Semi-automatic data transfer from Revit to BSim

For rapid and consistent modelling of indoor simulations

Building Energy Design

Master Thesis

Arturas Pranskunas

AALBORG UNIVERSITET

2019

AALBORG UNIVERSITY
STUDENT REPORT

Study Board of Civil Engineering
Thomas Manns Vej 23
DK - 9220 Aalborg Øst
Tlf. 99 40 93 09
www.aau.dk

Title:

Semi-automatic data transfer from Revit to BSim

Study programme:

Master of Science (MSc) in Technology in Building Energy Design

Project:

Master's Thesis

Project period:

September 2019 - January 2020

Supervisors:

Rasmus Lund Jensen

Nanna Dyrup Svane

Participant:

Arturas Pranskunas

Date of completion:10/01/2020

Number of copies: 1

Number of pages: 137

Synopsis:

This report investigates how the manual modelling practices with BSim could be improved by automating repetitive and time-consuming tasks to produce energy performance models. To achieve this goal, the author of this report scrutinizes the developed methodology by MOE engineering company, to develop a new practice that could be used to obtain the essential data for Danish BPS tool – BSim.

Furthermore, the developed semi-automatic methodology is based on scripts designed for Dynamo to translate and evaluate Autodesk Revit architectural model. Correspondingly, the data from Dynamo is exported in Microsoft Excel where supplementary model parameters are assigned, and exported for further data treatment outside the Dynamo environment.

The report main focus is on geometry translation methods in Dynamo environment that could be applicable for a wide range of architectural models. To do so, various approaches are used to translate and build BSim model. Particularly, a combination of Dynamo/Phyton scripts and external .NET Framework application was developed to accommodate information extraction, evaluation, parameter assignment and model build procedures.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author

Acknowledgements

Any research can be executed properly, without the presence of a second opinion. This why, I would like to express my gratitude to my supervisors Rasmus Lund Jensen and Nanna Dyrup Svane who helped me with this Master thesis. I would like to Thank You both for your valuable ideas, guidance, and patience whenever I was facing a tough challenge, and I had to make some difficult decisions.

Secondly, I would like to thank you Steffen Enersen Maagaard from MOE who spent his valuable time with me and provided their BIM tools with all encapsulated knowledge in it. This methodology was very helpful along the way of my thesis project.

Third, Kim Trangbæk Jønsson is definitely an expert with regards to BSim file structure, I am very appreciative of his participation at the beginning of my project, especially when all of it was completely new for me.

Moreover, I would also express my appreciation to my former colleague Stanislava Georgieva Raykova from 2nd and 3rd semester for commenting on my report drafts.

Finally, I would like to express my appreciation to my family for their understanding and endless support during this semester while I was busy with my thesis.

Abstract

This report investigates how manual modelling practices with BSim could be improved by automating repetitive and time-consuming tasks to produce energy performance models. To achieve this goal, the author of this report scrutinizes the developed methodology by MOE engineering company. This developed methodology is applied for heat loss calculation by using Dynamo for Revit and Microsoft excel. This is done in order to export and produce the required documentation. A Similar methodology is defined to translate architectural building models from Autodesk Revit to Danish BPS tool – BSim.

To begin with, a thorough analysis of the BSim dis (XML) file structure aimed to find how the geometry and the HVAC system data is structured is this file container. The analyses have been performed for a simple two-room model with a minimal amount of inputs. The conducted research on this building pointed out, that the model is formed by hierarchical tree relations between XML elements. Moreover, such elements are referenced by rid attributes. The correct rid attribute reference is vital for BSim and any wrong or duplicate assignment of it will cause BSim to crash on the process.

Furthermore, analyses were performed to define data collection structure which could be used to automate BSim dis (XML) file build. The mentioned data collections were identified by taking manual data modelling approach. A simple model in Autodesk Revit was measured with the annotation toolsets to obtain the required vertex coordinates. The outcome is that the model in SimView requires external surface data points. The investigation revealed that data collections could be simplified as BSim has a feature to determine inward-facing normal and is able to write missing elements such as normal vector elements and finish elements. However, to let BSim alter dis (XML) is not a decent practice, as it results in missing references.

To obtain the essential data from Autodesk Revit, the developed heat loss methodology by the MOE consulting company was used as a base to model space solids. Furthermore, this methodology was tested and identified, as the most suitable approach for obtaining the required SimView data frame model. The investigation revealed that solids have a close representation of the SimView wire-frame model. However, it turned that solids in the Dynamo environment are represented differently and do not have any relation to each other and must be modified.

Moreover, to define the necessary alterations to the base solid geometry, the studies were performed to the case house. Accordingly, the analysed base space solids resulted in overlapping geometries between two rooms that have complex boundary curve intersections. The studies revealed that the best practice would be to deconstruct space boundary curves and re-model space solids from individual boundary segments. However, such methodology will

require engineer intervention in the model generation process, as there are multiple possible situations of how the solids could be interpreted.

Similarly, like solids window/door entities do not have any relation to the altered space solids and not necessarily correspond to visual representation in Autodesk Revit. The studies pointed out that translated window/door openings with small approximations can be hosted on an appropriate host. However, an incorrect window and door family design will have a negative impact on the opening representation. Especially, such inaccuracies are hard to detect it pragmatically and can be only evaluated after the model build is complete.

Finally, it was observed that additional construction and HVAC system setup templates could be made in Microsoft Excel and linked in the final BSim model.

Abbreviations

A

API - Application programming interface, 3

B

BIM - Building Information Modeling, III, 1, 2, 3, 5, 35, 59, 60, 89
BPS - Building Performance Simulations, IV, 1, 2, 5, 55
BSim - Software name -Building Simulation, I, II, III, IV, V, VII, VIII, IX, X, XI, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 20, 21, 23, 24, 25, 26, 27, 29, 30, 31, 32, 34, 35, 36, 40, 43, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 62, 63, 66, 80, 84, 88, 89, 90, 93

C

CAD - Computer-aided design, 3, 8

D

DRY - The **Danish** Design **Reference Year**, 15

E

EU - European Union, 1

G

gbXML - Green Building XML, 1, 3, 59

H

HVAC - Heating, ventilation, and air conditioning, IV, V, 1, 2, 3, 5, 6, 12, 13, 32, 55, 57, 58

I

IFC - The Industry Foundation Classes, 1, 3, 59

M

Microsoft Access Database, 11, 13, 52, 57

P

PV - Photovoltaic, 8

R

Rid -Raw id attribute used to reference BSim hierarchical tree elements., IV, IX, 12, 17, 22, 24, 25, 26, 27, 51, 56

S

SGML - Standard Generalized Markup Language, 11

V

VBA - Visual Basic for Applications, 24

W

Windoor - Window and door opening in SimView, IX, XII, 20, 22, 23, 24, 26, 43, 45, 46, 56, 58, 62, 79, 80, 83, 84, 92

X

XML - eXtensible Markup Language, IV, VII, IX, XI, XIV, 3, 6, 7, 8, 10, 11, 12, 13, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 32, 44, 45, 48, 49, 50, 56, 58, 59, 60, 61, 62, 66, 67
XSD - XML Schema Definition, XIV, 12

Table of contents

List of Figures

List of Tables

Table of Listings

1 Introduction

Over the 20th century, atmospheric concentrations of the key greenhouse gases increased due to human activities. [1] There are many sources arguing about how much time is left until untraversable damage to the ecosystem is done. However, it is clear that by the end of this century the humankind has to find a carbon-neutral solution to satisfy its growing energy demands.

The EU is committed to ambitious climate policy. Its current target is to reduce greenhouse gas emissions 40% by 2030 compared to 1990 levels. [2] To fulfil such ambitions plans not just the fossil plants have to be discarded, but it requires to look for solutions, how to optimize energy consumption as well. As the buildings are responsible for approximately 40% of energy consumption in the EU, it makes them the largest energy consumer. [3] Therefore, there is a large room for improvement to reduce their energy demand.

Over the past 50 years, more than hundreds of building energy simulation tools have been developed. Such as BSim, BLAST, DeST, EnergyPlus, IES-VE, Energy-10, etc. These programs are designed to perform energy performance tests with various key aspects such as IEQ, atmospheric comfort, energy use, and heating demand. [4]

Each individual tool mentioned above this paragraph is better in some aspects than another. However, all those computer-aided tools do not necessarily apply for specific local standards. The Danish building performance (BPS) tool BSim is the only one that is adapted to the local standards for geometric measurements. Unfortunately, this tool lacks the capability to import models designed in another very popular drawing tool like Autodesk Revit.

In contrast, large BIM tools allow exporting building models in formats, such as IFC, gbXML. Unfortunately, the exported data cannot be used straight out the box, as in most cases it is inconsistent and strongly dependent on the modelling practices. For example, each architect has his own drawing style which has an impact on the overall model data representation. For this reason, in most cases models could not be used for energy performance simulations and must be adjusted for particular needs.

Traditionally, building simulation and the analysis starts only after an architectural and HVAC design phases are detailed enough and the extracted information is sufficient for the beginning of BPS model construction. For this reason, building simulations and analyses do not start in early design phases until major decisions are completed. Such decisions, potentially, might be critical to the energy performance of the future building. [5]

All in all, due to the never perfect interoperability and inconsistent modelling, the performance analyses are often performed with manual modelling, which causes late start and delivery of relevant analyses as well as ineffective use of resources.

1.1 Current modelling practices

At the present time, the models with BSim are made entirely from scratch. This is a very laborious and time-consuming process as all inputs have to be prepared and manually inserted in the simulation program. Furthermore, this work mostly consists of manual input reproduction of already existing information. Correspondingly, the manual data inputs often result in a human error which sometimes could be very difficult to detect. [5]

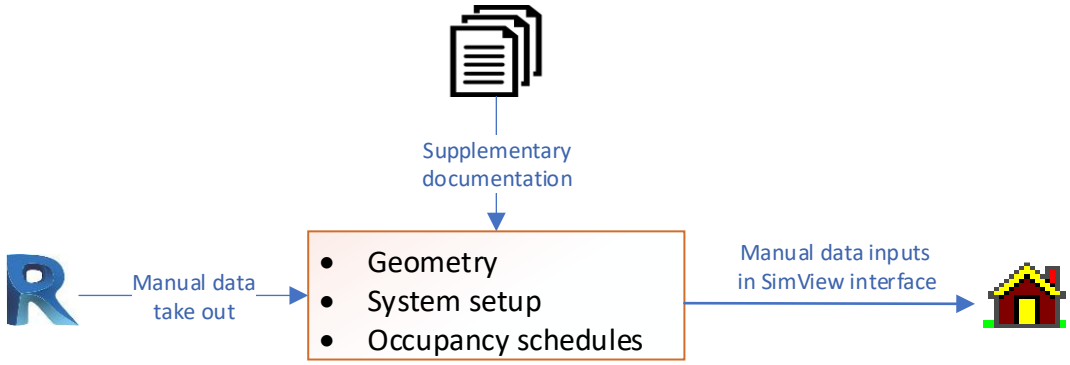

Figure 1-1 Current modelling practice with BSim

Figure 1-1 shows the manual data transfer methodology for BSim that is used until now. This type of methodology could be split into a few parts, such as model geometry, HVAC system setup, occupancy schedules, and other building data.

In the current practice, the building geometry is simplified by an engineer according to his or hers understanding, experience, skill, and knowledge. [5] In many cases due to lack of the resources and time constraints, an engineer aims to look for the shortcuts which sometimes results in even larger scale simplifications and assumptions that are made for the BPS models.

At the moment, there are only the construction definition database and the material definition database which could be reused in many projects by attaching it to a new project, while the rest of the information like, HVAC key parameters, system operation schedules, and occupancy schedules have to be set up manually for each new model. As a result, the information is typically set up from the supplementary documentation, such as pdf documents, excel documents or other specifically dedicated BIM tools that are responsible to store building-related information.

1.2 Data export possibilities

Another popular tool like Energy plus has not a single method to transfer building geometry data from Autodesk Revit to Energy plus. This can be achieved either by performing energy analysis directly in Autodesk Revit with the integrated energy plus engine or the 3rd party programs can be used to transfer building geometry data. [6]

As it was mentioned earlier in this chapter the BSim engine does not have any tools yet developed which would allow transferring building geometry data or HVAC system definitions from Autodesk Revit.

There are a few data exchange solutions that can be distinguished and used in cooperation with the BSim:

- Integration of simulation engine – tsbi in the BIM tool or direct coupling with Revit (API) [6]
- Export of the relevant information from the BIM tool to a file using the gbXML format and the subsequent import of that file into the simulation software. [6]
- Export of the entire BIM (or preferably the relevant parts of it) to the Industry Foundation Classes (IFC) format [6]
- Dynamo – visual programming tool that replaces conventional elaborate coding by visual sets of blocks of independent functionalities. [7]

The IFC schema manages to export the building's geometry data into a very compact and precise way that accommodates multiple geometry representations in one file. However, in some cases, the IFC data model classes contain an incorrect representation of data as the designer wrongly defined some elements or BIM authoring tool has flaws to handle some particular geometries. [8]

Unlike the IFC schema, the green-building XML schema (gbXML) is specifically designed to facilitate the transfer between CAD-based building information models, enabling intermobility between engineering analysis software tools. [9]. However, gbXML is not perfect in all cases. Studies indicate issues with the gbXML conversion, as some errors occur with incorrect shading surface definitions and omissions of some walls. [6] In addition, the gbXML produces inner space surface models. For this reason, models include empty gaps between rooms where partitions supposed to be located. Due to BSim measurement logic, such model representation would need to be heavily modified to eliminate any empty voids between related rooms.

For several years Engineer consultancy company – MOE was looking for a way to optimize and automate daily tasks with the help of the BIM technologies. After research, the company settled down with empowering Dynamo core for daily engineer tasks which would allow an engineer to select the most optimal solutions - economical and energy-wise.

Therefore, through the years they developed the Building Design tools that include: summer comfort simulations, daylight evaluations, data transfer to Building Energy 2018 – BE18 program.

As company representatives say: the use of the Dynamo eliminates a need to code all geometry treatment methods from scratch which would take a lot of time and resources. In addition, Dynamo has built-in visual tools that allow us to inspect any geometry change visually and take further actions to treat the related issues manually.

Because of aforesaid reasons, it was preferred to use Dynamo and the semi-automatic data transfer methodology developed by MOE for obtaining the required data from Autodesk Revit and transforming spaces to be suitable for the BSim simulation tool.

1.3 Problem statement

At the moment, in BSim building energy analysis or investigation of the critical spaces in the building is performed manually. In the current practice, as automated tools do not exist. The BSim modelling is performed by taking an architectural model, manually creating the topology model with HVAC system setups. Finally, the results are interpreted and suggestion documentation is prepared.

It is obvious that the exchange link between architectural and the BPS tool is missing which would allow to export and reflect simulated results in the BIM authoring tool. The main problem is that in Denmark the most popular BIM tool like Autodesk Revit does not have any developed tool or methodology which would allow exporting building topology model to BSim. In addition, modelling with the BSim includes a need for the detailed HVAC system setup. In many cases such setups are repetitive, and often could be adapted from the projects having similar requirements. This process is very slow and requires a lot of man-hours to produce a new model throughout building design phases. Especially, if several proposals are needed to be made or the building space geometry changed drastically. These restrictions cause a negative impact on the overall key decisions made with regard to energy performance. Moreover, the early made decisions might be very critical and could be very expensive to fix it, as soon as the building is ready for use.

Preferably, the best solution would be to develop a tool that would allow us to export the required information from Autodesk Revit and assemble a new BSim model with the predefined model parameters from an external database. Such a tool would allow to rapidly export models and assign adapted HVAC system templates. Furthermore, the results from the simulation could be imported back into Autodesk Revit in a form of schedules that could be used to give a visual representation on the floor plans.

The author of this thesis main focus is on defining the methodology in Dynamo that could be used to export building geometry data by selecting the required spaces in Autodesk Revit.

In addition, the author of this paper focuses on defining a methodology to improve the current model parameter setup logic, such as HVAC systems, people load, equipment load, construction assignment, site information.

The main problem formulation of this thesis is as follows:

How efficiently exported data from Autodesk Revit could facilitate to improve current modelling practices and allow energy performance simulations on earlier stages of the design phase?

In order to answer the main problem statement, the necessary work can be broken down into sub-tasks stated below this paragraph:

- Replicating the BSim dis (XML) structure.
- Develop a methodology in Dynamo to select spaces in Revit to export required space data.
- Transform space data to be suitable for the SimView topology model.
- Develop the methodology to define building parameters from supplementary databases.
- Construct BSim models automatically based on user-defined spaces in Revit.

1.4 Methodology

As this report aims to give answers to the model transfer problems with have gained interest over the past few years by the research community and the engineering companies, but still remains unresolved up until now. In addition, as this report uses the methodology and the tools from MOE company, the logical way is to start an investigation by review existing work and defining if these tools can be used for the model transfer.

The next step is to perform experimental tests on the Dynamo scripts and define the necessary alterations for the Autodesk Revit space model that would allow constructing valid SimView topology models. Equally important is to investigate the BSim dis (XML) file structure and identify XML hierarchical tree relations and its element requirements.

Moreover, it is important to define the data export steps that could be fully automated and the parts where the semi-automatic approach should be taken.

Lastly, define the optimal methodology for space and window/door data modelling. In addition, define the methodology for construction and HVAC system modelling for the BSim model.

1.5 Delimitation

The research in this report is focused on extracting the essential topology model data from Autodesk Revit and transferring it to BSim. Therefore, the investigation mainly focuses on the dis (XML) elements that are related to the BSim wire-frame model. In addition, another dis (XML) elements that belong to the system or the general data family are not discussed in depth. However, the report briefly specifies a reference between geometry and the system elements in the dis (XML) file, as well as proposes a solution to how the HVAC system information could be linked to the developed methodology in this report.

1.6 Structure of the thesis

This report is subdivided into the chapters where the first chapter of the report introduces with the more general knowledge and the following chapters will go more in detail with regards to a specific topic.

Each chapter in the report is made in the way of the text explanation and the graphical representation of a particular topic.

Moreover, in the paragraphs mentioned dis (XML) file elements are written in an *italic* style, so it would be easier for the reader to navigate and interpret the related figures, listings, and tables in this report.

2 BSim suite

BSim is a computer tool that offers advanced simulation facilities for building designers, engineers, architects, and other engaged parties. In analysing hygrothermal conditions, when planning, designing or analysing energy consumption and indoor climate in connection with the design for almost any type of buildings. [10]

This software has been used extensively over the past 20 years, previously under the name of tsbi3. Nowadays, BSim is the most commonly used tool in Denmark, and with increasing interest abroad, for energy design of buildings and for moisture analysis. [11] [10]

As the aim of this master thesis is to find a way of consistent information extraction from Revit to BSim. Therefore, it is crucial to define BSim software and its components. This chapter will focus on defining the BSim suite, requirements for model geometry and dis (XML) file structure.

2.1 BSim program modules

To begin with, the BSim suite consists of numerous separate modules that are responsible for certain tasks such as analysis of incident solar radiation, thermal simulation, daylight simulation, natural ventilation simulation, electrical yield analysis from PV cells, model editor, building environmental analysis and import of CAD drawings. [12] Each module and its purpose is presented in the bulleted list below this paragraph:

- SimView - Model editor. (3)
- XSun – Tool for analysis of incident solar radiation. (4)
- Tsbi5 - Thermal building simulations. (5)
- SimLight - Daylight calculations. (3)
- SimDXF - Tool for import of CAD drawings.
- SimPV - Calculation of the electrical yield from a building-integrated solar cell (PV) system.
- BEAT - Building Environmental Analyses Tool. (1)

2.1.1 SimView interface

The BSim program modules are built around SimView which is a central program, equipped with a user interface and model editor. Other modules are accessed through SimView tool-set that is presented in Figure 2-1.

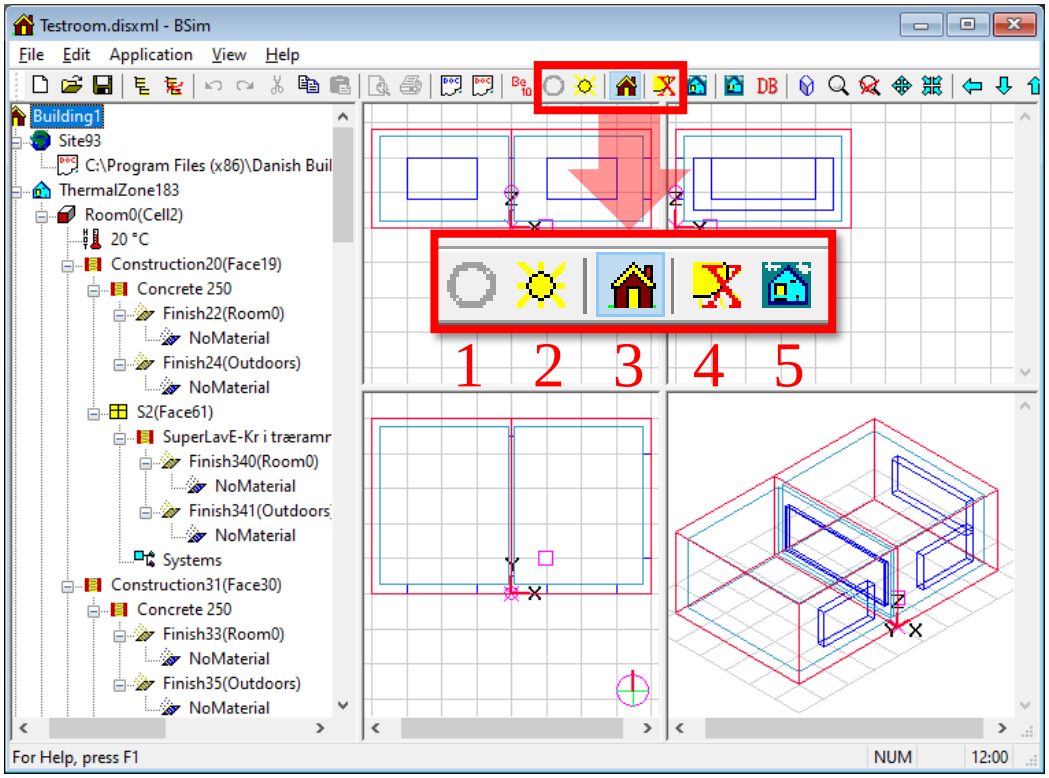

Figure 2-1 BSim user interface (1-BEAT, 2-Daylight, 3- SimView, 4- XSun, 5- tsbi5)

In the example, presented above the building model is showed in the form of a hierarchical tree summary and the graphical representation of 4 views, such as floor plan, two elevations and perspective 3D view [12]. Firstly, a building model in the SimView is created in the spatial coordinate system (X, Y, Z) in which X-axis has positive value towards the east, Y-axis has positive value towards the north and Z-axis has positive value pointing up. Simultaneously, with the geometrical model creation, the hierarchical tree is created in which the geometrical model can be enriched with the detailed building data. This data amount and requirements vary from model to model that depends on simulation purpose.

The following example, in Figure 2-2 is focused to illustrate the assembly of nodes in the hierarchical tree rather than nodes that are required for specific simulation.

Because of readability, duplicated nodes were removed from this example, it is worth to note that each *system* includes sub-nodes - *control/schedule* where control and the schedule are defined for the individual system unit. One example of *Cooling (Cooling348)* sub-node can be observed in Figure 2-2.

In essence, this hierarchical tree representation of the model in Figure 2-2 is closely related to the BSim file structure – dis (XML), that has to be assembled to be able to perform simulations in BSim. Those hierarchical tree nodes related to the file structure will be particularly covered in Chapter 2.2

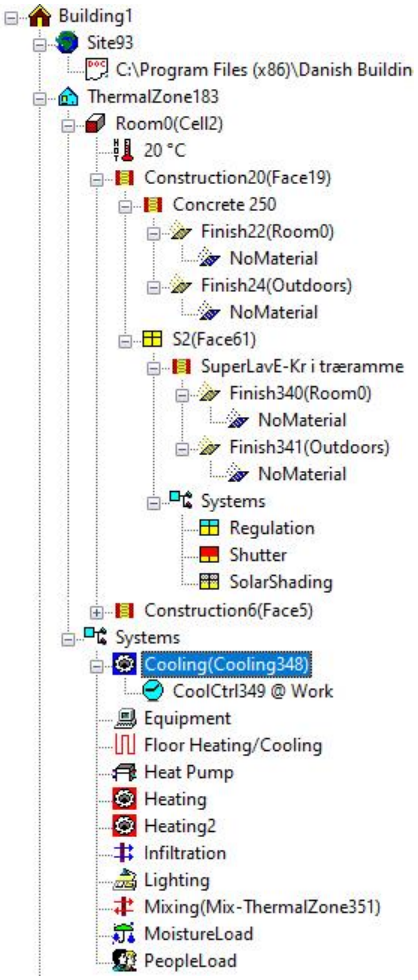

Figure 2-2 Hierarchical tree example

2.2 BSim input files

BSim simulation software input is divided into 3 separate files such as dis (XML), database (mdb) and weather database (dry). Each mentioned file is responsible for holding certain information that is required to perform simulations. In the following sub-chapters: 2.2.1; 2.2.2 and 2.2.3 these files and information it holds will be presented in detail.

2.2.1 XML file definition

Firstly, before looking into the contents of the BSim dis (XML) it is necessary, briefly to define: What XML document is, and how it is formed? The XML documents can be defined as eXtensible Markup Language. It is a simple and very flexible file format that is derived from SGML (ISO 8879). [13] This file type was designed with a goal to carry data across different platforms. All things considered, the most important fact is that this type of file format stores data in the plain text format. For this reason, such files can be read and effortlessly understood by humans. In addition, XML can be easily expanded or upgraded for new applications. [14]

XML documents are formed as a tree structure such structure starts at “the root”- element and trough branches go to “the leaves”- child elements [14]. Parent elements can have as many child elements as it is required. The child elements between each other will form a sibling’s relationship. In Figure 2-3 relationships between the root element and child elements can be examined in detail.

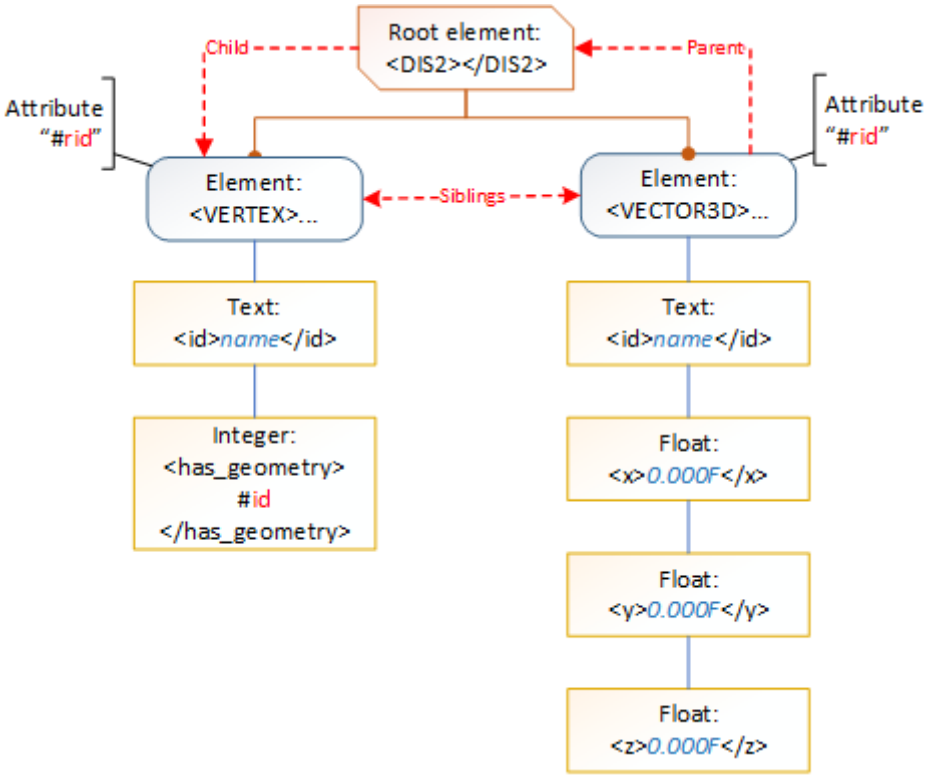

Figure 2-3 BSim XML file relationship between nodes

```
<xs:element name="VERTEX">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:string" />
            <xs:element name="has_geometry" type="xs:string" />*/ VECTOR3D rid ref.
          </xs:sequence>
          <xs:attribute name="rid" type="xs:string" use="required" />
        </xs:complexType>*/ rid = Prefix: # + int
      </xs:element>
      <xs:element name="VECTOR3D">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:float" />*/ Precision 0.000F
            <xs:element name="x" type="xs:float" /> */ Precision 0.000F
            <xs:element name="y" type="xs:float" /> */ Precision 0.000F
            <xs:element name="z" type="xs:float" /> */ Precision 0.000F
          </xs:sequence>
          <xs:attribute name="rid" type="xs:string" use="required" />
        </xs:complexType> */ rid = Prefix: # + int
      </xs:element>
```

Listing 2-1 VERTEX and VECTOR3D XSD schema example

By default, XML documents for specific applications are designed in such a way that while parsing[1] documents it does not break when the document is inevitably changed, or data is not a valid type. [15] For this reason, the XML document is designed with XML Schema (XSD) – describes the structure of an XML document. The main purpose of XSD is to define the legal element blocks in an XML document. For example, what kind of elements and attributes can appear, the number of child elements, data types for elements and attributes, default or fixed values for elements and attributes. [14] In Listing 2-1 XSD example of previously introduced *VERTEX* and *VECTOR3D* relationship in Figure 2-3 is presented in the form of the schema definition. As presented in this example, special attention should be paid to **rid** attribute as the expression of: "# + any number" is a string. However, in the BSim rid represents a unique - integer number with prefix: "#".

The entire BSim XSD example can be found in APPENDIX B

2.2.1.1 BSim dis (XML) file

In Figure 2-2 presented relationship between two elements indicates the location of *VERTEX* in the 3D space. The same principle is followed for the rest of the elements in the dis (XML) file that used to define building geometry, construction, HVAC systems data, schedules, etc.

The key for creating BSim's dis (XML) file is "*rid*"- attribute (see previous Chapter 2.2.1). This attribute with the unique number is being used to reference elements between siblings in dis (XML) file (see Listing 2-1). This reference strategy ensures that BSim can distinguish the

[1] analyse syntactically by assigning a constituent structure to (a sentence) [26]

same type of elements between each other and make proper references across all model elements.

Moreover, BSim dis (XML) file elements can be divided into several categories such as geometry, HVAC systems, and general data. In Figure 2-4 such categorization of elements is presented in the form of tables.

Geometry
BUILDING
CELL
ROOM
FACE
CONSTRUCTION
FACE_SIDE
FINISH
VERTEX
VECTOR3D
EDGE
WINDOOR
RELATIVE_POSITION
CONSTRUCTION_ELEMENT

Systems
THERMAL_ZONE
SYSTEM
SHADING
SHADING_CTRL
HEATING
HEATING_CTRL
LIGHTING
LIGHT_CTRL
PEOPLE_LOAD
VENTING
VENTING_CTRL
VENTILATION
FAN
RECOVERING_UNIT
ZONE_TEMP_CTRL

General data
DIS_PROJECT
SCHEDULE
TIME_DEFINITION
DAY_PROFILE
SITE
LOCATION
PARAM_LIST
PARAMS

Figure 2-4 BSim XML file categorized elements

The specified categories can be expanded even further by adding additional categories such as control, profiles, project settings, etc. However, this is fairly enough as the main aim is to define the methodology, how architectural models from Revit can be transferred to the BSim wire-frame model[2]. In addition, architectural Revit models do not include detailed information that is defined in the systems and general data categories. To sum up, the main focus stays with the elements of geometry category, therefore this category definition is sufficient for further geometry model analysis. The BSim geometry model construction in detail will be analysed in Chapter 3.

2.2.2 Database

Another significant part of the BSim software is the SimDB (mdb). This database is created with the Microsoft Access database management system that combines the relational Microsoft Jet Database Engine with a graphical user interface and software-development tools. [16]

[2] A wire-frame model, also wireframe model, is a visual representation of a three-dimensional physical object used in 3D computer graphics [25]

The SimDB database includes base information about the constructions, materials, doors, and windows. In general, SimDB consists of a combination of two databases such as Building element and Building material. The Building elements are made of groups of the building material that represents a complete database. [12]

The database uses the SfB number classification system which has been established by BYG-ERFA in 1977. The SfB system codes consist of digits and letters composed of a three-phase code. Let's take an example of an element with code – 21.10.00 (f12). First facet (21) indicates building part (exterior wall), the second facet indicates construction (walled construction), and the third facet, indicates resource material (concrete). [17]

Name	Description	Connected by
building_element	sfb index, name of building element	sfb
construction_material	Thermal properties of material	made_of
environment_material	Enviromental impact	made_of
finish_material	Emissivity, solar absobtion, reflection, color…	made_of
frame_material	Frame U-value	made_of
glazing_material	Glass properties	made_of
glazing_material_ex	Additional glazing properties	made_of
glazing_material_st	Glazing solar energy transmittance	made_of
layer	Construction layers	part_of
material	Common properties of materials	sfb
material_amount	Ammount of materials for constructions	part_of
moist_absorp	Sorption curve	part_of
moist_delta_rh	Delta_rh for the hygroscopic region	part_of
moist_absorp	Sorption curve	part_of
moist_desorp	Desorption curve	part_of
moist_material	Additional moisture properties	made_of
pcm_lambda	Thermal conductivity curve	part_of
pcm_melt	PCM melting curve	part_of
pcm_solid	PCM solidifying curve	part_of
pv_material	Pv materials	made_of
Version	Database version	--

Table 2-1 Database table description

This explained convention is used to identify and define new *building_element* with its corresponding material and material properties in the database. The SimDB tables presented in Table 2-1. are connected in hierarchical connections by corresponding SfB number. For example, the *building_element* table is linked to a *layer table*, furthermore, the *layer* table is linked to the *material* table and finally, a *material* table is linked with tables of material properties such as *construction_material, material amount, moist_absorb*, etc.

2.2.3 Weather file

Tsbi5 is using the special binary format for reading weather data. Therefore, in order to make alterations to the weather file, it has to be converted from ASCII – files with BSim suite built-in tool that provides such possibility. [12]

In order to generate a DRY file, it is required to provide weather input data as follows:

- Date (Hourly)
- Temperatures (°C or °F)
- Cloud cover (fraction (0 to 1) or octas (0 to 8))
- Solar incidence (J/cm² or W/m²)
- Humidity (at least one of:)
    - Enthalpy(kJ/kg)
    - Absolute humidity (kg H2O/kg)
    - Relative humidity (% or fraction (0 to 1)
- Wind direction (° (North = 0, East = 90))
- Wind speed (m/s)
- Atmospheric pressure (Pa)

2.3 Summary

This chapter described BSim suite modules and indicated the use of each module. In addition, the chapter focused on the SimView interface, as it is the main interface that connects all the remaining modules. It is worth to mention that, in general, SimView's hierarchical representation of the model is closely related to dis (XML) document structure as it has the same naming convention.

Furthermore, this chapter introduced the reader with diverse input files that are used by BSim modules. The main focus was kept for a dis (XML) file with regards to the geometry that is going to be referred and discussed in the following chapters. For this reason, the author classified all dis (XML) elements into categories.

Finally, this report introduced Database and weather input files. However, weather data inputs were not analysed in detail as it does not have an impact on generating dis (XML) file or model geometry creation.

3 BSim Model geometry

In the previous Chapter 2 input files for the BSim have been explained in detail. In addition, the chapter introduced with a glimpse of a vector position in spatial space. Moreover, the chapter suggested categories for examined XML file elements. Subsequently, in this chapter the main focus will be for the elements assigned to geometry category, simultaneously this Chapter will take one step further in the BSim dis (XML) file with regards to the geometry model.

3.1 Wire-frame model in BSim

This subchapter will take a simple box model and introduce elements in the dis (XML).

As it was introduced in Chapter 2.1.1, SimView produces 4 views in the Cartesian coordinate system. Probably the most representative view of all the views is a 3D view that provides depth sensation. Therefore, model definitions will be represented in a 3D view.

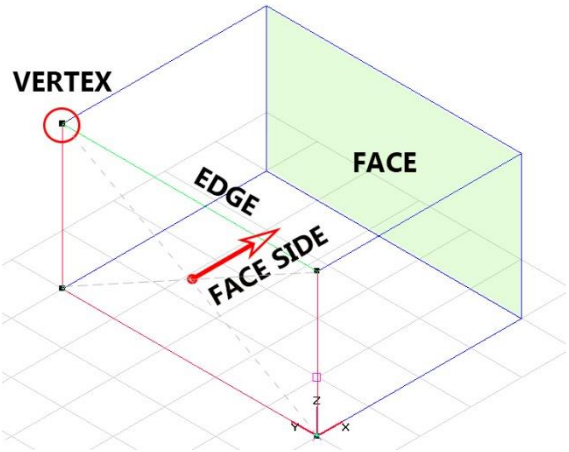

Figure 3-1 BSim model (Vertex, Face, Edge, Face side)

Figure 3-1 presents a simple box where vertex, edge, face and face side can be observed.

To begin with, in general, the model in the SimView is a combination of *VERTEX* points (see Listing 2-1). The two *VERTEX* points will form an *EDGE*, the number of *EDGE* forms closed polygon that is called *FACE*. Furthermore, the collection of faces will form a model

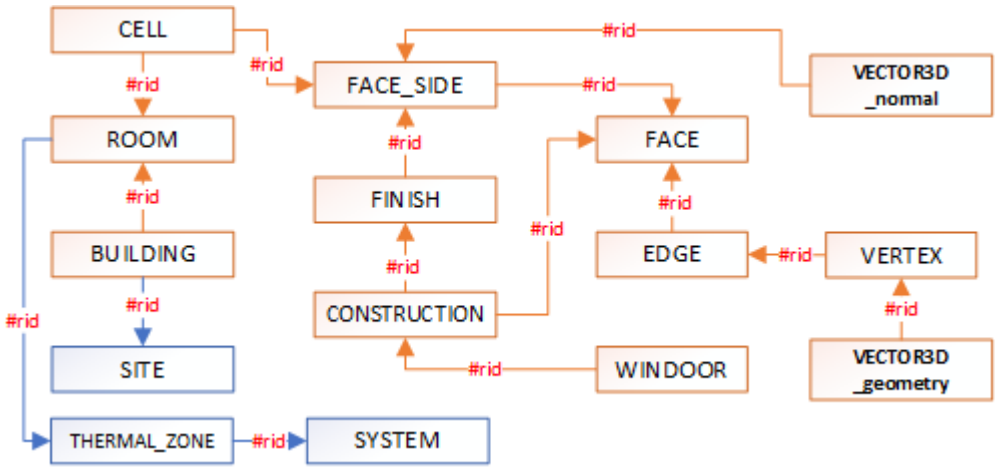

Figure 3-2 Hierarchical relationships between geometry elements.

shell that correspondingly can be named as *CELL* (see Figure 3-4*).* For example, in Figure 3-1 presented room will consist of 8 vertex points, 12 edges, 6 faces, and 1 cell.

As it was mentioned in chapter 2.2.1.1 SimView generates unique rid for each element. After that these elements are referenced to each other with respective rid. In Figure 3-2 such references are presented in detail. The arrows in the example show the direction of the reference flow between the elements, in general, it indicates that element has a rid reference of the element it originates from.

The complete hierarchical relationship between geometry elements and their respective reference nodes can be inspected in APPENDIX A.

3.1.1 Windoor hierarchical tree

In Figure 3-2 presented example is not complete for the *WINDOOR* element. This element has additional geometry references. As a rule, SimView uses the same strategy to describe *WINDOOR* element geometry as it would do for a partition element. Therefore, it includes elements such as *FACE, FACE_SIDE, FINISH, EDGE, VERTEX, VECTOR3D.* However, there are a few exceptions for the *WINDOOR* hierarchical tree that is worth to mention.

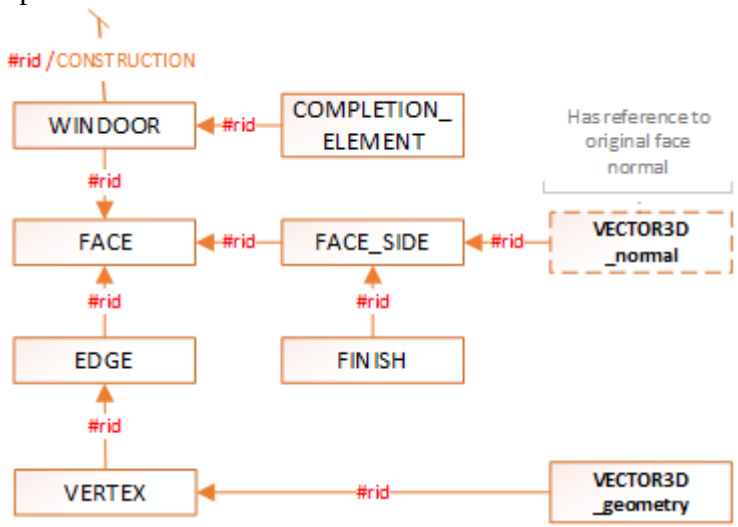

Figure 3-3 WINDOOR hierarchical tree

First of all, it does not create a new normal vector element (see Chapter 3.1.2), rather it utilizes the corresponding *FACE* element of the partitioning it is referred to. The example of the *WINDOOR* hierarchical tree is presented in Figure 3-3 above this paragraph:

3.1.1.1 Window side-fin

In the SimView there two possibilities how shadow objects around windows could be formed. For this reason, overhang or side fins could be modelled by creating room solid objects around the window or setting up side fin distance values as indicated in Listing 3-1. However, in the BSim solid room objects and side fin definitions are used for a different purpose. For example,

side fin definitions are used by the XSun module while room objects are used by the tsbi5 module. For this particular reason, the model has to be set up differently for a particular simulation. For example, if a model is intended to be used for thermal simulations, room objects are designed or if sun path studies are taken, side fins must be defined.

Moreover, this subchapter will partially describe, how overhangs or side-fins are intended to be created with a side-fin dialog box in the SimView.

The side-fins next to the windows are created differently than other faces in the dis (XML) file. It is created with only one additional *RELATIVE_POSITION* element for each side-fin in the model. This type of element does not have any *VERTEX* elements therefore, side fin is not represented in the SimView perspective model view.

```
<xs:element name="RELATIVE_POSITION">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="id" />
      <xs:element name="depth" type="xs:float" />
*/ Distance from FACE to outward pointing normal.
      <xs:element name="distance" type="xs:float" />
*/ Distance from the nearest window EDGE to the right, left or top side fin.
    </xs:sequence>
    <xs:attribute name="rid" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

Listing 3-1 Side fin schema representation.

Furthermore, this element includes two nodes such as depth and distance. The depth node represents the size of the fin while the distance node indicates, the length of the gap between the side fin and corresponding window edge. The Side fin schema can be observed in Listing 3-1.

3.1.2 Face side

The example in Figure 3-1 indicates *FACE SIDE* with red arrow, this face side reference to a normal vector which is perpendicular to the surface at the given point. The SimView calculates the inward-pointing normal (pointing forwards the interior of the *FACE*). The normal vectors are calculated for each *FACE* in order to determine the construction sides. Such a normal vector element in Listing 3-2 is presented in the form of schema representation. The normal vector point is generated one meter away from the internal side of the *FACE*. Therefore, in the given example where the model surface is perpendicular to X coordinate imaginary surface, the vector has an X value of 1.

The careful readers might notice that in the represented example, the normal vector in the same namespace[3] appears with the identical element name as the *VERTEX* coordinates. (see Listing 2-1 and Listing 3-2). Nevertheless, as it was mentioned in this report both equivalent type elements are allocated in different places of the XML hierarchical tree and should be treated properly.

```
<xs:element name="VECTOR3D"> */ FaceSides's Normal
  <xs:complexType>
    <xs:sequence>
      <xs:element name="id" type="xs:string" /> */ For example:
      <xs:element name="x" type="xs:float" />    */ x = 1
      <xs:element name="y" type="xs:float" />    */ y = 0
      <xs:element name="z" type="xs:float" />    */ z = 0
    </xs:sequence>
    <xs:attribute name="rid" type="xs:string" use="required" />
  </xs:complexType>
</xs:element>
```

Listing 3-2 Normal vector schema representation

The next sub-chapter 3.1.2.1 will introduce, how internal and external partitions are defined in accordance with normal vectors.

3.1.2.1 Construction side

Constructions in the SimView are applied with regards to normal vectors. Those vectors define the direction in which construction will take place. Additionally, external partitions have only one normal vector while internal partitions have two normal vectors. For this reason, external partitions are placed inwards while internal partitions are placed from its core.

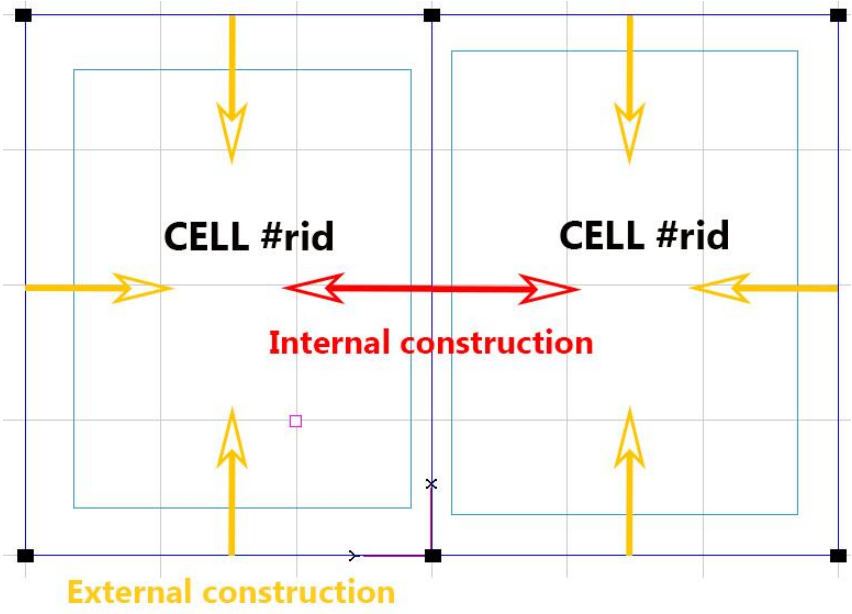

Figure 3-4 Construction placement direction for internal and external partitions (yellow - external partitions, red - internal partition)

[3] XML namespaces are used for providing uniquely named elements and attributes in an XML document [27]

To illustrate internal and external partition placement, the existing model from Figure 3-1 was extended with additional *CELL (room).* The new model is presented in Figure 3-4.

In Figure 3-4 presented wall construction, placement strategy in the floor plan will apply to floor partitions as well. For example, buildings with more than one story will have internal and external partitions, therefore internal partitions will be assigned with two normal vectors and external with one normal vector.

3.2 SimView model limitations

The SimView interface as the rest of the modules in the BSim suite is optimized for building performance simulations. Therefore, like other software in a market that is designed for a specific task, it has some limitations, and the SimView tool is not an exception.

One of the main limitations of the SimView is that it does not support curved lines. Therefore, models in many cases have to be simplified to the simple geometry that is made out of regular polygons such as triangle, quadrilateral or pentagon. The list of the main limitations with regard to the model geometry is presented below this paragraph.

- Curved lines are not supported.
- Structural elements such as columns cannot be included.
- One single window cannot be assigned to more than one face.
- Windows do not include mullions.
- Rooms must be designed in legal shapes.
- Rooms cannot include small partial walls.
- Models in special cases requires simplifications

3.3 Summary

This chapter introduced a wire-frame model, and specific elements in dis (XML) file, that are responsible for the model's geometry. In addition, this chapter focused on the normal vector definition and indicated its importance for defining the face side.

Furthermore, this chapter defined *WINDOOR* element which had the same creation strategy as the partition element. Consequently, the author introduced side fin creation and why these side fins are not visible in a 3D view.

Finally, this chapter concluded with SimView geometry limitations which definitely has influence for final model geometry in the SimView.

4 Model requirements

The previous, Chapter 3 introduced the reader with BSim file geometry and how this geometry is plotted in a specific dis (XML) file format. Consequently, the prior chapter pointed out the main limitations of BSim's wire-frame model.

This chapter will focus on the data that is required to form the BSim geometry model. More specifically, the chapter will define the exact data that is essential to be extracted from Autodesk Revit in order to convert the Architectural Revit model to the BSim wire-frame model. To do so, researches were carried out to re-create the BSim data model structure by taking a manual data export approach. Subsequently, the defined model structure is implemented to be used with the data that comes from Dynamo. All these studies are presented in the following chapters.

4.1 Study case

For re-creating SimView modelling logic, the same study case box model from chapter 3.1 will be used and continuously extended to define the modelling logic. Identically, a similar model is created in Revit for manual data take out. The final, side by side, Revit and SimView models are presented in Figure 4-1 below.

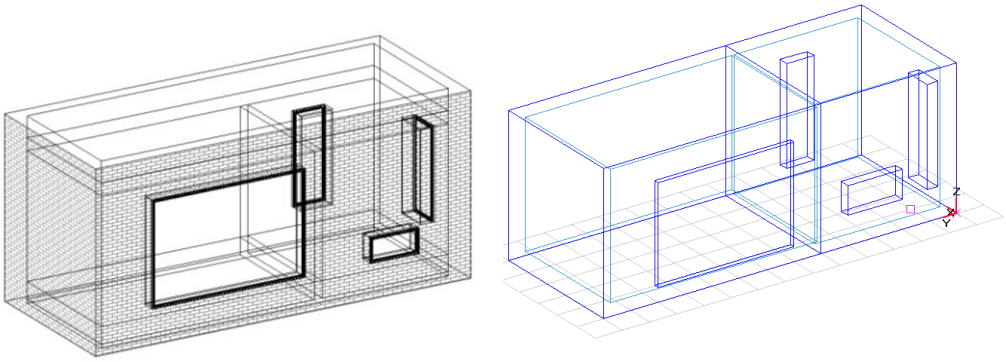

Figure 4-1 Side by side, Revit and SimView models

The required data vector points were gathered by creating imaginary origin (x, y, z) at the corner of the building. As a result, the rest of the vector coordinates were obtained by measuring the distance from the imaginary origin. The imaginary origin can be observed in Figure 4-1 with (x, y, z) indication.

4.1.1 Required data points

In sub-chapter 3.1.2.1 reader was introduced to the model design principles in SimView, and how partitions are defined in this software with respect to the facing side. This methodology is

closely related to the data points or vector positions in 3D space that is required by SimView. As it could be understood from the chapter 3.1.2.1 that external partitions in SimView are specified by outer surface vector locations and internal partition vector locations are obtained in the core of the element. In Figure 4-2, the required data points for partitions and *WINDOOR* elements are presented in the plan view.

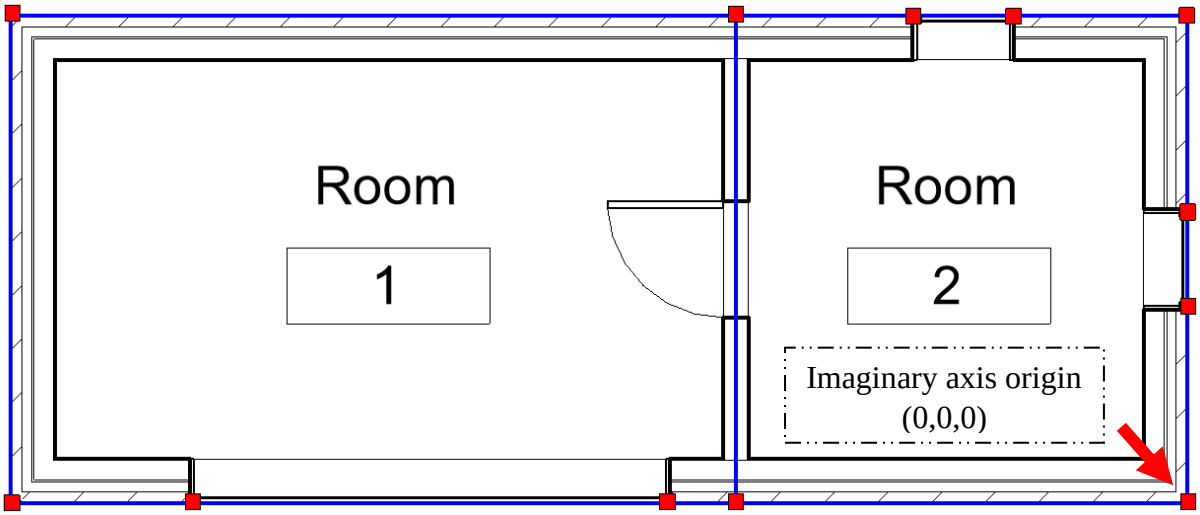

Figure 4-2 Required data points for SimView model

As it can be observed in Figure 4-2 above model requires to obtain external data points from Revit. The simple room geometry shown in the example may create a feeling of simplicity, where in fact, obtaining vector points becomes more complex simultaneously with the model In detail the topic will be presented in Chapter 5, while in the current Chapter 4 the automate rid referencing and re-created dis (XML) file structure will be introduced.

4.2 Data structure

As it was defined in chapter 3.1 windows, doors and opening geometry representation in SimView model have only a reference to the *CONSTRUCTION* element which subsequently, refers to the host - *FACE*. For this reason, it is more convenient to separate *WINDOOR* data from the *CELL* data. Especially this would allow more flexibility for an engineer who is going to make adjustments to the final model. For example, unnecessary *WINDOOR* components could be removed or easily resized depending on the engineer's needs. Hence, the model geometry data is separated into two parts: space data and windoor data.

4.2.1 Space data

To be able to build a 3D object representation in space it is important to define its location. Such object locations in Revit, SimView and other 3D applications are defined by collections of the data points. [18] Of course, geometry complexity is far more complex in Revit than it is in SimView. Despite this fact, it still uses a similar data collection approach to represent the model geometry.

Furthermore, based on SimView requirements space data was grouped into appropriate data collections that have a link to another collection. A short version of data collections can be found in Table 4-1 below this paragraph.

Vertex ID	X	Y	Z	EDGE	EdgeId	FACE consists of edges	FaceID, FinishID, Construction ID	Normal point	CELL	CONid	Has window	Cell bound	Has defined wall
1	0	0	0	1 2	1	1 2 3 4	1	100	1 2 3 4 5 6	1	2	0	21.10.21
2	0	5	0	2 3	2	5 6 7 8	2	200	0	2	2	0	21.10.21
3	0	5	5	3 4	3	9 4 10 8	3	020	0	3	2	0	21.10.22
4	0	0	5	4 1	4	10 3 11 7	4	002	0	4	2	0	27.10.00
5	5	0	0	5 6	5	1 12 5 9	5	001	0	5	0	0	23.10.01
6	5	5	0	6 7	6	11 2 12 6	6	020	7 6 8 9 10 11	6	0	3	22.10.10
7	5	5	5	7 8	7	13 2 14 15	7	100	0	0	2	0	27.10.00
8	5	0	5	8 5	8	14 18 17 16	8	020	0	0	0	0	21.10.21
9	0	13	0	1 5	9	20 6 19 17	9	200	0	0	0	0	21.10.21
10	0	13	5	4 8	10	13 16 19 12	10	001	0	0	0	0	21.10.21
11	5	13	5	7 3	11	15 18 20 11	11	002	0	0	0	0	23.10.01
12	5	13	0	2 6	12		0		0	0	0	0	0
0	0	0	0	2 9	0		0		0	0	0	0	0
0	0	0	0	9 10	0		0		0	0	0	0	0
0	0	0	0	10 3	0		0		0	0	0	0	0
0	0	0	0	9 12	0		0		0	0	0	0	0
0	0	0	0	12 11	0		0		0	0	0	0	0
0	0	0	0	11 10	0		0		0	0	0	0	0
0	0	0	0	12 6	0		0		0	0	0	0	0
0	0	0	0	11 7	0		0		0	0	0	0	0

Table 4-1 Space data collections

For a full-size table content used by BSim Data Modeler can be found in APPENDIX C.

As can be seen in the table above data geometry data is subdivided into collections that fallow SimView hierarchical tree in Figure 3-2. The defined collections are vertex (X, Y, Z), edges, faces, face side, normal, finish, construction and cell.

Moreover, each collection element is assigned with an ID that is being referenced to another collection element in a higher position of the hierarchical tree. For example, an edge with ID = 1 will result in a vertex with ID 1 and 2. The vertex 1 and vertex 2 will have coordinates as follows: (0,0,0) and (0,5,0). The same logic is kept for the rest of the collections in the data model.

4.2.2 Windoor data

The windoor data collections are very much alike as space data collections. It includes collections such as vertex (X, Y, Z), edges, faces, face-sides, finish, normal.

However, as it was mentioned in chapter 4.2 *WINDOOR* elements are only referenced in *CONSTRUCTION* element. Therefore, in order to create a *WINDOOR* element, it is required

to know the appropriate host that includes (*FACE* and *CELL*). The rest of the elements, such as *VERTEX*, *EDGE*, *FACE*, *FACE_SIDE*, *FINISH*, *COMPLETION_ELEMENT*, *RELATIVE_POSITION* and system elements are derived from *WINDOOR* element. Therefore, such elements can be modelled independently. Additionally, construction, systems, and schedules for the windows can be added even to existing models.

The windoor data collections can be found in APPENDIX C.

4.2.3 Rid - BSim Data Modeler

In the previous chapters, 4.2.1 and 4.2.2 presented data collections that do not have a unique rid - raw id, that is required to build model hierarchical relationships. In principle, this rid for each element could be inserted manually. In the situation of the study case model, it would not cause an issue as this model is not very complex. However, this process is very inefficient and most likely would take more time, than creating an entire model in the SimView interface. Therefore, this process should be automated.

Basically, the rid reference logic could be made with VBA. However, as the author is not familiar with VBA code, it was chosen to use C# language to make a standalone .NET framework application for rid referencing and afterwards, create dis (XML) file.

On the plus side, C# code can be transferred to Dynamo as a custom C# library that would allow building BSim models in the Dynamo environment. [19]

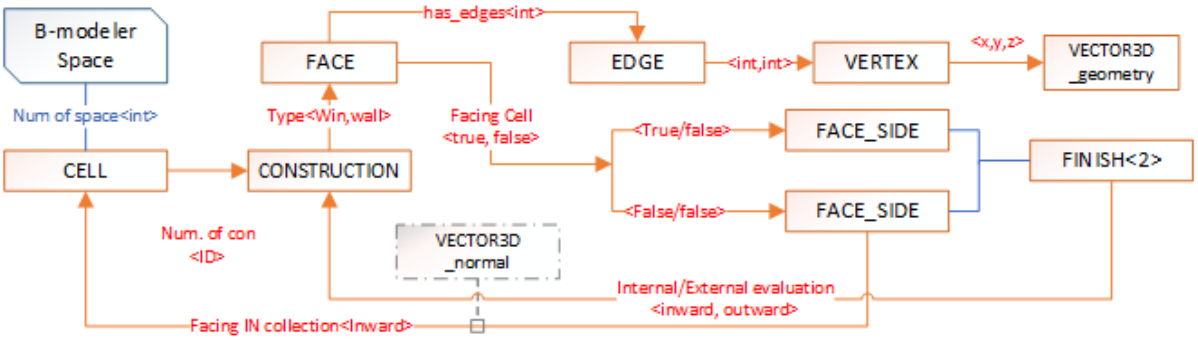

Figure 4-3 BSim Data Modeler geometry model

In Figure 4-3 presented the BSim Data Modeler logic differs from the actual SimView model logic. The main difference between created logic and the authentic model scheme is that formation in the BSim tool starts from *FACE_SIDE* element instead of *CONSTRUCTION* element. The newly created logic for creating the SimView model is presented in Figure 4-3 above. [20]

4.2.3.1 Modelling logic - without normal and finish elements

One of the reasons why modelling strategy was set in the way it was described in Figure 4-3, is that during the research, it turned out that SimView has an algorithm to take care of unspecified normal and finish elements and does not have to be present in the data model. For this reason, if such elements are not present in the dis (XML) model when the file is being saved, the SimView checks for the model integrity and generates a new model with missing elements. In this specific case when *NORMAL* and *FINISH* elements are not written in dis (XML) file it will be appended at the end of the file. A short version of the file's last lines is presented in Listing 4-1.

```
<EDGE rid="#1343">...</EDGE> // Last element before BSim generates missing elements
<FINISH rid="#134">
  <id>Finish198</id> // New element name
</FINISH>
<FINISH rid="#156">
  <id>Finish20</id>...</FINISH>// New element name
---...---
<VECTOR3D rid="#50">
  <id>FaceSide4's Normal</id><x>0</x><y>0</y><z>-1</z>
</VECTOR3D>
<VECTOR3D rid="#64">...</VECTOR3D>
<VECTOR3D rid="#72"></VECTOR3D>
---...---
```

Listing 4-1 Appended finish and normal elements in dis (XML) file

In practice, a *FINISH* element can be excluded from the modelling. In this scenario, BSim will create this element. Consequently, when the SimView creates *FINISH* elements, it will also generate a random *FINISH* element name, as such information was not available in the initial model (see Listing 4-1).

Therefore, this is not the best practice, if there is a need to perform specific simulations. For example, to perform light simulations, it would be required to define the material parameters such as reflectance, specularity, roughness. In order to define those parameters, it would require a new methodology to identify each surface finish layer and its facing directions implicitly.

All in all, it is crucial to write *FINISH* elements from the beginning while *NORMAL* elements can be excluded as they are not required for parameter variation.

4.3 Required geometry elements

As presented in subchapter 3.1 (Figure 3-2 and Figure 3-3) and discussed in Chapter 4, the model geometry is a group of hierarchical tree elements, that are referenced by rid attribute. Furthermore, the required and optional elements to form a simple geometry can be summarized in a list below this paragraph.

Required and optional elements of the topology model:

- VECTOR3D (_Normal) - *optional*
- VECTOR3D (_Geometry)
- VERTEX (window and door)
- EDGE (window and door)
- FACE (window and door)
- FACE_SIDE (window and door)
- COMPLETION_ELEMENT - *optional*
- CONSTRUCTION
- FINISH (window and door) - *optional*
- WINDOOR
- ROOM
- CELL
- BUILDING

The most important element that has the greatest impact on the presented list above is *VECTOR3D (_Geometry),* as the rest of the elements are strongly dependent on the location of these data points. Therefore, the correct definition of these points ensures the accurate representations of the final model.

4.4 Summary

This chapter introduced the reader with the required data points and indicated the importance of defining the right locations of it. In addition, this chapter defined study a house that was used for manual modelling and indicated how the required data points were obtained from the study house model.

Furthermore, this chapter defined the data structure which could be used for the model. Consequently, the author introduced with separate data collections for space and windoor data and why it is convenient to have this data separated.

In addition, the author gave a short introduction to BSim Data Modeler software which is responsible for rid referencing. Lastly, the chapter introduces rid logic inside the BSim Data Modeler tool and most importantly indicated the importance of a need to automate rid referencing.

Finally, this chapter ends with an analysis of the essential dis (XML) elements and what are the disadvantages of letting SimView to write optional dis (XML) elements.

5 Investigation of data export practice

The previous Chapter 4 identified the required data from Autodesk Revit and most importantly introduced a rid referencing strategy for SimView topology model. With all that set-in mind, it is possible to jump into the next chapter. And take a look at how this required data could be obtained from the Autodesk Revit database.

This chapter aims to introduce the reader with a methodology defined by the MOE engineering company.

Moreover, it will give a short introduction into space solids and most importantly, will define why such solids could be a good starting point for successful geometry transfer to SimView.

5.1 Data export practice by MOE

Currently, the engineering company MOE uses Dynamo core for Revit in order to obtain the essential information from the Revit model database. On the next step, the obtained data is transferred to Microsoft Excel where the engineer has control to check exported information whether it is valid or not.

Furthermore, an engineer completes the data model by specifying the additional parameters that are required for a particular case.

Finally, the computed results from Excel are sent back through the Dynamo link to Revit in order to produce a visual representation and documentation on a particular plan view.

In the next subchapters, the focus will payed to the methodology that is used by MOE for making heat loss calculations. Along with it, this chapter will specify why this is a good starting point for obtaining the essential data for the BSim simulations.

5.1.1 Space modelling

Figure 5-2 shows a step by step space transformation methodology, that is used to obtain required surface areas from the Autodesk Revit architectural model. This script summary is one of the three script sets that are used for the entire heat loss calculation method. According to the company, it is crucial to split the computation load in the Dynamo. In the case of huge building models, Dynamo could not properly handle the enormous number of objects at the same time. Due to this reason, in most cases, Dynamo tends to unexpectedly crash during the process.

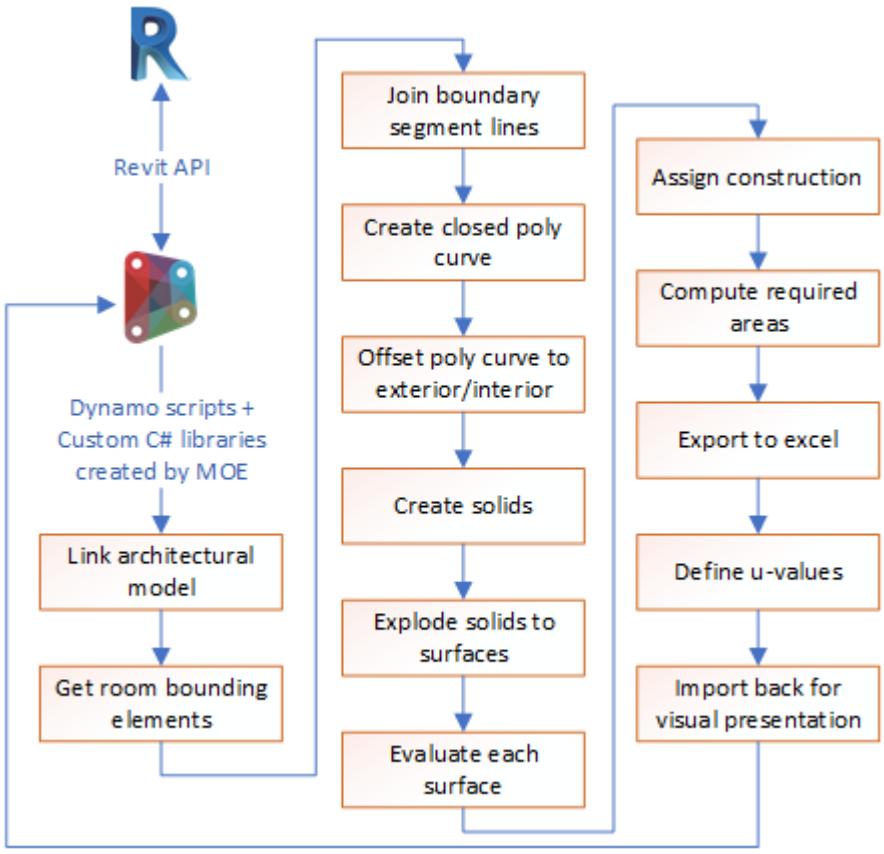

Figure 5-2 Current data export methodology for heat loss calculation by MOE

The methodology presented in Figure 5-2 uses an architectural model as a link to a predefined template, where an engineer is capable of tag building zones and appropriate spaces in it. The identified spaces are the major key to obtain room boundary segments that are later used to create joined polygons. Moreover, such polygons are offset towards the outdoor direction as it is required by DS418 [21]. Finally, new polygons are extruded towards z-plane by existing space height. A methodology that aims to transform room boundary segments and create space solids is visually presented in Figure 5-1 below this paragraph.

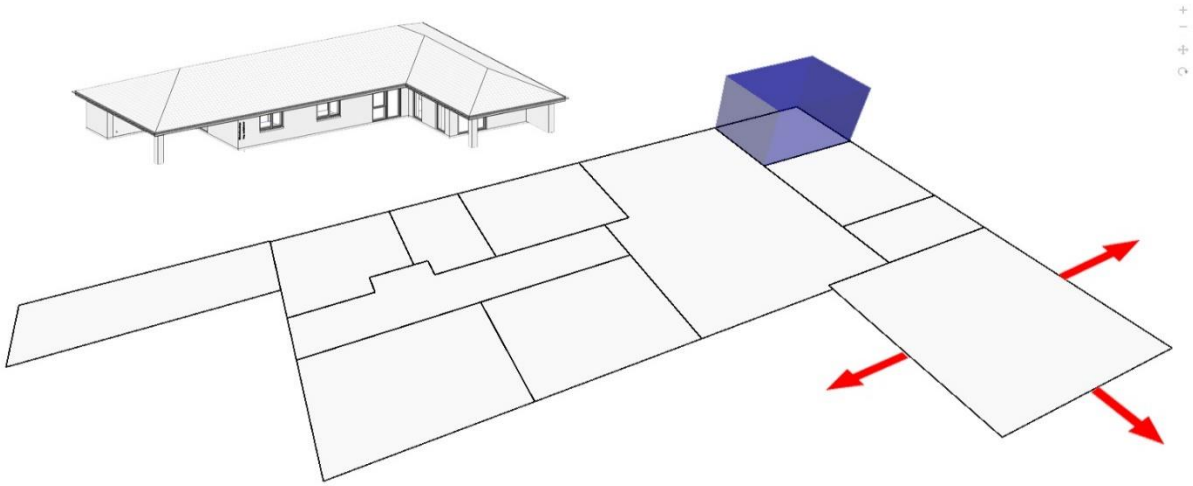

Figure 5-1 Space boundary and extruded space

A full methodology diagram for spaces and windows can be found in APPENDIX E.

5.2 Solids geometry to SimView

To begin with, this project will further exploit the methodology that is used for heat loss calculations. And will investigate how the transformed spaces into solids could help to build simplified SimView geometry models.

One of the reasons, why this particular methodology was selected instead of the BE18 method is that this script had already developed room boundary segment collection methods (see Figure 5-1). Another reason for this choice is that the requirements for the BSim parametric model geometry are very similar to the requirements specified in DS418:2011. [21]

Evidently, there are few exceptions with regards to storey partitions. In SimView, they have to be modelled with respect to the surrounding conditions, that were specified in Chapter 3.1.2.1, Figure 3-4.

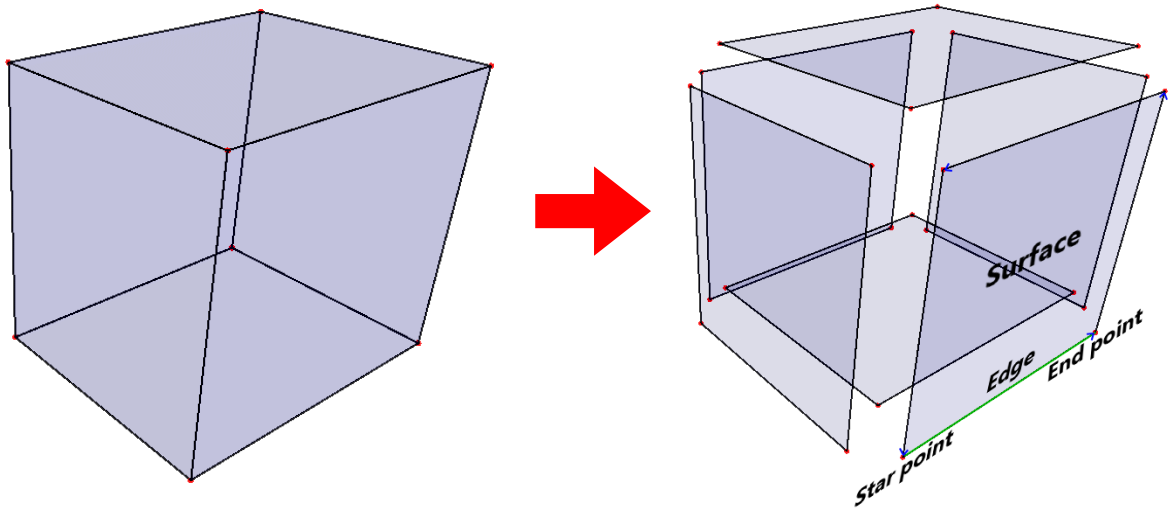

Figure 5-3 Solid equivalence to SimView geometry model

Moreover, exploded simplified space solids have a very close representation of the SimView topology model. As can be seen in Figure 5-3 exploded solid has the same structural parts as they were presented in Figure 3-1. On the other hand, it has a different naming disciple as solids in Dynamo are represented differently than *CELL* geometry in SimView (see Chapter 3.1).

The main difference can be found in edge representation as it is represented by two individual vector points. Generally speaking, each edge along the surface is formed from a start vertex and end vertex.

Later on, the previous edge end-point is used as a new starting point for the next edge that ordered clockwise on the surface. This scenario is presented in Listing 5-1 where *Edge_1* end vertex becomes the *Edge_2* start vertex. Subsequently, the same vector point is used for two individual edges. A careful reader might notice that such data collection includes duplicated point coordinates while SimView topology model is built with unique point coordinates. This is a severe issue for successful data transition and must be addressed correspondingly.

```
Edge_1 (StartVertex = Vertex(PointGeometry = Point(X = 10160, Y = 3480, Z = 0)),
        EndVertex =   Vertex(PointGeometry = Point(X = 5700, Y = 3480, Z = 0)))
Edge_2 (StartVertex = Vertex(PointGeometry = Point(X = 5700, Y = 3480, Z = 0)),
        EndVertex =   Vertex(PointGeometry = Point(X = 5700, Y = 3480, Z = 2500)))
….
```

Listing 5-1 Edge representation in Dynamo

A more detailed analysis of data collections and the data sorting is presented in Chapter 6.5 and Chapter 7.

All in all, solids in Dynamo look like a reasonable starting point for data transfer as it has a very close representation of BSim models and most importantly, it encapsulates all the essential data for the BSim model.

5.3 Summary

This chapter introduced a methodology that is used by the MOE consultancy company to perform heat loss calculations with a minimum effort. In addition, the methodology that was explained in detail gave an overview of how similar practices could be implemented for BSim data transfer. Especially, the author pointed out why there is great importance to split the workload between the separate Dynamo scripts.

Finally, this chapter concluded with an analysis of solid geometry representation in 3D space. To differentiate it, the author identified the space solids as a good representative for SimView models due to their similarities.

6 Development of data export methodology

The previous Chapter 5 introduced the reader with a data export method that uses Dynamo to covert Revit spaces into the solids. Hence, solids were used to compute the required gross and net areas. Unlike the heat loss calculation, the surface areas for the BSim models are too abstract values. Therefore, the script must be extended to export space solid basis data.

This chapter will define a semi-automatic data flow model that could be used to export fully functional, ready to simulate BSim models.

Secondly, there can be found a further investigation and adaption to the data export practice related to the one presented in Chapter 5. More specifically, the chapter will investigate space solid modelling methodology and will discuss the necessary alterations to solid geometry, so it could be used for SimView topology models.

Finally, a summary of the main findings will be provided in order to indicate the limitations of the current export methodology.

6.1 Methodology proposal

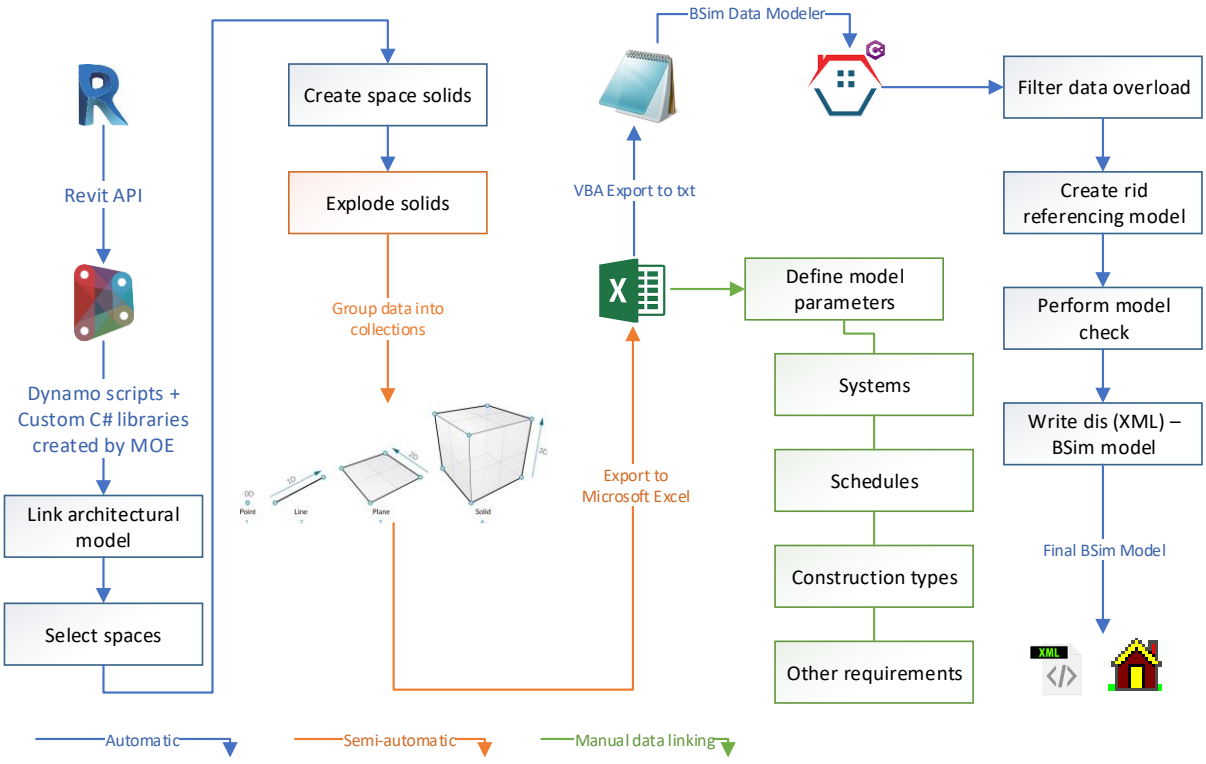

Figure 6-1 Proposed data transfer flow from Autodesk Revit to BSim

As can be seen in Figure 6-1 a proposed proposal, aims to significantly reduce man-hours spent to build a complete simulation model in BSim. Consequently, the importance of the proposed strategy would allow performing simulations with predefined presents, instead of setting it from scratch. Later on, during the design phase, pre-defined parameters could be adjusted and a new model could be built with a matter of minutes. As initial validations of the previous stage model are complete.

The proposed methodology in Figure 6-1 aims to use data export methods defined in heat loss calculations (see Figure 5-2). However, instead of using solid areas, a new set of Dynamo node blocks is designed to acquire 3D geometry representation of such solids (see Figure 6-2).

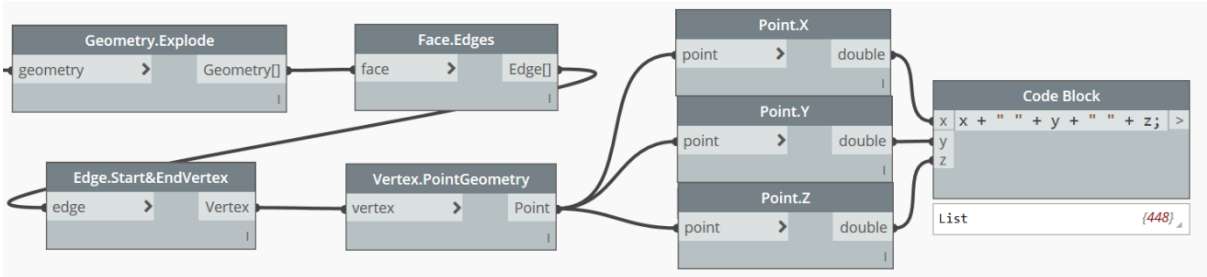

Figure 6-2 Solid exploding procedure (surface, edge, start vertex, end vertex, point (x, y, z)

Moreover, all structured geometry collections are exported into Microsoft Excel, where an engineer is responsible to assign required parameters to the exported architectural model. As the architectural models do not include the required parameters for energy simulations. For example, HVAC system definitions, occupancy, operation schedules, requirements, etc. Such information could be defined in Microsoft Excel by linking to the custom presents. Particularly, such data linking methodology will be discussed in Chapter 7.5.

Finally, the defined model data is exported to a structured text document and supplied to the BSim Data Modeler application which automatically creates dis (XML) model.

6.2 Space model analysis

This sub-chapter introduces the main model translation issues that result in incorrect space representation. Therefore, problems and common solutions will be presented in the following sub-chapters.

In addition, the Revit model floor plan used for analysis in this chapter could be found in APPENDIX F

6.2.1 Overlapping boundaries

First, and probably the most severe problem that occurs with the methodology presented in Chapter 5 is boundary overlapping. This situation occurs when two rooms have a corner joint between external and internal partitions. In principle, if the external partition wall changes its angle, it will develop additional boundary segments that will eventually overlap with each other.

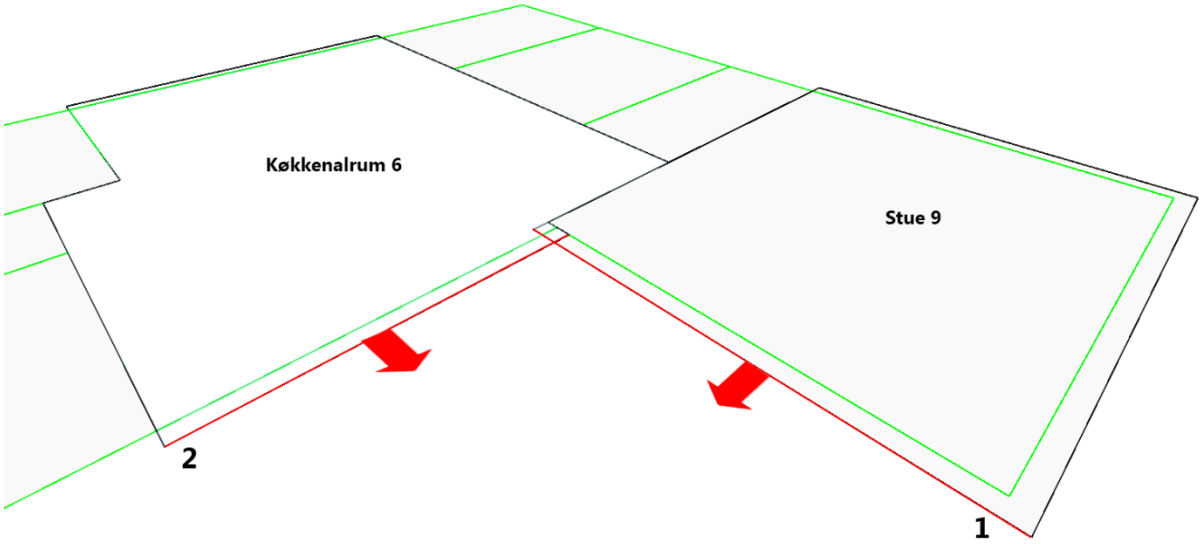

Figure 6-3 Boundary offset towards outdoor creates overlapping surfaces

Figure 6-3 gives a good visual representation of what happens when internal boundaries (green lines) are offset towards outdoor (black lines.). In the example, indicated red lines are offset by keeping a connection to their respective room boundary polygons, such as *Kokkenalrum 6* and *Stue 9*. For all of those reasons, a new overlapping geometry is produced by *Kokkenalrum 6* and *Stue 9* polygons.

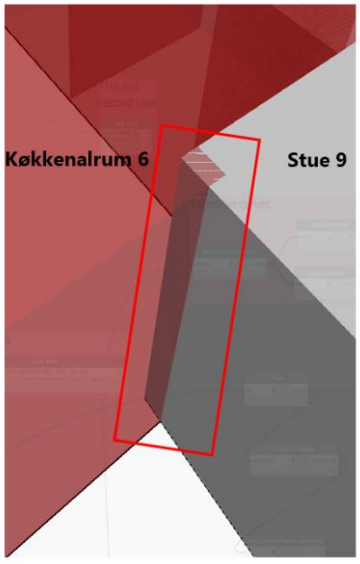

Figure 6-4 Overlapping solids

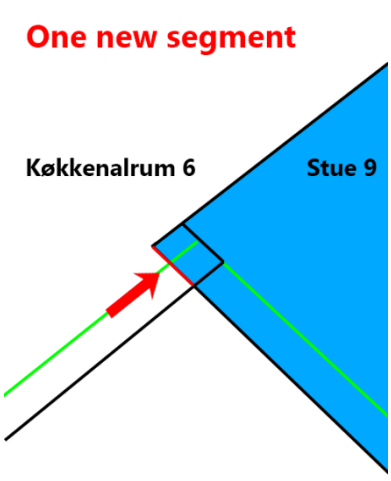

Figure 6-5 1st. crop possibility (green line- internal boundary, black line – external boundary, red line – new segment)

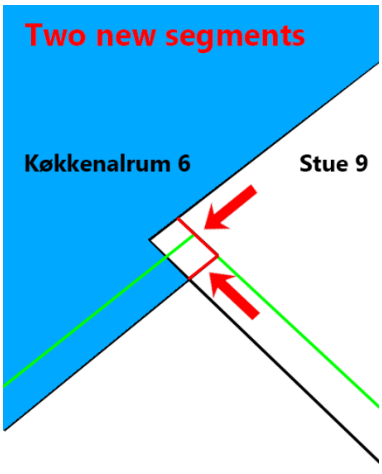

Figure 6-6 2nd. crop possibility (green line- internal boundary, black line – external boundary, red line – new segment)

In this situation when two polygons are extruded as solids it will overlap each other (see Figure 6-4). In the heat loss calculation, such small areas might be neglected as it will not have a significant impact on the final results. But in the case of a BSim model, such overlapping model spaces will definitely develop errors then SimView most likely will crash during the model clean up procedure.

Models, that are created by the engineer, will not face such issues, as the model designer would adjust geometry by moving internal or external wall. A few solutions could be proposed to tackle such a case. For example, to define a method that would allow to adjust polygons or simply crop it with respect to each other.

The first proposal would be unreasonable as Dynamo does not have such sophisticated methods to evaluate similar cases. Moreover, adding a methodology where the user could move polygons might be very complex. Furthermore, it will definitely not work for all cases. Generally speaking, Dynamo is not a drawing tool, but a data treatment tool. Therefore, it would easier to modify rooms or an entire building in Revit instead of doing such heavy modifications in the Dynamo environment.

The secondary proposal is presented in Figure 6-5 and Figure 6-6. There the aim is to evaluate and crop both polygons against each other, so the overlapping part can be discarded. However, this methodology is not perfect in all aspects as well. As can be noticed in the figures a new polygon form depends on the cutting tool (a polygon that is used to cut another polygon). For instance, if *Kokkenalrum 6* polygon was used to crop *Stue 9* polygon - it would result in two more additional surfaces (see Figure 6-6). During the contradictory cropping, only one additional face could be generated (see Figure 6-5).

Out of the box, Dynamo 1.3 does not have any block that could evaluate such a particular case and make a decision to crop the related polygons. Therefore, the author ended up creating a new Iron Phyton node to evaluate similar cases and to crop polygons by a bigger boundary area (see Figure 6-7). However, with regards to the model integrity and consistency, such an option should be left for an engineer to evaluate, how these polygons should be cropped.

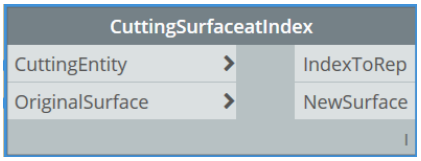

Figure 6-7 Custom polygon evaluation node in Phyton

6.2.2 Reorder of boundary data points

Overlapping boundary polygons are just a small part of the complications that have to be solved in order to have consistent data points in the final model. As it turned out, after polygons are cut, the data points remain on the poly curve. In general, such points remnants are ,, shadows" of previous intersecting line points. Thus, it results in a line of more than one segment.

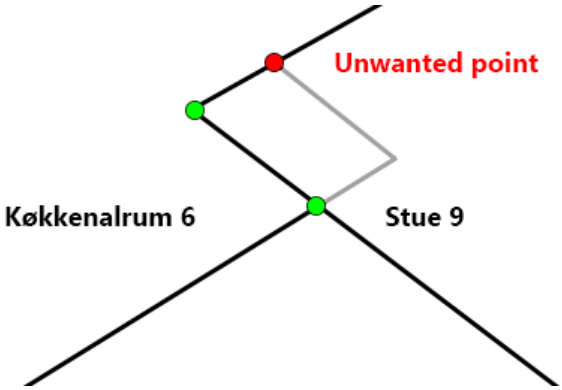

Figure 6-8 Unwanted data point after polygon crop

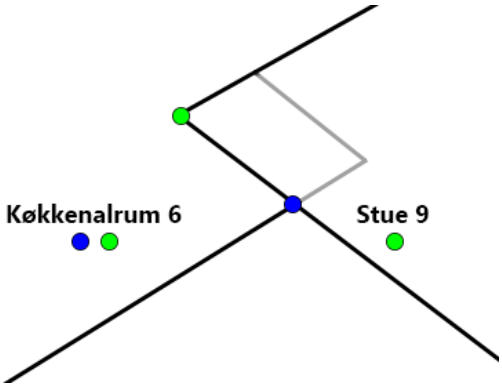

Figure 6-9 Incorrect data point collections between polygons

Figure 6-8 gives a good visual representation of the data points that are left on the polygon. This figure points out an unwanted point that remainders of *Stue 9* polygon. As it is indicated, this point is no longer necessary in a data model.

The unwanted point in polygon could be left out, but this point would compromise individual face creation methods. Therefore, such points have to be filtered out from the model. The reason why this point compromises individual face creations strategy could be read in APPENDIX J.

Neither Dynamo or MOE BIM tools had any libraries to prune such points from polygons. For this purpose, the author took inspiration from open source Springs nodes and with a few alterations to the code, the author made a practice to prune unwanted points from the closed polygons. [22]

Moreover, as soon as the data collections were grouped into collections and trails of building BSim model has been started. This action developed numerous problems. It was noticed that after the polygons crop and the unwanted data point removal, data point collections were not consistent. Especially, in the locations where crop procedures were performed.

Figure 6-9 indicates such a case when data points are not assigned to the right data point collections. As can be seen in Figure 6-9 *Kokkenalrum 6 polygon* has both green and blue points while *Stue 9* polygon has an only green point in its collection. The thoughtful reader might immediately identify why this point is not assigned to *Stue 9* data point collection.

Even though the blue point is at the spot where the line intersects with two rooms, but it is actually belonging only for *Kokkenalrum 6* room polygon. This polygon before offset was consisting of 8 curves while *Stue 9* polygon has only 4 curves (see Figure 5-1). Particularly, it points out that polygons do not keep any relation to each other and in 3D space are represented as individual objects.

Again, this data point collection inconsistency forced the author to take a look for custom solutions which would allow to remap data points between polygons and create proper collections. For this special purpose, another custom Iron Phyton node was made to ensure that the data points are allocated to the right polygon collections (see Figure 6-10)

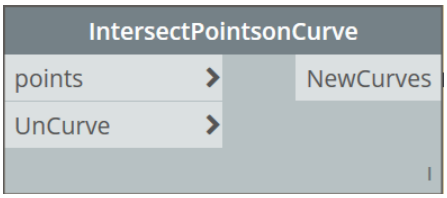

Figure 6-10 Data point integrity Iron Phyton custom node

A complete space data model is presented in APPENDIX D.

6.3 Surface evaluation

As individual solid surfaces could not be used directly, their geometry must be re-modelled as well. The chapter follows an individual surface modelling strategy that can be observed in APPENDIX J.

However, this practice is not valid if the coinciding surface of another room is taller than the parent room surface. The BSim recognizes such surfaces as two different surfaces. Therefore, it is a requirement to sub-divide such surfaces.

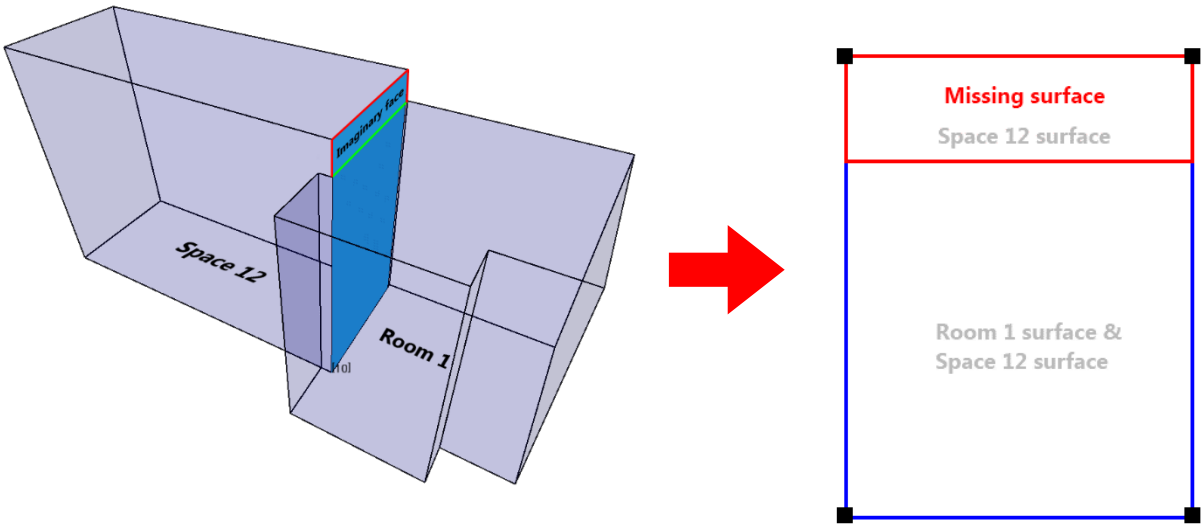

Figure 6-11 Missing surfaces for SimView topology model (red line indicates none existing surface and green lines indicates an upper level of room 1 surface).Surface marked with blue colour belongs to space 12 solid.

For example, Figure 6-11 gives a good illustration of the problem, as presented above in this paragraph. As can be seen in the illustration *Space 12* surface (blue area) consists of one surface. Such surface in SimView has to be represented as two individual surfaces. Unquestionably, Sim View would share *Room 1* surface and the remaining part of the missing surface indicated on the right side of the current illustration.

For this purpose, Space 12 surface must be split into two individual surfaces as shown in Figure 6-11. In the first step, such a surface has to be identified in order to take further action on splitting it. The currently developed strategy uses the surface area and highest altitude point to identify if the coinciding parent surface is taller than the surface on the other side. Based on this purpose, a custom Iron Python node was made to ensure that each surface is checked. Figure 6-12 presents this custom node logic.

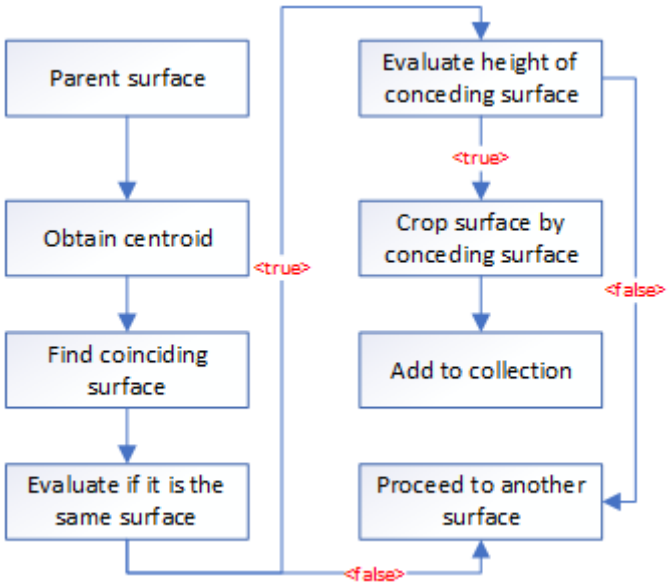

Figure 6-12 Surface evaluation logic (SurfaceEval&andSplit- Iron Phyton node)

6.3.1 Failure to find correct space for cropped surfaces

During the research, it turned out that the surface normal node not necessarily always returns the same result. For example, it might return direction towards the outer space or towards solid centroid. Due to this, the returned surfaces from the custom surface evaluation node in Figure 6-12 was not consistent and resulted in a wrong surface assignment to the space it belongs to. In Figure 6-13 similar case can be observed in detail.

To reassign cropped surfaces to the respective owners the custom surface evaluation node was upgraded with additional methods that are evaluating facing sides on each surface. To do so, a centroid point was generated. On the next step, this point is offset to normal and reverse normal direction. Finally, as soon as space solid intersections are checked, surfaces can be grouped by its respective spaces (see Figure 6-13).

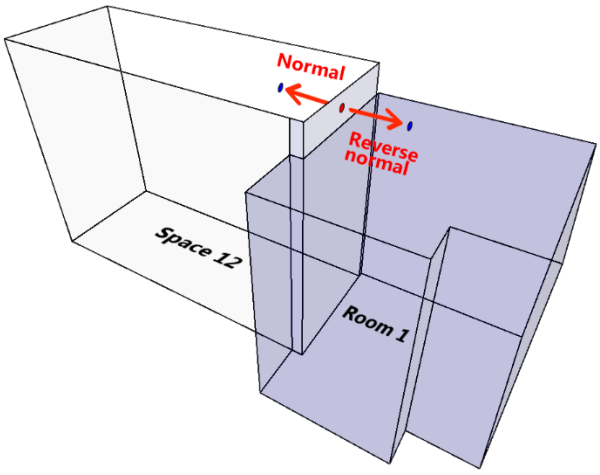

Figure 6-13 Surface allocation to correct spaces

6.4 Window and door geometry analysis

Further investigations take place into exporting the required data for windows and doors families from Autodesk Revit. During the research, it turned out that exporting such elements could lead to a severe error-prone model. For example, windows with multiple panels in some cases could not be exported correctly as it is very dependent on a family design. Moreover, it is important to match the window opening location with an outer solid surface as it could lead to incorrect opening placement.

6.4.1 Incorrect window and door assignment to space

To begin with, investigations took place into window/door export methodology used for heat loss calculation. The current window/door methodology used by MOE could be found in APPENDIX G. Unfortunately, as the methodology is used to obtain only areas, shadows, window fractions of exterior openings had to be discarded. For all of those reasons' methodology is not that suitable, as it was for space data export. Therefore, a new methodology was designed to obtain the essential data points from the Revit model by using: *Link Element Collector. Of Categories Intersecting Bounding Box* – custom dynamo node by MOE. This node can be found in APPENDIX G.

The Custom Element collector node creates a bounding box around each space solid. On the next step, it checks whether predefined category families (windows, doors) intersect to the created bounding box. In general, if a window or a door is assigned to the wrong categories, such families will not be collected by this element collector.

Moreover, during the test of this node, it turned out that this element collector node still requires improvements as it is very sensitive to the tolerance adjustments. For this reason, some of the doors and windows were not picked up from the test model (see APPENDIX G).

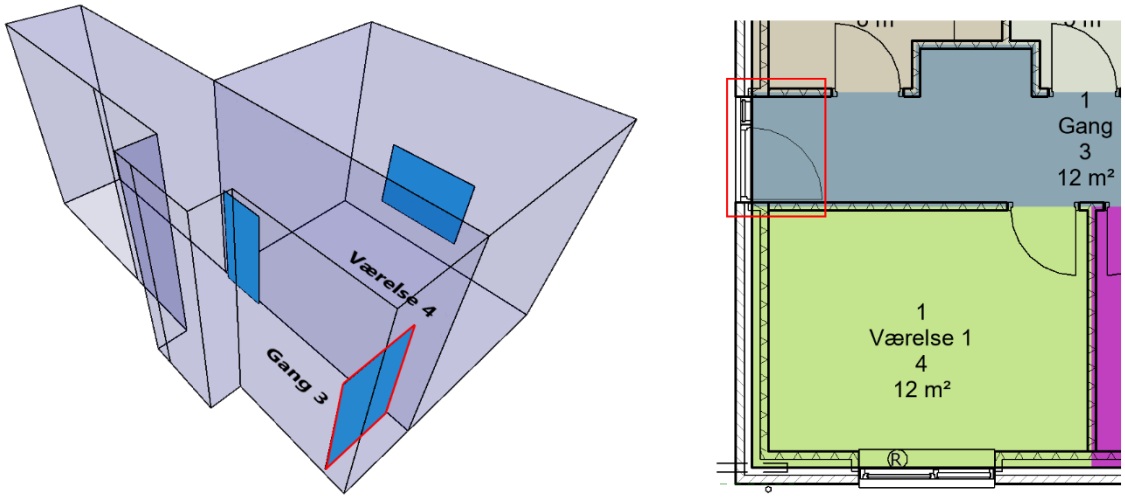

Figure 6-14 Windows and doors are assigned to wrong spaces.
LinkElementCollector.OfCategoriesIntersectingBoundingBox node in Dynamo is insecure

Another issue with the node is that it collects window/door families that do not even belong to the investigated space solid. Correspondingly, it was noticed that this particular issue happens when the window/door families are very close (distance <80 mm.) to the neighbouring space solid. Figure 6-14 gives a good representation of this issue when the opening (opening marked with red edges) is assigned to *Vaerelse 4* space. Finally, when the node loop encounters *Gang 3* space, the opening with red edges are collected once again in window/door data collection. This results, into the duplicated window/door families in the final collection.

In essence, at the moment developed procedures to collect and evaluate Revit families are very strongly depending on a custom element collector node. Particularly, this node failure will result in further errors in the evaluation procedure.

6.4.2 Visual representation might be deceptive

As the name of this chapter implies, window or door openings are not always characterizing its visual representation of the Autodesk Revit family. It is strongly depended on how a family block is designed. Figure 6-15 gives an example of a Revit family which is designed as one block (blue line) and on top of it, window element (red line) is inserted. This family design practice leads to an incorrect final representation window and door element.

To begin with, as soon as a family is designed in such manner the entire component could not be extracted correctly and it results in the wrong segment assignment. It seems though, that the Dynamo node in Figure 6-16 is not designed for such a case and loses one segment on top of

the window and door block while the green segment reference at the door threshold level is unclear.

Furthermore, as a segment is missing in Figure 6-16 it causes the following chain of reaction to further family data treatment. As can be observed in Figure 6-17 final polygons are snapped into one polygon.

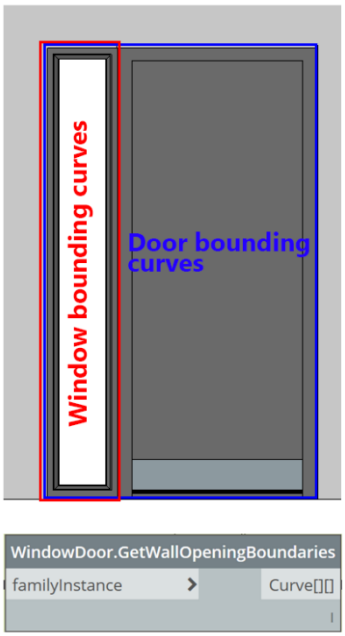

Figure 6-15 Autodesk Revit family element bounding curves

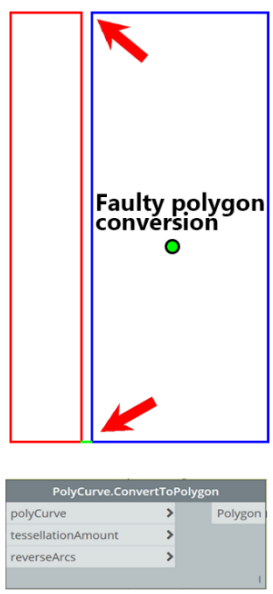

Figure 6-16 Boundary curves after conversion to independent polygons (it is unclear with polygon includes green line segment)

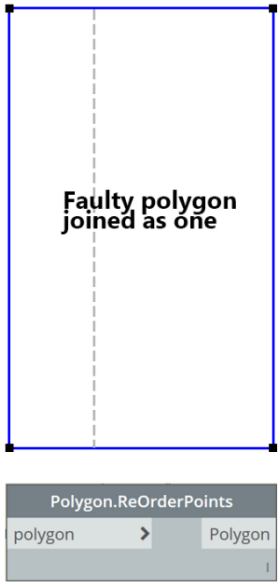

Figure 6-17 The Final polygon after point clean up

All things considered, *PolyCurve. Convert To Polygon* and *Window Door. Get Wall Opening Boundaries* nodes from the MOE package could be updated to evaluate such cases. At the moment an engineer should use this conversion method with care and if necessary, insert window and door manually. Particularly, this step should be done, if daylight simulations are performed for a modelled space.

6.4.3 Window and door placement on the surface

Exterior windows and doors like exterior walls in SimView are modelled by plotting it on an exterior surface. Thus, it is required to move the interior window/door opening polygons on an external solid surface as it is indicated in Figure 6-18.

This particular case can be tackled in many ways, such as by moving the entire polygon with *Geomerty.Translate* node, by intersecting solid with a wall surface or perform vector allocation in BSim Data Modeler. However, the most practical way is to perform all geometry translations

in Dynamo than moving this job to another environment, as it has visual representation capabilities and an engineer can observe a model transformation in a 3D view.

Moreover, it was chosen to use a solid method for obtaining window geometry coordinates on the external surface. These two transformation steps are presented in Figure 6-18 and Figure 6-19.

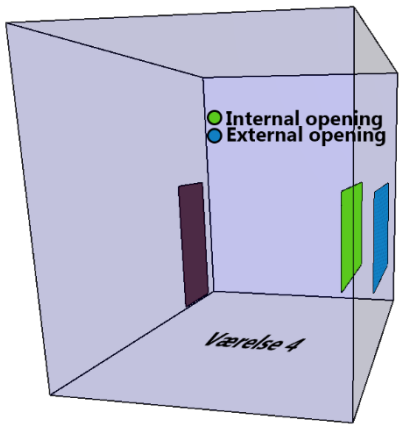

Figure 6-18 Window required geometry location (blue area)

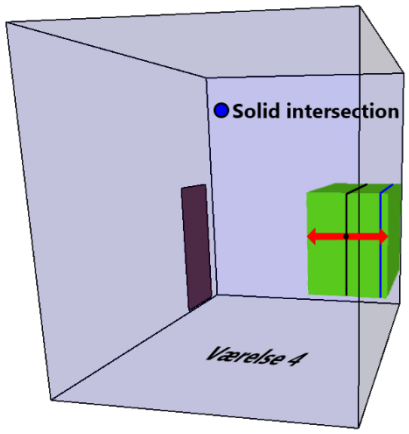

Figure 6-19 Window geometry location obtained by intersecting window polygon solid with the horizontal surface (intersection aria marked with blue line)

A solid intersection method for windows was chosen intentionally. This methodology helps to solve two matters at the same time. As it was mentioned in Chapter 6.4 windows and doors that consist of multiple panels with separating mullions, by window/door element collector would be recognized as two overlapping openings. Generally, if such data would be modelled in SimView it would create overlapping windows that would eventually crash the simulation tool. The related matter is represented in Figure 6-20 and Figure 6-21.

In other words, by extruding two overlapping solids, and in the next step cropping them by each other, new surfaces are obtained. Those surfaces with a common edge could be used without violating the SimView model integrity.

6.4.4 Overlapping window /door elements

The problem with separate window and door families situated next to each other is very similar to the issues discussed in Chapter 6.4.2. However, unlikely the previous example in Figure 6-15, this particular case in Figure 6-20 is fundamentally different and easier to automate without any need for alterations by an engineer.

In Figure 6-20 could be observed an example of window and door curves, such boundaries are not situated on each other, but unfortunately, they develop a small overlapping area that is not valid in the SimView model(see Figure 6-22).

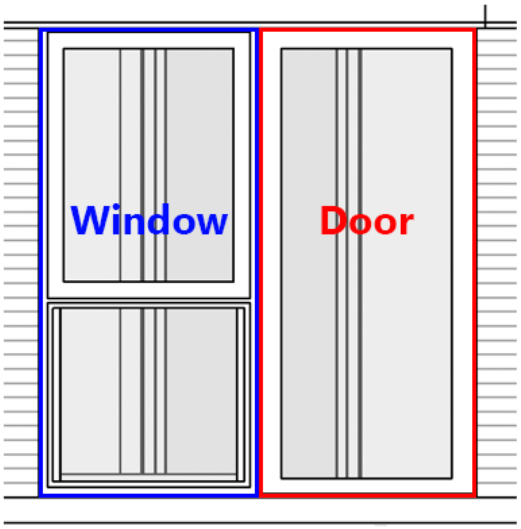

Figure 6-20 Window and door boundary curves in Revit interface

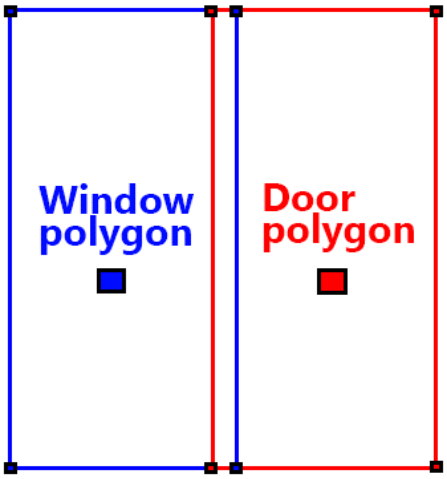

Figure 6-21 Window and door opening polygons in Dynamo (data point colours represents individual collections for door and window polygons). In this example, the polygon overlapping area is scaled for visual purposes.

For the visual purpose, a scaled version of overlapping polygons is presented in Figure 6-21. The thoughtful reader might notice two data point collections. A small point with a colour inside (blue or red) points out that data points are allocated to the right polygon collection. Therefore, such polygons could be cropped against each other by discarding a small overlapping area. Generally speaking, it would result in new polygons without overlapping areas. To do so, as it was mentioned in the previous chapter, such cases are evaluated together with the solid crop method.

As a matter of fact, there are a few options on how the opening overlapping situations can be handled inside the Iron Phyton node (*External_ Windor_ Evaluation*). Again, there is a possibility to set which opening should be used as a cutting tool. This preference would set which opening boundary should be kept unchanged while the following opening would lose a fraction of its area. For example, in Figure 6-22 presented door opening is cut by window opening boundary curves. Accordingly, the door polygon area is reduced by 0.0221 m^2.

Figure 6-22 Door opening polygon area change after the cropping procedure is applied. In this example, the polygon overlapping area is scaled for visual purposes.

On the other hand, no matter which way these polygons are cropped against each other, the total parameter area remains the same in all possible cropping scenarios.

6.4.5 Windoor evaluation with a custom node

To tackle down all window and door related issues that were addressed in previous chapter 6.4.3 and chapter 6.4.4. The author has developed a set of 3 custom nodes (*Internal_Door_Evalution, External_Windor_Evaluation, External.Windoor.Unq.Allocation*) for internal and external building openings. A node responsible for internal opening evaluation is slightly different than the nodes made for external opening evaluation, as it does not perform self-intersection checking methods. Despite this fact, all custom nodes follow the same strategy summarized in Figure 6-23.

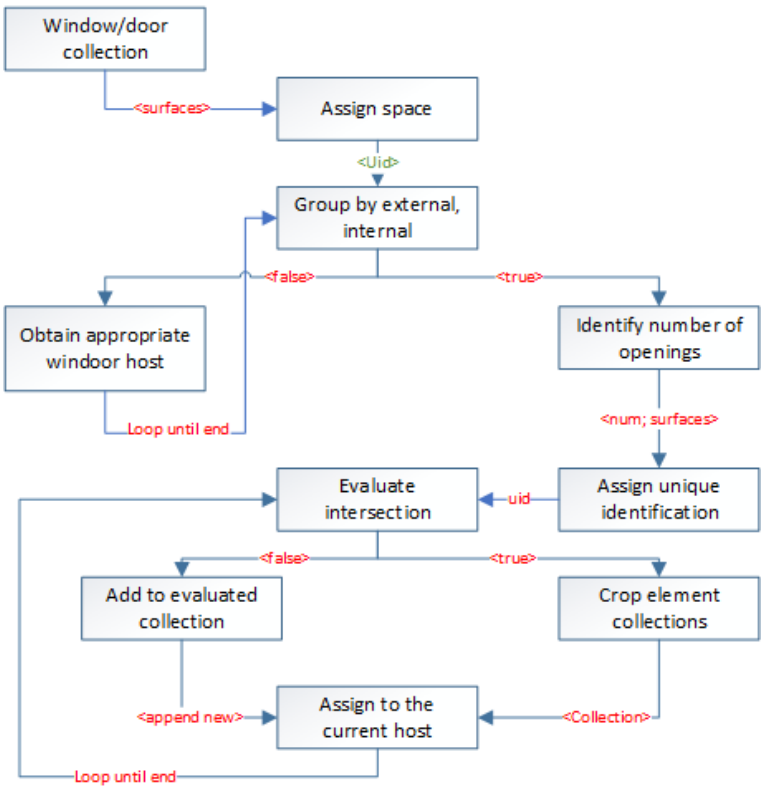

Figure 6-23 Windoor custom phyton node logic (External_Windor_Evaluation, External.Windoor.Unq.Allocation)

All in all, at the moment, developed methods certainly do not cover all the window/door cases such as skylights or windows that are shared between two spaces. Therefore, the methodology should be used with a caution of possible inaccuracies in the final model.

6.5 Data collections from Dynamo

As it was mentioned in chapter 5.1.1 due to performance reasons, it is crucial to divide tasks between discrete scripts, as Dynamo most likely will crash by handling the enormous amount of objects at the same time. For these several reasons, separated data collections were made, such as spatial data and window/door data to be exported from Dynamo separately. This sub-chapter will summarize the data collections that are imported into BSim Data Modeler.

6.5.1 Space data collections

All defined data collections in Figure 6-24 are essential to recreate the SimView topology model. In illustration presented data collections are raw exploded solid data groups that do not include unique ID referencing. It only has localized ID assignments to derived data collections. For example, the edges of the face will be represented by repeating face IDs.

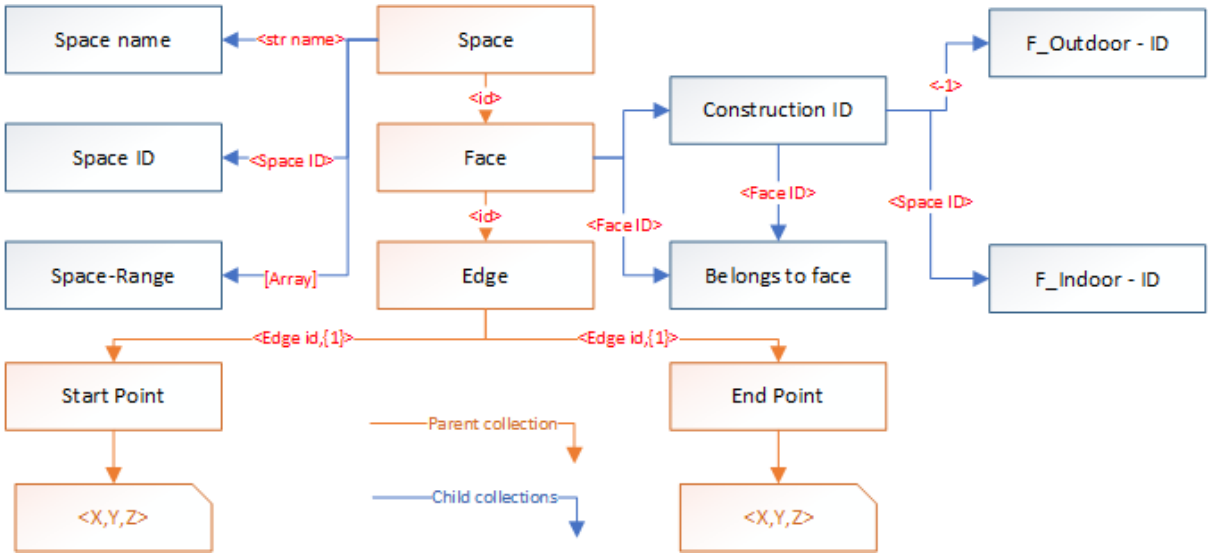

Figure 6-24 Space data collections from Dynamo

A full data table example can be found in the APPENDIX 0

All in all, defined space data collections contain numerous amounts of duplicated data that has to be treated before building a dis (XML) model. Following Chapter 7.2 will discuss in detail why it is relevant to filter data overloads.

6.5.2 Windoor data collections

In Figure 6-25 the presented windoor data collections are very much alike to the space data collections introduced in the previous chapter. The defined windoor data collections export window/door geometry representation as a segment of the *FACE* in SimView. Particularly, the most important part of this illustration is the correct reference to the <Host ID> in the space data collection. As it specifies an opening placement in the SimView model.

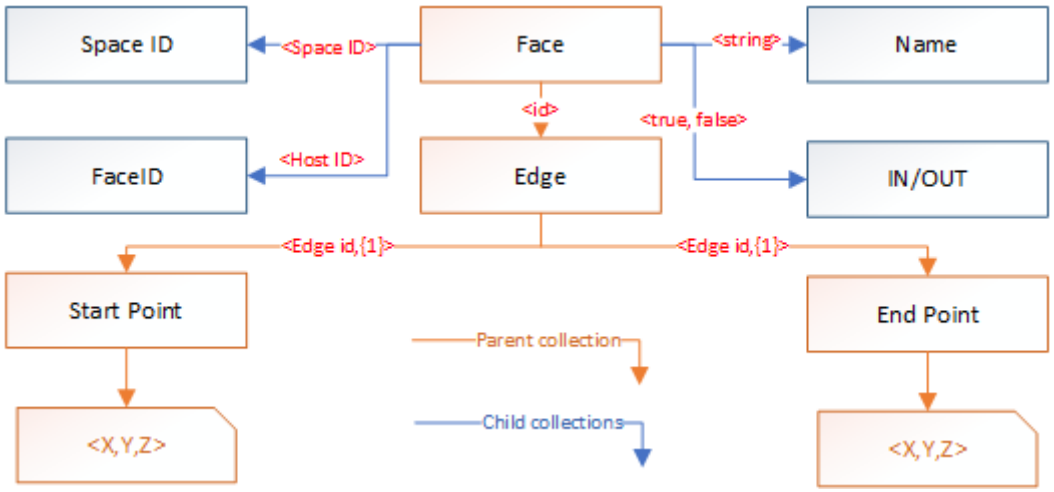

Figure 6-25 Windoor data collections from Dynamo

A full data table example of Figure 6-24, can be found in the APPENDIX H

To sum up, defined windoor data collections contain numerous amounts of duplicated data that has to be treated before creating a dis (XML) model. The following Chapter 7.2 will discuss in detail why it is relevant to filter data multiple occupancies in data collections.

6.6 Limitations

Currently, the developed space and window/door methodology has some application limitations as the author of this thesis covered most, but not all possible cases in the Revit architectural model. Accordingly, the methodology has to be tested on numerous models to define possible inaccuracies.

In addition, those limitations come due to the necessity of upgrading or rewriting custom Iron Phyton nodes that were presented throughout the report, into more sophisticated methods.

The current known limitations are:

- Sloped surfaces are not supported
- Skylights can only be assigned on a flat roof
- Room polygon crop function does not support multiple polygons
- Windows/door families of multiple segments my cause inaccuracies
- The multiple storey model might not work for some cases
- BSim Data Modeler ver. 0.1.7 has a limited amount of error check methods
- Dynamo and Phyton scripts require optimization

6.7 Summary

This chapter introduced the data export methodology and indicated the required steps that need to be taken, in order to obtain the required data from Autodesk Revit. In addition, this chapter has begun a study on a more complex model.

Firstly, this chapter moved on to the analysis of space solids. Consequently, the author identified the most critical issues related to boundary segments, such as polygon overlapping, different cropping possibilities, inconsistent data point allocation, surface modelling issues. As a solution to similar problems, the author introduced a custom Dynamo node logic to overcome the mentioned issues.

Secondly, the author moved on to a window/door analysis. The conducted analysis pinpointed another set of problems with regards to a window/door opening such as incorrect window and door assignment, Revit family's dependency, overlapping window polygons, incorrect placement. Correspondingly, through the chapter, the author defined the most universal solution for *WINDOOR* elements by extruding them as solids.

Third, this chapter introduced with defined data collections for space and door data. In addition, the author pointed out that this raw data in the defined collections is not final and it requires further data treatment.

Finally, this chapter concluded a list of limitations for the current methodology. Identically, it stated that such limitations are caused due to the requirement to build more sophisticated Dynamo nodes.

7 Solid data translation

As it was briefly introduced in the previous Chapter 5.2 even though solids are very much alike as SimView geometry models. In principle, they are presented very differently in 3D space. Thus, it requires a robust methodology, how this data could be transformed and filtered out to the level of SimView representation. For this reason, this chapter will break down the solid geometry and will discuss, how this geometry structure could be converted and be used for SimView models.

7.1 Solid internal surfaces

To begin with, it would be wrong to interpret that solids are equal as *CELL* in SimView without taking deeper analysis, how these solids are represented in 3D space. As soon as, two solids that are sharing the same internal surface, are exploded. It results, in two surfaces that belong to each individual solid. For example, this case is indicated in Figure 7-1 below this paraph. As can be noted from the figure, solid 1 and solid 2 does not keep any relation to each and produce two identical surfaces.

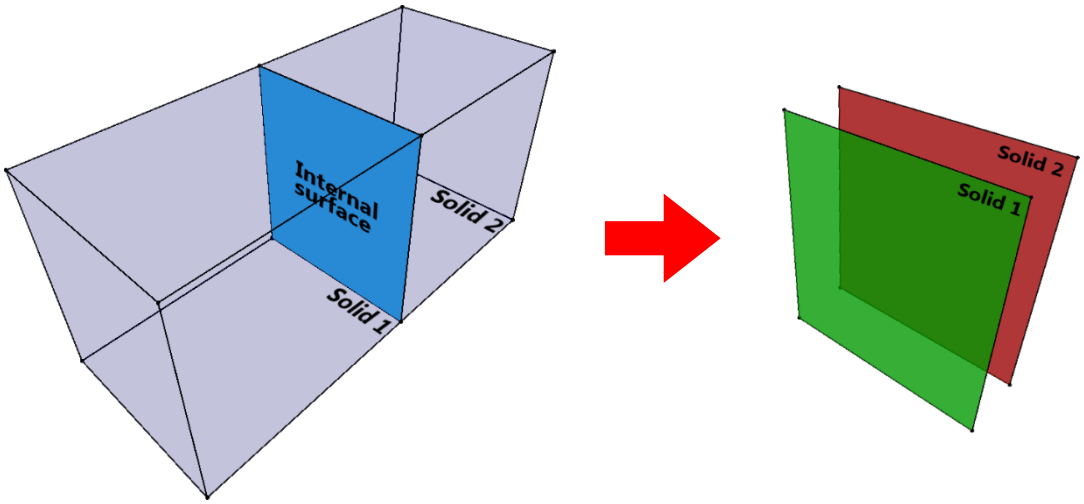

Figure 7-1 Junction of two solids

Consequently, in this case, solids surface data could not be directly used as it violates the SimView topology model, which shares the same internal surface for both spaces (*CELL*).

On the other hand, during the research, several trials were conducted to supply the SimView data model with duplicated internal surfaces. However, this is not the best practice due to numerous reasons that will be discussed in Chapter 7.2.

7.2 Data overload

During the research, several trials were conducted to supply the SimView data model with duplicated internal surfaces. The results were successful on a small-scale model like the one that was introduced in Chapter 3. However, such a modelling strategy is not the best choice due to missing references, amounts of data and performance related complications.

In the first place, it should be noted that BSim has model clean up logic allow not just to create new XML nodes as presented in chapter 4.2.3.1. It also removes duplicate data elements. However, this is a breakpoint where the multiple issues start.

7.2.1 Data model referencing

Probably the most severe problem occurs due to new referencing in the dis (XML) model. Listing 7-1 indicates such an issue in action, it points out what happens to dis (XML) file structure when the model is built with an individual *FACE* element for each *CELL*.

```
// Before SimView model clean up
<FACE rid="#42">// Internal surface as illustrated in Figure 7-1
  <id>Face8</id>
  <area>15</area>
  <round>16</round>
  <has_edge>#18 #19 #20 #21</has_edge>// Edge and vertex overload
  <has_face_side>#14 #17</has_face_side> // Face side overload
</FACE>
...
<FACE rid="#43">
  <id>Face9</id>
  <area>15</area>
  <round>16</round>
  <has_edge>#44 #45 #46 #47</has_edge>// Edge and vertex overload
  <has_face_side>#31 #32</has_face_side>// Face side overload
</FACE>
// After SimView model clean up
<FACE rid="#8">
  <id>Face5</id>// Reference lost – new name is generated by SimView
  <area>15</area>
  <round>16</round>
  <has_edge>#15 #16 #20 #19</has_edge>// New edges and data points ref.
  <has_face_side>#21 #22</has_face_side>// Reference lost
</FACE>
```

Listing 7-1 Surface data overload in SimView

As it can be observed in the first part of Listing 7-1, at the beginning model is built with two *FACE* elements such as `<id>Face8</id>` and `<id>Face9</id>`. Immediately, after the model clean-up function is performed in SimView, one of the *FACE* elements is removed from the data model. The biggest disadvantage that the new `<id>Face5</id>` id is written in the model and there is no way to implicitly identify which FACE element was deleted from the dis (XML) model.

Moreover, the *FACE* element is very high in a hierarchical tree of the BSim's file structure. Therefore, elements such as *FACE_SIDE*, *FINISH*, and *FINISH_MATERIAL* are removed or renamed from the final model as well. Consequently, to allow for BSim to take action on duplicated *FACE* data is not the best approach, as it causes a whole chain of reactions and inconsistent model referencing. For example, this will definitely lead to inconsistent models where there is a requirement to define *FINISH_MATERIAL* element for the particular room.

7.2.2 Data overload impact on performance

A thoughtful reader might notice another issue indicated in Listing 7-1. As soon as it is built this way each *FACE* is written into dis (XML) file with its respective geometry nodes. In this situation same multiple *VECTOR3D*, *VERTEX* and *EDGE* elements are plotted in the dis (XML) file. Following this would create a performance drawback not only on the BSim side but also on the BSim Data Modeler side. In principle, BSim Data Modeler will have to create bigger dis (XML) models and BSim has to read 2-3 bigger model files than necessary. For example, the case house mentioned in Chapter 5 was tested with duplicated data. As a result, the dis (XML) file size increased from 7936 lines to 17036 lines. Unfortunately, model after geometry clean up gets corrupted as some of the internal walls *VERTEX* was assigned to incorrect *FACE* elements.

All in all, the outcome of modelling multiple surfaces does not satisfy the model needs and results in a data leak. Most importantly, all of it results in an inconsistent model with missing references and a larger dis (XML) file.

7.3 Individual solid surface

In Chapter 5.2 briefly introduced solid's surface edge representation will be further investigated in this sub-chapter. The aim of this chapter is to define a methodology that could be used to covert solid surface elements to the SimView *FACE* element.

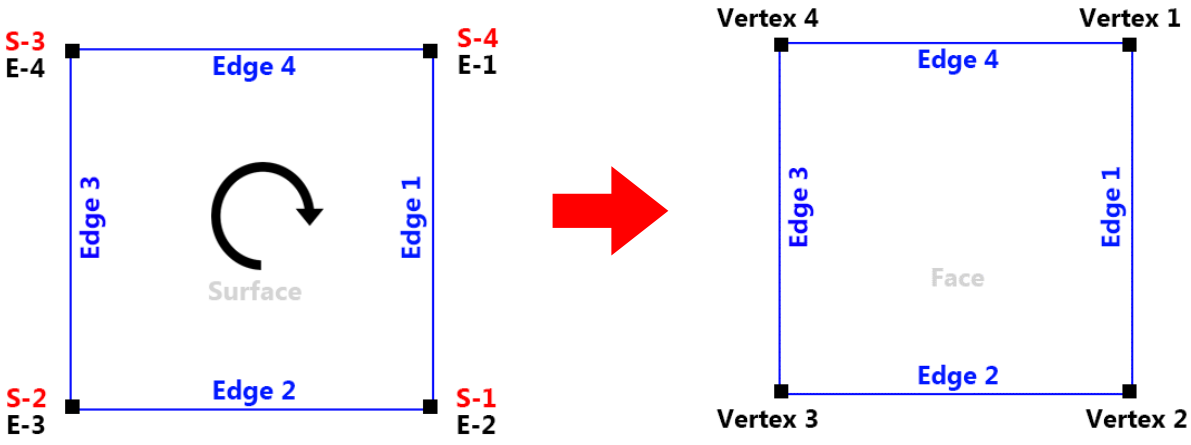

Figure 7-2 Surface conversion to Face

In Figure 7-2 represented side by side, surface and face elements fundamentally are very similar, but they require proper treatment before they could be used to write dis (XML), model. As it can be noticed in this figure, the surface element has 2 data points at the same location. Such points are *start-point* (S-1, S-2, S-3, E-4) and *end-point* (E-1, E-2, E-3, E-4). On the other hand, the face element has only one data point (Vertex 1,2,3,4) at the same location.

In principle, any *start-point* or *end-point* could be used for a new edge reference in the SimView model as coordinates are identical. However, it is not that simple as it might look from the first glimpse, as the edges in the SimView could be referenced in any order and at the same time, they could be shared by 2 to 4 faces. An example, when the edge is shared by 4 faces can be observed in Figure 7-3.

Moreover, as it was mentioned in the previous paragraph *EDGE* in SimView could be referenced in any order. In general, point order like in surface edges does not apply to face edges. For example, in Figure 7-3 shared edge could be represented as `<has_vertex>#3 #2</has_vertex>` or `<has_vertex>#2 #3</has_vertex>`. Both definitions would be valid and would not create any issues in the SimView topology model.

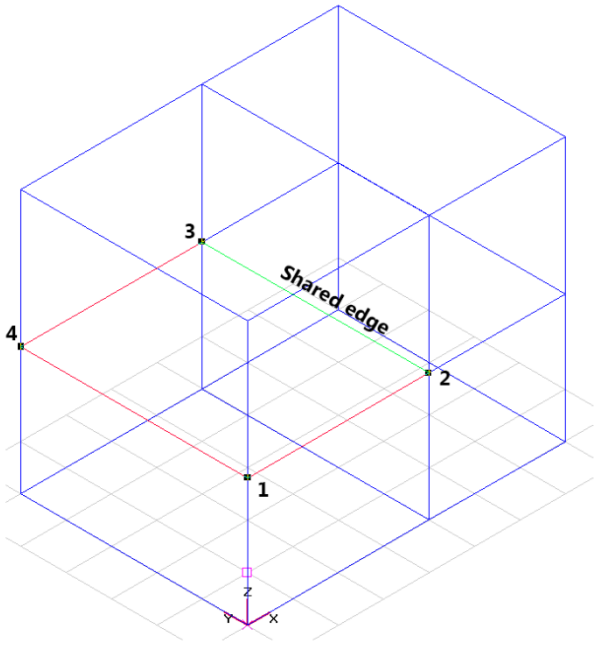

Figure 7-3 Face represented by 1,2,3,4 vertexes share a common edge (2,3) with other underestimating surfaces

Furthermore, face reference logic follows the same strategy as the one presented for edges. Hence, edges in the *FACE* element could be represented in any possible order. For example, indicated face (see 3 red lines and 1 green line) in Figure 7-3 has 4 *VERTEXES.* In this situation, a number of possible collections could be estimated as: 4!=4 × 3 × 2 × 1 = 24. This will result in 24 possible collections (E.g. `<has_edge>#1 #2 #3 #4</has_edge>`) that must be checked and assigned as equal.

In addition, in Figure 7-2 the presented solid surface start point does not necessarily always start at the same location. For this reason, two identical surfaces might be recognized as two individual surfaces.

7.3.1 Equality comparer

In order to eliminate multiple data points, edges and faces in exported Dynamo data collections, the additional library was coded to filter all data before model creation takes place in BSim Data Modeler. To do the job BSim Data Modeler was developed to use a rather simple but

effective *GetHashCode()* method to override default equality comparer. A hash code is a numeric value that is used to insert and identify an object in a hash-based collection. The get hash code method provides this hash code for algorithms that need quick checks of object equality. [23]

In Listing 7-2 Edge object equality comparer is overridden with a new set of rules that defines how these two vertexes in the node have to be compared. In the presented example, both 1st and second vertexes are compared against each other in order to define if the edge that is being compared already exists in the current data collection.

```
class Edges : EqualityComparer<VertexE>
{
    public override bool Equals(VertexE first, VertexE second)
    {
        if (first == null && second == null)
            return true;
        else if (first == null || second == null)
            return false;
        if (first.X == second.X && first.Y == second.Y)
        {
            return true;
        }
        else if (first.Y == second.X && second.Y == first.X)
        {

            return true;
        }
        else
        {
            return false;
        }
    }
    public override int GetHashCode(VertexE obj)
    {
        int hCode = obj.X ^ obj.Y;
        return hCode.GetHashCode();
    }
}
```

Listing 7-2 Edge object equality comparer

The same comparison strategy was used for the rest of the objects such as vectors (x, y, z) and faces.

7.4 Update for the rid referencing model

A more complex data pattern requires a more sophisticated rid referencing model. In Chapter 4.2.3 defined rid referencing model is still applicable. Unfortunately, BSim Data Modeler ver. 0.0.4 was made to create rid referencing in structured sequences. Overall, model rid referencing was made in linear sequence such as *VECTOR3D, VERTEX, EDGE, FACE, CELL, ROOM,*

BUILDING, SITE. This is definitely not a best practice when the model gets more complex and data collections specified in Chapter 6.5 no longer fit the data schema defined in Chapter 4.2.

For this reason, BSim Data Modeler was rewritten through a deeper object inheritance instead of looping through separate data collections.

7.5 Additional data parameters for the BSim model

This subchapter is equally important as the rest of the topics discussed in this thesis. Furthermore, these sub-chapters aim to introduce the actual implementation and use of user-defined parameters in the BSim Data Modeler. Especially, the author targets particular areas where predefined templates could be reused for similar models or adapted to particular cases.

7.5.1 Construction data assignment

In Figure 7-4 presented construction linking methodology for partition constructions targets BSim database (mdb) as an external data container. For example, the engineer identifies or creates a new construction type in the BSim database to suit the required model needs. On the next step database and construction elements could be linked by unique SfB number. [10]

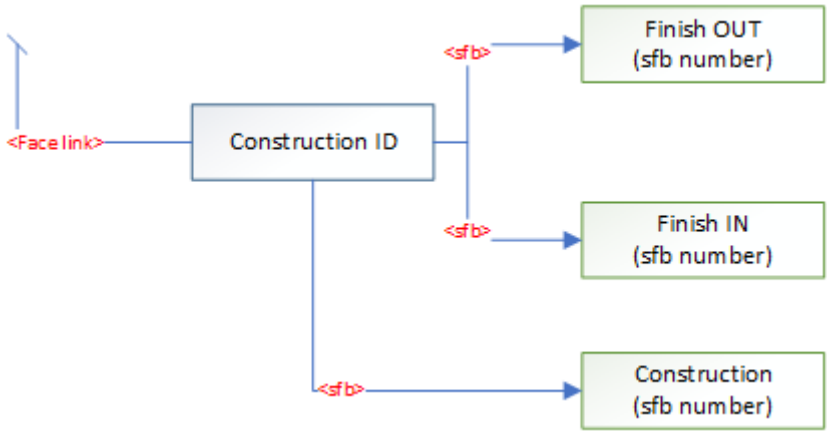

Figure 7-4 Implement external data linking for building partitions and windoor elements

The predefined template to achieve such functionalities in BSim Data Modeler could be found in APPENDIX I

7.5.2 Building systems

The similar element linking methodology that was presented in the previous Chapter 7.5.1 could be also used for building systems as well. Thus, allowing to reuse already defined templates for same and different BSim models at any stage of the project. Assuredly, this methodology would allow saving a significant amount of man-hours through the building design process.

In Figure 7-5 proposed solutions aim to indicate how systems could be grouped in predefined templates and linked to any BSim Model trough BSim Data Modeler. More specifically, this illustration aims to indicate an example of the *Ventilation* system template.

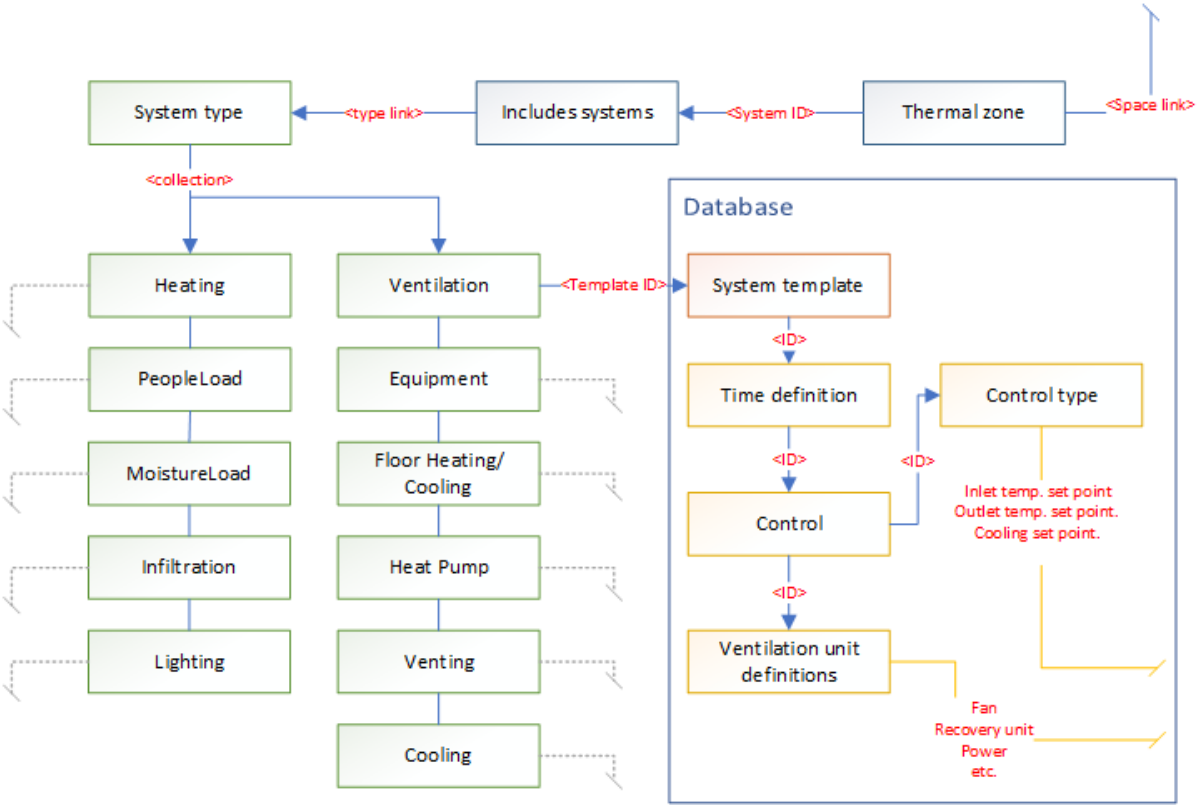

Figure 7-5 Proposed data linking methodology for building systems

First of all, to make it happen, it is important to choose an appropriate database solution where system-related information could be stored and occasionally modified upon a need. On the one side, Microsoft Excel itself could be used as a data container but with large amounts of data, it might be not the most optimal solution.

On the other hand, a database container could be used to store and manipulate the system data. There are a vast number of database containers that could be used to store building systems, such as Microsoft Access database, Microsoft SQL, MySQL, PostgreSQL… Implementation of such a database would allow manipulating data with a wide range of query capabilities. [24]

All in all, the modified templates could be linked with unique ids and supplied to BSim Data Modeler to build a final model.

7.6 Summary

This chapter introduced with a more thorough solid geometry representation that needs to be translated into the SimView topology model. A number of differences between two modelling tools forced to find a common solution for fluent data translation.

Firstly, the research started with solid geometric analysis. The outcome of the study showed that solids are individual objects in 3D space. Therefore, duplicated data in the output file is inevitable and has to be properly treated. Especially, as data overload causes larger and inconsistent models.

Secondly, the author gave a proposal on how the model surfaces, edges, and vertexes could be translated into SimView model, without any data copies. The proposed solution involves expanding BSim Data Modeler with additional libraries that would allow us to filter multiple data accuracies and assemble a new reference between geometry elements.

Finally, the chapter concluded with a proposal of how the additional parameters for BSim could be assigned. All in all, the author pointed out several parameters linking strategies that could improve model setup time as many templates can be reused and adapted from previous projects.

8 Conclusion

During the study period, there was a need to make numerous building energy performance models with BSim. The gained experience confirms the fact that BPS model preparation is a time-consuming task. Consequently, existing modelling practice is repetitive in many cases, as it requires to measure the room geometry in Autodesk Revit. Similar geometry often involves user interpretation and simplification. Moreover, building HVAC system setup in such rooms or other building models does not vary that much. Despite this fact, each individual model has to be set up with the SimView interface independently. Correspondingly, during the study years, building simulations were performed as soon as the key decisions had been made. Assuredly, a missing link between Autodesk Revit and BSim is of great importance, and the existence of such a tool will improve the overall design process at the early design stages.

The aim of this project was to develop a robust methodology that would create a missing link between Autodesk Revit and BSim. To do so, this project involved the usage of Dynamo, Excel, and custom .NET framework application.

As architectural models are not meant for the BPS and they do not fulfil the local requirements for the BSim model. The Dynamo was used to obtain and transform selected building spaces, as close to the BSim geometry model. Several model geometry alterations were investigated, it turned out that the wall intersection after the model boundary transformation is the most critical and often compromise the entire BSim model. Subsequently, these intersections cause the space solid overlap, this kind of room geometries continuously develop into small shared spaces. In essence, shared spaces require special evolutions. In fact, to move space solid surfaces it would require a sophisticated algorithm. Therefore, several methods were investigated and the most reasonable approach is to apply the boundary surface cropping tool. However, an engineer should decide upon cropping manner or deselect, join related spaces in the Autodesk Revit model.

Moreover, through the Dynamo space data script development process it was found that extruded space solids by the reference surface are not a perfect fit for BSim, as it does not have any relation to each other. Therefore, internal surfaces have to be evaluated with respect to the intersecting solid surfaces. The research showed that the best approach with the current methodology is to model vertical surfaces individually. However, Dynamo 1.3 base or nodes coded by MOE do not have such specific functionalities. For this reason, a set of Iron Python custom nodes was made to ensure data point integrity, to evaluate vertical walls and to assign it to proper space solids. After all the considerations were investigated, it was concluded, that the Iron Python node, which is responsible for the surface`s boundaries, should be extended.

By default, methods in the boundary surface evaluation node are set up to use a smaller room area as the cropping tool, the extension of this node would allow an engineer to select the manner how such surfaces should be cropped.

Furthermore, the second script that is responsible for collecting and extracting windoor data from Autodesk Revit is strongly depended on the window and door family's structure. The developed methodology is based on obtaining window and door openings. If more than two components are situated next to each other, such openings require geometry transformation. The designed script eliminates small intersecting areas which could be neglected. However, in the large windows, this area should be taken into consideration as it might become greater and reduce the window area more significantly.

Additionally, the combined window and door families visually look as two components while in Dynamo environment the translated data reveals its design flaws. The conducted investigations revealed that such widow/door components consist of two overlapping polygons that are not handled properly. Finally, it results in one merged polygon. Such opening elements should be handled by an engineer either by replacing families in the architectural model or model directly in SimView. As the window and the door openings are internal geometry representations, such geometries have to be plotted on an external surface. To do so, several possibilities were investigated. The most beneficial way is to extrude opening polygons as the solids and intersect them with a parallel surface. Due to the intersection methodology, the intersecting polygon's data points do not necessarily end up at the exact match of the surface plane. Fortunately, the BSim has methods to match openings coordinates on the closest face. For this reason, opening vector coordinates have to be supplied with the highest possible precision.

Data translation in this project has been just half of the work, as the exported data has to be assembled back to the 3D perspective model in SimView. For this reason, the best solution for the rid reference model is to move such a task to the external application which could create rid references, write dis (XML) file and perform error checking on the final model.

As solids do not have any relation to each other, the defined data collections contain duplicated data. The report suggested two possibilities to build dis (XML) model with individual solid objects, and perform solid data filtering. Both methods allow building BSim data models. Unfortunately, the use of the first method will cause reference loss, as the BSim topology cleanup methods will be forced to eliminate duplicates and create new unique XML tree elements. All in all, the author developed a methodology to filter solid data collections and included it in the BSim Data Modeler.

In addition, data collections were optimized throughout the methodology development process. A few possibilities were investigated in order to skip the *FINISH* element and the *NORMAL* element creation in the dis (XML) model. This is due to the fact, that SimView has methods to include such elements in the model. However, the research indicated that only normal elements can be left out as they only used for inner shell definition in the SimView model. On the other hand, the finish elements have to be created if there is a need to define the wall finish material in the BSim model.

Also, the author suggests external data linking methods for building constructions and HVAC systems. The implemented external data linking for the partition constructions aim to utilize the BSim mdb database to keep the construction data and assign it in excel tables by a unique number. The equivalent linking methodology is faster and more convenient than the original repetitive method in SimView. The presented construction's linking methodology is fully implemented in the BSim Data Modeler v. 0.1.7.0 Moreover, the author suggests to use a similar methodology for HVAC system linking, but to do so the database with related templates has to be created first in order to make such HVAC system linking possible.

In conclusion, the developed semi-automatic methodology allows for an engineer to select spaces in the architectural Revit model, set preferences for ideal model alterations, export model data to excel, define additional parameters and build a BSim model.

9 Future work

After being involved in the Dynamo environment not just theoretically but also practically, it can be concluded that Dynamo has some serious errors and a crashing tendency. Therefore, it should be considered to upgrade the current scripts from ver. 1.3 to the newest ver. 2.1. The newest Dynamo version would provide not just additional performance benefits, but it can also accommodate a script with newer nodes that are not present or sloppy in the current Dynamo version.

First of all, whether Dynamo is considered to be upgraded or not, to put the developed methodology to use, most custom Dynamo nodes should be updated and tested for different cases. Most importantly a custom node responsible for boundary surface crop should be upgraded to take into consideration more than two intersections. At the moment, it can handle only one intersection at the time. This will definitely be an issue for a complex model, as it will result in overlapping solids, as discussed in Chapter 6.2.1.

Secondly, during the research, it turned out that the window/door collection node is very sensitive and sometimes faulty. For this reason, this custom element collector node requires further improvements. Otherwise, some *WINDOOR* elements will be missing in the final BSim model.

Also, the BSim Data Modeler logic for HVAC systems are not complete. Therefore, to include HVAC system modelling in dis (XML), the tool has to be extended with additional methods that could handle the proposed linking method.

Additionally, the currently developed space data script does not support spaces at multiple levels at the same time. Consequently, spaces assigned at different levels will be assigned to the same building, but they will not be attached to the upper or lower spaces. To make this happen the script could be updated with a similar pattern by taking upper boundary segments or intersecting created solid with the upper ones. A new intersection method will be required again, as upper partitions will become internal partitions. In essence, such partitions must be shared between rooms, therefore it would require some additional surface cropping methods.

Finally, the current methodology could be extended to support inclined geometries that would expose a new possibility to model sloped constructions with openings on it.

10 Bibliography

[1] G.-K. P. R. K. a. P. F. Susan Solomon, “Irreversible climate change due to carbon dioxide emissions,” 28 01 2009. [Online]. Available: https://www.pnas.org/content/pnas/106/6/1704.full.pdf. [Accessed 18 12 2019].

[2] “ European Parliament,” 03 10 2019. [Online]. Available: https://www.europarl.europa.eu/news/en/headlines/society/20190926STO62270/what-is-carbon-neutrality-and-how-can-it-be-achieved-by-2050. [Accessed 18 12 2019].

[3] “Energy performance of buildings,” Europian Commision, 6 12 2019. [Online]. Available: https://ec.europa.eu/energy/en/topics/energy-efficiency/energy-performance-of-buildings/overview. [Accessed 18 12 2019].

[4] J. W. H. M. K. a. B. T. G. Drury B. Crawley, “Contrasting the capabilities of building energy performance simulation programs,” U S Department of Energy, Washington, DC, USA, Energy Systems Research Unit, University of Strathclyde, Glasgow, Scotland, UK, University of Wisconsin-Madison, Solar Energy Laboratory, Madison, Wisconsin, USA, National Renewable Energy Laboratory, Golden,, USA, UK, 2005.

[5] P. Vladimir Bazjanac, “IFC BIM-Based Methodology for Semi-Automated Building Energy,” Building Technologies Department Environmental Energy Technologies Division Lawrence Berkeley National Laboratory University of California Berkeley, CA 94720, California, 2010.

[6] W. D. 2. OFFICE of ENERGY EFFICIENCY & RENEWABLE ENERGY, “Autodesk Brings Detailed EnergyPlus HVAC Simulation to Revit,” 30 September 2019. [Online]. Available: https://www.energy.gov/eere/buildings/articles/autodesk-brings-detailed-energyplus-hvac-simulation-revit. [Accessed 10 11 2019].

[7] A. Andriamamonjy, D. Saelens and R. Klein, An automated IFC-based workflow for building energy performance simulation with Modelica, 2018.

[8] M. R. A. A. M. W. Y. Michael Bergin, “BIM-basedParametricBuildingEnergyPerformanceMulti-Objective Optimization,” in *eCAADe Education and Research in Computer Aided Architectural Design in Europe*, 2014.

[9] G. N. L. G. K. M. Á. G.-F. Á. G.-F. Georgios I. Giannakis, “A methodology to automatically generate geometry inputs for Energy,” 1 December 2015. [Online]. Available: https://www.researchgate.net/publication/286452825. [Accessed 21 12 2019].

[10] “The technology behind gbXML,” Green Building XML Inc., 2020. [Online]. Available: https://www.gbxml.org/About_GreenBuildingXML_gbXML. [Accessed 12 12 2019].

[11] S. Byggeforskningsinstitut, “Statens Byggeforskningsinstitut -Sbi,” 1997-2013. [Online]. Available: https://sbi.dk/bsim/Documents/PDF-docs/BSim%20brochure.pdf. [Accessed 23 09 2019].

[12] D. B. Crawley, J. W. Hand, M. Kummert and B. T. Griffith, “Contrasting the capabilities of building energy performance simulation programs,” *BSim version 1.0,* p. 3, July 2005.

[13] K. J. K. S. K. a. R. J. Wittchen, BSim User's Guide, version 7.13.10.1, Copenhagen: Danish Building Research Institute (SBi), 1999-2013.

[14] liam, “w3,” W3C, 2013-2015. [Online]. Available: https://www.w3.org/XML/. [Accessed 16 09 2019].

[15] R. Data, “w3school,” Refsnes Data, 1999-2019. [Online]. Available: https://www.w3schools.com/xml/xml_whatis.asp. [Accessed 16 09 2019].

[16] D. Obasanjo, “XML.COM,” O'Reilly Media, Inc., 1998 - 2008. [Online]. Available: https://www.xml.com/pub/a/2004/07/21/design.html. [Accessed 18 09 2019].

[17] A. Microsoft, “Office.com,” Microsoft , 1992. [Online]. Available: https://products.office.com/en-us/access. [Accessed 18 09 2019].

[18] B.-E. Foundation, “Byggetekniske erfaringer,” BYG-ERFA, 2019. [Online]. Available: https://byg-erfa.dk/en/node/5278. [Accessed 18 09 2019].

[19] B. V. Borrmann A., Principles of Geometric Modeling. In: Borrmann A., König M., Koch C., Beetz J. (eds) Building Information Modeling. Springer, Cham, Springer, Cham, 2018.

[20] none, “Dynamo Library,” primer.dynamobim.org, 2018. [Online]. Available: https://primer.dynamobim.org/03_Anatomy-of-a-Dynamo-Definition/3-3_dynamo_libraries.html. [Accessed 01 12 2019].

[21] S. B. -. SBi, Interviewee, *BSim topology model.* [Interview]. 6 11 2019.

[22] D. S. foundations, “Dansk Standart -DS418,” 2011. [Online]. Available: https://www.ds.dk/da. [Accessed 10 12 2019].

[23] D. Venkov, “Github SpringNodes,” Individual, 9 12 2018. [Online]. Available: https://github.com/dimven/SpringNodes/. [Accessed 14 12 2019].

[24] Kirti_Mangal, “C# | Object.GetHashCode() Method with Examples,” Geeks for Geeks, [Online]. Available: https://www.geeksforgeeks.org/c-sharp-object-gethashcode-method-with-examples/. [Accessed 13 12 2019].

[25] W. Solihin, C. Eastman, Y.-C. Lee and D.-H. Yang, “A Simplified Relational Database Schema for Transformation of BIM Data into a Query-Efficient and Spatially Enabled Database.,” *Automation in Construction 84* , p. 367–383. Web., 2017 December.

[26] Author, “Wikipedia,” Wikimedia Foundation, Inc., 15 01 2001. [Online]. Available: https://en.wikipedia.org/wiki/Wire-frame_model. [Accessed 18 09 2019].

[27] Author, “Vocabulary,” Vocabulary.com, 2019. [Online]. Available: https://www.vocabulary.com/dictionary/parse. [Accessed 18 09 2019].

[28] A. XML_namespace, “Wikipedia,” Wikimedia Foundation, Inc, 15 01 2001. [Online]. Available: https://en.wikipedia.org/wiki/XML_namespace. [Accessed 21 09 2019].

11 APPENDIX

Table of contents

A. XML hierarchical tree

1. Building hierarchical tree

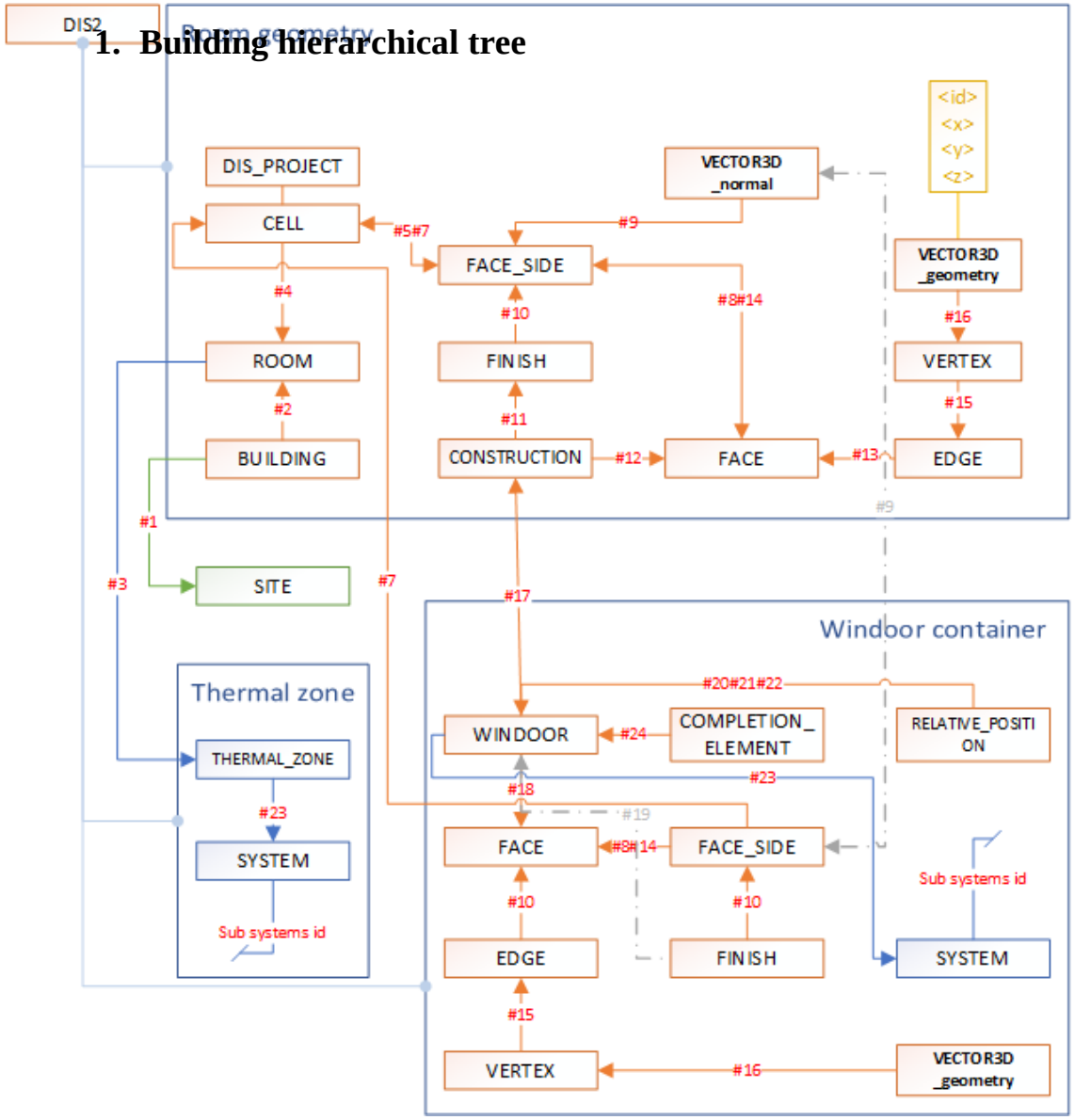

Table of references

ID	Element referenced by node:	
#1	<located_on_site>#1</located_on_site>	/BUILDING
#2	<composed_of>#2</composed_of>	/BUILDING
#3	<has_thermal_zones>#3</has_thermal_zones>	/BUILDING
#4	<represented_by_cell>#4<represented_by_cell>	/ROOM
#5	<bounded_by>#5</bounded_by>	/CELL
#6	<has_normal>#6</has_normal>	/FACE_SIDE
#7	<faces_cell>#7</faces_cell>	/FACE_SIDE
#8	<has_face>#8</has_face>	/FACE_SIDE
#9	<has_normal>#9</has_normal>	/FACE_SIDE
#10	<represented_by>#10</represented_by>	/FINISH
#11	<has_finish>#11</has_finish>	/CONSTRUCTION
#12	<represented_by>#12</represented_by>	/CONSTRUCTION
#13	<has_edge>#13</has_edge>	/FACE
#14	<has_face_side>#14</has_face_side>	/FACE
#15	<has_vertex>#15</has_vertex>	/EDGE
#16	<has_geometry>#16</has_geometry>	/VERTEX
#17	<includes_segments>#17</includes_segments>	/CONSTRUCTION
#18	<represented_by>#18</represented_by>	/WINDOOR
#19	<has_finish>#19</has_finish>	/WINDOOR
#20	<overhang>#20</overhang>	/WINDOOR
#21	<left_sidefin>#21</left_sidefin>	/WINDOOR
#22	<right_sidefin>#22</right_sidefin>	/WINDOOR
#23	<has_service>#23</has_service>	/WINDOOR and THERMAL_ZONE
#24	<of_type>#24</of_type>	/WINDOOR

2. SimView window hierarchical tree

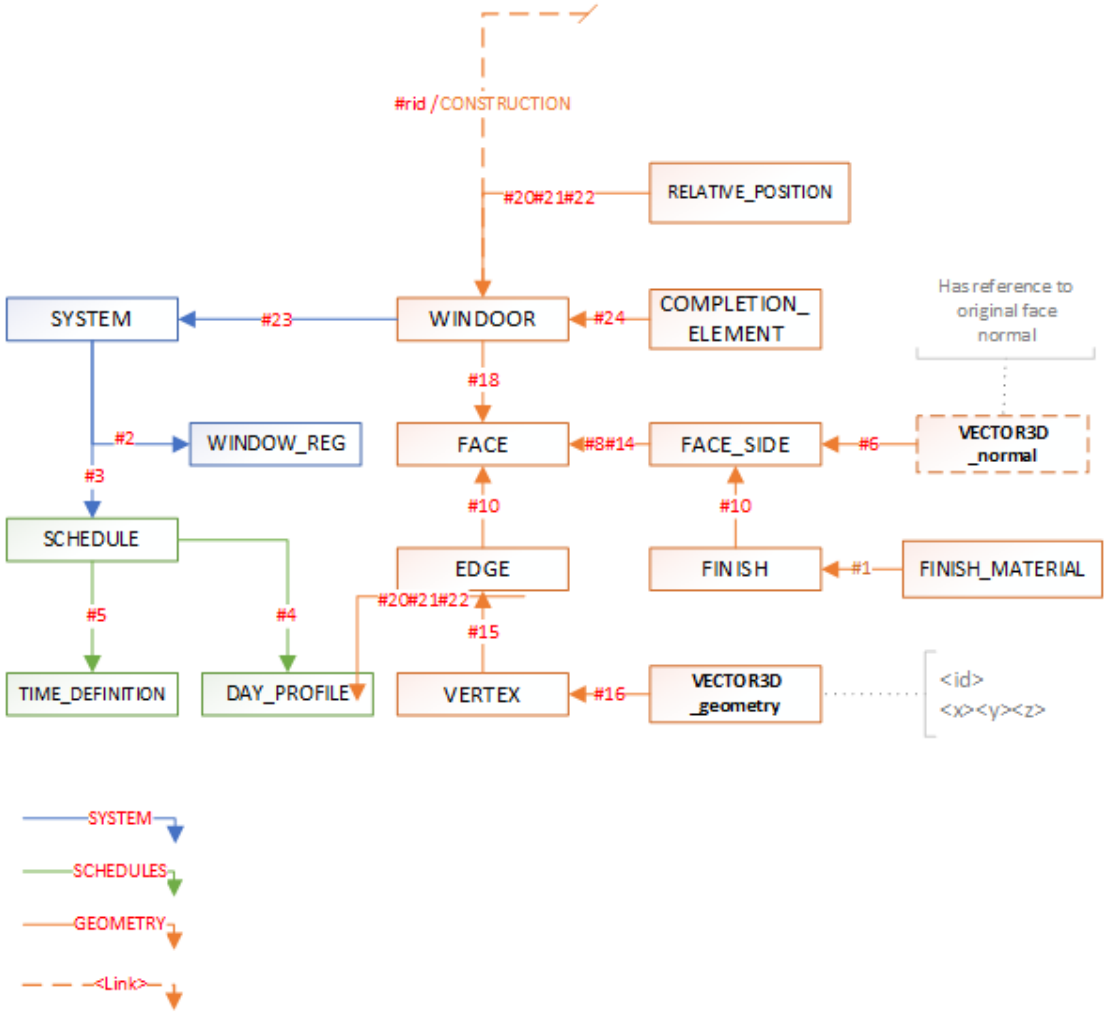

Table of references

ID	Element referenced by node:	
#1	<made_of>#1</made_of>	/FINISH
#2	<has_component>#2</has_component>	/SYSTEM
#3	<has_schedule>#3</has_schedule>	/SYSTEM
#4	has_control>#4</has_control>	/SCHEDULE
#5	<has_time_definition>#5</has_time_definition>	/SCHEDULE
#6	<has_normal>#6</has_normal>	/FACE_SIDE
#7	<faces_cell>#7</faces_cell>	/FACE_SIDE
#8	<has_face>#8</has_face>	/FACE_SIDE
#9	<has_normal>#9</has_normal>	/FACE_SIDE
#10	<represented_by>#10</represented_by>	/FINISH
#11	<has_finish>#11</has_finish>	/CONSTRUCTION
#12	<represented_by>#12</represented_by>	/CONSTRUCTION
#13	<has_edge>#13</has_edge>	/FACE
#14	<has_face_side>#14</has_face_side>	/FACE
#15	<has_vertex>#15</has_vertex>	/EDGE
#16	<has_geometry>#16</has_geometry>	/VERTEX
#17	<includes_segments>#17</includes_segments>	/CONSTRUCTION
#18	<represented_by>#18</represented_by>	/WINDOOR
#19	<has_finish>#19</has_finish>	/WINDOOR
#20	<overhang>#20</overhang>	/WINDOOR
#21	<left_sidefin>#21</left_sidefin>	/WINDOOR
#22	<right_sidefin>#22</right_sidefin>	/WINDOOR
#23	<has_service>#23</has_service>	/WINDOOR
#24	<of_type>#24</of_type>	/WINDOOR

B. XML schema investigation

In Chapter 3 discussed model was continuously extended to analyse how the SimView creates elements that form the unified model. The final model 3D geometry is presented in Figure 11-1

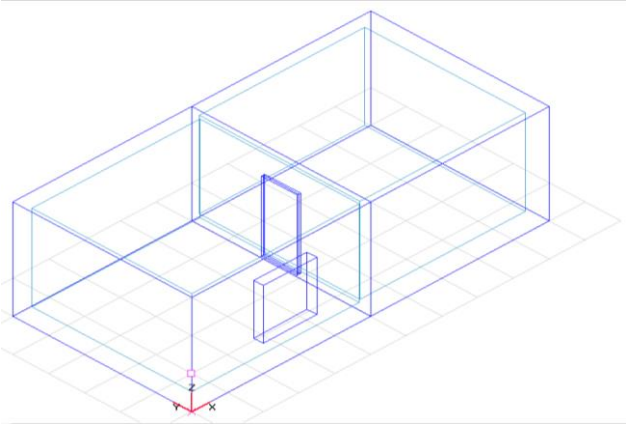

Figure 11-1 Analysed BSim model

BSim's dis (XML) file includes heading that contains FILE_DESCRIPTION and FILE_NAME nodes. The example of this heading is presented in Listing 11-2 Analysed model XML schema, part I -VI.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- FILE_DESCRIPTION(('DIS2 model', 'STEP.DLL version 7, 7, 10, 29','SBI','BSim,
Version 7.16.8.11(BS7121-255)')); -->
<!-- FILE_NAME('F:\Failai\AAU\4th-semester\1.Bsim file structure\1.Box
model\TestModel.disxml','26.09.2019 14.26'); -->
```

Listing 11-1 BSim dis (XML) file heading

However, not all of the represented elements in Figure must be included in dis (XML) to form the building's geometry. It is strongly depended on how detailed the model must be.

For example, the simplest model, that would only represent room solid, does not need to include elements such as:(see Figure 11-2).

- WINDOOR,
- RELATIVE_POSITION,
- CONSTRUCTION_ELEMENT,
- COMPLETION_ELEMENT,
- FINISH_MATERIAL,
- SITE,
- LOCATION,
- THERMAL_ZONE, SYSTEM,
- PARAM_LIST.

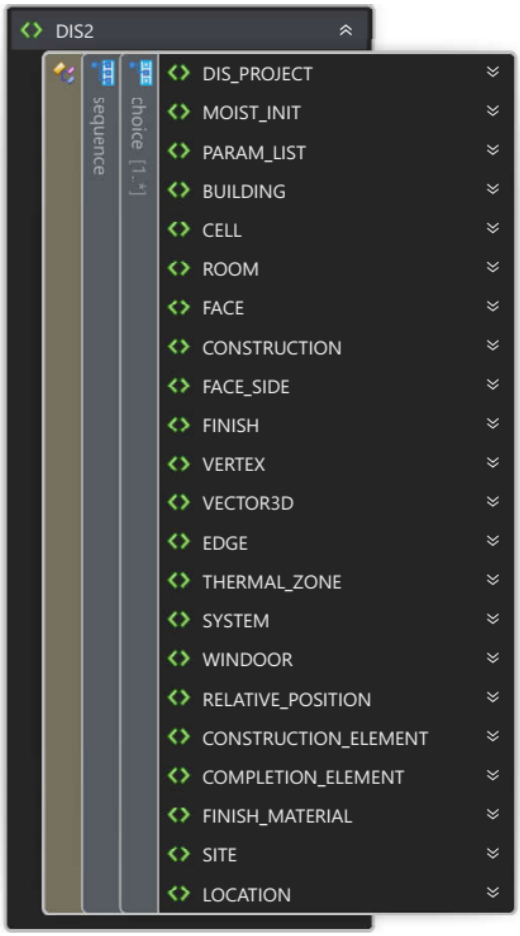

Figure 11-2 BSim hierarchical tree elements

1. XML SCHEMA

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DIS2">
    <xs:complexType>
      <xs:sequence>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="DIS_PROJECT">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" type="xs:string" />
                <xs:element name="description" />
                <xs:element name="database" type="xs:string" />
                <xs:element name="tstep" type="xs:int" />
                <xs:element name="options" type="xs:int" />
                <xs:element name="scale" type="xs:decimal" />
                <xs:element name="grid" type="xs:decimal" />
                <xs:element name="layer_thick" type="xs:decimal" />
                <xs:element name="start_time" type="xs:date" />
                <xs:element name="end_time" type="xs:date" />
                <xs:element name="has_design_parm" type="xs:string" />
              </xs:sequence>
              <xs:attribute name="rid" type="xs:string" use="required" />
            </xs:complexType>
          </xs:element>
          <xs:element name="MOIST_INIT">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" />
                <xs:element name="default_rh" type="xs:float" />
                <xs:element name="default_t" type="xs:float" />
                <xs:element name="default_first_lay" type="xs:float" />
                <xs:element name="default_expansion" type="xs:decimal" />
                <xs:element name="mc" type="xs:string" />
                <xs:element name="has_const_list" />
              </xs:sequence>
              <xs:attribute name="rid" type="xs:string" use="required" />
            </xs:complexType>
          </xs:element>
          <xs:element name="PARAM_LIST">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" type="xs:string" />
                <xs:element name="has_params" />
              </xs:sequence>
              <xs:attribute name="rid" type="xs:string" use="required" />
            </xs:complexType>
          </xs:element>
          <xs:element name="BUILDING">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="id" type="xs:string" />
                <xs:element name="rotation" type="xs:float" />
                <xs:element name="current" type="xs:float" />
                <xs:element name="height" type="xs:float" />
                <xs:element name="composed_of" type="xs:string" />
                <xs:element name="located_on_site" type="xs:string" />
                <xs:element name="has_thermal_zones" type="xs:string" />
```

Listing 11-2 Analysed model XML schema, part I

```
59              </xs:sequence>
60              <xs:attribute name="rid" type="xs:string" use="required" />
61            </xs:complexType>
62          </xs:element>
63          <xs:element name="CELL">
64            <xs:complexType>
65              <xs:sequence>
66                <xs:element name="id" type="xs:string" />
67                <xs:element name="volume" type="xs:decimal" />
68                <xs:element name="bounded_by" type="xs:string" />
69              </xs:sequence>
70              <xs:attribute name="rid" type="xs:string" use="required" />
71            </xs:complexType>
72          </xs:element>
73          <xs:element name="ROOM">
74            <xs:complexType>
75              <xs:sequence>
76                <xs:element name="id" type="xs:string" />
77                <xs:element name="represented_by_cell" type="xs:string" />
78                <xs:element name="ref_x" type="xs:float" />
79                <xs:element name="ref_y" type="xs:float" />
80                <xs:element name="ref_z" type="xs:float" />
81                <xs:element name="has_temp" type="xs:string" />
82                <xs:element name="behave_like" type="xs:string" />
83                <xs:element name="has_type" type="xs:string" />
84                <xs:element name="has_inner_shell" type="xs:string" />
85                <xs:element name="has_refpoint" />
86              </xs:sequence>
87              <xs:attribute name="rid" type="xs:string" use="required" />
88            </xs:complexType>
89          </xs:element>
90          <xs:element name="FACE">
91            <xs:complexType>
92              <xs:sequence>
93                <xs:element name="id" type="xs:string" />
94                <xs:element name="area" type="xs:decimal" />
95                <xs:element name="round" type="xs:decimal" />
96                <xs:element name="has_edge" type="xs:string" />
97                <xs:element name="has_face_side" type="xs:string" />
98              </xs:sequence>
99              <xs:attribute name="rid" type="xs:string" use="required" />
100           </xs:complexType>
101         </xs:element>
102         <xs:element name="CONSTRUCTION">
103           <xs:complexType>
104             <xs:sequence>
105               <xs:choice maxOccurs="unbounded">
106                 <xs:element name="id" type="xs:string" />
107                 <xs:element name="displacement" type="xs:float" />
108                 <xs:element name="hz0" type="xs:float" />
109                 <xs:element name="sfb" />
110                 <xs:element name="u_value" type="xs:decimal" />
111                 <xs:element name="exposure" type="xs:float" />
112                 <xs:element name="represented_by" type="xs:string" />
113                 <xs:element name="includes_segments" type="xs:string" />
114                 <xs:element name="has_finish" type="xs:string" />
115                 <xs:element name="thickness" type="xs:decimal" />
```

Listing 11-3 Analysed model XML schema, part II

```
116                 <xs:element name="of_type" type="xs:string" />
117                 <xs:element name="has_thermal_bridge" />
118               </xs:choice>
119             </xs:sequence>
120             <xs:attribute name="rid" type="xs:string" use="required" />
121           </xs:complexType>
122         </xs:element>
123         <xs:element name="FACE_SIDE">
124           <xs:complexType>
125             <xs:sequence>
126               <xs:element name="id" type="xs:string" />
127               <xs:element name="locked" type="xs:float" />
128               <xs:element name="has_normal" type="xs:string" />
129               <xs:element name="faces_cell" type="xs:string" />
130               <xs:element name="has_face" type="xs:string" />
131             </xs:sequence>
132             <xs:attribute name="rid" type="xs:string" use="required" />
133           </xs:complexType>
134         </xs:element>
135         <xs:element name="FINISH">
136           <xs:complexType>
137             <xs:sequence>
138               <xs:element name="id" type="xs:string" />
139               <xs:element name="r" type="xs:float" />
140               <xs:element name="z" type="xs:float" />
141               <xs:element name="rc" type="xs:float" />
142               <xs:element name="filtration" type="xs:float" />
143               <xs:element name="represented_by" type="xs:string" />
144               <xs:element name="made_of" type="xs:string" />
145               <xs:element name="facing" type="xs:string" />
146             </xs:sequence>
147             <xs:attribute name="rid" type="xs:string" use="required" />
148           </xs:complexType>
149         </xs:element>
150         <xs:element name="VERTEX">
151           <xs:complexType>
152             <xs:sequence>
153               <xs:element name="id" type="xs:string" />
154               <xs:element name="has_geometry" type="xs:string" />
155             </xs:sequence>
156             <xs:attribute name="rid" type="xs:string" use="required" />
157           </xs:complexType>
158         </xs:element>
159         <xs:element name="VECTOR3D">
160           <xs:complexType>
161             <xs:sequence>
162               <xs:element name="id" type="xs:string" />
163               <xs:element name="x" type="xs:decimal" />
164               <xs:element name="y" type="xs:decimal" />
165               <xs:element name="z" type="xs:decimal" />
166             </xs:sequence>
167             <xs:attribute name="rid" type="xs:string" use="required" />
168           </xs:complexType>
169         </xs:element>
170         <xs:element name="EDGE">
171           <xs:complexType>
172             <xs:sequence>
```

Listing 11-4 Analysed model XML schema, part III

```
172                 <xs:element name="id" type="xs:string" />
173                 <xs:element name="edge_length" type="xs:decimal" />
174                 <xs:element name="has_vertex" type="xs:string" />
175               </xs:sequence>
176               <xs:attribute name="rid" type="xs:string" use="required" />
177             </xs:complexType>
178           </xs:element>
179           <xs:element name="THERMAL_ZONE">
180             <xs:complexType>
181               <xs:sequence>
182                 <xs:element name="id" type="xs:string" />
183                 <xs:element name="solar_to_air_fact" type="xs:decimal" />
184                 <xs:element name="solar_to_ceil_ratio" type="xs:decimal" />
185                 <xs:element name="solar_to_wall_ratio" type="xs:decimal" />
186                 <xs:element name="solar_to_floor_ratio" type="xs:decimal" />
187                 <xs:element name="solar_lost_fact" type="xs:decimal" />
188                 <xs:element name="kappa" type="xs:float" />
189                 <xs:element name="top_height" type="xs:decimal" />
190                 <xs:element name="top_frac" type="xs:decimal" />
191                 <xs:element name="composed_of" type="xs:string" />
192                 <xs:element name="has_service" type="xs:string" />
193               </xs:sequence>
194               <xs:attribute name="rid" type="xs:string" use="required" />
195             </xs:complexType>
196           </xs:element>
197           <xs:element name="SYSTEM">
198             <xs:complexType>
199               <xs:sequence>
200                 <xs:element name="id" type="xs:string" />
201                 <xs:element name="active" type="xs:float" />
202                 <xs:element name="has_component" type="xs:string" />
203                 <xs:element name="has_schedule" />
204                 <xs:element name="system_type" type="xs:string" />
205               </xs:sequence>
206               <xs:attribute name="rid" type="xs:string" use="required" />
207             </xs:complexType>
208           </xs:element>
209           <xs:element name="WINDOOR">
210             <xs:complexType>
211               <xs:sequence>
212                 <xs:choice maxOccurs="unbounded">
213                   <xs:element name="id" type="xs:string" />
214                   <xs:element name="displacement" type="xs:float" />
215                   <xs:element name="hz0" type="xs:float" />
216                   <xs:element name="sfb" />
217                   <xs:element name="u_value" type="xs:decimal" />
218                   <xs:element name="exposure" type="xs:float" />
219                   <xs:element name="represented_by" type="xs:string" />
220                   <xs:element name="includes_segments" />
221                   <xs:element name="has_finish" type="xs:string" />
222                   <xs:element name="thickness" type="xs:float" />
223                   <xs:element name="of_type" type="xs:string" />
224                   <xs:element name="has_thermal_bridge" />
225                   <xs:element name="offset" type="xs:float" />
226                   <xs:element name="frame_area" type="xs:float" />
227                   <xs:element name="panel_area" type="xs:float" />
228                   <xs:element name="sf1" type="xs:float" />
```

Listing 11-5 Analysed model XML schema, part IV

```
229                 <xs:element name="sf2" type="xs:float" />
230                 <xs:element name="sf3" type="xs:float" />
231                 <xs:element name="round" type="xs:float" />
232                 <xs:element name="cd_coeff" type="xs:decimal" />
233                 <xs:element name="cnt_frac" type="xs:decimal" />
234                 <xs:element name="a_frac" type="xs:float" />
235                 <xs:element name="ka_coeff" type="xs:float" />
236                 <xs:element name="overhang" type="xs:string" />
237                 <xs:element name="left_sidefin" type="xs:string" />
238                 <xs:element name="right_sidefin" type="xs:string" />
239                 <xs:element name="has_service" />
240               </xs:choice>
241             </xs:sequence>
242             <xs:attribute name="rid" type="xs:string" use="required" />
243           </xs:complexType>
244         </xs:element>
245         <xs:element name="RELATIVE_POSITION">
246           <xs:complexType>
247             <xs:sequence>
248               <xs:element name="id" />
249               <xs:element name="depth" type="xs:float" />
250               <xs:element name="distance" type="xs:float" />
251             </xs:sequence>
252             <xs:attribute name="rid" type="xs:string" use="required" />
253           </xs:complexType>
254         </xs:element>
255         <xs:element name="CONSTRUCTION_ELEMENT">
256           <xs:complexType>
257             <xs:sequence>
258               <xs:element name="id" type="xs:string" />
259               <xs:element name="name" type="xs:string" />
260               <xs:element name="sfb" type="xs:string" />
261               <xs:element name="unit" />
262               <xs:element name="lifetime" type="xs:float" />
263               <xs:element name="thickness" type="xs:decimal" />
264               <xs:element name="resistance" type="xs:decimal" />
265               <xs:element name="composed_of" />
266             </xs:sequence>
267             <xs:attribute name="rid" type="xs:string" use="required" />
268           </xs:complexType>
269         </xs:element>
270         <xs:element name="COMPLETION_ELEMENT">
271           <xs:complexType>
272             <xs:sequence>
273               <xs:element name="id" type="xs:string" />
274               <xs:element name="name" type="xs:string" />
275               <xs:element name="sfb" type="xs:string" />
276               <xs:element name="unit" type="xs:string" />
277               <xs:element name="lifetime" type="xs:float" />
278               <xs:element name="thickness" type="xs:float" />
279               <xs:element name="resistance" type="xs:float" />
280               <xs:element name="composed_of" />
281             </xs:sequence>
282             <xs:attribute name="rid" type="xs:string" use="required" />
283           </xs:complexType>
284         </xs:element>
285         <xs:element name="FINISH_MATERIAL">
```

Listing 11-6 Analysed model XML schema, part V

```
286           <xs:complexType>
287             <xs:sequence>
288               <xs:element name="id" />
289               <xs:element name="sfb" type="xs:float" />
290               <xs:element name="unit" type="xs:string" />
291               <xs:element name="density" type="xs:unsignedShort" />
292               <xs:element name="name" type="xs:string" />
293               <xs:element name="emissivity" type="xs:decimal" />
294               <xs:element name="solar_absorptance" type="xs:decimal" />
295               <xs:element name="reflectance" type="xs:decimal" />
296               <xs:element name="surface_resist" type="xs:decimal" />
297               <xs:element name="specularity" type="xs:decimal" />
298               <xs:element name="roughness" type="xs:decimal" />
299               <xs:element name="red" type="xs:decimal" />
300               <xs:element name="green" type="xs:decimal" />
301               <xs:element name="blue" type="xs:decimal" />
302             </xs:sequence>
303             <xs:attribute name="rid" type="xs:string" use="required" />
304           </xs:complexType>
305         </xs:element>
306         <xs:element name="SITE">
307           <xs:complexType>
308             <xs:sequence>
309               <xs:element name="id" type="xs:string" />
310               <xs:element name="weather_file" />
311               <xs:element name="refl_rad" type="xs:decimal" />
312               <xs:element name="refl_light" type="xs:decimal" />
313               <xs:element name="horizon" type="xs:float" />
314               <xs:element name="emissivity" type="xs:decimal" />
315               <xs:element name="co2" type="xs:unsignedShort" />
316               <xs:element name="terrain" type="xs:float" />
317               <xs:element name="has_location" type="xs:string" />
318               <xs:element name="has_ground" type="xs:string" />
319             </xs:sequence>
320             <xs:attribute name="rid" type="xs:string" use="required" />
321           </xs:complexType>
322         </xs:element>
323         <xs:element name="LOCATION">
324           <xs:complexType>
325             <xs:sequence>
326               <xs:element name="id" />
327               <xs:element name="latitude" type="xs:float" />
328               <xs:element name="longitude" type="xs:float" />
329               <xs:element name="time_zone" type="xs:float" />
330               <xs:element name="elevation" type="xs:float" />
331             </xs:sequence>
332             <xs:attribute name="rid" type="xs:string" use="required" />
333           </xs:complexType>
334         </xs:element>
335       </xs:choice>
336     </xs:sequence>
337   </xs:complexType>
338 </xs:element>
339 xs:schema>
```

Listing 11-7 Analysed model XML schema, part VI

C. Manual model construction data tables

1. Space data – manual input method

VeretxID	X	Y	Z	EDGE	EdgeId	Face consists of edges	Face id	FACE_ SIDE_Inward	FACE_ SIDE_Outward	Link to face normal	Normal point	FinID	FINISH	CELL	RoomID	CONid	CONSTRUCTION	haswindow	cell_bound	Has defined wall
1	0	0	0	1 2	1	1 2 3 4	1	1	0	0	100	1	1	1 2 3 4 5 6	1	1	1 7	2	0	21.10.21
2	0	5	0	2 3	2	5 6 7 8	2	2	0	0	200	2	2	0	0	2	2 8	2	0	21.10.21
3	0	5	5	3 4	3	9 4 10 8	3	3	0	0	020	3	3	0	0	3	3 9	2	0	21.10.22
4	0	0	5	4 1	4	10 3 11 7	4	4	0	0	002	4	4	0	0	4	4 10	2	0	27.10.00
5	5	0	0	5 6	5	1 12 5 9	5	5	0	0	001	5	6	0	0	5	6 12	0	0	23.10.01
6	5	5	0	6 7	6	11 2 12 6	6	6	0	0	020	6	5	7 6 8 9 10 11	0	6	5 11	0	3	22.10.10
7	5	5	5	7 8	7	13 2 14 15	7	7	0	0	100	7	7	0	0	0	0	2	0	27.10.00
8	5	0	5	8 5	8	14 18 17 16	8	8	0	0	020	8	8	0	0	0	0	0	0	21.10.21
9	0	13	0	1 5	9	20 6 19 17	9	9	0	0	200	9	9	0	0	0	0	0	0	21.10.21
10	0	13	5	4 8	10	13 16 19 12	10	10	0	0	001	10	10	0	0	0	0	0	0	21.10.21
11	5	13	5	7 3	11	15 18 20 11	11	11	0	0	002	11	11	0	0	0	0	0	0	23.10.01
12	5	13	0	2 6	12		0	0	0	0	0	12	12	0	0	0	0	0	0	0
0	0	0	0	2 9	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	9 10	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	10 3	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	9 12	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	12 11	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	11 10	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	12 6	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	11 7	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0

Export_Data
(Check output folder)

Table 11-1 Space data collections for manual BSim modelling

2. Windoor data – manual input method

VeretxID	X	Y	Z	EDGE	EdgeId	Face consists of edges	Face has window	LinktoFace	has defined window
1	0	2	1	1 2	1	1 2 3 4	2	1	31.01.11
2	0	4	1	2 3	2	5 6 7 8	2	2	31.01.12
3	0	4	2	3 4	3	13 14 15 16	2	3	31.01.13
4	0	2	2	4 1	4	9 10 11 12	2	4	31.01.14
5	5	2	0.5	5 6	5	0	0	5	31.01.14
6	5	3	0.5	6 7	6	0	0	0	0
7	5	3	4	7 8	7	17 18 19 20	2	0	31.01.11
8	5	2	4	8 5	8	0	0	0	0
9	2	2	5	9 10	9	0	0	0	0
10	2	4	5	10 11	10	0	0	0	0
11	4	4	5	11 12	11	0	0	0	0
12	4	2	5	12 9	12	0	0	0	0
13	2	0	0.5	13 14	13	0	0	0	0
14	1	0	0.5	14 15	14	0	0	0	0
15	1	0	4	15 16	15	0	0	0	0
16	2	0	4	16 13	16	0	0	0	0
17	0	6	0.5	17 18	17	0	0	0	0
18	0	11	0.5	18 19	18	0	0	0	0
19	0	11	4	19 20	19	0	0	0	0
20	0	6	4	20 17	20	0	0	0	0

Table 11-2 Windoor data collections for manual BSim modelling

D. Dynamo models

1. Dynamo - Space Model

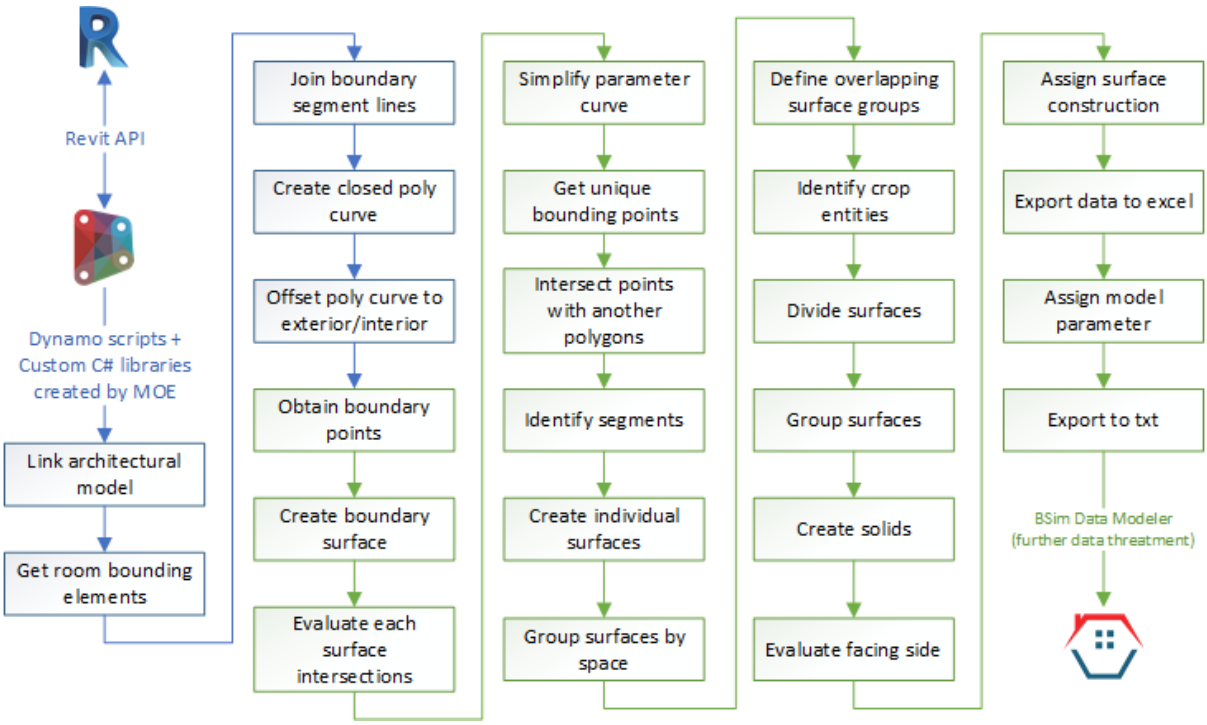

Figure 11-3 Designed architectural model translation to space solids.

2. Dynamo - Windoor Model

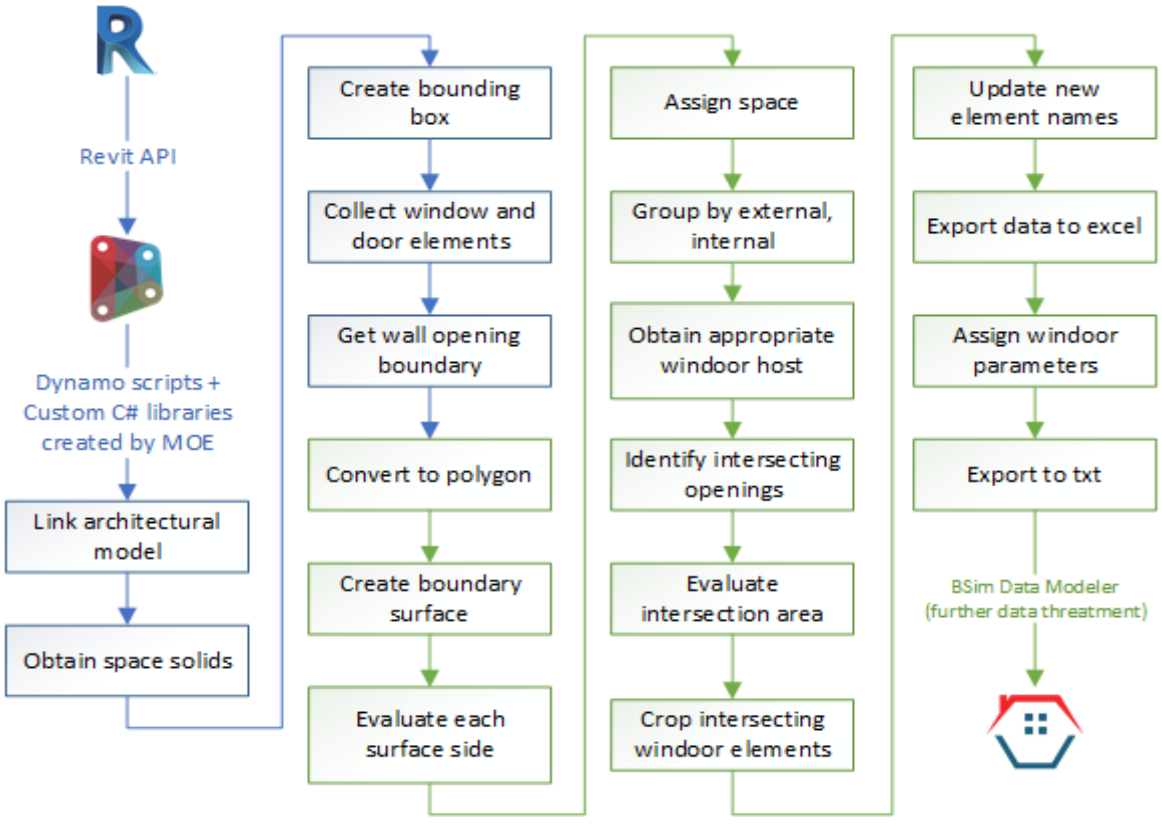

Figure 11-4 Designed architectural opening translation to individual surface segments

E. Dynamo data model by Moe

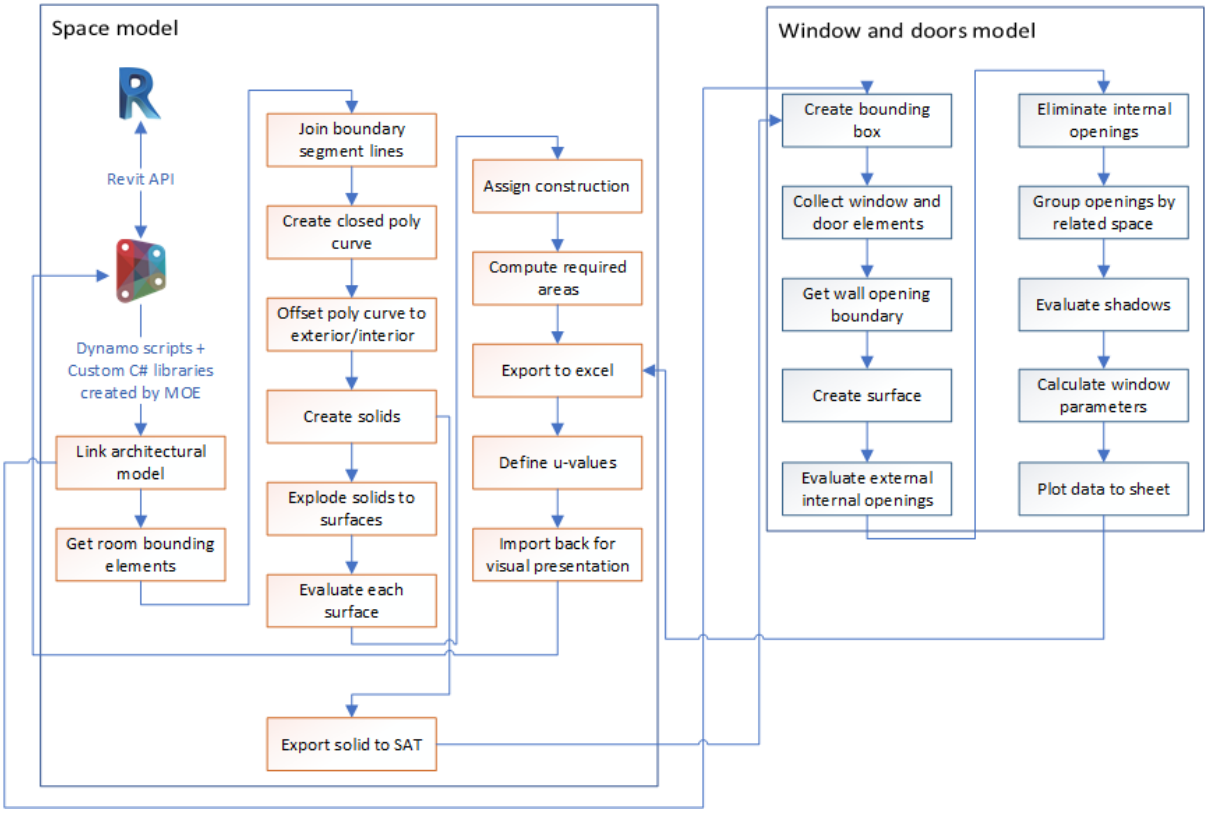

Figure 11-5 Heat loss calculation methodology

F. Investigated architectural model

1. Parcelhus_HusCompagniet_2019 – Revit Model from MOE

3D view

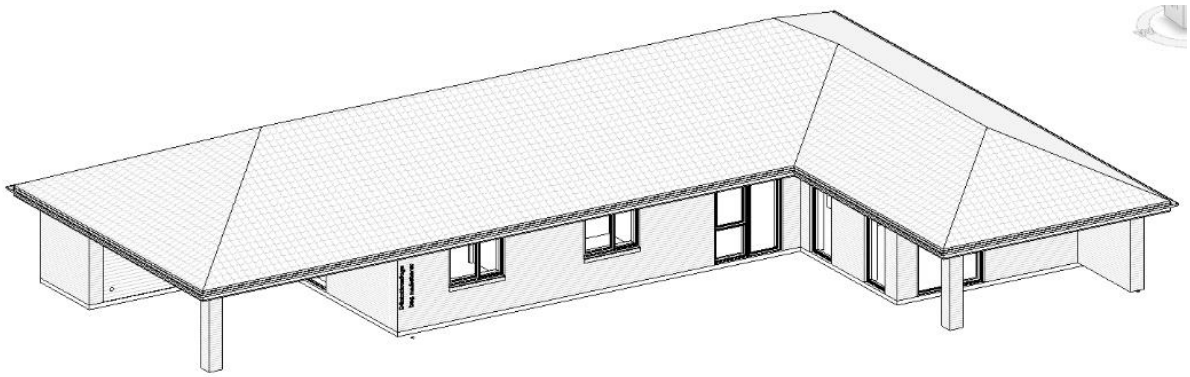

Figure 11-6 Parcelhus 3D view in Autodesk Revit

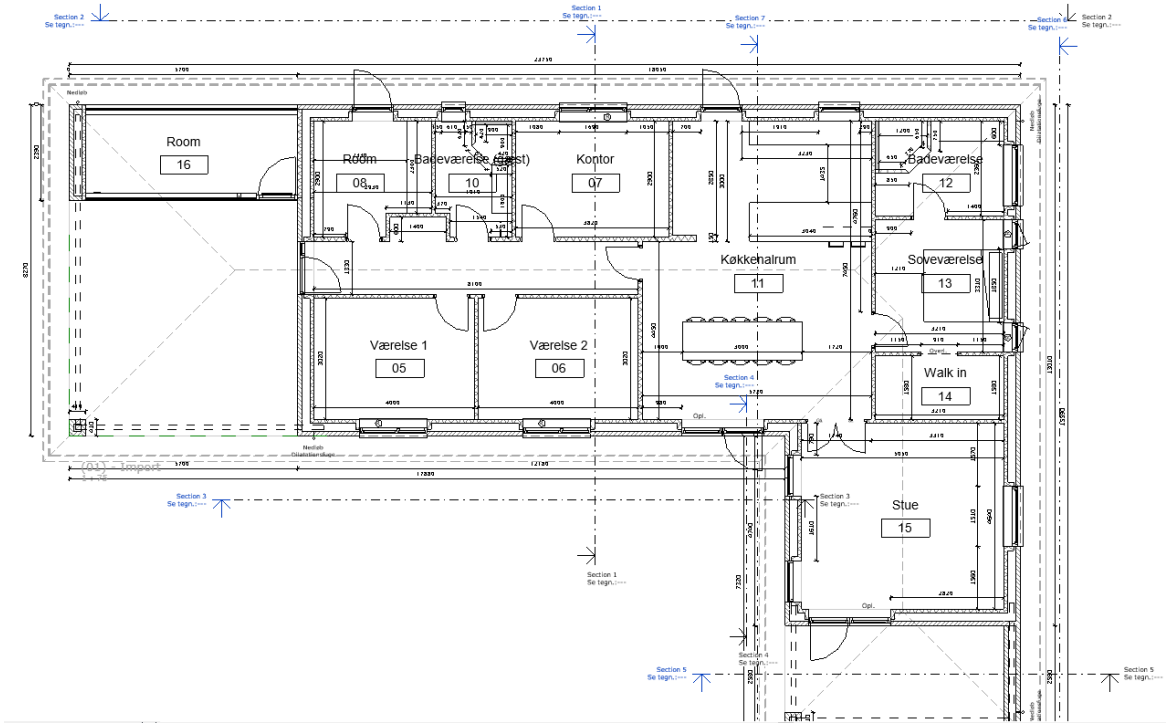

Figure 11-7 Ground floor plan

Figure 11-8 Marked spaces for space data export

2. Dynamo Space model

Topology model preview after ,, *Space data"* Dynamo script run in finished.

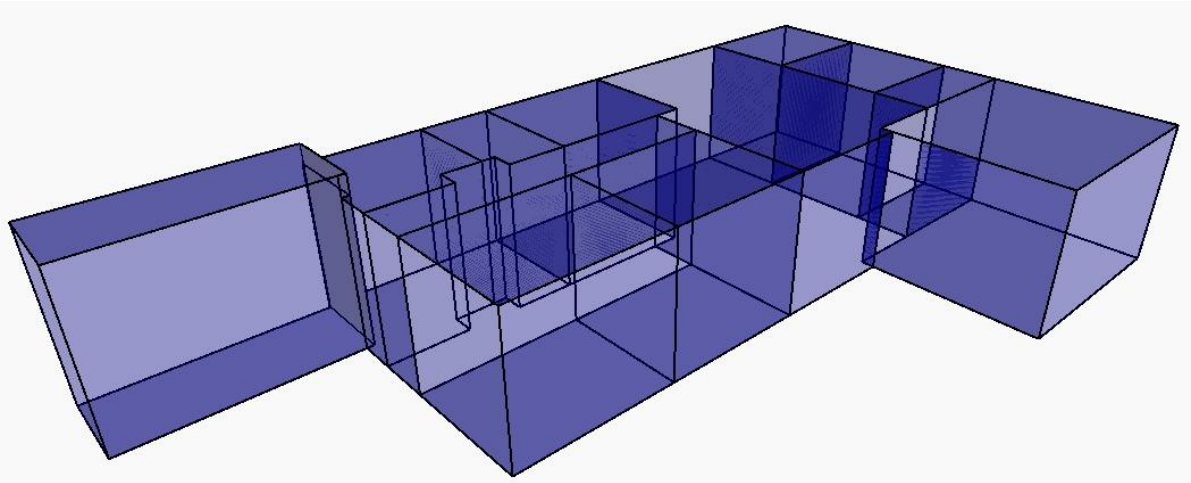

Figure 11-9 Parcelhus space model in Dynamo

3. Dynamo windoor data model

Windoor data preview after ,,Windoor *data* " Dynamo script run is finished. As it can be seen in illustration below this paragraph, the view angle is changed duo to window/door colour visibility.

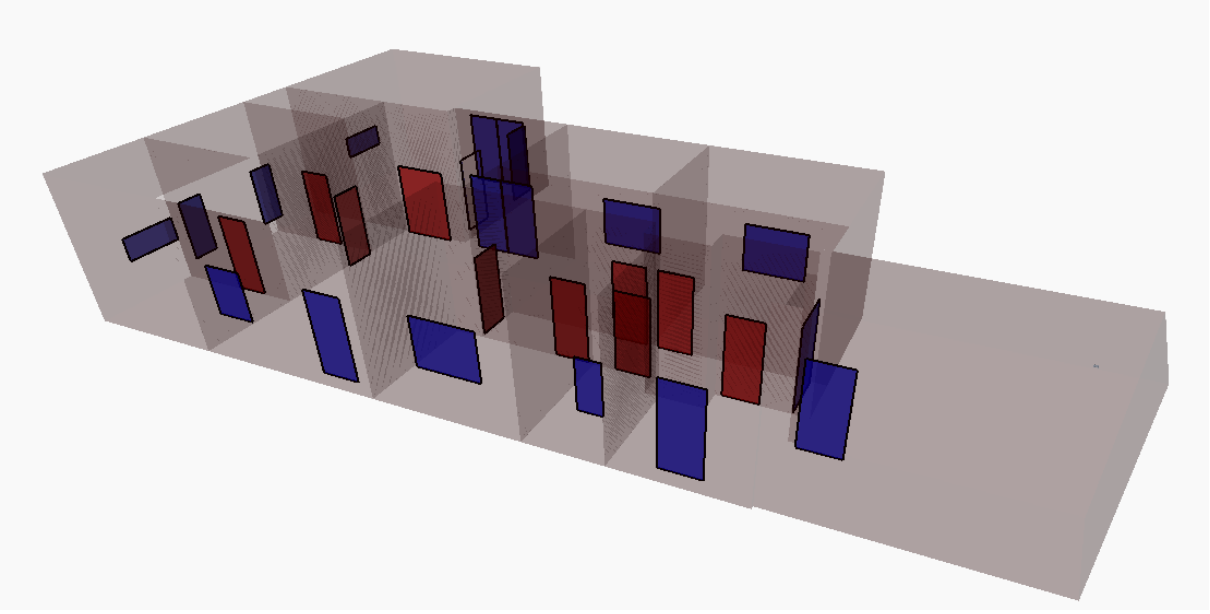

Figure 11-10 Parcelhus Windoor model in Dynamo

4. Built BSim model preview in BSim

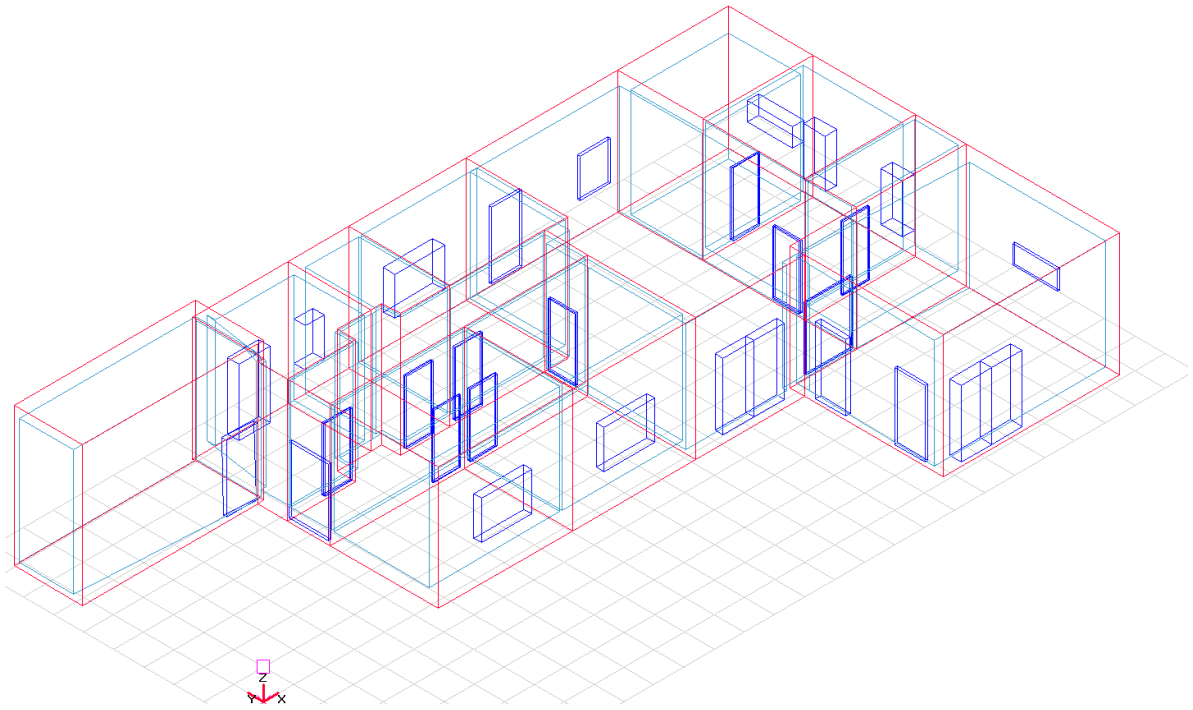

Figure 11-11 Parcelhus model built with BSim Data Modeler

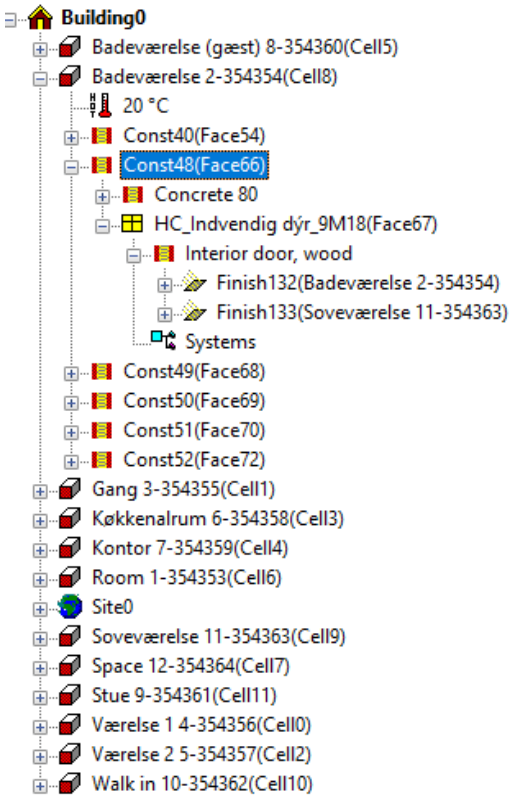

Figure 11-12 Parcelhus hierarchical tree in SimView

At the moment BSim Data Modeler from space and windoor data collections is capable to create BSim model with unlimited amount rooms and CELL relations. However, it does not have sophisticated methods to check if the room is a closed by bounded partitions. For this reason, the tool might develop some inaccuracies in the final model.

G. Opening analysis

LinkElementCollector.OfCategoriesIntersectingBoundingBox node block in Dynamo from MOE collects all windows and doors from Revit database. However, it is not a very robust solution, as it is very sensitive to tolerance adjustments and results of collecting duplicated window and door instances. In addition, the additional methodology is required to use this node.

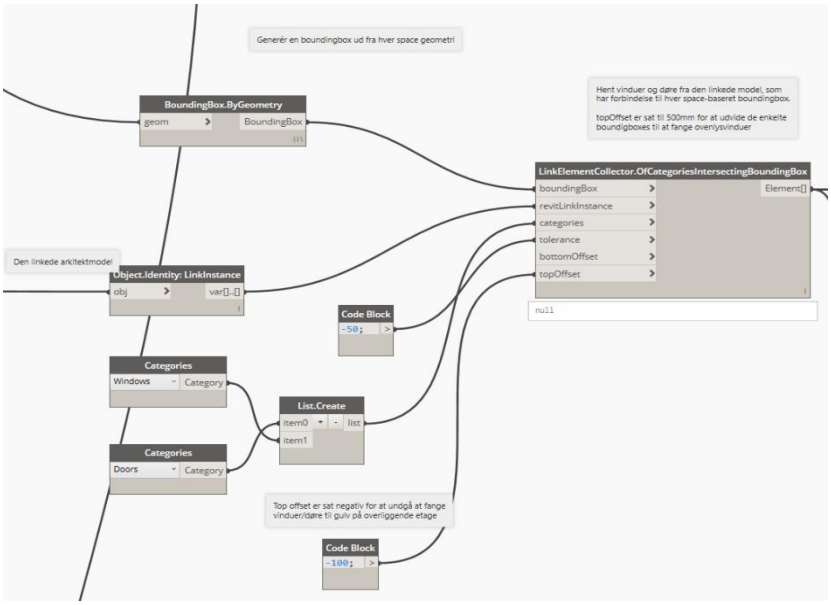

Figure 11-14 LinkElementCollector.OfCategoriesIntersectingBoundingBox node block in Dynamo

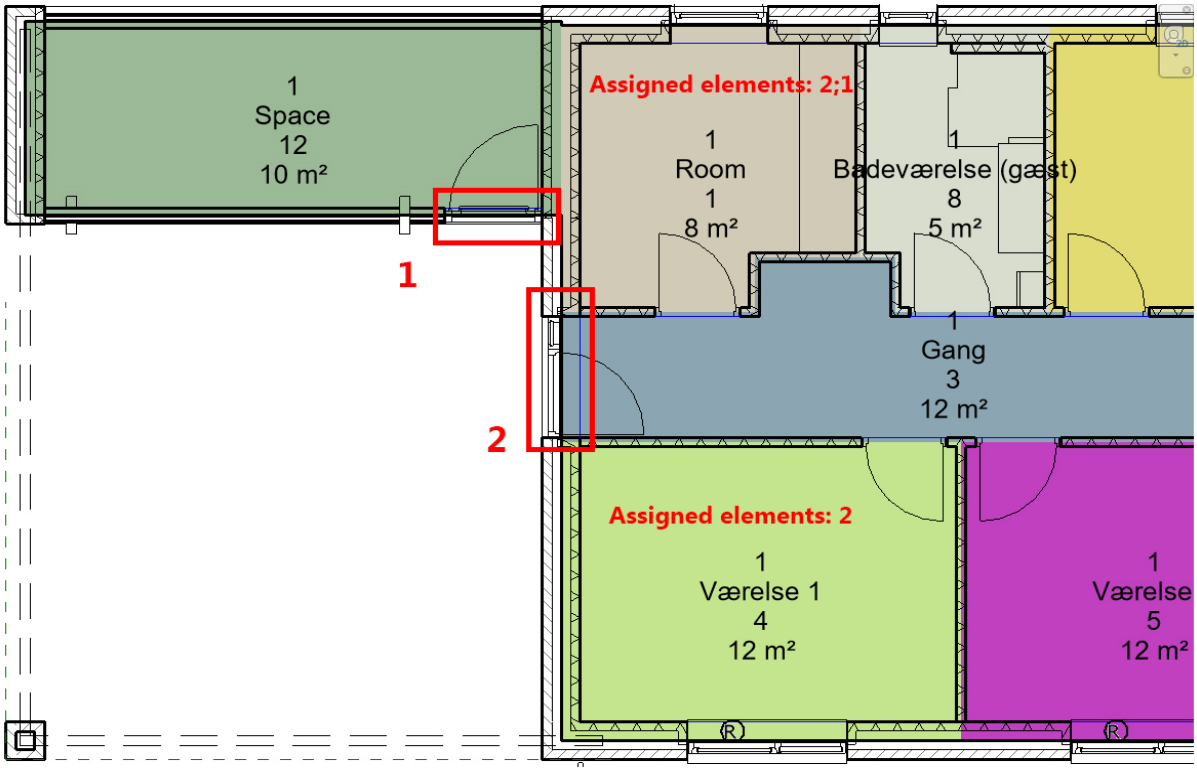

Figure 11-13 Incorrect opening assignment by element collector node

Window and door collector assign opening elements incorrectly. As illustrated in Figure 11-15 for the room marked with green colour this node acquires 3 openings.

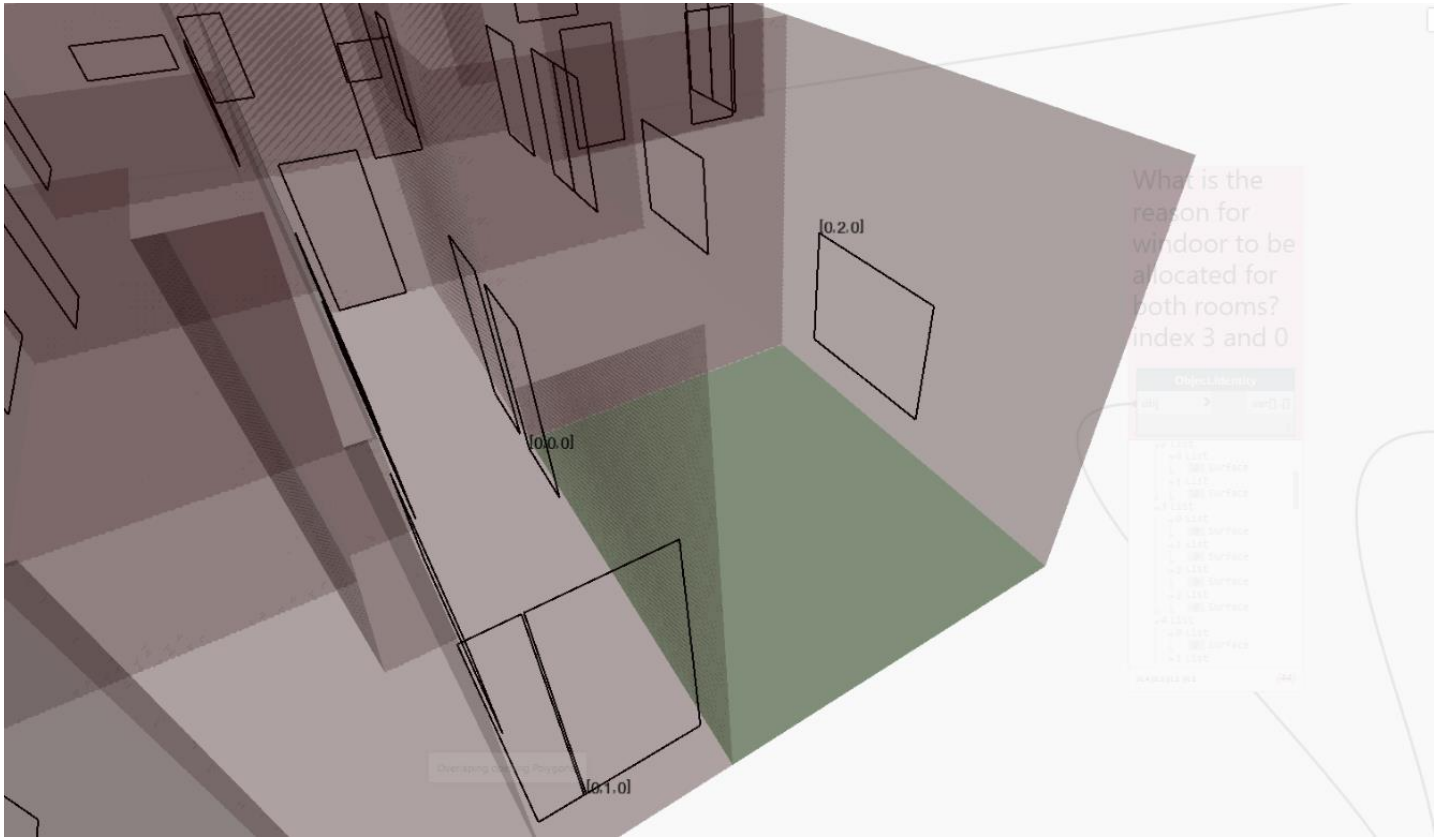

Figure 11-15 An example of incorrect Window/door element collection method

The openings that are situated next to each other cause polygon overlap which must be evaluated in order to be suitable for the SimView model.

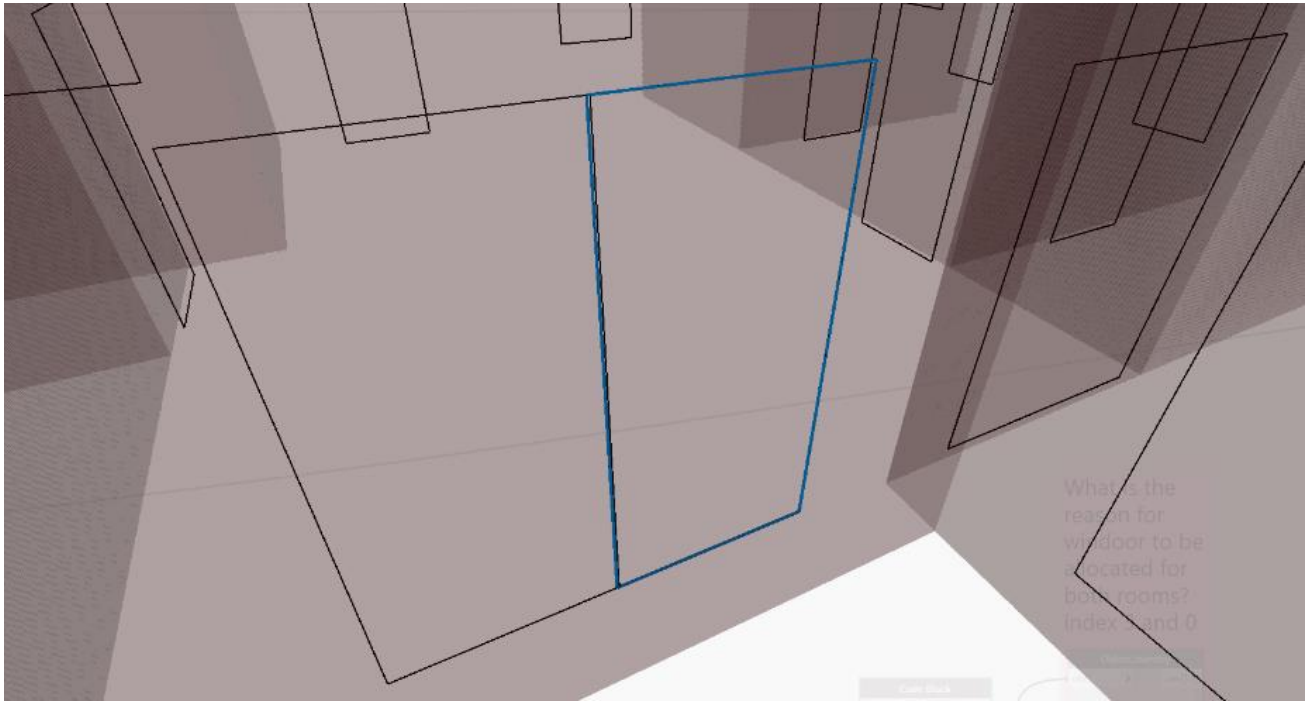

Figure 11-16 Overlapping external window polygons in Dynamo environment.

H. Defined data collections from Dynamo

StartPoint	EndPoint	Edge	Face	Space	SpaceID	Space-Range	Construction	BelongsToFace	F_Outdoor	F_Indoor	Space Name
10160 3480 0	5700 3480 0	1	1	1	354356	6	4893515	1	354355	354355	Værelse 1 4
5700 3480 0	5700 3480 3954	2	1	1	354357	12	4893582	1	-1	354356	Værelse 2 5
10160 3480 3954	5700 3480 3954	3	1	1	354359	19	4964720	1	354356	354356	Kontor 7
10160 3480 0	10160 3480 3954	4	1	1	354353	30	4893522	1	354357	354357	Room 1
10160 0 3954	10160 3480 3954	5	2	1	354355	43	4893503	1	-1	354356	Gang 3
10160 3480 3954	5700 3480 3954	6	2	1	354360	51	4893502	1	-1	354356	Badeværelse (gæst) 8
5700 3480 3954	5700 0 3954	7	2	1	354358	64	4893515	2	354355	354355	Køkkenalrum 6
5700 0 3954	10160 0 3954	8	2	1	354354	70	4893582	2	-1	354357	Badeværelse 2
10160 0 0	10160 3480 0	9	3	1	354363	76	4964720	2	354357	354357	Soveværelse 11
10160 3480 0	5700 3480 0	10	3	1	354362	82	4893516	2	354358	354358	Walk in 10
5700 3480 0	5700 0 0	11	3	1	354361	90	4893522	2	354356	354356	Stue 9
5700 0 0	10160 0 0	12	3	1	354364	99	4893502	2	-1	354357	Space 12
...	...	...	...	...	...	...	...	...	...	...	...

Table 11-3 Space data export from Dynamo

Start Point	End Point	Edge	Face	In/Out	Name	FaceID	SpaceID
10005.450000 3480.000000 2088.505000	9104.550000 3480.000000 2088.505000	1	1	TRUE	HC_Indvel	1	1
10005.450000 3480.000000 10.495000	10005.450000 3480.000000 2088.505000	2	1	TRUE	HC_Indvel	7	2
9104.550000 3480.000000 10.495000	10005.450000 3480.000000 10.495000	3	1	TRUE	HC_Indvel	14	3
9104.550000 3480.000000 2088.505000	9104.550000 3480.000000 10.495000	4	1	TRUE	HC_Indvel	24	4
11215.450000 3480.000000 2088.505000	10314.550000 3480.000000 2088.505000	5	2	TRUE	HC_Indvel	35	5
11215.450000 3480.000000 10.495000	11215.450000 3480.000000 2088.505000	6	2	TRUE	HC_Indvel	36	5
10314.550000 3480.000000 10.495000	11215.450000 3480.000000 10.495000	7	2	TRUE	HC_Indvel	37	5
10314.550000 3480.000000 2088.505000	10314.550000 3480.000000 10.495000	8	2	TRUE	HC_Indvel	38	5
12155.450000 4910.000000 10.495000	11254.550000 4910.000000 10.495000	9	3	TRUE	HC_Indvel	41	5
...	...	...	...	...	...	...	...

Table 11-4 Windoor data export from Dynamo

I. Data linking templates

In the following examples, the unique partitions and windoor elements are linked with unique SfB number from the construction and material databases. The SfB number is inserted in the columns marked with green colour. Finally, identified element information is exported together with all building topology model data.

Construction	Space Name	Space ID	Construction belongs to	Unique-Construction	Sfb	Finish_IN_sfb	Finish_Out_sfb
4893515	Værelse 1 4	354355	Gang 3	4893515	21.10.00	10	
4893582	Værelse 2 5	354356	Værelse 1 4	4893582	21.10.00	10	
4964720	Kontor 7	354356	Værelse 1 4	4964720	21.10.22	10	g2
4893522	Room 1	354357	Værelse 2 5	4893522	21.10.00	10	
4893503	Gang 3	354356	Værelse 1 4	4893503	21.10.00	10	
4893502	Badeværelse (gæst) 8	354356	Værelse 1 4	4893502	21.10.22	10	g2
4893515	Køkkenalrum 6	354355	Gang 3	4893516	21.10.00	10	
4893582	Badeværelse 2	354356	Værelse 1 4	4893514	21.10.00	10	
4964720	Soveværelse 11	354356	Værelse 1 4	4893534	21.10.00	10	
4893516	Walk in 10	354358	Køkkenalrum 6	4893591	21.10.00	10	
...	...	...	...	...	...	...	...

1.Update Construction

2.Export Space data

Table 11-5 Construction linking template

Space ID	Windoor	Windoor Host	Unique windoor	sfb	Finnish_In	Finnish_out
1	HC_Indvendig dør_9M	1	HC_Indvendig dør_9M	32.01.10	i1	i1
2	HC_Indvendig dør_9M	7	HC_Indvendig dør_15M	32.01.10	i1	i1
3	HC_Indvendig dør_9M	14	920 x 2135 mm door	32.01.10	i01	i01
4	HC_Indvendig dør_9M	24	Elem.247 - 1668 x 1258 m. bred opluk	31.01.14	a15	a15
5	HC_Indvendig dør_9M	35	Elem.247 - 1008 x 2188	31.01.10	a15	a15
5	HC_Indvendig dør_9M	36	Elem.247 - 1308 x 2188	31.01.10	a15	a15
5	HC_Indvendig dør_9M	37	Elem.247 - 588 x 1258	31.01.14	a15	a15
5	HC_Indvendig dør_9M	38	Elem.247 - 1008 x 1258	31.01.14	a15	a15
5	HC_Indvendig dør_9M	41	1488 x 588	31.01.14	a15	a15
5	HC_Indvendig dør_9M	42	Elem.247 - 708 x 1588 - Soveværelse	31.01.14	a15	a15
...	...	...	...	...	...	...

1.Update Windoor list

2.Export windoor data

Table 11-6 Windoor construction linking template

1. Extended data collections

Extended space data and windoor data collections from excel is exported to txt format file. Furthermore, this specified data is supplied to BSim Data Modeler.

Space Data

StartPoint	EndPoint	Edge	Face	Space	SpaceID	Space-Range	Construction	BelongsToFace	F_Outdoor	F_Indoor	Space Name	Con_Sfb	F_in	F_out
10160 3480 0	5700 3480 0	1	1	1	354356	6	4893515	1	354355	354355	Værelse 1 4	21.10.00	10	
5700 3480 0	5700 3480 3954	2	1	1	354357	12	4893582	1	-1	354356	Værelse 2 5	21.10.00	10	
10160 3480 3954	5700 3480 3954	3	1	1	354359	19	4964720	1	354356	354356	Kontor 7	21.10.22	10	g2
10160 3480 0	10160 3480 3954	4	1	1	354353	30	4893522	1	354357	354357	Room 1	21.10.00	10	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Table 11-7 Space data collections with specified construction types

Windoor Data

Start Point	End Point	Edge	Face	In/Out	Name	FaceID	SpaceID	Win_sfb	F_In	F_out
10005.450000 3480.000000 2088.505000	9104.550000 3480.000000 2088.505000	1	1	TRUE	HC_Indvendig dør_9M	1	1	32.01.10	i1	i1
10005.450000 3480.000000 10.495000	10005.450000 3480.000000 2088.505000	2	1	TRUE	HC_Indvendig dør_9M	7	2	32.01.10	i1	i1
9104.550000 3480.000000 10.495000	10005.450000 3480.000000 10.495000	3	1	TRUE	HC_Indvendig dør_9M	14	3	32.01.10	i1	i1
9104.550000 3480.000000 2088.505000	9104.550000 3480.000000 10.495000	4	1	TRUE	HC_Indvendig dør_9M	24	4	32.01.10	i1	i1
...	...	...	...	...	...	...	...	...	...	...

Table 11-8 Windoor data collections with specified construction types

J. Space solid creation walkthrough

This particular appendix visually illustrates the methodology used to create space solid. The representation in a form of Dynamo blocks can be found in APPENDIX M and APPENDIX N.

1. Step 1 – Obtain room boundary curves and apply an appropriate offset

In this step, inner leaf space boundary curves are collected and joined into polygons. Furthermore, each boundary curve segment is evaluated with regards to the OUT/IN orientation. Finally, polygons are offset towards the outdoor direction.

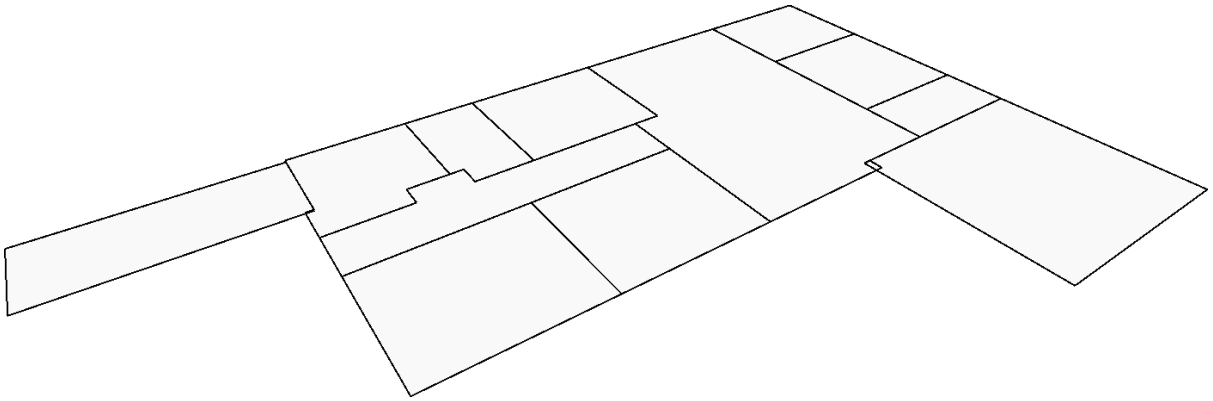

Figure 11-17 Parcelhus reference level boundary curves

2. Step 2 – Define boundary curve intersections.

Due to extrude methods and different model joints after conversion, space boundary polygons overlap on each other. Therefore, appropriate Semi-automatic evaluation should be made.

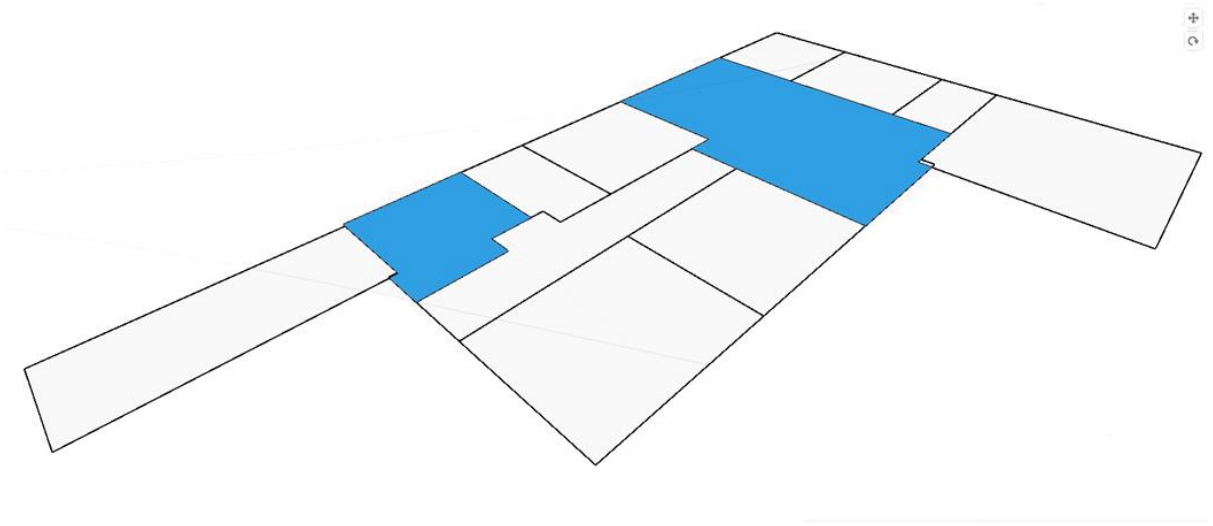

Figure 11-18 Cropped boundary surfaces

3. Step 3 – Evaluate room boundary curve points

New collections are assigned as previous data collections after boundary cropping are no longer valid.

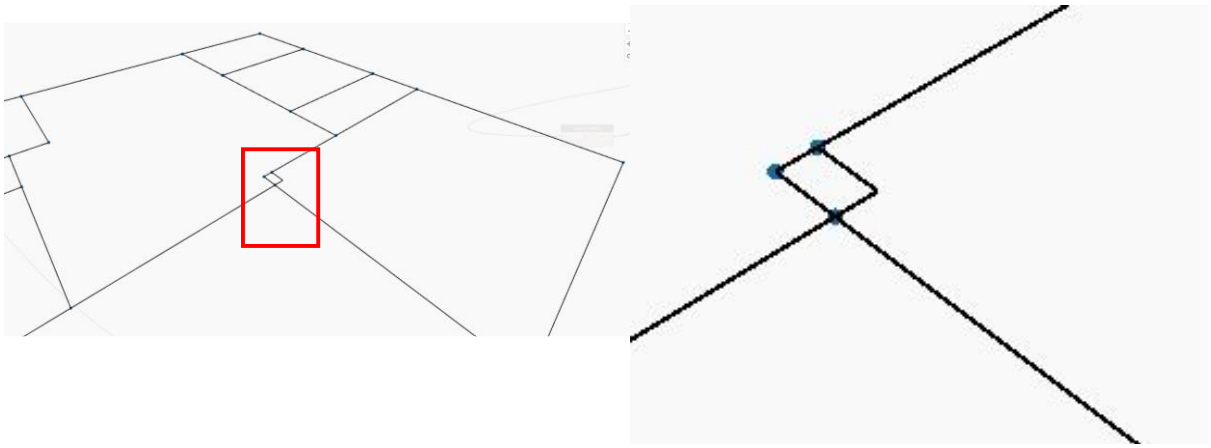

Figure 11-19 Overlapping boundary curves at intersection corner

4. Step 4 – Eliminate unwanted data points and remove unwanted data points

On this step, curve data points are evaluated to eliminate all points that are on the straight line, as it compromises individual surface creation. For example, in the illustration below extra data point would produce additional surface segment.

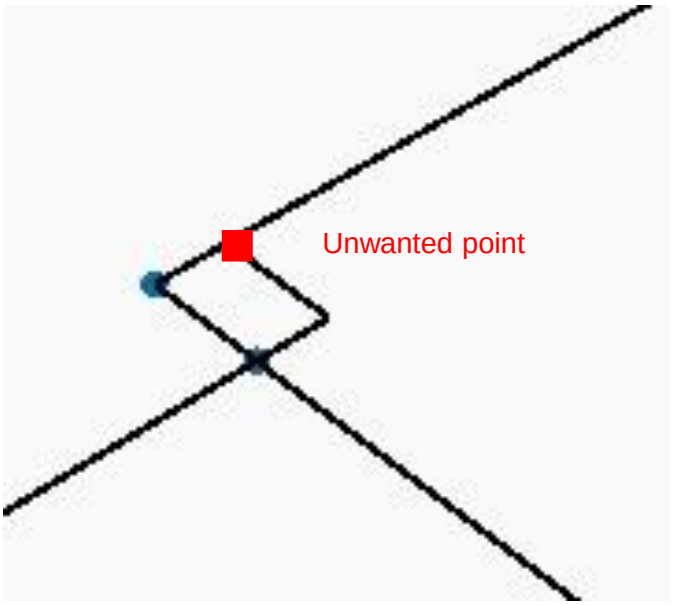

Figure 11-20 The Unwanted point on the straight line

5. Step 5 – Create new surfaces by corrected boundary segments

On this step, a new boundary segment collection is used to create an individual surface for each room.

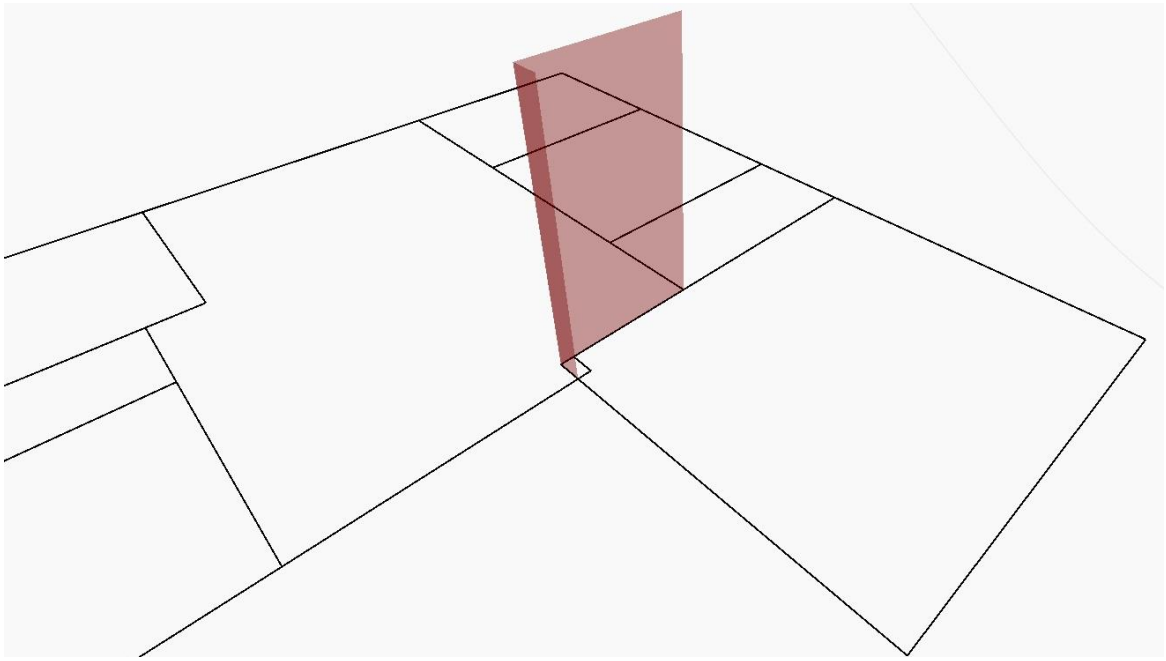

Figure 11-21 Surface creation example

6. Step 6 – Evaluate surface difference on opposite side and sub-divide surface into smaller segments.

As surfaces are created individually, additional evaluation is necessary to check whether the surface on another side is the same. In the cases where the surface is different in size, such surface is cropped and send back for evaluation and proper space assignment.

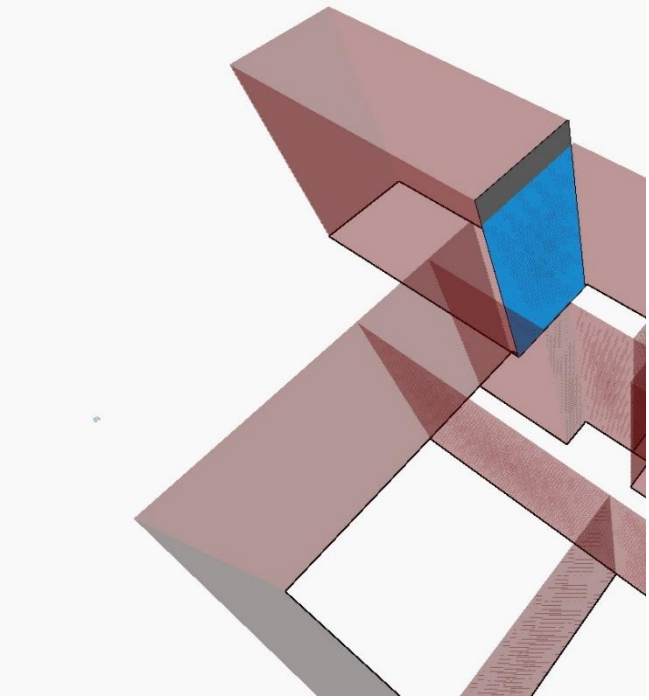

Figure 11-22 Surface sub-division example

7. Step 7 – Join all surfaces into appraise space solid

On this step, all individually created and evaluated surfaces are joined back into complete space solids. Such solids are a close representation of the SimView topology model and ready for export.

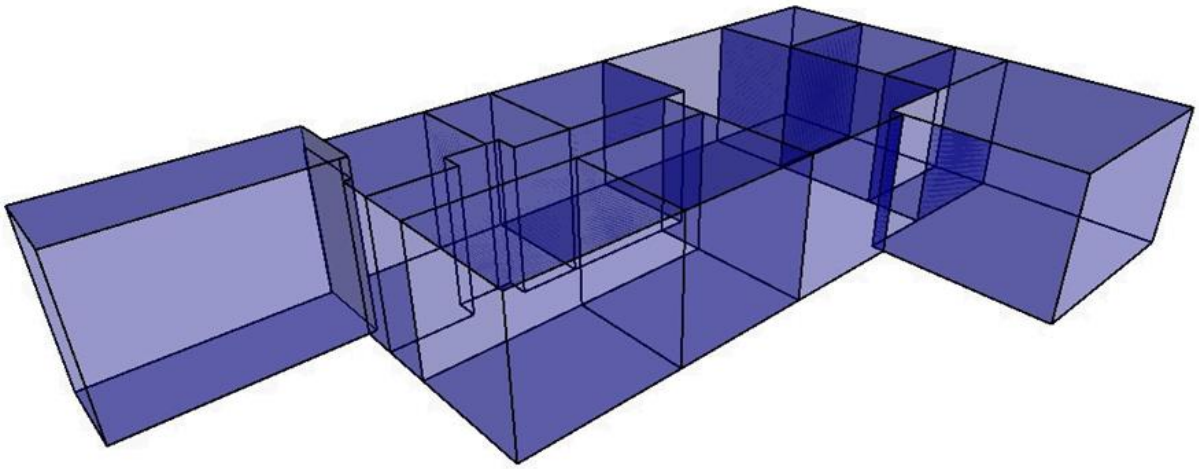

Figure 11-23 Joined surfaces into individual solids

From this stage, further solid data treatment is performed outside the Dynamo environment. For this particular data treatment and topology model modelling, the external stand-alone C# application – BSim Data Modeler is used.

K. BSim Data Modeler preview

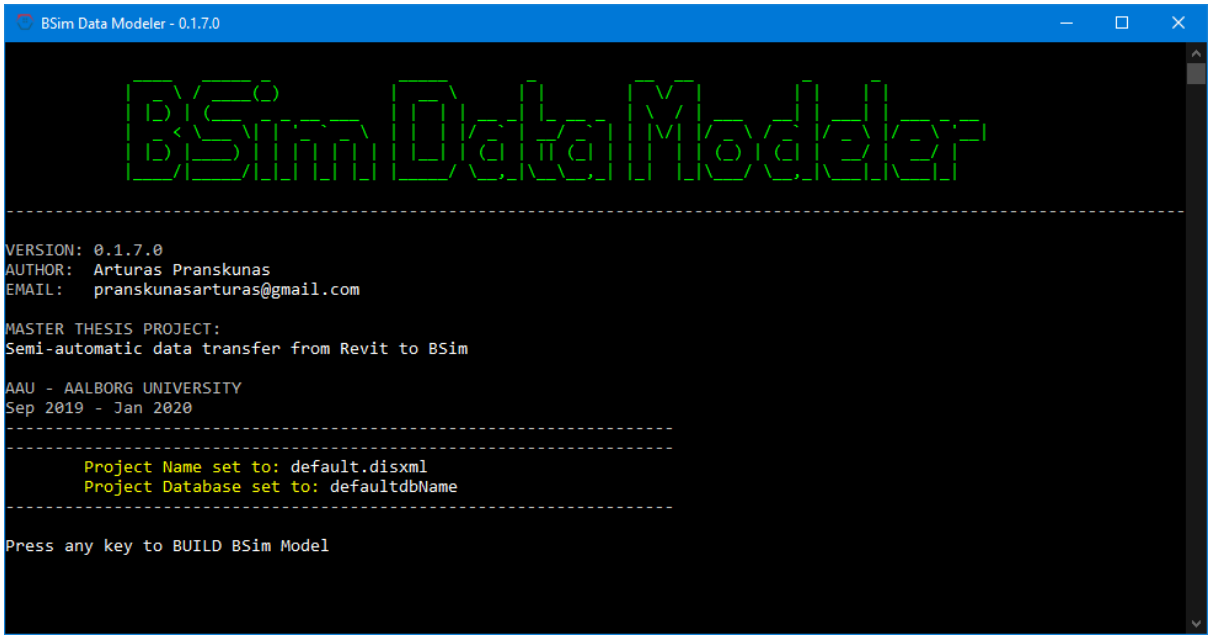

Figure 11-24 BSim Data Modeler interface

System requirements

- **Operating system:** Windows platforms with the support of .NET Framework 4.7.2
- **Memory:** 13-40 MB
- **Hard disk space:** 20 MB
- **Additional software:** BSim, Version 7.16.8.11 or disable BSim instance launch in the config file.

1. Workflow

1. Open Revit template: “BIM-Revit2BSim”
2. Tag building spaces with space tags.
3. Adjust space height.
4. Set output directory.
5. Run: “01_BSim-Spaces.dyn” - Dynamo script.
6. Run: “02_BSimWindoor” - Dynamo script.
7. Open Excel document: "InputData.xlsm".
8. Specify your Output location in "Settings"- sheet.
9. Refresh new data from Dynamo.
10. Go to sheets "Construction_Defintion and Windoor-parameters " to define optional parameters and click export.
11. For the first run time on a new computer "BsimDataModeler.exe" - will generate ,,BsimDataModeler.exe.config" where additional functionalities could be adjusted
12. Run "BsimDataModeler.exe" and follow instructions on the screen.
    1. Type project name

2. Type database name
13. Wait until BSim opens.
14. Press save model when closing BSim.
15. Close "BsimDataModeler.exe - your model is complete.

Please note that exported txt files from Excel should not be edited with a text editor, as it will compromise file structure and BSim Data Modeler will not be able to read output files.

2. Config file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="Debugger" value="true" />
    <add key="Modeler_log" value="true" />
    <add key="LogFiles" value="true" />
    <add key="BSim_startup" value="true" />
    <add key="DataBaseName" value="defaultdbName" />
    <add key="InternalDoors" value="true" />
    <add key="Windows" value="true" />
    <add key="ThermalZones" value="OFF" />
    <add key="Systems" value="OFF" />
    <add key="Schedules" value="OFF" />
    <add key="ExternalBuildings" value="OFF" />
  </appSettings>
</configuration>

// true – enabled
// false – disabled
// OFF – not fully implemented should be kept OFF
// InternalDoors – insert internal openings in the model
// Windows – insert external openings in the model
```

Listing 11-8 BSim Data Modeler configuration XML file

3. Changelog

RowNum	ID	Author	Date	Message
0	f791089e612214a4f9c0d55a94ff78f3e39a2ebf	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2020-01-08 22:43	Updated BSim start-up method. The previous method was buggy as it was starting multiple BSim instances. Updated to: 0.1.7.3 Final model before Project submission.
1	f84086c2acfb4bc21ab70a88a81c8b9c19eb241a	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2020-01-08 22:39	Fixed file name bug. The file name default variable was inherited between methods multiple times. Updated to: 0.1.7.2
2	dcde31b391fde81ba80e72c01db182b78efd2d7c	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2020-01-08 22:19	Added options to excluded internal or external openings from the model. Updated to: 0.1.7.1
3	b0b1beca2173af01dd8b97e77115e7cad8df25aa	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2020-01-08 22:14	Copyright Message update and setting option extension.
4	8a127a6649087a7cddaacf989c2452214d7a34884	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-09 14:17	Common debugger update - Bsim start-up hotfix. Added logic for missing Data log folder
5	f83d063feee850745910285f5b1bd3c20940db8c	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-09 14:13	Added windoor construction logic. Started to make methods for Surface- FINISH layers. At the moment it is disabled by default , as it missing logic for identifying facing sides.
6	3a82b107099eb7c97685a536d27a7076520282dd	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-07 21:51	Added CONSTRUCTION and CONSTRUCTION_ELEMENT logic. Sfb number can be defined in txt. Updated to 0.1.5.0
7	6d3b8fe3a3a3215616231bf9e4252ea36687c6c7	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-03 22:19	Extended parsing methods, added app configuration file. At the moment: debugger logs and data logs can be disabled separately. Application version updated to 0.1.4.0
8	477aefa56cf92df5a39d31b4e4c54fd34749f349	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-03 22:14	Extended current code to write finish elements for construction types and windoor elements.
9	6e4e02d5fb1dcd126c2b9baeb35519ddeecb6402	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-01 17:42	Updated to 0.1.2.0
10	2b278ec1bffef8af8caadf7929d015be9fbbe518	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-12-01 17:41	Added complete window geometry logic. Updated methods for more than one windoor per FACE. Fixed windoor method that was called multiple times due to this doors and windows were remapped in the global list more than 5 times...
11	b2bfcd6211fa5641b992271ba798bb745d9dcda0	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-28 21:49	Small bug fix. Updated ver. 0.1.1.0 -Stable
12	37b7f59ffbe11ed1ab30eb8184c9690c11d7d598	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-28 21:47	Completed windoor logic. At the moment windows and doors are fully functional in the model. However, original id of the windoor is remapped by the filter due to windoor overload. In principle windoor components could be filtered out in Dynamo as well then filter will keep original ID of the doors/windows.
13	4fef20b35880d5e1f64cbb925434626c45fd4972	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-28 03:34	Added window and door modelling logic.
14	b2c4b1ea2632f7bac8feecb24f31053d02f27b12	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-27 21:43	Room + space naming in SimView
15	a13fe853333e6024e830c539c27d58a5d0f3bc6b	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-27 21:42	Added site creation and room naming
16	6f575fea362241edc51fa56e865b9fa2eb1797ac	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:15	BSimDataModeler updated to ver 0.0.9
17	5403bc21c70eef51d083bc87975dfbe0a9726f17	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:13	Added new - rewritten class for changed data structure that comes from dynamo. Current limitations: modeler can only handle one building and it does not support windows.
18	04733f9dd5bfe7b6fa3557fe0d2e9fda7ee9fc29	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:07	Performance fix.
19	97d93ba1a232e15d3a3c99f3a5f523b49ef7c502	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:06	Extended debugger print methods. Visual fix;
20	488b9bbda31f96f8fea0e37c122a0a9b5dbbb7c2	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:04	Added external data enums
21	dda41b0f85aec4a0a24c48e2bdcfa689f52411ef	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-23 23:03	Fixed infinite loop when data points are retrieved from filter.
22	ced0cdd9909d41c0c6674796cee65d9f32aa73c8	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-21 19:06	Changed Face output method to new data structure that it is required by new data structure. Added: simple outdoor/ indoor evaluation and space data out for Modeler.
23	a755c038ae097281685f2211cc4c5f516bab424c	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-11 12:01	Added correct edge forming methods
24	7647c559a5ac89f4b0f37f3788c4f27cebf87b7c	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-11 11:54	Extended RID stacking procedure.
25	a5f3b3584ee463e50369093124b8fc162b3bd654	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-07 13:05	Backup
26	0b199cd06f26ecee84e4a4400bf73a13370ee493	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-06 15:52	Rewritten logic for data sorting. Data sorting begins from multiple edge check, continues to face and finishes at multiple vector check. Backup before class cleanup...
27	5faf2f9ac4f9bfc0d0b1793c8e71e7224bfece68	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-11-02 20:19	Duplicated Faces and Edges are completely removed. Methods for multiple data points are not correct it still produces repeated data points as Revit writes data in random order... Veretex code is updated to reference unique data points. Concat method for faces is not complete.
28	f22735473e455e7db929f1c2b115e2892e6e16a6	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-31 21:32	Updated Revit Data sorting - multiple data points and edges are removed. Face - implementation is still remaining.
29	2a44c236d19dde27eb78d4fb7a0607a3dd140d6d	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-30 18:38	Added Revit data purge classes - it removes duplicated data and makes new references for edges and faces...
30	7a5d7f4bbd309111ce2d47fa2c854b8faa35017d	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-28 15:19	Small fix: Removed Normal creation... Updated to ver. 0.008
31	b9e7fb6440c960be68ebe0e73985ee741870eb08	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-28 15:15	Attached SimView.exe instance - in order to finalize model geometry. Added Schema parser for XML validation. In addition to schema parser a simple node element validator for VERTEX, EDGE and SPACE was added.
32	560eb0c5293f3ccffcdbcf71752c74257c09b046	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-28 14:26	Code clean-up. Copyright Message moved to method.
33	f515623ad08f1522ed2bd950e3652b4fe1d575e6	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-28 11:30	Thermal zone fix. XML file validated with Schema that was generated from example files in BSim install directory.
34	dc8ba75b29d7cdaf31d566440b327ca0eadbe369	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-26 18:28	Normal and finish nodes are discarded as it is optional for model generation.
35	1f3ce7e6bc48c957d858e9f4eb470d2226062872	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-26 18:16	Updated to ver. 0.007
36	2953ae1172f8097d6c8c8a07944ea2f82eb52293	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-26 18:14	Fixed thermalZone bug, added BSim Launch function to test SimView crashed due to file consistency -error code is displayed in console. Fixed dot/ comma issue in output file - Current thread is set to CultureInvariant. Added dis(XML) file location logic.
37	45b046cc1cb16e9f17b451e867917df23df78670	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-25 07:56	Added SITE logic, wall construction, window construction. Fixed EDGE <edge_length> child node and crucial vertex referencing bug in this node that was crashing BSim model.
38	e5d56976472602b03fe1dd86890ed35958024913	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-14 18:34	Added SITE, LOCATION, extended debugger functions, and window completion element assignment.
39	09a8b2c6cdca8fb024780b88f7986a0a9eb46113	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-12 20:13	Extended CELL functionality up to 2 CELLS. There is crucial bug in cell FACE_SIDE referencing(referencing in a row is allowed! Otherwise, referencing is lost in the model). A proper method should be written for it.
40	01014598045c32c31e8e0f1dd17487f1893c91e4	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-12 14:59	Added Debugger, ver changed to 0.005
41	ed837ebfa43323fcbb8c4351d1ba90727944543a	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-11 19:19	Edge referencing model updated. Fixed bug with wrong edge IDs, right now is referenced correctly. Dummy cell code is written both rooms are place in same cell.
42	435bb4f2506f4afe57f33e2facf5b1d31f4671d3	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-10 22:57	Fixed multiple window data points.
43	06b2edcb65d00b0ebcb27eff766212ad386b3b34	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-10 13:32	Deleted unused code. Program updated to ver 0.0.3
44	249287275e03901b48ba3ba65533a3a28f42fad4	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-09 20:52	Functions for cell geometry and window have been rewritten. Furthermore one window per surface is fully functional, UniqueRid functions added, base for multiple rooms were set.
45	cddf6ff5a91de5fe3fe93f100f9dc4c11959aa40	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-09 00:31	Extended #RID functions
46	8f5e99267278d99380e79891cf93e3eb60fb5b11	Artūras Pranskūnas <pranskunasarturas@gmail.com>	2019-10-08 23:23	Initial ver.

L. Kindergarten test case

For testing purposes, a bigger model was chosen to test developed methodology robustness and identify required improvements to the current data export methods.

1. Kindergarten model overview

This around 1000 m2, the Kindergarten model consists of 37 rooms. For testing purposes, all building spaces were tagged for space and the windoor data export. As expected, this building export from Dynamo took way longer but did not throw any errors.

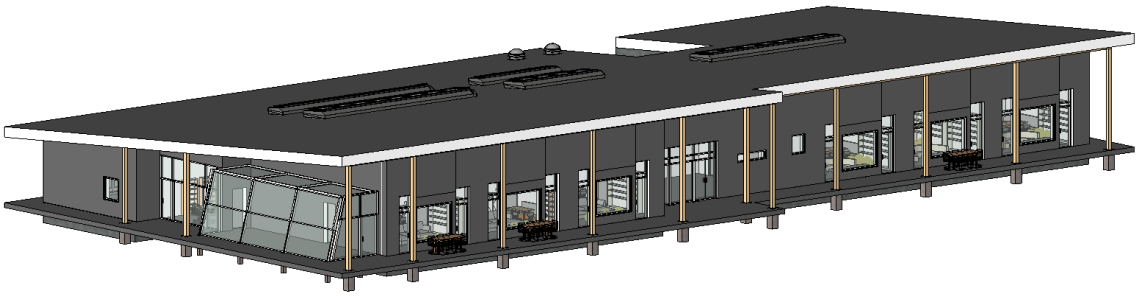

Figure 11-25 Kindergarten model 3D view in Revit

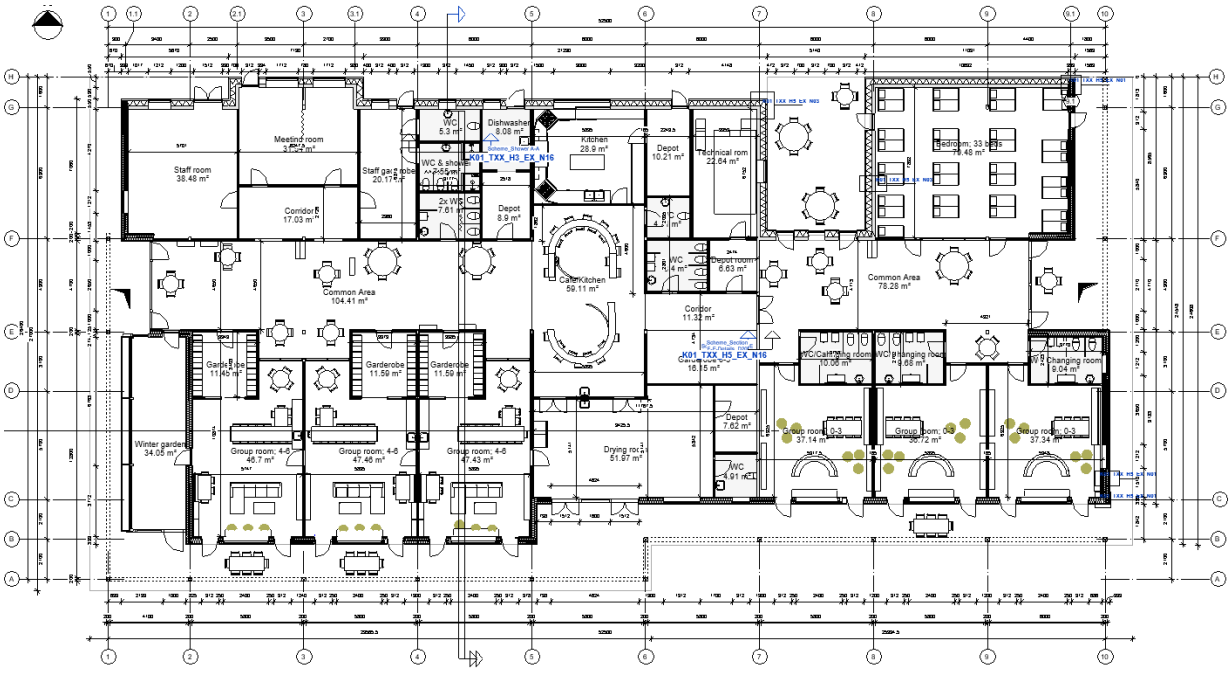

Figure 11-26 Kindergarten model ground floor plan

2. Result summary

The Kindergarten model is far bigger and more complex than the Parcelhus presented in APPENDIX F. For this reason, the space data Dynamo script took way longer to validate the entire building. There were not detected any crucial issues except that some rooms had overlapping geometries. The overlapping geometries in the model were produced due to incomplete Phyton script. This script is responsible for room boundary intersection evaluation. At the moment, the script can only handle a few intersection scenarios. To prevent such a scenario, the method requires an extension.

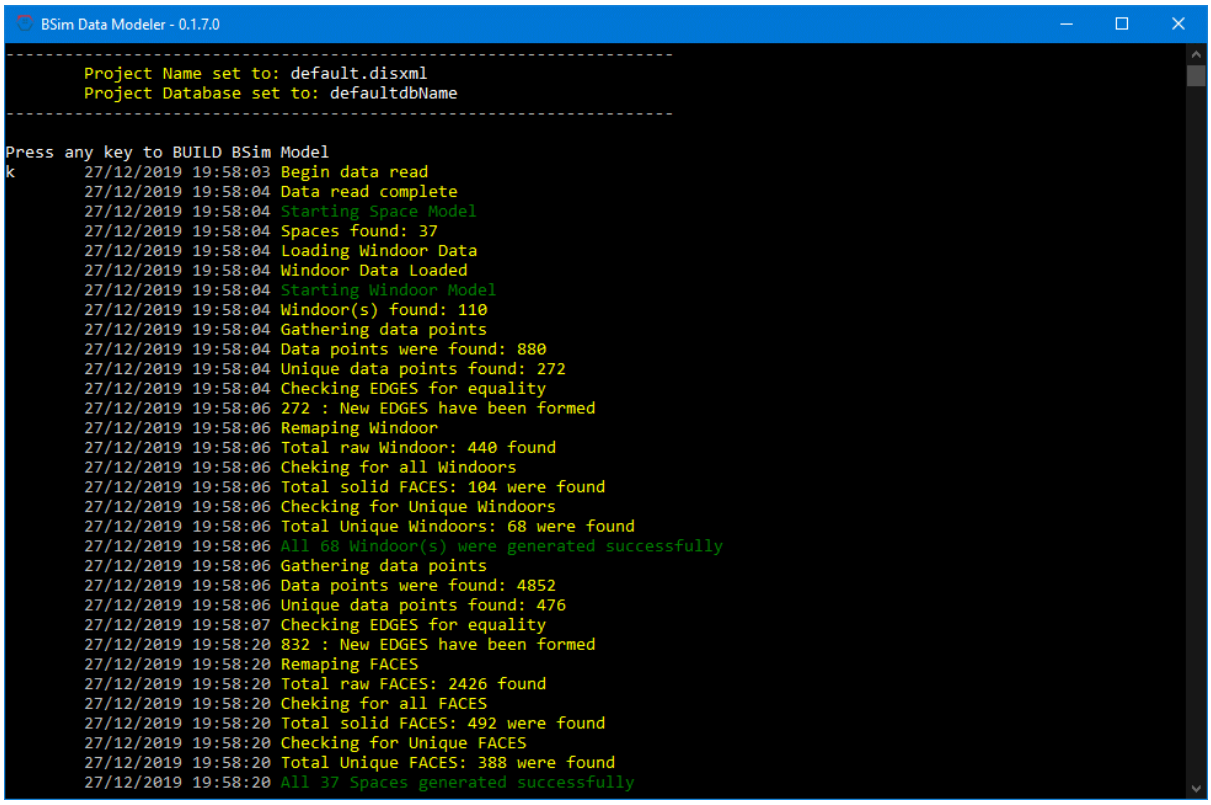

Figure 11-27 Kindergarten wire-frame model data statistics

Moreover, the BSim Modeler Data Modeler took much longer, about 17 seconds to produce a complete BSim model. As can be seen on the screenshot from BSim Data Modeler raw data has 4852 data points with means it has 2426 edges. Furthermore, all edges had to be evaluated on each shared face with took 13 seconds.

All in all, the Dynamo scripts and BSim Data Modeler produced a working BSim model with some overlapping geometries as expected. However, some windows got lost through export flow, as it got wrong references on Dynamo and BSim Data Modeler side.

The Final Kindergarten model after the export process can be observed below this paragraph.

3. Model in Dynamo environment

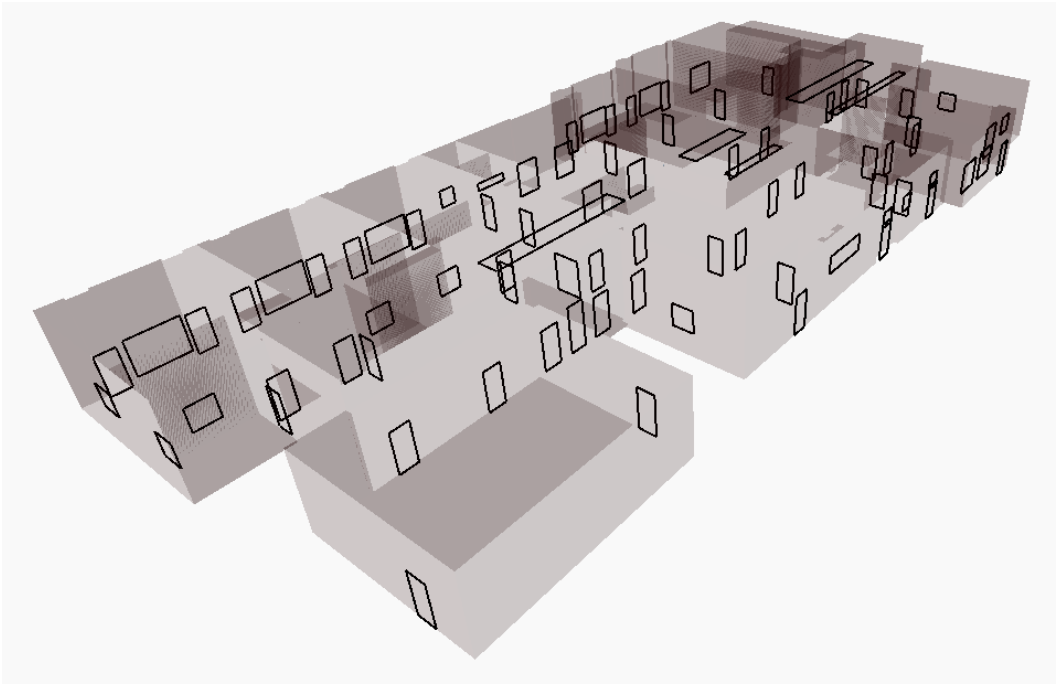

Figure 11-28 Kindergarten model 3D view in Dynamo

4. Model in SimView environment

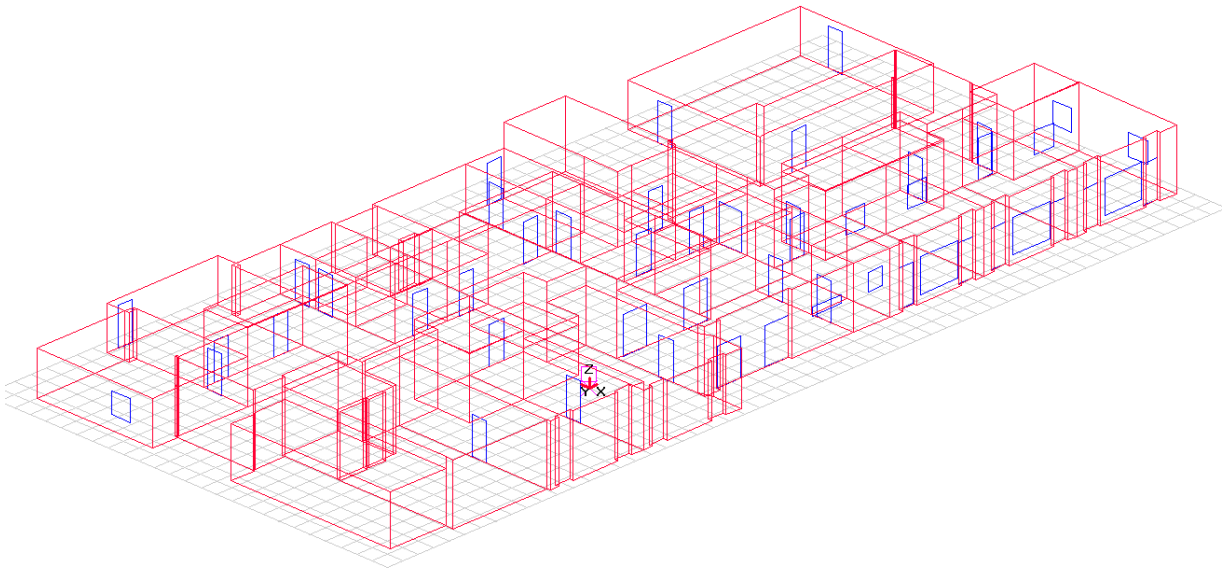

Figure 11-29 Kindergarten model 3D view in SimView

M. Space data Dynamo script walkthrough

1. Space data script overview

Please note that script screenshot captures presented in this APPENDIX is not cleaned. Therefore, it includes all nodes for testing cases that were presented in the report.

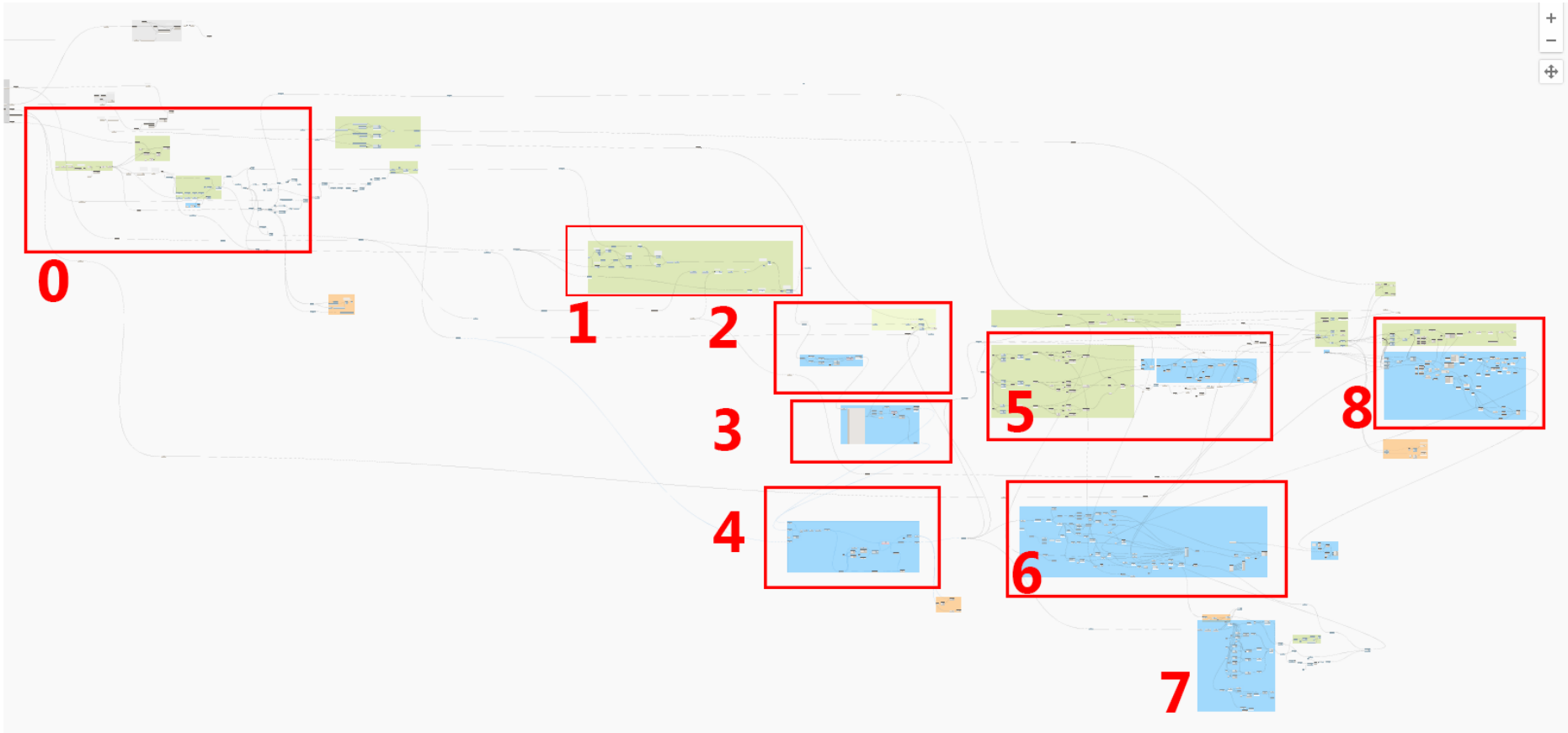

Figure 11-30 Space data Dynamo script overview

Explanation:

0. Script for obtaining space boundary segments and joining them into polygons.
1. Polygon offset rules.
2. Creates new boundary surface geometry, generated surfaces evaluated with regards to its intersections.
3. Evaluation of new room boundary segments, removal of unwanted points.
4. Vertical surface evaluation
5. Indoor/ outdoor surface evaluation
6. Space data collection sorting methods for export to Microsoft Excel.
7. Split surface evaluations
8. Construction family assignment to appropriate surface

2. Base surface evaluation num. – 2

If boundary surfaces intersecting between each other, the overlapping surface is discarded. At the moment, it only works for a few overlapping cases that were presented in the report.

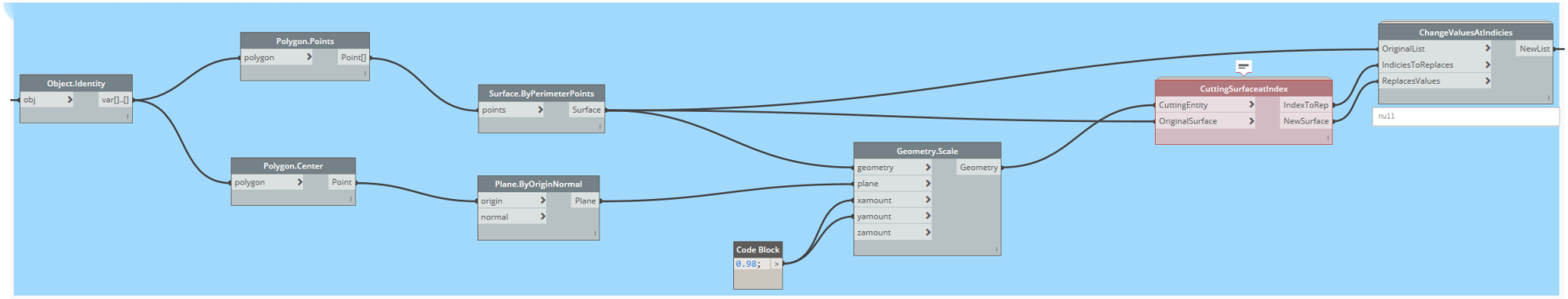

Figure 11-31 Base surface evaluation

3. Evaluate boundary curves and eliminate unwanted points num. -3

This script performs two evaluations. First, it removes any unwanted points from boundary curve collections. Secondly, it intersects newly created curves against each other.

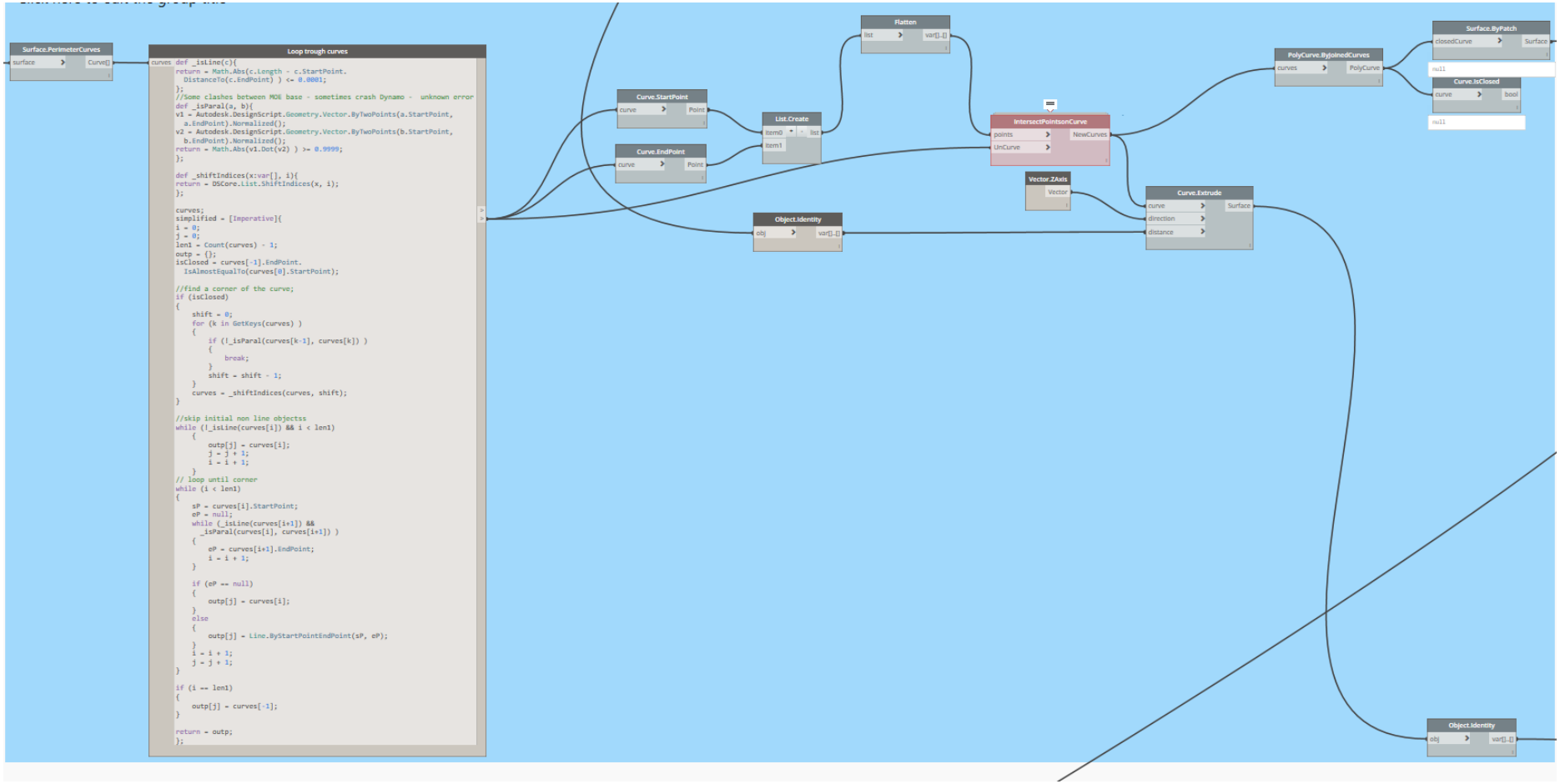

Figure 11-32 Boundary curve simplification

4. Individual surface evaluation script num. -4

This script consists of two parts, the first part offset existing base boundary by determined space hight, the second part of the script performs individual vertical surface evaluation and assigns it to appropriate spaces.

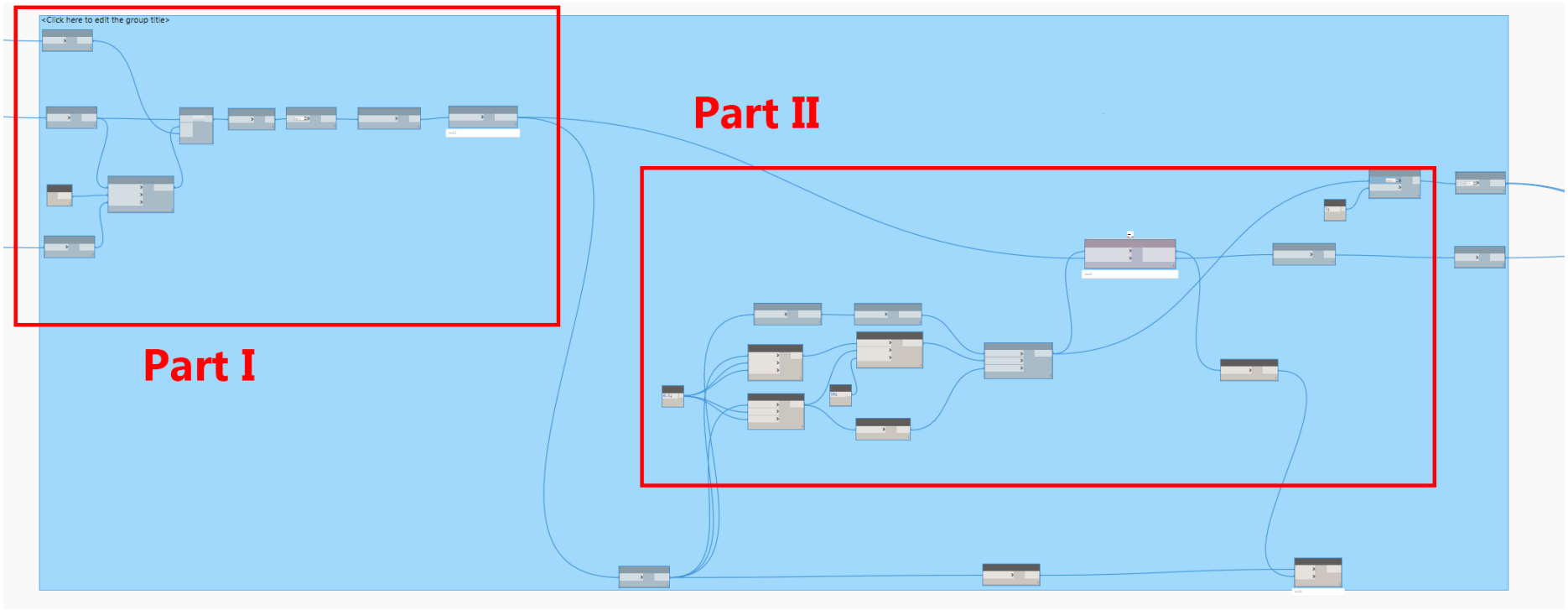

Figure 11-33 Individual surface evaluation scripts

Part I

Bottom surface remap method.

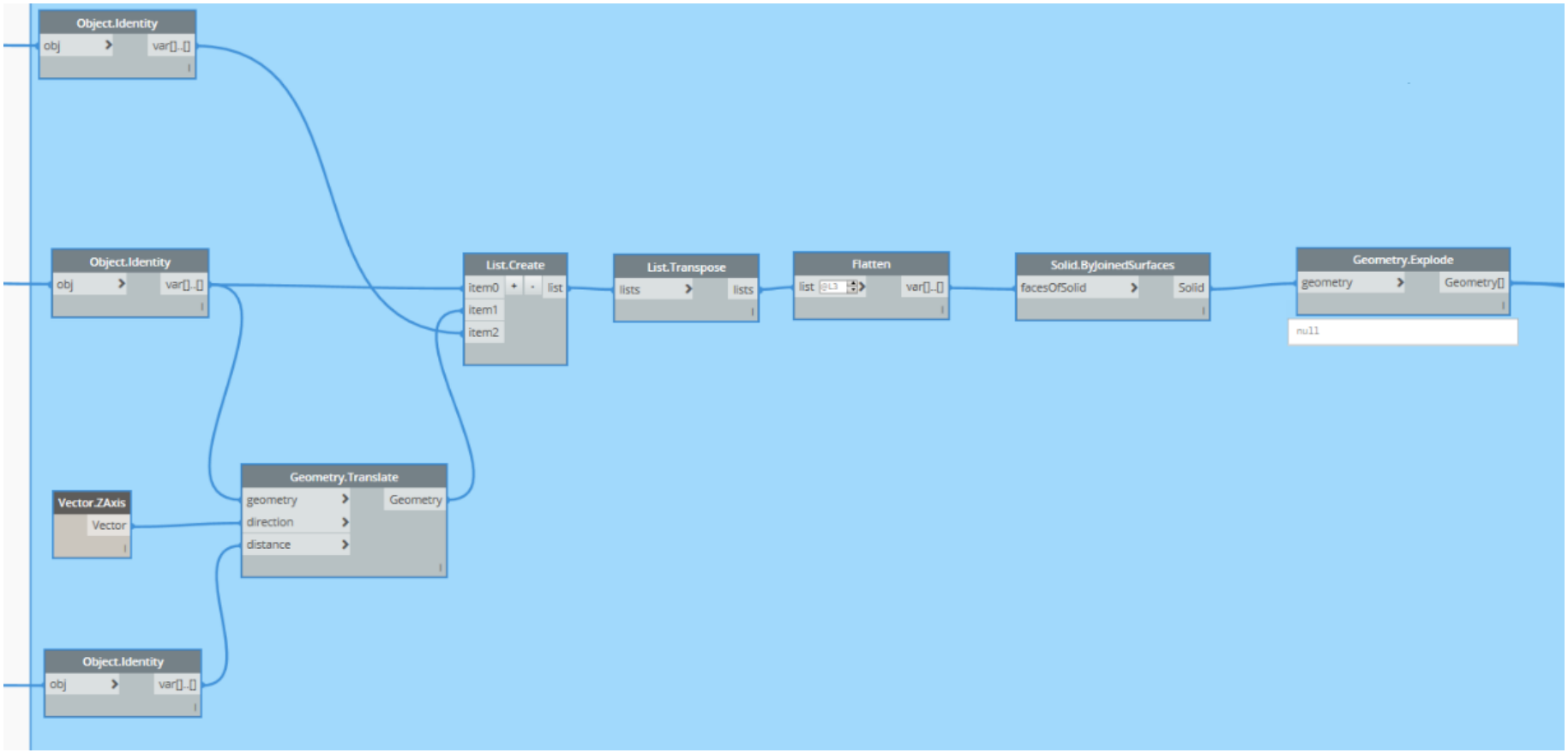

Figure 11-34 Solid base surface remap script

Part II

Vertical surface evaluation with regards to outward-facing surface.

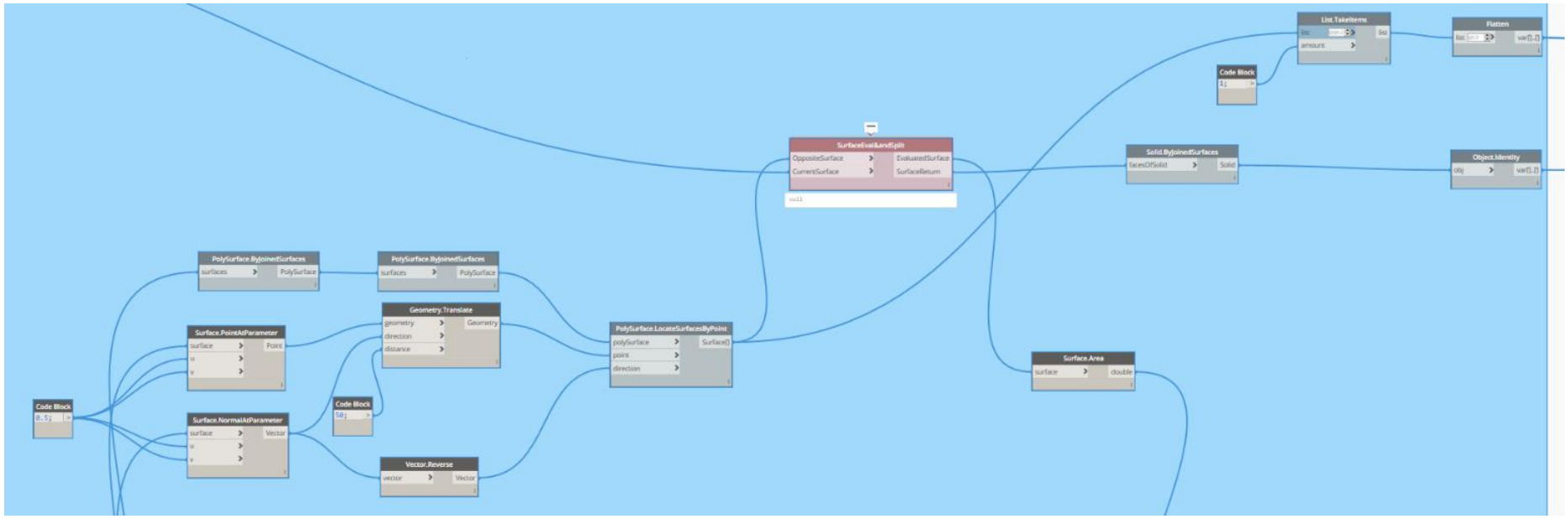

Figure 11-35 Vertical surface crop methodology.

5. Construction linking to the surface element num. – 8

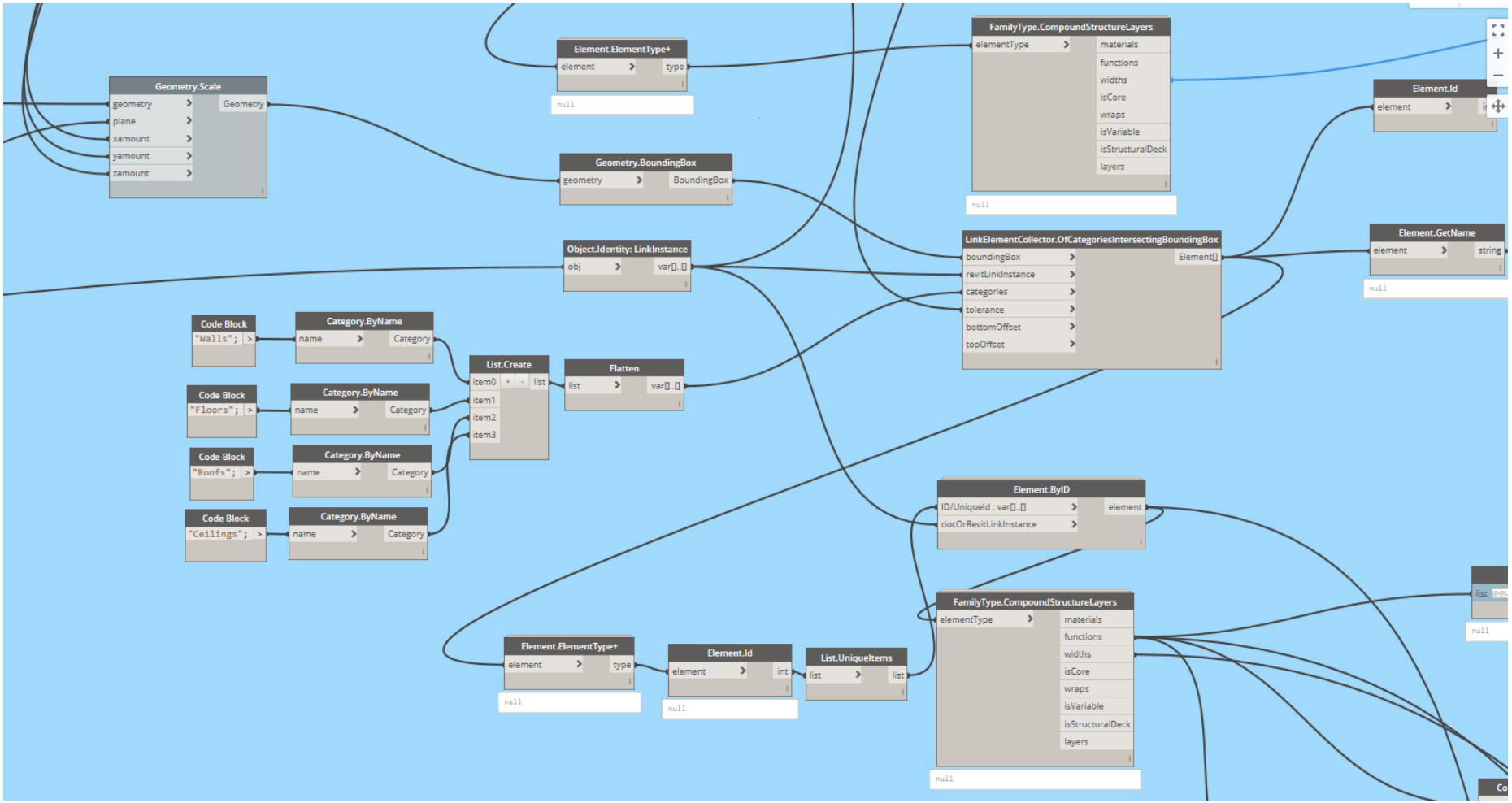

Figure 11-36 Construction assignment to related surface

6. Split surface evaluations num. - 5

This part is redundant at the current methodology as the custom Iron Phyton node (SurfaceEval&andSplit) evaluates each surface individually. This particular set of nodes were used to identify if the surface is shared by more than one space. Accordingly, a surface was cropped into appropriate segments and referenced between two spaces. However, it turned out that such evaluation is not the best approach as multiple points had to be modelled and it is strongly depended on wall size. At the moment, the data from this part of the script is used by BSim Data Modeler to identify possible inaccuracies in the vertical surfaces, such as surfaces that were formed after boundary curve transformation (see Figure 11-38).

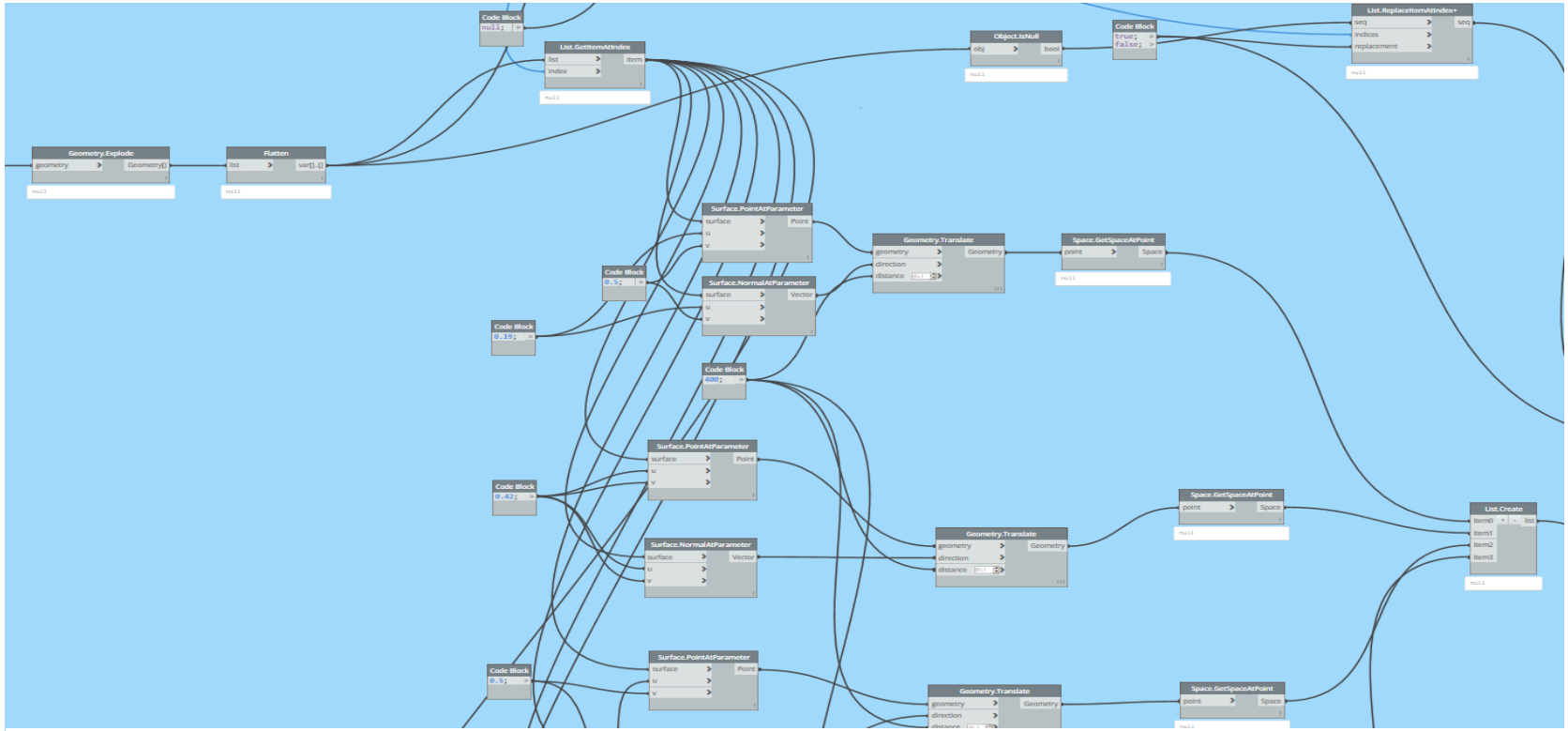

Figure 11-37 Surface evaluation at 5 points

Because of the different thickness of the external wall boundary curve after transformation changes completely. Therefore, it results in a new boundary segment which should be evaluated explicitly as in SimView similar surface must be modelled as a separate surface (see Figure 11-38).

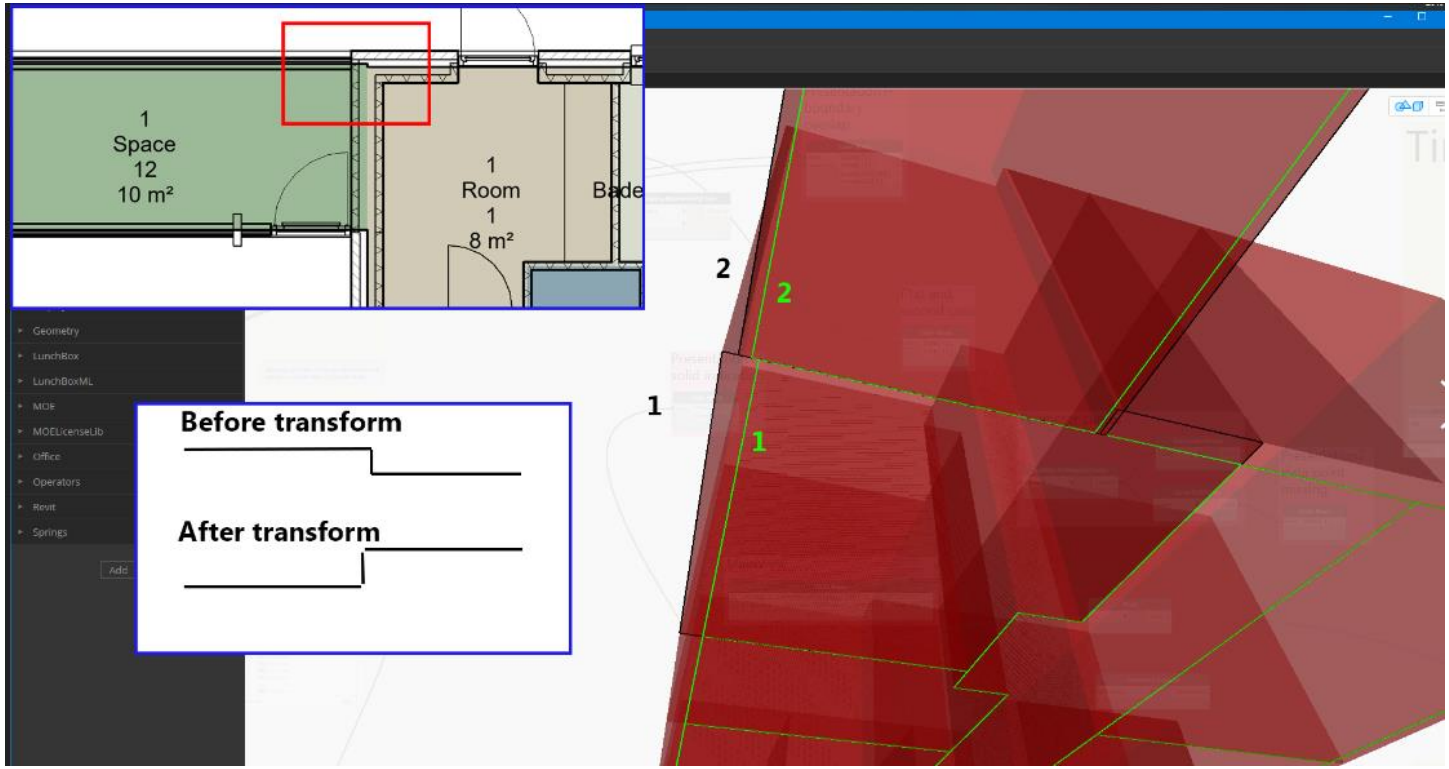

Figure 11-38 Boundary curve transformation (the green line indicates original boundary, and the black line indicates transformed boundary curve))

7. Indoor/outdoor 3-point evaluation methodology num. - 5

For the heat loss calculation, this particular script part was used to determine temperature difference across the partition. Three-point evaluation script was reused to define facing space id across the related partition.

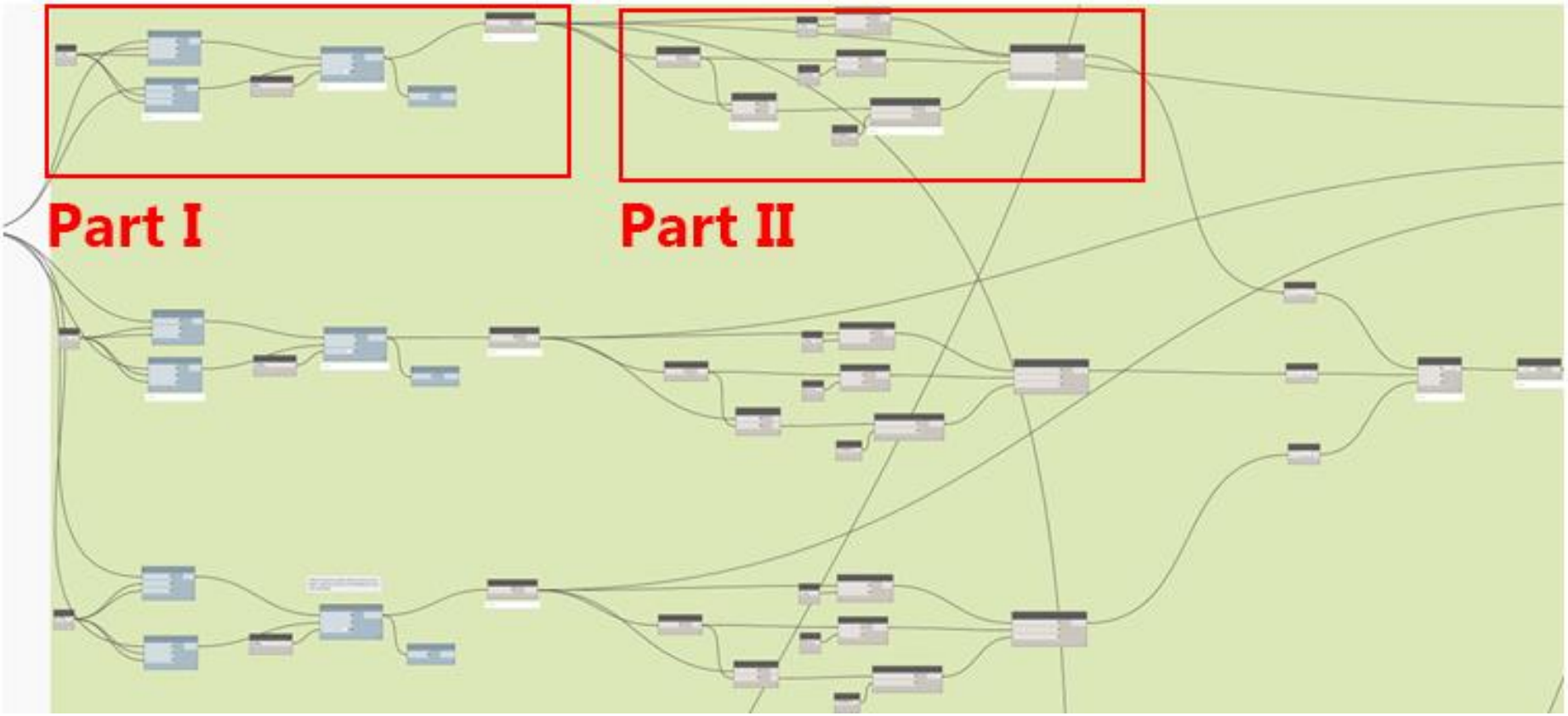

Figure 11-39 Outdoor/Indoor evaluation by projecting normal vectors

Part I

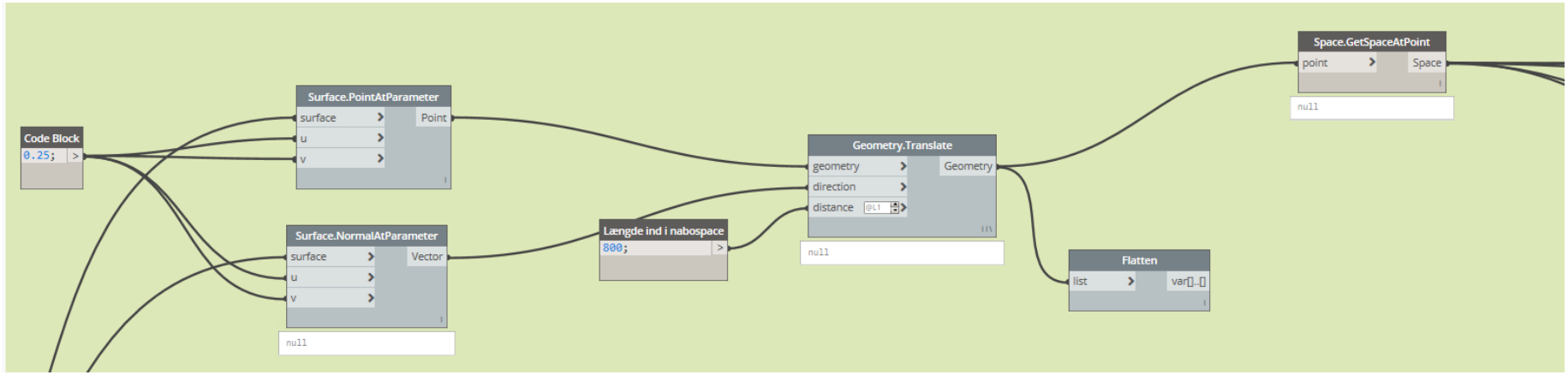

Figure 11-40 Surface point offset

Part II

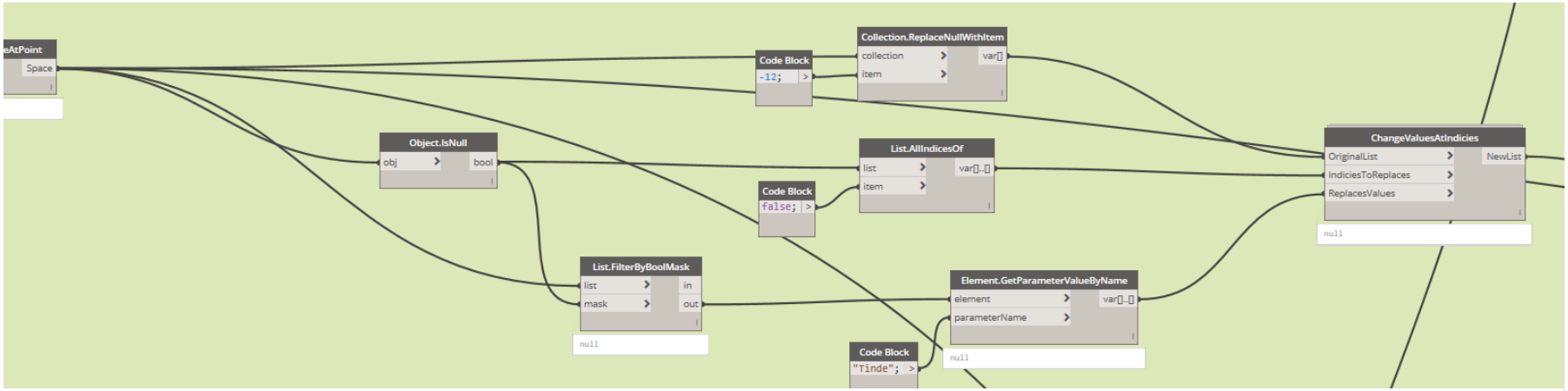

Figure 11-41 Filter surface side by indoor/outdoor direction

N. Windoor data Dynamo script walkthrough

1. Windoor data script overview

Please note that script screenshot captures presented in this APPENDIX is not cleaned. Therefore, it includes all nodes for testing cases that were presented in the report.

Figure 11-42 Windoor data Dynamo script overview

Explanation:

1. Solid import and window/door element collector methods
2. Indoor/outdoor evaluation
3. Obtaining window/door opening polygon, converting polygons to surface elements
4. Internal opening evaluation (trim intersections, obtaining appropriate host, remapping internal surface geometry on the host surface)
5. External opening evaluation (obtaining intersections, trimming overlapping entities, locating element host, remapping external surface geometry on the host surface, evaluating duplicated openings)

6. Joining internal and external openings into joined collections
7. Windoor data collection sorting methods for export to Microsoft Excel.

2. Solid import and window/door element collector method num. - 1

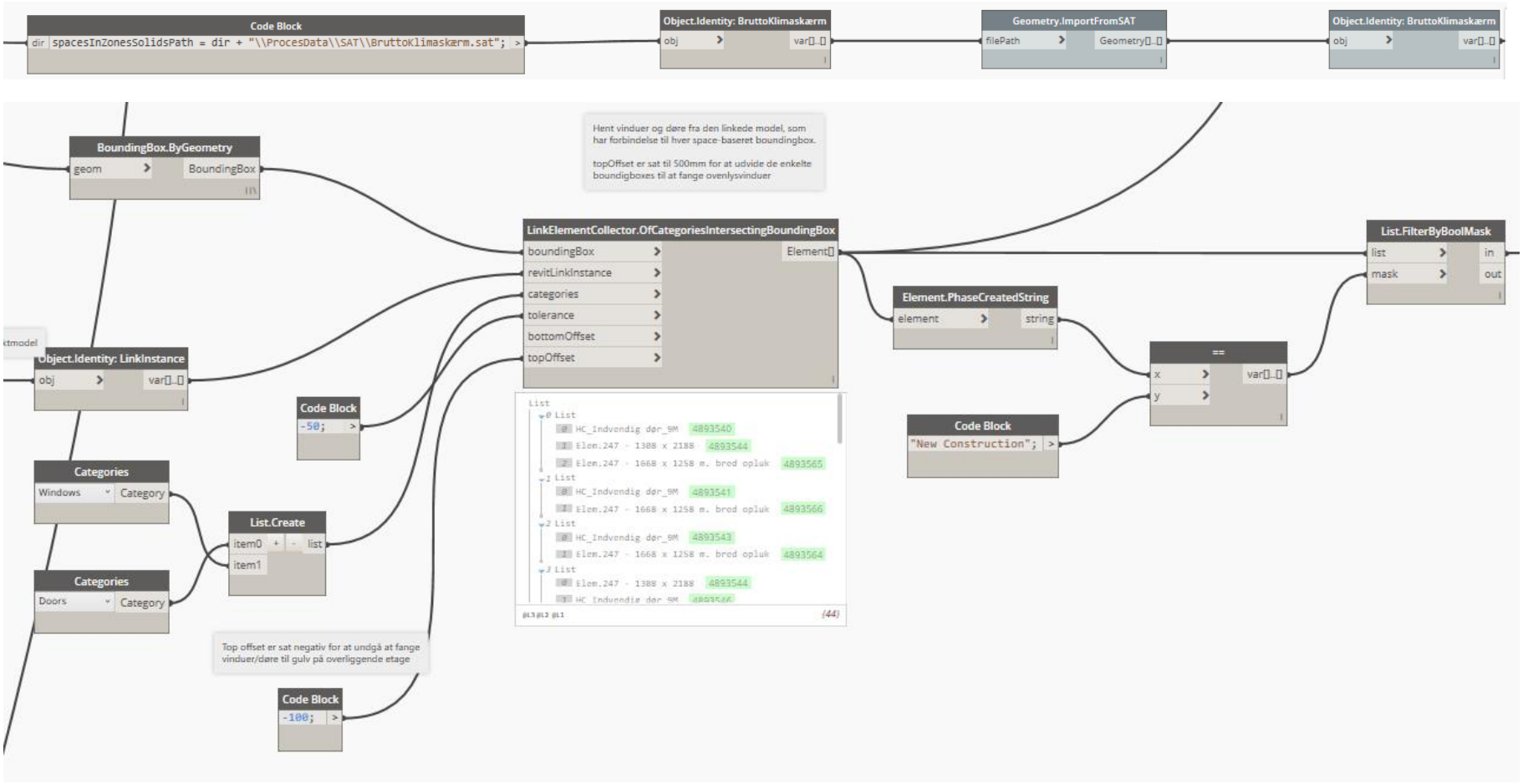

Figure 11-43 Window/door element collector method

3. Indoor/outdoor evaluation num. - 2

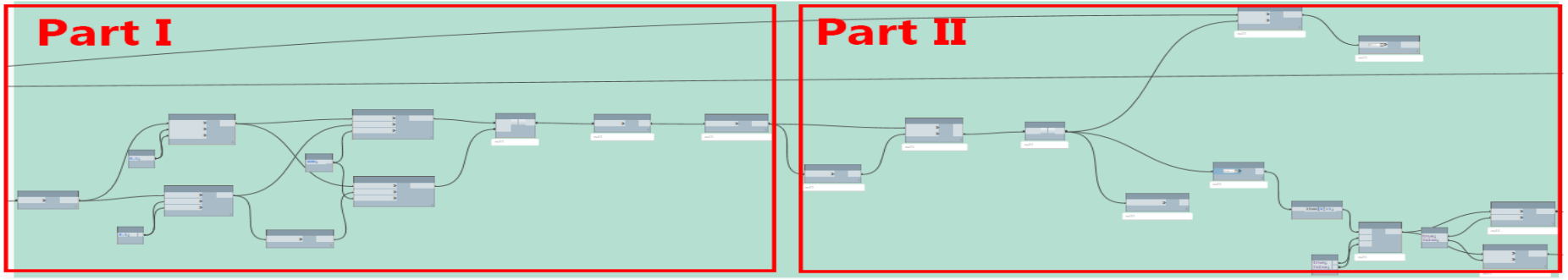

Figure 11-44 Indoor/outdoor opening evaluation

Part I

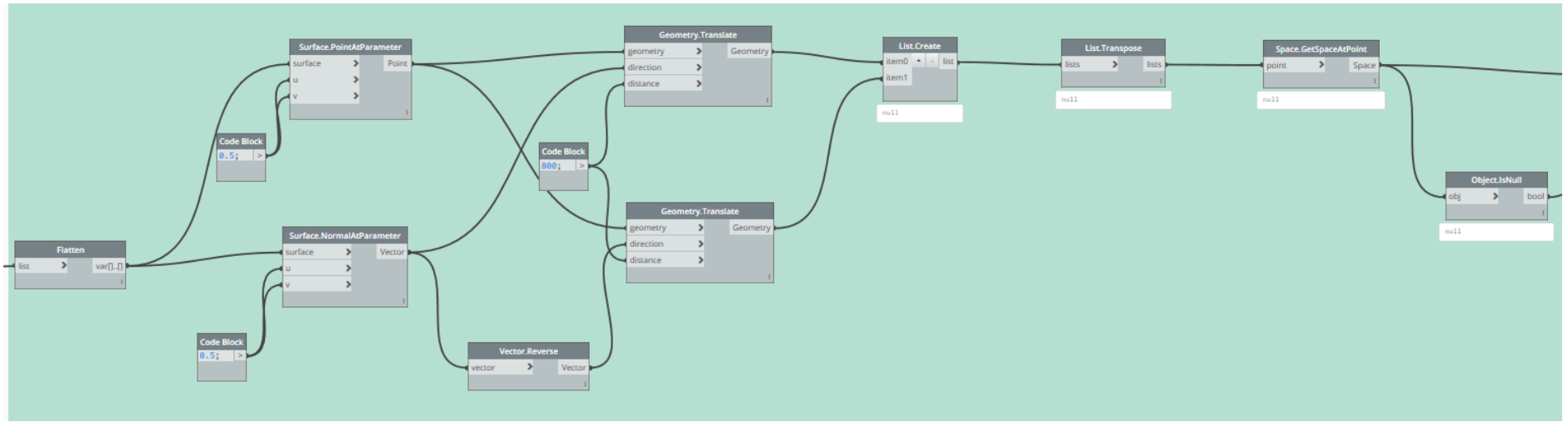

Figure 11-45 Space identification at point

Part II

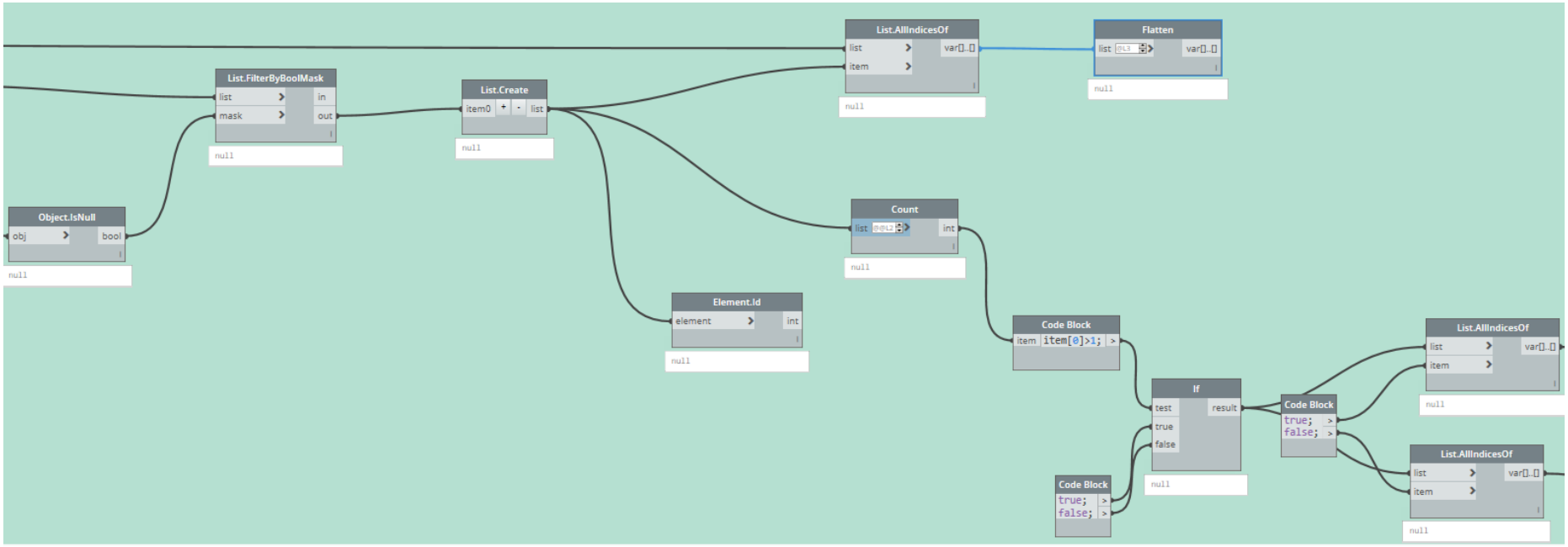

Figure 11-46 Indoor/outdoor opening sorting procedure.

4. Obtaining window/door opening polygon, converting polygons to surface elements num. - 3

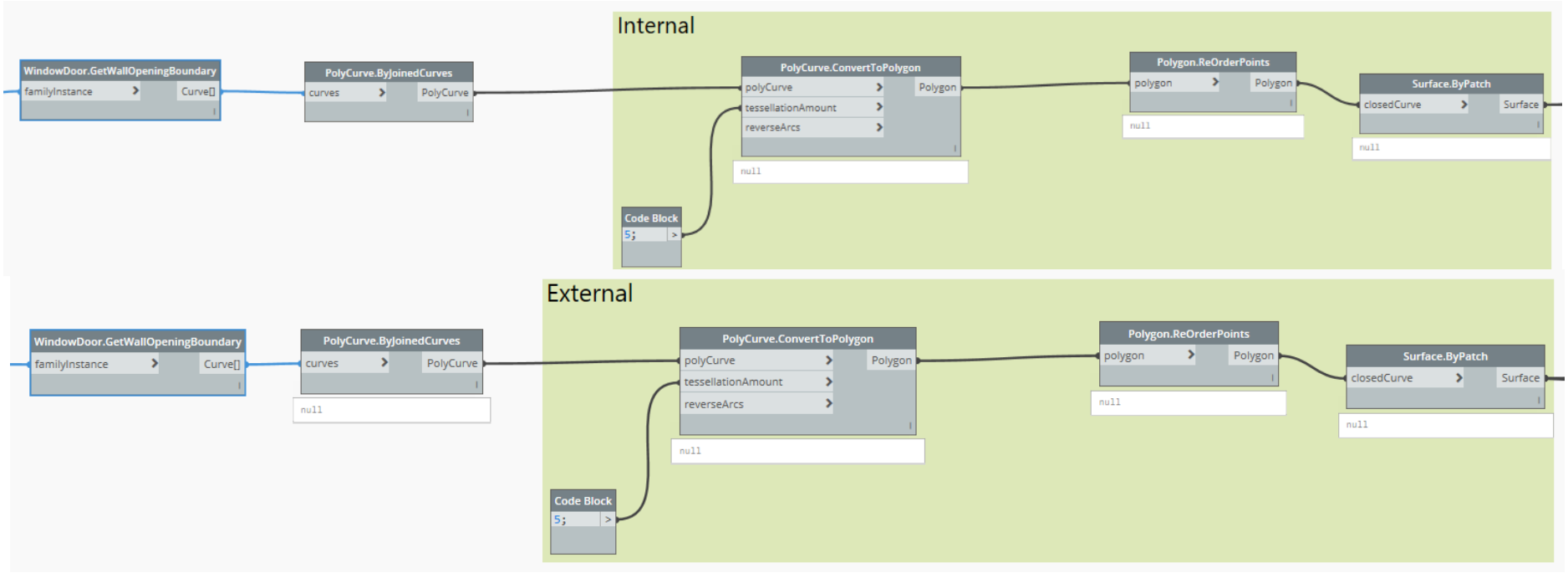

Figure 11-47 Internal/external polygon conversion to surface

5. Internal opening evaluation num. - 4

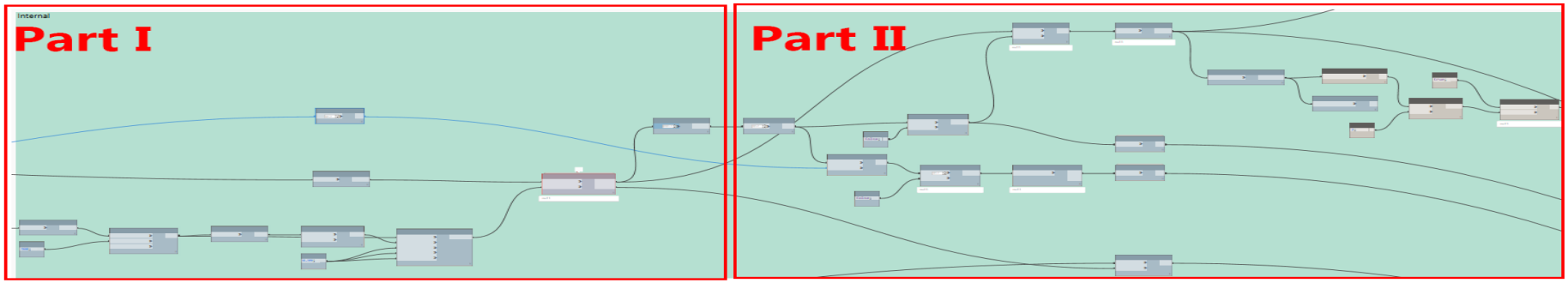

Figure 11-48 Internal opening evaluation overview

Part I

Internal openings at this point are rescaled by a fraction so it won't intersect with reference level edge. Afterwards, it is hosted on the surface.

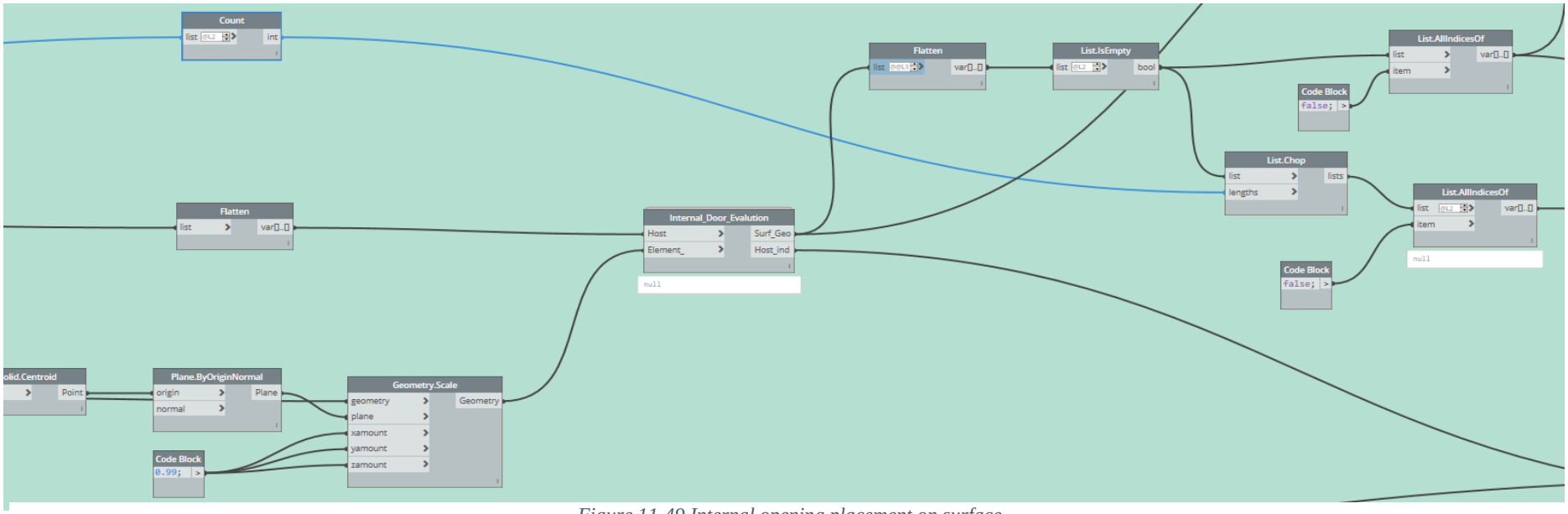

Figure 11-49 Internal opening placement on surface

Part II

This part of the script links selected openings to space and the host (the surface that includes opening).

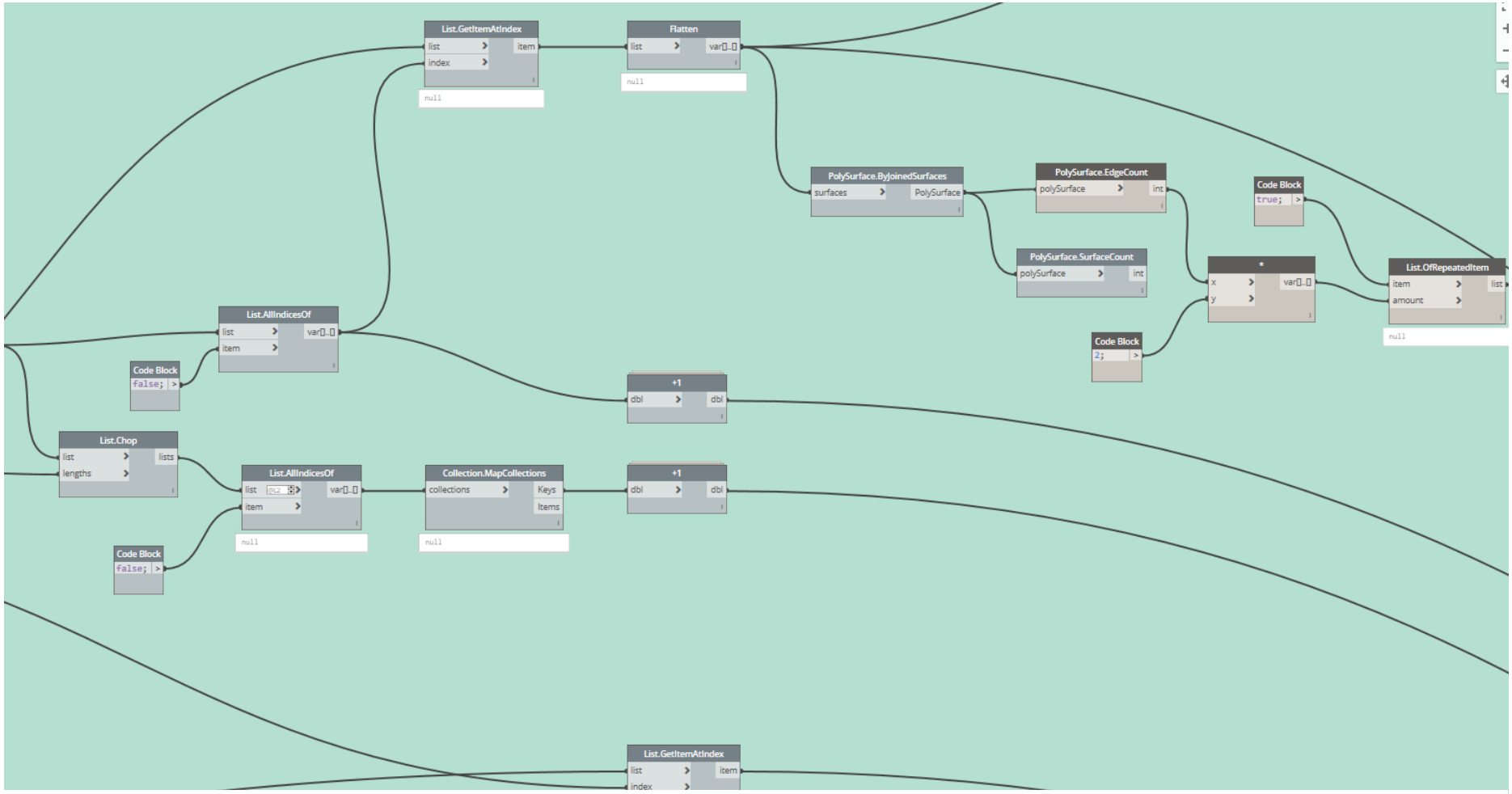

Figure 11-50 Internal opening assigned to related space and host element

6. External opening evaluation num. – 5

The external opening script was extended by a unique opening evaluation. During the project, it was determined that window/door collector gathers multiple external window and door openings. For this reason, the same logic could not be used, as it was used for internal openings. Moreover, the similar separation between external and internal openings allows control of disabling internal opening modelling in BSim Data Modeler.

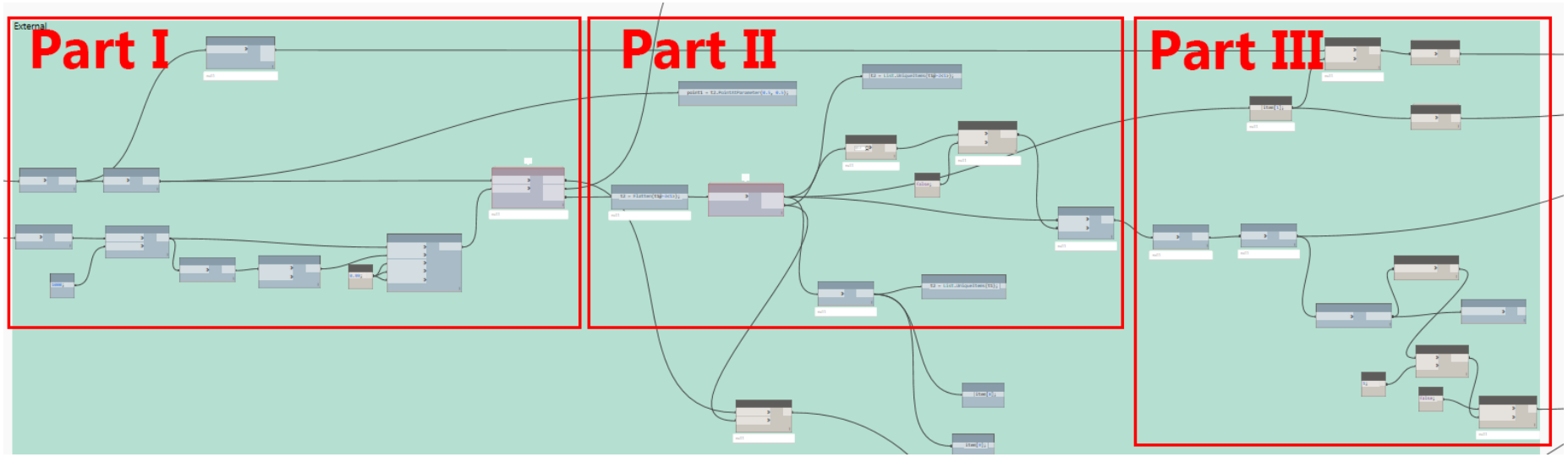

Figure 11-51 External opening surface evaluation procedure overview

Part I

External openings at this point are re-scaled by a fraction so it won't intersect with another opening. Afterwards, it is hosted on the surface.

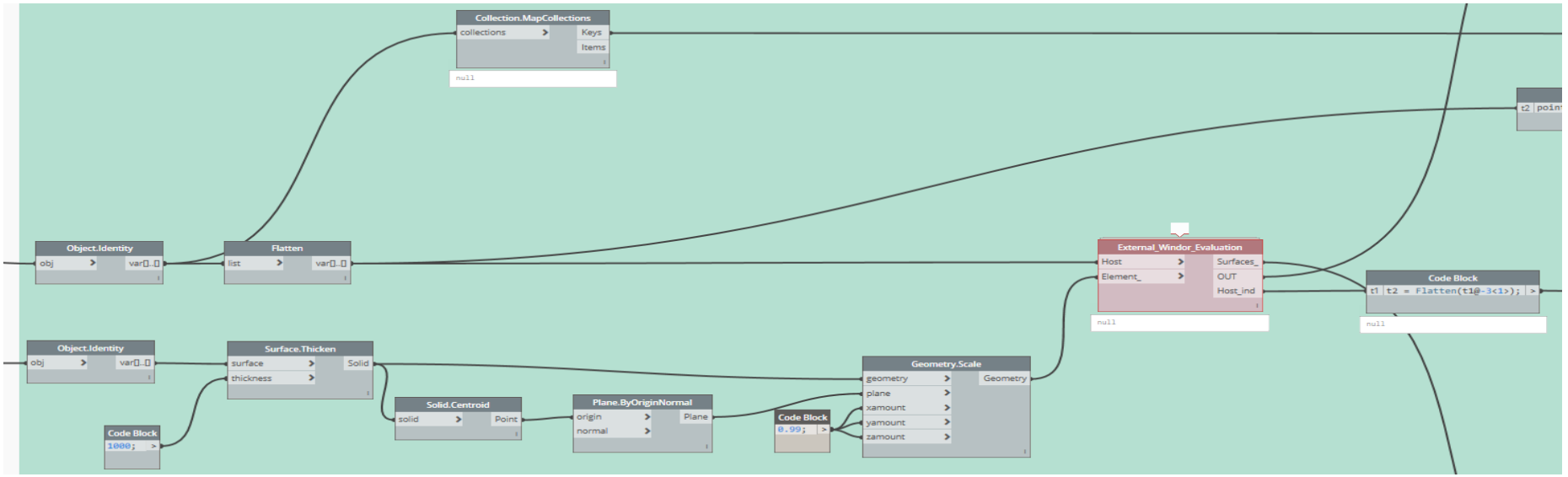

Figure 11-52 External opening evaluation with regards to vertical surface

Part II

Duplicated opening evaluation.

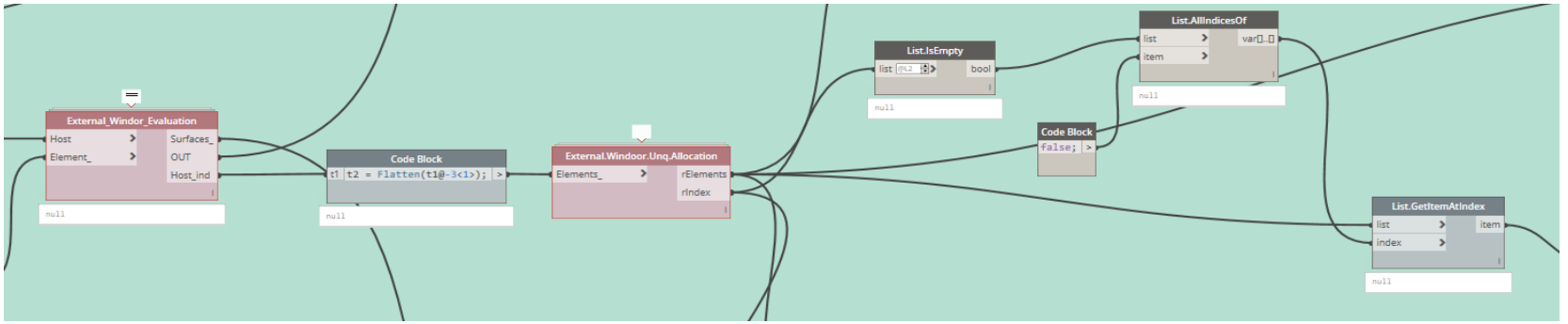

Figure 11-53 Locating unique openings

Part III

This part of the script links selected openings to space and the host (the surface that includes opening).

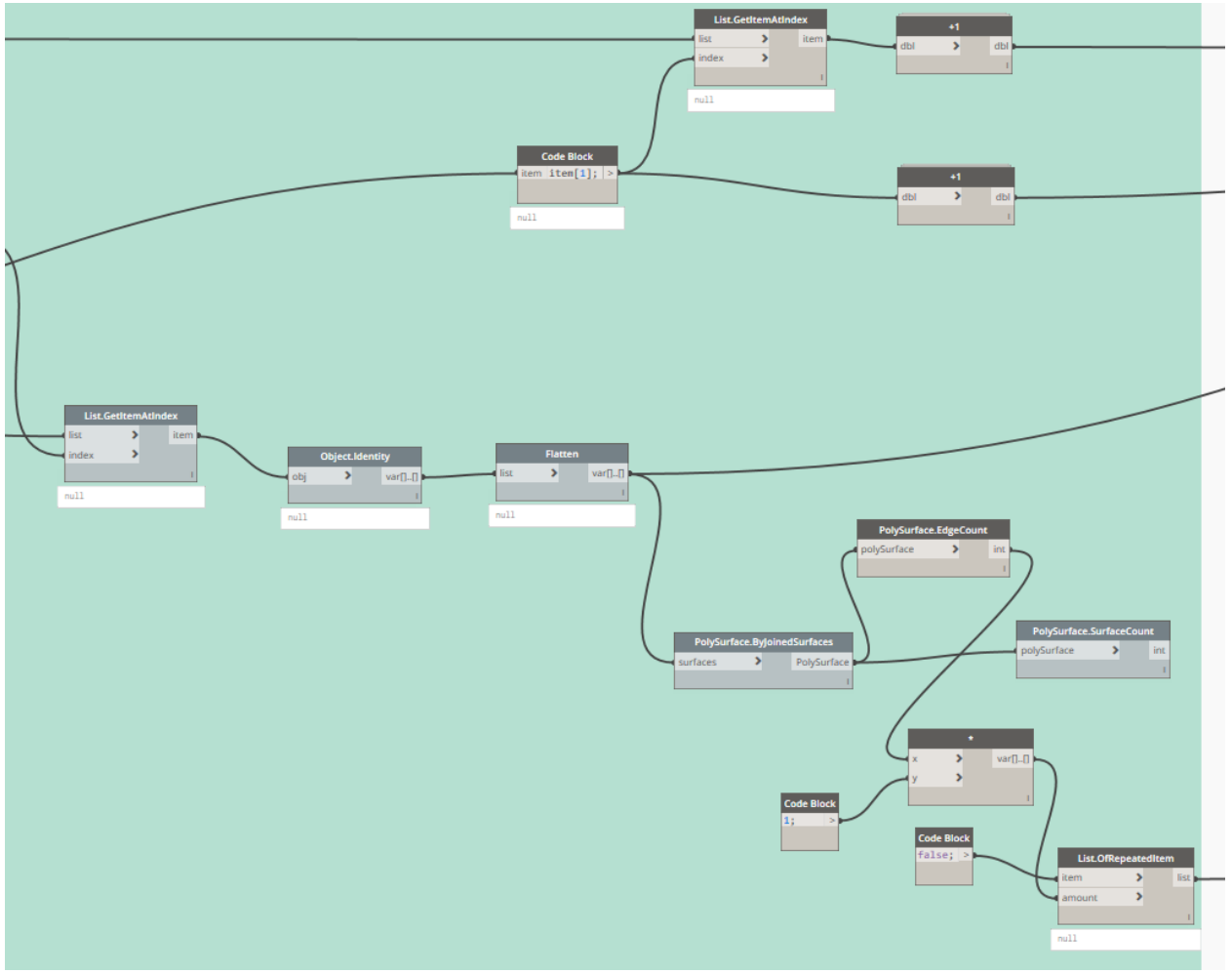

Figure 11-54 External opening assigned to related space and host element

7. Datapoint export methodology

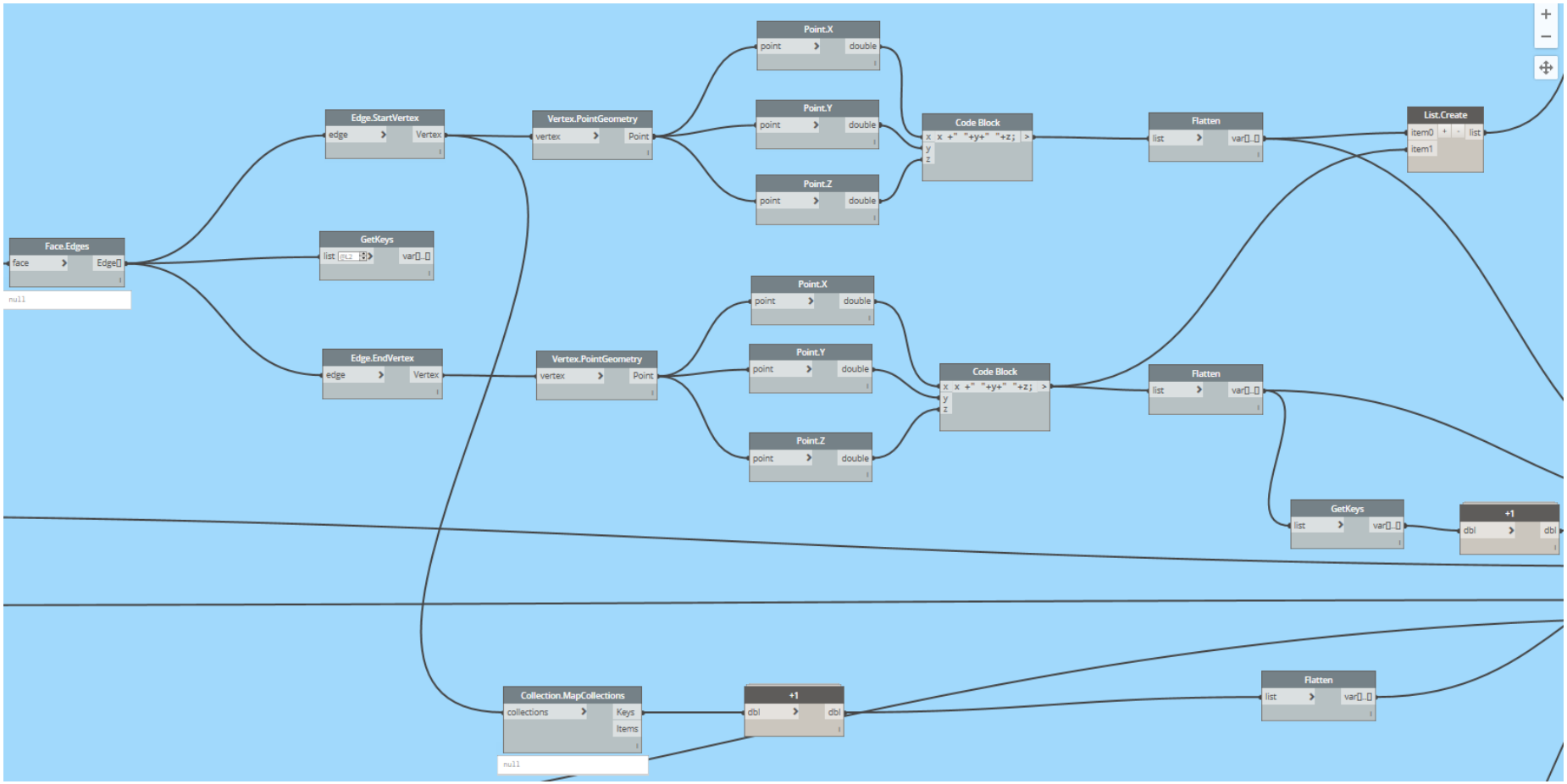

Figure 11-55 Windoor data point collections

O. Custom nodes code snippets

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
solids = IN[0]
csolids = IN[1]


lst = []
croppedindex = []
previousSpc = []

def wascrop(index):
    dublicate = 0
    for s in croppedindex:
        if s == index:
            dublicate = s
    return dublicate
def takemain(surfaces):
    surf1 = surfaces[0]
    surf2 = surfaces[1]
    if surf1.Area > surf2.Area:
       return surf1
    else:
       return surf2

for index, item in enumerate(solids):
    cs = index
    for index2, item2 in enumerate(solids):
        if cs != index2:
            if item.DoesIntersect(item2):
               if wascrop(index2) == 0:
                  #croppedindex.append(index)
                  croppedindex.append(index2)
                  allsurface = csolids[index].Split(csolids[index2])
                  if len(allsurface) > 1:
                      lst.append(takemain(allsurface))
                      previousSpc.append(index)
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = [lst, previousSpc]
```

Accept Changes Cancel

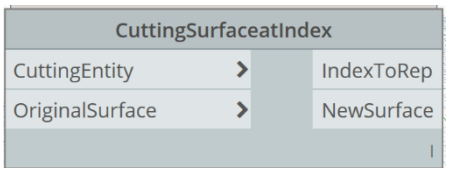

Figure 11-57 Cutting surface at index node

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

clr.AddReference('DSCoreNodes')
import DSCore as DS
from DSCore import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
points = IN[0]
curve = IN[1]
list = []
test = []
ncurves = []
returncurv =[]

def containsPoint(curve, pointss):
    if len(pointss) > 0:
       newcurves1 = curvesplit(curve, pointss)
       for point in pointss:
           if point.IsAlmostEqualTo(curve.StartPoint) == False and
           point.IsAlmostEqualTo(curve.EndPoint) == False:
              list.append(point)
               #list.append(curve)
    return newcurves1

def curvesplit(curve, points):
    newcurves2 = curve.SplitByPoints(points)
    ncurves.append(newcurves2)
    return newcurves2

pointlist = []
for index, item in enumerate(curve):
    subout = []
    for s in item:
        for pindex, point in enumerate(points):
            if point.DoesIntersect(s):
               pointlist.append(point)
               #containsPoint(s, point)
        surf = containsPoint(s,pointlist)
        del pointlist[:]
        subout.append(surf)
    returncurv.append(DS.List.Flatten(subout,2))
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = returncurv
```

Accept Changes Cancel

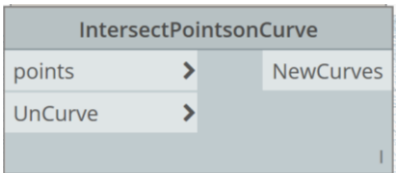

Figure 11-56 Intersecting points on boundary curve node

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
pirminis = IN[0]
esamaerdve = IN[1]

globlst = []

def Cerdve(pirminis, antrinis, ind1, ind2):
    evalFace = round(esamaerdve[ind1][ind2].Area,0)
    rFace =  round(pirminis.Area)
    if evalFace == rFace:
       return antrinis
    else:
       return pirminis

def evalA(first, second, ind1, ind2):
    fitem = round(first.Area,0)
    sitem = round(second.Area,0)
    if fitem == sitem:
        return first
    else:
        return Cerdve(first, second, ind1, ind2)


for ind, item in enumerate(pirminis):
    subout = []
    for ind2, s in enumerate(item):
       counter = len(s)
       if counter == 1:
          subout.append(s[0])
       elif counter == 2:
         subout.append(evalA(s[0], s[1], ind, ind2))

       else:
           raise Exception('Surface indetification failed')
    globlst.append(subout)
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = globlst
```

Accept Changes | Car

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
csurf = IN[0]
obsurf = IN[1]

rsurfaces = []
previousS = []
previousSurf = []
glb = []

def div(surfB, surfS):
    sp = surfB.Split(surfS)
    rsurfaces.append(sp)
    return sp

    #rsurfaces.append("Even")
def evaluate(cursurf, obssurf, ind, ind2):
    esurf1 = round(cursurf.Area,0)
    esurf2 = round(obssurf.Area,0)
    if esurf1 > esurf2:
       surfaces = div(cursurf, obssurf)
       return surfaces
        #rsurfaces.append("NOT")
    else:
        return 0


for ind, s in enumerate(csurf):
    subSpc = []
    for ind2, n in enumerate(s):
       isSurf = evaluate(n, obsurf[ind][ind2], ind, ind2)
       if isSurf != 0:
          subSpc.append(isSurf)
       else:
           subSpc.append(csurf[ind][ind2])

    glb.append(subSpc)

#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = [rsurfaces, glb]
```

Accept Changes | Cancel

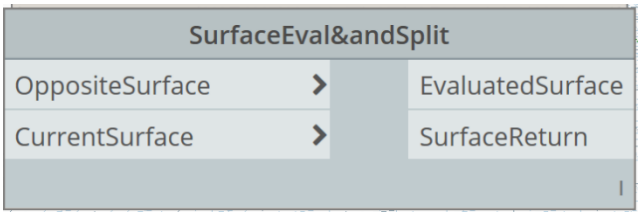

Figure 11-58 Surface evaluation and split node

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
surfaces = IN[0]
doors = IN[1]
outlst = []
centroid = []
trlist = []
dout = []
surfid =[]
did = []

def getsame(door, index, subout, dId):
    try:
        for item in subout:
            centroid1 = door.Centroid()
            centroid2 = item.Centroid()
            centroid.append(centroid1)
            if centroid1.IsAlmostEqualTo(centroid2):
               print("True")
               trlist.append("true")
               surfid.append(index +1)
               did.append(dId)
               return 0
            else:
                return 1
    except IndexError:
          return 0

def getdoorGeometry(surf, cdoor):
    geo = surf.Intersect(cdoor)
    return geo

for index, item in  enumerate(surfaces):
    subout = []
    suboutSurf = []
    for dind, door in enumerate(doors):
        if door.DoesIntersect(item):
           object = getsame(door, index, subout, dind)
           if object != 0:
              subout.append(door)
              suboutSurf.append(getdoorGeometry(item, door))
    outlst.append(subout)
    dout.append(suboutSurf)
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = [dout, surfid, did]
```

Accept Changes Cancel

Edit Python Script...

```python
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
surfaces = IN[0]
doors = IN[1]
outlst = []
centroid = []
trlist = []
dout = []
surfid =[]
did = []
nid = []

def getsame(door, index, subout, dId):
    try:
        for item in subout:
            centroid1 = door.Centroid()
            centroid2 = item.Centroid()
            centroid.append(centroid1)
            if centroid1.IsAlmostEqualTo(centroid2):
               print("True")
               trlist.append("true")
               surfid.append(index +1)
               did.append(dId)
               return 0
            else:
                return 1
    except IndexError:
          return 0

def getdoorGeometry(surf, cdoor):
    geo = surf.Intersect(cdoor)
    return geo

for index, item in  enumerate(surfaces):
    subout = []
    suboutSurf = []
    for dind, door in enumerate(doors):
        if door.DoesIntersect(item):
           object = getsame(door, index, subout, dind)
           if object != 0:
              subout.append(door)
              suboutSurf.append(getdoorGeometry(item, door))
              nid.append(dind)
    outlst.append(subout)
    dout.append(suboutSurf)
from Autodesk.DesignScript.Geometry import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = [dout, surfid, did, nid]
```

Accept Changes Cancel

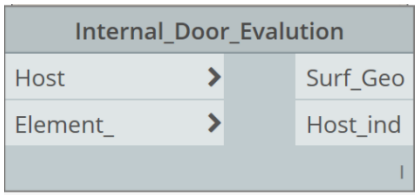

Figure 11-59 Internal door evaluation node

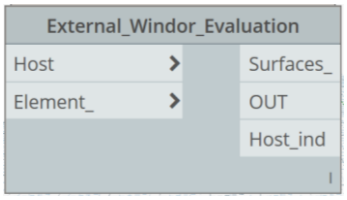

Figure 11-60 External window/door evaluation node

Edit Python Script...

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

clr.AddReference('DSCoreNodes')
import DSCore as DS
from DSCore import *
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
MainSurface = IN[0]
elInd = IN[0]
outlst = []
index = []
testlst = []
rmvInd = []

counter = 0
def GnSurf(surf):
    tempUniqC = []
    tempUniqS = []
    for s in surf:
       point = s.PointAtParameter(0.5, 0.5)
       global counter
       counter += 1
       if point in tempUniqC:
         rmvInd.append(counter +1)
         continue
       else:
           tempUniqC.append(point)
           tempUniqS.append(s)
       IntersectGr(tempUniqS)
    return tempUniqS

def GPoint (item, indO):
    tempCp = []
    for ind2, item2 in enumerate(item):
        #cpoint = item2.PointAtParameter(0.5, 0.5)
        tempCp.append(item2)
    #t2 = List.UniqueItems(tempCp)
    #outlst.append(tempCp)
    out = GnSurf(tempCp)
    if len(out) !=0:
      for i in out:
         index.append(indO)

    outlst.append(out)
    return out

def IntersectGr(unqobj1):
    tempWinds = []
    tempW = []
    for s in unqobj1:
       tempWinds.append(s)
    if(len(unqobj1) > 1):
      for ind, item in enumerate(unqobj1):
        tempWinds.pop(ind)
        for i in tempWinds:
            if(item.DoesIntersect(i)):
              tempW.append(i)
              testlst.append(item)
              testlst.append(tempW)
        tempWinds.append(tempW)
    return tempW

for ind, item in enumerate(MainSurface):
    unqobj = GPoint(item, ind)
#
# Copyright Arturas Pranskunas email: pranskunasarturas@gmail.com
#
OUT = [outlst, index, rmvInd]
```

Accept Changes Cancel

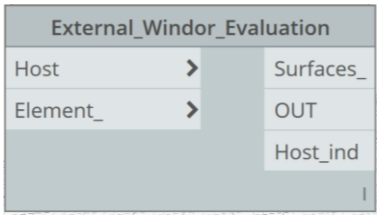

Figure 11-61 External window/ door evaluation node