Det Teknisk-Naturvidenskabelige Fakultet Første Studieår
AALBORG UNIVERSITET
Arkitektur Og Design

MATEMATIK OG FORM

5 March 2012 - Lecture GH2 (in English)
Matrices and Transformations in Grasshopper

Group 2	12:30-13:30	Lecture at Auditorium 7
		Lecturer: Dario Parigi, Jesper Christensen
	13:30-15:15	Task Check at group room
		Teachers: Martin Raussen, Jesper Christensen
Group 1	13:30-14:30	Lecture at Auditorium 7
		Lecturer: Dario Parigi, Mathias Jensen
	14:30-16:15	Task Check at group room
		Teachers: Dario Parigi, Mathias Jensen

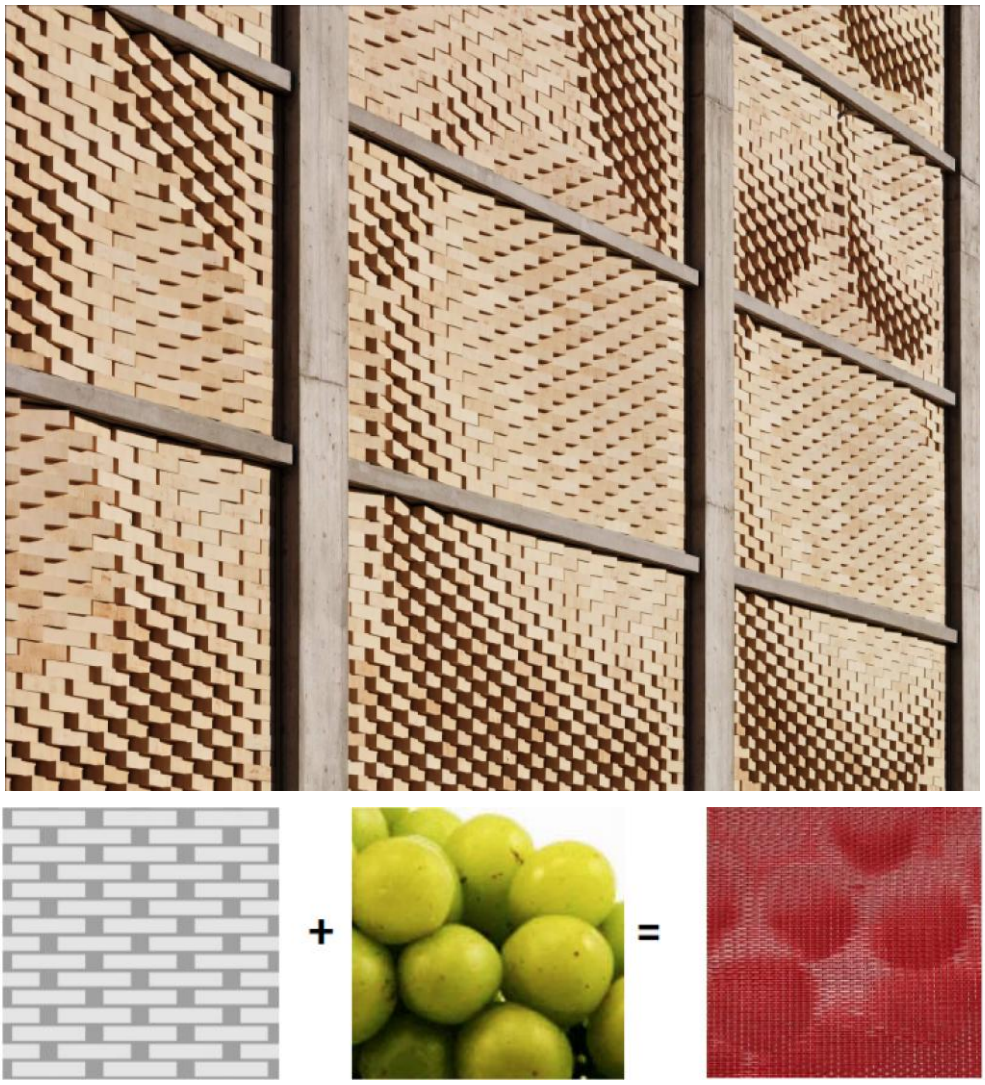
Aims and contents:
The lecture goal is to perform transformations in Grasshopper (moving, rotating, projecting, scaling) using the built-in components. It will be shown how to reveal the transformation matrices behind those operations. Before the task check it will be explained how grasshopper manages data in "tree structures".

Lecture schedule
-Transformations in Grasshopper
-Trees data structure

Tasks
group room task: see document

Literature
R. Issa, Essential Mathematics for computational design, 2nd ed. (pages 16-20)
Woojae Sung, Grasshopper Learning Material (pages 10-11)
Grasshopper Primer_Second Edition_090323 (pages 36-39)

PARAMETRIC WALL



What you will learn

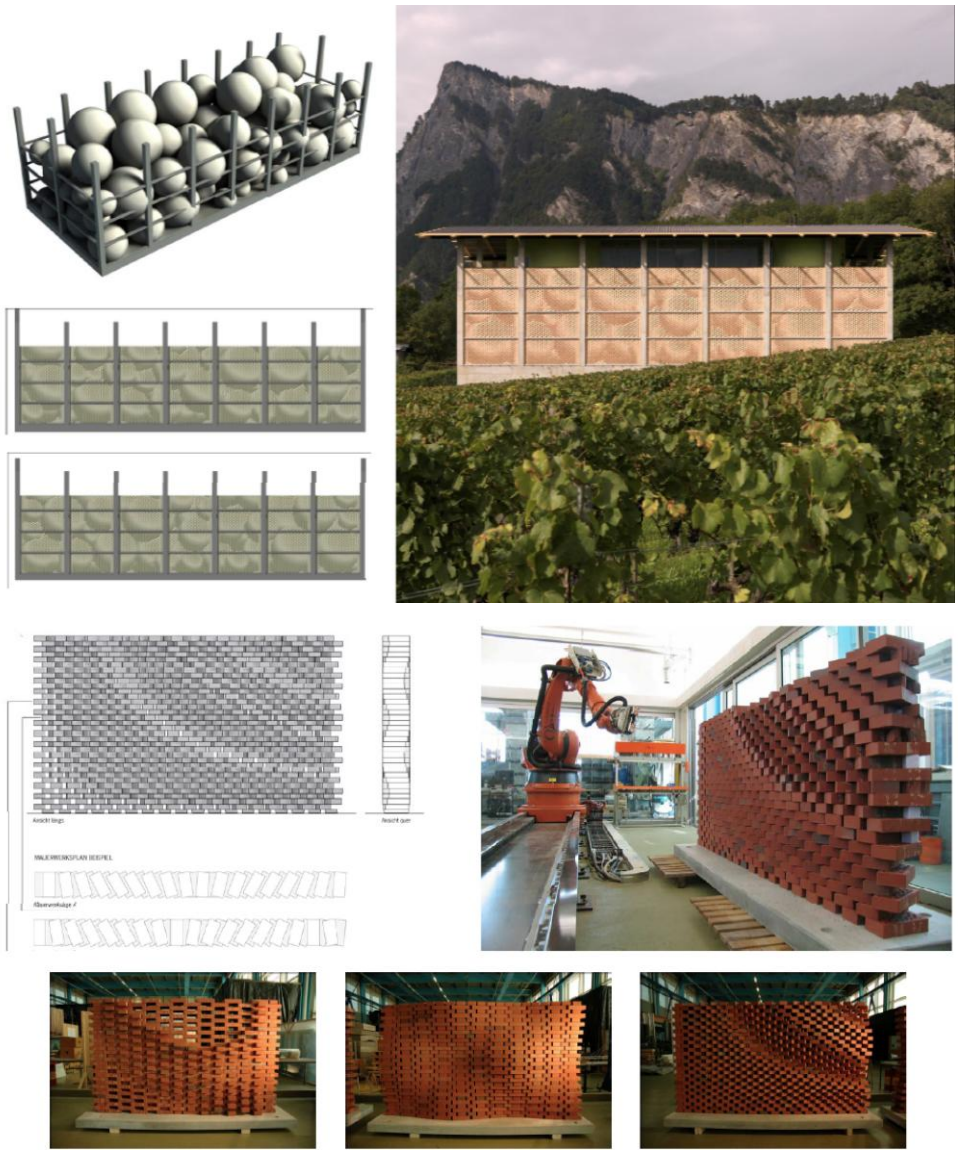
1. How to handle grasshopper “trees data structure”
2. To parametrically define relations between large set of objects

List of relevant components used

Recgrid
Tree branch
Flip matrix
Cull
Rotate3d
Image Sampler

Introduction

This exercise is inspired by the project for a winery in Gautenbein, Switzerland by Gramazio & Kohler, architects and researchers at ETH Zurich.
http://www.dfab.arch.ethz.ch/web/d/forschung/52.html
The “parametric wall”, where bricks position is defined by the geometric relations between bricks dimensions, and their rotation around the vertical axis is parametrically defined to recreate the image of grapes on the winery´s wall, characterizes the project.
In this exercise we are going to replicate the architects idea by using a picture of grapes to control the rotation of the bricks.



Vinery in Gautenbein, Switzerland. Arch. Gramazio & Kohler

Introduction to "trees data structure"

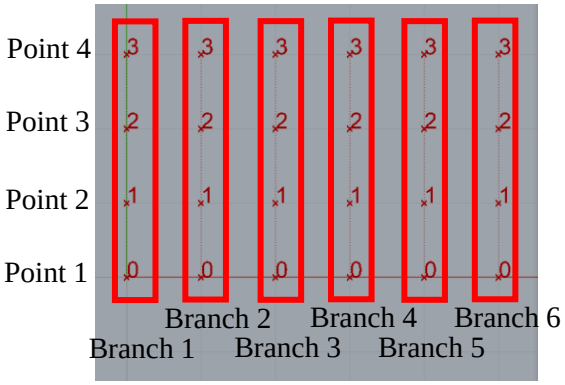
See also page 36-39 of the "grasshopper Primer"

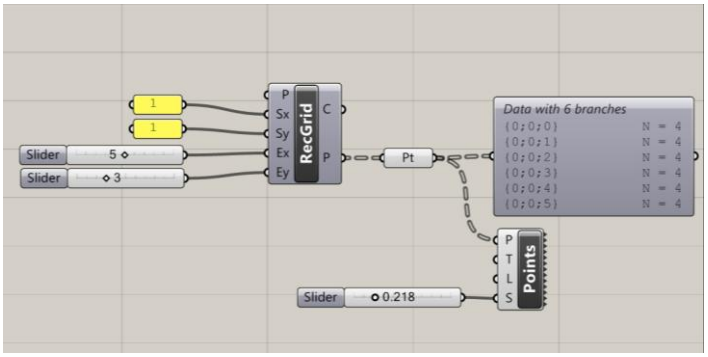
Grasshopper organize data with "trees data structures": data of any type (geometry, numbers, etc…) is stored in hierarchical branches you can navigate through by means of "paths",

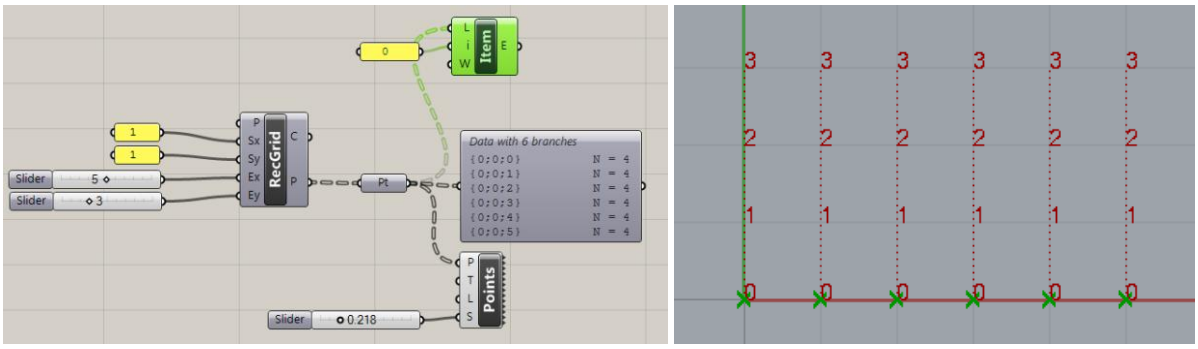
To introduce the concept, we use a rectangular grid of points with the component **recgrid** (vector>grid>rectangular), that creates a 2d grid with rectangular cells. Its inputs allows to define the size and number of cells, and the output P collects the points at grid corners.

If we use as inputs the values from the below screenshot (the sliders should be of the type integer when used to define the Number of grid cells in x and y direction), a set of 24 points at grid corners is created (six points for each of the 4 rows).

To visualize them, preview off the RecGrid component, collect the points from the P output into a Point component (Params>Geometry) and connect it to a Point list component. You can observe that the indexes that identify the points are not continuous from 0 to 23 for the 24 points, but they start over from 0 to 3 in each column of 4 points. In fact Grasshopper has automatically organized the points into trees data structure. We use a Param Viewer component (Params>Util) to visualize the details: its panel tells us that the points are organized in 6 branches, each branch is characterized by a numeric path on the left column (example:{0;0;0}) and contains 4 points (N=4). (If you double click on the panel a graphic rendition of the structure is also provided).

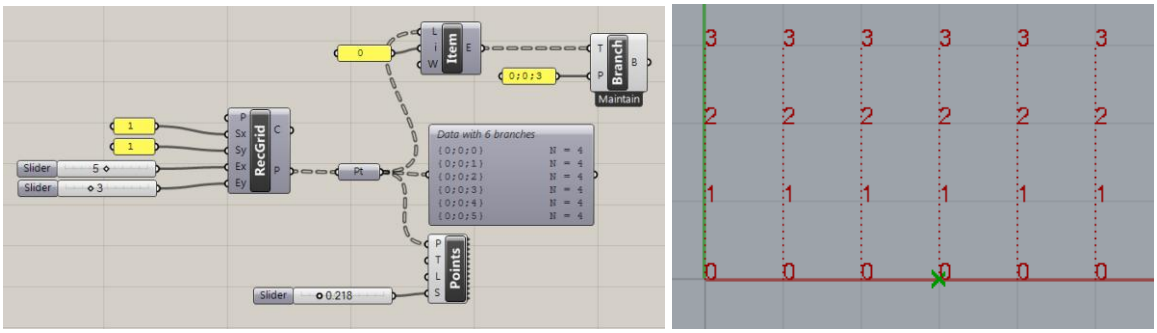




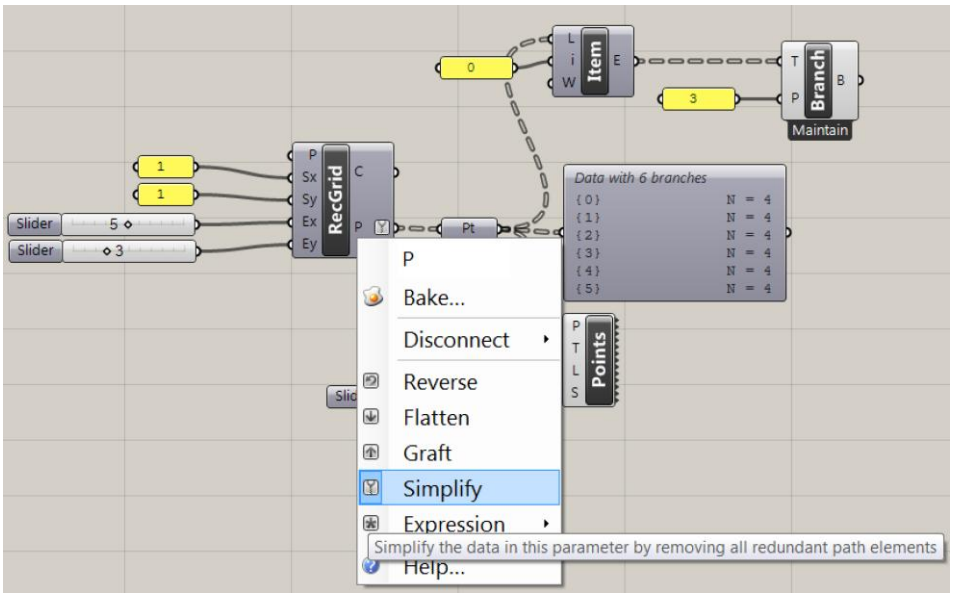


The indexes that identify the points start over and are independent inside each branch, therefore if we use a List Item component with the 0 index in the "i" input, it will identify at once the first point in all the branches.

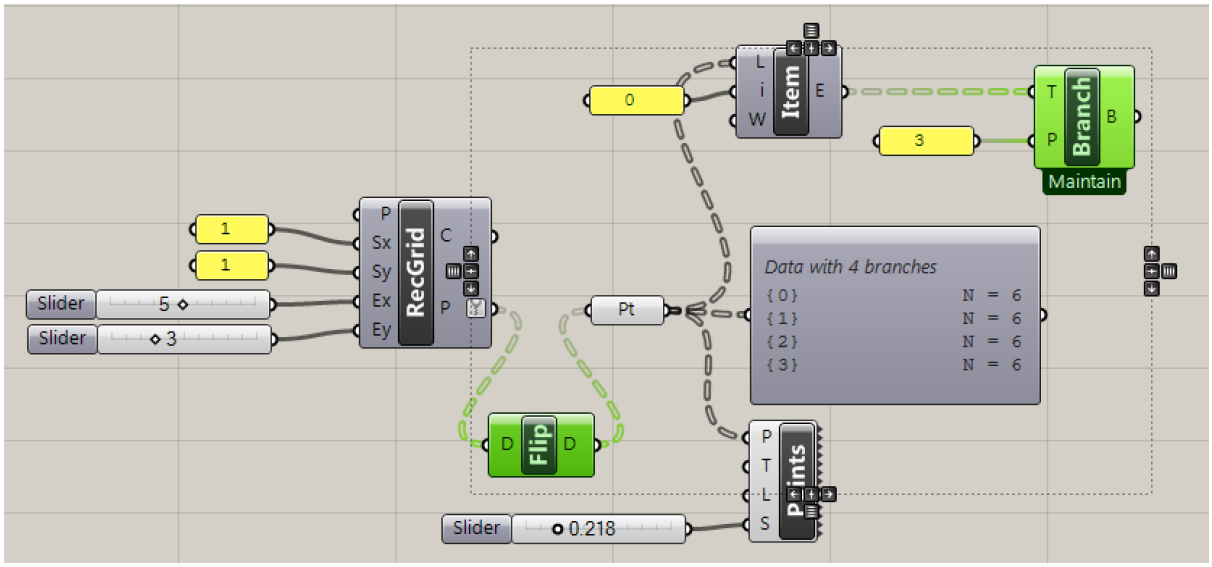
When data is organized in branches, we can also pick items just from single branches. We use a component Tree Branch (sets>Tree) and provide the correspondent branch path (that you can read between curly brackets in the Param Viewer panel) to select a single branch (in this case we pick branch 4, with the path 0;0;3).

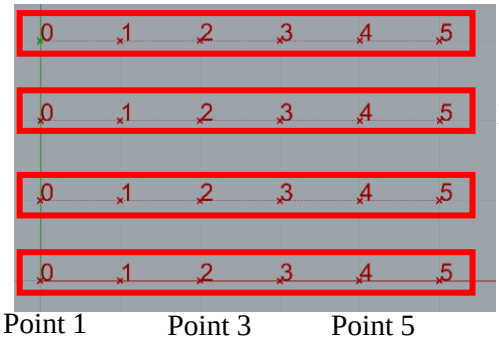


There is a way to simplify the "path" address. In the above example you can simplify and remove unnecessary or redundant path elements right clicking on the P output of RecGrid component and select "simplify". A small icon will appear besides P, and the path will be simplified by removing all the unnecessary paths (0;0;0 becomes 0 / 0;0;1 becomes 1 / etc..)

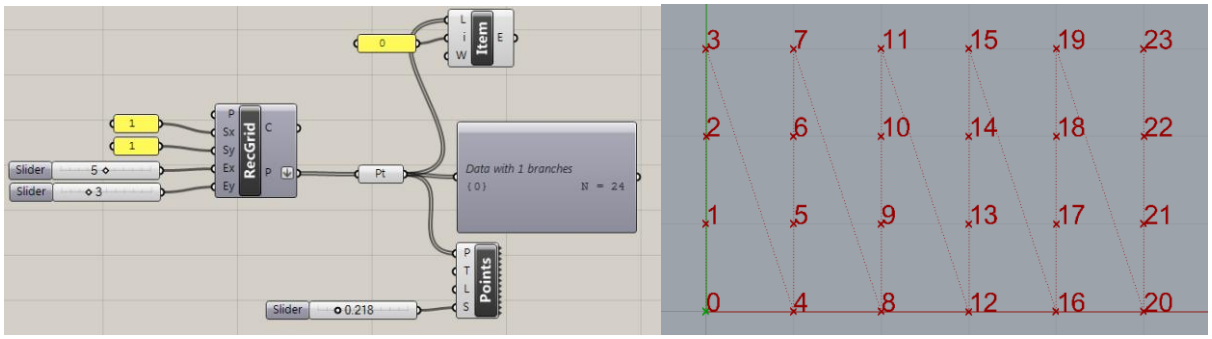


With the component **flip matrix** (sets>tree) it is possible to "flip" the tree data structure, and the points will be organized into 4 branches containing 6 points each (before “flipping” , points were organized in 6 branches containing 4 points each).

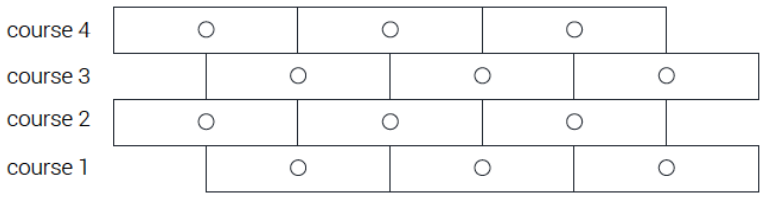


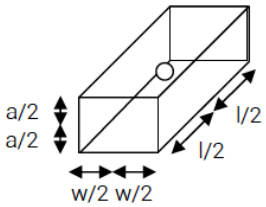


It is possible to eliminate at once all the branches by selecting the option “Flatten” that appears by right clicking on the P output from Recgrid. You can delete now the “Flip matrix” and “tree branch” component. Points are now ordered in a single list and all the branches are removed



Procedure for creating the parametric wall





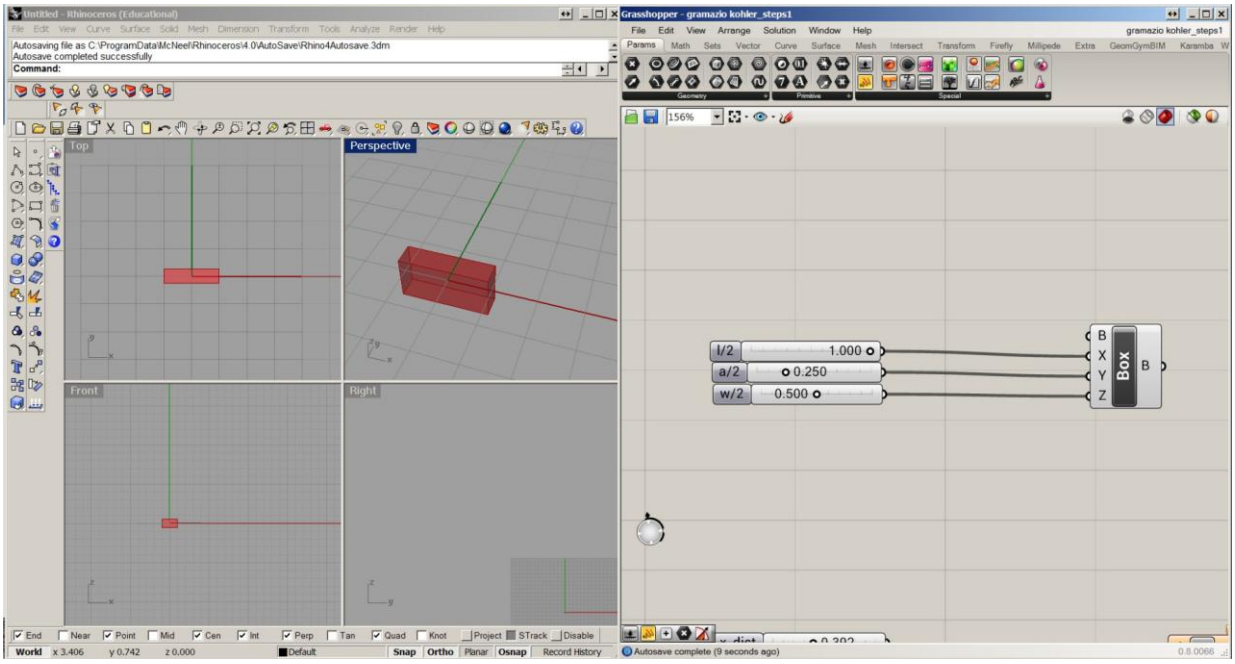
To define a parametric wall, the bricks position should be based on the brick dimensions (width **w**, height **a**, length **l**).

The geometry of the brick is created with the component **Centerbox**. The B input is the central point of the brick. As the dimensions are measured form the center point, we should define with sliders the following three parameters for the X, Y, Z inputs:

1) half of the width (w) of the brick

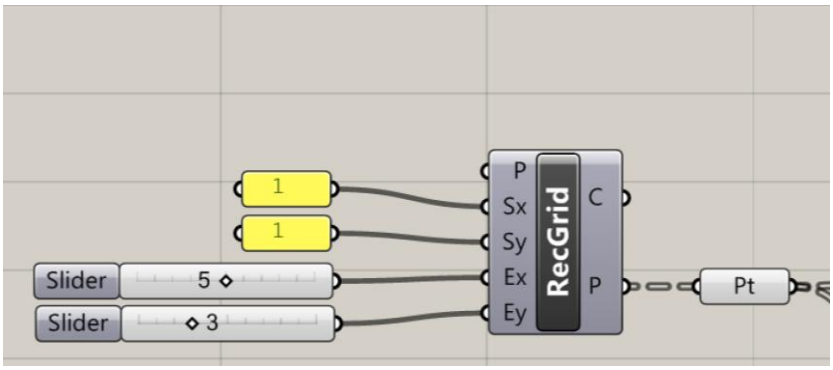
2) half of the height (a) of the brick

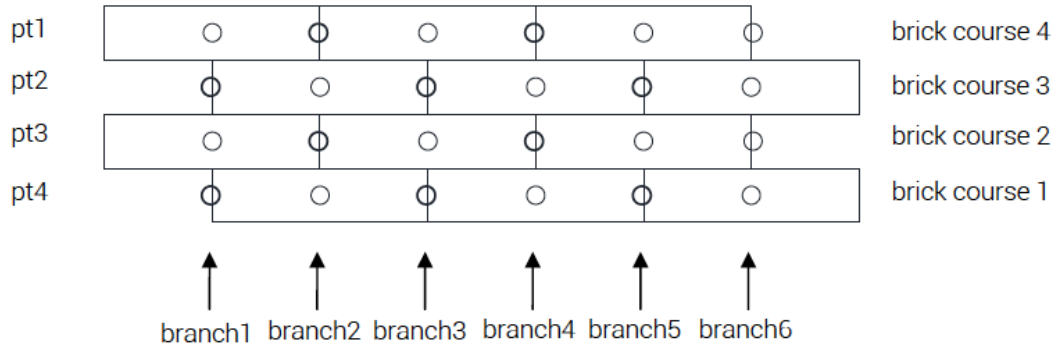
3) half of the length (l) of the brick



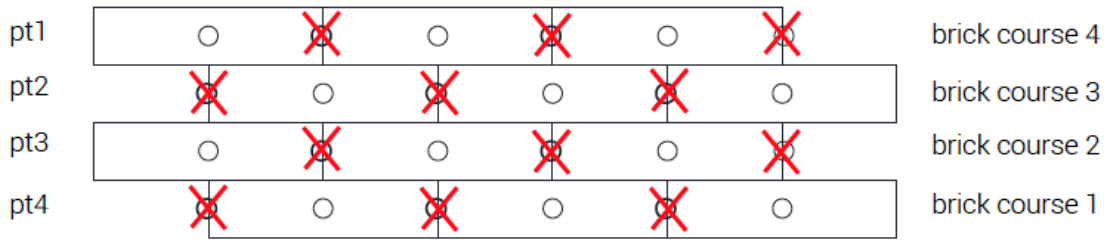
We create now a grid of points that represent the central points of the bricks in the wall. The spacing of the grid of points should be based on the dimensions of the bricks, so that the same wall definition can be used with different bricks dimensions.

We start from the **RecGrid** component and its inputs we used in the previous "Introduction to trees data structure" chapter (you can remove the remaining components used in that chapter).

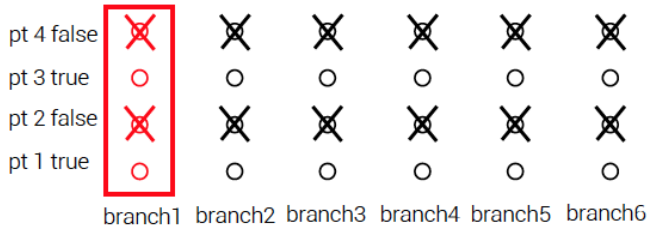


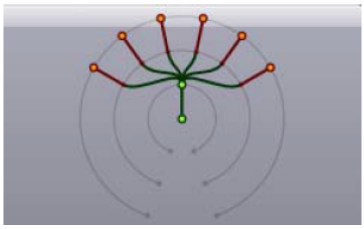


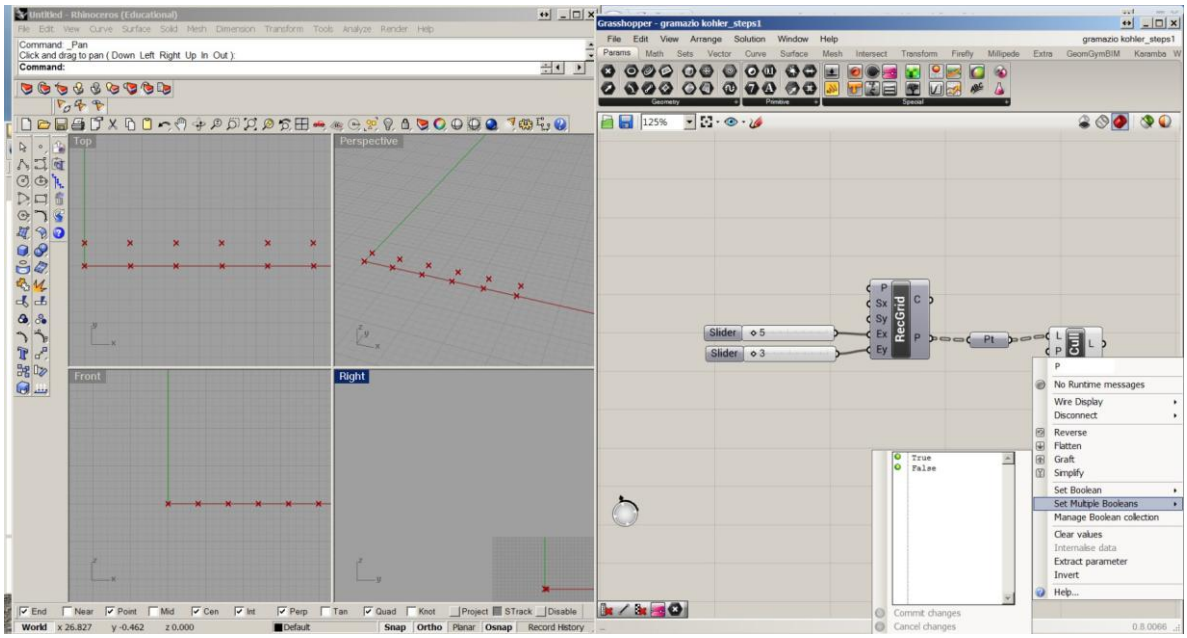
From the P grid of points of the **RecGrid** component we need to selectively remove the points that are not the central point of a brick, the one marked with red crosses below:



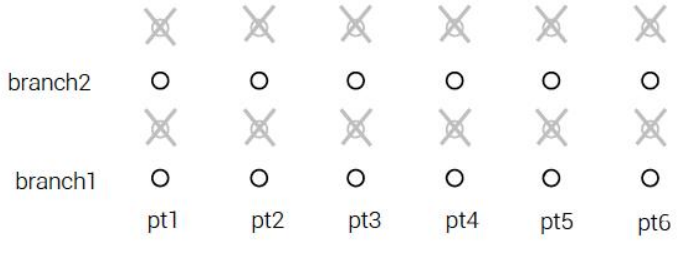
With the method proposed here, we will have to operate separately on the odd and even courses. First we will operate on the odd courses, and to isolate the odd courses we use a cull pattern (sets>sequence) component with the pattern "true; false" (right click on P > set multiple booleans), that removes the points in each branch according to the repeating pattern (true , false) where "false" delete the point, and true keeps the point, and has the effect to remove the points of the even courses.

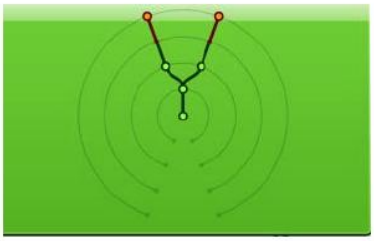




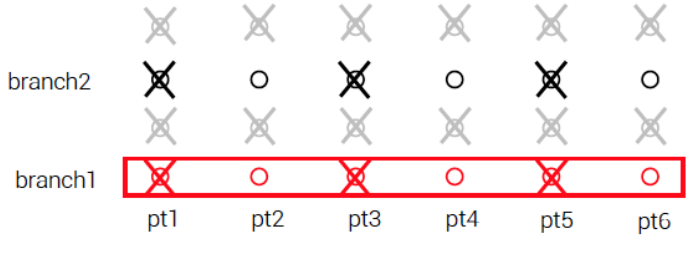


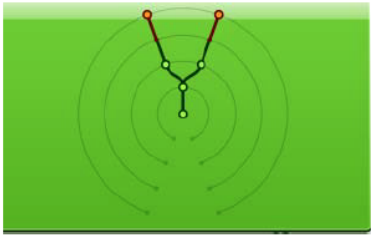
The next step is to eliminate among the points sitting in the odd courses just isolated, the points that are not the center points of a brick. To do so we need first to flip the tree data structure: points will be ordered in branches that each contains the points of an odd course.





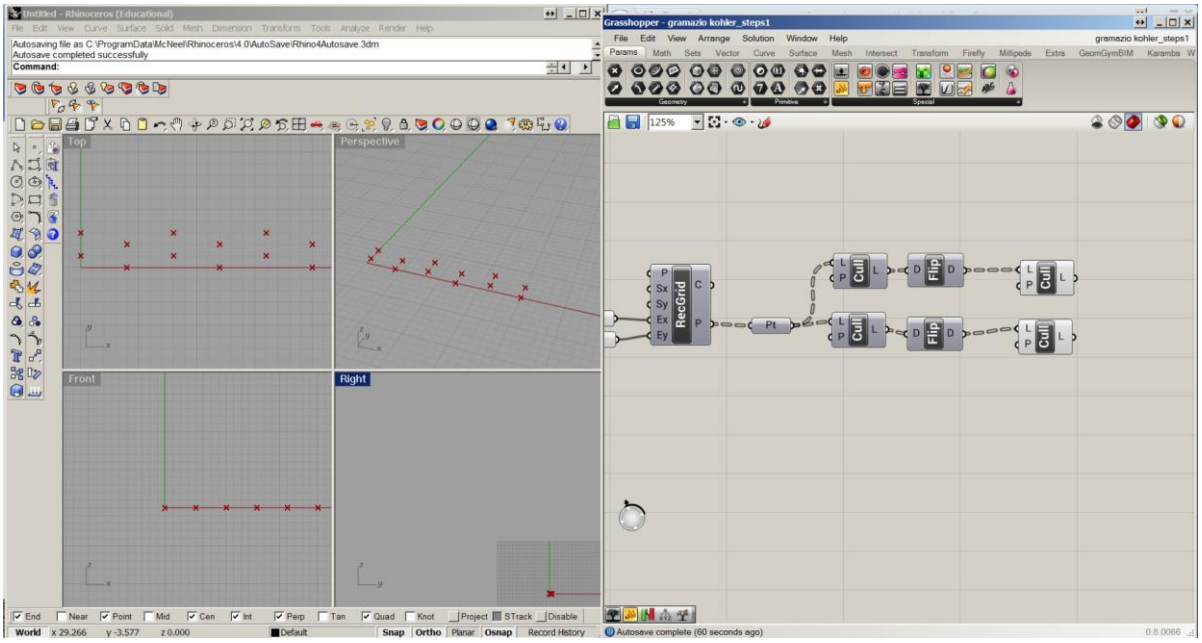
Then we apply a cull pattern that eliminate the points inside each branch with a pattern "false; true". With this step we remove the points that are not the central points of bricks, and what we have left are the center points of the bricks in the odd courses.



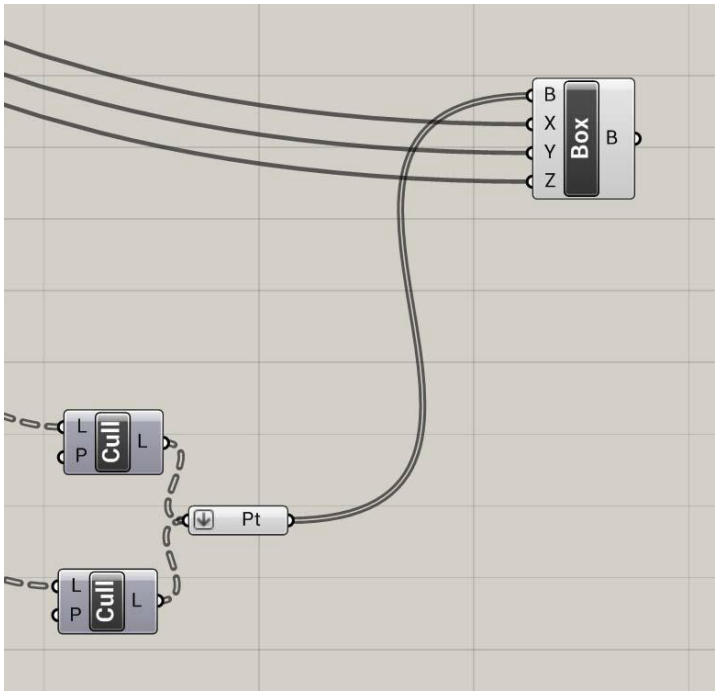


TO DO :

Use a similar procedure, but with different cull patterns, to create the center points of the bricks on the even courses.



Now you can merge the two sets containing the central points of the bricks of the odd and even courses in a **point** component (params>geometry) and flatten the list (right click on the point component) because we don't need anymore from now on the tree data structure.
Then connect the point to the box component that we created before.

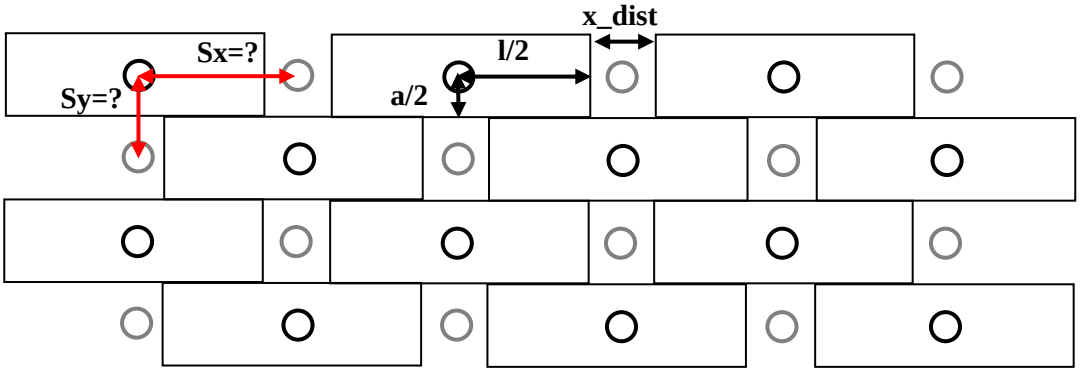


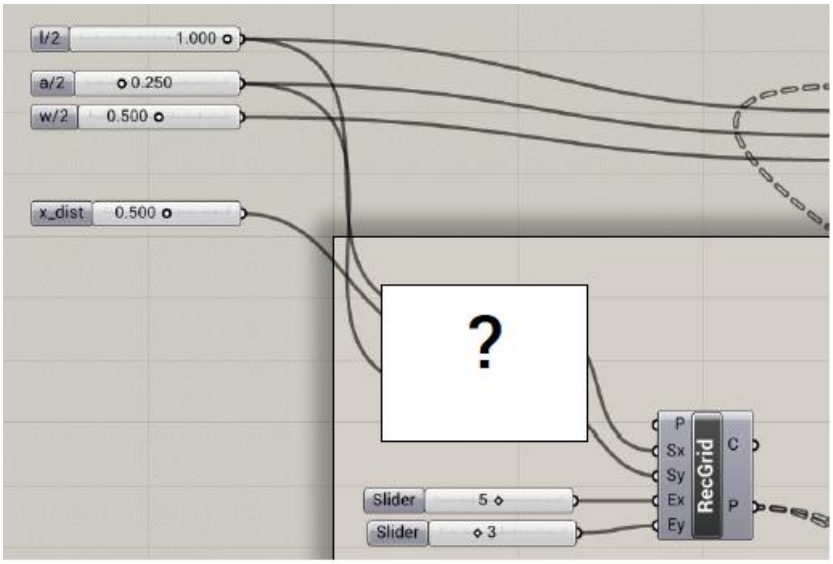
Now a brick is created in each point; still it is necessary to define the correct spacing of the initial grid to ensure the contact of the bricks in the y direction, and a spacing (**x_dist**) in the

x direction.

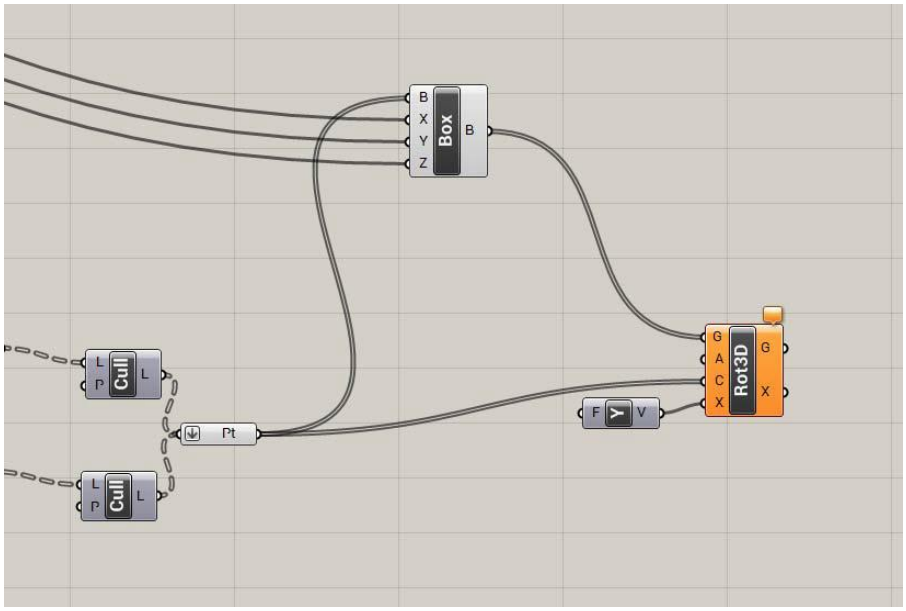
TO DO :

Define the spacing in the x and y direction of the RECGRID component (Sx and Sy), replacing the values that we gave at the beginning, using the dimensions of the brick and the x_dist value.





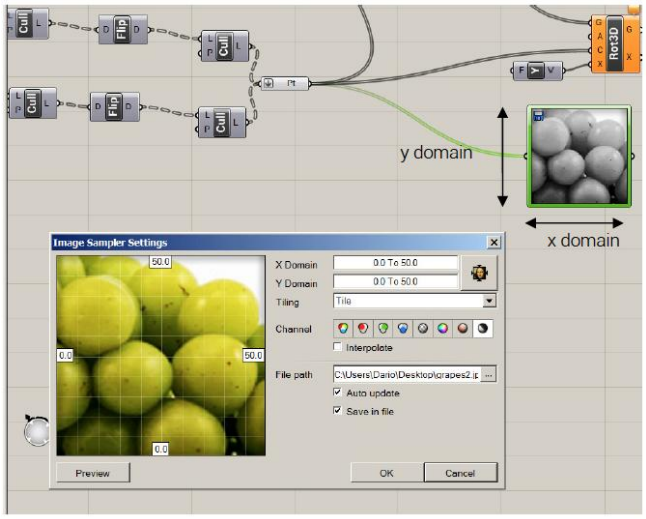
Now we are going to apply a rotation around the vertical axis of each brick, using a rotate 3d component (transform>euclidean), setting the boxes as geometry, a **unit y** vector as rotation axis, and the central points as center of rotation.

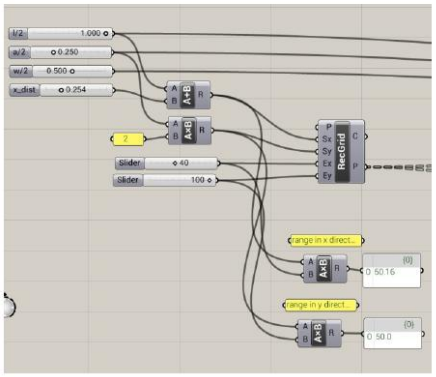


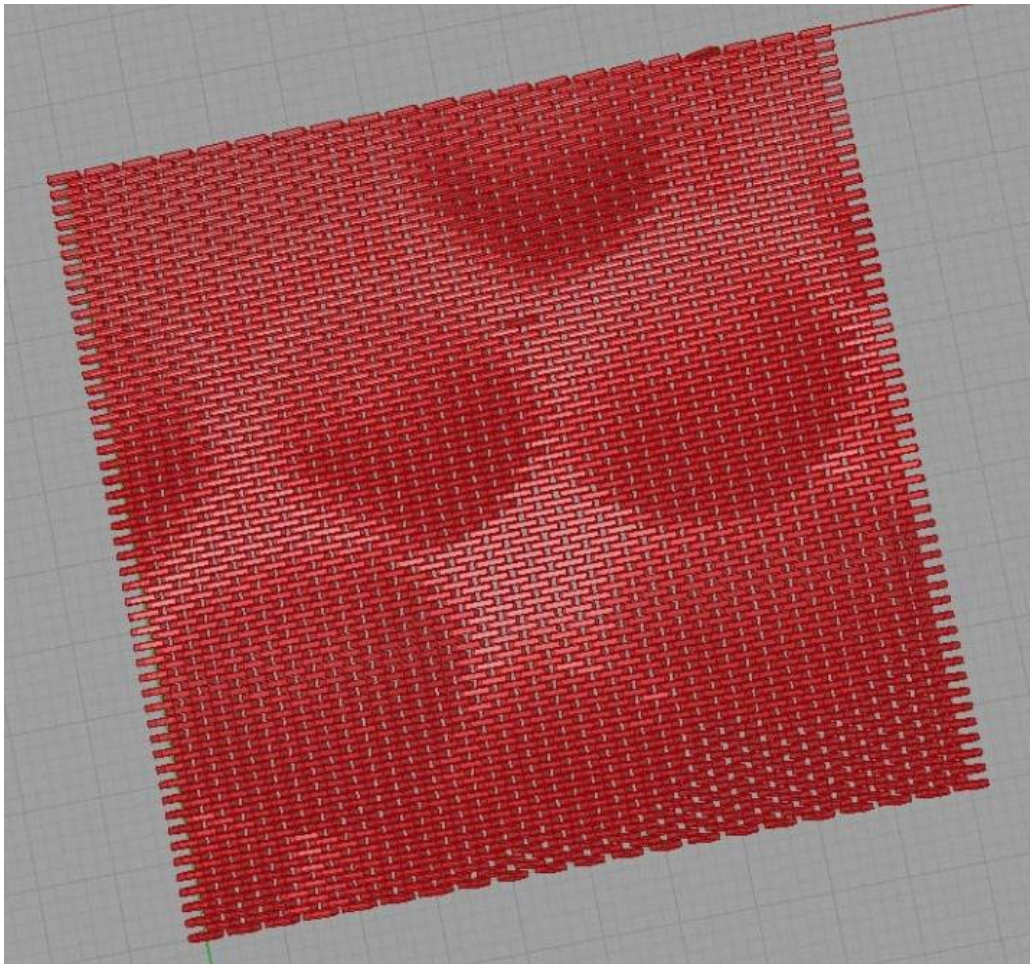
It is possible to input the rotation of each block in radians using values inherited from other sources. In our example we use an **image sample** (params>special) component to recreate the image of grapes. It accepts points as inputs, and it evaluates various characteristics of an image (saturation, brightness, etc..) in points positions given in the input . Right click on the component and enter settings to set the domain of the image (for example you can use "0 to 50" in both x and y domain), the channel (color brightness), and the image path. The X Y coordinates of the grid of points used as inputs should be included in the domain dimensions (0 to 50).

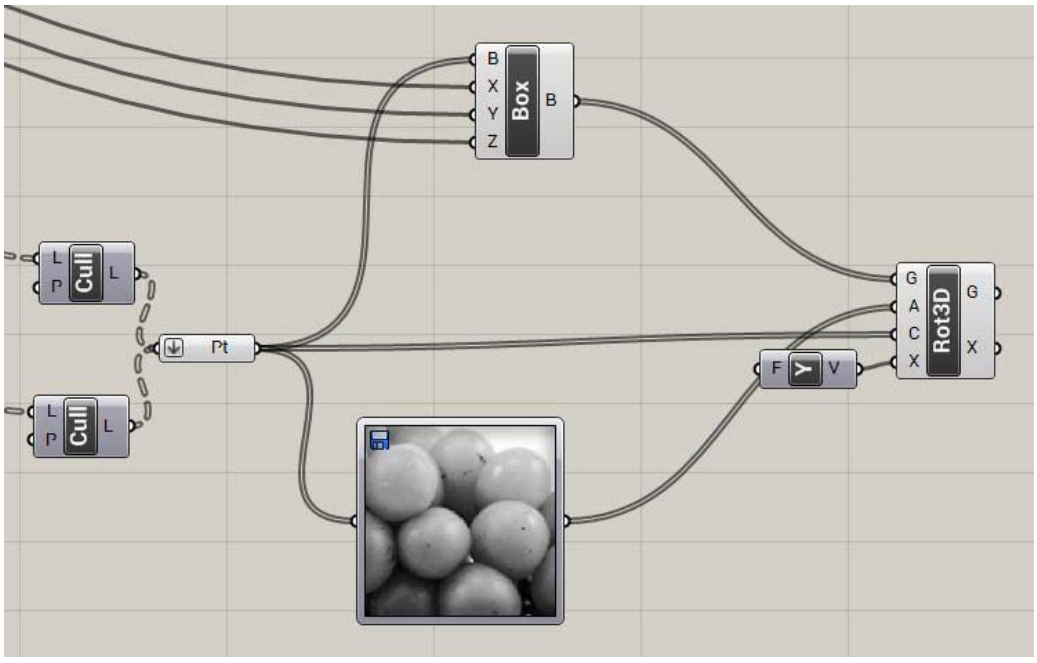
Use the output from the image sampler as values of the rotations. If we want the image to fill the entire wall, the dimension of the wall should match the domain of the image. To do so you should increase the number of grid cells in the RECGRID component in Ex and in Ey until the wall reaches the dimension of 50 in both x and y dimension to match the domain values of the image .

You can check the dimension of the wall by multiplying the spacing of the grid with the number of cells in both x and y direction








B
X
Y
Z
Box
B
L
P
Cull
L
Pt
L
P
Cull
L
G
A
C
X
Rot3D
G
X
F
Y
V

Note on transformation matrices in Grasshopper

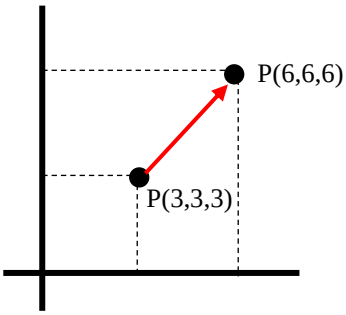
http://en.wikipedia.org/wiki/Transformation_matrix

Transformation matrices are used to represent linear transformations (example: rotation, scale, shear, translation, etc..). For linear transformations in the three dimensional space, transformation matrices have a dimension of 4x4.

Example: scaling transformation matrix applied to point P= (3,3,3)

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 3 \\ 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \\ 6 \\ 1 \end{bmatrix}$$

P'= (6,6,6)



In Grasshopper transformation matrices are "hidden" behind transformation components. However you can always visualize the transformation matrix by connecting the output X of the transformation component (scale, translate/move, rotate, shear …) to a matrix component (transform>util).