Change toolkit for digital building permit

Deliverable number	D3.3
Deliverable name	BIM to Geo conversion tool and procedure
Work package number	WP3 GeoBIM
Deliverable leader	Delft University of Technology
Dissemination Level	Public

Status	Final
Version Number	V1.0
Due date	M25, submitted M36 due to embargo
Submission date	30/09/2025

Project no.	101058559
Start date of project:	1 October 2022
Duration:	36 months
File name:	CHEK_101058559_D3.3-BIM to Geo conversion tool and procedure_V1.0_Final

Funded by the European Union

This project has received funding from the European Union under the Horizon Europe Research & Innovation Programme 2021-2027 (grant agreement no. 101058559).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Authors and contributors

Author	Organisation	E-mail
Jasper van der Vaart	TUD	J.A.J.vanderVaart@tudelft.nl
Ken Arroyo Ohori	TUD	K.Ohori@tudelft.nl
Jantien Stoter	TUD	j.e.stoter@tudelft.nl

Quality control

Author	Organisation	Role	Date
Peter Bonsma	RDF	WP leader	30/10/2024
Mayte Toscano	OGC	Reviewer	21/10/2024

Document history

Release	Description	Date	Author
V0.1	First Draft	03/10/2024	Jasper van der Vaart
V0.2	Second Draft	23/10/2024	Jasper van der Vaart
V1.0	Final version	29/10/2024	Jasper van der Vaart, Ken Arroyo Ohori, Jantien Stoter

Contents

1. Executive Summary

An automated solution for the conversion of a BIM model to a GIS building representation would make evaluations at a city scale easily accessible for both the design and DBP process. This could improve the structures' integration into the cityscape. This conversion would not only allow for visual evaluations but potentially unlock new evaluation methods as well, e.g. applied wind or local climate simulations. Finally, the GIS building representations from the BIM source can be used to update or enrich the existing GIS city models, improving datasets containing building representations based on airborne LiDAR data.

D3.3: BIM to Geo converter focusses on automated standard-compliant conversion from BIM models to GIS models. The automation of this conversion comes with a set of challenges. This is primarily due to the different structure and functionality of BIM and GIS models. In the past, there have been attempts to develop automated solutions to convert BIM to GIS, but these solutions had their own issues and limitations, preventing them from being widespread adapted in practice. Due to this, BIM is very rarely used as a GIS data source.

To improve the automated integration of BIM and GIS models and the enrichment of GIS models from BIM models, the IfcEnvelopeExtractor has been developed. This is a software application that automatically converts IFC files to format-compliant CityJSON files that can be used for further analysis. The tool can abstract and reconstruct the models according to the extended LoD standard presented by the TU Delft in 2016. It utilizes a combination of point cloud, voxel and ray casting processing to create the abstracted shapes. For exterior export, the available abstractions are LoD0.0, 0.2, 0.3, 1.0, 1.2, 1.3, 2.2, and 3.2. For interior, they are LoD0.2, 1.2, 2.2, and 3.2. Additionally, the tool can also output a voxelised shape.

The performance of the IfcEnvelopeExtractor is generally good when it was tested on the four CHEK IFC models that were created within the consortium. It can alleviate the user from lengthy and complex manual steps that would otherwise be required for a BIM to GIS conversion. However, due to its dependency on high quality IFC input models, it is recommended to validate the output of the tool manually to avoid errors in downstream processing.

This deliverable was finalised and reviewed according to the timeline in the DoA (to be ready by M25 – 31 October 2024). However, since a Journal paper was planned to be written based on this deliverable, the submission of the deliverable (with dissemination level 'Public') was postponed. Since October 2024, the methodology has been further developed, also based on further experiences in CHEK. A full report describing the final methodology is available: https://research.tudelft.nl/en/publications/bim2geo-converter

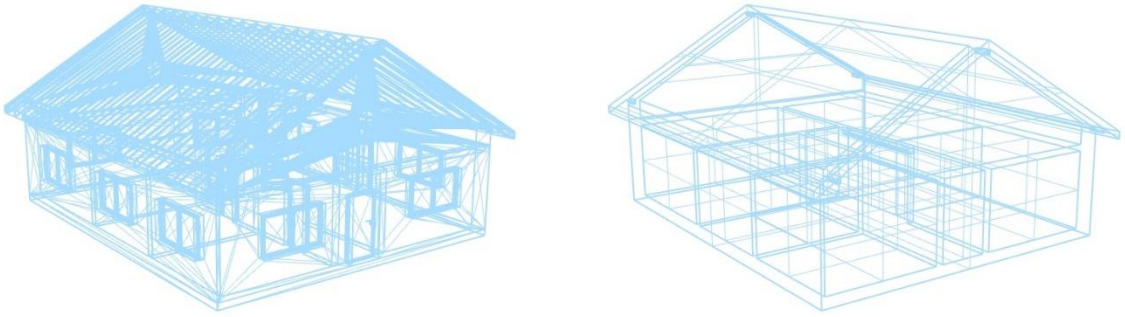
A preprint of the Journal paper based on the work is published on Preprints.org (ID 178274). The Journal paper is currently under review https://www.preprints.org/manuscript/202509.2284/v1.

2. Introduction

The aim of WP3 is to improve the integration of GIS and Building information modelling (BIM) models to support streamlining the DBP process. The deliverable D3.3: BIM to Geo converter (this one) focusses on automated standard-compliant conversion from BIM models to GIS models.

An automated solution for the conversion of a BIM model to a GIS building representation would make evaluations at a city scale easily accessible for both the design and DBP process. This could improve the structures' integration into the cityscape. This conversion would not only allow for visual evaluations but potentially unlock new evaluation methods as well, e.g. applied wind or local climate simulations. Finally, the GIS building representations that are obtained from the BIM source can be used to update or enrich existing GIS city models, improving datasets containing building representations based on airborne LiDAR data.

The automation of this conversion comes with a set of challenges. This is primarily due to the different structure and functionality of BIM and GIS models, see Figure 1. A BIM model is constructed out of many geometric elements representing different features that together construct the overall shape of a building. For example, separate walls, floors, doors, windows, beams, and columns are combined to construct a single building model. In contrast, a representation of a building in a GIS file is in general (only) constructed out of different shells. A shell can be seen as the boundary between the air and the tangible geometry of the building. The exterior of the building is usually modelled as a single shell, regardless of how many geometric objects its shape is constructed of. Each space within the building is modelled as its own unique shell as well, again, regardless of how many objects it is constructed of.



(a) (b)

Figure 1: Wireframe building representation in a BIM (a) and in a GIS (b) format

In the past, there have been attempts to develop automated solutions to convert BIM to GIS, but these solutions had their own issues and limitations preventing them from being widespread adapted in practice. Due to this, if in practice a BIM source is used to create a GIS building representation, this is done either automatically by one-to-one translating the BIM model or by recreating the model manually in 3D. Processing the file by hand is a labour-intensive process that does not scale well to larger building clusters or complex buildings. GIS models created by one-to-one translating a BIM model to a GIS format result in GIS models in name only. These GIS files might be visible when opened in GIS software, but they do not utilize the data models that are assumed when these formats are used, and they would not be usable for a significant amount of GIS related analyses, applications and processes.

Because of these challenges, BIM is rarely used as a GIS data source. Instead, the established primary source for GIS models is currently in-field measurements. This data source does come with its own issues. Acquiring these measurements is a cumbersome and expensive process that cannot be executed for a building that has yet to be realized. This means that it cannot be applied to the design of a (not yet existing) building or for the DBP process. Additionally, these measurements are often taken from aerial measuring devices, such as airborne LiDAR scanners. Unlike when a BIM source is used, these airborne LiDAR scanners will only properly record the building features that are visible from the sky. Thus, important features that are well documented in BIM models, such as a building's overhang, underground structures and interiors, are often ignored because they are not recorded via LiDAR.

To improve the automated integration of BIM and GIS models and the enrichment of GIS models, the IfcEnvelopeExtractor has been developed.

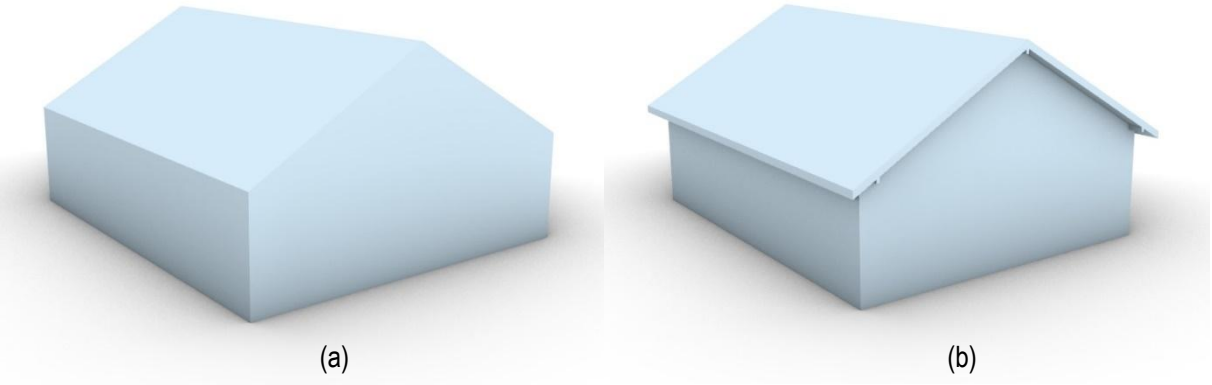


Figure 2: The quality of the GIS model that is possible to create from airborne LiDAR scanning (a) and from a BIM model conversion (b). It can be noted that the representation that has been created from LiDAR data (a) has no overhang included while the BIM sourced model (b) does.

3. The tool

The IfcEnvelopeExtractor is a cross platform (Windows & Linux) console application that accepts a BIM model as input and automatically converts it to a CityGML/CityJSON file with minimal user assistance. The application is lightweight and can be utilized on local systems and servers, depending on the requirements of the project.

The application utilizes the IfcOpenShell and OCCT libraries for accessing the BIM data and processing the geometry. The application is open source and freely available from: https://github.com/jaspervdv/IFC_BuildingEnvExtractor. The pre-built Windows executable can also be accessed from GitHub. In conjunction to the IfcEnvelopeExtractor the C++ library CJT was developed. CJT is a library that enables editing of CityGML/CityJSON files and the conversion of OCCT geometry to CityGML/CityJSON. This library is an integral part of the IfcEnvelopeExtractor but can also be used independent in other C++ applications. CJT is also open source and freely available from: https://github.com/jaspervdv/CJT.

3.1 Supported I/O file formats

The IfcEnvelopeExtractor application accepts BIM models in the open Industry Foundation Classes (IFC) format. However, only a subset of IFC versions is currently supported. The list of supported IFC formats can be seen in Table 1. The developed application also adheres to the georeferencing standards for IFC4. For IFC2x3 the property sets (Psets) approach developed for IfcGref (D3.2) are utilized. For IFC4x3 the same approach of IFC4 is applied.

Table 1 Supported IFC versions

Version	Name
2.3.0.1	IFC 2x3 TC1
4.0.2.1	IFC4 Add2 TC1
4.3.2.0	IFC4x3 ADD2

IfcEnvelopeExtractor outputs CityGML models in the CityJSON V2.0 encoding. Aside from a voxelised output it adheres geometrically to the extended Level of Detail (LoD) framework developed by the TUD in 2016, see Figure 3. The complete supported LoD output can be seen in Table 2.

Table 2 Supported Geometry LoD output

Level of Detail	Exterior	Interior	Semantics
LoD0.0	Yes	None	Standard
LoD0.2	Yes	Floors and rooms	Standard
LoD0.3	Yes	No	Standard
LoD1.0	Yes	None	Standard
LoD1.2	Yes	Rooms	Standard
LoD1.3	Yes	None	Standard
LoD2.2	Yes	Rooms	Standard
LoD3.2	Collection of surfaces	Rooms	None
LoDV (voxel)	Yes	Rooms	None

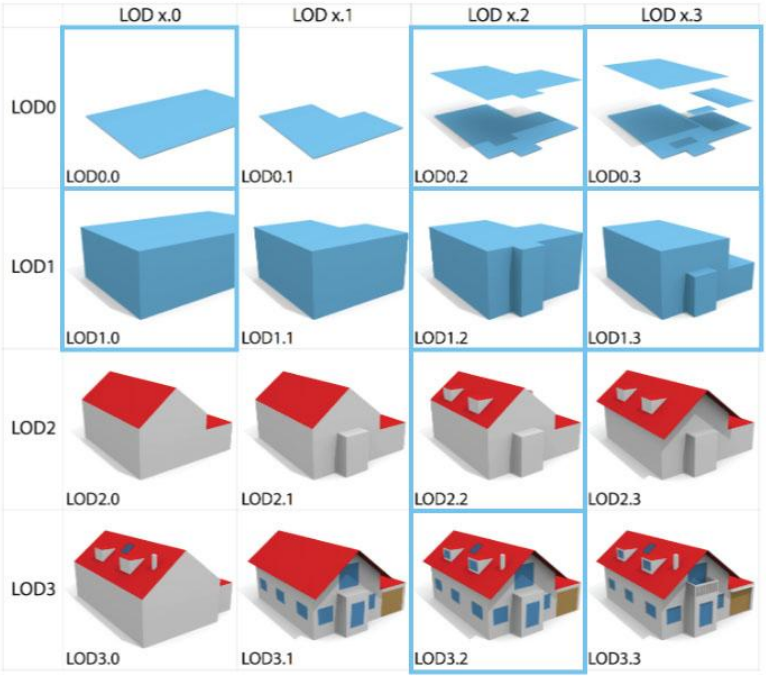


Figure 3: The LoD framework developed by the TU Delft in 2016 with the IfcEnvelopeExtractor supported exports highlighted in blue.

The flexible structure of CityJSON allows the output CityGML/CityJSON file to follow a specific structure. Every object that represents a building is split up into 2 building parts: The "Outer Shell" and the "Inner Shell". The “Outer Shell” represents the exterior of the building, the “Inner Shell” represents the interior of the building. The "Outer Shell" follows the regular structure that is often seen within established city models. It is populated with the different exterior LoD shells. The "Inner Shell" is, depending on the process settings and input file quality, split further over storeys and rooms. A complete overview of the structure can be seen in Figure 4.

```
CityJSON File
└── Building Object*
    ├── Outer Shell
    └── Inner Shell*
        └── Storey**
            └── Space/Room**
```

Figure 4: Structure of the output CityGML/CityJSON file. * Object that does not directly store any geometry. ** Depending on the quality of the input, the nature of the input and the user settings, these objects can occur many times.

3.2 The methodology

The processes that are executed by the IfcEnvelopeExtractor can be split into different parts. When the tool is used, it will always go through a preprocessing phase. This preprocessing is followed by the geometry processing phase. This second phase can be roughly split into four different subprocesses: Low-level, mid-level, high-level, and interior shell processing. Depending on the desired output some of these geometry related processes are executed while others might be partially or completely bypassed.

- Low-level processing:
  - Generates LoD0.0 and 1.0
  - Based on the BIM model's geometry point could
- Mid-level processing
  - Generates LoD0.2, 0.3, 1.2, 1.3 and 2.2
  - Based on the BIM model's roofing geometry
- High-level processing
  - Generates LoD3.2
  - Based on the BIM model's outer shell geometry
- Interior processing
  - Generates LoD0.2, 1.2, 2.2 and 3.2 interior spaces
  - Based on the BIM model's IfcSpace objects

These four subprocesses do not explicitly cover all the outputs of the tool. E.g. voxelised output is not mentioned in the second phase. This is because voxelisation is used internally as a geometry filter during both the mid-level processing and high-level processing. The voxelised shape is a useful byproduct, that comes with the creation of different shells, from which the outer surfaces are merged and exported in a later step.

3.2.1 Process 1: Prefiltering and simplification

During this first step, the application executes the filtering of objects that are assumed to construct the outer (or inner) shells. This filtering is coarse (and therefore called 'prefiltering') and based on the object's type. When using the default settings, every IfcWall, IfcCurtainWall, IfcWallStandardCase, IfcRoof, IfcSlab, IfcWindow, IfcColumn, IfcBeam, IfcDoor, IfcCovering, IfcMember and IfcPlate object is stored as a potential space dividing object. If interior spaces are to be converted, the IfcSpace objects are also stored, but these objects are not assumed to be part of the potential space dividing object list. The user has full control on which type of objects are used as potential space dividing object, so any IFC object type can be used or ignored if desired by the user. Spaces can however only be used as IfcSpace objects, no other IFC objects are allowed to be used.

The prefiltering process is based on the top-level object types, not their nested object types. Nested objects of any type are selected if their parent type is included in the filter list. For example, when IfcRoof types are utilized, the tool will also select IfcSlab objects if they are nested within the IfcRoof objects even if the IfcSlab type was not selected to be evaluated.

After the object type filtering the geometry of assumed complex objects for IfcDoor and IfcWindow objects is simplified. These objects have a lot of small details that are not relevant at a GIS scale and that can make the geometry complex,

e.g. detailed frames, hinges, and distinct panes. Because these objects are often box-like in nature, they can be approximated by replacing their geometry with an orientated smallest bounding box in full 3D space (X, Y, and Z rotation), see Figure 5. These objects are also often nested inside of IfcWall, IfcSlab or IfcRoof objects. If a complex object is placed in one of these nesting objects the void (i.e., Boolean set difference) is not applied to these nesting objects, eliminating a nested object at that location, see Figure 6 .

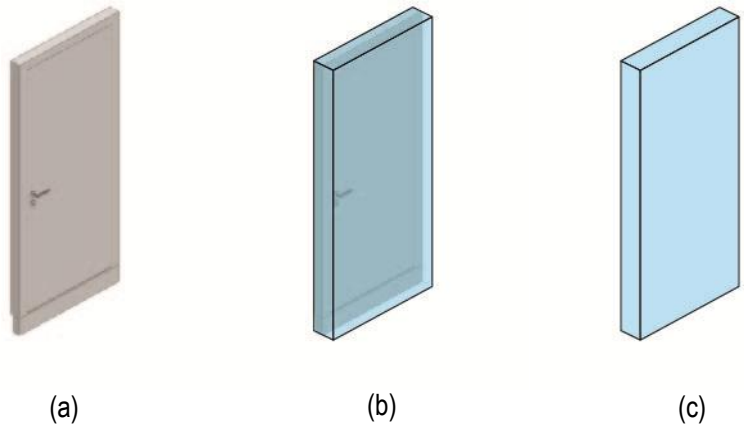


Figure 5: Example of the box simplification of an IfcDoor object. (a) The original IfcDoor geometry. (c) The simplified shape that has been approximated with the help of a bounding box. (b) Both shapes superimposed over each other.

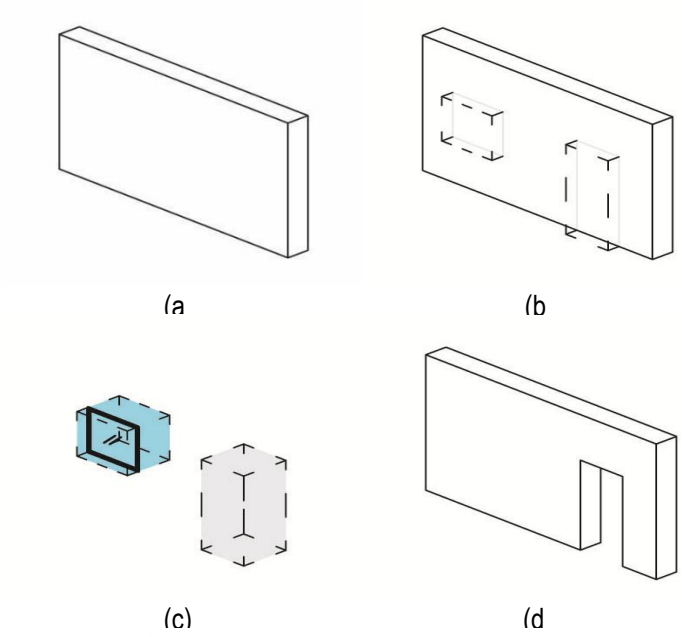


Figure 6: Example of void ignoring when the void is filled with an object. (a) A nesting wall with no void objects applied. (b) The original wall with the void objects superimposed on top of it. (c) The found IfcObjects encapsulated by these void objects, the grey void has no nested objects while the blue void has a nested IfcWindow object. (d) Only the void with no internal objects is applied, resulting in the simplified nesting wall geometry

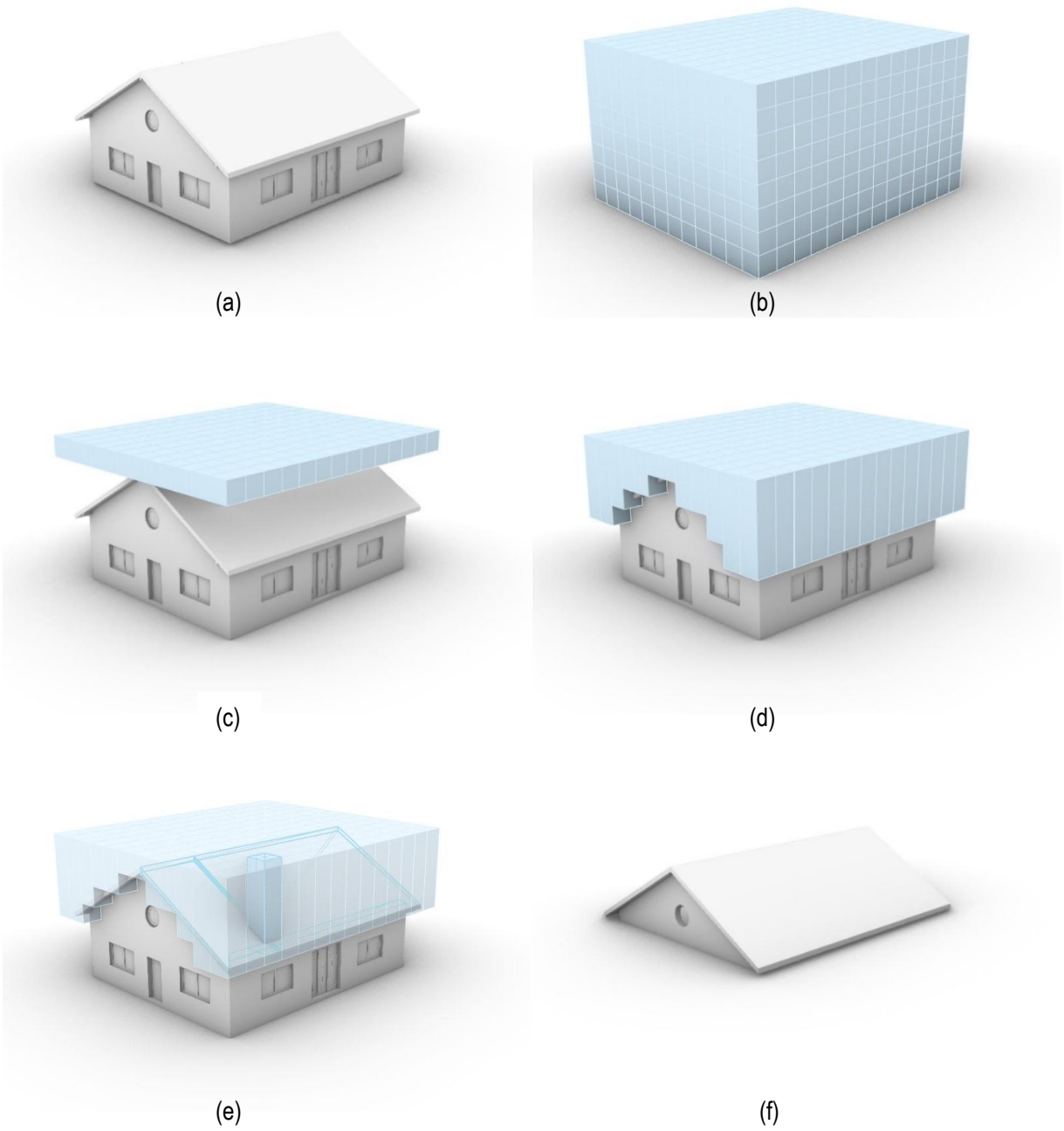


Figure 7: Example of the coarse roof object filtering by column voxelization. (a) The starting IFC model. (b) The full voxel grid. (c) The top slice of selected voxels. (d) The elongated collum voxels that hit geometry. (e) An isolated column voxel with the geometry it intersects with highlighted. (f) The resulting isolated geometry. Note that for the voxel examples the outer ring of non-intersecting voxels has been excluded to make the example more clear

Next, a single smallest bounding box in the XY plane is created that encompasses the whole point cloud - constructed from the geometry of the included objects - representing the potential space dividing objects. The bounding box is thus rotated around the model to result in the smallest possible volume box. If desired, the user can bypass this process and set the desired bounding box rotation themselves if the automated calculation does not yield a good result.

If mid-level or high-level processing is to be executed, a voxel grid will be created as well. The created bounding box is used as the base of the voxel grid domain. These dimensions are rescaled so that there is a ring of clear voxels surrounding the model and so that the actual user-selected voxel size will always be used. If mid-level processing is to be executed only, the voxel grid would be populated with voxel columns, which will be described in section 3.2.3. If high-level processing is selected to be executed or the voxelised shape is selected as the output, the voxel grid will be populated with full voxelization, which is described in section 3.2.4.

3.2.2 Low-level model processing

The LoD1.0 representation is a copy of the shape of the bounding box created in section 3.2.1. The four surfaces that have normal with a Z component of 0 are set to wall objects. Of the two unclassified surfaces that are left, the surface with the highest Z value is set as the roof object and the remaining surface as the ground floor object.

The LoD0.0 representation is the surface of the bounding box with the lowest Z value that has a normal with a Z component that is not equal to 0. This surface is isolated from the bounding box and stored in the output file. The surface is set to a roof object because this bounding box is based on the point cloud of the complete model and not just of the footprint. This means that if the building contains an overhang, this is included in the reconstructed surface, while a ground surface of a bounding box that is only based on the footprint could be smaller.

3.2.3 Mid-level model processing

The mid-level processing extracts the LoD envelopes that are primarily based on the model's roofing structure (LoD 0.2, 1.2, 1.3, 2.2). These roof-related elements are isolated in two steps: coarse filtering with column voxelisation and fine filtering via ray casting, see Figure 7 and Figure 8.

The column voxelisation starts by taking the top "slice" of the voxel grid. The voxels that are in this "slice" of the voxel grid will be the starting points of the columns. Each voxel will be elongated downwards with the voxel size dimension until the created column intersects with one or multiple objects. The growing of the column also ends if it hits the bottom of the voxel domain. E.g. if a voxel size of 1 is used, the column is 1 meter tall at the start, and then this will change to 2 meters, 3 meters, and so on until an intersection is found. The surfaces of the objects with which each of the columns intersects are collected.

The collected surfaces are fine filtered via a ray casting process where the rays are cast along the z-axis. Firstly, surfaces that have a normal with a Z component of 0 are removed from the surface list. The left-over surfaces are populated with a point grid which will function as the rays' origins. The points of this grid are placed by combining two different placement methods. The first method places the points on the surface at regular intervals along the U and V coordinates of the surface. This creates a regular grid covering the surface. However, depending on the shape and size of the surface, the resulting point grid could have no points placed near the edges of the surface. To avoid this, a second method is applied. The second method offsets the edges of the surface slightly inwards and places points at a regular interval along these offset edges. The offset edges are ignored after this.

From the grid points, rays are cast along the Z axis vertically upwards. If all the rays originating at the surface intersect with the other collected surfaces, the surface is assumed to be hidden and ignored. If the surface does have a single

ray (or more) that is cast that does not intersect with any of the collected surfaces, the surface is stored as a roof surface.

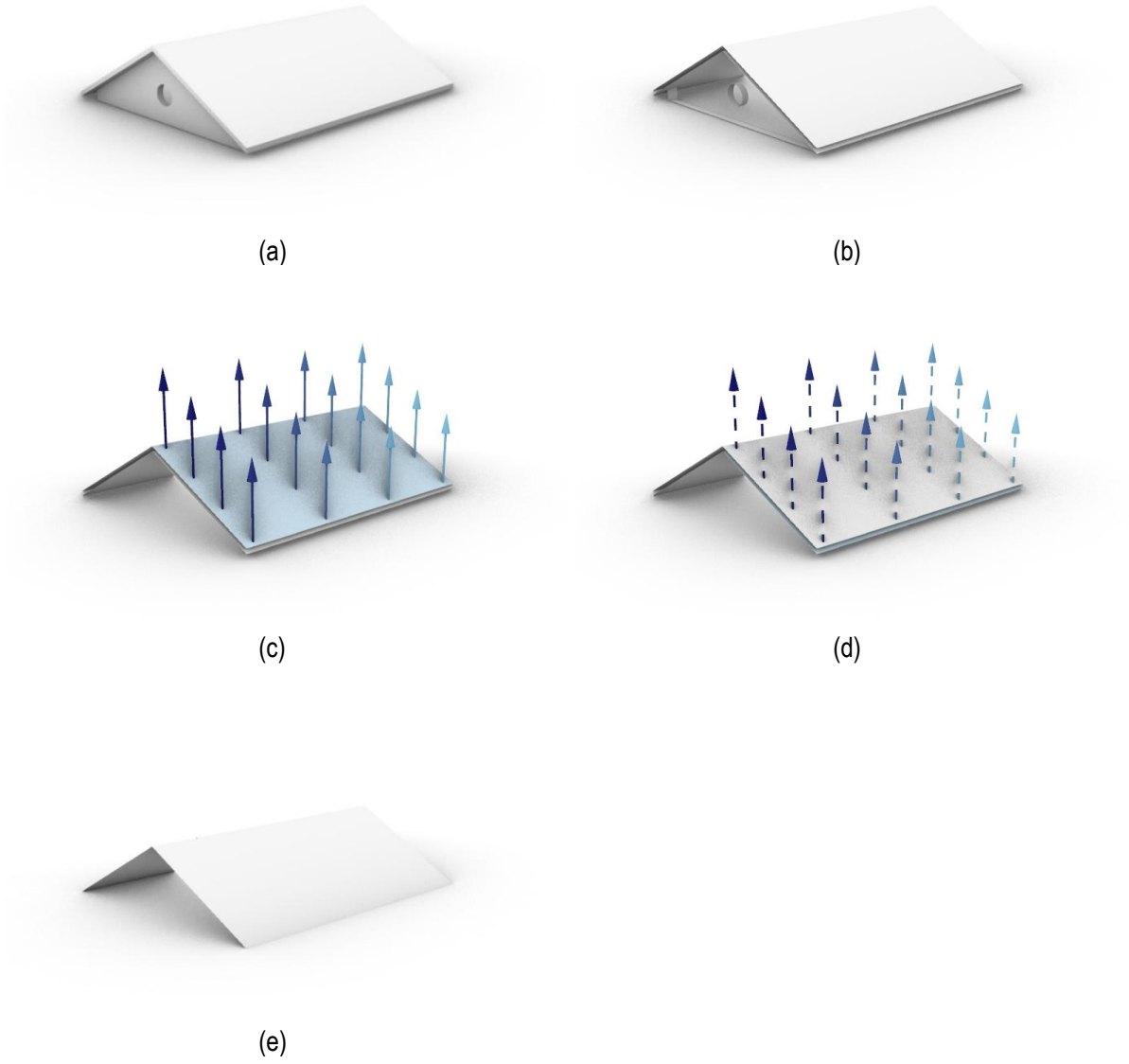


Figure 8: Example of the fine roof surface filtering process. (a) The starting geometry found by coarse filtering. (b) The surface collection after the surfaces with a normal with a Z component of 0 are removed. (c) The rays cast from the top closest to the camera surface of the roofing structure, all the rays are non-intersecting. (d) The rays cast from the surface directly underneath the surface in. (c) All the rays do intersect and thus the surface will be ignored. (d) The resulting fine filtered roof surfaces. Note that (c) & (d) display a selection of the surfaces and rays to make it easier to understand. The number of evaluated surfaces is in practice larger, and the resolution of the ray grid is much higher.

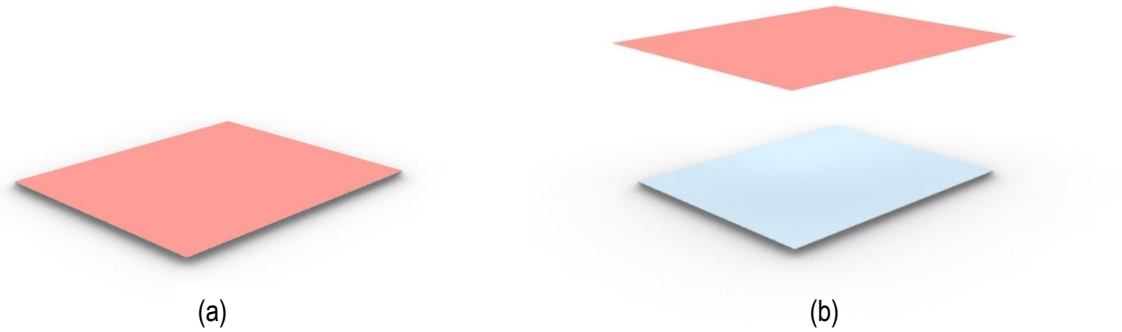


Figure 9: The different placement of the roof outline surface based on the desired output. (a) The roof outline is placed at the building's footprint height if no footprint output is desired. (b) The roof outline is placed at the building's max Z height if footprint output is desired.

To construct the LoD0.2 shell, these identified roof surfaces are all projected to the XY plane (z=0) and merged into a single surface: the projected roof outline. If footprint extraction will not be executed, the projected roof outline is stored as a roof surface at the ground level of the model, see Figure 9A. If footprint extraction is to be executed, the roof outline surface is translated to the max Z height of the building, see Figure 9B.

To construct the LoD1.2 shell, the roof outline surface at the ground level of LoD0.2 is taken and extruded to the max Z height of the building and stored as such. The same process for surface semantics that is applied for the LoD1.0 shape is applied here. The surfaces that have a normal with a Z component of 0 are set to wall objects. Of the two surfaces that are left, the top surface is set as the roof object and the bottom one as the ground surface.

The construction of the LoD1.3 and 2.2 shell is closely related. Both processes utilize the roof surfaces found by the ray-casting process. Only for LoD1.3, the roof surfaces are flattened so that the surfaces have a normal with X and Y components of 0. These flattened surfaces are translated to the top level of their unprojected counterpart. The rest of the process is identical:

Each surface is extruded to ground level and merged into a single solid. Because this is a very complex Boolean process, the extrusion and merging are done in a couple of simplified steps that focus on keeping the complex processes primarily in 2D. This is faster and more robust than working completely in 3D. The resulting solid shape can be stored in the output file. The same process for surface semantics that is applied for the LoD1.0 shape is applied here. The surfaces that have a normal with a z component of 0 are set to wall objects. From the remaining surfaces, the surface that has a normal with x and y components of 0 and with the lowest height is set as the ground surface, the rest of the surfaces are assumed to be roofing surfaces.

The LoD0.3 shell is a surface collection that is created by exporting the geometry from the first simplification sub step of the Boolean process that is executed for the LoD1.3 shell, see Figure 10. This collection is created by extruding all the flattened surfaces down to ground level. These extruded shapes are then used to split the flattened surfaces from which they were created. This results in a list of surfaces that, when looking along the Z axis in a negative direction, are either completely covered or uncovered by other surfaces. By taking a single point on each these surfaces and doing a ray cast along the Z axis in positive direction, the uncovered surfaces can be isolated. These surfaces are used for the LoD0.3 output and used for the further Boolean processing for LoD1.3.

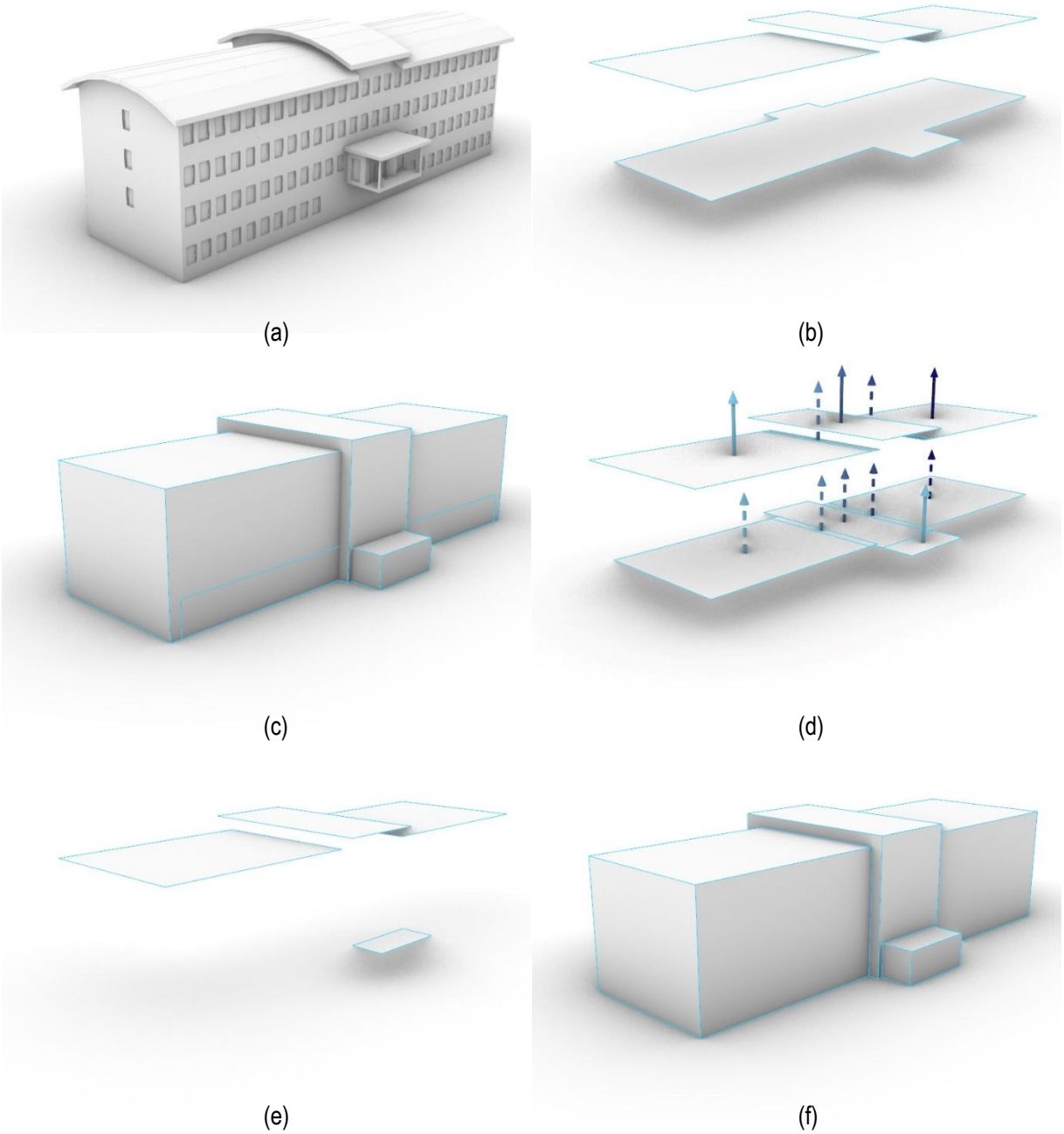


Figure 10: A selection of the steps that are taken for the creation of the LoD1.3 and 2.2 to highlight the LoD0.3 creation. (a) For this example, another model than in the former figures was picked. (b) The fine filtered roof surfaces that were isolated with the help of column voxelisation and ray casting. The surfaces were flattened. (c) The flattened surfaces are extruded to ground floor level and each original unextruded surface is split with the extruded shapes that intersect it. (d) The split surfaces are filtered with the help of a ray casting process, solid arrows do not intersect, dotted ones do. (e) the resulting LoD0.3 surfaces. (f) The LoD1.3 shape after extruding the LoD0.3 faces and joining and simplifying the resulting shapes.

3.2.4 High-level model processing

The high-level processing extracts the LoD envelopes that are based on the model's components that comprise the outer envelope (LoD3.2). These are isolated in two steps after the initial preprocessing steps: coarse filtering with voxelisation and fine isolation via ray casting.

The voxelisation process used for the high-level processing is straightforward. Every voxel is tested if it intersects with any of the space dividing objects. If it does intersect, it is marked as intersecting and the shapes the voxel intersects with are stored. The growth process of the outer shell is started by selecting a starting voxel which is outside of the building. This is a voxel from the outer ring of voxels mentioned in section 3.2.1. From here, the exterior "void" can be grown. Every "void" voxel can grow further while an intersecting voxel is not found. The intersecting voxels are collected and together form the voxelised outer shape. For each voxel in this shape the intersecting objects are collected. This list creates the coarsely filtered exterior objects.

The surfaces of these objects are sampled with a point grid following the same approach as described in section 3.2.3. From these points, rays are cast to a group of closest exterior voxel centers. If any of these rays from any of the grid points is unobstructed, the surface is assumed to be part of the exterior shell. These shapes are collected and exported as LoD3.2.

3.2.5 Interior processing

The interior processing steps are not based on the assumed space dividing object shapes. Instead, the creation of the interior spaces relies on the IfcSpace objects stored in the IFC file. Each IfcSpace object has its own geometry, this geometry is directly used by the IfcEnvelopeExtractor.

To construct the interior spaces of LoD0.2, LoD1.2, and LoD2.2, the top surfaces of the space geometry are selected with the help of a ray casting process. This process is similar to the extraction of the roofing surfaces described in section 3.2.3. First, the faces that have a normal with a Z component of 0 are ignored. The left-over faces get a point grid populated by taking the centre points of each triangle of their triangulated shape. From each of these centre points a ray is cast in the positive Z direction. If one of the rays of a surface does not intersect with another surface, the surface is assumed to be part of the top of the space.

To construct the LoD0.2 shell, these top surfaces are all projected flat to the XY plane at the lowest height of the original IfcSpace geometry. These flattened shapes are then merged into a single surface effectively creating a ceiling outline of the space. This ceiling outline is stored as a CeilingSurface. Because the interior LoD0.2 shapes currently only include the extraction of this singular shape, no top surfaces or a footprint extracted shape is stored.

To construct the space's LoD1.2 shell, the merged flattened room's roof outline is taken and extruded to the max Z height of the IfcSpace geometry. The logic that is applied for the surface semantics is the same as the one that was applied to the exterior LoD1.2 extraction, but only the types that are applied are related to interiors. The surfaces that have a normal with a Z component of 0 are set to InteriorWallSurface objects. Of the two surfaces that are left, the top one is set as the CeilingSurface object and the bottom one as the FloorSurface object.

To construct the space's LoD2.2 shell, the extracted space top surfaces are all extruded to the space's lowest Z height and merged. Just like with LoD1.2, the surfaces that have a normal with a Z component of 0 are set as InteriorWallSurface objects. Of the surfaces that are left, the lowest one is set as the FloorSurface object and the others as ceilingSurface objects.

To construct the space's LoD3.2, the geometry of the IfcSpace is preserved as is.

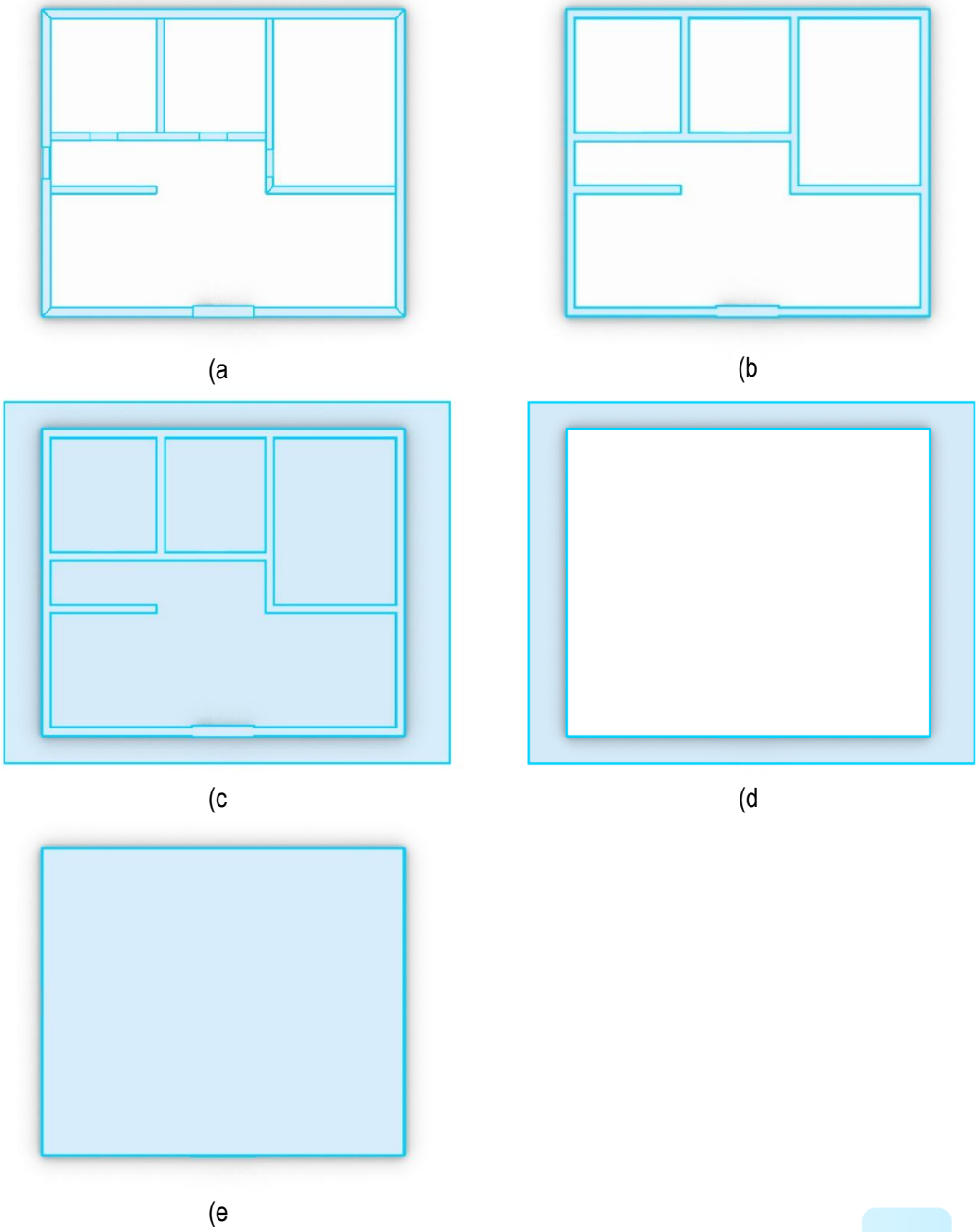


Figure 11: Example of the footprint and storey extraction process. (a) The outer wires from the horizontal cut have been converted to surfaces. (b) These surfaces are joined to reduce the complexity. (c) A horizontal face completely encompassing the surface cluster is added. This surface is split by the cluster. (d) The outer most split surface is isolated. (e) The inner ring is converted to a surface to represent the footprint or section.

3.2.6 Other processing steps

Some processes that are executed by the tool do not fall within the four subprocesses. For example: the footprint and storey geometry extraction.

Both the footprint and the storey geometry extraction use the same processes, see Figure 11. Horizontal sections are made through the entire model's space dividing objects at certain Z heights. For the footprint, this height is manually picked by the user; for the rest of the storeys, these heights are taken from the elevation attribute of the IfcBuildingStorey objects. If desired, the user can set an offset to this value, e.g. 1.2 meters to have it represent a regular floorplan section.

The horizontal section of the space dividing objects results in a horizontal wire for each object. These wires are converted into the boundaries of horizontal surfaces. These surfaces are merged and simplified, usually resulting in a drastic reduction of surface and edge count. This group of surfaces does not however represent the horizontal storey section yet. The interior of the building is now represented as a void and not as a surface, see Figure 11B. To resolve this, the created horizontal surfaces are used to split an oversized horizontal bounding surface that surrounds them. The outermost resulting shape is isolated. This shape represents the left-over area between the bounding surface and the building's horizontal section. However, its inner loops represent the outer loops of the section of the building. Converting these outer loops into surfaces themselves creates the simplified sections. These shapes are stored.

3.3 Input requirements

The IfcEnvelopeExtractor has been developed to function on a wide range of input models of different quality, including some somewhat unconventional models. During the development process, a leading principle was that it should work on models that have been created by people who may not be BIM experts. Additionally, the application has been developed to function on draft models. i.e. models that are not finished yet. This was done to enable architects and designers to export their draft designs and evaluate them at a city scale with the associated GIS tools.

Although the goal of the tool is to function on draft models, the complexity of the geometric algorithms demands that certain requirements are met to improve stability. The tool will function if not all these requirements are fully met, but the chances of success are influenced by it. For the exterior shell extraction, the requirements can be split up in the three different complexity levels (low, mid and high-level envelopes). These levels follow the three external shell processing methods that are described in section 3.2. The low-level shells are based on the point cloud derived from the geometries in the BIM model, mid-level shells are based on its roofing structure and high-level shells on the complete outer envelope.

Each level has unique requirements that are built on top of the requirements, stemming from the level before it. The higher the quality of the desired LoD envelope, the higher the required quality of the input IFC model. E.g. for mid-level shell extraction, the mid-level shell requirements + the low-level shell requirements should be adhered to. The requirements within one level are listed in order of importance.

These levels only apply the extraction of exteriors. For interior reconstruction, additional requirements are to be met, which extend the requirements of the highest level of detail.

The effect of ignoring requirements varies greatly. For example, the tool will end the process immediately if an invalid IFC file is submitted. In contrast, a non- watertight roofing structure could yield usable output in the best-case scenario. However, the tool could also return no output or crash. In general, if possible, it is recommended to adhere to the requirements. The software can potentially still output usable models if the requirements are not met but the chances

of success are reduced while computing time and the chance of crashes is increased. The requirements for each process level are as follows:

Low-level shell (LoD 0.0, 1.0 & voxel shells):

- Valid IFC4x3, IFC4 or IFC2x3 file
- Valid units
- Correctly classified objects* (recommended)
- No or limited use of IfcBuildingElementProxy objects (recommended)
- Site and building structurally separated (recommended)

Mid-level shell (LoD 0.2, 1.2, 1.3, 2.2):

- Correctly modelled and watertight roofing structure
- No or limited use of triangulated objects (recommended)

High-level shell (Footprint, storeys, LoD 3.2):

- Correctly modelled and watertight building exterior

Interior data:

- Correct IfcBuildingStorey use for storey creation
- No or limited use “half” stories.
- Correct IfcSpace use for space creation (both semantically and geometrically) **

The performance of the tool also depends on the complexity of the input model. The tool has proven to function well on models with simple objects but can struggle on complex shapes. For example, performance for triangulated complex shapes tends to be poor. So, models with rounded corners, domes and curved roofs perform poorly.

*The default behaviour of the tool utilizes IfcWall, IfcCurtainWall, IfcWallStandardCase, IfcRoof, IfcSlab, IfcWindow, IfcColumn, IfcBeam, IfcDoor, IfcCovering, IfcMember and IfcPlate objects. If default settings are used, it is recommended to have these objects correctly classified in the file. If different types are used, these types would need to be correctly classified as well.

**Note that software like Autodesk Revit natively has some issues that can prevent it from doing this properly,

3.4 User interfaces

To successfully convert a BIM model to a GIS model, a certain workflow must be followed, see section 3.5. To help the user control the application, two interfaces have been developed, a GUI and a configuration file based on JSON.

3.4.1 Graphical user interface

The Graphical User Interface (GUI) has been developed to give a user-friendly option to control the settings to process their BIM models, see Figure 12. This is a selection of settings that are often changed (e.g. desired output, file paths and voxel sizes) and does not require in-depth knowledge of programming or geometric processing, such as

voxelization logic, thread counts and error output. This does not mean that the hidden settings are not utilized when the GUI is used. The user has no direct control over these “advanced” settings, but the tool will automatically set them to reasonable values.

The GUI itself is a simple program created in Python. The GUI is available from the GitHub repository, both as a Python file and an executable. The executable has been added to allow users to run the GUI without the need to install Python on their system. This executable can also be moved around everywhere on the user’s system. However, the executable should always share the folder with the precompiled version(s) of the extractor tool.

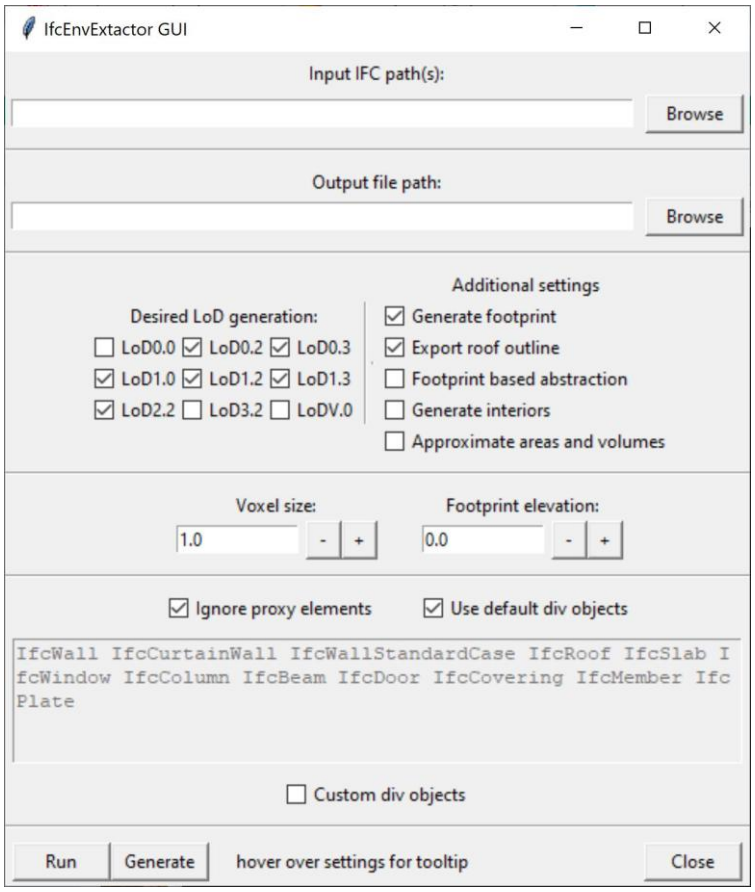


Figure 12: The GUI of the IfcEnvelopeExtractor

The settings available in the GUI are:

- Input IFC file path (file browser, allows multiple paths)
- Output IFC file path (file browser)
- Desired LoD generation (toggle)
    - LoD0.0, 0.2, 0.3, 1.0, 1.2, 1.3, 2.2, 3.2, and V.0 (voxel)
- Additional Settings (toggle)
    - Generate footprint

    - Export roof outline
    - Footprint based abstractions
    - Generate Interiors
    - Approximate areas and volumes
- Voxel size (field)
- Footprint elevation (field)
- Ignore proxy elements (toggle)
- Use Default div objects (toggle)
- Custom div objects (toggle and field)

The GUI generates a configuration JSON file and feeds it automatically to the correct precompiled version of the extractor. The GUI can also be used to create a configuration JSON file that can be adjusted or used later.

3.4.2 ConfigJSON

The configuration JSON file allows the user to adjust the settings of the application via the JSON format, see Figure 13. The configuration file gives the user more control over the tool compared to the GUI. The additional exposed settings are mostly related to improving performance on certain systems.

The additional settings that are being exposed by the configuration JSON are:

- Voxel logic (int 3 or 4)
    - 3 – center plane intersection
    - 4 – surface intersection
- IFC rotation (double or Boolean)
    - Allows for a custom rotation of the voxel grid
- Generate report (Boolean)
- Threads (int)
    - The max allowed number of threads to use simultaneously
- Horizontal section offset (double)
    - The offset height at which the storey geometry sections are taken
- Georeference (Boolean)
    - If true, the model will be georeferenced
- Merge semantic objects (Boolean)
    - If true, semantic objects are merged if they have identical attributes

```
{
    "Filepaths": {
        "Input" :
        [
            "path to IFC file",
            "Potential path to other IFC file"
        ],
        "Output" : "path to export (City)JSON file"
    },
    "LoD output": [
        5.0,
        0.0,
        0.2,
        1.0,
        1.2,
        1.3,
        2.2,
        3.2
    ],
    "Voxel":{
        "Size": 1,
        "Store values" : 0,
        "Logic" : 3
    },
    "IFC": {
        "Rotation" : false,
        "Default div": true,
        "Ignore proxy": true,
        "Div objects" : []
    },
    "JSON" : {
        "Footprint elevation": 1,
        "Footprint based" : 0,
        "Horizontal section offset": 1.2,
        "Generate footprint": 1,
        "Generate roof outline": 1,
        "Generate interior": 1,
        "Georeference" : 1,
        "Merge semantic objects": 1,
    },
    "Generate report": 1,
    "Threads": 12
}
```

Figure 13: Example of a Configuration JSON for the IfcEnvelopeExtractor

3.5 User Workflow

3.5.1 Input File Preprocessing

Preprocessing of the IFC files before using the IfcEnvelopeExtractor is only required for IFC files that have some issues. In theory, for models complying with the IFC standard, no preprocessing should be required. However, there are issues that occur quite often that can be resolved before further processing. This section will cover the most common ones and how to resolve them. Resolving these issues will improve both the quality of the IFC file and the quality of the output of the IfcEnvelopeExtractor.

The first step of preprocessing is to check if the hierarchical split between the building and its surrounding site is correct. Often parts of the site are incorrectly related to the IfcBuilding object, see Figure 14. The consequence is that the IFCEnvelopeExtractor is not able to distinguish between an IfcWall that is part of the building and an IfcWall that is not. Both will be processed as if they are part of the building. This can yield undesirable results if IfcWall objects are used as space dividing objects. The easiest way to preprocess IFC files with this issue is by eliminating all the objects that are not part of the building. This can be done in the BIM software directly before exporting to IFC. Alternatively, usBIM viewer or BlenderBIM can be used to directly edit the IFC files.

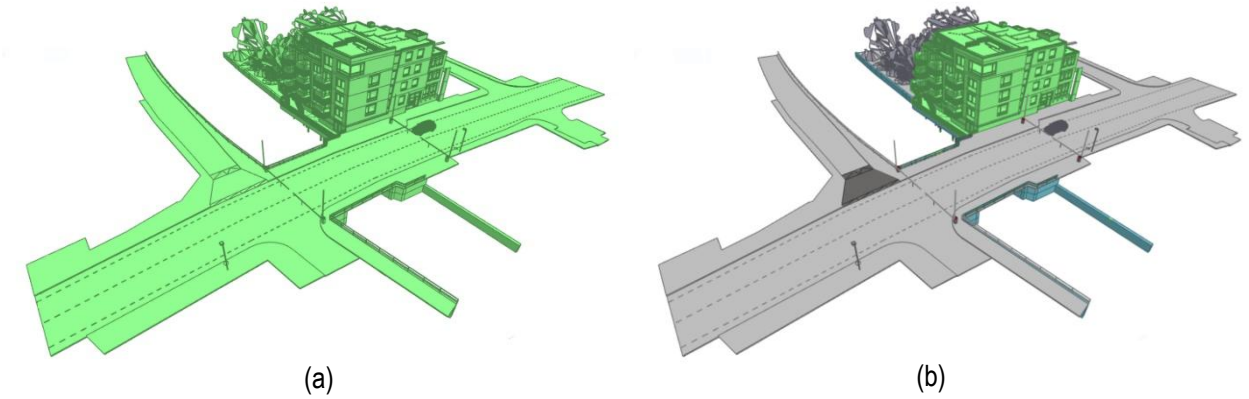


Figure 14: Example of the hierarchical split between the building and its surrounding site. Both (a) and (b) have the objects that are related to the IfcBuilding object selected. In (a) all the objects in the model are related to the IfcBuilding object, also the objects of site. (b) Shows the desirable situation where the site is not directly related to the IfcBuilding object.

The following step is checking the IfcSpace objects and their relationships. If there are no overlapping IfcSpace objects or no interior output is required, this step can be skipped. If there are IfcSpace objects encompassing other IfcSpace objects, it is recommended to check their CompositionType. The extractor only evaluates the IfcSpace objects with a CompositionType of ELEMENT. However, the CompositionType is almost always incorrect when nested IfcSpace objects are present. Nested IfcSpace objects often occur when the IFC file includes spatial data related to apartments or zones. The incorrect CompositionType seems to be an issue arising mostly from exporting IFC from BIM. It can be resolved by removing the IfcSpace objects that encompass other IfcSpace objects, or by changing their CompositionType to COMPLEX. It is recommended to do this with an IFC file editor like BlenderBIM because there is very little control from the BIM modelling software itself.

The final step is checking the object types that are present in the model. This is only required for the objects that will be used as space dividing objects for the extraction of the shell. All the other object types will be ignored by the

IfcEnvelopeExtractor, so there is no need for them to be correct. For example, IfcFurniture incorrectly typed as IfcBuildingElementProxy has no effect on the output if objects of these two types are not considered as space dividing objects. However, if part of the facade is modelled as IfcBuildingElementProxy, it will have effect on further processing. It is recommended to have all the object types that are part of the physical model classified with their correct IFC type and reduce the use of IfcBuildingElementProxy to a minimum. Correcting the type should be done directly in the BIM software This will also avoid problems for other applications. If the original file is not accessible, BlenderBIM can be used to change the type of objects.

3.5.2 Setting Software Variables (GUI)

The setting of the variables in the GUI is a fairly simple process. Most settings are easy to comprehend but there might be some deeper understanding required to effectively use the tool.

The **input IFC path(s)** allow the user to choose the files that will be evaluated. The tool supports multifile IFC models. However, all the models that are submitted will have to be parsed to check the file's IFC version, units and included object types. This means that all submitted files will be parsed even if they store no space dividing objects. For small models this will have little effect, but for large models this can slow down the process significantly. It is thus recommended to only set file paths to files that contain space dividing objects. Site, plumbing, or ventilation models are thus better excluded from this selection.

The **Output file path** allows the user to choose the storage location of the generated CityJSON file. The file extension should always be either ".json" or ".city.json". The application is not able to create new folders, so the output file path must utilize an existing folder structure. If not, the new folders have to be made prior before running the tool.

The **Desired LoD generation** toggles allow the user to choose the LoD abstraction shells that should be exported. As was mentioned in section 3.2, the desired LoD output has an effect on which algorithms are used. For more complex IFC models, it is recommended to only select the LoD abstractions that are required for further processing and not run every possibly output. The more algorithms are called, the slower and more crash prone the process will be. E.g. if LoD1.3 is the only one that is required for further processing, there is no reason to also generate 2.2 and 3.2.

The **Additional settings** toggles allow the user to tweak the LoD abstractions that are created. This could be desired if certain data is required from the input model in the exported output. At the time of writing, there are four additional settings available in the GU:

The **Generate footprint** settings will generate and store a footprint of the model in the LoD0.2 abstraction. This is done by taking a horizontal section through the building at the height of the **Footprint elevation** value. Toggling the generate footprint setting will force the tool to perform a full voxelisation. This could slow down the process if full voxelisation was not required for the chosen LoD abstractions. If the 'generate footprint' setting is toggled off, the roof outline will be projected vertically and placed at the Footprint elevation value to represent the ground surface.

The **Export roof outline** setting will store the roof outline of the model in the LoD0.2 abstraction. This shape is always created for LoD1.2. This means that adding it to LoD0.2 will effectively have no cost. However, if LoD1.2 is not required, exporting the roof outline for LoD0.2 abstraction will have an added computational cost.

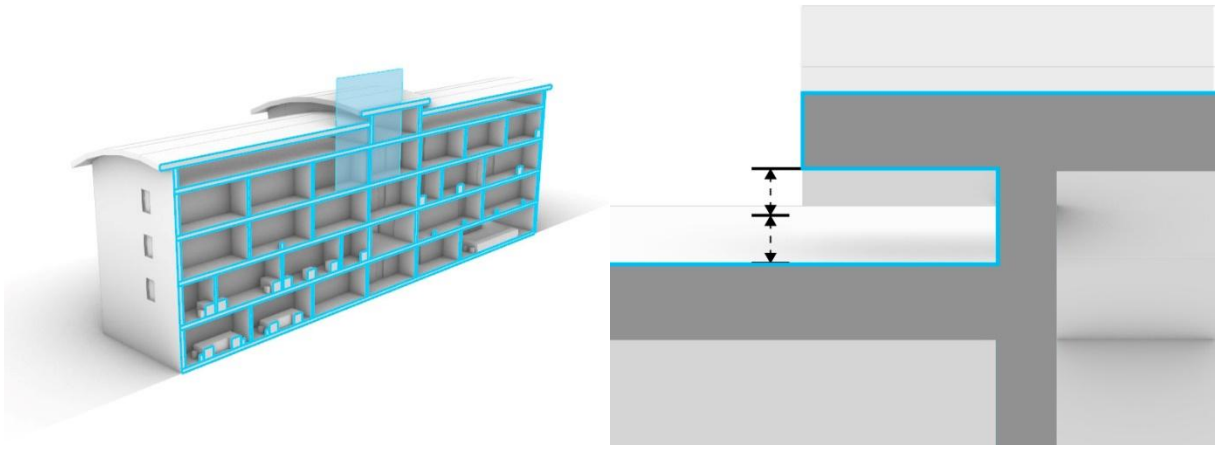
The LoD0.2 abstraction requires either the **Generate Footprint**, **Export roof outline,** or both to be active. If both are inactive, there is no exterior geometry that is added to the LoD0.2 abstraction.

The **Footprint based abstractions** setting can restrict the LoD1.2, 1.3 and 2.2 output to the footprint outline. Toggling this setting will however force the tool to apply more complex Boolean processes to generate the footprint geometry, which will slow down the processing time and increase the chance of failure.

Other settings:

The **Generate interiors** setting can create interiors for LoD0.2, 1.2, 2.2, 3.2, as well as voxelised output. It will however only generate the interiors of the selected LoD abstractions in the **Desired LoD generation** options. E.g. If only LoD0.2 is selected, only the interior for LoD0.2 will be generated and stored. The generation of interiors is, usually, a very simple process. The computing cost of this process depends on the number of rooms in the input model. If the input model is large, generating interiors can take a noticeable time.

The **approximate areas and volumes** setting will compute the areas and volumes of storeys, spaces and the exterior shell based on the voxelised shape. If this setting is toggled on, it will force the tool to do a full voxelisation even if this was not required for other required output. The resulting values are also subject to the chosen **voxel size.** A small size would result in different values than a large one. As the settings indicates, this is an approximation and not the actual value, the actual values could be very different.



(a) (b)

Figure 15: A simple model that dictates a voxel size restriction for successful LoD3.2 extraction. The full model can be seen in figure 9a. (a) Shows a section of the model within blue the area highlighted that is shown in 2D section in (b). It can be seen in (b) that to find the outer surfaces correctly the voxels should be able to grow underneath the overhang. This means that the voxel size should be maximal a half time the distance between the two roofing surfaces.

The **voxel size** allows the user to have control over the resolution of the used voxel grid. Selecting the optimal voxel size is however very complex due to how many algorithms rely on the voxel grid. The usual rule for voxel sizes is that the smaller the voxel size, the longer the processing time. And this is also true for this application in the sense that the voxelisation process is slowed down drastically when the voxel size is increased. However, the voxel size is also used for the coarse filtering of the mid and high-level shell extraction. Smaller voxel sizes tend to improve the coarse filtering process and avoid unnecessarily many fine filtering processes to be executed, which can cost a lot. So, smaller voxels improve the object filtering speed if those LoD abstractions are required.

The complexity of the building does however occasionally restrict the usable voxel size. The voxel grid must be able to grow to every object that could have surfaces that could be part of the shells. If the voxel size is too large to allow this,

the resulting shape could be incorrect or could have missing surfaces. Even simple models could already have this restriction, see Figure 15. It is recommended to manually measure the gaps between certain surfaces. For mid-level extraction only the roofing structure must be considered, for high-level extraction all the surfaces should be considered. The voxel size should be smaller than half the size of the smallest gap.

As mentioned before, the approximate areas and volumes setting will utilize the voxel grid to approximate the areas and volumes of the building's objects. The results of these approximations are heavily dependent on the used voxel size. If the gap measurements are not done, a rule of thumb would be to use a voxel size of between 0.3 to 0.5 meter. It is not recommended to use a voxel size that is larger than 1 meter or smaller than 0.1 meter.

The **footprint elevation** setting allows the user to select the height of the ground floor. This does however not mean that it has to be at that same elevation as storey number 0. This height can be picked based on the desires of the user. The desired location can be measured in an IFC viewer. This height is used for the footprint section height, but it is also used for the storey count approximation and the voxel area and volume approximation. This means that even if the footprint is not required by the output, it is still recommended to set this value correctly.

The **division objects** selection allows the user to select the types of objects that the IfcEnvelopeExtractor evaluates. The selected objects are the only objects that are evaluated by the tool. Selecting this carefully is the easiest way to improve computing speed performance of the tool. The default settings are created based on many different IFC models and should work on most normal input models. Some models use the occasional IfcBuildingElementProxy object as space dividing object. If this is the case, the option to ignore those can be toggled off with the **Ignore proxy elements** toggle. The object types to be used can be seen in the grey text box.

If the envelope of the input model is constructed out of different objects than the default list, extra objects can be added or removed. This can be done by toggling the **Custom div objects** settings. This will unlock the grey text box that is filled with the used object types. The types can be added to this list or removed from it. The types must be correct IFC types and separated by a space or tab. The tool will check if the entered types are valid. It is not case sensitive.

To do a quick check of the space dividing objects in an IFC model, the model can be opened in BIMvision. Under OBJECTS -> Types, the structural types in the file can be seen. Here, it is easy to toggle certain types off and on to see which ones could be part of the space dividing objects. This makes it easy to see if the IfcBuildingElementProxy objects are used, and at which locations.

When all these settings have been set properly, the tool can be started by pressing the **RUN** button. Alternatively, a configJSON can be generated by pressing the **Generate** button.

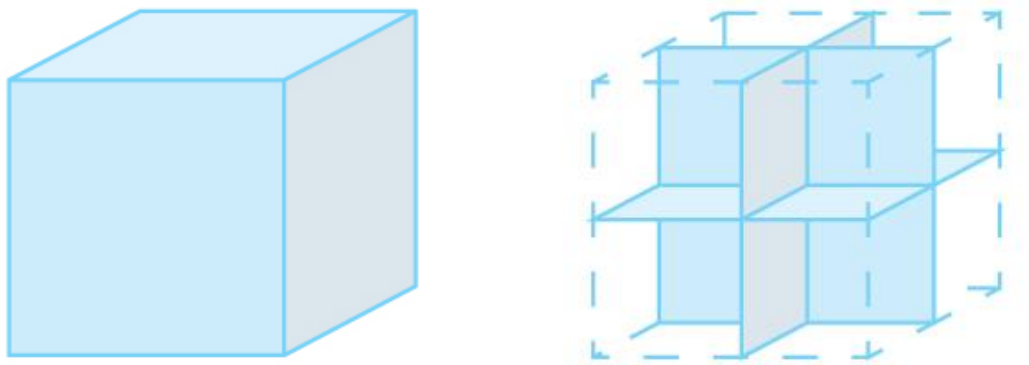
3.5.3 Setting Software Variables (ConfigJSON)

As mentioned in section 3.4.,2 the configuration JSON file allows the user to access advanced and more complex settings. This JSON file can be created completely from scratch by the user, or it can be generated by the GUI. The advantage of the generation via the GUI is that it will be created in the correct format and avoid typos.

Several of the settings available in the configuration JSON are accessible via the GUI, although possibly under a different name. This section will only cover the unique configuration JSON settings.

"**Voxel**" "**Logic**" determines the intersection method used for voxelisation. The value can be either 2 or 3. 2 will do a plane intersection, and 3 will do a full solid intersection, see Figure 16. A plane intersection means that an object is considered to be intersecting with a voxel if it intersects with an x, y, or z, plane that goes through the centre point of the voxel. A solid intersection means that an object is considered to be intersecting with a voxel if it intersects with any

of its exterior faces or is completely located inside of it. The solid intersection evaluation is computationally more costly due to more intersections that have to be checked, but the result is often more reliable for the further processing by the tool. The default value is 3.



(a) (b)

Figure 16: Difference between the two different voxel logics. (a) Shows the default logic, voxel logic 3, where each face of the voxel is used for the intersection. (b) Shows the alternative logic, voxel logic 2, where 3 planes are used for intersection.

"**IFC**" "**Rotation**" allows the user to override the automatic rotation of the model during processing. As described in section 3.2.1, the smallest bounding box in the XY plane is created around the model for voxelisation. By setting a floating-point value, the rotation process of this bounding box creation will be avoided, and the box will be placed according to the value, which is set in degrees. This can be useful when the voxelised output is used and the voxels are desired to be rotated to a certain angle. For example, this setting is desirable when multiple different models are voxelised and merged into a single GIS model for further processing. The custom rotation of the bounding box can however result in a bigger bounding box, which means that the voxel grid will include more voxels, all of which have to be evaluated for intersections, possibly slowing the process down. If no voxelised output is desired, it is recommended to not utilize this setting and keep it on the default value. If the auto rotation process is desired, the value has to be set to false. If the true north of the IFC model is to be used the value must be set to true. The default value is false.

"**JSON**" "**Horizontal section offset**" allows the user to add an offset to the evaluation of the footprint and storeys. This could be desirable if certain features of the building are to be avoided or included in the section process. The set value will be added to the values found in the IFC file. However, the created geometry will be placed back at the height of these IFC values. For example, if the offset is set to 1.2 meters and there is a storey with an elevation of 10 meters, the section to generate the storey will be done at 10 + 1.2 = 11.2 meters. The created geometry will be translated down from 11.2 back to 11.2 – 1.2 = 10 meters. So, it will reflect the elevation found in the storey object. The default value is 0.

"**JSON**" "**Georeference**" allows the user to set if the model is to be transformed according to the georeferencing data in the IFC. This data is often incorrect in IFC files, which can result in unexpected results. This can be avoided by settings the value to false, in which case the model will be placed close to the XY origin. The default value is true.

"**JSON**" "**Merge semantic objects**" allows the user to choose if the semantic objects that are stored in the output CityJSON file are to be compressed. This compression is done by merging the semantic objects that include identical attributes and values into a single semantic object. This can however cause issues in downstream GIS applications. To avoid these issues, the value can be set to false, this will avoid the compression process at the cost of larger output files. Default value is true.

"**Output report**" allows the user to choose if a JSON report file will be stored. This report file will have the same file name as the set output name with the addition of a "_report". This file includes a summary of the settings used and of the issues encountered during the export. This enables the user to see how the model was created, and if issues were encountered, what these issues were. The only cost for enabling the output report is the cost of writing the JSON file itself. The tool creates the summary in memory even if this output report is not desired. The default value is true.

"**Threads**" allows the user to choose the maximum number of threads the tool is allowed to use. Multithreading enables the tool to utilize the power of multicore processing. More threads mean that certain processes executed by the tool can be split in smaller parts. However, too many threads can overload the system, slowing it down considerably. The number of threads the system has available is often the CPU core number times 2. So, if a system has 4 cores the correct number of threads is 8. This formula is only valid for systems that are not in use by other processes. If other processes are in use, or the user wants to use the system while running the tool in the background, it is recommended to lower the number of threads. The default value of the tool is 0, which is interpreted by the tool as the number available threads according to the CPU specifications minus 2.

4. Results & discussion

To test the performance of the IfcEnvelopeExtractor tool, the IFC models of the of the four CHEK use cases were used, see Table 3, Figure 17a and the first figures of appendix 7.2, 7.3 and 7.4. These are models with complex geometries that could be encountered in practice. The models have been partially pre-processed. Most of the issues that were encountered were relayed to the designers and resolved. Only the site has been split from the rest of the building manually by editing the IFC file with BlenderBIM without consulting the original model's designer.

Table 3 properties of the pre-processed input models

Model Name	IfcObject count	ProxyElement Count	Storeys	Spaces
APC	1773	0	✓	✕
Gaia	990	3	✓	✓
Lisbon	586	0	✓	✓
Praha	3589	186	✕	✓

The quality of the outer shell abstractions that were created by the IfcEnvelopeExtractor can be seen in Table 4 , Figure 17 and appendix 7.1,7.2, 7.3 and 7.4. The processed APC model show a very promising output. For all four models, the generated LoD abstractions reflect the input model well. The Lisbon and Praha models show some issues with the LoD1.3 abstraction, but all the other abstractions are accurate. The Gaia model shows failed abstractions for the mid-level and high-level processing.

Table 4 Validity of the exterior output of the CHEK example models.

	LLoD		MLoD					HLod	VLoD
Model Name	0.0	1.0	0.2	0.3	1.2	1.3	2.2	3.2	voxel
APC	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gaia	✓	✓	✕	✕	✕	✕	✕	✕	✓
Lisbon	✓	✓	✓	✓	✓	✕	✓	✓	✓
Praha	✓	✓	✓	✓	✓	–	✓	✓	✓

The results are grouped based on low-level (LLod), mid-level (MLod) and high-level (HLoD) processing. The failing mid-level and high-level abstractions of the Gaia model are directly related to the geometry of the building. To keep the scope of the project within reasonable limits, the development has been focused on the processing of straight edges and flat surfaces. If models that have curved edges or curved surfaces are encountered, the processing performance of tool is reduced significantly, as was mentioned in section 3.3. The Gaia model has curved outer walls, and the poor mid-level and high-level performance is related to these parts of the model. Instead of crashing or avoiding the unexpected geometry, the tool processes the edges and surfaces as if they were flat, see Figure 18. This results in a shape that does not accurately reflect the actual building. Additionally, in the case of the Gaia model, there are small gaps that are formed when the curved surfaces are interpreted by the tool as flat surfaces. These gaps also made the 2D shapes invalid and hamper the generation of extruded shapes.

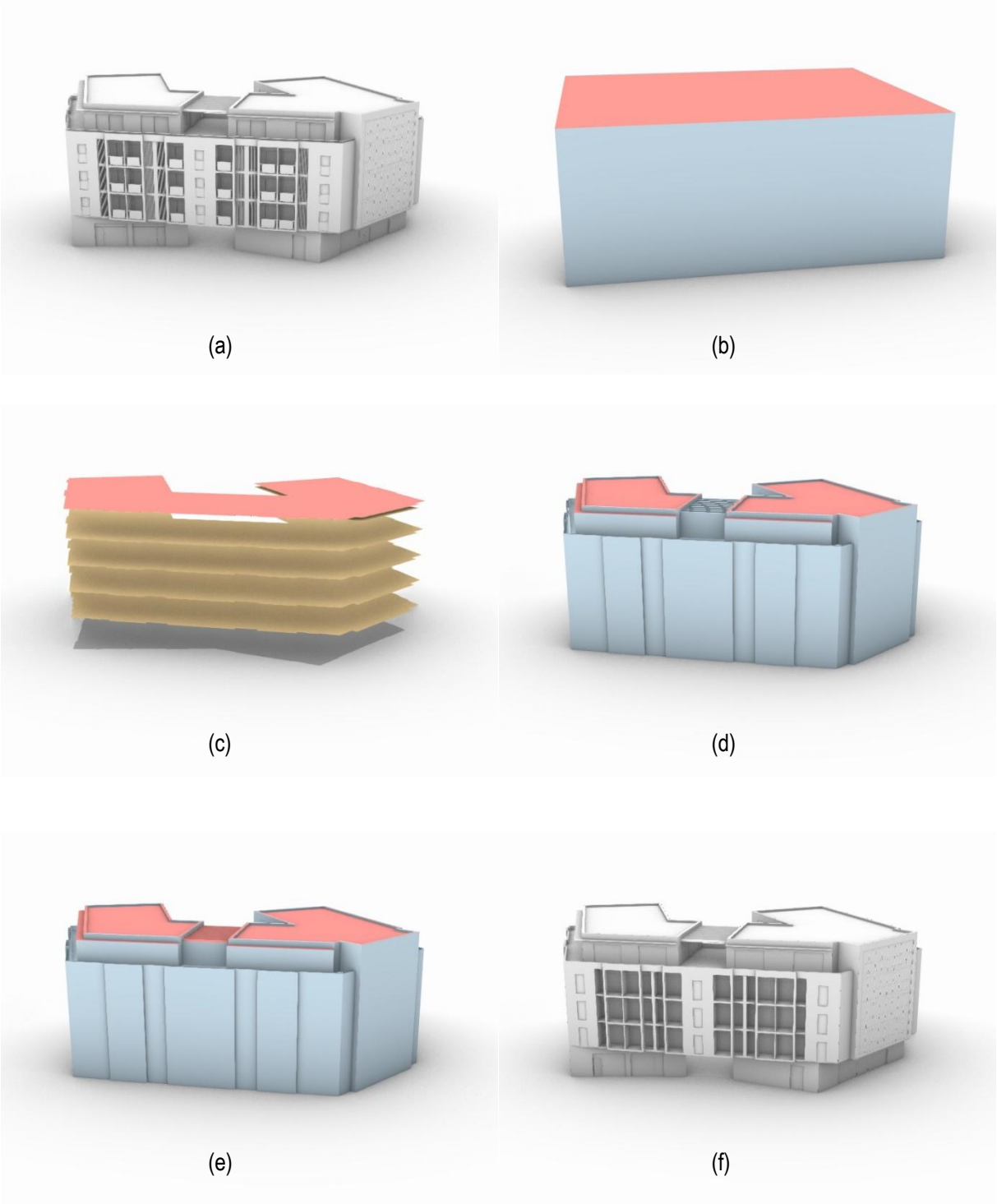


Figure 17: Selection of the abstractions created by the IfcEnvelopeExtractor based on the APC IFC model, see appendix 7.1 for the full results. Gray = ground surface, blue = wall surface, red = roof surface and orange = storey surface. (a) Shows the input IFC model. (b) Shows the LoD1.0 abstraction. (c) Shows the LoD0.2 abstraction. (d) Shows the LoD1.3 abstraction. (e) Shows the LoD2.2 abstraction. (f) Shows the LoD3.2 abstraction.

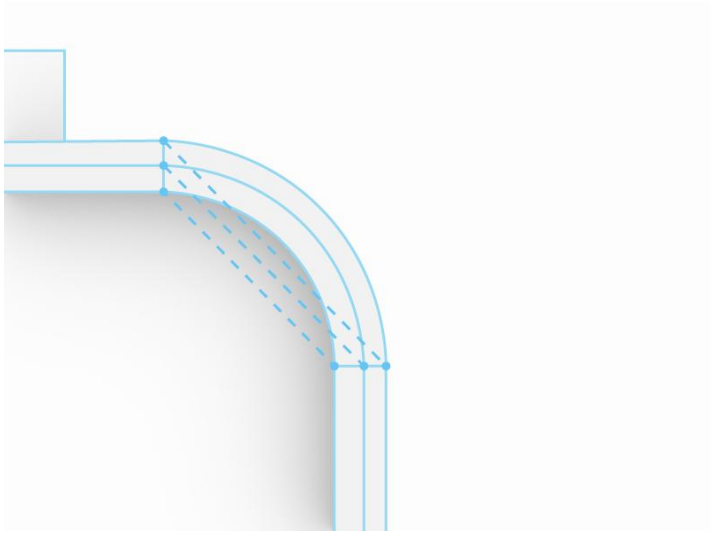


Figure 18: The incorrect interpretation of curved edges and surfaces by the IfcEnvelopeExractor. From this selection of the top projection of the Gaia IFC model it can be seen that the outlines are constructed out of curved edges (solid blue lines). The tool will however interpret these lines as straight between their endpoints (the dotted lines)

When looking at the results of the Praha model, an interesting issue can be noted in the LoD1.3 abstraction. The tool processes the Praha model perfectly if the IfcBuildingElementProxy in the file are ignored. However, as can be seen in Table 3, there are many proxy elements present in the model. These objects represent the large exterior staircase and the finishings of the roofing structure. If these objects are included in the process the LoD1.3 abstraction fails to include all the buildings' features, but the other abstractions are enriched by the added objects, see **Error! Reference source not found.**.

The results of the Lisbon model show that the LoD1.3 abstraction process runs into an issue as well. Incorrect void "columns" are pulled through the building. This is caused by the balcony surfaces being incorrectly ignored. Interestingly these missing surfaces are present in the LoD0.3 abstraction. This means that the tool does correctly find the surfaces, but somewhere in the extrusion and merging step of the LoD1.3 creation they are lost. Most likely the issue encountered in the Lisbon model, but also for the Praha model, is related to a too complex Boolean process. More pre-filtering is required to improve the output. Improving the pre-filtering step is work in progress

Table 5 Validity of the interior output of the CHEK example models

Model Name	0.2 Storeys	0.2 Spaces	1.2	2.2	3.2	voxel
APC	✓	✕	✕	✕	✕	✓
Gaia	✕	✕	✕	✕	✓	✓
Lisbon	✓	✓	✓	✓	✓	✓
Praha	✕	✓	✓	✓	✓	✓

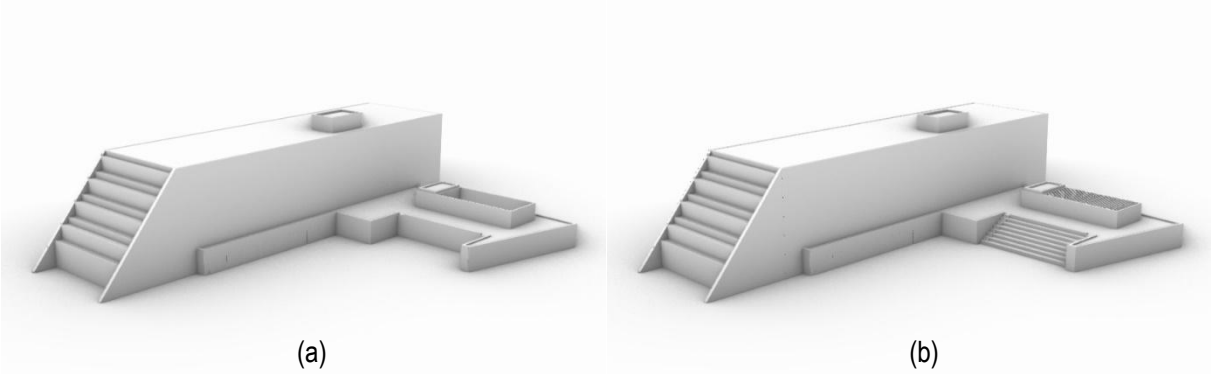


Figure 19: The difference between the LoD2.2 abstraction utilizing the IfcBuildingElementProxy. (a) shows the model without the shapes used. (b) show the model with the shapes used. If IfcBuildingElementProxy objects are used the model has more detail that could be considered important.

The quality of the inner shell abstractions generated by the IfcEnvelopeExtractor are shown in Table 5 . It shows that for none of the models a full successful interior output has been created. However, this is expected. The APC model includes no IfcSpace objects, and the Praha model includes no IfcBuildingStorey objects, see Table 3. As mentioned in 3.2, the tool requires those two object types in the input in order to create space and storey abstractions. So, the missing abstractions are expected and do not point to any unexpected failings of the tool itself. It could be reasoned however that if IfcSpace objects are missing in the IFC file, the tool should be able to generate spaces based on the space dividing objects. This is a topic for further development.

The Gaia model does include both IfcSpace and IfcBuildingStorey objects, but the tool was still unable to process the spaces in most instances. This is related to the earlier mentioned curved surfaces. The interior and exterior abstraction steps share subprocesses, so if the exterior export fails due to the presence of curved surfaces the interior export will as well. The LoD3.2 export is successful because for the interior export, this is a 1:1 translation of the shapes, so they are not processed in any way. This means that the presence of the curved corners does not influence any geometric operations.

Table 6 conversion statistics with the voxelisation export excluded.

Model Name	IFC file size (kb)	IFC polygon count	JSON File size (kb)	JSON polygon count	File Size reduction	Polygon count Reduction	Computing Time (min)
APC	5.828	135.802	305	20.619	94.7%	84.8%	4
Gaia	2.175	40.662	102*	7.643*	95.3%*	81.2%*	<1
Lisbon	14.912	317.464	209	13.052	98.6%	95.9%	3
Praha	42.452	3.670.796	631	32.594	98.5%	99.1%	6

A quantitative comparison between the source IFC files and their converted CityJSON counterparts can be seen in Table 6. This table excludes the voxelised output. This was done because the chosen voxel size can arbitrarily influence the polygon count and the file size, and this is up to the user's discretion. This makes it hard to make an objective comparison if they were included. Additionally, both IFC and CityJSON files do not usually consist of only triangulated

polygons. However, to be able to directly compare the geometric complexity, the geometry in both the IFC files and the CityJSON files has been triangulated.

Table 6 shows that the IfcEnvelopeExtractor drastically reduces both the geometry's complexity and the file sizes. The average file size reduction is 96.8% and the average polygon complexity reduction is 90.3%. This is not just comparing a single model file to another single model file. The CityJSON file includes the building's LoD0.0, 0.2, 0.3 1.0, 1.2, 1.3, 2.2 and 3.2 exterior abstractions and the LoD0.2, 1.2, 2.2 and 3.2 interior abstractions. Therefore, comparing the complexity of a single model CityJSON file to the IFC file will display even further reductions in file size and complexity.

The time to compute these shapes is also fast. Possibly, a human would be able to construct a LoD1.3 or 2.2 abstraction by hand in this time. But this would only be possible if the human was an experienced 3D modeller and/or if the model has a simple roofing structure. However, complex models would take a longer time to process by hand than the time the tool takes to process 8 different abstractions. These computing times were benchmarked on a consumer laptop (Intel i7-8750H processor). A more powerful system will be able to process these files at an even faster rate.

In its current state, the tool does not have a 100% success rate on every model. This means that the generated LoD abstractions should be validated afterwards by humans. However, the tool should function well in a supporting role, where it can execute an initial abstraction which is then refined by a human user. This will cut down on the processing time compared to a manual abstraction process and likely generate geometries closer to the source IFC model. In future work, the processes will also be refined based on the results presented in this report and overall experiences in CHEK.

5. Conclusion

To support the integration of BIM and GIS, the IfcEnvelopeExtractor has been developed. This is a software application that automatically converts IFC files to format compliant CityJSON files that can be used for further analysis. The tool can abstract the models according to the extended LoD standard developed by the TU Delft in 2016. It utilizes a combination of point cloud, voxel and ray casting processing to create the abstracted shapes. For exterior export, the available abstractions are LoD0.0, 0.2, 0.3, 1.0, 1.2, 1.3, 2.2, and 3.2. For interior processing, these are LoD0.2, 1.2, 2.2, and 3.2. Additionally, the tool can also output a voxelised shape. The IfcEnvelopeExtractor has been developed to be used on BIM models made by both BIM professionals and BIM non-experts and can thus function on IFC models of poorer quality. However, it is recommended that certain rules are adhered to regarding the input IFC so that geometric computations can be executed accurately.

The performance of the IfcEnvelopeExtractor is generally good. It can convert an input model to valid LoD abstractions in a time that is only potentially matched by experienced 3D modellers when doing it manually and only in models with a simple roofing structure. Some current limitations of the tool were found when the BIM mode contained curved surfaces. These limitations had an effect on the accuracy of the mid and high-level processes. Additionally, the tool is unable to reconstruct missing storey and interior space data. If the BIM model does not include this data, the data will also be missing in the output file. Due to these limitations, it is recommended to validate the output of the tool by hand to avoid errors in downstream processing.

The automatic conversion from BIM to GIS provides a practical, quick and scalable solution to facilitate the evaluation of buildings at a city level. This will optimize the design and digital building permit (DBP) process. During the design process the ability to automatically convert the draft BIM models to GIS enables GIS applications to be used as a design tool to incorporate environmental effects more effectively early in the design process. The improved workflow is however not limited to a validation/design tool for a building/structure. It can also be used to update existing GIS city models with more accurate BIM based information. Or it can be used to populate a GIS city model in case that no GIS model is available (e.g. because it has not been surveyed yet). This sort of integration will improve subsequent urban planning and infrastructure management.

The next steps in the development of this tool are enhancing its reliability and computing speed. Additionally, a set of improvements will be made based on the experiences of CHEK. Firstly, the output of LoD3.2 will be refined. In its current state, the tool creates a collection of surfaces that are part of the outer shell, but a closed outer shell is not created. Secondly, support for curved surfaces will be added. And thirdly, a way to validate or to extract the interior spaces without relying on the stored input BIM spaces will be developed.

6. References

6.1 List of Figures

6.2 List of Tables

6.3 List of used abbreviations

BIM	-	Building Information Modelling
DBP	-	Digital Building Permit
GIS	-	Geographic Information System
GUI	-	Graphical User Interface
IFC	-	Industry Foundation Classes
LoD	-	Level of Detail
WP	-	Work Package

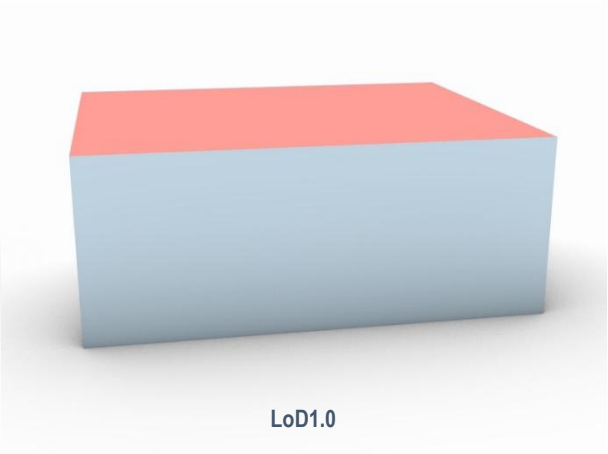
6.4 Glossary

BIM (Building Information Modelling)	-	The process of modelling and managing information related to a building or construction. Often executed at a building/architecture scale.
Boolean Operations	-	Operations that create composite geometry by forming unions, differences or intersections from two or more different input geometries
Bounding Box	-	A rectangular box that fully encloses a shape or group of shapes
CityGML (data model)	-	Standardised data model for GIS data storage
CityGML (encoding)	-	XML encoding of the CityGML data model
CityJSON	-	JSON encoding of the CityGML data model
Extrusion	-	Moving a 2D cross section along a path or direction to create 3D geometry
GIS (Geographic Information System)	-	System to store, process and manage geographic information
IFC (Industry Foundation Classes)	-	Open standard for storing BIM data
LiDAR	-	Acronym for “light detection and ranging”. A way of constructing a point by targeting an object with a laser and measuring the time it takes for a laser to bounce back from the object. Repeating this process while targeting from different angles or locations will result in a point cloud of the object
LoD (Level of Detail)	-	Term describing the amount of abstraction compared to the original data. Originated from the computer graphics field where this is a very flexible phenomenon. In GIS the abstraction shapes are restricted and adhere to set rules
OCCT (Open CASCADE Technology)	-	C++ 3D Modelling toolkit developed and maintained by Open Cascade SAS/ Capgemini
Point Cloud	-	A collection of data points in 3D space
Ray Casting	-	The process of creating a ray from a source to a target point. This ray can be checked for intersections with geometry
Semantic Objects	-	A JSON object representing the semantics of primitive of a geometry in the CityJSON encoding. A semantic object holds all the attributes of the primitive(s) it is related to
Triangulated Polygons	-	The partition of polygons into triangles
Voxelization	-	converting geometric data to a rasterized collection. In this document in a 3d rasterized grid of cube shapes

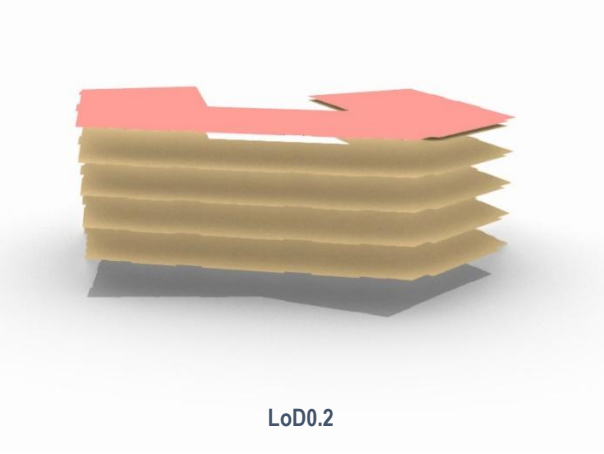
7. Appendix

7.1 Full results of the APC processing

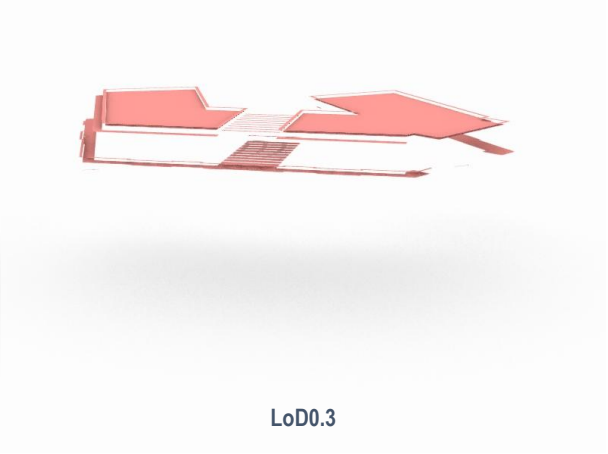
Source IFC model



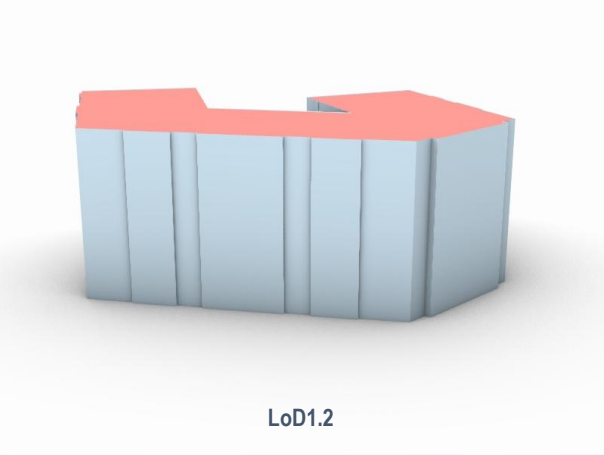
LoD1.0



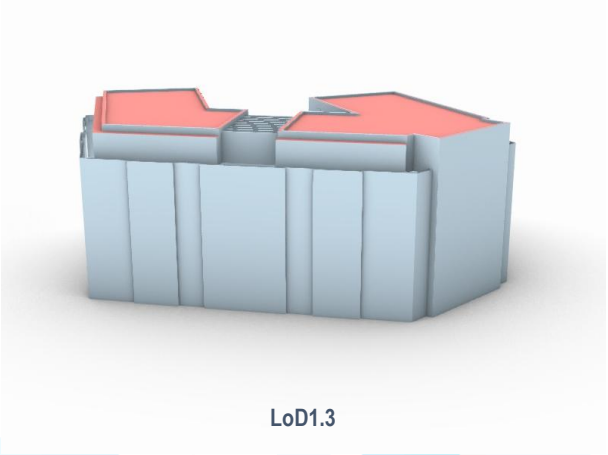
LoD0.2



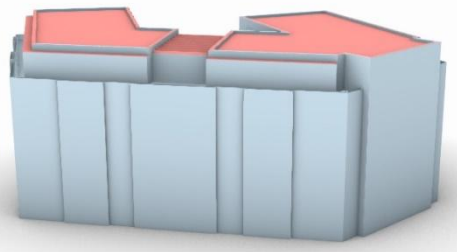
LoD0.3



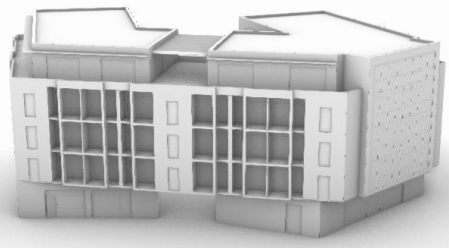
LoD1.2



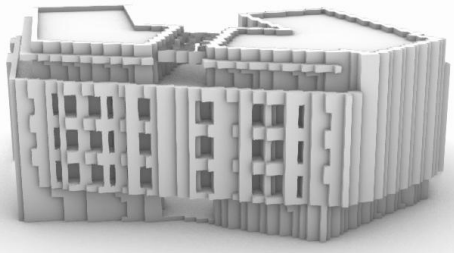
LoD1.3



LoD2.2



LoD3.2



Voxel

No geometry

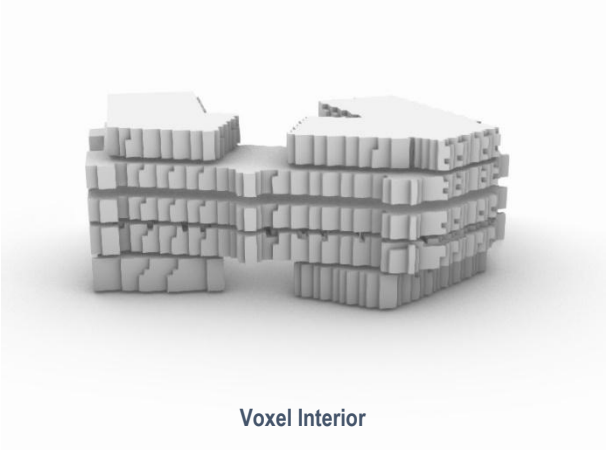
LoD1.2 Interior

No geometry

LoD2.2 Interior

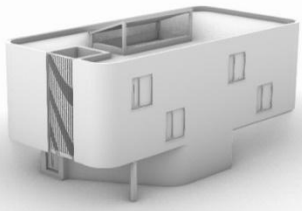
No geometry

LoD3.2 Interior

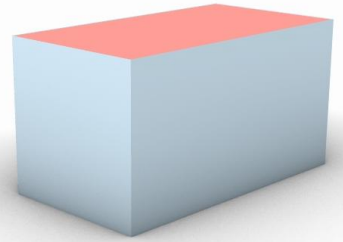


Voxel Interior

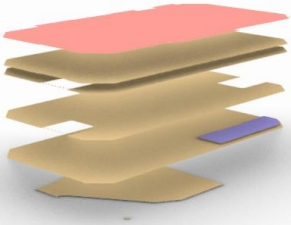
7.2 Full results of the Gaia processing



Source IFC model

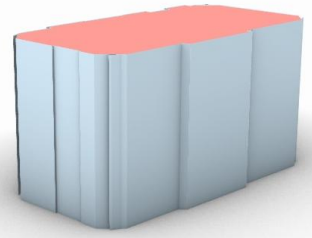


LoD1.0

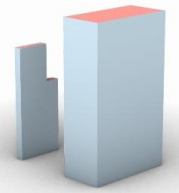


LoD0.2 (incorrect)

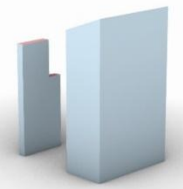
LoD0.3 (incorrect)



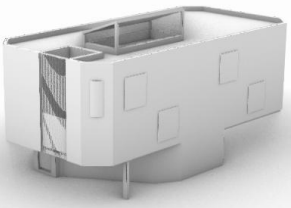
LoD1.2 (incorrect)



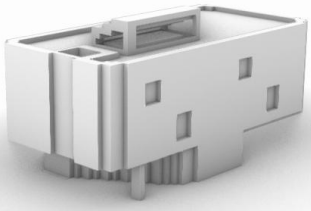
LoD1.3 (incorrect)



LoD2.2 (incorrect)



LoD3.2 (incorrect)



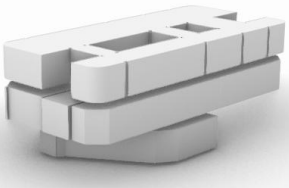
Voxel

No geometry

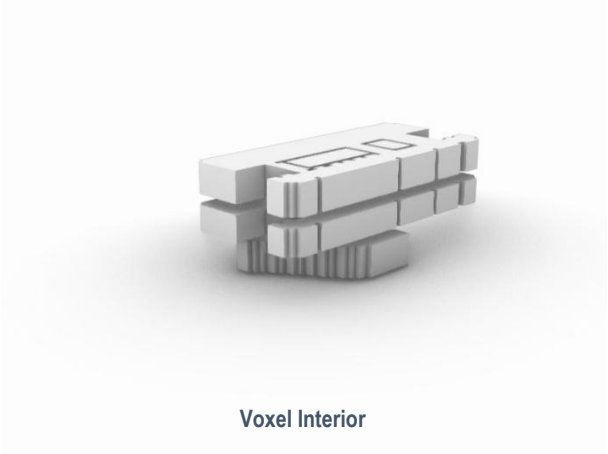
LoD1.2 Interior (incorrect)

No geometry

LoD2.2 Interior (incorrect)



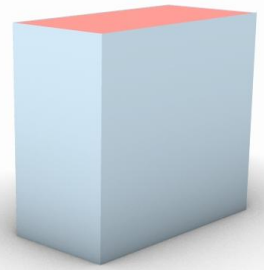
LoD3.2 Interior



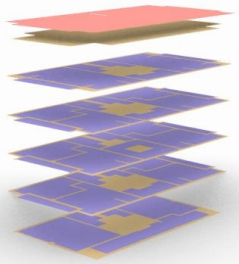
Voxel Interior

7.3 Full results of the Lisbon processing

Source IFC model

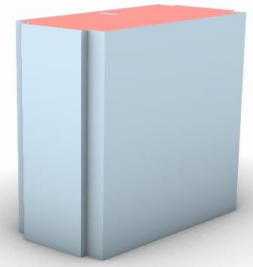


LoD1.0

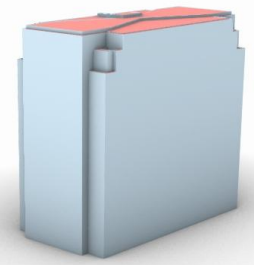


LoD0.2

LoD0.3



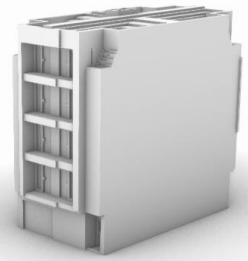
LoD1.2



LoD1.3 (incorrect)

LoD2.2

LoD3.2



Voxel

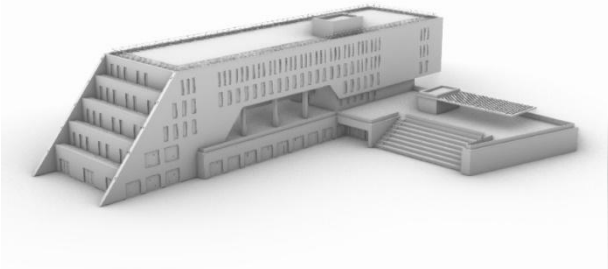
LoD1.2 Interior

LoD2.2 Interior

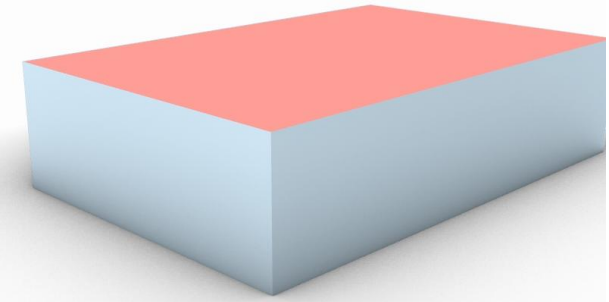
LoD3.2 Interior

Voxel Interior

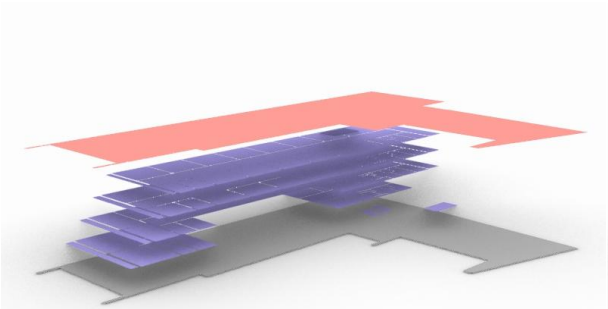
7.4 Full results of the Praha processing



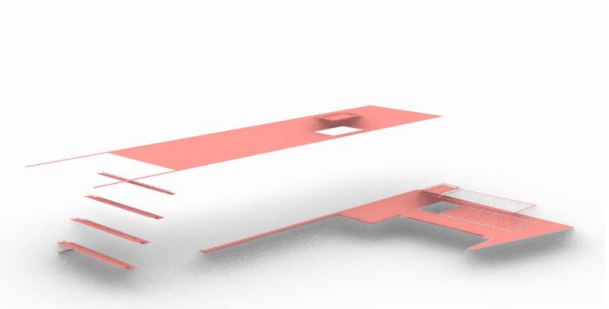
Source IFC model



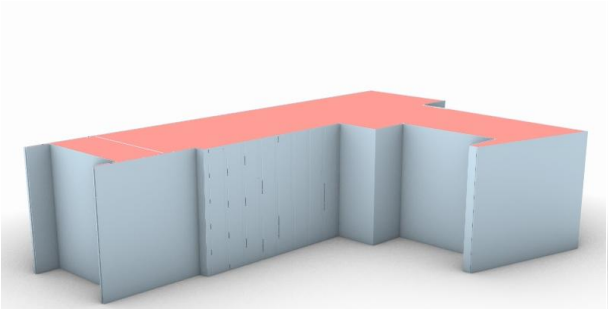
LoD1.0



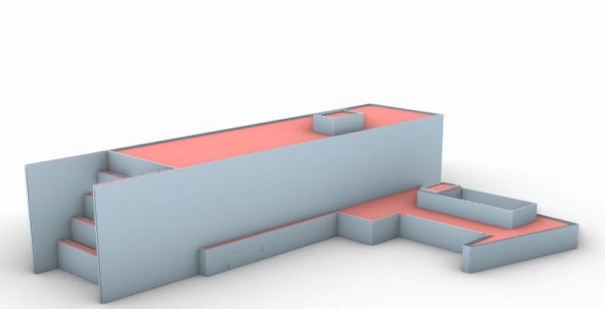
LoD0.2



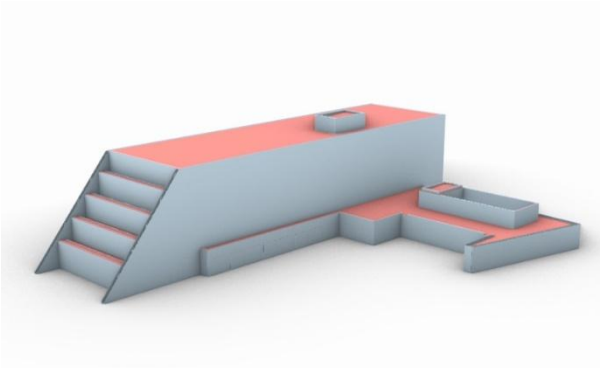
LoD0.3



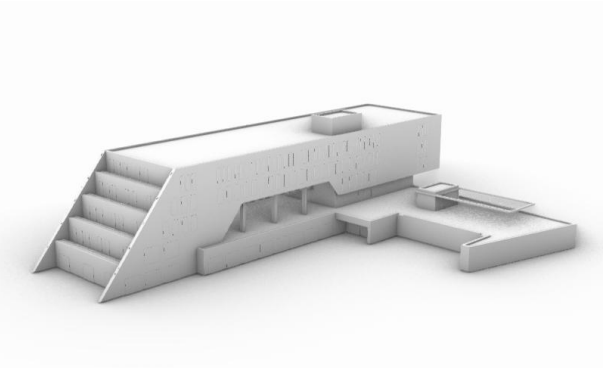
LoD1.2



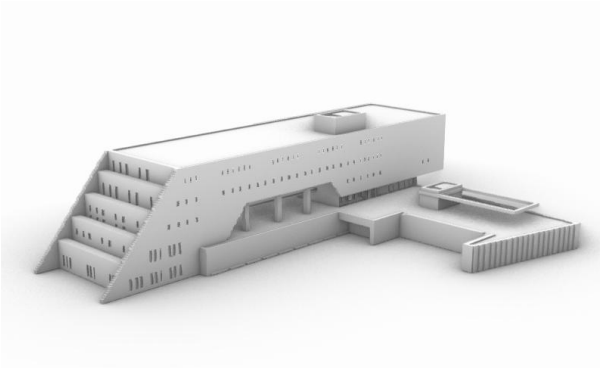
LoD1.3



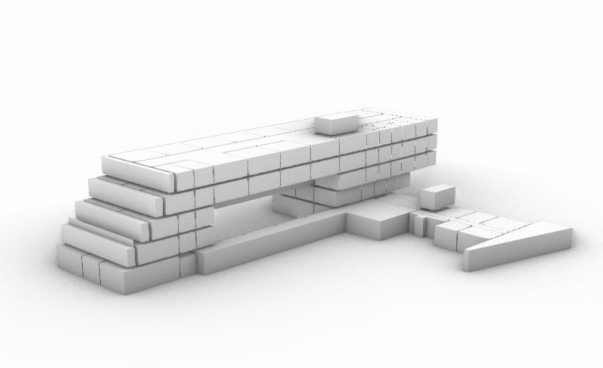
LoD2.2



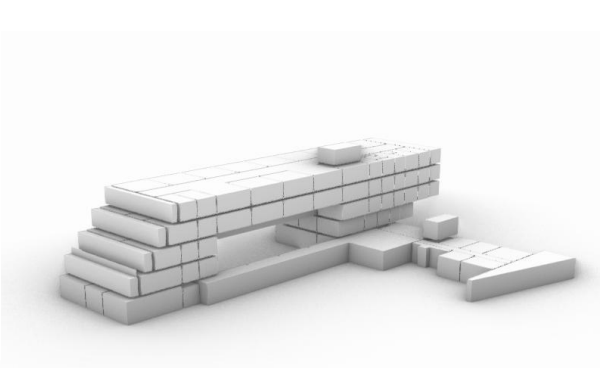
LoD3.2



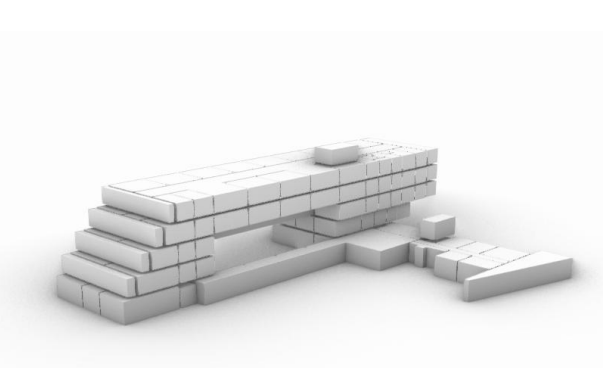
Voxel



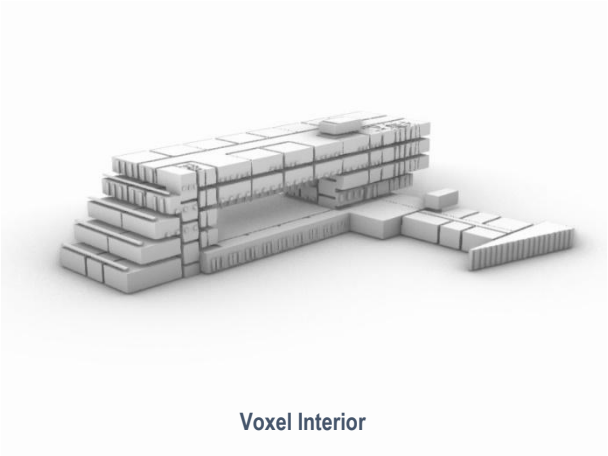
LoD1.2 Interior



LoD2.2 Interior



LoD3.2 Interior



Voxel Interior