Extending the 3D City Database 5.0 to support CityGML application in QGIS

Bing-Shiuan Tsai
student #5511461

1st supervisor: Giorgio Agugiaro
2nd supervisor: Camilo León Sánchez
External supervisors: Claus Nagel, Zhihang Yao

January 24, 2024

1 Introduction

Semantic 3D city models are commonly used for data visualisation and analyses in the realm of the built environment. Storage and management of semantic 3D city models data can be achieved in CityGML, which is an international standard adopted by the Open Geospatial Consortium (OGC).

Handling large amounts of data calls for a database encoding. For CityGML, there is a database encoding called the 3D City database (3DCityDB), an open source project developed for PostgreSQL and Oracle databases. The 3DCityDB in use is the version of 4.X, and it is derived from the CityGML v.2.0 conceptual model. Except for data storage, the database allows for data accessibility, enabling users to convert the city object associations to relations between predefined feature tables based on a set of mapping rules consistent to CityGML2.0 standards.

Although the 3DCityDB tool was developed to simplify the complexity of CityGML, its structure still remains rather complex for users with basic structured query language (SQL) skills. However, a plugin for QGIS, called "3DCityDB-Tools for QGIS", has been recently developed to facilitate the interaction with the 3DCityDB for a wider group of practitioners, and therefore to expand the usability of CityGML, narrowing the gap between them and the geo-information experts via an intuitive graphical user interface (GUI) in QGIS.

In the meanwhile, the OGC has published the CityGML v.3.0 standard in September 2021 [Kolbe et al., 2021], which introduces an improved conceptual data model and new modules such as logical space and physical space to address the spatial characteristics and the support of time-dependent IoT data, etc. The 3DCityDB is being updated to version 5.0 in order to add full support for CityGML v.3.0.

The goal of this research is to investigate how the new database structure of the 3DCityDB v. 5.0 can be coupled with the existing 3DCityDB-Tools plugin, in order to enable support not only for the existing version 4.x, but also for the upcoming version 5.0. The research will start from familiarising the mapping structure of the data in both CityGML v.2.0 and v.3.0 to the new schema, performing experiments on the database modification on the server-side, and eventually improving the user experience with the existing QGIS plugin.

2 Related work

2.1 CityGML

CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. It not only allows visualisation for 3D models but also enables thematic queries, analysis tasks, or spatial data mining, satisfying the information needs of various application fields [Kolbe et al., 2021].
The CityGML v.3.0 core model defines basic concepts and the base classes, the fundamental element of which is the abstract class "AbstractCityObject" (Figure 1). It is the superclass of all thematic modules in CityGML, the space concepts are derived from the core, which consist of two subclasses named "AbstractSpace" and " AbstractSpaceBoundary" for volumetric and areal extent separately. The geometry representations and its Level of Detail (LoD) are associated with the space concepts (Figure 2).

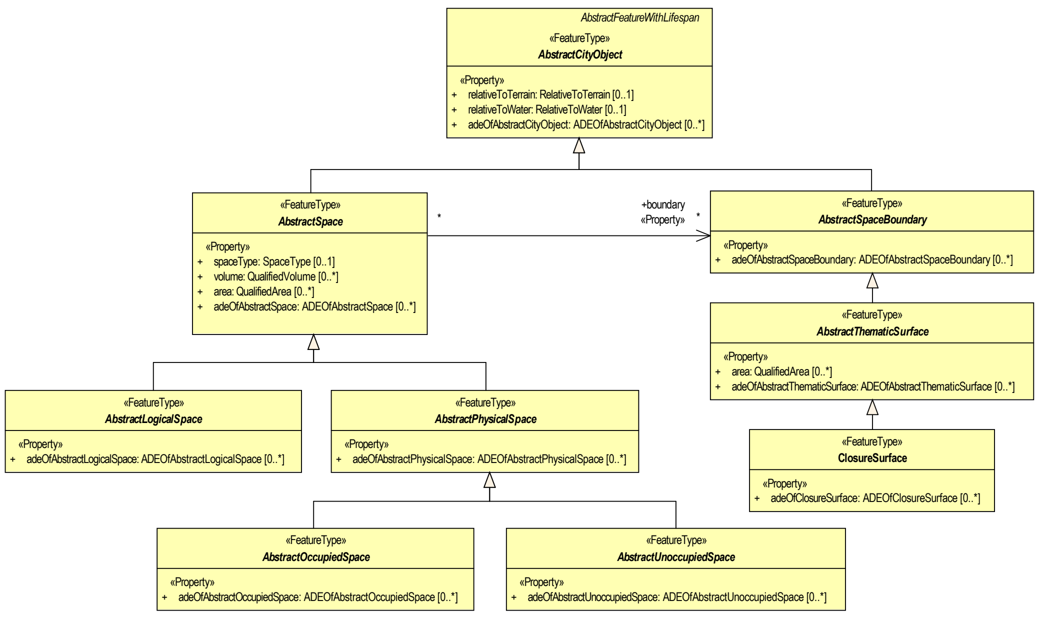

AbstractFeatureWithLifespan
«FeatureType» AbstractCityObject
«Property»
+ relativeToTerrain: RelativeToTerrain [0..1]
+ relativeToWater: RelativeToWater [0..1]
+ adeOfAbstractCityObject: ADEOfAbstractCityObject [0..*]
«FeatureType» AbstractSpace
«Property»
+ spaceType: SpaceType [0..1]
+ volume: QualifiedVolume [0..*]
+ area: QualifiedArea [0..*]
+ adeOfAbstractSpace: ADEOfAbstractSpace [0..*]
+boundary «Property»
«FeatureType» AbstractSpaceBoundary
«Property»
+ adeOfAbstractSpaceBoundary: ADEOfAbstractSpaceBoundary [0..*]
«FeatureType» AbstractThematicSurface
«Property»
+ area: QualifiedArea [0..*]
+ adeOfAbstractThematicSurface: ADEOfAbstractThematicSurface [0..*]
«FeatureType» ClosureSurface
«Property»
+ adeOfClosureSurface: ADEOfClosureSurface [0..*]
«FeatureType» AbstractLogicalSpace
«Property»
+ adeOfAbstractLogicalSpace: ADEOfAbstractLogicalSpace [0..*]
«FeatureType» AbstractPhysicalSpace
«Property»
+ adeOfAbstractPhysicalSpace: ADEOfAbstractPhysicalSpace [0..*]
«FeatureType» AbstractOccupiedSpace
«Property»
+ adeOfAbstractOccupiedSpace: ADEOfAbstractOccupiedSpace [0..*]
«FeatureType» AbstractUnoccupiedSpace
«Property»
+ adeOfAbstractUnoccupiedSpace: ADEOfAbstractUnoccupiedSpace [0..*]

Figure 1: Core - Space concepts of CityGML 3.0 UML diagram (Figure from [Kolbe et al., 2021])

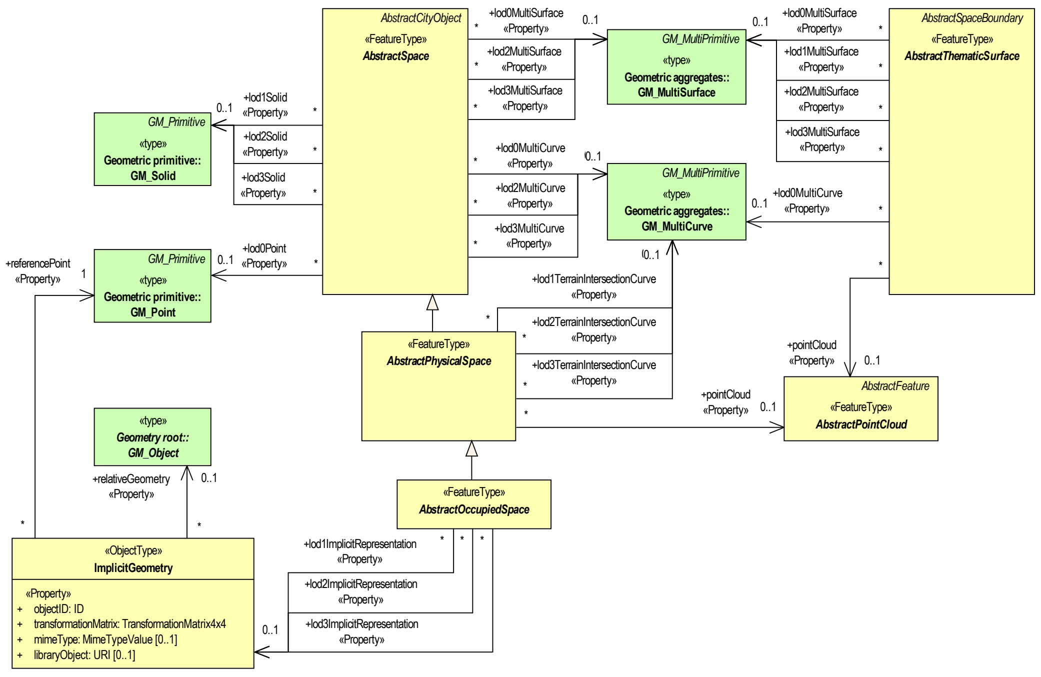

AbstractCityObject
«FeatureType» AbstractSpace
+lod0MultiSurface «Property»
+lod2MultiSurface «Property»
+lod3MultiSurface «Property»
GM_MultiPrimitive
«type» Geometric aggregates:: GM_MultiSurface
+lod1MultiSurface «Property»
AbstractSpaceBoundary
«FeatureType» AbstractThematicSurface
+lod1Solid «Property»
+lod2Solid «Property»
+lod3Solid «Property»
GM_Primitive
«type» Geometric primitive:: GM_Solid
+lod0MultiCurve «Property»
+lod2MultiCurve «Property»
+lod3MultiCurve «Property»
«type» Geometric aggregates:: GM_MultiCurve
+lod0Point «Property»
+referencePoint «Property»
«type» Geometric primitive:: GM_Point
+lod1TerrainIntersectionCurve «Property»
+lod2TerrainIntersectionCurve «Property»
+lod3TerrainIntersectionCurve «Property»
«FeatureType» AbstractPhysicalSpace
+pointCloud «Property»
AbstractFeature
«FeatureType» AbstractPointCloud
«type» Geometry root:: GM_Object
+relativeGeometry «Property»
«FeatureType» AbstractOccupiedSpace
+lod1ImplicitRepresentation «Property»
+lod2ImplicitRepresentation «Property»
+lod3ImplicitRepresentation «Property»
«ObjectType» ImplicitGeometry
«Property»
+ objectID: ID
+ transformationMatrix: TransformationMatrix4x4
+ mimeType: MimeTypeValue [0..1]
+ libraryObject: URI [0..1]

Figure 2: Core - Geometry and LoD concept of CityGML v.3.0 UML diagram (Figure from [Kolbe et al., 2021])

Features in the built environment are then mapped to the corresponding space concept based on its semantics. For example, the module “Building” contains two subclasses “building” and “buildingPart” which describe the physical volumetric extents of a building are derived from the “AbstractSpace” while its roofs, walls, etc. properties referring to the thematic surfaces are therefore derived from the areal “AbstractSpaceBoundary” (Figure 3).

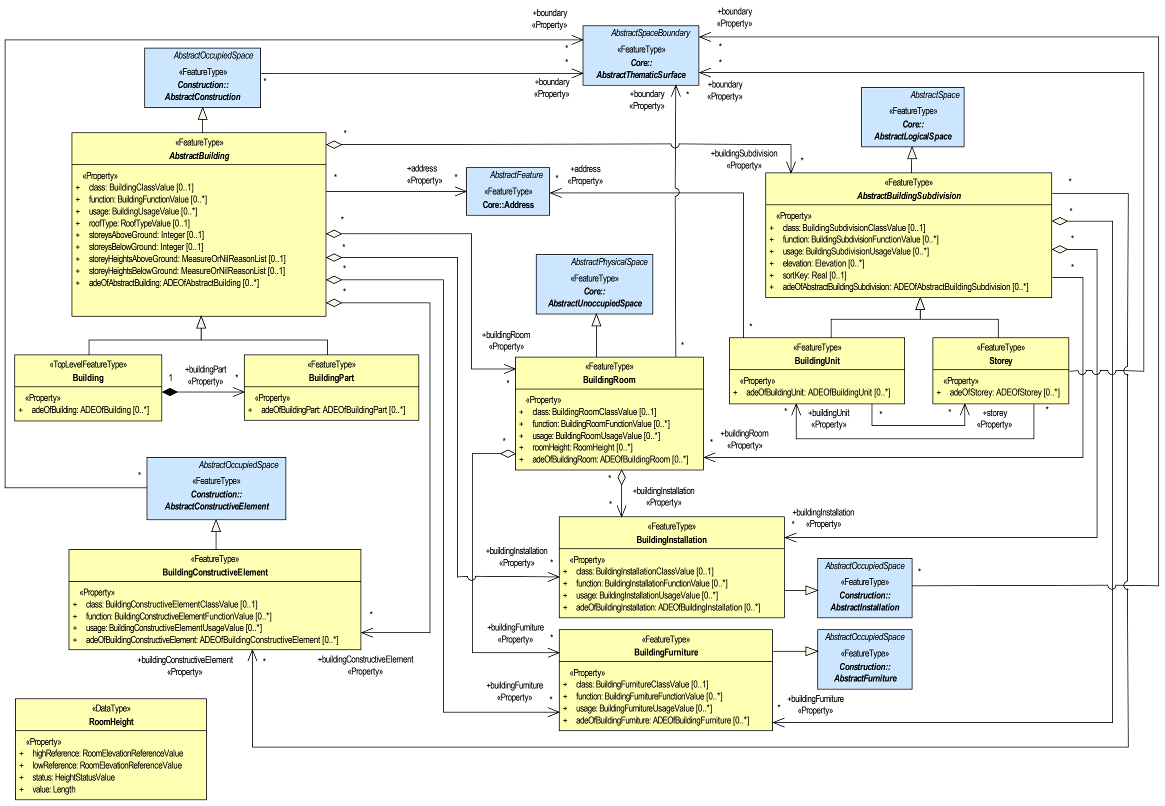
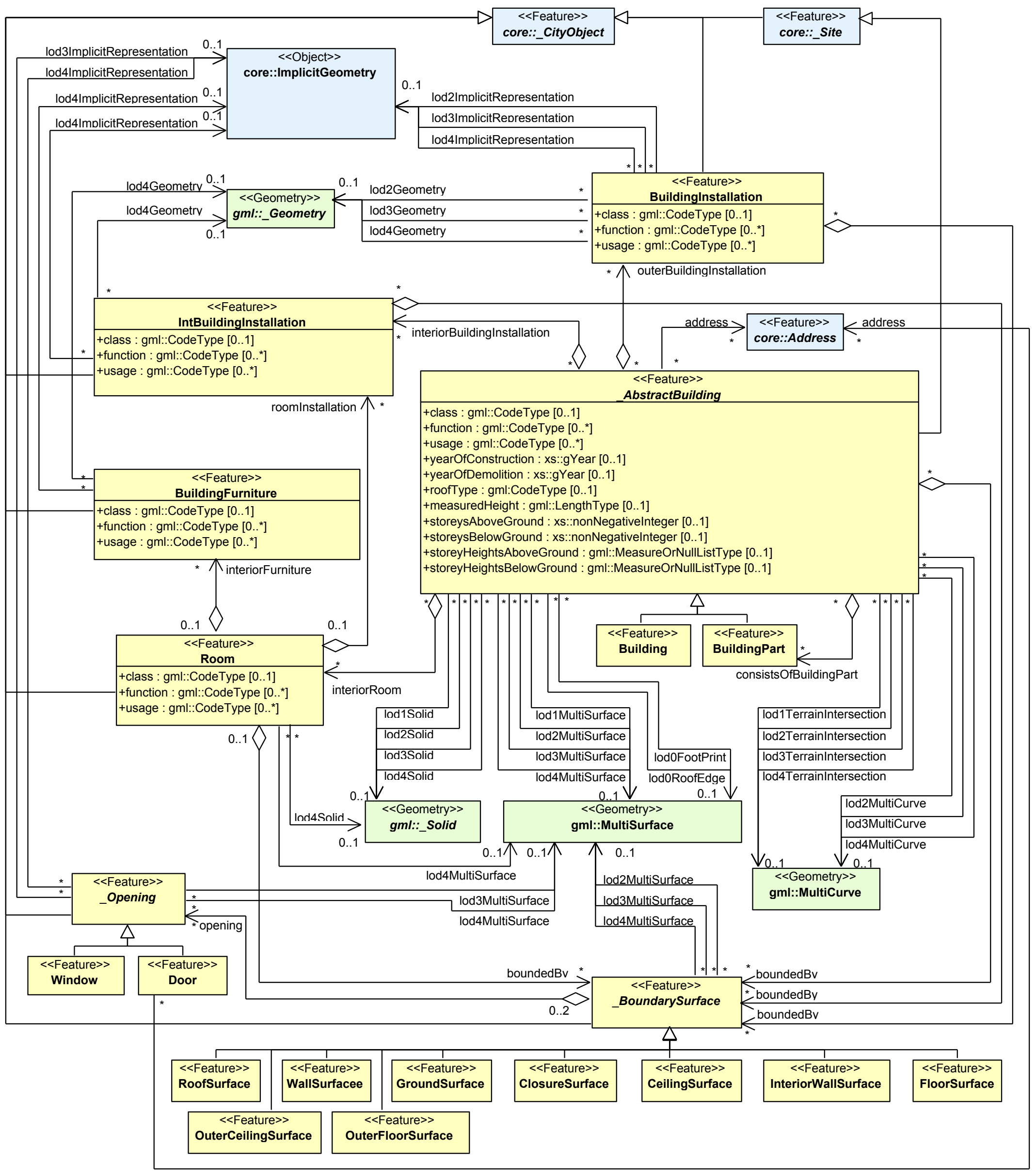

AbstractSpaceBoundary
«FeatureType»
Core::
AbstractThematicSurface
AbstractOccupiedSpace
«FeatureType»
Construction::
AbstractConstruction
«FeatureType»
AbstractBuilding
«Property»
+ class: BuildingClassValue [0..1]
+ function: BuildingFunctionValue [0..*]
+ usage: BuildingUsageValue [0..*]
+ roofType: RoofTypeValue [0..1]
+ storeysAboveGround: Integer [0..1]
+ storeysBelowGround: Integer [0..1]
+ storeyHeightsAboveGround: MeasureOrNilReasonList [0..1]
+ storeyHeightsBelowGround: MeasureOrNilReasonList [0..1]
+ adeOfAbstractBuilding: ADEOfAbstractBuilding [0..*]
AbstractFeature
«FeatureType»
Core::Address
AbstractSpace
«FeatureType»
Core::
AbstractLogicalSpace
«FeatureType»
AbstractBuildingSubdivision
«Property»
+ class: BuildingSubdivisionClassValue [0..1]
+ function: BuildingSubdivisionFunctionValue [0..*]
+ usage: BuildingSubdivisionUsageValue [0..*]
+ elevation: Elevation [0..*]
+ sortKey: Real [0..1]
+ adeOfAbstractBuildingSubdivision: ADEOfAbstractBuildingSubdivision [0..*]
AbstractPhysicalSpace
«FeatureType»
Core::
AbstractUnoccupiedSpace
«TopLevelFeatureType»
Building
«Property»
+ adeOfBuilding: ADEOfBuilding [0..*]
«FeatureType»
BuildingPart
«Property»
+ adeOfBuildingPart: ADEOfBuildingPart [0..*]
«FeatureType»
BuildingRoom
«Property»
+ class: BuildingRoomClassValue [0..1]
+ function: BuildingRoomFunctionValue [0..*]
+ usage: BuildingRoomUsageValue [0..*]
+ roomHeight: RoomHeight [0..*]
+ adeOfBuildingRoom: ADEOfBuildingRoom [0..*]
«FeatureType»
BuildingUnit
«Property»
+ adeOfBuildingUnit: ADEOfBuildingUnit [0..*]
«FeatureType»
Storey
«Property»
+ adeOfStorey: ADEOfStorey [0..*]
AbstractOccupiedSpace
«FeatureType»
Construction::
AbstractConstructiveElement
«FeatureType»
BuildingConstructiveElement
«Property»
+ class: BuildingConstructiveElementClassValue [0..1]
+ function: BuildingConstructiveElementFunctionValue [0..*]
+ usage: BuildingConstructiveElementUsageValue [0..*]
+ adeOfBuildingConstructiveElement: ADEOfBuildingConstructiveElement [0..*]
«FeatureType»
BuildingInstallation
«Property»
+ class: BuildingInstallationClassValue [0..1]
+ function: BuildingInstallationFunctionValue [0..*]
+ usage: BuildingInstallationUsageValue [0..*]
+ adeOfBuildingInstallation: ADEOfBuildingInstallation [0..*]
AbstractOccupiedSpace
«FeatureType»
Construction::
AbstractInstallation
«FeatureType»
BuildingFurniture
«Property»
+ class: BuildingFurnitureClassValue [0..1]
+ function: BuildingFurnitureFunctionValue [0..*]
+ usage: BuildingFurnitureUsageValue [0..*]
+ adeOfBuildingFurniture: ADEOfBuildingFurniture [0..*]
AbstractOccupiedSpace
«FeatureType»
Construction::
AbstractFurniture
«DataType»
RoomHeight
«Property»
+ highReference: RoomElevationReferenceValue
+ lowReference: RoomElevationReferenceValue
+ status: HeightStatusValue
+ value: Length
+boundary «Property»
+address «Property»
+buildingSubdivision «Property»
+buildingPart «Property»
+buildingRoom «Property»
+buildingUnit «Property»
+storey «Property»
+buildingInstallation «Property»
+buildingFurniture «Property»
+buildingConstructiveElement «Property»

Figure 3: Building module of CityGML v.3.0 UML diagram
(Figure from [Kolbe et al., 2021])

The geometry and LoD concept has changed fundamentally between CityGML v.2.0 and v.3.0. In CityGML v.2.0, geometry representations and LoD are associated with the thematic module and the degree of semantic decomposition, thematic surfaces were only allowed starting from LoD2 and the interior rooms only in LoD4 [Open Geospatial Consortium et al., 2021]. The geometry representation in the thematic module level requires each thematic features to have direct associations with geometries, which, for example, if building possesses a " lod2MultiSurface" representation, all the corresponding boundary surfaces in the same LoD are required to be aggregated for generating this representation (Figure 4). However, in CityGML 3.0, the geometry representations are associated with the newly added space concepts in core module, which significantly simplifies the models of the thematic modules since all features inherit the attributes of space and space boundary classes, there will only be at most 23 combinations of geometry representation depending on different thematic modules. As this research aims to extend the use of the current 3DCityDB developed for the CityGML v.2.0 to facilitate the application in QGIS, differentiating the standard difference of geometry representation is of importance. The new space concepts provide a clear scope for generating all possible "layers" that can be viewed in QGIS at the first step.



<<Feature>> core::_CityObject
<<Feature>> core::_Site
<<Object>> core::ImplicitGeometry
lod3ImplicitRepresentation
lod4ImplicitRepresentation
lod2ImplicitRepresentation
<<Geometry>> gml::_Geometry
lod2Geometry
lod3Geometry
lod4Geometry
<<Feature>> BuildingInstallation
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
outerBuildingInstallation
<<Feature>> IntBuildingInstallation
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
interiorBuildingInstallation
address
<<Feature>> core::Address
<<Feature>> _AbstractBuilding
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
+yearOfConstruction : xs::gYear [0..1]
+yearOfDemolition : xs::gYear [0..1]
+roofType : gml::CodeType [0..1]
+measuredHeight : gml::LengthType [0..1]
+storeysAboveGround : xs::nonNegativeInteger [0..1]
+storeysBelowGround : xs::nonNegativeInteger [0..1]
+storeyHeightsAboveGround : gml::MeasureOrNullListType [0..1]
+storeyHeightsBelowGround : gml::MeasureOrNullListType [0..1]
roomInstallation
<<Feature>> BuildingFurniture
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
interiorFurniture
<<Feature>> Building
<<Feature>> BuildingPart
consistsOfBuildingPart
<<Feature>> Room
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
interiorRoom
lod1Solid
lod2Solid
lod3Solid
lod4Solid
lod1MultiSurface
lod2MultiSurface
lod3MultiSurface
lod4MultiSurface
lod0FootPrint
lod0RoofEdge
lod1TerrainIntersection
lod2TerrainIntersection
lod3TerrainIntersection
lod4TerrainIntersection
lod2MultiCurve
lod3MultiCurve
lod4MultiCurve
<<Geometry>> gml::_Solid
<<Geometry>> gml::MultiSurface
<<Geometry>> gml::MultiCurve
<<Feature>> _Opening
opening
<<Feature>> Window
<<Feature>> Door
boundedBy
<<Feature>> _BoundarySurface
<<Feature>> RoofSurface
<<Feature>> WallSurface
<<Feature>> GroundSurface
<<Feature>> ClosureSurface
<<Feature>> CeilingSurface
<<Feature>> InteriorWallSurface
<<Feature>> FloorSurface
<<Feature>> OuterCeilingSurface
<<Feature>> OuterFloorSurface

Figure 4: Building module of CityGML v.3.0 UML diagram
(Figure from [Gröger et al., 2012])

2.2 3D City DataBase

The "3DCityDB is an Open Source software suite allowing to import, manage, analyze, visualize, and export virtual 3D city models according to the CityGML standard, supporting both versions 2.0 and 1.0." [Yao et al., 2018]. The database schema is established to accommodate the CityGML model for both storage and processing. However, some restrictions must be applied in order to convert the conceptual model into compacted relational tables, otherwise, a one-to-one mapping of CityGML data model will result in a vast number of tables and relations in between [Pantelios, 2022]. According to [Yao et al., 2018], "the super class shall be an abstract class that holds all attributes and associations which will be inherited by the concrete sub- classes and every of the sub-classes shall not have any further attributes or associated with other classes". To respond to this, certain tables are introduced to the schema with the primary and foreign keys to store the relations between them. The demonstration of converting the conceptual model to the relational tables of the building class in CityGML v.2.0 is shown in Figure 5 as an example.

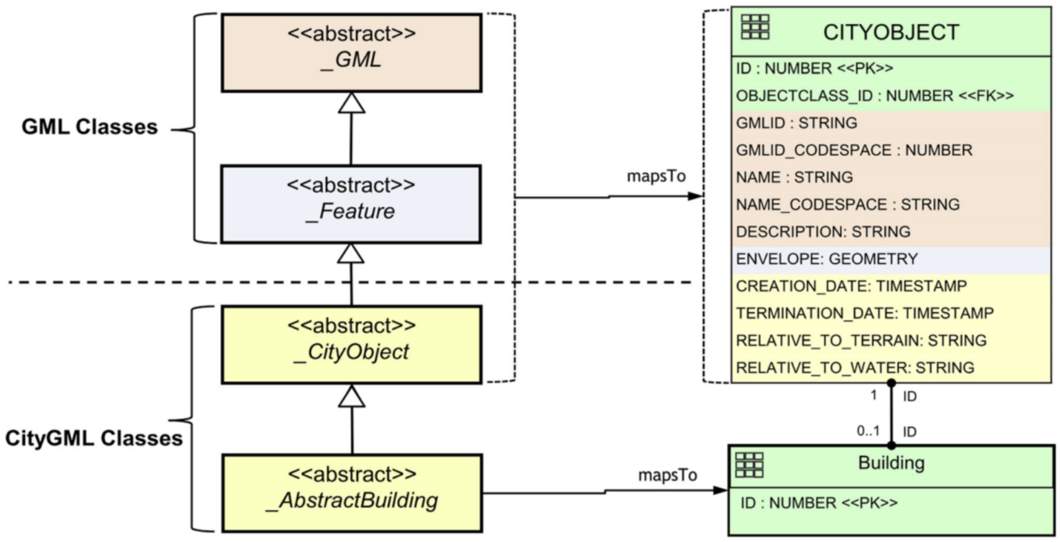


Figure 5: 3DCityDB Inheritance of building feature mapping in CityGML v.2.0 (Figure from [Yao et al., 2018])

The latest release of 3DCityDB v.4.4.0 maps classes in CityGML v.2.0 to 66 database relational tables in a PostgreSQL database schema. Since the CityGML v.3.0 testing version was released recently, 3DCityDB has been continuously updated, and by the time of writing, 3DCityDB v.0.6.0-beta is used in this research for management of the spatial data in accordance with CityGML v.3.0, which maps the classes to only 18 relational tables. Therefore, there is a substantial change in the mapping rules between the conceptual model (in UML) and the Entity-Relationship (ER) model of the database.

- Table ***Feature***: It stores all general information of all features within the given dataset, which contains primary keys like id, objectclass_id for further relational join.
- Table ***Property***: It accommodates all attributes of the existing features, which contains foreign keys like feature_id to link to corresponding features.
- Table ***Geometry_data***: It stores all the geometry representations of all existing features. Foreign keys like id, *feature_id* are set to link to corresponding features and properties.
- Table ***Implicit_geometry***: It stores the information of implicit geometry, which serves as an example representation that will be scaled and shifted to the reference places while creating viewable layers. Foreign key, *relative_geometry_id* is set to link to the corresponding geometry

The other tables which are not further explained here, are those used for appearances, etc. These tables are not currently relevant at the initial phase of this research. The details of the relational tables can be seen in the mapping schema of 3DCityDB v.0.6.0-beta, which is shown in Figure 6. In summary, 3DCityDB maps CityGML into a compact database schema that reduces operational complexity without introducing semantic ambiguity [Pantelios, 2022].

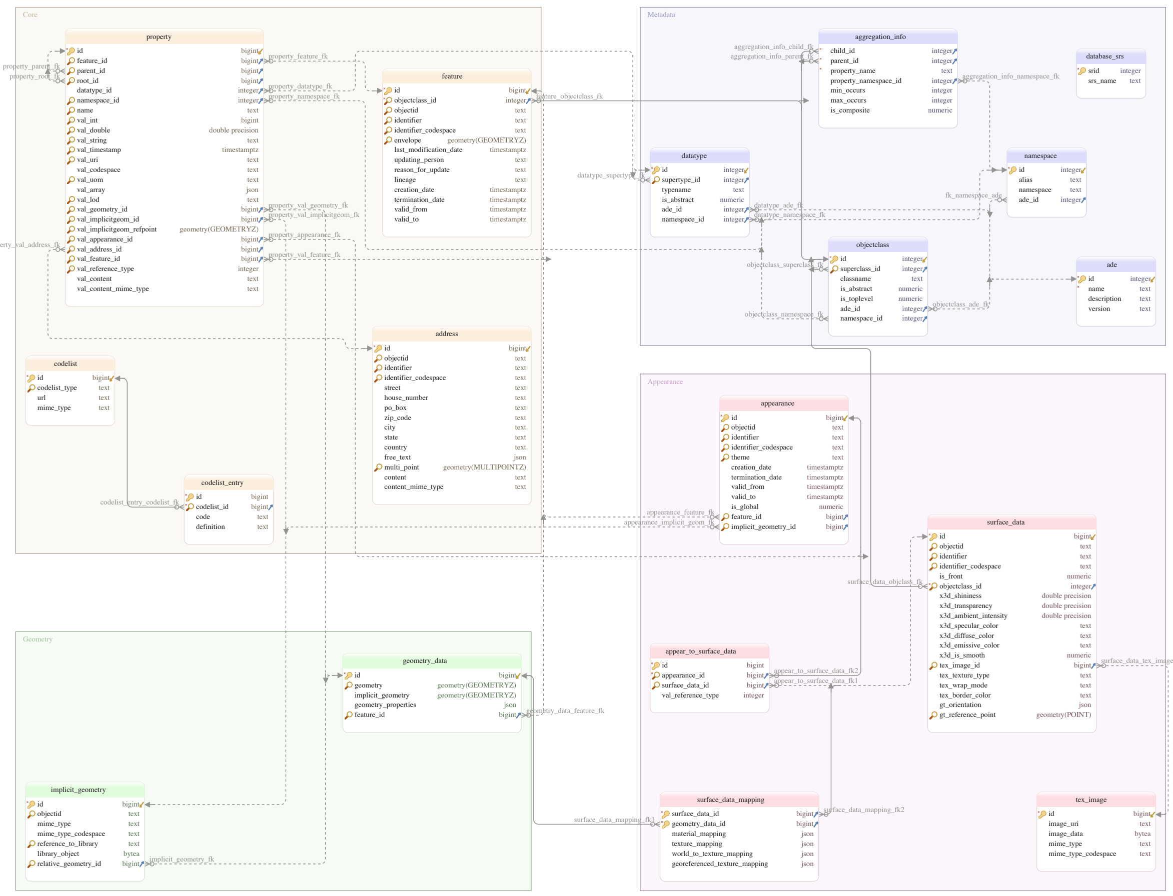


Figure 6: 3DCityDB schema of v.0.6.0-beta (Figure from [Nagel et al., 2023]

2.3 3DCityDB-Tools for QGIS

The existing plugin named "3DCityDB Tools for QGIS" to facilitate the use of CityGML data in QGIS is available through Github [Agugiaro et al., 2023]. According to the manual, the current 3DCityDB Tools v.0.8.7 supports the management and visualisation of data stored in the 3DCityDB with regard to the CityGML v.1.0 and v.2.0.

The plugin allows users to connect the local and remote 3DCityDB instances for PostgreSQL/PostGIS and load the data into QGIS. to create viewable layers, which are designed to facilitate users from different fields and expertise in their interaction with CityGML data encoded in the 3DCityDB. Once data layers are available in QGIS, the users can perform analyses, access and edit associated attributes, explore and visualise the data in 2D and 3D based on the built-in functions of QGIS and its other plugins.

The server-side part of the plugin, called "QGIS Package", provides capability for management of database users and data layers. It allows the users to define and create a layer by extracting a specific, selectable geometry (according to its LoD) of a feature and relate to its corresponding attributes. This structure of such a layer is in compliance with the Simple Feature Model (SFM). The client-side part of the plugin provides different tools for different uses:

- The QGIS Package Administrator tools allows database administrators to install the server-side part of the plug-in, as well as to set up database user access and user privileges.

- The Layer Loader tool allows users to load and interact with data in the 3D City Database directly from QGIS.

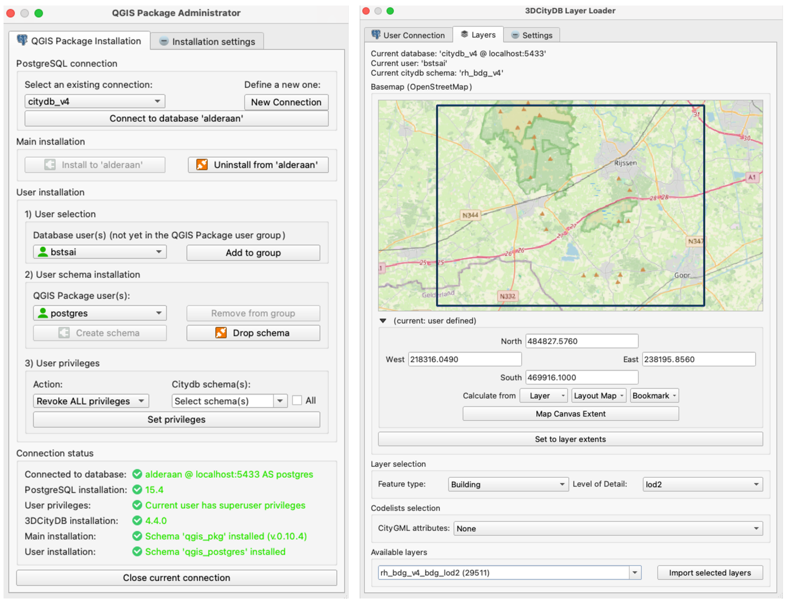
- The Bulk Delete tool allows users to delete features from the database, either at all at once, or by means of spatial and feature-related filters.

Users have the access to connect to the database using a friendly graphical user interface (GUI), set up the schema and grant privileges to the users in the server-side, building up all the functionalities for users and layers management. Since the current version of 3DCityDB-Tool plugin only supports 3DCityDB v.4.x. This research will mainly focus on extending the functionality of the plugin to support 3DCityDB v.5.0, and as consequence, to support CityGML 3.0. The work can be subdivided into two parts:

The first regards the server-side part, including creating feature geometries called views or materialized views and linking the feature attributes with the corresponding feature geometries to form the QGIS-usable SFM "layers". Views here are referring to the SQL transformation from a set of base tables to a derived table in PostgreSQL, which are recomputed every time when they are referenced or queried. The geometries of the features managed by 3DCityDB v.5 are stored in the geometry column of the **geometry_data** table, which then can be query via the primary keys of "*objectclass_id*" and other foreign keys, resulting in tables containing all the geometry with regard to each feature type.

Note that, since querying the geometry table on the fly would be time-consuming regarding the scale to join several tables. For efficiency reasons, the layer created by the 3DCityDB-Tools plugin is the result of linking a view with attributes to the materialized view with the geometries ([Pantelios, 2022], [Agugiaro et al., 2024]). Materialized views here are referring to the temporary virtual table which is used to store the proactively computed result of views in PostgreSQL. Index structures can be built on the materialized view to boost the performance while it is accessed in the database, the query execution time could be much faster compared to that of querying from a view.

The second regards the client-side part, including managing the 3DCityDB layers that are created by the process mentioned above and dealing with layers interaction functions such as updating attributes or features deletion etc.



Select an existing connection: citydb_v4
Define a new one:
Database user(s) (not yet in the QGIS Package user group): bstsai
QGIS Package user(s): postgres
Action: Revoke ALL privileges
Citydb schema(s): Select schema(s) ☐ All

Connected to database: ✓ alderaan @ localhost:5433 AS postgres
PostgreSQL installation: ✓ 15.4
User privileges: ✓ Current user has superuser privileges
3DCityDB installation: ✓ 4.4.0
Main installation: ✓ Schema 'qgis_pkg' installed (v.0.10.4)
User installation: ✓ Schema 'qgis_postgres' installed

Current database: 'citydb_v4 @ localhost:5433'
Current user: 'bstsai'
Current citydb schema: 'rh_bdg_v4'

North 484827.5760
West 218316.0490
East 238195.8560
South 469916.1000
Calculate from Layer Layout Map Bookmark

Feature type: Building
Level of Detail: lod2
CityGML attributes: None
rh_bdg_v4_bdg_lod2 (29511)

Figure 7: 3DCityDB-Tools GUI
(left: screenshot of the QGIS Package Administrator tool, right: Layer Loader tool)

3 Research questions

3.1 Objectives

This main goal of the research is to investigate how to add support for CityDB 5.x (and as a consequence CityGML v.3.0) to the 3DCityDB Tools plugin for QGIS. To achieve this goal, the following research questions need to be addressed.

- How does the new database structure of 3DCityDB v.5 affect the current methodology of the plugin to create layers which contain both geometries and attributes for a selected feature type following the SFS. In particular,
  - How do the new CityGML 3.0 concepts of space, LoD affect the process?
- With regards to geometries, can the same or a similar approach be reproduced?
  - Is it still necessary to rely on materialized views?
  - What are the performance differences between 3DCityDB 4.x and 5.x when retrieving geometries from the database?
  - How does the QGIS Package need to be restructured?
- With regards to attributes, can the same or a similar approach be reproduced?
  - Is it still necessary to rely on updatable views?
  - What alternatives are there?
  - How does the plugin front-end need to be restructured?
- How is the CityGML v.2.0 data mapped to the new schema of 3DCityDB v.5.0?
  - Can we deal with CityGML2.0 data as CityGML 3.0 data as long as it is stored in the 3DCityDB 5.x?

3.2 Scope of research

The scope of the research will focus on the thematic features of "Appearance", "Building", "Bridge", "CityObjectGroup", "generic", "Tunnel", "Vegetation", "Transportation", "Relief", "CityFurniture", "WaterBody", "Landuse" and their attributes, detail level and types of geometries. As the above-mentioned 12 features are the most frequently accessed information of a city, and they already existed in the CityGML v.2.0, understanding the layer creation process of these modules will facilitate the functions adaptation in compliance with CityGML v.3.0.

Consider the , "Dynamizer", "PointCloud" and "Versioning" are threenewly added modules in new CityGML standards. The 3DCityDB v.5.0 currently does not support these modules mapping to relational tables, and its development is still in progress. The research will take its development into consideration and attempt to offer conceptual suggestions of how to support the application of features from these modules in QGIS.

4 Methodology

The proposed methodology consists of two phases. The first phase focuses on the Plugin server-side in PostgreSQL, which takes the "*objectclass_id*" as the feature identifier, search in the 3DCityDB for its all possible geometry presentations and perform the SQL query among the 18 relational tables that are mapped by the 3DCityDB v.5.0 (see 2.2) to create the loadable layers with the corresponding attributes for further application in QGIS.

The second phase emphasises on the Plugin client-side in QGIS, which deals with the layer management of all available layers, testing and adapting the relative layers functions, e.g., inserts, updates and deletions. Although the two phases are described separately, they are highly interrelated since certain adjustments on the layer creation part could be necessary while working on the compatibility part and vice versa. The overview methodology is shown in Figure 8.

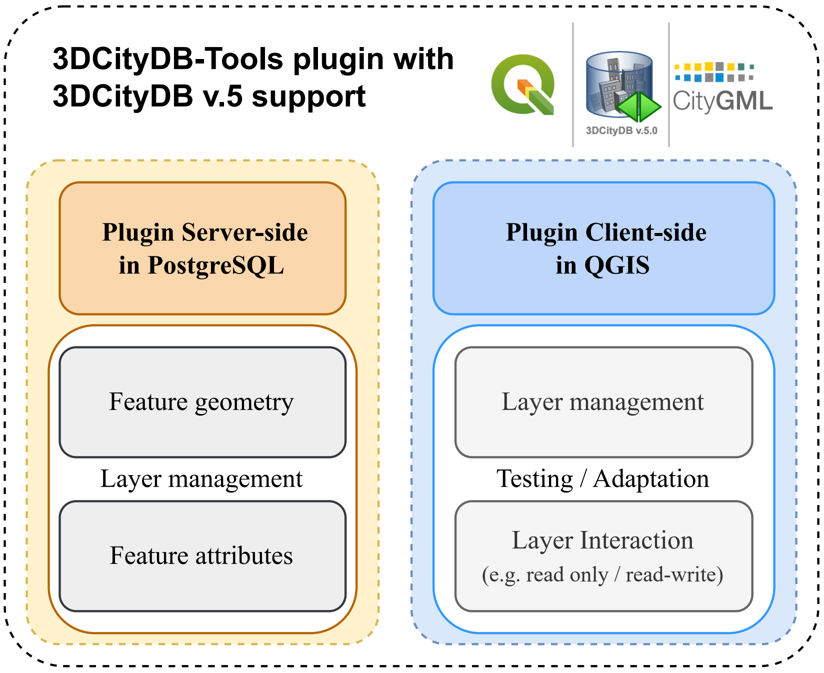


Figure 8: Overview of the proposed methodology

4.1 Plugin Server-side in PostgreSQL

4.1.1 Feature geometry

According to the CityGML v.3.0 UML diagram (Figure 2), the core geometry and LoD concept defines all the possible geometry representation of the abstract features of "AbstractSpace" and "AbstractSpaceBoundary". Since all the features in CityGML v.3.0 are derived from either two of the superclasses, there are only 23 possible combinations of geometry representation (Table 1) for each feature. This step will only take on the feature identifier, "objectclass_id", stored in the FEATURE table in 3DCityDB v.5.0 to gather its corresponding geometry views via SQL query.

At the first step of the research, both views and materialized view will be created among all feature types using the new relational table structure generated by 3DCityDB v.5.0 in order to test the time efficiency of performing a query from them.

4.1.2 Corresponding attributes attachment

According to the CityGML v.3.0 UML diagram (Figure 3), thematic features like buildings possess attributes such as class, usage and roofType, etc. These entity attribute values (EAV) are stored in the PROPERTY table and they are also able to be queried by sets of keys across different tables. The 3DCityDB-Tools plugin currently associates these attributes via the hierarchical relation class called Table Of Content (TOC) in QGIS, which sets upa Many-to-One relationship between fields of referenced and referencing layers [Pantelios, 2022]. The TOC relation class in QGIS is proposed to link the attributes in the property table to their corresponding main features, which will be investigated in the future work.

lod	representation	class	lod	representation	class
0	lod0point	space	0	lod0multisurface	boundary
0	lod0multisurface	space	0	lod0multicurve	boundary
0	lod0multicurve	space	1	lod1multisurface	boundary
1	lod1solid	space	2	lod2multisurface	boundary
1	lod1terrainintersectioncurve	space	3	lod3multisurface	boundary
1	lod1implicitrepresentation	space	x	pointcloud	boundary
2	lod2solid	space	x	envelope	boundary
2	lod2multisurface	space			
2	lod2multicurve	space			
2	lod2terrainintersectioncurve	space			
2	lod2implicitrepresentation	space			
3	lod3solid	space			
3	lod3multisurface	space			
3	lod3multicurve	space			
3	lod2terrainintersectioncurve	space			
3	lod3implicitrepresentation	space			
x	pointcloud	space			
x	envelope	space			

Table 1: Possible geometry representations of the two superclasses in CityGML v.3.0

4.2 Plugin Client-side in QGIS

4.2.1 Layer management

The concept of geometry representation has significantly changed between the CityGML v.2.0 and v.3.0, it was elevated to a higher core level not associated with the thematic level. Although the modification simplifies the thematic model representation, it also introduces some potential incompatibilities. For example, "lod2MultiSurface" of the building class used to be bounded by all its thematic surfaces, the geometry is represented by the aggregation of the corresponding LoD thematic surface (see Figure 4), which is then used to associated with the LoD-relating attribute in CityGML v.2.0. However, in the relational table mapped by the 3DCityDB v.5.0 in accordance with the CityGML v.3.0, building feature does not have "lod2MultiSurface" geometry representation initially, the "bounded by" relation is mapped to the "has" relation, indicating that building features possess thematic boundaries which are different features other than building. These potential incompatibilities arise as building features still have attributes like "LoD2 volume" but it has no geometry representation to be associated, which requires further solution to answer the research question in terms of attribute attachment.

The above-mentioned case is just an example, these potential incompatibilities derived from standard differences need to be solved in order to extend the use of CityGML data in 3DCityDB. The proposal is to discuss it case by case, as a possible solution to the example case could be performing the aggregation or including the geometry such as the envelope of the features as an alternative.

4.2.2 Layer interaction

The current 3DCityDB-Tools plugin sets up trigger functions that allow users to handle "update", "insert" and "delete" operations when creating the layers. The "insert" function is currently forbidden as its implementation does not handle new geometries while "update" operation is only available for updating attributes but not geometries ([Pantelios, 2022], [Agugiaro et al., 2024]). Considering the potential incompatibilities mentioned above, these functions could be modified to allow users to create aggregation of the building geometry in "lod2MultiSurface" and insert into the geometry table while simultaneously updating the property to assign "lod2MultiSurface" attribute to the building feature. This could be a possible solution to satisfy the association need of attributes with missing geometry. The function could be adjusted according to the potential incompatibilities.

5 Preliminary results

5.1 Querying geometries from the 3DCityDB v.5.0

In order to reproduce the layer creation procedure of the current 3DCityDB-Tools plugin, we need ideally to first deal with geometries (e.g. create views), then to attach attributes to them.

In other words, a geometry view is intended as a view (or a materialized view) associated to a query that extracts geometries from the ***geometry_data*** table according to the "*objectclass_id*" and its geometry representation - if data for such feature exists in the database. It is also relevant to investigate whether it is reasonable to create simple views or materialized views, as in the current implementation of the QGIS Plugin for CityDB 4.x [Agugiaro et al., 2024].

The pipeline shown in Figure 10 is built in Python to call and execute the SQL queries for creating views and materialized views of the geometries. The user first sets up the database connection, and executes the main function, which will ask for the user's input to create, delete or terminate. For creating views, the code runs the query to search for all possible "*objectclass_id*" from the date stored in the user-specified schema. For all "*objectclass_id*", the user chooses the view types, the code continues to call the query to search for all the possible geometry representations of the feature identified by the "*objectclass_id*" and the type of view will be created correspondingly.

The pipeline is currently capable of creating features from "Building", "Vegetation" and "Relief" modules, which are the most commonly used features in the built environment realm. The test dataset used for (materialized) views creation is the Rijsen-Holten and Vienna 3D models (see section 7), the outcome of the view creation is shown in Figure 12 and 13. The detail of the SQL queries for each testing feature module is explained as the following.

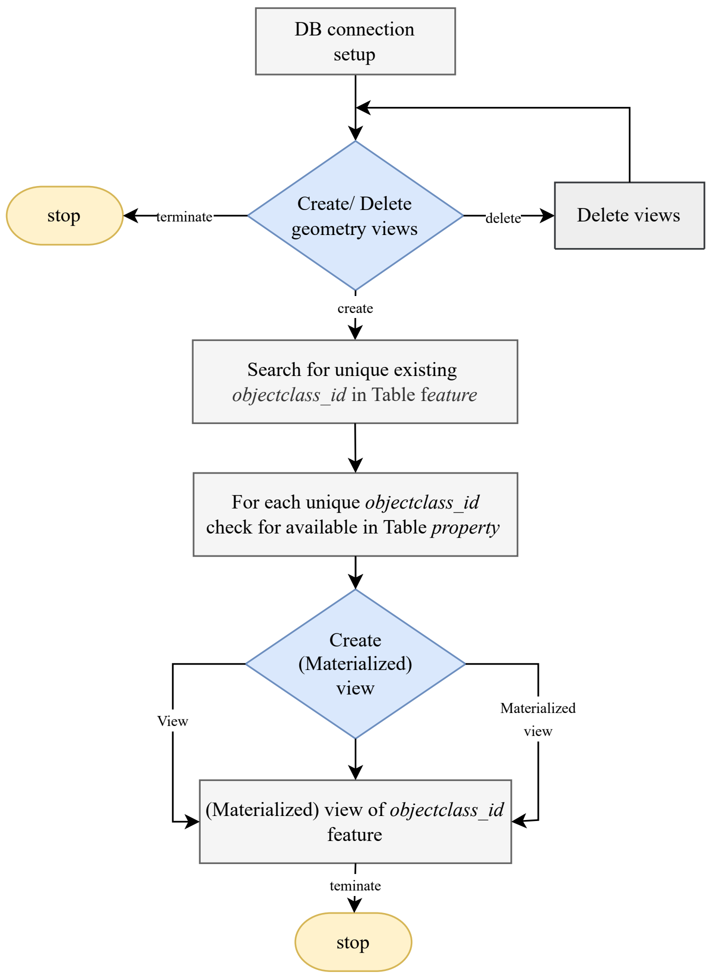

DB connection setup
Create/ Delete geometry views
stop
terminate
delete
Delete views
create
Search for unique existing *objectclass_id* in Table *feature*
For each unique *objectclass_id* check for available in Table *property*
Create (Materialized) view
View
Materialized view
(Materialized) view of *objectclass_id* feature
teminate
stop

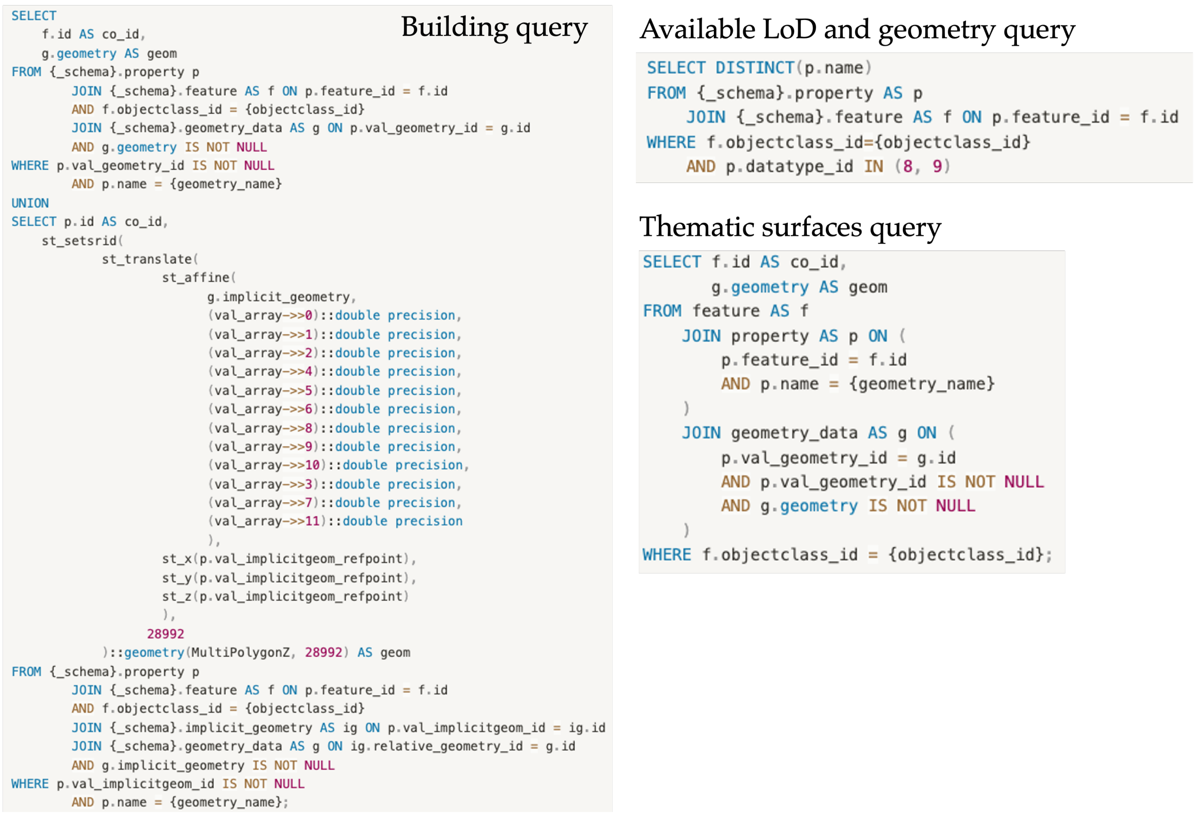
Figure 9: Layers creation pipeline

5.1.1 Building

In the test data, there are 6 different features with "textitobjectclass_id" relate to the building module, which are: Building (901), BuildingPart (902), RoofSurface (712), GroundSurface (710), WallSurface (709) and ClosureSurface (15). Building is derived from Abstract Space, which can have both geometry and implicit geometry representation while thematic surface is derived from Abstract Space boundary, which can only have geometry. The process is shown as the following pseudo-code with query blocks are shown in Figure 10.

Algorithm .1: Building views creation

```
Input: objectclass_id of building (901)
Output: view or materialized view of objectclass_id of 901
1 for objectclass_id = 901 do
2     execute (geometry query) ← gather geometry
3     ∪
4     execute (implicit geometry query) ← gather implicit geometry;
5 for objectclass_id = 15 or 709 or 710 or 712 do
6     execute (geometry query) ← gather thematic surface geometry
```


Building query

```sql
SELECT
    f.id AS co_id,
    g.geometry AS geom
FROM {_schema}.property p
        JOIN {_schema}.feature AS f ON p.feature_id = f.id
        AND f.objectclass_id = {objectclass_id}
        JOIN {_schema}.geometry_data AS g ON p.val_geometry_id = g.id
        AND g.geometry IS NOT NULL
WHERE p.val_geometry_id IS NOT NULL
        AND p.name = {geometry_name}
UNION
SELECT p.id AS co_id,
    st_setsrid(
            st_translate(
                    st_affine(
                          g.implicit_geometry,
                          (val_array->>0)::double precision,
                          (val_array->>1)::double precision,
                          (val_array->>2)::double precision,
                          (val_array->>4)::double precision,
                          (val_array->>5)::double precision,
                          (val_array->>6)::double precision,
                          (val_array->>8)::double precision,
                          (val_array->>9)::double precision,
                          (val_array->>10)::double precision,
                          (val_array->>3)::double precision,
                          (val_array->>7)::double precision,
                          (val_array->>11)::double precision
                          ),
                    st_x(p.val_implicitgeom_refpoint),
                    st_y(p.val_implicitgeom_refpoint),
                    st_z(p.val_implicitgeom_refpoint)
                    ),
                  28992
            )::geometry(MultiPolygonZ, 28992) AS geom
FROM {_schema}.property p
        JOIN {_schema}.feature AS f ON p.feature_id = f.id
        AND f.objectclass_id = {objectclass_id}
        JOIN {_schema}.implicit_geometry AS ig ON p.val_implicitgeom_id = ig.id
        JOIN {_schema}.geometry_data AS g ON ig.relative_geometry_id = g.id
        AND g.implicit_geometry IS NOT NULL
WHERE p.val_implicitgeom_id IS NOT NULL
        AND p.name = {geometry_name};
```

Available LoD and geometry query

```sql
SELECT DISTINCT(p.name)
FROM {_schema}.property AS p
    JOIN {_schema}.feature AS f ON p.feature_id = f.id
WHERE f.objectclass_id={objectclass_id}
    AND p.datatype_id IN (8, 9)
```

Thematic surfaces query

```sql
SELECT f.id AS co_id,
       g.geometry AS geom
FROM feature AS f
    JOIN property AS p ON (
        p.feature_id = f.id
        AND p.name = {geometry_name}
    )
    JOIN geometry_data AS g ON (
        p.val_geometry_id = g.id
        AND p.val_geometry_id IS NOT NULL
        AND g.geometry IS NOT NULL
    )
WHERE f.objectclass_id = {objectclass_id};
```

Figure 10: SQL query blocks for Building views creation

5.1.2 Vegetation

In the test data, only one tree feature with the "*objectclass_id*" of 1301 is available. According to the CityGML v.3.0 [Kolbe et al., 2021], vegetation is derived from Abstract Space, which can have both geometry and implicit geometry representation. Therefore, the view creation query is identical to that of the building. The process is shown as the following pseudo-code with the query shown in Figure 10.

Algorithm .2: Vegetation views creation

```
   Input: objectclass_id of vegetation (1300)
   Output: view or materialized view of objectclass_id of 1300
1  for objectclass_id = 1300 do
2      execute (geometry query query) ∪ execute (implicit geometry query)
```

5.1.3 Relief

In the test data, there are two different features with "*objectclass_id*" related to the relief module, which are: Relief feature (500), Relief (502). They are derived from the Abstract Space boundary that can only have geometry with LoD from 0 to 3 and no implicit geometry. Notice that the relief feature does not have corresponding geometry, its geometry representation is only available via the envelope of the feature. The relief component can have 4 different types of representation. The process is shown as the following pseudo-code with query blocks shown in Figure 11.

Algorithm .3: Relief feature views creation

```
   Input: objectclass_id of relief feature (500)
   Output: view or materialized view of objectclass_id of 500
1  for objectclass_id = 500 do
2      execute (search lods query)
3      execute (envelope query)
```

Algorithm .4: Relief component views creation

```
   Input: objectclass_id of relief component (502)
   Output: view or materialized view of objectclass_id of 502
1  for objectclass_id = 502 do
2      execute (search lod and types query) ← get available lods pairs
3      for lod_type_pair do
4          execute (geometry query)
```

Available LoD and geometry (Relief feature)

```sql
SELECT DISTINCT (p.val_int) AS lod
FROM {_schema}.feature AS f
  JOIN {_schema}.property AS p ON(
    f.id = p.feature_id
    AND f.objectclass_id = {objectclass_id})
WHERE p.name = 'lod'
```

Available LoD and geometry (Relief component)

```sql
SELECT DISTINCT (p.name),
  p.val_lod AS lod
FROM citydb_dtm.feature AS f
  JOIN citydb_dtm.property AS p ON(
    f.id = p.feature_id AND f.objectclass_id = 502)
WHERE p.datatype_id = 8
```

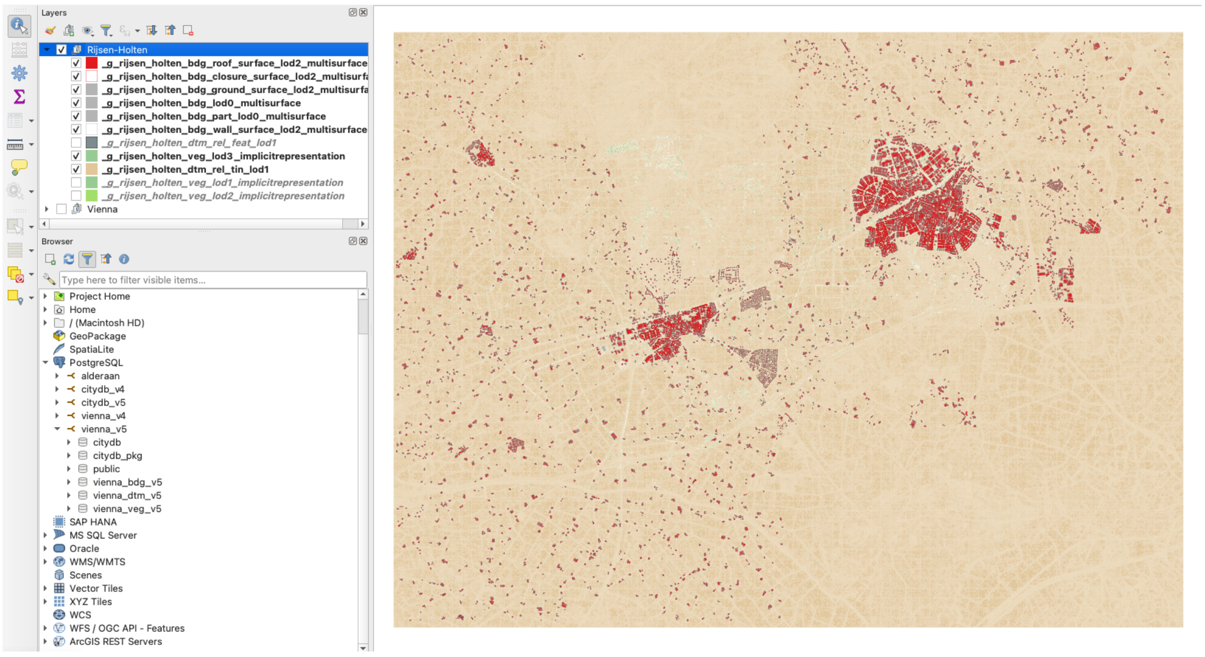
Relief feature query

```sql
SELECT
  f.id AS co_id,
  f.envelope AS geom
FROM {_schema}.property p
  JOIN {_schema}.feature f ON p.feature_id = f.id
  AND f.objectclass_id = {objectclass_id}
  AND p.name = 'lod'
WHERE p.val_int = {lodx}
```

Relief component query

```sql
SELECT
    f.id AS co_id,
    g.geometry AS geom
FROM {_schema}.property AS p
    JOIN {_schema}.feature AS f ON p.feature_id = f.id
    AND f.objectclass_id = {objectclass_id}
    JOIN {_schema}.geometry_data AS g ON p.val_geometry_id = g.id
    AND g.geometry IS NOT NULL
WHERE p.val_geometry_id IS NOT NULL AND p.name = {form}
    AND p.val_lod = {lodx}
GROUP BY f.id, g.geometry
```

Figure 11: SQL query for Relief layers creation



```
Layers
▾ ✓ Rijsen-Holten
    ✓ 🟥 _g_rijsen_holten_bdg_roof_surface_lod2_multisurface
    ✓ □ _g_rijsen_holten_bdg_closure_surface_lod2_multisurfa
    ✓ ▨ _g_rijsen_holten_bdg_ground_surface_lod2_multisurfa
    ✓ ▨ _g_rijsen_holten_bdg_lod0_multisurface
    ✓ ▨ _g_rijsen_holten_bdg_part_lod0_multisurface
    ✓ □ _g_rijsen_holten_bdg_wall_surface_lod2_multisurface
    □ ▨ _g_rijsen_holten_dtm_rel_feat_lod1
    ✓ 🟩 _g_rijsen_holten_veg_lod3_implicitrepresentation
    ✓ 🟫 _g_rijsen_holten_dtm_rel_tin_lod1
    □ 🟩 _g_rijsen_holten_veg_lod1_implicitrepresentation
    □ 🟩 _g_rijsen_holten_veg_lod2_implicitrepresentation
▸ □ Vienna
```

Figure 12: Layers created by the test pipeline loaded in QGIS (Rijsen-Holten dataset)

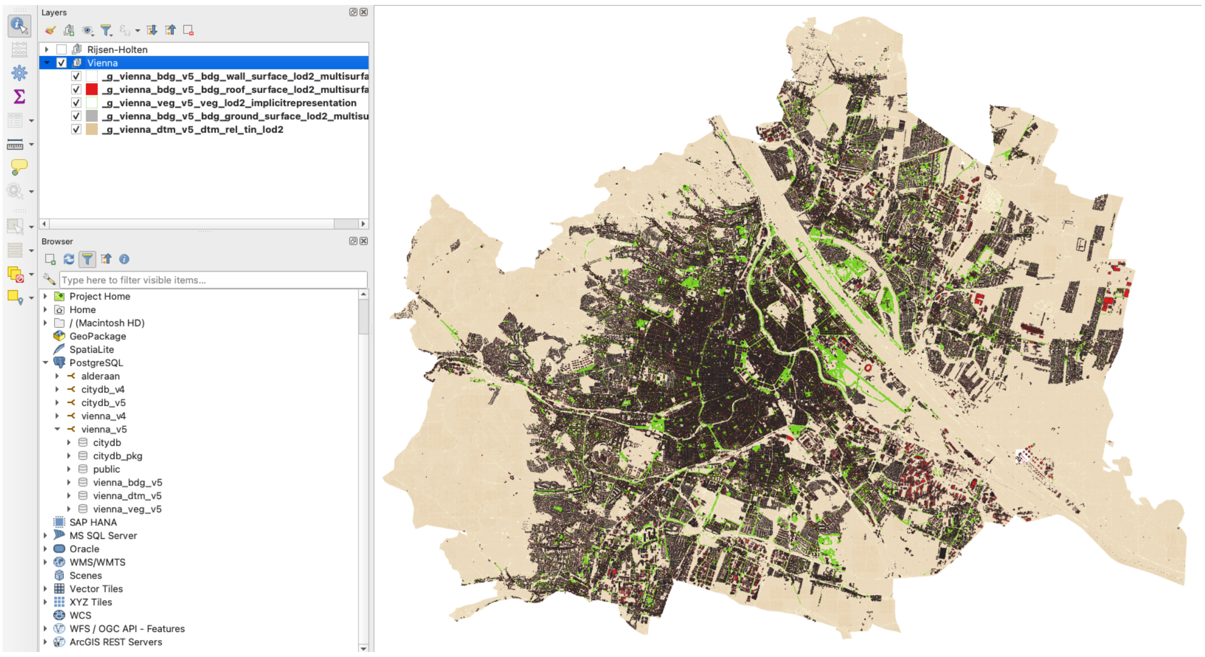


Figure 13: Layers created by the test pipeline loaded in QGIS (Rijsen-Holten dataset)

5.2 Views and Materialized views execution time comparison

The query execution time comparison between 3DCityDB v.4 and v.5 is established by performing a query to select features within a given extent on the views (V) and materialized views (MV) separately. It aims to test the time efficiency for querying them to understand whether it is reasonable to create simple views or materialized views as implemented by the current 3DCityDB-Tools Plugin. The result shown in Table 2 for the Rijsen-Holten and Vienna dataset. Since the space concept is associated with the core module but not thematic module in CityGML v.3.0, the pre-aggregate process of all the features like vegetation and relief tin model is not necessarily needed. The view query times of vegetation and relief with higher LoD show a huge improvement which also happens in the materialized view query times. The next stage of this research is to look into those time efficiency differences, and discuss the strategies for attaching attributes with the geometries for the layers creation.

6 Time planning

The schedule for the tasks related to the thesis is presented with a Gantt diagram in Figure 14.

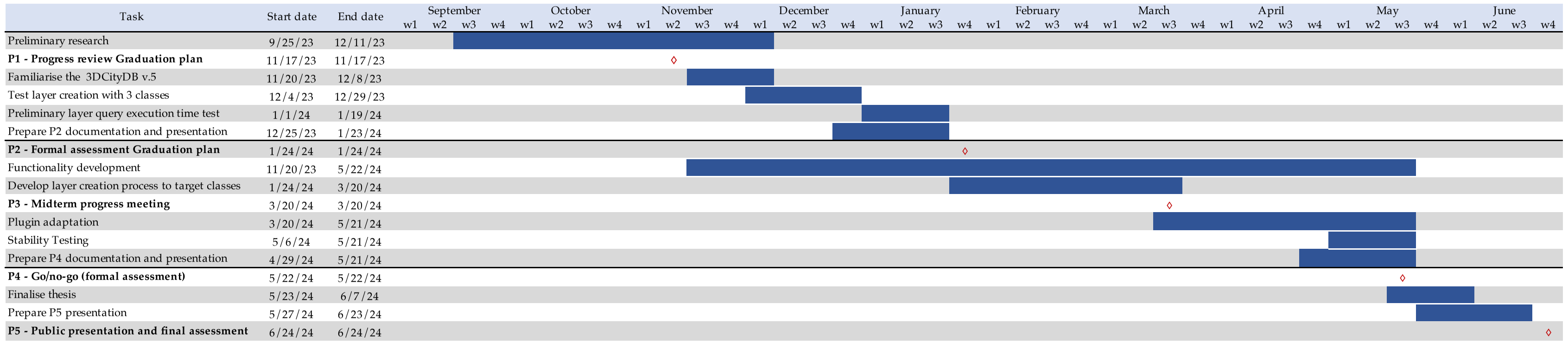
Attributes					3DCityDB v.4			3DCityDB v.5			Number of object selected
Dataset	Feature(CO)		LoD	Geometry	View	Materialized View		View	Materialized View		
					Query time	Refresh time	Query time	Query time	Refresh time	Query time	
Rijsen-Holten	Building		0	Multi Surface	00:01.0	00:03.1	00:00.2	00:02.0	00:03.7	00:00.2	30,449
	Building Part		0	Multi Surface	00:00.1	00:00.1	00:00.1	00:00.1	00:00.1	00:00.1	107
	Thematic surface	closure_surface	2	Multi Surface	00:01.1	00:02.9	00:00.2	00:01.4	00:01.5	00:00.1	50,902
		ground_surface	2	Multi Surface	00:01.0	00:02.6	00:00.2	00:01.8	00:02.5	00:00.2	29,618
		roof_surface	2	Multi Surface	00:01.5	00:04.5	00:00.4	00:01.8	00:03.5	00:00.3	94,210
		wall_surface	2	Multi Surface	00:05.3	00:16.2	00:01.1	00:03.0	00:08.4	00:01.0	558,362
	Vegetation (trees)		1	Implicit Representation	00:04.0	00:09.3	00:00.8	00:02.2	00:05.2	00:00.7	58,515
			2	Implicit Representation	00:03.8	00:06.3	00:00.6	00:01.5	00:04.3	00:00.5	58,515
			3	Implicit Representation	00:40.9	01:24.7	00:07.2	00:14.9	00:32.0	00:07.1	58,515
	Relief (Component)		1	tin	00:06.7	00:15.2	00:02.7	00:02.7	00:02.2	00:03.0	4,914

time unit: mm:ss.s

Attributes					3DCityDB v.4			3DCityDB v.5			Number of object selected
Dataset	Feature(CO)		LoD	Geometry	View	Materialized View		View	Materialized View		
					Query time	Refresh time	Query time	Query time	Refresh time	Query time	
Vienna	Thematic surface	ground_surface	2	Multi Surface	00:27.5	01:12.1	00:01.6	00:18.8	00:36.2	00:01.5	477,243
		roof_surface	2	Multi Surface	00:36.5	01:27.0	00:02.6	00:52.4	01:41.0	00:02.6	1,183,183
		wall_surface	2	Multi Surface	00:39.3	02:11.0	00:09.8	00:31.1	02:16.0	00:08.6	5,046,343
	Vegetation (trees)		2	Implicit Representation	00:08.9	00:17.2	00:01.4	00:03.8	00:11.0	00:01.3	187,359
	Relief (feature)		2	Surface	00:00.1	00:00.2	00:00.2	00:00.1	00:00.1	00:00.1	1,815
			3	Surface	00:00.1	00:00.1	00:00.4	00:00.1	00:00.1	00:00.1	1,815
	Relief (Component)		2	tin	01:06.8	02:10.8	00:06.0	00:09.3	00:07.7	00:05.8	1,815
			3	tin	01:28.4	03:01.1	00:15.3	00:16.7	00:09.6	00:13.8	1,815
			4	tin	02:01.4	04:16.8	00:28.0	00:30.9	00:35.3	00:25.6	1,815

time unit: mm:ss.s

Table 2: Query on view and materialized view execution time comparison (Rijsen-Holten & Vienna dataset)



Task	Start date	End date
Preliminary research	9/25/23	12/11/23
P1 - Progress review Graduation plan	11/17/23	11/17/23
Familiarise the 3DCityDB v.5	11/20/23	12/8/23
Test layer creation with 3 classes	12/4/23	12/29/23
Preliminary layer query execution time test	1/1/24	1/19/24
Prepare P2 documentation and presentation	12/25/23	1/23/24
P2 - Formal assessment Graduation plan	1/24/24	1/24/24
Functionality development	11/20/23	5/22/24
Develop layer creation process to target classes	1/24/24	3/20/24
P3 - Midterm progress meeting	3/20/24	3/20/24
Plugin adaptation	3/20/24	5/21/24
Stability Testing	5/6/24	5/21/24
Prepare P4 documentation and presentation	4/29/24	5/21/24
P4 - Go/no-go (formal assessment)	5/22/24	5/22/24
Finalise thesis	5/23/24	6/7/24
Prepare P5 presentation	5/27/24	6/23/24
P5 - Public presentation and final assessment	6/24/24	6/24/24

Figure 14: Gannt diagram of the thesis

7 Tools and datasets used

7.1 Tools

For this research, Python v.3.12 is selected as the programming language executed with Visual Studio Code as the development environment. Python is used in combination with PostgreSQL v.15 supported by the pgAdmin 4 for data processing executing SQL query for testing. The development of the 3DCityDB is going to be bound on QGIS 3.34.2-Prizren, 3DCityDB v.0.6.0-beta and 3DCityDB-Tools v.0.8.7. These versions are chosen based on their stability and are widely adopted. The operating system in which all the above tools are installed and the plugin is developed is macOS Ventura v.13.6.3.

7.2 Datasets

For the datasets that are going to be used in this research is the complete collection of buildings in the wider area of Rijsen-Holten, which is derived from the spatial portal of 3D Basisregistratie Adressen en Gebouwen (BAG) in the Netherlands. The 3D city model of Rijsen-Holten comes in GML format and is imported into 3DCityDB via its command line tool (v.0.6.0-beta). Moreover, other open datasets are going to be used for test purposes, such as the 3D models of Vienna available via Open Government Data Wien catalogue in Austria. Lastly, since this research is working on the CityGML v.3.0, the GML data stored with the new standard provided by the other sources is going to be tested if it is available.

References

Agugiaro, G., Pantelios, K., León-Sánchez, C., Yao, Z., and Nagel, C. (2024). Introducing the 3DCityDB-Tools plug-in for QGIS.

Agugiaro, G., Pantelios, K., Mbwanda, T., and Sánchez, C. L. (2023). 3DCityDB-Tools-for-QGIS. `https://github.com/tudelft3d/3DCityDB-Tools-for-QGIS`.

Gröger, G., Kolbe, T. H., Nagel, C., and Häfele, K.-H. (2012). OGC city geography markup language (CityGML) encoding standard.

Kolbe, T. H., Kutzner, T., Smyth, C. S., Nagel, C., Roensdorf, C., and Heazel, C. (2021). OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard. `http://www.opengis.net/doc/IS/CityGML-1/3.0`.

Nagel, C., Willenborg, B., Yao, Z., and Schwab, B. (2023). citydb-tool. `https://github.com/3dcitydb/citydb-tool/releases`.

Open Geospatial Consortium et al. (2021). OGC city geography markup language (CityGML) 3.0 conceptual model users guide. `https://docs.ogc.org/guides/20-066.html`.

Pantelios, K. (2022). Development of a QGIS plugin for the CityGML 3D City Database.

Yao, Z., Nagel, C., Kunde, F., Hudra, Donaubauer, A., Adolphi, T., and 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.