# Oasis

## Refuge from the busy city

**Final Changes Report**

Jonas Althuis, Hannah van der Sluijs, Adrian Wong, Benjamin Laagewaard

# Contents

This short report highlights the research and changes we considered following the final presentation of our project. These changes are a combination of work we had started before the presentation but not yet managed to finish in time as well as adjustments suggested by our teachers Pirouz Nourian, Hans Hoogenboom & Shervin Azadi after the final presentation. This report contains the following:

# Method for placing vegetation on our plot

## Critique

One of the critiques of our generative system at the time of our final presentation was the method we used for placing squares of vegetation on our plot.

The way we initially did it involved the following steps:

1. Placing points of interest of the neighborhood in the houdini model.
2. Finding the points nearest to these POIs in our grid of voxel points.
3. Placing a circle on these points such that the sum of the areas of each of these circles was equal to 30% of the plot, which is the amount of area specificied in the program of requirements as vegetation.
4. Calculating the amount of sunlight received at each of these circles.
5. Re-weighing the area of the circles to reflect the amount of sunlight it recieved, such that circles with more sunlight became bigger, as they were more suited for vegetation.

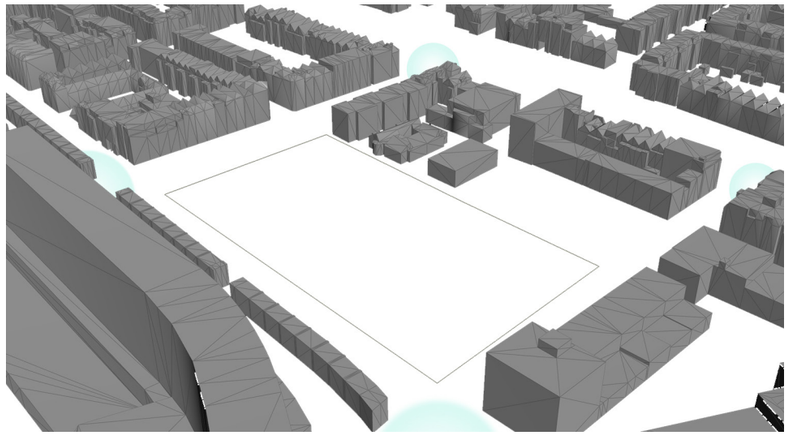


Fig. 1. POIs in neighborhood



Fig. 2. Circle on closest point of our plot

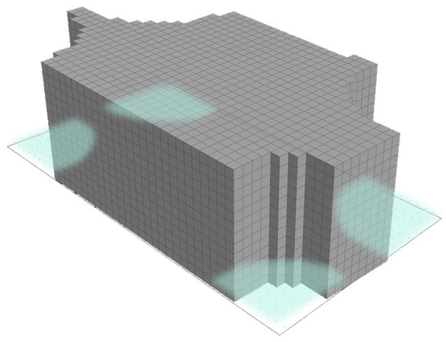


Fig. 3. Voxels removed from building envelope

# Improvement

Rather than shaping the vegetation on the plot using circles, a better method might be to measure the sunlight value of all the points on the bottom layer of our voxel grid. This would have to be done after the placement of our building as these locations would otherwise be in the shade of the building. The figures below show this process, dedicating the most suited 30% of our plot area to vegetation space.

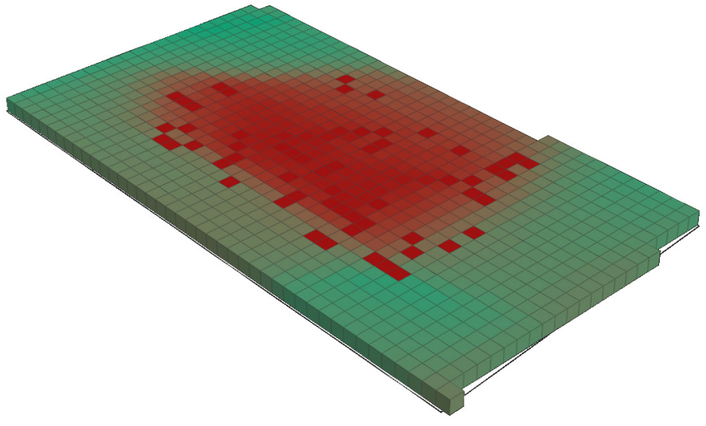


Fig. 4. Sunlight quality at each point, where green is the best and red is the worst.

The steps for this are the following:

1. Calculate sunlight value at bottom layer of grid points, including with our building.
2. Calculate 30% of plot area, and divide by voxel size to get necessary amount of voxels to fit vegetation in.
3. Keep only the necessary points, discard the rest.

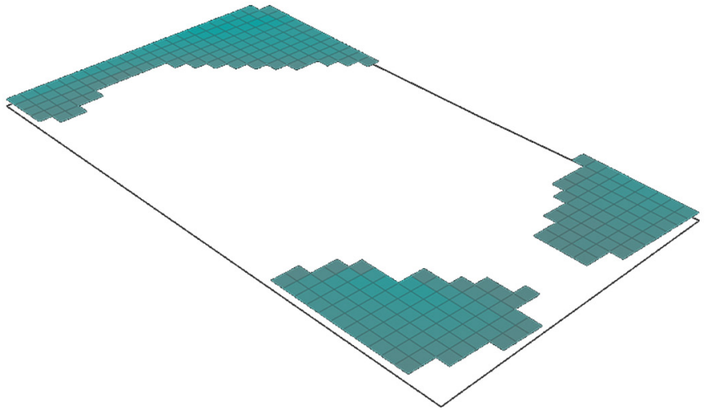


Fig. 5. Vegetation plots based on vegetation area needed and most suitable points.

# Research into clustering node

## Scope

As part of our method for generating corridors and shafts as circulation in our building, we used a Houdini node to cluster our functions together. This is shown in the two figures on the right.

The node used is called “Cluster Points”, and can be found in our *“Clustering”* geometry in the *“Circulation and Final Building Generation”* network box of our Houdini file.

While using the cluster points node, I noticed that it has many parameters for the way it works. The main input it gets is described as “points to cluster”, relatively straightforward. The main attribute used as a basis for clustering is called the “control attribute”, but multiple control attributes can also be used. Interestingly, the weight of this attribute can be controlled.

Using “P”, the point position vector, as control attribute creates clusters based on point position. The number of clusters can also  and a seed can also be specified.

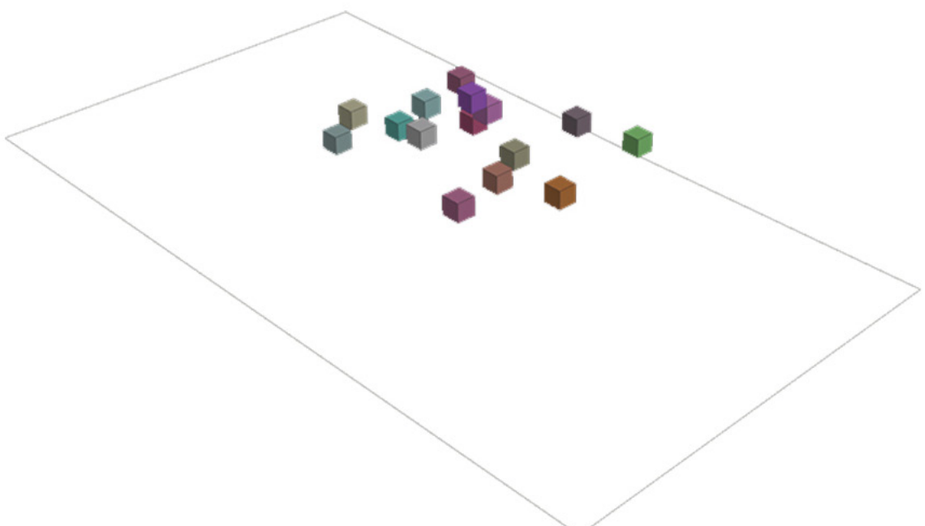


Fig. 1. Center points of functions after being placed with growth algorithm.

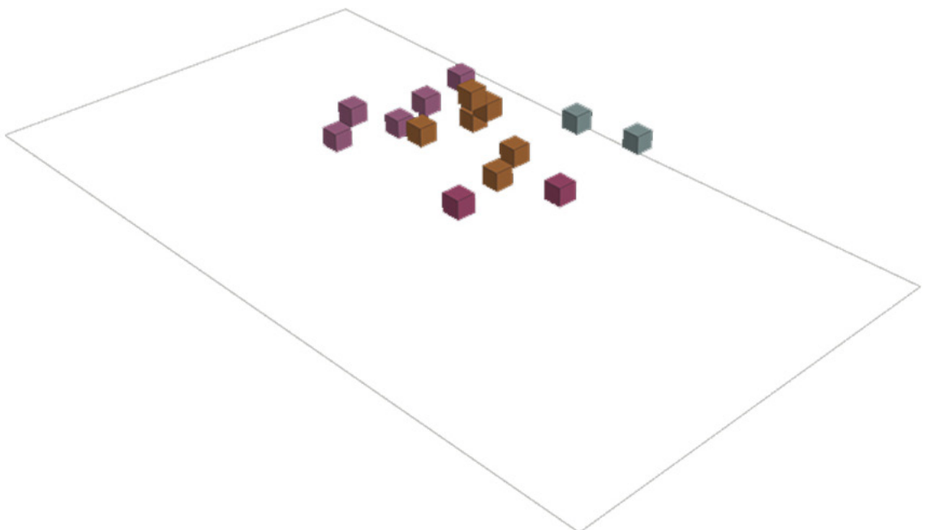


Fig. 2. Clusters of functions with Houdini ‘Cluster Points’ node.

# Clustering Method

To gain more insight on this clustering method, we looked at the documentation for this node, which can be found on the SideFX website, at the link below. This is also useful so that we know how to more effectively use the parameters that the node offers.

The documentation outlines how the node uses the K-means clustering algorithm to group points.

K-means clustering looks for a fixed number of clusters in a dataset, this can be controlled using the “Clusters” parameter in the node. It starts by creating centroids, and other data points are allocated to these centroids by trying minimize the sum of squares in each cluster. This is an iterative process, the number of iterations for which can also be defined in the node.

The sum of squares is a value that represents the average distance between the cluster center and the points in the cluster. By reducing this iteratively, the most “efficient” cluster configurations can be found.

This means that each cluster has an optimal center such that every member of the cluster has a relatively low distance to it’s center compared to the distance to other cluster centers.

Documentation:
https://www.sidefx.com/docs/houdini/nodes/sop/clusterpoints.html

# Variations of final result

## Changes in envelope

The system we've created for generating our building is quite versatile, and has many parameters that can be changed. One of these aspects is the envelope of the building, which is contolled by solar and acoustic thresholds. The shadow-casting threshold controls how the building casts shadow on the surroundings, effectively changing the height of the building. It can be found in our *"shadow casting"* geometry in the *"Envelope Shaping"* network box. The acoustic threshold removes points with a high acoustic pollution value. It can be found in our *"acoustic analysis"* geometry in the *"Data Analysis"* network box

Both of these can be changed to tweak the envelope of the building, as shown in the images to the right. There is of course a limit where there simple are not enough voxels left over to grow the entirety of the functions, which should not happen. To control over this, we created an area "checker" that calculates the percentage of area we have compared to the area that is required given our functions. This can be found in our *"area checker"* geometry in the *"Envelope Shaping"* network box.

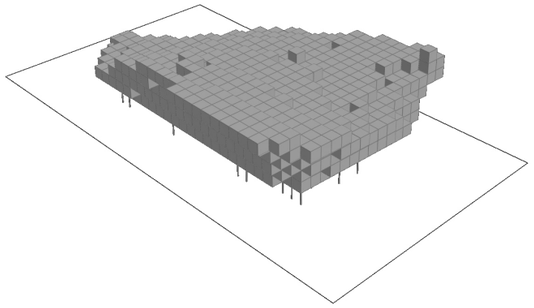


Fig. 1. Default setting, shadow-casting threshold at 0.46, acoustic threshold at 0.55.

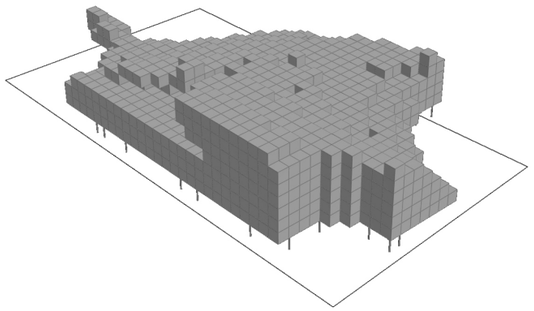


Fig. 2. Low acoustic, shadow-casting threshold at 0.46, acoustic threshold at 0.

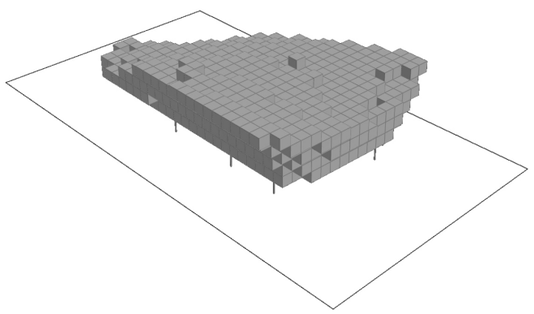


Fig. 3. High acoustic, shadow-casting threshold at 0.46, acoustic threshold at 0.7.

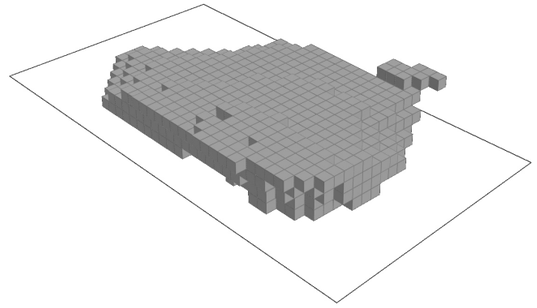


Fig. 4 Low shadow, shadow-casting threshold at 0.2, acoustic threshold at 0.55.

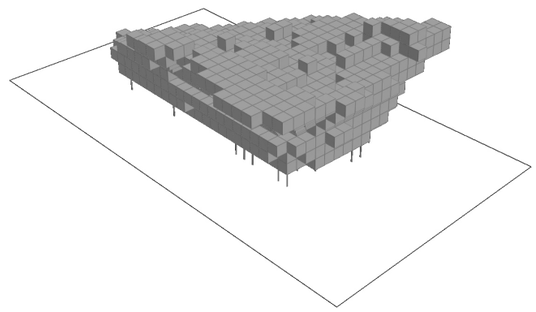


Fig. 5. High shadow, shadow-casting threshold at 0.6, acoustic threshold at 0.55.

It is important to note that some of these settings break certain features in the rest of our system, incorrectly generating columns for instance. These problems would need to be fixed and made more robust in a future iteration of our system. It is generally not the idea that these thresholds should change that much from their default values, which we have found to create good results.

# More variations

Different ways to generate more variations using our system could include changing the data weights for each function so that the weighted product calculation calculates completely different values, or changing the blockiness value of each function so that the system grows the functions differently. These changes are shown on the right.

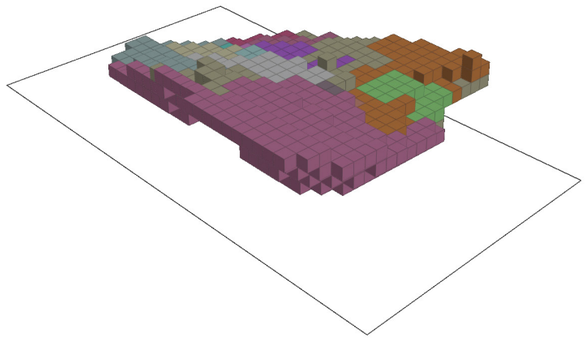


Fig. 6. Default model, no changes made to weighted product or blockiness.

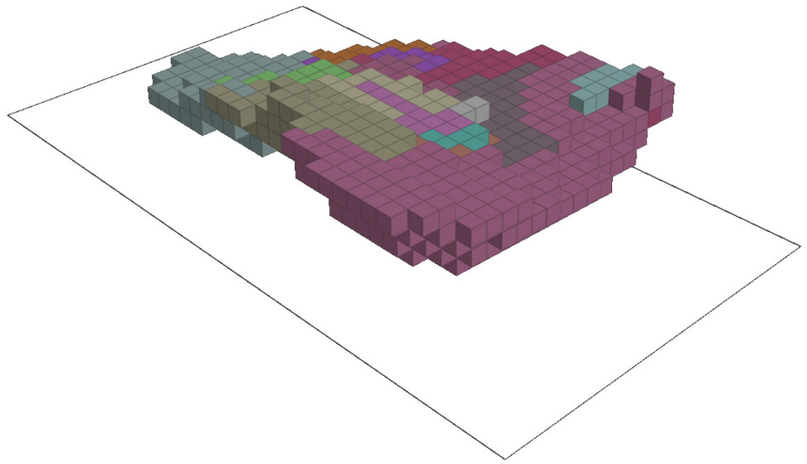


Fig. 7. Random generation of weighted product weight values.

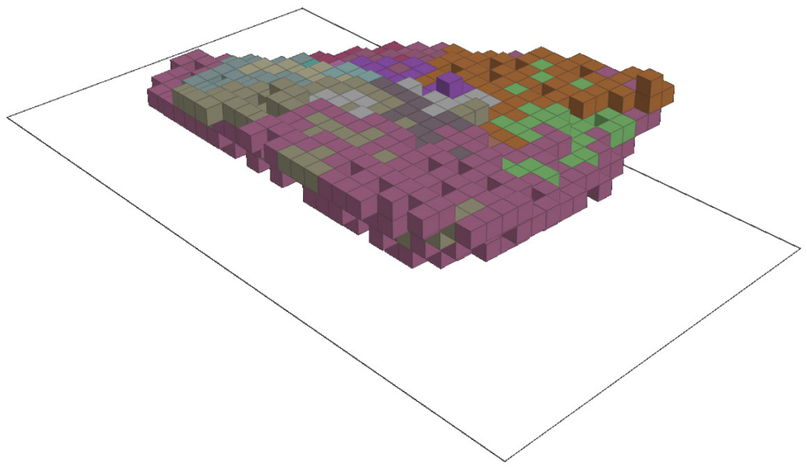
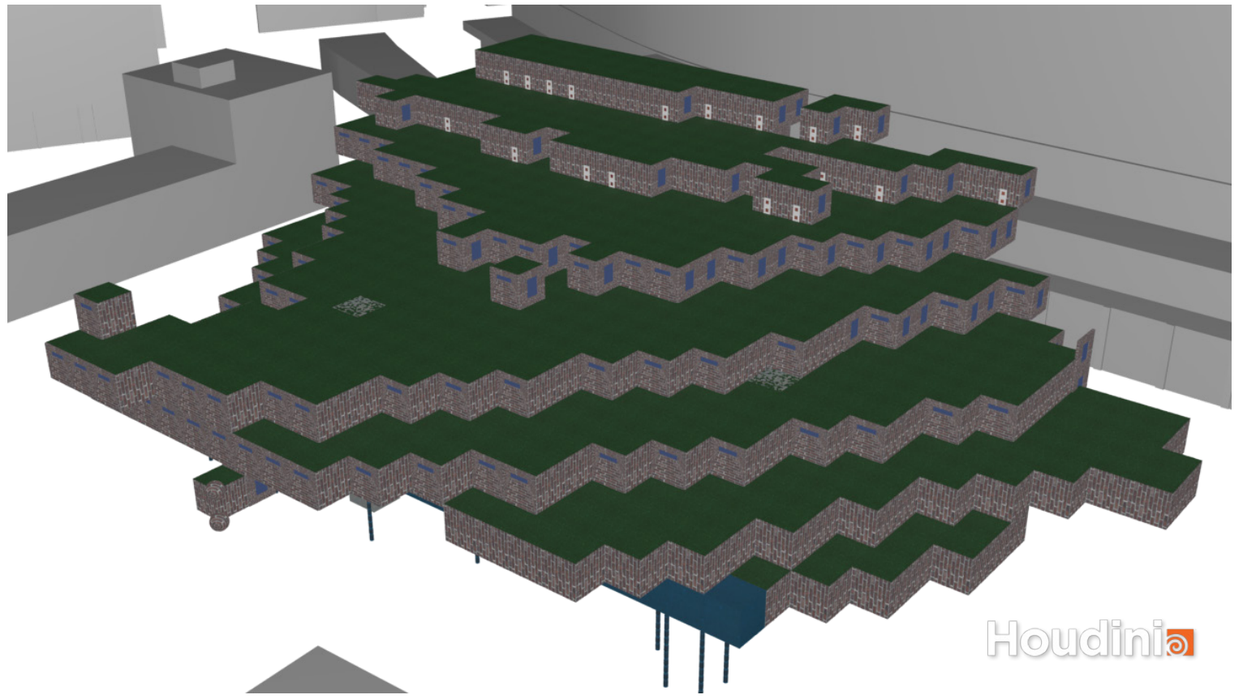
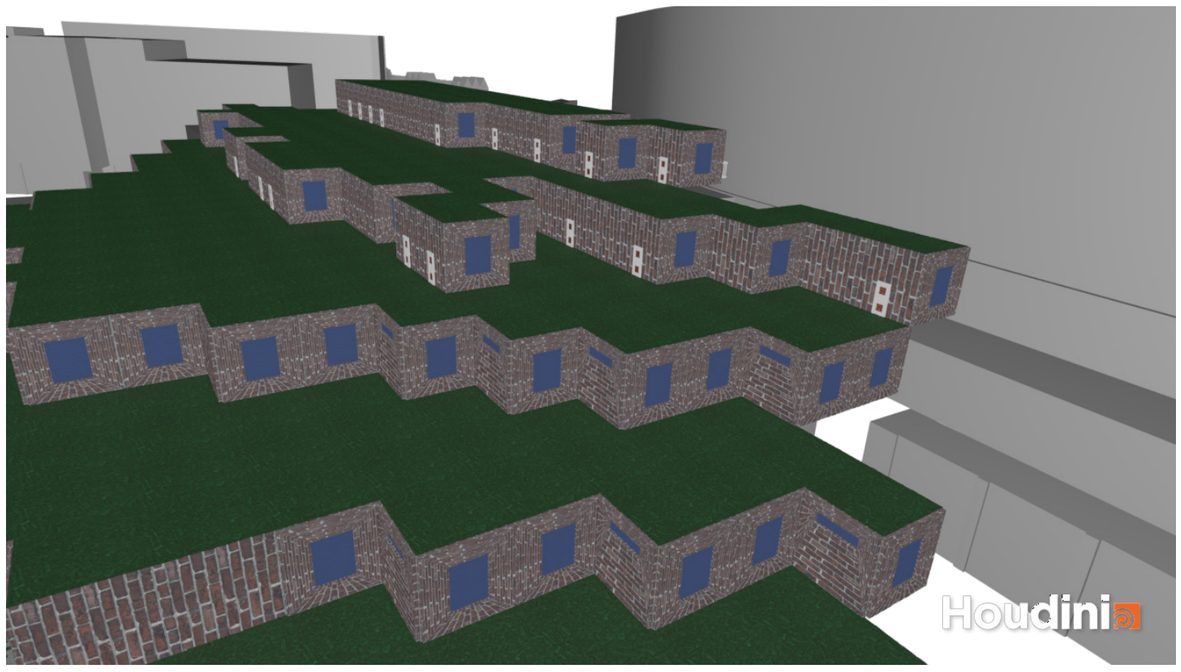
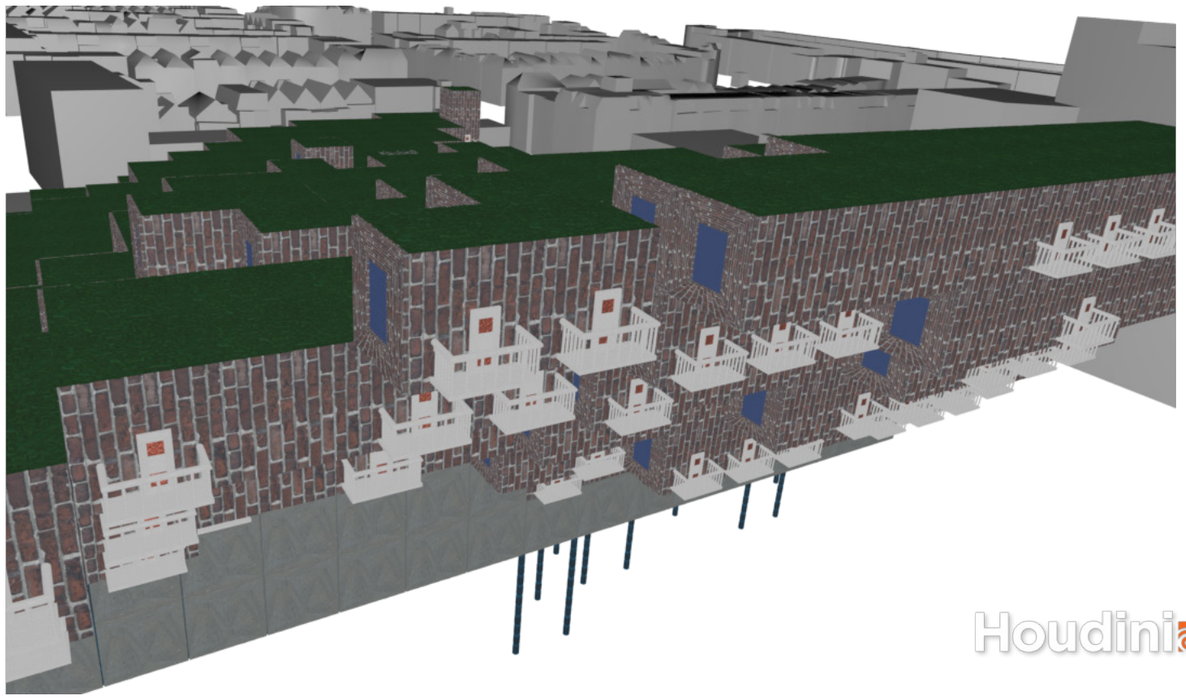


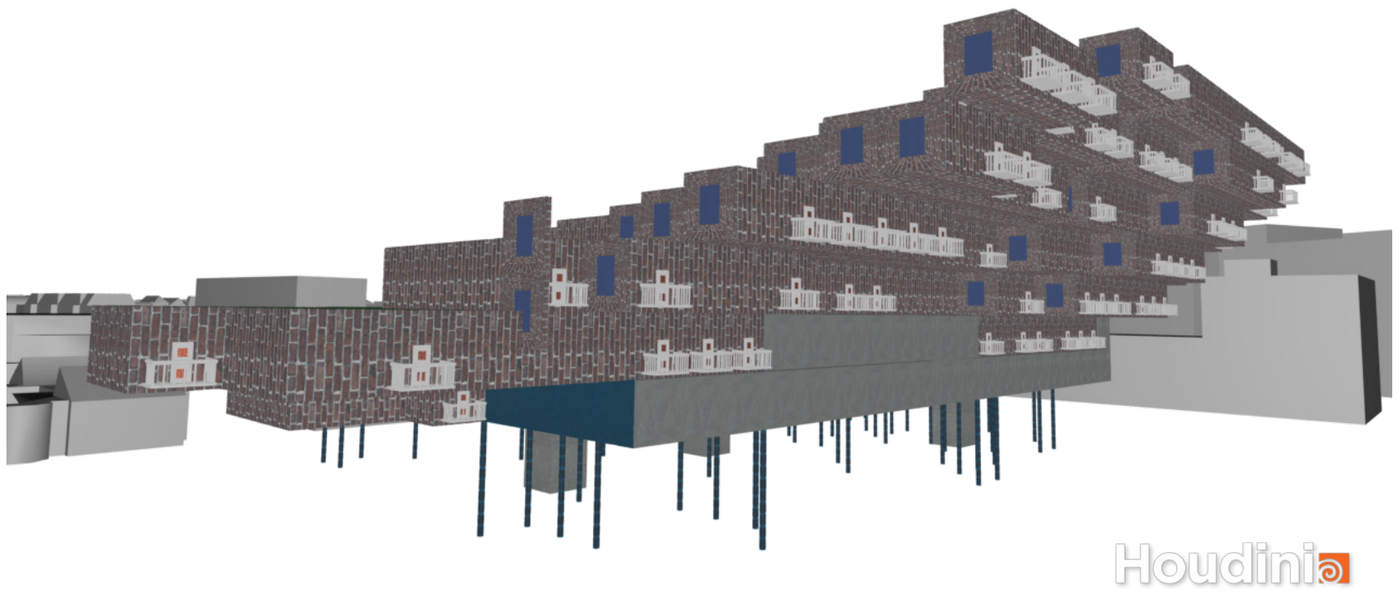
Fig. 8. Random generation of blockiness value for every function.

# Implementation of facades




Houdini


Houdini