Building façade renovation through parametric panel generation for Dutch housing

By T.R.L. Pluimers

TU Delft
Delft University of Technology
Faculty of Architecture
and the Built Environment

Msc. Graduation Thesis
Building Technology
June 2021

Title:
Building façade renovation through parametric panel generation for Dutch housing.

Author:
Ing. T.R.L. (Thijmen) Pluimers
Student nummer: 4992032

Mentors:
Dr. ing. T (Thaleia) Konstantinou MSc
Product Innovation
T.Konstantinou@tudelft.nl

Dr. M. (Michela) Turrin
Design informatics
M.Turrin@tudelft.nl

Preface

This thesis is the concluding piece of the Building Technology study master at the faculty of Architecture at the Delft University of Technology. The research described in this paper covers the last 9 months of the master program and is the graduation work of my masters.

During my studies façade has always interest since starting my master an interest in parametric has grown and this topic allowed me to use both while also working on a current problem in the building industry. Which is the need to renovate more than 1000 houses every day in order for them to be heated with electricity and disconnect all houses from the natural gas supply.

The research has been a great way of learning more about the scripting and coding aspect as well as the façade design for renovation. And although with the current Corona situation I learned and enhanced the skill I wanted to.

I would like to thank my mentors Thaleia Konstantinou and Michela Turrin for their guidance and support through this process. They helped me sharpen the thesis it's methodology and understood my ideas before I could even properly communicate them. Furthermore, I would like to thank Hans Hoogenboom for helping me with the process of transferring my initial ideas to Python code.

Abstract

The Dutch government has set the goal to disconnect all houses from the natural gas network that runs through our country. This to decrease emissions and meet the goals set in the Paris Agreement in an effort to halt the global mean temperature increase. To provide heat to the houses they will need to be heated with electric energy. This heating system is most efficient using lower temperatures therefore the houses will need to have a building envelope which has the right insulation levels. Meaning that a lot of older houses will need to be insulated better and therefore renovated. To reach this goal more than a 1000 house will need to be renovated every day. Ranging from small addition of insulation to houses that have no insulation at all.

To provide this solution this thesis researched the possibility of using parametric design to generate timber frame renovation panels. By doing so the time to engineer these panels could drastically drop while it would still be possible to adjust panels to building owner or designer's needs.
The boundary of this thesis is set to research the possibilities on prewar row houses as these have the largest impact with the consumption of 21 TWh every year. The process of renovation using prefab elements has been studied to determine the steps that are to be performed to complete the renovation. Within these steps the panels are engineered. This step has been disected into more detail to see what steps are taken to go from design to a panel that can be manufactured. The steps are then automized using python and grasshopper as software and the engineers decisions are implied by parameters.

The steps taken to generate the panels are: determining the to be panelized area, composing the panel contours, creating the panel geometries and calculating the panel specific data. These steps are to be automated by using the step specific parameters which are: the panel size limit, tolerances and geometry sizing and properties. These are decided by the engineer or are already established in the pre-engineering phase. Together they enable the tool to generate building information models of prefabricated building envelope panels which can be adjusted by its user if required.

Contents

Glossary

Thermology	Meaning
Adjustable	meaning that something can be adjusted throughout the design process.
Automation	An industrial process where mechanic tools normally used by human labors are automized to perform a specific task automatically. (Richard, 2004)
BIM	Building information model - a digital 3D model that contains information about the project. Generally, the level of detail (LOD) is determined to set a standard for the amount of detail in the model.
Computational method	A process that makes use of computing power. The method describes a certain process without an already defined solution to perform the process.
Computational tool	the tool performs the process described as the computational method.
Parametric method	a computational method that makes use of parameters as an input.
Parametric tool	the tool performs the process described as the parametric method.
Prefabricate	prefabrication is the process of fabricating before. Meaning the fabrication of elements or modules before the arrive on the building site.
robotization	One step further then atomization. The task that are done by the automized tools are now completed by robots. Meaning they can be programmed to do different tasks to different elements rather than being limited to one task and one element type. (Richard, 2004)
ZEB	Zero energy building, there are different opinions as to when a building is zero energy. In this thesis a building is zero energy as over the course of a year it does not use more energy than it generates.

1. Orientation phase

In this chapter the introduction to the subject and its framework is given. Starting with the background and through the problem statement forming an objective and a research question to provide a solution.

1.1. Background

The Dutch government aims to disconnect all houses from natural gas by 2050. According to a study done by the RVO (Rijksdienst voor Ondernemend Nederland) there are currently 7,9 million homes in the Netherlands. Since 2008 it is mandatory for new buildings to have an energy label. These are valid for ten years. The study of the RVO shows that more then 3,8 million houses currently have this label. Meaning that the other 4,1 million houses are not accounted for. (RVO, 2020)

Data from the CBS (Het Centraal Bureau voor de Statistiek) shows that the of the current building stock only 1,5 million has been built after 1995 (CBS, 2020). Meaning there are currently a lot of old houses without an energy label which need to be renovated to meet the required goal before 2050.

1.2. Problem statement

Renovating all these houses before 2050 is an enormous task especially because the majority of the houses are different and need a different approach to reach their energy usage target in the most efficient and effective way. Doing all these renovations at the rate they are done now will not meet the goal of 2050 (Filippidou F., 2017). The traditional way with onsite labor does not work. This is possibly also due to the decreasing number of skilled labors (CBS, 2020). This is substantiated further by the European Commission who acknowledges the necessity of prefabrication of modules for building renovation. In 2014 the horizon 2020 program was launched, including the issue Energy Efficiency (EE1: Manufacturing of prefabricated modules for renovation of building). Not only does the program aim for lowered pre-fabrication cost by using automation/robotization. It also calls for the combination with advanced computer-based tools like Building Information Modelling (BIM) in order to facilitate the industrialization process of the construction (Veld, 2015). Here in lies the problem however, to make accurate models that allow for de industrialization of the façade production a model with sufficient amount of detailing is necessary. The measurements of the façade should be exact and the panels fully detailed so they can be produced.

The mean size of a house in the Netherlands is 120 square meters (CBS, 2013). In order for this to be modelled it would take approximately a week (calculated at bimprice.com). This is without the modelling of the renovation adaptations. This shows that although we have the means to create a panel for renovation and mount it the next day, the real bottleneck is in the lies in the engineering of the renovation.

1.3. Objective

The object to be achieved in this thesis will be to help speed up the engineering process of prefab panels for building envelope renovation though parametric means. The final product will be a parametric tool that helps its user to speed up the model generating process.

1.4. Scope

In their paper about the five levels on industrialization (HU S.J., 2008) explains the levels and how the fifth level influences how the work is being done in the first four levels. Those levels are directly related to the physical process of manufacturing while the fifth is about the engineering process and making it “smarter” and more innovative to increase the efficiency in manufacturing. This thesis will focus on that fifth level by the creation of a parametric tool that provides the user with models for the prefabricated façade renovation they want to perform. The tool will be created by computational means using the software Rhino Grasshopper and its components.

There are a lot of parameters to take into account for the renovation of buildings. This research chooses to focus itself on the process of the integration of these parameters rather than dedicating itself to make sure the parameters are 100% accurate. Therefore, certain inputs are given into the tool manually which can also be calculated more accurately with programs. But as these are parameters they can be changed easily in future researches and application.

No specific material research will be done into the most optimal material for the façade renovations. For this research timber frame panels will be considered as in several papers (Pihelo P., 2017; Balkuv, 2017) about zero energy building renovations this is pointed to as the best option because it adds much isolation value compared to it thickness. Furthermore, building with timber frame elements has increased from 1-2% since the 1970’s to 5% between 2000 and 2011. It is expected to take a market share of 15% as a consequence of the CO2 policies of the government (Vis M.W., 2014). However, with the aim of the Dutch government to strongly reduce nitrogen being emitted into the atmosphere in order to protect nature reserves it could mean a real breakthrough for timber frame constructions as mentioned by Andy van Den Dobbelsteen (Belzen, 2019). Timber frame elements are widely used already in the construction industry by contractors so the knowledge about them is already there onto which this tool will build and it might increase the chance of the tool being further developed and implemented.

The façade creates the outer impression of the building. But this research focusses on the technical aspect of the process and development of the tool. The images shown in this research are a product of the author’s way of thinking. Nevertheless, the design in reality is subject to the architect creating it.

1.5. Research question

During the research the following research question will be answered:

How can a parametric tool be designed and what parameters are used for the renovation of poor energy performing buildings in order to provide designers and engineers with a building information model of prefabricated adjustable building's envelope panels?

Sub questions

To answer the research question the following sub-questions have been composed:

1. *Which poor energy performing buildings are being considered?*
2. *What façade renovation method is being considered?*
3. *How is the parametric method designed for the building renovation?*

1.6. Methodology

The research methodology describes how the research is being done. What type of research is being used for what topic and how it helps validate the research.

In the introduction the background and the problem are defined to do so desk study was used to clarify the background of the problem and literature to verify it.
Afterwards desk research will be used to solidify the theoretical framework for the thesis. First into the building stock for which literature will be reviewed to come up with numbers of houses and their energy performance. So that an educated discission can be made on what typology the research will be mainly focused.

Next possible renovation methods will be looked at through literature review first the different strategies will be analyzed before looking deeper into a specific case studies for the chosen building typology after which the prefabrication of this method will also be investigated. Once the literature review is completed a field research will be done into prefabricated elements.

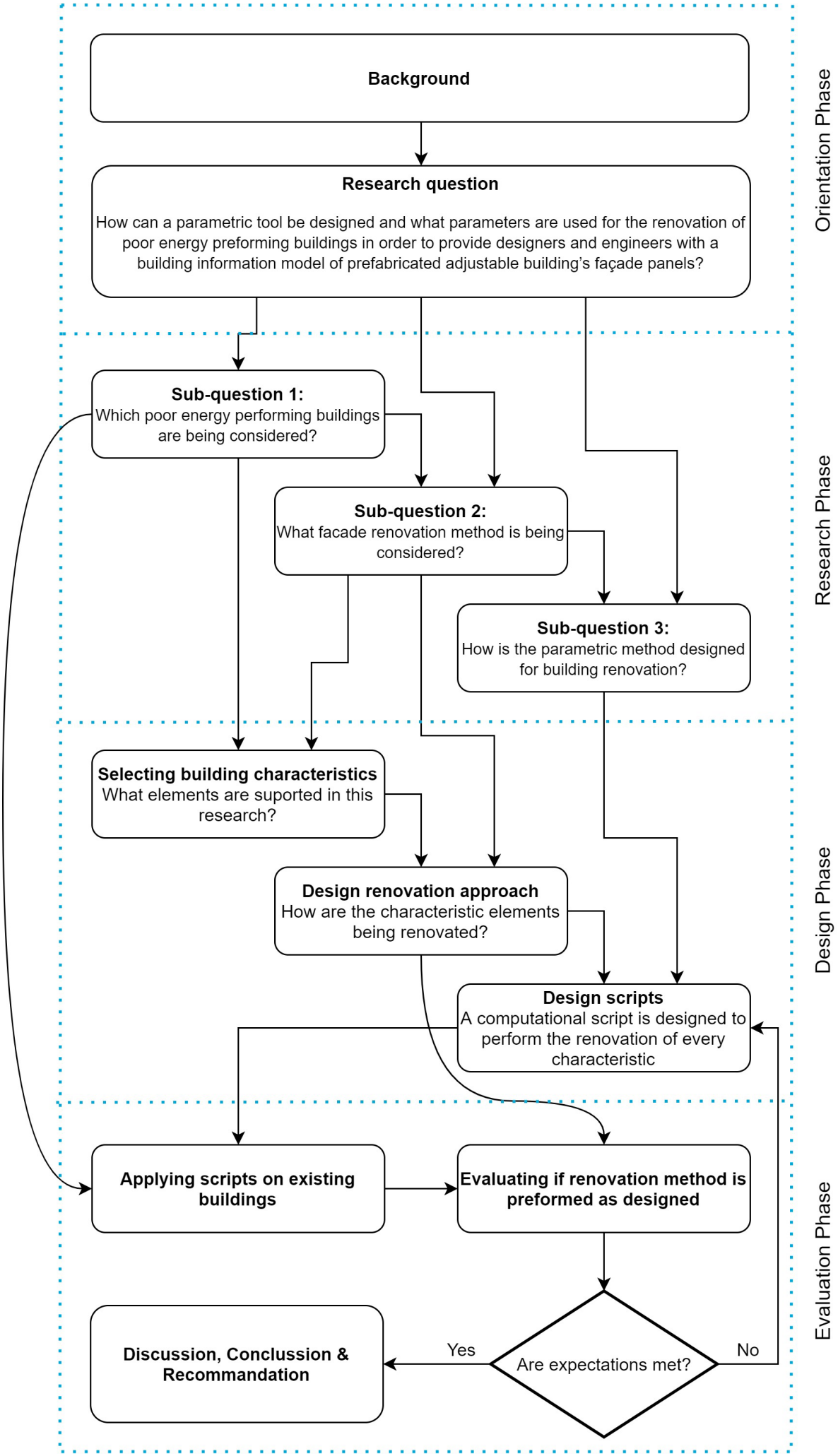
After that computational methods will be reviewed through desk research by looking at examples that fit the research scope and by reviewing literature about the topic. This will continue during the design stage of the research. Once working on and experimenting with the tool it becomes clearer what is needed. This is the research through experimentation.

During the design phase the characteristis of the building typology will be researchedon case study building to determine what characteristic can be supported with the research.

After which for these characteristics a renovation method is composed using the knowledge from the research phase. Then the steps taken in the process of renovation are identified for the next step.

The last step of the design is to design the parametric method which allows for automization of the renovation process. This is done by taking the steps identified in the previous step and evaluating if they are can be automized using a parametric tool. If so the parameters used to complete the step need to be determined and a scrip can be build.

The last part of the research will be the application of the tool, it is taken out of the design enviroment and applied on a case study building. any component that doesnt work will be adapted after which the final results will lead to the discussion, conclusion and recommandations for this thesis.


Background
Research question
How can a parametric tool be designed and what parameters are used for the renovation of poor energy preforming buildings in order to provide designers and engineers with a building information model of prefabricated adjustable building's façade panels?
Sub-question 1:
Which poor energy performing buildings are being considered?
Sub-question 2:
What facade renovation method is being considered?
Sub-question 3:
How is the parametric method designed for building renovation?
Selecting building characteristics
What elements are suported in this research?
Design renovation approach
How are the characteristic elements being renovated?
Design scripts
A computational script is designed to perform the renovation of every characteristic
Applying scripts on existing buildings
Evaluating if renovation method is preformed as designed
Are expectations met?
Yes
No
Discussion, Conclussion & Recommandation
Orientation Phase
Research Phase
Design Phase
Evaluation Phase

Report chapters

1. Introduction phase
    a. Background
    b. Problem statement
    c. Objective
    d. Research question
    e. Boundary conditions
    f. Research methodology
2. Literature phase
    a. Which poor energy preforming buildings are being considered?
        - What typology building is being considered and why?
    b. What renovation method is being considered?
        - What are the different strategies for building renovation?
        - How are buildings renovated in a prefabricated manner?
    c. How is the parametric method designed for the building renovation?
        - What are the inputs for the method?
        - What is the expected output?
        - General workflow of script
3. Design phase
    a. What are the characteristics that are common in row housing and that can be supported in the tool and this thesis?
    b. How can these characteristics be renovated?
    c. How can a computational method perform these renovation designs?
    d. Verification of the tool on existing building plans and adaptation on the tool to make it work.
4. Discussion
5. Conclusion
6. References
7. Appendices

Societal relevance

The subject of this research is a hot topic at this moment as the aim to renovate all the buildings becomes increasingly harder to achieve as time goes by and there isn't a solution that allows for mass customization. In this case it isn't the renovation that is posing the problem it is the time in which it has to be done. This is a known problem in the industry and there are already companies moving towards a more industrialized and digitalized solution for the renovation, this paper will contribute to that.

Scientific relevance

Although there has been plenty of research done on the renovation of existing housing, most of these researches looked into the method of the renovation. Little research has been done. the paper Building renovation adopts mass customization by Andrés F. Barco (Barco A.D., 2016) come closest to what is aspired to do with this research. Although it goes deeper into the preliminary stage of the renovation process rather than the creation of detailed elements. This is also a problem that is encountered in the literature survey while trying to gain knowledge of the state of art when it comes to the use of computational and parametric design in renovation or with existing facades.

1.7. Planning

	november				December			Januari				Februari		
	45	46	47	48	49	50	51	1	2	3	4	6	7	8
	1.10	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	3.1	3.2	3.3
1 - Topic definition														
1.1 topic selection														
1.2 information gathering and reading														
1.3 litrature review														
1.4 Research framework														
1.5 making planning														
P1														
2 - Research														
2.1. Building stock analyses														
2.2. Renovation method														
Strategies														
Case studies														
prefabricated renovation														
2.3 Computational design														
Input parameters														
Expected output														
Concept process plan														
P2								Prep time						
3 - Designing														
3.0 Computational method design plan														
Specify tool boundaries														
Specified process plan														
3.1 Computational method design														
Façade devision														
Penalization														
Evaluate and inprove														
P3														
P4														
4 - Finalization														
Write thesis														
Finalize research														
P5														

March				April					May				June				July	
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
3.4	3.5	3.6	3.7	3.8	3.9	3.10	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10	4.11	5.1
		Prep time																
										Prep time								
																Prep time		

2. Research Phase

Having constructed the framework of the graduation this chapter focusses on the literature that is already been made on this topic. In this way it creates a foundation for further construction of knowledge.

2.1. Which poor energy performing buildings are being considered?

There are a lot of different building typologies in the Netherlands. In this chapter the different typologies are analyzed and it is determined which building typology would have the biggest impact on the energy transition if it would be renovated to a higher energy label. This house will then be used as a starting point for the creation of the tool.

2.1.1. Current building stock analyses

A study by the Dutch government into housing typologies, there numbers and energy usage (Agentschap NL, 2011)there are 7 typologies. These will be the typologies further referred to in this thesis. Below the different typologies are described with the number of owner-occupied homes and portion of the total building stock. After which the distribution per building period and energy label will be discussed.

Detached

Detached houses are houses that is not joined by any other house (Collins, 2020). Because of this it also has the largest loss area compared to the volume as it has no external walls that are connected to other heated areas. A large part of the detached houses is owned by the inhabitants (circa 95%) at the time of the study there were 959.000 detached houses build before 2005, which is roughly 14% of the total building stock (Agentschap NL, 2011).

Semi-Detached

A semi-detached house is a house that is joined to another house on one side by a shared wall (Collins, 2020). A benefit of this is that the joint wall is between two heated area's and there is almost no heat being lost through this wall. Between 85 and 90% of the semidetached houses is owned by its residents. Finally, 824.000 semidetached houses were built before 2005, which is roughly 12% of the building stock (Agentschap NL, 2011).

Row house

A row house is a house that is part of a row of similar houses that are joined together by both of their side walls (Collins, 2020). Due to this it has less heat loss area. Roughly 60% of the row houses are owned by the dwellers. Furthermore 2.839.000 of these were built before 2005, which is roughly 42% of the building stock (Agentschap NL, 2011).

Duplex house

A duplex house is a house divided into two separate dwellings (Collins, 2020). After the second world war this was a temporally solution to the housing shortage but they are still being built in small numbers. A large part of the houses is being rented out as only 27% of the building is owned by the inhabitants. There is a total of 382.000 of this topology houses that was built before 2005, which accounts for 6% of the total building stock (Agentschap NL, 2011).

Gallery house

Gallery houses have an external passageway which provides entry to individual apartments on each floor (Collins, 2020). Due to this feature, they have two outside facing facades, one on the gallery side and one on the opposing side where a private balcony is situated. Most of the houses are built to be rented out as only 23% of them is inhabited by the owner of the apartment. There is a total of 465.000 Gallery houses in the Netherlands and the represent around 7% of the total housing stock (Agentschap NL, 2011).

Tenement apartments

A tenement apartment is a house that is part of a building which looks much like an apartment building. But rather than having a hallway connecting the houses, the front door of the houses is directly connected to a stair. At the time of the study around 14% of these houses were owned by the inhabitants. There are 847.000 of these houses which represents around 12% of the building stock (Agentschap NL, 2011).

Other flat apartments

The left-over flat apartments are a category where the apartments are reachable through a central hall of corridor. Around 26% of their apartments are owned by their residents with a steep increase in owner ship between 1992-2005 (47%). There are currently 485.000 apartment houses in the Netherlands and the represent about 7% of the housing stock (Agentschap NL, 2011).

2.1.2. Energy performance

To judge buildings on their energy performance energy labels are used, A being an energy efficient house and G being the least efficient. Having given a general overview of the building typologies. This paragraph will go into the numbers of the number of houses that were built in a certain time period and their energy labels.

Building stock per time period and their energy label

To further analyze the building stock separations into different groups are needed. The government research has already done so for houses being built in a specific time span that had the same energy regulations. The years by which they are grouped are as follows:

1946-1964: Tenement apartments and row houses have a division between prewar and post war houses.
1965-1974: The first regulations about insulation were made.
1975-1991: Minimal thermal insulation with an Rc of 2,5 m^2K/W and double glazing.
1992-2005: The building code was introduced.

Table 1: Showing the number of houses per typology and building period with their energy label (Agentschap NL, 2011).

	<1945	<1964/1946-1964	1965-1974	1975-1991	1992-2005
Detached		441.000	119.000	221.000	178.000
Energy label		G	F	D	B
Semi-detached		285.000	142.000	224.000	173.000
Energy label		F	E	C	B
Row house	523.000	478.000	606.000	879.000	353.000
Energy label	G	F	E	D	C
Duplex house		226.000	22.000	94.000	40.000
Energy label		G	D	C	B
Gallery house		69.000	174.000	109.000	113.000
Energy label		D	E	C	B
Tenement apartment	256.000	267.000	112.000	142.000	70.000
Energy label	F	E	D	C	B
Other flat aparments		99.000	125.000	125.000	136.000
Energy label		E	E	C	B

Calculation of energy labels

The letters used for the energy labels stand for a range of numbers that are calculated with formula 1. The formula calculates the ratio between the used energy and the floor- and loss area of a building. If energy usage is decreased of floor- or loss area is increased compared to the other the Energy Index number is lowered. The calculated number represents the energy efficiency of the house (Majcen D., 2013).

Table 2: Energy indexes and their corresponding energy labels (source: https://www.rvo.nl/onderwerpen/duurzaam-ondernemen/gebouwen/wetten-en-regels/bestaande-bouw/energie-index)

Label	A	B	C	D	E	F	G
EI	≤1.20	1.21 – 1.40	1.41 – 1.80	1.81 – 2.10	2.11 – 2.40	2.41 – 2.70	≥2.70

To see where the biggest impact of renovation will be it is important to compare all different typologies. In order to give an indication of the impact of renovating a certain typology from a certain time period the amount of said houses and their energy labels must be combined. In order to do so the energy labels must be converted in to back into their energy index.

$$EI = \frac{Q_{total}}{155 \cdot A_{floor} + 106 \cdot A_{Loss} + 9560}$$

Equation 1: Calculation of the Energy index (Source: (Majcen D., 2013))

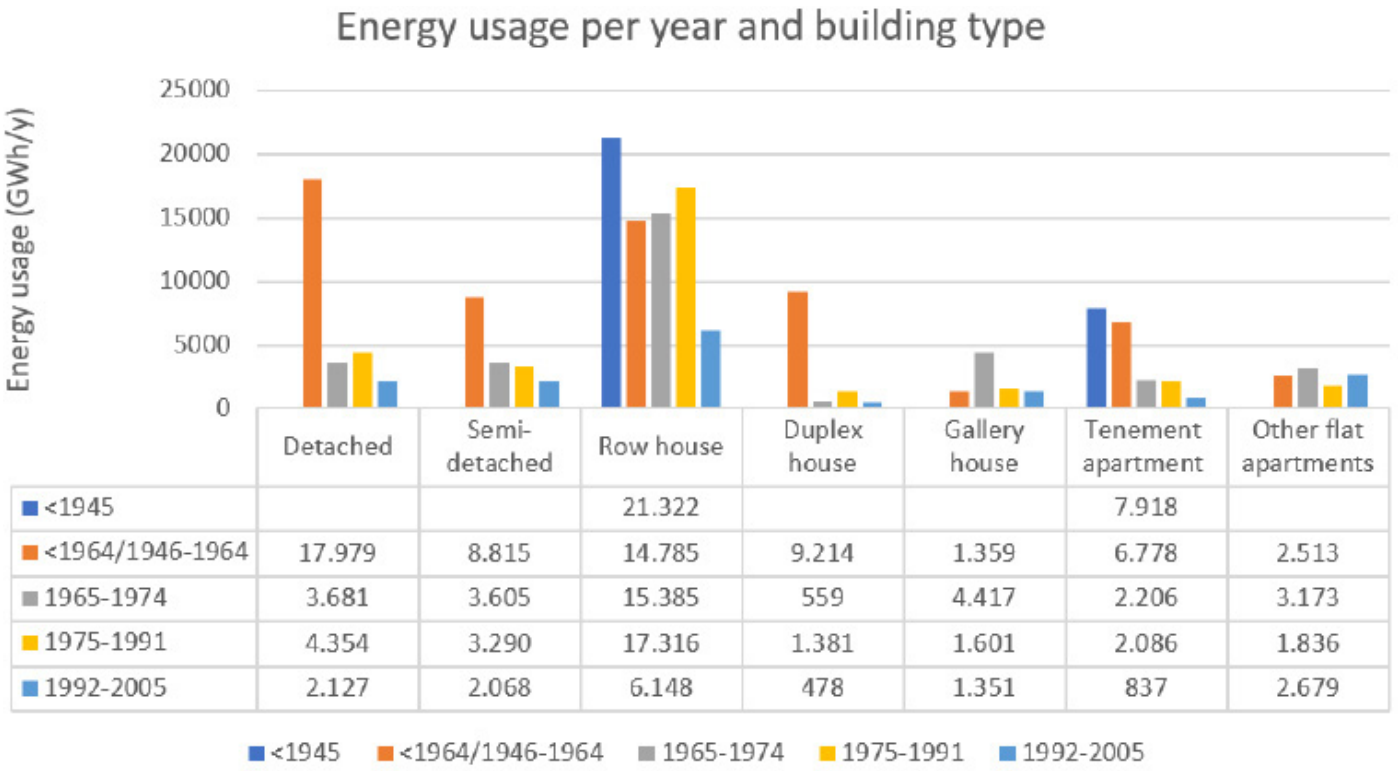
With these energy indexes also comes a mean theoretical primary energy consumption. Combining these with the average size of dwellings, the theoretical primary energy consumption of that part of the building stock can be calculated. The table below shows all the numbers needed for such a calculation. The results of these calculations are seen in table 4.

Table 3: showing the energy index that belongs to the energy labels(Sources: Labels and energy index (Agentschap NL, 2011), primary energy consumption gathered from (Filippidou F., 2017), Average dwelling size source (Majcen D., 2013).

Label	Energy index	Mean theoretical primary energy consumption (kWh/m2 ·y)	Average size of dwelling (m2)
A	0.78	96.8	105.1
B	1.18	132.5	90.2
C	1.46	161.6	90.9
D	1.81	207.8	94.8
E	2.21	265.0	95.8
F	2.66	328.0	94.3
G	2.9	426.9	95.5

Multiplying the energy usage per house by the number of houses the mean energy usage of the complete building stock can be calculated. The graph below shows this grouped per typology and building period.

Table 4: Impact chart showing the impact of typologies of buildings, there quantity and energy label. Information for this chart is taken from table 2 (Agentschap NL, 2011; Filippidou F., 2017; Majcen D., 2013).



	Detached	Semi-detached	Row house	Duplex house	Gallery house	Tenement apartment	Other flat apartments
<1945			21.322			7.918	
<1964/1946-1964	17.979	8.815	14.785	9.214	1.359	6.778	2.513
1965-1974	3.681	3.605	15.385	559	4.417	2.206	3.173
1975-1991	4.354	3.290	17.316	1.381	1.601	2.086	1.836
1992-2005	2.127	2.068	6.148	478	1.351	837	2.679

2.1.2. Conclusion

The purpose of this chapter was to see where the biggest impact for an intervention can be made. As can be seen in the energy usage per year and building type in almost every time period row houses have the biggest impact other than the oldest group of detached houses. Of course, this chart is made using an interpretation of energy labels but using all mean numbers gives a good general overview of the big picture. However, with row houses being 47% of the current building stock it is hard not to see why this group could not have the biggest impact. Literature study has also shown that the least amount of research has been done on pre-war row housing. Combining these facts pre-war row houses have been chosen as a case study building for this research thesis.

2.2. What façade renovation method is being considered?

In this chapter the renovation methods are discussed, as this thesis is focused on the development of the tool it is considering which subjects are important for the tool. First the strategies of renovation will be discussed. Afterwards cases are studied to see what is the best way to renovate the case study building and to see how prefabricated panels are used for renovation. This all to give guidelines for the creation of the parametric method.

2.2.1. Strategies

To renovate buildings there are several strategies in this chapter the different strategies are discussed. To determine these strategies different studies have been consulted (Konstantinou, 2014; Ebbert, 2010).

Replace

Replacement is a strategy in which old building panels are removed and replaced by new panels. In doing so any downsides that came with the old façade are taken away and resolved with new panels. By replacing the panels and not adding a new layer it is possible to increase the buildings energy performance without increased width of the wall. However, by taking out the original façade the space becomes (partly) inhabitable during the construction period. Furthermore, the strategy only works with not load-bearing walls. The whole process will also be more energy consuming as the original façade needs to be taken off and transported away.

Add-in

Adding in is a practice of adding an extra façade elements or insulation on the interior side of the façade. This is a common practice when it comes to buildings with monumental protection, this because it does not have any influence on how the exterior of the building looks. Limitations to this approach are that the insulation line doesn't continue all the way throughout the building's envelope. This results in Thermal bridges where floor slabs and walls connect to the façade. Furthermore, because the insulation is on the interior side condensation could be a problem if it is not the air- and vapor proving is not handled correctly. Lastly there is the fact that adding to the inside of the façade takes away interior space and interferes with the inability of the building during the construction period.

Wrap-it

This strategy suggests adding an insulating layer to the outside of the façade directly, naturally this can only be done if there are no restrictions for the building design. This strategy replaces windows or places additional ones in front of the existing ones and adds insulation and new cladding. This technique can be executed on most buildings that have load bearing facades. However, the limiting factor is the structural capacity of the original façade as mostly this approach is fixed directly on the existing façade. There are different approaches to applying this technique; it can be done on site by attaching insulation panels and plaster over them. Also mounting prefabricated panels can be a way to execute this approach.

Add-on

Add-on shares some similarities with the wrap-it strategy in that it adds to the outside of the building. But rather then replacing the windows and adding insulation onto the existing façade this strategy adds an extra layer all around the building. This could be a secondary skin to form a double-skin façade. And thus, avoiding wind and rain loads on the original façade. It also could help with the natural ventilation of the building.

Cover-it

Similar to the add-on approach the cover it approach adds a structure to the building but rather then adding it in front of the façade this strategy is about covering building courtyards and using it as interior space. It also has the benefit that it takes away wind and rain load from the façade. Naturally to use this approach the building is required to have a courtyard that is allowed to be covered. Also, this approach is not a solution for all sides of the building.

This strategy is an upgrade to a building or group of buildings in which they are connected via a roof that coffers outdoor space. The benefit of this method is that it creates extra indoor space, and the old exterior facades are now in between two heated spaces so the heat loss is reduced. A limitation to this strategy is that not all buildings can use this method and it still requires additional measures for the other parts of the facades.

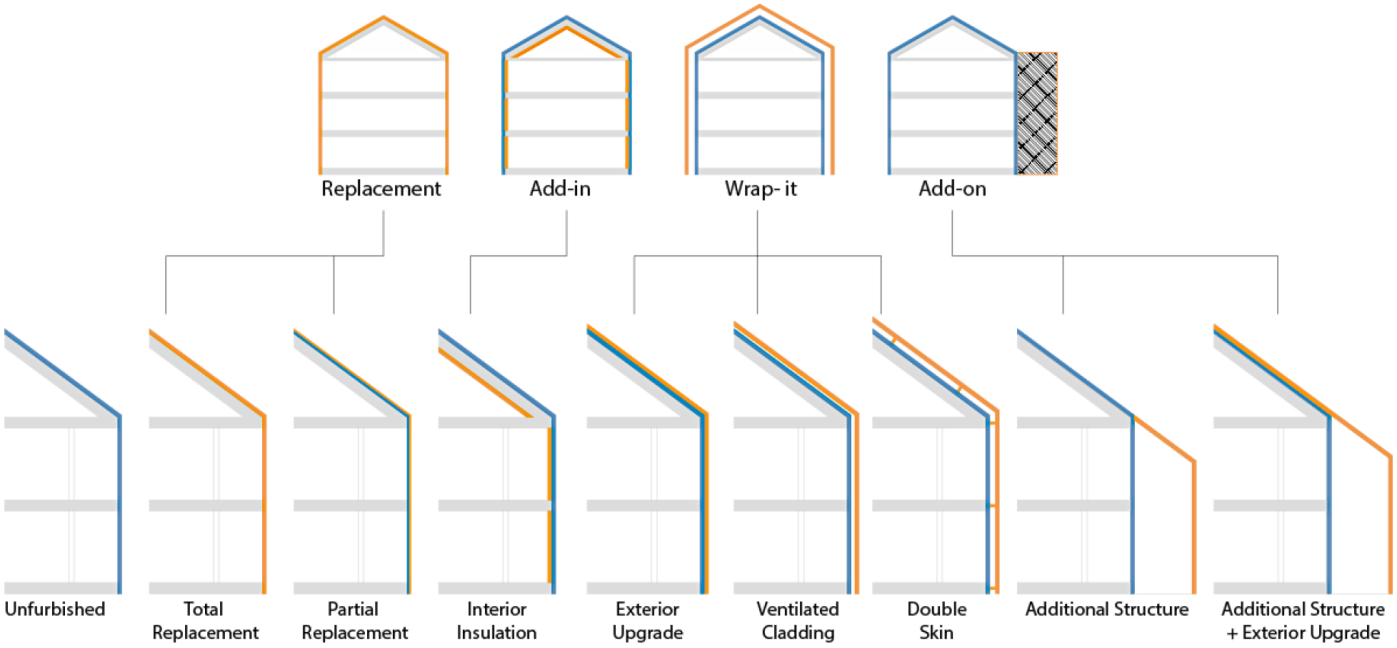


Image 1: Possible refurbishment strategies for building envelope based on (Konstantinou, 2014) (source of image: (Balkuv, 2017))

2.2.2. Case studies

Having looked into the different strategies, in this chapter case studies will be analyzed to study the way in which the renovation has been done. First the optimal way of renovating a building with the row house typology is studied to see what measures are taken in the façade for the renovation. After this a renovation with prefabricated elements is researched.

2.2.2.1. Zero energy refurbishment of pre-war houses.

This case study is based on a research paper from Faik Nebil Balkuv who researched the best way to renovate a cluster of row houses situated in Haarlem that was built before 1945 with an energy index of 2,16 (energy label E) (Balkuv, 2017).

He did so by determining what renovation approach would be best and testing it. In the thesis two refurbishment strategies are compared. The first is the most realistic approach in which there are only moderate interventions to the building with most of the retrofitting only being done by adding on the outside. The method is referred to in the thesis as Refurbishment-1 and is based on the Wrap-it refurbishment strategy (Konstantinou, 2014). The second approach is a more extensive refurbishment option with large-scale interventions in this case the exterior wall and the roof are removed and replaced with a better preforming version. This method is referred to as Refurbishment-2 and is based on the Replace refurbishment strategy (Konstantinou, 2014). One of the main differences between the two strategies is the impact on the dwellers. In Refurbishment-1 most of the work is done on the outside, as a result of this the building will be inhabitable during the construction period. While in Refurbishment-2 the construction process is of such an impact that the building cannot be inhabited for a maximum of 10 days.
After having determined the most suited strategy for the refurbishment the thesis looked into 5 case studies in which 60+ houses were renovated from poor energy performance to A++ for most of them. Having learned from those case studies the thesis continues with the adaptation of these to the building cluster in Haarlem. It does so by calculating all options for refurbishment and choosing the best.
The final results chosen for the renovation are shown on the next pages.

Refurbishment-1

As stated, before this refurbishment is the one where the least impact on the dwellers is considered.
For the building envelope different building envelope parts were upgraded to meet the regulations and to bring the dwelling up to a zero-energy building.

Table 5: Shows the Rc-values that are chosen for the façade in refurbishment option 1 (Source: A remake of the table in (Balkuv, 2017).)

Envelope part	Description	Existing		New	
External Wall	Wall cavity is filled with Loose Cellulose filling. Rigid insulation is added on the outside with stone strips on the outside	0,8 m2K/W		4,5 m2K/W	
Ground floor	The ground floor is insulated from the crawlspace with a mineral wool fiber blanket. The crawlspace walls are insulated from both sides to reduce heat leakage through the thermal bridges.	0,32 m2K/W		3,5 m2K/W	
Pitched Roof	Insulation is added from the inside.	1 m2K/W		6 m2K/W	
Dormer roof	No changes	2,53 m2K/W		2,53 m2K/W	
Dormer sides	No changes	2,53 m2K/W		2,53 m2K/W	
Glazing	Triple glazing (Rehau Geneo) on West and Double Glazing (Rekord Basic) on East facing façades.	Single: 6,1 W/ m2K	Double: 2,5 W/ m2K	Rekord: 1 W/ m2K	Rehau: 0,9 W/ m2K
Window frames	Window frames are replaced with PVC framing. The leaking stone is replaced with an insulated steel version.	Wood: 3,6 W/ m2K	UPVC: 3,4 W/ m2K	Rekord: 1,1 W/ m2K	Rehau: 0,78 W/ m2K
Doors	Doors are replaced with better insulated alternatives	3,5 W/m2K		0,75 W/m2K	

Refurbishment-2

This refurbishment focusses less on the inability of the house but more on the energy performance of the house in the end. For the building envelope different building envelope parts were upgraded to meet the regulations and to bring the dwelling up to a zero-energy building.

Table 6: Shows the Rc-values that are chosen for the façade in refurbishment option 2 (Source: A remake of the table in (Balkuv, 2017).)

Envelope part	Description	Existing		New	
External Wall	Outer brick wall is removed and replaced with prefabricated insulated panel.	0,8 m2K/W		4,5 m2K/W	
Ground floor	The ground floor is insulated from the crawlspace with a mineral wool fiber blanket. The crawlspace walls are insulated from both sides to reduce heat leakage through the thermal bridges.	0,32 m2K/W		3,5 m2K/W	
Pitched Roof	Roof is totally removed and replaced with prefabricated insulated panel	1 m2K/W		6 m2K/W	
Dormer roof		2,53 m2K/W		6 m2K/W	
Dormer sides		2,53 m2K/W		4,5 m2K/W	
Glazing	Triple glazing (Rehau Geneo) on West and Double Glazing (Rekord Basic) on East facing façades.	Single: 6,1 W/ m2K	Double: 2,5 W/ m2K	Rekord: 1 W/ m2K	Rehau: 0,9 W/ m2K
Window frames	Window frames are replaced with PVC framing. The leaking stone is replaced with an insulated steel version.	Wood: 3,6 W/ m2K	UPVC: 3,4 W/ m2K	Rekord: 1,1 W/ m2K	Rehau: 0,78 W/ m2K
Doors	Doors are replaced with better insulated alternatives	3,5 W/m2K		0,75 W/m2K	

Services

To both of these refurbishments have almost the same installations were added.

For heating (and cooling) a Water-to-Water heat pump with low temperature source and low temperature (radiators in refurbishment-1)/ (floor heating in refurbishment-2). For domestic hot water the can, be boosted. For both there is a back-up electrical system that can support the heat pumps on very cold days.

For the Ventilation refurbishment-1 uses natural inlet and mechanical extraction. But refurbishment-2 uses mechanical supply and extraction is used with heat recovery. During the summer natural ventilation can be used and the heat recovery is bypassed. Lastly for energy generation there are PV-panels 18,4 m2 on the east- and 6,7 m2 west facing roof.

2.2.2.2. Prefabricated façade renovation – 2ND skin

In this paragraph a façade renovation approach called 2nd skin is studied. Second skin is a modular renovation system where different prefabricated façade elements are used to update a builds energy efficiency. This can be done up to different levels up to zero energy (Nul op de meter) depending on the wishes of the owner. The design vision that sets 2nd skin apart from the rest is the integration of services into the façade while keeping panels lightweight so they don't need additional foundation support. The prefabrication of the panels allows the project not only to integrate the services into the facades effectively but also limits the disturbance for the occupants (Silvester S., 2016).

2nd skin provides certain packages to give an indication of the intervention needed to reach a certain goal. Those packages are shown below.

Table 7: The different Packages that can be chosen for the second skin approach of renovating existing buildings. (Source: (2ND skin, 2020))

Packages	Premium	Hybrid	All Electric	Zero energy
Façade RC	4,5	4,5	6,0	6,0
Roof RC	6,0	6,0	6,0	6,0
Floor RC	3,5	3,5	4,5	4,5
Glazing type	HR++	HR++	HR+++	HR+++
Ventilation system	Ventilation rosters above glass with mechanical CO2 driven extraction.	Ventilation rosters above glass with mechanical CO2 driven extraction.	Ventilation pipes over façade. Heat recovery system.	Ventilation pipes over façade. Heat recovery system.
Heating installation	Existing boiler Existing radiators	Heat pump 3kW Existing boiler Existing radiators	Heat pump 6kW Existing radiators	Ground source heat pump
Solar panels	-	8 PVT panels	16 PVT panels	15 PV panels
Disturbance for occupants	Circa 3 days	Circa 5 days	Circa 10 days	Circa 10 days
Energy Index	1.00 (Label A)	0.67 (Label A+)	0.08 (Label A++)	0 (Nul op de meter)

As explained in the energy performance paragraph of chapter 2.1. these energy indexes have corresponding energy labels (Shown in table 2).
The packages used by 2ND skin are arbitrary as they are based on some choices made by the company to showcase the impact of certain renovation measures. However, the four levels correspond with levels of renovation that measures people might want to take.

The first level is Premium, in this level only the building envelope is updated to a better insulation value. In the second level a heat pump and PVT panels are added to this so decrease the energy needed for heating of the house and a small portion of electrical energy generation.
In the third level this is taken a step further by insulating the house better and placing a better heat pump with more panels. This allows the house to be all electric, meaning the house is no longer connected to the natural gas network. The last level is Zero energy. There are different definitions of what it means to be zero energy. However, since the energy index is revered to as "Nul op de meter" it means that the building's net energy consumption is reduced to zero (rvo.nl). Meaning that over the course of a year a house does generate at least as much energy as it consumes.

2.2.3. Prefabricated façade renovation

In this paragraph the process of renovating a façade through prefabricated façade panels is studied. Prefabricated façade renovation is done through the means of using panels. For this there are a lot of different options including: concrete panels with rigid insulation and a stone strip or concrete finish (Mjörnell, 2016) or Wooden framing with insulation infill and vapor barriers are most commonly used (Pihelo P., 2017).
In this chapter first the focus points that are important in every renovation (with prefabricated panels) are discussed by the use of literature. Then the design rules for prefabricated panels will be analyzed using drawing from companies that practice the engineering and fabrication of such panels. Then in chapter 2.3. an over view is given of the parameters that can be input for the parametric method.

2.2.3.1. Key points for façade renovation using prefabricated panels.

Several different papers (Mjörnell, 2016; Pihelo P., 2017; Colinart T., 2019; Aldanondo M., 2014) have shown similar critical points when it comes to renovating of existing buildings using prefabricated elements. These points are discussed below;

Energy performance and moisture

When renovating a façade of course the energy performance of the element (opaque parts and the windows, doors and other openings) is of great importance, next to that there is the airtightness of the exterior wall. In order to prevent mold from growing in the construction it is important to have a waterproof exterior barrier that does allow the damp that is trapped inside the element to exit. Also, the damp proofing of the interior wall needs to be assessed and addressed. If there are seams or cracks in the wall the damp that is generated inside can flow into the construction which is unwanted (Pihelo P., 2017).

Existing situation

Next to this there is the fact that the existing building has to be taking into account. There will always be some unevenness in the walls due to the manufacturing and use of the construction. To make sure every element fits in the plan de façade needs to be measured or scanned to assess the tolerance. After this a reference plane can be placed. This is an imaginary frame that is offset parallel to the façade to ensure clearance between the elements and the original façade (Aldanondo M., 2014). The space between the element and the façade will be filled with non-rigid wool insulation to ensure that trapped vapor can ventilate. Also, the opening should be assessed because like the evenness of the wall they could be not perfectly square.

Transportation

Furthermore, there is the transportation of the panels. Panels can be made in all different sizes but are constrained by some factors namely: First manufacturing process as the factories have a certain maximum size of panel, they can produce on their production lines. Secondly assembly process as in some cases large panels are hard to place in certain positions or it might not be accessible for a crane. Lastly the transportation because the panels are most commonly transported using lorries they need to fit inside the trailer.

Mounting

And lastly there is the fixing system. Panels can be hung directly from the façade using brackets if the façade has the loadbearing capabilities (Aldanondo M., 2014). Otherwise, there are different solutions to solve this. In the renovation of the row house (Balkuv, 2017) this is solved by adding a concrete block at the bottom of the lowest panel onto the façade. The panels are stacked on top of each other and the load is transferred through them, the panels still need support from the façade for wind and rain load however. Another example given is a metal subframe with its own foundation is added Infront of the façade (Aldanondo M., 2014).

2.2.3.2. Panel & process limitations

The whole process of renovation process limits on the design of the panels. These limits are discussed in this paragraph and will be implemented into the final parametric method. According to previous research (Barco A.D., 2016) an industrialized renovation can be done in six stages. These six being:

1. Information collection.

   Information about the façade is collected, this can be done in different ways such as taking measurements and translating them to drawings of by making 3D scans. This part is important because as-build buildings slightly differ from their original drawings due to manufacturing, assembling and settlement of the buildings.

   This stage will pose the limitations on the size of the panels, location of windows and other openings in the façade and defines the unevenness in the façade which the panels need to adopt to.

2. Semantic enrichment

   The model that has been built in stage one is enriched with important detailing such as load bearing capabilities of specific areas or the Rc-value of the different parts of the façade.

   This stage sets goals for the panels. How much insulation needs to be in them in order to reach the goal of the energy performance of the building. And where the panels can be mounted and how much weight they can support (this also dictates the weight of the façade panel. As the bigger the panel gets the more weight has to be distributed over the mounts.

3. Envelope configuration

   In this stage the panels are being made according to the limitations that are set in all other stages. It is important that attention is given to the location of windows and doors on the panels. As a panel cannot end with an opening, they always need to be framed. This will pose a challenge in the creation of the parametric tool.

4. Manufacturing

   In the manufacturing stage the designed panels are being constructed. In this stage the limitations posed on the panels are due to the manufacturing equipment. Limiting the panels in their maximum size.

5. Transportation

   The panels have to be transported to the building site this brings its own set of limitations. For example, the size of the lorry that transports the panels. But also, the weight limit or dimension limit of certain roads, bridges or tunnels.

6. Installation

   Lastly the panels are installed on the building. The limitations in this part come from the method of installation. For example, is a crane able to reach the position where the panel needs to be mounted. This might limit the size and weight of the panels.

2.2.3.3. Design rules prefabricated panels

Having discussed the theoretical part of the prefabricated elements this paragraph goes into detail about the practice om designing these panels. In this instance they are timber framing panels. The information is gathered through contact with Mr. W. Lubbers from the company De Groot Vroomshoop who are specialized in the engineering and fabrication of timber framing panels. The drawings that were shared have been analyzed and the design rules that have been discovered are featured in this paragraph. In the first paragraph the literature is researched to see how panels are made generally to validate the information gathered. After that the drawings are analyzed.

Element build layering

Timber framed panels are generally known as lightweight panels as their strength comes from a framework of timber beams with insulation in between. This also makes the panel more efficient in thickness. As prefabricated panels that use concreate for their structural performances need a layer of concrete which would not add to the insulation value of the panel while adding thickness. Windows are also incorporated in the framework to make sure their weight is transferred down properly. And most of the time on the inside a wooden board is added for stability of the panels. This is not necessary when it relies on the stability of another element however.
When using wooden panels, it is important to take moisture into account. This is why mostly there are foils incorporated in the panels that allow moisture to escape when it gets in the panel and another one that does not allow moisture to enter. Lastly on the exterior an aesthetical finish is added, this can be done in different ways. It is always important to make sure there is ventilation in front of the water-repellent foil to make sure any access moisture can ventilate away (Knaack U., 2012).

The description above and the drawings of De Groot Vroomshoop are generally similar. As explained the panels are but up out of different layers every layer has its own function and is explained here. These layers and functions are a general description however. In practice variations from this might occur.

Table 8: General buildup of prefabricated panels (dimensions and layers can vary a lot depending on desired design and performance) (Source: own analyses of drawings)

Layer	General thickness	Function
	Interior	
Interior finish	-	-
Wooden board	12-18 mm	Provides stability if the panel is structurally loaded
Vapor repellent foil	0,2 mm	Makes sure vapor from the inside doesn't enter the panel.
Insulation in timber framework	Varies depending on desired Rc-value. Usually depended on standard timber dimensions.	Provides the required insulation value.
Water repellent foil	0,2 mm	Makes sure water from the exterior cannot enter while vapor from within can leave.
Timber battens	12-24 mm	Enables the ventilation behind the cladding to make sure any access water is ventilated away.
Cladding	12+ mm	Esthetical and shields the frame of direct rain.
	Exterior	

Tolerances

In prefabricated building tolerances are of great importance as the material behaviors, production system and the management of the elements cannot be predicted 100% of the time (Jingmond M., 2010). To ensure all elements fit together and none of them are too big a certain tolerance is considered. In the first image below, it can be seen that between the elements and exterior construction elements a tolerance of 20mm has been used. The second image shows tolerance between two elements that are mounted against each other, in this case the tolerance is roughly 12mm. As these tolerances differ per material, element size and production system (Jingmond M., 2010) it would be wise to make them an input parameter into the tool.

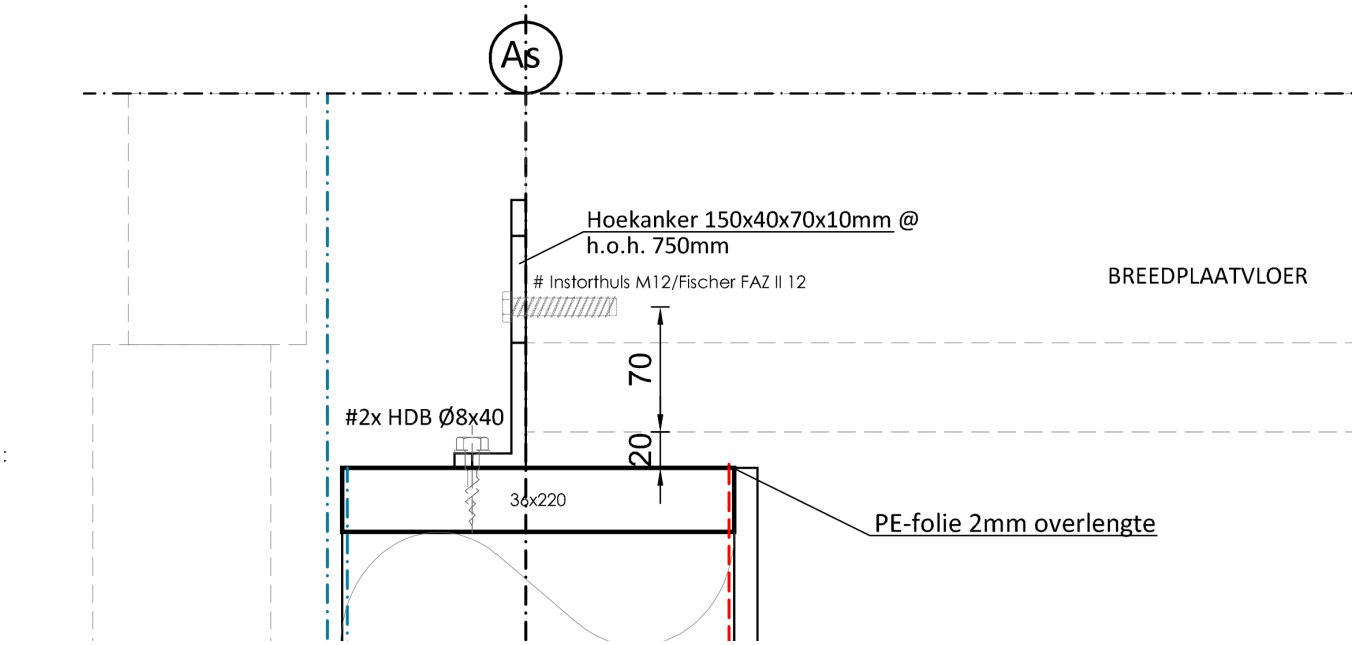


Image 2: Detail showing the mounting of a prefabricated element and the tolerances (source: personal communication with De Groot Vroomshoop)

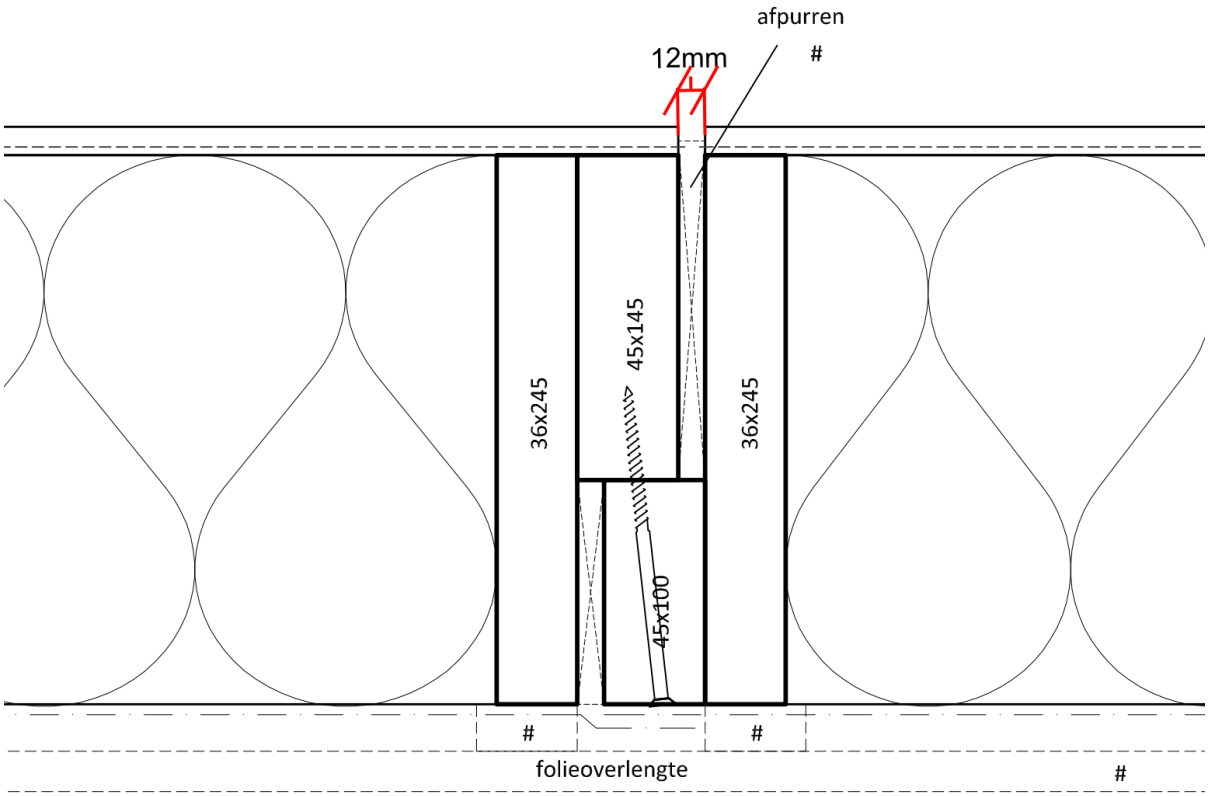


Image 3: Detail showing the connection between two elements and its tolerance (source: personal communication with De Groot Vroomshoop)

Vapor resistant foils

Vapor balance in constructions is very important. If moisture gets trapped in a construction it can start to degrade the elements. This is especially important when wood is being considered as construction material or as wools are considered as insulation. In most buildings there is a vapor pressure from inside to outside due to the temperature difference and the moisture production inside (Künzel H.M., 1996). For this reason, vapor barriers are placed in timber framing elements. They're referenced drawings also show these barriers. They are shown in image 2 and 3. The interior barrier is a vapor proof barrier that does not allow any type of moisture (liquid or vapor) through. While the outside foil is a water proof vapor open barrier. This allows the vapor pressure in the construction to be balanced by the exterior vapor pressure. And thus, making sure no condensation takes place. To make sure that the vapor barrier is closed it is important that there are no seams on the joints. To ensure this generally the foil is continued outside the panel. This way when the panel gets on the building site the foils may the taped closed ensuring a closed vapor barrier.

Wood dimensions and intermediate distance

The dimensions of the wooden profiles are dependent on the milling which differs per supplier. For this reason, the sizes of the profiles should be something that can be input by hand. As for the distance between the profiles. This depends on the manufacturing process. There is always a maximum distance, in some cases this depends on the structural performance of the panel. However, most of the time it is dictated by insulation width. As rolls or blankets of insulation are mostly 60cm wide the space between the wooden profiles is set to 60cm to not waste any material. Nonetheless De Groot Vroomshoop uses a machine to blow insulation into the panels. So, there is no fixed size to ensure this is also possible the distancing should also be changeable.

Windows

Windows will be framed within the timber frame and there are multiple options for fitting them in. One is shown below on image 3 where a separate framework is placed with the window in it. Another way is mounting the window directly on the timber framing like shown in image 4, in this case the timber framing of the window needs to be smaller than the timber of the panel itself in order for the window to fit in the panel. However, the window is applied it is generally not fabricated in the same factory as the prefab elements. Meaning they are also not modelled by the company. Instead, they are provided as models.

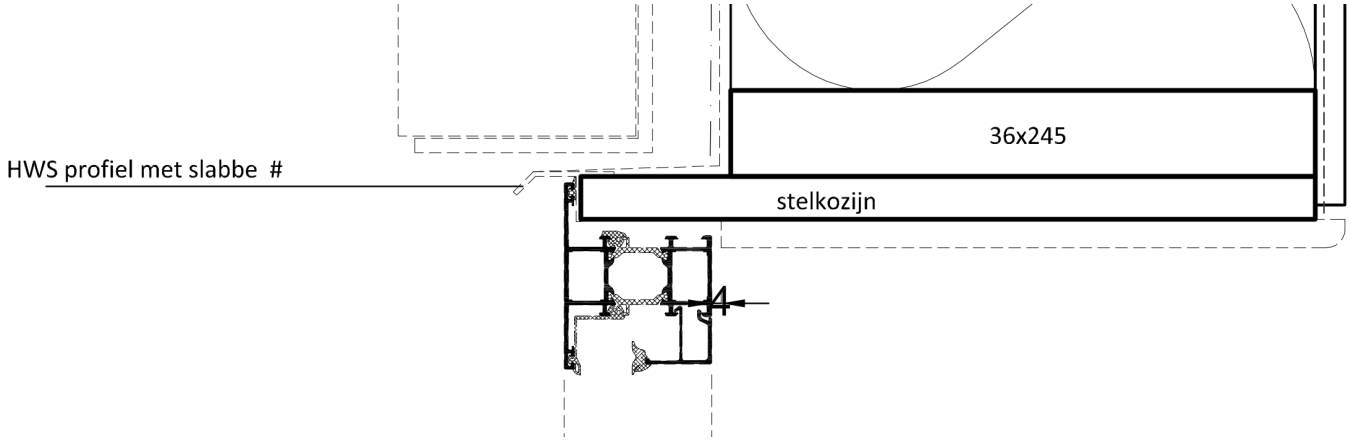


Image 4: Detail showing how a metal/plastic window frame might be fitted in the fade panel by using an additional framework (source: personal communication with De Groot Vroomshoop)

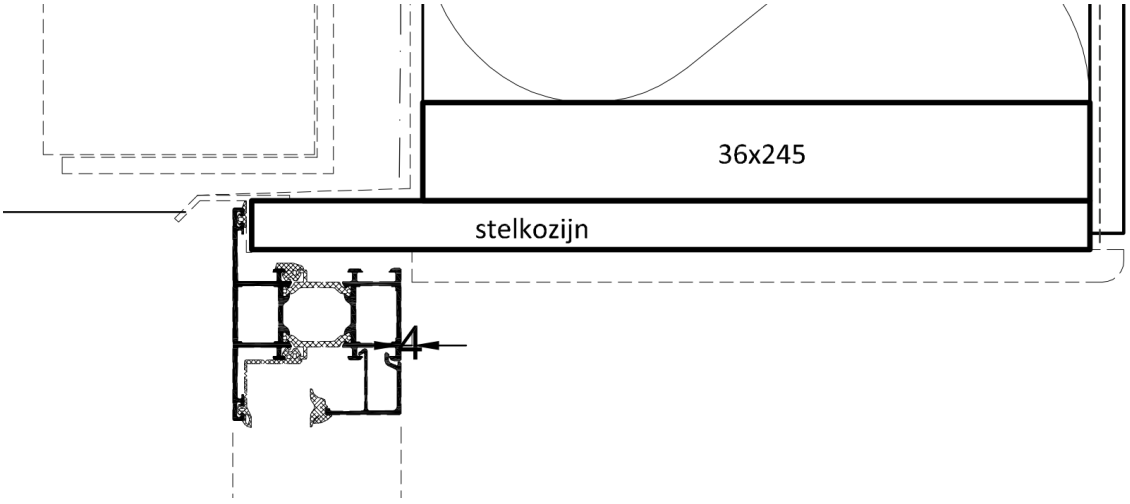


Image 5: Detail showing how a wooden window frame might be fitted in the fade panel by using the existing timber frame of the panel (source: personal communication with De Groot Vroomshoop)

2.2.4. Conclusion

Having investigated the renovation strategies and the case studies it is clear that the general way of renovating the building envelope is through the strategy of wrapping and in some cases adding in. However, since in the case of this thesis prefabricated modules are considered it is most likely that the wrap-it approach the most optimal one as it allows for the use of larger prefabricated panels. The case studies gave some clarity about the preferred façade renovation strategies. In the case of the row-house renovation about what is required in order to achieve zero energy. And the 2ND skin gave some insight in all-in-one renovation through prefabricated panels which is further substantiated by the documentation from the prefabrication firm. How these inputs are included in the tool is further elaborated on in the section about input parameters (2.3.2).

2.3. How is the parametric method designed for the building renovation?

In this chapter the parametric aspect of this thesis is researched. With computational design a design process with the assistance of computers (computer aided design or CAD) intended. Since decades computers are a part of the design process. They help speed up the process of engineering and building. Where conventional design with computers was mostly about modeling or drawing, parametric design is about the relationship between elements. In the conventional design process, an engineer modelled what he thought to be the right solution. If this had to change everything depended on it needed to change as well, all by hand. However, in parametric design the aim is to establish relationships between parts. This does require more time and requires the engineer not to think in solutions but rather in relations. The end result is a model that can be changes by changing one of the parameters (Woodbury, 2010). It is therefore interconnected and a lot of options can be rapidly produced. By doing this the engineer does not have to come up with the single best solution. He can select from many solutions. In the case of this thesis, it is about generating panels and adapting them to the parameters given by the engineer, who can judge the outcome and change the input parameters accordingly.

The general aspects of the parametric tool are discussed in this chapter. First the current methods are discussed after which the input parameters and the expected outcome is described. Finally, the general process of getting from the input to the output is shown.

2.3.1. Current methods

In his paper (Montali J., 2018) Montali writes about the growing tendency to bring knowledge, which is normally used in later stages to the design stage. Which helps the process to achieve the design for manufacturing and assembly which reduces the failure costs of the building process. However, as Montali describes the currently available tools do not address the current design-manufacturability gap in the façade sector. The paper does not refer to renovation. However, its principles reviewed by Montali are the ones also used in this thesis. From a literature has been reviewed to look at the current level of knowledge and to establish the gap that this thesis hopes to bridge. To do so different criteria are included in the search terms. First there is the manufacturing part, this is important as the tool need to design façade panels that can be manufactured. Then there are the parametric design search terms which is the basis of the tool. The building façade term because it is about building facades. And lastly there are the existing constrains, because in renovation there are existing constrains that need to be taken into account. The keywords that were used to search for literature are listed below.

Table 9: Key words used to search for literature. (source: self-made)

Manufacturing	Parametric design	Building façade	Existing constrains
Design for manufacturing Dfm*	Parametric design Paramet*	Façade* Enveloppe	Existing Constraints Renovation

The search query gave a total result of 51 papers. All papers were screened using the title, abstract and key words. The papers left are listed below and have fully been reviewed a summary is written about their findings on the next pages. After then the gap in the knowledge which is necessary for this thesis is pointed out.

Table 10: Resulting literature of the search query that is related to this thesis. (source: self-made)

Reference	Author	Title	Year
(Tan T. M. G., 2019)	T. Tan et al.	BIM-enabled Design for Manufacture and Assembly	2019
(Sun Y., 2020)	Y. Sun et al.	Constraints Hindering the Development of High-Rise Modular Buildings	2020
(Tan T. L. W., 2020)	T. Tan et al.	Construction-Oriented Design for Manufacture and Assembly Guidelines	2020
(Chen K., 2018)	K. Chen et al.	Design for Manufacture and Assembly Oriented Design Approach to a Curtain Wall System: A Case Study of a Commercial Building in Wuhan, China	2018
(Holzer D., 2007)	D. Holzer et al.	Parametric Design and Structural Optimization for Early Design Exploration	2007
(Austern G., 2018)	G. Austern et al.	Rationalization methods in computer aided fabrication: A critical review	2018

BIM-enabled design for manufacture and assembly

In the paper Tan et al. (Tan T. M. G., 2019) Has reviewed several papers in order to determine the gap between BIM-enabled engineering, design for manufacture and design for assembly. “The main purpose of DfMA is to assist designers in optimizing and increasing productivity by integrating downstream knowledge and information into the design stage.” The study mainly focusses on facades but there are some reviewed papers about weather seals and bridges. The conclusion of the paper is that BIM in combination with Design for Manufacturing and Assembly (DfMA) can adopt intelligent technologies. However, in the discussion it is stated that these intelligent technologies in the manufacturing industry have gone through more than twenty years of development. But the corresponding research in the construction industry is still lacking. But with the last developments of construction industrialization an opportunity is provided to fill this gap.

Constraints Hindering the Development of High-Rise Modular Buildings

In his paper Sun et al. (Sun Y., 2020) researches the constrains that are holding back the development of modular high-rise buildings. This is done by a literature survey to determine constrains and by a group of 12 experts in the field that review these constrains. The biggest constrain in this case is the Lack of experience and expertise in the development process of the building. In de discussion Sun refers to the fact that the most experienced engineers know what to take into account in the beginning of the process in order to not form a problem during the manufacturing stage. This experience is what is lacking with engineers that just start or architectural firms who do not have this type of expertise in house. It is therefore mentioned that DfMA is an important aspect to fix this experience gap. By taking decisions that are made in the end upstream and projecting them at the end result better decisions can be made.

Construction-Oriented Design for Manufacture and Assembly Guidelines

In this paper Tan et al. (Tan T. L. W., 2020) compares DfMA of the manufacturing industry where it has been used for a reasonable period to the DfMA that can be used in the building industry where it is still infancy. The paper points out that the industries are different and thus the guidelines for the one might need to adapt in order for it to work for the other. The paper concludes with a set of five guide lines for construction-oriented DfMA. First, the DfMA must consider context-based design as the project is attacked to land and its, physical, natural and cultural context. Second, building technology-rationalized DfMA should help in de consideration of the technology used by the means of their availability and efficiency. Third, logistics and supply chain management are rarely considered in manufacturing but is of great importance in the construction industry, therefore it is important that the logistics-optimized design principle is considered in the DfMA. Fourth, the level of prefabrication and integration of parts greatly differs. Therefore, DfMA must consider component-integrated designs. Fifth, Materials are related to all the presented guiding principles. The use of lightweight structural efficient materials is a principle that should be included in the DfMA. The study shows that these guidelines can operate both individually or collectively.

Design for Manufacture and Assembly Oriented Design Approach to a Curtain Wall System: A Case Study of a Commercial Building in Wuhan, China

In the paper Chen et al. (Chen K., 2018) describes that studies reporting DfMA-oriented design approaches to curtain wall systems (CWS) are extremely rare. His paper reports a case study of a successfully applied DfMA-orientated design with a CWS. But as described might be held back by the fact that no parametric design was used. Meaning that any re-engineering forced the engineers to redo their drawings, calculations and reports. This is then also the sole discussion point proposed in the paper. As the rest states that DfMA helped decrease the material costs (nearly 40% in this particular case), improved the CWS quality and decreased the pollution and waste materials during the manufacturing process. the paper shows the potential DfMA might have.

Parametric Design and Structural Optimization for Early Design Exploration

The article written by Holzer et al. (Holzer D., 2007). Focuses on the question how engineering and architectural expertise can be assisted by a process of digital engineering. The article describes the process of the creation of a roof shell. The roof is created using parametric models. The paper concludes that the implementation of this method and the type of parameters chosen are dependent on the progress of the project according to the design stages. Meaning that the parameters differ greatly depending on the chosen design concept but also the process. It goes on to state that the architect immediately gets visual feedback of the structural performance of his design. Because of this he can design with this already in mind rather than having to adapt his ideas and concepts to what is structurally possible later on. Furthermore, the architect and the engineer get a better understanding of each other's working methodology. Lastly it is stated that the project would not have been possible without the iterative experimentation which is the basis of any parametric model.

Rationalization methods in computer aided fabrication: A critical review

In this paper about rationalization of the design using Computer aided fabrication (Austern G., 2018) the writes describe the important roll robots are playing in industrialization. But that they have not really arrived in the building industry yet. In order for robotization and fabrication in general rationalization is of great importance. The paper shows the different stages at which different disciplines tend to rationalize their design. Researchers before the process while designers, and engineers tend to do this during or after the process. According to the author this is due to the fact that academics are typically also in charge of the fabrication and thus have good cause to incorporate the constraints of it early in the project. In typical design-bid-build projects this expertise is added later in the project where changes in order to make it manufacturable are harder to implement (the academics practice DfMA while in typical projects this is not the case). Also, it is explained that rationalization during the process is hard to implement if there is not a tool that helps the experts. Parametric tools such as Grasshopper, Digital Project or Generative Component are named as examples to help the engineer to do so. These tools can help its user to rationalize the design in real time. The paper concludes to recommend future research directed towards development of easy-to-use parametric co-rationalization methods that can operate in real time.

Concluding

The papers that have been reviewed all show the importance of design for manufacturing and assembly. It is also noted that taking these design discissions to earlier in the process is not an easy task if there is no computational or parametric aid for the project team. In his paper Austern et al. described parametric tools that could help achieve this. There is no literature that connects DfMA to renovation however, Tan et al. mentions in their paper Construction-oriented design for manufacture and assembly guidelines that in every case the context is of great importance and should be one of the guidelines for the process. However, since there is no mention of a desired way in which to do so this will be a gap in the knowledge this thesis tries to fill.

2.3.1. Input parameters

In this paragraph the inputs of the tool are discussed. An input is whatever the tool requires from the user to do its job. These inputs are gathered mainly from chapter 2.2. where the renovation and prefabrication requirements are discussed.

Existing facade

The biggest input of the tool will be the limitations set by the current façade. As the panels and their windows and doors need to align with the windows and doors of the existing façade. This existing façade can be input as a model that is enriched with important details like where there should not be a panel (for mounting purposes). Also, the strength of the façade behind the panels needs to be taken in to account in order for the tool to calculate if the panels are not too big and therefore to heavy.

Tolerances

The tolerances used should be specified. Possibly there should be a difference between tolerances between the panels themself and the tolerances between panels and other structures.

Rc-value

The need for renovation starts with the need for better insulation. This should be an input so that the tool can calculate the Rc-value and determine the thickness that is needed for the panel in order to achieve this Rc-value.

Panel limitations

The panel itself has limitations related to its size. A panel cannot be bigger than the façade, it should be able to be manufactured in a given factory. But also, transportation and placement should be considered. In the end it comes down to one maximum panel size. In width and in height.

Layer and Frame dimensions

Because each company might work with a different supplier for their layering and framing materials of choice in this tool its dimensions and characteristics can be chosen as an input.

2.3.3. Tool output

The output of the tool should be a building information model that is ready for manufacturing. The engineer should only have to check it and possibly make minor adjustments to project specific areas that the tool does not produce correctly. The level of detailing should be such that all individual elements are modelled separately (into layers of the panel and its framework) and are thus distinguished for the manufacturing process. This corresponds with the LOD300 (level of detail) a term mostly used in the modeling world for the amount of detailing a model should have. It should also provide the user with the information it needs separate from the manufacturing, for instance the materials that are needed to make the panel. The weight, RC-value and eventual size.

2.3.4. Tool process

As stated in the introduction of chapter 2.2. there is a lack of literature on the topic presented in this thesis. This is why the process described below is a general overview of how the process could be designed. However, through consulting experts in the field, experimenting and research while designing the tool will be solidified and try to bridge the gap in the body of knowledge. In the discussion, conclusion and recommendation of this thesis this will be evaluated.

1. Façade division

In order to make the prefabricated panels the façade needs to be divided in multiple frame (if the panel limitations make it so that one panel is not sufficient to cover the whole façade). Important aspects for this are: the location of the windows/doors in the frames. As they cannot be divided over two frames due to the required strength and stiffness of the panels. Also, tolerances and mounting positions need to be taken into consideration.

2. Penalization of frames

The frames that the façade is divided into are to be penalized. This is done by taking the user input about the layers and building a model of it. If there is a framework desired in the case of a timber frame panel this should be specified by the user. The sizes of the framework and insulation layers can be specified by the user as they are most comely standardized. The panels are being assessed in terms of energy and structural performance as explained below. Possibly they have to be adapted after the results

3. Energy assessment

To make sure the façade achieves its goal of the required heat resistance it needs to be calculated in the parametric method. The thermal resistance of a construction is generally expressed in Rc-values with m^2K/W as unit or for windows and doors it is common to talk about the U-value which is 1/Rc.

The Rc-value of a building component is calculated by dividing the thickness of the material by the Thermal conductivity of the material. For example: $r=d/\lambda=0.006/1.0=0.006$ m^2K/W. In most instances the wall construction is layered. If is concerns a solid layered construction (without air cavities) only conduction has to be taken into account to calculate the Rc value of the panel itself. To calculate the total Rc value of these components the following formula can be used Rc=r_1+r_2+r_3+.. . If certain layers of the construction consist of more than one part the r-value should be divided into percentages of that layer equal to the material. For example, a layer has steel sections (5%) and insulation (95%). That layer would be calculated as r_total=(r_(metal studs)·0,05)+(r_insulation·0,95) (Hagentoft, 2003). (The heat transmission from surface to air (re or ri) is not taken into account in this part as it is already there in the original wall and the value added in the parametric method should be the thermal resistance the user wants to add to the building.)

The heat resistance is calculated using the user input for thickness of the element due to the standard wood sizes. If the calculated RC-value is lower than the desired Rc-value the user is notified.

4. Structural assessment

In order to make sure the panels can be hung from the façade their weight can be divided of the loading capacity of the mounting brackets, so that the engineer knows how much brackets are needed to carry the panel and can decide it this is desired or not.

5. Information user

The final step is to inform the user of what has been done. If something has gone wrong the user should be notified and told what they can change to make the tool work right. But if everything goes right the user should be informed with the final shape of the panels and their specifications. For example, the Rc-value or weight of every panel and materials used might be something the user is interested in.

2.3.5. Overview of tool

Having determined the input, output and the general process steps the layout of the tool might look something like the image below.

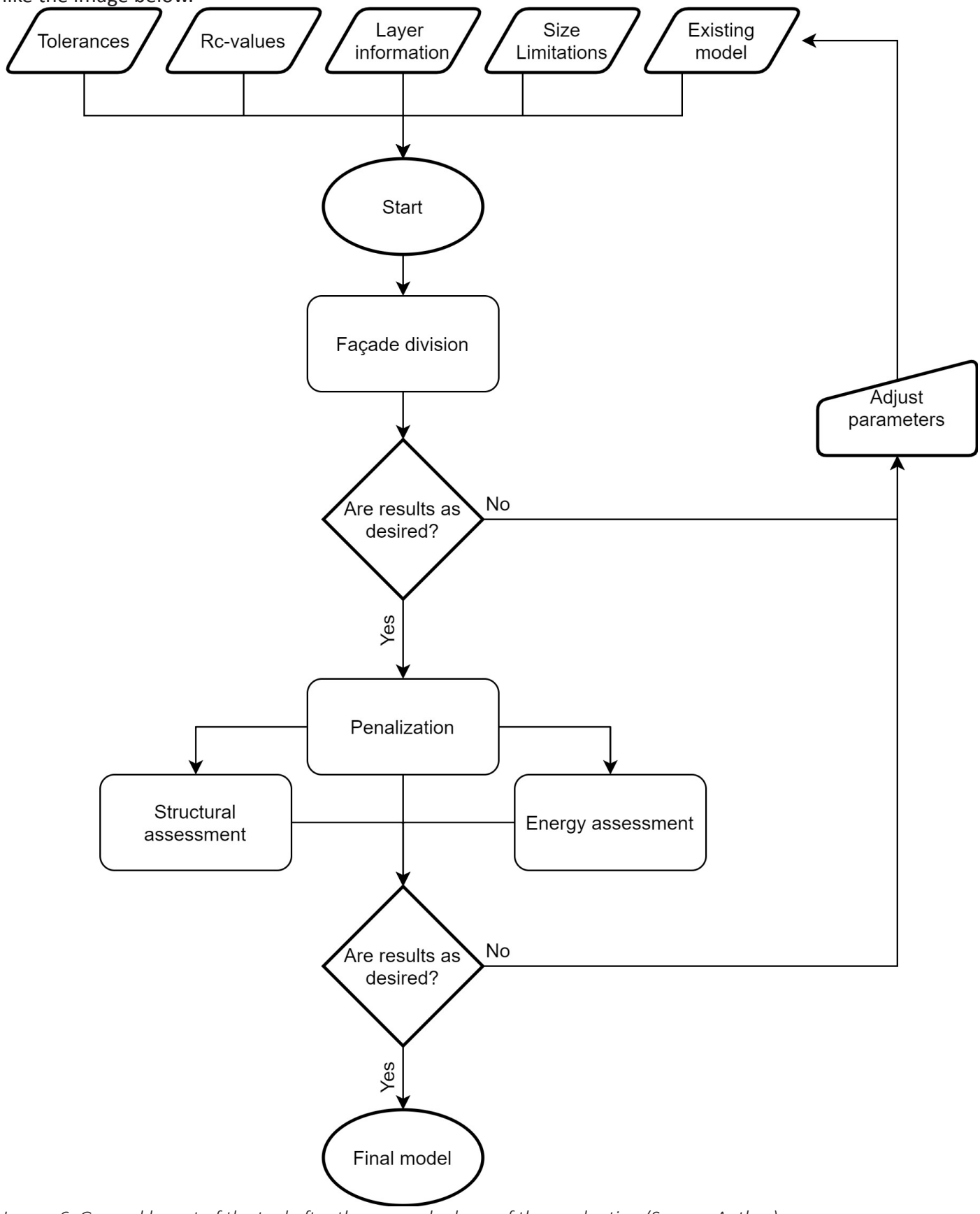


Image 6: General layout of the tool after the research phase of the graduation (Source: Author)

3. Design phase

With the research phase of the thesi
research question: How can a param
order to provide designers and engi
this question, first the design of the p
designed elements. To test this meth
compared with the original design. L

is completed the aim of the design phase is to take the research as a foundation and provide an answer to the main
netric method be designed and what parameters are used for the renovation of poor energy preforming buildings in
neers with a building information model of prefabricated adjustable building's envelope panels? In order to answer
panel has to be determined. After which this design is taken and a parametric method will be created to generate the
nod, it will be applied on a case study building in order to generate renovation panels. Which are then assessed and
Lessons learned from this are implemented in the tool if it did not function properly.

3.1. Design of the panel

In order to make a good approach for the design of the refurbishment and the automatization of this process with the use of computational design. This paragraph will analyze the most common building envelope characteristics that are to be found in a rowhouse to determine if it is feasible to preform them with prefabricated elements and/or parametric design. After which the elements will be discussed in detail.

3.1.1. Characteristics of facades

In order to make a good approach for the design of the refurbishment and the automatization of this process with the use of computational design. This paragraph will analyze the most common building envelope characteristics that are to be found in a rowhouse to determine if it is feasible to preform them with prefabricated elements and/or parametric design. After which the elements will be discussed in detail.

Basic elements

In order to enclose any space, there are three basic elements that are always required: a floor, walls and a roof. In the case of this thesis the focus only lies on the elements which can be prefabricated for the renovation of row houses. This limits the possibilities of the basic elements and eliminates the floor as these are generally replaced with non-prefab system or the existing floor is insulated.

Image 7: example of a pre-war rowhouse (source: (Agentschap NL, 2011))

The walls of row houses are very similar in most cases: a straight façade with openings for doors and openings. These elements lend themselves perfectly for renovation with prefabricated elements. The openings can be fitted in the panels in order to already place the windows and doors in the panels.

For the roofs there is a clear distinction between two types: the slanted roof and the flat roof. The slanted roof of most row houses is already constructed out of timber although not in a prefabricated manner. Nowadays however most slanted roofs for houses are constructed with prefabricated timber frame elements. The roof can have different openings like a skylight which can be integrated in the element like a window is integrated in the walls. Another common roof element are dormers, these require an opening in the roof and are put on top of the roof on the building site after the roof itself is fixed in place. (Lubbers, 2021) Next to the slanted roofs there are also the flat roofs. These can also be prefabricated but in practice it is more commonly chosen to apply a complete insulation layer on top of the (existing) construction layer. As this means there are no weak links in the insulation in the form of construction members this type of roof build-up is known as “warm dak” which refers to the construction being on the warm side of the roof (Joostdevree, 2021).

Having discussed the basic elements there are some elements that are not used in every rowhouse but are still commonly found. These elements will be discussed next.

Bay windows

A commonly seen addition to row houses is bay windows. They allow for an extension of the livable space. They consist of separate parts, there is the façade part which is closed and can be renovated and fitted with prefab wall panels easily. But there is also the part with the windows. This part is more different than the windows in the normal façade in the way that the windows are not integrated in the façade. They are their own component as can be seen in the image. This is done to reduce the profile and not have the complete depth of the wall at every corner of the bay window.

Image 8: Row house with a bay-window (source: geschiedenisvanzuidholland.nl)

Balconies

Another commonly addition to rowhouses are balconies. However, at the time it was common practice to not thermally disconnect the balconies from the house its construction. Because of that the whole balcony is a big thermal bridge. Insulating this balcony would solve the thermal issue but the added thickness would mean that if there are windows or doors below ore above, they will have to be adapted in order for them to open properly. But it is more commonly preferred to remove the balcony and replace it with one that is disconnected from the structure like in the 2ND skin project. In any case this is not a part of the project that is normally executed in prefab timber framing and so it will not be included in this thesis.

Image 9: Row house balconies (source: (Agentschap NL, 2011))

Porches

The last characteristic which is sometimes seen in rowhouses is when there is a Porch. This is an area that is outside but that is surrounded by inside areas at the sides and on top. This case is in a way similar to a balcony in that the construction is continued outside the thermal barrier. But with addition of walls on the side. Which makes it tricky to renovate considering the limited space that is available. So also, this characteristic will not be included in the thesis.

Image 10: Row house portal entrance (source: self made)

Conclusion

Having gone through the most frequent building envelope characteristics it is clear that the characteristics that lend themselves for renovation though prefab elements should be the ones that will be designed themselves and later turned into a computational method. These characteristics are: Walls, slanted roofs and bay-windows. In the end all characteristics need to be able to interact with each other. So, the computational method will need to allow its user to make adjustments to allow for these connections.

3.1.2. Wall panels

In order to design a panel, first the design of panels has to be studied. This has partially been done in chapter 3.2.3. – Prefabricated façade renovation in this paragraph that knowledge will be elaborated on further.

3.1.2.1. The theory

The timber frame element is composed of different parts. The image below shows a load-bearing timber frame element. The structural part of these elements generally consists of 2 parts; The framework which provides the form and strength. And the plating, which ensures the stability of the element. The bottom wooden member of the framework is called a Bottom plate (1), the top element a Top plate (3). These are connected by a series of studs (2) which are commonly spaced 400mm to 600mm apart. Depending on the structural load there will be a Lintel above the window (4) to transfer the forces from above the window to the studs next to the window. Which can be double the thickness of normal studs like the image below to ensure they are strong enough.

The gaps between the studs are filled with insulation materials to increase the resistance to thermal conductivity. A foil to ensure vapor is not moving from the interior of the building into the element is placed on the interior. While a foil that ensures any damp that might come into the insulation part can transfer out without water getting in is placed on the exterior side. Lastly the plating is added (5) to ensure the stability of the element. (Woude, 2004)

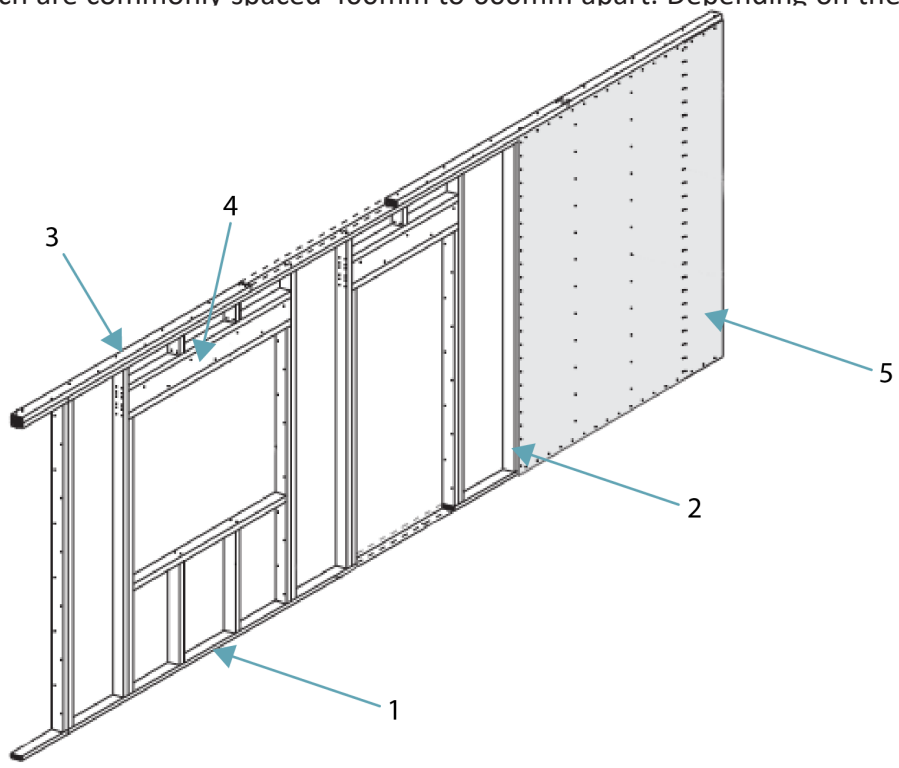


Image 11: Row house portal entrance (source: self made)

As already stated in the research part of this thesis the sizes of the prefab panels are dependent on different parameters. However, most manufacturers use the transportation limitation as a general limit to the size of their panels. As panels with a size larger than that can be transported in a normal truck increases the cost of transport drastically. Especially considering that there are a lot of trucks needed to transport all the panels that are made. The sizes limits of the normal transported prefab panels are a height of 3,6 meter and a length of 6,3 meter. (Raab Karcher, 2021; Gerrits, 2008) in case of houses and especially row houses this means that the complete width of a façade and a floor height can be covered. This also allows for the panels to be properly mounted to the structure of the building. As will be shown in the next paragraph.

Mounting

In order for a timber frame panel to stay in place they need to be fixed using mounting brackets. These brackets can be placed anywhere on the panel where there is access to a supporting structure behind the panel. But generally, they are placed at the edges of a panel. The brackets need to be fixed at a load-bearing part of the wall. Which can be the whole wall in case the wall itself is load-bearing. But it can also be that the interior walls are load-bearing in the other direction in which case the side of those walls and the side of the floor slaps might be used to mount the panels to. In the case study done in paragraph 3.2.3. – prefabricated façade renovation the project's prefab elements are supported around the panel. In the image below these mounts are shown. The bottom mount ensures the panel is supported in all 3 axes. The other mounts are only there to ensure the panel does not fall out of the façade.

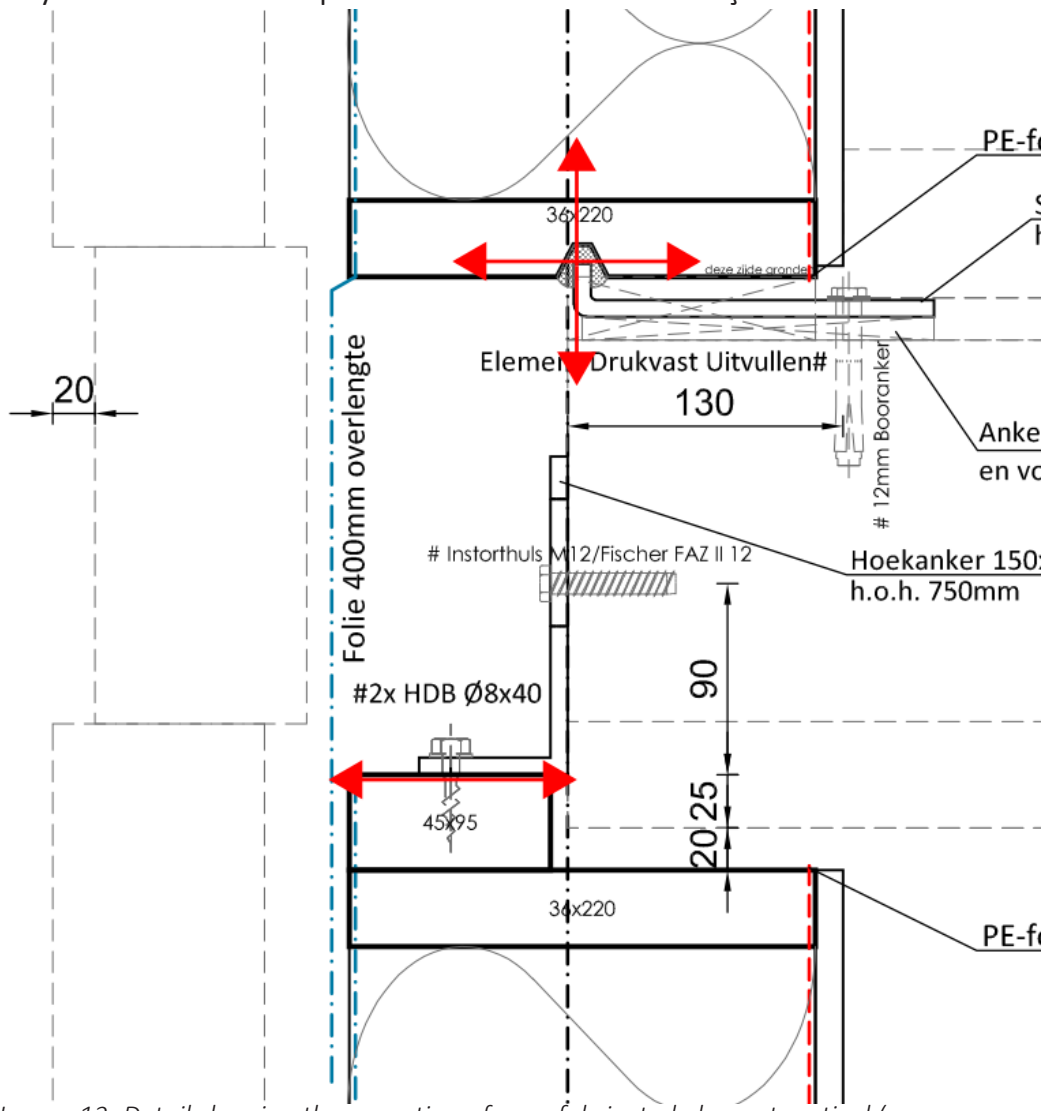


Image 12: Detail showing the mounting of a prefabricated element vertical (source: personal communication with De Groot Vroomshoop)

The side of the panel is generally fixed at the partition wall between two houses. As panels are mostly made to the size of a house width (in case of row houses) and level height (as anything over 4 meters requires a special transport). The space between these panels is generally left open with such a distance that is easy to apply proper insulation to counter any thermal bridge.
In the case of this thesis the mounting will not be as straight forward as is shown in these images, because

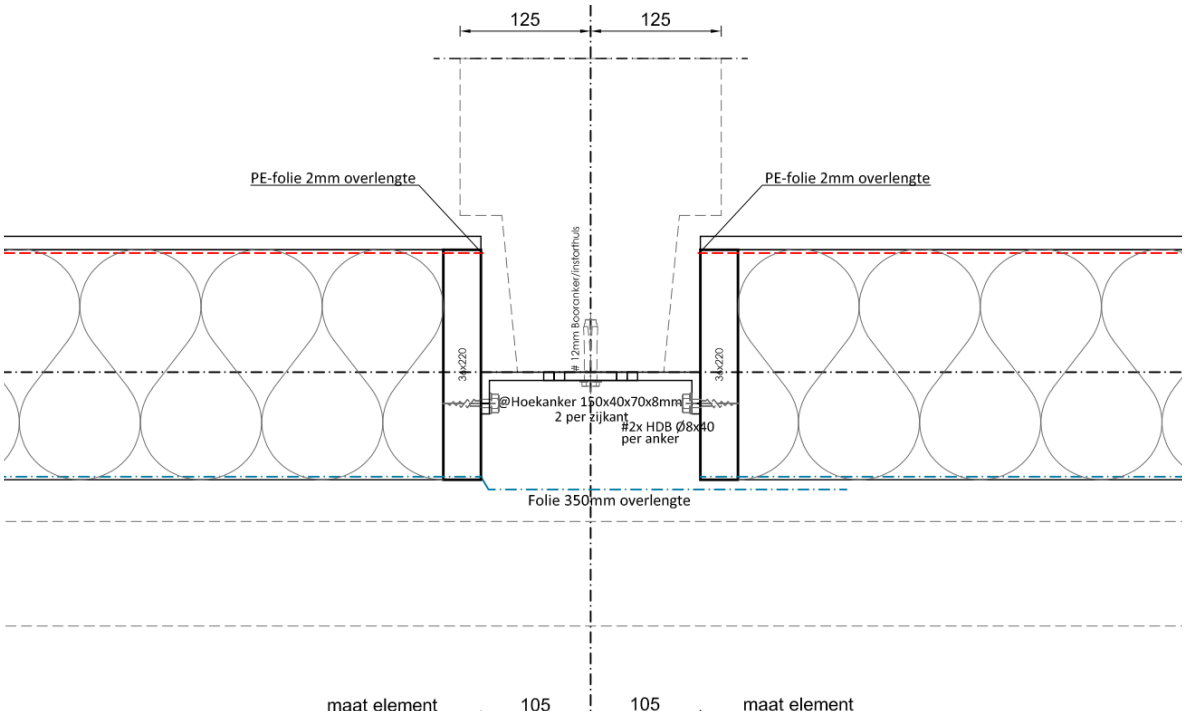


Image 13: Detail showing the mounting of a prefabricated element horizontal (source: personal communication with De Groot Vroomshoop)

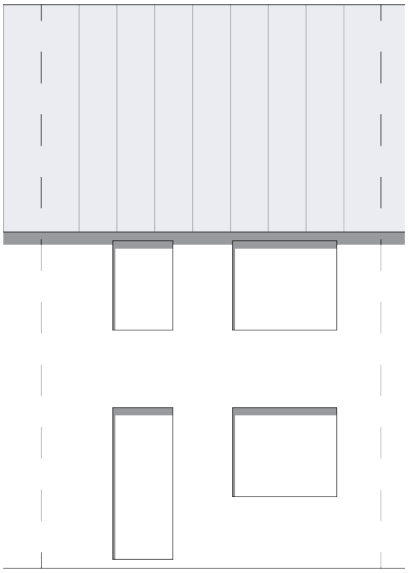
these elements do not need to take into consideration a façade that is already in place and that might not be as straight and level as the panel itself. Furthermore, it also will not be mounted directly onto floor slabs and walls. Because of this there are a couple of possibilities:

1. The panels are mounted directly on the façade as it is strong enough to transfer the forces to the foundation.
2. A construction is made which allows the panels to transfer their vertical loads to the foundation while relying on the façade for the vertical stability.
3. The panels are mounted directly onto the structure of the building through the means of consoles that penetrate the outer layer of the façade.

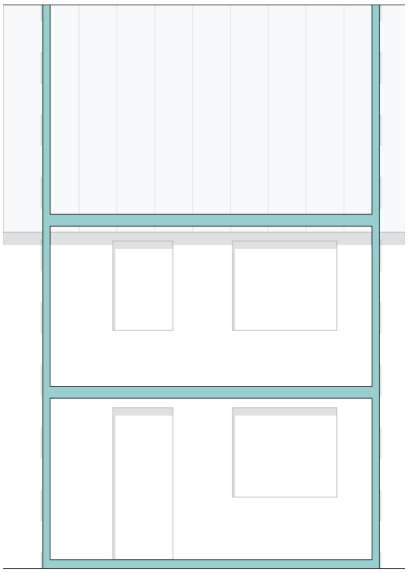
Of course, also combination between these options is possible.

3.1.2.2. The panel design

Now that the wall panels and their characteristics are known in this part, the design is formulated which will be used for the thesis and which is the basis for the parametric tool. This panel design will be a generic design as the parametric tool is a solution not for a single façade layout. Later this design will be tested on a case study building. In the image to the right an illustration is shown of a simple row house façade. It has four openings, every opening needs to be completely surrounded by one panel. This is done so that the window and door frames can already be fitted in the factory, by doing so the concept of prefabrication is optimally applied. This can be a challenge when the façade openings are large and because of that decrease the stability of the panel. However, this is generally not the case with older row houses.



The division of the panels depends on the construction of the building. If the façade itself is strong enough to support the weight of the panels then the penalization of the façade is only limited by the transport size and the best way to divide the façade from a mounting and engineering point of view. However, in most of the cases the loadbearing structure has to be addressed in order to fix the panels. This sets the first sizing limit for the panels as they need to be within these borders in order to properly fix them to the construction and also be able to insulate them in a good manner.



Having determined the panel boundaries, the layout of the panels themselves can be generated. This is similar as the panel layout that is shown in the beginning of the paragraph. The top and bottom plate together with the left- and rightmost stud form the outline of the panel.
The panel is filled with studs that are offset 600mm every time. Next the studs are placed next to the openings and a sill trimmer is added to the bottom of the window with a lintel covering the top. In this case the linter is a normal size timber element as it does not have to support any element resting on top of it. The bottom plate for the door is only removed on the building site to ensure the stability of the panel during transport. However, if it is in the way of the placement of the door it will be removed and the stability has to be ensured by placing another temporarily element.

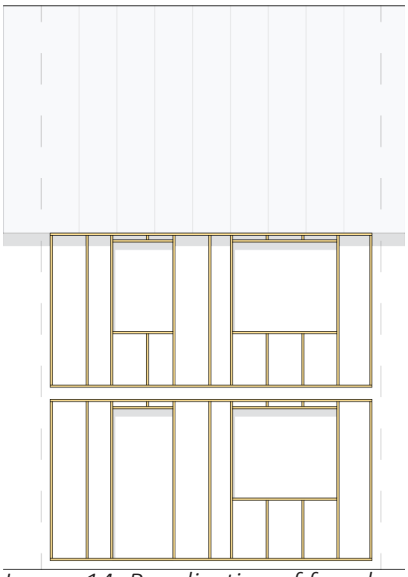


Image 14: Panelization of facade

Placement

The panels need to be mounted on the original façade. But in order to ensure enough clearance to the façade the engineer needs to define at which point the panels are placed. Depending on the mounting method this plane might be further away from the façade but the general consideration in the placement of the plane is to provide a straight plane that is not affected by any irregularities in the original façade. The type of mount used to fix the panels highly depends in the strength of the sub structure. It might be fixed to the interior wall like shown in image 15, to the exterior wall if it is strong enough or it might need a separate structure altogether. These different mounting strategies are however not in the scope of the thesis. The attachment of the panel on said mounting bracket however is part of the thesis. To make sure the seams between panels are as little as possible and to ensure a unitized connection the unitized façade anchors shown below are used as an inspiration. As shown the brackets on top have a double connection while to bottom bracket has a single connection. This will be used in order to mount two elements to the same bracket.

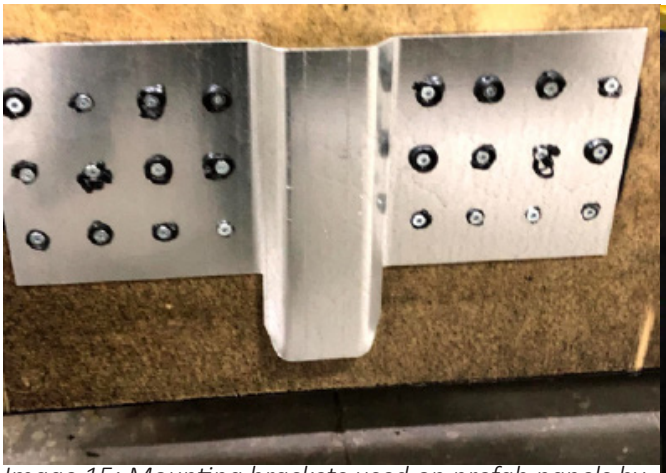


Image 15: Mounting brackets used on prefab panels by RC-panels (Source: R. Nuñez Larios)

The detail below shows the essence of mounting the panels. The seam between the panels has been indented to ensure no water gets behind the panels and also prolongs the thermal bridge the seam forms for the façade. The panels are mounted on the mounting brackets with the use of the sleeves. In order to mount a panel first, the lower brackets are put in place. After which the panel is lower over them allowing the brackets to slide into the sleeves. Then the bracket on top is slid into the sleeve and fastened in doing so the panel is secure. The process is then repeated. At the ground level there is a section of rigid insulation which is water resisted. This prevents standing water from getting in touch with the panel and in doing so damaging it.

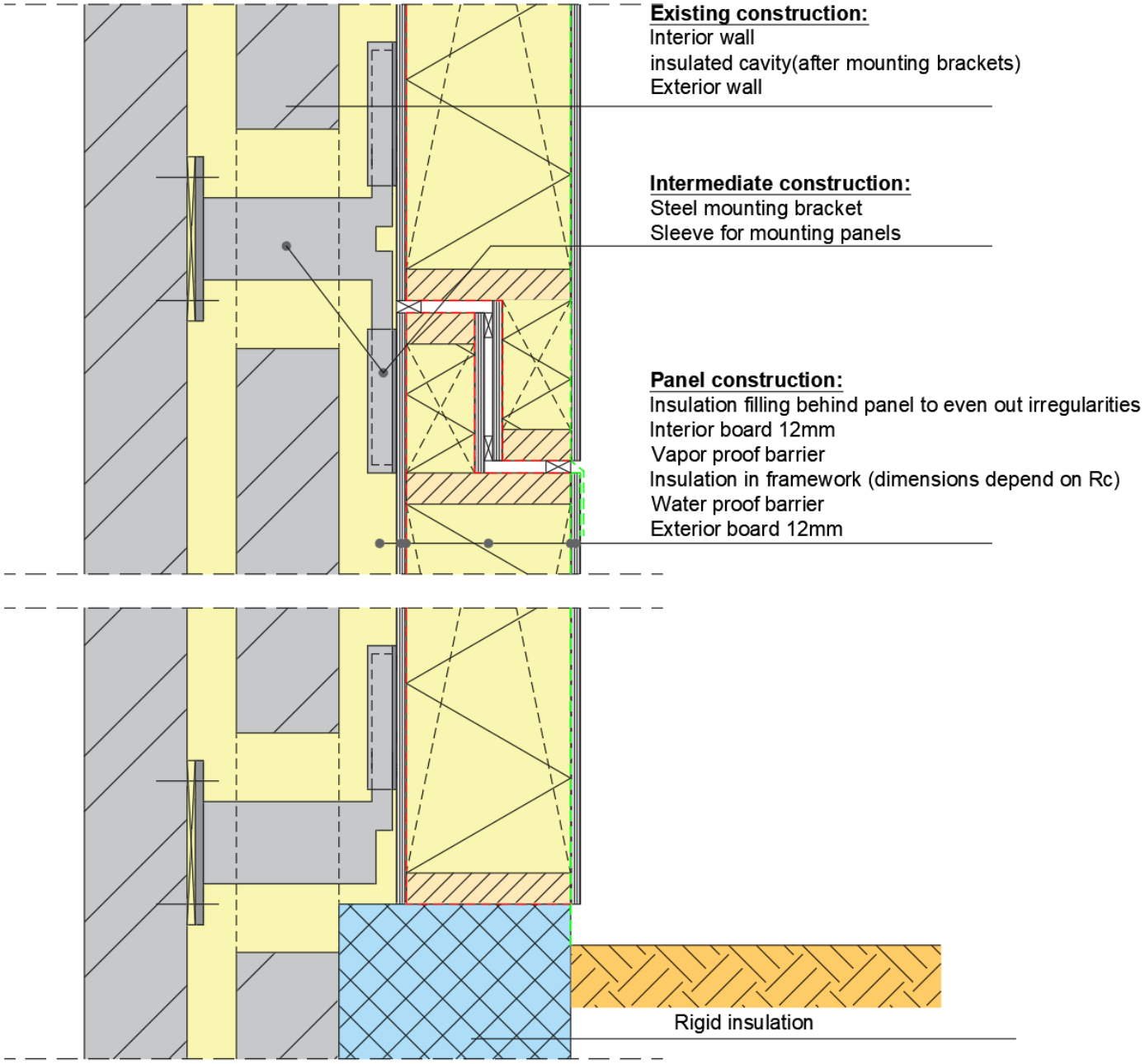


Image 16: Detail of the panel placement and mounting (Source: self-made)

Window openings

The incorporation of windows in panels can be done in different ways depending on the framing system, material, and if the frames are made by the company themselves or by different sub-contractors. As can be seen below the connection is different for a wooden or metal frame. There is however a clear transition from where the panel framing ends and the elements for mounting the window framing begins.

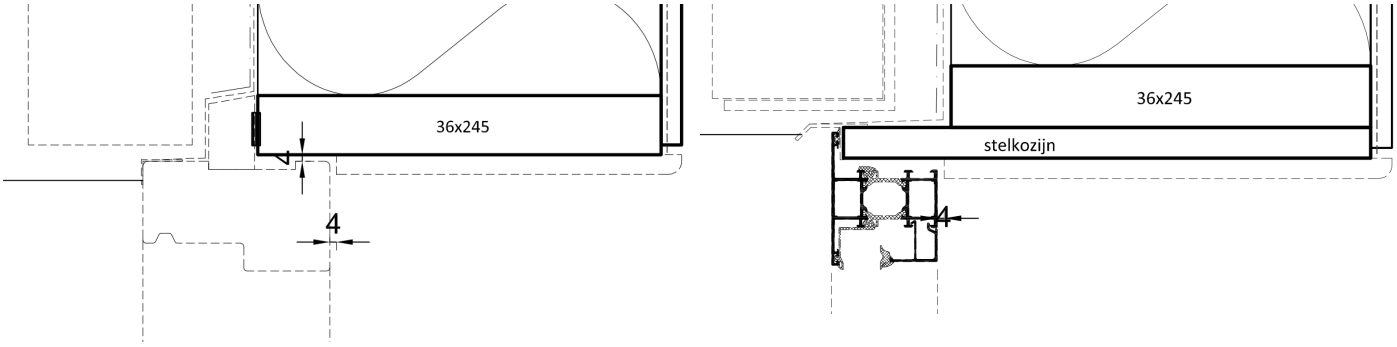


Image 17: Different approaches of windows frames in prefab timber panels (source: personal communication with De Groot Vroomshoop)

This allows for flexibility in the approach of the generation of these infills.

So rather than designing a single window framing and formulate a computational method that places it in all panels. It is preferred that the windows can be an input to the script. That takes the window and places it in the assigned location in the panel. To do so a standard wooden framework for triple glazing will be used to generate the windows. To make sure the finishing looks good on the inside there will need to be some work done on site. The detail below shows how any inequalities in the right-angles of window frames can be solved.

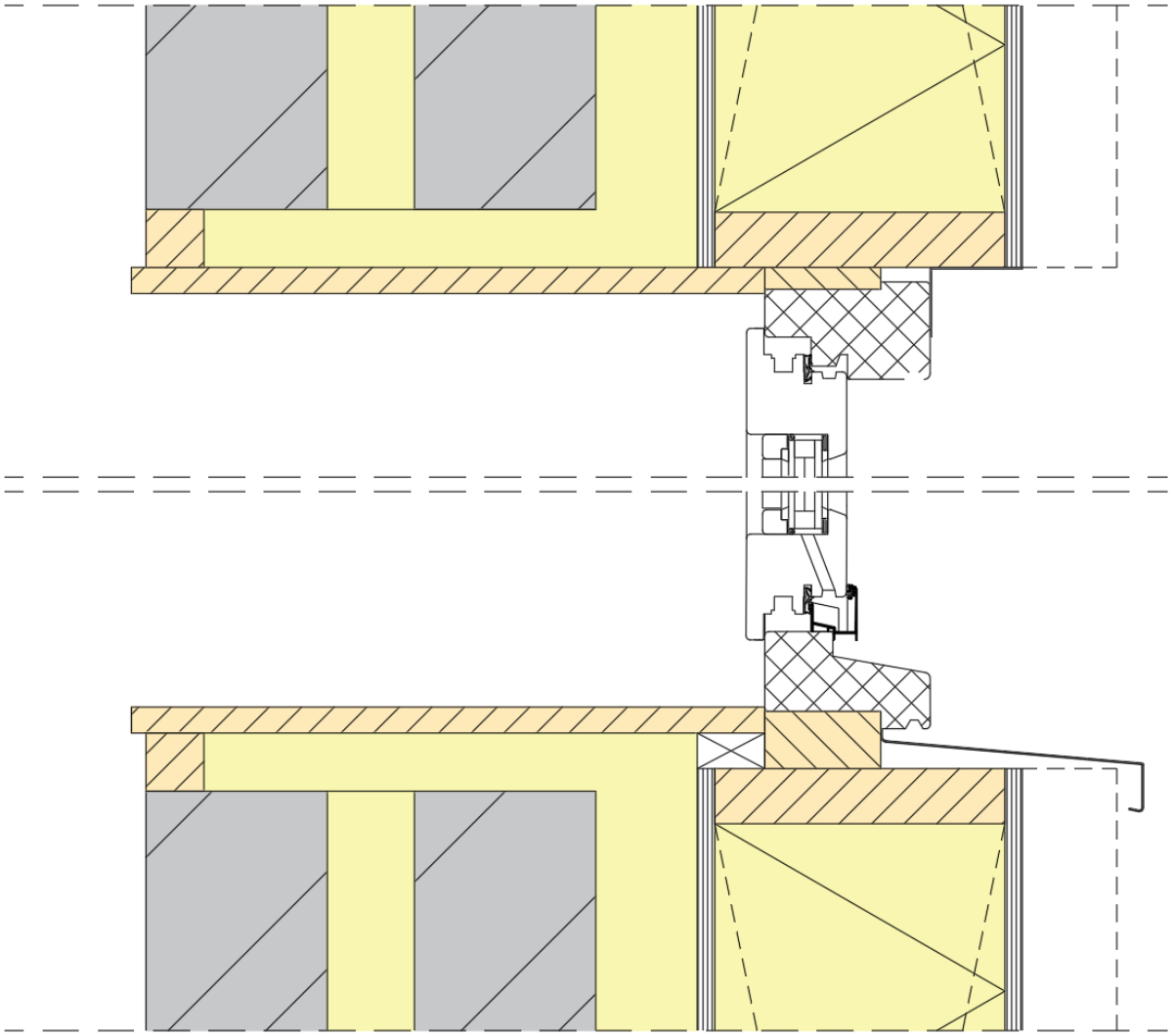


Image 18: Detail showing the integration and interior finishing of window openings (Source: own work)

3.1.3. Roof panels

In older row houses the roof is generally not insulated as can be seen in the detail below. During the renovation the existing roofs can be insulated both from the inside and from the outside. However, with the selected approach the new outside face of the wall will be further offset outward. This means that if the roof is insulated from the inside there is no good connection between the insulation of the wall and the roof, this can lead to cold bridges. For this reason, in this thesis the exterior will be the side that will be insulated with the prefab panels. The rest of this paragraph will go into the theory of how such roof panels can be designed and afterwards this knowledge will be used and further specified to the design for this thesis.

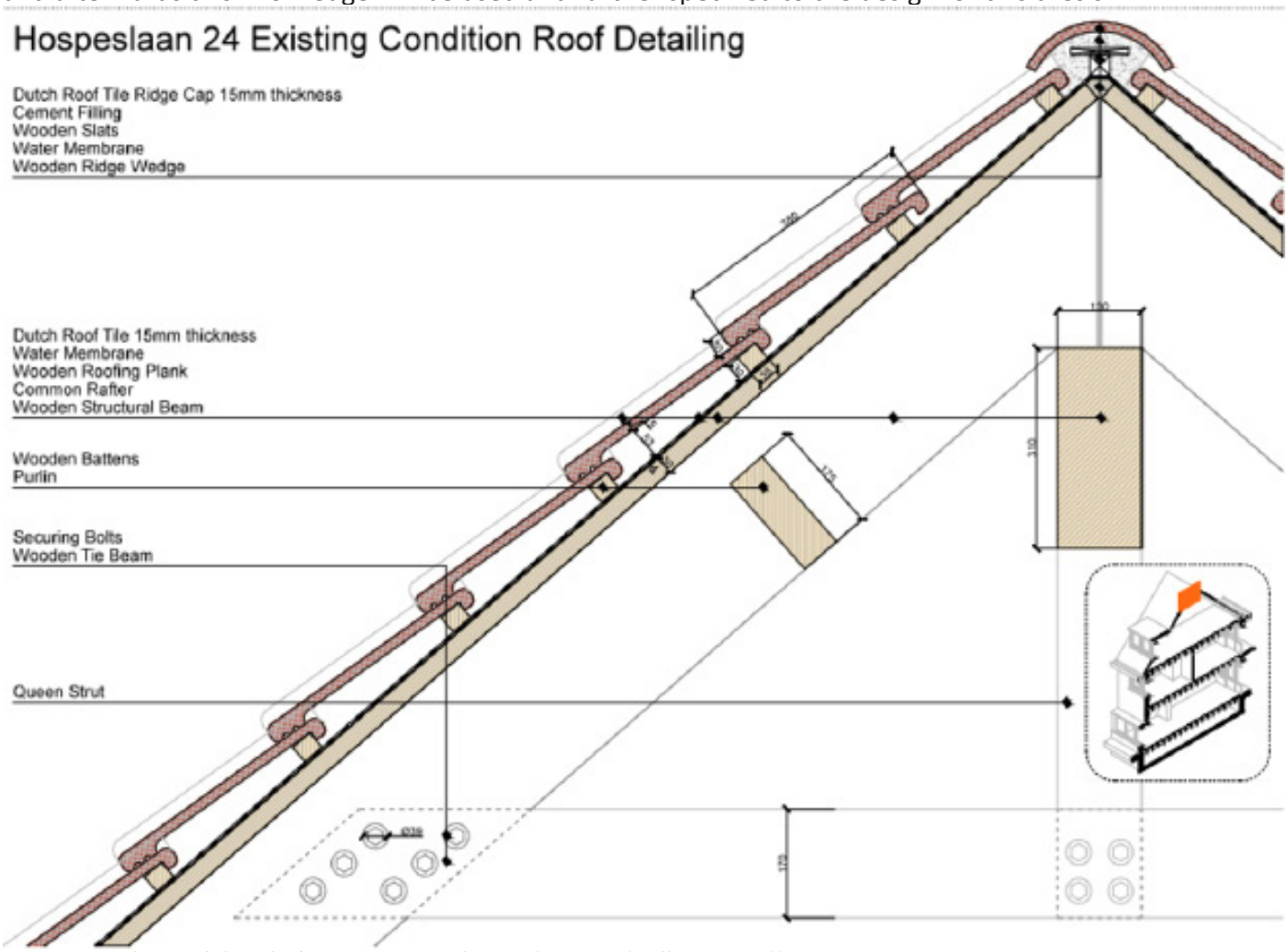


Image 19: The roof detail of a pre-war rowhouse (source: (Balkuv, 2017))

3.1.3.1. The theory

Roofs can be made in a number of different ways, such as a rafter roof or a purlin roof which indicate the direction of the main construction members. Both of these types of construction also have a prefabricated form. The purlin prefab roof commonly spans from one house partition wall to the next. While the rafter version spans from eaves to ridge. Generally, these elements are connected to their counterpart on the other side of the roof by means of a hinge (Woude van der, 2005). In this thesis the hinge roof will be designed further as 8 out of every 10 roofs for new build houses uses this roof (de Leeuw, 2020) and thus the knowledge is already widely known.

Structure

The structure of a hinge roof is similar to the structure of a timber frame wall in terms of its constructional components. The image below shows these different elements: the roof rests on a wall beam (1) that transfers the horizontal forces from the roof into the floor and the vertical forces into the wall. The interior of the panel is generally cladded with timber board for interior finishing. Behind this layer there is a vapor proof barrier similar to the wall panel. The core structure consists out of rafters that are in between the insulation layer of the panel (2). The distance between these studs depends on the weight they need to support and the span from the eaves to the ridge. On the exterior side of the insulation layer there is a water proof barrier again similar to the wall panel.

Depending on the panel there can be boarding on the outside as well. Onto that the structure to support the tiles is mounted first a vertical layer of lath's and then a horizontal layer (3). The distance between these depends on the distance between the rafters and type of tile used for the roof. On the top of the roof there is the hinge which gives the element its name (4). Once in place the ridge (5) is placed on site.

The sizes of these elements depend on the span and tile division but are generally limited at 10 m long and 3,6 m wide as a result of the transportation limits. (Woude van der, 2005)

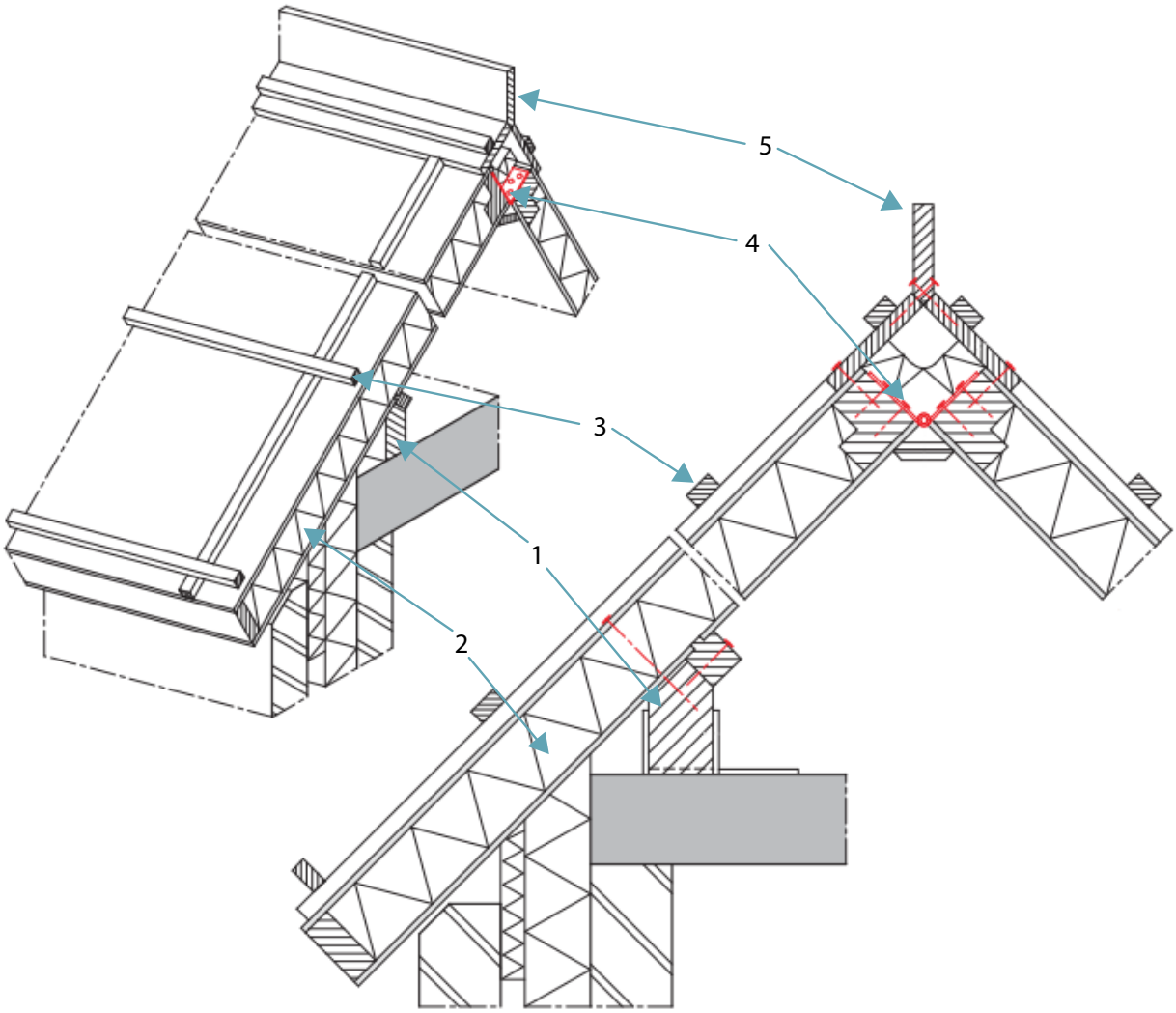


Image 20: Detailed section of a hinge roof (source: (Woude van der, 2005))

Openings

There are different openings that can be made into a roof panel, the four main categories are (Woude van der, 2005):

- Construction outlets – Such as chimneys
- Installation outlets – such as ventilation, plumbing vents or connections for solar panels.
- Dormers
- Skylights

The element that is applied to the opening after its creation is different for each category listed above. However, the process of making the opening is equal in most cases and is similar to the generation of a wall element. In this way the roof penalization can be viewed separately from the components that are applied. Which are mostly prefab elements from other companies such as the building service supplier or dormers or skylights by Velux or any other supplier.

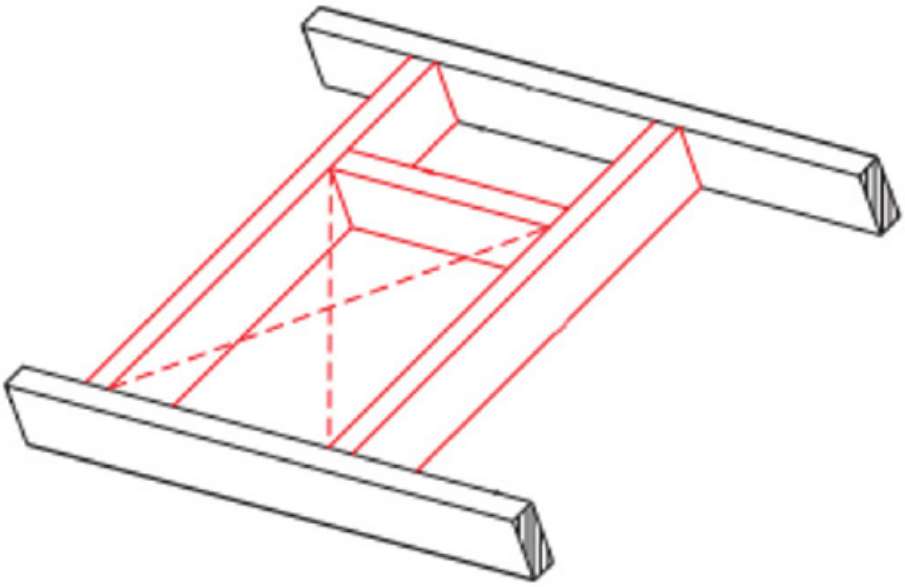


Image 21: Construction around roof opening (Source: (Woude van der, 2005))

An opening has to be surrounded by wooden members that transfer the load to the closest rafter as can be seen in the image. If the opening is so big that it crosses rafters the loads need to travel to the rafters next to the opening. Because of this the size of the wood framing around the openings might increase depending on the number of rafters that are cut.

3.1.3.2. The panel design

As already mentioned in the theory survey the prefab roof elements naturally are constructed using the same design rules as wall elements in the way that at its core it can be considered a framework. However, in order to fulfill its functions some parts have to be modified. Starting with the exterior, Lath's are fixed on the outside in order to carry the tiles. The size of the tiles dictates the distance between the laths and when one wants to do it properly the openings in the roof need to take into consideration this measurement as well to ensure no tiles need to be cut and.

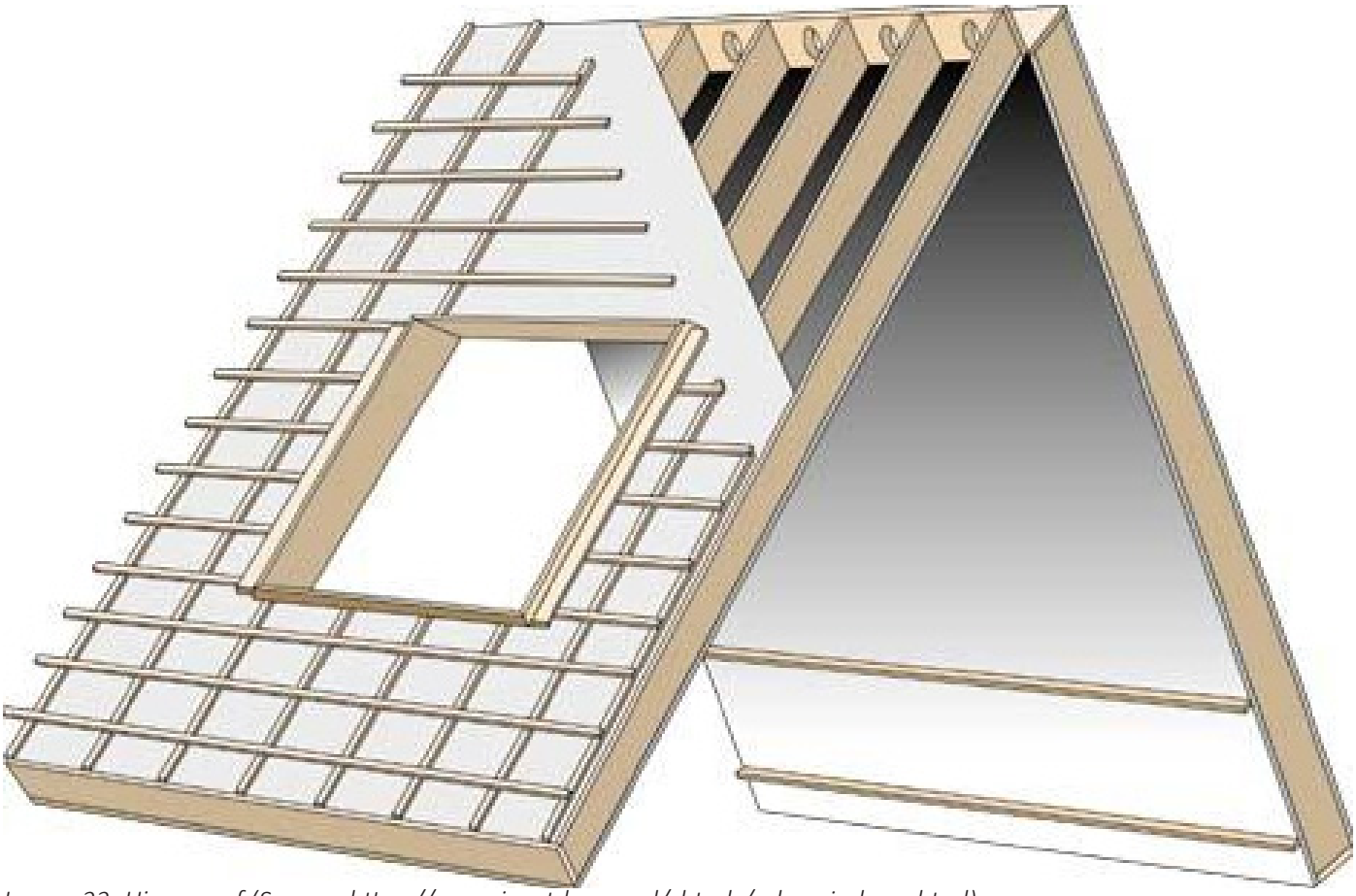


Image 22: Hinge roof (Source: https://www.joostdevree.nl/shtmls/scharnierkap.shtml)

Eaves

The detailing at the eaves of the roof is perhaps the most complicated detailing to be found in a roof as everything comes together here; the roof with the wall, the force transfer, the insulation line and esthetics.
Starting with the mounting of the roof. The wall beam that which is set on site supports the roof and transfers the forces through the bracket to either the floor if it is stiff enough to handle the forces or to a mounting bracket on the other side so the horizontal forces will get canceled out.
Next, it is clear to see where the wall element ends and where the roof element begins. However, the intermediate space is important to design properly. In order for the thermal barrier to be unbroken and as efficient as possible. The same goes for the water barrier. As can be seen in the detail below, there is an overhang added to the roof for esthetical purposes. Right now, it is set but in the computational part of the research this will be something that can be changes by its parameter. This part is not insulated as it not between an inside and outside space. Lastly the detail contains some finishing (the gutter and some wooden boards) these are just there as an indication of how this might be done.

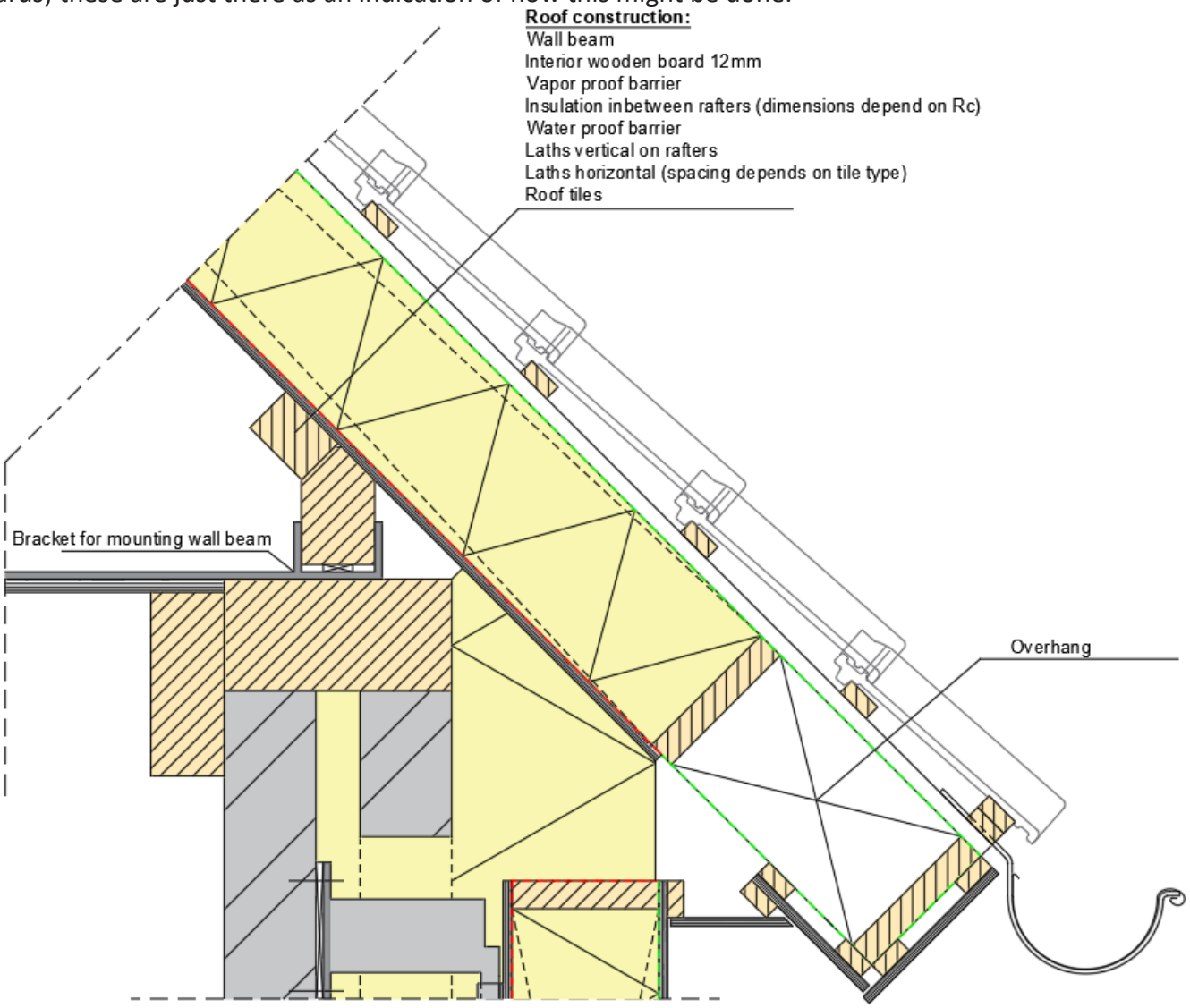


Image 23: Detail of roof base (source: self-made)

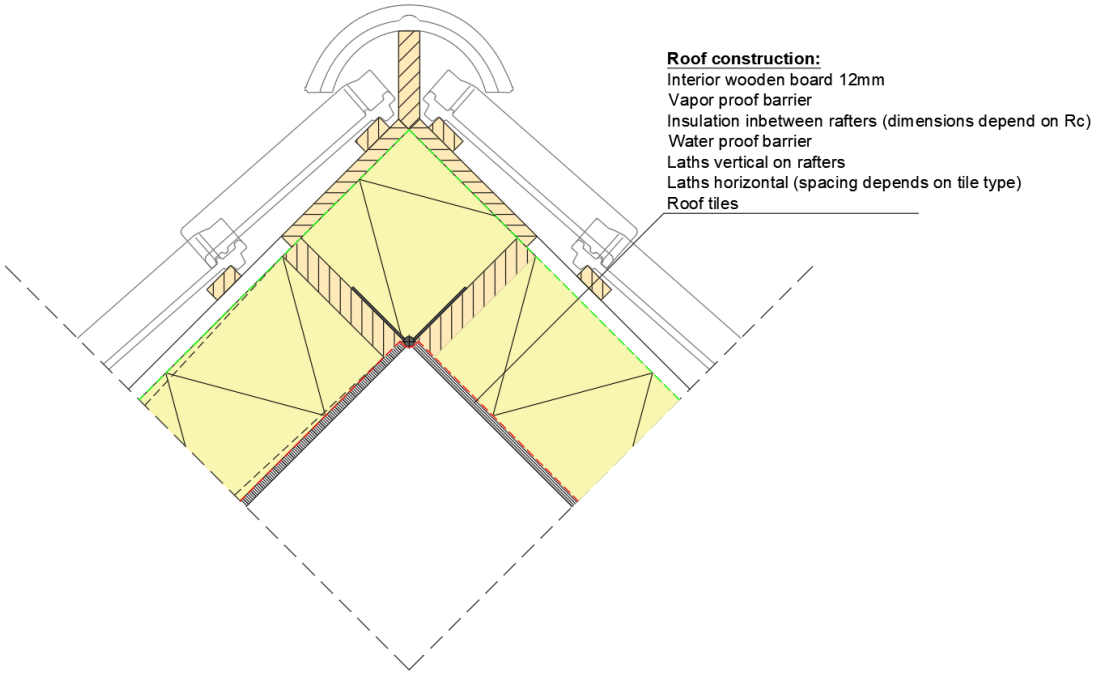


Image 24: Detail of roof ridge (Source: self-made)

Ridge

At the ridge the two separate panels come together and are joined by a hinge. In the image shown below everything above the hinge lines (insulation, laths and water barrier) has to be added on site.

3.1.4. Bay-window panels

Bay-windows are a commonly used method among row-houses to extend the space in their living rooms and sometimes they also provide a balcony.

Image 25: Two typologies of bay-windows (Source: geschiedenisvanzuidholland.nl (L), Own photo(R))

The image above shows the two most common typologies of bay windows. They both serve the same function however the bay window on the left is a complete layer of windows between two segments of wall. The right image however has the windows included in the wall. The design of both cases is similar to how the wall panels are already generated. However, the connection is different as it needs to be inclined.

The bay window on the left will have a layer of wall panels below the window frames, then the framework elements and then another layer of panels above the window frames. These elements can only be joined on site as otherwise the window frame might take unexpected forces. However, the elements can be prefabricated. The bay window on the right however can be prefabricated using complete panels as the windows can be included in the.

Connection between panels

As only the connection between the existing panels and the bay window panels themselves is different to the wall panel generator this part has to be designed and can then be added to the parametric method of the wall generator.

The detail below shows how an inclined connection between two panels can be made. First the connection between the panel of the wall and the panel of the bay window. In which the last stud of the wall panel is kept straight to minimize the number of panels with angular members. The connection between the bay window panels is split half to allow for a flush alignment.

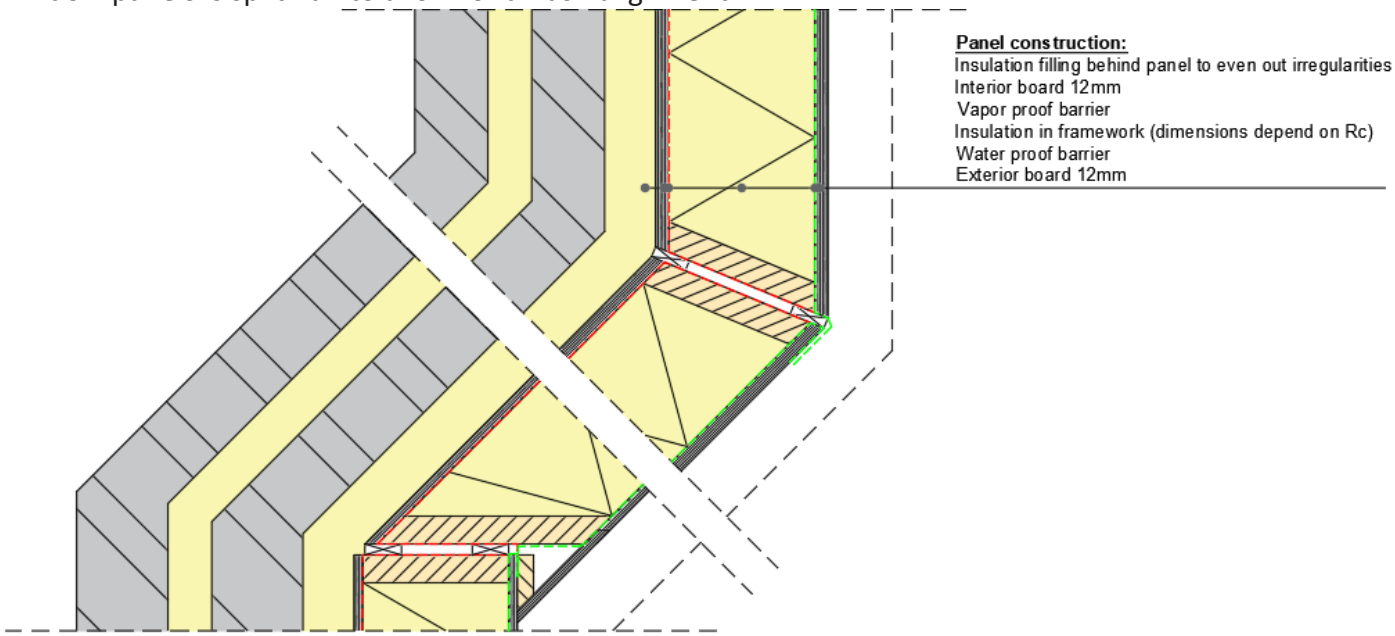


Image 26: Detail showing the connection between bay window panels and normal panels (Source: self made)

3.2. Design of the tool

As already discussed in the research phase of this thesis there is already a lot of research on how to panelize facades in a computational way. However, the amount of research catered towards the same subject for renovation buildings is little. The research found was mostly written by A. F. Barco such as his paper about building renovation adopts mass customization. This paper has been an inspiration for the work done in this thesis.

But, where their research focusses on the design of the façade as a whole with the penalization as an output for further design (using parametric for optimization) this research focusses on the process and an output ready for manufacturing with a high level of detail within the panels (using parametric for atomization). This by taking the design for manufacturing and assembly principle into account in the panel generation. Therefore, the end level of the panels detailing is exact meaning that the materials, properties and positioning of all individual elements is at a final design stage (LOD 300). In doing so parametric method can be used to speed up the modelling of these panels for manufacturing as mentioned in the problem statement. This chapter will go into the process of renovation and automate the steps of engineering the panels

3.2.1. Generic tool design

Having designed the renovation method for the row houses. The next step is to take the renovation process and automating it using a parametric tool. But before a process can be automated first the process and its deciding factors needs to be clarified. In this chapter the steps in the renovation process will be evaluated.

Generic design process

The building process knows different stages of design. In a normal design assignment, it starts with a sketch design in which the needs and requirements for the project site are established. In this stage the general direction for the project is determined. In a renovation project like is the scope of this research this phase is limited to inspecting the existing situation, and determining the demands of the new façade system as well as making a cost estimate.

The next step is the preliminary design in which the design gets more concrete to the point where the final design is made. The measures and their impact are determined as well as the renovation approach. In this stage it is decided what renovation method is to be used. The outcome of this stage are the required data for the execution of the final design such as the RC-values.

The next stage is the engineering for the execution. This is the stage where all components of the panels are designed and where everything is made ready for manufacturing and execution. This part of the design process where this thesis is implemented. The details and geometry of every part are engineered in this stage. It therefore relies heavily on the predetermined requirements from the previous stages.

When the engineering is done the next stage is the manufacturing and assembly of the design. Where all de decisions and designs come together.

The last stage of a building design is the monitoring and maintenance stage. In this stage the building is checked and maintained accordingly to ensure a long lifespan of the building and to ensure the performance, comfort and safety of the building.

With the generic steps of the process clear next the steps specific to the scope of this thesis and the following steps of the renovation will be described along with the required input, its source and the output.

0 – Design phase
The project starts with the design phase in which the state of the building is surveyed and the goal of the renovation is decided. With this starting point and the goal of the design is set. These are used to make the preliminary design. This involves deciding the construction approach, required RC-values and cost estimates.

1 – Data gathering
The first step in the process of engineering the renovation panels is the gathering of data which is used to create the elements that are used for the renovation. This can be done by referring to as-built drawings, 3D-scans or on-site measurements. As long as the measurements are accurate and any deformations are recorded. This information is generally already gathered during the design phase of the project.

2 – Determine area to be panelized
The next step is to take the data and determine what area should be panelized and what panels should be used (Roof or wall). The result of this step will be the contours of the surface that will be divided into panel contours. This contour is created by taking into account the space required for connections to exterior parts and other panels. As well as the mounting system to be used. The result is acceptable if it can be panelized properly. Meaning the panels used to fill the surface area are according to the manufacturing standards. No panel should be too small nor too big to be able to fill the area, the openings can be fitted and the panel can be mounted to the surface as well as connected to other panels/exterior elements such as window openings.

3 – Create panel contours
With the surface area to be panelized known, the panel contours can be created. This is done by filling the surface contour with panels. The goal is to fill the surface with as few panels as possible while taking into account the maximum panel size which is based on transportation, manufacturing and assembly limits. As well as a minimum panel size limit which is based on constructability. Furthermore, in between the panels there needs to be space for tolerances which depend on the building method and panel sizing. The output is a group of contours of the panels which is acceptable when the mentioned parameters are met.

4 – Create panel geometry
After the boundary of the panels are determined the infill needs to be engineered. The basis of this infill is the framework and insulation layer. Which is created to ensure the required RC-value which was calculated in the EPC calculation for the building and is achieved by the thickness of the insulation layer. The wooden members in this layer provide the strength for the panel and their size is determined by structural calculations as well as standard timber sizes. Surrounding these layers are the vapor and moisture barrier to keep the moisture out of the panel. Then there is a layer of board that provides the stability of the panel. This layer's dimensions depend on the amount of shear force it needs to withstand and the standard sizing's of the material. As a last part the windows are engineered into the panels. The size of the window is dictated by the opening implied with the surface contours and is dependent on the required natural light that has to enter the building. The output of this step is acceptable if the required RC-values are met, the panels can be manufactured and the external connections can be made.

5 – Calculations
With the panels generated their specific weight and RC-value can be calculated. This is done to ensure the required RC-values are met and to inform the engineer about the weight of the panel for the engineering of the mounting brackets. But also, to document the data. The output is acceptable if the panel RC-requirements are met and the weight can be carried with the mounting brackets assigned to the panel. If not, the panel needs redesigning.

6 – Engineer connections
Next the connections between the panel and external components such as reveals, panel to panel connections and the mounting brackets can be engineered. These connections depend on the geometry of the component they need to connect and the requirements to this connection. For the mounting brackets the strength of the bracket is a limiting factor and the number of brackets is determined by structural calculations and the weight of the panel. For the connections between the panels building physics determines how the connections are made. They need to be waterproof and not create a hole in the insulation.

7 – Manufacturing & transport
After the engineering is done the plans for the engineering are handed to the manufacturing site where the panels are made along with all their attributes. When manufactured the panels are transported to the construction site to be installed.

8 – disassemble
In order to renovate the building first the obsolete parts need to be removed such as the window openings and existing roof panels. This so that the panels and their connections as they have been designed can be placed. All these elements will be marked on a disassembly plan to be communicated to the construction site workers.

9 – Mounting brackets
Next the building is prepared for the attachment of the panels by installing the mounting brackets on their designated location. This is a very precise operation as the mounting brackets are already attached to the panels, and so decide how the panel will be mounted. The locations of these brackets are engineered in along with all other connections.

10 – Mounting panels
With everything ready the panels are mounted to the façade. The location of the panels on the façade needs to be checked to ensure the panel is placed as it was intended (within the tolerances that are taken into account during the design). Also, there connections to other panels and the reveals are installed. These connections need to be checked as they are the “weak points” in the design because it’s the only part that ensures a fully closed façade that is installed on site.

11 – Apply cladding
After the panels are installed, the technical renovation is completed, with the panels the house now complies with the building regulations and the RC-value of these surfaces is sufficient for heating with low temperature heating. However, cladding of any kind will need to be applied to comply with the aesthetic standards required by the local aesthetics committee. Depending on the cladding material additional layers might be required. This is the last layer to be applied to the structure.

12 – Post construction
With the cladding and finishing applied to the building envelope the renovation is complete. The building complies with the demands set out in step 0. To ensure the performance, safety and comfort of the building it needs to be monitored periodically and maintained to ensure it stays in a good state.

Those are the general steps in the process of renovation. The excel sheet on the next page shows the steps in more detail along with their input and output. In order to give a complete overview of the process.

Table 11: Engineering steps, there sub operations and the required input and expected output.(Source: self made)

step	Operation	Comment	sub-operation	In
0	Data gathering	Collect and manage data from the building	Acquire 3D scan/model/ drawing	
1	Surface area	Decide what area of the surface will be panelized	What type of panel should be used	Roof/wall
			Create boundary contour	External conn
				Sizing of area
2	Panel contour	Creates boundary contour for the panels	Create surface & opening domains	Surface conto
			Create panel boundaries	Panel size lim
				Tolerances
3	Panel geometry	Geometries of all panel members are generated	Generate wooden members	Member sizin
				Layer thickne
			Generated insulation	Layer thickne
			Generate solid layers	Layer thickne
			Place windows & doors	Window geor
				Window tags
4	Calculations	Perform calculations to validate panel performance	Calculate RC-value	Layer thickne
				Lambda value
			Weight calculation	Geometry vol
				Density
5	Engineer connections	Engineer connections to all exterior parts	Mounting brackets	Panel weight
			Opening connections	Sizing & connection
			panel to panel connections	Gap between
6	Manufacturing & transport	The creation an shipping of the panels	-	Generate geo
7	Disassembly	Disassembling window frames and other objects in the way of the to be placed panels	-	Disassembly c
8	Mounting brackets	Placing mounting brackets	-	Locations and brackets
9	Mounting panels	Mounting the panels to the surface and applying all connections to external parts	-	Fabricated pa
10	Apply cladding	Apply cladding and finishing to the surface	-	Type of mounting pro

Parameters

nput	Source	Output	Goal
		Model of the building envelope	Inform all other steps
	Nature of the surface	Decide panel generation method	Inform all other steps
nections	Openings	Locations of connections to reveals	Engineer reveal connections
	Other surfaces to be panelized	Locations of panels in other planes	Engineer connections to panels
	Minimum panel sizes	Go/No go	Inform if whole area can be panelized
ours	Data step 0	Mathmathical domains	Inform panel distribution
its	Transport size limits	Maximal panel size limit	Inform panel distribution
	Factory size limits		
	Assembly size limits		
	Building method & panel sizes	Distance between panels	Inform panel distriburtion
	ISO/NEN 2768		
g's	Structural calculations earlier design stage	Wooden member geometry	BIM for manufacture & assembly
ss			
ss	EPC calculation	Insulation geometry	
ss	Structural calculations	Solid layers geometry	
metry	Building plans earlier design stage	Window geometry	
sses	Panel geometry step 3	RC-value panels	Check and document performance According to design stage
e	Material property		
ume	Panel geometry step 3	Weight panels	Inform engineering of mounts
	Material property		
& sub structure	Structural calculations	Locations and dimensions of brackets	Inform construction site workers
location of	Design & data step 0	Geometry for connection reveal	
panels	Tolerance step 2	Geometry of panel to panel connection	
metry panels	Step 3	Fabricated panel	Building site
drawing	Earlier design stage	Disassembled openings ready for renovation	To be covered up with panels and reveal connections
d dimensions of	Step 5	Brackets located on the building envelope	To mount the panel using the brackets
nels	Step 6	Mounted façade panel	To match the design
cladding and ocedure	Earlier design stage by architect	Finalized façade	To match the design

All steps and parameters used in these steps are pieces of information that are communicated between professionals that each contribute to the process. Each of them needs information from previous stages. This is shown in the table on the previous page but to give a better overview the diagram shown here has been made showing the flow of information from one profesional to an other and what they do with this information. The flow sugests a liniar flow of information however in reality a more intergrated desgin process is prefered in which profesionals work closely together and data flows between all steps.

Final generic tool scheme

Having composed an overview of the renovation process and the information flow a final scheme for the computational method can be drawn up. This scheme is composed of the scheme from the research phase chapter 2.3.5. and the steps and in-/output of the table is added. The end result is a scheme of on the next page. Showing the tool input and what parameters it is based on. As well as all the operations conducted to get from the input to a Building Information Model.

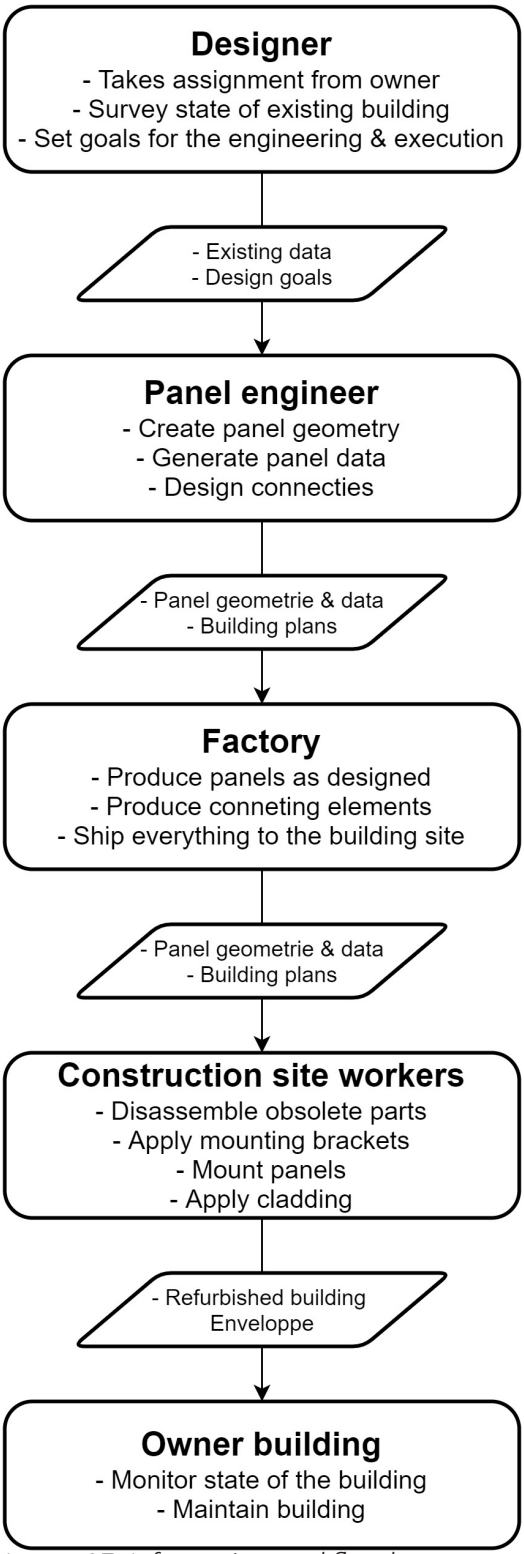


Image 27: Information workflow between professionals in the renovation process

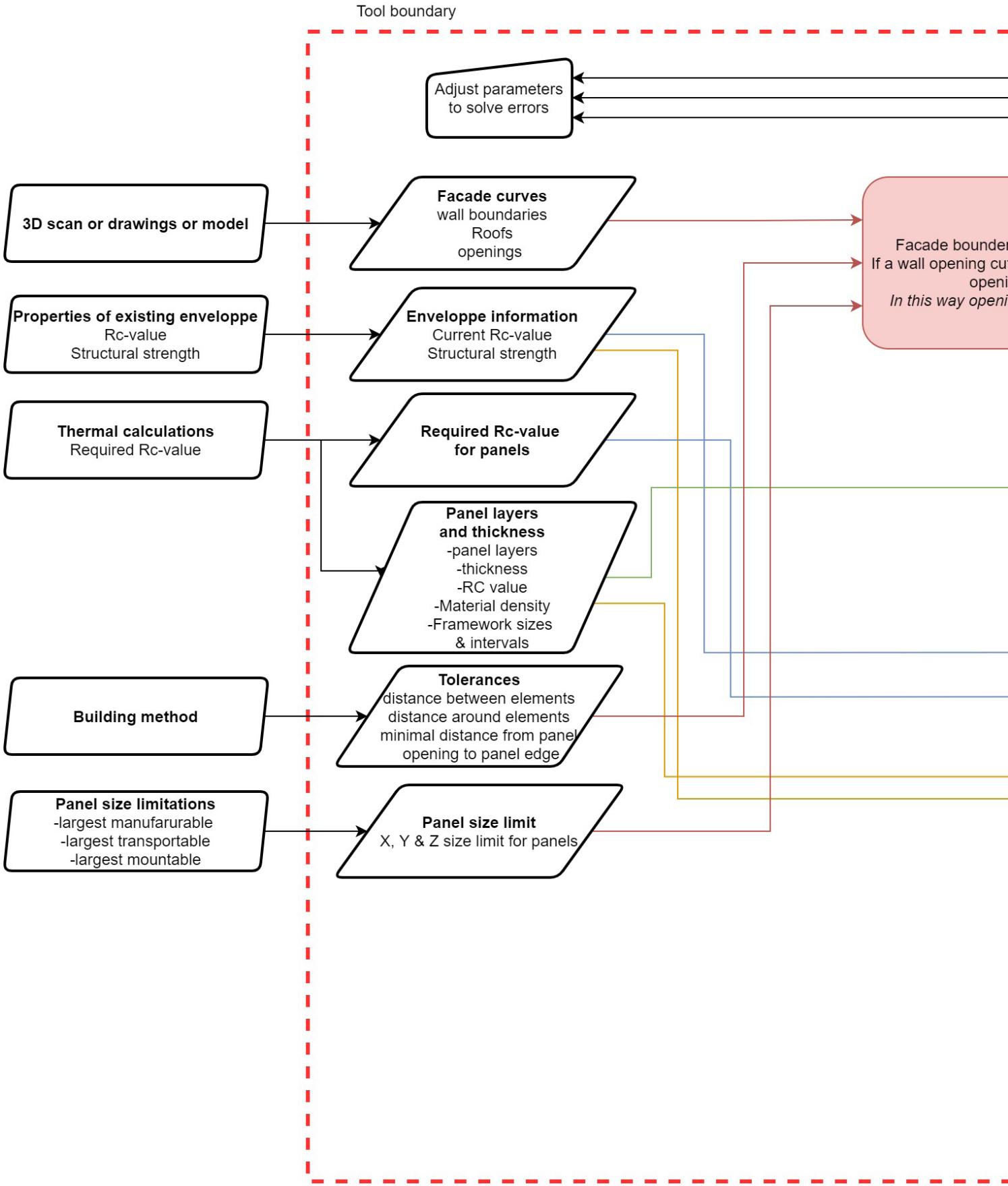
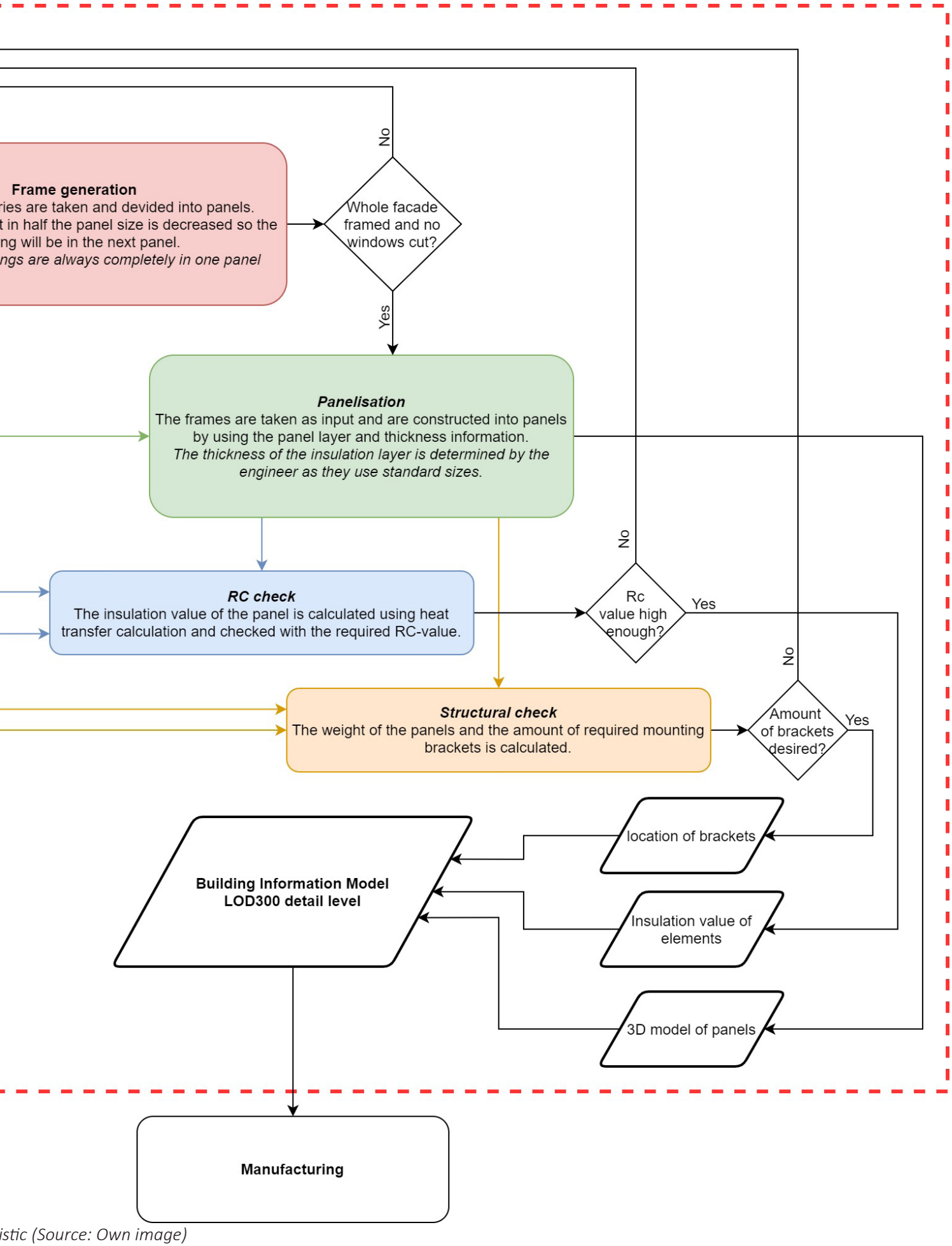


Image 28: The final scheme showing the general method for the design of the tool. Which will be further specified per character



istic (Source: Own image)

3.1.7. Wall generation tool

The essence of the wall generation tool is to take the input of the wall and its specifics and combine it with the design rules provided by the user in order to provide the user with 3D models of the panels and information about their thermal performance and weight. The process gone through is explained in this chapter, an overview of this process has been given in the flow chart below. It is up to the user to decide if he/she want the computational method to generate the panel contours or if he/she wants to provide them. However, the structure in this chapter follows the chronological work order within the tool.

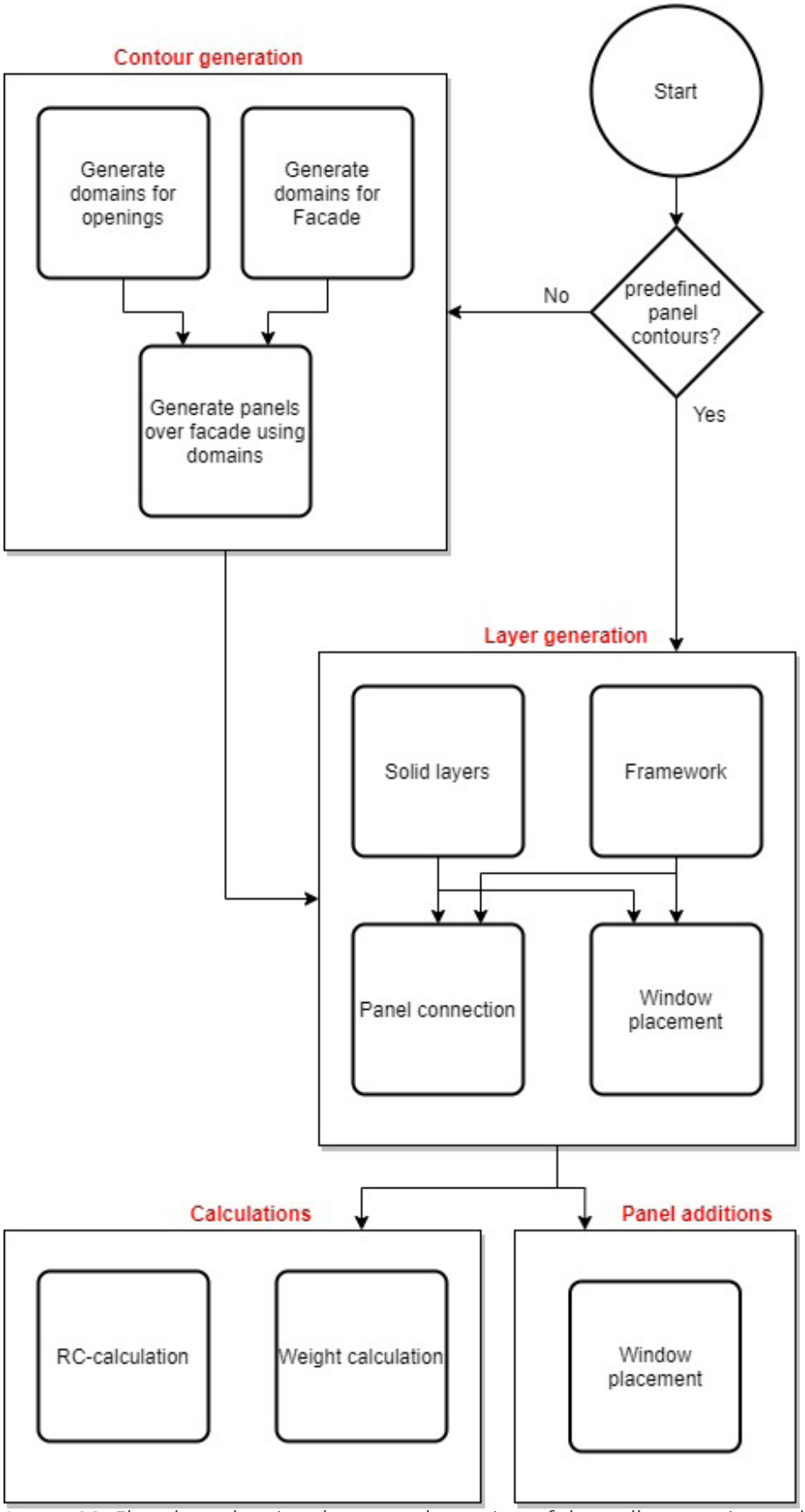


Image 29: Flowchart showing the general overview of the wall generation tool (Source: own image)

Grasshopper script

The workflow diagram above has been translated into grasshopper as can be seen on the image below.
The green groups on the left side represent the inputs per category, the purple elements are the components that process this input, and the cyan group on the right represents the output of the script in every component's geometry, the panel weight and RC-value
The rest of this chapter will go into detail as to how these individual components function. The grasshopper script can be found in appendix 7.1.

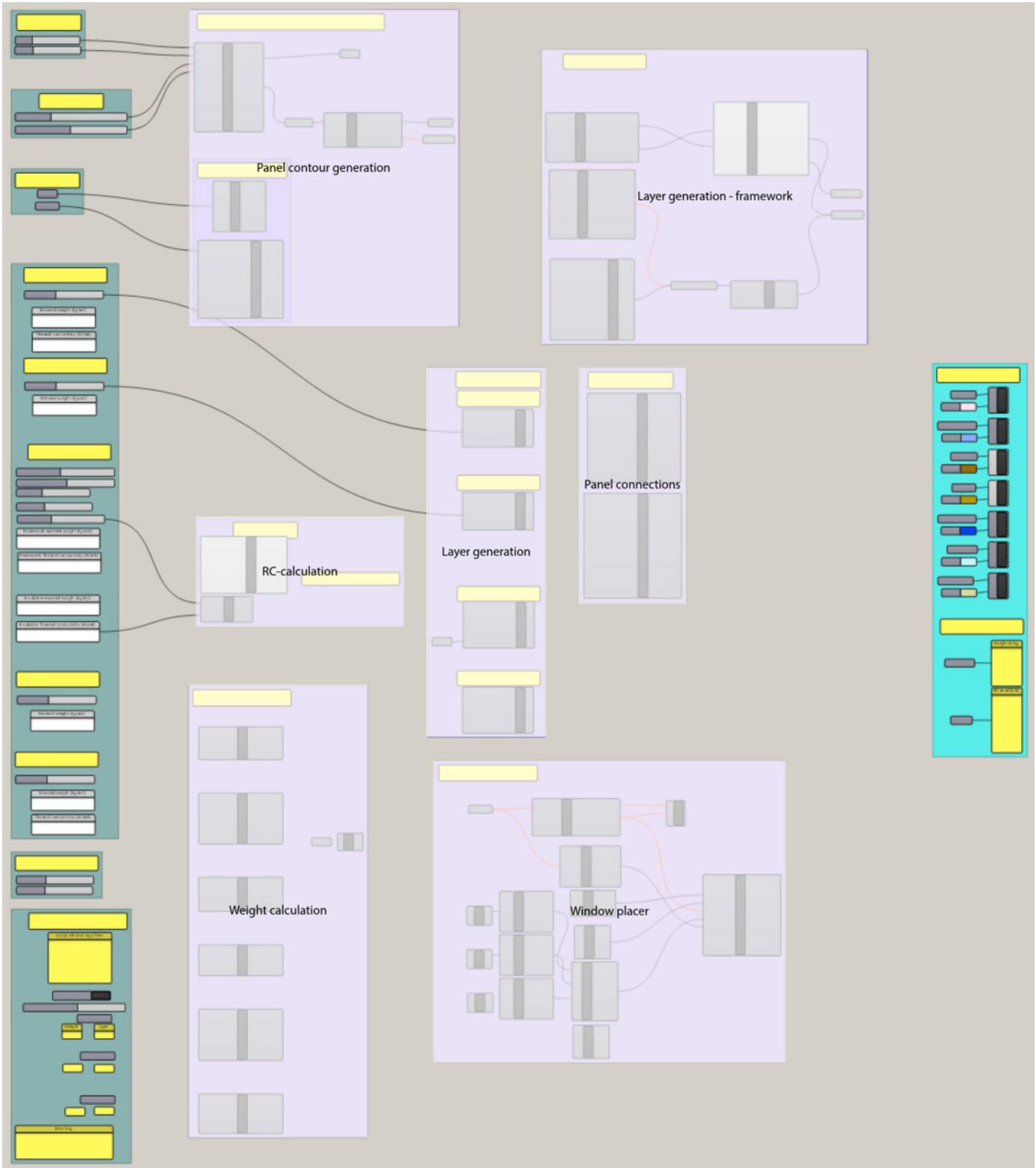


Image 30: Grasshopper file that performs the generation of wall elements (Source: own image)

3.1.7.1. Contour generation

To start of the generation of the timber wall panels, first their contours need to be determined. This paragraph will first go into the inputs required for the computational method to provide said contours after which the process of generating them will be discussed.

Input

In order for the method to operate it requires inputs, these are listed and discussed in this section.

- *Edge tolerance*
  To take into account the tolerances required around the edge of the panels this parameter is used.
- *Tolerance between panels*
  To ensure ample spacing between the different panels this parameter is used
- *Façade curves*
  To generate the domains for the façade the curves are used as an input.
- *Opening curves*
  To generate the domains for the openings the curves are used as an input.
- *Stud width and plate height*
  These parameters are used to ensure enough spacing between panel edges and window edges
- *Maximum panel dimensions*
  The maximum height and width a panel can have in order to keep the panels within these limits

Process

The process of generating the panels is done in three steps. The first is setting the limits of the panels by generating domains in which they can be generated. After that the panels themselves are generated. Lastly the generated panels are structured within the grasshopper logic. This process is chosen over an algorithm like used in building renovation adapts mass-customization (Barco A.D., 2016) because the facades are far less complex in their panel layout. It might even be that the user provides their own panel layout.

<u>Domain generation - Theory</u>

In order to properly generate the panel contours. First the domain of the façade and the locations of the windows within this domain are established. So, to ensure the locations at which there cannot be any panel boundaries. The process of the generation of these domains is illustrated in image 31. The façade domain (blue) shows the range within which the panels can be generated. The opening domains (red) show where there cannot be any panel boundaries. These are generated for X and Y separately. This approach works for rowhouses well because the layout of the openings in the façade is generally in line with each other. Meaning that openings are positioned above each other and on the same height rather than in other lay outs for example a chessboard pattern on which this approach would not work.

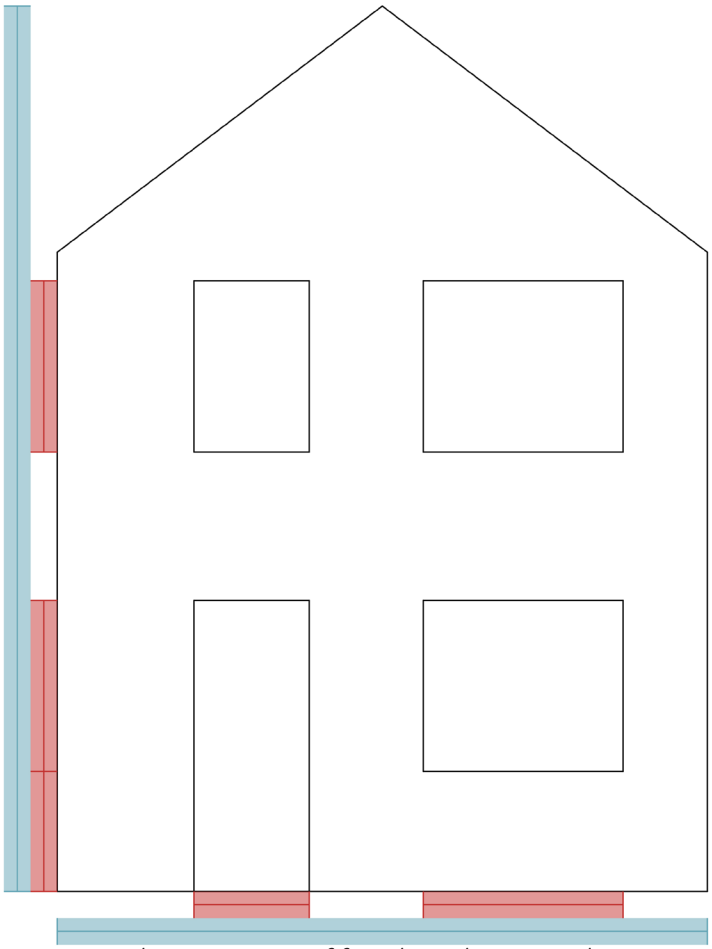


Image 31: The generation of façade and opening domains (source: own image)

Domain generation - Workflow

The workflow of the generation of the domains is displayed in the workflow diagram below the domains for the openings (left) take into account an additional offset to ensure a stud and plate can always run around the opening and thus support the framework. The domain of the façade has no need for additional information.

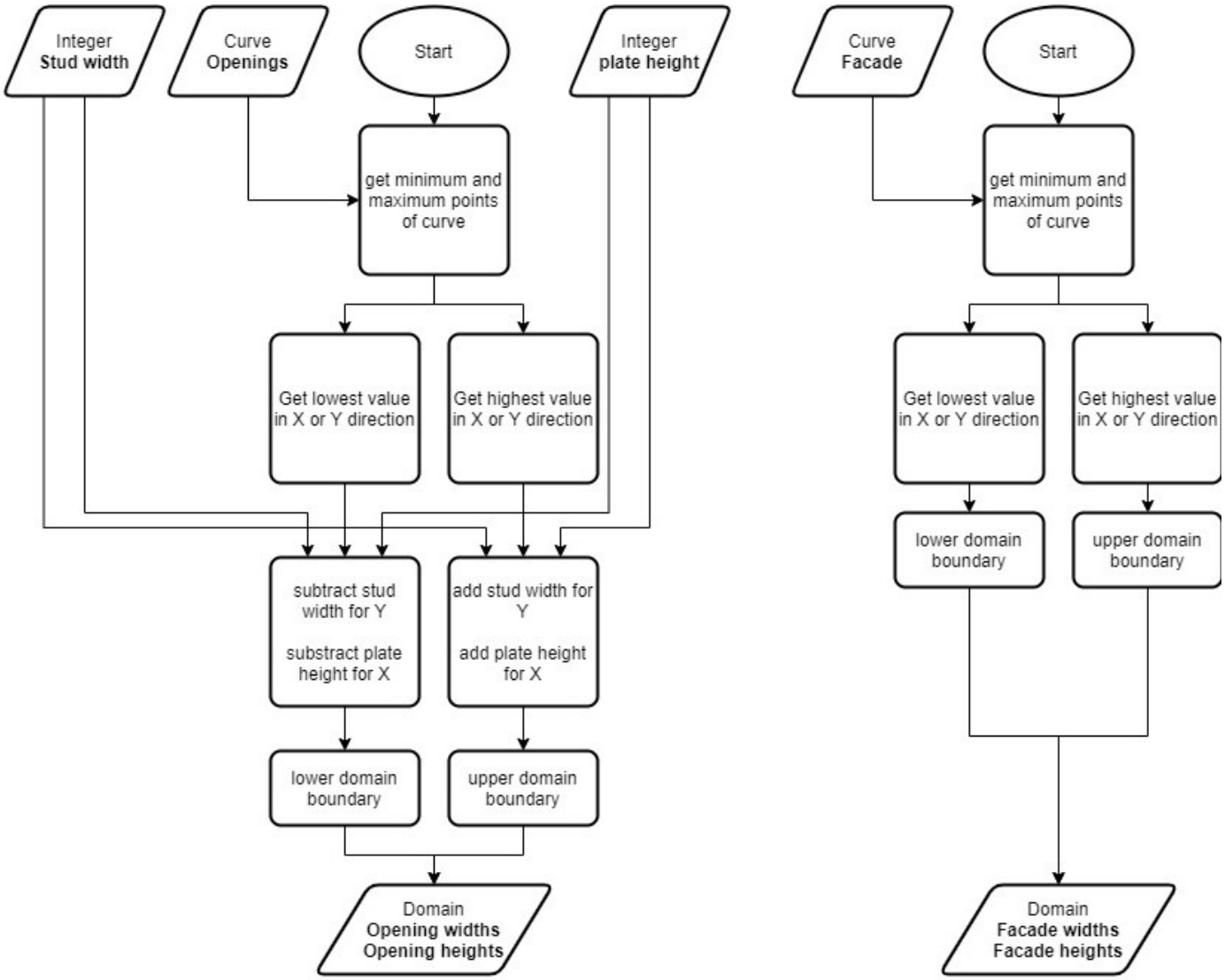


Image 32: Workflow of the process of generating domains for the windows (Source: Self made)

Domain generation - Grasshopper

As the generation of domains take a geometry as input and a domain as output only a single element is used that is coded using python. The python script of these components is added in appendix 7.1.1.

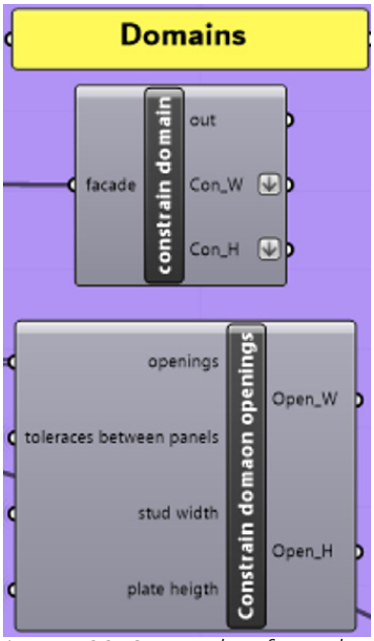


Image 33: Screenshot from the grasshopper elements creating the domains

Contour generation - Theory

The domains will be input for the grasshopper component that determines the positions of the corner points of the panels. It does so in both X and Y direction separately and takes the previous determined point (in the first iteration this point is the corner point of the panel.) as a starting point and adding the maximum panel size to this point. It then checks if the point is in the domain of a window or outside of the façade domain. If so, it places the point back to the last proper location. In case of an intersection with an opening the point is placed back to the edge of the opening while taking into consideration the minimum distance it needs to keep from this edge (in order to fit a timber frame around the window). If the generated point is both within the façade domain and outside the opening domains the point is accepted and the component moves on to the next point. The image on the right shows the end result of this process as the corner points are joined by a line. The edges tolerance and tolerances between the panels can be seen in this image. For this generation the panel size has been set in such a way that it intersects with the right windows. Which is handled by placing the contour of the panel to the left of the opening. This process results in the blue rectangular panels which are then clipped by the red lines to form the final panels for the façade. The final panels are shown in the lower illustration.

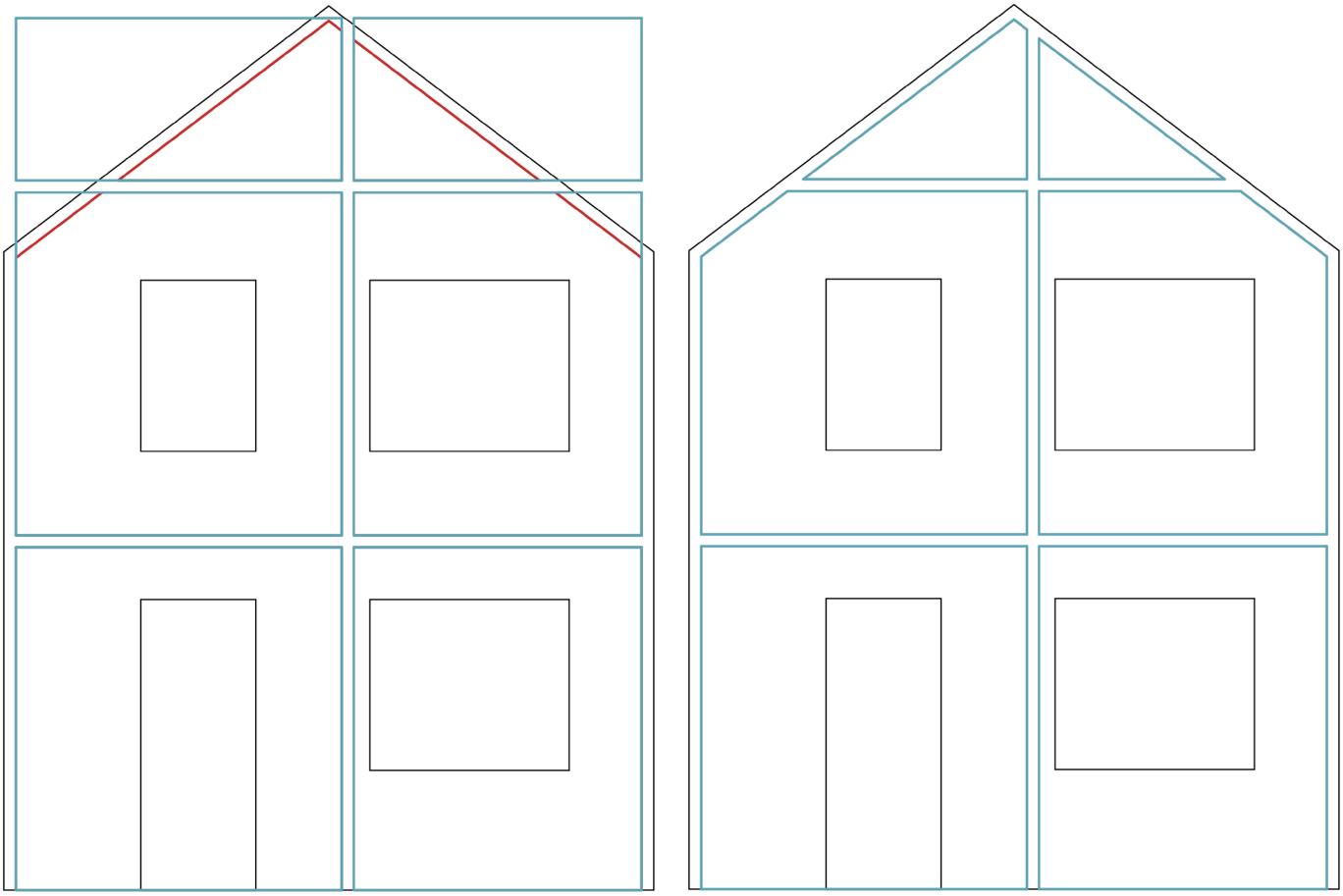


Image 34: Visuallisation of the panel contour generation (Source: own image)

Contour generation - Workflow

In order to create these panel contours the workflow shown below is used. It does so by generating a list of x and Y coordinates is generated separately using a while loop, which means it runs until a condition is met. In this case that condition is the last point in the generated list being the last placeable point on the façade. When the coordinates are found they are used as corner points for the façade and thus create rectangular panels. These panels are then cut using a panel boundary curve to fit them in the façade.

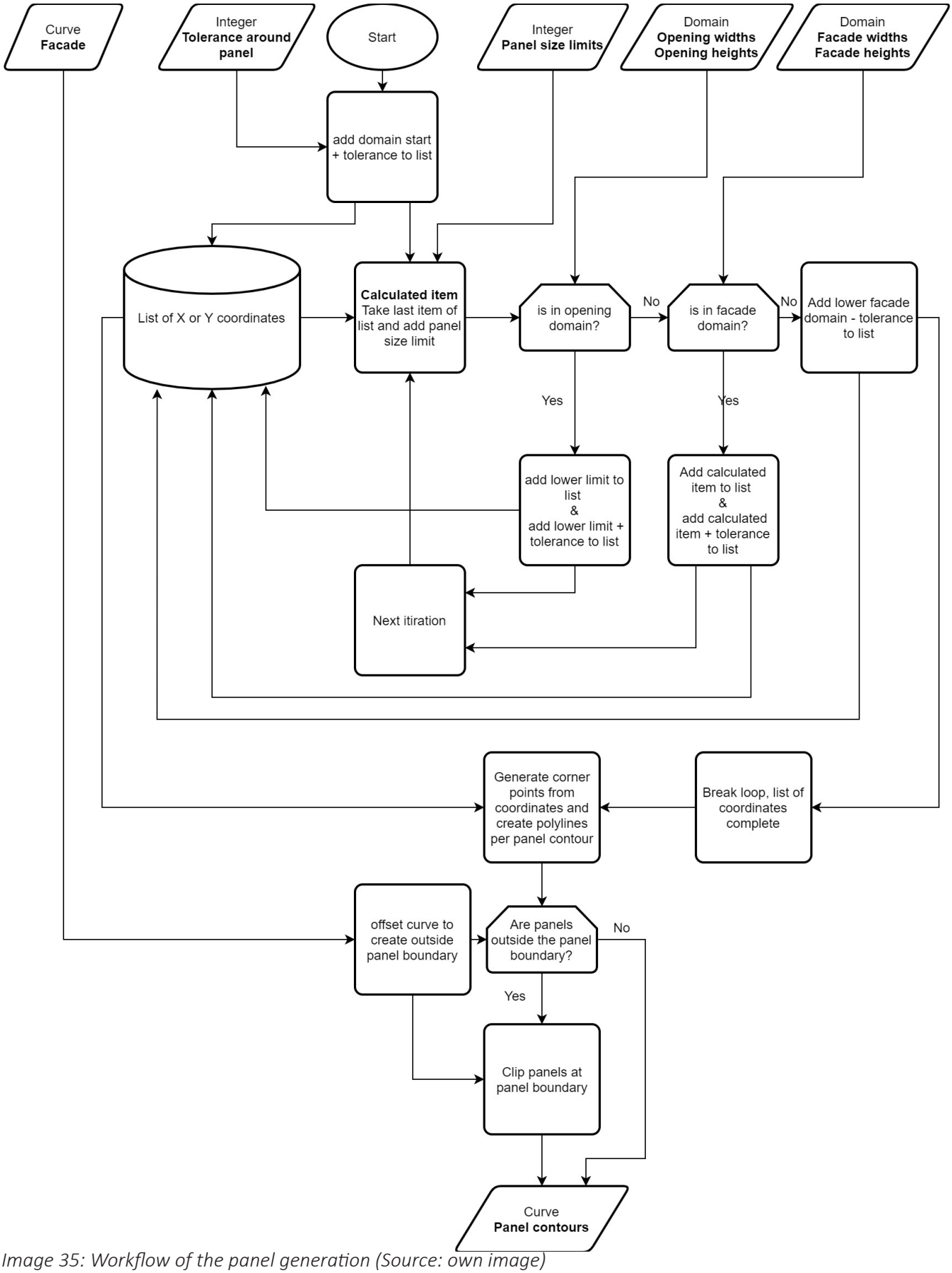


Image 35: Workflow of the panel generation (Source: own image)

Contour generation - Grasshopper

The workflow as shown on the previous page is executed in grasshopper the buildup is shown below, further detailing about the components and their python scripts can be found in appendix 7.1.2. First the façade is divided into rectangular panels, then the exterior curve of the façade is offset inward to create a panel boundary which is used to clip or cut the panels that stick out. When this is done the gaps in the contour created by the clipping or cutting are closed to form a continuous panel contour within the façade boundary.

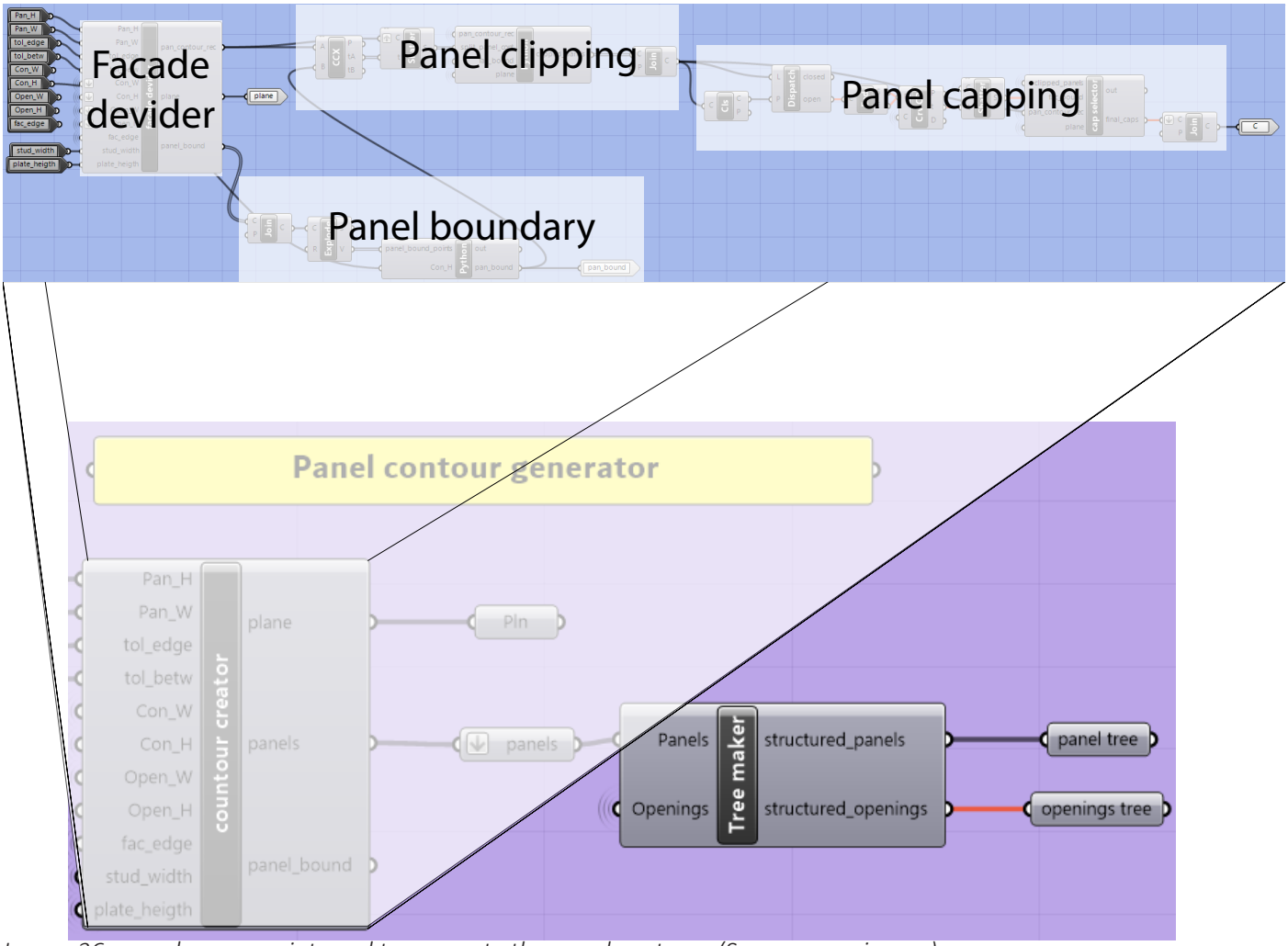


Image 36: grasshopper script used to generate the panel contours.(Source: own image)

Structuring

As a last part of the contour generation and as a base of reference for the rest of the method the panels generated and the openings provided are structured in the grasshopper tree structure in such a way that every panel is in its own branch and that openings within those panels are assigned to the same branch (within another tree). This enables the method and later the user to refer to the panel individually and pull their information. The principle of structuring is shown in the workflow below.

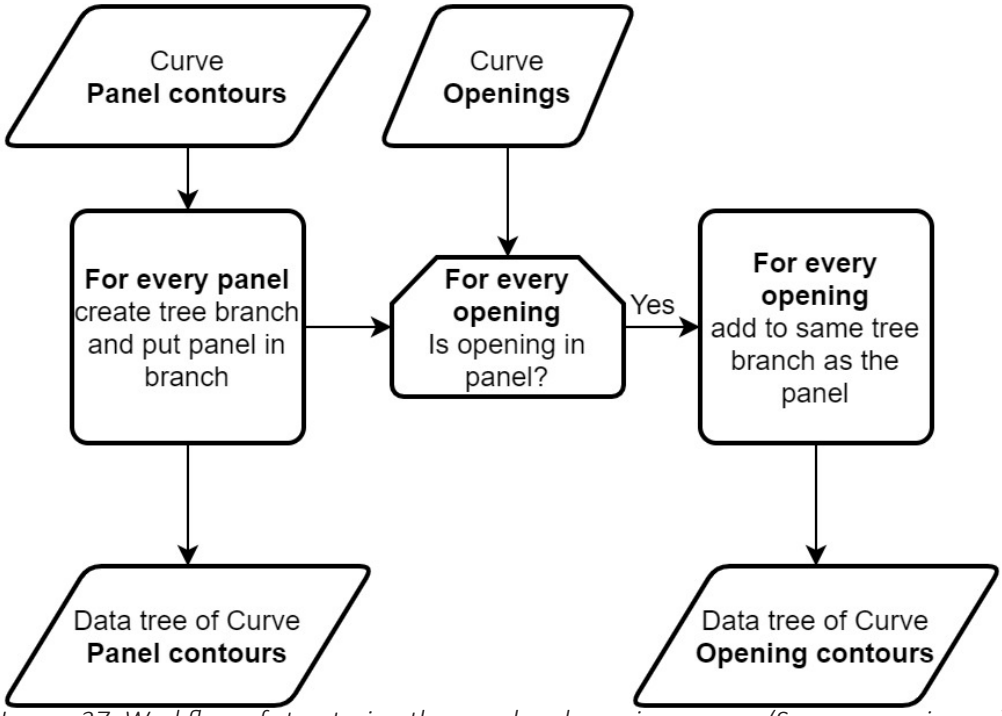


Image 37: Workflow of structuring the panel and opening curves. (Source: own image)

To make this workflow function in grasshopper a python script has been used, it is added in the appendix 7.1.3. and because its size is small it's also added below. This block first creates the two data trees for panels and openings. After this it runs over all panels and generates a tree branch to place them in. while also checking for openings in the panels. If there are openings, they are added in the corresponding tree branch. If there are no openings an empty branch is generated in order for the rest of the tool to function properly.

```
structured_panels = DataTree[rg.Curve]()
structured_openings = DataTree[rg.Curve]()

for i,pan in enumerate(panels):
    myPath = GH_Path(i)
    structured_panels.Add(pan,myPath) ## adds panel to data tree
    open_in_pan = False
    for j,check in enumerate(is_inside.Branch(0,i)):
        if check == 2: ## 2 means the center point of the opening is inside the panel
            structured_openings.Add(openings[j],myPath) ## adds opening to data tree
            open_in_pan = True
    if open_in_pan == False: ## if there is no opening in the panel it generates an empty branch
        structured_openings.EnsurePath(myPath)
```

Block of code used to structure panels and their respective openings in grasshopper

3.1.7.2. Layer generation

In this chapter the generation of the different panel layers will be discussed excluding the framework which will be discussed in the next chapter. The layers generated in this chapter are solid layers that are the size of the panel and have openings where there are openings in the panels. In theory the number of layers can be as much as required. For the purpose of this research, it was limited to five being: 1) interior board, 2) vapor proof layer, 3) framework, 4) water proof layer, 5) exterior board. The process of all layers except the framework is identical and so only the generation of one layer will be explained.

Input

In order for the method to operate it requires inputs, these are listed and discussed in this section.

- *Structured panels*
  The structured panels are the boundary of the layers.
- Structured openings
  The structured openings allow the layer generator to cut the openings out of the layer per panel.
- Layer thickness
  The layer thickness provides the thickness the layer should have.
- Thicknesses of layers before
  In order to get the layer in the proper location in the model it needs to be offset from the plane with the distance of the sum of all the previous layers their thicknesses.

Process

The process of generating these layers illustrated in the image below. First the surface of the layer is generated per panel (left). Then the windows are cut out of this surface(middle) lastly the surface is moved to the right location within the panel using the thicknesses of all layers before and extruded using the thickness provided(right).

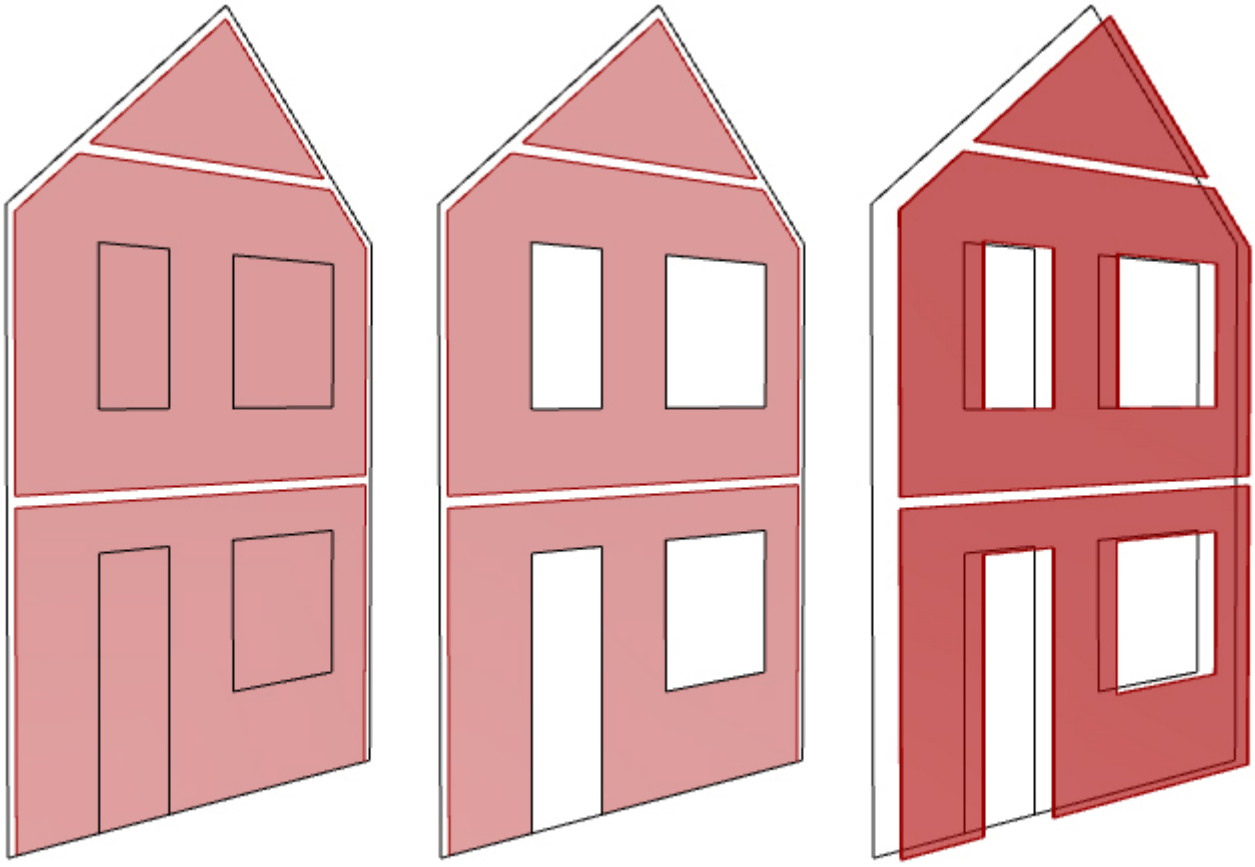


Image 38: The generation of a solid layer(Source: own image)

Layer generation - Workflow

In order to execute the process as described on the previous page the workflow diagram shown below is used.

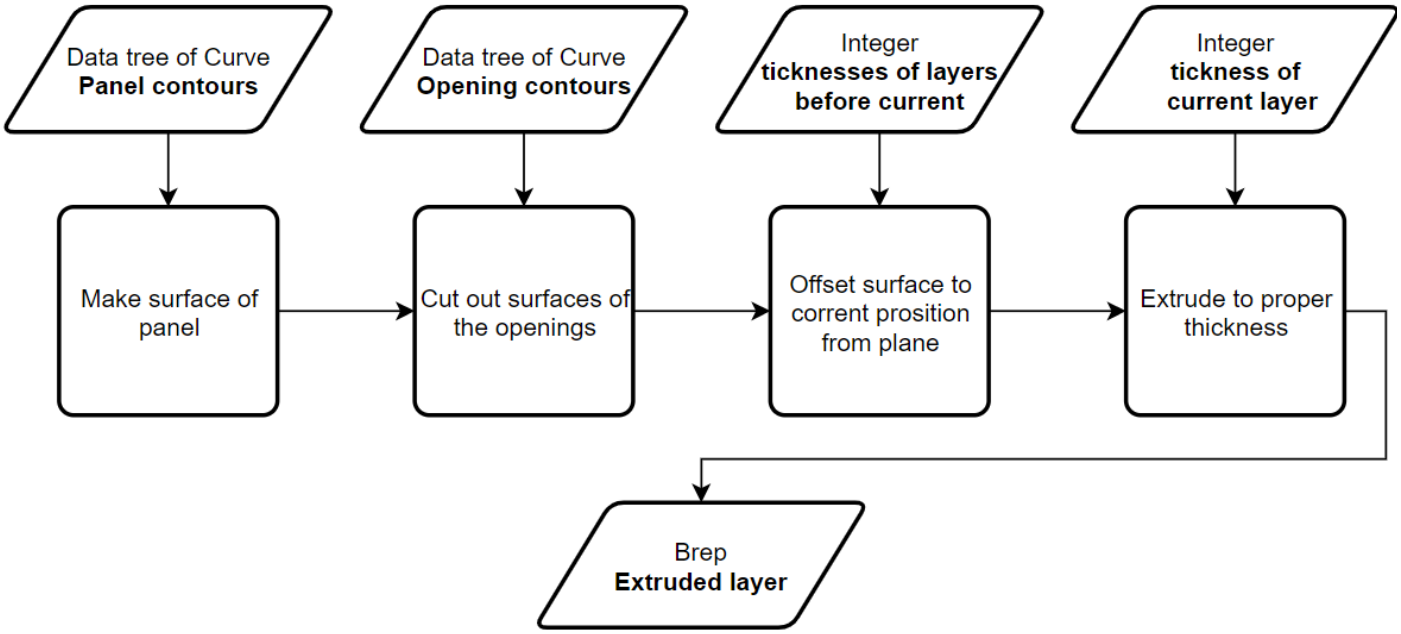


Image 39: Workflow for extruding panel layers(Source: own image)

Layer generation - Grasshopper

To execute this process in grasshopper the script below has been used as a cluster to be reused for all layers individually. This makes it easier for future users to modify the number of layers. The top row generates the surface and takes out the windows after which the middle row of components moves it into position and extrudes it to the correct thickness.

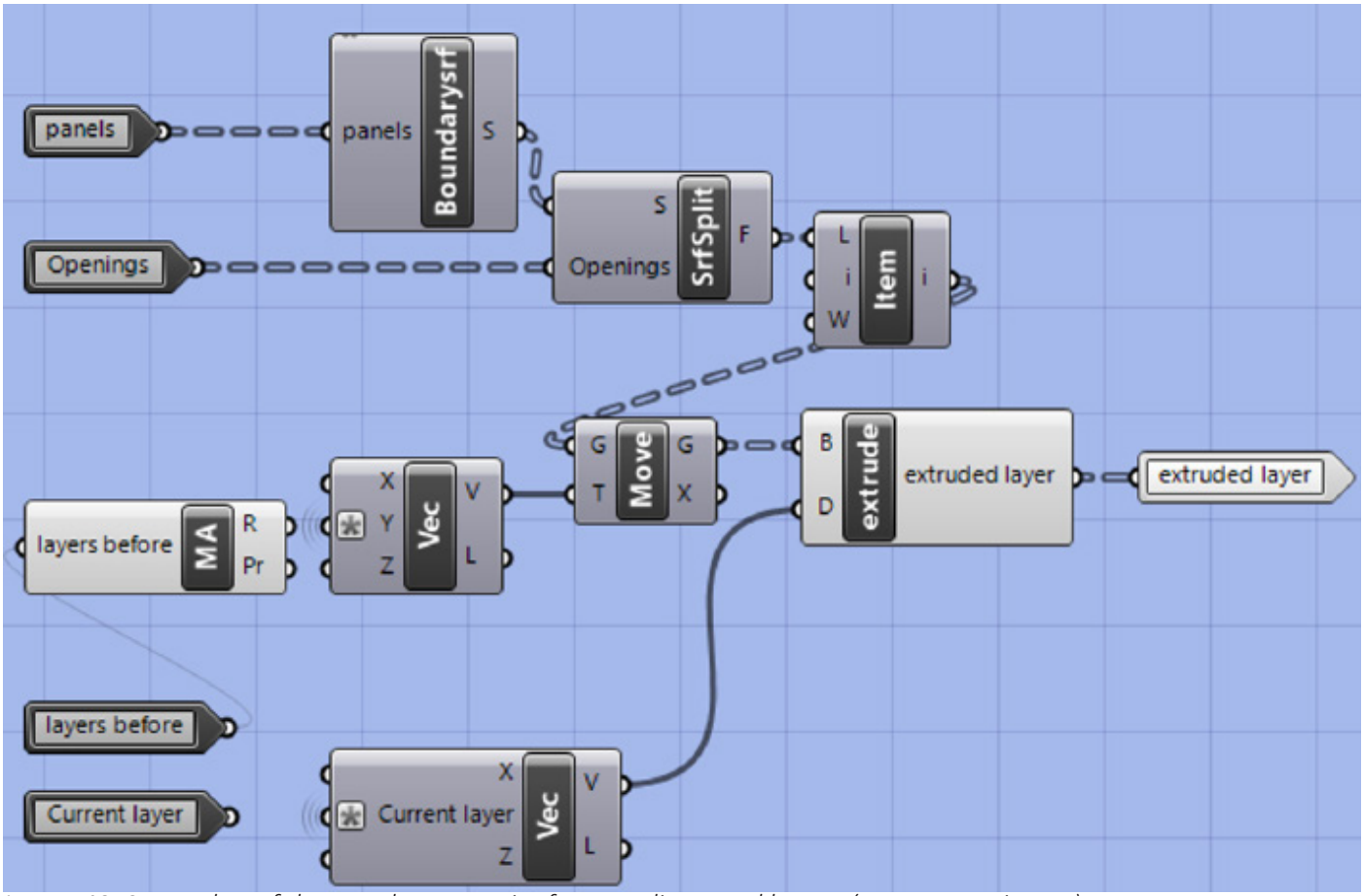


Image 40: Screenshot of the grasshopper script for extruding panel layers. (source: own image)

3.1.7.3. Layer generation - Framework

The generation of the framework is more complex than the normal layer generation it still relies on the surfaces, moving and extruding them. However, the way in which these surfaces are generated is more complex as it needs to take into account the panels edges, openings and the wooden members of the framework itself.

Input

In order for the method to operate it requires inputs, these are listed and discussed in this section.

- Structured panels and their windows
  A tree structure with all panels in a separate branch and a different tree structure with openings in the same branch as their panels are in.
- Plane
  Plane in which the façade is orientated.
- Plate height
  Integer in which the height of the plate members is set.
- Stud interval
  Integer in which the spacing between studs is set.
- Stud width
  Integer in which the height of the stud members is set.
- Min distance between studs
  Integer in which the minimum distance between studs is set, used to check if studs are not positioned to close to studs at the end of the façade of next to windows.
- Layer thickness
  Integer in which the thickness of the framework layer is described.
- Thickness layers before
  Integer in which the thicknesses of all layers before the framework are added. So that the framework can be positioned in the proper location.

Process

The process of generating the framework is divided into several different steps all of them are described per different component generated

<u>Boundary members - Theory</u>

The boundary members which are the wooden members of the framework that form the outline of the panel. First the horizontal members are generated as these are generally the base for the panel or the connection surface between two panels which need to be straight. The process of generating these members is done by taking the horizontal lines of the panels and offsetting them to the inside of the panel. these lines together with the original horizontal panel lines form the horizontal plates that are clipped by the panel boundary and by any opening cutting through them. Having created the horizontal members, the curves of the panel boundary and their offset counterparts that are left over from the rest of the boundary members. These members are generated in the same manner however they are clipped by the horizontal plates to ensure that the horizontal plates run to the edge of the panel.

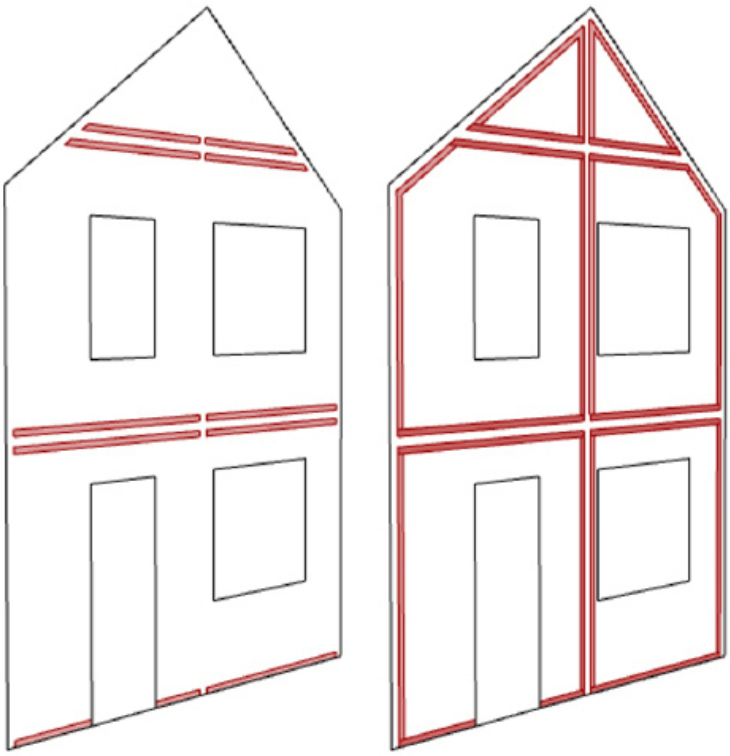


Image 41: the generation of the boundary members (Source: own image)

Boundary members - Workflow

The workflow used to generate these boundary members is shown below. It splits the panel contours in horizontal and non-horizontal parts and then transforms them into boundary plates using the method explained above.

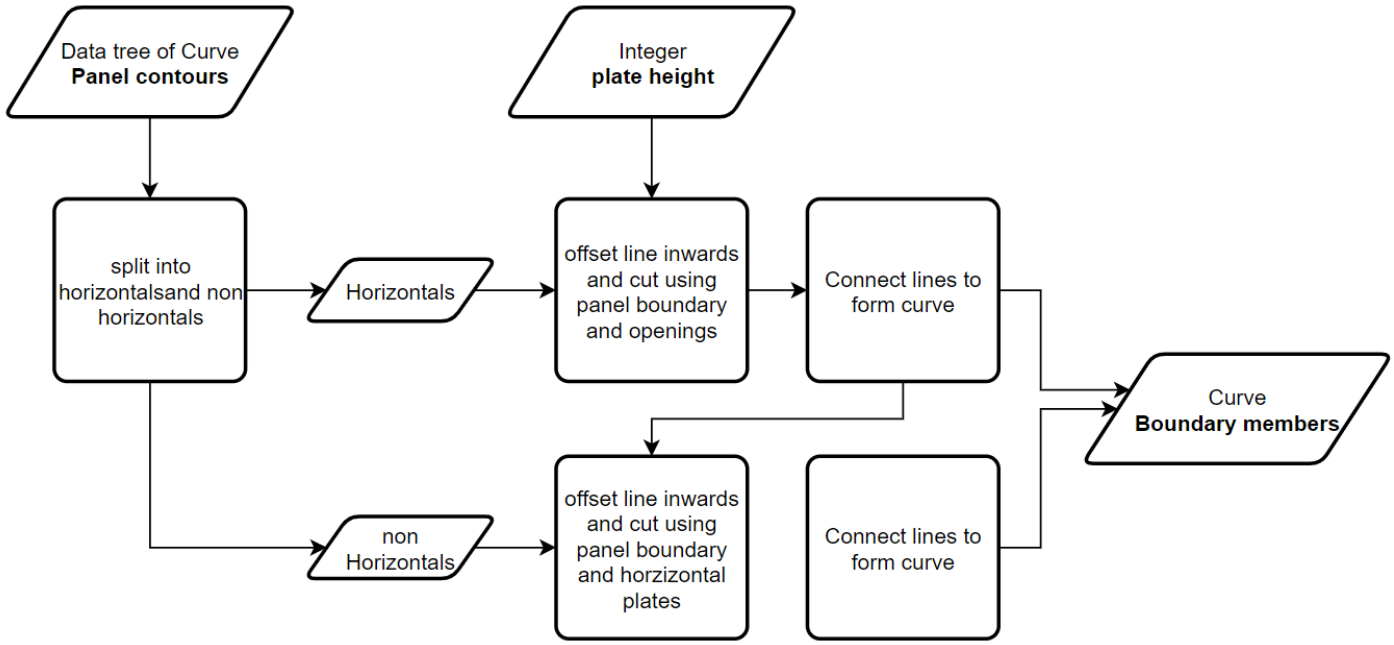


Image 42: workflow of the boundary member generating process (Source: own image)

Boundary members - Grasshopper

This workflow is then scripted into rhino. It starts off with a component that splits the boundaries in horizontal a non-horizontal part. The horizontal parts are turned into plate curves within the same component and the non-horizontal parts are taken to a second component which makes them fit between the horizontal parts. How this is done exactly with the python scripts can be seen in appendix 7.1.5.

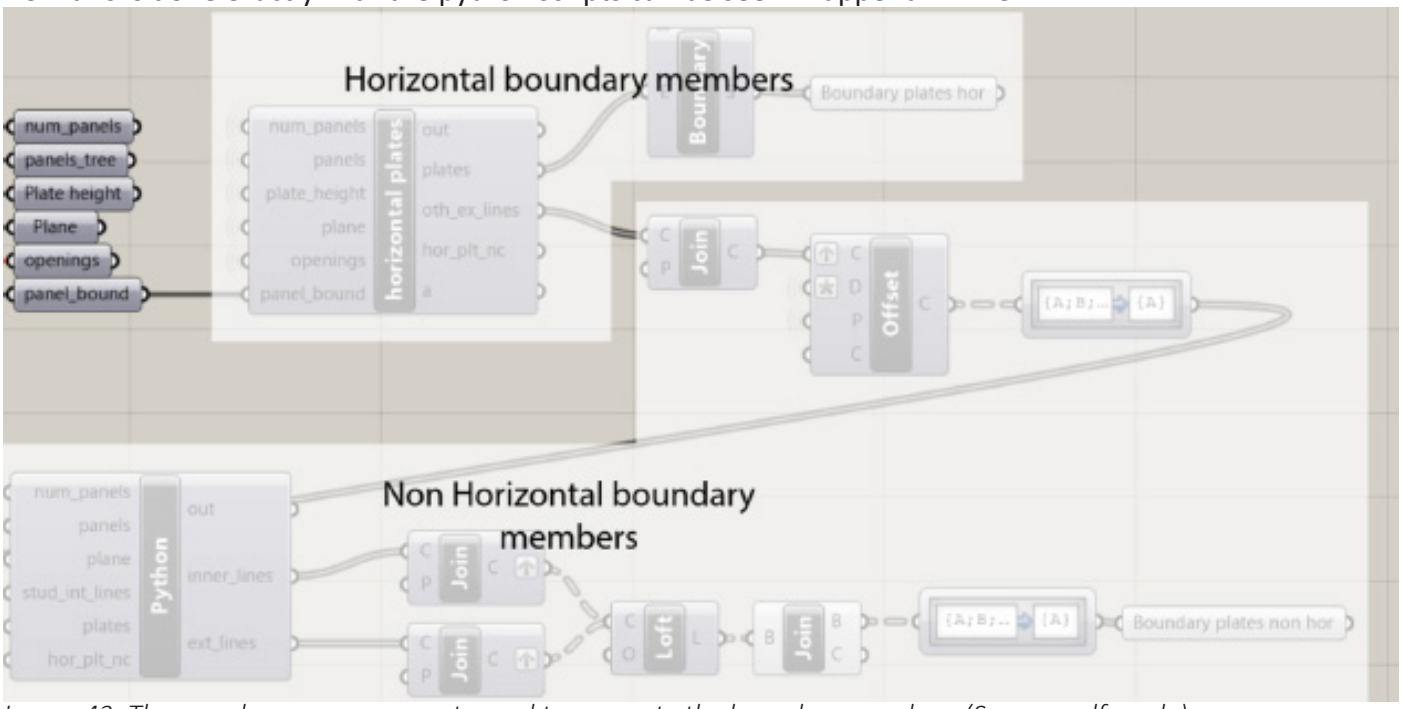


Image 43: The grasshopper components used to generate the boundary members (Source: self made)

Opening plates

Next to the boundary plates also the opening plates are generated. These are the plates that are at the top and bottom of every window opening. With an exception for the openings that are partially on the edge of the panel like the door opening in this case. It uses the same principles as the boundary members. But uses the horizontal curves of the openings as input.

The workflow used to generate this is shown below. It takes the horizontal lines of the opening contours, offsets them outward and checks if it is in panel.

This workflow diagram is translated into grasshopper using the components shown in the image at the bottom of the page. this includes a python component of which the script can be found in appendix 7.16.

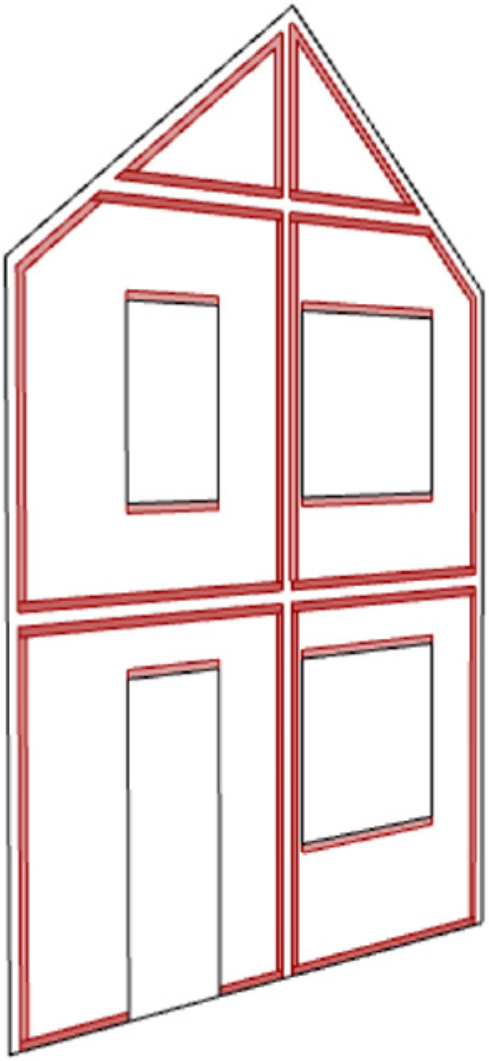


Image 44: Opening plates added to the framework members. (Source: own image)

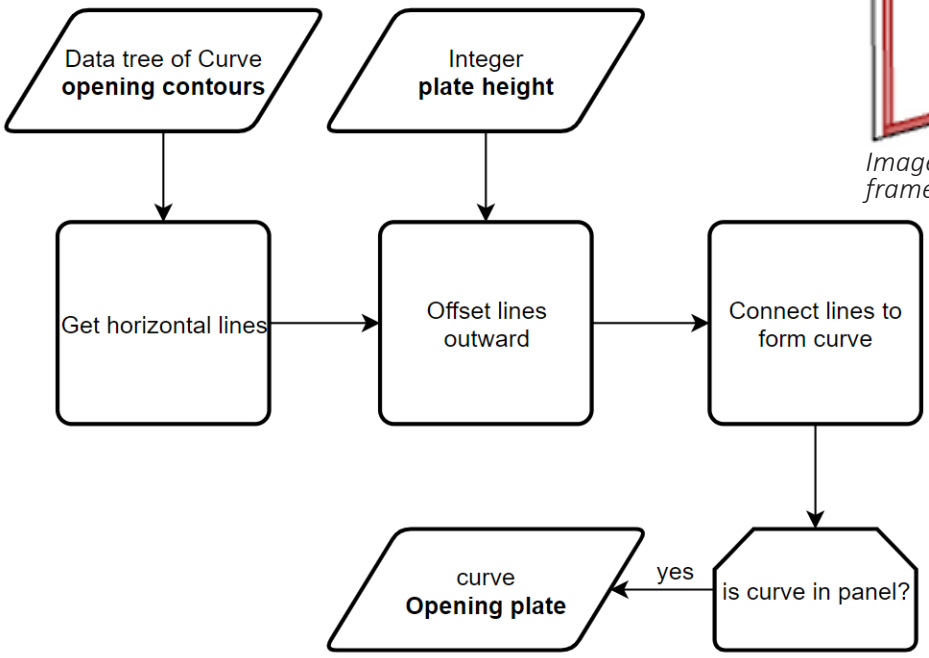


Image 45: Workflow of the generation of the window plates. (Source: own image)

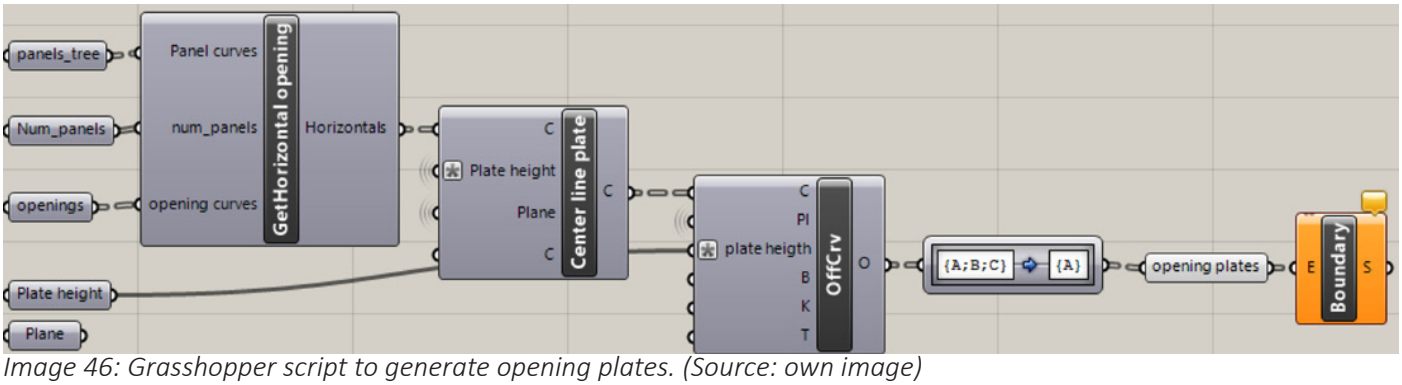


Image 46: Grasshopper script to generate opening plates. (Source: own image)

Studs - Theory

The studs are the most common members of the prefab panel and also the most complex part to place. It does not only need to take into account all the other members of the panel it also needs to take into account the openings, intervals between the members that cannot be too small and studs always need to be right next to an opening. This process is a bit similar to the generation of the panel contours. It generates X coordinates for the center lines of the studs it does so by starting at the boundary of the panel and adding the stud interval to this x coordinate. It then checks if the location of the stud is acceptable. For the stud to be in an acceptable location it needs to be A) within the panel. B) not within a given proximity of an opening as is set with a parameter to ensure there is enough space between the studs to apply insulation properly. As can be seen in image 46 the stud on the left side of the right window is move slightly left to ensure the minimum distance which in this case is 100mm is met. In order to achieve this, it checks if the coordinate is between the window edge ± 100mm and moves it accordingly while also placing the stud at the window edge. C) if the previous coordinate is before a window edge and the next one is after a window edge (but both fall out of the range if ± 100mm) instead the stud next to the window is placed to ensure that most studs have an interval of in this case 600mm. (the blue dashed line in the image below) D) if the last stud is too close to the boundary member it is also placed back with the 100mm parameter.

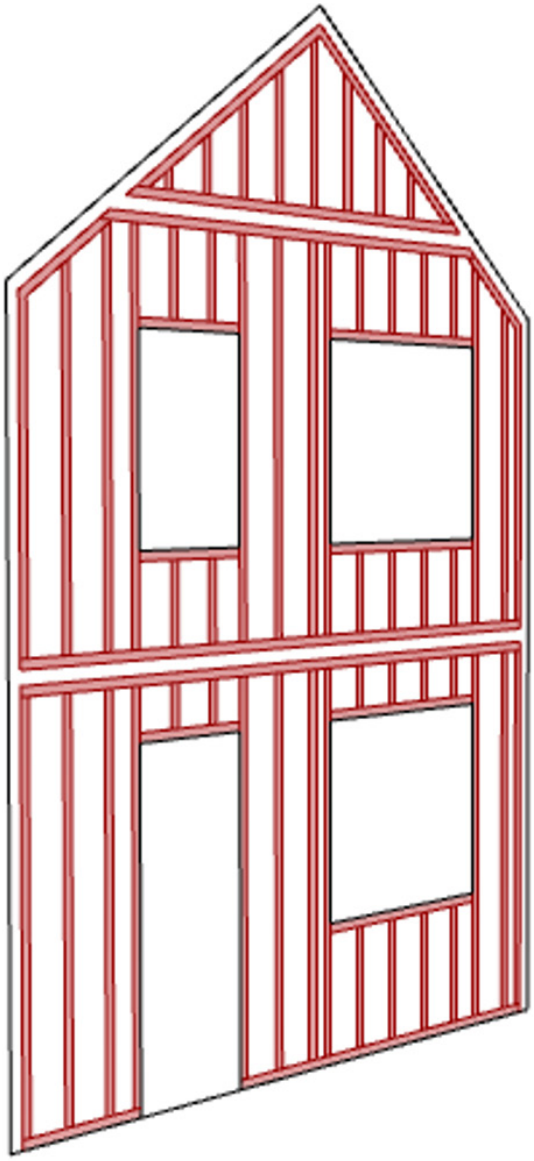


Image 47: Studs added to the framework members. (Source: own image)

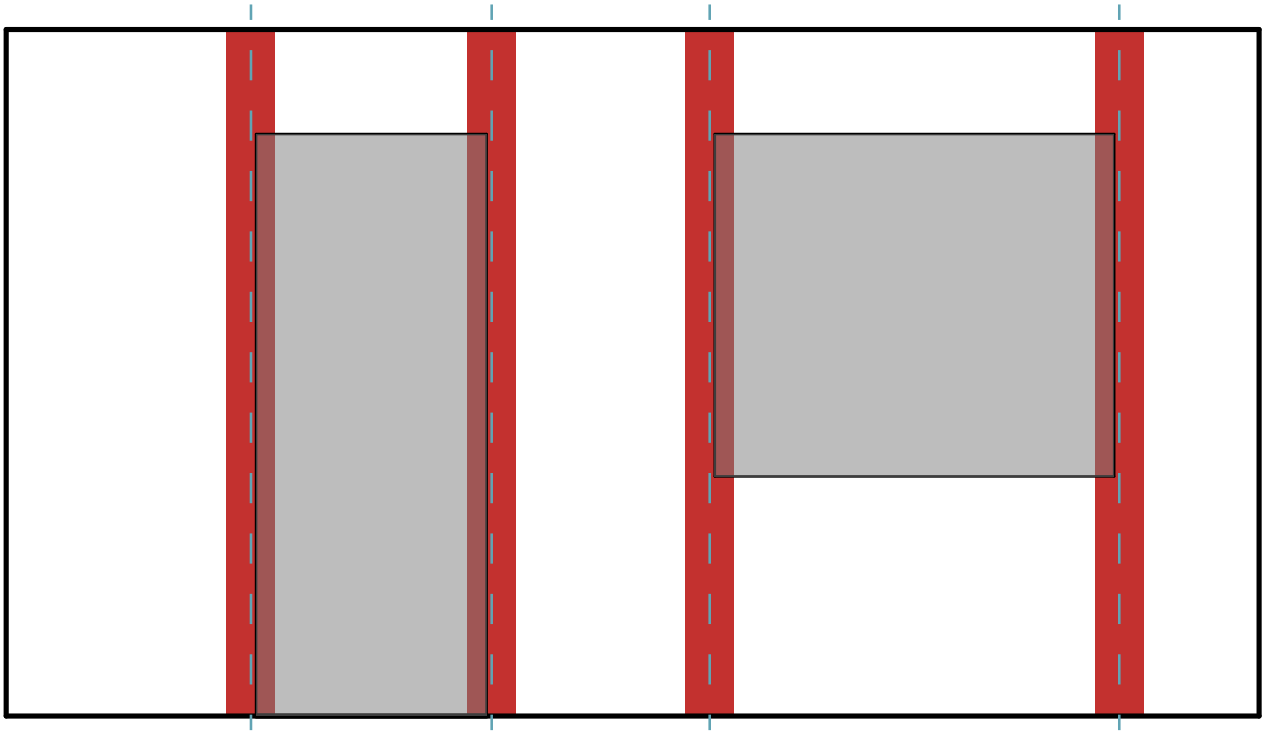


Image 48: Illustration of there no studs can be placed (red) and where a stud has to be placed (blue dashed line) (Source: own image)

Studs - Theory

The workflow of the method which is described at the previous page is shown here. As stated, a list of stud coordinates is generated within the domain of the panel. This while taking into account the provided minimum proximity to the openings. These coordinates are used to generate the stud outlines which are then cut by the plates and openings leaving studs that are all in the intended position and with the desired length as shown in image 46.

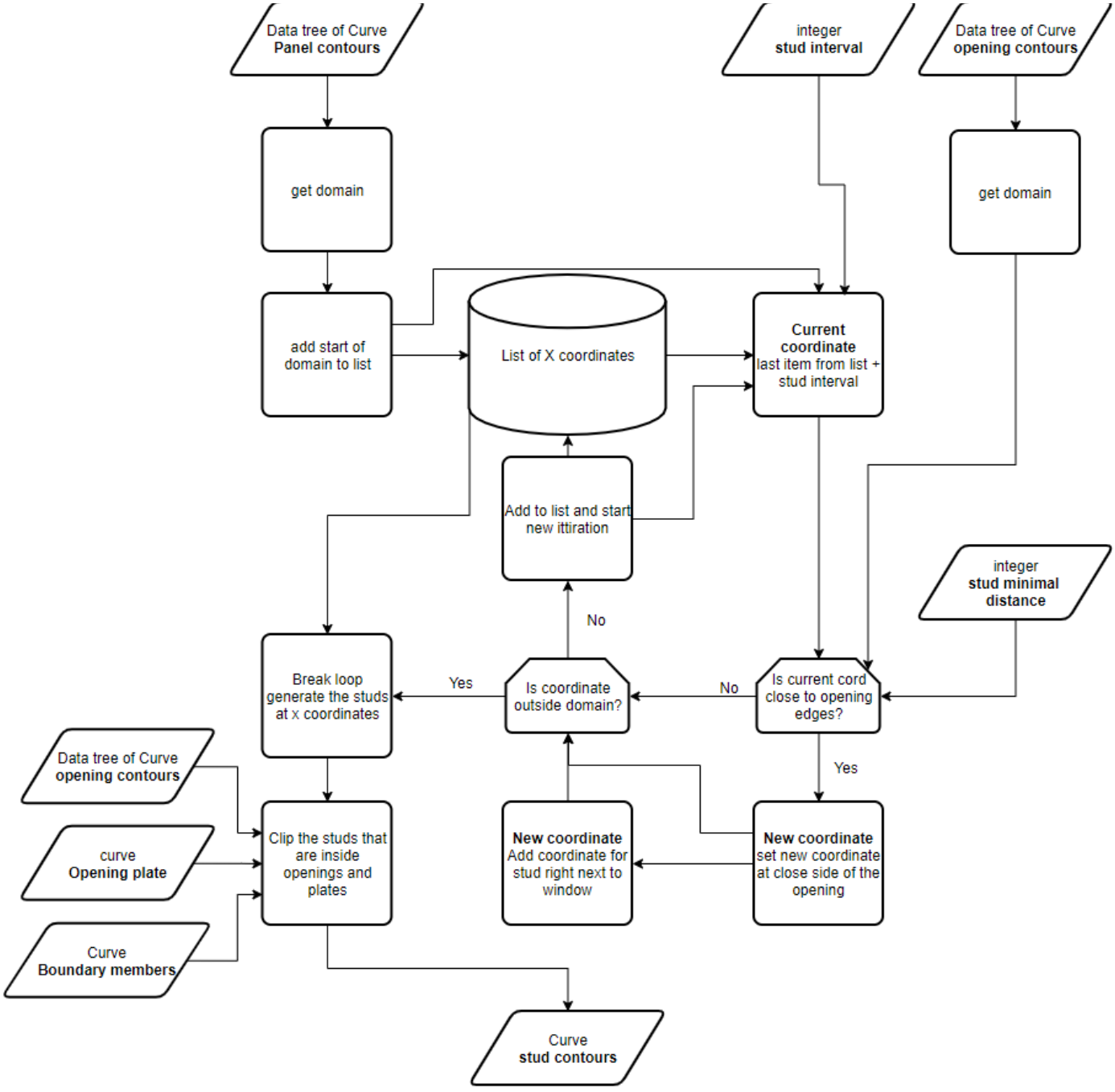


Image 49: Workflow diagram of the generation of the studs(Source: own image)

Studs - Grasshopper

The workflow diagram shown on the previous page has been translated into grasshopper using the components below. The top part takes the exterior lines of the panels. The bottom part then generates the studs and clips them to the panels (this only happens if the panels are not rectangular). More detail about this script and the python components used can be found in appendix 7.1.7.

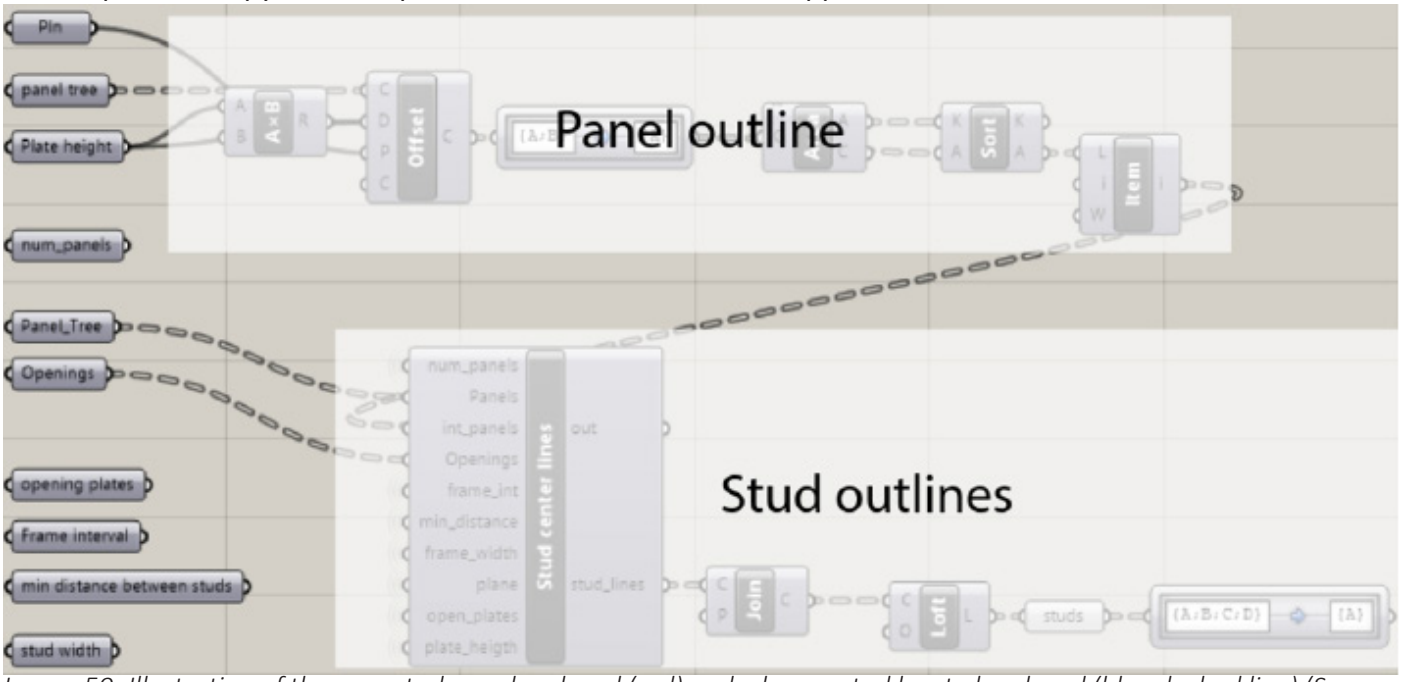


Image 50: Illustration of there no studs can be placed (red) and where a stud has to be placed (blue dashed line) (Source: own image)

Insulation generation - Theory

The generation of the insulation is similar to the normal generation of panels. A surface is constructed from the panel and the openings are cut out. However, after the openings also the wooden members are cut from the surface, in this way only the to be insulated surfaces remain.

Insulation generation - Workflow

To generate the insulation the workflow shown at the bottom of the page is used. First the surface of the panels is generated and then the surface of the already generated elements (studs, boundary members, plates and the openings) is subtracted leaving only the insulation

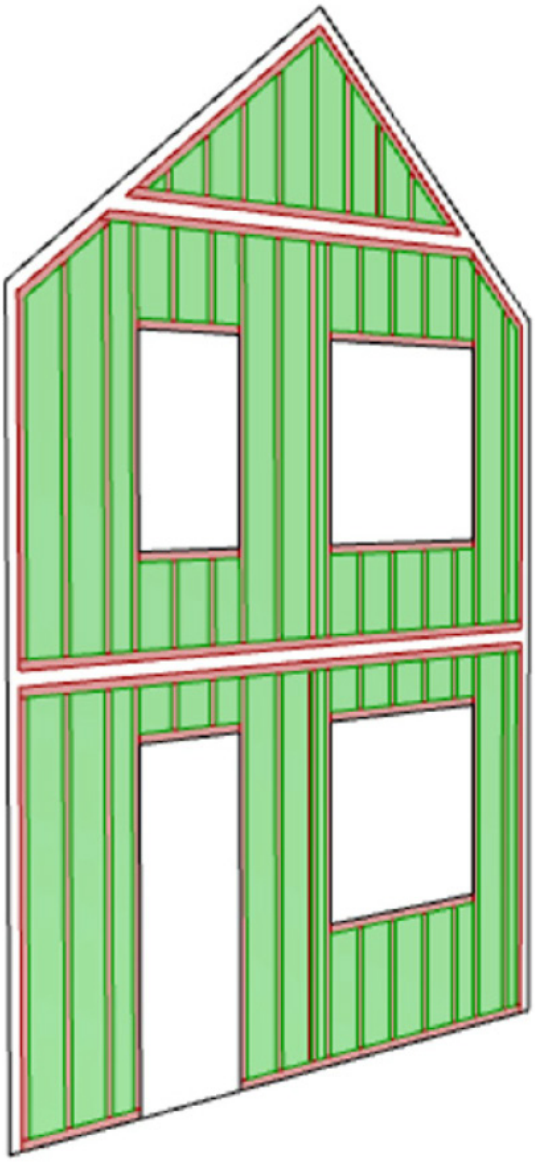


Image 51: Insulation added in between the framework members. (Source: own image)

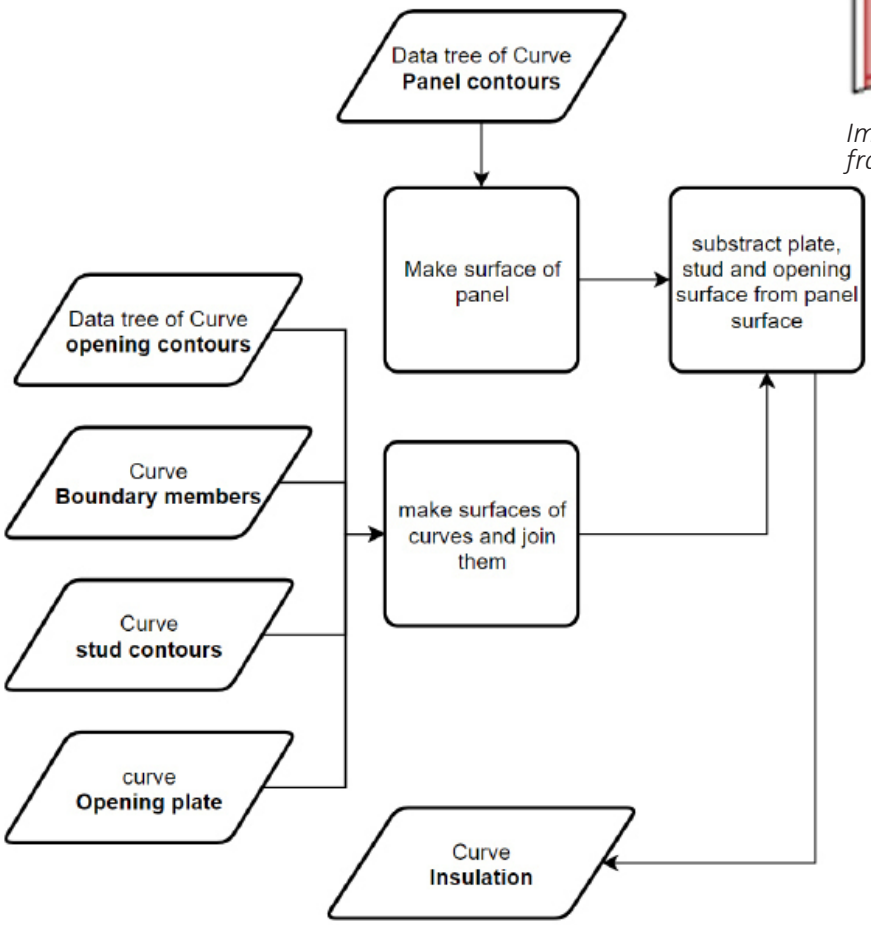


Image 52: Workflow illustrating the generation of the insulation within the framework layer (Source: self-made)

Insulation generation - Grasshopper

To generate the insulation the surfaces of the wooden members and openings are joined (cutting objects) and subtracted from the panels surface. In this way the only surfaces left are the surfaces where insulation needs to be applied. The grasshopper components to do this are shown below.

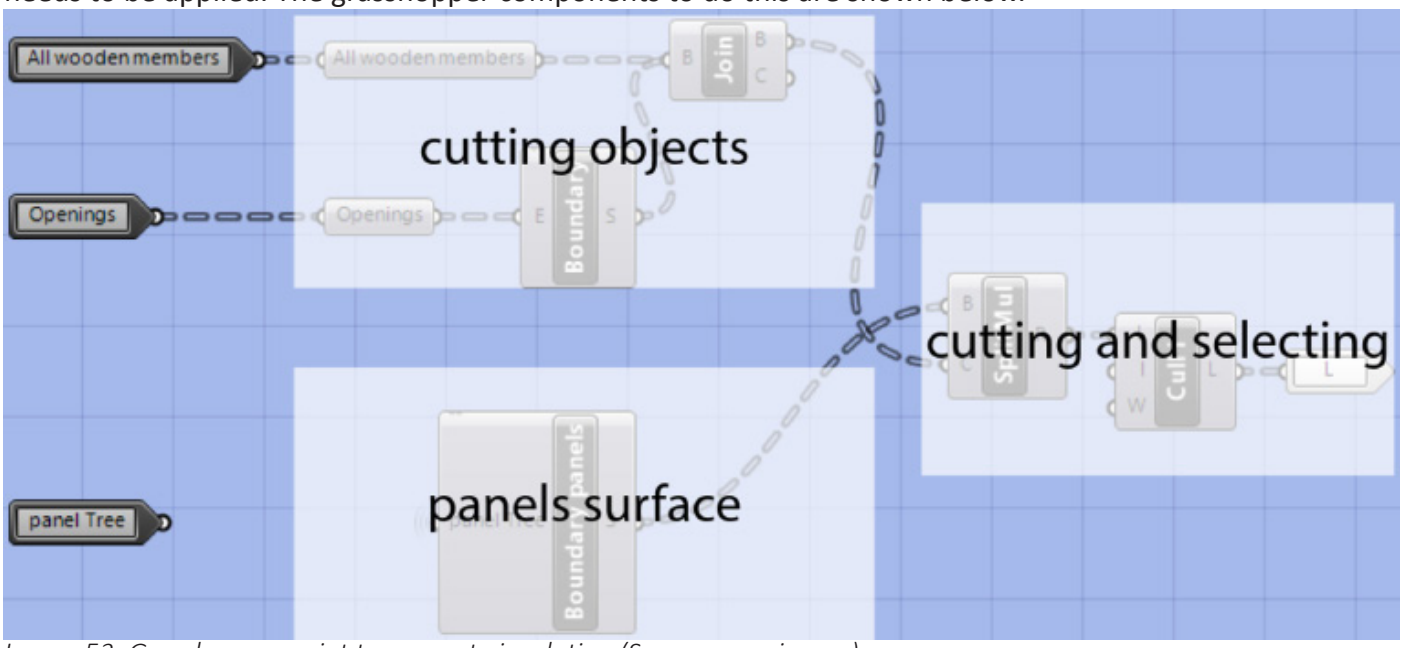


Image 53: Grasshopper script to generate insulation (Source: own image)

Extruding and moving

After all the surfaces have been generated, they are moved to their position in the panel (depending on the layers that are in front of them) and extruded. This process is identical to the generation of the normal solid layers. As can be seen in the workflow diagram of this process which is shown at the bottom of the page.

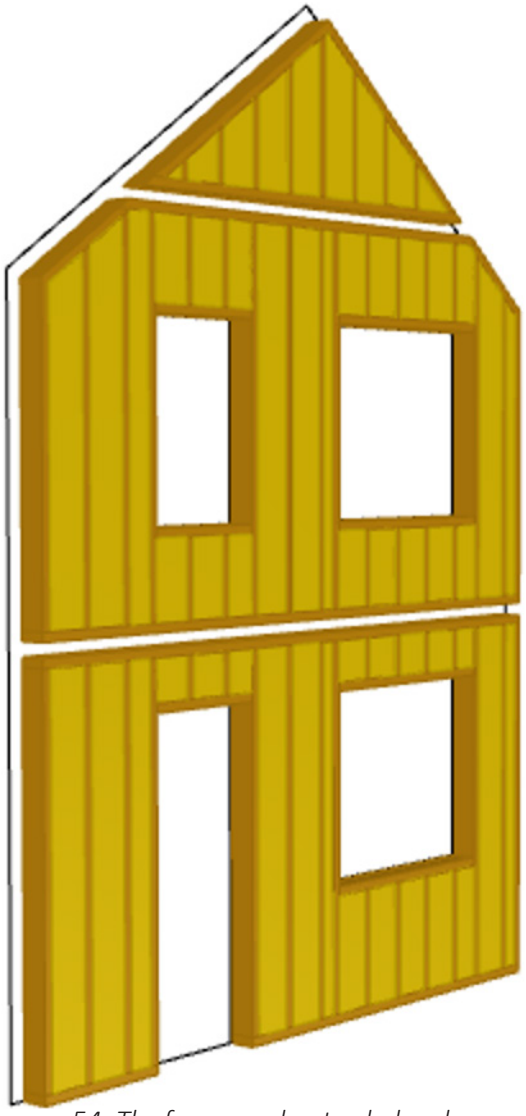


Image 54: The framework extruded and moved to the correct position. (Source: own image)

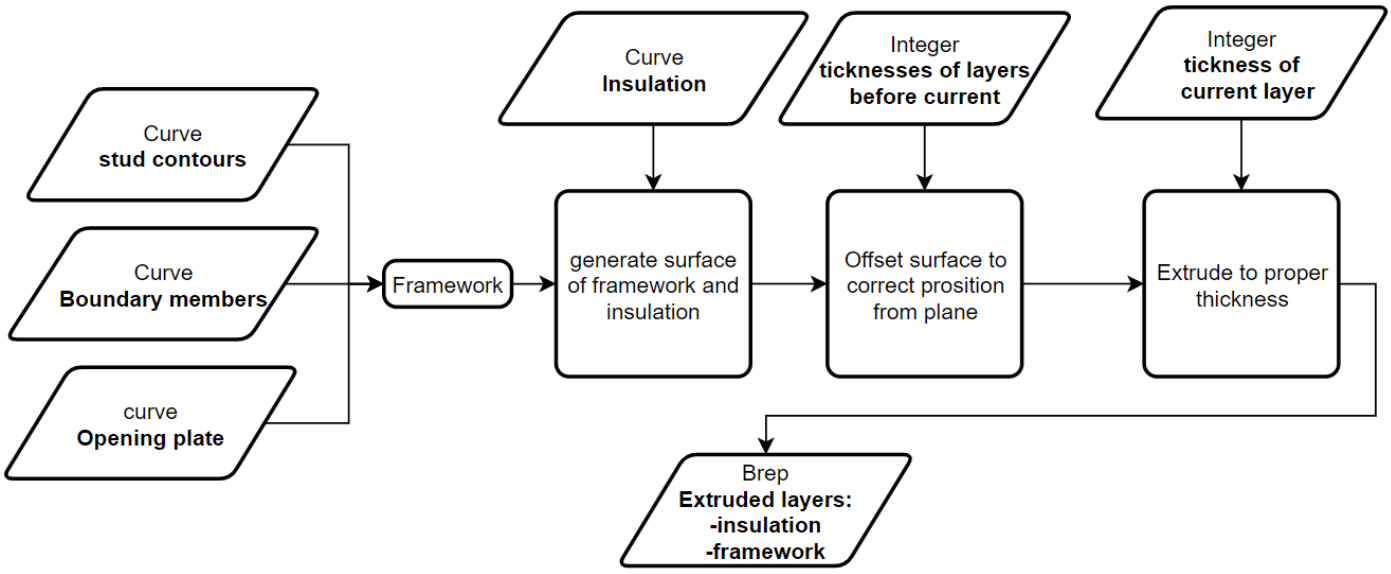


Image 55: Workflow diagram of the extrusion of framework and insulation layer. (Source: own image)

3.2.2.4. Panel connections

In order to create the indentation between the panels a separate component is needed. As these are created outside the panels in the intermediate space.

Input

- *Structured panels*
  A tree structure with all panels in a separate branch in order to select the top and bottom line
- *Plane*
  Plane in which the façade is orientated
- *Plate height*
  Integer in which the height of the plate members is set
- *Stud interval*
  Integer in which the spacing between studs is set
- *Stud width*
  Integer in which the height of the stud members is set
- *Min distance between studs*
  Integer in which the minimum distance between studs is set, used to check if studs are not positioned to close to studs at the end of the façade.
- *Layer thickness*
  Integer in which the thickness of the framework layer is described. Of which the tolerance between the indentation will be subtracted and the integer will be halved.
- *Thickness layers before*
  Integer in which the thicknesses of all layers before the framework are added. So that the framework can be positioned in the proper location.

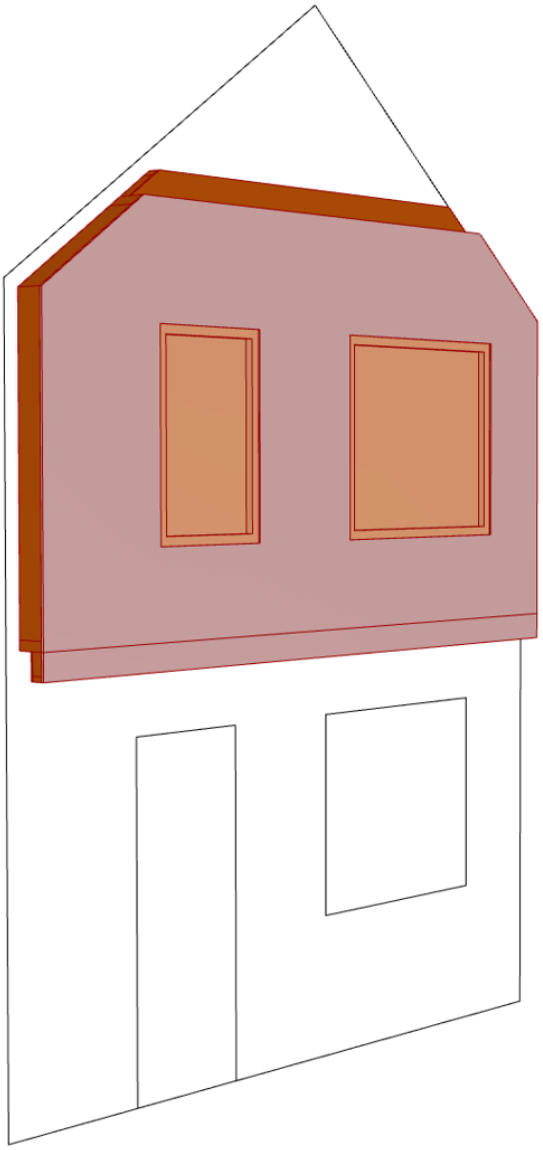


Image 56: A façade panel showing the indentation designed (Source: own image)

Process

The process starts off with identifying the top and bottom lines of the panels that are connected to other panels. For the top lines the backwards indentation is then created with the same materials as the board and framework. The dimensions are the same as the original framework only the thickness is halved and tolerance is taken into account. The same is done for the front indentations. The layers are then moved to the right position and extruded.

Panel connections - Workflow

The workflow of the generation of these indentations or panel connections is shown below, it relies on the same logic to generate plates and studs as is done by their respective components as described before only in this case it does not need to take into account any openings and thus the studs are all spaces with the desired interval requiring less wood and less work to get the insulation in.

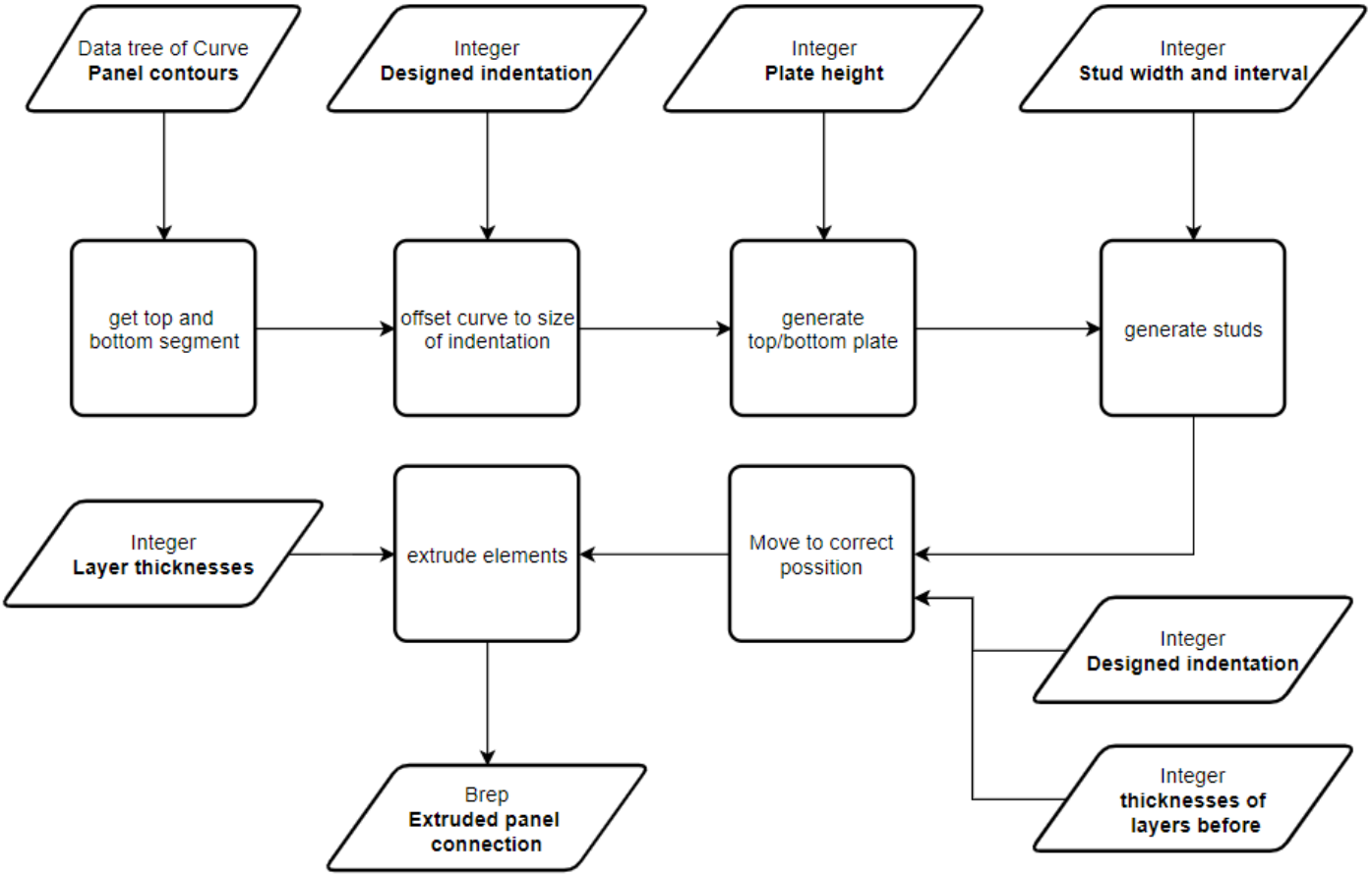


Image 57: Workflow diagram showing the process designed for the generation of the connections between the panels. (Source: own image)

Panel connections - Grasshopper

This workflow has been translated into grasshopper as shown below. First a python component generates the framework members and their boundary. Then these are used to generate the insulation and finishing. After which all the layers are moved in place. More detail about this can be found in appendix 7.1.8.

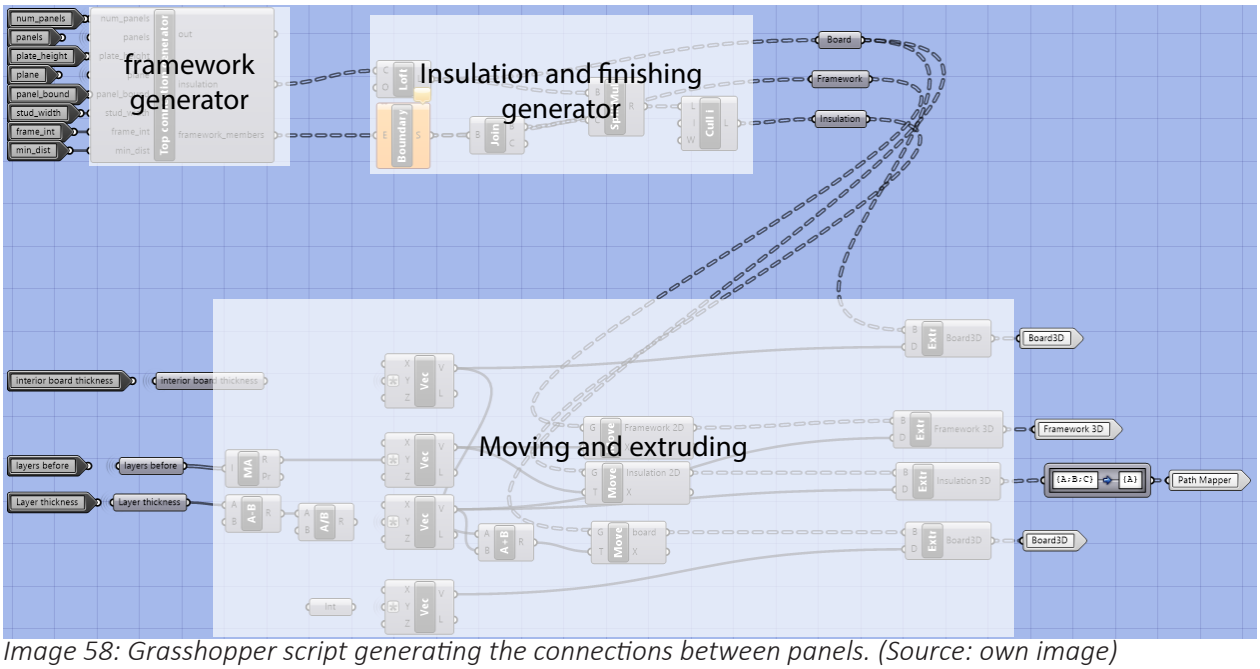


Image 58: Grasshopper script generating the connections between panels. (Source: own image)

3.2.2.5. RC-calculator

To ensure the wall panels have a sufficient thermal resistance the RC value is calculated within the tool.

Input

Stage one

- *Insulation thickness*
  Integer containing the thickness of the insulation
- *Lambda of insulation*
  Integer containing the lambda value of the insulation

Stage two

- *Framework thickness*
  Integer containing the thickness of the insulation layer
- *Lambda of framework elements*
  Integers containing the lambda values of the wooden members and insulation
- *Area of frame members*
  Area of the wooden members and insulation to determine the mean RC-value

Process

The calculations are done in two stages; the first stage is being calculated before the process of generating the framework for the panels. It does so by dividing the thickness of the layer by the lambda value of the specific material which is provided by the user. Which is the way to calculate Rc values for solid constructions. (Hagentoft, 2003) This calculation gives an estimate of the upper boundary of the RC value of the panel. As the addition of more wooden framework members (which have a lower lambda) reduces the Rc value of the panels. As the percentage of wooden members is different for every panel due to its layout the lowest value is selected and fed back to the engineer so that he can decide if it is sufficient. The image below shows a panel within grasshopper giving the feedback to its user about the two calculated Rc values. The scripts used can be found in appendix 7.1.9.

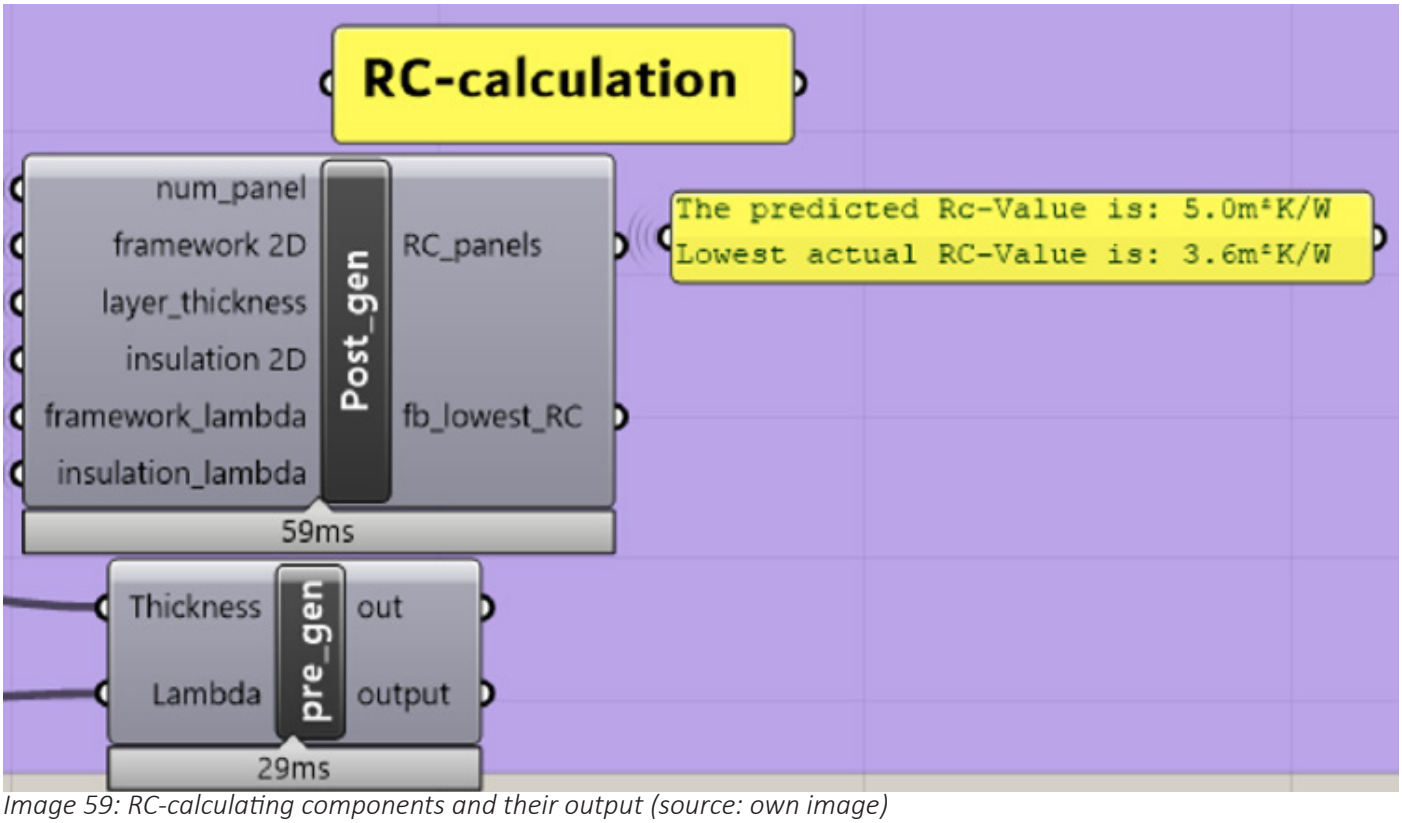


Image 59: RC-calculating components and their output (source: own image)

3.2.2.6. Weight calculator

To enable the user of the tool to assess how much mounting points the panel needs (depending on the weight and the strength of the supporting structure) the weight of the panels is calculated.

Input

- *Density per material*
  Integer containing the density of the materials used
- *Volume per material*
  Volume of the materials in the panels. Gathered from the extruded elements

Process

In order to calculate the weight of the panels all their extruded elements are taken and multiplied by the by the user provided density. As with all the other elements in the computational script the weight is separated in a branch per panel. A mass addition provides a simple overview of the weight per panel as shown in the image below.

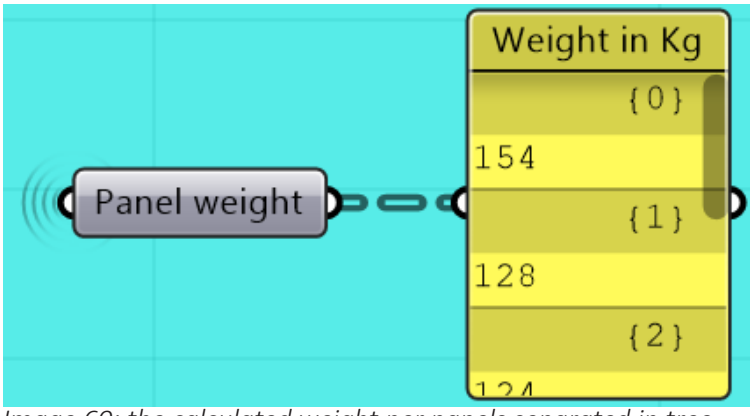


Image 60: the calculated weight per panels separated in tree branches (Source: own image)

3.2.2.7. Window placement

The windows placed in prefab elements are not created together with the panel they are generally a complete component that is placed inside the panels. For this reason, they are not generated in the script. They are however moved to the proper location within the panel so that they can be fixed in that location upon the manufacturing of the panel.

Input

- *Openings*
  The opening curves where the windows need to be placed. Which will also need to be assigned a window tag by the used.
- *Frame tags*
  Frame tag per different Frame type so that the script can refer to the different frames and place them accordingly.
- *Weight per window*
  Weight per frame so that it might be added to the total panel weight.
- *Offset*
  Offset of the plane to place the windows in the correct depth in the panel.

Process

This placement is done by the user assigning window tags to openings and providing the information per unique tag such as their geometry and the weight. The script then takes the geometries and moves them from there reference corner point to the reference corner point of the opening. The placed frames can be offset to the position desired by the design.

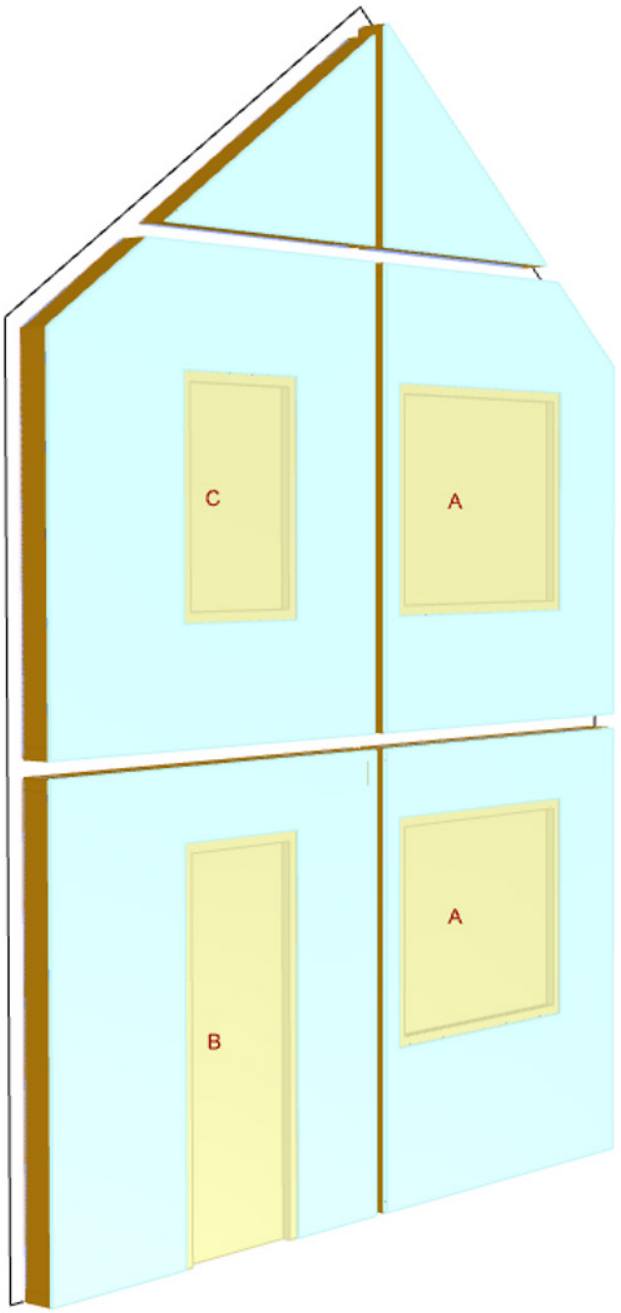


Image 61: Placed windows and their tags (Source: own image)

Window placement - Workflow

The process described on the last page is visualized by the work flow below. The user provides a list of tags that correspond to a certain opening. The component then cross reference the tags. If there is a match the Brep corresponding to the tag is moved to the location of the window.

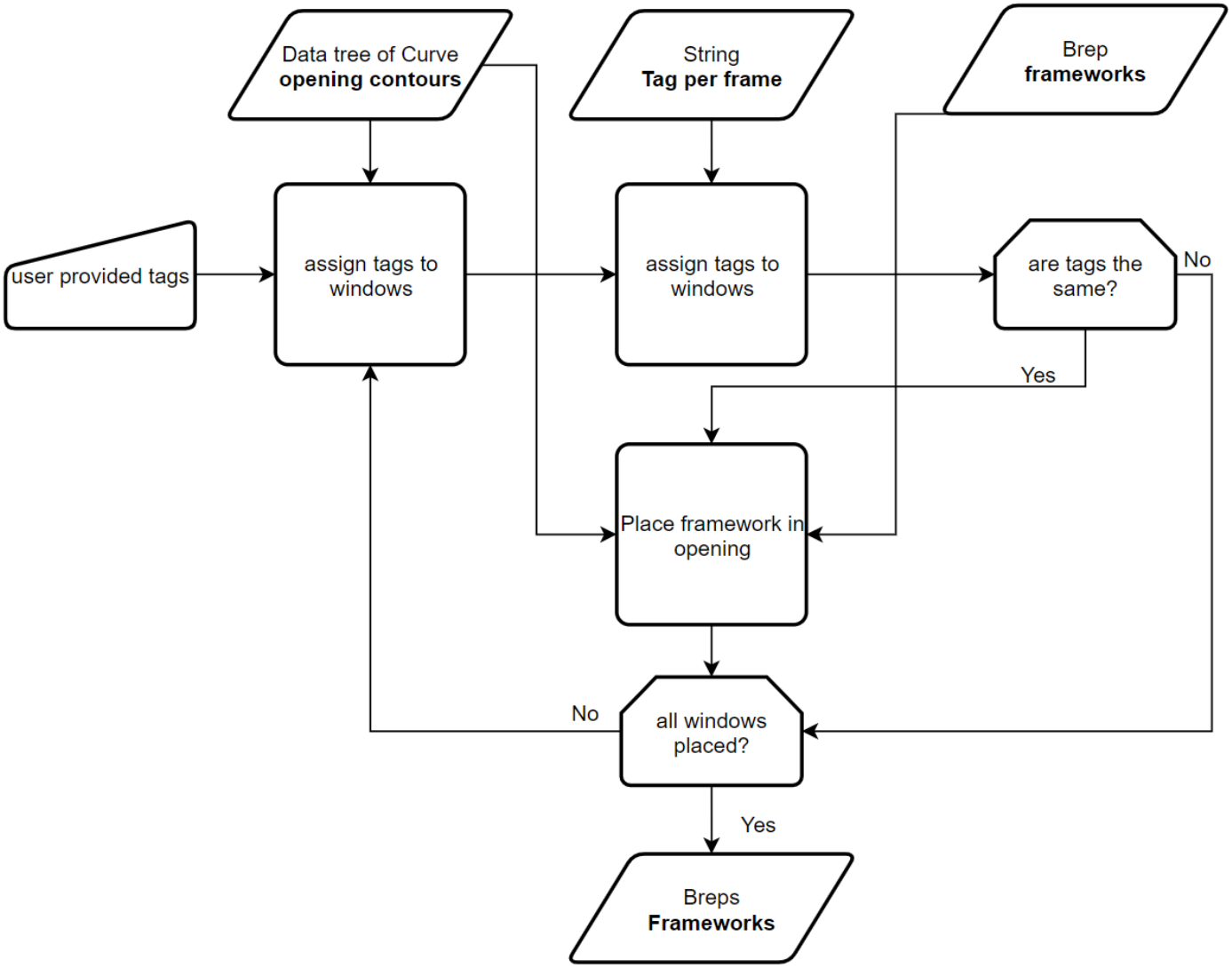


Image 62: Workflow diagram of the window placement component

Window placement - Grasshopper

The workflow shown on the previous page is translated into grasshopper with the script below. First the tags are assigned by the user and registered to their corresponding window using a python component. The provided geometries and weights with the windows and their tags are processed and placed to the correct opening by the window placer. More detail about how these components work and their python scrip can be found in appendix 7.1.10.

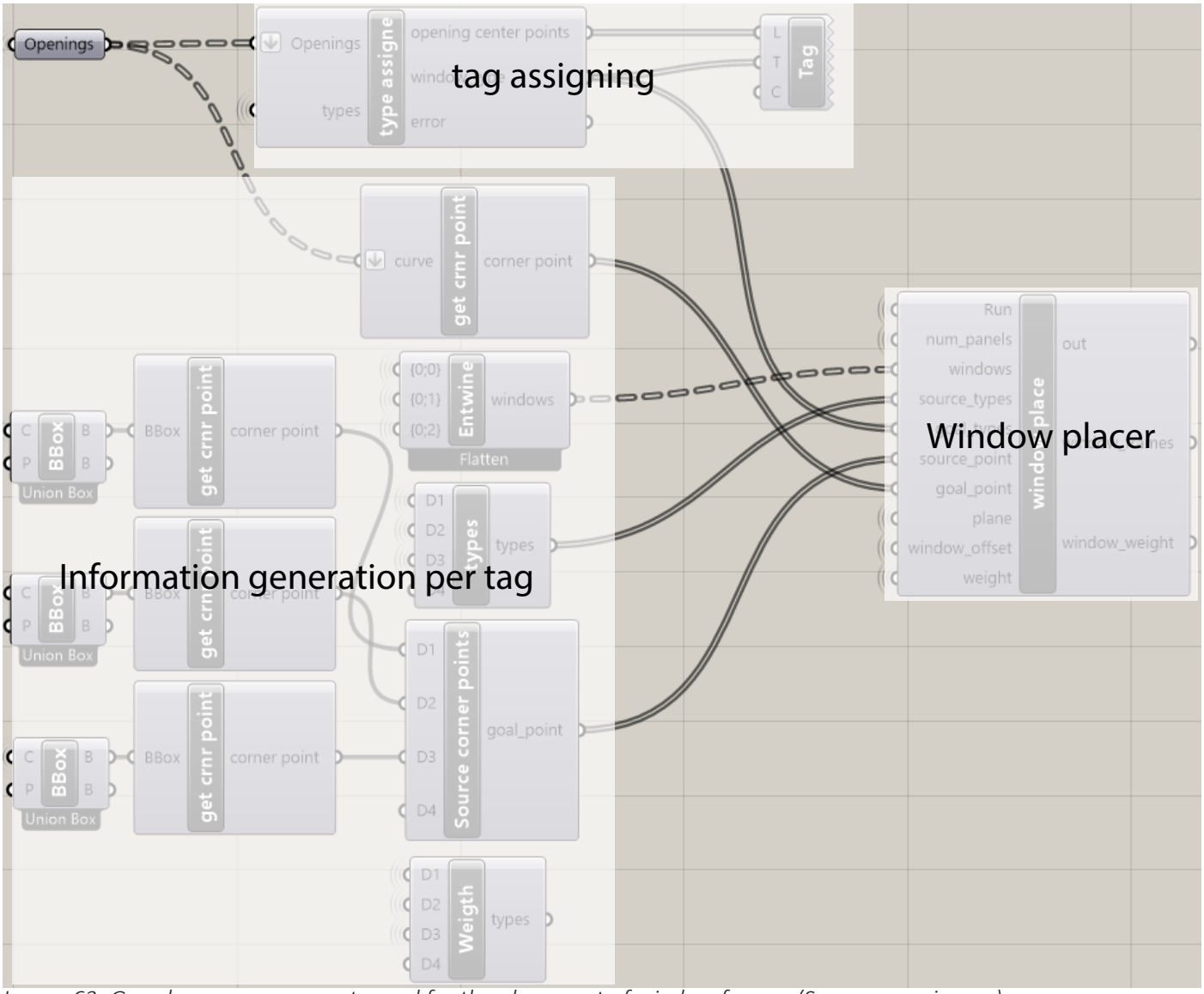


Image 63: Grasshopper components used for the placement of window frames (Source: own image)

3.2.2.8. Conclusion

Those were all the individual parts that together form the computational method that allows the user to generate wall panels. The scheme on the next page shows the integration of all the individual parts of the method in the big picture as well as there in and output.

The inputs the user provides to the tool can be split in numerical and non-numerical inputs. The numerical inputs such as wooden member sizing's allows the user to change the size of all components in the panel. The non-numerical inputs, such as the boundary contours of the façade allows the user control by which he can dictate exactly where the panels should be generated.
The method always outputs the geometry, weight and RC-values of every panel. This method will be tested in chapter 3.3. on a case study

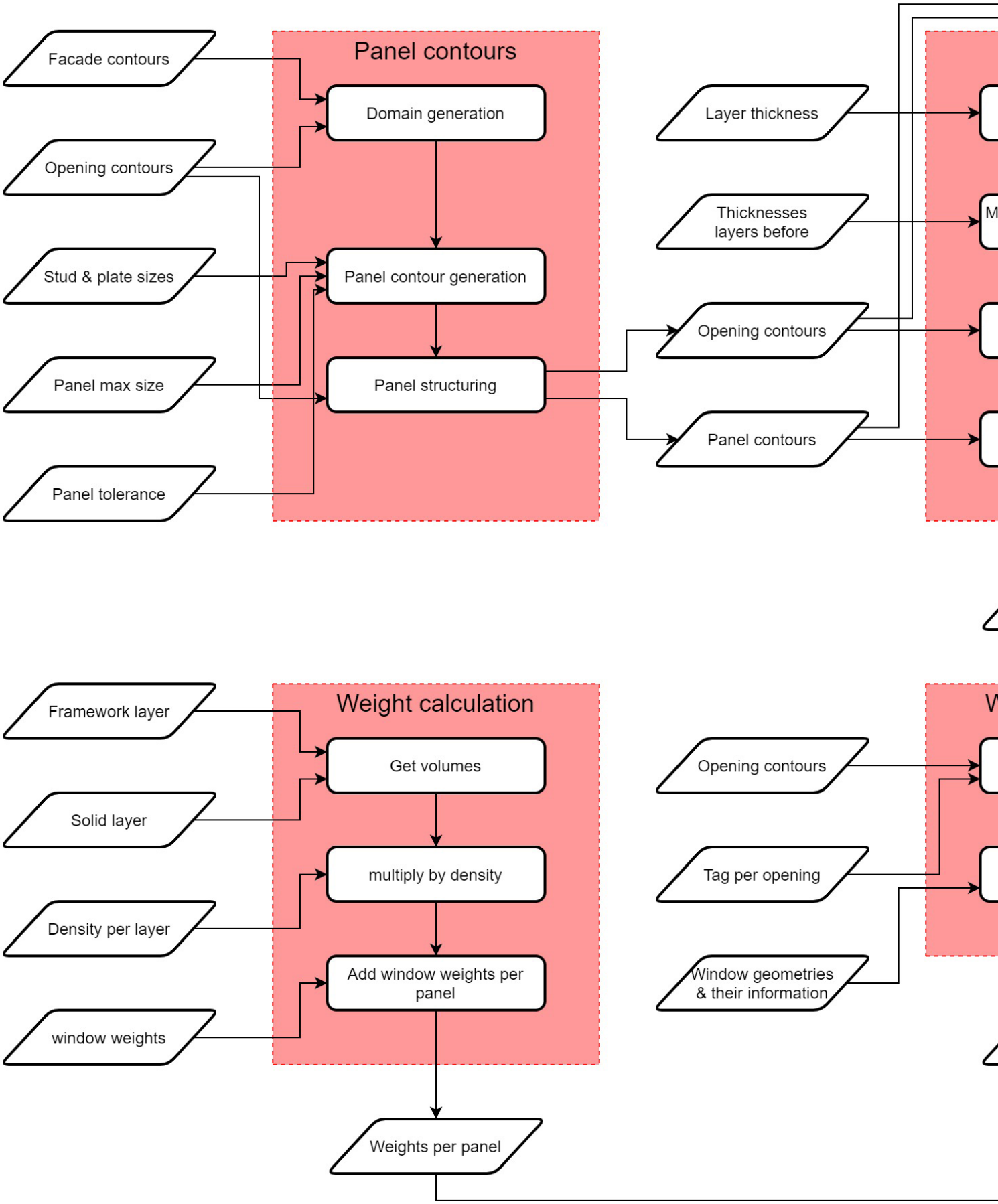
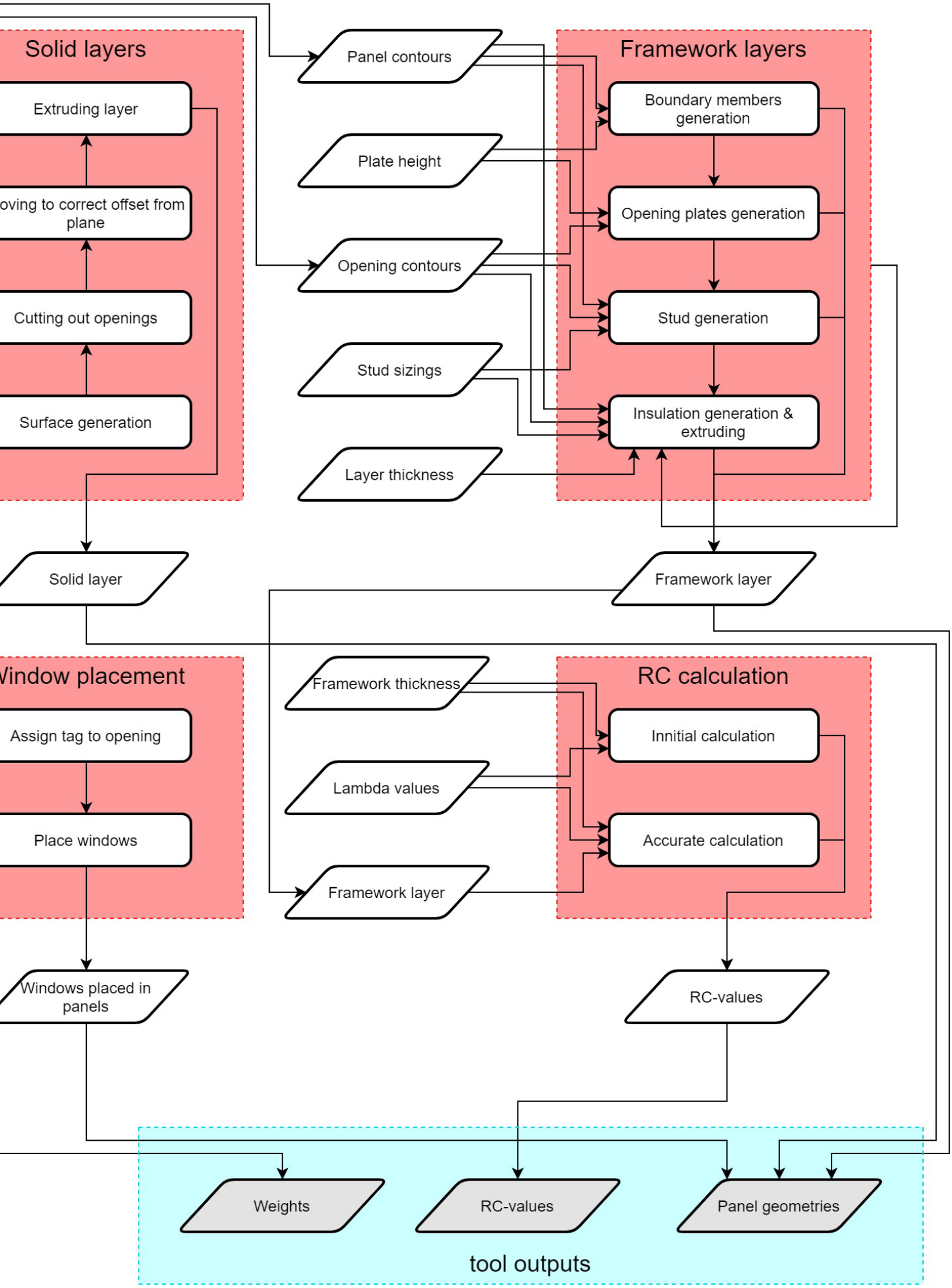


Image 64: Overview of the process gone through in the wall generation tool (Source: self-made)


Solid layers
Extruding layer
oving to correct offset from plane
Cutting out openings
Surface generation
Solid layer
Panel contours
Plate height
Opening contours
Stud sizings
Layer thickness
Framework layers
Boundary members generation
Opening plates generation
Stud generation
Insulation generation & extruding
Framework layer
Window placement
Assign tag to opening
Place windows
Windows placed in panels
Framework thickness
Lambda values
Framework layer
RC calculation
Innitial calculation
Accurate calculation
RC-values
Weights
RC-values
Panel geometries
tool outputs

3.2.3. Roof generation tool

The roof generation tool is similar to the wall generation tool at the basis. It takes the same steps in the generation of its components, however it does so at an angle which needs a different approach then used in the wall generation tool. Non the less it is based on the same input, and methodical steps. With the addition of the roof specific elements such as the wall beam and the laths. When gone through this process the outcome will still be a 3D model with the information about the panels in terms of weight and RC-value. At the start of this process the user can decide if he wants to provide the panel contours or if the tool should generate them according to the roof size and the maximum panel sizes. After this the layers for the panels are generated including the wall beam and the laths. Lastly the calculations and window placement are executed to finalize scheme.

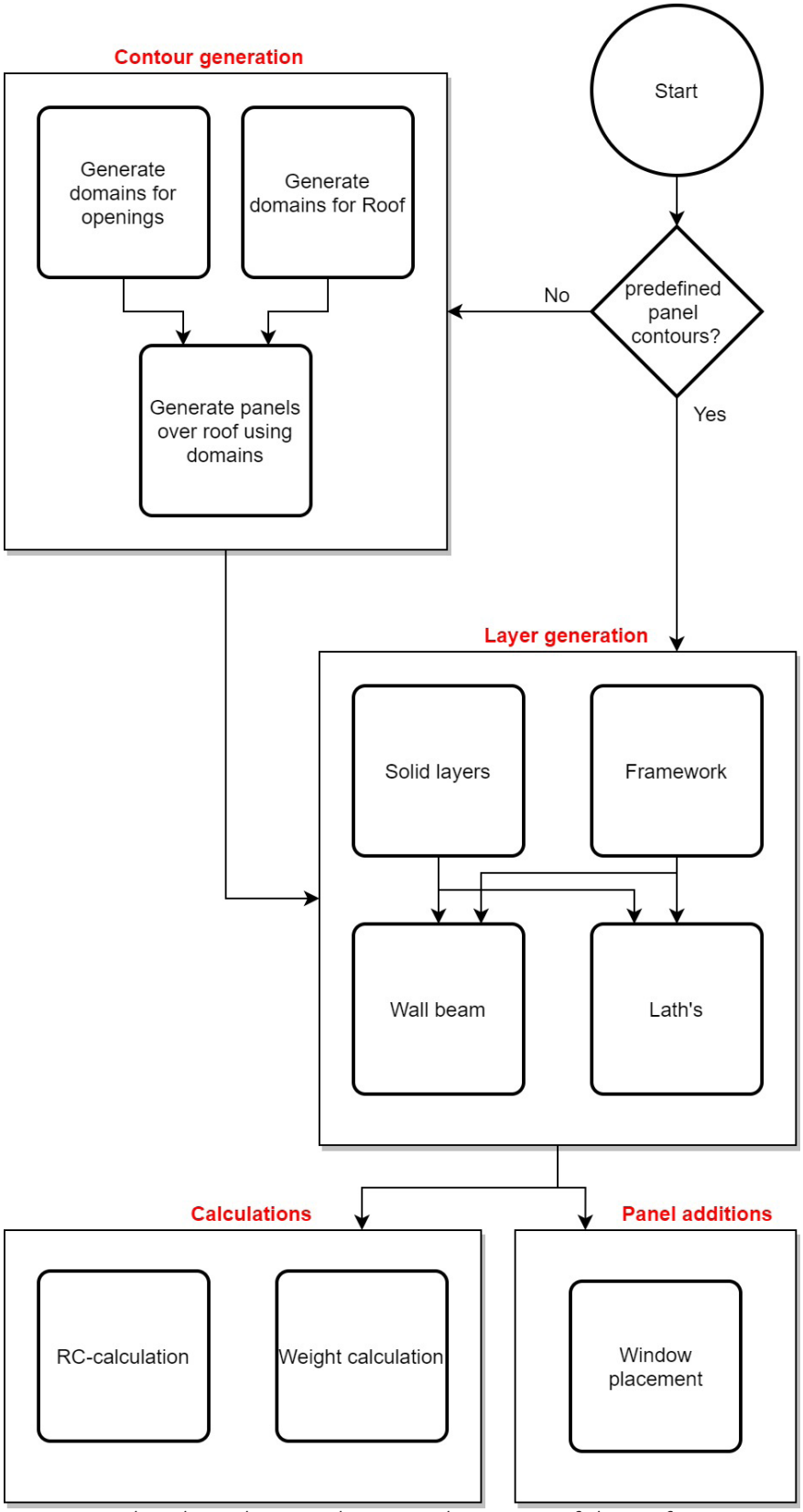


Image 65: Flowchart showing the general overview of the roof generation tool (Source: own image)

To execute the method the grasshopper script shown below is used. The groups on the left side are the inputs for the components of the script. The components themselves in purple all preform a small task in generating the roof panels and provide output to the cyan colored groups. How they perform these tasks and what information is there output will be explained in detail in the rest of this chapter. The grasshopper script can be found in appendix 7.2.

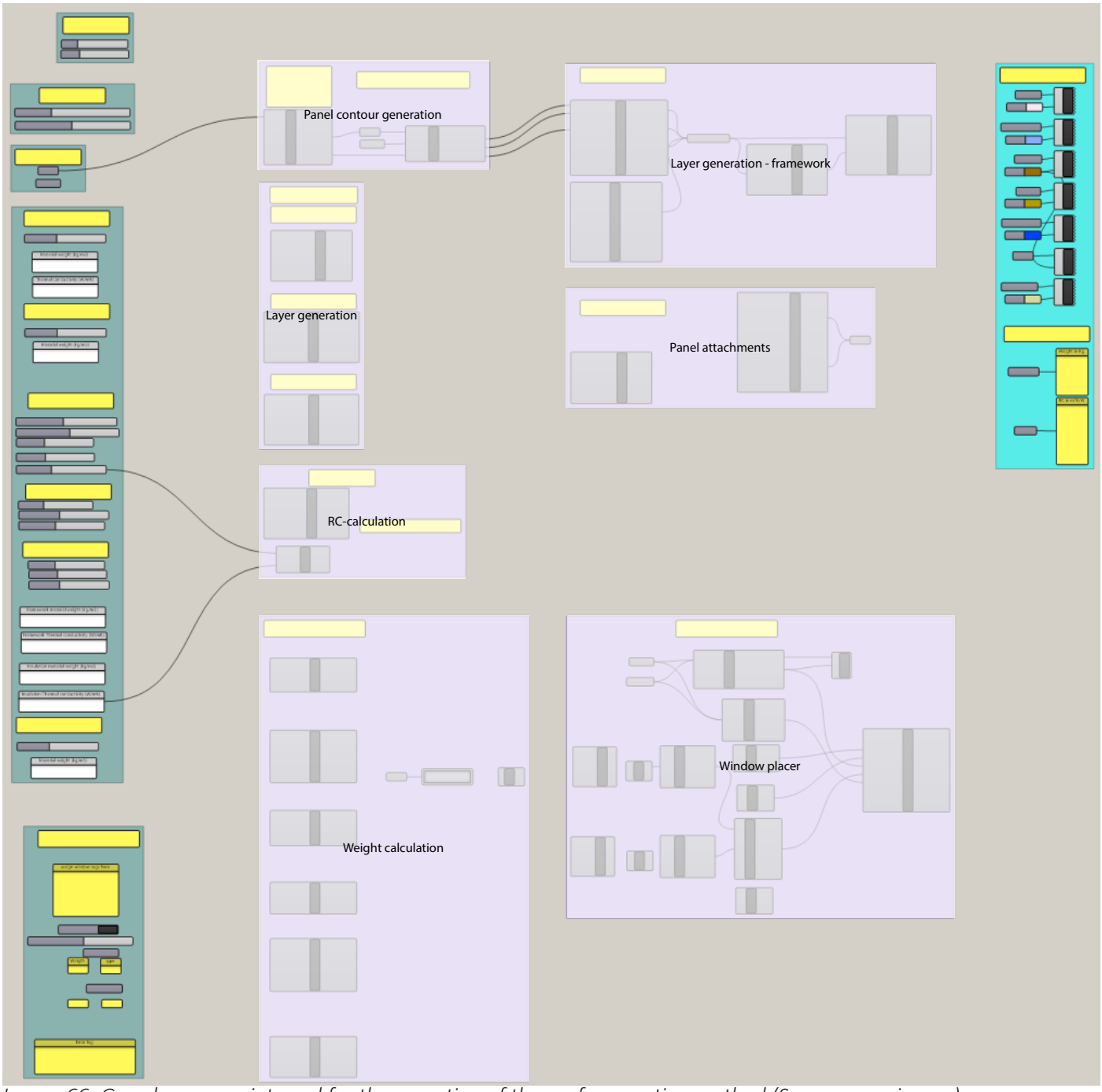


Image 66: Grasshopper script used for the execution of the roof generation method (Source: own image)

3.2.3.1. Contour generation

In order to generate the panel and their layers first the contours need to be determined. This can be done by the user of the tool so they can have full control and skip this step. They would then provide the contours to the next component of the script. But they can also be generated by the tool. In that case the used provides the maximum panel sizing and the tool will make a layout for the roof.

Input

- *Roof curve*
  This input provides the boundary in which the panels have to be placed.
- *Tolerance at edge*
  The edge tolerance implies the minimum distance the edge of the panel should have to de edge of the roof.
- *Tolerance between panels*
  This tolerance implies the minimum distance every panel edge should have from the next panel edge
- *Panel width*
  Sets the limit for the maximum panel width. Panel height is not a parameter as the panels will need to span from the eaves to the ridge.

Process

The generation of the panel contours exists out of two steps. The first step is the generation itself. The second step is the structuring of the data for future use in the tool. This includes assigning panels to tree branches (a data structuring system within grasshopper) and assigning openings to the same branch.

Contour generation - Theory

The generation of these panel contours is done similar but more simplified than the wall generation. The openings form no constraint in this. If an opening need to be in a panel the user will need to adapt the parameters. This approach is chosen because most of the time the openings will be for dormers that are right in the center of the roof. And as the panels need to span from top to bottom, they cannot be fitter around openings as this would leave them too weak to carry the load. In practice the roofs are generally split in 2 as the maximum with that is used is most commonly 3,6m as already mentioned in the research phase.

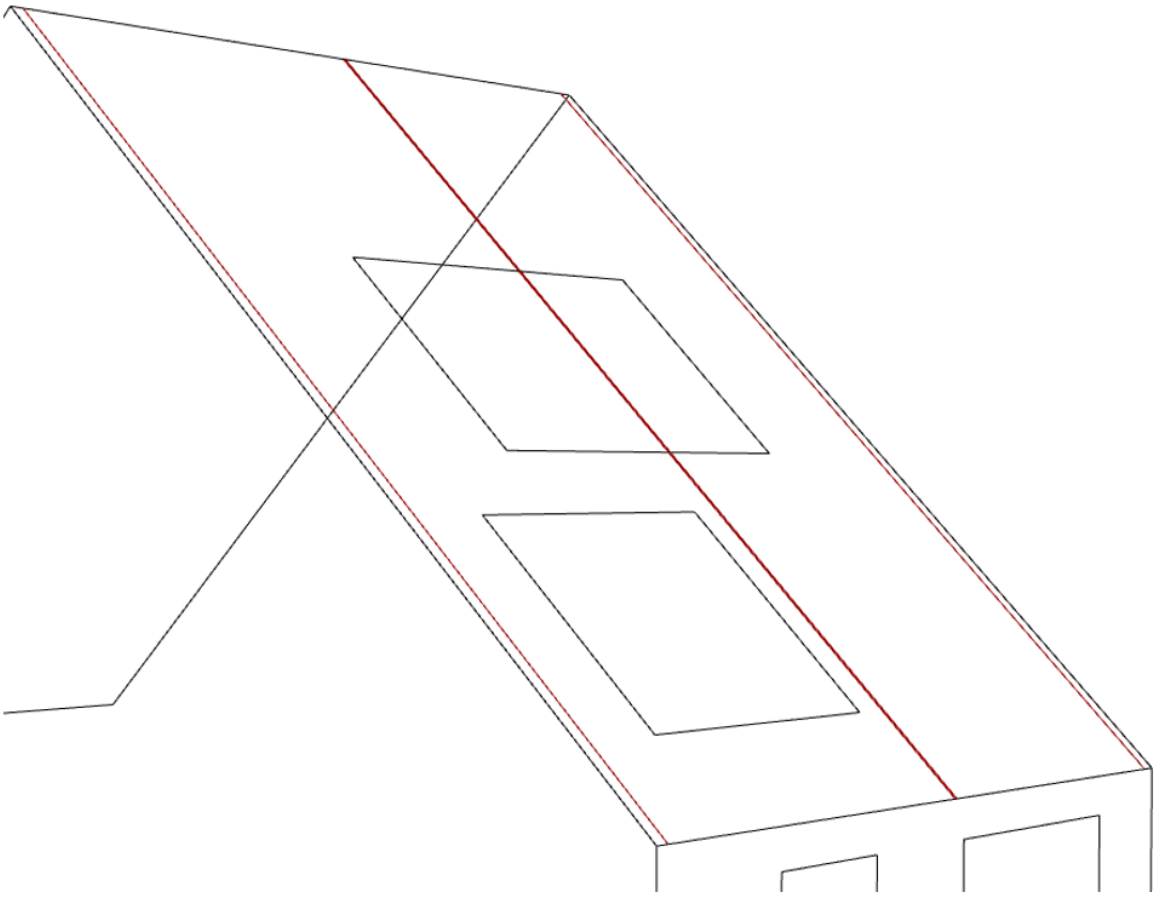


Image 67: Showing the generated panel edges of the roof panels (Source: own image)

Contour generation - Workflow

The way in which these contours are achieved is illustrated using the workflow diagram below. It starts off with the roof curve and generates a domain in X direction which functions as a limit for the panel contour boundaries. This is then taken and divided using the panel width and the tolerances. The output is a polyline curve that represents the contour of the panel that is to be made.

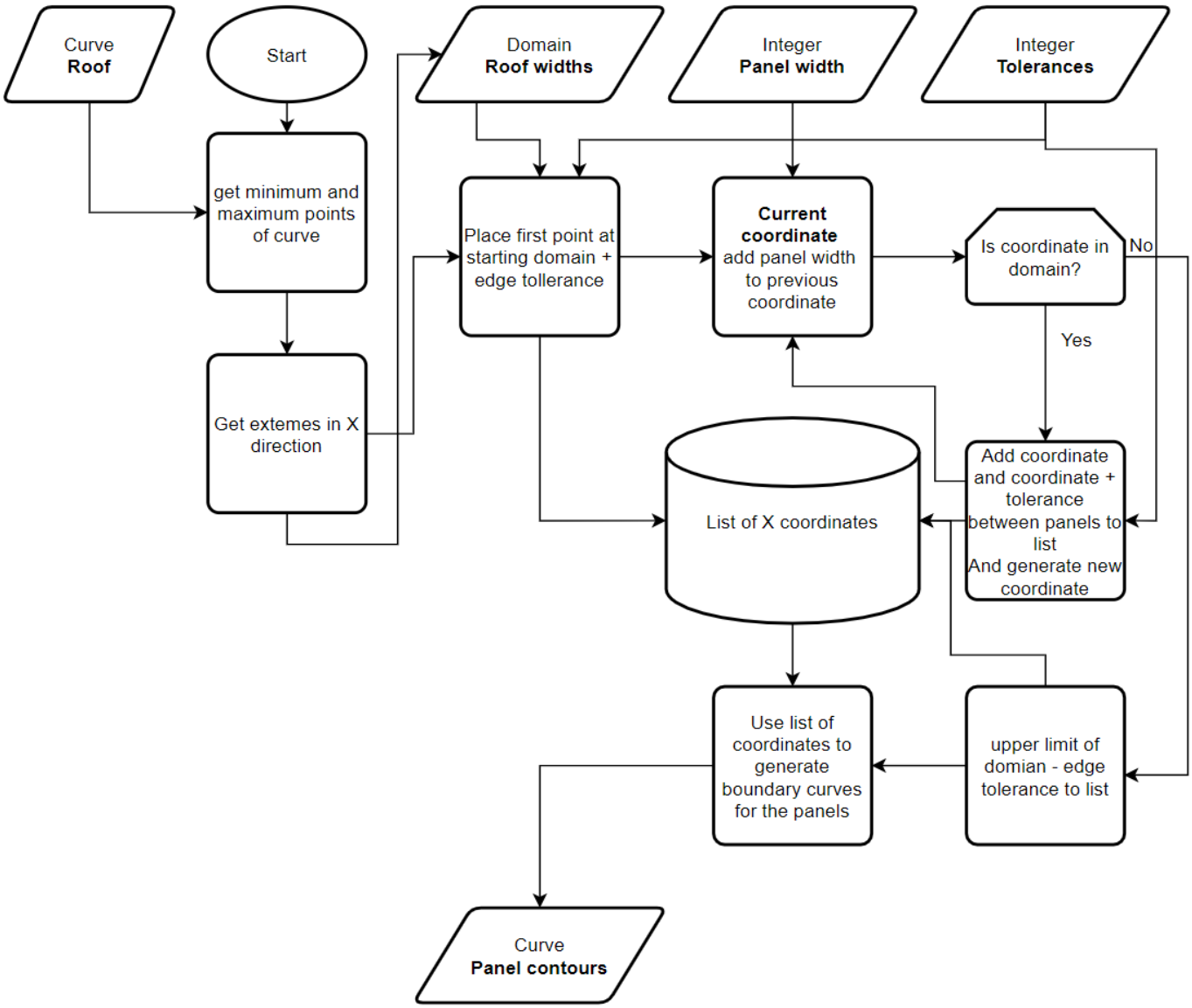


Image 68: Workflow diagram showing the workings of the generation of the panel contours (Source: self made)

Contour generation - Grasshopper

The execution of this method is done by a grasshopper python component. The code used in this component is shown below but also in appendix 7.2.1. The code starts with the Isolation of the horizontal members of the roof curve. Then the domain is constructed from these members. It then creates a list for the coordinates to go in and fills it. Afterwards this list is used to generate the panel exterior boundaries. Lastly a plane is generated that is in plane with the roof to use as a guide for the rest of the script to orient itself.

```
temp_roof = rg.PolylineCurve.DuplicateSegments(roof)
hor_lines = []
for line in temp_roof: ## isolates horizontal segments of the roof curve
    p1 = (rg.LineCurve.PointAtNormalizedLength(line,0))
    p2 = (rg.LineCurve.PointAtNormalizedLength(line,1))
    if abs(p1[2]-p2[2]) < 10:
        if p1[0] < p2[0]:
            hor_lines.append(rg.LineCurve(p1,p2))
        else:
            hor_lines.append(rg.LineCurve(p2,p1))
roof_dom =
[math.ceil((rg.LineCurve.PointAtNormalizedLength(hor_lines[0],0))[0]),math.ceil((rg.LineCurve.Poin
tAtNormalizedLength(hor_lines[0],1))[0])] ##constructs domain from the lines begin and end points
roof_dom.sort()
pan_cords = [roof_dom[0]+edge_tol]    ##creates list for the x coordinates and adds the first
                                      coordinate
while pan_cords[-1] < (roof_dom[1]-edge_tol-pan_w): ## generates the corner coordinates
    temp_point = pan_cords[-1]+pan_w
    temp_point2 = temp_point+betw_tol
    pan_cords.append(temp_point)
    pan_cords.append(temp_point2)
pan_cords.append(roof_dom[1]-edge_tol) ##Adds coordinate at the end of the panel

panel_contours = []
isplane = False
for i,cord in enumerate(pan_cords):
    if i%2:
        pass
    else: ##transforms coordinates in panel contours
        p1 = rg.LineCurve.PointAtLength(hor_lines[0],cord-roof_dom[0])
        p2 = rg.LineCurve.PointAtLength(hor_lines[1],cord-roof_dom[0])
        p3 = rg.LineCurve.PointAtLength(hor_lines[1],pan_cords[i+1]-roof_dom[0])
        p4 = rg.LineCurve.PointAtLength(hor_lines[0],pan_cords[i+1]-roof_dom[0])
        plp = [p1,p2,p3,p4,p1]
        panel_contours.append(rg.PolylineCurve(plp))
        if isplane == False:                ##generates a plane which is used to orient the rest of the
            plane = rg.Plane(p1,p2,p4)      tool to the angle of the roof.
            isplane = True
```

Block of code used to generate the panel contours in grasshopper

Structuring - Workflow

As a last part of the contours and as a base of reference for the rest of the method the panels generated and the openings provided are structured in the grasshopper tree structure in such a way that every panel is in its own branch and that openings within those panels are assigned to the same branch (within another tree). This enables the method and later the user to refer to the panel individually and pull their information. Contrary to the wall generation method with the roof is might be possible for an opening to be in multiple panels. These openings will be put in a different list as they need to be treated differently in the rest of the script. The generation of this structing is shown in the workflow below.

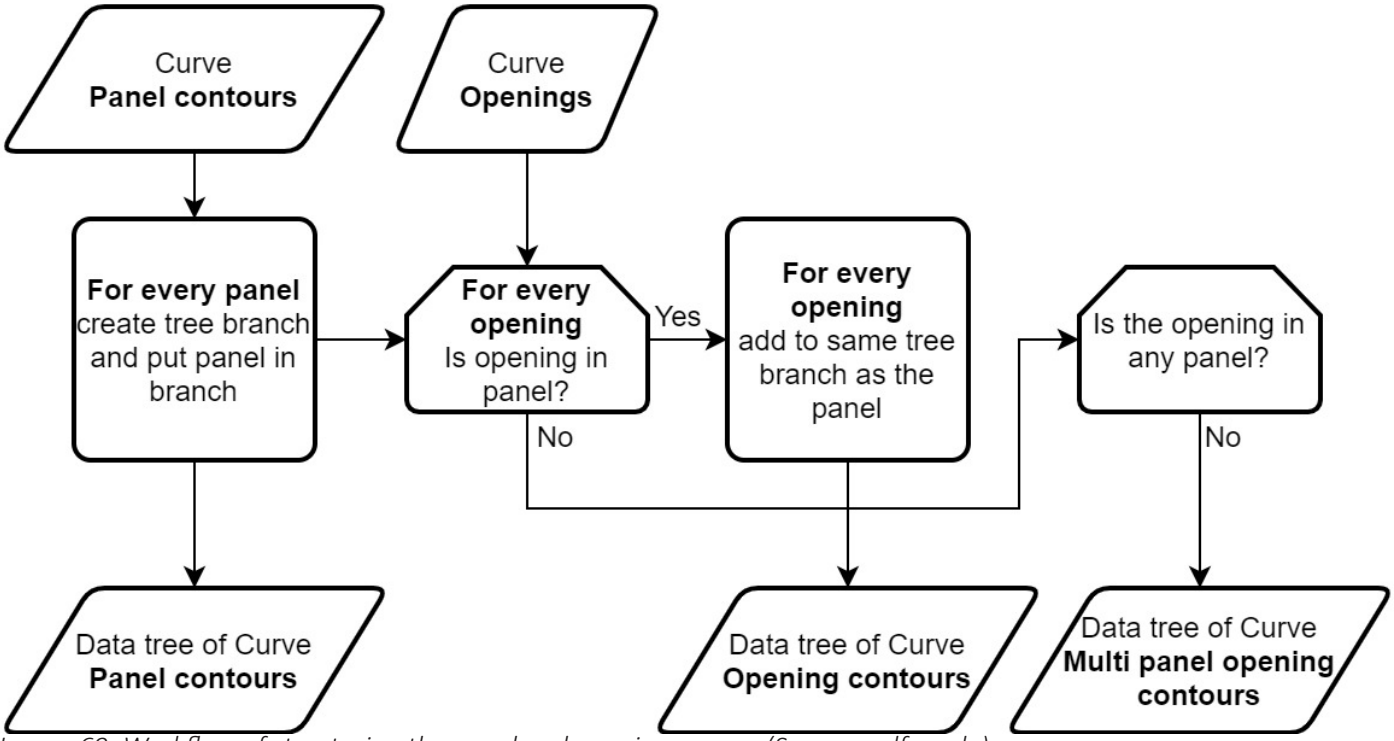


Image 69: Workflow of structuring the panel and opening curves. (Source: self made)

Structuring - Grasshopper

To make this workflow function in grasshopper a python script has been used, it is added in appendix 7.1.3 and because its size is small it's also added below. This block first creates the two data trees for panels and openings. After this it runs over all panels and generates a tree branch to place them in. while also checking for openings in the panels. If there are openings, they are added in the corresponding tree branch. If there are no openings an empty branch is generated in order for the rest of the tool to function properly. If the opening crosses multiple panels it is put in a separate list so it can be referred to separately later.

```
for i,pan in enumerate(panels):
  myPath = GH_Path(i)
  structured_panels.Add(pan,myPath) ##creates branch for every panel
  open_in_pan = False
  for open in openings:
    contain = []
    segs = rg.PolylineCurve.DuplicateSegments(open)
    for seg in segs:
      point = seg.PointAtNormalizedLength(0.5)
      contain.append(rg.Curve.Contains(pan,point,plane,0))
    if rg.PointContainment.Outside not in contain: ##put openings in branch if they are in one panel
      open_in_pan = True
      check += 1
      structured_openings.Add(open,myPath)
    elif rg.PointContainment.Inside in contain and rg.PointContainment.Outside in contain and check <
num_open:
      check += 1
      multi_pan_open.append(open) ##if the opening is in multiple panels it is put in a separate list
  if open_in_pan == False:
    structured_openings.EnsurePath(myPath)
```

Block of code used to structure panels and their respective openings in grasshopper

3.2.3.2. Layer generation

In this chapter the generation of the different panel layers will be discussed excluding the framework which will be discussed in the next chapter. The layers generated in this chapter are solid layers that are the size of the panel and have openings where there are openings in the panels. In theory the number of layers can be as much as required. For the purpose of this research, it was limited to four being: 1) interior board, 2) vapor proof layer, 3) framework, 4) water proof layer. The process of all layers except the framework is identical and so only the generation of one layer will be explained.

Input

In order for the method to operate it requires inputs, these are listed and discussed in this section.

- *Structured panels*
  The structured panels are the boundary of the layers.
- *Structured openings*
  The structured openings allow the layer generator to cut the openings out of the layer per panel.
- *Layer thickness*
  The layer thickness provides the thickness the layer should have.
- *Thicknesses of layers before*
  In order to get the layer in the proper location in the model it needs to be offset from the plane with the distance of the sum of all the previous layers their thicknesses.

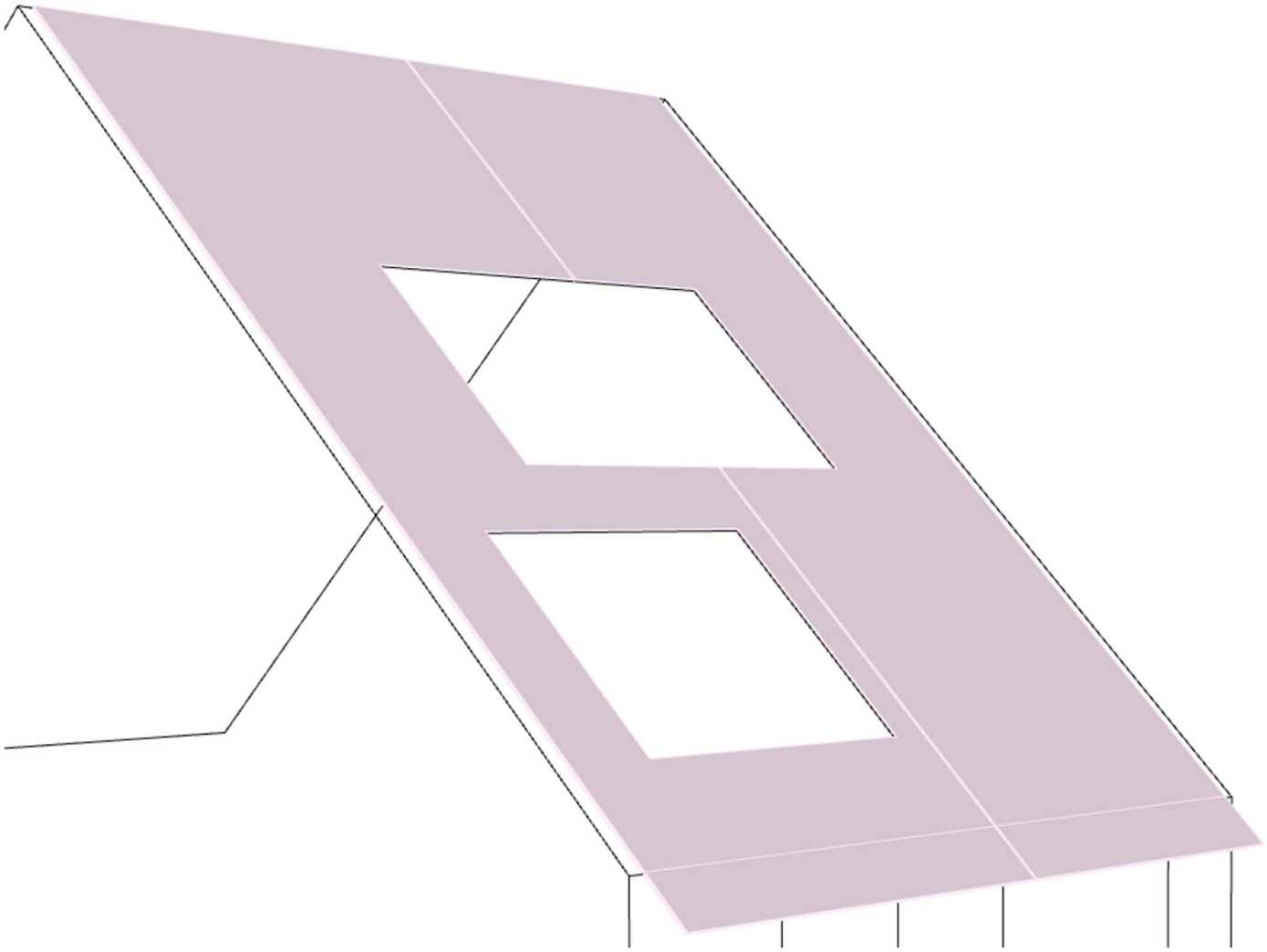


Image 70: The generated solid interior layer (Source: own image)

Process

The generation of the layers is done as illustrated in the workflow diagram below. First a surface is constructed from the panel contours. Then the openings are subtracted from this surface. Next, the surface is moved to the correct position in the panel depending on the layers that are used before the current layer. Lastly the layer is extruded to the correct thickness as defined by the user.

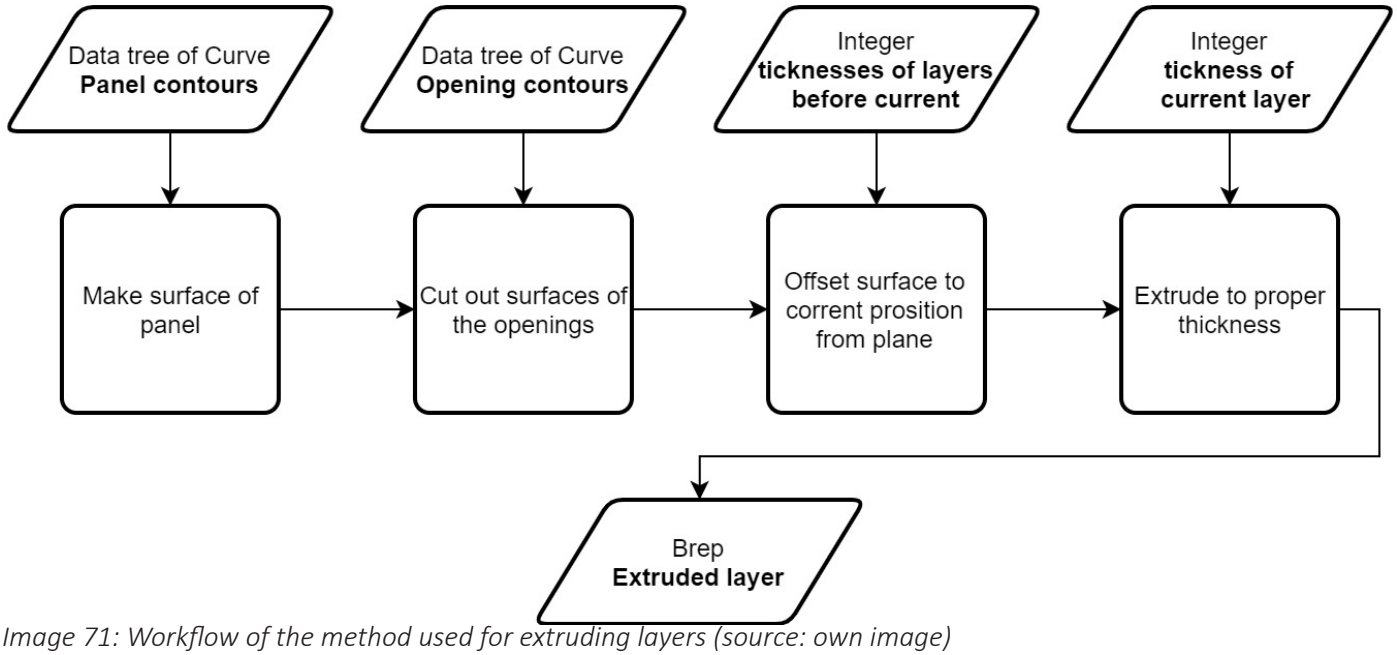


Image 71: Workflow of the method used for extruding layers (source: own image)

This workflow is translated into grasshopper with the components shown below. First the surfaces are formed and the opening surface is subtracted from the panels. A python component is used to select which surfaces are inside openings the python script can be found in appendix 7.2.2. Then the moving and extruding is executed.

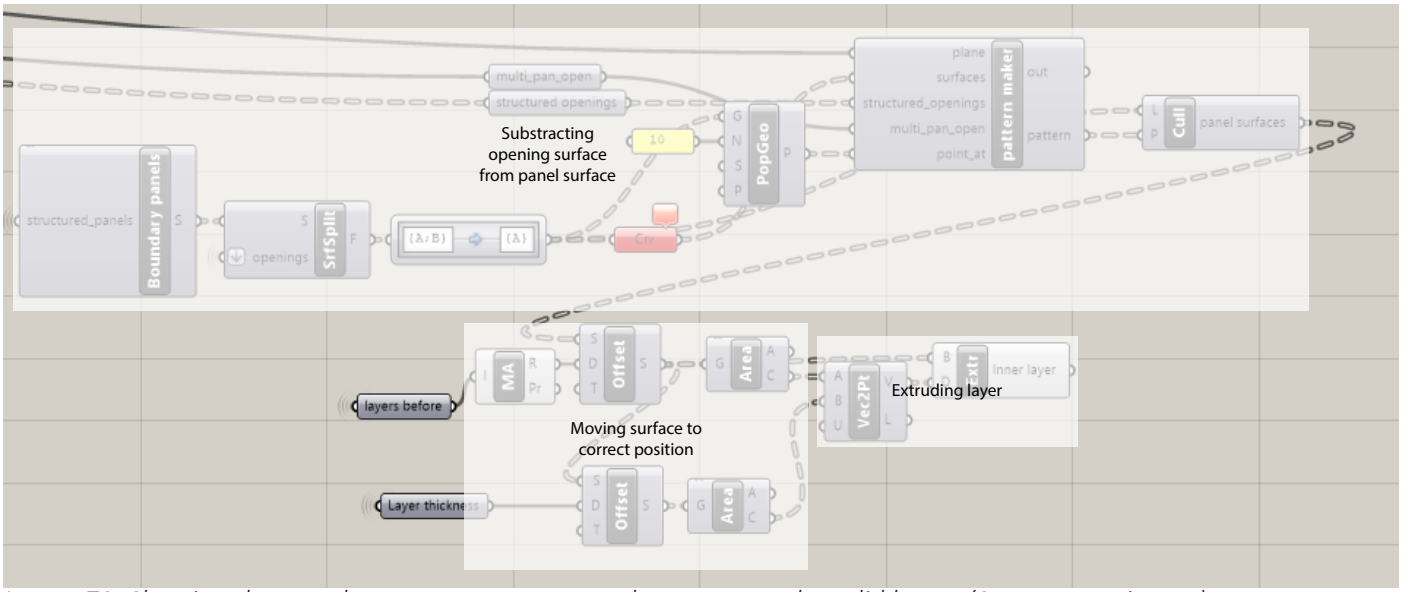


Image 72: Showing the grasshopper components used to generate the solid layers (Source: own image)

3.2.3.3. Layer generation – Framework

In this chapter the generation of the framework layer is explained. The framework is the skeleton of the panel and is generated in different steps. Which are shown in the grasshopper script at the bottom of the page and will be explained further in the rest of this chapter. First the plates and rafters are explained after which the overhang is gone over. Then the insulation generation and lastly the extruding of all these elements.

Inputs

• *Panels*

The panels are used as a boundary for all framework members and are the basis that is used to generate every member of the framework

• *Openings*

The openings are inputted to ensure the framework is framed around the openings and thus supporting the window frames

• *Multi panel openings*

Multi panel openings are similar to the openings only that they cross multiple openings. They also need framing around them but are not mounted in the factory.

• *Rafter width*

Rafter width is input to generate the rafters with the desired width.

• *Plane*

The plane which has been generated with the generation of the panel contours is input to ensure all operations executed by the script are done in the same plane as the roof.

• *Rafter minimum distance*

This minimum distance is used to check if when the rafters are generated, they have a minimum distance between themselves.

• *Rafter interval*

This is the maximum distance between the rafters.

• *Plate height*

This is the height used to generate the plates.

• *Overhang size*

The offset from the lower roof boundary used to generate the overhang.

• *Layer thickness*

Thickness of the framework layer, used to extrude the layer.

• *Thickness of layers before*

Thickness of the layers before, used to move the layers to the appropriate location within the panel to ensure no intersections.

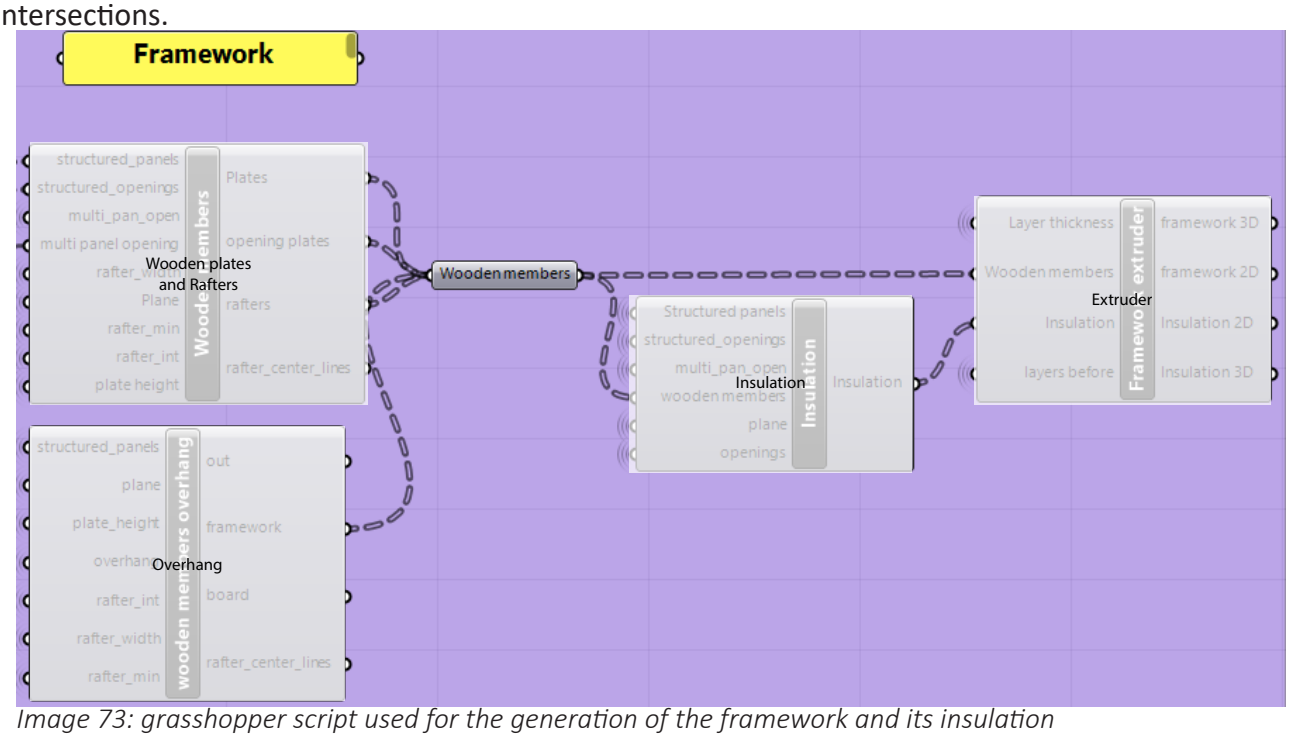


Image 73: grasshopper script used for the generation of the framework and its insulation

Process

In order to generate the framework for the panel several different components need to be generated first their contours will be generated after which they will be extruded.

Plates - Theory

The first step in generating the wooden members are the plates located at the ridge and eaves of the panel. These are generated by offsetting the horizontal curve segments of the panels. And connecting these curves to form the outline of the plate.

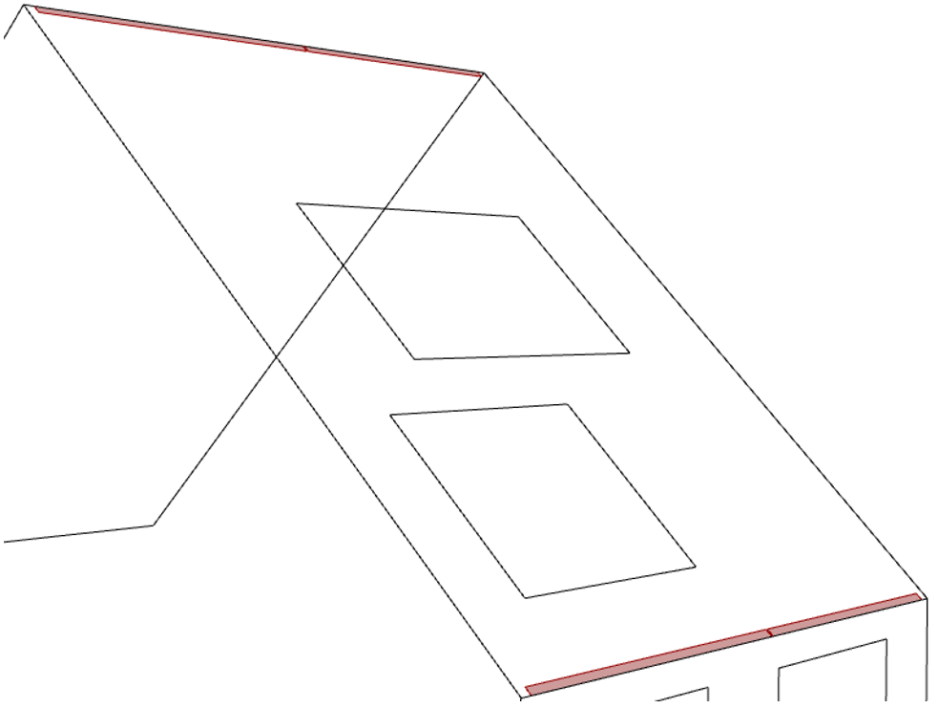


Image 74: Contours of the boundary plates (Source: own image)

Plates - Workflow

The method used to perform this action is illustrated using this workflow diagram. It uses the plate height provided by the user to offset the horizontal members of the panel. The offset and original members are connected to form the outer contour of the plates.

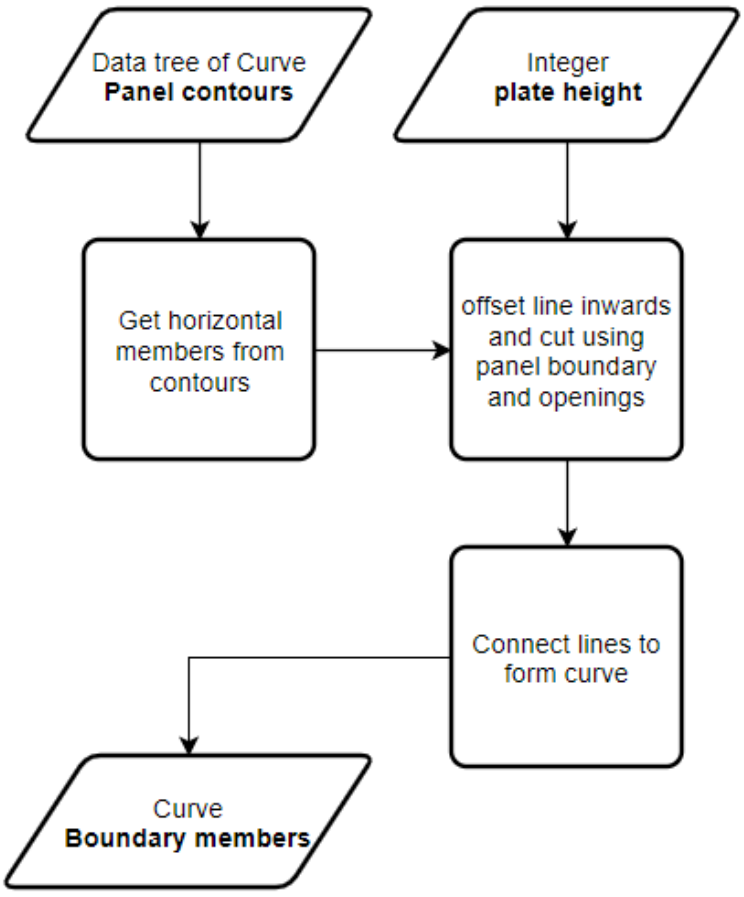


Image 75: Method used to generate the panel boundary plates (Source: own image)

Plates - Grasshopper

To translate the method to a functioning grasshopper scrip python coding has been used. The code is explained in detail in appendix 7.2.3. But is also shown below.

```
for i in range(structured_panels.BranchCount):
  myPath = GH_Path(i)
  for pan in structured_panels.Branch(i): ## loops through every panel
    segs = rg.PolylineCurve.DuplicateSegments(pan)
    horizontals = []
    for seg in segs: ## Checks every segment
      p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
      p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
      if abs(p1[2]-p2[2]) < 10:
        horizontals.append(seg) ## Isolates horizontal segments
    for hor in horizontals: ##offsets horizontals to the inside
      t1 = hor.PointAtNormalizedLength(0)
      t2 = hor.PointAtNormalizedLength(1)
      temp1 = hor.Offset(plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
      temp_line = temp1[0]
      temp2 = hor.Offset(plane,-plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
      point = temp_line.PointAtNormalizedLength(0.5)
      contain = rg.Curve.Contains(pan,point,plane,0)
      if contain == rg.PointContainment.Inside: ## connects the curves to form a plate contour
        t3 = rg.LineCurve.PointAtNormalizedLength(temp_line,1)
        t4 = rg.LineCurve.PointAtNormalizedLength(temp_line,0)
        bt = [t1,t2,t3,t4,t1]
        plate = rg.PolylineCurve(bt)
        plates.Add(plate,myPath)
      else:
        t3 = rg.LineCurve.PointAtNormalizedLength(temp2[0],1)
        t4 = rg.LineCurve.PointAtNormalizedLength(temp2[0],0)
        bt = [t1,t2,t3,t4,t1]
        plate = rg.PolylineCurve(bt)
        plates.Add(plate,myPath)
```

Block of code used to structure panels and their respective openings in grasshopper

Opening plates - Theory

The process of generating the opening plates is similar to the generation of the boundary curves, it also takes the horizontal lines of the curve. Which is the opening in this case. But instead, it offsets the lines outward as they need to be on the exterior of the opening. It then connects the lines to form the contour.

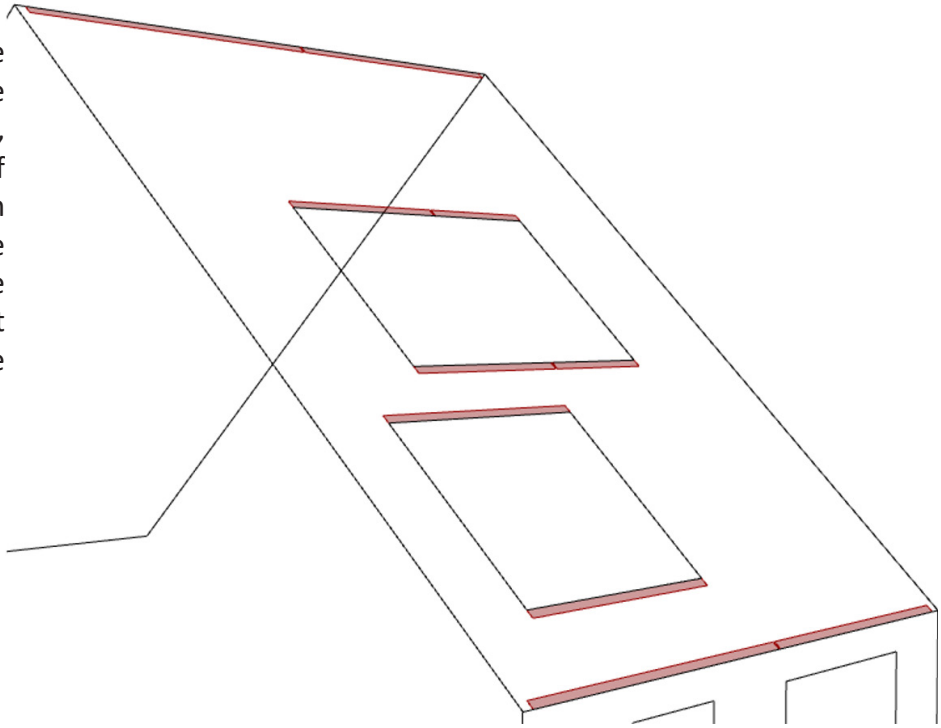


Image 76: Contours of the boundary and opening plates (Source: own image)

Opening plates - Workflow

To generate these contour lines the method illustrated in the workflow here is used. It has an additional step which checks if the plate is within the panel. To ensure openings that are on the boundary do not generate plates outside the panel.

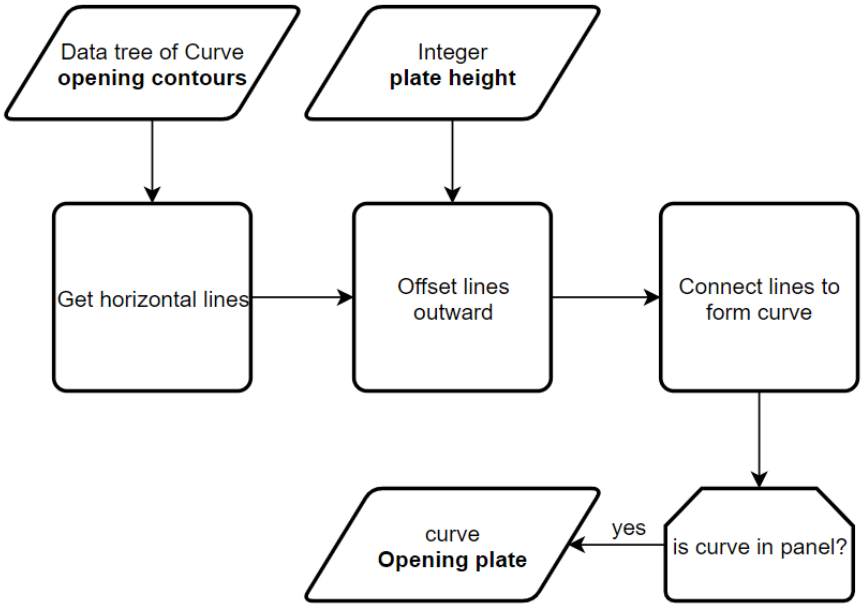


Image 77: Method used for the generation of the opening plates. (Source: own image)

Opening plates - Grasshopper

To perform the operation described on the previous page python code has been used. Which is added in appendix 7.2.3. Also, part of this code is shown below. The code is separated into two parts, the part shown below works for openings that are in one panel. And then there is the code that works for openings that spans multiple panels that only generate plates within the currently selected panel. In this way no duplicates are generated.

```
for open in structured_openings.Branch(i): ##runs through every opening
        segs = rg.PolylineCurve.DuplicateSegments(open)
        horizontals_open = []
        for seg in segs: ## gets horizontal segments
          p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
          p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
          if abs(p1[2]-p2[2]) < 10:
            horizontals_open.append(seg)
        for hor in horizontals_open: ## offsets segments outward generates the contour from them
          t1 = hor.PointAtNormalizedLength(0)
          t2 = hor.PointAtNormalizedLength(1)
          temp1 = hor.Offset(plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
          temp_line = temp1[0]
          temp2 = hor.Offset(plane,-plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
          point = temp_line.PointAtNormalizedLength(0.5)
          contain = rg.Curve.Contains(open,point,plane,0)
          if contain == rg.PointContainment.Outside:
            t3 = rg.LineCurve.PointAtNormalizedLength(temp_line,1)
            t4 = rg.LineCurve.PointAtNormalizedLength(temp_line,0)
            bt = [t1,t2,t3,t4,t1]
            plate = rg.PolylineCurve(bt)
            open_plates.Add(plate,myPath)
          else:
            t3 = rg.LineCurve.PointAtNormalizedLength(temp2[0],1)
            t4 = rg.LineCurve.PointAtNormalizedLength(temp2[0],0)
            bt = [t1,t2,t3,t4,t1]
            plate = rg.PolylineCurve(bt)
            open_plates.Add(plate,myPath)
```

Block of code used to structure panels and their respective openings in grasshopper

Rafters - Theory

Having generated all the plates within the boundary of the panel the rafters can be generated. The generation of these members is more complicated than the generation of the plates. This is due to the fact that the rafters interfere with all other wooden elements and openings. They need to connect to the side of the windows but cannot go through the opening. Also, there is no predetermined number of rafters as there are with plates. Because this depends on multiple parameters.

This approach is very similar to the approach of the wall panel studs.

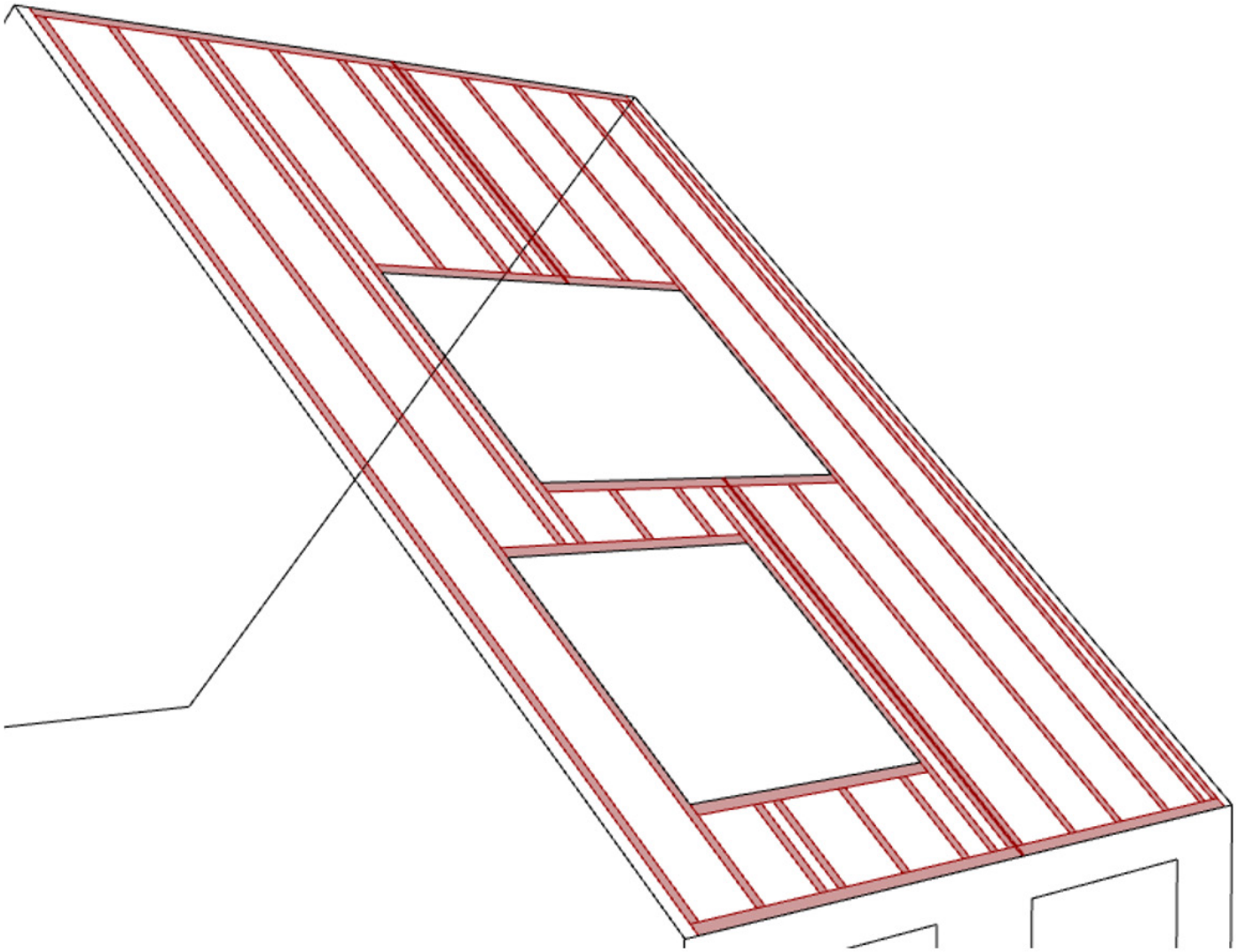


Image 78: Generation of rafters (Source: Own image)

Rafters - Workflow

In order to get this, result the method shown below is used. First the panel contours are used to generate a domain of the panel in x direction. This domain functions as a limit within which the x coordinates for the rafters are generated. It adds the first coordinate of the loop to the list and then starts a loop that only stops when the end of the domain is reached. For every iteration in this loop, it takes the last coordinate of the list and adds the rafter interval, it then checks if it is close to the edge of the window. If it is to close it moves it away to ensure insulation can properly be applied and it ensures that there is always a stud right next to the window opening. If the new coordinate is outside the panel domain the loop is broken and the upper limit of the domain is added. With these coordinates center lines for the rafters are generated. And cut to ensure no lines are in openings or plates. Then the centerlines are used to generate the contours.

In order to generate these this method in grasshopper a python component is used the component layout and its script are shown in appendix 7.2.4.

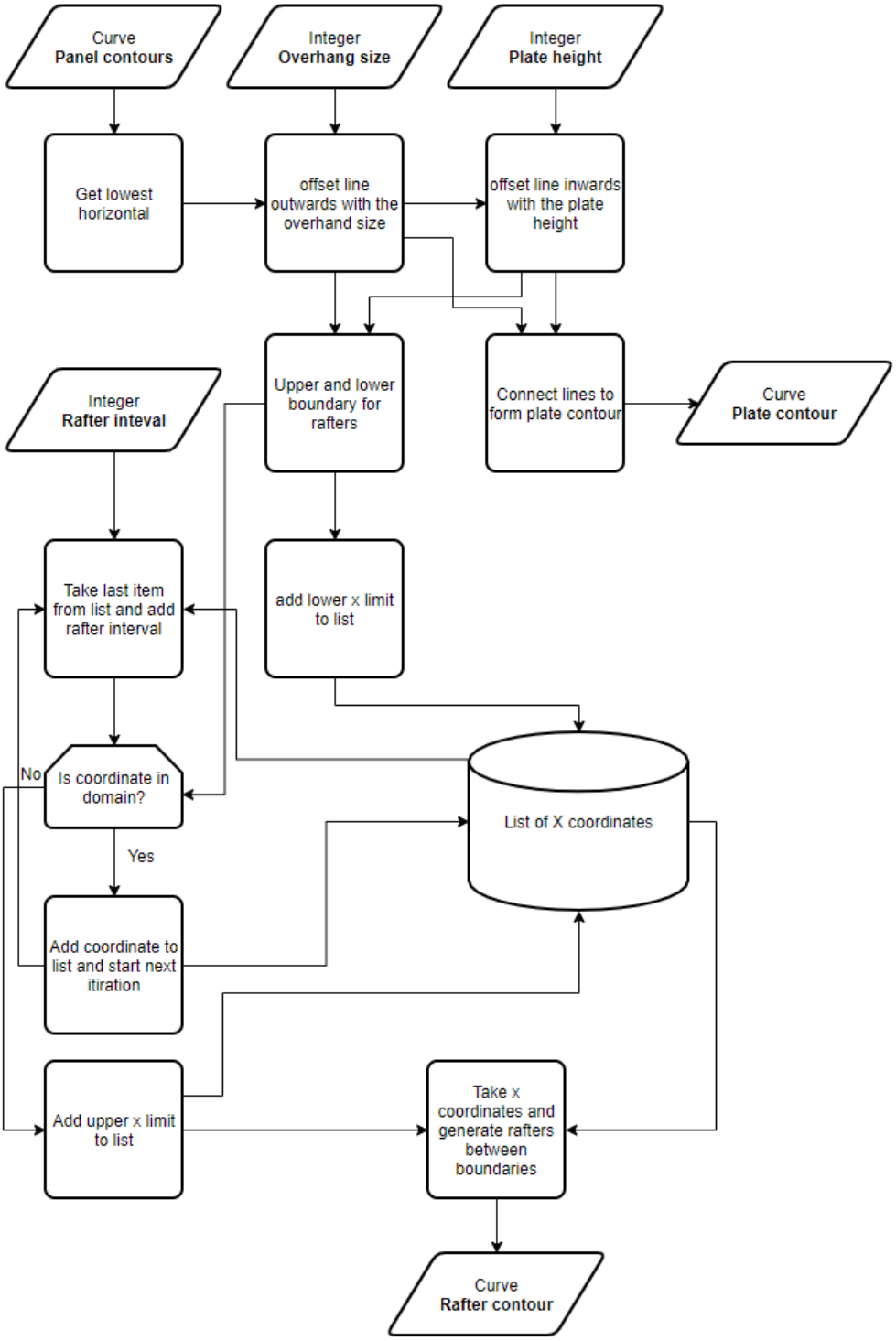


Image 79: Method used for the generation of rafters (Source: Own image)

Insulation - Theory

The generation of the insulation is similar to the normal generation of insulation in the wall panels. A surface is constructed from the panel and the openings and the wooden members are cut out. In this way only the to be insulated surfaces remain. The overhang isn't fitter with insulation as is designed to be outside the thermal boundary.

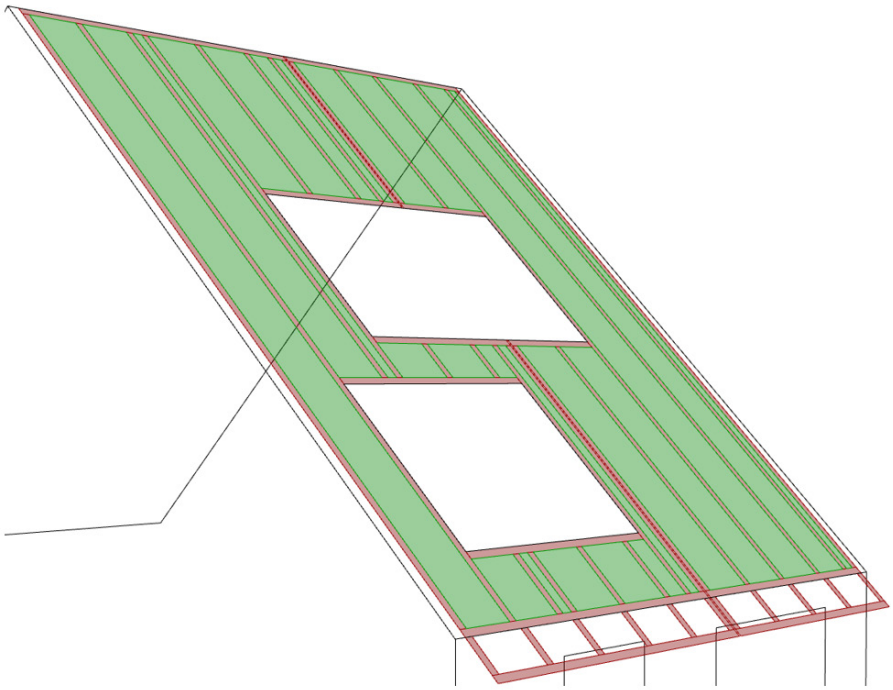


Image 80: Insulation for the roof panels (Source: own image)

Insulation - Workflow

To generate the insulation the workflow shown at the bottom right is used. First the surface of the panels is generated and then the surface of the already generated elements (rafters, plates, opening plates and the openings) is subtracted leaving only the insulation.

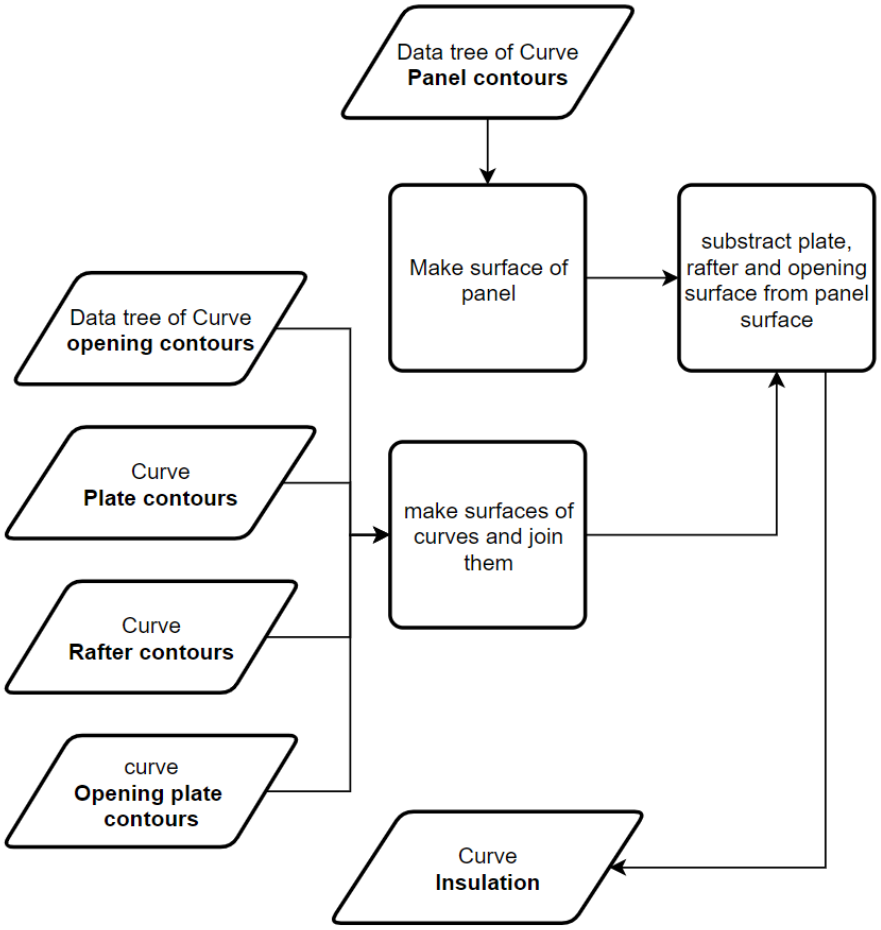


Image 81: Workflow illustrating the generation of the insulation within the framework layer (Source: own image)

To generate these contours the grasshopper script below is used. First the surfaces are split, after which the surfaces are selected by checking if surfaces are within openings or wooden members.

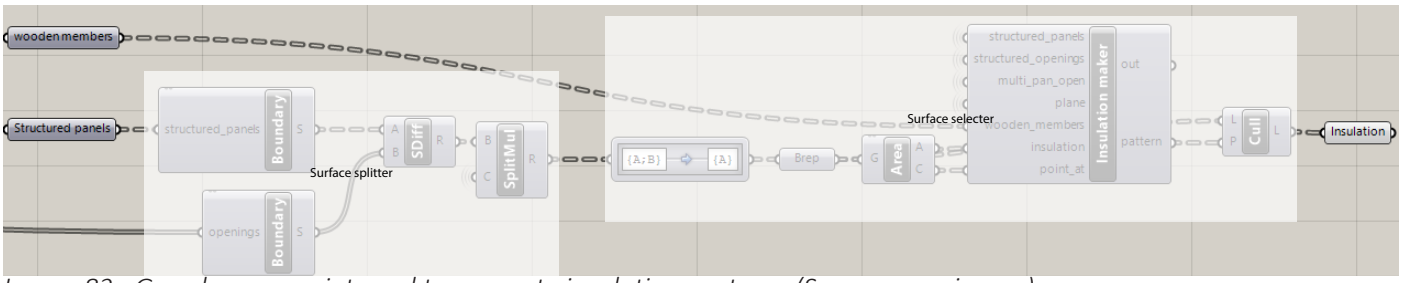


Image 82: Grasshopper script used to generate insulation contours (Source: own image)

Extruding and moving

After all the surfaces have been generated, they are moved to their position in the panel (depending on the layers that are in front of them) and extruded. This process is identical to the generation of the normal solid layers. As can be seen in the workflow diagram of this process which is shown at the bottom of the page.

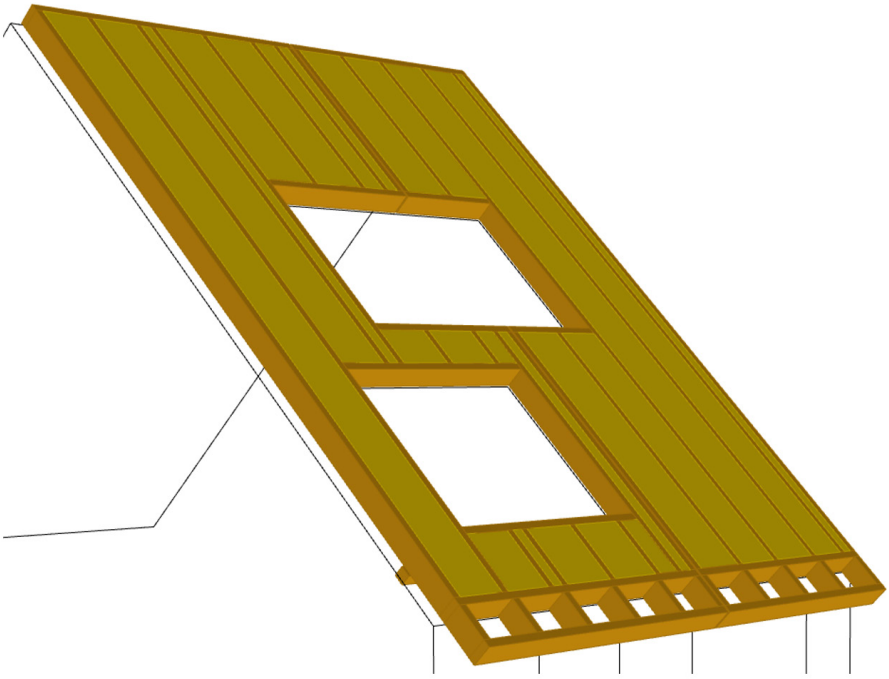


Image 83: Extruded framework members and insulation (Source: Own image)

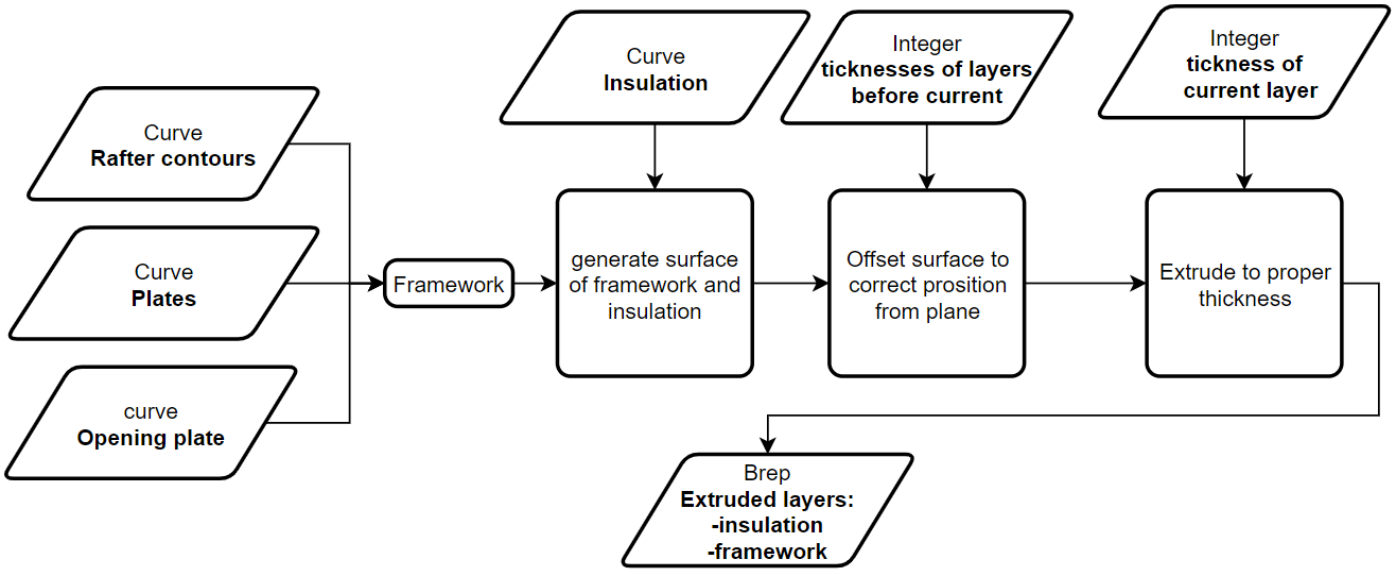


Image 84: Workflow diagram of the extrusion of framework and insulation layer (Source: Own image)

3.2.3.4. Panel attachments – Wall beam

The wall beam is the most important structural component of the panel as it transfers all the forces from the roof to the supporting structure below. In this chapter the generation of this element is discussed.

Input

- *Panel contours*
  The contours of the panels are used to generate the wall beams.
- *Plane*
  The pane which is used to ensure the wall beam is generated in the same plane as the roof panels
- *Bottom of wall beam*
  This parameter allows the user to specify where the bottom of the wall beam should be.
- *Sizing of wall beam*
  The size of the wall beam which can be calculated to ensure it can transfer the forces.

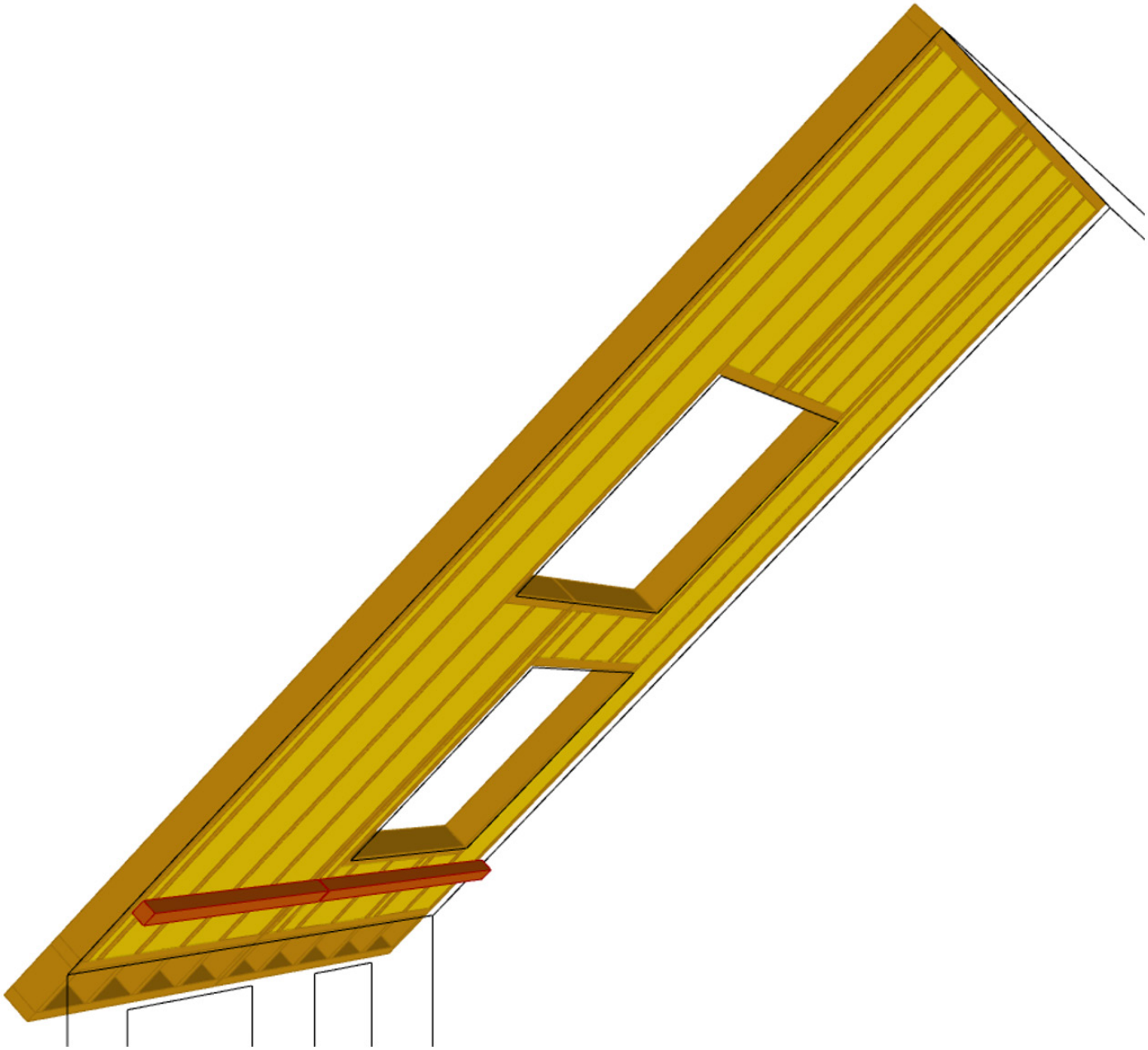


Image 85: The wallbeam at the bottom of the panels (Source: Own image)

Process

To create the wallbeam the bottom horizontal curve of the panel is isolated. This curve is then offset inwards to the location of the bottom of the wall beam. This curve is then offset again with the height of the wall beam to provide the top of the wall beam. These curves together from the contour which is extruded to create the actual wall beam.

This

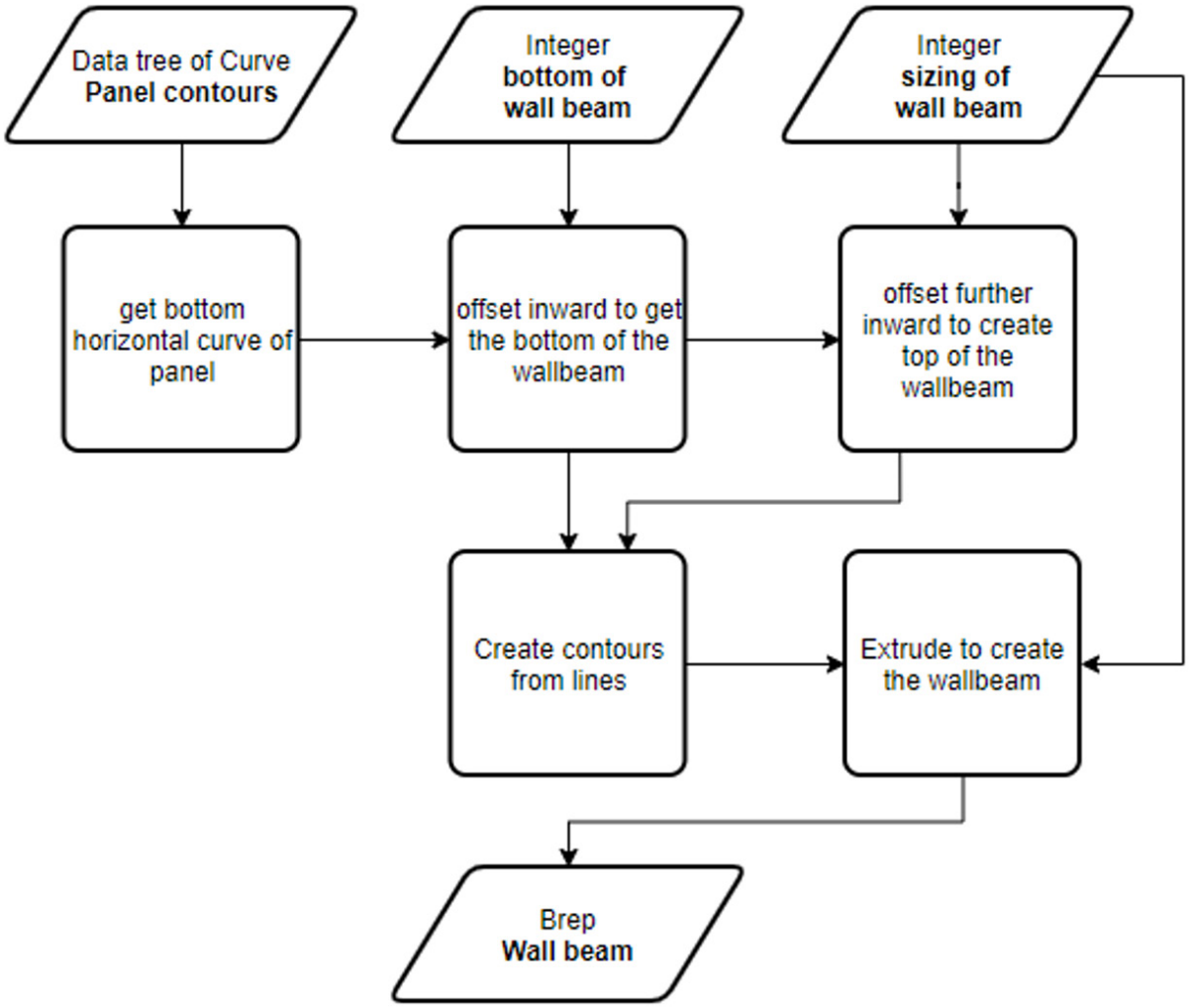


Image 86: Method used for the generation of the wall beams (Source: Own image)

method is translated into grasshopper using the components shown below. The contour is generated in python of which the script can be found in appendix 7.2.6. The extrusion is created with grasshopper components to create a vector which is used as direction for the extrusion.

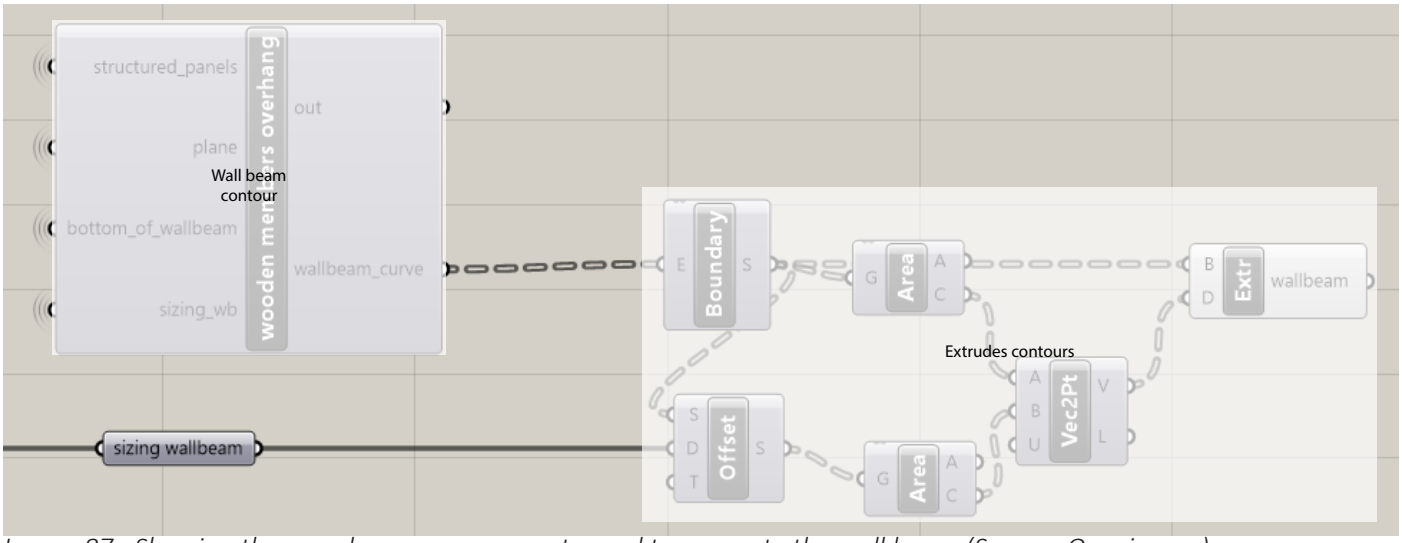


Image 87: Showing the grasshopper components used to generate the wall beam (Source: Own image)

3.2.3.5. Panel attachments – Laths

The laths form the structure of the roof that allows for the tiles to be mounted. There are two rows of laths. The first row is vertical and are on top of the rafters, these allow for the ventilation beneath the tiles. The second layer are the horizontals which are the laths that allow for mounting of the tiles. Most inputs of this tools will be focused on the horizontals as these need to be adjusted the most.

Inputs

- *Panel contours*
  The contours which are used as a boundary for the creation of the lath's
- *Plane*
  The plane is used to ensure any operation done to the lath's are done in the correct plane.
- *Overhang contours*
  The overhang contours are added to the panel contours as there will also need to be laths in the overhang.
- *Openings*
  The openings are used to cut away lath's that go over openings
- *Rafter centerlines*

The rafter centerlines are used to generate the vertical lath's

- *Plate height*
  The plate height is used to ensure the vertical lath's go up until the edge of the panel rather than stop where the rafters stop
- *Lath's width*
  Used when generating the lath's
- *Lath's interval*
  Used when populating the panel with horizontal lath's
- *Lath's thickness*
  Used for the extrusion of the lath's
- *Thickness of layers before*
  Used to move the lath's to the proper location

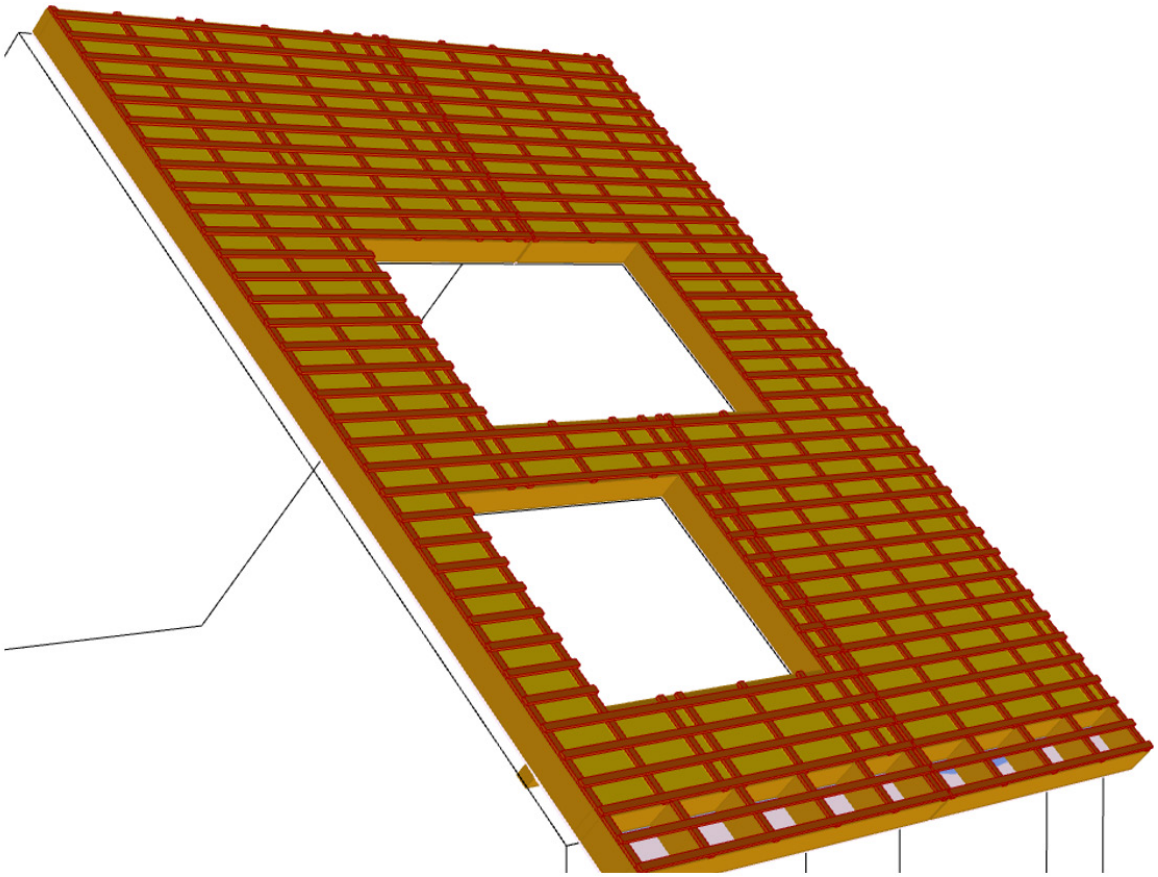


Image 88: The generated Laths on the roof panel (Source: Own image)

Process

As mentioned, the lath's are generated separately for the vertical and horizontal lath's. This process is illustrated using the workflow diagram shown below. The verticals are generated by using the rafter centerlines which are generated with the creation of the rafters. These lines are extended by the plate height to ensure they go up until the boundary of the panel. then they are offset to both sides with half the lath thickness to generate the laths contours.

The horizontals are made similar to the rafters by the generation of coordinates within a domain. However, the interval between the lath's is most important as this depends on the tile size so the laths won't move for the openings. When these coordinates are generated, they are turned into centerlines which are cut when they are inside a window opening. Lastly, they are similar to the verticals offset to both sides and turned into contours. After this they are moved and extruded at the correct position which has already been explained in the layer generation chapter.

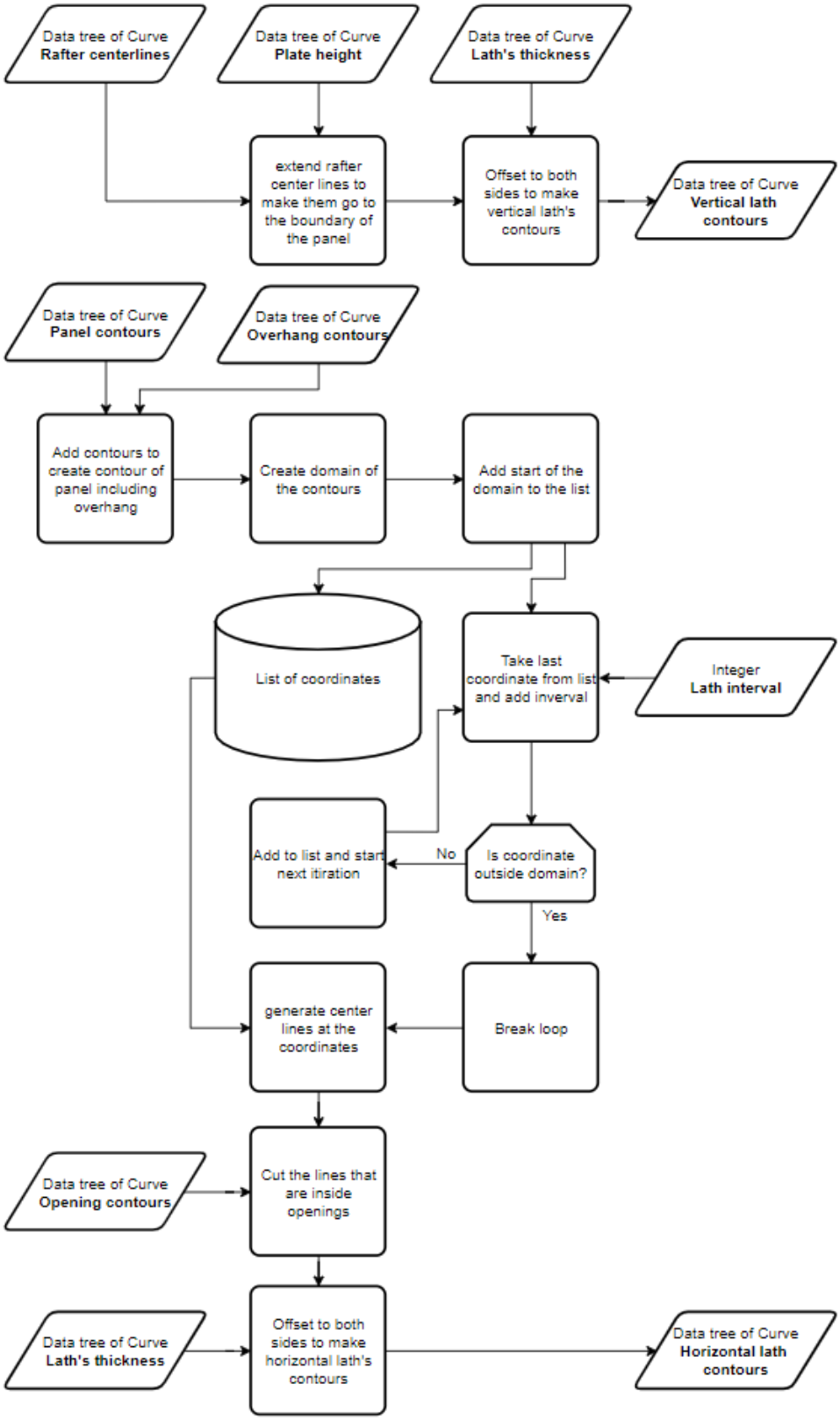


Image 89: Workflow diagram showning the process of generating laths (Source: own image)

This process is translated to grasshopper which is shown below. The lath contours are generated using a python component of which the code can be found in appendix 7.2.7. Then the verticals and horizontals are moved and extruded separately as they need to be on different level within the panel.

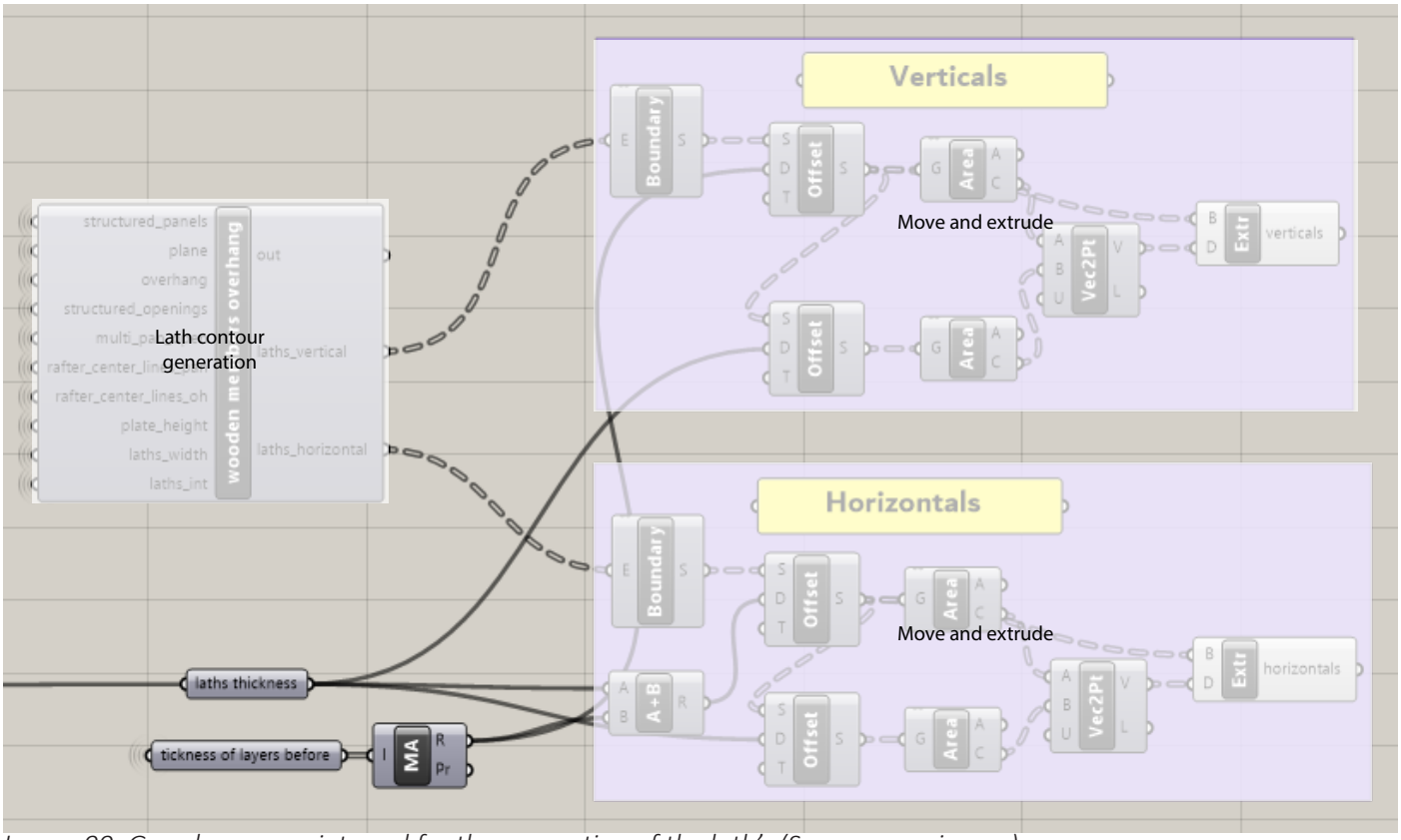


Image 90: Grasshopper script used for the generation of the lath's (Source: own image)

3.2.3.6. RC-calculator

To ensure the wall panels have a sufficient thermal resistance the RC value is calculated within the tool.

Input

Stage one

- *Insulation thickness*
  Integer containing the thickness of the insulation
- *Lambda of insulation*
  Integer containing the lambda value of the insulation

Stage two

- *Framework thickness*
  Integer containing the thickness of the insulation layer
- *Lambda of framework elements*
  Integers containing the lambda values of the wooden members and insulation
- *Area of frame members*
  Area of the wooden members and insulation to determine the mean RC-value

Process

The calculations are done in two stages; the first stage is being calculated before the process of generating the framework for the panels. It does so by dividing the thickness of the layer by the lambda value of the specific material which is provided by the user. Which is the way to calculate Rc values for solid constructions. (Hagentoft, 2003) This calculation gives an estimate of the upper boundary of the RC value of the panel. As the addition of more wooden framework members (which have a lower lambda) reduces the Rc value of the panels. As the percentage of wooden members is different for every panel due to its layout the lowest value is selected and fed back to the engineer so that he can decide if it is sufficient. The image below shows a panel within grasshopper giving the feedback to its user about the two calculated Rc values. The scripts used can be found in appendix 7.1.9.

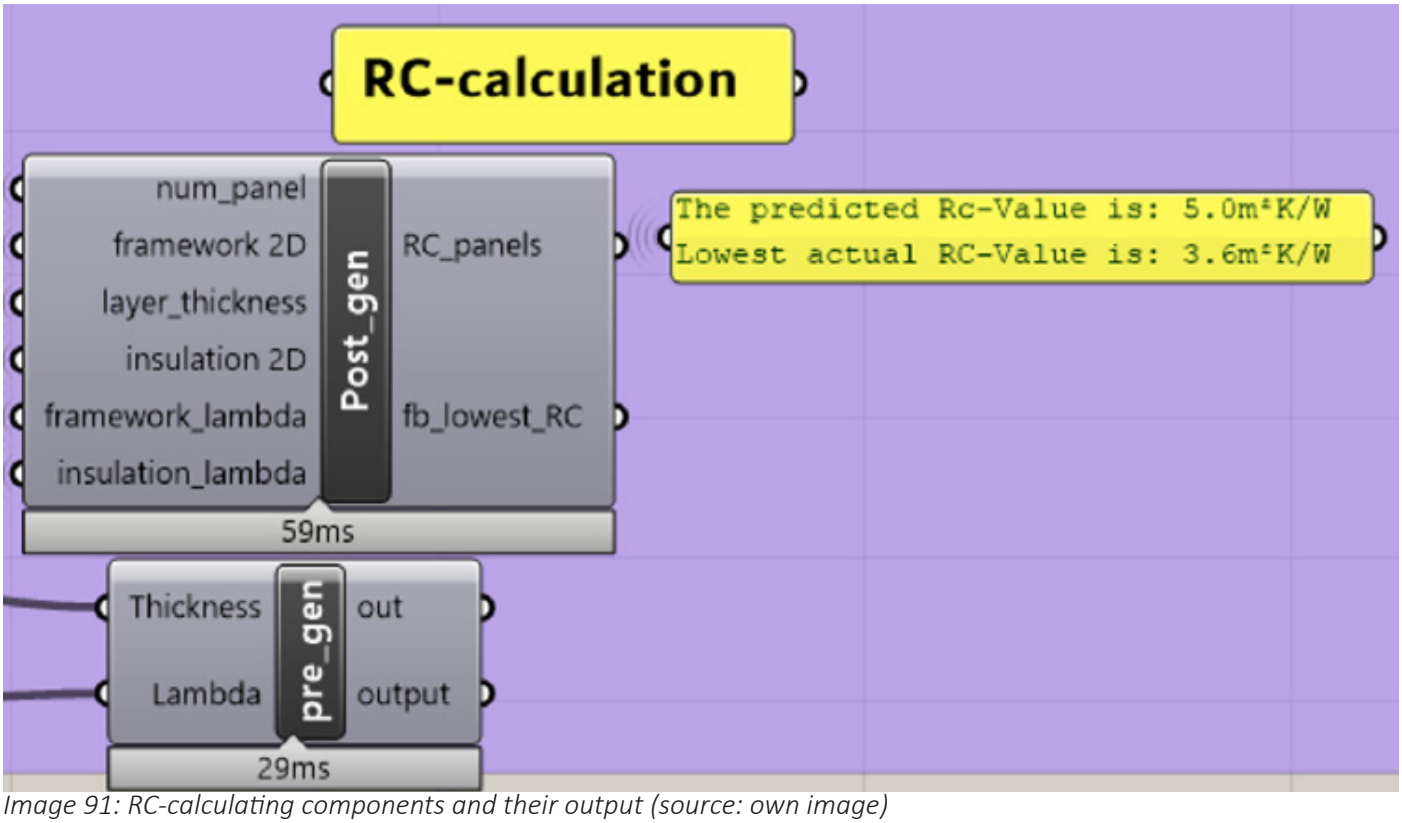


Image 91: RC-calculating components and their output (source: own image)

3.2.3.7. Weight calculator

To enable the user of the tool to assess how much mounting points the panel needs (depending on the weight and the strength of the supporting structure) the weight of the panels is calculated.

Input

- *Density per material*
  Integer containing the density of the materials used
- *Volume per material*
  Volume of the materials in the panels. Gathered from the extruded elements

Process

In order to calculate the weight of the panels all their extruded elements are taken and multiplied by the by the user provided density. As with all the other elements in the computational script the weight is separated in a branch per panel. A mass addition provides a simple overview of the weight per panel as shown in the image below.

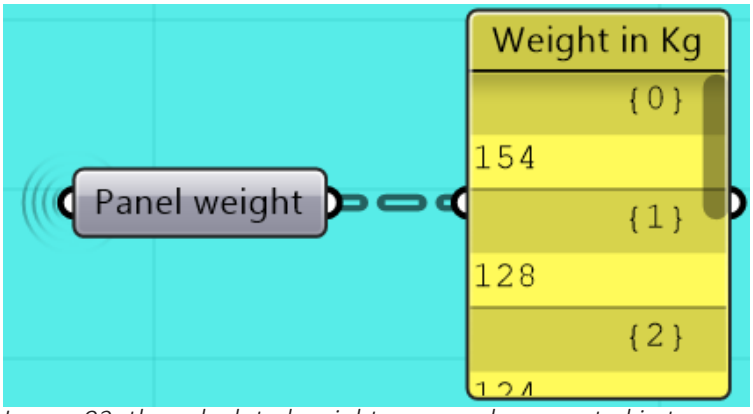


Image 92: the calculated weight per panels separated in tree branches (Source: own image)

3.2.3.8. Window placement

The windows placed in prefab elements are not created together with the panel they are generally a complete component that is placed inside the panels. For this reason, they are not generated in the script. They are however moved to the proper location within the panel so that they can be fixed in that location upon the manufacturing of the panel.

Input

- Openings
  The opening curves where the windows need to be placed. Which will also need to be assigned a window tag by the used
- *Frame tags*
  Frame tag per different Frame type so that the script can refer to the different frames and place them accordingly
- Weight per window
  Weight per frame so that it might be added to the total panel weight
- Offset
  Offset of the plane to place the windows in the correct depth in the panel.

Process

This placement is done by the user assigning window tags to openings and providing the information per unique tag such as their geometry and the weight. The script then takes the geometries and moves them from there reference corner point to the reference corner point of the opening. The placed frames can be offset to the position desired by the design.

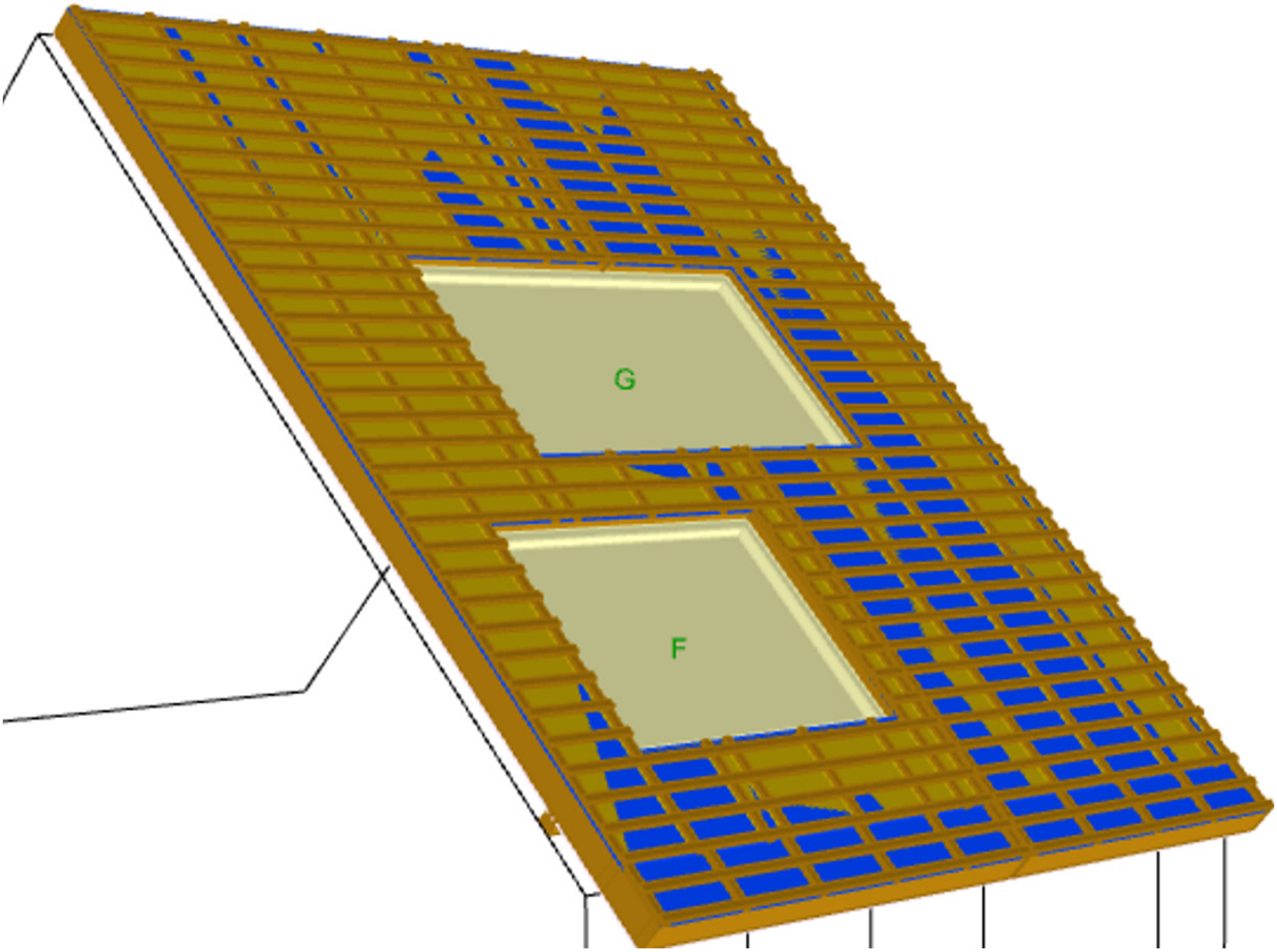


Image 93: Placed windows and their tags (source: own image)

The process described on the last page is visualized by the work flow below. The user provides a list of tags that correspond to a certain opening. The component then cross reference the tags. If there is a match the Brep corresponding to the tag is moved to the location of the window.
This workflow is translated into a grasshopper script, which can be found in appendix 7.1.11.

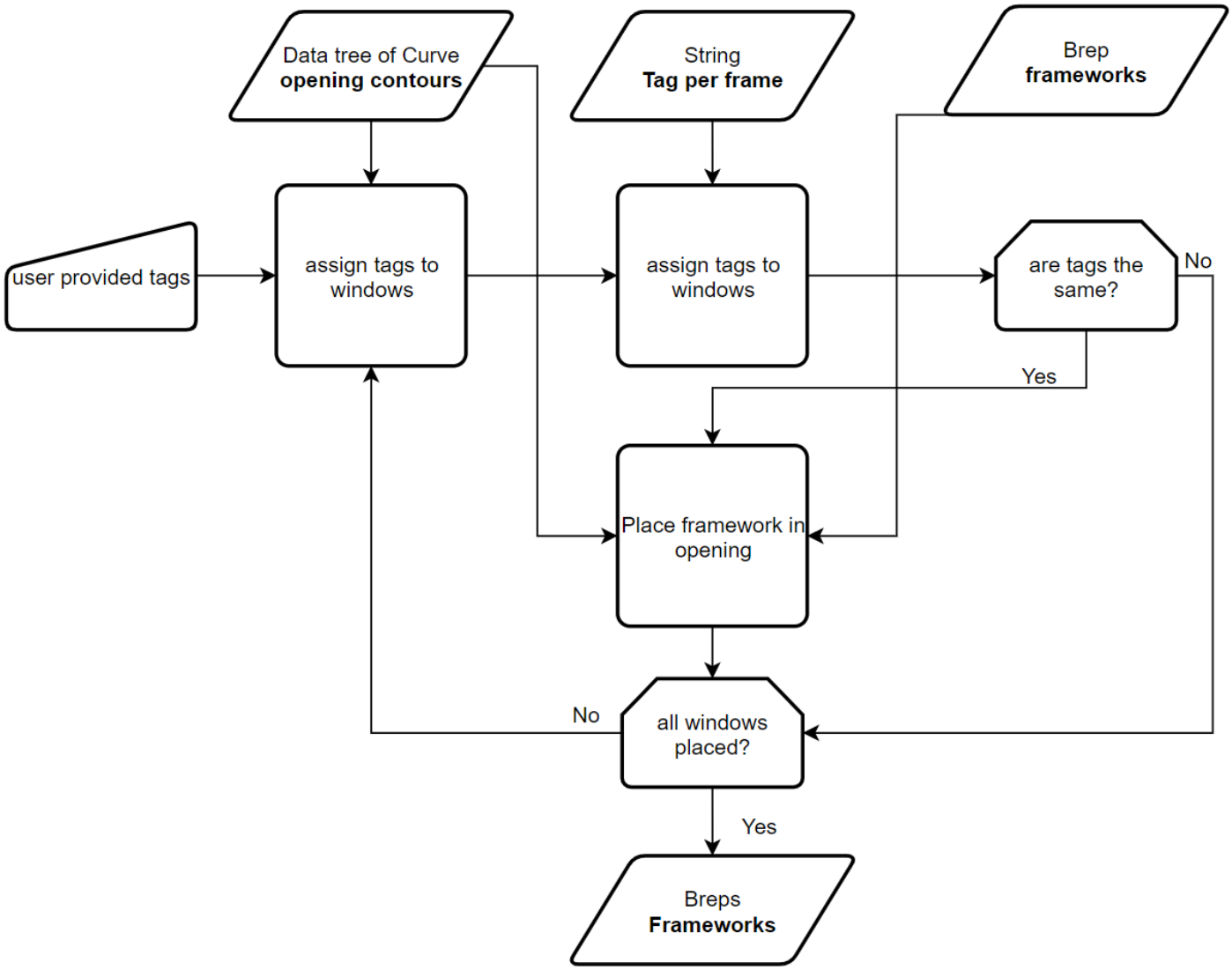


Image 94: Workflow diagram of the window placement component (source: own image)

3.2.3.9. Conclusion

Those were all the individual parts that together form the computational method that allows the user to generate roof panels. The scheme below shows the integration of all the individual parts of the method in the big picture as well as there in and output.

The inputs the user provides to the tool can be split in numerical and non-numerical inputs. The numerical inputs such as wooden member sizing's allows the user to change the size of all components in the panel. The non-numerical inputs, such as the boundary contours of the roof allows the user control by which he can dictate exactly where the panels should be generated.

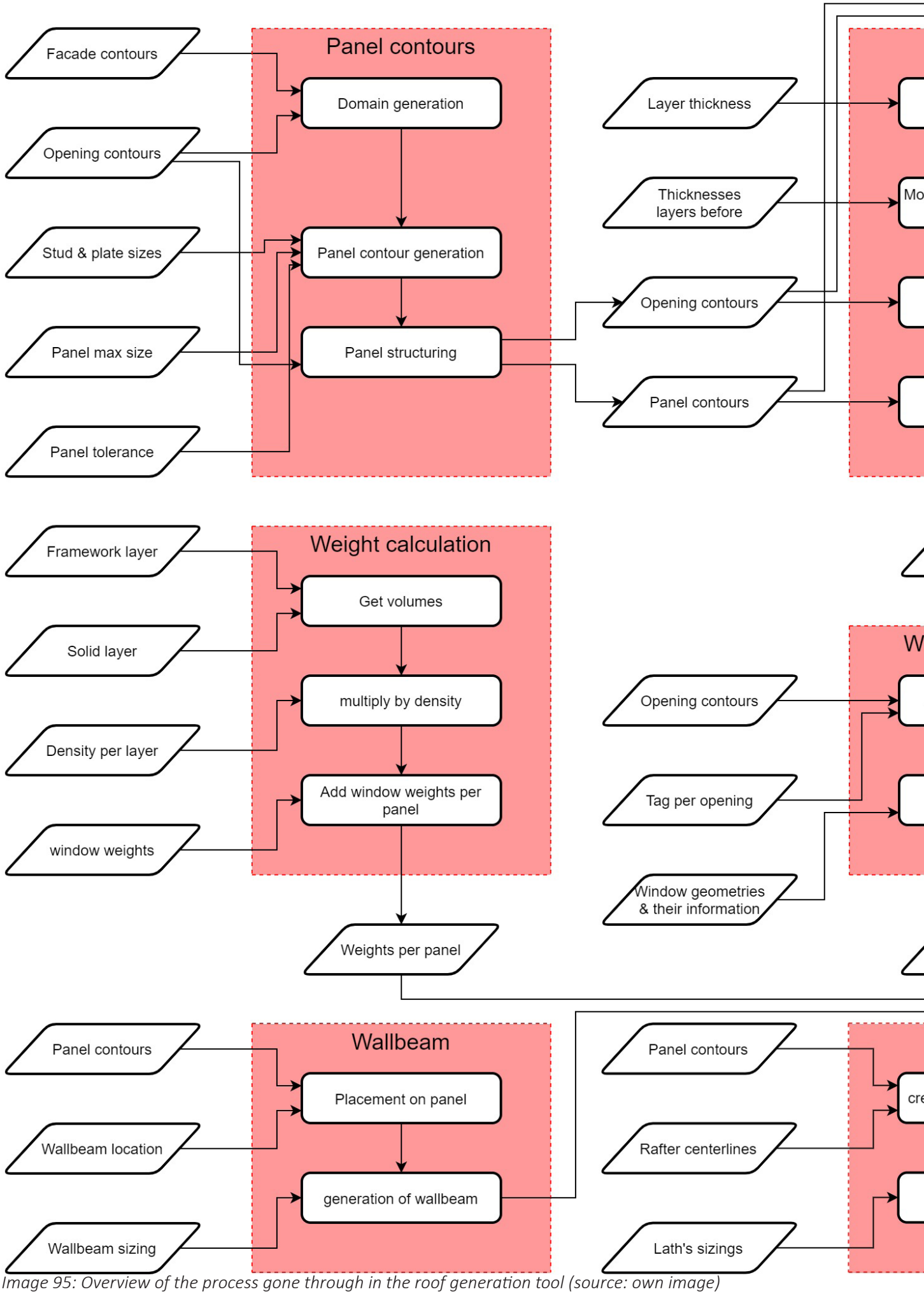
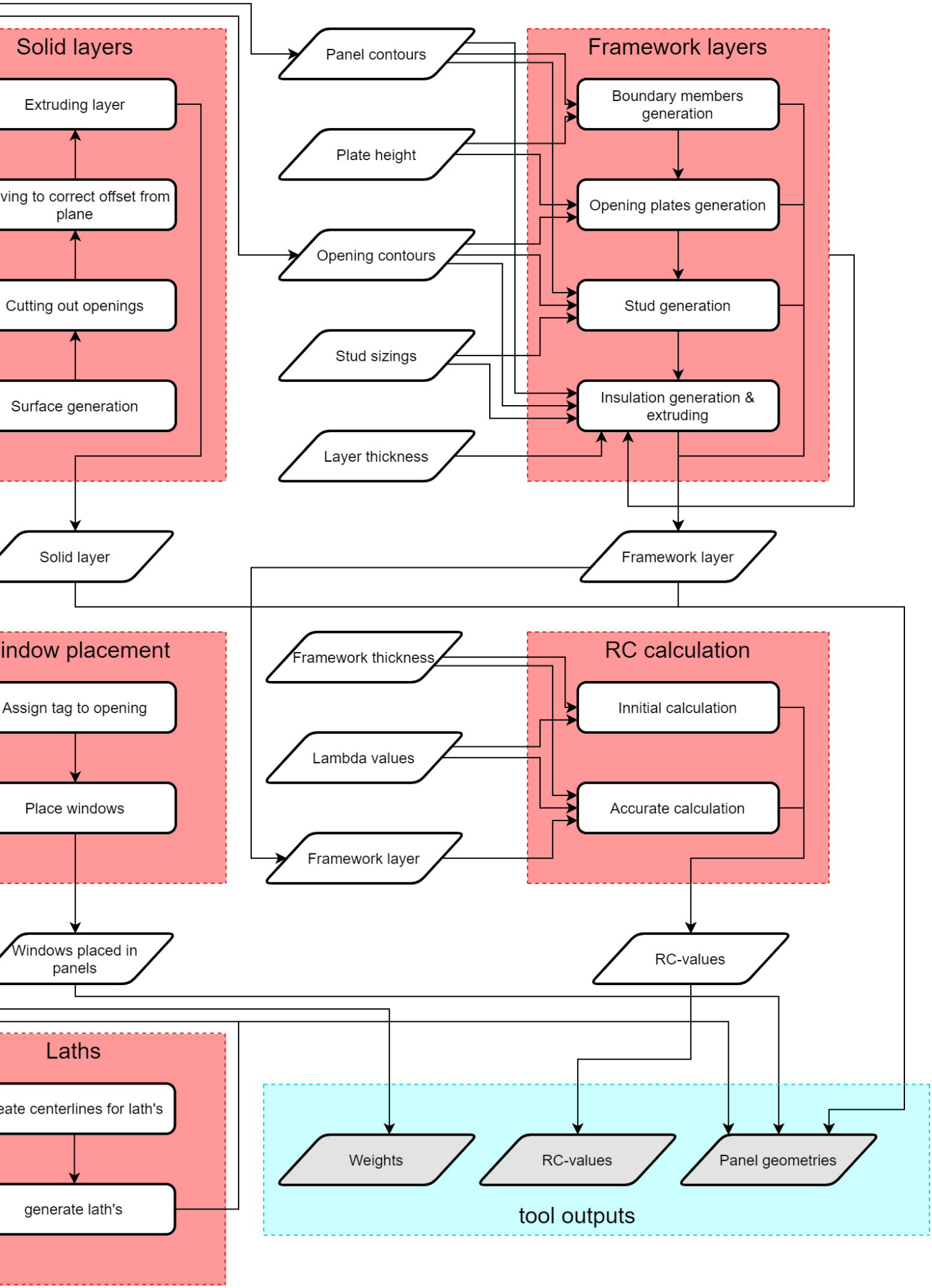


Image 95: Overview of the process gone through in the roof generation tool (source: own image)


Solid layers
Extruding layer
ving to correct offset from plane
Cutting out openings
Surface generation
Solid layer
Panel contours
Plate height
Opening contours
Stud sizings
Layer thickness
Framework layers
Boundary members generation
Opening plates generation
Stud generation
Insulation generation & extruding
Framework layer
indow placement
Assign tag to opening
Place windows
Windows placed in panels
Framework thickness
Lambda values
Framework layer
RC calculation
Innitial calculation
Accurate calculation
RC-values
Laths
eate centerlines for lath's
generate lath's
Weights
RC-values
Panel geometries
tool outputs

3.2.4. Bay windows

As already mentioned in the design chapter the bay windows are a variation on normal walls which in the light of the approach for the renovation and the generation of the different methods is only a light alteration on the existing method. So, this method is used as a basis with an alteration to the layer generation of the framework. Which allows its vertical boundary members (meaning the left and right most stud) to be put at an angle.

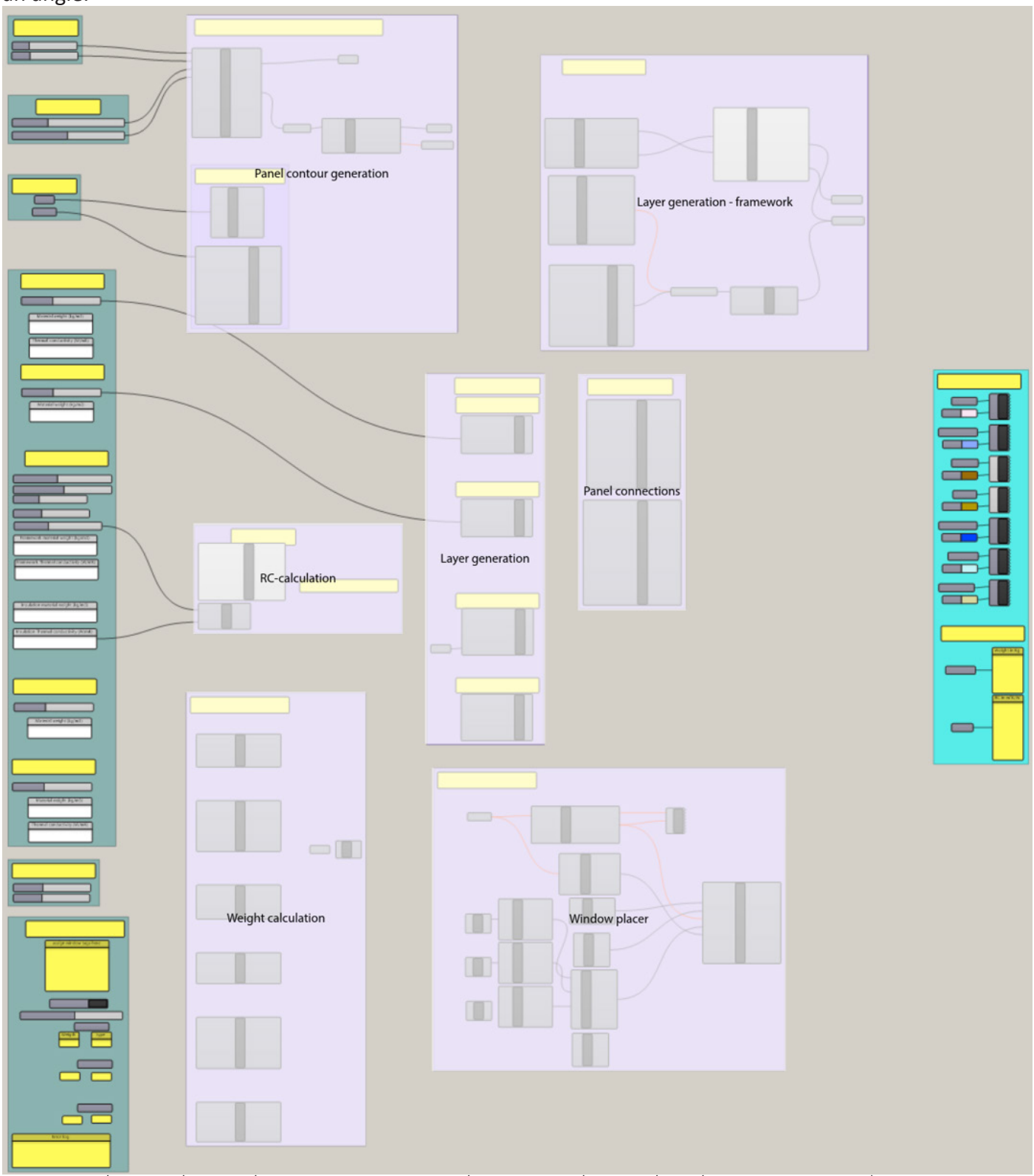


Image 96: Showing the grasshopper components used to generate bay windows (Source: own image)

Input

In order for the method to operate it requires inputs, these are listed and discussed in this section.

- *Vertical boundary members*
  Curves of the plate boundary members that will need to be adjusted to the angle of the boundary studs
- *Horizontal boundary members*
  Curves of the stud boundary members that will be rotated according to the angles
- *Angle a*
  Angle on the one side of the panel that is input user to rotate the edge of the panel.
- *Angle b*
  Angle on the other side of the panel that is input user to rotate the edge of the panel.
- *Layer thickness*
  Thickness of the framework layer, used to extrude the layer.
- *Layers before*
  Thicknesses of the layers before the framework, used to put the framework layer into the correct position.
- *Openings*
  Curves used to cut out insulation that is generated within window openings.

Process

As mentioned, the difference for bay windows is that the vertical boundary members will need to be generated at an angle for them to be able to connect to the next panel. so, the generation of the boundary members and insulation will need to be altered in order to make this function.

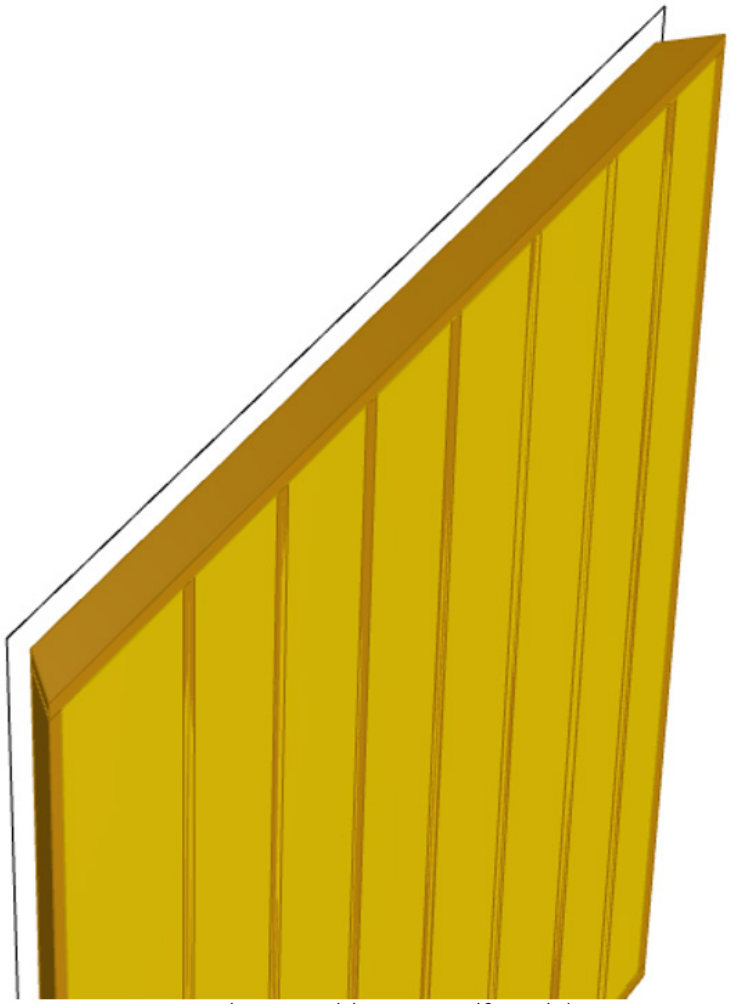


Image 97: Bay-window panel (Source: self-made)

Bay-windows - Workflow

These geometries are made using the workflow below. It takes in the boundary members to split them into horizontal and vertical members. The vertical members are the boundary studs which are rotated according to the user defined angles and the horizontals which are the plates are adapted accordingly. Then a Boundary brep Is made to create the insulation. To do so all wooden members and openings are cut out of this brep. It is similar to the original insulation generation however this one uses breps instead of surfaces.

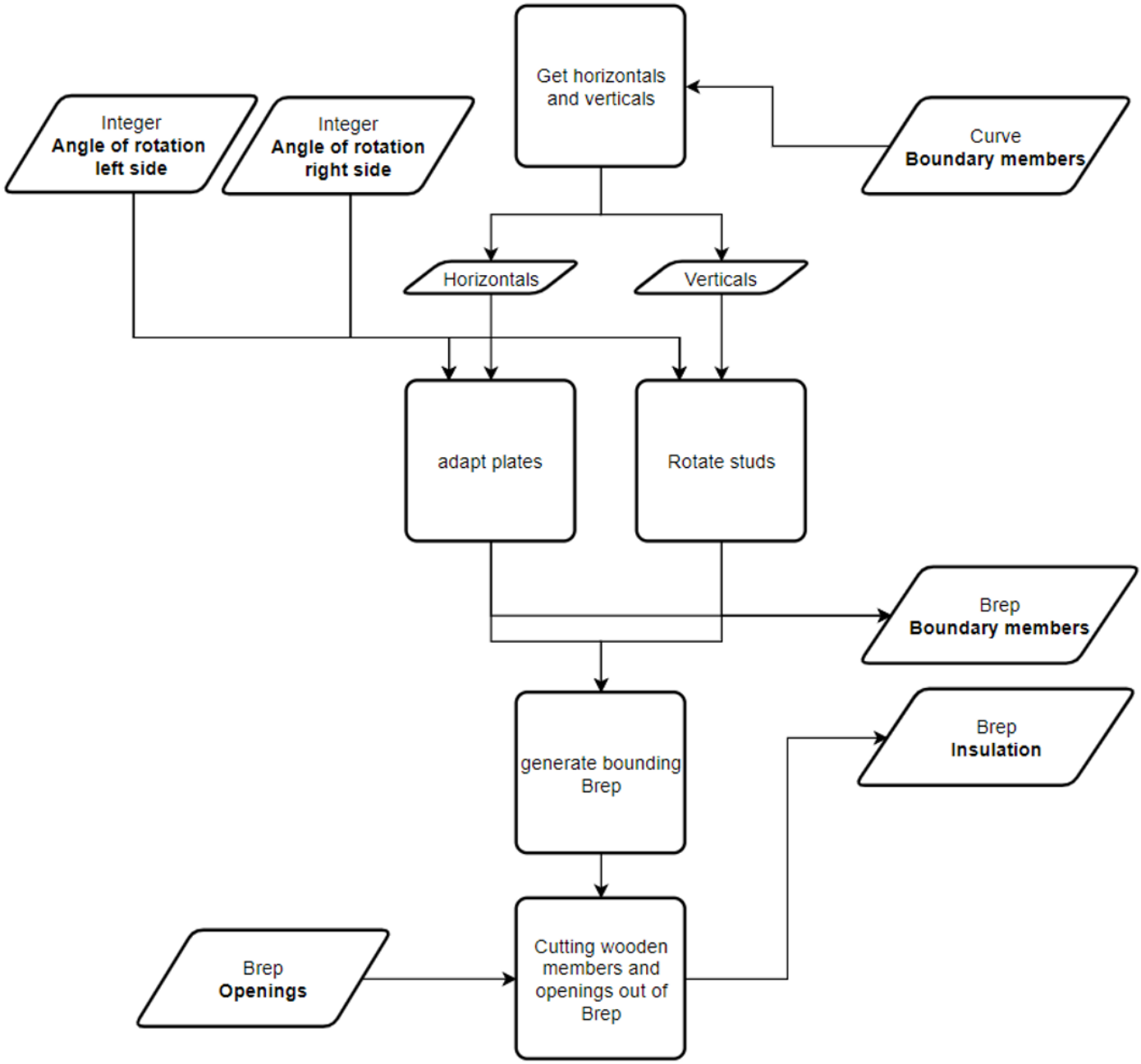


Image 98: Workflow of the generation of the angular connection (Source: self made)

Bay-windows - Grasshopper

In order to translate the workflow diagram to grasshopper the script below has been used. It takes the verticals and horizontal members of the boundary members and uses the assigned angles to rotate the studs and adjust the plates accordingly in the boundary plates generator. Using all these breps the insulation is generated and a curve that represents the exterior of the framework which is used to generate the exterior foil and finishing layer. This script can be found in appendix 7.3.

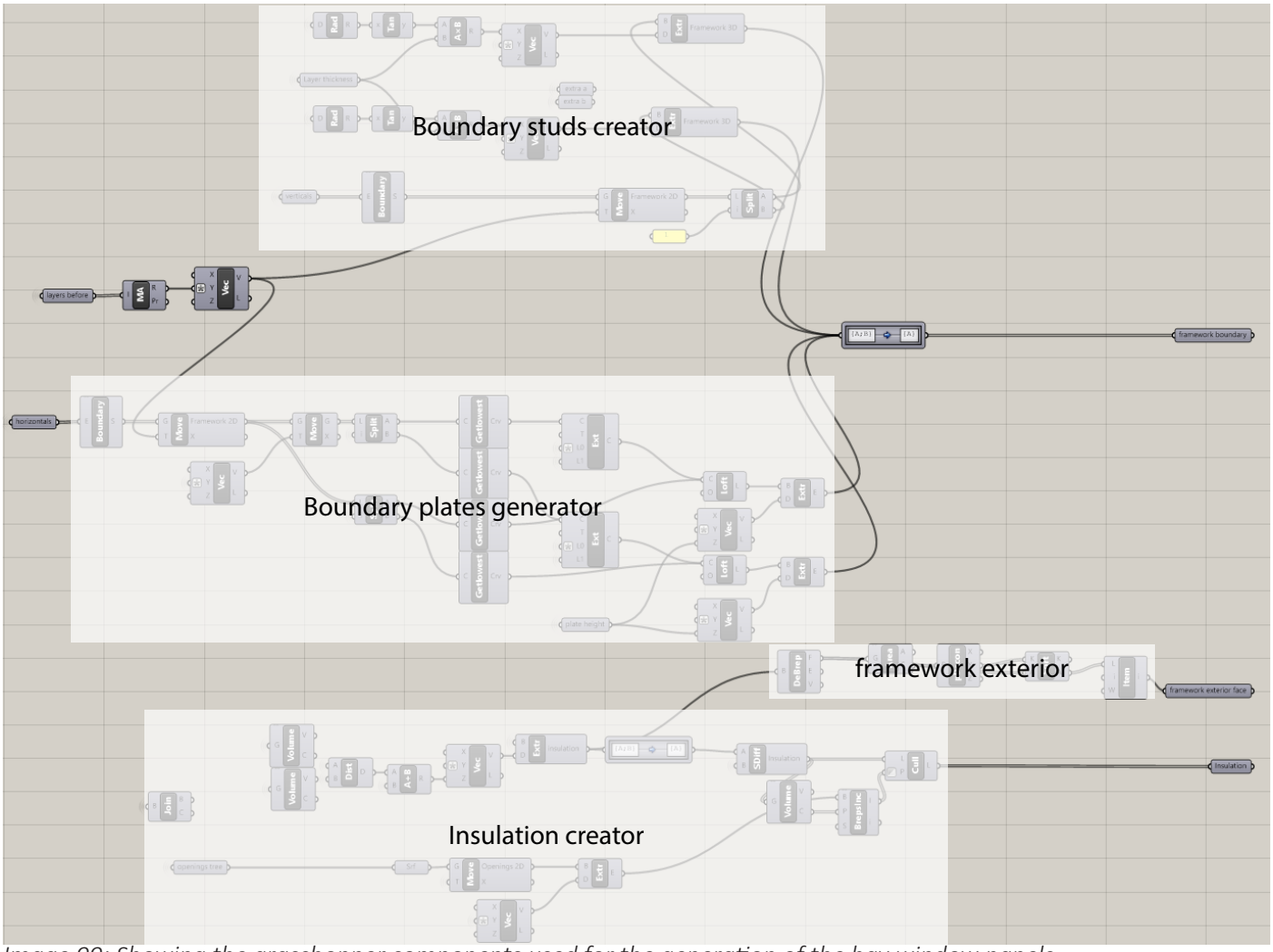


Image 99: Showing the grasshopper components used for the generation of the bay window panels (Source: own image)

3.3. Case study application

In this phase of the research the design of the panel and the tool used to apply it are tested on a case study building. Also, the process of using the tool is explained. In this way the tool is tested in a different environment than the one it was created in and this provides an opportunity to fix the last issues. This also allows for a stepped approach that can be used to apply the tool.

To apply the tool several steps, need to be taken, in this paragraph these steps are listed and explained.

Starting point

The first step of implementing the tool is the starting point, whatever this may be, it can be a model that is conceived from a 3D scan of the building, old digital drawings or newly generated drawings of the building's envelope. This information will then be transferred over to Rhino in order to be able to run the grasshopper script. In the example using in this case the building that was also used in the research: Zero Energy Building Refurbishment & Energy Neutral Urban Clusters in Haarlem by Faik Nebil Balkuv is used. This because it meets all the characteristics handled in this thesis. It is a prewar rowhouse, with facades, a roof and a bay window so every part of the tool can be tested. The RC-values and their requirements are as stated in the chapter 3.2.2. case studies.

Table 12: Existing and required Rc-values of the case study facades (Source: (Balkuv, 2017))

Envelope part	Existing	New
External Wall	0,8 m²K/W	4,5 m²K/W
Roof	1 m²K/W	6 m²K/W

Image 100: Rhino model of the rowhouses located on the Hospeslaan in Haarlem (Source: self made)

Selecting methods

In order to generate the panels for a surface first the method for the surface has to be determined, it being a wall, bay-window or roof. This is not an input for the tool however it does determine which script is used for which part of the building envelope. The image below shows the visualization of this process. with the wall panel parts shown in green, the roof in orange and the bay-window in blue.

Image 101: Different surfaces colorized by their panel generation method (Source: self made)

Creating contours

Next the plane on which the panels are to be generated needs to be determined. This depends on the straightness of the wall and the minimal clearance needed between the panel and the existing wall. For example, to mount the panels. In this thesis this distance will be 70mm as already discussed in Chapter 4.1. Design of the panel. Onto this plane the contours of the walls and openings can then be projected theses wall contours might need to be altered by the user to allow for better connections between the panels. In this example this is shown at the top of the wall which leaves a gap that allows the roof to go over the wall panel. Also, at the bay-window a gap is shown that takes into account the thickness of the wall panel.

Image 102: Wall panel and opening contours on the panel plane (Source: self made)

Panel contour generation

With all the steps before completed the user is now ready to implement the tool. Starting with importing the created contour curves in the correct tool. As well as providing the panel size limits and the tolerances. The user can then check if the panel contours are correct and if the panel distribution is to his/her satisfaction. If not the contours, tolerances or maximum panel sizes can be changed. If that doesn't provide the correct result. The user can decide to generate the panel contours and provide them to the next step of the tool. For the case study building the panel heights are set to the floor height. Which results in the panel contours shown in red in the image below.

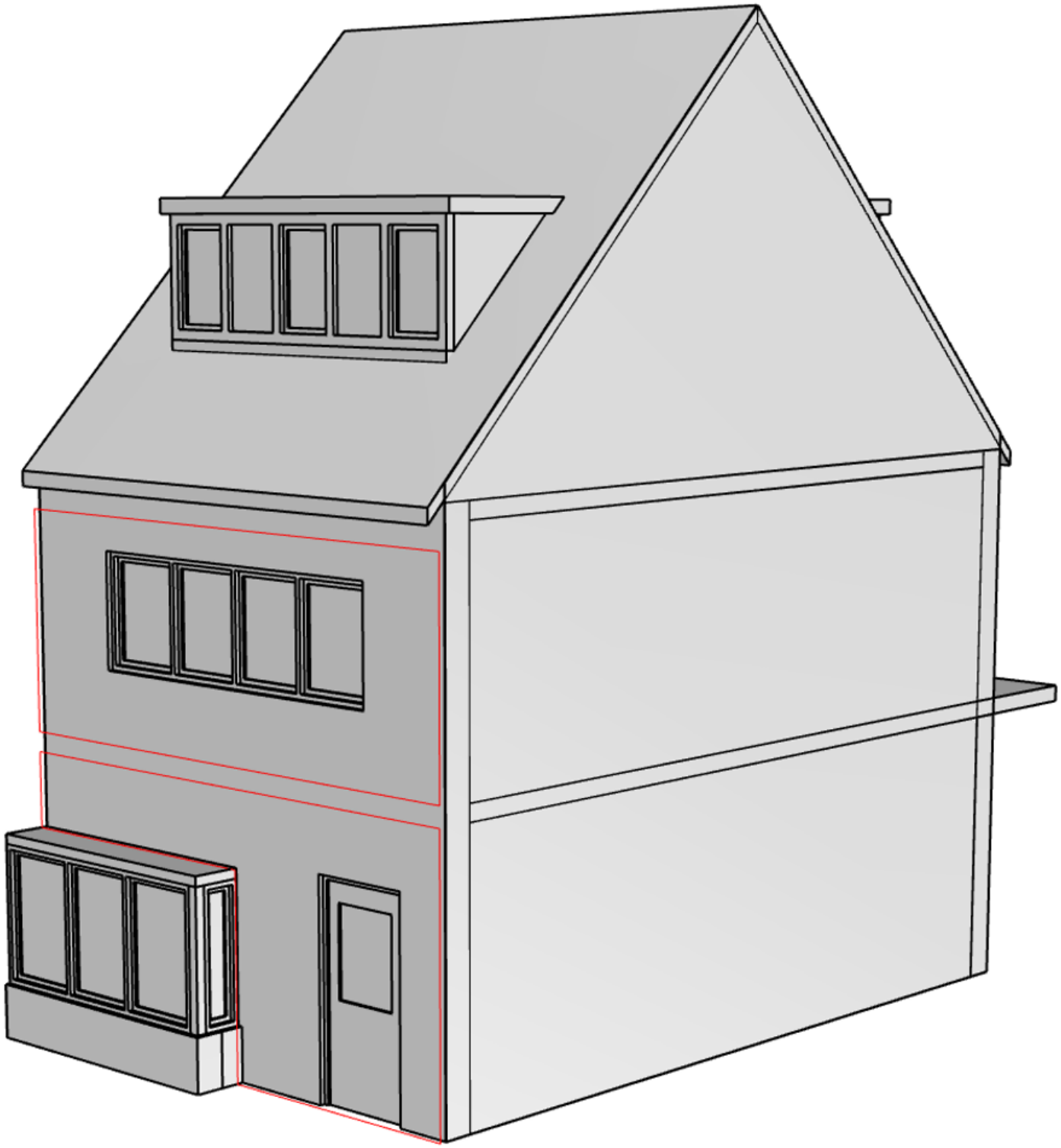


Image 103: Showing the panel contours generated by the tool on the case study building(Source: self made)

Layer specific inputs

Now that the panel contours have been made the layers can be generated. The information for the generation of these elements should be provided next. In the wall method there are five layers by default, they can be removed or added if needed. For the solid layers the thickness of the layers needs to be provided in order to generate the elements. Also, the user will need to provide the density of the layer's material in order to calculate the weight of the panel.

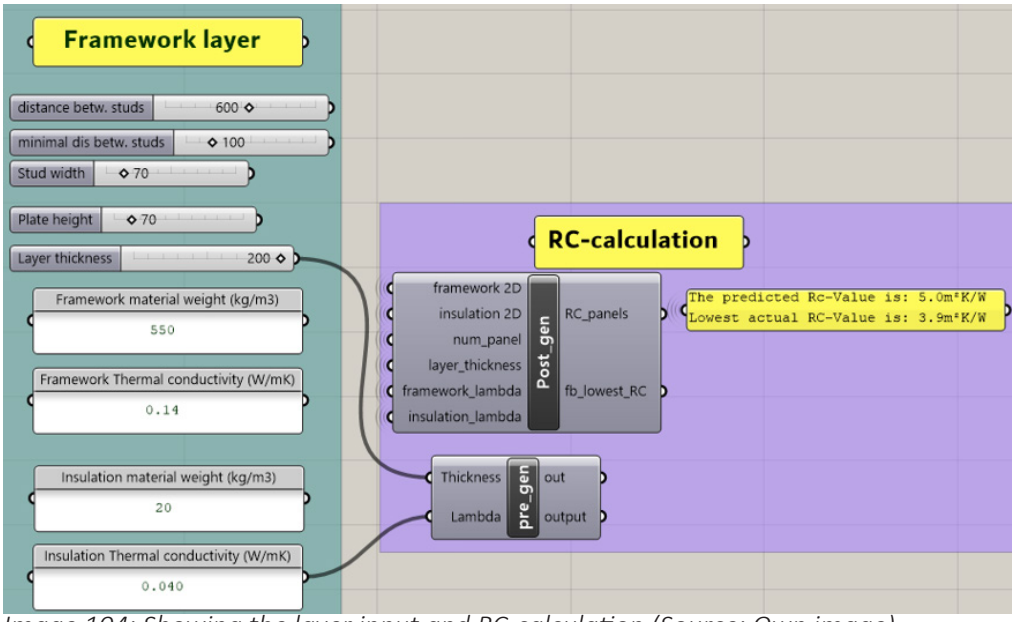


Image 104: Showing the layer input and RC-calculation (Source: Own image)

Wall panels

These inputs are also required for the framework layer and the insulation. However, in this layer also the minimum and maximum distance between the studs and their width as well as the height of the plate should be specified for the generation of these elements. The thickness of the layer and the thermal conductivity determines the RC-value of the panels. As seen below the lowest value of all panels in 3,9 m2K/W which for this case study is more than sufficient as it needs to upgrade the wall from 0,8 to 4,5 m2K/W. And with the original wall still in place and there being an insulated cavity between the original façade and the newly added panels this requirement will be more then met. The layers will then be generated as shown on the right (the exterior layers are left out to show the framework)

Image 105: Wall panel framework as generated on the exterior of the façade (Source: self made)

Bay-windows

For the generation of the bay-windows the user will also need to specify the angle of the left and right corner of the panel to ensure the most optimal connection between the panels. As can be seen below the right side of the bay-window has a 45-degree angle. While the left side is kept straight (0-degree angle).

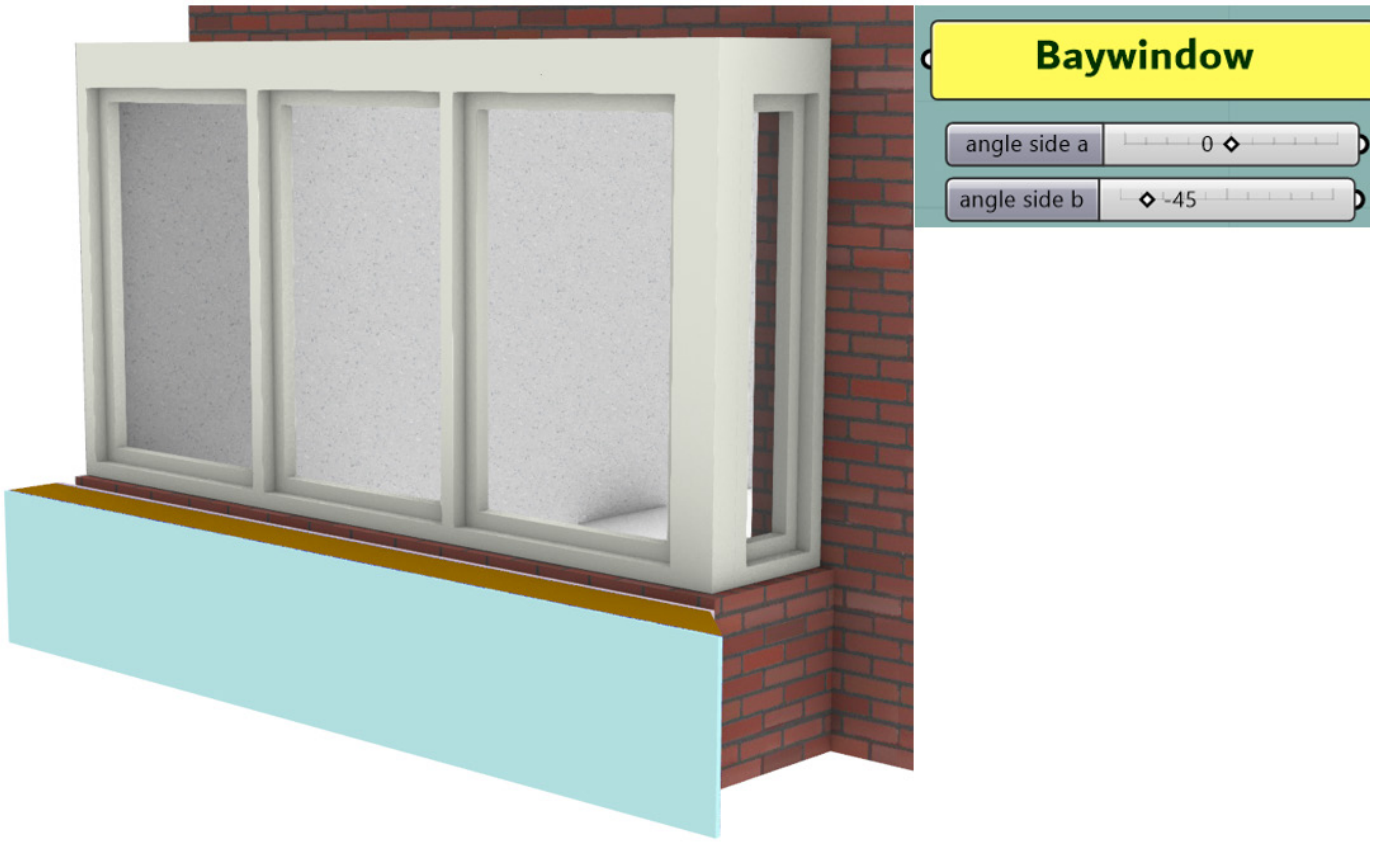


Image 106: Showing the angle on the side of the bay-window panel and its method specific input (Source: Own image)

Roof panel

For the generation of the roof panels additional inputs are also required. So, the user has to input the next inputs. For the eaves the size of the overhang has to be specified so that it might be generated. Along with the location and size of the wall beam. For the tile finish the laths sizing and interval needs to be specified. Especially the interval is important as it changes depending on the tile manufacturer and type.

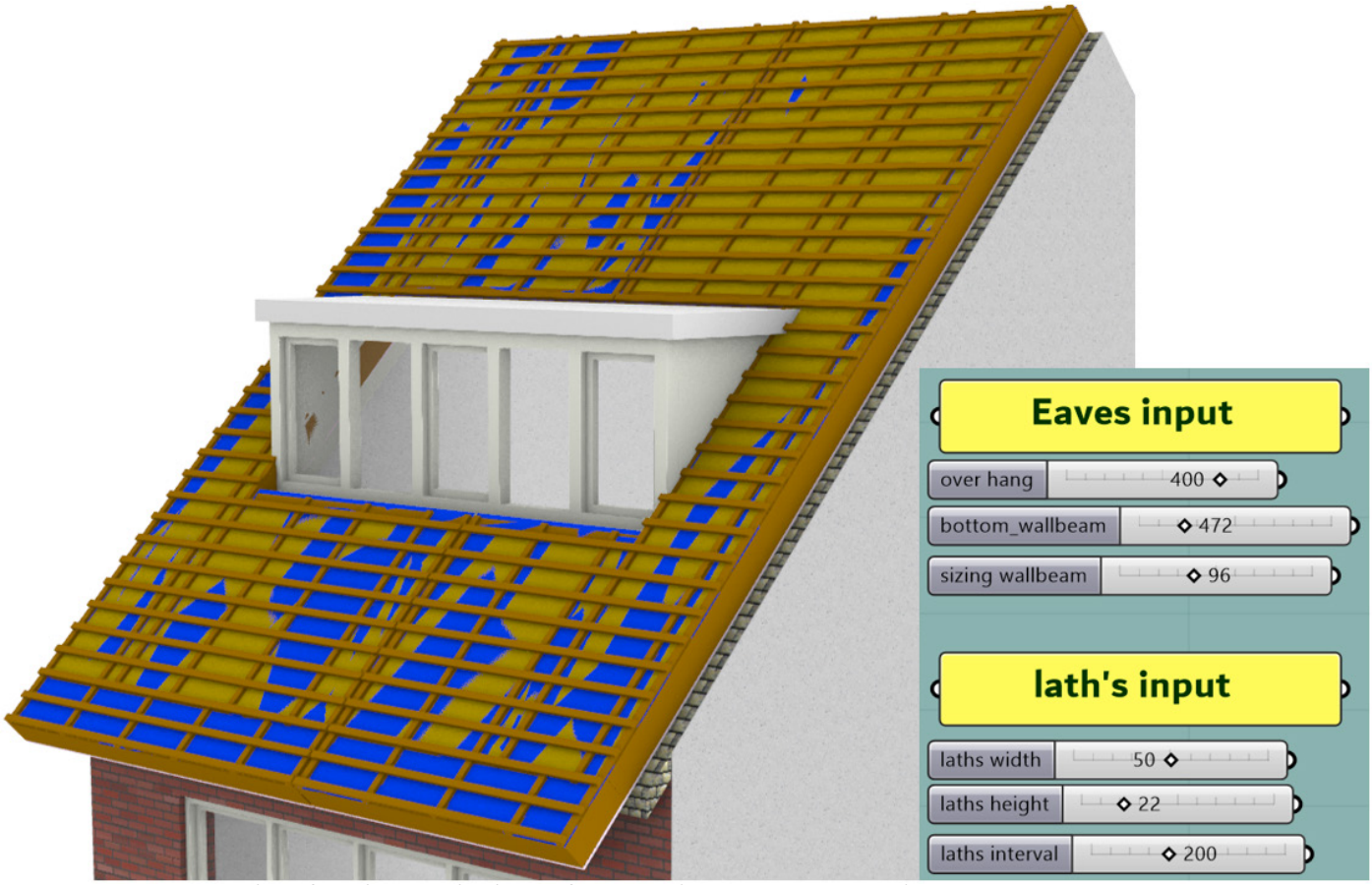


Image 107: Generated roof, and its method specific inputs (Source: own image)

Framework placement

The last input required from the user for the tool to function is for the placement of doors and windows. Before anything is input by the user the openings will show that no type has been assigned to them like below as well as the order in which they are listed and should be assigned.

Next the user should provide a geometry, weight and type per framework. As well as an offset distance from the plane. This is the distance from the inside of the panel to the inside of the framework. The user can then Assign the types to the openings. All inputs are shown below and the final result is shown in the bottom right of the page.
The error log gives feedback to the user if the tags don't match or if to many or to little are provided.

If the user wants to add more frameworks it can be done by duplicating the inputs of an existing framework

Image 108: Opening with no tag assigned to their openings yet (Source: own image)

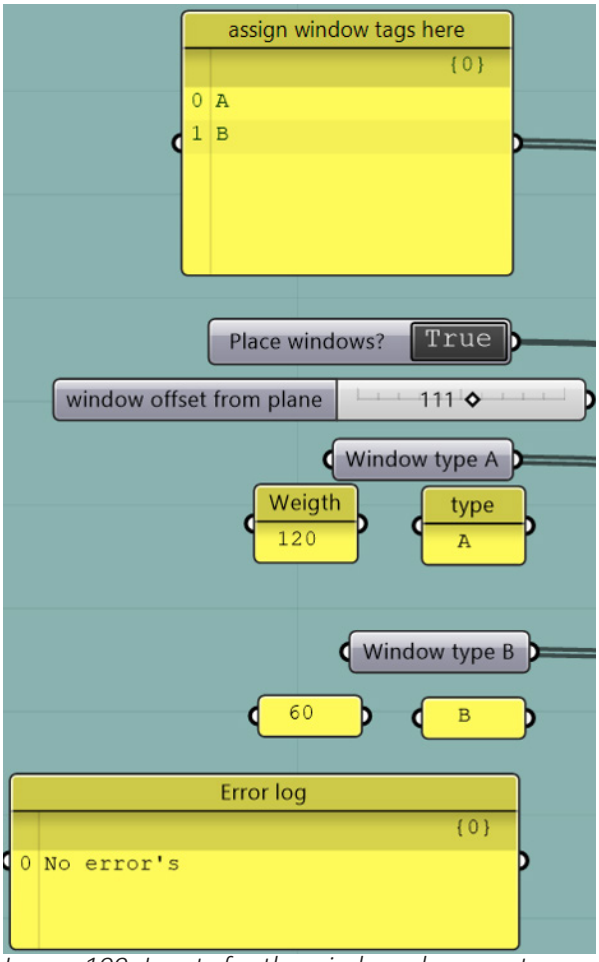


Image 109: Inputs for the window placement (Source: Own image)

Image 110: Showing the placed frameworks and their tags (Source: Own image)

Conclusion

To further clarify the process the scheme shown below has been made. It shows the steps the user needs to take and the steps the tool takes when provided the information. The information that needs to be provided is also detailed.

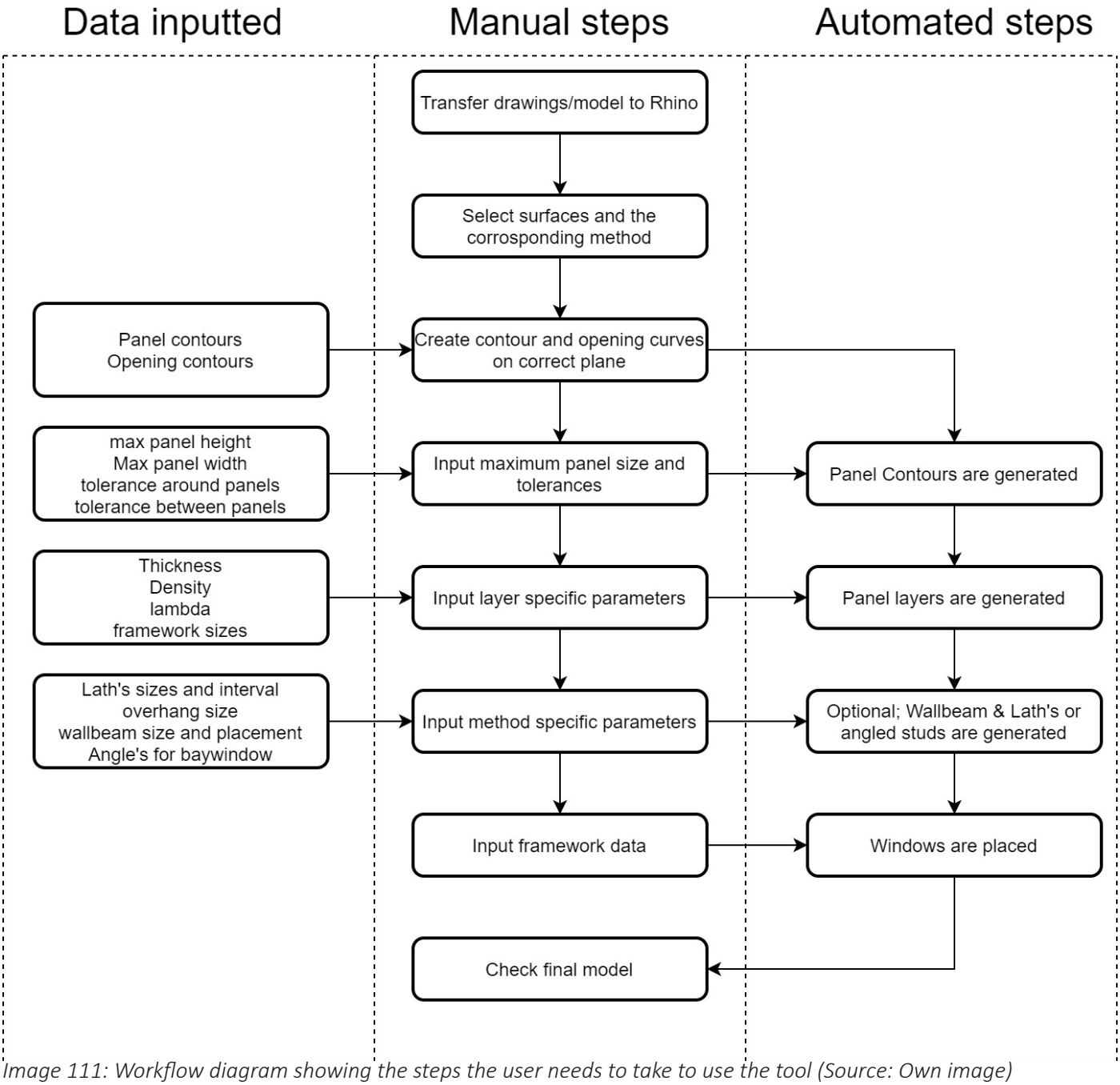


Image 111: Workflow diagram showing the steps the user needs to take to use the tool (Source: Own image)

The workflow shows that before the tool can be used the user needs to take steps to prepare everything. First gather the information and transfer it to rhino in order to make the tool able to interact with it. Then select the proper method per surface every method is made for the specific case and has limitations when applied somewhere else. The wall tool can handle complex geometries but only in its plane. The bay window can add angles but does not work with non-rectangular geometry. The roof allows for addition of lath's but only the panel width can be changed. After this the user draws the contours of the to be panelized face and their openings. In this the user already needs to take into account where he wants the panels to stop. For example, to cover the sides of panels that can be inside Porches. This input allows the most manipulation within the tool as the user can mandate where the panels begin and stop. Next the user dictates maximum panel sizes and desired tolerances so the tool might generate the panel contours. After this the layer specific parameters are inputted and the layers are generated. If the roof or bay window tool is used, additional input is required. Lastly the information about the windows is to be provided. After the tool is done the final model will need to be checked. If the user wants to make improvements, he/she might change the desired parameters accordingly.

The tools allow the user to change the sizing of all elements (framework members, exterior plating and so on) so the user is able to control a lot. However, there are limitations, only one façade can be constructed at a time using the methods. Also, it is not possible within the tool to change the sizing of a single object without changing the other related objects (For example all wooden studs have the same size). Furthermore, the connection between the façade panels has been hard coded into the script as it was designed. Lastly, inputs such as the angles for bay-windows, the length of the overhang and the wall beam are now inputted by the user. However, in the future it might be possible that these are automatically generated within the tool as it becomes a more integrated tool rather than separated parametric methods.

Post tool engineering

With that the use of the tool is concluded. However, the building process does not stop there. Before installation some other elements still need to be engineered.

One of those elements is the mounting, which has been designed in the design chapter. This isn't implemented in the tool so the user will need to take the weights and layout of the panels provided by the tool and configure where the mounts will be placed and the panel and wall.

Also, head on connections of the panels will need to be covered. The tool does do provide this option, this could also be a next step in its engineering. Before the panels can be mounted onto the façade the space between the panels needs to be insulted. For this insulation might be attached to the wall that allows for compression when the panel is mounted so no false cavity is created. Lastly the edges where the panel connects to the existing structure need to be engineered and applied on site. These are: the rigid insulation panel below the wall panels to prevent water from getting to the panel from the ground. The reveals that need to be finished to cover the old wall. The connection between the wall panel and the roof panels and the ridge once the roof is placed. All these elements have been discussed in the panel design chapter.

4. Discussion

In the research the process of engineering a prefab renovation is dissected in order to try and compose a paramtric way of generating prefab facade panels to speed up the engineering. The steps in the process are derived from existing research papers on the process but the process of generating a panel and generating is a product of this paper. To determine if the tool can provide an answer to the research question and to asses its validity a swot analysis has been made.

SWOT analysis

A SWOT analysis is a way of assessing the (S)trengths, (W)eaknesses, (O)pportunities & (T)hreats of a product. The strengths and weaknesses address the internal positives and negatives while the opportunities and threats cater towards the external positives and negatives.

For this thesis the scheme below shows the components of the SWOT analysis which are further discussed in the rest of this chapter.

Table 13: SWOT analysis scheme (Source: self-made)

Internal	**Strengths**	**Weaknesses**
	New & Faster approach to generating	Code isn't written as effective as possible
	Customizability	Overwhelming for inexperience users
	Immediately visual feedback	still needs to be check by the user
External	**Opportunities**	**Threats**
	can be applied on millions of renovation cases	Using the tool requires the user to go through a learning curve first
	Can be transferred to other applications in the build environment	Architects can view it as generalizing the façade design and restricting their architectural freedom
	can be adapted to companies specific building methods	

Strengths

the tool sets itself apart by its new and fast approach of the atomization of prefab building envelope panels. By this it allows a user to rapid prototype different façade layouts in order to find the best possible layout. Furthermore, because the tool is made in grasshopper and the components are clearly divided it allows for customization of single elements which allows it to be upgraded are changed entirely.

Weaknesses

The weaknesses of the tool are that it uses python code which is written by someone who didn't have any experience in python prior to starting the thesis. So, it isn't as efficient as it possibly can be. Furthermore, the grasshopper script and its components can seem overwhelming to inexperienced users. As there is a lot going on, on the screen. Lastly the tool is catered towards the generation of timber frame so in this way it is limited to generating other panels and the panels created will need to be checked by an engineer before being transferred to manufacturing.

Opportunities

The main big opportunity for the tool is in it catering towards a fast solution for the generation of prefab renovation panels. There are millions of houses that need to be renovated and the majority of them can be engineer with a tool like this one. It might even be applied (with some changes) to other building typologies or buildings outside the Netherlands. Furthermore, it might also be that the techniques used in the tool can be translated to other applications within the build environment.

Threats

The tool automizes the application of the engineering of a certain building method. Because of this the variation between the panels is limited (within the domains). Even though the panels that are constructed

allow for a large variety of finishing due to the lay out this can still be limiting for the materials used for this finishing. Also, the tool might work in a test environment and on a case study but still the implementation of it being the maker of the tool means one knows exactly how every part works and functions. However, for new users there will be a learning curve before the tool can be used properly.

SWOT Conclusion

As it is designed the tool does what it is supposed to do. It generates the panel 3D elements and provides the user with instant feedback in the form of visuals and data. All while allowing for the customizability of the input that can differ per company. With this it can help speed up the engineering of the millions of renovation panels that need to be made. While even allowing further customization because of its transparency.
However, there are some obstacles to overcome before the tool could function on a large scale. Which are; the code used in the tool isn't as effective as it possibly could be so corner cases that didn't come to light in this research might not be solved in the correct way. This would be the next step in engineering the tool for use in the market.
Further testing needs to be done to see if the tool provides the correct panel everything before it can be sent directly to the manufacturing site. Rather than first be reviewed by an engineer. Also, the steps that the engineer needs to take after the tool has done its job can be automated. This can all be part of further research onto the implementation of the tool.

Furthermore, architects can view the panels as a generalization of the façade and feel restricted in their architectural freedom as the materials and mounting systems to be used are limited by the panel system. However, there is still a lot of freedom in the façade design and the façade design of the current row houses doesn't use any materials that cannot be used on the exterior of the panels. And with next versions of the tool and added features the freedom of design will only increase.
Lastly the script might look overwhelming to users that do not have experience with grasshopper and even grasshopper users will first need to find their bearings before they will be able to customize the tool. What could help for users of the tool might be an interface rather than working with the actual grasshopper file.

5. Conclusion & Recommendation & Reflection

This thesis tries to provide a partial solution to the problem of having all houses disconnected from natural gas before 2050. It does so by trying to speed up the engineering by automating it. In order to provide this solution, the following question was composed. How can a parametric tool be designed and what parameters are used for the renovation of poor energy performing buildings in order to provide designers and engineers with a building information model of prefabricated adjustable building's envelope panels?

Conclusion

The scope of the renovation of these houses is set on row houses as they consume 47% of all energy required for heating all the houses in the Netherlands. This typology has been further narrowed down to pre-war row houses as the research showed this is the largest energy consumer in the category with 30% of all row houses. The Approach chosen for the renovation of these houses is the "wrap-it" approach in which the exterior of the house will be insulated. This was chosen because it is the least interfering process for the inhabitants and the most efficient for using prefab panels. Cases have been studied in order to generate the innitial design which later would be translated to a computational method. To give suggestions on how this could be taken to a parametric tool literature and case studies have been conducted. Which gave a clear indication of how to formulate a process in order for it to be automized.

So the engineering process has been evaluated and broken down to the steps that make up the process. Which are: 1) The preliminary design stage, in which the initial design is made and boundaries aswell aswell as goals are set. 2) Engineering, in which the final design is engineered and made ready for production and assembly. 3) Manufactory, in which the elements needed for the realisation are made and transported to the building site. 4) Assembly, in which the design is executed. 5) Maintanance, the period after assembly untill end of life in which the building is monitored and kept in prestine condition to ensure a long lifespan. This already showed a clear flow of information in which every step is dependent on the previous steps in the process. But, a good executable design will need communication in two directions so that the project may be designed for manufacturing and assembly rather then trying to fix design flaws in later stages. For the creation of a parametric tool further detail into the engineering and post engineering steps was necessary. This in order to determine which are the steps that are to be completed in the engineering phase of the project as well as to analyze which can be automated by a parametric method.

In order to provide an engineer with a building information model geometry and its required data needs to be generated. For this research RC-values and weight has been deemed as required data. The research has shown that the generation of the panels happens in two steps. First the panel contour generation which determines within the set limit what the boundary of the panels will be. After which the panels are generated. For the panel contour generation, the following three inputs are essentials: 1) Façade and opening contours, in oder to be able to divide de facade into panels. 2) Maximum panel sizes, to set a limit to the size of the panel. 3) tolerances, to ensure enough space between the panels so the design can be executed.

Together these inputs allow for adjustable panel contours. In oder to be manufactured these need to be filled with geometry of there individual elements. For the generation of these elements the information of those components is essential. For the solid layers this is only there thickness so that they may be created and their density and lambda to provide the weight and RC data.

However, for the framework it is more complicated. Every component sizing's needs to be input. All inputs for this layer are: its Thickness, the plate sizing's in order to generate the plates at the panel boundary and above and below windows. The stud sizing and its minimum and maximum interval distance so they can be placed everywhere. Also, the openings are needed here to place the wooden members at the correct position. For these members also their density and weight and lambda are required to provide the weight and RC data.

The roof requires additional input for the length of the overhang and the location and sizing of the wall beam so it can be generated. Furthermore, sizing's and intervals of the lath's are to be provided for the roof generation. When the bay window method is used an additional input is the required angle of the wooden boundary member at each side to be generated.

Lastly the windows placement requires the user to provide the geometry, weight and tag per window unique window type and assign tags to openings and a preferred offset for the windows. Those are all the steps and parameters needed to generate the building envelope renovation panels.

To conclude with an answer to the research question:
A parametric tool can be designed by taking the rudimental steps of the engineering process which are: determining the to be panelized area, composing the panel contours, creating the panel geometries and calculating the panel specific data. These steps are to be automated by using the step specific parameters which are: the panel size limit, tolerances and geometry sizing and properties. These are decided by the engineer or are already established in the pre-engineering phase. Together they enable the tool to generate building information models of prefabricated building envelope panels which can be adjusted by its user if required.

Recommendation

The research shows the potential of using parametric tools for the generation of prefab renovation panels. However it also exposed areas which need more attention. As it stands now the tool works within the design and test environment of the research. But there are still improvements to be made before it can be applied in the working environment especially in the following areas: robustness, user-experience and customizability.

Robustness
The tool and its code is only test on a small number of walls/facades. In order for the tool to be able to be implemented it will need to be tested and adapted to work with all different kinds of facade sizes, orientations and layouts. Furthermore the python coding used in this research is mostly self taught and not reviewed by an other coder because of this unexpected bugs might occur and the script itself could be more efficient.

User-experience
The tool is a hybrid between grasshopper components and python components used within grasshopper. In order to operate it the grasshopper script must be used. This requires users not familiar with grasshopper to learn the basics and experienced grasshopper users to learn how the script itself functions before being able to apply it. To cut this learning curve an user interface might be used which only allows it user to see and alter the parameters he needs to get to the desired output rather then seeing and dealing with the complete script all the time.

Customizability
Although the methods within the tool are build in a way that each component has its own task and these are clearly separated in order to allow the user to update certain components. The tool as a whole does not allow much customizability when it comes to other building components then walls, roofs and bay-windows. Furthermore, it can only generate timber frame elements. If this building method is not chosen the tool cannot be used as of now. If this is desired methods for the other approaches also need to be created. This is also true for the architectural finishing of the facade. The current version of the tool requires the engineer to input the sizings and limitations of the substructure into the tool. While doing so the engineer needs to take into account the finishing the architect would like to use on the facade. A feature to be added might be a component that helps the architect with adjusting the sub-structure to the aesthetical facade design.

Reflection

This research not only tries to answer the question of how a parametric tool can help an engineer speed up the design process. It comes from a bigger interest; taking the parametric scripting which is mostly used in the design and make it more practical. The goal of the research was not necessarily to make a tool but to show how this process is done. Starting with analyzing the design chain and then focussing on one step of the process. To further break it down into the steps done by the responsible person in this step and to determine what is his input and what is it based on. And then when this is clear it provides a clear guideline for someone to make the tool.

6. References

2ND skin. (2020, 12 11). 2ND skin whitepaper. Retrieved from 2ND skin: https://www.2ndskin.nl/download-whitepaper/

Aerts, C. (2020). Automated robotic manufacturing for building prefabrication. Delft: Technical University Delft.

Agentschap NL. (2011). Voorbeeldwoningen 2011. Den Haag: Ministerie van Binnenlandse Zaken en Koninkrijksrelaties.

Aldanondo M., B.-S. A. (2014). Towards a BIM Approach for the High Performance Renovation of Apartment Buildings. Paris: IFIP.

Austern G., G. C. (2018). Rationalization methods in computer aided fabrication: A critical review. Haifa, Israel: Elsevier.

Balkuv, F. (2017). Zero Energy Building Refurbishment & Energy Neutral Urban Clusters in Haarlem. Delft: Delft University of Technology.

Barco A.D., V. E. (2016). Building renovation adopts mass customization - Configuring insulating envelops. New york: Springer Science.

Belzen, T. v. (2019, 10 8). Revolutionaire bouwers dolblij met stikstof: 'Dit kan weleens dé motor van verandering zijn'. Retrieved 11 8, 2020, from Cobouw: https://www.cobouw.nl/duurzaamheid/nieuws/2019/10/van-stikstofnegatief-naar-biobased-en-modulair-dit-kan-weleens-de-motor-van-verduurzaming-zijn-uw-101277340

CBS. (2013, 1 21). Twee derde van alle woningen eengezinswoningen. Retrieved from Centraal bureau voor de statestiek: https://www.cbs.nl/nl-nl/achtergrond/2013/04/twee-derde-van-alle-woningen-eengezinswoning#:~:text=De%20gemiddelde%20vloeroppervlakte%20van%20woningen,keer%20zo%20groot%20als%20appartementen.

CBS. (2020, October 6). Voorraad woningen. Retrieved from CBS: https://opendata.cbs.nl/statline/#/CBS/nl/dataset/82550NED/table?fromstatweb

CBS. (2020, 11 23). Werkenden in de bouw. Retrieved from cbs.nl: https://www.cbs.nl/nl-nl/maatwerk/2018/34/werkenden-in-de-bouw

Chen K., L. W. (2018). Design for Manufacture and Assembly Oriented Design Approach to a Curtain Wall System: A Case Study of a Commercial Building in Wuhan, China. Hong Kong, China: MDPI.

Colinart T., B. M. (2019). Building renovation with prefabricated ventilated facade elements: A case study. Lorient: Elsevier.

Collins. (2020, 11 24). Collins. Retrieved from Collinsdictionary: collinsdictionary.com

de Leeuw, M. (2020, 07 20). Bouwdirecteur over verzakt dak Heesch: '80 procent van de nieuwbouw heeft deze constructie'. Cobouw. Retrieved from https://www.cobouw.nl/bouwkwaliteit/nieuws/2020/07/bouwdirecteur-over-verzakt-dak-heesch-80-procent-van-de-nieuwbouw-heeft-deze-constructie-101286818

Ebbert, T. (2010). RE-FACE: Refurbishment Strategies for the Technical Improvement of Office Façades. Delft: Delft University of Technology. Retrieved from http://resolver.tudelft.nl/uuid:b676cb3b-aefc-4bc3-bbf1-1b72291a37ce

Filippidou F., N. N. (2017). Are we moving fast enough? The energy renovation rate of the Dutch non-profit housing using the national energy labelling database. Delft: Elsevier.

Gerrits, J. (2008). Draagconstructies Basis. Delft: TU Delft.

Hagentoft, C. (2003). Introduction to building Physics. Lund, Sweden: Studentlitteratur AB.

Holzer D., H. R. (2007). Parametric Design and structural Optimisation for Early Design Exploration. International journal of architectural computing, 625-643.

HU S.J., Z. X. (2008). Product variety and manufacturing complexity in assembly systems and supply chains. Michigan: Elsevier.

Jingmond M., L. T. (2010). Identifying Causes of Additional Cost in Tolerance Compliances Failure in Buildings. Salford, United Kingdom: TG65 & W065-special Track 18th CIB World Building Congress (p. 554).

Joostdevree. (2021, 02 25). warm dak. Retrieved from joostdevree: https://www.joostdevree.nl/shtmls/warm_dak.shtml

Knaack U., C.-K. S. (2012). Prefabricated systems: principles of construction. Brickhäuser: Walter de Gruyter.

Konstantinou, T. (2014). Facade Refurbishment Toolbox: supporting the design of Residential Energy Upgrades. Delft: TU Delft.

Künzel H.M., K. K. (1996). Calculation of heat and moisture transfer in exposed building components. Holzkirchen, Germany: Pergamon.

Lubbers, W. (2021, 02 25). De stap naar volledige houtskeletbouw-woningen is kleiner dan je denkt. en de urgentie is groter dan ooit. Retrieved from hout. Natuurlijk van nu.: https://www.houtnatuurlijkvannu.nl/de-stap-naar-volledige-houtskeletbouw-woningen-is-kleiner-dan-je-denkt-en-de-urgentie-is-groter-dan-ooit/

Majcen D., I. L. (2013). Theoretical vs. actual energy consumption of labelled dwellings in the netherlands: Discrepancies and policy implications. Delft: Elsevier.

Mjörnell, K. (2016). Experience from Using Prefabricated Elements for Adding Insulation and Upgrading of External Facades. Singapore: Springer Science+Business Media.

Montali J., O. M. (2018). Knowledge-Based Engineering in the design for manufacture of prefabricated facades: current gaps and future trends. Dartford, United Kingdom: Taylor & Francis.

Nederlands Normalisatie-instituut. (1990). NEN 2881:1990 Maattoleranties voor de bouw. Delft: Nederlands Normalisatie-instituut.

Nederlands Normalisatie-instituut. (1995). Maattoleranties voor de bouw Instructies en voorbeelden voor de berekening. Delft: Nederlands Normalisatie-instituut.

Pihelo P., K. T. (2017). nZEB Renovation with Prefabricated Modular Panels. Trondheim: Elsevier.

Raab Karcher. (2021, 1 29). prefab beton wanden. Retrieved from Raab Karcher: https://www.raabkarcher.nl/prefab/wanden/betonwanden

Richard, R. (2004). Industrialised building systems: reproduction before automation and robotics. Montréal: Elsevier.

RVO. (2020, Maart 1). Energielabels woningen 2010-2019. Retrieved from Compendium voor de Leefomgeving: https://www.clo.nl/indicatoren/nl0556-energielabels-woningen

Silvester S., K. T. (2016). 2nd skin: Zero energy apartment renovation via an intergrated facade approach. Delft: Technical University Delft.

Sun Y., W. J. (2020). Constraints Hindering the Development of High-Rise Modular Buildings. Perth, Australia: MDPI.

Tan T., L. W. (2020). Construction-Oriented Design for Manufacture and Assembly Guidelines. Londen, United Kingdom: American Society of Civil Engineers.

Tan T., M. G. (2019). BIM-enabled Design for Manufacture and Assembly. Lonon, United Kingdom: University College London.

Veld, P. O. (2015). MORE-CONNECT: Development and advanced prefabrication of innovative, multifunctional building envelope elements for modular retrofitting and smart connections. Maastricht: ELSEVIER.

Vis M.W., R. P. (2014). Cascading in the wood sector. BTG biomass technology group B.V. Den Haag: Netherlands Enterprise Agency RVO. Retrieved from http://www.btgworld.com/nl/nieuws/cascading-wood-sector-final-report-btg.pdf

Woodbury, R. (2010). Elements of parametric design. Retrieved from http://cw.routledge.com/textbooks/9780415779876/parametric.asp

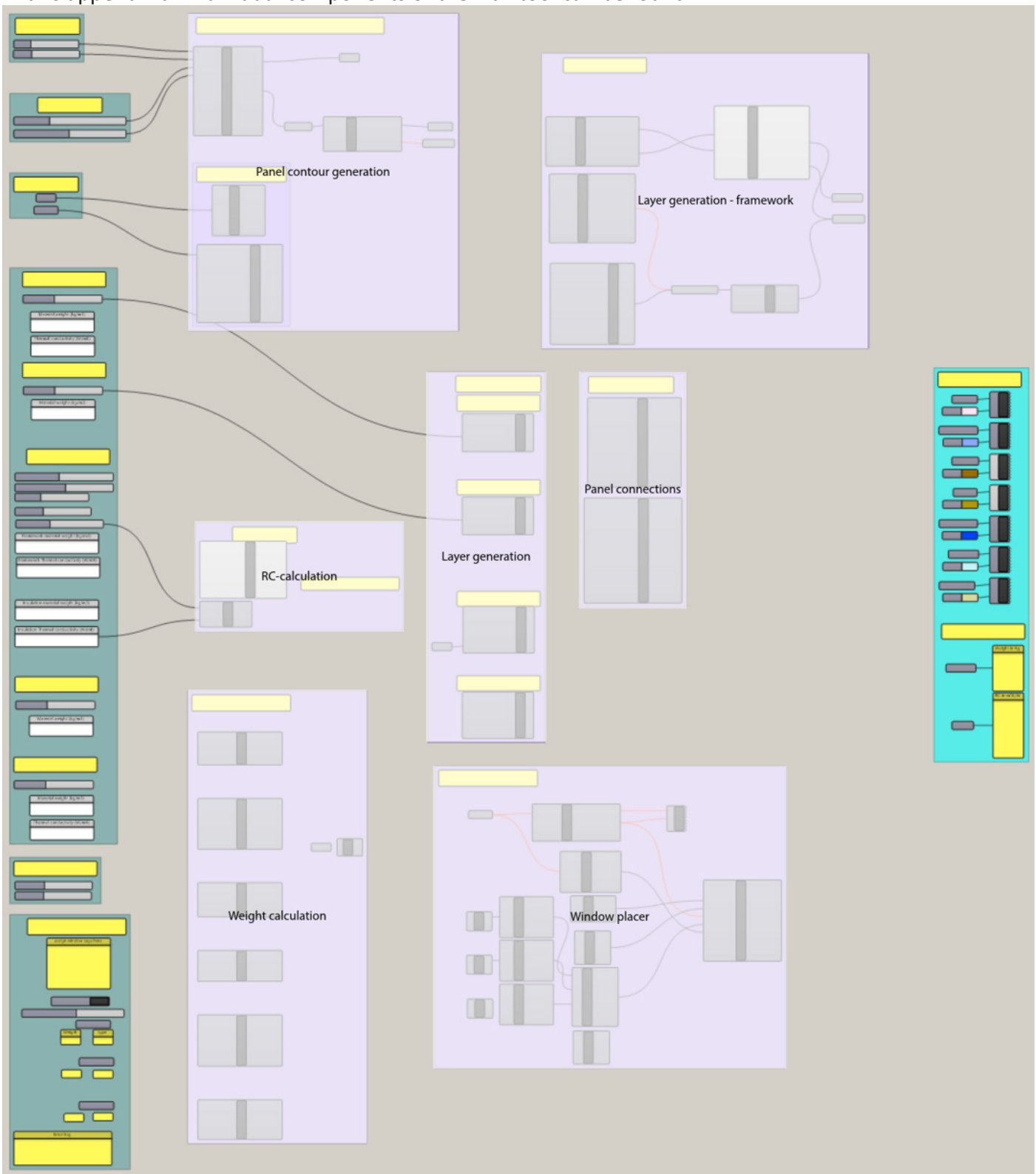
Woude van der, D. (2005). Jellema 4A Omhulling - Prestatie-eisen/Daken. Utrecht/Zutphen: ThiemeMeulenhoff.

Woude, v. d. (2004). Jellema 3 - Draagstructuur. Utrecht/Zutphen: ThiemeMeulenhoff.

7. Appendix

7.1. Wall tool script

In this appendix all individual components of the wall tool can be found.



7.1.1. Domain generation script

This script finds the coordinates of the openings so that they can prevent panel boundaries from crossing windows

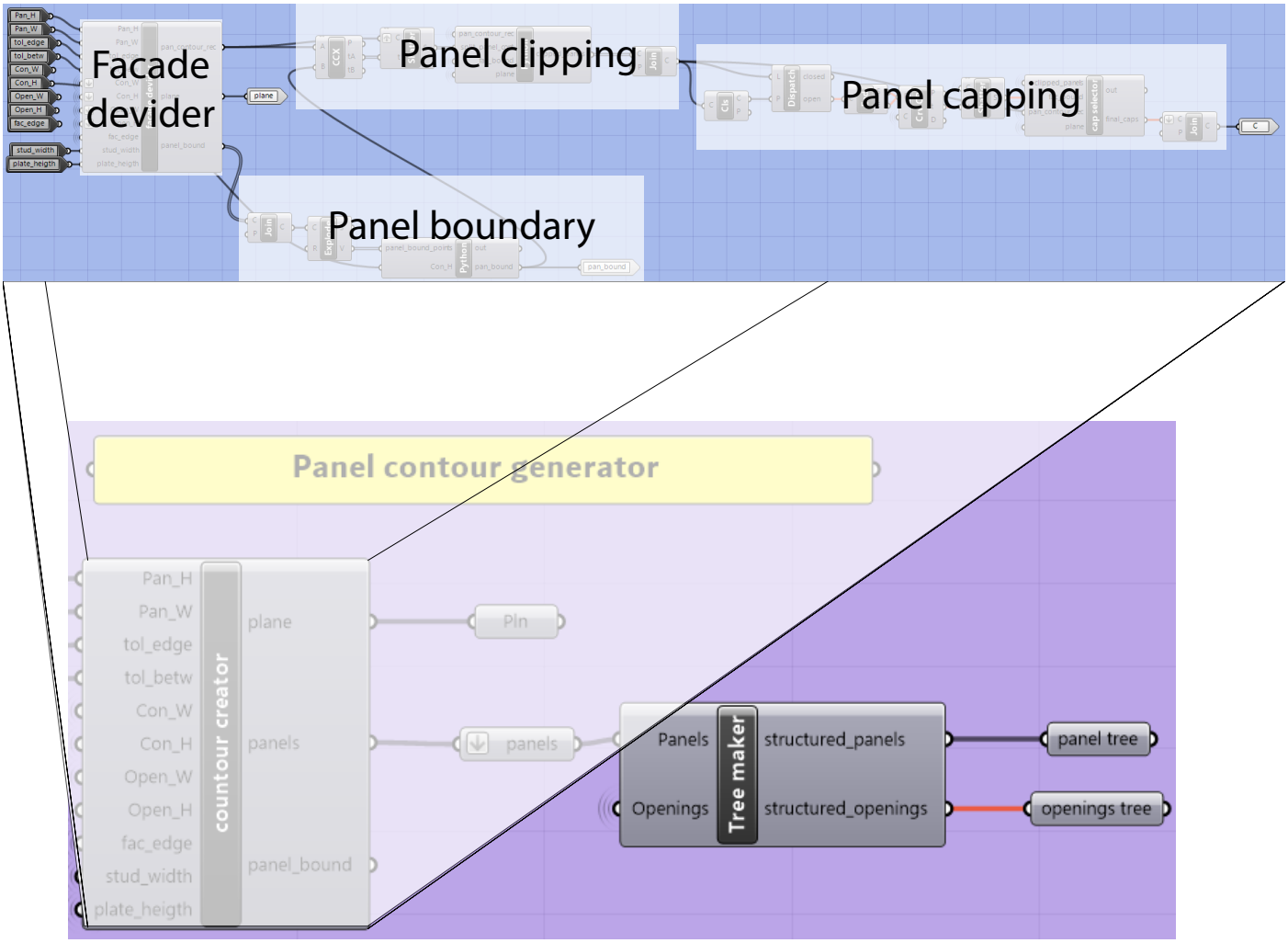
```
def find_extremes(curves):
    parameter = Rhino.Geometry.Curve.DivideByLength(curves, 1, True)
    points = []
    for para in parameter:
        temp_point = Rhino.Geometry.Curve.PointAt(curves, para)
        points.append(temp_point)
    Point_X = []
    Point_Y = []
    Point_Z = []
    for point in points:
        Point_X.append(round(point[0]))
        Point_Y.append(round(point[1]))
        Point_Z.append(round(point[2]))
    Extremes_X = min(Point_X), max(Point_X)
    Extremes_Y = min(Point_Y), max(Point_Y)
    Extremes_Z = min(Point_Z), max(Point_Z)

    return Extremes_X, Extremes_Y, Extremes_Z

extremes_combined_openings = []
for open in openings:
    extreme = find_extremes(open)
    for item in extreme:
        if item[0] == item[1]:
            pass
        else:
            extremes_combined_openings.append(item)

Open_W = []
Open_H = []
for i, extremes in enumerate(extremes_combined_openings):
    if i%2 == 0:
        Open_W.append(extremes)
    elif i%2 == 1:
        Open_H.append(extremes)
```

Very similar to the last script this script finds the coordinates of the boundary of the façade.

```
def find_extremes(curves):
    parameter = Rhino.Geometry.Curve.DivideByLength(curves, 1, True)
    points = []
    for para in parameter:
        temp_point = Rhino.Geometry.Curve.PointAt(curves, para)
        points.append(temp_point)
    Point_X = []
    Point_Y = []
    Point_Z = []
    for point in points:
        Point_X.append(round(point[0]))
        Point_Y.append(round(point[1]))
        Point_Z.append(round(point[2]))
    Extremes_X = min(Point_X), max(Point_X)
    Extremes_Y = min(Point_Y), max(Point_Y)
    Extremes_Z = min(Point_Z), max(Point_Z)
    return Extremes_X, Extremes_Y, Extremes_Z

extremes_combined_facade = []

extreme = find_extremes(facade)
for item in extreme:
    if item[0] == item[1]:
        pass
    else:
        extremes_combined_facade.append(item)

temp_Con_W = []
temp_Con_H = []
for i, extremes in enumerate(extremes_combined_facade):
    print(extremes)
    for item in extremes:
        if i%2 == 0:
            temp_Con_W.append(item)
        elif i%2 == 1:
            temp_Con_H.append(item)
        else:
            print("more then 2 items in extremes")
Con_W = (min(temp_Con_W),max(temp_Con_W))
Con_H = (min(temp_Con_H),max(temp_Con_H))
```

7.1.2. Panel contour generation script

the scripts used in the façade divider, panel boundary and clipping and capping are shown on the next pages.



The script used to divide the façade in rectangular panels is shown below.

```
segs_fac =rg.PolylineCurve.DuplicateSegments(facade)
boundary_points = []
for seg in segs_fac:
    pb = seg.PointAtNormalizedLength(0)
    pe = seg.PointAtNormalizedLength(1)
    if pb not in boundary_points:
        boundary_points.append(pb)
    if pe not in boundary_points:
        boundary_points.append(pe)

boun_x = []
boun_y = []
boun_z = []
for point in boundary_points:
    boun_x.append(int(point[0]))
    boun_y.append(int(point[1]))
    boun_z.append(int(point[2]))
resultx = all(elem == boun_x[0] for elem in boun_x)
resulty = all(elem == boun_y[0] for elem in boun_y)

Con_H = [min(boun_z),max(boun_z)]
if resultx == False:
    Con_W = [min(boun_x),max(boun_x)]
    pan_depth = boun_y[0]
else:
    Con_W = [min(boun_y),max(boun_y)]
    pan_depth = boun_x[0]

##H Cords
Con = Con_H
Open = Open_H
Pan = Pan_H
Cords = [Con[0]]
while Cords[-1] < Con[1] - tol_edge: # creates facade division for X coordinates
    temp_cord = [] #new list for the temp coordinates
    for x in Open: #checks all window opening sizes
        if Pan + Cords[-1] >= Con[0] + tol_edge and Pan + Cords[-1] < Con[1] - tol_edge: #panel fits in
contours
            if Pan + Cords[-1] >= x[0] and Pan + Cords[-1] <= x[1]: #panel crosses window
                if x[0] == Cords[-1]: #if panel is smaller then window size this if statement makes sure the
while loop does not keep looping and just makes panels that are of smaller size so they can be
placed around the window
                    temp_cord.append(Pan+Cords[-1])
                elif Cords[-1] >= x[0] and Cords[-1] < x[1]:
                    temp_cord.append(Pan+Cords[-1])
                else:
                    temp_cord.append(x[0]-open_off)
            else: #panel does not cross widow
                temp_cord.append(Pan+Cords[-1])
        else: #panel to big for contours
            temp_cord.append(Con[1] - tol_edge)
```

```
if min(temp_cord) == Cords[-1]: #checks if while loop does not rune indefinitly
        print("panel size is to small to incorporate windows")
        break
    Cords.append(min(temp_cord)) #gets lowest value and adds it to the list for x divisions
    if Cords[-1] != Con[1] - tol_edge:
        Cords.append(Cords[-1] + 200)
H_Cords = Cords

##W Cords
Con = Con_W
Open = Open_W
Pan = Pan_W
Cords = [Con[0] + tol_edge]
while Cords[-1] < Con[1] - tol_edge: # creates facade division for X cordinates
    temp_cord = [] #new list for the temp coordinates
    for x in Open: #checks all window opening sizes
        if Pan + Cords[-1] >= Con[0] + tol_edge and Pan + Cords[-1] < Con[1] - tol_edge: #panel fits in
contours
            if Pan + Cords[-1] >= x[0] and Pan + Cords[-1] <= x[1]: #panel crosses window
                if x[0] == Cords[-1]: #if panel is smaller then window size this if statement makes sure the
while loop does not keep looping and just makes panels that are of smaller size so they can be
placed around the window
                    temp_cord.append(Pan+Cords[-1])
                elif Cords[-1] >= x[0] and Cords[-1] < x[1]:
                    temp_cord.append(Pan+Cords[-1])
                else:
                    temp_cord.append(x[0]-open_off)
            else: #panel does not cross widow
                temp_cord.append(Pan+Cords[-1])
        else: #panel to big for contours
            temp_cord.append(Con[1] - tol_edge)
    if min(temp_cord) == Cords[-1]: #checks if while loop does not rune indefinitely
        print("panel size is to small to incorporate windows")
        break
    Cords.append(min(temp_cord)) #gets lowest value and adds it to the list for x divisions
    if Cords[-1] != Con[1] - tol_edge:
        Cords.append(Cords[-1] + tol_betw)
W_Cords = Cords

pan_contour_rec = []
temp_pan_contour = []

Rec_W = len(W_Cords) - 1
Rec_H = len(H_Cords) - 1

for i,Rec_W in enumerate(range(Rec_W)):
    if i%2 == 0:
        for j,Rec_W in enumerate(range(Rec_H)):
            if j%2 == 0:
                myPath = GH_Path(i,j)
                b1 = rg.Point3d(W_Cords[i], pan_depth, H_Cords[j])
```

```
            b2 = rg.Point3d(W_Cords[i + 1], pan_depth, H_Cords[j])
            b3 = rg.Point3d(W_Cords[i + 1], pan_depth, H_Cords[j + 1])
            b4 = rg.Point3d(W_Cords[i], pan_depth, H_Cords[j + 1])
            b_temp = [b1, b2, b3, b4, b1]
            bt = rg.PolylineCurve(b_temp)
            pan_contour_rec.append(bt)

##Plane constructor
p1 = rg.PolylineCurve.Point(pan_contour_rec[0],0)
p2 = rg.PolylineCurve.Point(pan_contour_rec[0],1)
p3 = rg.PolylineCurve.Point(pan_contour_rec[0],2)
plane = rg.Plane(p1,p2,p3)
```

The script below is used to generate in curve that is the external curve of all panels. Used to clip the panels made in the previous script.

```
new_points = []
Z_check = []
for point in panel_bound_points:
   Z_check.append(math.floor(point[2]))
for i,check in enumerate(Z_check):
   if check ==(min(Z_check)):
      new_points.append(rg.Point3d(panel_bound_points[i][0],panel_bound_points[i][1],Con_H[0]))
   else:
      new_points.append(panel_bound_points[i])

pan_bound = rg.PolylineCurve(new_points)
```

The script below uses the panel boundary to clip the rectangular panels. Both are generated with the previous scripts.

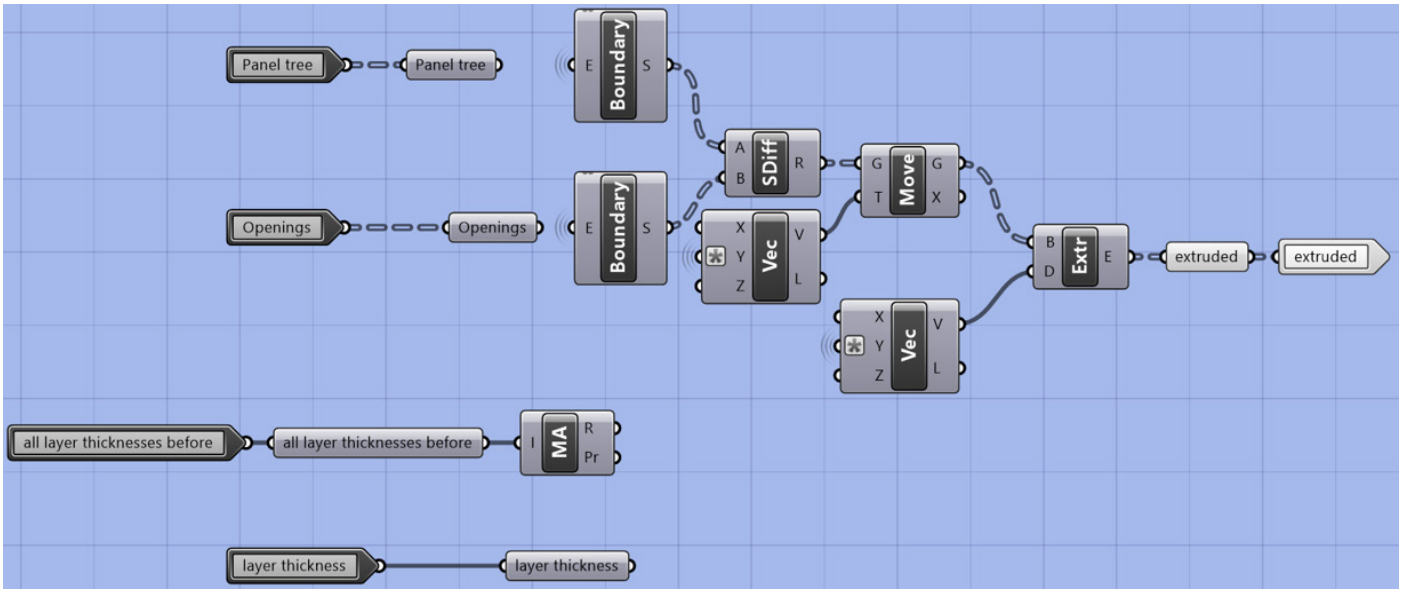
```
clipped_panels = []
un_clipped_panels = []
caps = []
for rec in pan_contour_rec:
    params = []
    events = rg.Intersect.Intersection.CurveCurve(rec,facade,0,0)
    for event in events:
        params.append(event.ParameterA)
    if params == []:
        un_clipped_panels.append(rec)
    else:
        params = []
        clipped_pan = []
        events = rg.Intersect.Intersection.CurveCurve(rec,pan_bound,0,0)
        for event in events:
            params.append(event.ParameterA)
        temp = rec.Split(params)
        temp_lines = []
        for item in temp:
            point = item.PointAtNormalizedLength(0.5)
            contain = rg.Curve.Contains(pan_bound,point,plane)
            if contain != rg.PointContainment.Outside:
                clipped_pan.append(item)
                clipped_panels.append(item)
        params_caps = []
        for line in clipped_pan:
            pb = line.PointAtNormalizedLength(0)
            pe = line.PointAtNormalizedLength(1)
            params_caps.append(rg.PolylineCurve.ClosestPoint(pan_bound,pb)[1])
            params_caps.append(rg.PolylineCurve.ClosestPoint(pan_bound,pe)[1])
        temp_cap = pan_bound.Split(params_caps)
        for item in temp_cap:
            point = item.PointAtNormalizedLength(0.5)
            contain = rg.Curve.Contains(rec,point,plane,0)
            if contain == rg.PointContainment.Inside:
                clipped_panels.append(item)
```

7.1.3. Panel structuring script

```
structured_panels = DataTree[rg.Curve]()
structured_openings = DataTree[rg.Curve]()

for i,pan in enumerate(panels):
    myPath = GH_Path(i)
    structured_panels.Add(pan,myPath)
    open_in_pan = False
    for j,check in enumerate(is_inside.Branch(0,i)):
        if check == 2:
            structured_openings.Add(openings[j],myPath)
            open_in_pan = True
    if open_in_pan == False:
        structured_openings.EnsurePath(myPath)
```

7.1.4. Solid layer generation script

For this only grasshopper components are used they are all shown below.



7.1.5. Boundary member generation script

```
for i in range(len(num_panels)):
  myPath = GH_Path(i)
  for j,pan in enumerate(panels.Branch(i)):
    segs = rg.PolylineCurve.DuplicateSegments(pan)
    horizontals = []
    non_horizontals = []
    for inner in interior_plate_line.Branch(i):
      point = rg.PolylineCurve.PointAtNormalizedLength(inner,0.5)
      contain = rg.Curve.Contains(pan,point,plane)
      if contain == rg.PointContainment.Inside:
        inner_line = inner
    inner_segs = rg.PolylineCurve.DuplicateSegments(inner_line)
    inner_hor = []
    inner_non_hor = []
    for innerseg in inner_segs:
      pb = rg.LineCurve.PointAtNormalizedLength(innerseg,0)
      pe = rg.LineCurve.PointAtNormalizedLength(innerseg,1)
      if abs(pb[2]-pe[2]) < 10:
        inner_hor.append(innerseg)
      else:
        inner_non_hor.append(innerseg)
    for seg in segs:
      pb = rg.LineCurve.PointAtNormalizedLength(seg,0)
      pe = rg.LineCurve.PointAtNormalizedLength(seg,1)
      if abs(pb[2]-pe[2]) < 10:
        horizontals.append(seg)
      else:
        non_horizontals.append(seg)
        oth_ex_lines.Add(seg,myPath)
    inner_lines = []
    for line in horizontals:
      out_line = line
      inner_line_extended = []
      temp = rg.LineCurve.Offset(line,plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)[0]
      point = rg.LineCurve.PointAtNormalizedLength(temp,0.5)
      contain = rg.Curve.Contains(pan,point,plane,0)
      if contain == rg.PointContainment.Inside:
        inner_lines = temp
      else:
        inner_lines = (rg.LineCurve.Offset(line,plane,-
plate_height,0,rg.CurveOffsetCornerStyle.Sharp)[0])
      temp =
rg.LineCurve.Extend(inner_lines,rg.CurveEnd.Both,plate_height*2,rg.CurveExtensionStyle.Line)
      params = []
      events = rg.Intersect.Intersection.CurveCurve(temp,pan,0,0)
      for event in events:
        params.append(event.ParameterA)
      temp_line = temp.Split(params)
      for obj in temp_line:
        point = obj.PointAtNormalizedLength(0.5)
        contain = rg.Curve.Contains(pan,point,plane,0)
```

```
            if contain == rg.PointContainment.Inside:
                inner_line_extended.append(obj)
        inner_line_cut = []
        for line in inner_line_extended:
            params = []
            temp_lines = []
            for obj in inner_non_hor:
                events = rg.Intersect.Intersection.CurveCurve(line,obj,0,0)
                for event in events:
                    params.append(event.ParameterA)
            temp = line.Split(params)
            for obj in temp:
                point = obj.PointAtNormalizedLength(0.5)
                contain_pan = rg.Curve.Contains(pan,point,plane,0)
                contain_inn = rg.Curve.Contains(inner_line,point,plane,0)
                if contain_pan == rg.PointContainment.Inside and contain_inn !=
rg.PointContainment.Inside:
                    temp_lines.append(obj)
            inln =
rg.LineCurve(rg.LineCurve.PointAtNormalizedLength(temp_lines[0],0),rg.LineCurve.PointAtNormali
zedLength(temp_lines[-1],1))
        #outer line
        temp =
rg.LineCurve.Extend(out_line,rg.CurveEnd.Both,plate_height*2,rg.CurveExtensionStyle.Line)
        params = []
        for obj in non_horizontals:
            events = rg.Intersect.Intersection.CurveCurve(temp,obj,0,0)
            for event in events:
                params.append(event.ParameterA)
        temp_line = temp.Split(params)
        outer_line_extended = []
        for obj in temp_line:
            point = obj.PointAtNormalizedLength(0.5)
            contain = rg.Curve.Contains(pan,point,plane,0)
            if contain != rg.PointContainment.Outside:
                outer_line_extended.append(obj)
        if len(outer_line_extended) == 1:
            outln = outer_line_extended[0]
        else:
            temp_ln =
rg.LineCurve(rg.LineCurve.PointAtNormalizedLength(outer_line_extended[0],0),rg.LineCurve.Point
AtNormalizedLength(outer_line_extended[-1],1))
            params = []
            for obj in inner_non_hor:
                events = rg.Intersect.Intersection.CurveCurve(temp_ln,obj,0,0)
                for event in events:
                    params.append(event.ParameterA)
            temp_ln_sp = temp_ln.Split(params)
            for obj in temp_ln_sp:
                point = obj.PointAtNormalizedLength(0.5)
                contain_pan = rg.Curve.Contains(pan,point,plane,0)
```

```
      contain_inn = rg.Curve.Contains(inner_line,point,plane,0)
            if contain_pan != rg.PointContainment.Outside and contain_inn ==
rg.PointContainment.Outside:
               outln = obj
         ## opening cutter
         params_in = []
         params_out = []
         for open in openings.Branch(i):
            open_segs = rg.PolylineCurve.DuplicateSegments(open)
            non_hor = []
            for seg in open_segs:
               pb = rg.LineCurve.PointAtNormalizedLength(seg,0)
               pe = rg.LineCurve.PointAtNormalizedLength(seg,1)
               if abs(pb[2]-pe[2]) > 10:
                  non_hor.append(seg)
            for obj in non_hor:
               events = rg.Intersect.Intersection.CurveCurve(inln,obj,0,0)
               for event in events:
                  params_in.append(event.ParameterA)
               events = rg.Intersect.Intersection.CurveCurve(outln,obj,0,0)
               for event in events:
                  params_out.append(event.ParameterA)
         if params_in != []:
            in_lines = []
            out_lines = []
            temp_in = inln.Split(params_in)
            temp_out = outln.Split(params_out)
            for obj in temp_in:
               point = obj.PointAtNormalizedLength(0.5)
               contain = []
               for open in openings.Branch(i):
                  contain.append(rg.Curve.Contains(open,point,plane,0))
               if rg.PointContainment.Inside not in contain and rg.PointContainment.Coincident not in
contain:
                  in_lines.append(obj)
            for obj in temp_out:
               point = obj.PointAtNormalizedLength(0.5)
               contain = []
               for open in openings.Branch(i):
                  contain.append(rg.Curve.Contains(open,point,plane,0))
               if rg.PointContainment.Inside not in contain and rg.PointContainment.Coincident not in
contain:
                  out_lines.append(obj)
            for q,line in enumerate(in_lines):
               inb = line.PointAtNormalizedLength(0)
               ine = line.PointAtNormalizedLength(1)
               outb = out_lines[q].PointAtNormalizedLength(0)
               oute = out_lines[q].PointAtNormalizedLength(1)
               tp = [inb,ine,oute,outb,inb]
               plates.Add(rg.PolylineCurve(tp),myPath)
         else:
```

```
            inb = inln.PointAtNormalizedLength(0)
            ine = inln.PointAtNormalizedLength(1)
            outb = outln.PointAtNormalizedLength(0)
            oute = outln.PointAtNormalizedLength(1)
            tp = [inb,ine,oute,outb,inb]
            plates.Add(rg.PolylineCurve(tp),myPath)
        ##non horizontal boundary plates
        for line in non_horizontals:
            outer_line = line
            temp = rg.LineCurve.Offset(line,plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)[0]
            point = rg.LineCurve.PointAtNormalizedLength(temp,0.5)
            contain = rg.Curve.Contains(pan,point,plane,0)
            if contain == rg.PointContainment.Inside:
                inner_line = temp
            else:
                inner_line = (rg.LineCurve.Offset(line,plane,-
plate_height,0,rg.CurveOffsetCornerStyle.Sharp)[0])
            plate_hr = []
            for plate in plates.Branch(i):
                segs = rg.PolylineCurve.DuplicateSegments(plate)
                for seg in segs:
                    pb = rg.LineCurve.PointAtNormalizedLength(seg,0)
                    pe = rg.LineCurve.PointAtNormalizedLength(seg,1)
                    if abs(pb[2]-pe[2]) < 10:
                        plate_hr.append(seg)
            params_in = []
            params_out = []
            for seg in plate_hr:
                temp = seg.Extend(rg.CurveEnd.Both,plate_height,rg.CurveExtensionStyle.Line)
                events = rg.Intersect.Intersection.CurveCurve(outer_line,temp,0,0)
                for event in events:
                    params_out.append(event.ParameterA)
                events = rg.Intersect.Intersection.CurveCurve(inner_line,temp,0,0)
                for event in events:
                    params_in.append(event.ParameterA)
            ## inner line
            if params_in != []:
                temp_lines = inner_line.Split(params_in)
                for obj in temp_lines:
                    point = obj.PointAtNormalizedLength(0.5)
                    contain = []
                    for item in plates.Branch(i):
                        contain.append(rg.Curve.Contains(item,point,plane,0))
                    if rg.PointContainment.Inside not in contain and rg.PointContainment.Coincident not in
contain:
                        inln = obj
            else:
                inln = inner_line
            ##outer line
            if params_out != []:
                temp_lines = outer_line.Split(params_out)
```

```
            for obj in temp_lines:
                point = obj.PointAtNormalizedLength(0.5)
                contain = []
                for item in plates.Branch(i):
                    contain.append(rg.Curve.Contains(item,point,plane,0))
                if rg.PointContainment.Inside not in contain and rg.PointContainment.Coincident not in
contain:
                    outln = obj
        else:
            outln = outer_line
        inb = inln.PointAtNormalizedLength(0)
        ine = inln.PointAtNormalizedLength(1)
        outb = outln.PointAtNormalizedLength(0)
        oute = outln.PointAtNormalizedLength(1)
        tp = [inb,ine,oute,outb,inb]
        plates.Add(rg.PolylineCurve(tp),myPath)
```

7.1.6. Opening plate generation script

```
plate_center_line = DataTree[rg.LineCurve]()
a = []
for i in range(panel_tree.BranchCount):
    myPath = GH_Path(i)
    for pan in panel_tree.Branch(i):
        for open in openings.Branch(i):
            segs = rg.PolylineCurve.DuplicateSegments(open)
            for seg in segs:
                pb = seg.PointAtNormalizedLength(0)
                pe = seg.PointAtNormalizedLength(1)
                if abs(pe[2]-pb[2]) < 10:
                    temp =
rg.LineCurve.Offset(seg,plane,plate_height/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
                    point = temp.PointAtNormalizedLength(0.5)
                    contain = rg.Curve.Contains(open,point,plane,0)
                    if contain == rg.PointContainment.Outside:
                        temp = (temp)
                    else:
                        temp = (rg.LineCurve.Offset(seg,plane,-
plate_height/2,0,rg.CurveOffsetCornerStyle.Sharp)[0])
                    point2 = temp.PointAtNormalizedLength(0.5)
                    contain = rg.Curve.Contains(pan,point2,plane,0)
                    if contain == rg.PointContainment.Inside:
                        plate_center_line.Add(temp,myPath)
```

7.1.7. Stud generation script

```
stud_lines = DataTree[rg.PolylineCurve]()

for i in range(Panels.BranchCount):
  myPath = GH_Path(i)
  for panel in Panels.Branch(i):
#this part sets up the script in generating the domains for the panels and the windows
    segs = rg.PolylineCurve.DuplicateSegments(panel)
    boundary_x = []
    boundary_y = []
    boundary_z = []
    for seg in segs:
      boundary_x.append(seg.PointAtNormalizedLength(0.5)[0])
      boundary_y.append(seg.PointAtNormalizedLength(0.5)[1])
      boundary_z.append(seg.PointAtNormalizedLength(0.5)[2])
    resultx = all(elem == boundary_x[0] for elem in boundary_x)
    resulty = all(elem == boundary_y[0] for elem in boundary_y)
    pan_H = [int(round(min(boundary_z))),int(round(max(boundary_z)))]
    if resultx == False:
      pan_W = [int(round(min(boundary_x))),int(round(max(boundary_x)))]
      planex = True
      pan_d = boundary_y[0]
    else:
      pan_W = [int(round(min(boundary_y))),int(round(max(boundary_y)))]
      planex = False
      pan_d = boundary_x[0]
    pan_cords = [pan_W[0]+frame_width/2]
    cords_open = []
    for open in Openings.Branch(i):
      segs_open = rg.PolylineCurve.DuplicateSegments(open)
      for seg in segs_open:
        pb = seg.PointAtNormalizedLength(0)
        pe = seg.PointAtNormalizedLength(1)
        if abs(pb[2]-pe[2]) > 10:
          temp_seg =
rg.LineCurve.Offset(seg,plane,frame_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
          point = temp_seg.PointAtNormalizedLength(0.5)
          contain = rg.Curve.Contains(open,point,plane,0)
          if contain == rg.PointContainment.Outside:
            if planex == True:
              cords_open.append(int(round(point[0])))
            else:
              cords_open.append(int(round(point[1])))
          else:
            point = (rg.LineCurve.Offset(seg,plane,-
frame_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]).PointAtNormalizedLength(0.5)
            if planex == True:
              cords_open.append(int(round(point[0])))
            else:
              cords_open.append(int(round(point[1])))
    cords_open.sort()
    while pan_cords[-1] < pan_W[1]:
```

```
        temp_cord = pan_cords[-1]+frame_int
        for cor in cords_open:
            if pan_cords[-1] < cor and temp_cord > cor:
                temp_cord = cor
            elif temp_cord > cor-frame_width-min_distance and temp_cord < cor:
                pan_cords.append(cor-frame_width-min_distance)
                temp_cord = cor
            elif temp_cord > cor and temp_cord < cor+frame_width+min_distance:
                temp_cord = cor
        pan_cords.append(temp_cord)
    pan_cords.pop(0)
    if pan_cords[-1] > pan_W[1]:
        pan_cords.pop(-1)

##line generation
    for cord in pan_cords:
        pb = rg.Point3d(cord,pan_d,pan_H[0])
        pe = rg.Point3d(cord,pan_d,pan_H[1])
        temp_cent_ln = rg.LineCurve(pb,pe)
        ##take out lines outside panels
        params = []
        events = rg.Intersect.Intersection.CurveCurve(temp_cent_ln,panel,0,0)
        for event in events:
            params.append(event.ParameterA)
        tmp_cl_pan = temp_cent_ln.Split(params)
        line_clipped_pan = []
        for line in tmp_cl_pan:
            point = line.PointAtNormalizedLength(0.5)
            con_pan = rg.Curve.Contains(panel,point,plane,0)
            if con_pan == rg.PointContainment.Inside:
                line_clipped_pan.append(line)
        line_clipped_plate = []
        plate_hor = []
        for plate in bound_plate.Branch(i):
            segs = rg.PolylineCurve.DuplicateSegments(plate)
            for seg in segs:
                pb = rg.LineCurve.PointAtNormalizedLength(seg,0)
                pe = rg.LineCurve.PointAtNormalizedLength(seg,1)
                if abs(pb[2]-pe[2]) < 10:
                    plate_hor.append(seg.Extend(rg.CurveEnd.Both,70,rg.CurveExtensionStyle.Line))
        for line in line_clipped_pan:
            params = []
            for obj in plate_hor:
                events = rg.Intersect.Intersection.CurveCurve(line,obj,0,0)
                for event in events:
                    params.append(event.ParameterA)
            temp = line.Split(params)
            for obj in temp:
                point = obj.PointAtNormalizedLength(0.5)
                contain = []
                for plate in bound_plate.Branch(i):
```

```
                contain.append(rg.Curve.Contains(plate,point,plane,0))
            if rg.PointContainment.Inside not in contain and rg.PointContainment.Coincident not in
contain:
                line_clipped_plate.append(obj)
        line_clipped_open_pl = []
        center_lines = []
        for line in line_clipped_plate:
            params = []
            for open in Openings.Branch(i):
                events = rg.Intersect.Intersection.CurveCurve(line,open,0,0)
                for event in events:
                    params.append(event.ParameterA)
            if params == []:
                center_lines.append(line)
            else:
                openings_clipped = []
                temp = line.Split(params)
                for obj in temp:
                    point = obj.PointAtNormalizedLength(0.5)
                    contain = []
                    for open in Openings.Branch(i):
                        contain.append(rg.Curve.Contains(open,point,plane,0))
                    if rg.PointContainment.Inside not in contain:
                        openings_clipped.append(obj)
                for obj in openings_clipped:
                    params = []
                    for plate in open_plates.Branch(i):
                        events = rg.Intersect.Intersection.CurveCurve(obj,plate,0,0)
                        for event in events:
                            params.append(event.ParameterA)
                    temp = obj.Split(params)
                    for ins in temp:
                        point = ins.PointAtNormalizedLength(0.5)
                        contain = []
                        for plate in open_plates.Branch(i):
                            contain.append(rg.Curve.Contains(plate,point,plane,0))
                        if rg.PointContainment.Inside not in contain:
                            center_lines.append(ins)
        for line in center_lines:
            temp = line
            l1 = line.Offset(plane,frame_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
            l2 = temp.Offset(plane,-frame_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
            params1 = []
            params2 = []
            p1b = l1.PointAtNormalizedLength(0)
            p1e = l1.PointAtNormalizedLength(1)
            p2b = l2.PointAtNormalizedLength(0)
            p2e = l2.PointAtNormalizedLength(1)
            tp = [p1b,p1e,p2e,p2b,p1b]
            stud_lines.Add(rg.PolylineCurve(tp),myPath)
```

7.1.8. Panel connection generation script

```
framework_members = DataTree[rg.PolylineCurve]()
insulation = DataTree[rg.LineCurve]()
a = []
for i in range(len(num_panels)):
    myPath = GH_Path(i)
    for j,pan in enumerate(panels.Branch(i)):
        sharp = rg.CurveOffsetCornerStyle.Sharp
        myPath = GH_Path(i)
        plate_ex_curves = rg.PolylineCurve.DuplicateSegments(pan)
        pan_hor_lines = []
        pan_bound_points = []
#get horizontal lines from ever panel
        for line in plate_ex_curves:
            pb = line.PointAtNormalizedLength(0)
            pe = line.PointAtNormalizedLength(1)

            if pb not in pan_bound_points:
                pan_bound_points.append(pb)
            if pe not in pan_bound_points:
                pan_bound_points.append(pe)
            hor_int_nclip = []
            temp_lines = []
            if abs(pb[2] - pe[2]) < 10: #checks if lines are (nearly) horizontal
                pan_hor_lines.append(line)
        z_points = []
        for point in pan_bound_points:
            depth = point[0]
            z_points.append(point[2])

        pan_dom = [int(round(min(z_points))),int(round(max(z_points)))]
#check top and bottom line
        for line in pan_hor_lines:
            point = int(round(rg.LineCurve.PointAtNormalizedLength(line,0.5)[2]))
            if point == pan_dom[0]:
                bot_line = line
            elif point == pan_dom[1]:
                top_line = line
#creating top connection
    #top plate
        if (len(pan_hor_lines)) != 1:
            framework_members_curves = []
            temp_top_plate = []
            motion1 = rg.Transform.Translation(0,0,157)
            motion2 = rg.Transform.Translation(0,0,195)
            temp1 = rg.LineCurve(top_line)
            temp2 = rg.LineCurve(top_line)
            temp1.Transform(motion1)
            temp2.Transform(motion2)
            temp_top_plate.append(temp1)
            temp_top_plate.append(temp2)
            insulation.Add(top_line,myPath)
```

```
    top_plate = []
    for line in temp_top_plate:
        params = []
        events = rg.Intersect.Intersection.CurveCurve(line,panel_bound,0,0)
        for event in events:
            params.append(event.ParameterA)
        temp = line.Split(params)
        for obj in temp:
            point = obj.PointAtNormalizedLength(0.5)
            cont = rg.Curve.Contains(panel_bound,point,plane)
            if cont == rg.PointContainment.Inside:
                top_plate.append(obj)
    if top_plate != []:
        l1pb = top_plate[0].PointAtNormalizedLength(0)
        l1pe = top_plate[0].PointAtNormalizedLength(1)
        l2pb = top_plate[1].PointAtNormalizedLength(0)
        l2pe = top_plate[1].PointAtNormalizedLength(1)
        temp_points = [l1pb,l1pe,l2pe,l2pb,l1pb]
        framework_members_curves.append((rg.PolylineCurve(temp_points)))
        framework_members.Add((rg.PolylineCurve(temp_points)),myPath)
#beginning and ending stud
    zc = []
    begin_stud = []
    end_stud = []
    if top_plate != []:
        for line in top_plate:
            point = line.PointAtNormalizedLength(0.5)
            zc.append(point[2])
        min_ind = zc.index(min(zc))
        max_ind = zc.index(max(zc))
        insulation.Add(top_plate[max_ind],myPath)
        pbbe = top_line.PointAtNormalizedLength(0)
        pebe = top_line.PointAtNormalizedLength(1)
        pbte = top_plate[min_ind].PointAtNormalizedLength(0)
        pete = top_plate[min_ind].PointAtNormalizedLength(1)
        lbe = rg.LineCurve(pbbe,pbte)
        lee = rg.LineCurve(pebe,pete)
        pbbi = top_line.PointAtLength(plate_height)
        pebi = top_line.PointAtLength(top_line.GetLength()-plate_height)
        pbti = top_plate[min_ind].PointAtLength(plate_height)
        peti = top_plate[min_ind].PointAtLength(top_plate[min_ind].GetLength()-plate_height)
        lbi = rg.LineCurve(pbbi,pbti)
        lei = rg.LineCurve(pebi,peti)
        begin_stud.append(lbe)
        begin_stud.append(lbi)
        end_stud.append(lee)
        end_stud.append(lei)
        temp_points1 = [pbbe,pbte,pbti,pbbi,pbbe]
        temp_points2 = [pebe,pete,peti,pebi,pebe]
        framework_members.Add((rg.PolylineCurve(temp_points1)),myPath)
        framework_members.Add((rg.PolylineCurve(temp_points2)),myPath)
```

```
    framework_members_curves.append((rg.PolylineCurve(temp_points1)))
    framework_members_curves.append((rg.PolylineCurve(temp_points2)))
#other studs
    temp_x = []
    temp_x.append(( top_plate[min_ind].PointAtNormalizedLength(0))[0])
    temp_x.append(( top_plate[min_ind].PointAtNormalizedLength(1))[0])
    cons = [min(temp_x),max(temp_x)]
    x_cords = [cons[0]]
    while x_cords[-1] < cons[1]-frame_int:
        x_cords.append(float(x_cords[-1]+frame_int))
    x_cords.pop(0)
    if x_cords[-1] > cons[1]-min_dist-plate_height:
        x_cords.pop(-1)
        x_cords.append(float(cons[1]-min_dist-plate_height))
    for cord in x_cords:
        temp_p = top_line.PointAtNormalizedLength(0)[2]
        temp_d = top_line.PointAtNormalizedLength(0)[1]
        pb = (rg.Point3d(cord,temp_d,temp_p))
        pt = pb + rg.Point3d(0,0,157)
        tline = (rg.LineCurve(pb,pt))
        stud_lines = []
        tline1_temp = rg.LineCurve.Offset(tline,plane,stud_width/2,0,sharp)
        tline2_temp = rg.LineCurve.Offset(tline,plane,-stud_width/2,0,sharp)
        for item in tline1_temp:
            stud_lines.append(item)
        for item in tline2_temp:
            stud_lines.append(item)
        p1b = stud_lines[0].PointAtNormalizedLength(0)
        p1e = stud_lines[0].PointAtNormalizedLength(1)
        p2b = stud_lines[1].PointAtNormalizedLength(0)
        p2e = stud_lines[1].PointAtNormalizedLength(1)
        temp_points_studs = [p1b,p1e,p2e,p2b,p1b]
        temp_line_stud = rg.PolylineCurve(temp_points_studs)
        framework_members.Add(temp_line_stud,myPath)
else:
    framework_members.EnsurePath(myPath)
    insulation.EnsurePath(myPath)
```

7.1.9. RC-calculation script

This script calculates the average RC-value per panel using the areas per material and their thickness and lambda.

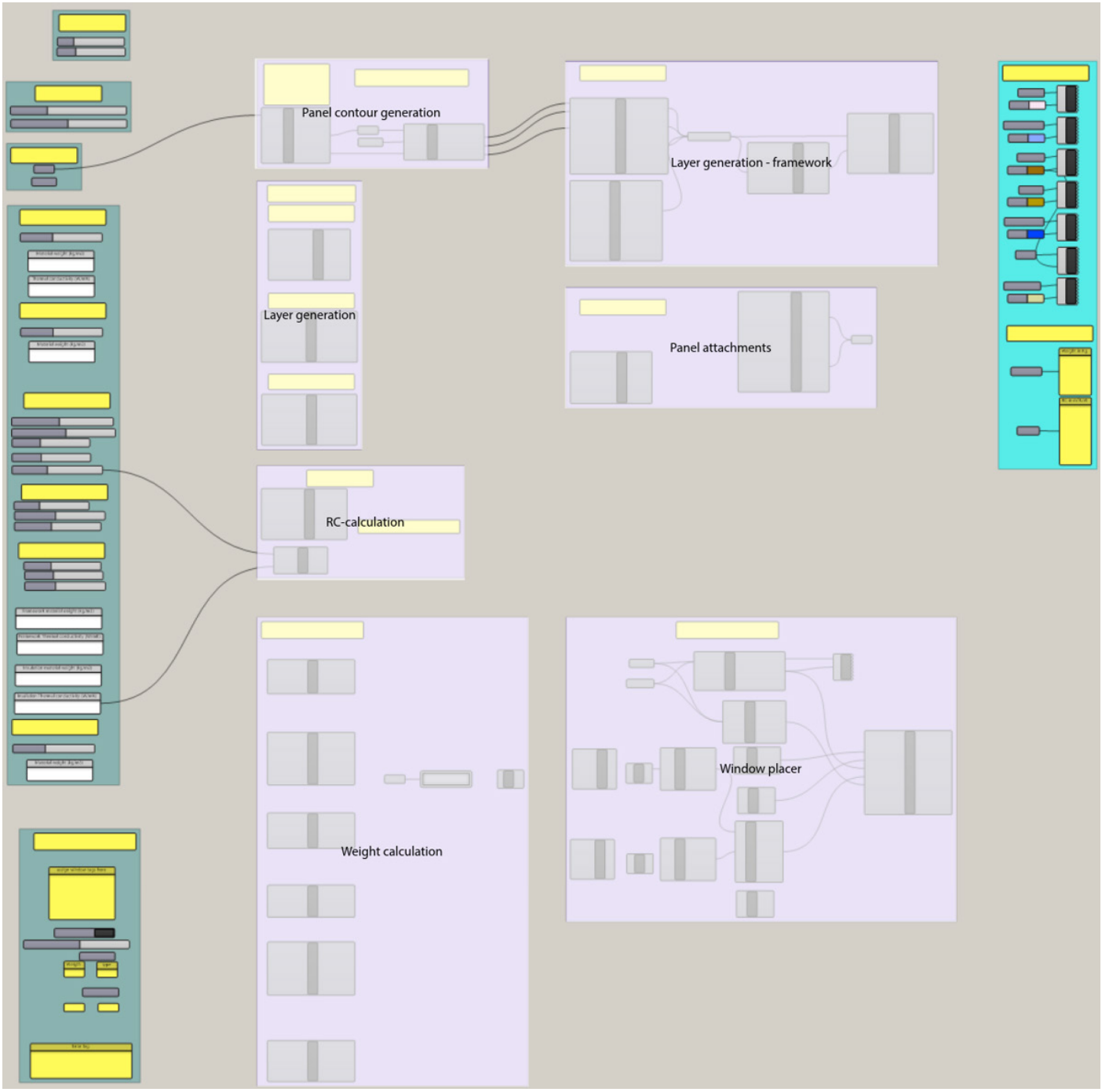
```
RC_panels = DataTree[float]()
temp_RC = []

for i in range(len(num_panel)):
    myPath = GH_Path(i)
    for item in framework_area.Branch(i):
        temp_frame_area = item
    for item in insulation_area.Branch(i):
        temp_ins_area = item
    total_area = (temp_frame_area + temp_ins_area)
    per_ins = (temp_ins_area/total_area)
    per_frame = 1 - per_ins
    RC_ins = ((layer_thickness/1000)/insulation_lambda)
    RC_frame = ((layer_thickness/1000)/framework_lambda)
    RC = round(RC_ins*per_ins + RC_frame*per_frame,1)
    RC_panels.Add(RC,myPath)
    temp_RC.append(RC)
lowest_RC = min(temp_RC)

fb_lowest_RC = "Lowest actual RC-Value is: {}m²K/W".format(lowest_RC)
```

7.1.10. Window placement script

```
if Run == True:
    window_frames = DataTree[rg.Brep]()
    window_weight = DataTree[float]()
    for p,gt in enumerate(goal_types):

        for j,st in enumerate(source_types):
            if gt == st:
                for i,pan in enumerate(num_panels):
                    tree_branch = GH_Path(i)
                    window_in_pan = False
                    contain = rg.Curve.Contains(pan,goal_point[p],plane,0)
                    if contain != rg.PointContainment.Outside:
                        window_in_pan = True
                    else:
                        window_frames.EnsurePath(GH_Path(i))
                    if window_in_pan == True:
                        motion = rg.Transform.Translation(rg.Vector3d(goal_point[p]-source_point[j]))
                        window_weight.Add(weight[j],tree_branch)
                        for obj in windows.Branch(0,j):
                            temp = obj.Duplicate()
                            temp.Transform(motion)
                            offset = rg.Transform.Translation(0,-window_offset,0)
                            temp.Transform(offset)
                            window_frames.Add(temp,tree_branch)
```

7.2. Roof tool script

In this appendix all individual components of the roof tool can be found.

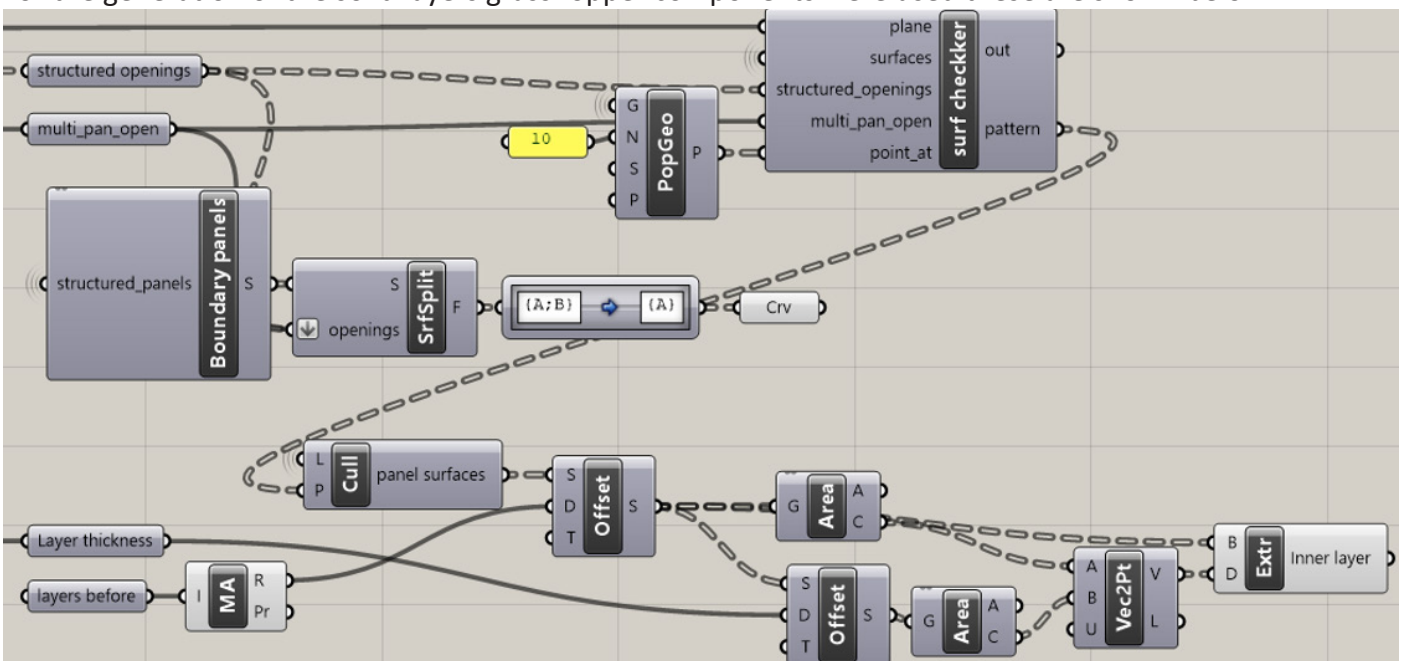


7.2.1. Panel contour generation script

```
temp_roof = rg.PolylineCurve.DuplicateSegments(roof)
hor_lines = []
for line in temp_roof:
    p1 = (rg.LineCurve.PointAtNormalizedLength(line,0))
    p2 = (rg.LineCurve.PointAtNormalizedLength(line,1))
    if abs(p1[2]-p2[2]) < 10:
        if p1[0] < p2[0]:
            hor_lines.append(rg.LineCurve(p1,p2))
        else:
            hor_lines.append(rg.LineCurve(p2,p1))
roof_dom =
[math.ceil((rg.LineCurve.PointAtNormalizedLength(hor_lines[0],0))[0]),math.ceil((rg.LineCurve.Poin
tAtNormalizedLength(hor_lines[0],1))[0])]
roof_dom.sort()
pan_cords = [roof_dom[0]+edge_tol]
while pan_cords[-1] < (roof_dom[1]-edge_tol-pan_w):
    temp_point = pan_cords[-1]+pan_w
    temp_point2 = temp_point+betw_tol
    pan_cords.append(temp_point)
    pan_cords.append(temp_point2)
pan_cords.append(roof_dom[1]-edge_tol)

panel_contours = []
isplane = False
for i,cord in enumerate(pan_cords):
    if i%2:
        pass
    else:
        p1 = rg.LineCurve.PointAtLength(hor_lines[0],cord-roof_dom[0])
        p2 = rg.LineCurve.PointAtLength(hor_lines[1],cord-roof_dom[0])
        p3 = rg.LineCurve.PointAtLength(hor_lines[1],pan_cords[i+1]-roof_dom[0])
        p4 = rg.LineCurve.PointAtLength(hor_lines[0],pan_cords[i+1]-roof_dom[0])
        plp = [p1,p2,p3,p4,p1]
        panel_contours.append(rg.PolylineCurve(plp))
        if isplane == False:
            plane = rg.Plane(p1,p2,p4)
            isplane = True
```

7.2.2. Solid layer generation script

For the generation of the solid layers grasshopper components were used these are shown below



To check if a split surface is inside a opening the script below is used

```
for i in range(surfaces.BranchCount):
    myPath = GH_Path(i)
    for j,sur in enumerate(surfaces.Branch(i)):
        contain = []
        for point in point_at.Branch(i,j):
            for open in structured_openings.Branch(i):
                contain.append(rg.Curve.Contains(open,point,plane,0))
            for open in multi_pan_open:
                contain.append(rg.Curve.Contains(open,point,plane,0))
        if rg.PointContainment.Inside not in contain:
            pattern.Add(True,myPath)
        else:
            pattern.Add(False,myPath)
```

7.2.3. Plate contour generation script

```
plates = DataTree[rg.Curve]()
open_plates = DataTree[rg.Curve]()

for i in range(structured_panels.BranchCount):
    myPath = GH_Path(i)
    for pan in structured_panels.Branch(i):
        ## top and bottom plates
        segs = rg.PolylineCurve.DuplicateSegments(pan)
        horizontals = []
        for seg in segs:
            p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
            p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
            if abs(p1[2]-p2[2]) < 10:
                horizontals.append(seg)
        for hor in horizontals:
            t1 = hor.PointAtNormalizedLength(0)
            t2 = hor.PointAtNormalizedLength(1)
            temp1 = hor.Offset(plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
            temp_line = temp1[0]
            temp2 = hor.Offset(plane,-plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
            point = temp_line.PointAtNormalizedLength(0.5)
            contain = rg.Curve.Contains(pan,point,plane,0)
            if contain == rg.PointContainment.Inside:
                t3 = rg.LineCurve.PointAtNormalizedLength(temp_line,1)
                t4 = rg.LineCurve.PointAtNormalizedLength(temp_line,0)
                bt = [t1,t2,t3,t4,t1]
                plate = rg.PolylineCurve(bt)
                plates.Add(plate,myPath)
            else:
                t3 = rg.LineCurve.PointAtNormalizedLength(temp2[0],1)
                t4 = rg.LineCurve.PointAtNormalizedLength(temp2[0],0)
                bt = [t1,t2,t3,t4,t1]
                plate = rg.PolylineCurve(bt)
                plates.Add(plate,myPath)

        for open in structured_openings.Branch(i):
            segs = rg.PolylineCurve.DuplicateSegments(open)
            horizontals_open = []
            for seg in segs:
                p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
                p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
                if abs(p1[2]-p2[2]) < 10:
                    horizontals_open.append(seg)
            for hor in horizontals_open:
                t1 = hor.PointAtNormalizedLength(0)
                t2 = hor.PointAtNormalizedLength(1)
                temp1 = hor.Offset(plane,plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
                temp_line = temp1[0]
                temp2 = hor.Offset(plane,-plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
                point = temp_line.PointAtNormalizedLength(0.5)
                contain = rg.Curve.Contains(open,point,plane,0)
```

```
        if contain == rg.PointContainment.Outside:
            t3 = rg.LineCurve.PointAtNormalizedLength(temp_line,1)
            t4 = rg.LineCurve.PointAtNormalizedLength(temp_line,0)
            bt = [t1,t2,t3,t4,t1]
            plate = rg.PolylineCurve(bt)
            open_plates.Add(plate,myPath)
        else:
            t3 = rg.LineCurve.PointAtNormalizedLength(temp2[0],1)
            t4 = rg.LineCurve.PointAtNormalizedLength(temp2[0],0)
            bt = [t1,t2,t3,t4,t1]
            plate = rg.PolylineCurve(bt)
            open_plates.Add(plate,myPath)
for open in multi_pan_open:
    segs = rg.PolylineCurve.DuplicateSegments(open)
    horizontals_open = []
    for seg in segs:
        p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
        p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
        if abs(p1[2]-p2[2]) < 10:
            horizontals_open.append(seg)
    for hor in horizontals_open:
        temp1 = hor.Offset(plane,plate_height/2,0,rg.CurveOffsetCornerStyle.Sharp)
        temp_line = temp1[0]
        temp2 = hor.Offset(plane,-plate_height,0,rg.CurveOffsetCornerStyle.Sharp)
        point = temp_line.PointAtNormalizedLength(0.5)
        contain = rg.Curve.Contains(open,point,plane,0)
        if contain == rg.PointContainment.Outside:
            c_line = temp_line
        else:
            c_line = temp2[0]
        params = []
        events = rg.Intersect.Intersection.CurveCurve(c_line,pan,0,0)
        for event in events:
            params.append(event.ParameterA)
        temp = c_line.Split(params)
        for line in temp:
            point = line.PointAtNormalizedLength(0.5)
            contain = rg.Curve.Contains(pan,point,plane,0)
            if contain == rg.PointContainment.Inside:
                plate_c_line = line
        temp1 = plate_c_line.Offset(plane,plate_height/2,0,rg.CurveOffsetCornerStyle.Sharp)
        temp2 = plate_c_line.Offset(plane,-plate_height/2,0,rg.CurveOffsetCornerStyle.Sharp)
        temp_line1 = temp1[0]
        temp_line2 = temp2[0]
        t1 = temp_line1.PointAtNormalizedLength(0)
        t2 = temp_line1.PointAtNormalizedLength(1)
        t3 = temp_line2.PointAtNormalizedLength(1)
        t4 = temp_line2.PointAtNormalizedLength(0)
        bt = [t1,t2,t3,t4,t1]
        plate = rg.PolylineCurve(bt)
        open_plates.Add(plate,myPath)
```

7.2.4. Rafter contour generation script

```
rafter_min = rafter_min+rafter_width
studs = DataTree[rg.Curve]()
stud_center_lines = DataTree[rg.Curve]()
a = []
for i in range(structured_panels.BranchCount):
    myPath = GH_Path(i)
    for pan in structured_panels.Branch(i):
        segs = pan.DuplicateSegments()
        x_cords_pan = []
        horizontals = []
        for seg in segs:
            point = seg.PointAtNormalizedLength(0.5)
            x_cords_pan.append(point[0])
            p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
            p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
            if abs(p1[2]-p2[2]) < 10:
                horizontals.append(seg)
        dom_pan = [round(min(x_cords_pan)),round(max(x_cords_pan))]
        open_stud_cords = []
        for open in structured_openings.Branch(i):
            segs = open.DuplicateSegments()
            open_cords = []
            for seg in segs:
                point = seg.PointAtNormalizedLength(0.5)
                open_cords.append(point[0])
            open_stud_cords.append(round(min(open_cords))-rafter_width/2)
            open_stud_cords.append(round(max(open_cords))+rafter_width/2)

        for open in multi_pan_open:
            segs = open.DuplicateSegments()
            open_cords = []
            for seg in segs:
                point = seg.PointAtNormalizedLength(0.5)
                open_cords.append(point[0])
            temp_dom_min = round(min(open_cords))
            temp_dom_max = round(max(open_cords))
            if temp_dom_min > dom_pan[0] and temp_dom_min < dom_pan[1]:
                dom_min = temp_dom_min-rafter_width/2
            elif temp_dom_min <= dom_pan[0]:
                dom_min = temp_dom_min-(2*rafter_min+2*rafter_width)
            else:
                dom_min = temp_dom_min+(2*rafter_min+2*rafter_width)
            if temp_dom_max > dom_pan[0] and temp_dom_max < dom_pan[1]:
                dom_max = temp_dom_max+rafter_width/2
            elif temp_dom_max <= dom_pan[0]:
                dom_max = temp_dom_max-(2*rafter_min+2*rafter_width)
            else:
                dom_max = temp_dom_max+(2*rafter_min+2*rafter_width)
            open_stud_cords.append(dom_min)
            open_stud_cords.append(dom_max)
        stud_cords = [dom_pan[0]+rafter_width/2]
```

```
while stud_cords[-1] < dom_pan[1]:
    temp_cord = stud_cords[-1]+rafter_int
    changed = False
    for open in open_stud_cords:
        if temp_cord > open-rafter_min and temp_cord < open:
            changed = True
            stud_cords.append(open-rafter_min)
            stud_cords.append(open)
        elif temp_cord >= open and temp_cord < open+rafter_min:
            changed = True
            stud_cords.append(open)
        if stud_cords[-1] < open-rafter_min and temp_cord > open+rafter_min:
            changed = True
            stud_cords.append(open)
    if changed == False:
        stud_cords.append(temp_cord)
stud_cords.pop(-1)
if stud_cords[-1] > dom_pan[1]-rafter_width/2-rafter_min:
    stud_cords.pop(-1)
    stud_cords.append(dom_pan[1]-rafter_width/2-rafter_min)
stud_cords.append(dom_pan[1]-rafter_width/2)
stud_c_lines = []
for cord in stud_cords:
    t1 = rg.LineCurve.PointAtNormalizedLength(horizontals[0],0.5)
    t2 = rg.LineCurve.PointAtNormalizedLength(horizontals[1],0.5)
    p1 = rg.Point3d(cord,t1[1],t1[2])
    p2 = rg.Point3d(cord,t2[1],t2[2])
    tline = (rg.LineCurve(p1,p2))
    params = []
    for plate in plates.Branch(i):
        events = rg.Intersect.Intersection.CurveCurve(tline,plate,0,0)
        for event in events:
            params.append(event.ParameterA)
    for plate in o_plates.Branch(i):
        events = rg.Intersect.Intersection.CurveCurve(tline,plate,0,0)
        for event in events:
            params.append(event.ParameterA)
    temp = tline.Split(params)
    for line in temp:
        point = rg.LineCurve.PointAtNormalizedLength(line,0.5)
        plate_cont = []
        open_cont = []
        for plate in plates.Branch(i):
            plate_cont.append(rg.Curve.Contains(plate,point,plane,0))
        for plate in o_plates.Branch(i):
            plate_cont.append(rg.Curve.Contains(plate,point,plane,0))
        for open in structured_openings.Branch(i):
            open_cont.append(rg.Curve.Contains(open,point,plane,0))
        for open in multi_pan_open:
            open_cont.append(rg.Curve.Contains(open,point,plane,0))
```

```
            if rg.PointContainment.Inside not in plate_cont and rg.PointContainment.Inside not in
open_cont:
                stud_c_lines.append(line)
        for line in stud_c_lines:
            stud_center_lines.Add(line,myPath)
            temp1 = rg.LineCurve.Offset(line,plane,-rafter_width/2,0,rg.CurveOffsetCornerStyle.Sharp)
            temp2 = rg.LineCurve.Offset(line,plane,rafter_width/2,0,rg.CurveOffsetCornerStyle.Sharp)
            temp1_line = temp1[0]
            temp2_line = temp2[0]
            p1 = temp1_line.PointAtNormalizedLength(0)
            p2 = temp1_line.PointAtNormalizedLength(1)
            p3 = temp2_line.PointAtNormalizedLength(1)
            p4 = temp2_line.PointAtNormalizedLength(0)
            bt = [p1,p2,p3,p4,p1]
            temp = rg.PolylineCurve(bt)
            studs.Add(temp,myPath)
```

7.2.5. Overhang generation script

```
framework = DataTree[rg.Curve]()
rafter_center_lines = DataTree[rg.Curve]()
board = DataTree[rg.Curve]()

for i in range(structured_panels.BranchCount):
  for pan in structured_panels.Branch(i):
    myPath = GH_Path(i)
    ##bottom member generator
    segs = rg.PolylineCurve.DuplicateSegments(pan)
    horizontals = []
    for seg in segs:
      p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
      p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
      if abs(p1[2]-p2[2]) < 10:
        horizontals.append(seg)
    temp1 = rg.LineCurve.PointAtNormalizedLength(horizontals[0],0.5)
    temp2 = rg.LineCurve.PointAtNormalizedLength(horizontals[1],0.5)
    if temp1[2] < temp2[2]:
      eaves_line = horizontals[0]
    else:
      eaves_line = horizontals[1]
    bottom_line = (eaves_line.Offset(plane,overhang,0,rg.CurveOffsetCornerStyle.Sharp))[0]
    contain = rg.Curve.Contains(pan,bottom_line.PointAtNormalizedLength(0.5),plane,0)
    if contain == rg.PointContainment.Inside:
      bottom_line = (eaves_line.Offset(plane,-overhang,0,rg.CurveOffsetCornerStyle.Sharp))[0]
      inverse = True
    else:
      inverse = False
    if overhang >= plate_height:
      bottom_member_size = plate_height
    else:
      bottom_member_size = overhang
    if inverse == True:
      top_plate_line =
(rg.Curve.Offset(bottom_line,plane,bottom_member_size,0,rg.CurveOffsetCornerStyle.Sharp))[0]
    else:
      top_plate_line = (rg.Curve.Offset(bottom_line,plane,-
bottom_member_size,0,rg.CurveOffsetCornerStyle.Sharp))[0]
    bp1 = rg.LineCurve.PointAtNormalizedLength(bottom_line,0)
    bp2 = rg.LineCurve.PointAtNormalizedLength(bottom_line,1)
    tp1 = rg.LineCurve.PointAtNormalizedLength(top_plate_line,1)
    tp2 = rg.LineCurve.PointAtNormalizedLength(top_plate_line,0)
    bt = bp1,bp2,tp1,tp2,bp1
    bot_plate = rg.PolylineCurve(bt)
    framework.Add(bot_plate,myPath)
    ###generation of rafters
    if overhang > plate_height:
      dom = [round(tp2[0]),round(tp1[0])]
      dom.sort()
      x_cords = [dom[0]+rafter_width/2]
      while x_cords[-1] < dom[1]-rafter_width/2-rafter_int:
```

```
            x_cords.append(x_cords[-1]+rafter_int)
        if x_cords[-1] > dom[1]-rafter_width/2-rafter_min:
            x_cords.pop(-1)
            x_cords.append(dom[1]-rafter_width/2-rafter_min)
        x_cords.append(dom[1]-rafter_width/2)
        temp_top = eaves_line.PointAtNormalizedLength(0.5)
        temp_bot = top_plate_line.PointAtNormalizedLength(0.5)
        for cord in x_cords:

rafter_center_lines.Add((rg.LineCurve(rg.Point3d(cord,temp_bot[1],temp_bot[2]),rg.Point3d(cord,t
emp_top[1],temp_top[2]))),myPath)
            tp1 = rg.Point3d(cord-rafter_width/2,temp_top[1],temp_top[2])
            tp2 = rg.Point3d(cord+rafter_width/2,temp_top[1],temp_top[2])
            bp1 =  rg.Point3d(cord-rafter_width/2,temp_bot[1],temp_bot[2])
            bp2 =  rg.Point3d(cord+rafter_width/2,temp_bot[1],temp_bot[2])
            bpt = [tp1,tp2,bp2,bp1,tp1]
            rafter = rg.PolylineCurve(bpt)
            framework.Add(rafter,myPath)
    t1 = eaves_line.PointAtNormalizedLength(1)
    t2 = eaves_line.PointAtNormalizedLength(0)
    t3 = bottom_line.PointAtNormalizedLength(0)
    t4 = bottom_line.PointAtNormalizedLength(1)
    tpp = [t1,t2,t3,t4,t1]
    board.Add(rg.PolylineCurve(tpp),myPath)
```

7.2.6. Wall beam generation script

```
wallbeam_curve = DataTree[rg.Curve]()

for i in range(structured_panels.BranchCount):
    for pan in structured_panels.Branch(i):
        myPath = GH_Path(i)
        ##bottom member generator
        segs = rg.PolylineCurve.DuplicateSegments(pan)
        horizontals = []
        for seg in segs:
            p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
            p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
            if abs(p1[2]-p2[2]) < 10:
                horizontals.append(seg)
        temp1 = rg.LineCurve.PointAtNormalizedLength(horizontals[0],0.5)
        temp2 = rg.LineCurve.PointAtNormalizedLength(horizontals[1],0.5)
        if temp1[2] < temp2[2]:
            eaves_line = horizontals[0]
            ridge_line = horizontals[1]
        else:
            eaves_line = horizontals[1]
            ridge_line = horizontals[0]
        el0 = rg.LineCurve.PointAtNormalizedLength(eaves_line,0)
        el1 = rg.LineCurve.PointAtNormalizedLength(eaves_line,1)
        rl0 = rg.LineCurve.PointAtNormalizedLength(ridge_line,0)
        rl1 = rg.LineCurve.PointAtNormalizedLength(ridge_line,1)
        if el0[0] < el1[0]:
            bpl = el0
            bpr = el1
        else:
            bpl = el1
            bpr = el0
        if rl0[0] < rl1[0]:
            tpl = rl0
            tpr = rl1
        else:
            tpl = rl1
            tpr = rl0
        lef_line = rg.LineCurve(bpl,tpl)
        right_line = rg.LineCurve(bpr,tpr)
        bottom_left = rg.LineCurve.PointAtLength(lef_line,bottom_of_wallbeam)
        top_left  = rg.LineCurve.PointAtLength(lef_line,bottom_of_wallbeam+sizing_wb)
        bottom_right = rg.LineCurve.PointAtLength(right_line,bottom_of_wallbeam)
        top_right  = rg.LineCurve.PointAtLength(right_line,bottom_of_wallbeam+sizing_wb)
        points = [bottom_left,top_left,top_right,bottom_right,bottom_left]
        wallbeam_curve.Add(rg.PolylineCurve(points),myPath)
```

7.2.7. Lath's generation script

```
laths_vertical = DataTree[rg.Curve]()
laths_horizontal = DataTree[rg.Curve]()

for i in range(structured_panels.BranchCount):
  for pan in structured_panels.Branch(i):
    myPath = GH_Path(i)
    ##vertical laths
    vertical_lines = []
    for line in rafter_center_lines_pan.Branch(i):
      line = rg.LineCurve.Extend(line,rg.CurveEnd.Both,plate_height,rg.CurveExtensionStyle.Line)
      vertical_lines.append(line)
    for line in rafter_center_lines_oh.Branch(i):
      line = rg.LineCurve.Extend(line,rg.CurveEnd.Start,plate_height,rg.CurveExtensionStyle.Line)
      vertical_lines.append(line)
    for line in vertical_lines:
      temp1 = rg.LineCurve.PointAtNormalizedLength(line,0)
      temp2 = rg.LineCurve.PointAtNormalizedLength(line,1)
      p1 = rg.Point3d(temp1[0]-laths_width/2,temp1[1],temp1[2])
      p2 = rg.Point3d(temp1[0]+laths_width/2,temp1[1],temp1[2])
      p3 = rg.Point3d(temp2[0]+laths_width/2,temp2[1],temp2[2])
      p4 = rg.Point3d(temp2[0]-laths_width/2,temp2[1],temp2[2])
      temppoints = [p1,p2,p3,p4,p1]
      laths_vertical.Add(rg.PolylineCurve(temppoints),myPath)
    segs = rg.PolylineCurve.DuplicateSegments(pan)
    horizontals = []
    for seg in segs:
      p1 = rg.LineCurve.PointAtNormalizedLength(seg,0)
      p2 = rg.LineCurve.PointAtNormalizedLength(seg,1)
      if abs(p1[2]-p2[2]) < 10:
        horizontals.append(seg)
    temp1 = rg.LineCurve.PointAtNormalizedLength(horizontals[0],0.5)
    temp2 = rg.LineCurve.PointAtNormalizedLength(horizontals[1],0.5)
    if temp1[2] < temp2[2]:
      eaves_line = horizontals[0]
      ridge_line = horizontals[1]
    else:
      eaves_line = horizontals[1]
      ridge_line = horizontals[0]
    el0 = rg.LineCurve.PointAtNormalizedLength(eaves_line,0)
    el1 = rg.LineCurve.PointAtNormalizedLength(eaves_line,1)
    rl0 = rg.LineCurve.PointAtNormalizedLength(ridge_line,0)
    rl1 = rg.LineCurve.PointAtNormalizedLength(ridge_line,1)
    if el0[0] < el1[0]:
      bpl = el0
      bpr = el1
    else:
      bpl = el1
      bpr = el0
    if rl0[0] < rl1[0]:
      tpl = rl0
      tpr = rl1
```

```
    else:
        tpl = rl1
        tpr = rl0
    left_line_temp = rg.LineCurve(bpl,tpl)
    right_line_temp = rg.LineCurve(bpr,tpr)
    left_line =
rg.LineCurve.Extend(left_line_temp,rg.CurveEnd.Start,overhang,rg.CurveExtensionStyle.Line)
    right_line =
rg.LineCurve.Extend(right_line_temp,rg.CurveEnd.Start,overhang,rg.CurveExtensionStyle.Line)
    domain = [0,int(rg.LineCurve.GetLength(left_line))]
    cords = [domain[0]+laths_width/2]
    while cords[-1] < domain[1]-laths_int:
        cords.append(cords[-1]+laths_int)
        if laths_int == 0:
            break
    hor_center_lines = []
    for cord in cords:
        pl = rg.LineCurve.PointAtLength(left_line,cord)
        pr = rg.LineCurve.PointAtLength(right_line,cord)
        temp_line =(rg.LineCurve(pl,pr))
        params = []
        for open in structured_openings.Branch(i):
            events = rg.Intersect.Intersection.CurveCurve(temp_line,open,0,0)
            for event in events:
                params.append(event.ParameterA)
        for open in multi_pan_open:
            events = rg.Intersect.Intersection.CurveCurve(temp_line,open,0,0)
            for event in events:
                params.append(event.ParameterA)
        if params == []:
            hor_center_lines.append(temp_line)
        else:
            temp_lines = temp_line.Split(params)
            for line in temp_lines:
                point = line.PointAtNormalizedLength(0.5)
                contain = []
                for open in structured_openings.Branch(i):
                    contain.append(rg.Curve.Contains(open,point,plane,0))
                for open in multi_pan_open:
                    contain.append(rg.Curve.Contains(open,point,plane,0))
                if rg.PointContainment.Inside not in contain:
                    hor_center_lines.append(line)
    for line in hor_center_lines:
        temp1 = rg.Curve.Offset(line,plane,laths_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
        temp2 = rg.Curve.Offset(line,plane,-laths_width/2,0,rg.CurveOffsetCornerStyle.Sharp)[0]
        p1l = rg.LineCurve.PointAtNormalizedLength(temp1,0)
        p1r = rg.LineCurve.PointAtNormalizedLength(temp1,1)
        p2l = rg.LineCurve.PointAtNormalizedLength(temp2,0)
        p2r = rg.LineCurve.PointAtNormalizedLength(temp2,1)
        temp_points_hor = [p1l,p1r,p2r,p2l,p1l]
        laths_horizontal.Add(rg.PolylineCurve(temp_points_hor),myPath)
```

7.3 Bay window script