Supporting Document for Call for Code Submission

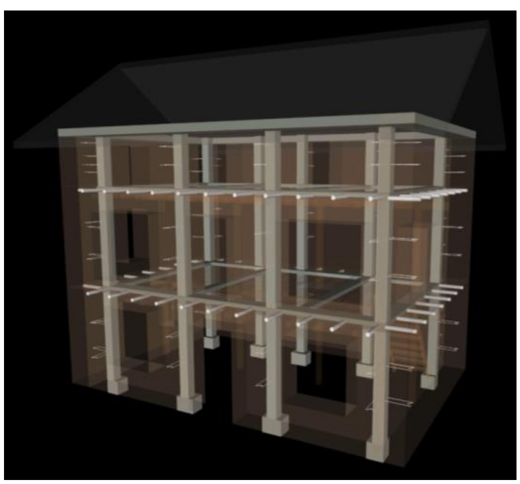
Post-Disaster Rapid Response Retrofit: PD3R

Introduction

Natural disasters have been a determinant factor for the development of civilizations. From landscaping the environment to vanishing entire cities, the forces of nature play an important role in our lives. The impact of these events has been significant, especially when they strike on developing countries. For instance, between 1971 and 1995 Natural Disasters took more than 3 million lives, and affected more than 136 million around the world (HRC, 2004). Historical evidences show how vulnerable we are to the power of earthquakes, typhoons, hurricanes, and more. However, our mindset towards Natural Disasters has changed through time as modern society has focused more on the scientific facts rather than super natural explanations. This change of approach has led to the development of tools that can predict, quantify and reduce risk. As governments and local authorities implement these new tools, life and economic loss after a disasters reduce drastically.

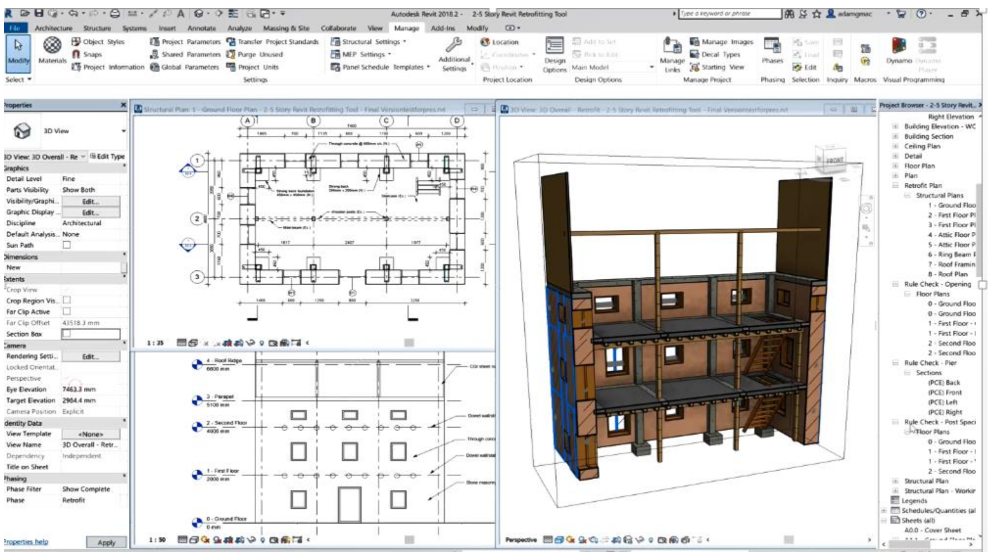
Build Change, Earthquakes and Technology

One of the most significant achievements within our organization, Build Change, in Nepal is the development of a Retrofit Type Design which allow our structural designs to be used massively. The Type Design is the result of years of engineering calculations and the consistent building styles found in many of the districts in Nepal. In few words, the Type Design is one retrofit solution that suit thousands of houses. This enabled us to rapidly asses the condition of a house and use the same structural designs, removing the need to engineer a solution for each house separately. Specifically, our Retrofit Type Design consists of adding structural elements to the house such as columns, beams, slabs and eventually closing or moving openings like doors and windows.



3D Retrofit Model of the Type Design, including concrete columns, beams, slab strips and footings.

After the approval of our Retrofit Type Design by the Government of Nepal, the demand scaled up to thousands of houses that where eligible to be retrofitted. In order to respond to this increase in the demand, in Build Change we developed a tool to produce Type Design in just hours after receiving the request from the field. The way of reducing the time by each Retrofit solution from days to hours is using technology. First, staff from the field takes the measurement of houses with a mobile app, then submits those measurements to the cloud and from the main office in Kathmandu we process the data and create a complete set of construction drawings ready to be implemented. The main improvement in productivity occurs in the measurement processing phase. This is done through a Building Information Modeling (BIM) tool which speeds up and automate a series of tasks, shortening the working time from a days to a couple of hours.



Screenshot of the Revit Tool used to increase the production of retrofit solutions.

Our BIM tool consists of a 3D model of a house which includes all the existing elements (such as walls, windows, doors, roof, framings, etc.) and also the new elements to be built from the Retrofit Type Design (such as concrete columns, beams, slab strips and footings). After receiving the measurements from the mobile app in the field, our BIM tool reads the measurements and adjusts our 3D model to be exactly as the real house. After adjusting all the components we produce a set of construction drawings to obtain the construction permit and start construction as soon as possible.

Objectives

Following Build Change´s main premise to *Build Disaster Resistant Buildings and Change Construction Practices Permanently*, PD3R Team´s main objective is to improve the safety conditions of buildings and reduce human and economic loss after the occurrence of a natural disaster. Now, this is a broad and ambitious objective, so our main specific objectives can be summarized in the following points:

- Develop a methodology based on an AI/Machine Learning System that enables post-disaster retrofitting as soon as possible once the disaster strikes.

- Train an AI/Machine Learning System to determine if a house is eligible for retrofitting, by feeding it with a large volume of digital images generated through a BIM software.
- Design the methodology in a way that allows it´s application globally, adapting the construction types, materials and structural calculations to the context of a particular country or geographic area.
- Respond in a matter of days to a disaster, reducing the time that affected people spends waiting for their house to be retrofitted.

Dataset Production using Python

A special feature of the houses in Nepal is their similarity in the construction type. Most of them resemble a rectangular shape of 4 mud and stone walls, varying from one to three stories. As mentioned in the introduction, this feature was one of the factors that led to the Retrofit Type Design, which can be used massively on thousands of houses with the same configuration. The image below shows a typical house with the rectangular structure mentioned.

Traditional Nepali house affected by the earthquake.

Upon the Type Design houses the engineering team in Build Change could determine the key factors that define if a house is safe under an earthquake. The structural calculations where broken down into 3 simple rule calculations concerning the opening dimensions and their location along the walls. This implies that only knowing the openings location and dimension we can determine if the house is eligible for a retrofit. The areas where we obtain these dimensions are shown in the image below.

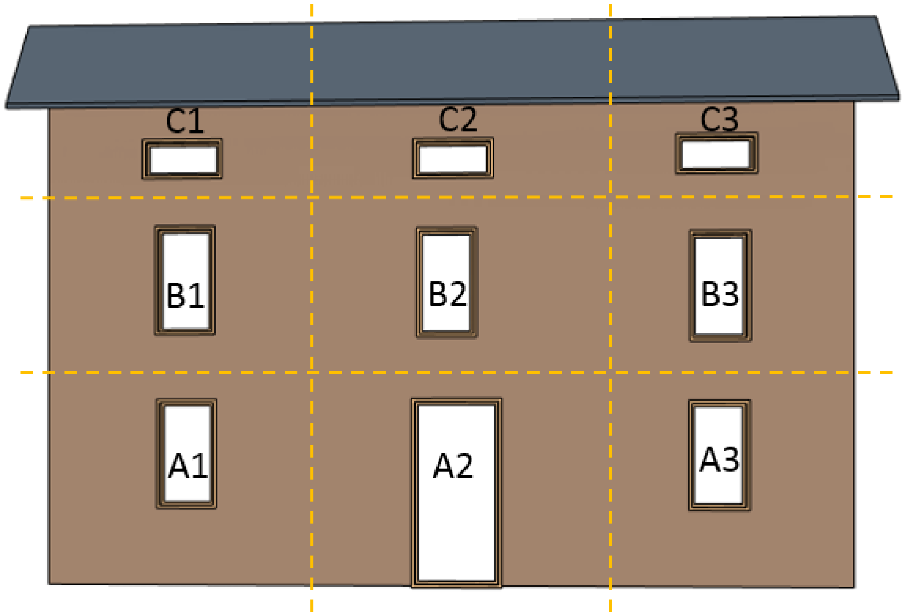
Key areas to measure for the front wall of a house in a retrofit evaluation.

Going back to our second specific objective which aims to train an AI/Machine Learning System, our approach was consistent with the type design and the use of technology such as BIM software. In particular, the automation through BIM, (Autodesk Revit and Dynamo for our case) was crucial because it allowed us to generate millions of possibilities varying the window position on our type design. So, in order to train the system we first had to generate all the possible combinations using a Python Code and a modular approach for varying every window independently according to the typical construction types found in the field.

```python
# Get all combinations for the Wall
length = 8

for elem in itertools.product(*c3):
    c3_comb.append(elem)

allmodsw1   = [a1_comb,a2_comb,a3_comb,b1_comb,b2_comb,b3_comb,c1_comb,c2_comb,c3_comb]
allcombw1   = []


# Cartesian Product of Combinations for all segments to get 4^9 labeled datasets of "Go" and "No-go" scenarios

for num, elem in enumerate(itertools.product(*allmodsw1)):

    #Calculate Opening Percentage per wall
    sum_opw1 = elem[0][1]+elem[1][1]+elem[2][1]
    sum_opw2 = elem[3][1]+elem[4][1]+elem[5][1]
    sum_opw3 = elem[6][1]+elem[7][1]+elem[8][1]

    opercent1  = (sum_opw1/length)*100
    opercent2  = (sum_opw2/length)*100
    opercent3  = (sum_opw3/length)*100

    # Output Status based on Opening Percent and Pier Calculation Checks
    status    = ["go" if opercent1<=35 and opercent2<=35 and opercent3<=35 and calcPier(elem,length) == "go" else "no-go"]

    # Merge lists of lists to get flattened rows per combination
    flattenedrow = [val for sublist in elem for val in sublist]
    allcombw1.append([num+1, *flattenedrow,*status])


#Export Data to Excel
df_w1 = pd.DataFrame(allcombw1)
df_w1.to_excel(filename,index=False)
```

Screenshot of the Python code, applying the List Cartesian Product to generate all combinations of windows for a front wall.

At first, the Python script, “01 Create Labeled Data.py”, is used to create a labeled dataset of 4^9 iterations of possible housing configurations for Stone Mud Mortar (SMM) houses in Nepal. To create the unique combinations, the façade is divided into 9 segments, with four varying possibilities of window position and size per segment. After taking the Cartesian product of the lists of configurations for each segment, a total of 262144 possible combinations were generated.



9 segments to vary the window position and generate all possible houses.

The data was then classified into categories of "Go" and "No-Go", to identify each possible combination as either retrofittable or non-retrofittable. The classification was done based on the Opening percentage limit and the calculated criteria for the amount of solid wall at the edges, and in between openings. This is determined by a thorough engineering analysis done to produce the "Type Design" for retrofitting SMM houses. The classified data was then exported as an excel file to feed into the BIM software for image generation.

The BIM software's Visual programming interface used the labeled dataset in the excel file "01 labeled_dataset.xlsx", to generate a separate 3D model reflecting each combination and to output the corresponding view of the façade as an image.

3D Model Generation

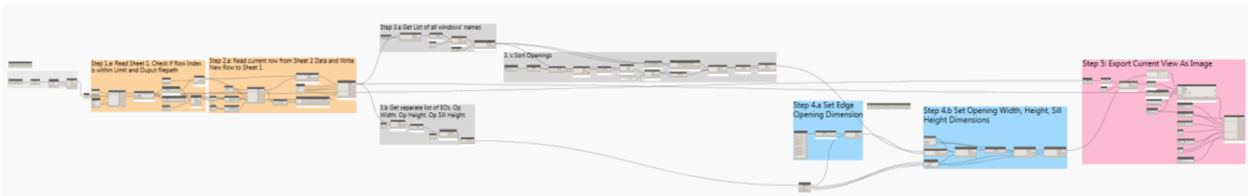
Once we had an Excel file with all the possible combinations for nine openings on the front wall of a house, we started the second part of the process concerning the generation of all those cases in a BIM model to extract pictures and train the AI/ML system. To achieve this we used a Visual Programing software over the 3D BIM model in order to automate operations and loop through the hundreds of thousands of different houses created in the Dataset. The software we used to iterate over all the possible houses is Dynamo, which enables automation of repetitive tasks in a periodic runtime.

In summary, there are four fundamental steps Dynamo follows to generate each case are pointed below.

1. Reads the first row (i) of the Dataset and stores the 36 measurements for the front wall (9 windows and 4 measurements per each).
2. Adjusts the model by shifting the openings to match those 36 measurements.

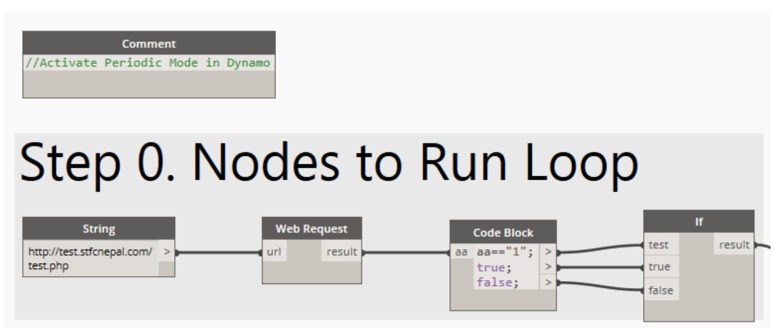
3. Creates a unique file path and takes a picture of the front wall.
4. Repeats step one with the second row (i+1) and repeat until reaching the last value.

The image below shows a screenshot of the complete Dynamo Script which loops over all the houses. The different groups (colored rectangles) perform sets of operations from reading the dataset to shifting windows in the BIM 3D Model.

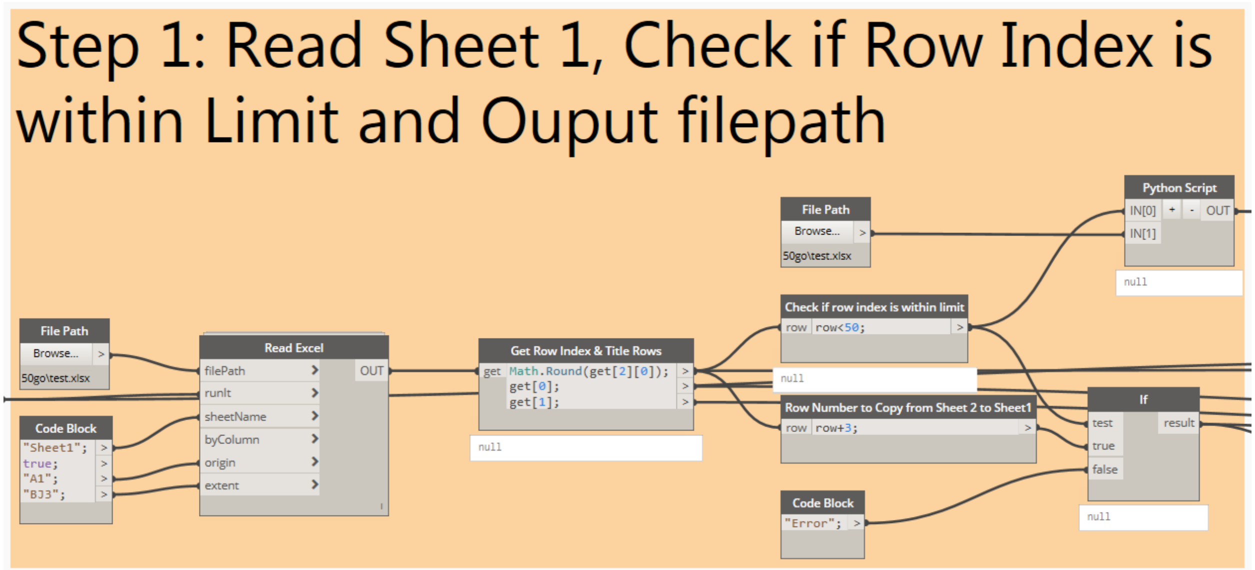


Full Dynamo Script for the model generation of all possible combinations of houses.

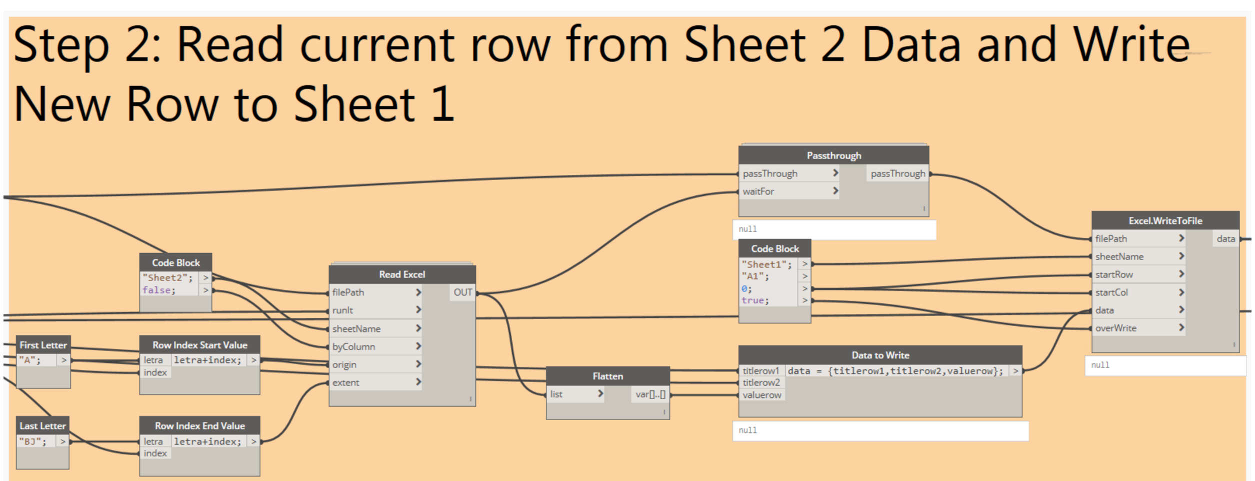
In depth, the Dynamo script uses nodes and wires to connect information to create algorithms that can be run repetitively. Using built in nodes and python code blocks, we are able to adjust the geometry of the model using values taken from a Dataset, which in our case is an Excel file. With this in mind, the 4 step process shown above can be broken down into the following.



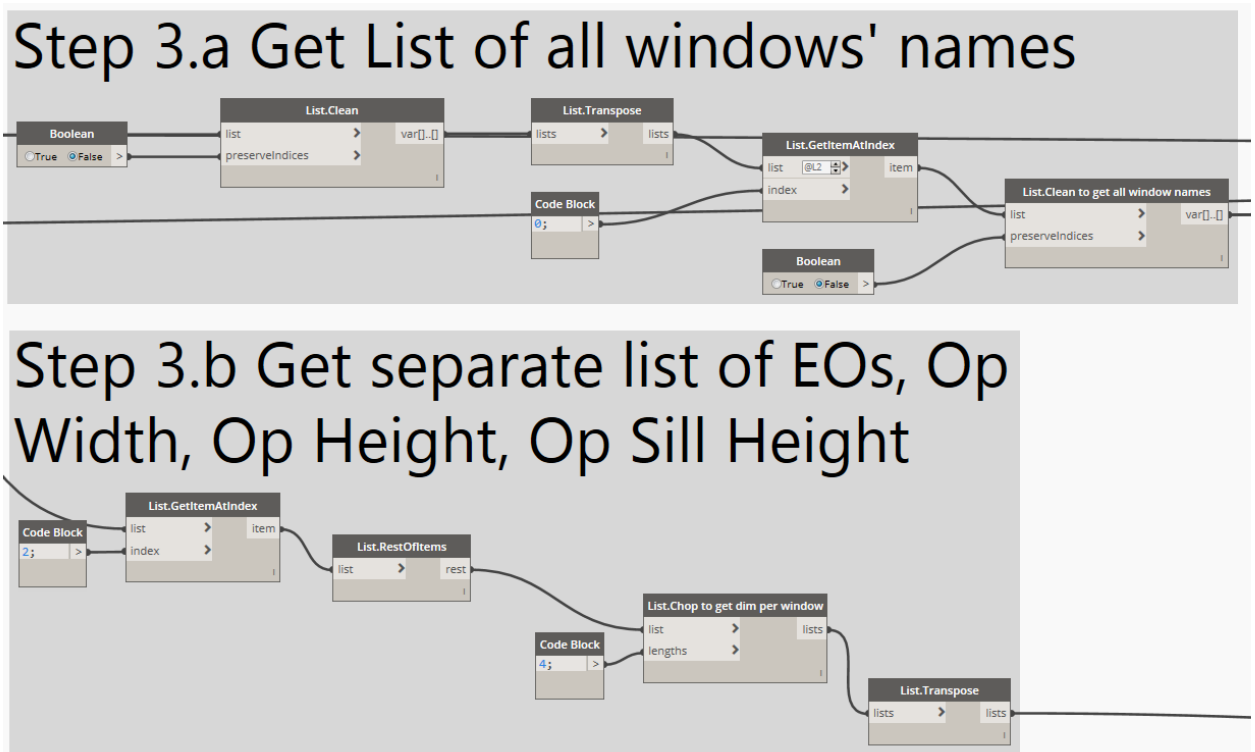
The basic principle to loop and generate all the possible houses is to run periodically our Dynamo Script (also called graph). This means that the algorithms in the graph are executed in time intervals defined by the user. In our particular case our script will run every 10 seconds. To enable the periodic Run in Dynamo we had to include a Web Request Node that uses the data coming from a website we designed. The website http://test.stfcnepal.com/test.php gives the number 1 or 2 every time you refresh it. With this we could set the script "on" and "off" for every run and allow the information in the script to be new for every house. The group of nodes Step 0 gives the whole script the ability to reproduce different houses every 10 seconds with completely new measurements for openings.



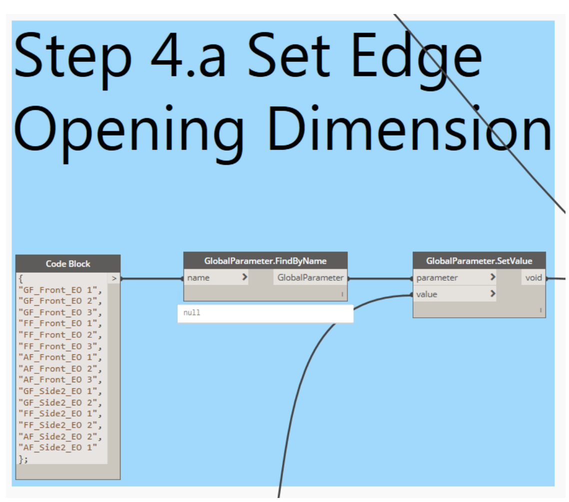
The Step 1 group of nodes reads from the first Sheet of the Dataset to start the looping process. Our Dataset uses two sheets to store all the information: the first sheet stores only one row (corresponding to one house) that is used for the script to know which of the hundreds of thousands of houses is being represented in the BIM 3D model. The second sheet stores all the rows defining all the possible combinations of openings for different houses. Understanding this, the Step 1 group of nodes is in charge of reading the index of the first sheet and verifying that it is still inside the limit (in this case 50 rows). If it is inside the range of houses to be run, it ads three cells to go over the heading rows and then select the next row representing a new house to be modeled.



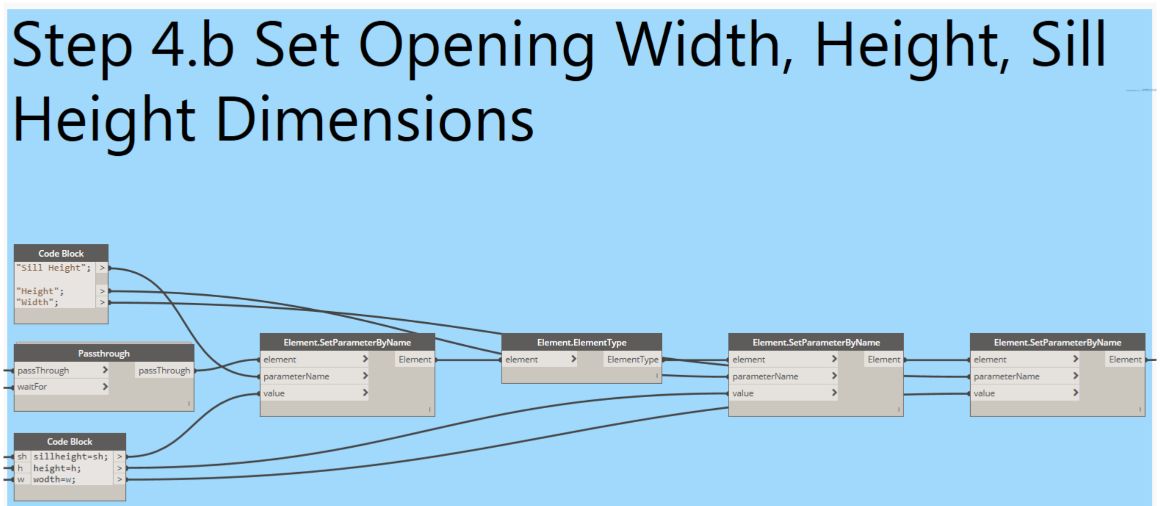
Step 2 takes the resulting row of the previous group of nodes and goes to the Sheet 2 where all the houses are stored in rows. Then, it selects that row including the 36 measurements of the 9 windows on the façade to generate a new house and writes it in the Sheet 1. This process can be described as the updating of the row, from the first house to the second. It is crucial to write the next house in the first sheet because this way the script knows which house is currently being modeled (Steps 3 and 4).



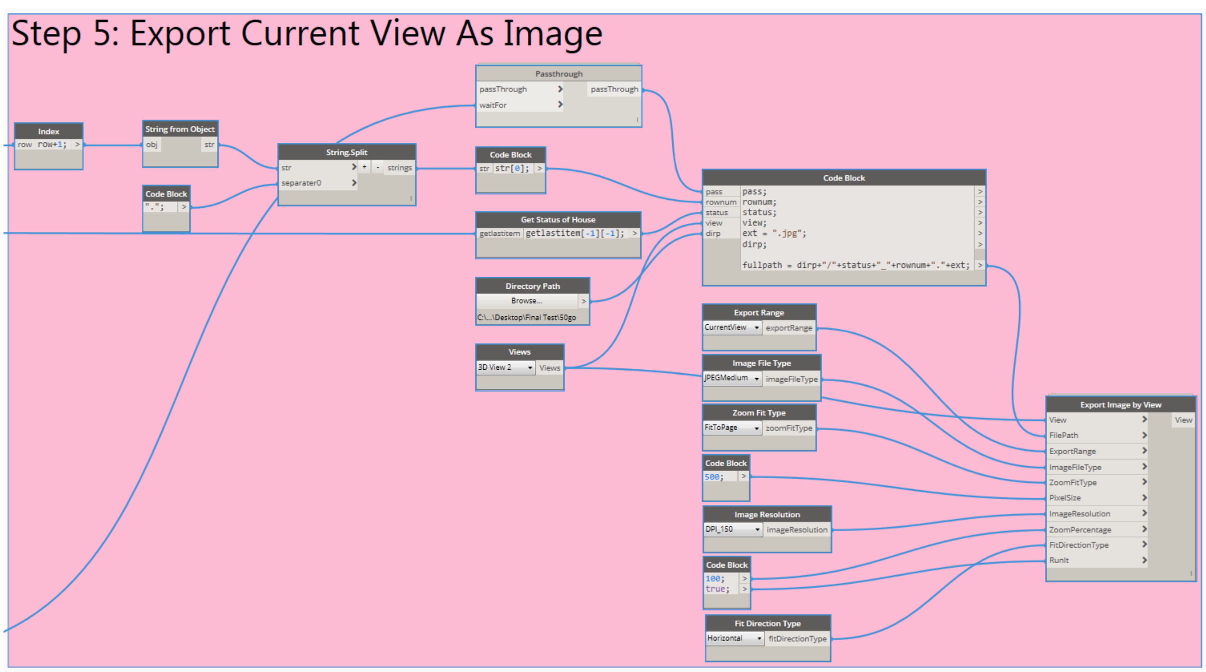
Once the previous group of nodes obtained the window measurements for the next house, the Step 3a and 3b gather the data and sorts it to match the windows in the BIM 3D model. Each window has 4 parameters that describe its position along a wall and those values have to be sorted in lists with specific structure to be read by Revit. These are Edge Opening, Width, Height and Sill Height. Once the Step 3 sets accordingly the 36 values for the new house the script can read and shift the windows.



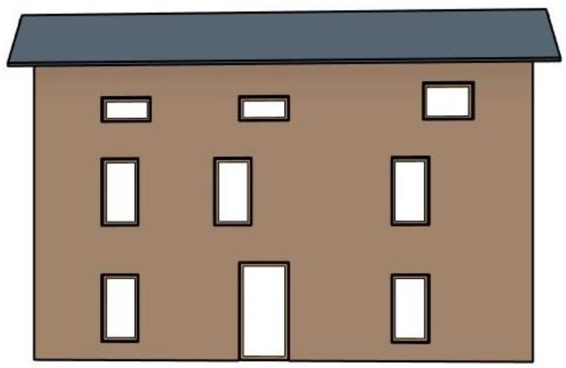
Step 4a sets the Edge Opening for each of the nine windows in the façade of the house. The Edge Opening describes the distance from the perpendicular left side wall to the edge of the window or door. These values were previously generated by the python script to resemble the typical configuration of a house in Nepal.



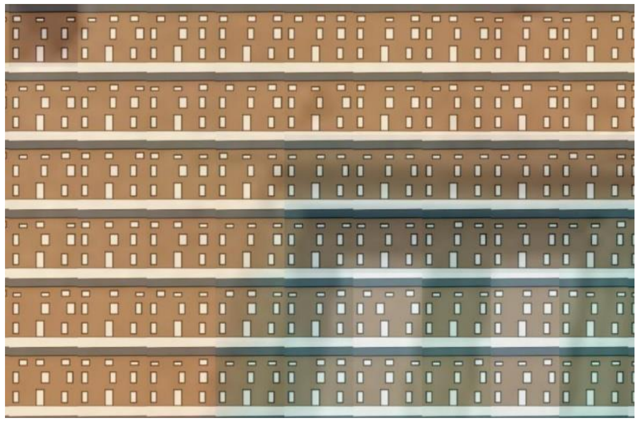
The second part of the Step 4 Group of nodes adjusts the other 3 parameters of openings. Width, Height and Sill Height are changed by setting the parameter of each window to match the one in the Dataset. After the Step 4.b Group of nodes is executed, the BIM 3D model is a complete representation of the house.



The fifth and final group of nodes takes the current view of the BIM 3D model and exports it as an image. With this final task, every house that was run periodically through dynamo is saved as an image to feed the AI/Machine Learning system. After running different combinations of openings through the Dataset we obtained a large volume of JPEG images previously labeled as Go or No-Go for our Retrofit Type Design. The result is shown below.



Random example of a house after varying the position and configuration of the 9 openings in the front wall.



Mosaic artistic representation of the large volume of virtually generated houses using the 3D Modeling software.

With a large volume of virtually generated images through a BIM 3D model, we are capable of training an AI/Machine Learning system to judge on real pictures of houses and determine if it's retrofittable or not. The key idea for this project is representing real world photos with BIM models produced massively for disaster preparedness.

Use of Digital and Real Images for Training an AI Model

After generating 262144 images for "Go" and "No-Go" cases, the images were pre-processed for training the AI model.

For preprocessing part, firstly the generated images were mixed with few real images of houses of Go and NoGo types. Then using OpenCV's Canny edge extraction filter, all the data were converted into edge features so that both digital and real images looked similar. Then the custom made convolutional neural network model is trained on this dataset.

For the edge filter conversion, "convert_data.py" file is used. The output of the file is saved as numpy file. For training the model, "train.py" file is used. Here, firstly the numpy files are loaded, then converted into Tensorflow's TF.data format for better performance. Then the model is constructed to support the data type and training.

Following are the specifications of neural network model, created using Tensorflow's TF.Keras api.

Convolutional Blocks: 6

- ➔ Convolution 2D Layers : 6
- ➔ Maxpool 2D layers : 6
- ➔ BatchNormalization layer :6
- ➔ Dropout layers: 6

Fully Connected Layers: 2

- ➔ First Fully connected layer : 1024 Neurons
- ➔ Second Fully Connected Layer: 256 Neurons

Output Layer: Sigmoid

Optimizer Details:

Optimizer: ADAM Optimizer with 0.001 Learning Rate

Loss Function: binary cross entropy

Metrics during training: Accuracy

The model is trained for 100 epochs.

<u>Final Accuracy and Metrics</u>

Training Accuracy: 86%

Validation Accuracy: 78%

Test Accuracy on only Real Data: 65

When the training image is passed to the network, it passes through several layers of deep convolutional neural network, where there are total of six convolution blocks, of which consists of a convolution 2d layer, max pool 2d layer and a batch normalization layer. After 3 consequent Convolution Blocks, a dropout layer is introduce to regularize the weights and reduce the overfitting in the model. After 6 convolution block, the output is flattened using Flatten() function and two fully connected layer is introduced, which has 1024 and 256 output neurons, between which another dropout layer is also introduced with dropout rate of 30%. Finally, the input is passed to the output layer which has sigmoid activation and does binary classification in our input data. Over the period of 100 epochs the model is trained with binary cross entropy loss function and ADAM optimizer for gradient updates.

Final Training accuracy of the model is 86% and Validation accuracy is 78%. When the model is tested on real non-generated images, the accuracy is 65%.

Future Plan

The current model of the project is a proof of concept model, where we have demonstrated how digital images along with few real images, when mixed together can create an image recognition AI, which can classify between houses that are fit to retro fitting and that are not. For the future, we will need to add more digital data, as well as real data along with supporting measurement values and will need to add labelling in image data to increase the accuracy of the model and make AI more precise.