

Pre-culling geometric linked building data for lightweight viewers

Linked Data

Master's dissertation submitted in order to obtain the academic degree of

Master of Science in de ingenieurswetenschappen: architectuur

Supervisor: Prof. ir.-arch. Paulus Present

Counselors: Ir.-arch. Jeroen Werbrouck
Prof. dr. ir. arch. Ruben Verstraeten

Philippe Soubrier 01702837 philippe.soubrier@ugent.be
Academic year: 2022–2023

Contents

| | | |
|-------|---|----|
| 1 | Introduction | 6 |
| 1.1 | Linked Data | 8 |
| 1.1.1 | RDF and triples | 8 |
| 1.1.2 | Ontologies and reasoning | 10 |
| 1.1.3 | Triplestores and SPARQL | 11 |
| 1.1.4 | Complexity of the data graph | 11 |
| 1.2 | Research questions | 12 |
| 1.2.1 | Can LDBIM be culled? | 13 |
| 1.2.2 | Can existing semantics be used? | 13 |
| 2 | State of the art | 14 |
| 2.1 | Existing BIM viewers and ontologies | 14 |
| 2.1.1 | Qoniq and LOD Streaming for BIM | 14 |
| 2.1.2 | ld-bim.web.app | 15 |
| 2.1.3 | AEC related ontologies | 15 |
| 2.1.4 | GIS related ontologies | 17 |
| 2.2 | On the market viewers comparison | 18 |
| 2.2.1 | General Features | 18 |
| 3 | Culling approaches | 20 |
| 3.1 | AEC related ontologies | 20 |
| 3.1.1 | BOT | 20 |
| 3.1.2 | FOG and OMG | 20 |
| 3.2 | GIS related ontologies | 20 |
| 3.2.1 | geoSPARQL | 20 |
| 4 | Setup | 21 |
| 4.1 | Pages | 21 |
| 4.1.1 | index | 21 |
| 4.2 | Components | 23 |
| 4.2.1 | ARViewer | 23 |
| 4.2.2 | Navbar | 24 |

| | | |
|-------|------------------------------|----|
| 4.2.3 | QueryPannel | 26 |
| 4.3 | Modules | 28 |
| 4.3.1 | Viewer | 28 |
| 4.3.2 | fetchSPARQL | 32 |
| 4.3.3 | useCacheManagement | 34 |
| | References | 37 |
| | Referenced webistes | 39 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | Sequence diagram - basic concept | 7 |
| 1.2 | Illustration of culling principle | 8 |
| 1.3 | Triple structure | 10 |
| 1.4 | Evolution of LOD during the life-cycle of a building | 11 |
| 2.1 | Illustration of the BOT ontology | 16 |
| 2.2 | Illustration of the FOG and OMG ontologies | 17 |
| 2.3 | Performance viewers | 19 |

List of Tables

| | | |
|-----|--|----|
| 1.1 | Size of test-models in Johansson et al., 2015 | 7 |
| 2.1 | Acceleration techniques used by tested viewers | 19 |

Short abstract

This is my short abstract.

Abstract

This is my abstract

Chapter 1

Introduction

From 2D, to 3D and now to [BIM](#). The evolution of the Architecture, Engineering and Construction ([AEC](#)) industry has been a long and complex one. The introduction of 3D modeling was the first major step in the industry's evolution, as it allowed for more accurate representations of buildings. No longer solely relying on 2D drawings, a 3D model of a building can be used to create various representations, from a simple 2D floor plan to a full 3D model. Following the adoption of 3D modeling, the implementation of Building Information Modelling ([BIM](#)) emerged as another significant milestone. [BIM](#) adds an extra layer of information on top of the 3D model. As the digital representation of a building's physical and functional characteristics, BIM serves as a repository for semantics originating from various applications throughout the design and construction processes, including cost estimation, energy analysis, and production planning.

However, as mentioned in Werbrouck, [2018](#), the next challenge for the [AEC](#) industry is related to the ecosystem of current [BIM](#) softwares, which remains closed off to other disciplines. This data management challenge is currently being addressed by the Linked Building Data Community Group ([LBD-CG](#)) and other research entities, such as the University of Ghent, through the use of Web of Data technologies¹. This emerging milestone will be discussed in this thesis under the term Linked Data [BIM](#) ([LDBIM](#)).

Proposal

Each of these evolutions has brought, and will continue to bring, a significant amount of data together. This volume is expected to grow exponentially in the future as the industry shifts towards a more digital approach and opens up to other stakeholders. The data graphs will not only expand in terms of semantics but also in geometry. This makes visual querying, or simply put, 3D exploration of models, an increasingly difficult task. Especially when looking at newer devices used in the industry such as mobile

¹W3C, [2023](#).

phones, and tablets, which are becoming more and more powerful, but still have limited computational resources in comparison to office computers.

To bring this volume of geometric data in perspective, Table 1.1 shows the size of the test-models used in Johansson et al., 2015 , a study from 2015 on the performance of BIM viewers for large models with the following description:

“ Although the Hotel model contains some structural elements they are primarily architectural models. As such, no Mechanical, Electrical or Plumbing (MEP) data is present. However, all models except the Hospital contain furniture and other interior equipment. ” (Johansson et al., 2015)

| Model | # of triangles | # of objects | # of geometry batches |
|---------------|----------------|--------------|-----------------------|
| Library | 3 685 748 | 7318 | 11 195 |
| Student House | 11 737 251 | 17 674 | 33 455 |
| Hospital | 2 344 968 | 18627 | 22 265 |
| Hotel | 7 200 901 | 41 893 | 62 624 |

Table 1.1: Size of test-models in Johansson et al., 2015

These models demonstrate how basic BIM models can already contain a significant amount of data. LDBIM will not only bring together new stakeholders but also have the capability to keep track of multiple geometry versions for each object, should they occur. Therefore, this thesis proposes a new approach to the visual querying of LDBIM models, wherein viewers will not have to load the entire model into memory. Instead, after filtering at the source, only the geometry needed for the visual tasks at hand will be loaded, while maintaining the original link to each resource for further processing and use cases. This filtering step is commonly referred to as culling in the computer graphics industry and is illustrated in Figures 1.1 and 1.2.

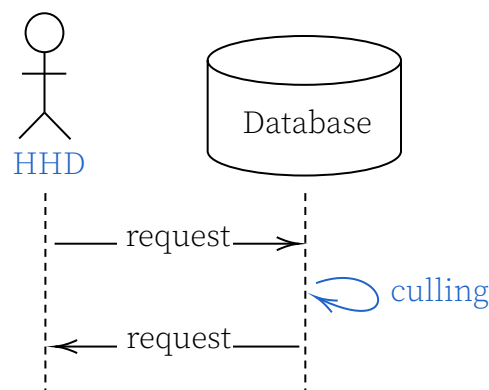


Figure 1.1: Sequence diagram - basic concept

Figure 1.1 illustrates the basic idea of this thesis, presenting an extra step in the communication between a user, represented here by a Hand Held Device (HHD), and a

database storing the model. An [HHD](#) has been chosen to exemplify a low-powered device used in the field, which requires a lightweight 3D viewer to visualize and explore the digital twin of the building. The [HHD](#) is assumed to have no knowledge of the [LDBIM](#) model and only receives the geometry that needs to be displayed from the database. On the other hand, the database is assumed to possess, or have access to, all the knowledge of the model and the necessary semantics to perform the culling.

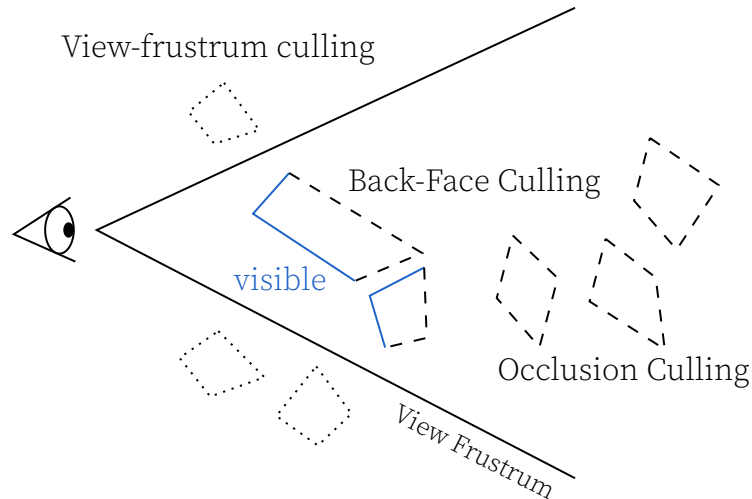


Figure 1.2: Illustration of culling principle, based on Cohen-Or et al., [2003](#)

Figure [1.2](#) showcases multiple culling techniques to showcase some culling principles. The first technique, *frustum culling*, is used to determine which objects are visible to the user. The second technique, *occlusion culling*, is used to determine which objects are occluded by or behind other objects. And lastly, *back-face culling*, is used to determine which faces, and not whole objects, are facing away from the user.

1.1 Linked Data

As mentioned in the [Introduction](#), the evolution from [BIM](#) to [LDBIM](#) is an evolution of the data *management* layer. “Linked Data”, as stated by the World Wide Web Consortium ([W3C](#)), is a collection of interrelated datasets on the Web, formatted in a standard way that is accessible and manageable by Semantic Web tools. The same applies to the relationships among them.² The following collection of Semantic Web technologies explores the required environment to achieve this goal.

1.1.1 [RDF](#) and triples

At the core of the Semantic Web is the Resource Description Framework ([RDF](#)), a data model for describing resources on the Web. RDF is a graph data model that consists of

²W3C, [2015b](#).

triples, which are statements about resources. A triple consists of a subject, a predicate, and an object. The subject is the resource that is being described, the predicate is the property of the subject, and the object is the value of the property. Both the predicate and the object can, in turn, become the subjects of other triples. Listing 1.1 shows an example of an [RDF](#) database described in the Turtle format.

```
@prefix fog: <https://w3id.org/fog#> .
@prefix omg: <https://w3id.org/omg#> .
@prefix bot: <https://w3id.org/bot#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix flupke: <http://flupke.archi#> .

flupke:room1 rdf:type bot:Zone ;
  bot:containsElement flupke:coneOBJ ;
  bot:containsElement flupke:cubeGLTF .

flupke:coneOBJ omg:hasGeometry flupke:coneOBJ_geometry-1 ;
  rdf:type bot:Element .

flupke:cubeGLTF omg:hasGeometry flupke:cubeGLTF_geometry-1 ;
  rdf:type bot:Element .

flupke:coneOBJ_geometry-1 rdf:type omg:Geometry ;
  fog:asObj_v3.0-obj
  ↪ "https://raw.githubusercontent.com/flo13622/AR-Linked-BIM-viewer/
  ↪ main/public/assets/database_1/coneOBJ.obj"^^xsd:anyURI
  ↪ .

flupke:cubeGLTF_geometry-1 rdf:type omg:Geometry ;
  fog:asGltf_v1.0-gltf
  ↪ "https://raw.githubusercontent.com/flo13622/AR-Linked-BIM-viewer/
  ↪ main/public/assets/database_1/cubeGLTF.gltf"^^xsd:anyURI
  ↪ .
```

Listing 1.1: Example of an [RDF](#) database in turtle format

The basic, yet versatile, structure of a triple is illustrated in Figure 1.3. Both the subject and object are considered as nodes in the data graph, and they are linked by the predicate, which is referred to as an edge. Multiple triples can thus create and link multiple nodes or enrich a connection between two nodes by creating new edges between them. Each element contains a single resource that can be one of the three types: a [URI](#), a literal, or a blank node. A Uniform Resource Identifier ([URI](#)) points to a resource on the web and, as its name states, is unique and unambiguous, thus enabling queries and reasoning of the same nature. A literal is a value, and a blank node is an anonymous

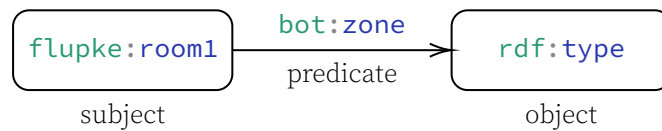


Figure 1.3: Triple structure

resource, sometimes used as a placeholder when the exact resource is not known or not necessary to specify. Due to their nature, a subject must be either a [URI](#) or a blank node, a predicate exclusively a [URI](#), and the object may be any of the three types. As [URI](#) descriptions can be very long, a prefix can be used to shorten them. This is illustrated in Listing 1.1 with the `@prefix bot: <https://w3id.org/bot#>`, which declares that `bot:Zone` refers, in its full length, to the address `<https://w3id.org/bot#Zone>`.

This basic concept can be extrapolated to describe and store any kind of data. The advantage for the [AEC](#) industry would be to allow any stakeholders to describe and enrich the knowledge base of a building.

1.1.2 Ontologies and reasoning

When looking at Listing 1.1, a distinction can be made between two types of statements: some refer to classes or properties, such as `flupke:room1`, while others refer to facts associated with them, like `rdf:type`. The former is referred to as TBox for “terminology”, and the latter is referred to as ABox for “assertions”. The TBox is the part of the ontology that describes the classes and properties of the domain, while the ABox is the part of the ontology that describes the facts about the domain.

By developing an ontology, the domain of interest and the relationships between the classes and properties can be described. This is achieved by defining the classes and properties of the domain and their relationships. The ontology is then used to reason about the domain, inferring new facts based on the ontology and the existing facts within the domain. This is done by a reasoner, which is software capable of performing the reasoning itself on the ontology and associated data. As mentioned, the reasoner can be used to infer new facts, check if created facts are consistent with the ontology, and check if the ontology itself is consistent.³ It is often integrated with [RDF](#) databases, also known as triplestores or graph databases.

Classes, properties, and their relationships can be defined using Resource Description Framework Schema ([RDFS](#)), which is a vocabulary for describing [RDF](#) schemas using a basic set of constructs. As an extension of [RDFS](#), Web Ontology Language ([OWL](#)) is a vocabulary for describing ontologies using a more expressive set of constructs tailored to the needs of ontologies. Both [RDFS](#) and [OWL](#) are considered to be formal ontologies themselves, as they describe the classes and properties of the domain of [RDF](#).

³W3C, 2015a.

1.1.3 Triplestores and SPARQL

As briefly discussed in 1.1.2, triplestores are [RDF](#) databases that store data in the form of a graph. They are used to store and query Linked Data and are often integrated with a reasoner. The data itself is retrieved and modified using the SPARQL Protocol and [RDF](#) Query Language ([SPARQL](#)).⁴ In contrast to Structured Query Language ([SQL](#)), [SPARQL](#) queries are able to work across multiple triplestores, called [SPARQL](#) endpoints. These are known as federated queries, and their results are combined into a single result set. This is useful when the data is distributed across multiple triplestores in a decentralized manner.⁵ For example, multiple stakeholders participating in a project, each with their own database.

1.1.4 Complexity of the data graph

The complexity of the data graph is a major concern when working with [LDBIM](#). This section discusses the origins of the different sources of geometric data that enrich it.

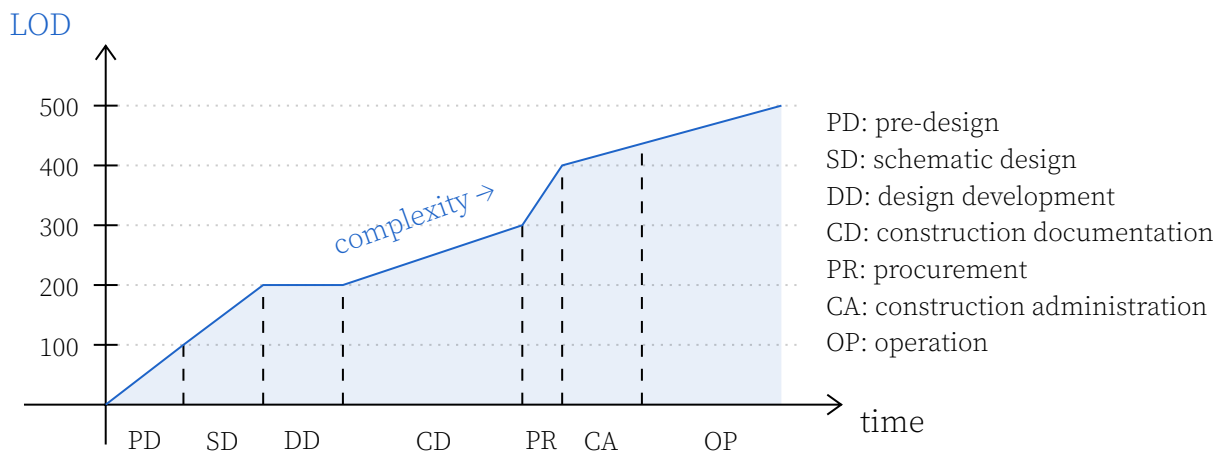


Figure 1.4: Evolution of [LOD](#) during the life-cycle of a building.

Based upon the Macleamy Curve (Ilozor & Kelly, [2012](#))

random values at the moment

1.1.4.1 BIM geometry

The 3D model of a building consists of a multitude of sub-models, describing objects for all the different stakeholders participating in the project. Some describe very large objects, and some very small parts. Both can be defined in their most simple and abstract form or have an intricate and complex geometry. For instance, a door can simply be defined as a box, or up to the level of the screw-thread for the hinge system. The level of abstraction is here described as the Level of Detail ([LOD](#)), which is most of

⁴W3C, [2015c](#).

⁵Ontotext, [2022](#).

the time pre-selected for the needs of a [BIM](#) model, and is applied throughout a single model.

As shown in Figure 1.4, a standard BIM workflow goes through multiple phases, each with their associated model and [LOD](#). This makes it an important concept in the [AEC](#) industry, as it allows for a very efficient workflow. The modeling step is approached from a top-down perspective, starting with rougher geometries describing the broader ideas of a concept model and evolving to a more refined model for the construction documentation phase. As the last and longest-standing model, a higher [LOD](#) can be used to describe subtle changes in the evolution of a building during the operation phase.

1.1.4.2 [LDBIM](#) geometry

The interconnectivity of semantics can also be applied to geometry descriptions. This could allow the co-existence of multiple [LODs](#) in a single model database. Besides storing the evolution of a single element's geometry, it enables the linking of the different [LODs](#), described in 1.1.4.1, to each other. Not only that, but extending onto the size of the models described in Table 1.1, already existing Mechanical, Electrical and Plumbing ([MEP](#)), structural, alongside many other stakeholders' geometry can be added.

1.2 Research questions

Johansson et al., 2015 presented in their paper a new [BIM](#) viewer equipped with the powerful Coherent Hierarchical Culling algorithm ([CHC](#))++. This is a third-generation occlusion culling algorithm developed by Mattausch et al., 2008a, the first being the [CHC](#) (Bittner et al., 2004), followed by the Near Optimal Hierarchical Culling ([NOHC](#)) (Mattausch et al., 2008b). Their conclusion stated that although occlusion culling is very efficient, it is still bound to the scene size, which is limited by hardware capabilities. More precisely, the Graphics Processing Unit ([GPU](#)), Video Random Access Memory ([VRAM](#)), and Random Access Memory ([RAM](#)) capacities.

This thesis proposes the introduction of culling algorithm technology within the context of [LDBIM](#) to address the previously mentioned issue of the scene's size, by culling the scene at its source prior to sending it to the viewer. As culling algorithms have been extensively researched and continue to evolve, as described in the previous paragraph, the research questions in this thesis concentrate on assessing the feasibility of introducing such algorithms in [LDBIM](#). It aims to propose a set of possible solutions tailored to this specific problem, while highlighting possibilities for future research and specific use cases.

1.2.1 To what extent can [LDBIM](#) geometry be culled to be streamed to lightweight viewers?

This thesis focuses on computing with data snippets or triples inside a [LDBIM](#) model, not within. Meaning that the smallest unit of data that can be culled is the one described in one triple, which is in the most likely scenario, a single [LOD](#) of a single element. It implies that geometry is defined and separated at the object-level. It also implies that culling techniques such as back-face culling will not be handled in this thesis, and will be left to the viewer itself, not the database.

Which snippets of data are needed by the viewer? Is part of the question. The basic needs of the viewer consist firstly of the geometry itself, selecting the right geometry format for the application as well as the additional visual information such as color, texture, etc. Secondly, the identifier of each element is of crucial importance to maintain the link to other semantic resources in the graph. This enables the viewer to retrieve those resources for a multitude of use cases, transforming it into a user-friendly visual query tool.

1.2.2 Can existing semantics and ontologies be used to feed possible culling algorithms?

In contrast to the computer graphics industry, this interconnected context already contains both explicit and implicit, through inferencing, relationships inside the graph. Similarly to Johansson and Roupé, [2009](#) and their paper where they used the semantics of a [BIM](#) model in Industry Foundation Classes ([IFC](#)) format to develop culling techniques. However, this thesis will focus on the use of Semantic Web resources. Therefore, it will analyze both [AEC](#)-specific and [AEC](#)-related ontologies, such as Geographic Information System ([GIS](#))-related, to see if they can be used to feed culling algorithms.

Chapter 2

State of the art

As mentioned in Johansson et al., 2015, existing research on the performance of currently used BIM viewers is quite limited. This state-of-the-art research will, therefore, focus on the overall features of some promising newer viewers and the ontologies that will be used in this thesis.

2.1 Existing BIM viewers and ontologies

2.1.1 Qoniq and LOD Streaming for BIM

Qoniq focuses on developing an open platform BIM viewer. With the use of Unity to enable cross-platform compatibility, they focused on two main aspects: performance and aesthetics. The latter refers to the visual quality of the viewer, offering both a seamless experience for the viewer as well as a pleasant one, with, for example, the implementation of ambient lighting and shadow castings. The first and most researched aspect of their viewer, the performance, is mainly focusing on a LOD culling algorithm.

(T. Strobbe, personal communication, November 25, 2022)

2.1.1.1 Qoniq's approach to LOD streaming

Their core research is developing a dynamic LOD streaming model. Starting from the geometry and semantics of an IFC file, they compute an LOD hierarchy tree of the model. Through multiple mesh decimation algorithms, they reduce the number of triangles of each object's mesh, regardless of the semantics associated with that object. On top of that, a filtering algorithm is implemented in the streaming model to filter out objects, regarding their semantics, that are not relevant to the current camera position. In doing so, they both reduce the size of models far from the viewpoint and evaluate the need to show certain objects based on their nature, extracted from semantics in the IFC file, and their distance to the camera. The resulting dynamic LOD streaming

model is reevaluated at each camera move in Unity.
(T. Strobbe, personal communication, November 25, 2022)

Unity was chosen as it allows for writing once and deploying everywhere. This means that the viewer can be used on any platform, including mobile devices and browsers. The performance results are thus related to the hardware capabilities of each device, with the exception of the browser, where the performance of Unity’s WebGL build is limited to a scene size of 2Gb.¹

2.1.1.2 Advantages and trade-offs

Being able to run on many platforms, offering a smooth viewer experience and a pleasing aesthetic makes it an ideal candidate for lightweight viewers on the job site. However, the LOD library has to be computed on every model update. The decimation algorithms are furthermore computational results that are not humanly reviewed. This means that the quality of the resulting meshes is not guaranteed for the lower LODs, which are, as illustrated in Figure 1.4, already modeled in previous design phases. LDBIM could, by interconnection, recall previous LODs in the viewer’s scene. Without the need for computational remodeling. Nevertheless, Qonic serves as this thesis’s goal, outside the LDBIM context.

2.1.2 ld-bim.web.app

“The purpose of the app is to showcase our LBD toolset and to demonstrate the capabilities of Linked Building Data to newcomers.”²

<https://ld-bim.web.app/> demonstrates a viewer built around an RDF database. It separates the data from an IFC file into semantics, stored in the previously mentioned graph, and a glTF model, together with a JavaScript Object Notation (JSON) file containing a reference table. Extra local or remote graphs can be added to the User Interface (UI). As it contains a SPARQL engine to query and visualize, in the form of highlighting, the results of the query in a 3D viewer. The viewer is based on the ifc.js project, which is itself based on the three.js 3D JavaScript library.

2.1.3 AEC related ontologies

As mentioned in the second research question 1.2.2, this section will discuss AEC-related technologies, some of which are not yet approved by the W3C but are actively researched by the LBD-CG³

¹Unity, 2023.

²Rasmussen and Schlachter, n.d.

³LBD-CG, 2022.

2.1.3.1 BOT

The Building Topology Ontology (BOT) proposes a set of classes and properties, “which provides a high-level description of the topology of buildings including storeys and spaces, the building elements they may contain, and the 3D mesh geometry of these spaces and elements.” (Rasmussen et al., 2020), as illustrated in Figure 2.1. This high-level description could be fed to portal-culling algorithms in a situation where the visibility is contained within one `bot:Space` or `bot:Storey`, or it could extend the scope to `bot:adjacentZone`⁴. Additionally, it could play a part in the construction of the Bounding Volume Hierarchy (BVH) needed for other occlusion culling algorithms, such as the CHC++ (Johansson et al., 2015).

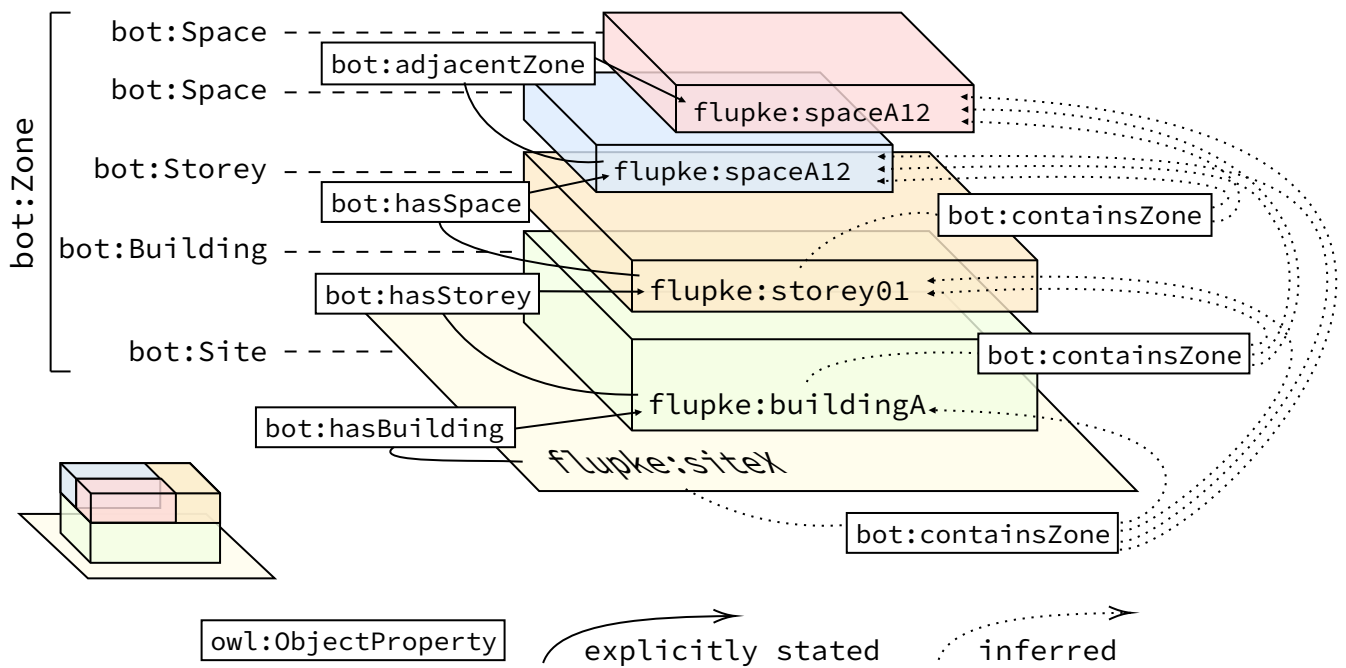


Figure 2.1: Illustration of the BOT ontology, based on Rasmussen et al., 2020.

2.1.3.2 FOG and OMG

With the help of File Ontology for Geometry formats (FOG) and Object Management Group (OMG), geometry descriptions can be linked in the data graph. The innovation lies in the choice to store it either inside or outside the graph, by means of one triple referring to a literal or an URI. Listing 1.1 showcases multiple examples of objects assigned with a geometry description using an URI (Bonduel et al., 2019).

```
flupke:coneOBJ_geometry-1 fog:asObj_v3.0-obj "https://..."^^xsd:anyURI .
```

Listing 2.1: Example of FOG usage

⁴Linietsky et al., 2023.

Listing 2.1 describes a subject of datatype `xsd:anyURI` from the Extensible Markup Language (XML) Schema Definition (XSD)⁵. The versatile approach of Bonduel et al., 2019 also proposes the following datatypes: `xsd:string` for American Standard Code for Information Interchange (ASCII)-based geometry descriptions or `xsd:base64Binary` for binary geometry descriptions.

The format of the geometry is assigned directly by the predicate in Listing 2.1, which is `fog:asObj_v3.0-obj`. This further infers the statements in Listing 2.2.⁶

```
f1upke:coneOBJ_geometry-1 fog:asObj "https://..."^^xsd:anyURI ;
ex:LOD "100"^^xsd:integer .
```

Listing 2.2: FOG inference examples

Bonduel et al., 2019 refers to the proposal of the LBD-CG stated in Wagner et al., 2019 “to allow the modeling of properties on three levels”. The first and second levels are illustrated in Figure 2.2. Level 2 allows assigning multiple geometry descriptions to a single object, each with, for example, a different LOD.

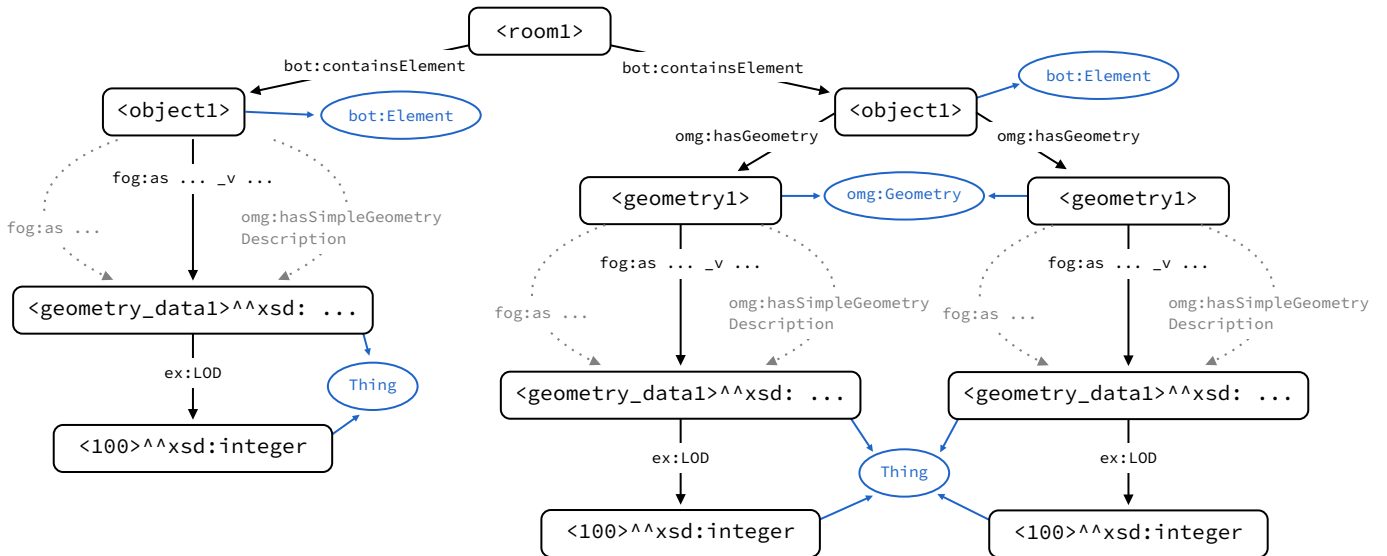


Figure 2.2: Illustration of Level 1 (left) and Level 2 (right) of the FOG and OMG ontologies, based on Bonduel et al., 2019. LOD can't be assigned to literal, needs to be changed

2.1.4 GIS related ontologies

The technological field of study, Geographic Information System (GIS), is closely related to the BIM domain. The central standards organization, Open Geospatial Con-

⁵Carrol and Pan, 2006.

⁶Bonduel et al., 2020.

sortium (OGC), which actively maintains the GIS standards, is also prominent on the Semantic Web scene.⁷ With standards such as:

2.1.4.1 GeoSPARQL

“The OGC GeoSPARQL standard supports representing and querying geospatial data on the Semantic Web. GeoSPARQL defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data. In addition, GeoSPARQL is designed to accommodate systems based on qualitative spatial reasoning and systems based on quantitative spatial computations.”⁸

As multiple triplestores and SPARQL endpoints support the GeoSPARQL extension, it is a viable candidate for spatial and LOD culling algorithms. Such algorithms require spatial data, such as the distance from the viewpoint to the object. Spatial query functions proposed in this extension are needed for this purpose. The functions can compute on nodes of geospatial geometry as if they are expressed using Well-Known Text (WKT) or the Geography Markup Language (GML). These expressions can be assigned by using the predicates `geo:asWKT` or `geo:asGML`. However, GeoSPARQL comes with some limitations that are less prevalent in the GIS domain, which mostly requires 2D data (Perry & Herring, 2012), in contrast to BIM where 3D distance functions would be needed. Despite such limitations, GeoSPARQL remains a viable solution for spatial querying, and workarounds could be employed to address them.

2.2 On the market viewers comparison

Johansson et al., 2015 mentioned in their paper the lack of research about objective BIM viewers comparison and made one as a result. The size of the model they tested can be found in Table 1.1. They evaluated the following viewers:

- DDS CAD Viewer
- Tekla BIMsight
- Autodesk Navisworks
- Solibri Model Viewer

2.2.1 General Features

Their study had two main goals. Firstly, evaluating existing viewers and their capabilities, they identified the acceleration techniques used, which are presented in Table 2.1.

⁷OGC, 2023a.

⁸OGC, 2023b.

| BIM viewer | Acceleration technique |
|----------------|--|
| Solibri 9.0 | VFC DC (optional) HAGI (optional) |
| Naviswork 2015 | VFC DC (optional) CPU OC (optional) GPU OC (optional) |
| BIMsight 1.9.1 | VFC |
| DDS 8.0 | VFC DC (optional) |
| DDS 10.0 | VFC DC |

Table 2.1: Acceleration techniques used by tested viewers from Johansson et al., 2015. (View Frustum Culling (VFC), Drop Culling (DC), Hardware Accelerated Geometry Instancing (HAGI), Central Processing Unit (CPU), Occlusion Culling (OC))

Secondly, they implemented modern culling algorithms and strategies such as CHC++. The worst-case scenarios are shown in Figure 2.3 against the Solibri viewer. The results are quite promising, but as concluded by the authors, the gains are limited to the capacities of the GPU, VRAM, and RAM, as discussed in 1.2.

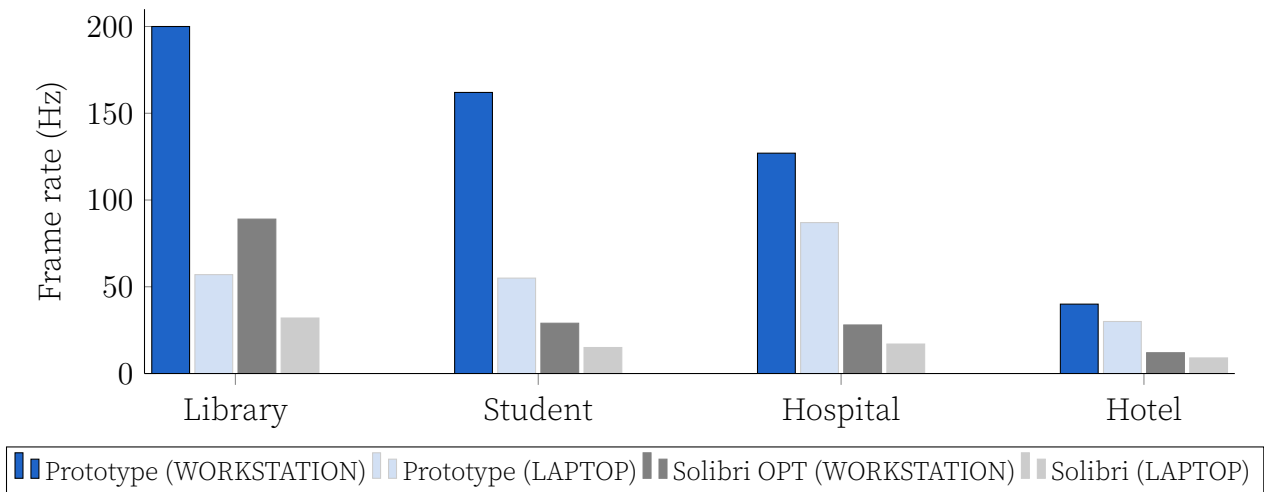


Figure 2.3: Comparison in rendering performance.
from Johansson et al., 2015

Chapter 3

Culling approaches

3.1 [AEC](#) related ontologies

3.1.1 [BOT](#)

3.1.2 [FOG](#) and [OMG](#)

3.2 [GIS](#) related ontologies

3.2.1 [geoSPARQL](#)

Chapter 4

Setup

4.1 Pages

4.1.1 index

```
import dynamic from "next/dynamic";
import Head from "next/head";
import { Navbar, Querypanel } from "~/components";

const ARViewer = dynamic(() => import("~/components/ARViewer"), {
  ssr: false,
});

export default function Home() {
  return (
    <>
      <Head>
        <title>Create T3 App</title>
        <meta name="description" content="Generated by create-t3-app" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <main>
        <div className="absolute top-14 h-[calc(100vh-3.5rem)] w-full
→ overflow-hidden">
          <ARViewer />
        </div>
        <Navbar />
        <Querypanel/>
      </main>
    </>
  );
}
```

```

import dynamic from "next/dynamic";
import Head from "next/head";
import { Navbar, Querypanel } from "~/components";

const ARViewer = dynamic(() => import("~/components/ARViewer"), {
  ssr: false,
});

export default function Home() {
  return (
    <>
      <Head>
        <title>Create T3 App</title>
        <meta name="description" content="Generated by create-t3-app" />
        <link rel="icon" href="/favicon.ico" />
      </Head>
      <main>
        <div className="absolute top-14 h-[calc(100vh-3.5rem)] w-full
→ overflow-hidden">
          <ARViewer />
        </div>
        <Navbar />
        <Querypanel/>
      </main>
    </>
  );
}

```

4.2 Components

4.2.1 ARViewer

```
import { Viewer } from "@xeokit/xeokit-sdk";
import { useRef } from "react";
import { useRecoilValue, useSetRecoilState } from "recoil";
import { cleanStart, endpoint, freezing, lruLimit, uiQuery } from
  ↳ "~/atoms";
import { useCacheManagement } from "~/modules/useCacheManagement";
import { LoaderType, useInitViewer } from
  ↳ "~/modules/viewer/useInitViewer";
import { useLoadGeometry } from "~/modules/viewer/useLoadGeometry";

export default function ARViewer() {
  const clean = useRecoilValue(cleanStart);
  const uiQueryValue = useRecoilValue(uiQuery);
  const setFreeze = useSetRecoilState(freezing);
  const endpointValue = useRecoilValue(endpoint);
  const lruLimitValue = useRecoilValue(lruLimit);

  const viewerRef = useRef<Viewer>();
  const loaderTypesRef = useRef<LoaderType>();

  // LRU cache management
  const { clearLRU, evalLRU } = useCacheManagement(
    lruLimitValue,
    viewerRef.current
  );

  // initialize the setup
  useInitViewer(viewerRef.current, loaderTypesRef, clean, setFreeze);

  // fetch the main query
  useLoadGeometry(uiQueryValue, endpointValue, loaderTypesRef, evalLRU);

  return (
    <>
      <canvas id="myCanvas" className="mt-2 h-full w-full"></canvas>
      <canvas
        className="fixed right-0 bottom-0 h-40 w-40"
        id="myNavCubeCanvas"
      ></canvas>
    </>
  );
}
```


4.2.2 Navbar

```
import InfoIcon from "@mui/icons-material/Info";
import { TextField, Tooltip } from "@mui/material";
import Autocomplete from "@mui/material/Autocomplete";
import { RocketIcon, TrashIcon } from "@radix-ui/react-icons";
import { useState } from "react";
import { useRecoilState, useRecoilValue } from "recoil";
import { cleanStart, defaultEndpoints, endpoint, freezing } from
  ↪ "~/atoms";

function Divider() {
  return <div className="border-black self-stretch border-l
  ↪ border-dashed" />;
}

export default function Navbar() {
  const [clean, setClean] = useRecoilState(cleanStart);
  const [endpointValue, setEndpoint] = useRecoilState(endpoint);
  const dftEndpoints = useRecoilValue(defaultEndpoints);
  const freezingValue = useRecoilValue(freezing);
  const [tempEndpoint, setTempEndpoint] = useState("");

  const updateEndpoint = () => {
    setEndpoint(tempEndpoint);
    console.log("updated endpoint:", endpointValue);
  };

  const handleKeyDown = (event: React.KeyboardEvent) => {
    if (event.key === "Enter") {
      updateEndpoint();
    }
  };

  return (
    <div className="border-black absolute z-10 flex h-16 w-full
  ↪ items-center justify-between gap-2 border-b bg-white p-2 shadow">
      <div className="flex h-full flex-grow items-center gap-2">
        <Tooltip title="Clean viewer">
          <TrashIcon className="mx-2" onClick={() => setClean(!clean)} />
        </Tooltip>
        <Divider />
        <Autocomplete
          disabled={freezingValue}
          size="small"
          sx={{ flexGrow: 1, maxWidth: 600 }}
          freeSolo
          options={dftEndpoints}
          onChange={(_, input) => {
            setTempEndpoint(input);
          }}
        />
      </div>
    </div>
  );
}
```

```

    }}
    onKeyDown={handleKeyDown}
    renderInput={({params}) => (
      <TextField {...params} label="Type endpoint" />
    )}
  />
  <Tooltip title="Got to database">
    <button disabled={freezingValue} onClick={updateEndpoint}>
      <RocketIcon
        className="mx-2"
        style={{ color: freezingValue ? "gray" : "black" }}
      />
    </button>
  </Tooltip>
</div>
<Divider />
<h1 className="text-right">
  Pre-culling geometric linked building data
  <br />
  for lightweight viewers
  <br />
</h1>

  <Tooltip title="info">
    <div className="mx-2 flex items-center">
      <InfoIcon />
    </div>
  </Tooltip>
</div>
);
}

```

4.2.3 QueryPannel

```
import { PaperPlaneIcon } from "@radix-ui/react-icons";
import TextareaAutosize from "@mui/base/TextareaAutosize";
import { useEffect, useState } from "react";
import { useRecoilState, useRecoilValue } from "recoil";
import { freezing, uiQuery } from "~/atoms";

export default function Querypanel() {
  const freezingValue = useRecoilValue(freezing);
  const [queryValue, setQuery] = useRecoilState(uiQuery);
  const [tempQuery, setTempQuery] = useState("");

  useEffect(() => {
    setTempQuery(queryValue);
  }, []);

  const updateQuery = () => {
    setQuery(tempQuery);
    console.log("updated query");
  };

  const handleKeyDown = (e: any) => {
    if (e.key === "Enter" && e.shiftKey) {
      updateQuery();
    }
  };

  return (
    <div className="absolute bottom-4 left-4 flex w-[400px] flex-col
    ↪ rounded border bg-white p-2 shadow-lg">
      <h3 className="pb-1">Query</h3>
      <hr />
      <TextareaAutosize
        minRows={4}
        aria-label="maximum height"
        placeholder="Write query"
        defaultValue={tempQuery}
        spellCheck="false"
        style={{
          width: "100%",
          padding: "0.5rem",
          fontFamily: '"Fira code", "Fira Mono", monospace',
          fontSize: "0.7rem",
          opacity: freezingValue ? 0.3 : 1,
        }}
        disabled={freezingValue}
        onChange={(e) => setTempQuery(e.target.value)}
        onKeyDown={handleKeyDown}
      />

```

```

    {freezingValue && (
      <div className="absolute flex h-full w-full items-center
↪ justify-center ">
        <p className="bg-white p-2 ">
          Query is frozen while the viewer is initializing
        </p>
      </div>
    )}
    <div className="my-2 flex w-full items-center justify-between gap-2
↪ px-2 ">
      <PaperPlaneIcon className="cursor-pointer" onClick={updateQuery}
↪ />
      <p className="text-sm opacity-50">shortcut: Shift + Enter</p>
    </div>
  </div>
);
}

```

4.3 Modules

4.3.1 Viewer

4.3.1.1 useInitViewer

```
import {
  GLTFLoaderPlugin,
  NavCubePlugin,
  OBJLoaderPlugin,
  STLLoaderPlugin,
  Viewer,
} from "@xeokit/xeokit-sdk";
import { useEffect } from "react";

export type LoaderType = {
  [key: string]: {
    loader: any; // could not find a general type for loaders
    params?: any;
    litParam?: string;
    uriParam?: "src";
  };
};

export function useInitViewer(
  viewer: Viewer | undefined,
  loaderTypes: React.MutableRefObject<LoaderType | undefined>,
  clean: any,
  setFreeze: (state: boolean) => void
) {
  useEffect(() => {
    // freeze the query and endpoint inputs
    setFreeze(true);
    // clear the existing canvas
    viewer?.scene.clear();

    // initialize the viewer
    viewer = new Viewer({
      canvasId: "myCanvas",
      transparent: true,
    });

    // initialize the navcube
    new NavCubePlugin(viewer, {
      canvasId: "myNavCubeCanvas",
      visible: true,
    });

    // identify the scene and camera
  });
}
```

```

const scene = viewer.scene;
const camera = scene.camera;
camera.projection = "perspective";

// initialize the loaders
const gltfLoader = new GLTFLoaderPlugin(viewer);
const objLoader = new OBJLoaderPlugin(viewer, {});
const stlLoader = new STLLoaderPlugin(viewer);

// store the loaders in a ref
loaderTypes.current = {
  "https://w3id.org/fog#asGltf": {
    loader: gltfLoader,
    params: { edges: true },
    litParam: "gltf",
    uriParam: "src",
  },
  "https://w3id.org/fog#asStl": {
    loader: stlLoader,
    params: { edges: true, rotation: [180, 0, 0] },
    litParam: "stl",
    uriParam: "src",
  },
  "https://w3id.org/fog#asObj": {
    loader: objLoader,
    uriParam: "src",
  },
};
console.log("viewer initialized");
setFreeze(false);
}, [clean]);
}

```

4.3.1.2 useLoadGeometry

```
import { useEffect } from "react";
import { getEntities, getGeometry } from "../fetchSPARQL";
import { EntryLRU } from "../useCacheManagement";
import { LoaderType } from "../useInitViewer";

async function loadGeometry(
  uiQuery: string,
  endpoint: string,
  loaderTypes: React.MutableRefObject<LoaderType | undefined>,
  evalLRU: (entry: EntryLRU) => boolean
) {
  await getEntities(uiQuery, endpoint, (bindings: any) => {
    try {
      const entry = {
        id: bindings.element.value,
        metadata: {
          format: bindings.fog_geometry.value,
          datatype: bindings.geometryData.datatype.value,
        },
      };
      as EntryLRU;

      // cache management, stop if needed
      if (!evalLRU(entry)) throw new Error("Already in cache");

      // else fetch geometry data
      const loaderType = loaderTypes.current?.[entry.metadata.format];
      getGeometry(entry.id, entry.metadata.format, endpoint)
        .then((data) => {
          console.log(data);
          // if the data is a literal, and is supported
          if (
            entry.metadata.datatype ===
            "http://www.w3.org/2001/XMLSchema#string" &&
            loaderType?.litParam
          ) {
            loaderType.loader.load({
              ...loaderType.params,
              id: entry.id,
              [loaderType.litParam]: data,
            });
          }
          // if the data is a uri, and is supported
          else if (
            entry.metadata.datatype ===
            "http://www.w3.org/2001/XMLSchema#anyURI" &&
            loaderType?.uriParam
          ) {
            loaderType.loader.load({
```

```

        ...loaderType.params,
        id: entry.id,
        [loaderType.uriParam]: data,
    });
}
// if the data source is not supported
else console.log("unsupported / undefined data source",
↵ entry.id);
})
.catch((error) => {
    console.error("Error fetching geometry data:", error);
});
} catch (error) {
    console.error("Error loading geometry:", error);
}
});
}

export function useLoadGeometry(
    uiQuery: string,
    endpoint: string,
    loaderTypes: React.MutableRefObject<LoaderType | undefined>,
    evalLRU: (entry: EntryLRU) => boolean
) {
    useEffect(() => {
        if (endpoint) {
            // ensures no loading on first render
            loadGeometry(uiQuery, endpoint, loaderTypes, evalLRU);
        }
    }, [endpoint, uiQuery]);
}

```


4.3.2 fetchSPARQL

```
import { SparqlEndpointFetcher } from "fetch-sparql-endpoint";
import { Format } from "../viewer/types";

const myFetcher = new SparqlEndpointFetcher();

// -----
// Fetch entities for a given query
// -----
async function getEntities(
  query: string,
  endpoint: string,
  forEachEntry: (bindings: any) => void
): Promise<void> {
  const bindingsStream = await myFetcher.fetchBindings(endpoint, query);

  bindingsStream.on("data", (bindings: any) => {
    forEachEntry(bindings);
  });

  bindingsStream.on("error", (error: any) => {
    console.error("Error fetching entities:", error);
  });
}

// -----
// Fetch geometry data of a given entity
// -----

function getGeometryQuery(id: any, format: Format): string {
  return `SELECT ?geometryData
  WHERE {
    <${id}> <${format}> ?geometryData .
  }
  LIMIT 1`;
}

async function getGeometry(id: any, format: Format, endpoint: string) {
  return new Promise(async (resolve, reject) => {
    const bindingsStream = await myFetcher.fetchBindings(
      endpoint,
      getGeometryQuery(id, format)
    );

    bindingsStream.on("data", (bindings: any): void => {
      const data = bindings.geometryData.value;
      resolve(data);
    });
  });
}
```

```
bindingsStream.on("error", (error: any): void => {
  reject(error);
});

bindingsStream.on("end", (): void => {
  reject(new Error("No data found"));
});
}

export { getEntities, getGeometry };
```

4.3.3 useCacheManagement

```
import { LRUMap } from "lru_map";
import { Datatype, Format } from "../viewer/types";
import { useRef } from "react";
import { Viewer } from "@xeokit/xeokit-sdk";

export type MetadataLRU = {
  format: Format;
  datatype: Datatype;
};

export type EntryLRU = {
  id: string;
  metadata: MetadataLRU;
};

export function useCacheManagement(limit: number, viewer: Viewer |
↳ undefined) {
  const LRU = useRef<LRUMap<string, MetadataLRU>>(new LRUMap(limit));

  LRU.current.shift = function () {
    let entry = LRUMap.prototype.shift.call(this);
    viewer?.scene.models[entry?.[0]]?.destroy();
    return entry;
  };

  function clearLRU(): void {
    LRU.current.clear();
  }

  function addLRU(entity: EntryLRU): void {
    LRU.current.set(entity.id, entity.metadata);
  }

  function evalLRU(entity: EntryLRU): boolean {
    if (LRU.current.get(entity.id) === entity.metadata) return false;
    addLRU(entity);
    return true;
  }

  return { clearLRU, evalLRU };
}
```

The Xeokit Software Development Kit ([SDK](#)) ...

List of Acronyms

| | | |
|--------|--|----|
| AEC | Architecture, Engineering and Construction | 6 |
| ASCII | American Standard Code for Information Interchange | 17 |
| BIM | Building Information Modelling | 6 |
| BOT | Building Topology Ontology | 16 |
| BVH | Bounding Volume Hierarchy | 16 |
| CHC | Coherent Hierarchical Culling algorithm | 12 |
| CPU | Central Processing Unit | 19 |
| DC | Drop Culling | 19 |
| FOG | File Ontology for Geometry formats | 16 |
| GIS | Geographic Information System | 13 |
| GML | Geography Markup Language | 18 |
| GPU | Graphics Processing Unit | 12 |
| HAGI | Hardware Accelerated Geometry Instancing | 19 |
| HHD | Hand Held Device | 7 |
| IFC | Industry Foundation Classes | 13 |
| JSON | JavaScript Object Notation | 15 |
| LBD-CG | Linked Building Data Community Group | 6 |
| LDBIM | Linked Data BIM | 6 |
| LOD | Level of Detail | 11 |
| MEP | Mechanical, Electrical and Plumbing | 12 |
| NOHC | Near Optimal Hierarchical Culling | 12 |
| OC | Occlusion Culling | 19 |
| OGC | Open Geospatial Consortium | 17 |
| OMG | Object Management Group | 16 |
| OWL | Web Ontology Language | 10 |
| RAM | Random Access Memory | 12 |
| RDF | Resource Description Framework | 8 |
| RDFS | Resource Description Framework Schema | 10 |
| SDK | Software Development Kit | 34 |

| | | |
|--------|--|----|
| SPARQL | SPARQL Protocol and RDF Query Language | 11 |
| SQL | Structured Query Language | 11 |
| UI | User Interface | 15 |
| URI | Uniform Resource Identifier | 9 |
| VFC | View Frustum Culling | 19 |
| VRAM | Video Random Access Memory | 12 |
| W3C | World Wide Web Consortium | 8 |
| WKT | Well-Known Text | 18 |
| XML | Extensible Markup Language | 17 |
| XSD | XML Schema Definition | 17 |

References

- Bittner, J., Wimmer, M., Piringer, H., & Purgathofer, W. (2004). Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23, 615–624. <https://doi.org/10.1111/J.1467-86>
- Bonduel, M., Wagner, A., Pauwels, P., Vergauwen, M., & Klein, R. (2019). Including widespread geometry formats in semantic graphs using rdf literals. <http://lib.ugent.be/catalog/pug01:8633665>
- Cohen-Or, D., Chrysanthou, Y., Silva, C., & Durand, F. (2003). A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9, 412–431. <https://doi.org/10.1109/TVCG.2003.1207447>
- Ilozor, B. D., & Kelly, D. J. (2012). Building information modeling and integrated project delivery in the commercial construction industry: A conceptual study. *Journal of Engineering, Project, and Production Management*, 2, 23–36. <https://doi.org/10.32738/JEPPM.201201.0004>
- Johansson, M., & Roupé, M. (2009). Efficient real-time rendering of building information models. <https://www.researchgate.net/publication/220758081>
- Johansson, M., Roupé, M., & Bosch-Sijtsema, P. (2015). Real-time visualization of building information models (bim). *Automation in Construction*, 54, 69–82. <https://doi.org/10.1016/j.autcon.2015.03.018>
- Mattausch, O., Bittner, J., & Wimmer, M. (2008a). Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum*, 27, 221–230. <https://www.academia.edu/14388994>
- Mattausch, O., Bittner, J., & Wimmer, M. (2008b). Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum*, 27, 221–230. <https://www.academia.edu/14388994>
- Perry, M., & Herring, J. (2012). *Ogc geosparql - a geographic query language for rdf data* (OGC Implementation Standard 11-052r4). Open Geospatial Consortium. <http://www.opengis.net/doc/IS/geosparql/1.0>
- Rasmussen, M. H., Lefrançois, M., Schneider, G., & Pauwels, P. (2020). Bot: The building topology ontology of the w3c linked building data group. *Semantic Web*, 11, 1–20. <https://www.researchgate.net/publication/342802332>

- Wagner, A., Bonduel, M., Pauwels, P., & Uwe, R. (2019). Relating geometry descriptions to its derivatives on the web. *Proceedings of the 2019 European Conference for Computing in Construction*, 304–313. <http://dx.doi.org/10.35490/ec3.2019.146>
- Werbrouck, J. (2018). *Linking data : Semantic enrichment of the existing building geometry*. <http://lib.ugent.be/catalog/rug01:002494740>

Referenced webistes

- Bonduel, M., Wagner, A., & Pauwels, P. (2020). *Fog: File ontology for geometry formats*. <https://mathib.github.io/fog-ontology/>
- Carrol, J. J., & Pan, J. Z. (2006). *Xml schema datatypes in rdf and owl*. <https://www.w3.org/TR/swbp-xsch-datatypes/#sec-xmils-dt>
- LBD-CG. (2022). *A list of ontologies related to linked building data*. <https://github.com/w3c-lbd-cg/ontologies>
- Linietzky, J., Manzur, A., & the Godot community. (2023). *Portal — documentation de godot engine (stable) en français*. https://docs.godotengine.org/fr/stable/classes/class_portal.html
- OGC. (2023a). *Geosparql - a geographic query language for rdf data*. <https://www.ogc.org/standard/geosparql/>
- OGC. (2023b). *Open geospatial consortium*. <https://www.ogc.org/>
- Ontotext. (2022). *What is sparql?* <https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/>
- Rasmussen, M. H., & Schlachter, A. (n.d.). *Ld-bim - bim meets linked data*. <https://ld-bim.web.app/>
- Unity. (2023). *Manual: Memory in unity webgl*. <https://docs.unity3d.com/2023.2/Documentation/Manual/webgl-memory.html>
- W3C. (2015a). *Semantic web: Inference*. <https://www.w3.org/standards/semanticweb/inference>
- W3C. (2015b). *Semantic web: Linked data*. <https://www.w3.org/standards/semanticweb/data>
- W3C. (2015c). *Semantic web: Query*. <https://www.w3.org/standards/semanticweb/query>
- W3C. (2023). *Linked building data community group*. <https://www.w3.org/community/lbd/>