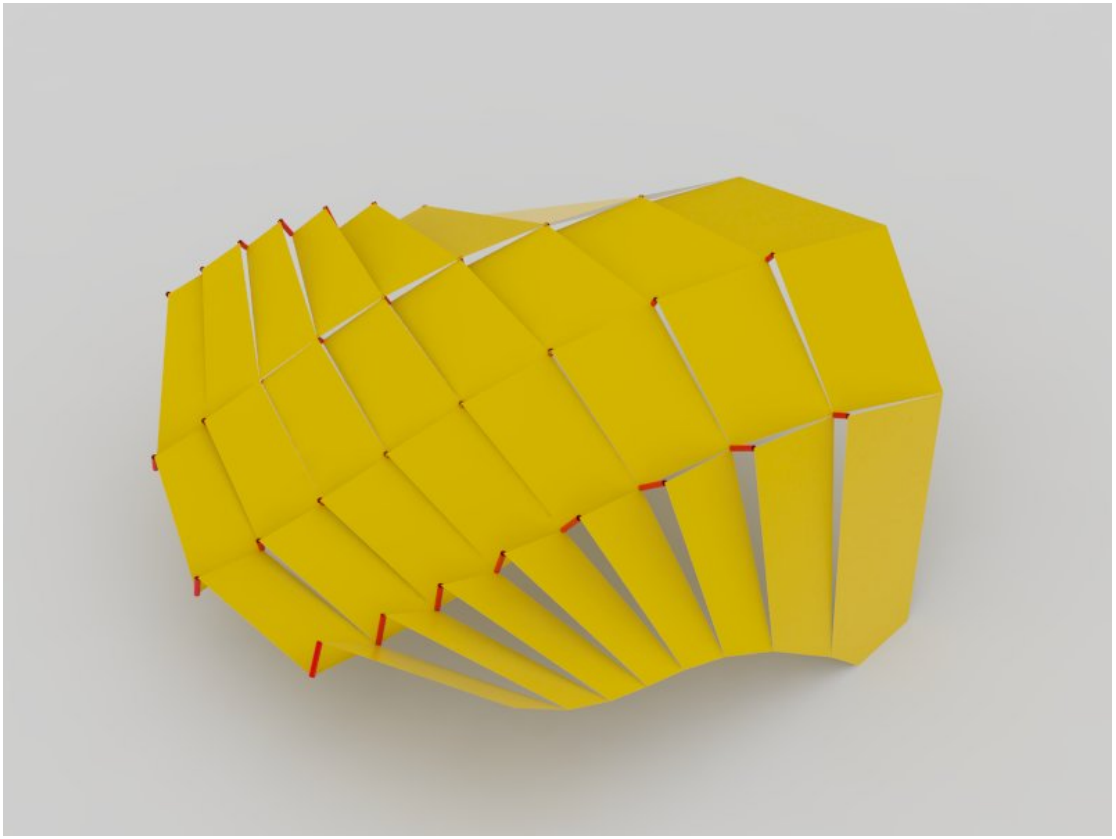
MATEMATIK OG FORM -8 may 2012 - MINIPROJECT 1 - grasshopper exercise

PLANAR QUADRILATERAL MESH



Map any free form surface with a set of discrete planar elements.
The method allows you to build surfaces by cutting elements from a single sheet of material.
However we will face the problem that in general four points do not lie in the same plane.
We will use this characteristic to create a surface that allows the light in from the slits created from one planar face to the other.

Starting from a group of points we want to create a surface made of planar quadrilateral elements.
In this example the group of points is simplified in a regular grid structure.

step1: execute station 1

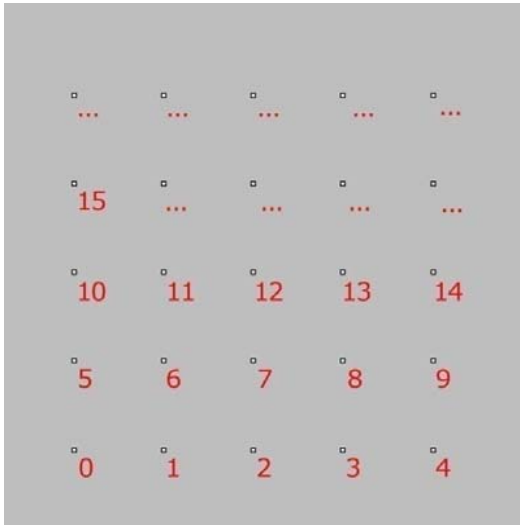
Created a grid of points in the domain (0,1) for both x and y coordinates

step 2: execute station 2

Using the previously created grid of points to create a similar grid of point along a three dimensional surface using the component "evaluate surface" (surface> analysis)

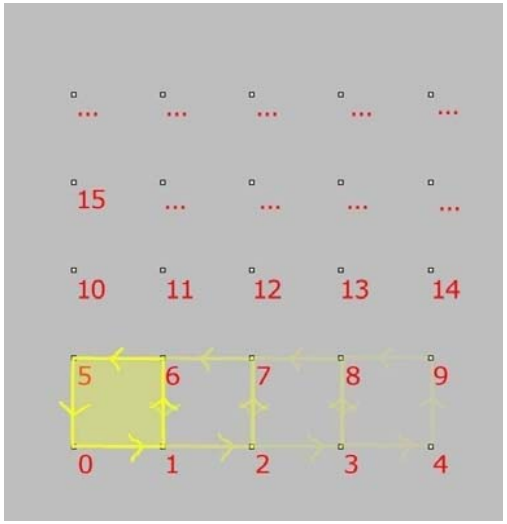
step3 : understanding the indexing of the points

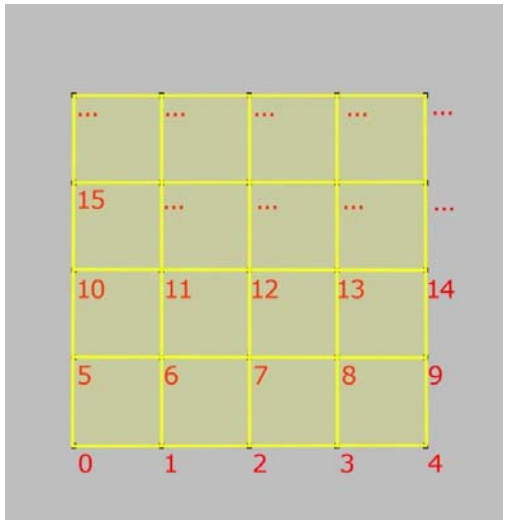
to create the planar quadrilateral mesh we should define the four vertices of each element.
If we look at the point grid, we see that there is an index number for each point in the grid. We know each point by its index number instead of coordinates in order to deal with its topology.



To define mesh faces, we need to call every four corners that we assumed to be a face and put them together and give them to the <4 points surface> component to be able to make the mesh surface.

In a given point grid, a simple quadrant face defines by an order of points that if you connect by a curve, you can make a face. This curve starts from a point in the grid, goes to the next point, then goes to the same point of the next row and then goes to the back column point of that row, and by closing this curve, you see the first face of the mesh finds its shape. Here the first face has points [0,1,6,5] in its face definition. The second face has [1,2,7,6] and so on.





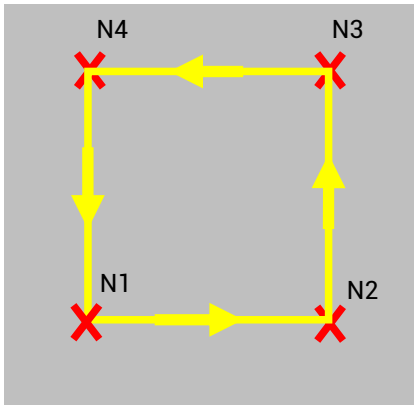
To define the whole quadrilateral elements, we should find the relation between these points in order to generate automatically, through a set of operations, the indexes of the four vertices of each element.

For each face , if the starting vertex has index n, the next vertex has index **n+1**. The following vertex has index **n+1+c**, where c is the number of columns.. The fourth vertex has index **n+c**. you can double-check this statement with the indexing of the first face in the above example grid, where **c**=5 and **n**=0.

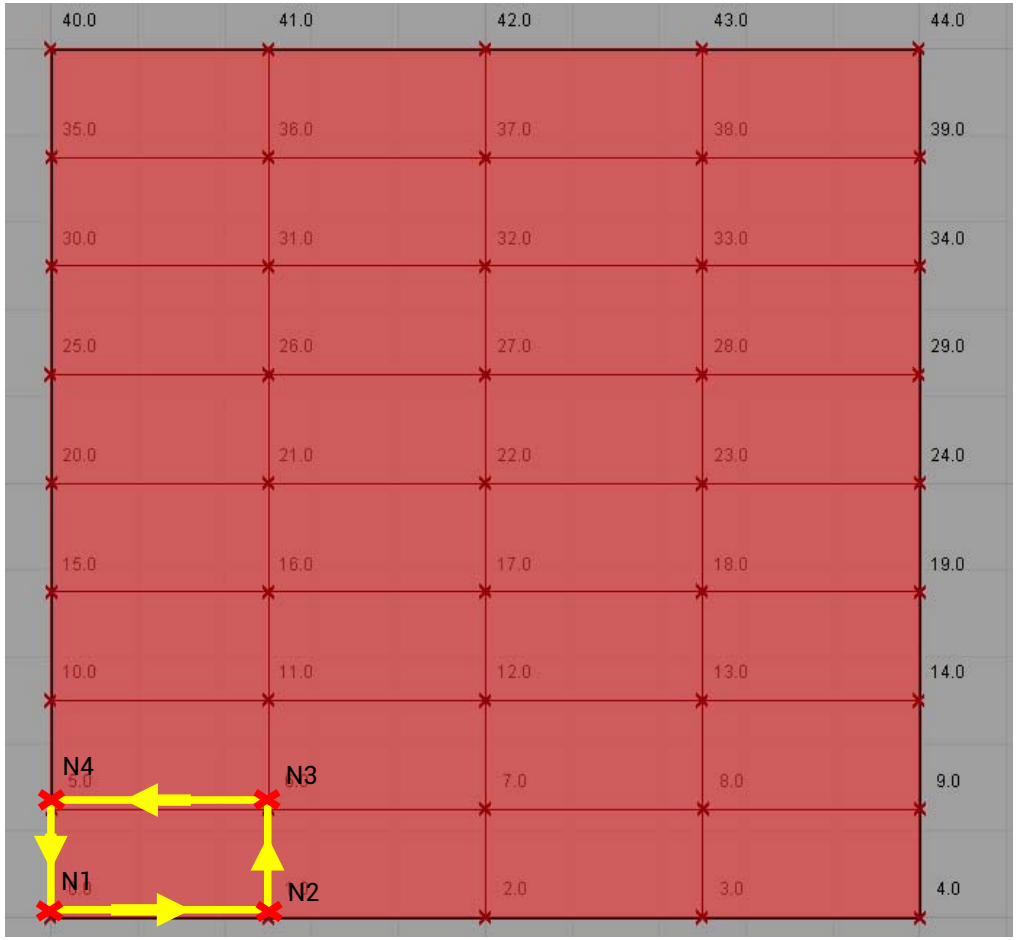
All we need is then a list of the indexes of the starting vertexes of each face, and then to perform the above simple operations to get the indexes of the remaining vertices.

If you look at the grid of points in the above example, you see that points on the last column and last row could not be start points of any face. So basically in the list of points, we need to omit the points of the last row and last column and then start to generate face matrices.

step 4: execute station 3



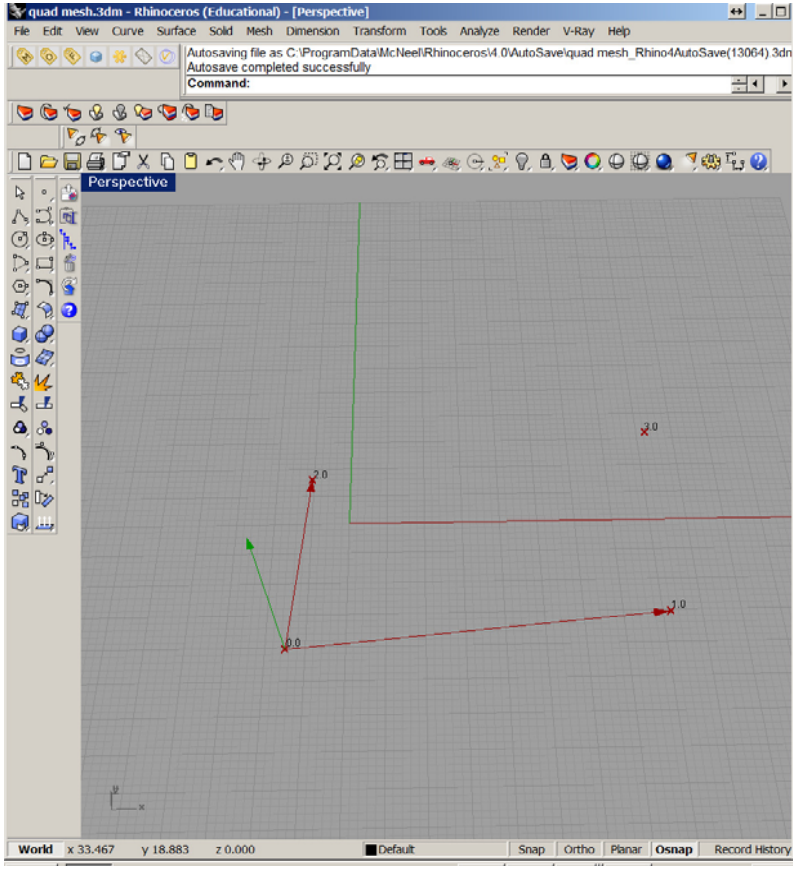
Exclude this column from the starting point indexes

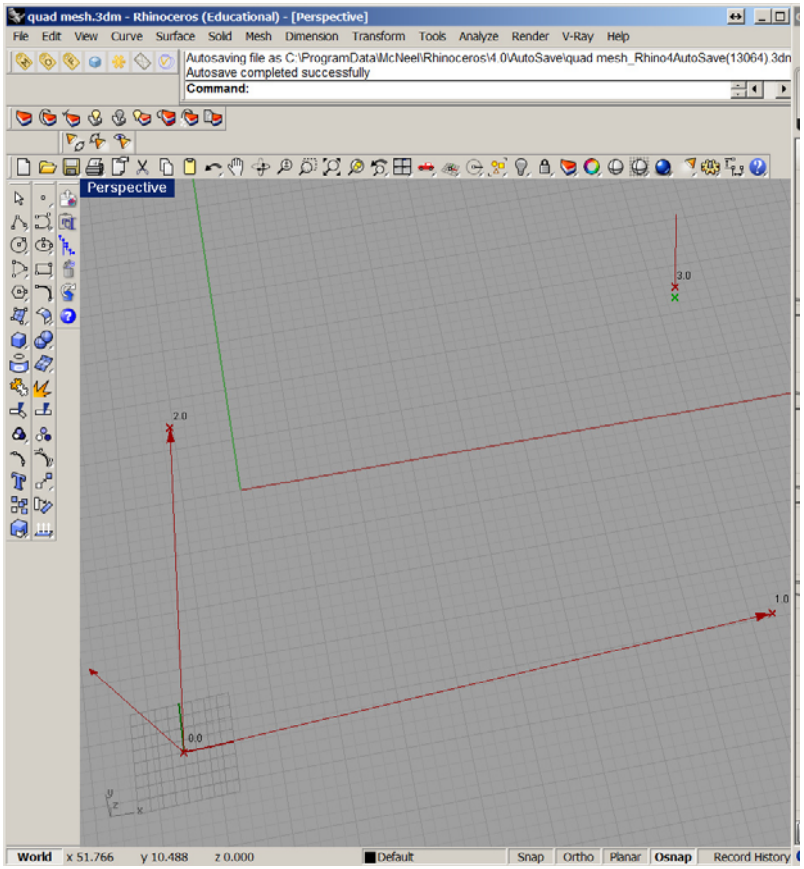


Exclude this row from the starting point indexes

step 5: execute station 4

In this stage we will execute the operations that will quickly allow us to define the quadrangular meshes . We will look at the operations in a single face, however the operations are applied to all the faces simultaneously, if we feed these operations with the set of vertexes just created in the previous step.
First we calculate the cross product between vector that goes from the starting vertex to the second vertex, and the vector that goes from the starting vertex to the fourth vertex.





Next we create a plane perpendicular to the vector obtained with the **cross product** with the component vector>plane>**plane normal,** having origin in the starting vertex. This plane passes through the starting vertex, and vertex 2 and 4. Vertex 3 in general does not lie on the plane, so if we used it to create our quadrilateral element we would obtain a non-planar element. Our goal is then to find the point closest to vertex 3 but lying on the plane.
To find this point we create a line passing through vertex 3, perpendicular to the plane, using the component curve>primitive>**line SDL**, that creates a line from its starting point and its direction (which direction should you use?), and we calculate the intersection between the line and the plane (intersect>mathematical>**line plane**). The result of the intersection is the new third vertex, and the last one we needed to create the planar quadrangular element.
Use the component surface>freeform>4 points surface to create the planar elements, connecting the stating vertex, the vertex 2, the vertex 3 lying on the plane, and the vertex 4.
You can also create a geometry that highlight the distance between the original vertex 3 and the vertex 3 lying on the plane, connecting the 2 points with a line, and then creating a pipe around it (surface.freeform>pipe). You can transfer the geometry in Rhino using the command **bake** on a specific component