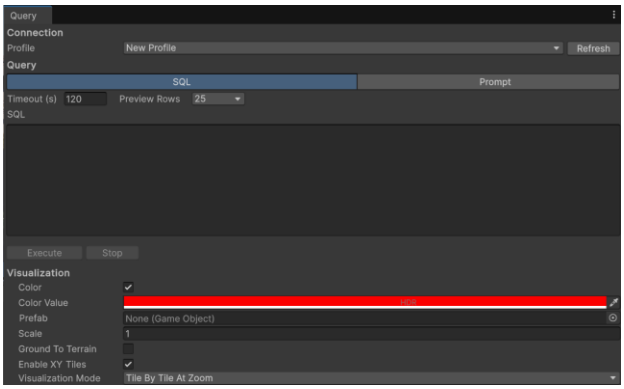

Figure 2. SQL query execution panel allowing to preview and visualise geospatial data

Regarding LLMs and MCP, the idea is to provide an additional interaction layer that simplifies access to spatial data and metadata. Figure 3 shows the architectural diagram, illustrating the Unity-based system architecture integrated with LLMs via the MCP. The core process begins with user interaction through an Editor/UI integration window in Unity, which feeds user actions and prompts to the Unity Orchestration layer. This centralised layer then coordinates the prompt-building process and communicates to two distinct clients within Unity. On one track, it communicates to the Ollama client, which uses HTTP to access an external Ollama LLM model. The Ollama model is responsible for complex tasks like query construction, explanation of retrieved results, and fixing broken SQL. On the other side, the Unity Orchestration layer communicates to the MCP client, which utilises stdio to communicate with an external Postgres MCP Server. This server acts as a specific tool bridge, allowing the LLM, once it has generated a query, to utilise this specific tool bridge to generate and execute read-only queries against a linked PostgreSQL DB. The query results then flow back through the entire system to the user interface. This structure facilitates a seamless workflow between the LLM and the application's data, preserving the underlying PostGIS-based query workflow. Crucially, while the MCP facilitates the interaction with PostGIS, the Ollama model is the one that provides the 'intelligence'. This makes the system more accessible for non-expert users while preserving the underlying workflow. The application allows users to select and download from a list of currently available state-of-the-art open-source LLM models that are available on Ollama and are the most appropriate to run locally on user's local machine.

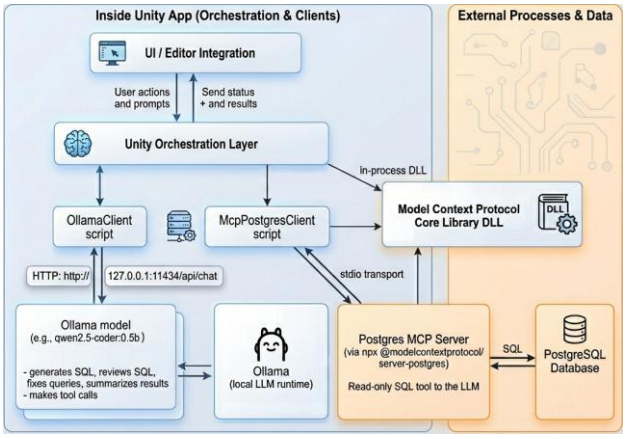

Figure 3. Unity and PostGIS system integration through MCP and LLMs

Figure 4 shows what kind of help is possible to provide to a user when answering the prompt. In general, six different options are identified, including to generate and execute the SQL, generate only the SQL, review an existing SQL statement, fix broken SQL, suggest more efficient SQL, if possible, summarise query results in plain language. At the same time, user can select specific tables that should be considered during the SQL generation.

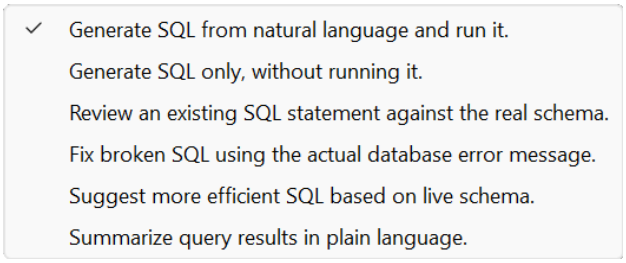

Figure 4. Helping SQL prompt options using LLMs and MCP

3.3 Unity interaction and spatial data preparation

A comprehensive session management has been implemented in Unity to support robust communication with PostgreSQL. The system provides persistent connection profiles, and secure storage of persisted credentials. In practice, this enables the application to manage multiple databases efficiently while reducing repeated checking (Figure 5).

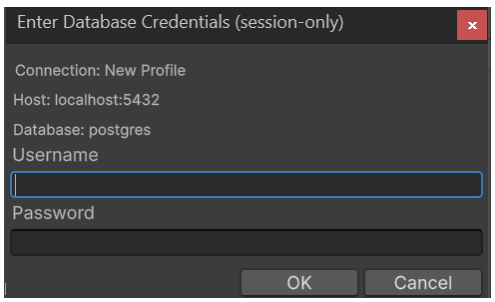

Figure 5. Session credentials prompt to establish a connection with PostgreSQL

The visualisation workflow enables Unity to read spatial data types, and render results through geometry-specific procedures, targeting points, lines, and polygons, as well as dedicated handling exist for TIN and PolyhedralSurface geometries. The implementation does not rely on a single rendering pathway; instead, it supports both direct Unity-based rendering and server-backed overlay visualisation through GeoServer particularly for 2D data.

A dedicated preprocessing stage is implemented to improve rendering performance for repeated visualisation tasks. The system can construct Unity-ready mesh table that eliminates the need for repeated triangulation during loading. This preprocessing approach represents an optimisation strategy, which allows shifting computationally expensive operations from runtime to an offline preparation stage.

The system includes an explicit tiling and streaming subsystem designed for large-scale spatial datasets. This subsystem is based on uniformed XY tiling and supports the creation of a tiled layer using ST_Tile function in PostGIS. In Unity it is possible to choose between tile-by-tile loading, if the intent is to load entire layer, which is more suitable when loading data in the Editor, or through camera-driven streaming of visible tiles, which is usually needed in Unity Play mode. Figure 6 shows all loading options for a layer that a user can select among when loading data in the Editor. The same logic is available in the Play mode to retrieve data.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026 | © Author(s) 2026. CC BY 4.0 License.

A camera-driven visibility mechanism is implemented to support view-dependent loading of spatial content from PostGIS as explained in 3.1 section, while allowing dynamic content occlusion, which is natively implemented in Unity. On loading all objects are kept separately so interactions such as camera occlusion culling, selection, highlight, hide/show, move, change materials, adding metadata for an individual object are possible.

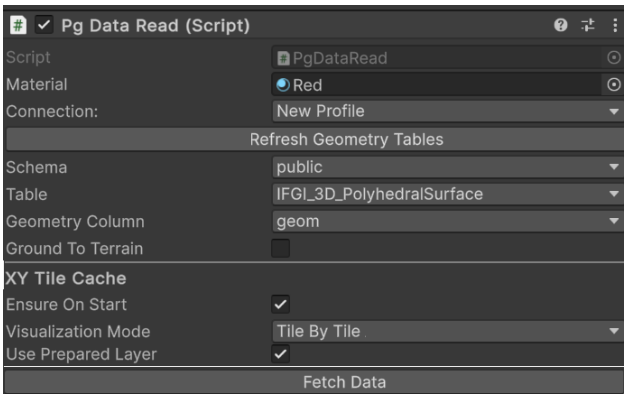

Figure 6. PostGIS layer loading options in Unity Editor

In addition, selected 2D datasets or raster layers can be published to GeoServer automatically and subsequently visualised in Unity as WMS raster terrain overlays. As mentioned in the system design section, this approach is recommended to showcase 2D data over 3D terrain regardless of Cesium terrain resolution.

When it comes to metadata, they are not fetched by default together with the geometry to speed up the visualisation of data in the scene. This does not mean that metadata cannot be fetched for all objects at once. To obtain metadata, user needs to click on an object, in which case a ray is casted in this direction and hit point is determined. Based on the hit point position, ray direction, and selected layer 3D line intersection is performed with the original geometry in PostGIS. A question can be asked why ID values are not simply matched. The reason is that many times layers do not have by default set up unique identifiers (e.g., unique ID column).

Regarding georeferencing of 3D data, raycasting is used, grounding object bottom surface to the terrain. The user can beforehand, rotate and move objects in the scene.

The implemented framework supports the export of Unity scene content into PostGIS through multiple geometry representations, including PointZ, LineStringZ, PolygonZ, and PolyhedralSurface. For mesh-intensive 3D content, the system also supports the generation of an additional Unity-oriented mesh representation stored in database tables, allowing subsequent reloads to bypass repeated reconstruction steps. The export workflow further accommodates material extraction, colour preservation, and merging of coplanar faces to create polyhedralsurfaces with faces that are polygons instead of simply triangles. As a result, the Unity application can function not only as a 3D scene representation tailored to real-time visualisation, but also as a source for exporting 3D georeferenced data to PostGIS.

4. Demonstration

In the following text, the proposed functionalities are tested, investigating the usability of the system in different scenarios. All experiments are conducted on a Windows 11 computer with an Intel Core i7-12700K CPU, 32 GB RAM, and an NVIDIA RTX 4070 GPU.

4.1 Geospatial data visualisation

To only pull visible tiles as explained before the system performs an intersection between the tiles and 3D camera view (Figure 7). The Unity can further omit for visualisation the objects that are not within the 3D view, reducing the number that needs to be rendered and improving the frame rate of the application.

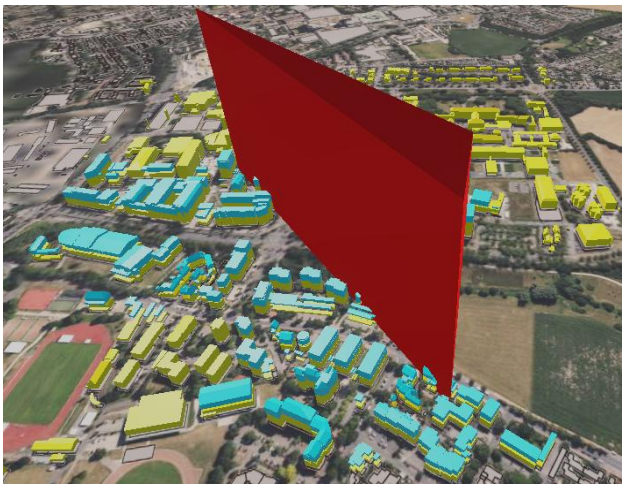

Figure 7. Comparison between visualising all layer tiles (in yellow) and the ones intersect with the 3D camera (in blue). The camera view is visualised in red.

When it comes to 2D data, the system automatically publishes a selected layer or SQL results using the GeoServer capabilities. Figure 8 shows 2D European roads dataset over the Cesium terrain through WMS. It is also possible to retrieve metadata from data by clicking on a specific geometry part.

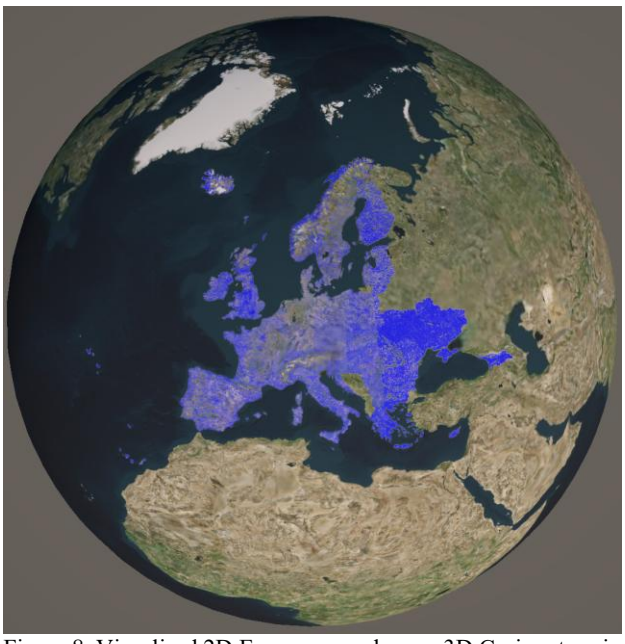

Figure 8. Visualised 2D European roads over 3D Cesium terrain by automatic publishing from PostGIS to Unity via GeoServer

4.2 SQL Execution and LLM support

Apart from layer retrieval, it is possible to write SQL queries directly in Unity, preview results and visualise. Figure 9 shows the execution of a SQL query as well as the visualisation of the results.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026 | © Author(s) 2026. CC BY 4.0 License.

Figure 9. Visualising geospatial data through direct SQL querying in Unity

Regarding the use of LLM and MCP, several queries are explored, investigating the usability of the set-up architecture, and the available helping options to create valid SQL queries. The model used is called *gemma4:e4b*, which is one of the most powerful models suggested by our Unity application while considering the configuration of the computer used.

Prompt 1 (generate SQL and execute + assigned tables): "Retrieve the first 10 rows from `IFGI_PolyhedralSurface` table. Determine how floor elements are represented, using only the schema and those 10 rows. After that, write the final SQL query that returns all first-floor elements."

SQL 1 (42 s): "SELECT * FROM IFGI_PolyhedralSurface WHERE name LIKE 'LEVEL1_%'"

Prompt 2 (generate SQL only + assigned tables): "Return the first 100 trees from the Muenster trees layer, which are 10 meters away from the Muenster LOD2 buildings."

SQL 2 (73 s): "SELECT t.* FROM muenster_trees t
WHERE EXISTS (SELECT 1 FROM muenster_lod2 l WHERE ST_DWithin(t.geom, l.geom, 1)) LIMIT 100;"

Prompt 3 (fixing SQL): "SELECT st_3distance(geometry) FROM building_footprints
Error: 42883: function st_3distance(geometry) does not exist"

SQL3 (0.5 s): "SELECT st_3ddistance(a.geometry, b.geometry) FROM building_footprints a
JOIN building_footprints b ON a.ctid <> b.ctid;"

Prompt 4 (suggest more efficient query): "SELECT t.* FROM muenster_trees t JOIN muenster_lod2 b ON ST_Distance(t.geom, b.geom) <= 10 LIMIT 100;"

SQL4 (60 s): "SELECT t.* FROM muenster_trees t JOIN muenster_lod2 b ON ST_DWithin(t.geom, b.geom, 10) LIMIT 100;"

The SQL queries showed the system can generate appropriate SQL queries and suggest fixes. For some prompts the solution was more difficult to find, requiring more interactions between the LLM, MCP and the database. By specifying/selecting the exact tables the system concentrates only on constructing the SQL, and not on initial database exploration. As always, the quality of the generated SQL depends strongly on how precisely the task is formulated and how much contextual guidance is provided to the model.

4.3 Data writing and retrieval

Table 2 shows the time it takes to overlay a 2D dataset on top a 3D terrain in Unity from preparing PostGIS data, transfer and visualisation. The dataset used for testing is 98280 building footprints covering the area of the city of Muenster. The direct transfer of raster data (i.e. 0.01 MB) resulted in the least transferred data to Unity, while the GeoJSON approach needed to transfer all vector data first to Unity followed by rasterisation, while the other two approach transferred only images. Since GeoServer processes all the data on the fly, the preparation was 0 seconds. Once the data are prepared the raster overlay approach was the quickest to render. However, the GeoServer approach resulted in the quickest approach in total.

Dataset	Transferred data (MB)	Preparation (s)	First tile rendered (s)	Rendering time (s)
GeoJSON overlay	26.48	0.97	0.29	1.3
Raster overlay	**0.01**	2.04	**0.05**	**1.05**
GeoServer WMS	0.5	**0** (live rendering)	0.32	2.16

Table 2. Computational times to render 2D data

Table 3 shows times needed to write, prepare, and read 3D data for two different datasets: Muenster-based 1100 CityGML LOD2 buildings and one 3D building model with 2050 objects. The results show that it takes more time to write data to PostGIS than to prepare or fetch data. The processing of 3D data into Unity readable format can take some time, while the time to visualise data in Unity is usually quick, and 3D models appear in general immediately. It should be indicated that preparing a layer is not a requirement, but it represents a way to save time, which will go in the fetch time, if the layer is not prepared. A comparison between pg2bd3m and our approach is performed, where pg2b3dm has a slight edge in rendering 3D data in Unity, while the data preparation is quicker using our approach. Thus, the full cycle of rendering 3D from PostGIS to Unity resulted being quicker using our approach.

Dataset (approach)	Triangles / objects	Write (s)	Prepare (s)	Fetch (s)
CityGML LOD2 buildings (pg2b3dm)	670250 / 13936	/	13.8	**0.1**
CityGML LOD2 buildings (our approach)	670250 / 13936	4.9	**11.3**	0.3
3D building (pg2b3dm)	1172150 / 2050	/	17.8	**0.4**
3D building (our approach)	1172150 / 2050	10.5	**11.4**	0.5

Table 3. Computational times to write, prepare and fetch 3D data

As explained previously there are several options when exporting geospatial data to PostGIS, allowing to pick the geometry type, CRS, colour and materials, as well as merge coplanar faces. Once the data are stored in PostGIS, they can be visualised back in Unity or in other applications like QGIS.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026 | © Author(s) 2026. CC BY 4.0 License.

Figure 10 shows the differences between these two systems. In QGIS, the data seems flatten a bit compared to how that look in Unity. Also, Unity has much better support for shaders, rendering more realistically the building elements and the trees.

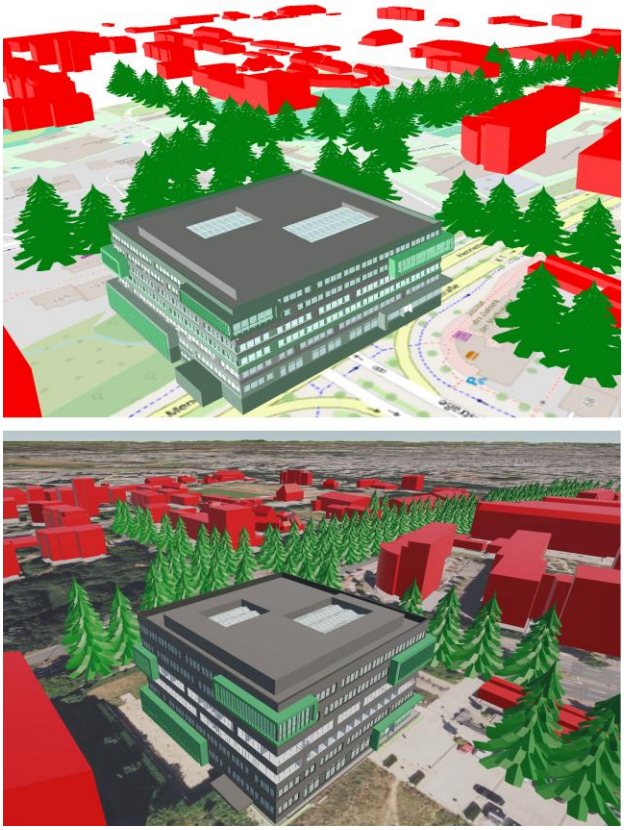

Figure 10. Visualiastion of PostGIS stored 3D data in QGIS (upper image) and Unity (lower image)

As indicated the metadata retrieval is based on raycasting hit point that is possible to obtain when clicking in a direction in Unity and 3D line intersections in PostGIS (Figure 11).

Figure 11. Metadata retrieval using raycasting hit point and PostGIS 3D querying

5. Discussion

The presented framework shows that a direct PostGIS-to-Unity workflow can be a practical alternative to more common server-based pipelines, especially in cases where the application requires immediate access to database content, dynamic querying, selective retrieval, and live interaction. Rather than treating the database only as a storage backend, the implementation demonstrates that PostGIS can operate as an active component for filtering, metadata delivery, and scene preparation, while Unity handles rendering, interaction, and user-specific logic.

One of the main strengths of the proposed approach is the tile-based retrieval strategy. By performing 3D camera–tile intersection, the system loads only the portion of the dataset that is relevant to the current view. At the same time, storing auxiliary tile structures in a separate helper schema is useful from a data-management perspective, because optimisation structures can be created or removed without modifying original datasets.

The reported timings show that visualisation in Unity is fast especially once data are stored. The preprocessing of 3D data required more time to adjust it for visualisation in Unity either by our application or pg2b3dm.

Another important outcome is that the system supports several kinds of interactions, including a metadata retrieval mechanism, the ability to export and fetch from PostGIS. This highlights that the framework is not limited to visualisation alone but can also support lightweight modifications. This is a relevant distinction from many existing engines and viewers that mainly consume already prepared geospatial datasets. Regarding 3D georeferencing, grounding the bottom surface of objects is implemented, while more complex integration with 3D terrain would be useful to support (Yan et al., 2019), since Cesium allows access to the geometry of the terrain tiles. On the other hand, the terrain tiles constantly switch between LODs depending on the camera position, making the process more challenging.

Direct overlay of 2D geometries on a dynamically changing 3D terrain is less reliable due to the Cesium terrain LOD changes, which can cause features to become partially hidden. The use of image overlays, therefore, is the recommended solution. All tested options performed similarly, with a slight edge GeoServer has compared to GeoJSON and raster approach. However, GeoServer introduces an additional component into the workflow.

The LLM–MCP component should also be interpreted carefully. The experiments show clear value in syntax repair, query suggestions, and efficiency-oriented reformulation, but for very complex prompts, the results become incorrect. This suggests that the proposed LLM–MCP workflow is especially useful as an assistive layer for query drafting, debugging, and refinement. The results indicate that the quality of the response improves when the model is constrained through table selection and clearer task framing, which means that the usefulness of the LLM is strongly tied to how well the database context is exposed to it.

From a broader perspective, the proposed solution is best to understand not as a replacement for standards such as 3D Tiles, nor as a substitute for mature GIS platforms, but as a complementary architecture for cases where direct database communication is beneficial. For instance, it can be used for local or research-oriented applications, digital twin prototypes, scene editing environments, and systems where SQL-driven filtering and immediate database access matter. The presented approach offers a more direct and controllable alternative.

6. Conclusion and future work

This paper presented a direct integration between PostGIS and Unity for scalable 2D/3D geospatial data management and visualisation. The proposed framework shows that PostGIS can be used not only as a storage backend, but also as an active runtime component for spatial querying, tile-based data retrieval, metadata access, and georeferenced 3D content management. At the same time, Unity provides the interactive

https://doi.org/10.5194/isprs-annals-XII-4-W1-2026-17-2026

3D environment needed for real-time rendering, user interaction, and scene-based exploration. The demonstrated results indicate that such a database-centred approach can serve as a practical alternative for local, research-oriented, and exploratory applications. The results suggest that visualisation of 3D data is not the main issue and the attention should be placed on quicker processing of PostGIS data into triangles.

The implementation further showed that the combination of tile-based loading, preprocessing of Unity-ready mesh representations, and selective metadata retrieval can support efficient handling of larger datasets while preserving interactive performance. Also, the integration of LLMs and MCP suggests that natural-language assistance can improve accessibility of spatial querying, especially for SQL writing assistance.

A natural next step is a more systematic evaluation of scalability and query quality. The current demonstration confirms feasibility, but future work should benchmark larger datasets under varying scene complexity. For the LLM-supported component, future evaluation should also examine prompt design and error categories in more detail. Such analysis would help clarify when LLM assistance is dependable enough for non-expert use and when expert supervision remains essential. The current integration only investigated the use of open-source LLMs, while the support to more advanced cloud-based models (e.g., ChatGPT and Claude) should be also integrated. Also, models like *gemma4:e4b* are multimodal, supporting image analysis, which is useful for visual validation of SQL results.

Future work should extend the framework toward more advanced management of large and heterogeneous 3D geospatial datasets too. Also, one important direction is the automatic generation of HLODs from 3D data, enabling more efficient rendering of city-level models, something what Cesium for Unity already uses for their terrain. Another relevant extension is the support for point clouds and time-series data, which would broaden the applicability of the appliacation. Further work should also investigate more advanced georeferencing of 3D models based on external spatial datasets, such as alignment with OpenStreetMap 2D building footprints (Diakite and Zlatanova, 2020) and point cloud data, to facilitate better placement and spatial consistency. In addition, future work could investigate stronger support for semantic 3D standards, richer metadata structures, and more seamless handling of bidirectional editing between Unity and PostGIS.

References

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015. Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, *4*(4), 2842-2889. https://doi.org/10.3390/ijgi4042842

Buyuksalih, I., Bayburt, S., Buyuksalih, G., Baskaraca, A. P., Karim, H., Rahman, A. A., 2017. 3D modelling and visualization based on the unity game engine–advantages and challenges. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*, 161-166. https://doi.org/10.5194/isprs-annals-IV-4-W4-161-2017

Diakite, A. A., Zlatanova, S., 2020. Automatic geo-referencing of BIM in GIS environments using building footprints. *Computers, Environment and Urban Systems*, *80*, 101453. https://doi.org/10.1016/j.compenvurbsys.2019.101453

Hou, X., Zhao, Y., Wang, S., Wang, H., 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.1145/3796519

Jeddoub, I., Nys, G. A., Hajji, R., Billen, R., 2023. Digital Twins for cities: Analyzing the gap between concepts and current implementations with a specific focus on data integration. *International Journal of applied earth observation and geoinformation*, *122*, 103440. https://doi.org/10.1016/j.jag.2023.103440

Labetski, A., Kumar, K., Ledoux, H., Stoter, J., 2018. A metadata ADE for CityGML. Open Geospatial Data, Software and Standards, 3, 16. https://doi.org/10.1186/s40965-018-0057-4

Laksono, D., Aditya, T., 2019. Utilizing a game engine for interactive 3D topographic data visualization. *ISPRS International Journal of Geo-Information*, *8*(8), 361.

Lee, A., Chang, Y. S., Jang, I., 2020. Planetary-scale geospatial open platform based on the Unity3D environment. *Sensors*, *20*(20), 5967. https://doi.org/10.3390/s20205967

Liu, M., Wang, X., Xu, J., Lu, H., Tong, Y., 2025. NALSpatial: a natural language interface for spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, *37*(4), 2056-2070. https://doi.org/10.1109/TKDE.2025.3525587

Marnat, L., Gautier, C., Colin, C., Gesquière, G., 2022. Py3DTilers: an open source toolkit for creating and managing 2d/3d geospatial data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *10*, 165-172.

Pierdicca, R., Muralikrishna, N., Tonetto, F., Ghianda, A., 2025. On the use of LLMs for GIS-based spatial analysis. *ISPRS International Journal of Geo-Information*, *14*(10), 401. https://doi.org/10.3390/ijgi14100401

Rantanen, T., Julin, A., Virtanen, J. P., Hyyppä, H., Vaaja, M. T., 2023. Open geospatial data integration in game engine for urban digital twin applications. *ISPRS International Journal of Geo-Information*, *12*(8), 310. https://doi.org/10.3390/ijgi12080310

Shaji, R., Bouveyron, T., Willmes, C., 2025. 3DTiler: A tool to georeference 3D Models and generate 3D Tiles. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *48*, 157-163. https://doi.org/10.5194/isprs-archives-XLVIII-4-W15-2025-157-2025

Subramaniyam, B., Shylesh, D., Ramasamy, J., Kumar, N., 2024. A Virtual Reality Tool for Accuracy Assessment of 3D Models in an Immersive Virtual Environment. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *10*, 333-340. https://doi.org/10.5194/isprs-annals-X-4-2024-333-2024

Yan, J., Zlatanova, S., Aleksandrov, M., Diakite, A. A., Pettit, C., 2019. Integration of 3D objects and terrain for 3D modelling supporting the digital twin. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*, 147-154. https://doi.org/10.5194/isprs-annals-IV-4-W8-147-2019

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T. H., 2018. 3DCityDB-a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, *3*(1), 1-26. https://doi.org/10.1186/s40965-018-0046-7