

BLD226681

Automated Structural Load Calculation using Dynamo and Revit

Achintya Bhat
Stantec

Soojung Kim
Stantec

Learning Objectives

- Learn how to develop an efficient workflow to automate rain load for irregularly shaped roofs using the architectural model and Dynamo player
- Learn how to develop a Dynamo script to calculate/annotate snow load for irregularly shaped roofs using Revit model and engineering standards
- Learn how to automate calculation of wind loads using Dynamo
- Learn how to increase Revit modeling productivity by automating load annotations in the Revit model using Dynamo player

Description

This class will introduce the attendees to the current Revit workflow for calculating structural loads using Revit software and their limitations. As the roof gets more complicated, it is difficult to calculate the rain and snow loads using the traditional Revit workflow. By integrating this logic into Dynamo, we can increase the efficiency and usability of the load calculation process. We will demonstrate how to use the architectural model to automate the rain load and snow load calculations on irregular roofs. We will show how to consider roof slopes and relationships between drainages to calculate rain volume and how to extract snow load source areas to calculate the snow drift load. Further, we will demonstrate how we can expand the workflow to automate the same for wind load. At the end of the class, attendees will learn an efficient workflow to automate the structural load calculation and annotation in Revit with the click of a few buttons using Dynamo player, thus reducing 1-2 days of work to less than an hour.

Your AU Experts

Achintya Bhat is a computational designer at Stantec who works within the Innovative Technology Development team. She has a background in engineering and a master's in Project and Construction Management from the University of British Columbia. Achintya specializes in using Business Intelligence (BI) tools, and visual scripting tools like Dynamo to automate workflows for architects, engineers and project managers. Recently, building upon her engineering background, she has been working towards increasing the efficiency in structural engineering by using visual programming and algorithmic thinking to automate structural design calculations. Achintya strives to advance data-driven design across AEC with automation of the profession's mundane chores to allow more time for creativity.

achintya.bhat@stantec.com

Soojung Kim is a BIM designer in Stantec and she acts as a BIM lead for the engineering team. She provides BIM supports including BIM modeling, visual programming, and data management. Recently, she's focusing on creating visual programming scripts to improve engineering analysis and modeling process. She has a background in Architectural engineering

and Civil engineering and pursued her master's degree studying BIM for Project/Construction management from the University of British Columbia.

soojung.kim@stantec.com

Rain Load Calculation

In this section, we have explained the concepts of the logic used in the script to automate the calculation of roof slopes, rain ponding height, and in turn, the rain load. Firstly, we will briefly introduce our current workflow.

Current Workflow

The current workflow involves the structural BIM designer manually placing masses using the Model in-place function in Revit. Then, the BIM designer calculates the rain volume based on the rainfall statistics of the city and the projected area of the roof. Next, the height of the Model in-place mass is varied gradually to meet the rain volume. Figure 1 and Figure 2 below illustrate the workflow.

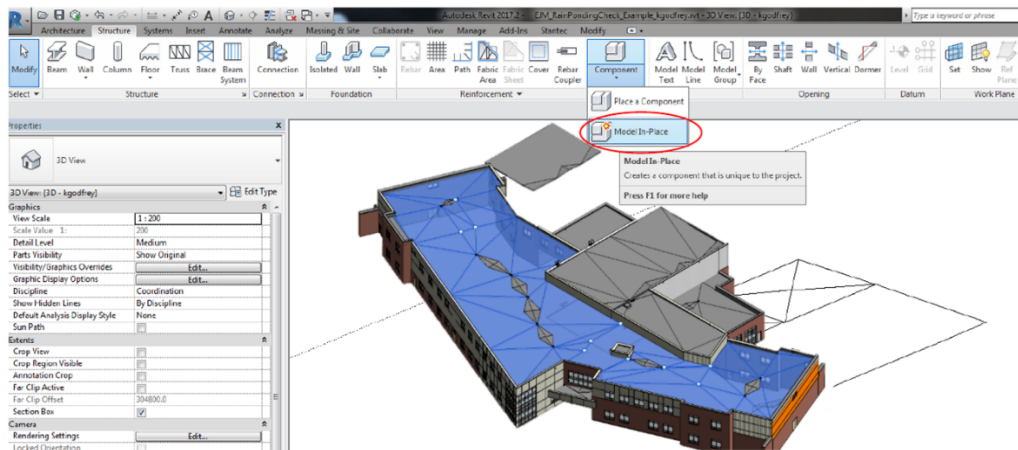


Figure 1: Revit Conceptual Mass illustrating Roof geometry

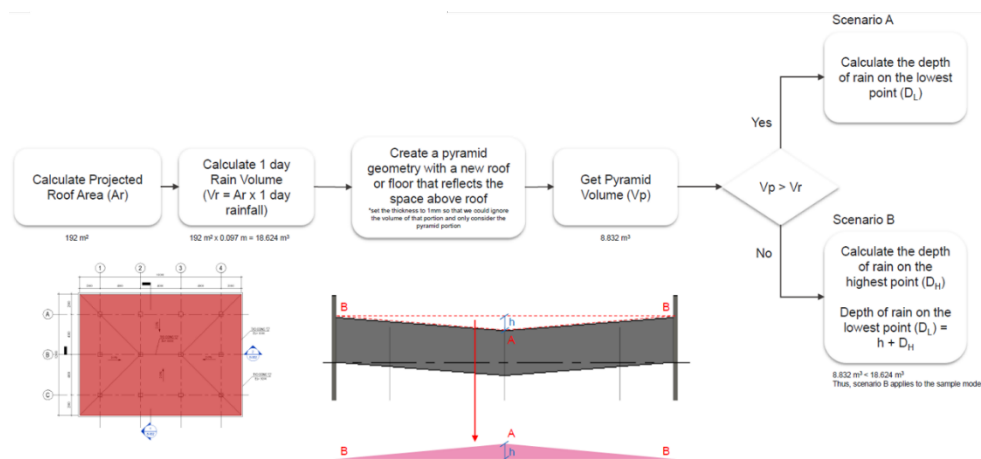


Figure 2: Current Workflow

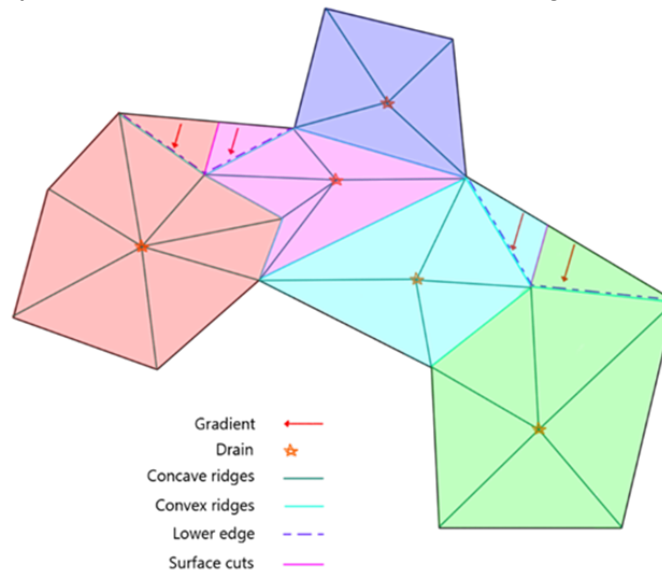
Dynamo Workflow

Assumptions

- Roofs are modeled in the architectural model with sloped in 3D
- Surfaces drain equally to all the drains if it has more than one drainage point

In this section, we will discuss the logics used in the script to automate the rain load calculations. As one can observe from Figure 2, there are several parameters that need to be identified or calculated in this script to calculate the final rain load. In addition to that, there are some assumptions that we have made based on our structural current practices and experience. In this sub section, we will explain the concepts of the logic used to convert our institutional knowledge to a dynamo script.

We divided the roof into multiple Drainage Areas as shown in Figure 3 below. Each drainage area is made up of surfaces which intersect and/or slope towards a drainage point. Next, we have grouped these Drainage areas into Overflow groups. This is done to identify the neighboring drainage areas the rain water would overflow into once each drainage area fills up. We also need to identify at what elevation, it will start overflowing (Refer Figure 4).



Drainage Areas: Grouping roof surfaces based on spatial intersection and slope gradient

Figure 3: Drainage Areas based on Grouping

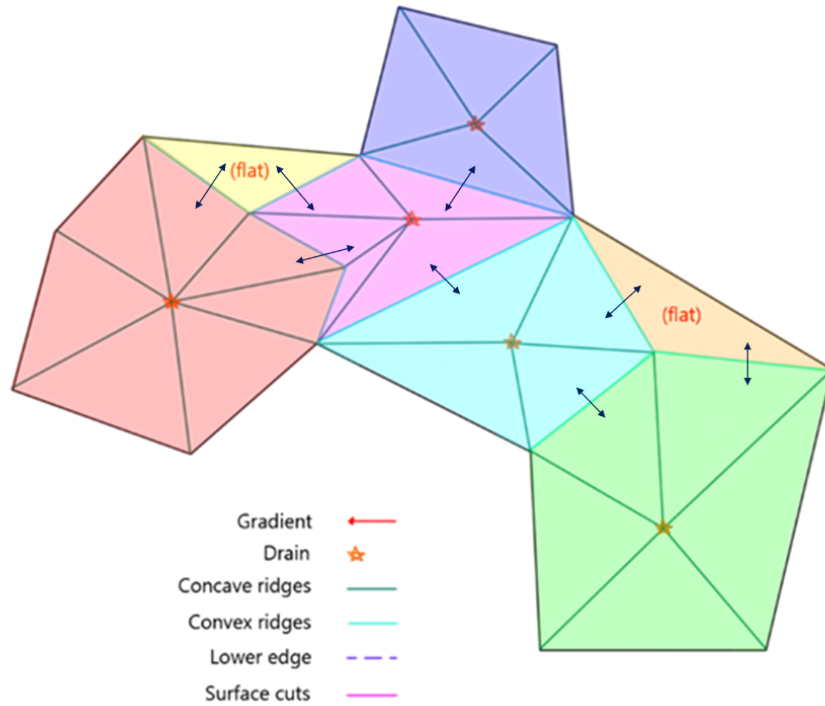


Figure 4: Grouping Drainage Areas Based on Overflow Elevation

We have divided the Dynamo script into five main sections/functions as shown in Figure 5. In this class we will talk in detail about section I, section II, section III and briefly about section IV and section V. Section I talks about finding the roof surface's slopes. Under this section, we will also explain how to use this data to automate slope annotations. Section II will discuss about grouping the surfaces to create drainage areas and section III will discuss grouping drainage areas into overflow groups after determining the overflow elevations. Sections IV will talk briefly about the custom python node which was developed to determine the rain ponding height and section V will talk about automation of rain load annotations on sheets.

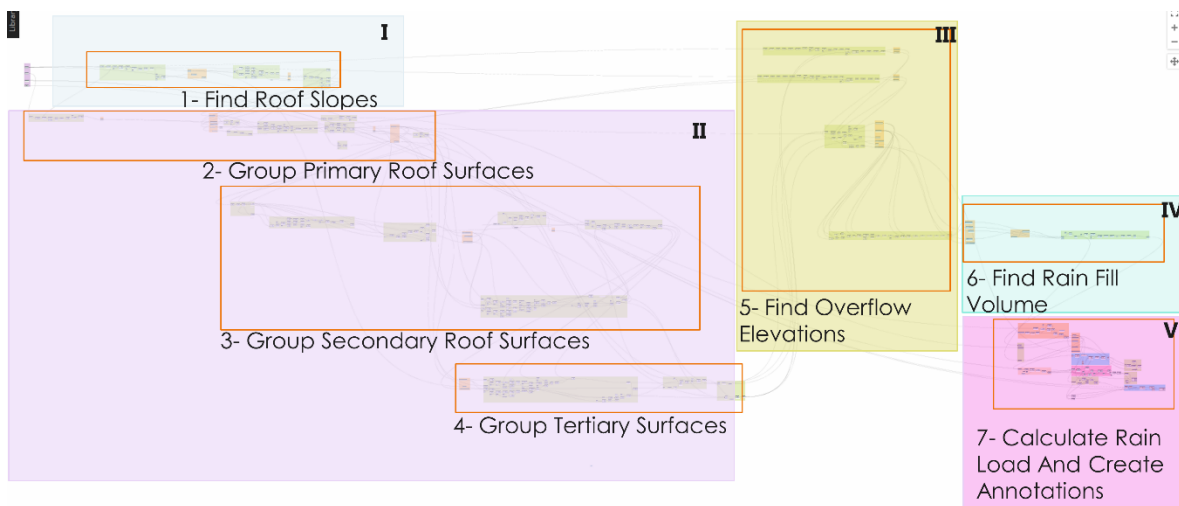


Figure 5: Rain Load Automation Dynamo Script Overview

Finding Roof Slope

In this step, there are four main parts:

- A. Getting Roof top surfaces
- B. Calculating Surface slopes
- C. Calculate Slope Vector direction
- D. Creating slope annotations**

A. Getting Roof top surfaces

When we extract the geometry from a Revit element like a roof, by default, the surface normal of the element solid is as shown in Figure 6.

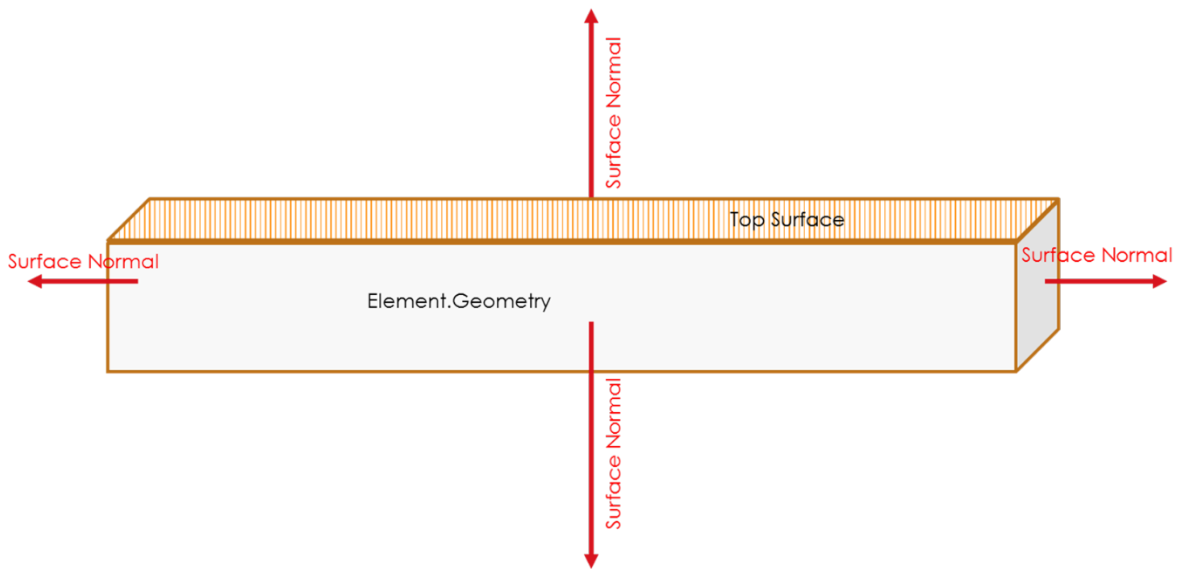


Figure 6: Dynamo Geometry Surface Normal

Hence, for the top surface, the Z vector is positive as shown in Figure 7.

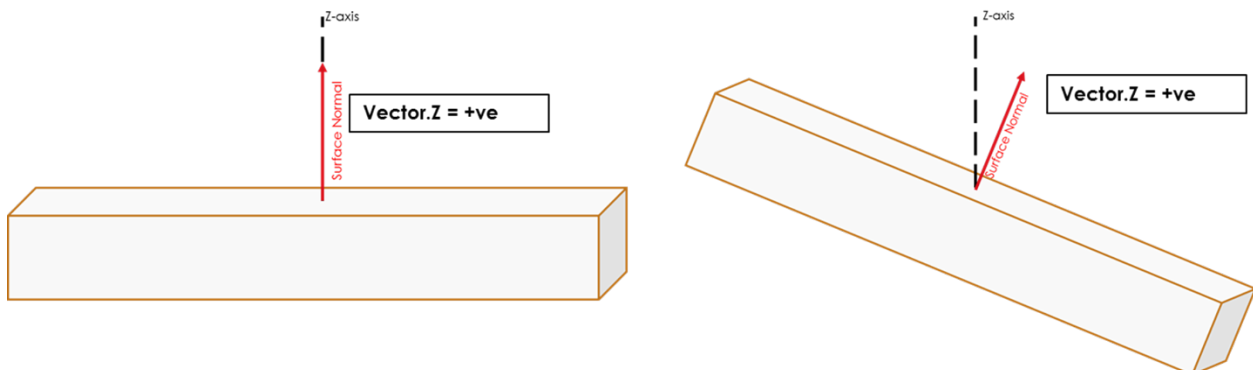


Figure 7: Top Surface Normal Vector Z is Positive

Figure 8 shows the dynamo script used to replicate this logic.

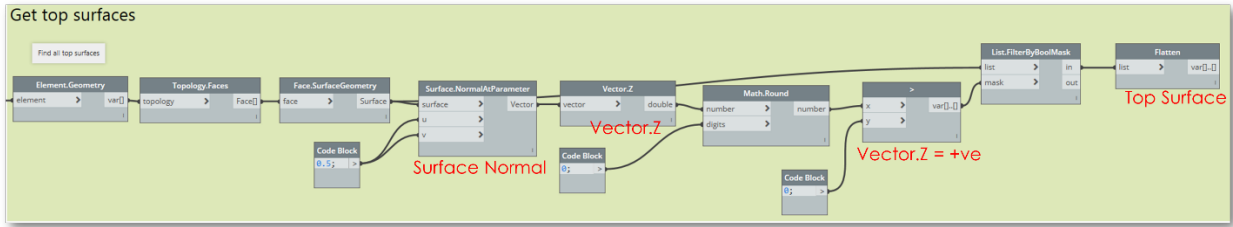


Figure 8: Getting Top Surface in Dynamo

B. Calculating Surface slopes

Now that we have identified the roof top surfaces, the next step is to determine the surface slope. We calculate roof slope as show in Figure 9.

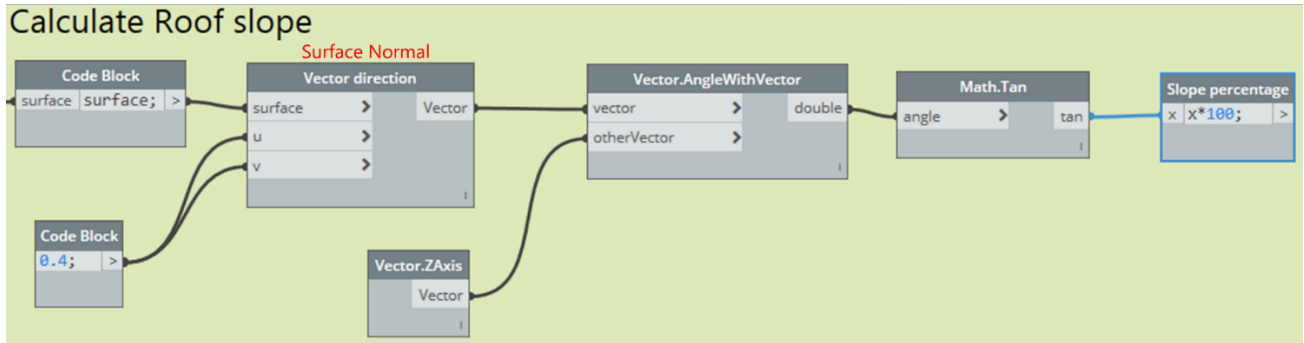
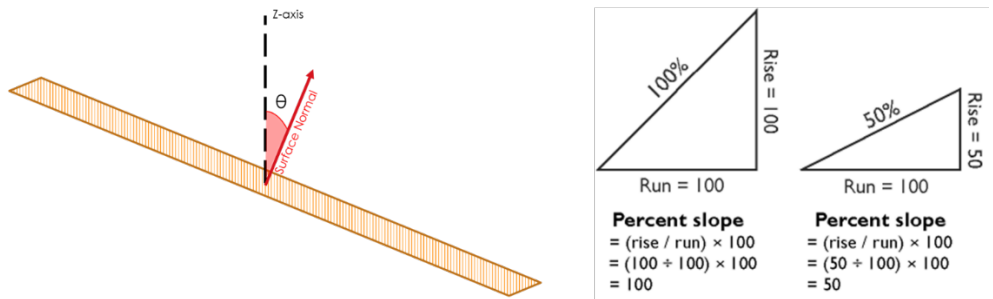


Figure 9: Calculate Roof slope gradient

C. Calculate Slope Vector direction

Now that we have the slope percentage, next we need to determine the slope vector direction as shown in Figure 10. We need the slope vector direction for the next steps in the rain load calculation script.

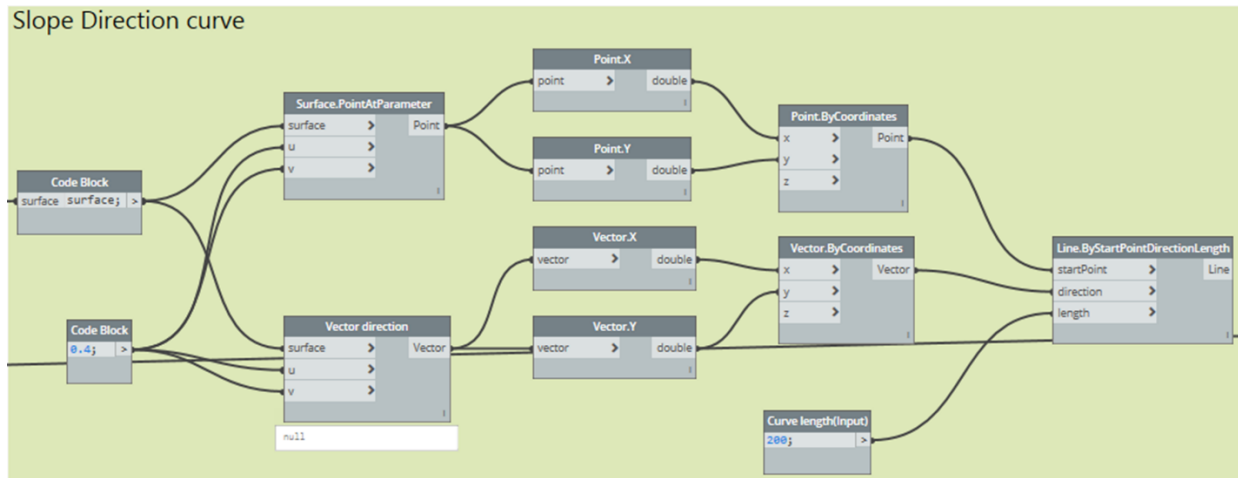


Figure 10: Determining Slope Vector Direction

D. Creating slope annotations

Figure 11 illustrates the dynamo script logic to create slope annotations. There are four sub steps for this step.

- Create annotation family instance
- Determine Rotation angle
- Rotate slope annotation instances
- Add slope % text

This ends Section I of the rain load calculation script.

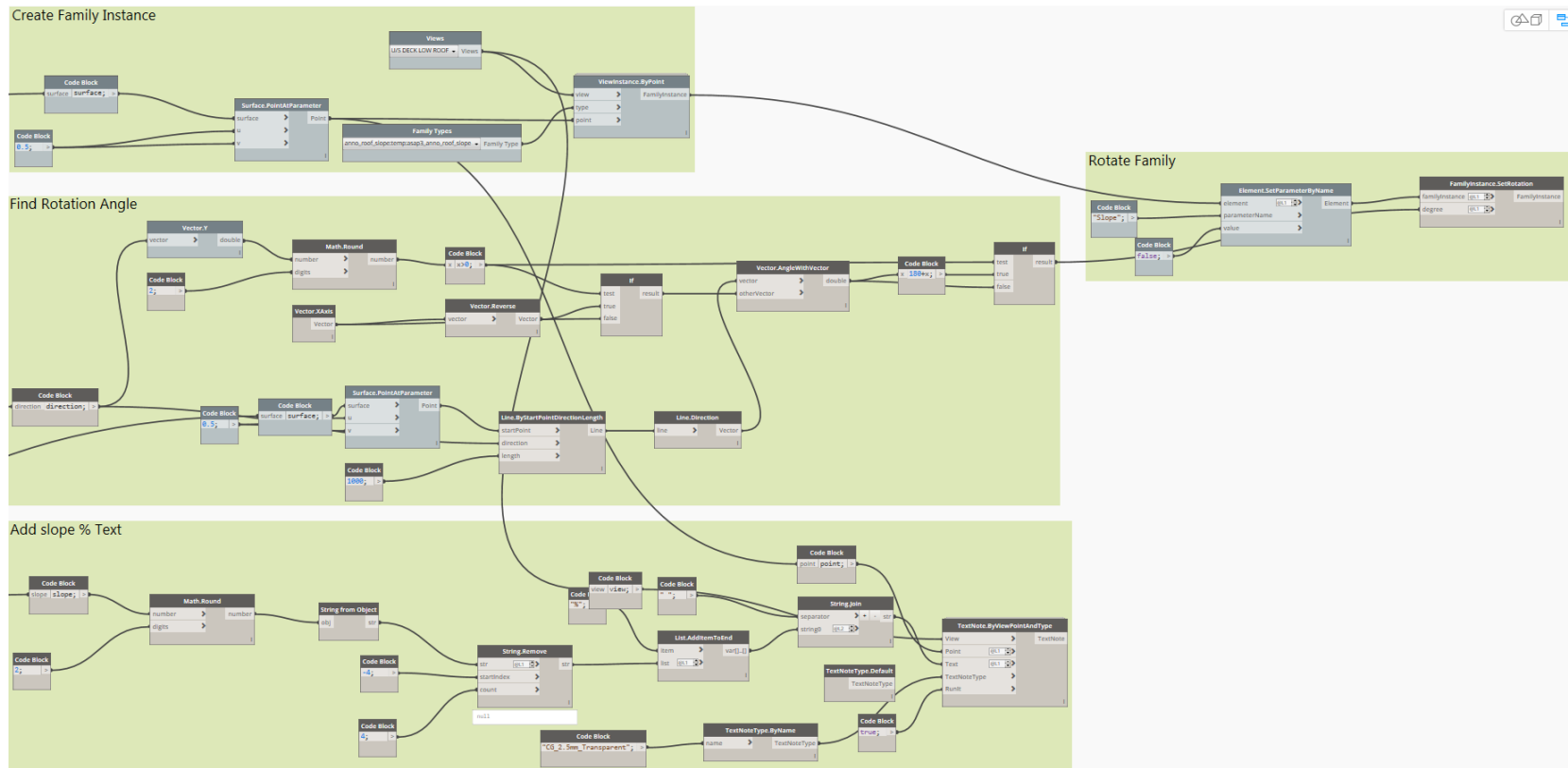


Figure 11: Dynamo script for Slope Annotation

Creating Drainage Areas Based on Spatial Intersection

The first step under the Section II of the rain load script is to identify all the drainage points with respect to the roof element under consideration. Based on our experience, we observed that the drainage elements are not always placed in the Revit model at the initial phase or if placed, the elements are sometimes not hosted or hosted with an offset which will fail the *Geometry.DoesIntersect* function in dynamo. Hence, we created a custom point-based family to manually identify all the drainage points to make sure we don't miss any of the drainage points. Depending on the project/team Revit standards, this step can be ignored. Figure 12 shoes how we extracted these drainage points into the dynamo platform and checked for intersection with the roof element. Then we filtered the list of drainage points to keep the ones that are on the roof under consideration for the script.

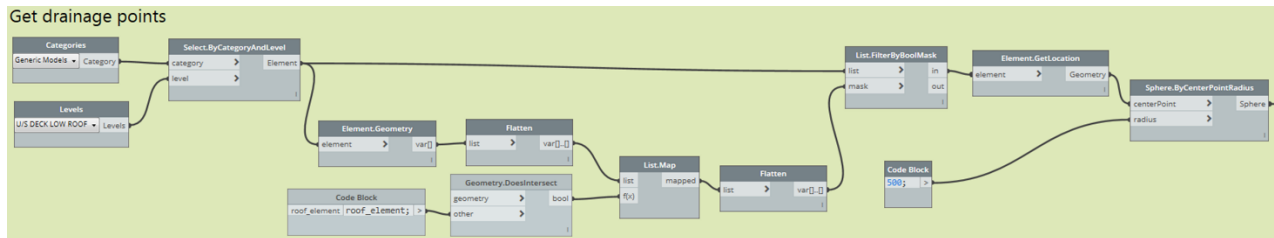


Figure 12: Dynamo script to Find Drainage Locations

Creating Surface ID

We find it useful to create a dictionary for the surfaces, or basically, giving them ID's helps with list management as you will see in this section. Figure 13 illustrates how we did this.

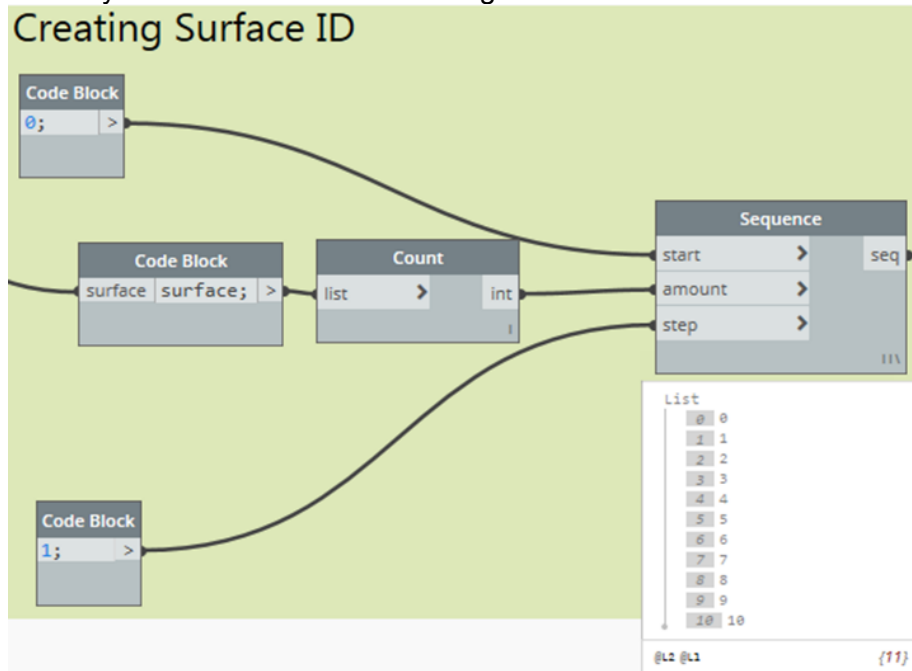


Figure 13: Creating Surface Dictionary for a roof element

Drain Relationship

We have divided the surfaces into primary, secondary and tertiary surfaces as illustrated in Figure 14. Primary surfaces are the surfaces which directly drain into drainage points. In other words, they intersect with drainage points. Secondary surfaces drain into primary surfaces, which means they intersect with primary surfaces. Finally, tertiary surfaces drain into secondary surfaces.

Drain Relationship

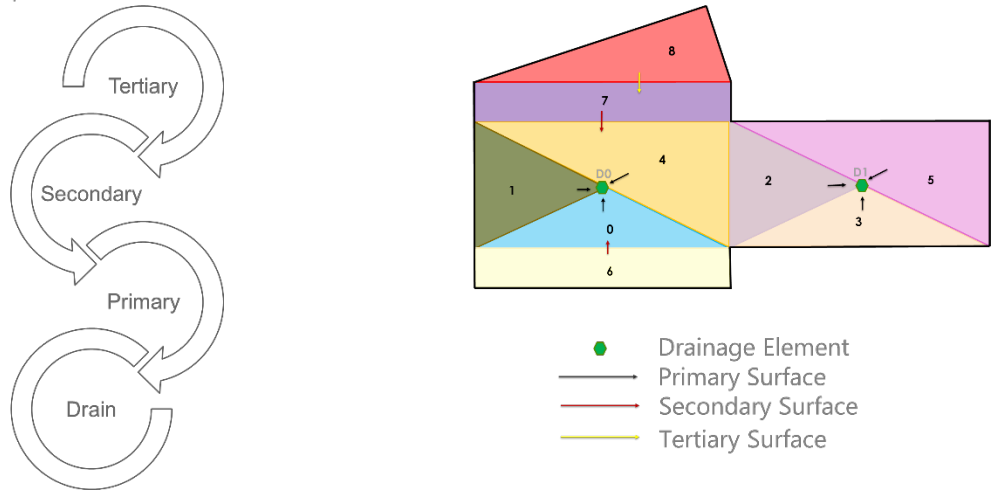


Figure 14: Drain and Surface Relationship

Finding Primary tributary surfaces

Figure 15 illustrates the primary surface relationship. As one can observe, we check for intersection with the drainage points and group surfaces by intersection with drainage points.

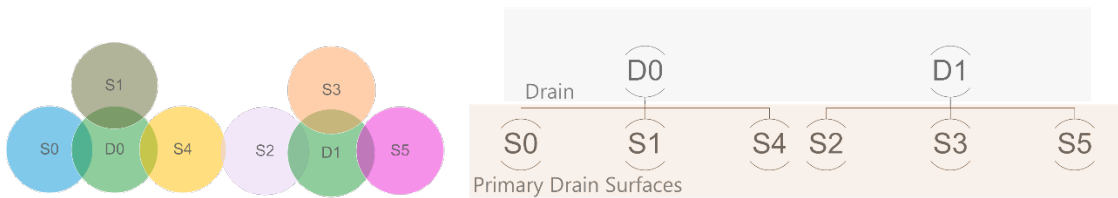


Figure 15: Primary Surface Relationship

We use the *List.Map* function to map the `geometry.doesintersect` function to all surfaces with the drainage points list. Then the list of indices of true values gives us the grouping of the primary surfaces as shown in Figure 16.

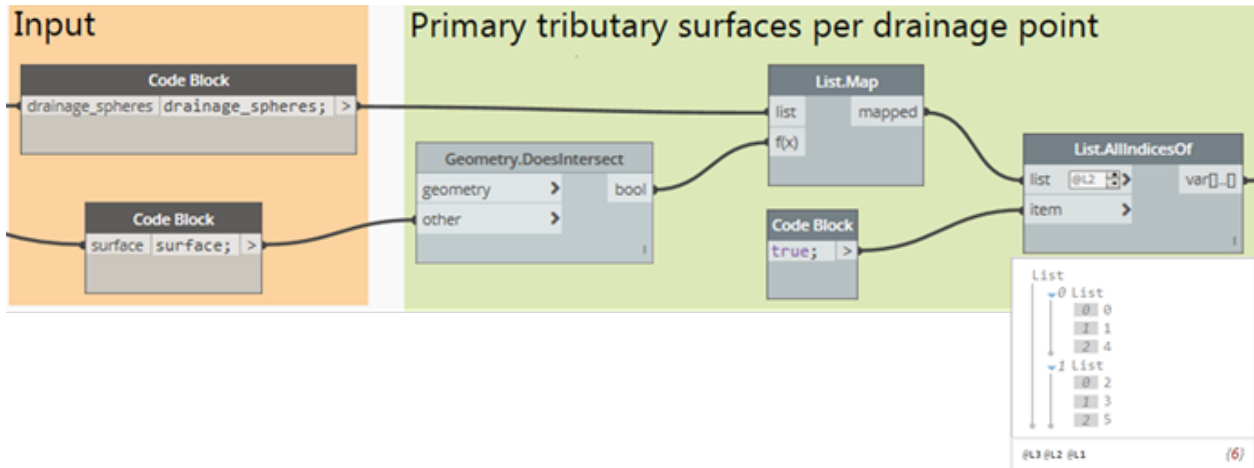


Figure 16: Finding and Grouping Primary Surfaces in Dynamo

Finding Secondary Tributary surfaces

Figure 17 illustrates the secondary surface relationship.

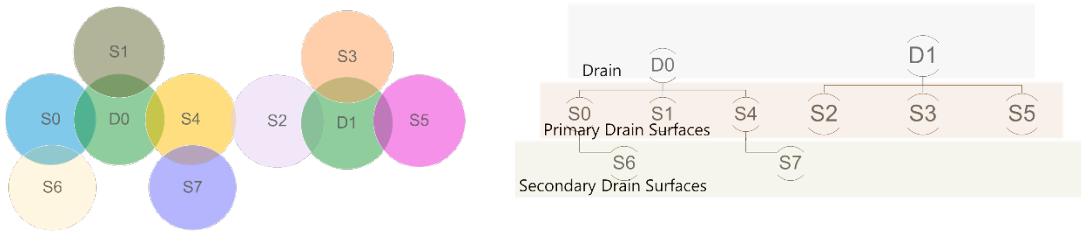


Figure 17: Secondary Surface Relationship

Here, we make a list of surfaces which were not included in the primary surface list. Then we use *List.Map* to map the *Geometry.DoesIntersect* function to all the remaining surfaces to the primary surface list. The list of indices of true values gives us the parent primary surface each of the remaining surfaces. If the parent is null, that means it's not a secondary surface. Hence, we deal with it in the tertiary group. Figure 18 shows this part of the dynamo script.

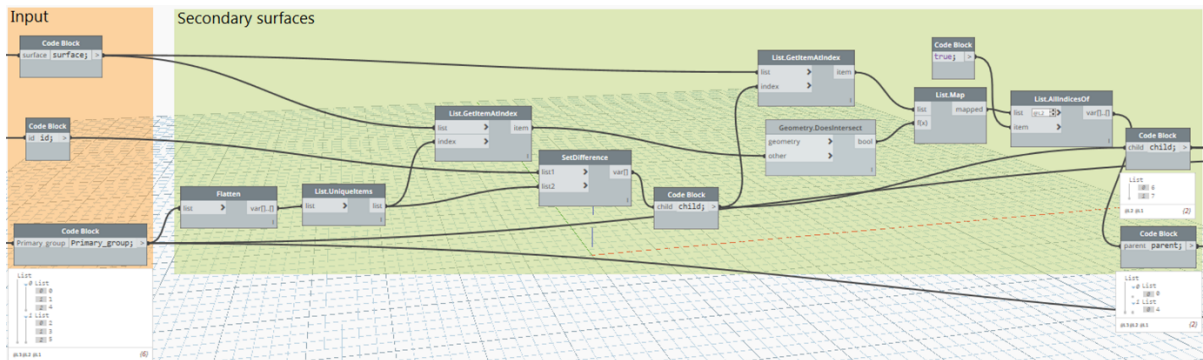


Figure 18: Finding Secondary Surfaces in Dynamo

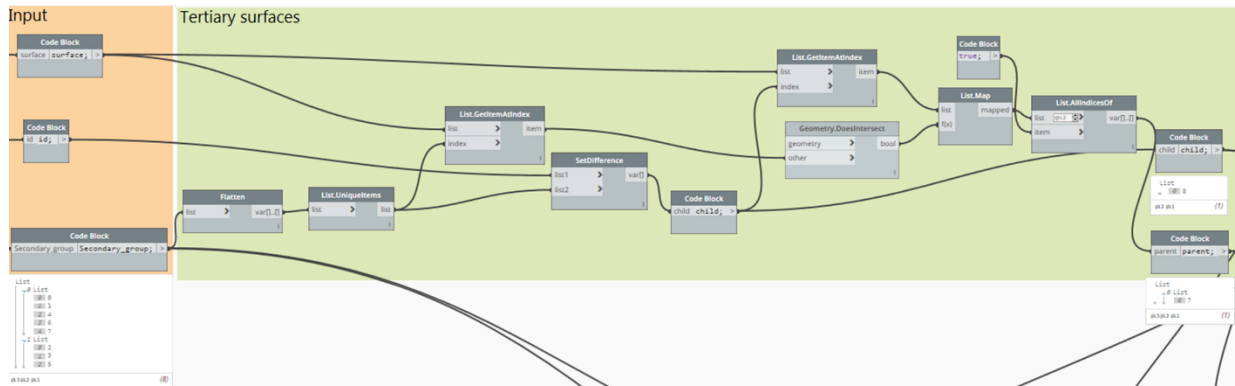


Figure 21: Finding Tertiary Surfaces in Dynamo

Once we have determined the tertiary surface and their parent surfaces, the next step is to add the tertiary surfaces to the group which contains its parents as shown in Figure 22.

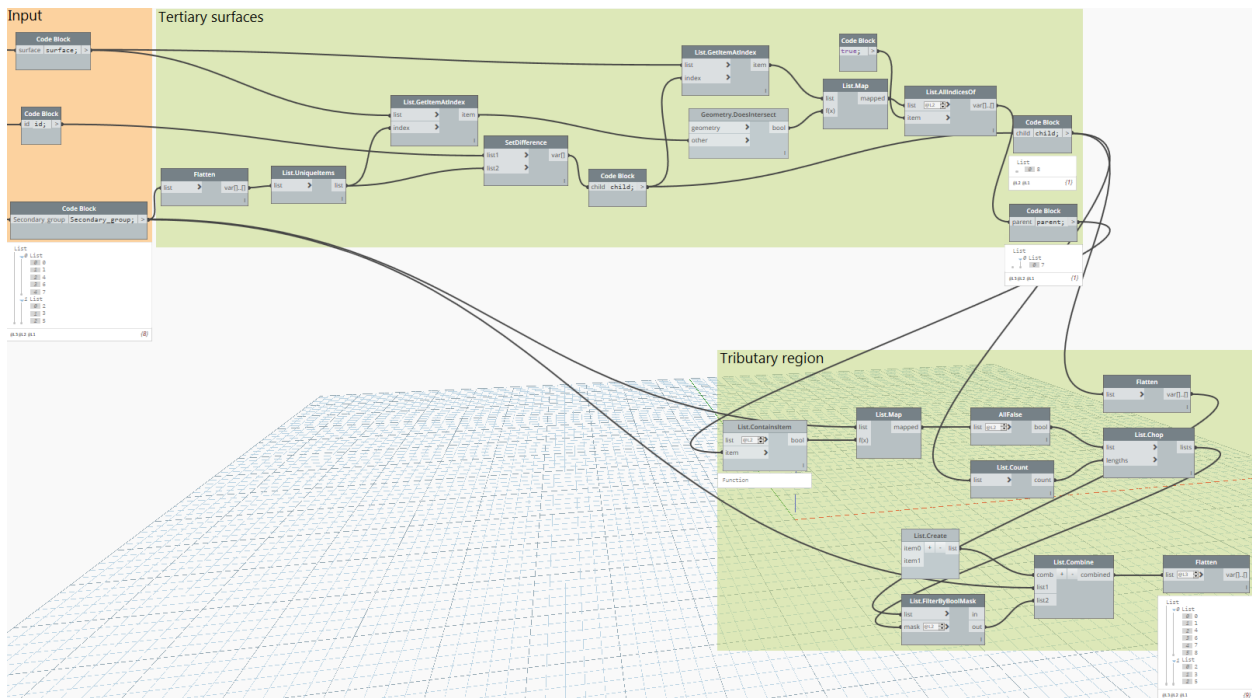


Figure 22: Grouping Tertiary Surfaces in Dynamo

Dealing with Dual Parent Surface Relationship Based on Slope Vector Direction

Another thing to consider here is, when a surface drains into two surfaces which drain into two different drainage points, we need to add an additional check (refer to Figure 23). We need to check the slope vector direction to identify which surface/s it would drain into. Sometimes the surface might be intersecting but the slope might be in another direction. In this case, we can neglect one of the parent surfaces (refer to Figure 24). But, when it drains into both the

surfaces, then we need to make an assumption. Based on our practice, we assumed that the surface with more than one parent would be divided as shown in Figure 25.

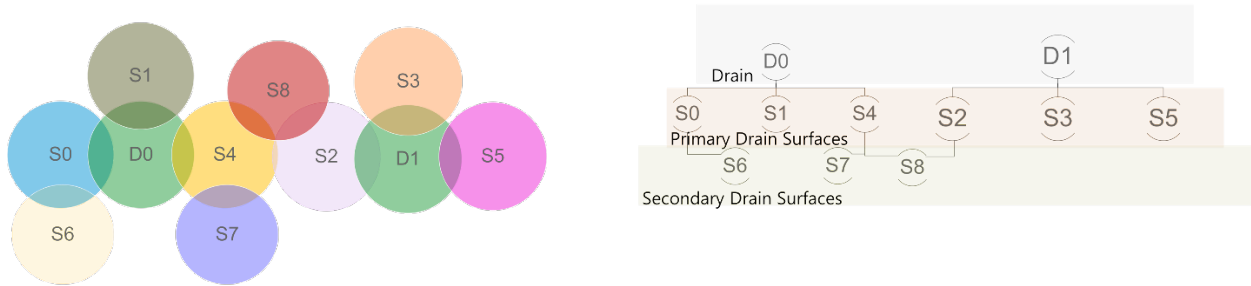


Figure 23: Dual Parent Relationships

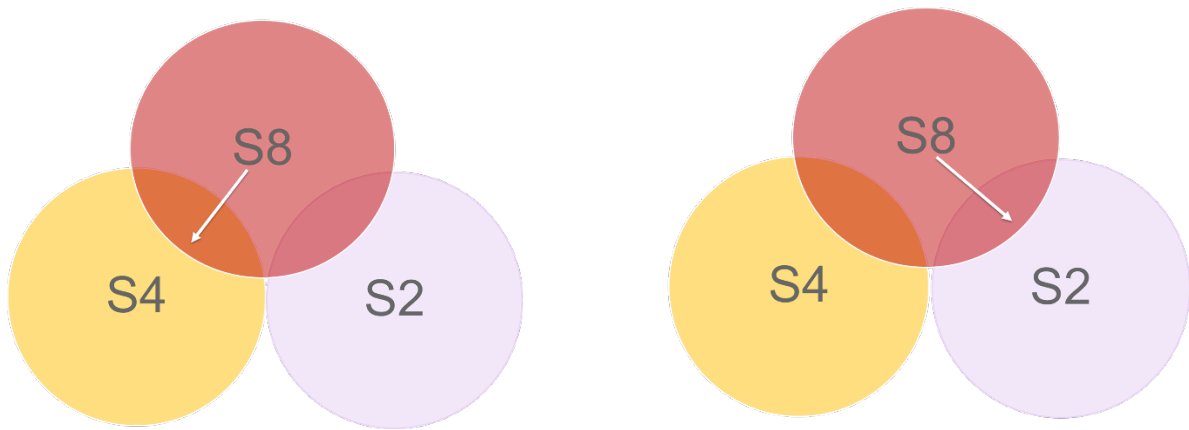


Figure 24: Check for Slope Direction

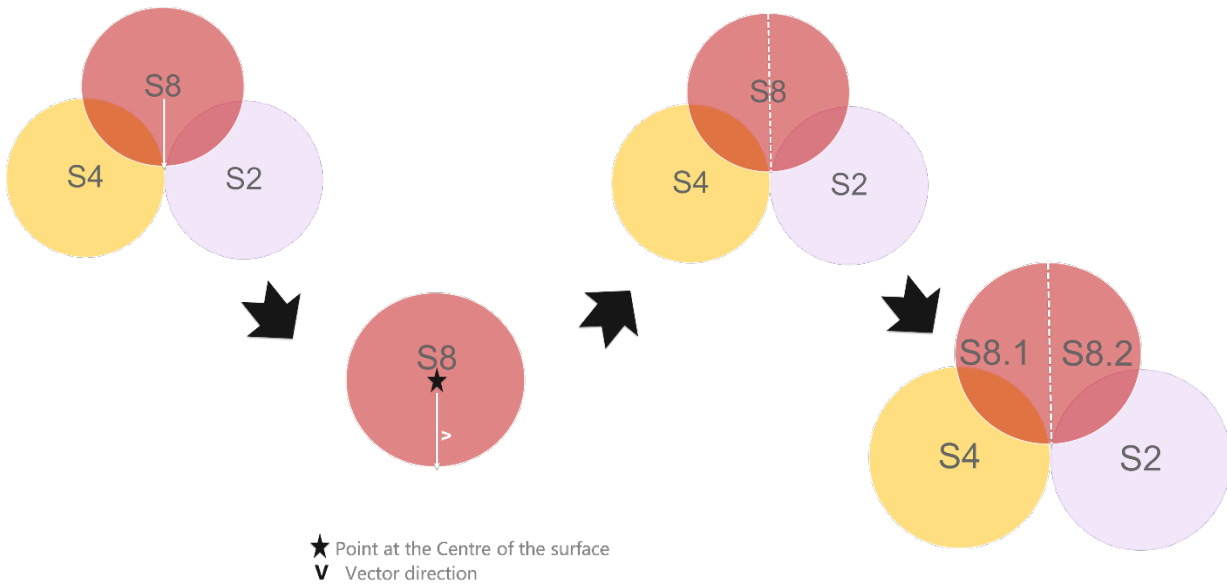


Figure 25: Splitting Surface with Dual Parent Relationships

Form Tributary Regions

The final step under Section II is to create the Drainage areas as shown in Figure 26 and give them ID's.

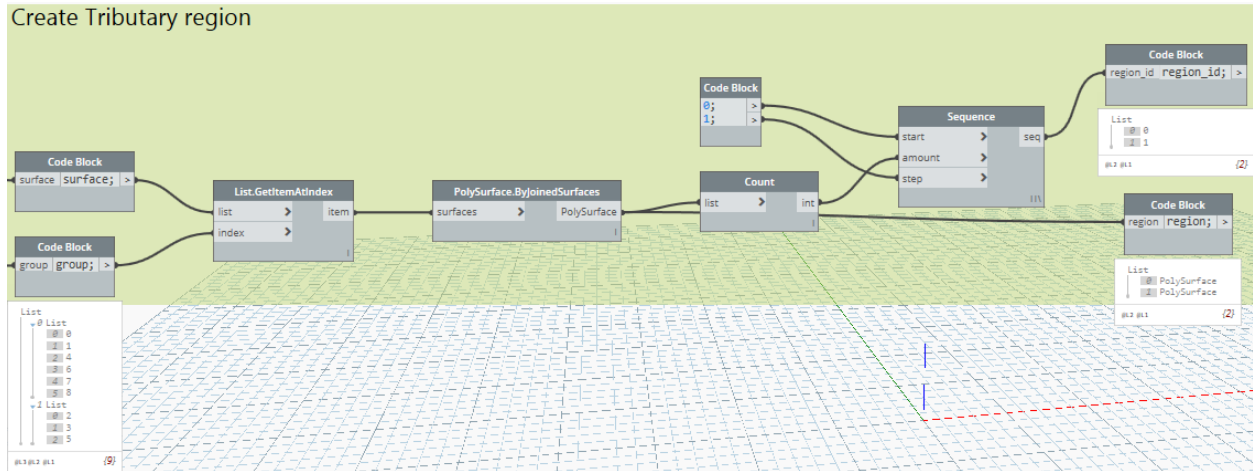


Figure 26: Creating Drainage Areas

Create Overflow Groups

In Section III, there are three main sub steps:

- A. Find neighboring drainage areas
- B. Find overflow elevation
- C. Create overflow groups

Find Neighboring Regions

We will use an example to illustrate this logic (refer to Figure 27 – 30). For Drain Region 0 in the figures below, drain regions 1, 2, 3 and 4 are the neighboring regions. Figure 29 shows the dynamo logic for this part. But, Drain Region 2 is not considered as it does not share an edge to allow the overflow. So, we include an object type check for the intersection geometry as shown in Figure 30.

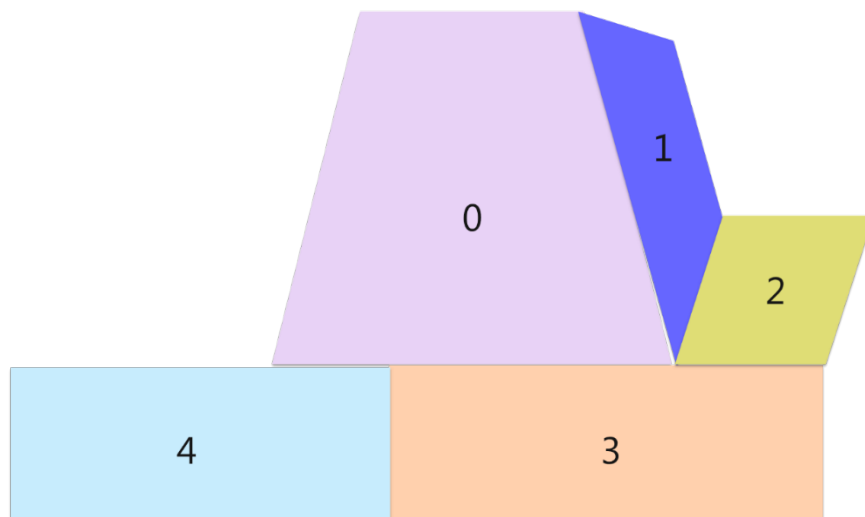


Figure 27: Drainage Areas

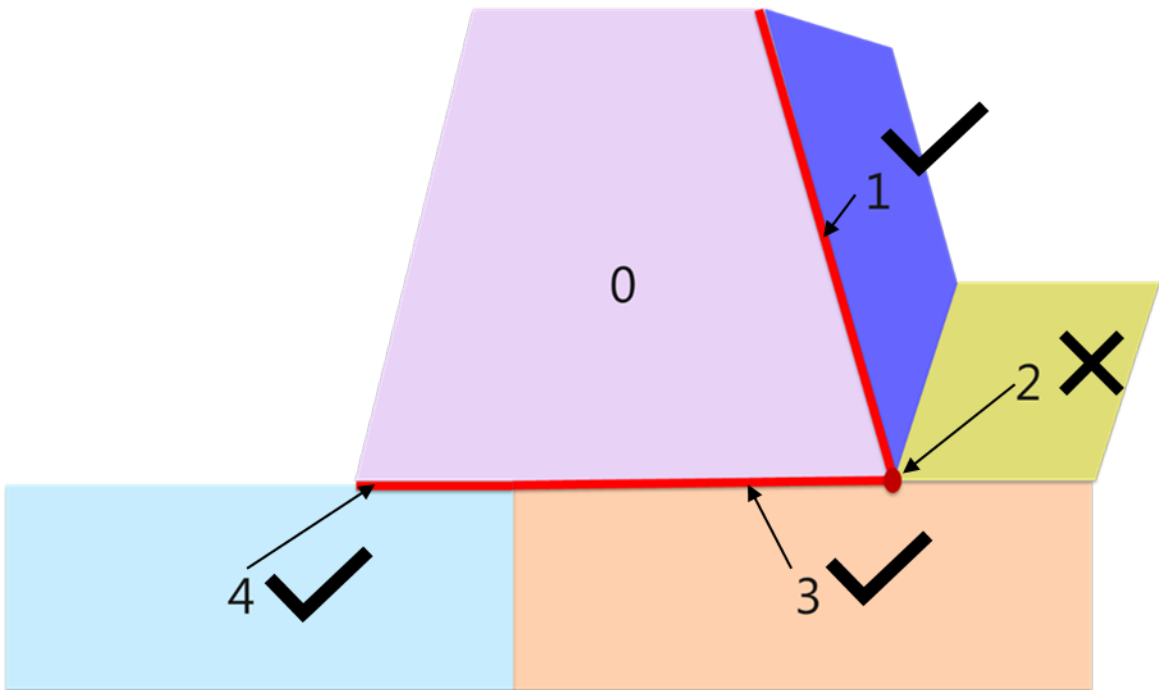


Figure 28: Drainage Area with its Overflow group comprising of the neighboring drainage areas with which it shares an edge

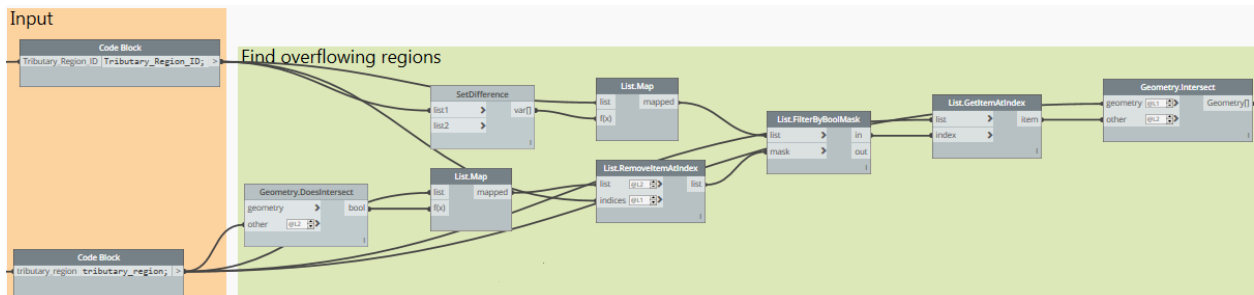


Figure 29: Finding Overflow Groups in Dynamo-Part 1

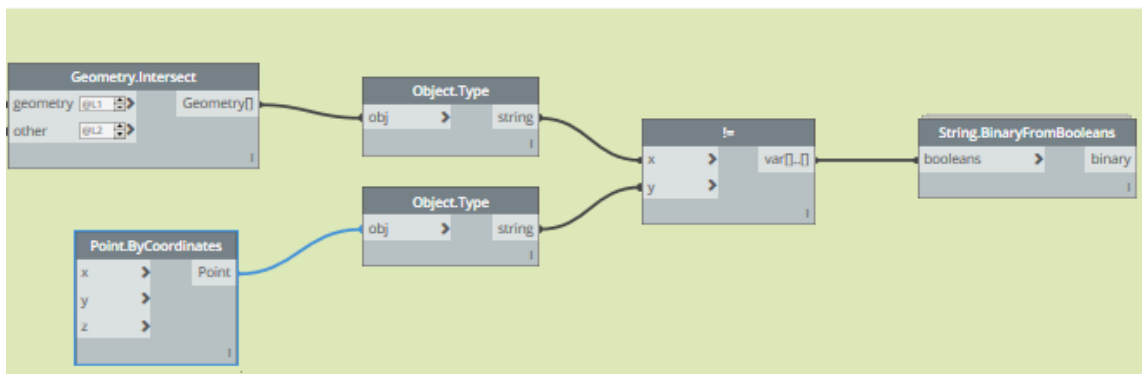


Figure 30: Finding Overflow Groups in Dynamo-Part 2

Determine Overflow Elevation

Once we determine the neighboring regions, we determine the elevation at which the water will overflow between the two drain regions by creating a bounding box around the intersection geometry and then taking the Point.Z value of the minimum point of the bounding box as shown in Figure 32.

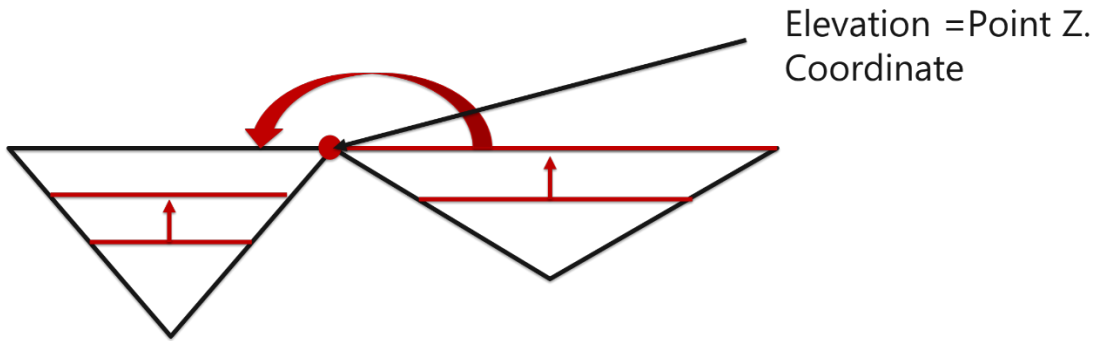


Figure 31: Overflow Elevation

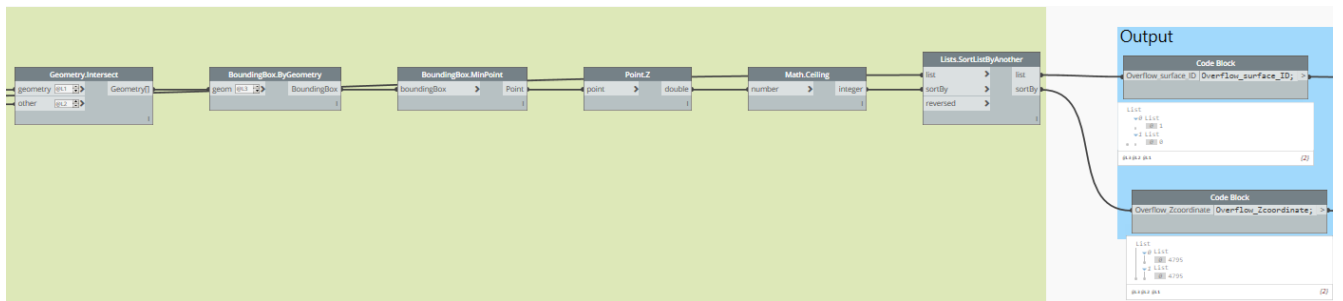


Figure 32: Finding Overflow Elevation in Dynamo

Calculate Projected Area and Volume

The projected area of each of the Drainage areas is calculated as shown in Figure 33 below. Then the volume of rain per drainage area is calculated by multiplying it with the rainfall in the region of the project.

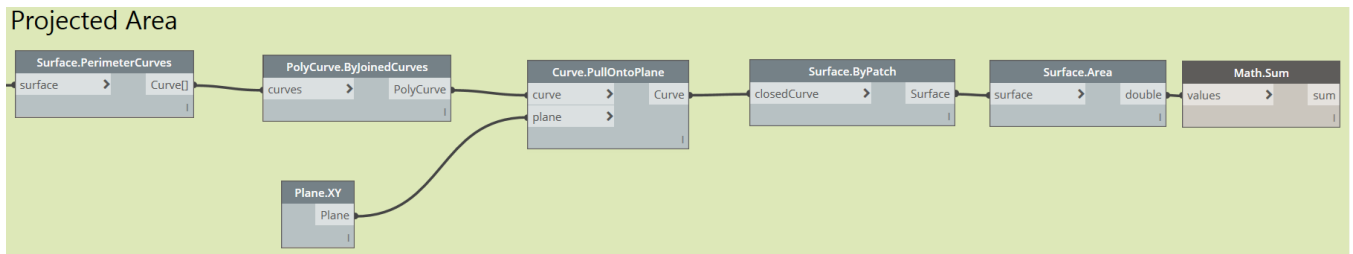


Figure 33: Calculating Projected Area in Dynamo

Find Rain Fill Volume

Now that we have all the different drainage regions, overflow groups, overflow elevation and the volume per drain. The next step is to do the iteration to fill up the drains until it meets the rain volume and to get the fill solid, fill elevation and excess overflow volume. In order to do this, we developed a custom python node to replicate the iterations.

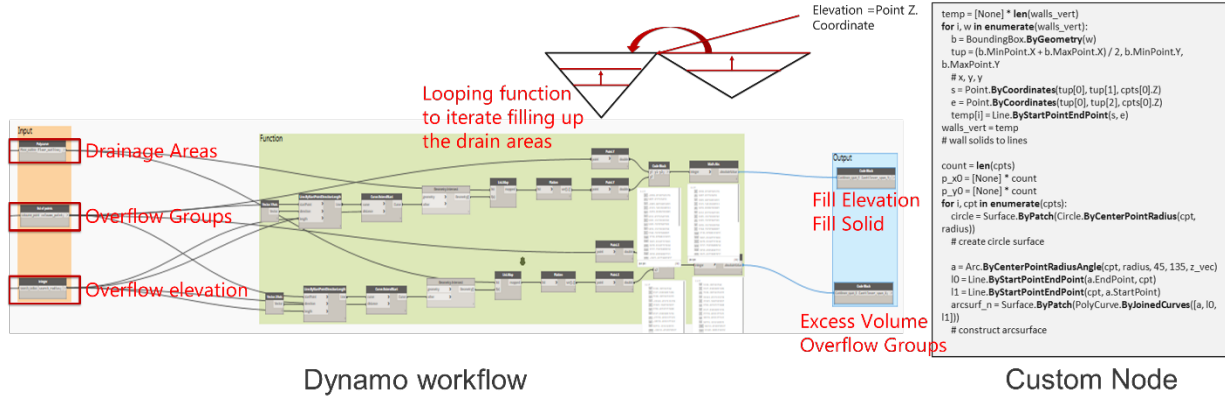


Figure 34: Rain Fill custom Python Node

Calculate Rain Load

The rain load, calculated using the formula given in Figure 35 and Figure 36, shows the same inside dynamo. Projected area is calculated using the same logic mentioned in the previous section (Figure 33). We find the top surfaces of the fill solid for this.

- Rain Filled solid → Projected Area of Top surface of fill = a
- Elevation of fill = E1
- Elevation of Drain = E2
- Excess Volume of Rain = V



$$H1 = E1 - E2$$

$$H2 = V/a$$

$$H = H1 + H2$$

$$\text{Rain Load} = H(\text{in mm}) * 0.00981 \text{ kpa}$$

Figure 35: Rain Load Calculation

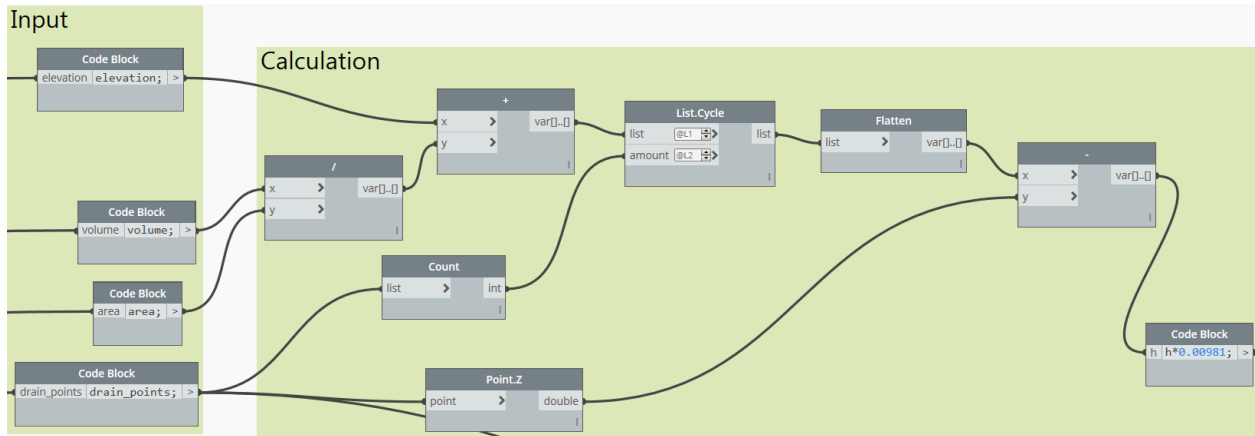


Figure 36: Rain Load Calculation Logic in Dynamo

Automate Rain load and Slope Annotation

The rain load that is calculated is converted to a string and then annotated as shown in Figure 37 below.

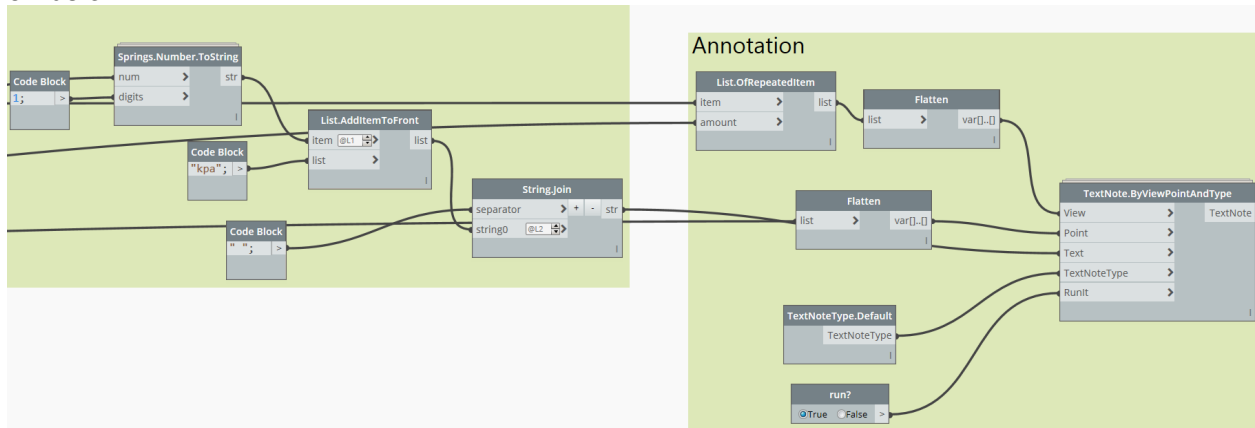


Figure 37: Rain Load Annotation using Dynamo

Setting up Dynamo Player

To set up this workflow for dynamo player, right click on the input nodes that you want the user to have control on and make sure “is Input” is enabled in the dropdown. Then open dynamo player and select the folder where the script is located to open the script. Then click on the edit panel icon to edit/run the script as shown in Figure 38.

Dynamo Player setup

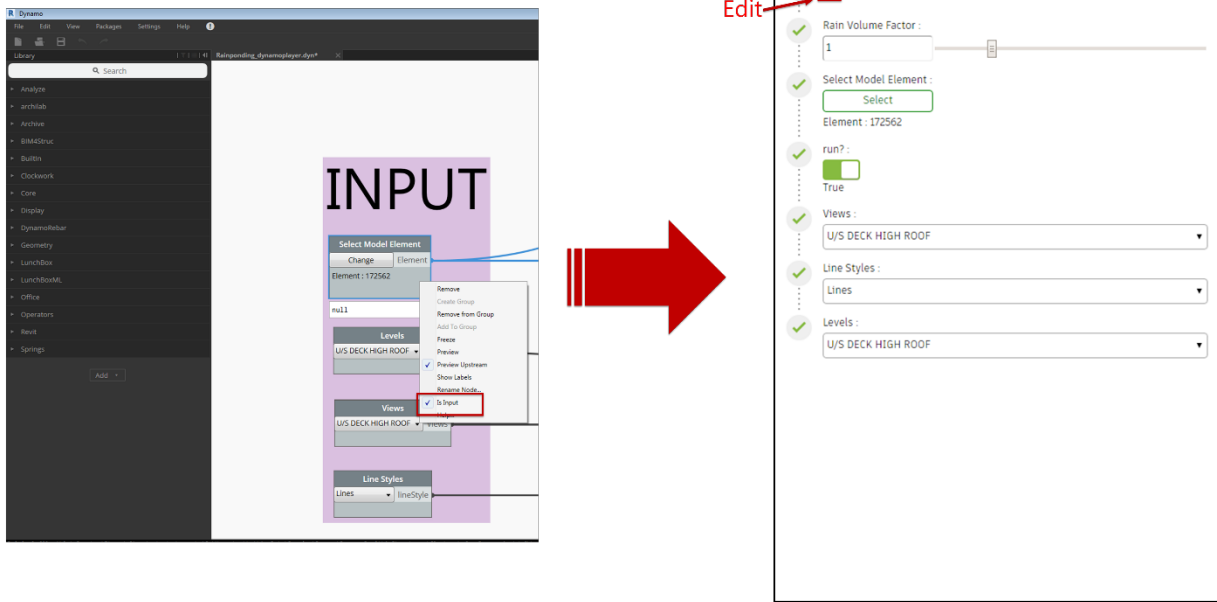


Figure 38: Dynamo Player Setup

Snow Load Calculation

In this section, we will explain the concepts of the logic used in the script to automate the calculation and annotation of snow load on roofs. First, we will briefly introduce the terminologies we use in this document and explain the current workflow of load calculation/annotation.

Terminologies

Snow load is the downward force on a building's roof by the weight of accumulated snow and ice. There are two snow loads we need to consider, one called 'uniform load' and another called 'drift load'. Uniform load is a value determined for a flat, wide open roof free of obstructions and protrusions. Drift load is an unbalanced snow loading resulting from wind transporting snow from one portion of the roof to another. Here, we call the roof we calculate the loads on as 'lower roof'.

Snow drifts often form on a lower roof in the wind shadow next to roof projections such as roof equipment and parapets (Figure 39, we will call this 'roof projection condition' in this document). In this case, the drift loads occur around the roof projections.

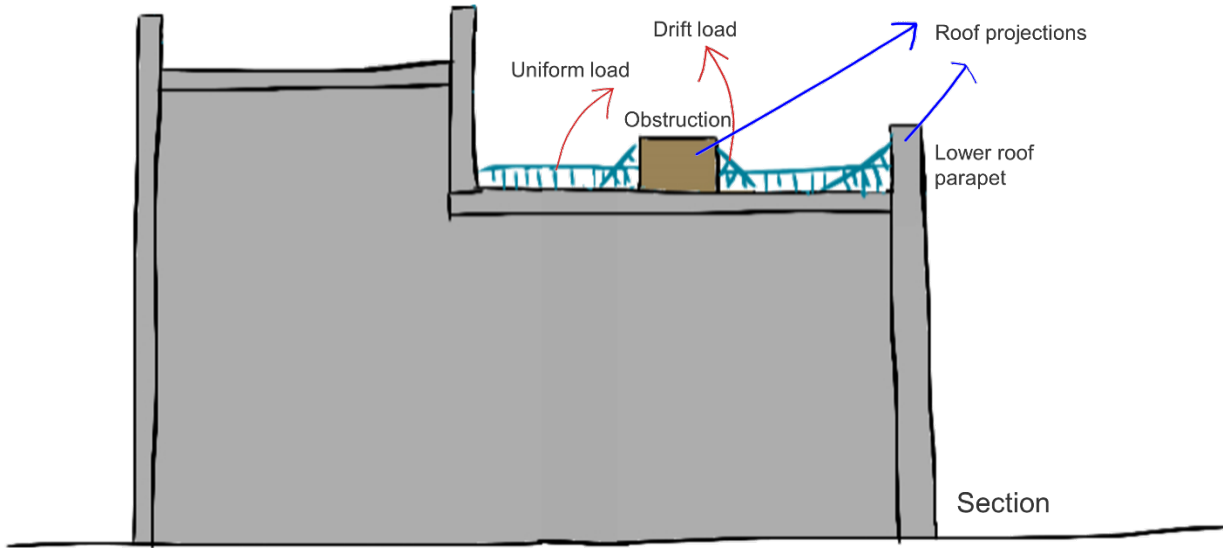


Figure 39: Snow loads on roof projections

Snow can also be blown up against the wind shadow of the higher portions of the building (upper roof parapet), which occurs when there is an adjacent roof (Figure 40, we will call this 'adjacent roof condition' in this document). This drift load is either determined by the drift from upper roof (Case 1) or lower roof (Case 2), whichever is conservative.

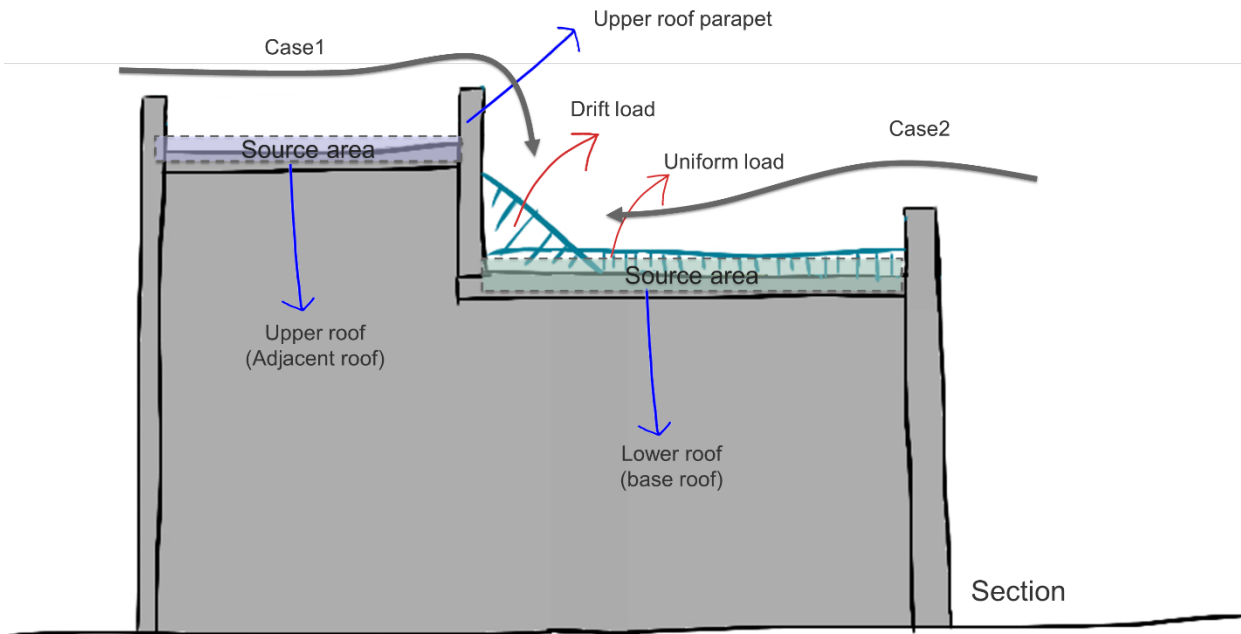


Figure 40: Snow loads on adjacent roofs

Current Workflow

Snow load is calculated roof-by-roof and each calculation process contains repetitive steps of getting dimensions of roof/projections, calculating loads using multiple formulas, and comparing the loads to choose conservative values.

- 1) First, the factors required in the code need to be retrieved. This includes wind exposure factor, importance factor, slope factor, etc.
- 2) Get width/length/height of the lower roof from the model.
- 3) Calculate uniform load.
- 4) For roof projection condition, repeat the following steps.
 - Get width/length/heights of roof projections from model
 - Calculate loads in spreadsheet
- 5) For adjacent roof condition, repeat the following steps (A-C).
 - A. Calculate Case 1
 - Get width/length/height of upper source area and parapets from model
 - Calculate loads in spreadsheet
 - B. Calculate Case 2
 - Get width/length/height of lower source area and parapets from model
 - Calculate loads in spreadsheet
 - C. Compare Case 1, 2 to get conservative values.
- 6) After calculating the above loads for each roof, the engineers need to make markups for technologists.
- 7) Then the technologists create snow load profile for each drift location and annotate the loads in the model. Lastly, the engineers backcheck the load plan.

Dynamo Workflow

As you can see above, the snow load calculation process includes repetitive steps including getting geometric information from the model, inputting those values to formulas, comparing values to get the final load. In this section, we will explain how we turned this process into the Dynamo script, which eventually reduced most of the manual steps.

Software

Revit 2018

Dynamo 1.3.2

Custom packages used: archilab, Clockwork, Springs, SteamNodes

Assumptions

There are a few assumptions we made developing this script. First, the calculation process is based on National Building Code 2015. Second, the roof slopes are smaller than 15 degrees. Additional nodes can be added to this script to consider bigger slopes, which we are planning to work on afterward. Third, all elements that need to be considered are modelled accurately in 3D. Here, we used a sample project model to demonstrate the script (Figure 41). To simplify the explanation, a part of the model is used for the demonstration.

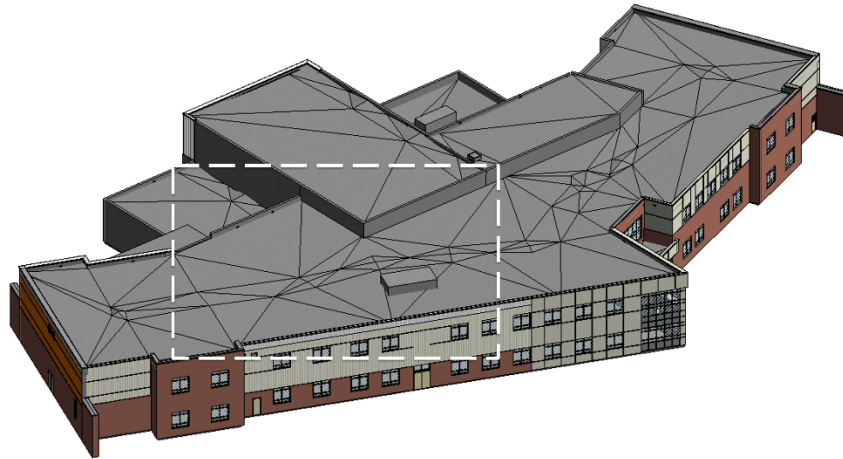


Figure 41: Sample project model

Overall Flow

We have divided the Dynamo script into four main sections/functions as shown in Figure 42. In Section I, we explain how we select and filter the elements that need to be considered for each condition (for adjacent roof and roof projections) of the load calculation. In Section II, we demonstrate how the scripts extract the drift surfaces and source areas for snow drift. Section III will retrieve the plan dimensions of the objects that need to be put in load calculation formulas. Section IV shows how these values are being input into the formulas using Python. Lastly, Section V describes how the snow load profiles and load annotations are created in the model.

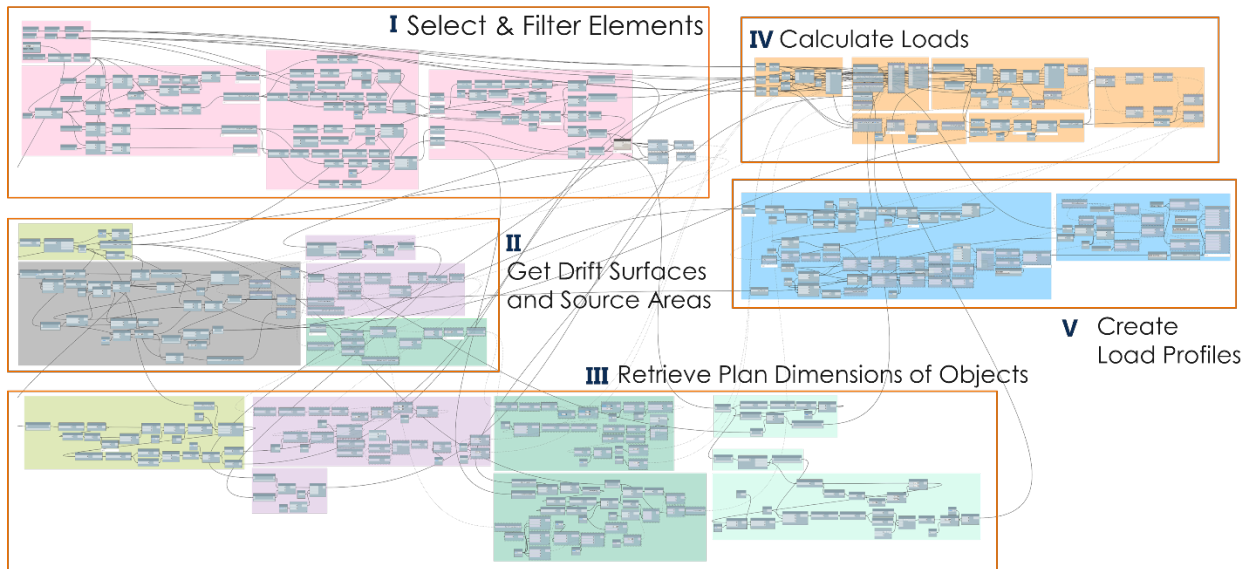


Figure 42: Snow load script

I. Select & Filter Elements

Input Factors

Both uniform load and drift load use the same formula shown in Figure 43. According to the formula, we need seven values to input to get the snow load value. Among those, five can be input by the users before running the script. Most of the values depend on

the location of the building, so the values are consistent throughout the same project. The following are the values used in the sample project, assuming the building is in Edmonton, AB, Canada. The γ at the end of the list, is the specific weight of snow, which we will need later for another calculation.

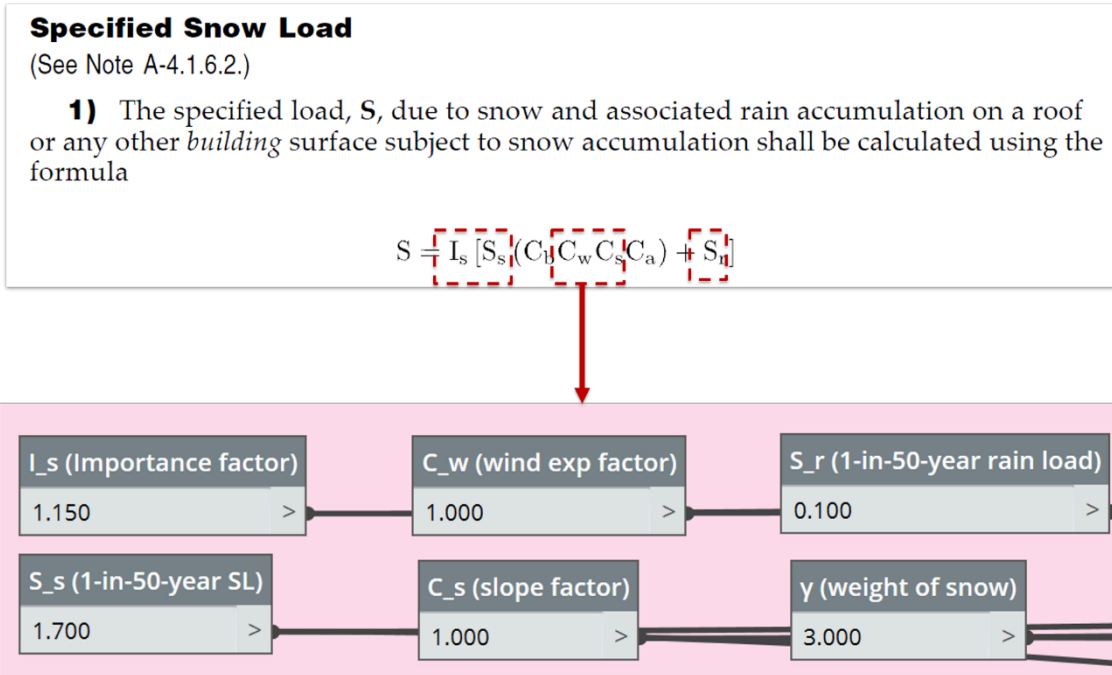


Figure 43: Snow load calculation formula

- J. I_s (Importance factor) = 1.0 when the building is in 'Normal' Importance Category
- K. S_s (1-in-50-year ground snow load) = 1.7 kPa if the building is in Edmonton
- L. C_w (Wind exposure factor) = 1.0 if the building is in Edmonton
- M. C_s (slope factor) = 1.0 when the roof slope < 15°
- N. S_r (1-in-50-year associated rain load) = 1.0 kPa if the building is in Edmonton
- O. γ (Specific weight of snow) = 3.0 kN/m³ if the building is in Edmonton

Assign Name to Elements

Before we select the elements, we need to assign names to each element in their 'Comments' parameters (Figure 44). Users can decide how they will name the elements, but here we named as below.

- 1) Lower roof parapets: 'LP' (No need to number them)
- 2) Upper roof parapets: 'UP1', 'UP2', 'UP3', ... (Number them)
- 3) Upper roofs: 'R-UP1, UP2', 'R-UP3' (To include the name of roof parapets they are attached to)
- 4) Lower roof obstructions: 'OB' (No need to number them)

Element properties

Identity Data	
Image	
Comments	R - UP1,UP2
Mark	
Workset	BE_ENVELOPE
Edited by	Soojung.Kim@stantec.com

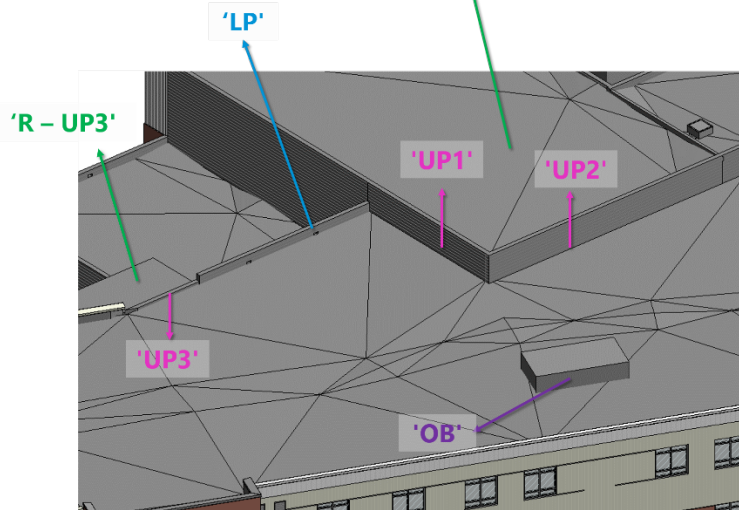


Figure 44: Assign element names

Select Elements

Now we can select the lower roof, which is the base element we want are looking into (Figure 45). As mentioned before, we refer to this base roof as 'lower roof' to distinguish it from adjacent roofs, which will be refer to 'upper roofs'. The geometric information of this roof will be used to calculate both uniform load and drift loads.

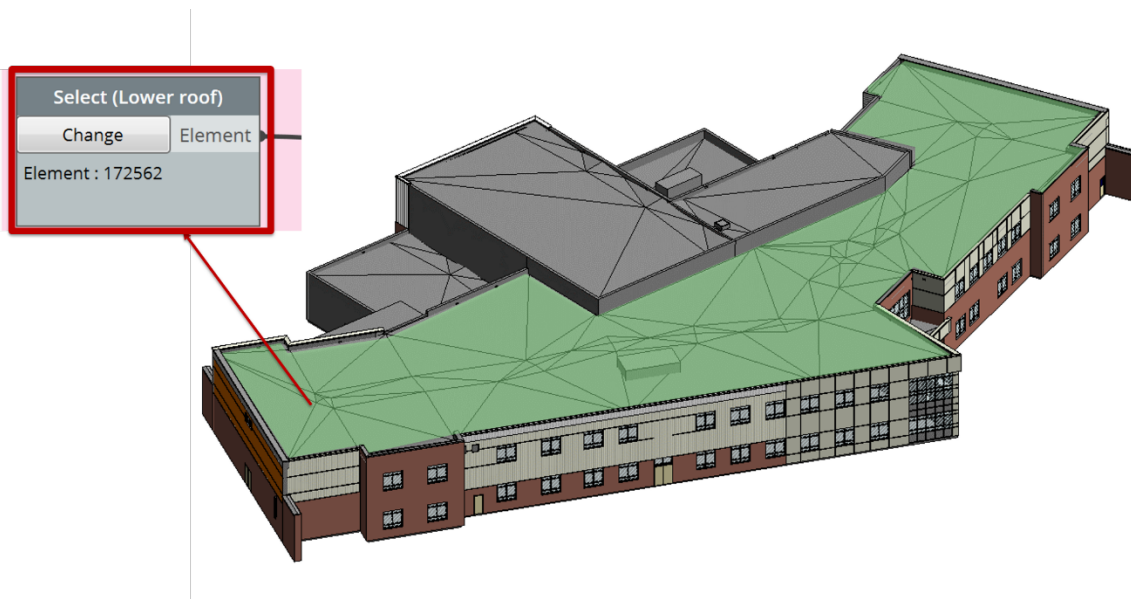


Figure 45: Select element to be considered

Next, we need to select the elements we need for drift loads. This would include the upper roofs and upper roof parapets for adjacent roof conditions, and lower roof parapets and obstructions for roof projection conditions. After we select the elements, the script starts to run.

Filer with Names

The script filters out the elements into four categories by using 'List.FilterByString' node shown in Figure 46. The reason why we name the upper roofs to include the parapet names is to group and sort each list in alphabetic order to match each other.

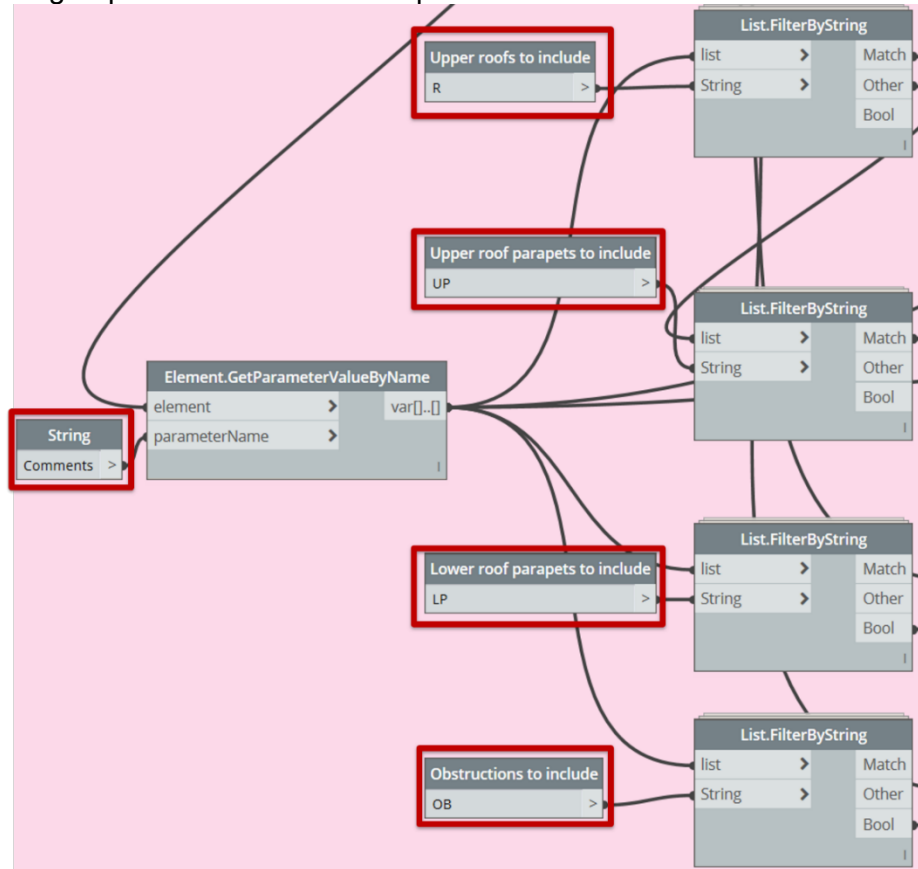
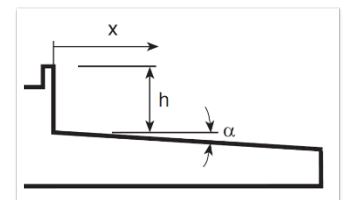


Figure 46: Sort elements by name

Filter Out Negligible Upper Roof Parapets & Roofs

According to the codes, we can ignore the drift load occurring on adjacent roof locations if the height of the upper roof parapet is smaller than $0.8 \cdot S_s / \gamma$.



(1) If $a > 5 \text{ m}$ or $h \leq 0.8 S_s / \gamma$, drifting from the higher roof need not be considered.

Figure 47: Negligible upper roof parapets (NBC2015)

To get the h value of the upper roof parapets, we use 'Geometry.BoundingBox' to get the bounding boxes of the parapets and get the maximum z value of the bounding boxes using 'BoundingBox.MaxPoint'. Then we subtract the lower roof height from h values (we

will explain the process of obtaining lower roof height in the next section) and get the indices of parapets with $h < 0.8 * S_s / \gamma$. Using these indices, we filter out the upper roofs and upper roof parapets to get the final lists.

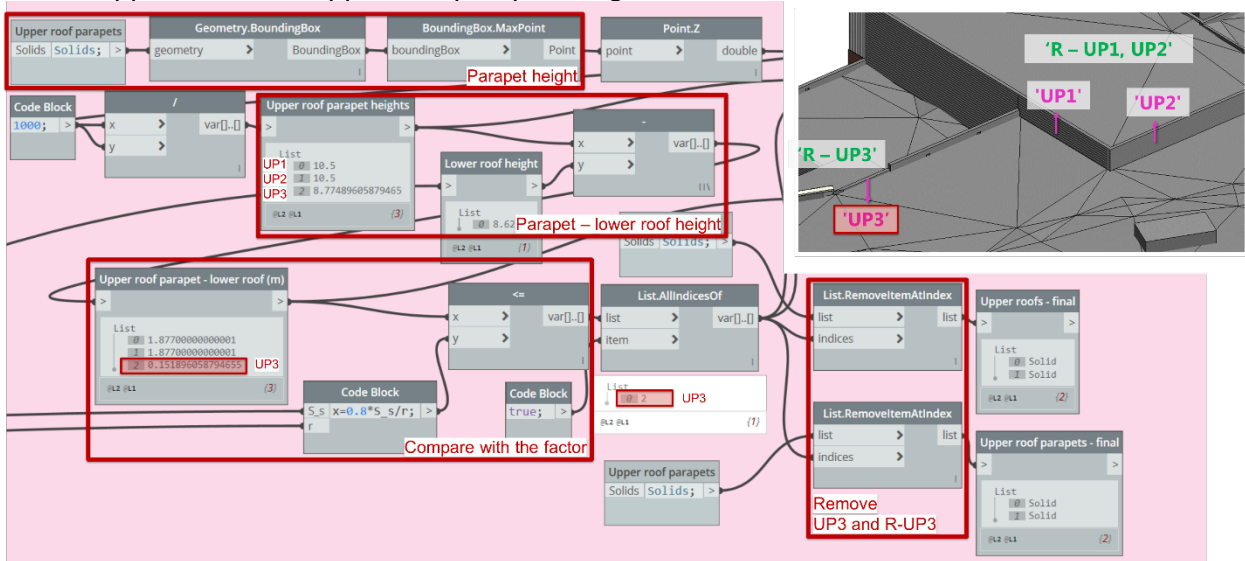


Figure 48: Filter out upper roofs and parapets with negligible heights

II. Get Drift Surfaces and Source Areas

Get Drift Surfaces (+Drift Base Lines)

To get the information we need for load calculation, we need to extract drift surfaces and base lines first. To get the drift surfaces, the faces of the drift objects (upper roof parapets and roof projections) are filtered in multiple steps. First, using the 'Surface.NormalAtParameter', the horizontal surfaces of the drift objects are filtered out (Figure 49).

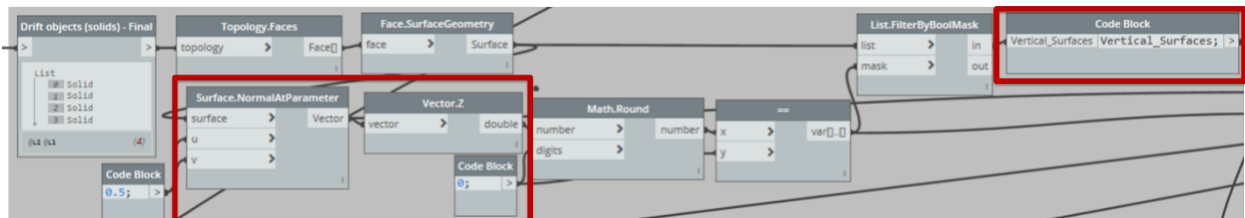
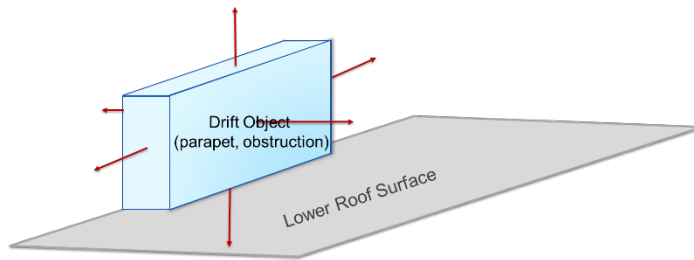


Figure 49: Filter surfaces/base lines of drift objects - 1

The remaining vertical surfaces are then offset vertically to see if the offset surfaces intersect with lower roof surface. If the offset surfaces don't intersect with the lower roof, they are filtered out. Additionally, if the surfaces intersect with the lower roof, but in a

negligible length (here we set to 1m), we ignore those surfaces for the drift loads. The `Geometry.Translate` node is used to create the offset surfaces and `Geometry.DoesIntersect` and `Geometry.Intersect` nodes are used to filter the surfaces.

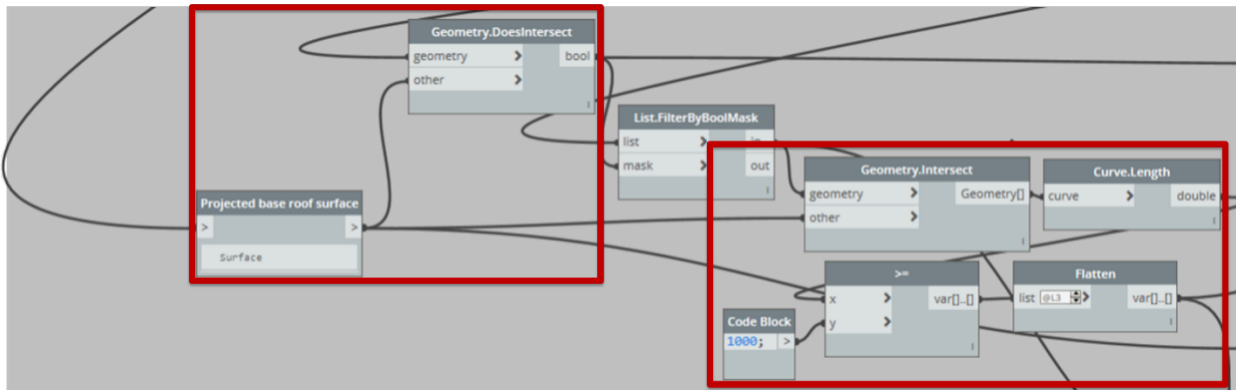
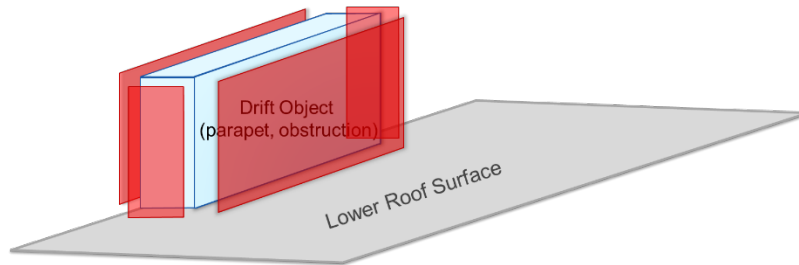


Figure 50: Filter surfaces/base lines of drift objects – 2

This finally leaves out the drift surface we need to consider for drift load as shown in Figure 51. Then we can get the base line of the drift by extracting intersecting lines of the drift surfaces and the lower roof. The final two lists in the image consist of the drift surfaces and drift base lines of four drift objects. Note that the fourth object has four drift surfaces and base lines since the obstruction above the lower roof can have drift loads on its four vertical faces.

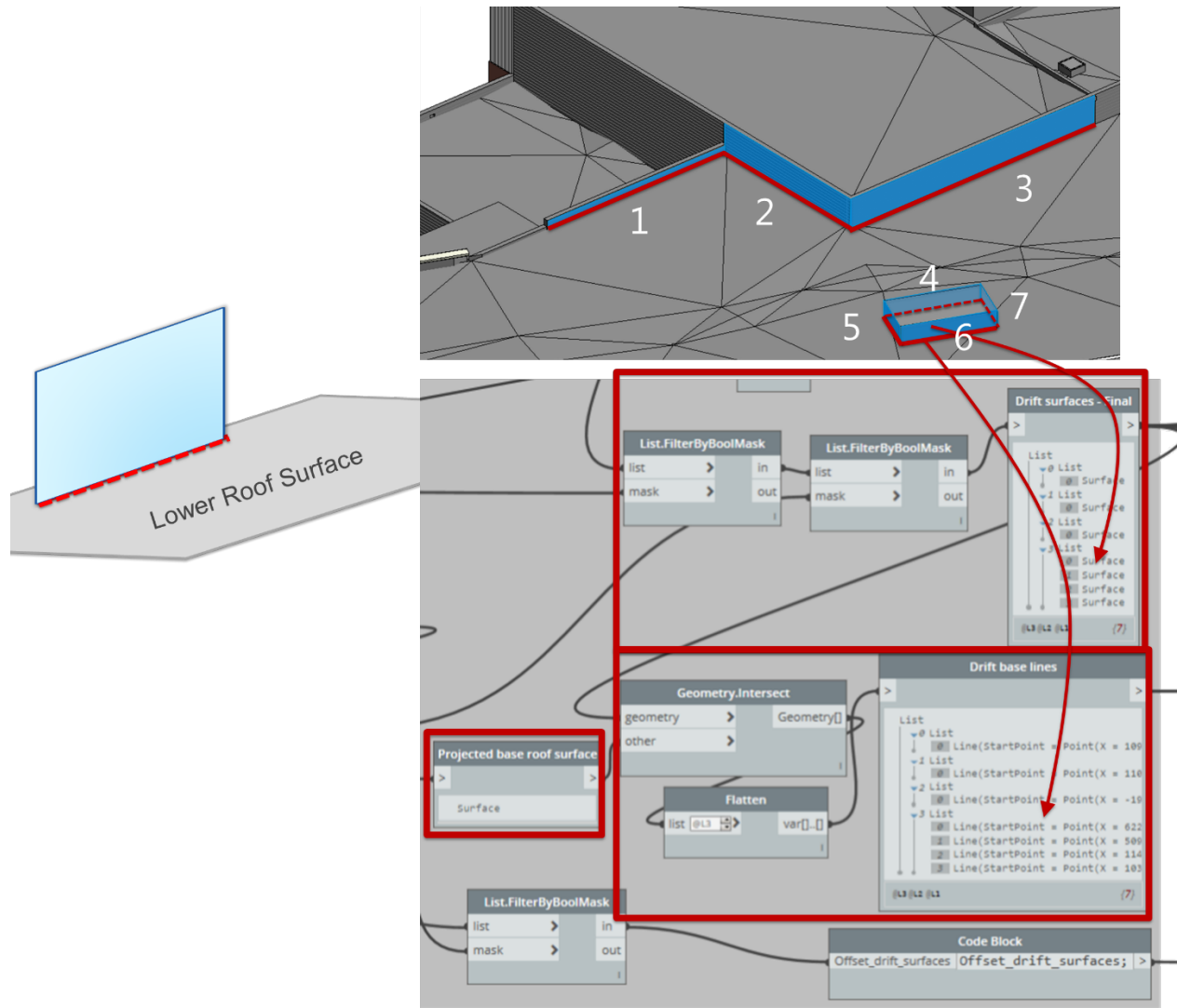


Figure 51: Filter surfaces/base lines of drift objects – 3

Determine Source Areas

According to the NBC code, if there is an adjacent roof that contributes to the drift load on the upper roof parapet, we need to consider both drift from upper roof (Case 1) and lower roof (Case 2) and get the conservative value. Therefore, we first need to determine the source areas of each case and get the geometric information of the source areas. In Figure 52, you'll notice that the source areas do not necessarily include the entire upper or lower roof area, but only the part that contribute to the drift. It is clearer if you see Figure 53, where it shows the entire lower roof area in red dotted line and the source area of Case 2 with green shade. In this sample model, the source area of Case 1 is the same as the entire upper roof surface in this image since the entire roof is at the other side of the drift surface.

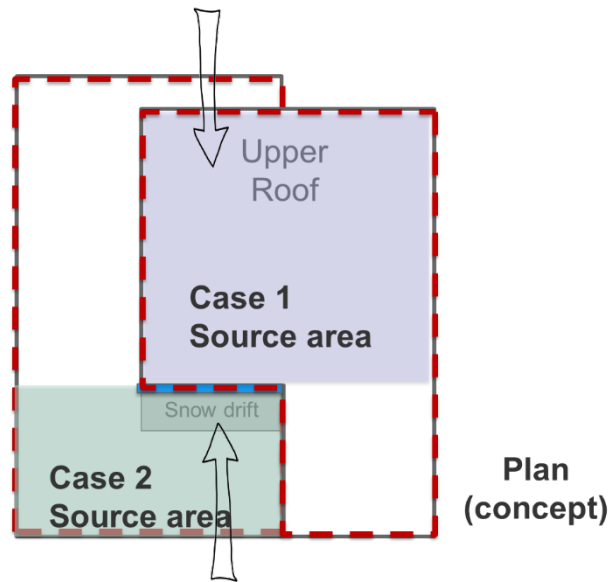
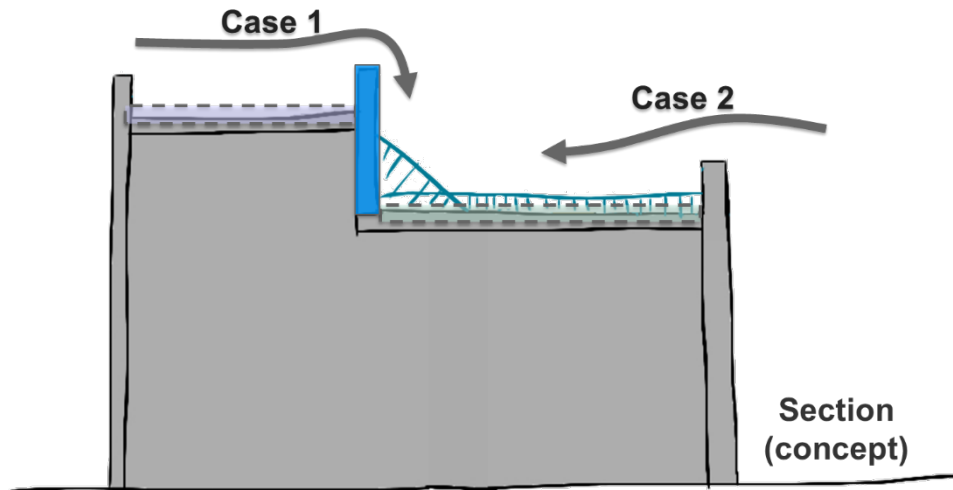


Figure 52: Source areas of adjacent roof drift - 1

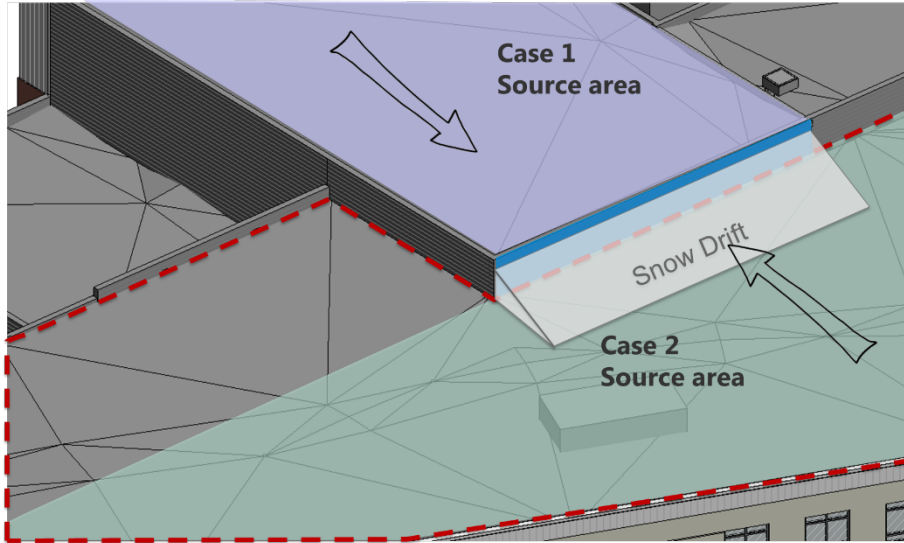


Figure 53: Source areas of adjacent roof drift - 2

We will explain the logic of extracting the source area taking Case 2 as an example. To get the source areas we want, first we create a base plane using the drift surface and split the lower roof surface with the plane (Figure 54). This will create surface 1 and surface 2 (Figure 55). Then we use the offset drift surface we got in the previous section (Figure 50) to filter the split surface that intersects with the offset drift surface (Figure 56). You might have realized that when using this same logic to get the source area of Case 1, we need to get the 'out' output of '*List.FilterByBoolmask*' at the end since the source area of Case 1 should not intersect with the offset drift.

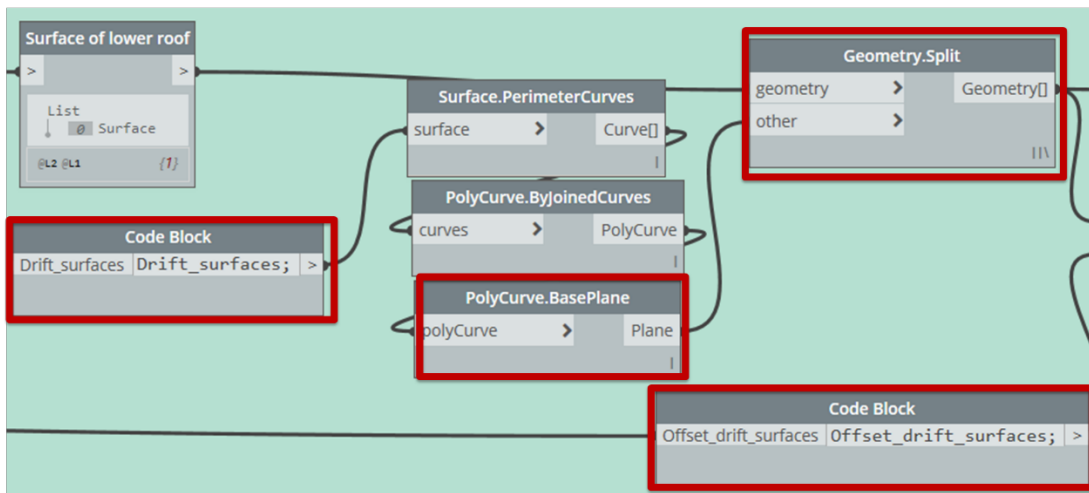
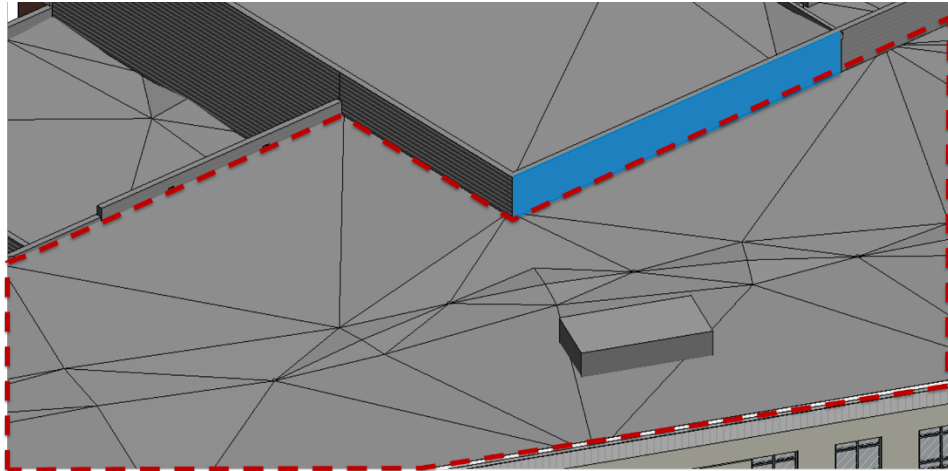


Figure 54: Split lower roof surface with drift surface base plane

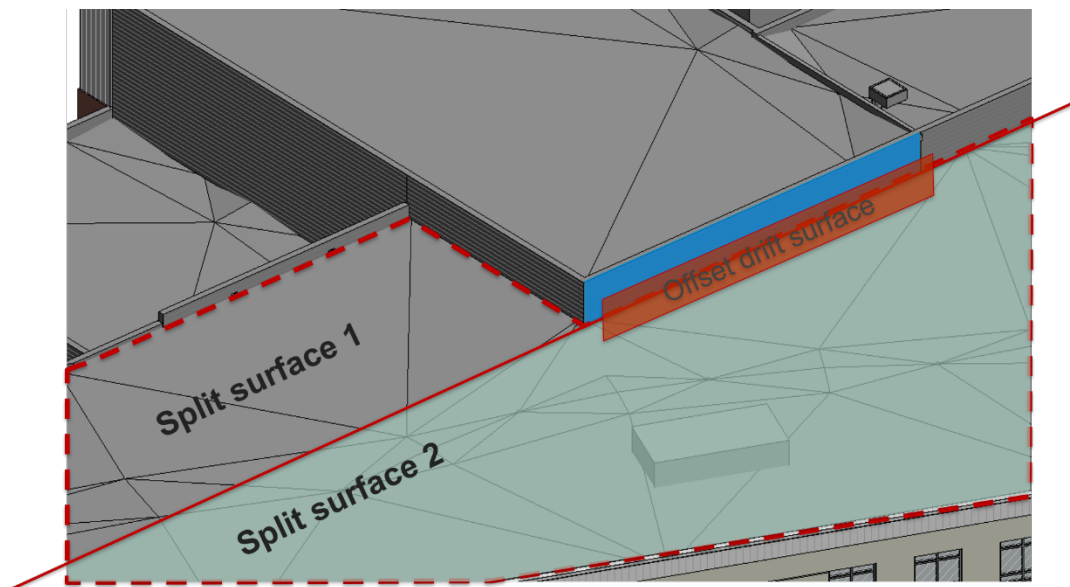


Figure 55: Getting source area for Case 2

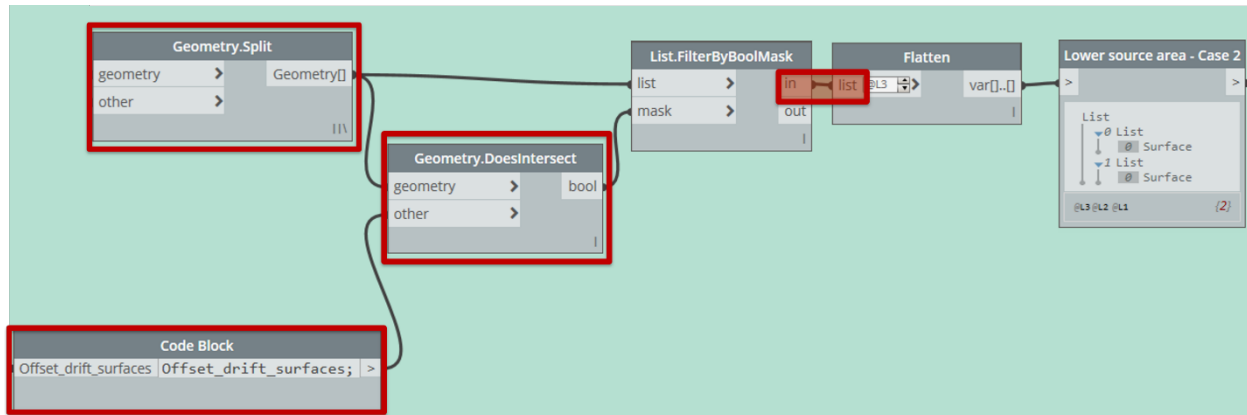
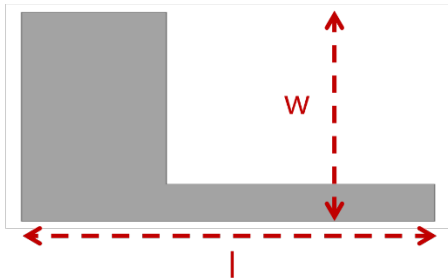


Figure 56: Filter surface that contributes to drift

III. Retrieve Plan Dimensions of Objects

According to the code, we need the plan dimensions of all related objects (roofs, projections, and source areas) to calculate the snow loads (Figure 52).



Lower Roof

l_c = characteristic length of the upper or lower roof, defined as $2w - w^2/l$, in m,
 w = smaller plan dimension of the roof, in m, and
 l = larger plan dimension of the roof, in m,

Roof Projections

x_d shall be taken as the lesser of $3.35h$ and $(2/3)l_0$, where
 h = height of the projection, and
 l_0 = longest horizontal dimension of the projection.

Source Areas

l_{cs} = characteristic length of the source area for drifting, defined as $l_{cs} = 2w_s - \frac{w_s^2}{l_s}$, where w_s and l_s are respectively the shorter and longer dimensions of the relevant source areas for snow drifting shown in Figure 4.1.6.5.-B for Cases

Figure 57: Dimensions needed for load calculation

Using the bounding box node, we can easily get the longest and shortest dimensions of objects. However, one thing to remember about the bounding box node is that it creates the bounding box aligned to the x axis, not the longest dimension of the object. So, if the object is not aligned to the x axis, like A below, we end up getting different dimensions from what we want. Therefore, we need to rotate the objects parallel to the x axis to get the correct dimensions we want (refer to B below).

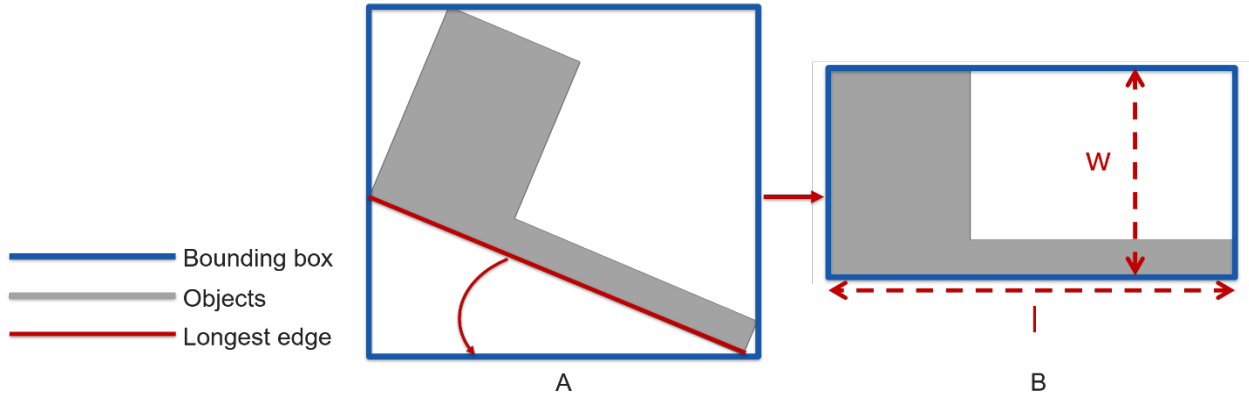


Figure 58: Rotate objects to get correct bounding box

First, we get the perimeter curves of the projected top surfaces (refer to Rain load section about getting projected top surfaces). Then we sort the curves by their lengths using 'Curve.Length' (Figure 59). Get the longest curve and get its relative angle to X axis by using 'Curve.TangentAtParameter' and 'Vector.AngleWithVector'. Rotate the object with the angle on XY plane (Figure 60). Lastly, use 'BoundingBox.ToCuboid' to get the length and width of the cuboid and sort them out by length to get the 'smaller dimension' and 'larger dimension' (Figure 61).

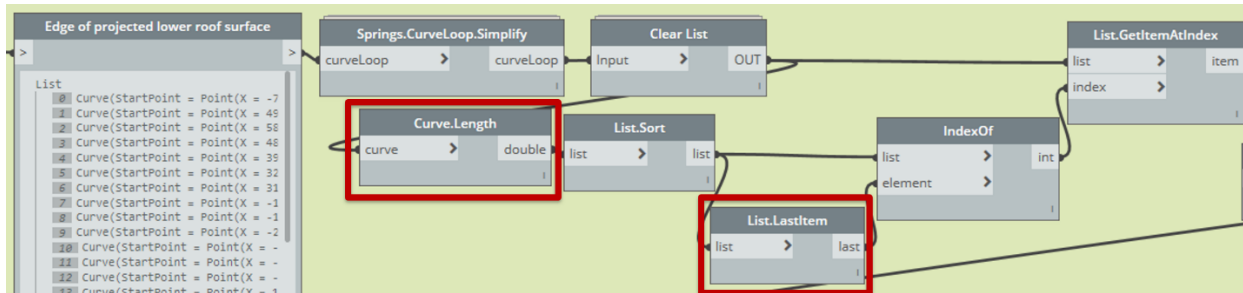


Figure 59: Get longest perimeter curve

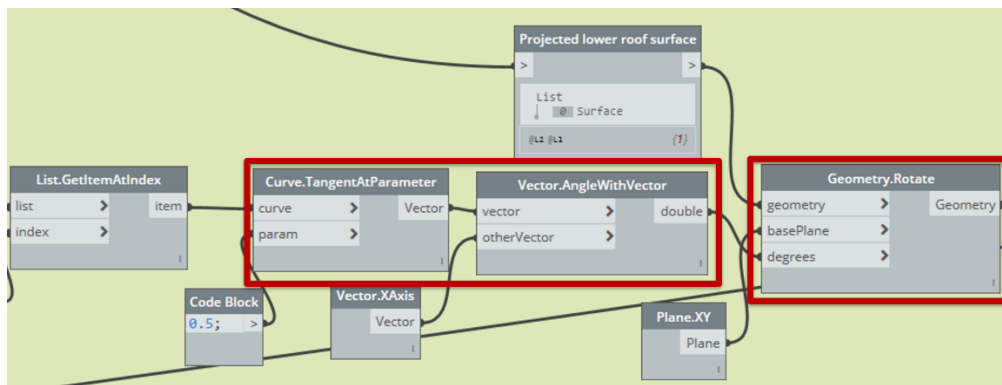


Figure 60: Rotate object to X axis

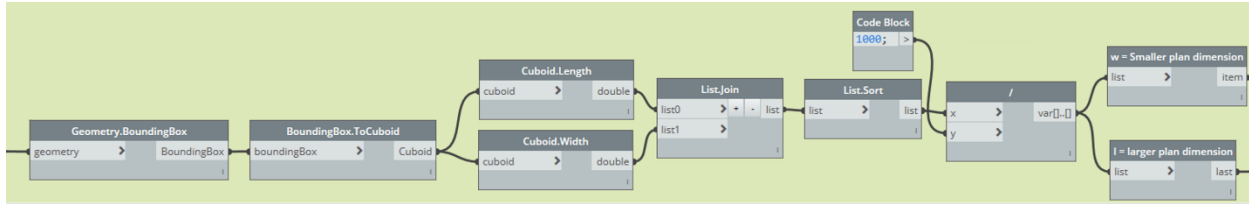


Figure 61: Get smaller and larger dimension of object

IV. Calculate Loads

Now we can use the geometric data we extracted from the model to calculate the snow loads. To create the snow load profile, we need to know the uniform load, maximum drift load, and drift length for each drift location (Figure 62). Since there are multiple sub-lists to handle and multiple factors to be input to the formula, here we use a Python node to calculate both uniform load and drift loads.

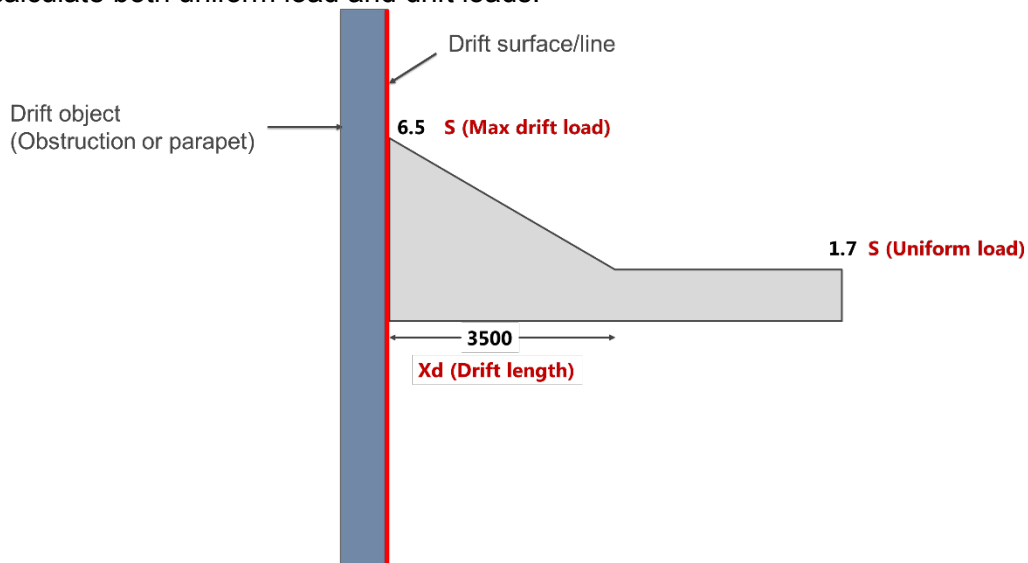


Figure 62: Snow load profile

Calculate Uniform Load

The uniform load is consistent for the same roof, so we calculate the uniform load first. As it was mentioned earlier, additional to the factors we manually input, we need C_b (basic load factor) and C_a (accumulation factor) to calculate the loads we want.

$$S = I_s [S_s (C_b C_w C_s C_a) + S_r]$$

Figure 63: Snow load calculation formula

According to the codes, C_b is determined as below. Here l_c is the characteristic length of the objects that can be calculated with the smaller and larger dimensions of those objects, which we extracted in the previous section.

i)

$$C_b = 0.8 \text{ for } l_c \leq \left(\frac{70}{C_w^2} \right), \text{ and}$$

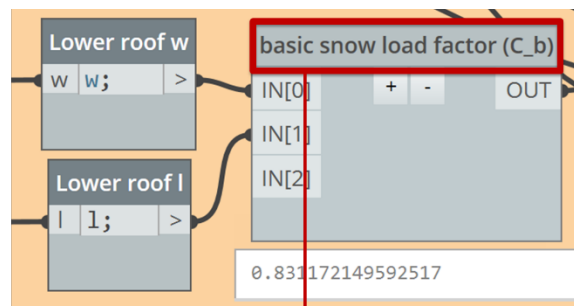
ii)

$$C_b = \frac{1}{C_w} \left[1 - (1 - 0.8C_w) \exp \left(-\frac{l_c C_w^2 - 70}{100} \right) \right] \text{ for } l_c > \left(\frac{70}{C_w^2} \right)$$

l_c = characteristic length of the upper or lower roof, defined as $2w-w^2/l$, in m,
 w = smaller plan dimension of the roof, in m, and
 l = larger plan dimension of the roof, in m,

Figure 64: Basic snow load factor

By inputting the values, we use 'if/else' logic to get the C_b for the lower roof as Figure 65 shows. With this C_b output and C_a (1.0 according to the codes), we can calculate the uniform load as Figure 66 indicates.



```

7 w=IN[0]
8 l=IN[1]
9 C_w=IN[2]
10
11 l_c=2*w-(w**2/l)
12 if l_c<=70/C_w**2:
13     C_b=0.8
14 else:
15     C_b=(1/C_w)*(1-(1-0.8*C_w)*math.exp(-(l_c*C_w**2-70)/100))
16
17 #Assign your output to the OUT variable.
18 OUT = C_b

```

Figure 65: Python script to calculate C_b

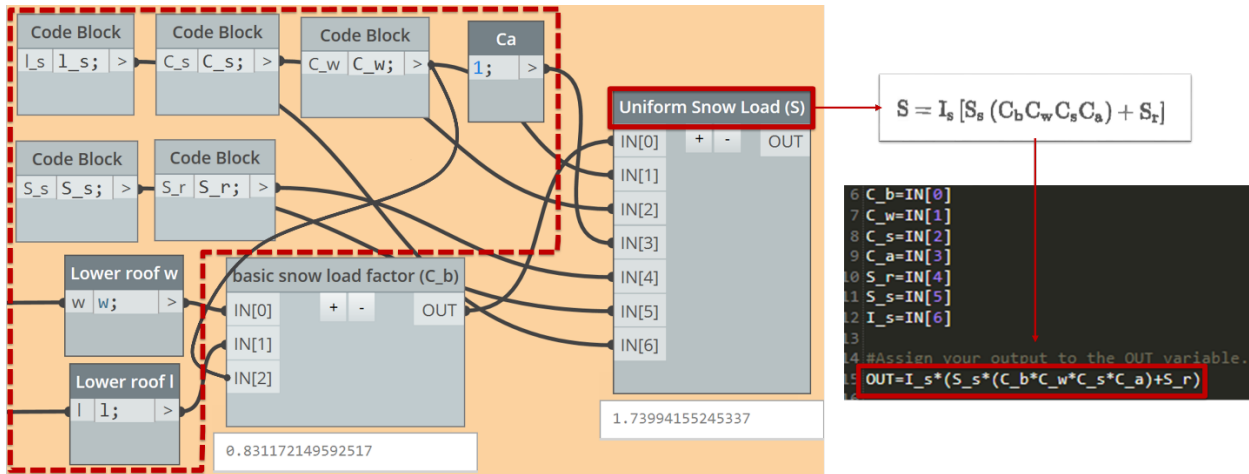


Figure 66: Python script to calculate uniform load

Calculate Maximum Drift Load (Roof Projection Locations)

For roof projections, as Figure 67 shows, Ca_0 is calculated with 'Minimum' function in the Python node. Then maximum drift load can be calculated using this Ca_0 .

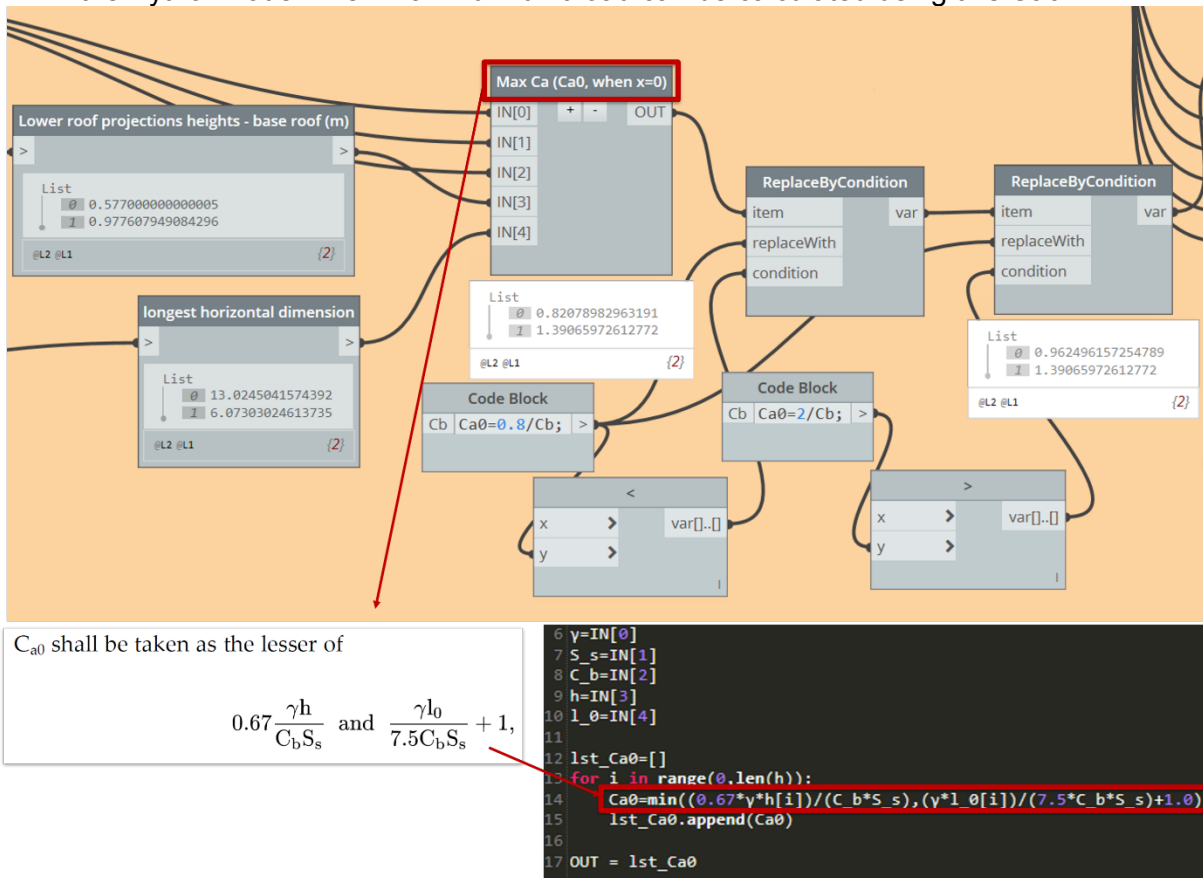


Figure 67: Calculate Ca_0 for Roof Projection locations

Calculate Maximum Drift Load (Adjacent roof Locations)

To calculate the maximum drift load, we need to first calculate the maximum accumulation factor Ca_0 . According to the code, Ca_0 is determined using the following logic:

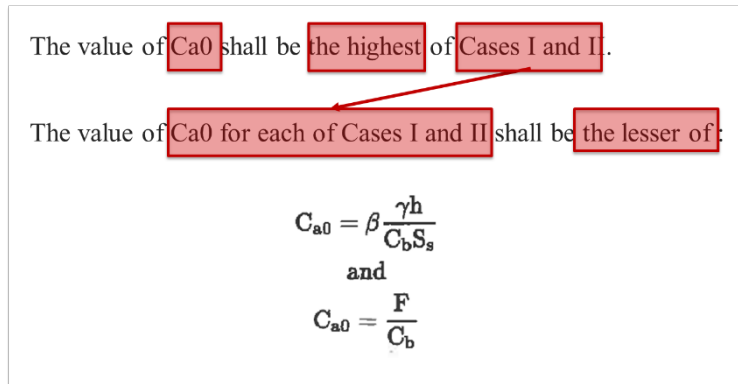


Figure 68: Maximum accumulation factor (C_{a0}) formula for adjacent roof

The manual input values and geometric data we extracted are set as the inputs. Since we are calculating C_{a0} for Case 1 and Case 2 of the same drift location and getting the greater value, the inputs are grouped as values for Case 1 and values for Case 2 ('lst_drifts_case 1' and 'lst_drifts_case2' in Figure 69).

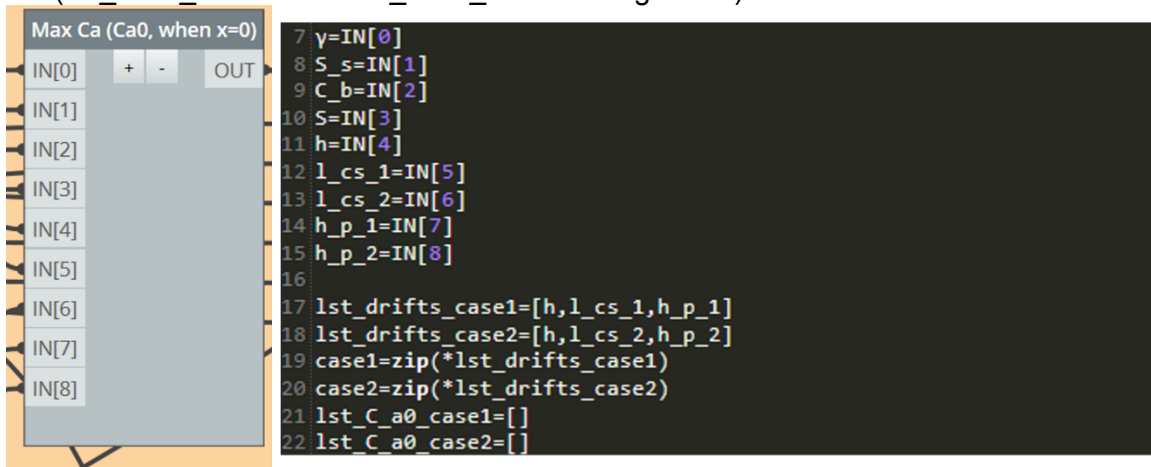


Figure 69: Grouping inputs for C_{a0} calculation

Then for each drift location, (i) the C_{a0} of Case 1 and (ii) the C_{a0} of Case 2 are calculated using minimum function. Then at last (iii) C_{a0} of Case 1 and Case 2 are compared to get the greater value using maximum function (Figure 70).

```

25 for drift in case1:
26     H_p=min(drift[2]-0.8*S/y,drift[1]/5.0)
27     if H_p<0: H_p=0.0
28     F1=min(5.0,0.35*1.0*math.sqrt(y*(drift[1]-5.0*H_p)/S_s)+C_b)
29     C_a0_1=min(1.0*y*drift[0]/(C_b*S_s),F1/C_b) (i)
30     lst_C_a0_case1.append(C_a0_1)
31 for drift in case2:
32     H_p=min(drift[2]-0.8*S/y,drift[1]/5.0)
33     if H_p<0: H_p=0.0
34     F2=min(5.0,0.35*0.67*math.sqrt(y*(drift[1]-5.0*H_p)/S_s)+C_b)
35     C_a0_2=min(0.67*y*drift[0]/(C_b*S_s),F2/C_b) (ii)
36     lst_C_a0_case2.append(C_a0_2)
37 lst_greater = [None] * len(lst_C_a0_case1)
38 for i in range(len(lst_C_a0_case1)):
39     lst_greater[i] = max(lst_C_a0_case1[i], lst_C_a0_case2[i]) (iii)
40 OUT =lst_greater

```

Figure 70: Calculating C_{a0} using Python

With the Ca0 values, the maximum drift load is calculated (Figure 71).

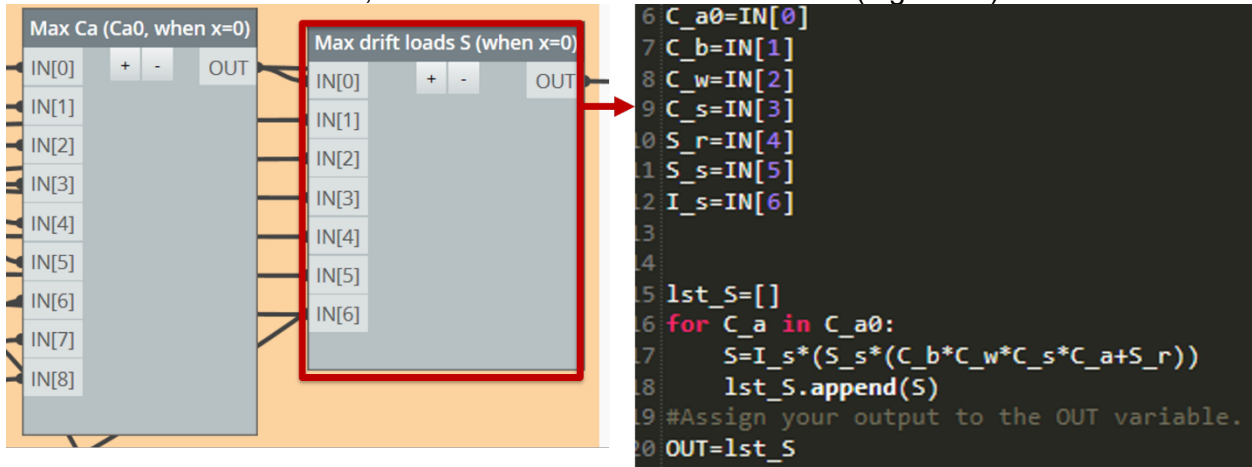
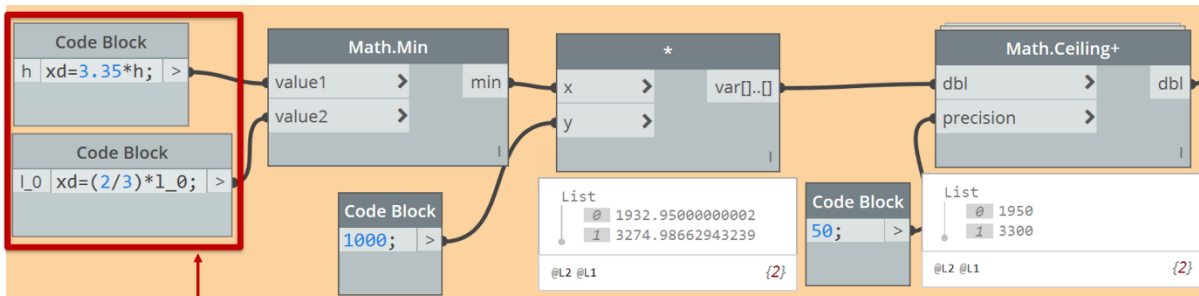


Figure 71: Calculating maximum drift load S

Calculate Drift Length (Roof Projection Locations)

The drift lengths for roof projections are simply calculated with *Math.Min* node and *Math.Ceiling+* node.



x_d shall be taken as the lesser of $3.35h$ and $(2/3)l_0$, where
 h = height of the projection, and
 l_0 = longest horizontal dimension of the projection.

Figure 72: Calculating drift length X_d

Calculate Drift Length (Adjacent roof Locations)

We can calculate the drift length by using a simple code node. Then, *Math.Ceiling+* is used to round up the lengths.

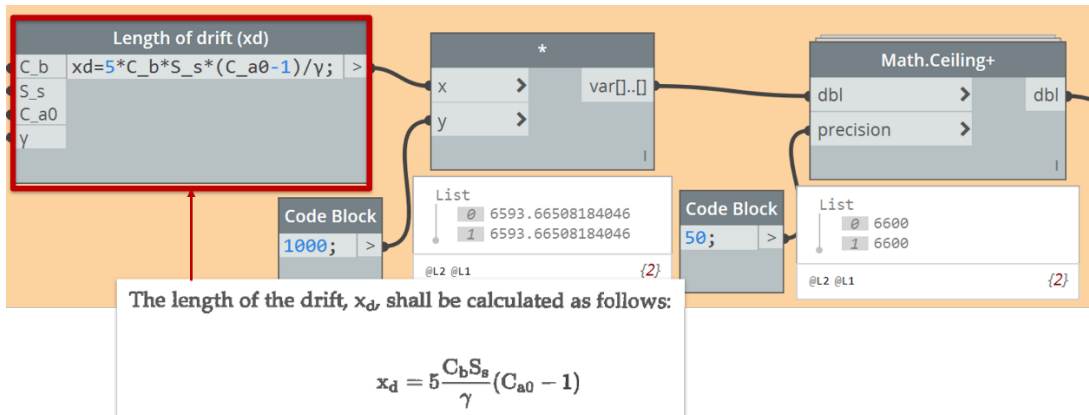


Figure 73: Calculating drift length x_d

V. Create Load Profiles

Create Points to Create Snow Load Profiles

The technologist usually creates filled regions for the snow load profiles. To make the filled regions, we need the perimeter curves, which means we need the perimeter points to create the curves. There are five points we need to create (Figure 74). The load profile should be attached to the drift surface in the plan as shown. Point A is the base point, Point B is the maximum load point, Point C is the point where the drift load becomes as same as the uniform load, and Point D and E are the extent to where we are showing the uniform load. We are going to use the Tangent and Normal vectors of the drift surface to create the points.

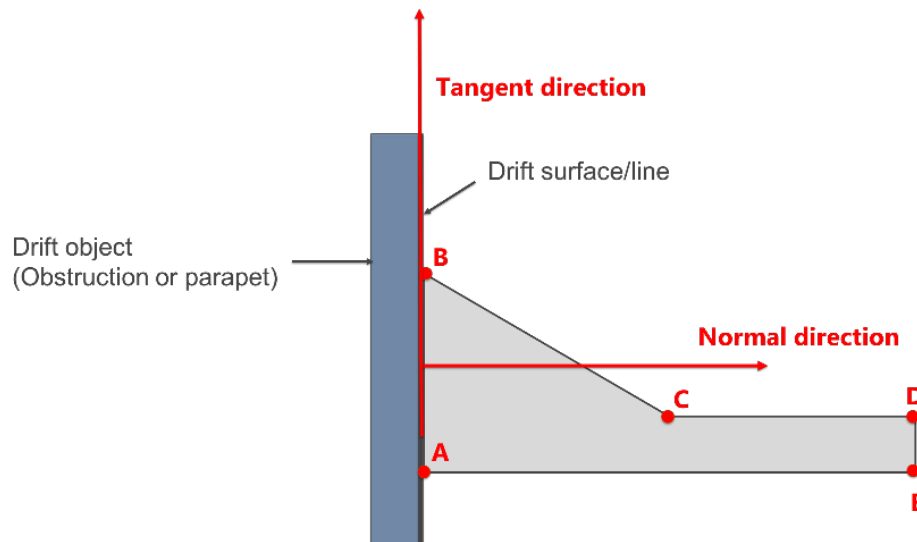


Figure 74: Points to create snow load profiles

First, Point A is created at the middle of the drift base line using 'Curve.PointAtParameter' node. Point B is created by translating Point A to the tangent direction of the drift surface using 'Curve.TangentAtParameter' node. The maximum drift load is remapped by 500 here (ie. if the load is 3.0 kPa, the length from Point A to B in the plan will be 1500mm), but it could be adjusted by the users' preference. Then Point C is created by translating Point A to the normal direction of the drift surface by drift length (ie. move Point A 3000mm to the right) and again translating it tangent direction by remapped uniform load (ie. if the uniform load is 1.0 kPa, move the point 500mm

upward. Lastly, translate Point C in to the normal direction by certain length (i.e. here we moved 1000 mm) to make the endpoint of the load profile. Point E can be created by the same logic with Point A.

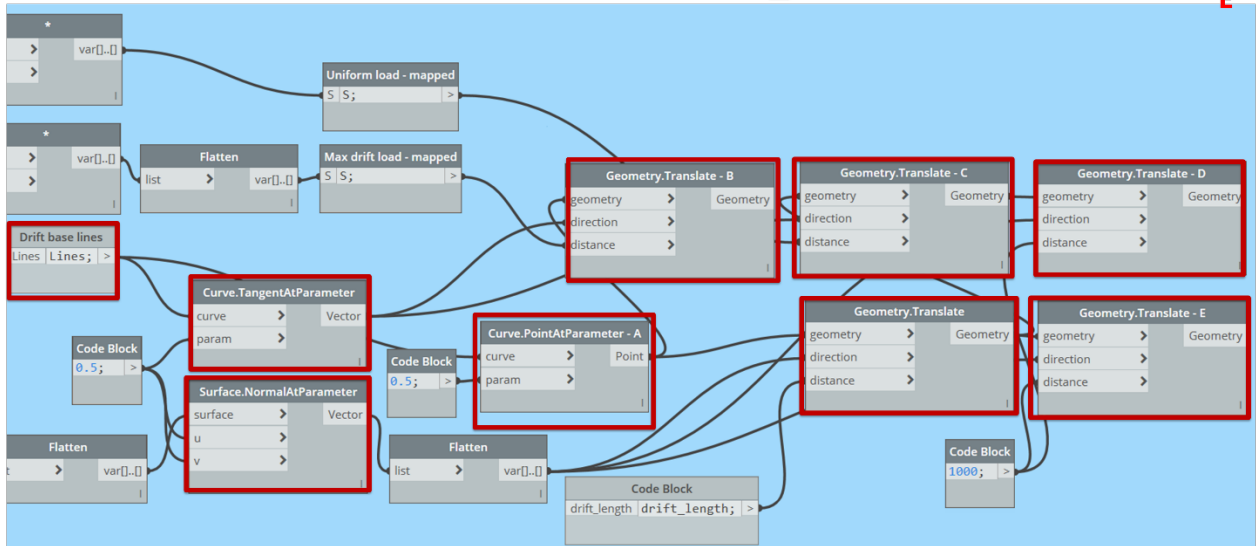
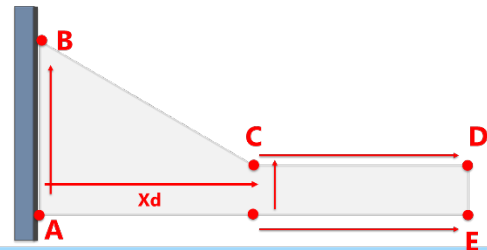


Figure 75: Create points for snow load profiles

Create Lines and Make Filled Regions

With the points, the profile perimeter curves are created using 'Line.ByStartPointEndPoint' node and lastly the filled regions are created. The filled region type can be selected by the user.

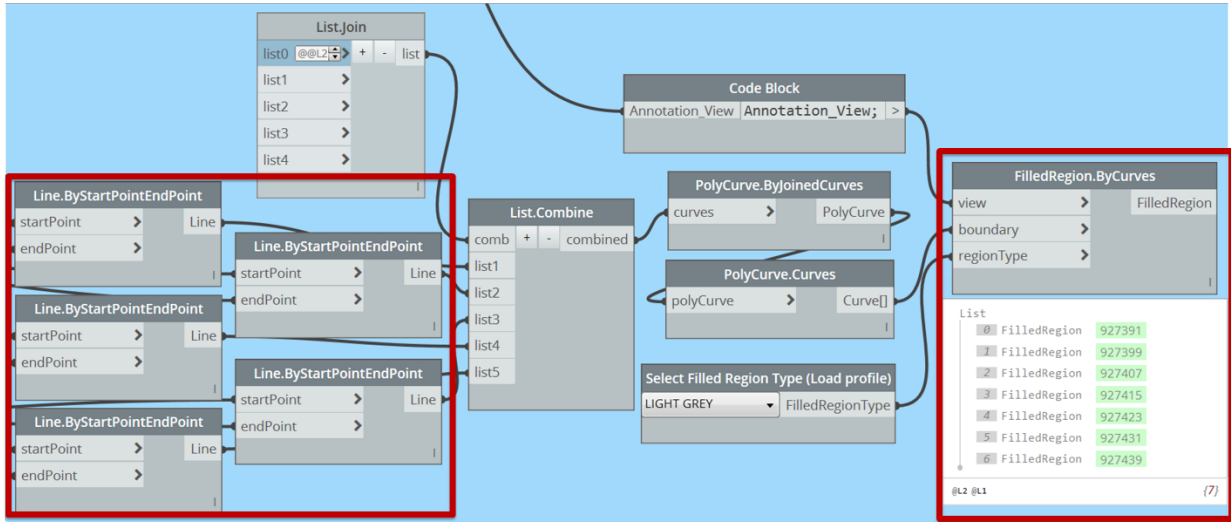
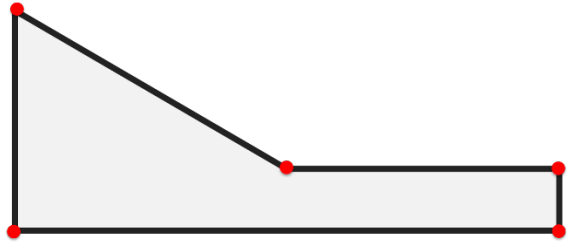


Figure 76: Create lines and filled regions

Set Load Parameters

To annotate the loads on the load profiles, the loads are assigned to filled region element parameters (Figure 77). Here, we used 'Comments' parameter for maximum load and 'Marks' parameter for uniform load. Now each filled region (load profile) has two loads set in their instance parameters.

Create Tags

Lastly, the tags are created (Figure 78). Here we put the Comments (max load) tag at Point A and Marks (uniform load) tag at Point D. The tag families are created beforehand and selected by the users. 'Passthrough' node is used since the parameter values need to be set before we tag those values. The node waits until the *Parameter.SetValue* nodes are run before it will run the *Create Annotation Tag* nodes. Figure 79 shows the final view of the snow load plan.

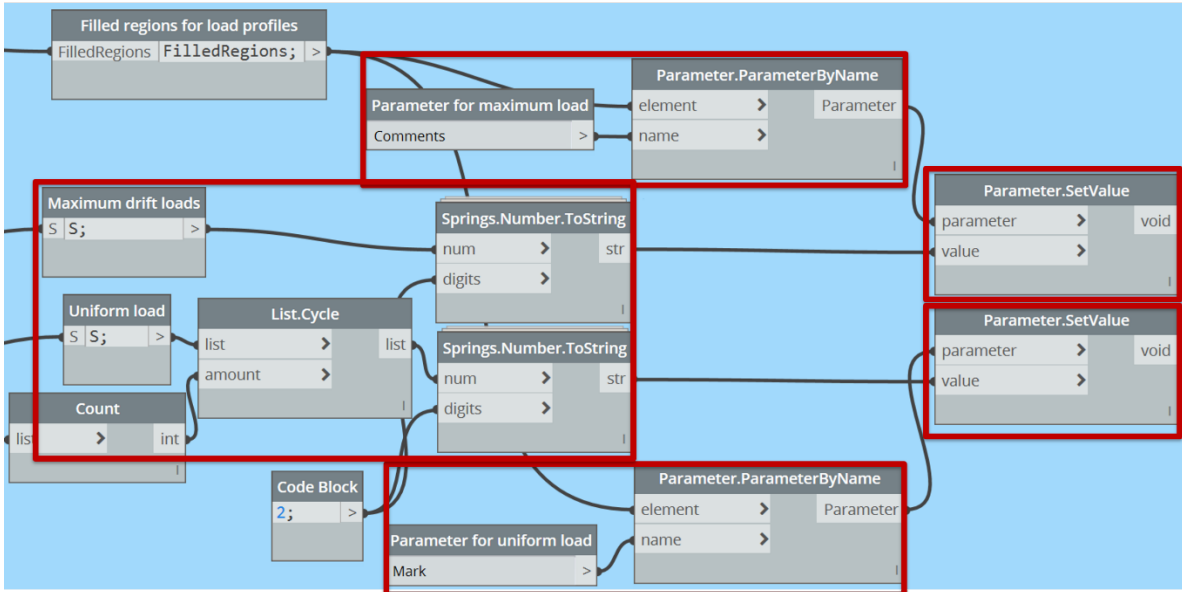
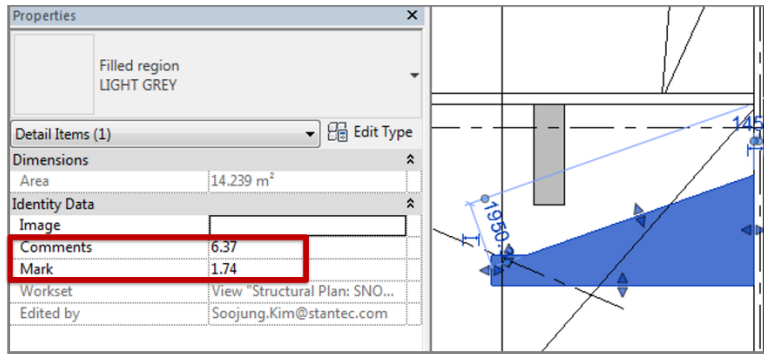


Figure 77: Assign parameter values of load profiles with snow loads

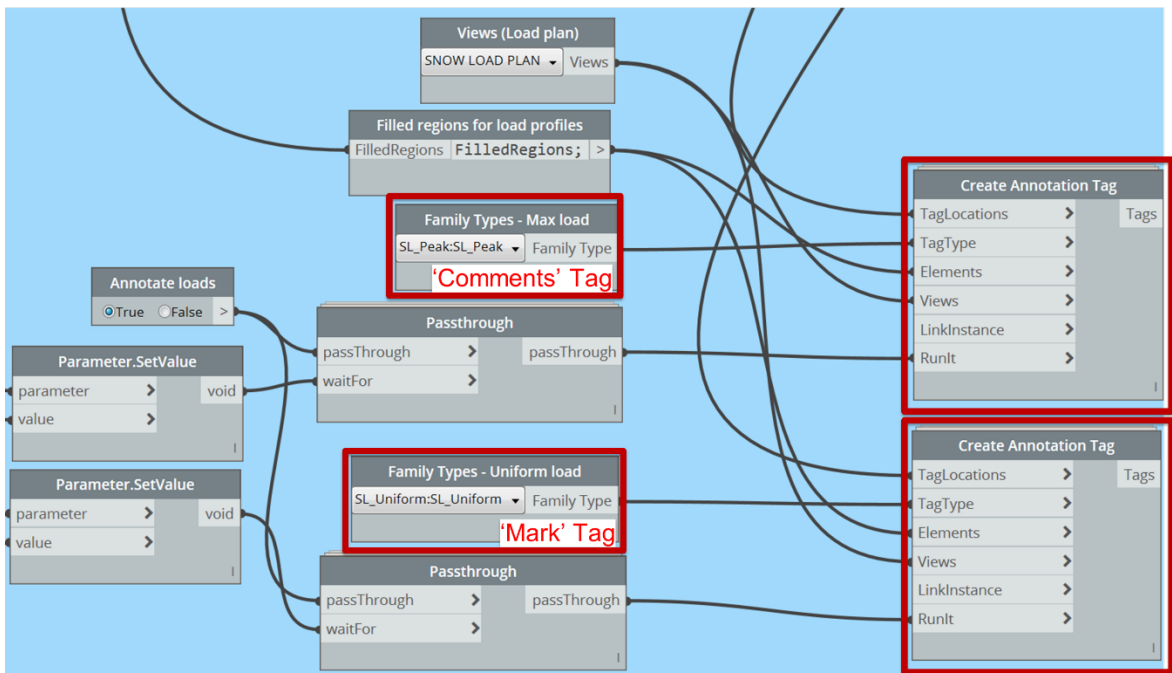


Figure 78: Create annotation tags

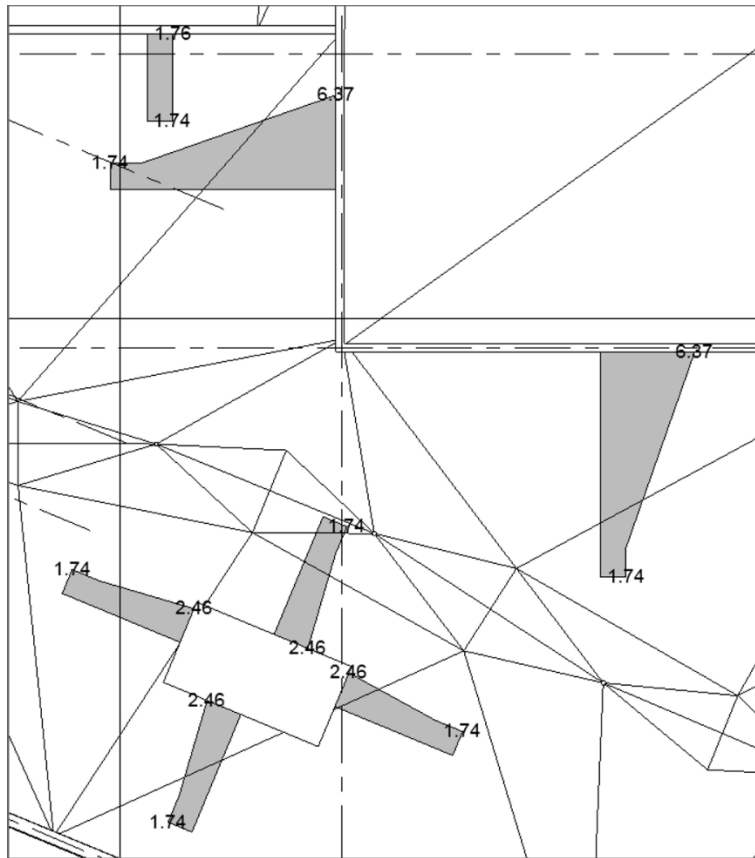


Figure 79: Snow load profiles created

Running with Dynamo Player

To run this script with dynamo player, open the dynamo player and locate the script file. Make sure to assign names to the elements before running the script. First, select the lower roof and input the factors we need. Select Filled Region Type you want, and the view you want to annotate the loads. If you run the script, the script will let you select the elements (adjacent roofs, parapets, and obstructions). After selecting all the objects you need, use Esc button to escape from selection and allow the script to run.

Wind Load Calculation

In this section, we will describe the wind load annotation workflow.

Current Workflow

In the current workflow, engineers follow the code book and calculate the wind load manually based on the tributary region areas of the roof. Then it is passed on to the technologist who annotates it on the sheets manually.

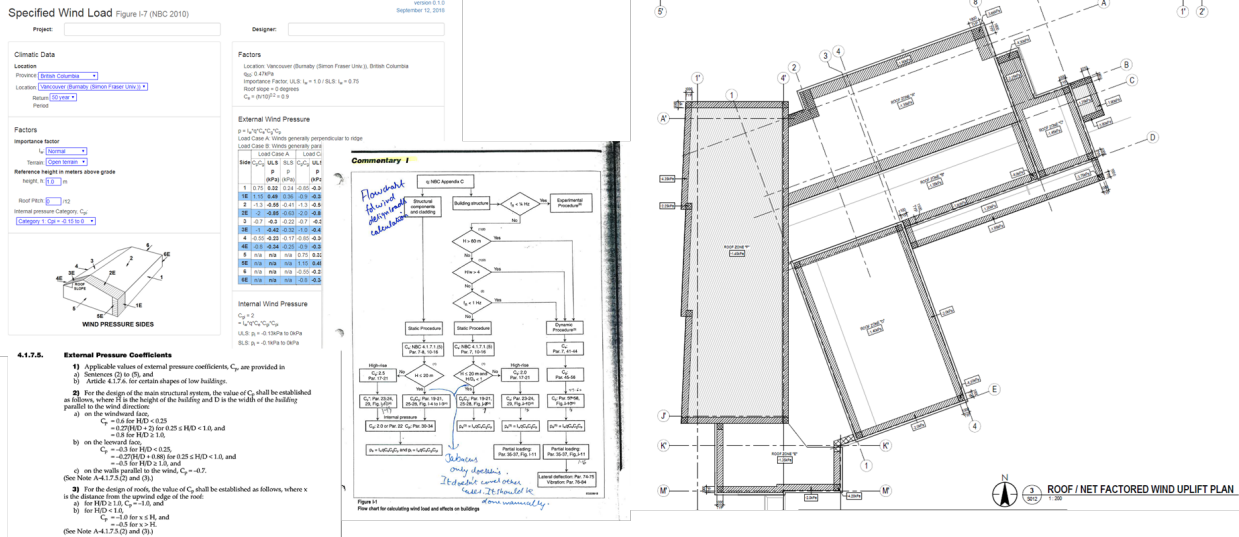


Figure 80: Wind Load Current Workflow

Assumptions

- National Building Code of Canada, 2015
- Case of Roof with slopes less than 7°
- Static method only
- For Ultimate Limit state

Terminologies

According to the code book, the three zones for loading on a roof are named as shown in Figure 81 below.

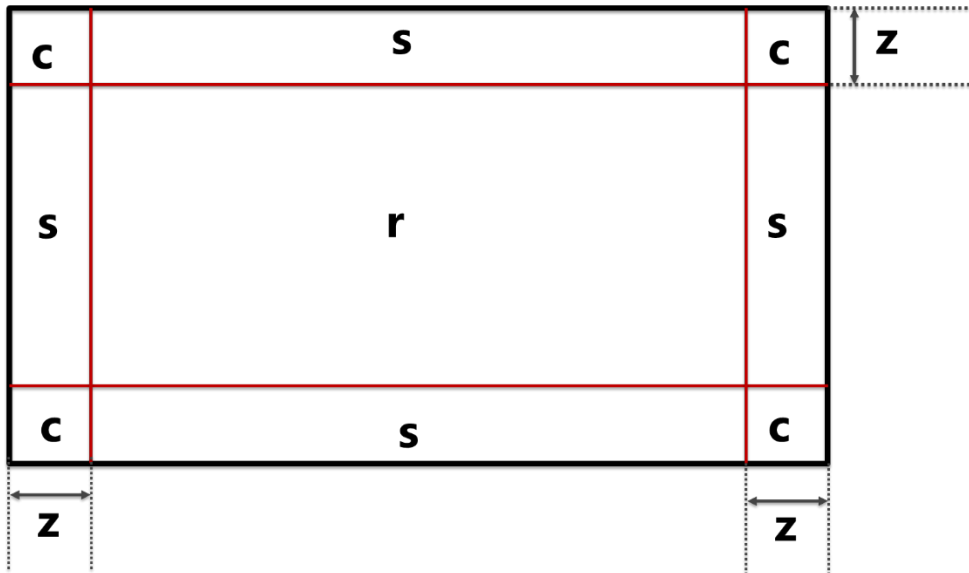
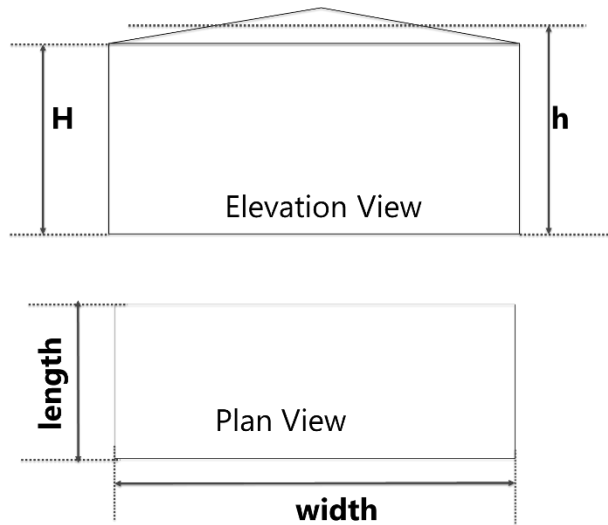


Figure 81: Roof Zones

And the dimensions are named as shown in Figure 82 below.



$$D = \text{Min}(\text{Width}, \text{Length})$$

Figure 82: Roof Dimensions

Wind load is calculated as shown below:

Wind Load = External Pressure – Internal Pressure

External Pressure = $I_w q C_e C_p C_g$

Internal Pressure = $= I_w q C_e C_{pi} C_{gi}$

Where,

I_w = Importance Factor

Q = wind pressure

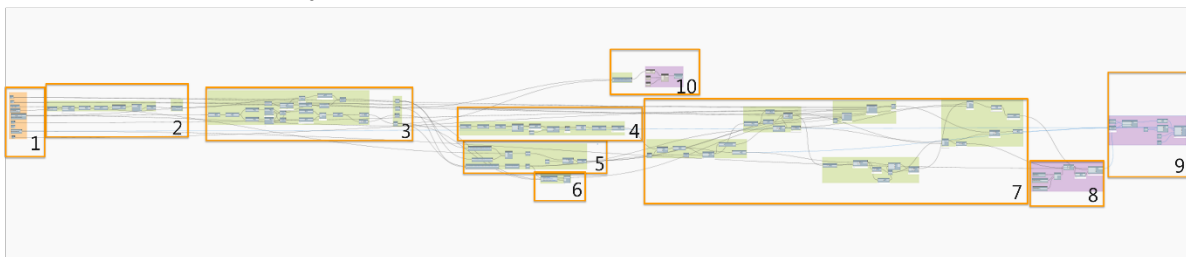
C_e = Exposure Factor

C_g = Gust Factor

C_p = External Pressure Factor

Dynamo Workflow

In this section we will describe the logics used to develop the script. Here we will explain steps 1, 3, 5, 6, 7, 8 and 9 briefly.



- 1- Inputs
- 2- Calculate Base Elevation
- 3- Calculate Roof Dimensions
- 4- Find roof boundary curves
- 5- Calculate C_e and edge load width Z
- 6- Calculate areas and $C_p C_g$
- 7- Create boundary for filled regions
- 8- Create Filled regions
- 9- Create Load Annotations
- 10- Check for Static Method

Figure 83: Wind Load Automation Dynamo Script Overview

Inputs

The figure below shows the inputs for this script.

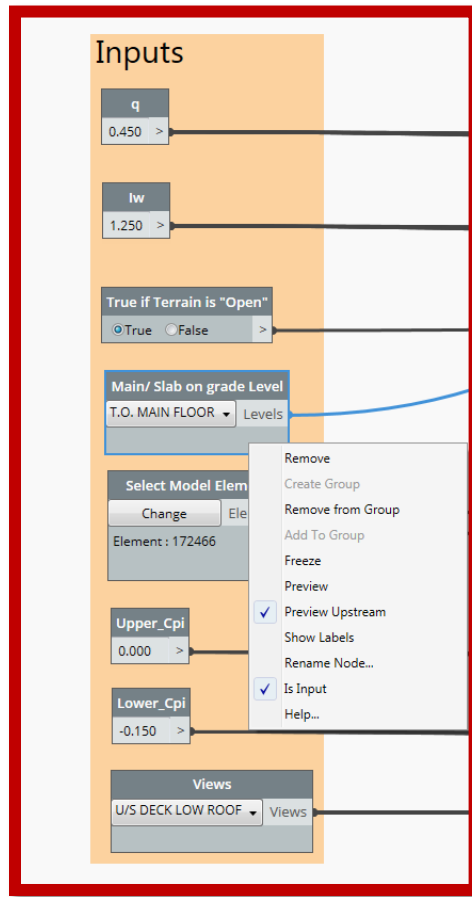


Figure 84: Wind Load Inputs

Calculate Roof Dimensions

The roof height and plan view dimensions are calculated as shown in Figure 85. The concept here is very similar to what we have followed in the earlier two scripts, *i.e.*, using bounding boxes to calculate the dimensions.

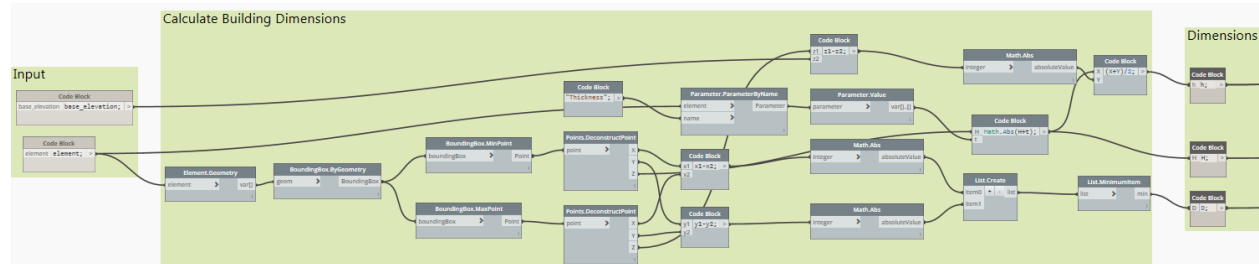


Figure 85: Calculating Roof Dimensions in Dynamo

Calculate Exposure Factor and edge width

We used code blocks as shown in the figure below to do most of the calculations in this script (Refer Figure 86).

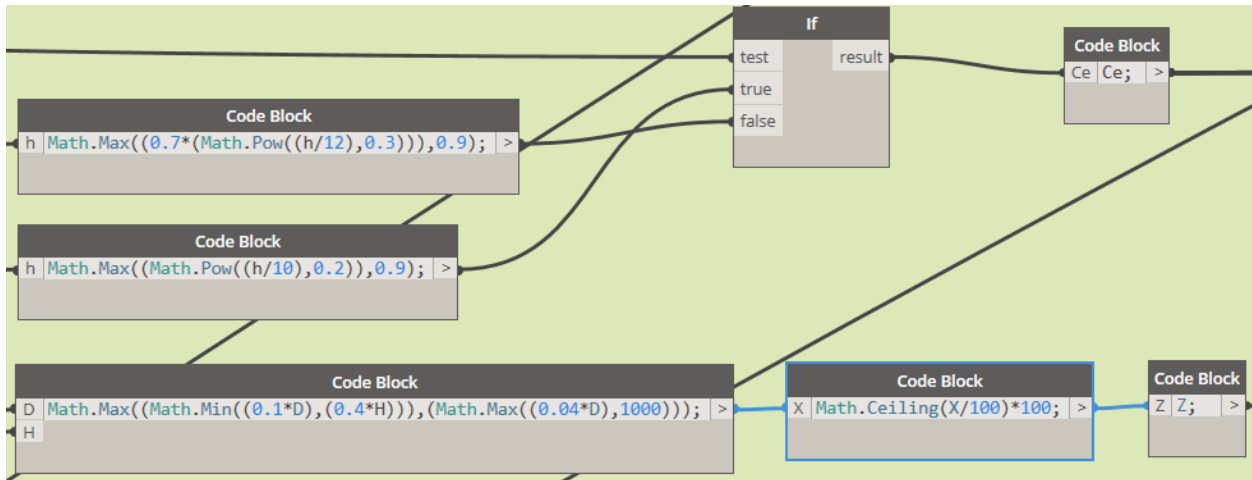


Figure 86: Calculating Ce and Z as per Code Requirement in Dynamo Using Code Blocks

Calculate Areas and CpCg

We used code blocks to calculate the area and a simple for loop using python script node to associate the CpCg values based on the tributary area size as shown in Figure 87:

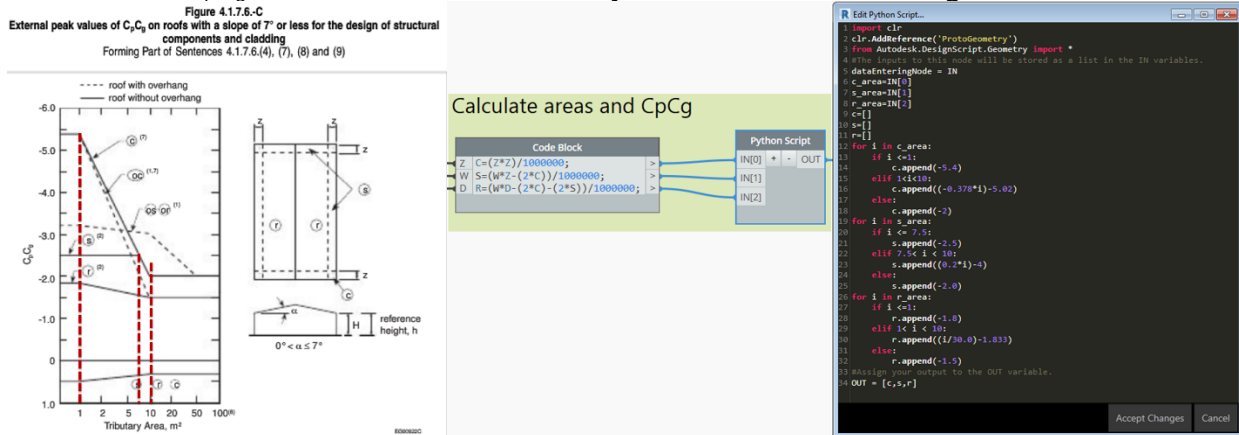
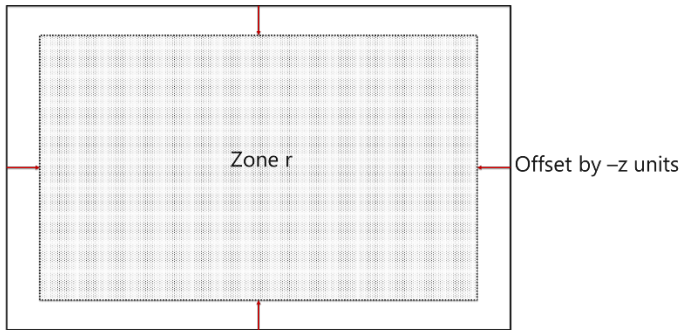


Figure 87: Calculating Tributary Areas and CpCg using Code blocks and Python Blocks for Formulae

Creating Filled Regions

First, we create the filled region for the central r zone as shown in Figure 88. We offset the roof boundary curved inwards by “z” units.



Create R filled region

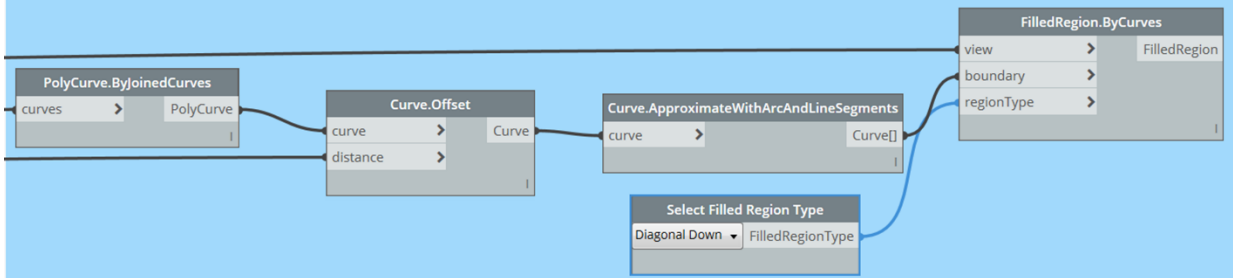
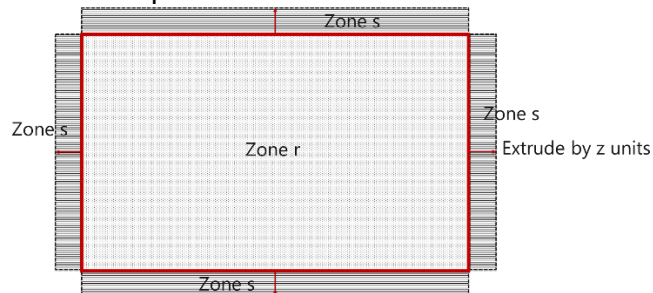


Figure 88: Creating Filled Regions for r Zone

Next, we create the filled region for the edge regions, *i.e.*, s zone as shown in figure below. We extrude the perimeter curves of the zone r surface by z units as shown in Figure 89.



Create S filled region

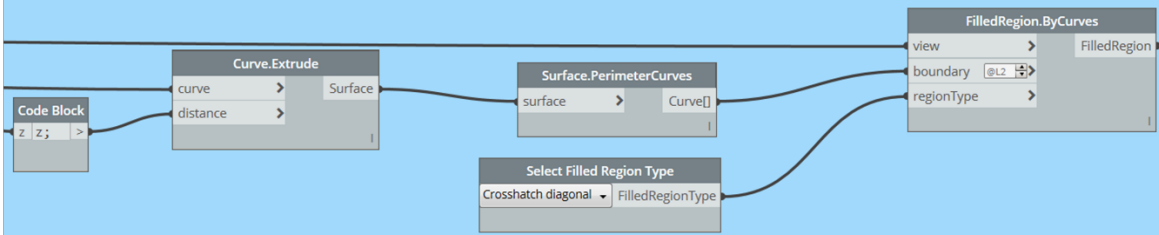


Figure 89: Creating Filled Regions for s Zone

Finally, we create the filled regions for corners, *i.e.*, c zone as shown in figure below. We take the remaining parts of the roof surface by using the geometry split function as shown in Figure 90.

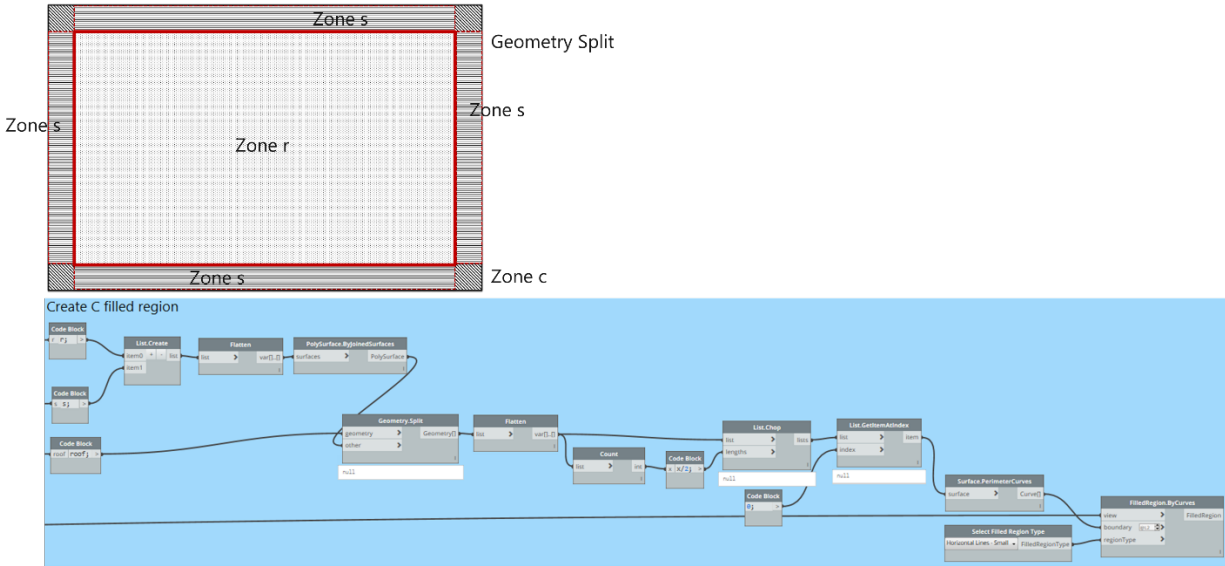


Figure 90: Creating Filled Regions for c Zone

Annotation

Finally, we enter the calculated load to the filled regions as comments and then create tags as shown in Figure 91.

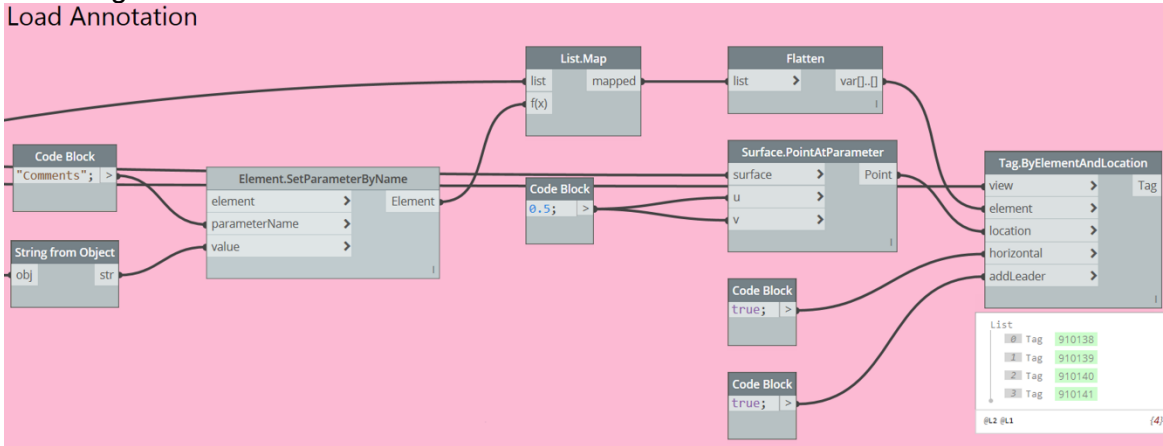


Figure 91: Annotating Filled Regions

Additional Materials: Click [here](#) for Images, scripts and presentation material.