CityJSON Management Using Multi-Model Graph Database to Support 3D Urban Data Management

Muhammad Syafiq
3D GIS Research Lab, Universiti Teknologi Malaysia, Johor Bahru, Malaysia

Suhaibah Azri[1]
3D GIS Research Lab, Universiti Teknologi Malaysia, Johor Bahru, Malaysia

Uznir Ujang
3D GIS Research Lab, Universiti Teknologi Malaysia, Johor Bahru, Malaysia

Abstract

The prevalence of 3D city models in urban applications is increasing due to their lightweight and flexibility, making them adaptable to various applications. However, effective data interoperability remains an issue. Managing 3D city models within a database can improve urban data management applications such as data enrichment and efficient querying. Motivated by the need for better interoperability of 3D city models, this paper proposes a novel method for storing CityJSON using the concept of a multi-model graph database as a foundation for enriching its semantics. The proposed approach involves decomposing CityJSON objects into smaller JSON components, which are then abstracted into graph elements. Parent-child and other relationship attributes are modelled to capture the hierarchical and associative structures of the CityJSON data. A specific programme is employed to preprocess CityJSON data based on several conditions before being loaded into the graph database. Our multi-model approach allows three types of queries: document, graph, and hybrid. The latter combines both document and graph query. Comparative evaluation against relational databases demonstrates that the proposed method outperforms in terms of query performance. The improved query performance is attributed to the advantage of graph database that reduces the need for joins and the ability to efficiently index and navigate JSON data. The findings of this study establish a foundation for semantic enrichment of 3D city models to improve interoperability and support advanced urban data management.

2012 ACM Subject Classification Information systems → Graph-based database models; Information systems → Geographic information systems; Information systems → Database design and models

Keywords and phrases CityJSON, Graph Database, 3D City Model, 3D GIS, Interoperability

Digital Object Identifier 10.4230/LIPIcs.GIScience.2025.2

Funding This work is supported by the Ministry of Higher Education through the Fundamental Research Grant Scheme (FRGS/1/2022/WAB07/UTM/02/3). The highest appreciation is offered to UTMNexus scholarship for sponsoring the study of the first author.
Suhaibah Azri: Fundamental Research Grant Scheme (FRGS/1/2022/WAB07/UTM/02/3).

1 Introduction

Urban management reliance on 3D city models has grown steadily in recent years, driven by their role in various urban applications such as energy demand modelling, indoor navigation, and sustainability studies [27]. A virtual representation of city and urban data is required for meeting the demands of modern urban applications to enable effective decision-making, efficient resource allocation, and adept strategic planning [28, 10, 19]. 3D city models are

[1] Corresponding author

primarily developed to visualise and represent urban objects [26, 14]. However, they can be stored in a database for advanced querying, analysis, and integration of urban objects with associated urban data.

CityJSON is a 3D city model format encoded in JSON, which serves as an exchange format for CityGML [12]. It carries most of the schema exists in CityGML, allowing it to represent urban objects, such as buildings, bridges, vegetation, and city furniture, in 3D and multiple Level of Details (LoD). CityJSON is also capable of storing attributes based on their data structure. Their data structure adopts a hierarchical and nested design that enables clear parent-child relationships among urban objects. This organisation allows the attributes, geometries, and semantic information to be assigned directly to their corresponding urban objects. However, their structure can get heavily nested where querying information from the whole document would be complex and inefficient. Therefore, decomposing their structure into a more readable and less nested approach is a more intuitive method.

Research into storing 3D city models has explored both relational (RDBMS) and non-relational (NoSQL) databases. In RDBMS, 3D city model components are typically stored in tabular formats, with relationships like parent-child are handled through additional tables. This structure introduces limitations due to the reliance on numerous joins, which reduce efficiency and increase query complexity [16]. RDBMS also lacks native support for hierarchies and inheritance, making it less suited for representing real-world urban objects and their complex relationships [33, 5]. Consequently, relational databases struggle with object-oriented representations and nested structures, leading to inefficiencies in querying 3D city model data [4]. The lack of flexibility to represent 3D urban data in an object-oriented manner is thus open for further research. Moreover, object-oriented approaches have been recognised for effectively modelling complex relationships in 3D GIS, supporting urban applications and enabling detailed structural analyses [33, 21, 11].

NoSQL databases have also been explored as a replacement to address RDBMS limitations, particularly regarding inflexible schema. They are capable of structuring information using object-oriented approach, which allows for modelling hierarchies and inheritance relationships. This capability is highly relevant for managing 3D urban data that involves deep hierarchies and complex information associations [24]. Document-based and graph-based databases have been used to store 3D city models, with data decomposition being a common method of data insertion. CityJSON, which is encoded in the JSON format, further facilitates data insertion into NoSQL databases like MongoDB and ArangoDB as they readily accept data in JSON format. This compatibility reduces the need for data format conversion as CityJSON can be stored directly in its native format. However, storing CityJSON as a whole before unnesting its components is cumbersome as it will further complicate querying and analysis.

This study addresses the limitations of storing CityJSON in relational databases, particularly the challenges of handling its nested structure and object-oriented modelling, by using ArangoDB, a multi-model graph database. CityJSON components are decomposed and stored as documents, which also serve as nodes in a Labelled Property Graph (LPG) structure. Relationships are modelled to reflect parent-child hierarchies and the inheritance of geometry and semantic attributes, while attributes themselves are represented as edges linked to their respective City Objects. This graph-based transformation emphasises semantic decomposition over geometric detail and offers improved query performance compared to relational models.

2 Related Works

CityJSON is a lightweight 3D city model exchange format for CityGML designed to enhance the interoperability of 3D city models. Its JSON-based encoding simplifies storage and parsing compared to the XML-based CityGML format, which is more verbose and often complicated to handle. The lightweight nature of CityJSON makes it preferred by programmers due to reduced complexity when building applications around it [12].

Furthermore, JSON is a human and machine-readable format that simplifies the process of data manipulation and information retrieval. This makes CityJSON more accessible for integration with web applications, APIs, and databases that inherently support JSON. CityJSON widespread compatibility reduces some challenges in its various applicability, which ultimately improves the usability and interoperability of 3D city models across many urban applications. This practical advantage underlines CityJSON as a logical, more intuitive choice to improve the interoperability and utility of 3D city model data.

When storing information in a DBMS, relational databases are typically the first choice following their widespread use and more functionality. 3DCityDB [29] is a relational database schema designed specifically to store OGC-standard 3D city models. It is built based on spatially enhanced relational databases of either PostgreSQL with PostGIS extension or Oracle Spatial. 3DCityDB provides several functionalities including storing, managing, visualising, analysing, and exporting 3D city models in CityGML format. The initial development of 3DCityDB did not offer support for CityJSON; however, subsequent developments introduced the capability to import and export CityJSON. Another relational database solution to store CityJSON is CityREST [13], which is a RESTful API designed to stream CityJSON datasets over the web. It is built on top of PostgreSQL, which offers several key functionalities like retrieving city objects, filtering city objects within a specified bounding box, and data filtering. A more recent approach is CJDB, which is a relational database schema designed for storing CityJSON built on top of PostgreSQL. It is developed as a more efficient alternative to the 3DCityDB schema for storing and managing CityJSON data. Unlike 3DCityDB, CJDB significantly reduces the large number of tables required to store similar datasets. This design simplifies data management, which in turn reduces memory usage [17].

Concerns have been raised by [1] towards the unsuitability of RDBMS for storing OGC standard data models due to the risk of impedance mismatch. This issue arises when attempting to map object-oriented data models into relational schemas can potentially lead to the loss of critical information or relationships. Despite its extended capabilities, ORDBMS still depends on table joins, making it less suited for modelling the hierarchical structure of 3D city models. For instance, representing parent-child relationships requires additional tables and duplicating City Objects as foreign keys, which increases complexity and reduces performance. In contrast, LPG-based graph databases handle such hierarchies more intuitively by directly linking nodes without duplication, offering better node reusability and efficiency. Moreover, storing and querying JSON data in relational databases involves specialised operations that degrade performance as data size and complexity grow. Furthermore, the storage and querying of JSON-based data in relational databases require specialised operations. These operations may impact query performance, particularly as the complexity and size of the data increase.

Existing solutions point towards the use of NoSQL databases that are object-oriented to store information and relationships more efficiently than relational database that lacks the schema and flexibility to represent a real-world entity [8]. The process of locating object-oriented data in the tabular format relevant to relational databases can be difficult and

prone to misrepresentation of information [15]. This limitation arises because of relational databases that are not designed to manage hierarchical, nested data structures typically found in object-oriented data like CityJSON. As a result, alternative approaches have been developed to address the challenges of storing CityJSON in NoSQL databases.

Furthermore, NoSQL databases possess a flexible schema and the ability to naturally model hierarchies and complex relationships. It provides a more intuitive and efficient solution for managing the intricate structure of CityJSON data. Document and graph-oriented NoSQL databases have been explored as alternatives to CityJSON to address the limitations of relational databases in handling object-oriented data. Both MongoDB document and RDF-based graph databases have been widely utilised for this purpose. MongoDB was explored by Nys and Billen [16] and Karin et al. [22] for storing CityJSON, with evaluations comparing its performance with PostgreSQL. Nys and Billen [16] proposed a simplified schema for a document database where CityJSON components are decomposed into first-order or discriminated schemas. Their work also included a visualisation framework built using a MERN stack API architecture. Meanwhile, Karin et al. [22] focused on the querying capabilities of MongoDB to compare the API querying of CityJSON data via GraphQL against PostgreSQL. Both studies found that the querying performance of CityJSON data using MongoDB is promising. This advantage is attributed to MongoDB reduced reliance on tables and joins compared to relational database, resulting in a simpler and more efficient query of CityJSON components.

Akin and Cömert [2] developed a converter that maps CityJSON components into RDF triples by using CJIO to transform CityJSON into a DataFrame, which is then translated into RDF triples in Neo4j. While their work demonstrates the potential of graph databases for storing CityJSON, it lacks evaluation or comparison with other databases, leaving the practical effectiveness of RDF for this purpose underexplored. RDF triples, structured rigidly as subject-predicate-object, are less suited for representing the object-oriented and nested nature of 3D city models. RDF also struggles with the semantic richness and hierarchical depth of 3D city models due to its lack of internal node structure. Attributes must be expressed as additional triples rather than embedded directly into nodes, resulting in a more complex and verbose graph structure. This limitation hampers the representation of semantically rich data, making RDF less ideal for dynamic urban applications [7]. In contrast, LPG allows attributes to be directly embedded in nodes and edges, offering a more flexible and expressive approach for preserving and enriching the semantics of 3D city models. Additionally, RDF requires joins between triples for deep graph traversals, which increases query complexity [20], whereas LPG supports native and efficient traversal operations, thus enhancing performance for complex queries.

LPG graphs models consist of fundamental graph elements (i.e., nodes and edges), which can be enriched with key-value properties. The ability to annotate both nodes and edges enhances the expressiveness of the graph, allowing it to capture complex, object-oriented structures more naturally. In this study, LPG graph is utilised to model and store the nested structure of CityJSON by taking advantage of its ability to annotate both nodes and edges with properties. This flexibility allows information to be stored contextually where descriptive properties of each JSON object are embedded as node attributes, while associative attributes like those linked to City Objects are captured through edge attributes. Each JSON object is represented as an individual node, enabling a clear and expressive mapping of the hierarchical and semantic relationships inherent in CityJSON data.

3 Methodology

Although RDF-based graphs have been explored for CityJSON data management, we argue that LPGs are better suited for managing 3D city model data. As 3D city models are developed to improve interoperability across diverse applications, storing them as attributed nodes and edges within an LPG structure is more appropriate since it allows better expressiveness of information compared to RDF, particularly as LPG graphs are built on a key-value pair [30]. This approach enables the seamless association of attributes relevant to various urban applications that facilitate the semantic enrichment of 3D city models.

Junxiang et al. [32] have raised concerns about the limitations of RDF triples for storing and querying building information data. Specifically, RDF graphs and their query languages lack efficiency to support graph traversals, which poses challenges for graph querying and analysis. As a result, RDF triples often must be converted into LPG graphs to enable scalable graph analytics and fully capture complex semantic relationships [30, 3, 18, 9]. Furthermore, LPG graphs simplify data integration compared to RDF graphs [31]. This makes LPGs a more effective choice for managing 3D city models that support semantic complexity and allow efficient graph traversals and analytics. Therefore, this study aims to develop a schema model for storing CityJSON in ArangoDB, a multi-model graph database that supports the LPG graph structure.

3.1 Schema for Multi-Model Graph Database

Our approach focuses on managing 3D urban data using a multi-model graph database. The structure of an LPG-based graph database is considered a more intuitive approach compared to RDF-based graph databases and relational databases for handling complex 3D urban data.

3D urban data are characterised by their complex structure, nested relationships, and semantically rich attributes. Representing such data using an object-oriented approach is more suitable, which can be achieved effectively with an LPG-based property graph. LPG structures mirror real-world object relationships more naturally, making them ideal for 3D spatial data where objects like buildings, geometry, and attributes can be abstracted through graph elements.

In ArangoDB, records are stored in JSON format as documents in document collections, while relationships between records are established and stored in edge collections where each edge references the unique key of the connected records. This design essentially treats each record in the document collection as a node, whereas the edges can be connected between nodes to represent their relationships, hence the multi-model nature of ArangoDB. Storing an entire CityJSON document as a single entity is possible in ArangoDB; however, querying and retrieving specific information can become complex due to the deeply nested structure of CityJSON components. Filtering data requires the query process to traverse the deep hierarchical levels of the CityJSON document, which can be inefficient and time-consuming. To address this issue, it is necessary to decompose CityJSON files into distinct components and store them as separate documents within a document collection.

Therefore, we propose a schema for multi-model graph database to store CityJSON based on three collections, one document collection, and two edge collections (see Figure 1). The document collection gathers all the decomposed CityJSON components as individual documents. First-level objects, such as City Objects, are stored as separate documents, while second-level objects, including geometries nested within City Objects, and third-level objects, like the semantics nested within geometry, are further decomposed and stored as individual

documents. Additionally, each City Object ID is stored as a document to explicitly associate City Objects with their attributes and facilitate parent-child relationships. The content of each decomposed document is shown in Figure 2.

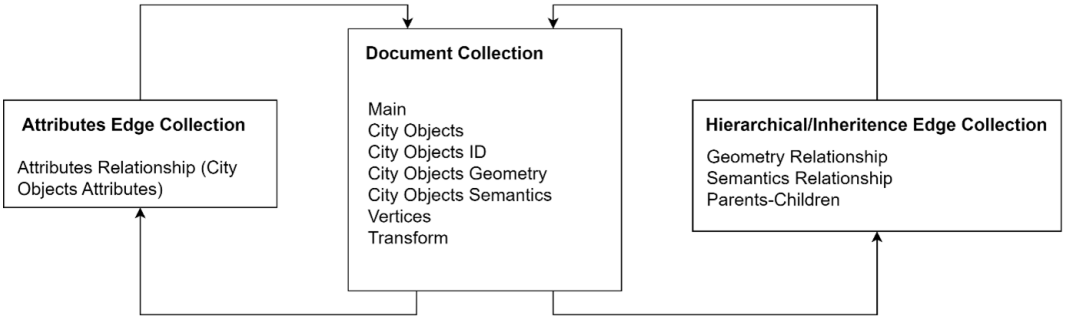


Figure 1 CityJSON Multi-Model Graph Database Schema.

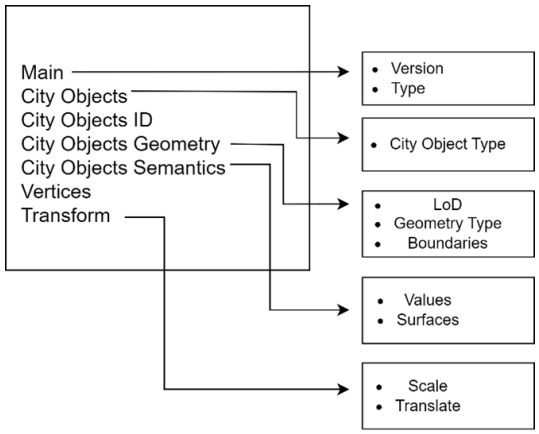


Figure 2 Components of Decomposed CityJSON Documents.

Two edge collections are used to model relationships. The first edge collection links City Object documents to their City Object IDs and stores attributes as edge attributes. Attributes originally stored within the "Attributes" key of each City Object in the original CityJSON file are separated and mapped as key-value pairs in the edge collection. This separation of attributes aims to avoid the attributes being kept nested under their City Objects. It ensures that attributes can be queried more efficiently, and new attributes can be added or modified using the basic insert and update database operations. In the case where the City Objects do not contain any attributes, the relationship between the City Objects document and each City Object ID will still be established. This ensures that any future attributes relevant to any City Objects can be inserted.

The second edge collection models the relationships between geometry, semantics, and their corresponding City Objects alongside the parent-child hierarchies among City Objects. Figure 3 illustrates the representation of the relationship between the decomposed components of CityJSON.

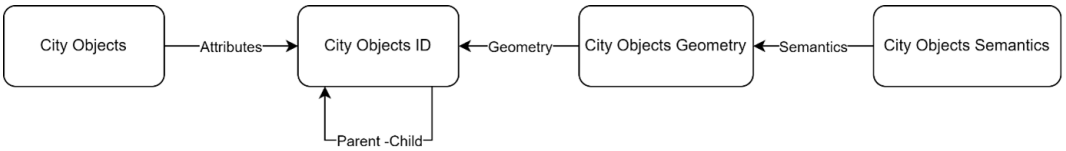


Figure 3 Relationship Representation between Decomposed CityJSON Components.

3.2 Storing CityJSON into Multi-Model Database

A program is developed to store a CityJSON file into ArangoDB based on the schema explained in Section 3.1. The programme workflow is shown in Figure 4.

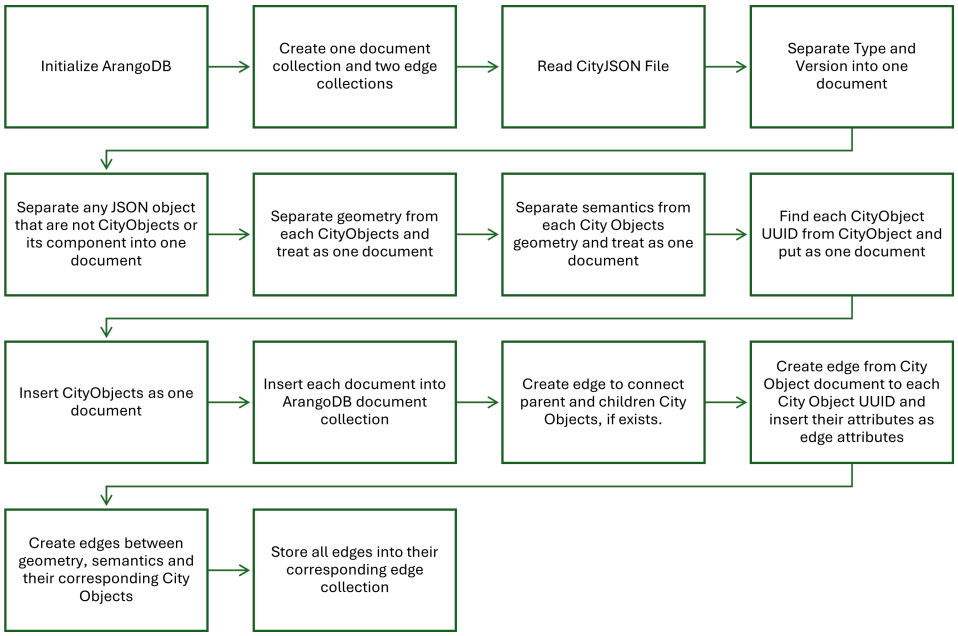


Figure 4 CityJSON to LPG Graph Workflow.

The integration of CityJSON into the graph database process begins by setting up the ArangoDB environment, a graph database to store and manage CityJSON data. First, the ArangoDB graph is initialised and a dedicated database is created. Within this database, a document collection is established to store the CityJSON components. Edge collections are created to represent the relationships between these components. One edge collection is for connecting CityJSON components with their parent-child relationship and other inheritance relationship, while another edge collection is created to store CityObject attributes.

The workflow then reads the CityJSON file where the content will be decomposed into individual components with smaller and manageable JSON scripts containing the decomposed JSON objects. The code will analyse the file to identify and separate each CityJSON component with the exception of CityObjects and its components, which will be processed at a later stage. The type and version components are stored as one document (Main document, see Figure 1), which is named based on the file name of the input CityJSON file. Each of these identified components is then inserted as individual documents into the document collection, which has been created at the database initialisation step.

Next, the workflow proceeds to parse the City Objects and their components. The City Object components will be analysed and decomposed based on the following considerations:

1. Each CityObject is examined to determine its individual ID (CityObjectUUID). For each City Object ID, a document is created and inserted into the document collection.
2. The hierarchical structure of CityObjects is addressed by examining the parent-child relationships among CityObjects. If such relationships exist, edges are created to represent this relationship.
3. If a CityObject contains geometry information, the geometry is separated and stored in a dedicated document and later kept in the document collection. Edges are created to link the geometry data to its respective CityObjects.
4. If the geometry of the CityObject contains semantic information, the semantic is separated and stored in a dedicated document and kept in the document collection. Edges are created to link the semantic information to its respective geometry documents.
5. Edges are created between CityObject and all CityObject UUID and stored in the attribute edge collection regardless of whether the CityObject has attributes or not. If the CityObject contains attributes, it is inserted as edge attributes; otherwise, the edge will act as a placeholder for future attributes.

At the end of the workflow, all CityJSON components (objects, semantics, and geometry) and their attributes are stored as structured documents in the database. Relationships between these components are encoded as edges, making it possible to query and analyse the data using graph-based operations. The attributes of City Objects are stored inside a dedicated collection, which allows users to dynamically include any information regarding the CityObjects pertinent to any applications in the future. The integration process transforms the CityJSON data into a form that is highly suitable for advanced applications like urban data management and semantic querying, thus transforming urban management decision-making towards knowledge-driven initiatives.

4 Results and Analysis

4.1 CityJSON as Graph

A graph-based representation of the CityJSON data can be constructed by adhering to the workflow for storing CityJSON data in ArangoDB as outlined in Section 3.2 and structured according to the schema described in Section 3.1. For implementation and evaluation, we use three tiles of CityJSON data retrieved from the 3DBAG website [2]. The dataset includes multiple LoD, resulting in multiple geometries and corresponding semantics for each LoD.

The graph in Figure 5 shows the CityJSON data structure based on the components and relationships outlined in Figure 3 where the City Object documents serve as central nodes and all City Objects ID converge. It visualises how the decomposed components are interconnected and captures the hierarchical parent-child relationships among the City Objects as well as the inheritance of semantics and geometry back to their corresponding City Objects. The nodes represent the main JSON objects in CityJSON. Other information like LoD is represented as queryable attributes inside the nodes. Additionally, the graph is capable of illustrating the multiple geometries and semantics associated with each LoD, providing a comprehensive view of the structure of the original CityJSON dataset and its relationships.

[2] `https://3dbag.nl`

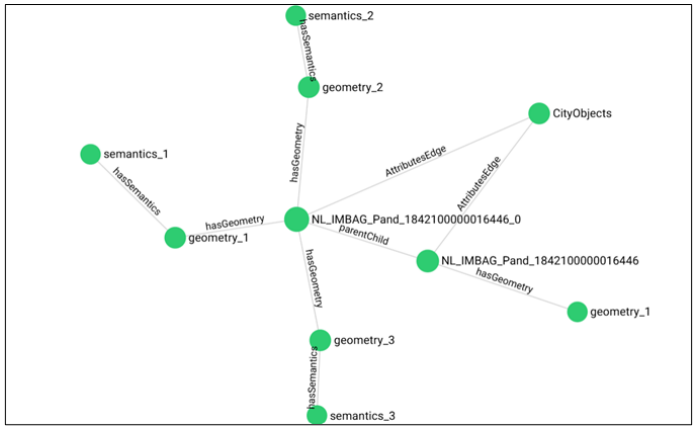


Figure 5 Representation of CityJSON Structure as Graph.

4.2 Evaluation against Relational Database

Our approach is evaluated against PostgreSQL based on the CJDB schema [17]. It uses three tables to store the CityJSON components and their relationship. The first table (`city_object`) stores the City Objects and the information describing the City Objects. The second table (`cj_metadata`) stores the information of the imported CityJSON file. Finally, the third table (`city_object_relationship`) stores the relationship between City Object, such as the parent-child relationship. The CJDB schema is shown in Figure 6.

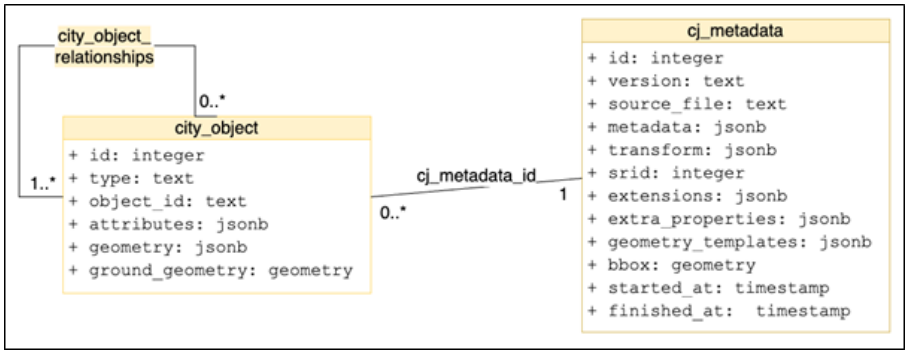


Figure 6 CJDB Schema.

As ArangoDB is a multi-model database, we evaluate our approach according to document-based query, graph-based query, and hybrid query (combination of document-based and graph-based query). It involves using three tiles from the CityJSON dataset retrieved from the 3DBAG website. Table 1 contains a description of the CityJSON datasets used for the implementation and evaluation of our approach, while Figure 7 illustrates the visualisation of the datasets using Ninja CityJSON viewer [3] [25].

The data is a multiple LoD data based on the improved LoD specification by [6]. Table 2 explains the query and the query type for the evaluation of our approach.

[3] `https://ninja.cityjson.org/`

Table 1 CityJSON Datasets Retrieved from 3DBAG Website.

Dataset	File Size (Kb)	Number of City Objects	LoDs
F-8-264-552	1409	396	LoD0, LoD1.2, LoD2.2, LoD2.3
E-10-278-556	2954	892	LoD0, LoD1.2, LoD2.2, LoD2.3
G-8-328-528	19868	6966	LoD0, LoD1.2, LoD2.2, LoD2.3

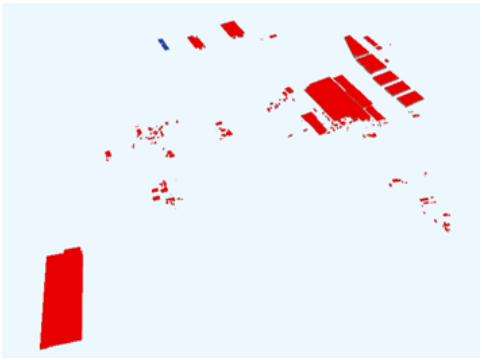
Table 2 Queries for Benchmarking with PostgreSQL.

No.	Query	Query Type
Q1	Query all City Objects with "Building" type	Document
Q2	Query all LoD 1.2 City Objects	Document
Q3	Query all City Objects with slanted roof	Graph
Q4	Query City Objects with specific child	Graph
Q5	Insert "owner" attributes for all City Objects with "Building" type	Hybrid
Q6	Delete "owner" attributes for all City Objects with "Building" type	Hybrid

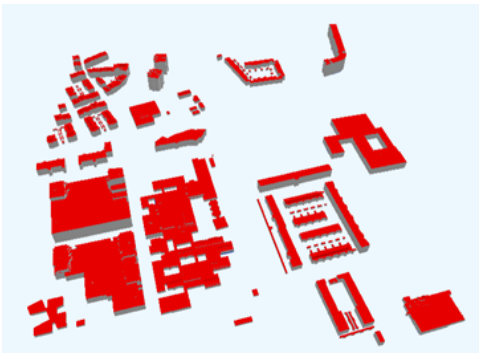
All benchmarks were conducted on a machine running on AMD Ryzen 5 CPU, 16 GB RAM, and a 516 GB SSD. ArangoDB 10.1 and PostgreSQL v17 with PostGIS extension were used. Each query was executed three times and the average execution time was recorded. All benchmarks were performed under warm cache conditions.

Query 1 and Query 2 are document-based query. The City Objects' type is stored within City Object documents in the document collection, whereas the LoD is stored in geometry document. In Q3, graph-based querying is employed to identify City Objects with slanted roofs by querying the edge attributes between City Objects document and its City Object ID. Similarly, Q4 leverages graph traversal to retrieve buildings with specific child elements by querying the hierarchical relationship between City Object ID. Q5 and Q6 are document and graph queries, respectively. The City Objects type is stored in the City Objects document, while the attributes are stored as edge in the attributes edge collection. Therefore, both Q5 and Q6 must navigate the elements in the document and graph structures to complete the query. The query performance comparison is visualised in Figure 8.

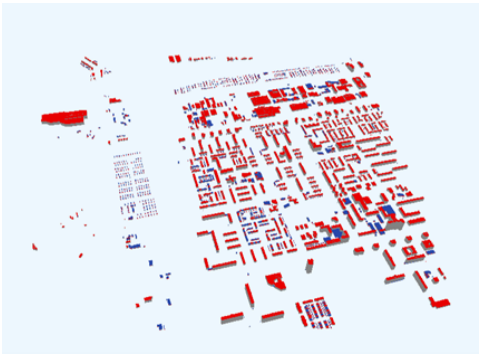
The evaluation shows that our multi-model schema excels in most evaluation cases than PostgreSQL based on the CJDB schema. ArangoDB stores information natively in JSON format, while CJDB stores CityJSON information in JSONB format. Although both are stored in document-based format, ArangoDB, which is purposely built as a multi-model database that natively supports document-based data, is inherently more efficient for querying such data compared to PostgreSQL, which relies on its table-based schema to manage JSONB. This advantage is evident in Q1 and Q2 where ArangoDB achieves significantly lower execution times for querying document-based data. Additionally, ArangoDB stores information as a single document, while PostgreSQL may rely on The Oversized-Attribute Storage Technique (TOAST) table to store oversized data. This will introduce additional overhead as join operation with the TOAST table is needed when accessing oversized documents. The design advantage enables ArangoDB to store and retrieve large CityJSON documents more efficiently, making it better suited for querying large datasets.



(a) F-8-264-552.



(b) E-10-278-556.

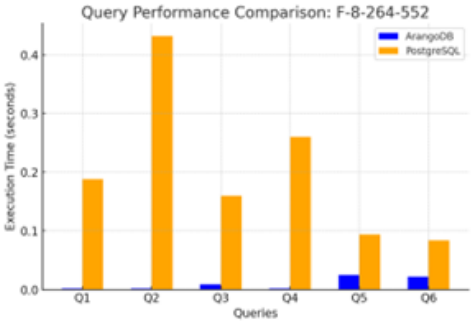


(c) G-8-328-528.

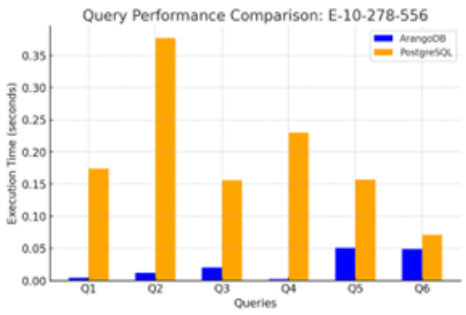
Figure 7 Visualization of each Evaluation Dataset.

Q3 in ArangoDB is a graph traversal operation to retrieve City Objects attributes stored as edge attributes. The edge points towards City Object ID, which is the object that the attributes belong to. Edge attributes that are relevant to LPG make filtering more efficient since the query interacts directly with the graph structure. On the other hand, relational database relies on parsing the JSONB column and filtering its key-value pair based on JSONB operators. This approach requires a scan or index-based lookup of the JSONB column, which is computationally more expensive than edge filtering in ArangoDB. When compared to RDF, they do not natively support attributes on edges and require workarounds like reification, which will further complicating queries [30].

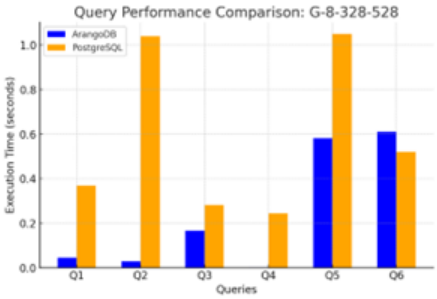
Meanwhile, Q4 is also a graph traversal operation to navigate the parent-child relationships between City Objects. ArangoDB handles this straightforwardly by establishing a relationship between a parent and their children. Meanwhile, PostgreSQL handles this by establishing a new table to join parent and children. Therefore, the join operation is unavoidable for PostgreSQL to query parent-child relationships, which usually will result in additional computational overhead than graph traversals. This is evident by the execution time shown in Table 3. Compared to RDF-graph, LPG-based graph databases are better suited for this purpose than RDF triples as the cost of graph traversals in RDF is higher compared to LPG [3]. RDF may require joining of multiple triples to accomplish deep graph traversal operations. Therefore, traversing the deep hierarchical relationship between parent and child is more efficient using LPG.



(a) F-8-264-552.



(b) E-10-278-556.



(c) G-8-328-528.

Figure 8 Query Performance Comparison for each Dataset.

Q5 and Q6 in ArangoDB are hybrid queries because they involve accessing data stored in both the document collection and the edge collections. The City Object types are filtered in the document collection, while the attributes are stored as edge attributes in the graph structure. This dual-querying process introduces overhead following the need to navigate multiple data structures and perform cross-collection traversals. The overhead is particularly noticeable in Q6 (Delete operation) for the G-8-328-528 dataset where the deletion process required slightly more time than PostgreSQL. This can be attributed to the larger dataset size, which increases the complexity of traversing and modifying edge attributes after initial document filtering. While ArangoDB handles graph traversals efficiently, the combination of document filtering and edge modification introduces additional steps that slightly affect performance in large datasets. In contrast, PostgreSQL manages the deletion operation more efficiently in this specific case due to optimised JSONB operations for direct attribute modification. However, performance advantage in Q6 is limited to this specific scenario as the overall querying process still suffers from complex joins and schema rigidity in other query types. Future work could explore strategies to optimise hybrid queries in ArangoDB, such as pre-indexing relationships or implementing batch processing techniques to reduce the traversal overhead in large datasets. Despite the overhead observed in hybrid queries, ArangoDB maintains superior performance in most cases, particularly in queries involving complex relationships and semantic enrichment.

5 Conclusion and Future Works

This study adopts an object-oriented approach to abstract urban components and their relationships as graph elements using a multi-model graph database. CityJSON is decomposed into individual JSON scripts, which are stored as document nodes and linked via unique keys. Two edge collections are used: one connects City Object documents to their IDs for attribute storage while the other captures parent-child and semantic-geometry relationships. Queries are executed using document-based, graph-based, and hybrid approaches, which show better performance compared to PostgreSQL based on the CJDB schema. This demonstrates the scalability and flexibility of the proposed method.

The strength of our approach lies in the reusability of City Object nodes. Enrichment of attributes can be achieved by modifying or updating the edge attributes relevant to the City Objects to allow better expressivity. Furthermore, relationships can be established without necessitating node duplication owing to the node reusability of City Objects. This requires no joins of tables and allows a better query through graph operations, which have been demonstrated to be more time-efficient compared to query on relational model.

Representing 3D city models through object-oriented abstraction simplifies their complexity by reducing them into manageable, modular structures. Each component is treated independently, allowing flexible storage, updating, and querying. Semantic enrichment is supported by attaching attributes to nodes and edges, while topological relationships can be modelled via edges, thus enabling spatial queries [33], [23]. Future work can extend this approach to support spatial queries, including bounding box operations essential for location-based urban applications. This involves computing bounding boxes for all City Objects and storing them in the database, further enhancing spatial query capabilities for 3D city models.

References

1 Amgad Agoub, Felix Kunde, and MARTIN Kada. Potential of graph databases in representing and enriching standardized geodata. *Tagungsband der*, 36:208–216, 2016. URL: https://www.researchgate.net/profile/Felix_Kunde/publication/305701542_Potential_of_Graph_Databases_in_Representing_and_Enriching_Standardized_Geodata/links/579a93ea08ae2e0b31b1591a/Potential-of-Graph-Databases-in-Representing-and-Enriching-Standardized-G.

2 A T Akın and Ç Cömert. "CITYJSON2RDF" A Converter for Producing 3D City Knowledge Graphs. In Isikdag U. and Bayram B., editors, *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, volume 48, pages 15–20, KTU, Engineering Faculty, Trabzon, 61080, Turkey, 2024. International Society for Photogrammetry and Remote Sensing. doi:10.5194/isprs-archives-XLVIII-4-W9-2024-15-2024.

3 Alex Johannes Albertus Donkers, Dujuan Yang, and Nico Baken. Linked data for smart homes: Comparing RDF and labeled property graphs. *CEUR Workshop Proceedings*, 2636:23–36, 2020. URL: https://ceur-ws.org/Vol-2636/02paper.pdf.

4 Suhaibah Azri, Francois Anton, Uznir Ujang, Darka Mioc, and Alias A Rahman. *Crisp Clustering Algorithm for 3D Geospatial Vector Data Quantization*, pages 71–85. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-12181-9_5.

5 Suhaibah Azri., Uznir Ujang., F. Anton, D. Mioc, and A. A. Rahman. 3D nearest neighbour search using a clustered hierarchical tree structure. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 41(July):87–93, 2016. doi:10.5194/isprsarchives-XLI-B2-87-2016.

6 Filip Biljecki, Hugo Ledoux, and Jantien Stoter. An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, 59:25–37, 2016. doi:10.1016/j.compenvurbsys.2016.04.005.

7 Valeriy Chernenkiy, Yuriy Gapanyuk, Anatoly Nardid, Maria Skvortsova, Anton Gushcha, Yuriy Fedorenko, and Richard Picking. Using the metagraph approach for addressing rdf knowledge representation limitations. In *2017 Internet technologies and applications (ITA)*, pages 47–52. IEEE, 2017.

8 Linfang Ding, Guohui Xiao, Albulen Pano, Mattia Fumagalli, Dongsheng Chen, Yu Feng, Diego Calvanese, Hongchao Fan, and Liqiu Meng. Integrating 3D city data through knowledge graphs. *Geo-Spatial Information Science*, pages 1–31, 2024. doi:10.1080/10095020.2024.2337360.

9 A. E.Hadi Hor, G. Sohn, P. Claudio, M. Jadidi, and A. Afnan. A semantic graph database for BIM-GIS integrated information model for an intelligent urban mobility web application. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4(4):89–96, 2018. doi:10.5194/isprs-annals-IV-4-89-2018.

10 Mohamad Yusoff Izham, Ujang Muhamad Uznir, Abdul Rahman Alias, Katimon Ayob, and Ismail Wan Ruslan. Influence of georeference for saturated excess overland flow modelling using 3D volumetric soft geo-objects. *Computers and Geosciences*, 37(4):598–609, 2011. doi:10.1016/j.cageo.2010.05.013.

11 Noraidah Keling, Izham Mohamad Yusoff, Habibah Lateh, and Uznir Ujang. *Highly Efficient Computer Oriented Octree Data Structure and Neighbours Search in 3D GIS*, pages 285–303. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-25691-7_16.

12 Hugo Ledoux, Ken Arroyo Ohori, Kavisha Kumar, Balázs Dukai, Anna Labetski, and Stelios Vitalis. CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1), 2019. doi:10.1186/s40965-019-0064-0.

13 Xiaoai Li. CityREST: CityJSON in A Database + RESTful Access, 2021.

14 Zulaikha Hana Mohd, Uznir Ujang, and Tan Liat Choon. Heritage house maintenance using 3D city model application domain extension approach. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 42(4W6):73–76, 2017. doi:10.5194/isprs-archives-XLII-4-W6-73-2017.

15 Billy Montolalu, Siti Rochimah, and Daniel Siahaan. Sql and nosql object-database mapping using property graphs in relational cases. In *2024 11th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 515–520. IEEE, 2024.

16 Gilles Antoine Nys and Roland Billen. From consistency to flexibility: A simplified database schema for the management of CityJSON 3D city models. *Transactions in GIS*, 25(6):3048–3066, 2021. doi:10.1111/tgis.12807.

17 Leon Powałka, Chris Poon, Yitong Xia, Siebren Meines, Lan Yan, Yuduan Cai, Gina Stavropoulou, Balázs Dukai, and Hugo Ledoux. cjdb: A Simple, Fast, and Lean Database Solution for the CityGML Data Model. *Lecture Notes in Geoinformation and Cartography*, pages 781–796, 2024. doi:10.1007/978-3-031-43699-4_47.

18 Sumit Purohit, Nhuy Van, and George Chin. Semantic Property Graph for Scalable Knowledge Graph Analytics. *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, pages 2672–2677, 2021. doi:10.1109/BigData52589.2021.9671547.

19 Nurfairunnajiha Ridzuan, Uznir Ujang, Suhaibah Azri, and Tan Liat Choon. Visualising urban air quality using AERMOD, CALPUFF and CFD models: A critical review. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 44(4/W3):355–363, 2020. doi:10.5194/isprs-archives-XLIV-4-W3-2020-355-2020.

20 Awny Sayed and Amal Almaqrashi. Scalable and Efficient Self-Join Processing technique in RDF data. *arXiv preprint arXiv:1409.4507*, 11:43–50, 2014. arXiv:1409.4507.

21 Wenzhong Shi, Bisheng Yang, and Qingquan Li. An object-oriented data model for complex objects in three-dimensional geographical information systems. *International Journal of Geographical Information Science*, 17(5):411–430, 2003. doi:10.1080/1365881031000086974.

22 Karin Staring, Stelios Vitalis, Linda Brink, and Balazs Dukai. Combination of cityjson with postgresql, mongodb and graphql. Master's thesis, Delft University of Technology, 2020.

23 Muhammad Syafiq, Suhaibah Azri, and Uznir Ujang. Navigating Immovable Assets: A Graph-Based Spatio-Temporal Data Model for Effective Information Management. *ISPRS International Journal of Geo-Information*, 13(9):313, 2024. doi:10.3390/ijgi13090313.

24 Uznir Ujang, Francesc Anton Castro, and Suhaibah Azri. Abstract topological data structure for 3D spatial objects. *ISPRS International Journal of Geo-Information*, 8(3), 2019. doi:10.3390/ijgi8030102.

25 S. Vitalis, A. Labetski, F. Boersma, F. Dahle, X. Li, K. Arroyo Ohori, H. Ledoux, and J. Stoter. Cityjson + Web = Ninja. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VI-4/W1-20(September):167–173, 2020. doi:10.5194/isprs-annals-vi-4-w1-2020-167-2020.

26 Jochen Wendel, Alexander Simons, Alexandru Nichersu, and Syed Monjur Murshed. Rapid development of semantic 3D city models for urban energy analysis based on free and open data sources and software. *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics, UrbanGIS 2017*, 2017-Janua, 2017. doi:10.1145/3152178.3152193.

27 Nevil Wickramathilaka, Uznir Ujang, Suhaibah Azri, and Tan Liat Choon. Influence of Urban Green Spaces on Road Traffic Noise Levels: - a Review. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 48(4/W3-2022):195–201, 2022. doi:10.5194/isprs-archives-XLVIII-4-W3-2022-195-2022.

28 Bruno Willenborg, Maximilian Sindram, and Thomas H. Kolbe. Applications of 3D City Models for a Better Understanding of the Built Environment. In Martin Behnisch and Gotthard Meinel, editors, *Trends in spatial analysis and modelling: decision-support and planning strategies*, Geotechnologies and the Environment, pages 167–191. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-52522-8.

29 Zhihang Yao, Claus Nagel, Felix Kunde, György Hudra, Philipp Willkomm, Andreas Donaubauer, Thomas Adolphi, and Thomas H. Kolbe. 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), 2018. doi:10.1186/s40965-018-0046-7.

30 Zhanfang Zhao, Sung Kook Han, and Ju Ri Kim. LPG representation of the reification of RDF. *International Journal of Engineering and Technology(UAE)*, 7(3.34 Special Issue 34):562–566, 2018. doi:10.14419/ijet.v7i3.34.19382.

31 Junxiang Zhu, Heap Yih Chong, Hongwei Zhao, Jeremy Wu, Yi Tan, and Honglei Xu. The Application of Graph in BIM/GIS Integration. *Buildings*, 12(12), 2022. doi:10.3390/buildings12122162.

32 Junxiang Zhu, Peng Wu, and Xiang Lei. IFC-graph for facilitating building information access and query. *Automation in Construction*, 148, 2023. doi:10.1016/j.autcon.2023.104778.

33 Siyka Zlatanova, Alias Abdul Rahman, and Wenzhong Shi. Topological models and frameworks for 3D spatial objects. *Computers and Geosciences*, 30(4):419–428, 2004. doi:10.1016/j.cageo.2003.06.004.