Student thesis series INES nr 666

Methods for storing simulation output in semantic 3D city models

Miras Kozgan

2024

Department of

Physical Geography and Ecosystem Science

Lund University

Sölvegatan 12

Miras Kozgan (2024).

Methods for storing simulation output in semantic 3D city models.

Metoder för att lagra simuleringsresultat i semantiska 3D-stadsmodeller.

Master's degree thesis, 30 credits in *Geography information Systems and Remote Sensing*

Department of Physical Geography and Ecosystem Science, Lund University

Level: Master of Science (MSc)

Course duration: *January* 2024 until *June* 2024

Methods for storing simulation output in semantic 3D city models

Miras Kozgan

Master thesis, 30 credits, in *Geography information systems and remote sensing*

Supervisor
Per-Ola Olsson
Department of Physical Geography and Ecosystem Science, Lund

Supervisor
Karolina Pantazatou
Department of Physical Geography and Ecosystem Science, Lund

Acknowledgments

I would like to thank my supervisors Per-Ola Olsson and Karolina Pantazatou for the feedback and support on every stage of the master thesis. The proposed topic idea was exciting to explore and left so many things to discover in the end. The experience of writing a scientific paper in Lund University after my previous experience was a fresh breath with all freedom I had as an author and guiding hands of my supervisors and, of course, my opponent Felix Hansson.

I am very grateful to my life partner Joana Kartovickaja who always supported me when SQL queries and FME transformers attacked me with errors and warnings. Without her I would have been very sad by the end of this thesis work.

My family was of great support to start this master program and they deserve a mention in this part: thanks to my father Yerlan, mother Saltanat, brother Aizhas and sister Aidana. Each of them helped me in so many ways that it takes another master thesis to describe everything they did.

Thanks to my small birdie pet Garfield, whose cheeping made me do what I have done, and thanks to developers of the app Finch.

I want express appreciation to FME, 3DCityDB, CityGML, PostGIS community members, whose posts and articles gave me clues how to resolve majority of issues I had.

Abstract

Currently, there is no direct support for storing simulation output within semantic 3D city models (CityGML). This master thesis explores a potential methodology for this purpose, with a focus on using appearances of 3D geometries instead of creating generic city objects and attributes or adding an Application Domain Extension (ADE). The caveat with the last two methods is that they require additional changes or extensions to the CityGML schema and introduce interoperability issues. It means that 3D models and simulation output cannot be processed and/or visualized in one software simultaneously, instead this must be done utilizing different software. This workaround increases the processing time of the simulation output. To address these challenges, this thesis proposes a third option: utilizing the 3DCityDB appearance module, which can store surface-related visualization, including non-visible specifications like daylight and sunlight simulation outputs. This method leverages existing CityGML and 3DCityDB structures to integrate simulation outputs without extending the schema, thereby simplifying the process for urban planners or other decision makers while maintaining model efficiency.

The thesis aims to bridge the gap between utilizing extensive simulation data—specifically sunlight and daylight simulation outputs—and the capacity of 3D city models to effectively incorporate this information. By implementing and testing the use of the appearance module, the proposed methodology demonstrates its feasibility for storing, querying (SQL), and visualizing simulation outputs.

The thesis work concludes that the appearance module of CityGML can be used to store sunlight and daylight simulation data in 3DCityDB successfully. The integration process has been validated, ensuring accurate implementation within the CityGML schema. Additionally, the thesis work confirms that querying and retrieving appearance data may simplify established work practices related to the urban planning process, further promoting the utilization of CityGML models in this branch. This advancement provides a valuable tool for decision-makers, particularly in assessing compliance to daylight regulations at different stages of the urban planning process.

Contents

Acknowledgments iv
Abstract v
List of abbreviations vii
List of figures viii
1. Introduction 9
  1.1 Background 9
  1.2 Problem statement 10
  1.3 Aim 11
  1.4 Limitations 12
2. Theoretical background 12
  2.1 CityGML 12
    2.1.1 Overview 12
    2.1.2 Geometry model in CityGML 13
    2.1.3 Level of detail (LOD) in CityGML 15
  2.2 3DCityDB 16
  2.3 Appearance module 16
  2.4 Daylight and sunlight simulation outputs 18
    2.4.1 Obstruction angle 19
    2.4.2 Direct Sun Hours (DSH) 20
    2.4.3 Total Annual Solar Irradiance 21
3. Methods 22
  3.1 Obstruction Angle and Direct Sun hours 22
  3.2 Total Annual Solar Irradiance 26
  3.3 Direct Sun Hours 29
4. Results 29
  4.1 Created CityGML models 29
  4.2. Querying appearances from a 3DCityDB 30
5. Discussion 34
6. Conclusion 37
7. References 37

List of abbreviations

0-3D	0-3-Dimensional
3DCityDB	3-Dimensional City Database
ADE	Application Domain Extension
CityGML	City Geography Markup Language
CityJSON	City JavaScript Object Notation
DSH	Direct Sun Hours
EN 17037	European Standard 17037
FME	Feature Manipulation Engine
GIS	Geographic Information System
GML	Geography Markup Language
ID	Identifier
LOD	Level of detail
OA	Obstruction Angle
OGC	Open Geospatial Consortium
PostgreSQL	Postgre Structured Query Language
RDBMS	Relational database management system
UUID	Universally Unique Identifier
XML	Extensible Markup Language

List of figures

Number	Name	Page
Figure 1.1a	Example of single value sunlight simulation output	10
Figure 1.1b	Example of grid data sunlight simulation output	10
Figure 1.2	CityGML module overview.	11
Figure 2.1	Appearances of buildings in CityGML 2.0	13
Figure 2.2	UML diagram of City GML's geometry model	14
Figure 2.3	Boundary Representation used for object modelling and corresponding graph	15
Figure 2.4	Geometry complexes and aggregates	15
Figure 2.5	Level of detail in CityGML	16
Figure 2.6	Appearances database schema	17
Figure 2.7	Different appearances applied to surfaces	18
Figure 2.8	Obstruction angle	20
Figure 2.9	OA simulation output example	20
Figure 2.10	Direct Sun Hours example	21
Figure 2.11	Total annual solar irradiance example in Rhino environment	22
Figure 3.1	Workflow of implementation OA as an appearance module	23
Figure 3.2	FME workflow of processing the simulation output	23
Figure 3.3	FME workflow of processing wall surfaces geometries	24
Figure 3.4	FME workflow of creating roof surfaces	25
Figure 3.5	Writing CityModel and Building to the model	25
Figure 3.6	Turning facade grids into an appearance	26
Figure 3.7	First section of a FME workflow related to wall surfaces	26
Figure 3.8a	Applying one raster as an appearance to a simple wall	27
Figure 3.8b	Applying one raster as an appearance to a complex wall	27
Figure 3.9a	Custom transformers *gml_finder*	27
Figure 3.9b	Custom transformer *rastercreator*	27
Figure 3.10	Custom transformer *Offset_fixer*	28
Figure 4.1	Resulting models with window appearances before importing them into 3DCityDB	29
Figure 4.2	Resulting models with window appearances after importing them into 3DCityDB	30
Figure 4.3.	Visualization issues in FZK Viewer	30
Figure 4.4	Required tables to query to receive appearances from 3DCityDB	31
Figure 4.5	FME workflow to extract geometries with appearances from a 3DCityDB	32
Figure 4.6	The result of the created workflow	32
Figure 4.7.	Result of the modified query to show specific values of OA	32
Figure 4.8	Required tables to retrieve wall appearances from a 3DCityDB	33
Figure 4.9	FME workflow to visualize exported geometries and appearances	33
Figure 4.10a	Appearances retrieved from the 3DCityDB by SQL Queries	34
Figure 4.10b	Appearances exported from 3DCityDB with the 3DCityDB Importer/Exporter tool	34
Figure 5.1	Appearance distortions	35
Figure 5.2	Appearance orientation issues	35

1. Introduction

1.1 Background

With the growing rate of urbanization leading to changes in global landscape, planning cities more efficiently is of a crucial importance. In this context, semantic three-dimensional (3D) city models have become an impactful tool. They can be described as detailed digital representations of urban objects (including buildings, streets, parks, bridges, etc.) combining geometries with additional semantic data (e.g., façade material or color, year of construction, etc.). Digital models are used in different urban applications, from environmental impact analysis to civil infrastructure management, functioning as a basis for the simulation and visualization of a complex urban-landscape system (Biljecki et al., 2015). However, the efficiency of these models in representing the dynamic nature of complex urban landscapes is somewhat lacking when it comes to data management strategies for e.g., storing simulation outputs (Biljecki et al., 2018). Having results from different simulations stored within a 3D model offers the opportunity to execute queries and obtain results without the need to use separate software for processing different types of data formats. This will enable urban planners to use simulation results from different simulations multiple times after getting them from consultants without additional software costs. Providing a way to store simulation results within 3D models can potentially aid urban planners in making more informed decisions when it comes to sustainable urban planning. In addressing these challenges, the exploration of standardized ways of storing simulation output in semantic 3D city models becomes impactful, laying the foundation for enhanced data management of simulation outcomes.

City Geography Markup Language (CityGML; Open Geospatial Consortium, n.d.; Kutzner et al., 2020) serves as an open standard for the storage of semantic 3D city models. Versions 1.0 and 2.0 are built upon Extensible Markup Language (XML), utilizing geometries based on Geography Markup Language (GML) alongside additional features.

The most recent version, CityGML 3.0, introduced in 2023 by the Open Geospatial Consortium (OGC), expands its capabilities beyond GML to include JSON-encoded schemas (OGC, 2023). This evolution allows for greater flexibility in data representation and interoperability, aligning with contemporary standards and practices in data management.

Moreover, it's important to mention CityJSON (City JavaScript Object Notation), a format that has implemented many functionalities of CityGML 2.0 (Ledoux et al., 2019). Notably, CityJSON is also an OGC community standard (CityJSON OGC web page, n.d.). Unlike CityGML, which uses GML for storing geometries and XML for semantic data representation, CityJSON uses JSON for semantic information and GeoJSON for geometries.

It's worth noting that deploying CityGML and CityJSON models effectively, necessitates specialized environments to efficiently manage and utilize the data. This is attributed to their complexity and richness in representing urban environments. 3D City Database (3DCityDB) is an open-source software geo database for storing, representing, and managing CityGML models. This database supports several database types such as PostgreSQL, Oracle, PolarDB (3DCityDB documentation, n.d.) and the execution of spatial queries. The structure of the database aligns with the CityGML standard including geometry, semantic information, appearance, building, land use, vegetation, water bodies and other modules with spatial indexing and extensions to features. CityJSON models are also supported by 3DCityDB (3DCityDB Documentation, n.d.).

Daylight simulation recommendations are stated in the European standard EN 17037:2018 and guide urban planners on how to estimate daylighting conditions in planned buildings. Ensuring adequate daylight access is beneficial not only for citizens, e.g., mental health, sleep latency, job burn-out, alertness, etc. (Aries et al. 2013, Borisuit et al. 2015) but may also bring economic benefits through increased productivity of workers and increasing value of real estate (Turan et. al, 2020; Yang & Nam, 2010). Simulating daylight conditions over 3D city models provides urban planners and authorities with more information that can assist them in making better decisions. Storing the simulation results in the 3D city model and making them easily accessible, aids decision-makers assessing daylight conditions in multiple planned buildings. Spotting lighting problems beforehand and solving them may be beneficial for all stakeholders. There is also the option of storing simulation results from different types of simulations (e.g., daylight, noise, flood) in the same 3D city model. Storing simulation outputs with options to utilize these data will help these decision-makers to assess daylight and/or noise conditions of planned buildings simultaneously. Enabling quick access to the data with visualizations of simulation results, may highlight "daylight access"-related issues before a point of no return.

In this thesis work, two types of simulation output were utilized: daylight and sunlight simulations. These are commonly required at multiple stages of the urban planning process (Kanters & Wall, 2014). Sunlight simulations are a qualitative and quantitative estimation of direct sunlight incident on a surface (e.g., window, façade, etc.), while daylight simulations assess both direct and reflected sunlight. The common format of results from these simulations are either a single value related to the geometry of a certain part of a building (e.g., a window) (figure 1.1.a) or a regular grid of values placed on the surface of different building parts (e.g., windows, facades, or roofs) (figure 1.1.b). The utilization of two distinct formats of outputs poses a notable challenge that requires the implementation of different methods for storing them.

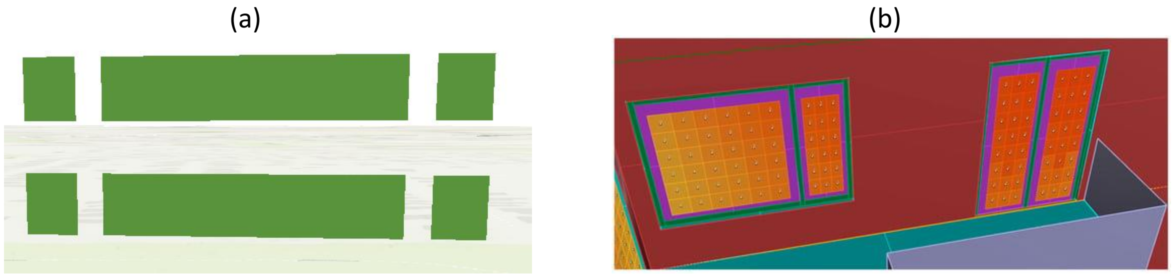


Figure 1.1. a) Example of single value sunlight simulation output (done by author) and b) Example of grid data sunlight simulation output (Source: Ericson et al. 2019).

1.2 Problem statement

At present, there is no direct support for storing simulation output in CityGML 3D models. However, by studying the structure of the CityGML schema, one could identify two possible ways: (1) using *generic city object* and *attributes* or (2) by adding an *Application Domain Extension* (ADE). Nevertheless, both ways have shortcomings (Uggla et al., 2022). Generic city objects and attributes (i.e., manually created objects and attributes within a 3DCityDB extending a CityGML model) do not demand a schema; therefore, these objects and attributes cannot be validated against a schema. An ADE is added as a new schema and it allows to create own classes and attributes, but it creates interoperability and data exchange issues, as classes and attributes are not standardized. It also increases the complexity and the size of a 3D model, considering the condition that there might be a need to store several sets of simulation data in a model, e.g., noise and daylight or sunlight simulations of a model. Developing software applications for reading, storing, and processing 3D city models with stored simulation output according to either one of the aforementioned options would be a cumbersome and time-consuming task. It is therefore very likely that no such applications would be developed, something that would repeat the observations of Noardo et al. (2021) regarding CityGML and what Biljecki et al. (2015) have observed for CityGML ADEs. To overcome these issues, it is worth considering a third option for storing simulation output in semantic 3D city models. This option would include the use of the CityGML *appearance* module which can be stored in 3DCityDB as part of a 3D city model.

The appearance module (figure 1.2) is used to store information on object surfaces. It relates not only to visible surface characteristics like color or texture, but also to non-visible specifications as infrared radiation (3DCityDB, n.d.). It may be a promising way of integrating simulation outputs within models since the appearance model stores 2D array information of color and texture. So, we could borrow this structure and use it for storing 2D array simulation output instead of color or texture. Therefore, existing visualization formats, classes, and objects of 3DCityDB may be used without extending CityGML. This could also simplify the process for developers to extend their software to support importing and visualizing simulation outputs stored this way.

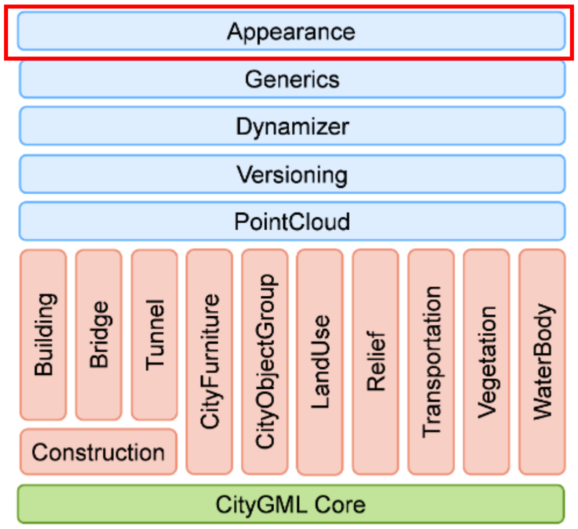


Figure 1.2. CityGML module overview. Source: 3DCityDB documentation, n.d.

This master thesis is addressed to answer the need for creating simulation results storage methodologies within the framework of CityGML models. It tries to bridge the existing gap between the expansive data produced by sunlight and daylight simulations and the capability of 3D city models to accommodate this information effectively. The thesis will focus on implementing and testing the proposed mechanism of storing data.

1.3 Aim

The primary aim of this study is to assess the feasibility of storing simulation output within the appearance module of a CityGML model in 3DCityDB. The thesis questions are:

1. Can the appearance module in CityGML be used for storing the datasets generated by daylight and sunlight simulations?

2. What are the steps required to integrate simulation outputs into the appearance module of 3DCityDB?
3. Is it possible to efficiently query and retrieve simulation output stored as appearances in a 3DCityDB for analysis and visualization purposes?

1.4 Limitations

The thesis is limited by the following factors:

1. The study focuses on daylight and sunlight simulation outputs, which may not be generalizable to other types of simulation data (e.g., noise, flood simulations). However, since daylight access is regulated by Swedish law, daylight simulations are a mandatory part of the urban planning process. Future research should also consider simulation output data formats of other types of simulations whose execution is mandatory in the urban planning process (e.g., noise, flood, landslide, etc.).
2. The thesis concentrates on a limited set of daylight and sunlight simulation outputs. This selection was driven by the daylight and sunlight simulation metrics specified in the Swedish legislation and the Swedish and European recommendations.
3. The choice of 3DCityDB over other relational database management systems (RDBMS) was influenced by its specific support for CityGML and its capabilities for spatial data management. While 3DCityDB is a powerful tool, other RDBMS solutions need special extensions to support CityGML, increasing difficulty of the proposed implementation.
4. The implementation is limited to the current version of 3DCityDB. Future versions may offer additional functionalities or improvements that are not considered in this study.
5. The performance and scalability of querying and visualizing large datasets within 3DCityDB have not been tested, which may impact the efficiency of the proposed solution in real-world scenarios.
6. Although CityJSON is supported by 3DCityDB and offers benefits such as simplified JSON and GeoJSON encoding, this study focuses on CityGML. The decision to use CityGML was based on its widespread adoption and comprehensive feature set as well as because the new semantic 3D city model national profile for Sweden, 3CIM, is based on it (Uggla et al., 2023). In any case, it should be noted that CityJSON could also be a viable option for similar applications (3DCityDB Importer/Exporter, n.d.; 3DCityDB, n.d.).
7. This thesis only deals with the representation of simulation output on semantic 3D city model geometries that correspond to the same LOD as the one used in the simulation that produced them.

2. Theoretical background

This chapter provides a theoretical background for the thesis, introducing concepts of daylight simulation outputs which will be used to achieve the aim of the study. Furthermore, it explains the structure of CityGML, 3DCityDB, and CityGML's appearance module.

2.1 CityGML

2.1.1 Overview

In 2008, the Open Geospatial Consortium (OGC) announced CityGML as an open standard for representing and exchanging 3D urban models (Gröger et al., 2012). Since then, CityGML was developed further, leading to the release of CityGML 2.0 in 2012 and CityGML 3.0 in 2020 (Kutzner et al. 2012,2020). CityGML enables the representation of urban structures' geometries, themes, and visual attributes (Gröger and

Plümer, 2012). Now, CityGML is widely used to present the thematic and semantic attributes of urban systems, including buildings and their constituents (roofs, walls, windows, etc.), terrains, water bodies, and vegetation (Gröger and Plümer, 2012). CityGML not only defines the semantic attributes of objects but also delineates their relationships, such as between a building and the underlying terrain. Additionally, CityGML data is typically stored in a relational database (e.g., 3DCityDB) to facilitate efficient data management and querying.

CityGML provides clear definitions for semantic information, attributes, relationships, and 3D geometrical representations for all urban objects, including buildings (Gröger and Plümer, 2012). CityGML is organized into modules, allowing for the combination of different modules to meet specific use-case requirements (Gröger and Plümer, 2012). Apart from CityGML's *Building* theme, which has been extensively utilized in a variety of applications, the remaining themes have primarily been used for visualization purposes and only sparsely been used in other applications. (Biljecki et al., 2015).

CityGML enables the description of building objects with visual appearances on their outer boundaries, such as walls or roofs (Gröger and Plümer, 2012). For each geometry (roof, wall) a different appearance (e.g., see figure 2.1) is stored in a separate table of a 3DCityDB. These appearances are represented through various types of categorical data (e.g., low, medium, or high pollution) used in analytical tasks, including noise pollution, sun exposure, wind simulations, shadow cast simulations, and energy demand estimations (Biljecki et al., 2015). These capabilities allow 3D city models to not only serve visualization purposes but also facilitate further analysis of urban structures in space.

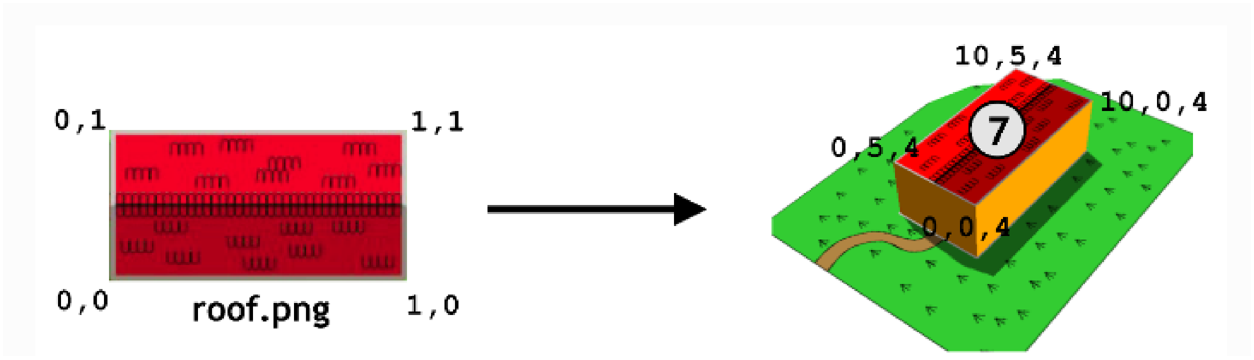
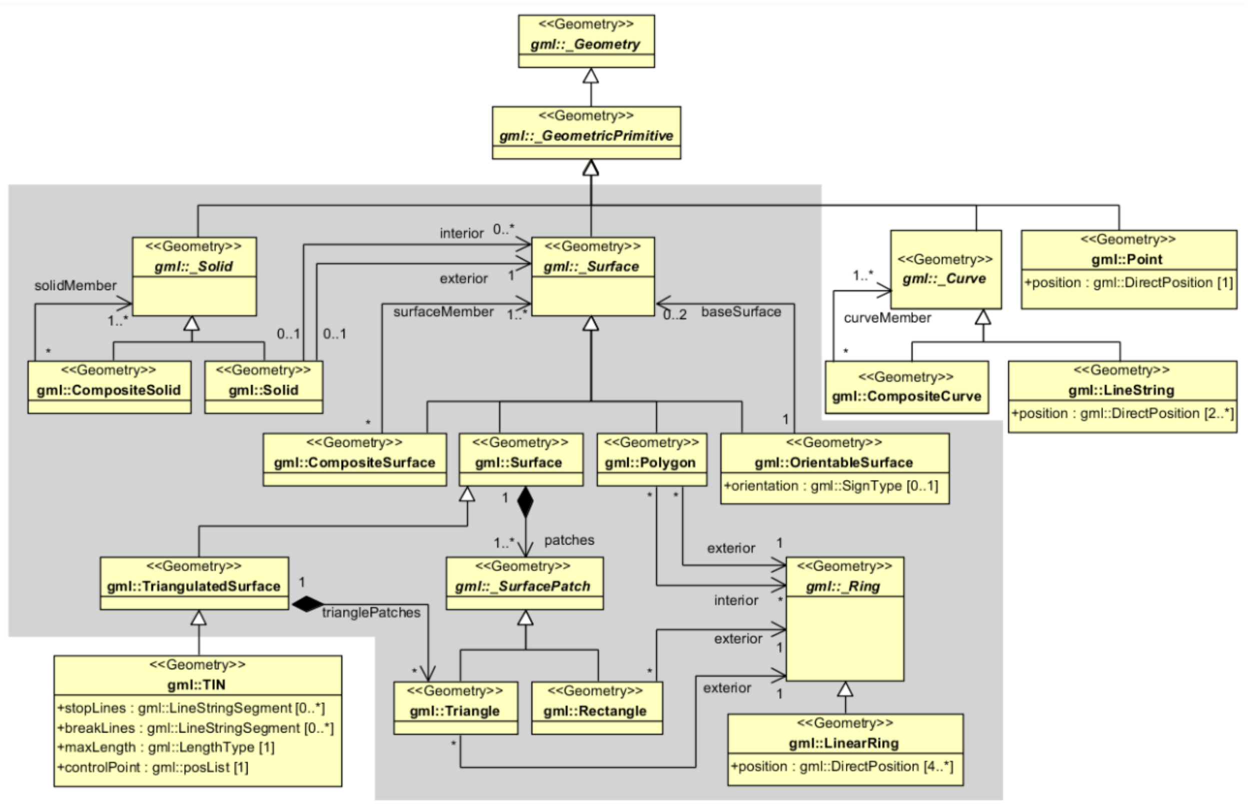


Figure 2.1. Appearances of buildings in CityGML 2.0. Source: 3DCityDB Documentation, n.d.

Various use-cases in 3D city models demand functionalities beyond the scope of CityGML (Gröger and Plümer, 2012). These specific cases require additional attributes, feature types, or relationships not provided by CityGML. To address this, CityGML introduced the Application Domain Extension (ADE), a customizable application schema allowing users to extend the functionality of the standard CityGML schema (Biljecki et al., 2018). By incorporating new relationships and attributes, users can create tailored feature types in CityGML to suit specific purposes (Kumar et al., 2017). Furthermore, CityGML supports the simultaneous use of multiple ADEs, enhancing the versatility of 3D city models (Biljecki et al., 2018).

2.1.2 Geometry model in CityGML

Spatial characteristics of features in CityGML are based on objects of GML 3 geometry model, which was developed upon the spatial schema of the standard ISO 19107. It starts with geometric primitives: a zero-dimensional point, a one-dimensional curve with height and a two-dimensional surface having length and width (figure 2.2).



<<Geometry>> gml::_Geometry
<<Geometry>> gml::_GeometricPrimitive
<<Geometry>> gml::_Solid
<<Geometry>> gml::_Surface
<<Geometry>> gml::_Curve
<<Geometry>> gml::Point
+position : gml::DirectPosition [1]
<<Geometry>> gml::CompositeSolid
<<Geometry>> gml::Solid
<<Geometry>> gml::CompositeCurve
<<Geometry>> gml::LineString
+position : gml::DirectPosition [2..*]
<<Geometry>> gml::CompositeSurface
<<Geometry>> gml::Surface
<<Geometry>> gml::Polygon
<<Geometry>> gml::OrientableSurface
+orientation : gml::SignType [0..1]
<<Geometry>> gml::TriangulatedSurface
<<Geometry>> gml::_SurfacePatch
<<Geometry>> gml::_Ring
<<Geometry>> gml::TIN
+stopLines : gml::LineStringSegment [0..*]
+breakLines : gml::LineStringSegment [0..*]
+maxLength : gml::LengthType [1]
+controlPoint : gml::posList [1]
<<Geometry>> gml::Triangle
<<Geometry>> gml::Rectangle
<<Geometry>> gml::LinearRing
+position : gml::DirectPosition [4..*]
solidMember
interior
exterior
surfaceMember
baseSurface
curveMember
patches
trianglePatches
0..*
1..*
0..1
0..2
1
*

Figure 2.2. UML diagram of City GML's geometry model. Source: CityGML specifications, OGC, 2006.

The representation of geometry for solids is defined by the Boundary Representation (B-Rep, Foley et al., 1995). B-Rep is a method for describing the shape and structure of a 3D object. In this representation, a solid object is defined by its surface boundaries, which include faces, edges, and vertices (figure 2.3):

- A face is a flat or curved surface that forms part of the boundary of a solid object. Each face is typically defined by a closed loop of edges.
- An edge is a line segment where two faces meet. Edges define the boundaries of faces and are essential for constructing the overall shape of the object.
- A vertex is a point where edges meet. Vertices serve as the corner points of the object and are crucial for defining the geometry of the edges and faces.

This method allows for precise and detailed modeling of complex shapes by focusing on the external surfaces of the object rather than its internal structure. Figure 2.3 illustrates the components of B-Rep, showing how faces, edges, and vertices are used to define the geometry of a solid.

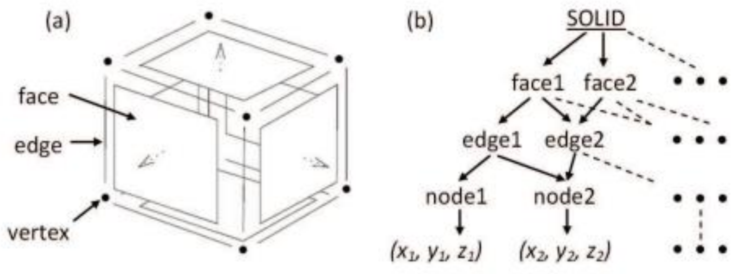


Figure 2.3. Boundary Representation used for object modelling and corresponding graph. Source: Lou, 2011

By using B-Rep, CityGML can accurately represent the geometry of various urban features, ensuring that the models are both detailed and true to their real-world counterparts.

Solid is bounded by surfaces, curve by a line, and *Surface* is represented by Polygons. A combination of primitives may be used to create such complexes and aggregates as *MultiSolid*, *MultiSurface*, *MultiCurve*, *MultiPoint* to *GeometricComplex*, *GeometricPrimitive* and *AbstractGeometricAggregate* (figure 2.4).

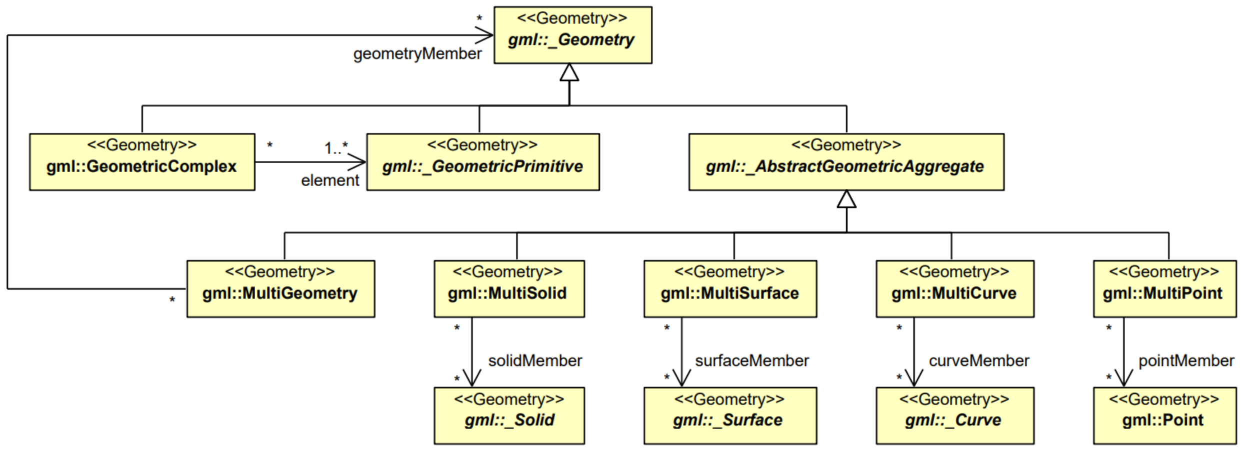

<<Geometry>> gml::_Geometry
geometryMember
<<Geometry>> gml::GeometricComplex
element
<<Geometry>> gml::_GeometricPrimitive
<<Geometry>> gml::_AbstractGeometricAggregate
<<Geometry>> gml::MultiGeometry
<<Geometry>> gml::MultiSolid
<<Geometry>> gml::MultiSurface
<<Geometry>> gml::MultiCurve
<<Geometry>> gml::MultiPoint
solidMember
surfaceMember
curveMember
pointMember
<<Geometry>> gml::_Solid
<<Geometry>> gml::_Surface
<<Geometry>> gml::_Curve
<<Geometry>> gml::Point

Figure 2.4. Geometry complexes and aggregates. Source: CityGML specifications, OGC, 2006.

For example, a building can be represented by defining its surfaces (walls, roof, etc.) using polygons. These surfaces are bounded by edges, which in turn are defined by vertices.

2.1.3 Level of detail (LOD) in CityGML

The CityGML 2.0 standard divides 3D building models into five levels of details (LOD) (Figure 2.5). Gröger & Plümer, (2012) describe each LOD in detail. They start with LOD0 as the 2D footprint of the building. A building object in LOD1 is represented as a 3D box model using the building footprint as base and the roof height as height. LOD 2 is more complex because of additional *RoofSurface*, *WallSurface* and *GroundSurface* objects added to the LOD 1 representation. 3D building representation in LOD 3 contains information of façade elements: doors and windows which are included as features of roof and walls. Buildings represented in LOD 4, also contain elements of the building interior (e.g., internal walls, floors, etc.).

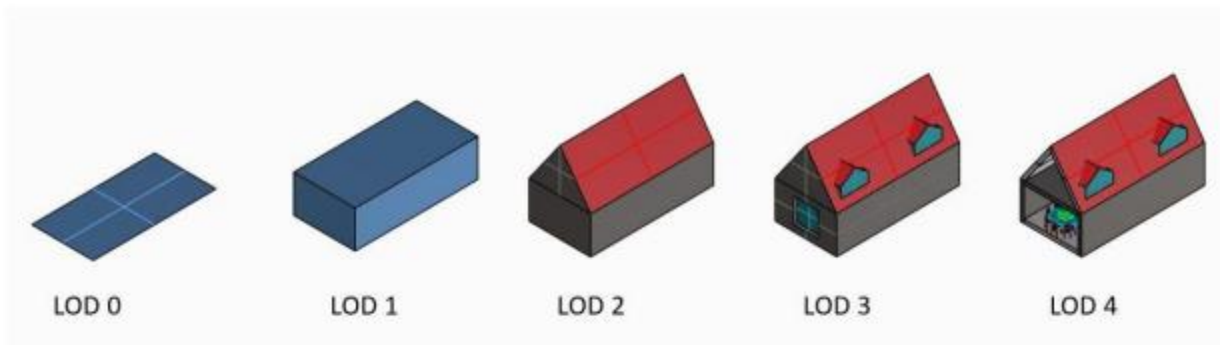


Figure 2.5. Level of detail in CityGML. Source: Hartman K., 2022

2.2 3DCityDB

The 3D City Database (3DCityDB) is an Open-Source extended database schema that aligns with the CityGML standard (Kolbe et al., 2019). It functions as a supplementary database schema within a spatially enhanced relational database management system (SRDBMS) (Kolbe et al., 2019), and accommodates two database systems: the commercial Oracle and the Open-Source PostgreSQL/PostGIS (Kolbe et al., 2019).

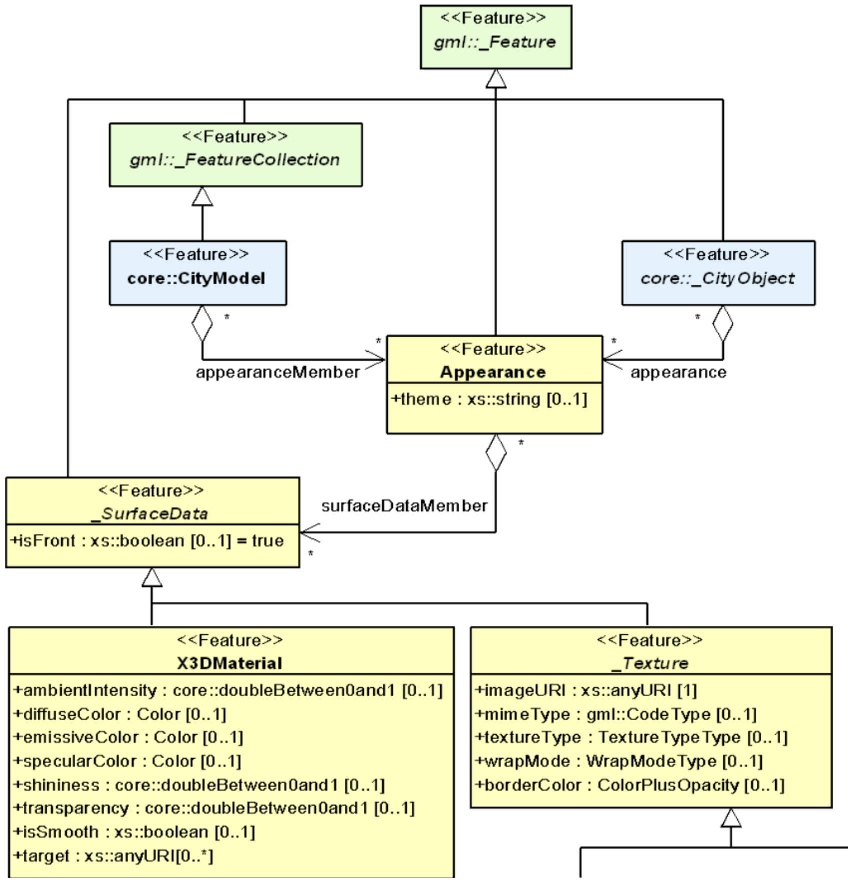
Aligned with the CityGML 2.0 standard, 3DCityDB's primary function is to store CityGML models of varying sizes, at different levels of details (LOD), and geometries (Kolbe et al., 2019). This functionality is facilitated through a graphical user interface (GUI), featuring tools for importing, exporting, examining, and managing stored city models. The database schema currently supports CityGML 1.0 and 2.0, but future implementation of CityGML 3.0 is anticipated (Kutzner et al., 2020).

Offering additional support for users of 3D city models, 3DCityDB provides tools for exporting data to Keyhole Markup Language (KML), COLLAborative Design Activity (COLLADA), and the GL Transmission format (glTF) (Barnes and Finch, 2008; Burggraf, 2015; Bhatia et al., 2017). These formats are primarily utilized in 3D applications for visualization and exchange, and can also be employed in Google Earth, ArcGIS explorer, and Cesium (Kolbe et al., 2019). Moreover, 3DCityDB manages additional extensions to CityGML files by creating custom Application Domain Extensions (ADEs).

2.3 Appearance module

The *Appearance* module is used to store data about object surfaces. It relates not only to visible surface characteristics like color or texture, but also to non-visible specifications as, for example, noise simulation or infrared radiation (3DCityDB Documentation, n.d.). The Appearance table is stored in 3DCityDB on top of the CityGML core (figure 1.2). The UML diagram of the module is shown in figure 2.6.

The table contains details regarding the surface characteristics of objects, which are stored in the DESCRIPTION attribute, while their corresponding categories are stored in the THEME attribute. For instance, the DESCRIPTION attribute might hold information about the texture or color of an object's surface, while the THEME attribute categorizes it by labeling it as "urban," "natural," or "industrial." This classification helps in organizing and identifying objects based on their visual attributes. Every city object has its appearance information stored, facilitated by the *Appearance* table, which establishes relationships with the *CityObject* and *CityModel* base classes through foreign keys. In addition to its role as a feature class, the Appearance class enables referencing via GML identifiers such as *GMLID* within the table, which makes it possible to query it directly within a 3DCityDB.



<<Feature>>
gml::_Feature
<<Feature>>
gml::_FeatureCollection
<<Feature>>
core::CityModel
<<Feature>>
core::_CityObject
appearanceMember
<<Feature>>
Appearance
+theme : xs::string [0..1]
appearance
<<Feature>>
_SurfaceData
+isFront : xs::boolean [0..1] = true
surfaceDataMember
<<Feature>>
X3DMaterial
+ambientIntensity : core::doubleBetween0and1 [0..1]
+diffuseColor : Color [0..1]
+emissiveColor : Color [0..1]
+specularColor : Color [0..1]
+shininess : core::doubleBetween0and1 [0..1]
+transparency : core::doubleBetween0and1 [0..1]
+isSmooth : xs::boolean [0..1]
+target : xs::anyURI[0..*]
<<Feature>>
_Texture
+imageURI : xs::anyURI [1]
+mimeType : gml::CodeType [0..1]
+textureType : TextureTypeType [0..1]
+wrapMode : WrapModeType [0..1]
+borderColor : ColorPlusOpacity [0..1]

Figure 2.6. Excerpt of appearance module. Source: 3DCityDB Documentation, n.d.

The SURFACE_DATA table contains various data types corresponding to each surface. These data types are then linked to specific geometries referenced within the APPEARANCE table. The SURFACE_DATA table has the following properties:

1) IS_FRONT describes the side a surface data object applies to (1: front face; 0: back face).
2) The *OBJECTCLASS_ID* column denotes if materials (just a simple color with specific properties, e.g., how light is diffused by a surface, as walls are presented in the figure 2.7 of the building on the left) or textures (e.g., bricks of the building on the right in fig. 2.7) are used for the visual representation of the object of the class.
3) Materials are specified by the attributes X3D_xxx which define its graphic representation. Details on using georeferenced textures, such as orientation and reference point, are contained in attributes GT_xxx (figure 2.6).
4) TEX_IMAGE_ID is an ID of a raster representation of a surface texture (for example, taking a photo of a building and applying it to the building).

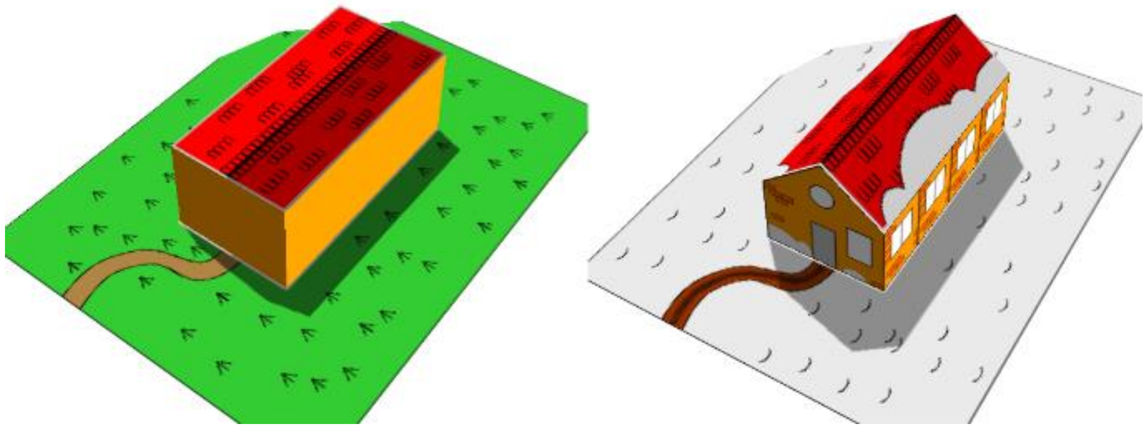


Figure 2.7. Different appearances applied to surfaces. Source: 3DCityDB Documentation, n.d.

There are several applications of appearance implementation with a combination of semantic data in 3D models. The project done by Law et al. (2006) introduced using noise simulation outputs as a graphical representation of buildings in 3D model, however it was done for a GIS based 3D model and not for a CityGML model. The second approach was done by Kolbe (Kolbe et al., 2015), where RGB and infrared photos of buildings were used as textures. Unlike simulation outputs, these textures serve as direct representations of actual facades within the built environment. There is a knowledge gap regarding the absence of articles addressing the utilization of simulation outputs as appearances for buildings within CityGML models which are possible to query in 3DCityDB.

2.4 Daylight and sunlight simulation outputs

Daylight conditions in Sweden are regulated by the building regulations of National Board of Housing, Building and Planning (Boverket, 2011). These rules apply to frequently used rooms, regulating adequate access to sunlight or daylight. It's suggested that these rooms should have at least one window so the daytime and seasonal changes can be followed by inhabitants.

Urban planners in Sweden seldom consider sunlight access in outdoor environments and building facades or incorporate active use of daylight (e.g., solar systems) in their planning processes (Kanters et al., 2021). This is attributed to a lack of legislation, established practices, input data, and suitable tools (Kanters et al., 2021). The new European standard for daylight simulations (EN 17037:2018, SS-EN 17037:208 in Sweden) suggests recommendations on daylight conditions, including solar access in rooms and the duration of direct sunlight. These recommendations do not state strict regulations, leaving room for interpretation. Additionally, it's essential to consider the latitude of the study area when conducting daylight simulations. In northern countries like Sweden, the sun angle remains very low for a significant portion of the year, resulting in long shadows and highlighting the significance of reflectance from one building to another for indoor daylighting.

Daylight simulation tools based on CAD and BIM are prevalent (e.g., Jakica, 2017) and support not only single-building simulation but also small neighborhood simulations. On the contrary, 2.5D or 3D GIS tools, e.g., a Sun-shadow volume tool for ArcGIS Pro (ArcGIS Pro Documentation, n.d.) and for the open-source QGIS Urban Multi-scale Environmental Predictor (QGIS UMEP SEBE Documentation, n.d.), which seldom incorporate interior daylight simulations, are often employed to cover multiple buildings over larger areas.

3D city models are considered suitable input data for daylight simulations as they potentially encompass all required geographical information. However, depending on the LOD of a model, different outputs might be generated, because there are more details to influence the output of a simulation (Kanters et al., 2021). In Sweden, most urban planning offices have 3D city models over the municipality in LOD2 and lack essential window information (Harrie et al., 2021). Geometries and reflectance properties of windows for planned buildings may be derived from BIM (W. Huang, Olsson, Kanters, & Harrie, 2020), while for existing buildings, street view pictures (Dogan & Knutins, 2018; Kong & Fan, 2021; Lee & Nevatia, 2004) or previous studies of architectural decisions (Schindler & Bauer, 2003) are utilized to provide window information for daylight simulations.

Light simulations are not complicated in rural areas, because meteorological conditions are the main factor of energy potential (Freitas et al., 2015). However, complexity increases in urban environments due to spatial constraints (e.g., one building blocking light to the other) limiting incoming sunlight. Urban-level simulations must encompass building details, various building types and designs, outdoor areas (open spaces, street layouts, materials, etc.), and reflectance pf surfaces(Freitas et al., 2015). Consequently, multiple simulation tools are often employed simultaneously to conduct comprehensive analyses (SHC, 2021).

In this thesis, the following simulation outputs were used: *obstruction angle* as an estimation of sunlight access, *direct sun hours* as a variable of lighting conditions in rooms, and *total annual solar irradiance* as an amount of solar energy incoming to facades of buildings. These simulation outputs are used to regulate daylight and sunlight conditions of buildings in Sweden (Boverket, 2011).

2.4.1 Obstruction angle

One of the primary daylight simulation outputs addressed in this thesis is the obstruction angle (OA). OA is defined as the angle between the line of an imaginary horizontal plane and the line connecting the middle of a window and the top part of an object positioned directly in front of the window, such as a neighboring building (Swedish Standard SS914201) (figure 2.8). According to Alenius et al. (2019), the obstruction angle dictates the amount of skylight (diffuse light) that penetrates the interior of a building or apartment. For direct sunlight to infiltrate a building, the windows must have unobstructed access to both the sky and sunlight. Consequently, in urban environments, particularly at steep sun angles, reflected light prevails, leading to limited availability of direct sunlight and skylight. For example, a window on a lower floor has higher OA and less access to the direct sunlight than a window located at a higher floor with lower OA and less access to the direct sunlight.

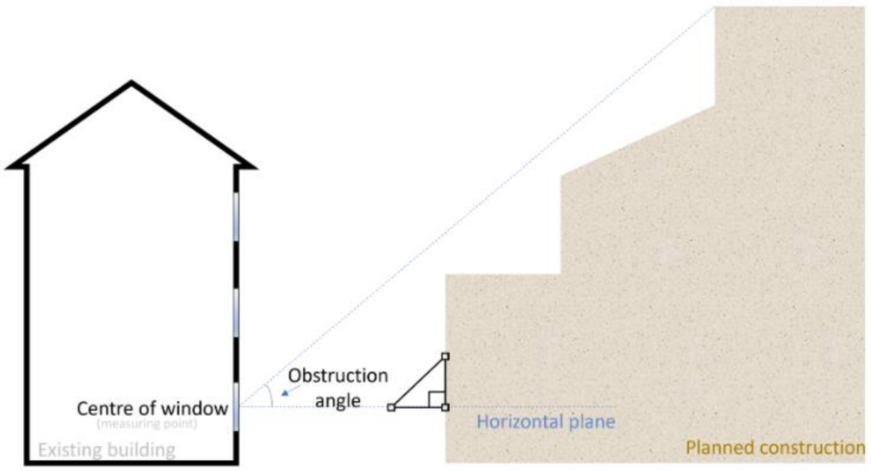


Figure 2.8. Obstruction angle. Source: Pantazatou et al., 2023

The result of the obstruction angle calculation is an obstruction angle attributed to a point located at the centroid of each window. This value can be linked to the geometry of a window (figure 2.9).

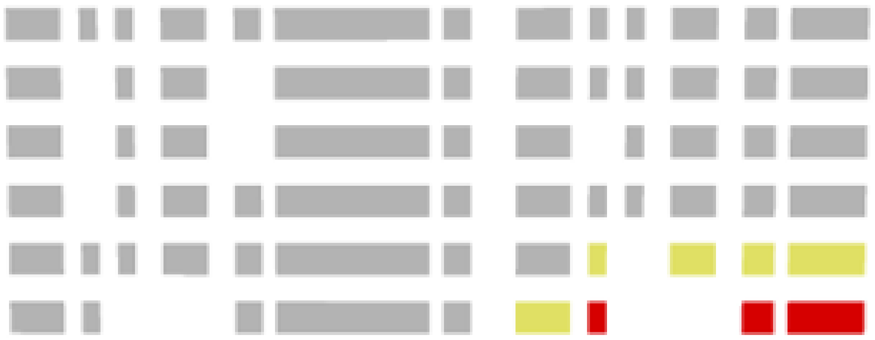


Figure 2.9. OA simulation output example (red: OA over 30 degrees, yellow: between 27 and 29). Source: Daylight simulation exercise report, Miras Kozgan, Christopher James Wear, Daniël Zegeling, 2023)

The simulation output of OA used in this thesis is a polygon ESRI shapefile of with OA values in the attribute table of it.

2.4.2 Direct Sun Hours (DSH)

The term "Direct sun hours" is a sunlight measurement stated in the European Standard EN 17037 – Daylight in Buildings. It is expressed as the total number of hours the Sun directly shines towards a window of a room in a building at a defined height (0.3 m) above the lower part of the frame of a window. The height of a point which DSH are simulated for is 30 cm above a window frame, and it is recommended by the European standard (European Standard, EN 17037). The data used in the thesis was created by executing a DHS simulation within the CAD 3D software environment Rhino (Rhino 3D, n.d.) using the Ladybug (Ladybug, n.d.) and Grasshoper plugins (Pantazatou et al., 2024).

The simulation output is calculated in the following steps:

1. Defining a virtual scene using the latitude of the location and a time of the simulation (March 21st)
2. Calculating a path of the Sun over the scene

3. Adding geometries: both study surfaces (windows and balcony doors) and objects which create obstruction to sunrays (all buildings in the study area)
4. Placing a grid of points with a spatial resolution of 0.1m over apertures (windows).
5. With the Ladybug module Direct Sun Hours calculating the output for each point of the grid and exporting two tables in a .TXT format: coordinates and direct sun hours for each point of the grid
6. Joining both tables together in a GIS software and converting the resulting KML file into a multipatch Shapefile.
7. Compute the DSH for the window centroid.

Example of the raw data produced as output from computing the DSH output is presented in figure 2.10.

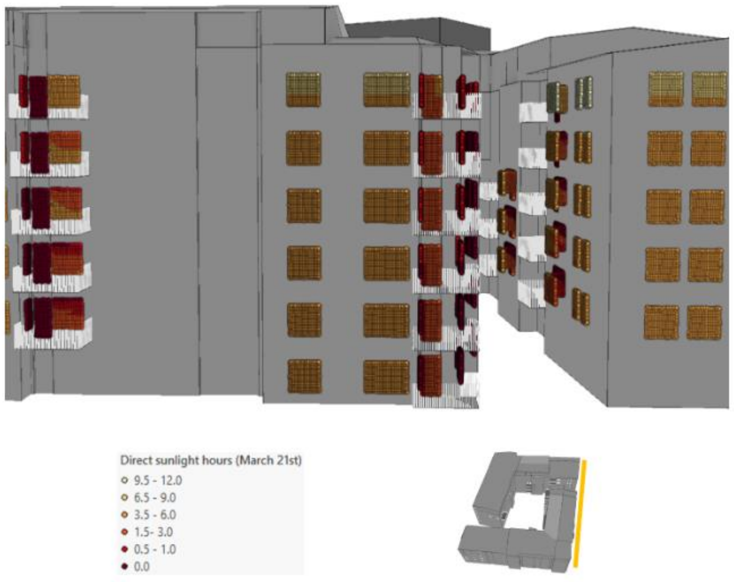


Figure 2.10. Direct Sun Hours example. Source: Pantazatou, 2023

2.4.3 Total Annual Solar Irradiance

Total annual solar irradiance is another simulation output measuring the quantity and quality of irradiance incident to a study surface. This measurement is calculated as a transmission of solar energy per time to a point of a grid which is placed over a study surface in kWh/m^2. The annual irradiance is computed using ClimateStudio, a plugin to Rhino CAD-environment (Solemma LLC, n.d.). This simulation allows to assess the potential of building energy efficient buildings equipped with solar panels. The output of this simulation is a grid of points with a defined density for a façade or a roof of a building. The example of the output result is depicted in figure 2.11.

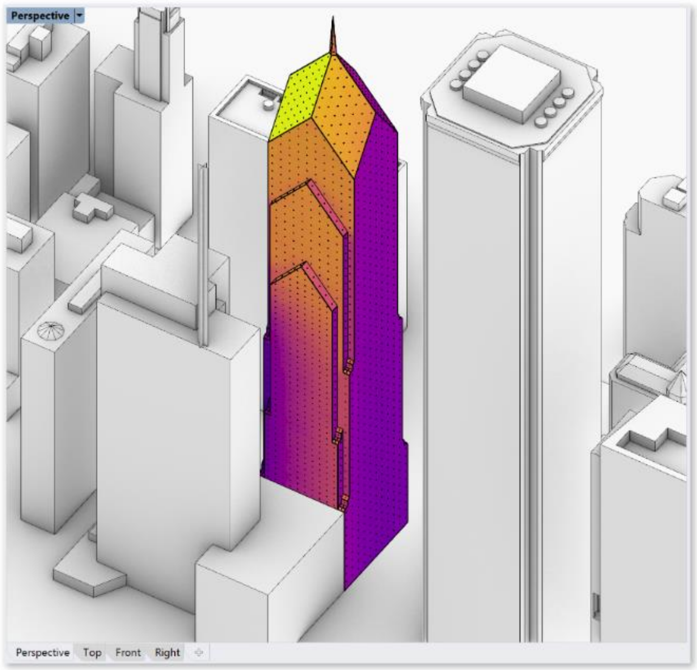


Figure 2.11. Total annual solar irradiance example in Rhino environment. Source: Solemma LLC, ClimateStudio, n.d.

The simulation output of the total annual solar irradiance is a multipoint grid in the format of ESRI shapefile for facades with values for each point in the attribute table.

3. Methods

This section describes the workflow of methods applied to add simulation outputs as appearances to CityGML geometries. Simulation outputs are presented by two different types of ESRI shapefiles: obstruction angle is a polygon shapefile with OA values for each window, while direct sun hours and total annual solar irradiance are multipoint grids for windows and facades respectively. However, DSH simulation output may be calculated not for the entire window, but just for the centroid of it. Therefore, the method used for the OA may be also applied to DSH.

All methods that are described below have similar parts in common: finding spatial relationships of simulation output with geometries, writing a correct ID of a target geometry, creating appearances and writing them into a CityGML model. Simulation outputs like OA and DSH, which have only one value for a given geometry (e.g., window), may be expressed as a color depending on interval of values. On the other hand, multipoint grids require to create rasters with coloring depending on values and after that applied as an appearance to geometries of walls or windows. The main tool to manipulate the data was FME Workbench.

3.1 Obstruction Angle and Direct Sun hours

This section details the workflow for integrating simulation outputs into a CityGML model, focusing on the OA (Obstruction Angle) simulation output. The goal is to process simulation outputs to generate visual appearances for windows, making it easier to understand and analyze the data. The workflow is divided into several parts, each addressing a specific aspect of the process, from data preparation and transformation to spatial relationship analysis and final output integration. By following this structured approach, the workflow ensures accurate and meaningful representation of simulation results within the 3D city model. An overview of the entire workflow is presented below (figure 3.1).

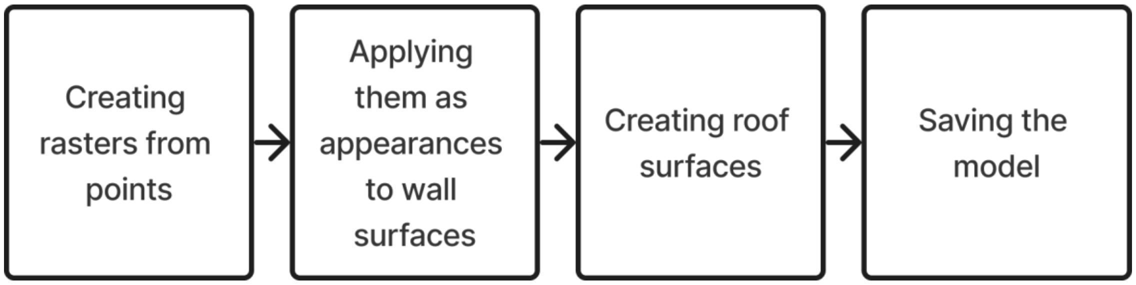

Creating rasters from points
Applying them as appearances to wall surfaces
Creating roof surfaces
Saving the model

Figure 3.1. Workflow of implementation OA as an appearance module

The workflow can be broken down into 4 parts:

Part 1. Processing simulation output (figure 3.2). In this part of the workflow, appearances of windows are created depending on values of OA.

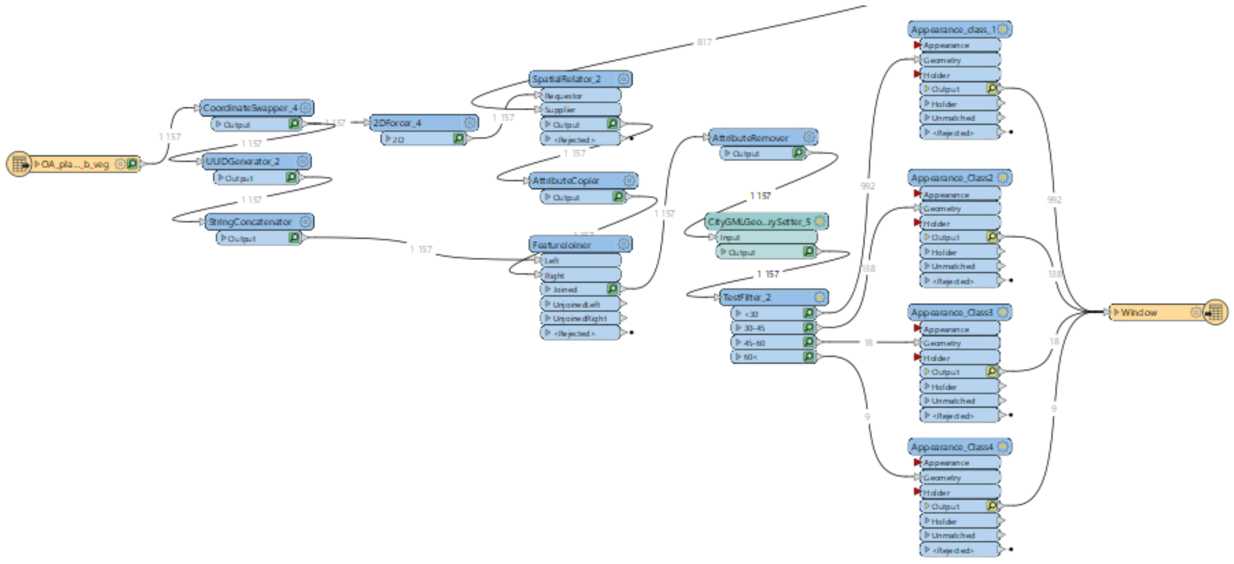


Figure 3.2. FME workflow of processing the simulation output

After reading the shapefile there is a block of transformers to uniform input: coordinate swapper is used to change X and Y coordinates, because of difference in the order of coordinates in FME features and ESRI shapefiles. *UUID Generator* and String Concatenator transformers are applied to create unique IDs to each output and create a *gml_id* by concatenating “GML_” and ID created in *UUID Generator*.

Then geometries of the simulation output are used to find a spatial relationship with geometries of walls. For this purpose, both geometries are forced to 2D by an FME transformer. 2D wall geometries were buffered by 1 meter. A *SpatialRelator* transformer finds all objects of 2D simulation output which are within the buffer zone of wall geometries and joins attributes, where *gml_ids* of walls are used as parent IDs to write appearances correctly into a CityGML model with corresponding geometries. This is performed by renaming the attribute *gml_id* derived from the wall surfaces to *gml_parent_id* by an

AttributeCopier transformer. The next step is to join these attributes with a 3D representation of the simulation outputs and remove attributes which are no longer used.

For correctly writing geometries into a CityGML model it is vital to create two attributes: *Citygml_lod_name* and *Citygml_feature_role* and with values *LOD4MultiSurface* and *opening.* This helps to indicate a level of detail and geometry type. In this case *opening* means that the geometry belongs to a window. To achieve this, a custom transformer *CityGMLGeometrySetter* was used. This custom transformer is available in FME Community Hub (FME Community Hub, n.d).

The next step was to break all OA values in intervals to apply different colors to each interval. For this purpose a *TestFilter* transformer was used to channel all obstruction angle values in different breakdowns which in this case was taken from "Dagsljus i Planering" by Alenius et al., (2015): less than 30°, 30°-45°, 45° -60° and more than 60°.

All outputs from test filter are channeled to own *AppearanceSetter* transformer where a chosen color is given to a geometry. The color palette for OA is presented in the table 1.

Table 1. Color palette of OA value intervals.

Color	OA value	RGB color code
🟨	< 30°	1, 1, 0.698039
🟨	30°- 45°	0.996078, 0.8, 0.360784
🟧	45° - 60°	0.992157, 0.552941, 0.235294
🟥	> 60°	0.890196, 0.101961, 0.109804

The color codes are important to save, because they allow to query specific windows with the color code. The last step was to write all features as windows to a CityGML writer.

Part 2. Join the output to a wall surface (figure 3.3). This part is responsible for finding spatial relationships between windows and wall geometries. It is important to find which windows belong to which walls to correctly write geometries into a model.

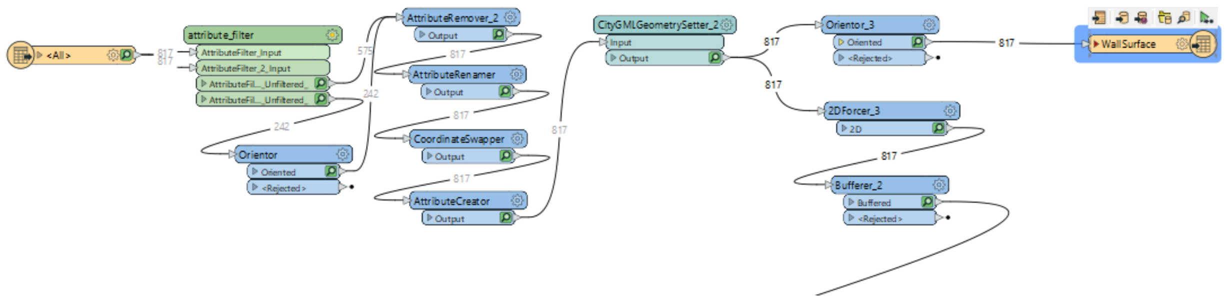


Figure 3.3. FME workflow of processing wall surfaces geometries

This step was partially described in part 1 of the workflow. Wall geometries were used to find spatial relationships with windows as an attribute supplier.

First, the custom transformer called *attribute_filter* was used to fix orientations of wall surfaces. Half of the wall surfaces were oriented incorrectly, which caused inaccurate representation of wall transparency. Walls and wall parts were manually picked and oriented backwards.

The following block of transformer is aimed to edit and delete unnecessary attributes. Coordinates were swapped and a transformer called *CityGMLGeometrySetter* sets wall surfaces to *LOD4MultiSurface* and *BoundedBy* feature role to write geometries properly to the model. It is important to mention that a transformer *Orientor* was used twice, because after importing the model to a 3DCityDB all orientations of objects are inverted so at the export the same model is in the correct orientation.

<u>Part 3. Creating roof surfaces (figure 3.4).</u> This part describes how roof surfaces were created and written into the CityGML model.

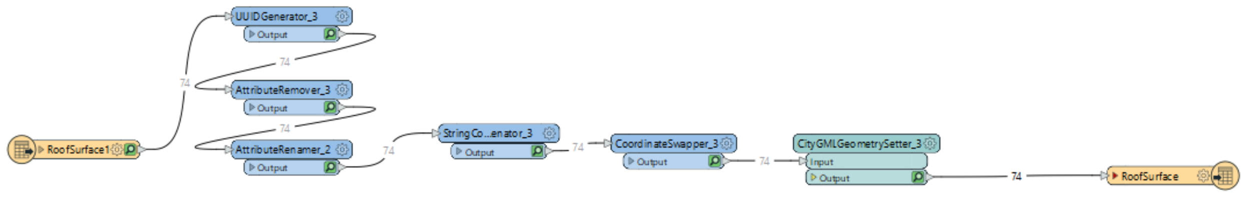


Figure 3.4. FME workflow of creating roof surfaces

Roof surfaces written from the initial CityGML model did not work with the rest of the building parts. After changing coordinates, coordinate reference systems, extracting to a shapefile and importing roofs were still invisible. However, these geometries were still visible and match spatially in a different software (ArcGIS Pro), but in FME the same geometries did not match. Therefore, it might be an issue specific to FME.

The solution to this issue was to create footprints of walls and extrude them for a Z-value of walls in ArcGIS Pro. The result was imported in FME to prepare the feature and write it to the final model. The transformers do the same as previously for windows: creating unique IDs via *UUID Generator* and *StringConcatenator*, swapping coordinates and setting geometry to *LOD4MultiSurface* and the *BoundedBy* feature role.

<u>Part 4. Writing the rest of geometries (figure 3.5).</u>

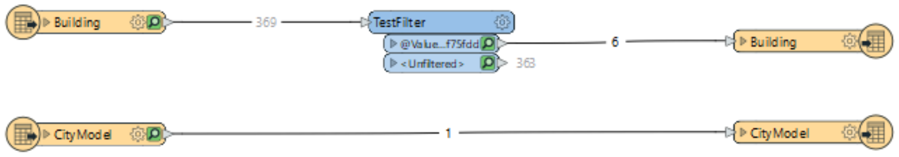


Figure 3.5. FME workflow of writing CityModel and Building to the model

In this part, the CityModel was directly written from the original model. Building objects were filtered by building *gml_ids* to extract all affected buildings with windows and obstruction angle values to write them to the building part of the output model.

Combining all parts a CityGML model was created. It was imported to a local 3DCityDB using the 3DCityDB Importer/Exporter tool.

3.2 Total Annual Solar Irradiance

This simulation output is represented as a grid of points with solar irradiance value for a façade of a building. To apply these grids to a building, the following workflow was created (figure 3.6).

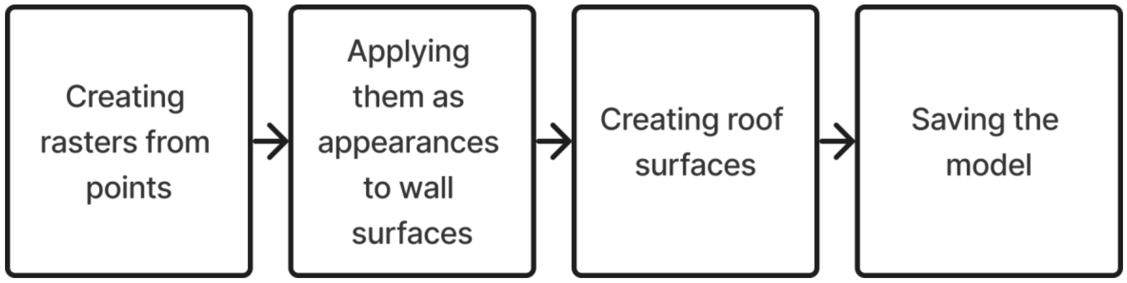

Creating rasters from points
Applying them as appearances to wall surfaces
Creating roof surfaces
Saving the model

Figure 3.6. Turning facade grids into an appearance

The workflow can be broken down into the following sections:

<u>Part 1. Rasterizing annual solar irradiance grids and writing wall surfaces (figure 3.7).</u> The first part is the most important in this workflow, as it is necessary to transform the total annual solar irradiance grids into images which may be applied to facades as appearances.

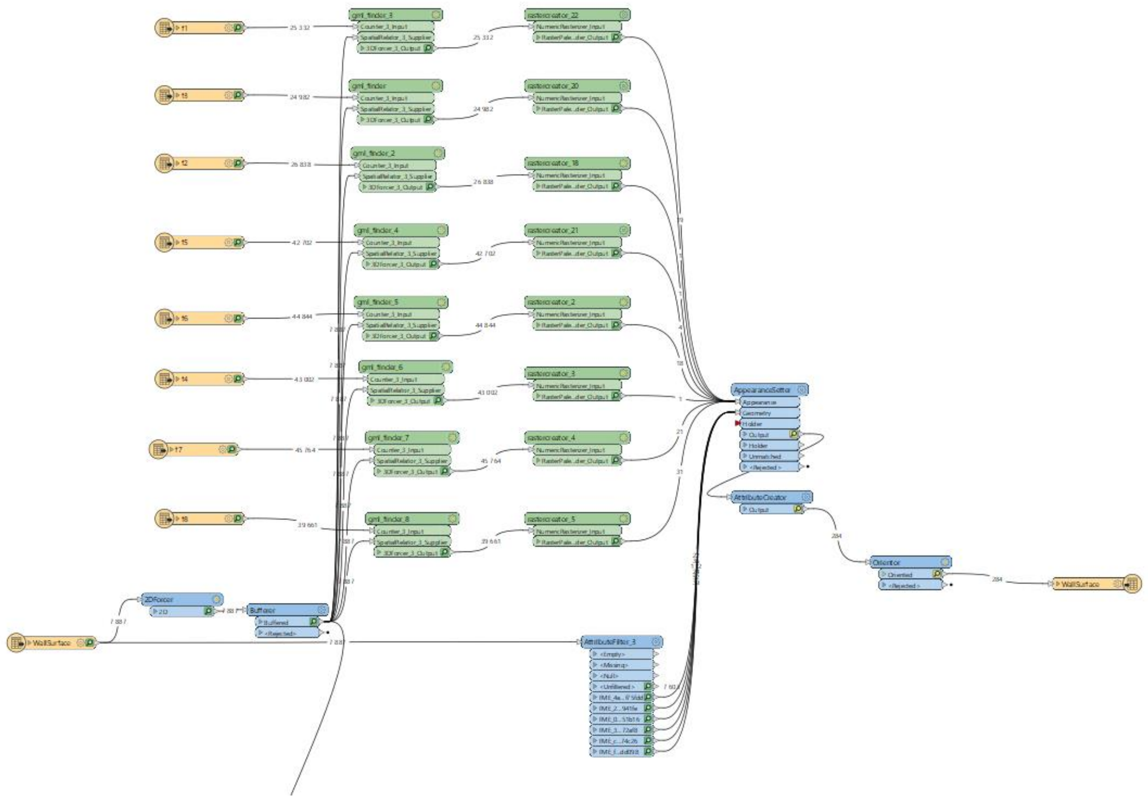


Figure 3.7. First section of a FME workflow related to wall surfaces

The first approach was to create raster images and apply them to wall geometries. It was unsuccessful because some wall geometries comprised several parts and were more complex. This method was working for simple walls consisting just of one aggregate without parts (figure 3.8a), but it applied the same raster for each section if a geometry consisted of several parts (figure 3.8b).

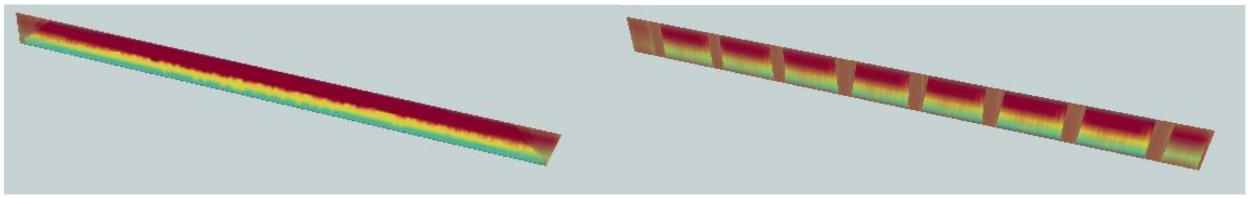
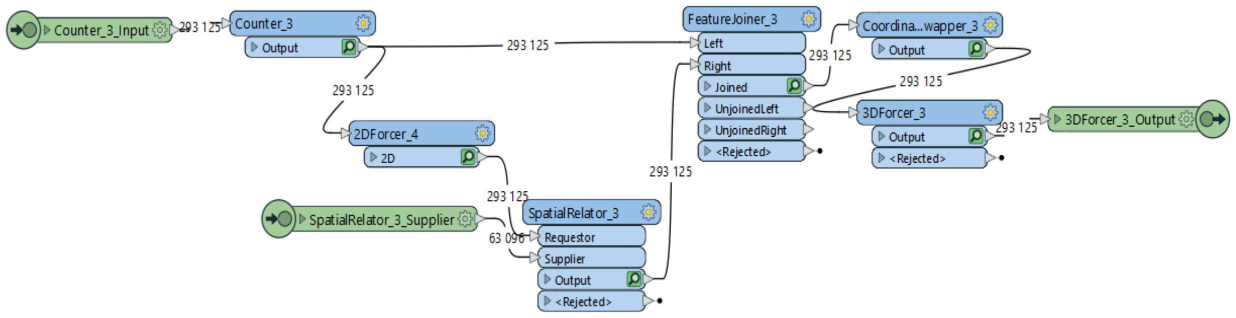


Figure 3.8. Applying one raster as an appearance to a) a simple and b) a complex wall

This way even if applied to a simple wall surface the appearance is still distorted (figure 3.8a). To resolve this issue custom transformers *gml_finder* and *rastercreator* were made (figure 3.9a and 3.9b).



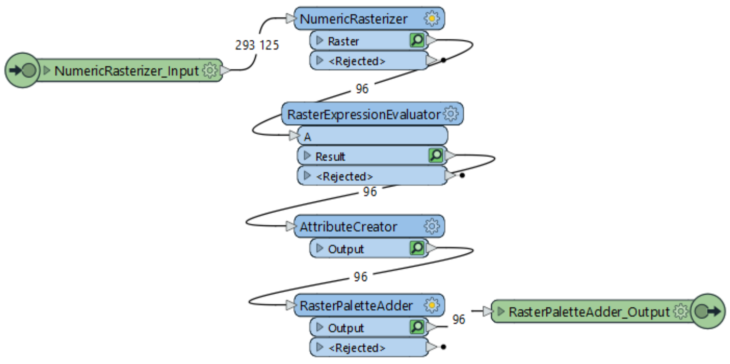


Figure 3.9. Custom transformers: a) gml_finder (top) and b) rastercreator (bottom).

The transformer *gml_finder* (figure 3.9a) is based on the previous workflow, when points and wall surfaces are forced to 2D, walls are buffered to 0.1 meter and spatial join was executed, which helps to find all points within a buffer zone. The *Bufferer* transformer creates a buffer zone around the 2D walls, in other words from lines polygons were created. The minimal buffer zone was used because there were some intersections with other walls, as points have no size, this was resolved by the less buffer zone around walls. Joining features allowed to assign *gml_id* of walls for each point of the simulation output grid without errors.

The subsection of this custom transformer is swapping coordinates from Y to Z so the grid will be projected to the ground to create a raster for a wall, because without editing coordinates raster will be one pixel wide just in X and Y coordinates. A *3DForcer* transformer assigns solar irradiance values as a Z coordinate to create a grid of Z values which is an input to the next transformer.

The *rastercreator* transformer (figure 3.9b) rasterizes grids received and gives it a color based on solar irradiance value. The first transformer is *NumericRasterizer* which creates raster grouping by *gml_ids* of wall parts received by a previous transformer. This means that for walls where only one *gml_id* is given only one raster is produced and in the opposite if there are many *gml_ids* provided several rasters will be created and grouped according to this value.

RasterExpressionEvaluator classifies rasters into 9 classes, *AttributeCreator* is to create a value of palette color to each class, *RasterPaletteAdder* gives cells of raster a color depending on the class. The intervals and color palette is presented in table 2.

Table 2. Color palette for annual solar irradiance intervals.

Color	Annual solar irradiance value
🟨	<100 kWh/m^2
🟨	100-200 kWh/m^2
🟩	200-300 kWh/m^2
🟩	300-400 kWh/m^2
🟩	400-500 kWh/m^2
🟩	500-600 kWh/m^2
🟩	600-700 kWh/m^2
🟩	700-800 kWh/m^2
🟩	>800 kWh/m^2

<u>Part 2. Adding OA simulation output as an appearance to windows and writing the rest parts of the model.</u>

This part of the workflow was taken from the previous (figure 3.2) as it was decided to combine both simulation outputs in one model where annual solar irradiance as an appearance was applied to walls and obstruction angle to windows. In the resulting model wall geometries covered window geometries, therefore windows were not visible. The solution was to create a custom transformer *offset_fixer* (figure 3.10).

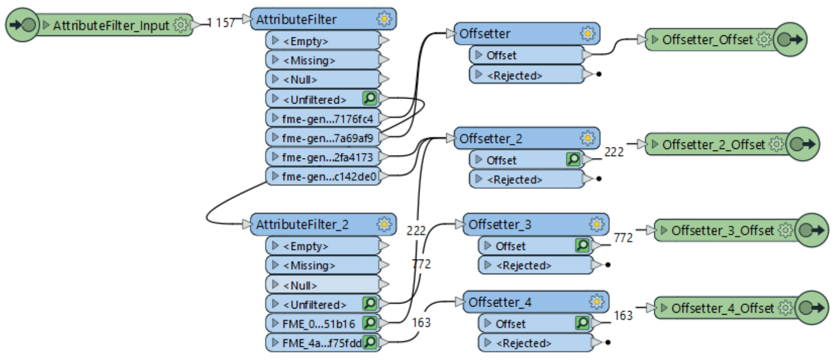


Figure 3.10. Custom transformer Offset_fixer

The main purpose to filter out windows by IDs of walls and buildings which need to be moved in 4 directions: positive and negative in X axis and Y axis for 0.1 degrees of cartesian coordinates in order not to detach windows from walls.

<u>Part 3. Creating roof surfaces and writing the rest of geometries.</u> This part is the same as in the previous workflow. The resulting CityGML model was imported to a local 3DCityDB using the 3DCityDB Importer/Exporter tool.

3.3 Direct Sun Hours

The same workflow (figure 3.6) was executed to create rasters from multipoint grids of DSH output and apply them as an appearance to windows.

The only difference of the workflow was that a spatial join with *2DForcer* and *Bufferer* did not work, because several windows could be joined spatially in 2D. Instead of these transformers a *Near 3D* tool in ArcGIS Pro was used to find IDs of windows and join it to each point of DSH grid.

The workflow worked and created rasters were applied as an appearance of windows, but while writing resulting geometries into a CityGML model an error of texture coordinates was encountered. To resolve it, texture coordinates for geometries were generated, first, automatically, and then manually, but the issue persisted. Afterwards, the created rasters were clipped by the window geometries, raster cell sizes were changed, which did not solve the issue. At the end, geometries from a CityGML model and not from a shapefile were used to avoid possible issues in coordinates. These attempts to resolve the issue did not help.

4. Results

4.1 Created CityGML models

The resulting CityGML model with added appearances of walls and windows described in the previous section was saved locally. Afterwards, the model was imported into a local 3DCityDB and exported from it using 3DCityDB Importer/Exporter tool. Before importing and after exporting the model FME Data Inspector was used to check if the model was written correctly (figures 4.1 and 4.2).

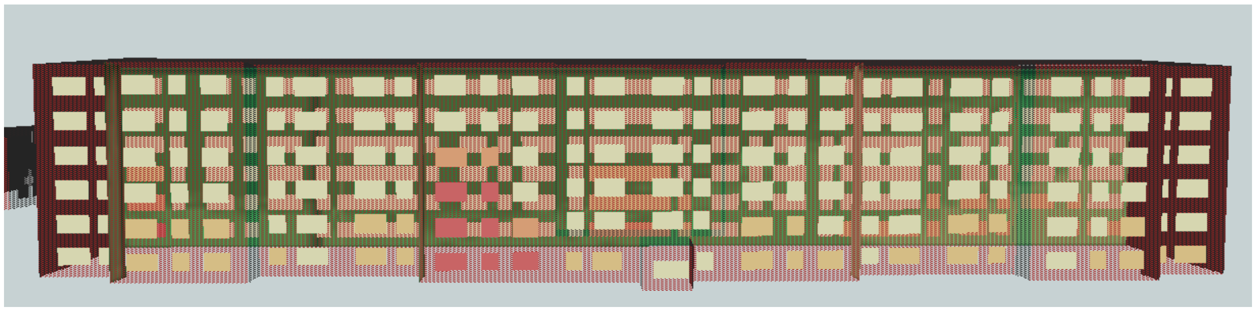


Figure 4.1. Created CityGML model with OA as appearance of windows and total annual solar irradiance as appearance of walls before importing into 3DCityDB.

In figure 4.1 the appearance is inverted because of before mentioned behavior of 3DCityDB Importer/Exporter when exporting the model turns back the appearance to the correct state (figure 4.2).

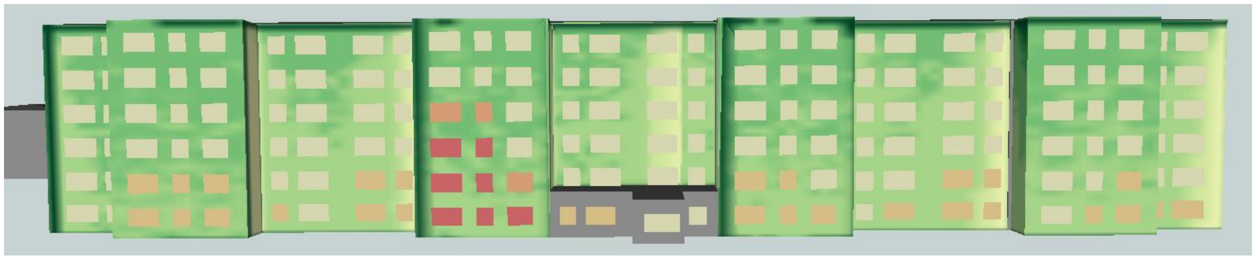


Figure 4.2 The same CityGML model exported from 3DCityDB.

The model was validated without errors by the 3DCityDB Importer/Exporter tool. In figure 4.2 the appearances of windows present OA values. Wall geometries have as appearance the total annual solar irradiance value. The color palette for both appearances was described in the part 1 of both workflows (sections 3.1 and 3.2 respectively).

The resulting CityGML models were tested in FZK Viewer (FZK Viewer, KIT, n.d.) but geometries with appearances did not show properly. In the process of the thesis appearances were applied as on a global level and not directly to geometries, which may cause these visualization issues (figure 4.3).

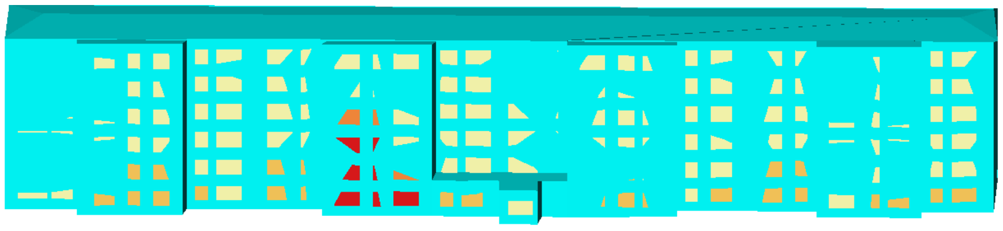


Figure 4.3. Visualization issues in FZK Viewer

4.2. Querying appearances from a 3DCityDB

One of aims of this thesis was to test possibilities of retrieving a CityGML model with simulation outputs as appearances from a 3DCityDB. Exporting it with the Importer/Exporter tool of 3DCityDB is the easiest way to receive a result, but SQL queries allow to retrieve specific appearances from a database.

Considering 3DCityDB schema (figure 2.6) four tables are necessary to join (figure 4.4):

1) *citydb.surface_geometry* contains geometries of all objects in a model;
2) *citydb.opening* with IDs of objects only for windows to filter incoming geometries;
3) *citydb.surface_data* with parameters of appearances (e.g., a color of a geometry);
4) *citydb.textureparam* which is a bridge between *citydb.surface_data* and *citydb.surface_geometry* as it contains IDs of both tables;

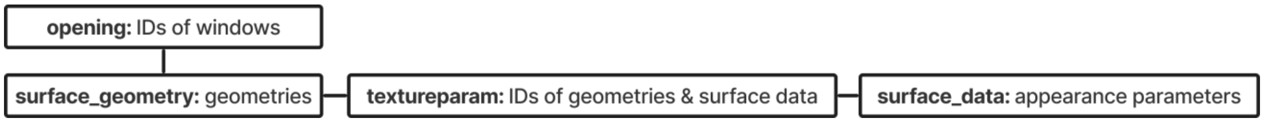

opening: IDs of windows
surface_geometry: geometries
textureparam: IDs of geometries & surface data
surface_data: appearance parameters

Figure 4.4. Tables to query to receive window appearances from 3DCityDB

To receive window geometries with the appearances, the following SQL queries were created and executed using a *SQLCreator* transformer in an FME environment:

1) A query to extract all geometries of a model

```
SELECT sg.id, sg.cityobject_id, sg.geometry
FROM citydb.surface_geometry sg
WHERE sg.geometry IS NOT NULL
```

2) A query to extract all *BuildingWindow* IDs. This attribute is used to filter out only window geometries from the previous table. It is worth noting that the opening table also contains doors if there are any in the model. A condition with a specific *objectclass_id*, 38 for *BuildingWindow* (3DCityDB Documentation, n.d.), should be applied in that case.

```
SELECT * FROM citydb.opening
```

3) A query to receive appearance parameters, color, and shininess in this case.

```
SELECT sd.id AS sd_id, sd.x3d_diffuse_color, sd.x3d_shininess, sg.cityobject_i
FROM citydb.surface_data sd
INNER JOIN citydb.textureparam tp ON tp.surface_data_id = sd.id
INNER JOIN citydb.surface_geometry sg ON sg.id = tp.surface_geometry_id
WHERE sd.tex_image_id IS NULL
```

After extracting all features from the database, the following FME workflow was created (figure 4.5).

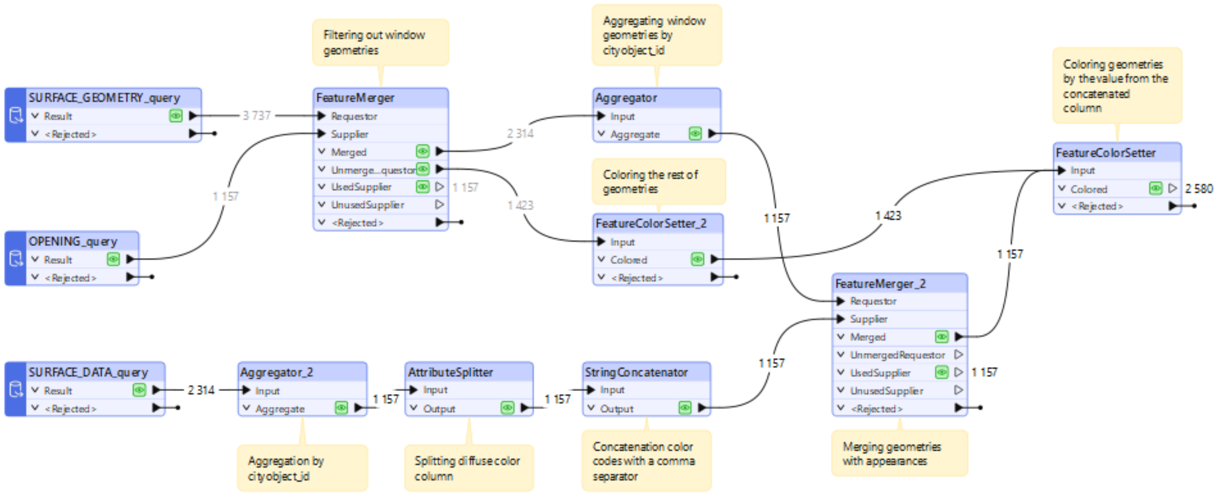


Figure 4.5. FME workflow to extract geometries with appearances from a 3DCityDB

The result of the workflow is a 3D model with appearances received from a 3DCityDB (figure 4.6).

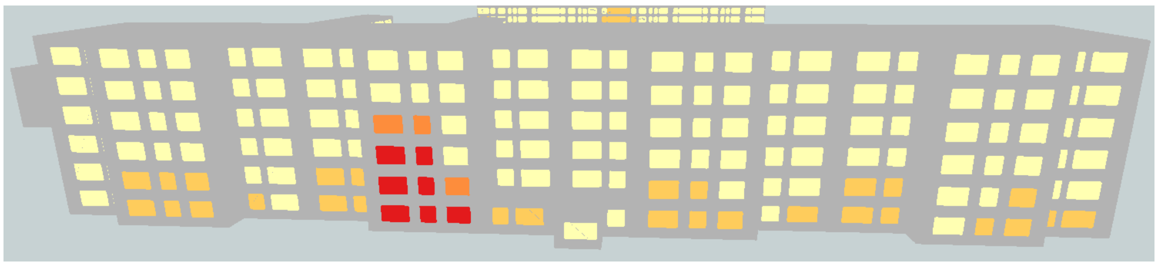


Figure 4.6. The result of the created workflow

The proposed queries can be modified to extract appearances of windows within specified range of OA by adding a condition WHERE with a known color code to the *SURFACE_DATA* query (figure 4.5). For example, a query to extract all OA less than 30 degrees was created:

```
SELECT sd.id AS sd_id, sd.x3d_diffuse_color AS color, sd.x3d_shininess, sg.cityobject_id
FROM citydb.surface_data sd
LEFT JOIN citydb.textureparam tp ON tp.surface_data_id = sd.id
LEFT JOIN citydb.surface_geometry sg ON sg.id = tp.surface_geometry_id
WHERE sd.x3d_diffuse_color = '1.0 1.0 0.698039'
```

The result of the modified query is presented in figure 4.7.

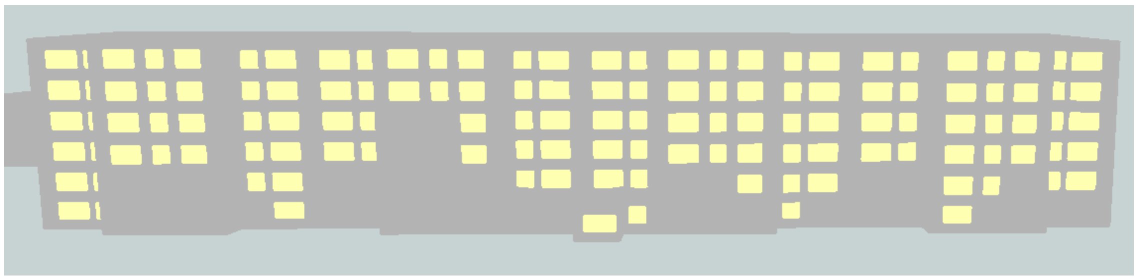


Figure 4.7. Result of the modified query to show specific values of OA

The workflow may be modified to add geometries of the rest of windows, but their color code should be additionally written as an attribute. If it is not done, color will be random.

The extraction of total annual solar irradiance as appearances for walls from a 3DCityDB with SQL queries was also performed. In this case only one query was created with condition *WHERE* to filter all geometries without raster images.

```
SELECT sd.id, sd.tex_image_id, sg.id AS sg_id, sg.geometry, ti.tex_image_uri, ti.tex_image_data
FROM citydb.tex_image AS ti
INNER JOIN citydb.surface_data sd ON sd.tex_image_id = ti.id
INNER JOIN citydb.textureparam tp ON tp.surface_data_id = sd.id
INNER JOIN citydb.surface_geometry sg ON sg.id = tp.surface_geometry_id
WHERE ti.tex_image_uri IS NOT NULL
```

The main purpose of the query is to retrieve joined geometries with raster images which belong to each geometry. To execute this a connection of two tables *citydb.surface_geometry* and *citydb.surface_data can be established via citydb.textureparam* which contains IDs of these tables. *Citydb.surface_data* contains keys to *citydb.tex_image* which stores images (figure 4.8).

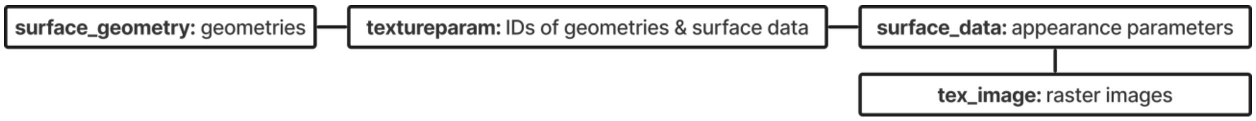

surface_geometry: geometries
textureparam: IDs of geometries & surface data
surface_data: appearance parameters
tex_image: raster images

Figure 4.8. Required tables from a 3DCityDB to retrieve wall appearances

To process retrieved data from the database a FME workflow was constructed (figure 4.9).

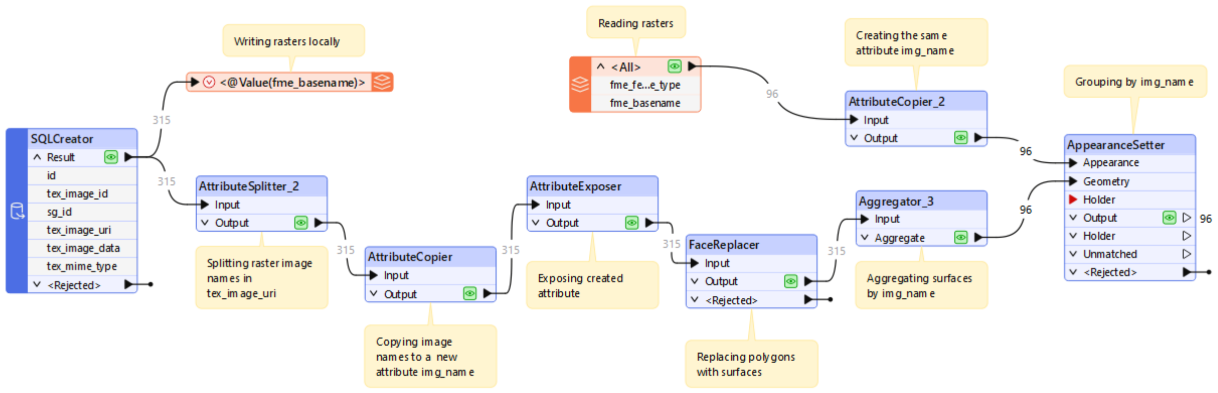


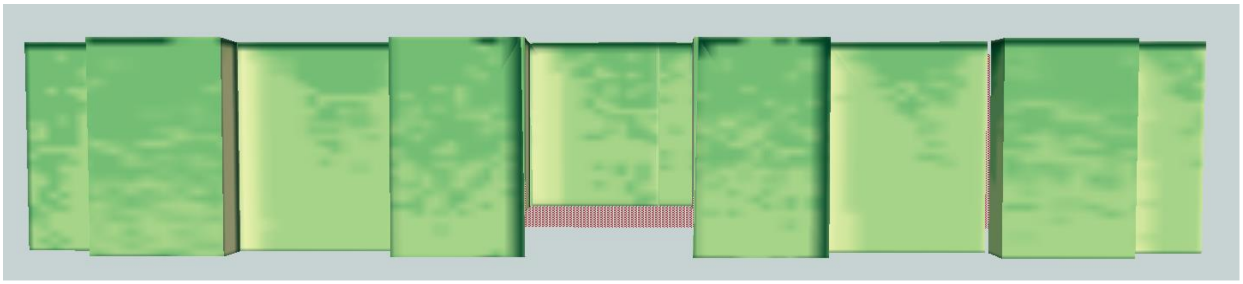
Figure 4.9. FME workflow to visualize exported geometries and appearances

The main difference of this workflow is that previously created rasters had to be saved locally before being applied to geometries. Encoded rasters directly retrieved from the 3DCityDB cannot be used as an appearance. *AppearanceSetter* transformer requires *fme_raster* feature type, and without writing rasters locally, they have *fme_sql_creator* feature type. After reading images again, feature type changes to *fme_raster*.

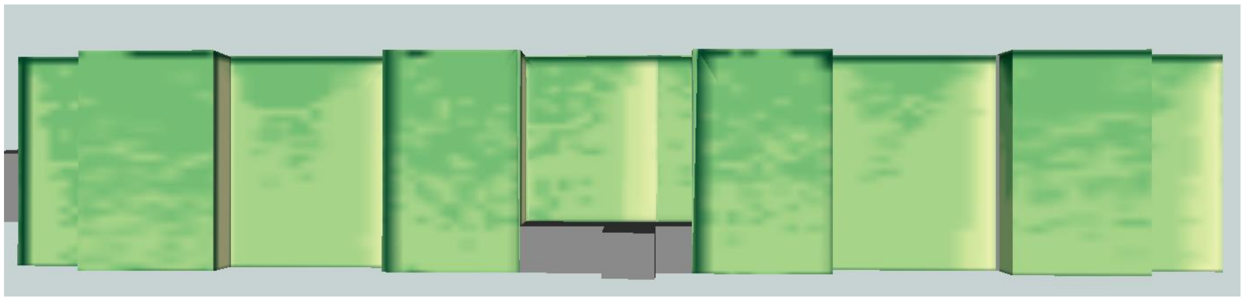
Geometries retrieved from the database have feature type fme_polygon, which is also unsuitable for the *AppearanceSetter* transformer. To resolve the issue a transformer *FaceReplacer* was added, which converts the feature type to *fme_surfaces*, what was required.

The only value which helps to join a raster to its related/corresponding geometry is the image name stored within the model. Therefore, an attribute storing raster names for geometries and rasters is necessary to perform feature merge between two datasets. It allows to group wall geometries by image names and apply raster data as appearances to them.

After querying raster files from a 3DCityDB it was noticed that appearances retrieved by SQL queries are mirrored (figures 4.10a and 4.10b).



a)



b)

Figure 4.10 a) Appearances retrieved from the 3DCityDB by SQL Queries, b) Appearances exported from 3DCityDB with the 3DCityDB Importer/Exporter tool

This issue was resolved by changing texture wrap pattern in *AppearanceSetter* to *Mirror.*

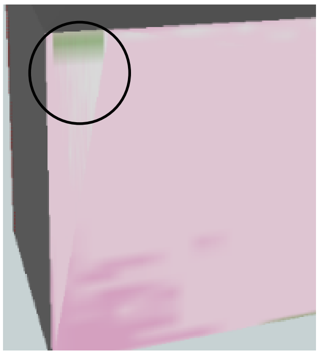
5. Discussion

This master thesis builds upon the limited previous works that have utilized appearances in 3D models for visualization purposes by coloring geometries based on semantic data (Law et al. 2006, Kolbe et al. 2015). Previous applications were limited to visualization purposes, but proposed methods now enable efficient querying and manipulation of 3D city models' appearances directly from 3DCityDB. This capability not only enhances the visual representation of simulation data but also facilitates more intuitive and accessible urban analysis. For urban planners, the ability to dynamically visualize simulation results within 3D city models, can lead to more informed decision-making and streamlined project workflows. Consultants can easily share these visualizations, making collaboration and communication more effective.

The proposed methodology offers significant benefits by allowing simulation outputs to be immediately visible upon opening a 3D city model in an FME environment, eliminating the need for additional generic attributes or Application Domain Extensions (ADEs). This approach addresses an interoperability issue, enabling different users to visualize simulation outputs within the same software environment, thereby improving consistency and reducing potential misinterpretations (Biljecki et al. 2015, 2018, Bilen et al., 2014).

It is evident that the ability to visualize simulation data dynamically may significantly impact urban planning and development (Kanters et al., 2021). Earlier studies highlighted limitations such as the static nature of visualizations and the complexity of data manipulation within 3D city models. This thesis addresses these challenges by providing a dynamic and user-friendly approach to visualizing simulation results. However, the following issues were discovered while exploring the results:

1) Geometries impact the appearance raster distortions (figure 5.1)



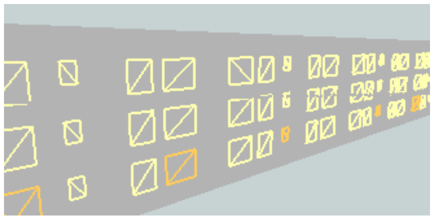
Even after creating an individual raster for each wall section which was shown in figures 3.8a and 3.8b, there are still some inaccuracies which are caused by vertices of geometries. In the figure 5.1 these distortions are visible within a circle. At the same time, a change in the pattern of the appearance in the figure proves that a new raster for each part of a complex wall geometry is created, resolving the issue in figures 3.8a and 3.8b. Therefore, in cases where the accuracy of annual solar irradiance on edges is of great importance this method may impact.

Figure 5.1. Appearance distortions

2) Too many classes without representativity.

Great difference in values of the whole study area increases category classes for rasters and at the same time the low difference of measurements in one façade does not allow to show this difference. When it is necessary to see differences in values in a single façade less than intervals of a neighborhood (e.g., in the created model intervals of solar irradiance was 100 kWh/m^2, but differences less than 10 are studied), it is better to create a raster specifically for that façade. Instead of analyzing the entire model's simulation output breakdowns, this approach allows for more detailed observation. This issue may be resolved by applying different color palettes to different breakdowns of values, for example, yellow-red palette for irradiance values between 0 and 100, and white-green palette for values between 100 and 200.

3) Appearance orientation issues when querying appearances from a 3DCityDB (figure 5.2)



After extracting color codes from the *citydb.surface_data* and applying it to geometries, it was discovered that a part of windows was exported with incorrect orientation. This issue can be resolved with an Orientor transformer.

Figure 5.2. Appearance orientation issues

4) Color of windows retrieved by querying does not look identical to colors in the original CityGML model, however, the color codes are retrieved directly from the 3DCityDB. It is related to additional parameters

of the transformer *AppearanceSetter* as shininess, which is not available in the transformer *FeatureColorSetter* in the querying workflow.

5) Previously mentioned issue of visualizing appearances between different CityGML model viewers (figure 4.3). It may be solved by storing appearances for specific parts of the model and not for all geometries. This may be explored in future works.

6) Creating raster files from simulation output grids, as it was done for total annual solar irradiance, raises a question on how to query simulation output values directly from a database. Therefore, there is no flexibility to select specific geometries by simulation output value. However, it is possible to extract certain geometries which have appearances. For example, there is no opportunity to query an interval of solar irradiance, because created rasters do not contain simulation output values Values are categorized into groups and colorized. Adding generic attributes may be helpful, if there is a way to associate simulation outputs with appearances or geometries. In this case, it will allow to retrieve data with a defined value or interval.

7) Using the appearance modules increases importance of metadata in datasets if this methodology is used. A CityGML model without simulation value intervals and color codes for these intervals described in metadata loses utility. Without knowing this information from metadata users cannot interpret what appearances mean. It will be only a visualization outside of context. Metadata connects a specific color with a specific interval of simulation output values.

Building on the findings of this thesis, several avenues for further development are possible to enhance the integration and utility of simulation outputs within CityGML models.

Firstly, expanding the range of surface types with the appearance module could be explored. For example, the grid of points may be used as an appearance for window geometries. In the process of the thesis, it was tested but technical issues were encountered.

Secondly, storing simulation outputs as generic attributes may be possible. The main limitation of this proposed methodology is that values of simulation outputs are not actually stored in appearances. Raster cells represent intervals. It is impossible to retrieve a value for a specific cell of a raster. Generic attributes can be saved within a model, but still cannot be validated against the schema. This may cause additional challenges, because the proposed method attempting to store appearances without extensions of CityGML. Generic attributes may complement with detailed simulation output values and appearances may be used to assess the context in general.

Thirdly, a combination of appearances may be written into one CityGML model. Therefore, different simulation outputs with the same data format (e.g., point grid for walls but in different seasons) may be applied to the same geometries under individual names. For example, it is possible to store several datasets of one simulation output (obstruction angle with planned buildings, without planned buildings or with vegetation) as an appearance of geometries. However, this method requires a way to distinct which appearance represents which simulation output.

Furthermore, the appearance module can use not only plain colors but also colors with additional effects which may be used in various graphical software, for example, an emissive color which is a color emitted by a surface while reflecting a light. It may add possibilities to visualize simulation outputs.

Moreover, it should be possible to create rasters for defined intervals of simulation output values and leave other cells with NODATA value. This way several rasters will be grouped by one geometry and overlayed as an appearance over each other. This modification can make possible querying specific appearances, because only known values will be presented visually.

By pursuing these developments, the potential of CityGML models as a powerful tool for urban planning and simulation output integration can be further extended, offering benefits to urban planners and the contribute to the creation of better conditions for sustainable urban development.

6. Conclusion

In conclusion, this study successfully demonstrates the feasibility of storing sunlight and daylight simulation outputs within the appearance module of a CityGML model in 3DCityDB. The research confirms that the CityGML appearance module can store datasets generated by daylight and sunlight simulations with modifications, enabling that these datasets can be imported into and retrieved from a 3DCityDB database. This may be useful to decision makers (urban planners) when they check if planned buildings do conform to the Swedish daylight regulations.

Furthermore, the thesis outlines a clear set of steps required to integrate simulation outputs into the appearance module, providing a practical guide for future implementations. This integration process has been validated, ensuring that the data provided by simulations can be stored in semantic 3D city models utilizing 3DCityDB modules in the current CityGML schema.

Finally, the thesis work shows that querying and retrieving appearance data from 3DCityDB can be done efficiently, facilitating detailed analysis and visualization. This capability enhances the utility of CityGML models, allowing urban planners to examine if buildings adhere to the daylight or sunlight recommendations. The proposed methods can lead to better strategic planning, implementation of policies and better utilization of funds.

7. References

3DCityDB documentation, n.d., https://3dcitydb-docs.readthedocs.io/en/version-2023.0/.

3DCityDB Importer/Exporter Github website, n.d., https://github.com/3dcitydb/importer-exporter

Alenius, M., Dahlberg, J., Lundgren, M., & Cederström, C. (2019). Dagsljus i Stadsplanering. https://whitearkitekter.com/se/wpcontent/uploads/sites/3/2019/04/20190408_WRL_Dagsljus-i-Stadsplanering-1.pdf

ArcGIS Pro Documentation, n.d., https://pro.arcgis.com/en/pro-app/latest/tool-reference/3d-analyst/sun-shadow-volume.htm

Aries M, Aarts M, van Hoof J. Daylight and health: A review of the evidence and consequences for the built environment. Lighting Research & Technology. 2015;47(1):6-27. doi:10.1177/1477153513509258

Barnes, M., and E. L. Finch. (2008). COLLADA – Digital Asset Schema Release 1.5.0. Specification.

Bhatia, S., P. Cozzi, A. Knyazev, and T. Parisi. (2017). glTF Specification 2.0. .

Billen, Roland & Cutting-Decelle, Af & Marina, Ognen & Almeida, J.-P & Caglioni, Matteo & Falquet, Gilles & Leduc, Thomas & Métral, Claudine & Moreau, Guillaume & Perret, Julien & Rabino, Giovanni & García, Roberto & Yatskiv, Irina & Zlatanova, Sisi. (2014). 3D City Models and urban information: Current issues and perspectives. 10.1051/TU0801/201400001.

Biljecki, F., Kumar, K. & Nagel, C. (2018). CityGML Application Domain Extension (ADE): overview of developments. Open geospatial data, softw. stand. 3, 13, https://doi.org/10.1186/s40965-018-0055-6.

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

Borisuit A, Linhart F, Scartezzini J-L, Münch M. Effects of realistic office daylighting and electric lighting conditions on visual comfort, alertness and mood. Lighting Research & Technology. 2015;47(2):192-209. doi:10.1177/1477153514531518

Bournas, I. (2021). Swedish daylight regulation throughout the 20th century and considerations regarding current assessment methods for residential spaces. Building and Environment, 191, 107594. https://doi.org/10.1016/j.buildenv.2021.107594

Boverket. (2011). Boverket´s mandatory provisions and general recommendations, BBR: BFS 2011:6 with amendments up to BFS 2018:4. https://www.boverket.se (BFS 2011:6).

Burggraf, D. (2015). OGC KML 2.3, Version 1.0.

CityJSON official website, n.d., https://www.cityjson.org/citygml/v20/#citygml-features-supported

CityJSON Open Geospatial Consortium website, n.d., https://docs.ogc.org/cs/20-072r5/20-072r5.html

Eriksson, S., Waldenström, L., Tillberg, M., Österbring, M., & Kalagasidis, A. S. (2019). Numerical simulations and empirical data for the evaluation of daylight factors in existing buildings in Sweden. Energies, 12(11), 2200. https://doi.org/10.3390/en12112200

Foley, J., van Dam, A,. Feiner, S., Hughes, J. (1995): Computer Graphics: Principles and Practice. Addison Wesley, 2nd Ed

FME Community Hub, CityGMLGeometrySetter custom transformer, n.d.,https://hub.safe.com/publishers/safe-lab/transformers/citygmlgeometrysetter

FZK Viewer, Karlsruhe Institute of Technology, n.d., https://www.iai.kit.edu/english/1648.php

Gröger, G., & Plümer, L. (2012). CityGML–Interoperable semantic 3D city models. ISPRS Journal of Photogrammetry and Remote Sensing, 71, 12-33.

Harrie, L., Kanters, J., Mattisson, K., Nezval, P., Olsson, P. O., Pantazatou, K., . . . Fan, H. (2021). 3D CITY MODELS FOR SUPPORTING SIMULATIONS IN CITY DENSIFICATIONS. Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLVI-4/W4-2021, 73-77. doi:10.5194/isprsarchives-XLVI-4-W4-2021-73-2021

Jakica, N. (2017). State-of-the-art review of solar design tools and methods for assessing daylighting and solar potential for building-integrated photovoltaics. Renewable and Sustainable Energy Reviews, 81. doi:10.1016/j.rser.2017.05.080

Kaden, R. and Kolbe, T. H.: CITY-WIDE TOTAL ENERGY DEMAND ESTIMATION OF BUILDINGS USING SEMANTIC 3D CITY MODELS AND STATISTICAL DATA, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., II-2/W1, 163–171, https://doi.org/10.5194/isprsannals-II-2-W1-163-2013, 2013.

Kanters, J., Gentile, N., & Bernardo, R. (2021). Planning for solar access in Sweden: routines, metrics, and tools. Urban, Planning and Transport Research, 9(1), 347-367. doi:10.1080/21650020.2021.1944293

Kanters Jouri & Wall Maria (2014), The impact of urban design decisions on net zero energy solar buildings in Sweden, Urban, Planning and Transport Research, 2:1, 312-332, DOI: 10.1080/21650020.2014.939297.

Kolbe, Thomas & Gröger, Gerhard & Plümer, Lutz. (2005). CityGML - Interoperable access to 3D city models. Geo-information for Disaster Management. 10.1007/3-540-27468-5_6.

Kolbe, Thomas & Nagel, C. & Stadler, A.. (2009). CityGML-OGC standard for photogrammetry?. Photogrammetric Week. 265-277.

Kolbe, T. H., H. Nguyen Son, K. Chaturvedi, B. Willenborg, A. Donaubauer, C. Nagel, Z. Yao, H. Schulz, et al. (2019). 3D City Database for CityGML Version 4.2

Kumar, K., Ledoux, H., Commandeur, T. J. F., and Stoter, J. E.: MODELLING URBAN NOISE IN CITYGML ADE: CASE OF THE NETHERLANDS, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W5, 73–81, https://doi.org/10.5194/isprs-annals-IV-4-W5-73-2017, 2017.

Kutzner, T., Chaturvedi, K. & Kolbe, T.H., (2020). CityGML 3.0: New Functions Open Up New Applications. PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 88, 43–61. https://doi.org/10.1007/s41064-020-00095-z.

Ladybug, n.d., https://docs.ladybug.tools/ladybug-primer/

Law, C.W.; Lee, C.K.; Tai, M.K. Visualization of complex noise environment by virtual reality technologies. In Proceedings of the Symposium of the campaign "Science in the Public Service", Hong Kong, China, 27 April 2006; p. 8

Ledoux, H., Arroyo Ohori, K., Kumar, K. et al. CityJSON: a compact and easy-to-use encoding of the CityGML data model. Open geospatial data, softw. stand. 4, 4 (2019). https://doi.org/10.1186/s40965-019-0064-0

Littlefair, P. (2001). Daylight, sunlight, and solar gain in the urban environment. Solar Energy, 70(3), 177-185. doi:https://doi.org/10.1016/S0038-092X(00)00099-2

Littlefair, P. (2011). Site layout planning for daylight and sunlight. https://images.reading.gov.uk/2021/07/BRE-Oct-2011-Site-layout-planning-for-Daylight-andsunlight.pdf

Lou, Ruding. (2011). Modification of semantically enriched FE mesh models.

Mardaljevic, J., Heschong, L., & Lee, E. (2009). Daylight metrics and energy savings. Lighting Research & Technology - LIGHTING RES TECHNOL, 41, 261-283. doi:10.1177/1477153509339703

Noardo, F., Ohori, K. A., Biljecki, F., Ellul, C., Harrie, L., Krijnen, T., … & Stoter, J. (2020). Reference study of citygml software support: the geobim benchmark 2019—part ii. Transactions in GIS, 25(2), 842-868. https://doi.org/10.1111/tgis.12710

Open Geospatial Consortium, City Geography Markup Language, n.d.. https://www.ogc.org/standard/citygml/.

Open Geospatial Consortium, OGC adopts CityGML 3.0 Part 2: GML Encoding as an official OGC Standard, (2023), https://www.ogc.org/press-release/ogc-adopts-citygml-3-0-part-2-gml-encoding-as-an-official-ogc-standard/

Pantazatou, K., Kanters, J., Olsson, PO. et al. Input data requirements for daylight simulations in urban densifications. Urban Info 2, 2 (2023). https://doi.org/10.1007/s44212-023-00024-6

Pantazatou K., (2023), Direct Sun Hours. a script for the Grasshoper plugin. https://github.com/Sustainable3DCities/Direct_Sun_Hours?tab=readme-ov-file

Pantazatou, K., Kanters, J., Mattisson, K., Olsson, PO., Harrie, L. (2024). Recommendation for Vegetation Information in Semantic 3D City Models Used in Urban Planning Applications. In: Kolbe, T.H., Donaubauer, A., Beil, C. (eds) Recent Advances in 3D Geoinformation Science. 3DGeoInfo 2023. Lecture Notes in Geoinformation and Cartography. Springer, Cham. https://doi.org/10.1007/978-3-031-43699-4_1

Rhino 3D, n.d., https://www.rhino3d.com/

Solemma LLC, ClimateStudio, n.d., https://climatestudiodocs.com/index.html

Turan, I., Chegut, A., Fink, D., Reinhart, C., (2020): The value of daylight in office spaces. Building and Environment, 168, 106503.

Uggla M, Olsson P, Abdi B, Axelsson B, Calvert M, Christensen U, Gardevärn D, Hirsch G, Jeansson E, Kadric Z, et al. Future Swedish 3D City Models—Specifications, Test Data, and Evaluation. ISPRS International Journal of Geo-Information. (2023); 12(2):47. https://doi.org/10.3390/ijgi12020047.

Urban Multi-scale Environmental Predictor Documentation, n.d., https://umep-docs.readthedocs.io/projects/tutorial/en/latest/Tutorials/SEBE.html

Yang, I.-H., & Nam, E.-J., (2010): Economic analysis of the daylight-linked lighting control system in office buildings. Solar Energy, 84(8), 1513–1525.