Linking Data: Semantic enrichment of the existing building geometry

Scan-to-graph plugin documentation

Jeroen Werbrouck

Supervisors: Prof. Pieter Pauwels, Willem Bekers
Counsellor: Mathias Bonduel

CONTEXT:
Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de ingenieurswetenschappen: architectuur

Department of Architecture and Urban Planning
Chair: Prof. dr. ir. Arnold Janssens
Faculty of Engineering and Architecture
Academic year 2017-2018

GHENT UNIVERSITY

This document contains the install information for the Scan-to-Graph plugin that was created in context of the Master's Dissertation "Linking Data: Semantic enrichment of the existing building geometry" at Ghent University by Jeroen Werbrouck, supervised by prof. Pieter Pauwels and Willem Bekers and counselled by Mathias Bonduel. The text of the dissertation is available at the Library of the Department of Architecture and Urban Planning at the Faculty of Engineering and Architecture (https://lib.ugent.be/). The plugin itself can be downloaded at the main GitHub repository of the dissertation (https://github.com/JWerbrouck/scan-to-graph). This document slightly differs from Chapter V in the dissertation text, since some improvements and changes were made after the text was submitted.

The author gives permission to make this document available for consultation and to copy parts for personal use.

In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

Content

Linking Data: Semantic enrichment of the existing building geometry

PLUGIN DOCUMENTATION

JEROEN WERBROUCK

Supervisors: Prof. Pieter Pauwels, Willem Bekers
Counsellor: Mathias Bonduel

CONTEXT:
Master's dissertation submitted in order to obtain the academic degree of
Master of Science in de ingenieurswetenschappen: architectuur

Faculty of Engineering and Architecture – Ghent University
Department of Architecture and Urban Planning
Chair: Prof. dr. ir. Arnold Janssens
Academic year 2017-2018

Dissertation Abstract

Building Information Modelling (BIM) has changed the way buildings are conceived, planned and executed. Apart from its frequent use for as-planned buildings, BIM has now been used for some years as a useful tool for digitizing existing buildings as well. mostly by performing a 'scan-to-BIM'. However, some inherent characteristics of existing buildings are complicating such a modelling process: uncertainties, imperfections in the built object and an interdisciplinarity that may go beyond the traditional Architecture, Engineering and Construction (AEC) topics. In this thesis, a method is proposed to integrate the concepts of scan-to-BIM into a Linked Data context, which could provide some answers to these complications. Current modular Semantic Web ontologies (BOT, PRODUCT, GeoSPARQL) are combined and extended to provide an alternative, minimalist framework for creating a semantically enrichened building model of existing buildings, in the form of an RDF graph. This 'scan-to-Graph' framework combines building topology, geospatial data and geometry with product classes, source linking and the documentation of assumptions, with a focus on point cloud sources. An example plugin for Rhinoceros 6 has been developed to link such semantic information to capricious geometries based on real-world objects. Finally, as a proof of concept, the process is demonstrated in a case study of the Presence Chamber of the Gravensteen castle in Ghent. With point clouds as a basis, a geometric 3D model of this room was created, semantically enriched and serialized to an RDF graph, including geometry.

Keywords: Linked Data, existing buildings, scan-to-BIM, scan-to-graph, remote sensing

A – INSTALLATION

This section contains, in bullet points, the information on how the plugin is to be installed and on which other applications need it depends.

1 Dependencies

Rhino 6 (https://www.rhino3d.com/download)

- The built-in version of IronPython should work; otherwise: the plugin was made using IronPython 2.7.8 (2.7.8.0) (https://github.com/IronLanguages/ironpython2/releases)
- The plugin will not work on Rhino 5 as the Eto GUI framework is not implemented there
- The E57 FILE IMPORT plugin by Dale Fugier should be installed to allow importing *.e57 point clouds (http://www.food4rhino.com/app/e57-file-import)

Python 2.7

- (https://www.python.org/downloads/)
- python release: 2.7.1
- extra packages: rdflib ('pip install rdflib') (https://github.com/RDFLib/rdflib)
- default folder set in the plugin : C:/Python27/python.exe
  (If installed elsewhere; the folder can be changed in the main application file ("scan-to-Graph_plugin.py") at line 21)

CloudCompare (v2.8.1 Stereo)

- (http://www.danielgm.net/cc/)
- default folder set in the plugin :
  "C:\Program Files\CloudCompareStereo\CloudCompare.exe"
  (If installed elsewhere If installed elsewhere; the folder can be changed in the main .py file ("scan-to-Graph_plugin") at line 24)

Stardog (stardog-5.2.3)

- https://www.stardog.com/
- information on how to set up a database https://www.stardog.com/docs/

  stardog/bin needs to be set to the PATH, otherwise : change in the main .py file ("scan-to-Graph_plugin") at line 1877 and 1887 "stardog.bat" to the full installation path (e.g. "C:\data\stardog-5.2.3\bin\stardog.bat")
- for SPARQL queries that do not yield geometric results, it is recommended to use the Stardog Admin Web Console, which is an optimized environment

2 Installation

- The plugin is not a .rhi installation folder, so another installation method is required
- Download the plugin installation folder and place it at the preferred location (e.g.: C:\Users*username*\AppData\Roaming\McNeel\Rhinoceros\6.0\scripts, other installation folders are also allowed)
- In Rhino, right click on the toolbar > "N*ew Button…"*
  - Choose the appearance (e.g. *Text only* => STG)
  - Tooltip: e.g. "Scan-to-Graph plugin"
  - Command: !-RunPythonScript "Installation_folder\Scan-to-Graph_Plugin.py"

!-RunPythonScript "C:\Users*username*\AppData\Roaming\McNeel\Rhinoceros\6.0\scripts\STG_plugin\Scan-to-Graph_Plugin.py"

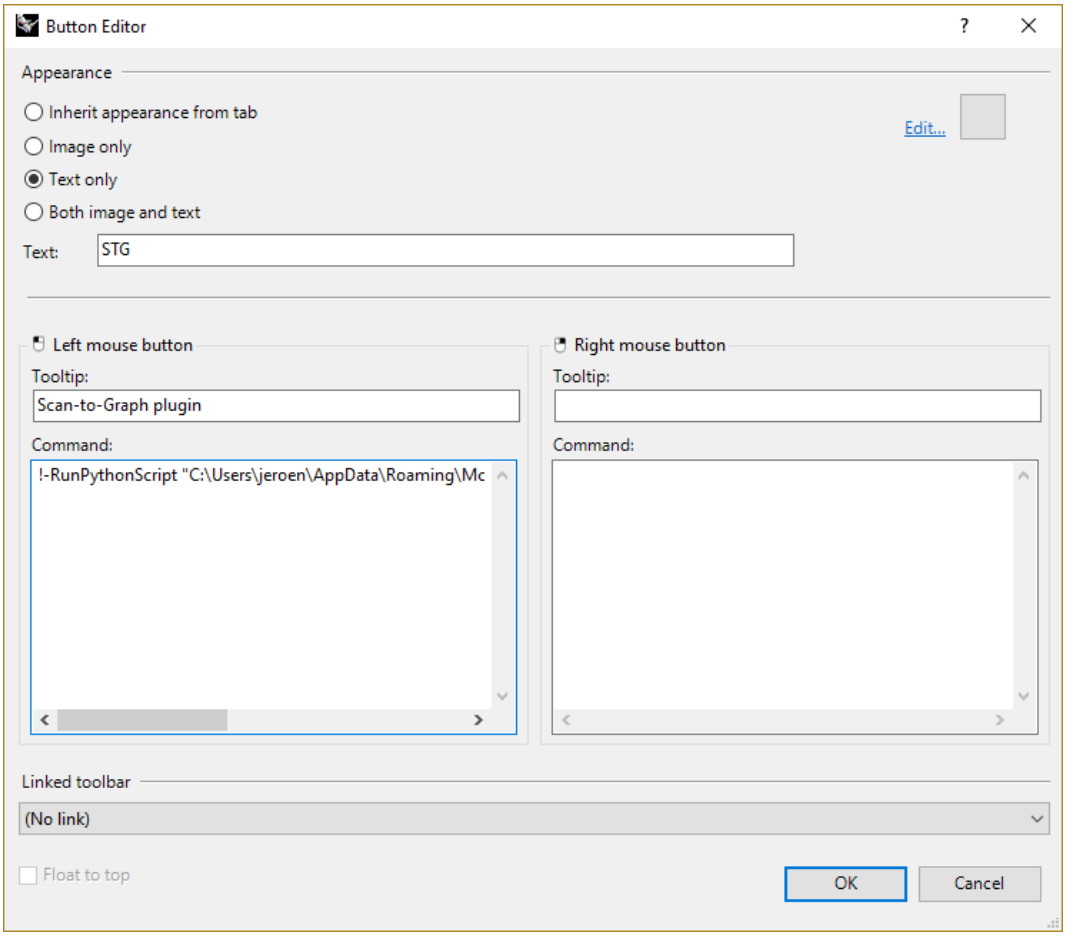

Fig. 1: Setting the shortcut button to start the plugin

The plugin should now be called when clicking this button. If this does not work for some reason, the file can be called manually by opening it with the "EditPythonScript" command and executing the script there. Feel free to contact me when problems arise.

B - FUNCTIONALITY

1 General information

The plugin allows to define a building site, building, storeys and spaces, which contain objects or are adjacent to them, **in a Linked Data context** (using the Resource Description Framework RDF). It was constructed in context of a master thesis at Ghent University ("Linking Data: Semantic enrichment of the existing building geometry")[1]. Objects are primary characterized by a layer that contains its representations, such as the point clouds or the reconstructed 3D geometry. It is possible to further specify sub-objects with a main object (which has a layer) or another sub-object as a parent. URIs are constructed primary using the name of the instance in Rhino. In the current version of the plugin, objects and storeys should have a unique name to avoid errors in the graph. The URI of a space is based on its own name and the name of the storey it belongs to, so different storeys can have a space with the same name. Sub-objects only need to have a unique name within the 'range' of their parent object but may share their name with a sub-object from another parent object, e.g. two columns can each have a sub-object 'Capital'.

At the top of the plugin, the name of the graph is to be defined (Fig. 2). The folder is set by clicking the [...] button. **It should be located on the C-Disk to work**. The graph will be serialized in Turtle. The [...] button also allows to load previously defined graphs (.ttl) (in test phase – may not load everything). Graphs that are loaded will be serialized again, **possibly overwriting the original file**. After a graph is loaded, a new name can be set in the textbox or using the [...] button. The textbox below serves for stating an overall URI, which serves as a basis for giving all instances their own URI (it will be used as a prefix 'inst:' in the Turtle files). Another option to save unfinished work is to save the created information in a .txt file, which contains the python dictionaries and lists that are used in the plugin itself, so this does not relate to RDF or linked data in general.

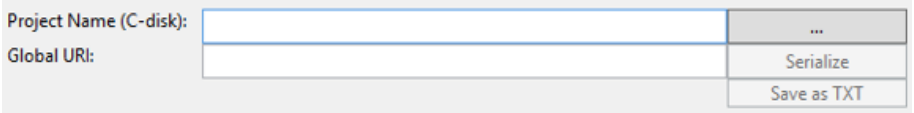

Fig. 2: General information settings at the top of the plugin

2 Project Info Tab

In this tab (Fig. 3), the overall topology of the building is laid out, by use of bot:Zone data instances and a coordinate system. Although a bot:Site can contain multiple bot:Building elements in theory, in the plugin this is currently restricted to only one. If it would nevertheless be necessary to define multiple buildings sharing one site, this is possible by either storing them in the same database or merging the graphs together outside the plugin environment. Building storeys and their spaces of the building are also part of the building topology and are defined at the bottom of the tab.

Lambert72 is currently the default Geographic Coordinate System, as it was used in the master thesis. Using another GCS is possible by copying its openGIS URI (hyperlink by "Coordinate

[1] Jeroen Werbrouck – supervisors: Prof. Pieter Pauwels, Willem Bekers, Mathias Bonduel

System") into the coordinate system widget. Georeferenced point clouds that are imported using the plugin will be translated to the origin using the project coordinates (if they have been set); large geocoordinates often result in the project's canvas becoming too big and unfit for modelling (see further). Note that this only works with cartesian coordinates, since the Rhino canvas uses a cartesian system. In the current plugin version, it is only possible to include only one GCS, although it is in theory possible to add multiple.

The Project Info Tab also includes a checkbox for including STEP representations in the graph as Literal strings. As a test option (**unstable**), it is possible to load the STEP files that are included in an RDF graph. This can be done by checking the checkbox "import STEP geometry …" and parsing an RDF graph **in an empty Rhino Document** (.3dm). This function is currently not compatible with the rest of the plugin, as changing the old Rhino IDs into the new Rhino IDs (from the imported geometry) is not fully supported yet. Results may vary and depend on the validity of the geometries being imported ("bad objects" and larger files increase the possibility of getting problems). So currently, this importing option can serve for visualizing, but not for graph adaptation.

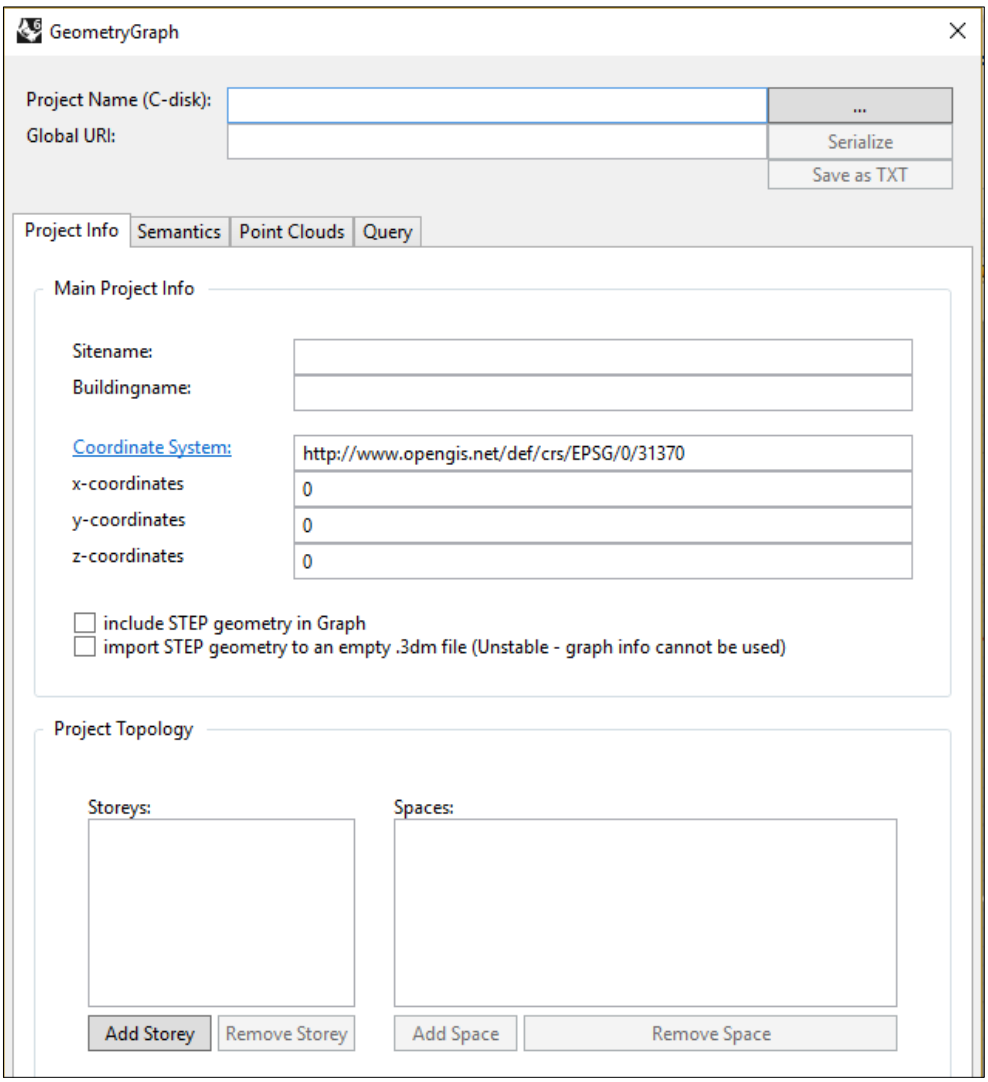

Fig. 3: Plugin tab to define the topology and geographic location of the building (additionally: the option to include geometric information directly in the graph)

4 Point Clouds Tab

When the basic project topology is defined, one can start modelling the as-is geometry based on point clouds (or other sources). The plugin streamlines this importing process for as-is modelling by several steps (Fig. 4). First of all, the large file format of huge detailed point clouds can give performance problems. Therefore, subsampling the point clouds by calling **CloudCompare** before importing them is given as an option. When using the subsampling option, the user locates a folder **which contains only the files that should be subsampled** (when it contains other files or folders, it is not sure whether the import will happen without problems, results may vary). The subsampled point clouds are then stored in a separate subfolder and imported. Subsampling options include: octree, spatial and random, as performed by CloudCompare. Each point cloud is imported as a separate layer, which takes the name of the point cloud and serves as the representation of the 'semantic object'. Apart from the point cloud, the layer also contains the modelled representation of the element. If well-chosen, the name of the point cloud can give an indication for later object classification: i.e. if it contains a Product Ontology class (e.g. column-COLUMN_Column1_PresenceChamber.e57 will give a classification suggestion for product:Column-COLUMN).

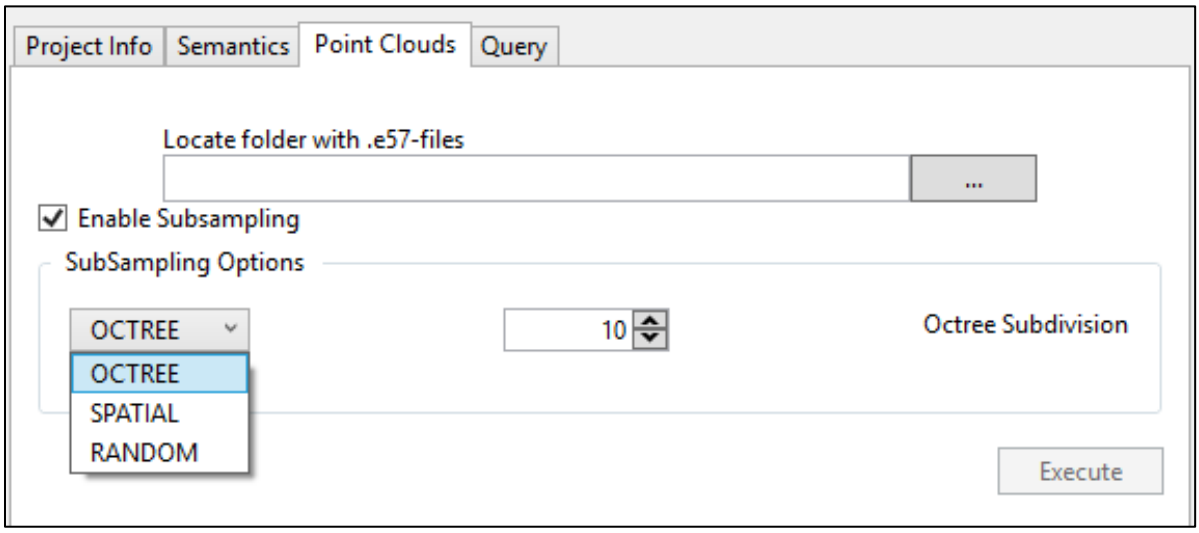

Fig. 4: Plugin tab for importing and subsampling of point clouds

5 Semantics Tab

This tab contains the core of the plugin (Fig. 5). The 'Main Objects', which may have a point cloud representation, refer to a layer of the document (and take its name as the element's name). An object is located in a space, which is part of a storey, both defined in the 'Project Info' tab. The default relationship between a zone and an object is 'bot:containsElement'. The checkbox 'Adjacent' provides the option to override this default relationship and set it as 'bot:adjacentElement'. In this case, it is possible to define the zone on the other side, in the plugin limited to a space. A 'Main Object' may be hosted by another element (bot:hostsElement), e.g. a door can be hosted by a wall. As indicated in the section about the Point Cloud Tab, the element's type is guessed from the name of the point cloud (strictly spoken, from the name of the Layer), but can be changed without consequences. A widget to link the geometries to the represented LoA after a geometric deviation analysis is provided. Although this widget is currently mapped to an entire object in the User Interface, in the graph, this LoA will be linked to each individual geometry, except those that are denoted as occlusions.

An object may have sub-objects, either hosted or aggregated. Apart from this parent-child relationship, a sub-object has a name and a type. An option is provided to set a sub-object as an aggregate/hosted element of another sub-object, hereby enabling the core Linked Data property of virtually unlimited detail. Note that, from a certain point, the object types will have to be custom-defined, since they will not be included in the Building Product Ontology anymore. In the current version, this can only manually be done by adding the URIs of these objects to the .csv-file 'custom_types.csv', located at the plugin's installation folder. As this is not fully supported yet, please use the stgp URI that is already present in the .csv file, so it will be mapped to the graph.

Each object intrinsically aggregates 3D geometry that its corresponding layer contains. This is denoted by the item 'self' in the 'sub-objects' list. These geometries are visible in a list below the sub-object's properties ("Sub-Object has Geometries:"). When such an element-ID is selected in this list, the corresponding object will also be selected in the viewer. Further on, selecting an element makes it possible to add one or more written notes to this element in the form of a modelling remark (a string linked to stg:ModellingRemark) or an occlusion (stg:OccludedGeometry), in the plugin both listed in the lowermost list of the Tab. When 'self' is selected in the sub-object list, the geometry-list also contains the point clouds that are stored in the layer, which are, unlike 3D objects, not serialized as a geo:Geometry, but as stg:PointCloudFile. When selecting another sub-object, there is an option to pick the 3D objects (surfaces or polysurfaces/volumes) in the viewer that assemble this sub-object (restricted to the geometries in the Object Layer and Default layer), again displayed in the listbox.

To check if the information has been stored correctly, a button at the bottom of the tab is provided to print all information that is currently stored on the command line, in a schematic way. This includes the Project Topology, geolocation, the assumptions that are mapped to object IDs and all Main Object attributes ('point cloud', 'type', hosted or not …), their sub-objects and the attributes of these sub-objects.

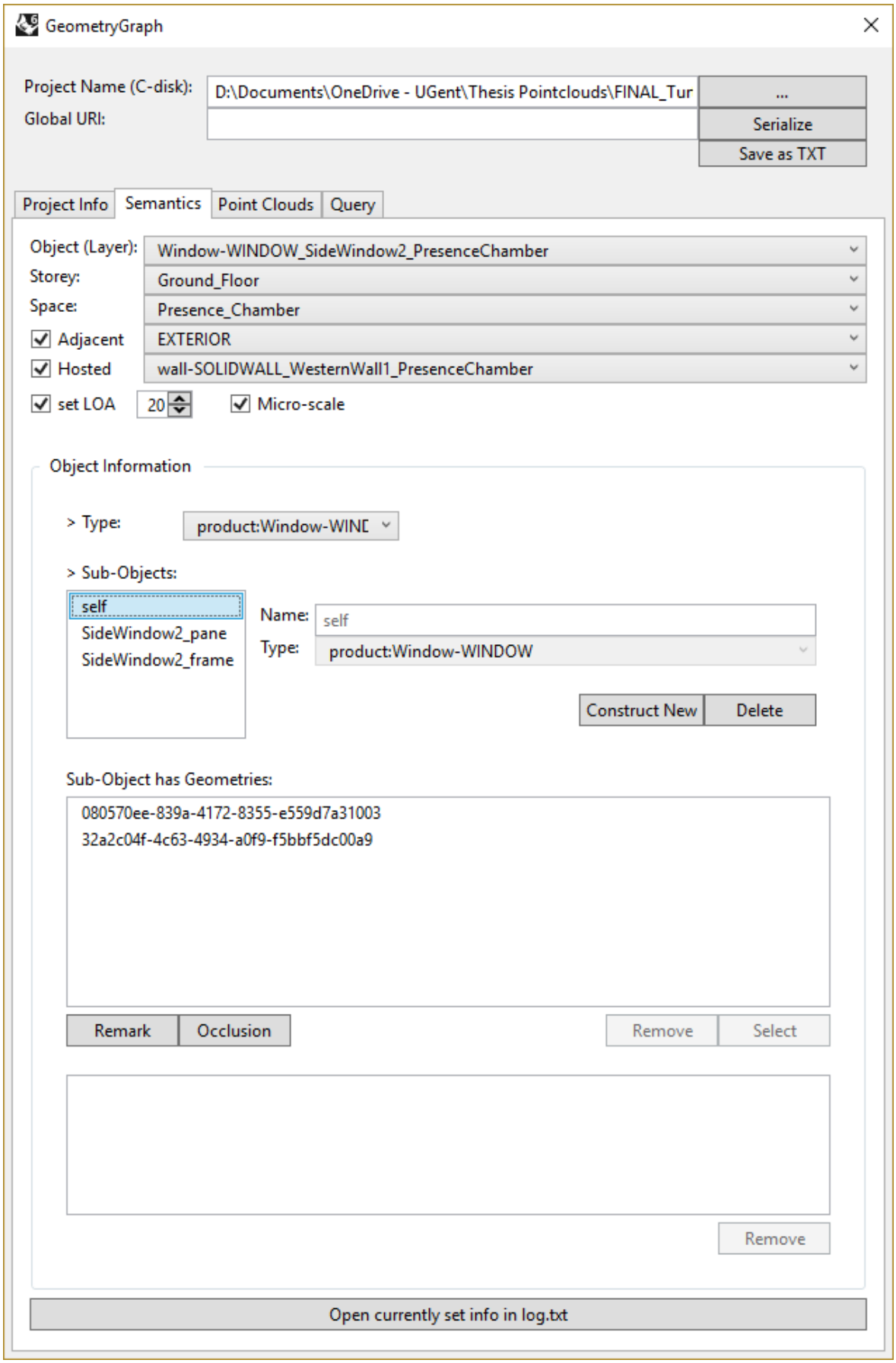

Fig. 5: Plugin tab for assigning object attributes, sub-objects and geometries, as well as labeling 3D objects with modelling remarks

6 SPARQL query tab

The last tab (Fig. 8) is a tab for performing SPARQL queries and visualizing geometry that matches the query. The graph should be loaded in an active Stardog Database before running a query. To enable reasoning, the corresponding ontologies should also be loaded into this Stardog database.[2] This can be done by going to the local server (default at *localhost:5820,* username: *admin*, password: *admin*), opening *Database* > *Browse* > *Data* > *+Add.* A local graph file (ttl; rdf; owl …) containing the ontology definitions can then be uploaded. To prevent that all graphs in the database are also queried when inferencing is not enabled, the ontologies can be added as 'named graph'. In our case, this will be done for BOT, Product, GeoSPARQL and the custom ontology STG.

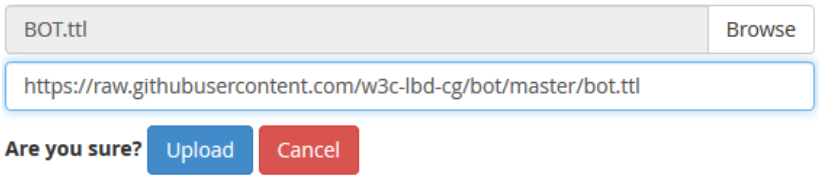

Fig. 6: uploading ontologies as named graphs

The Query tab itself contains an input box where the name of the Stardog database should be stated. Below, there is the SPARQL query input box[3], with the query button and a function to enable/disable reasoning.[4] Results are displayed at the table "Query Results", each variable a column. When the query results contain Rhino IDs, the overall display mode changes to 'Wireframe', the objects related to the IDs in the results are individually set to a 'Rendered' mode, which makes them clearly visible in the viewport (Fig. 7). Selecting a row in the results that contains a Rhino ID will also select the corresponding item in the viewport. For SPARQL queries that do not yield geometric results, it is recommended to use the Stardog Admin Web Console, which is an optimized environment

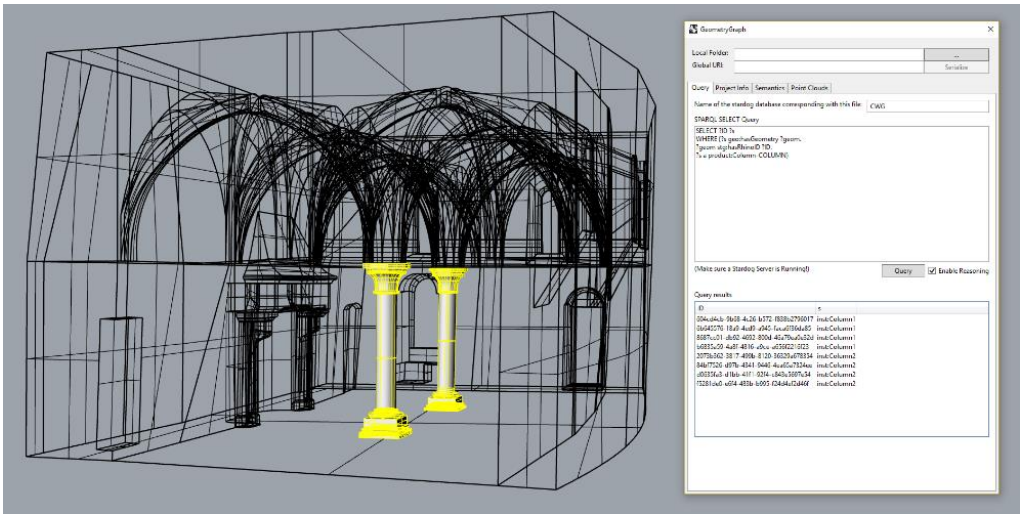

Fig.7: highlighting and selecting objects via a SPARQL query

[2] […]>stardog-admin db create -o spatial.enabled=true -n "DB-name" "Path-to-DB"

[3] Apparently, when querying a Stardog Database via the command line with "SELECT ?s ?p ?o WHERE {?s ?p ?o}", Literals with a certain length (e.g. the Literal strings containing STEP geometry) are not interpreted correctly and are split at random places. Results may vary.

[4] Make sure, when entering a query in the plugin interface, that a space is added between the command (SELECT, INSERT) and WHERE, also when working with multiple lines.

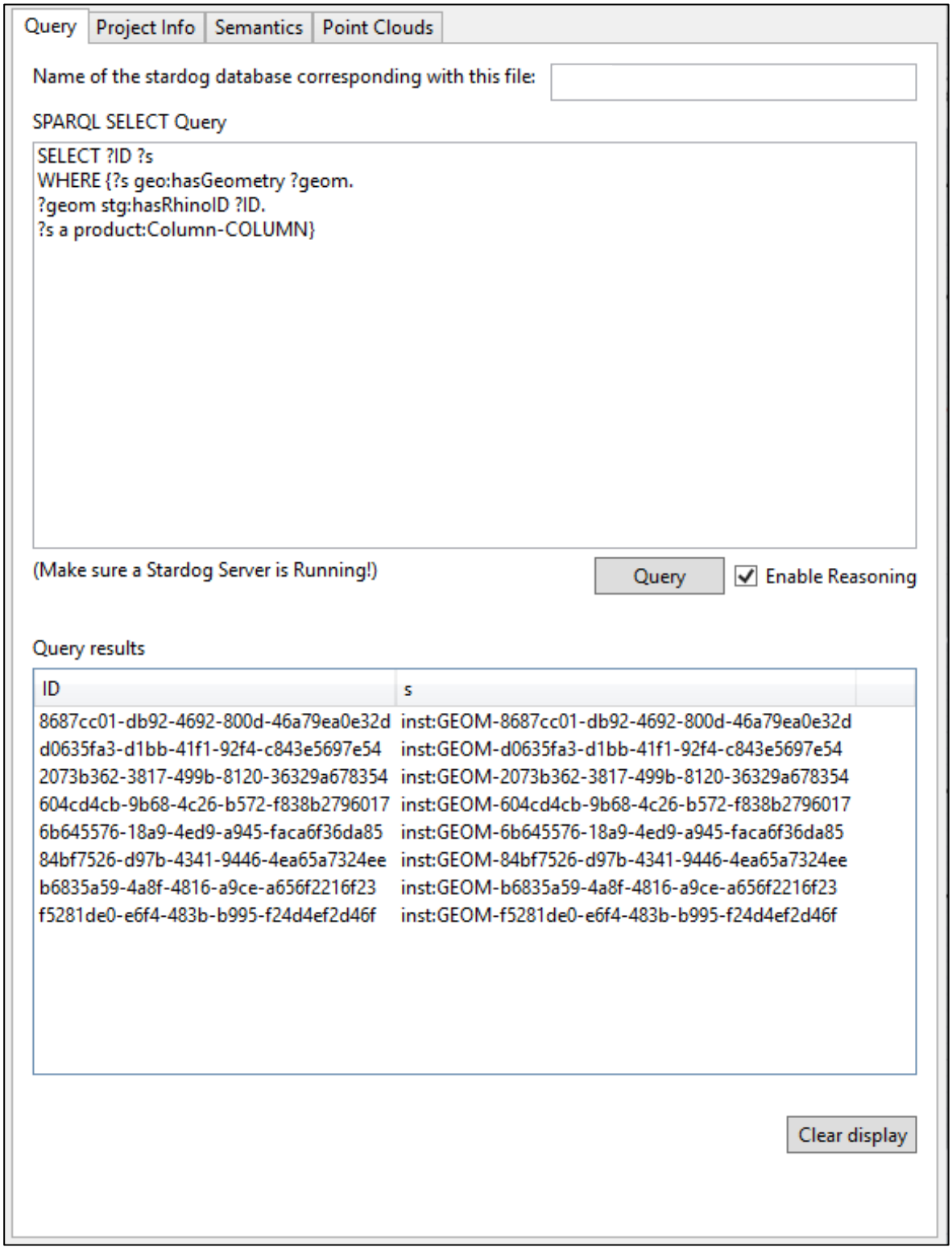

Fig. 8: Plugin tab for querying with SPARQL; visualization of the results as a table and in the active viewport.

7 Additional content

The Plugin provides a UI for constructing graphs with the structure outlined in Chapter III. However, it is probable that some additional content needs to be added to the initial graph. The plugin does not provide a UI to add 'additional content', since this is too generic: a UI that uses Linked Data but does not expect Linked Data expertise from its user should focus on documenting only a part of the information, related with specific ontologies. This UI thus deals with providing the basic topology and geometry of an existing building. A future research project could focus on the development of an interface to implement historical information (e.g. using the structure of E-CRM and CHML as a basis). When a UI for the type of information that has to be included in the graph (historical, technical, geographical …) is yet to be developed, this issue can be solved by performing SPARQL INSERT queries, stating the information in SPARQL. Stardog (or another RDF store) can be used for this. For example, if someone wants to add some historical information about the purpose of the Presence Chamber in the Gravensteen during the Middle Ages (on the Case Study in the Master Dissertation). The integration of such information in the graph (using E-CRM) takes the following steps:

E-CRM has a property 'P103_was_intended_for' (domain: E71_Man-Made_Thing; range: E55_Type)[5]. The subject of the relationship will be:
'inst:Ground_Floor_Presence_Chamber', which is already present in the graph as a bot:Space, but is hereby also classified as a 'E71_Man-Made_Thing'. The object will have to be defined newly, as 'inst:giving_audience', now implicitly classified as an E-CRM 'E55_Type'. The INSERT query will thus be as follows:

```
PREFIX ecrm: <http://erlangen-crm.org/140617/>
INSERT {inst:Ground_Floor_Presence_Chamber ecrm:P103_was_intended_for
        inst:giving_audience}
WHERE {}
```

The prefix is set because the E-CRM namespace is not yet included as a namespace from the database (which can be changed in the main settings). The WHERE brackets can stay empty, since there are no variables assigned here (it would be different when we would want to say that all bot:Spaces had the purpose of giving audience). After adding ecrm as a database Prefix (in the Stardog Web Console), the success of above INSERT can be controlled by performing a quick SPARQL ASK query:

```
ASK WHERE {?s ecrm:P103_was_intended_for
inst:giving_audience}
```

Which returns 'true': such a triple is present in the database. Note that to enable reasoning, the ontology has to be loaded into the Stardog Database. In the case of larger ontologies, such as E-CRM or CHML, this can slow down querying.

[5] http://erlangen-crm.org/docs/ecrm/current/index.html#anchor-2008326450 (accessed 13/05/2018)

C – General Remarks

- The folder of the graphs that are created with the plugin should be located at the C: disk;

- If the installation folders of the applications on which the plugin depends are not the default installation folders, please change them in the main plugin .py file as specified at page 4 (A.1 Dependencies);

- Parsing a previously created .ttl graph into the plugin may vary qua results. If your work is unfinished, information loss is avoided by saving to .txt and loading it later. The extension needs to be changed manually (.txt => .ttl), either in the textbox or by setting a new name/folder with the [...] button. The default extension at the [...] button is Turtle.

- The plugin does not identify identic names for objects, storeys or spaces. Caution is required; since setting identic names may cause a crash.

- The layer names should not contain spaces, as these will be used for the object URIs in the graph. Spaces are automatically converted to underscores when serializing, but loading a graph may cause some problems (e.g. the object is saved as “Column_2” but the layer’s name is “Column 2”. An error will occur and Rhino will crash.

- Requirements before serializing:
    - There should be a specified project name and folder;
    - There should be a specified global URI
    - The opened Rhino (.3dm) file should have a filename

D – Acknowledgements

The construction of the Plugin would not have been possible without the general support of the thesis by Prof. Pieter Pauwels, Willem Bekers (UGent) and Mathias Bonduel (KU Leuven), who gave me valuable feedback during the course of the thesis and on the plugin development. Also many thanks to Prof. Jakob Beetz of RWTH Aachen, who taught me the basics of Linked Data and coding in Python at the course "M1 Project: Architekturinformatik" during my Erasmus exchange to RWTH Aachen.