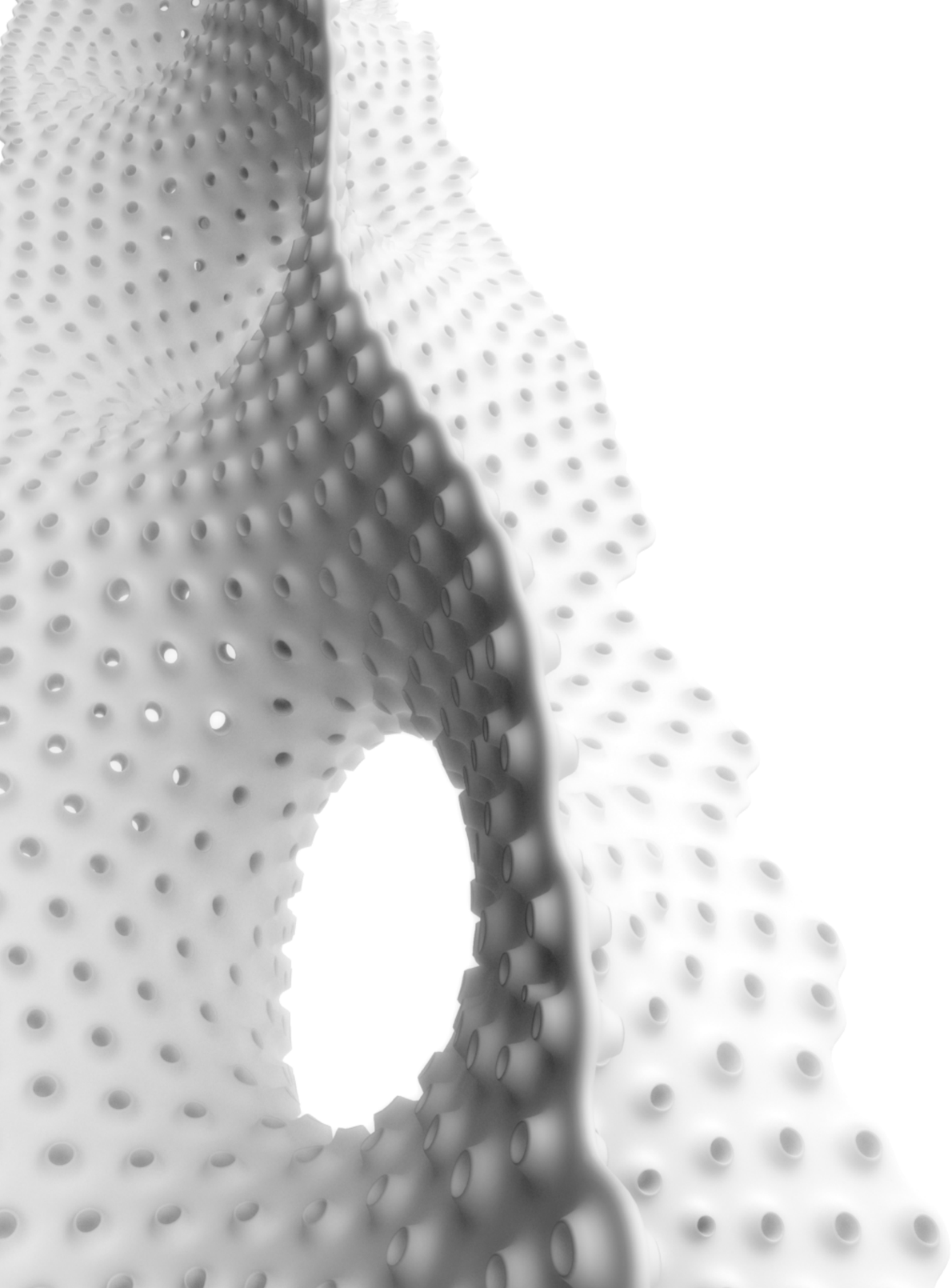


Basic course

Grasshopper

Vicente Soler

2013/14
TOOLS TRAINING
ESCUELA DE ARQUITECTURA
UNIVERSIDAD EUROPEA

Escuela de Arquitectura
Universidad Europea de Madrid
C/ Tajo s/n. Villaviciosa de Odón
28670 Madrid
arquitectura.uem.es

Autor Vicente Soler
Autor Óscar Liébana Carrasco
Director de la colección Óscar Liébana Carrasco
Diseño José Valero
ISBN-10: 84-695-9946-1
ISBN-13: 978-84-695-9946-4
Depósito Legal
Publicado simultaneamente en España
2013/2014

INDEX

WHAT IS TOOLS TRAINING?

Tools Training are courses for specific tools requiring constant upgrade for students in the School of Architecture (degree, postgraduate and vocational training) for a better academic progress of the subjects in their regulated course. Priority areas are the software of digital fabrication, parametric design, BIM and use of specific machinery of architecture workshops. They are made with professors linked to school and preferably alumni. The learning system will be held through practical examples on workshops. Students will bring their equipment and educational licenses shall be used. Digital documents as well as promotional videos of the courses will be made.

Oscar Liébana [@oliebana] [@BIMLabUEM]

Founder of BIM Lab UEM

INDEX

INTRODUCTION

This course will introduce the student to the basics of the Grasshopper plug-in for Rhino 3D. In most 3D modeling software the end user is able to automate the creation of geometry and other tasks using some form of scripting. Taking advantage of this feature requires programming knowledge. The Grasshopper plug-in exposes a visual programming interface, allowing the users to design algorithms by manipulating graphical elements, akin to drawing a flowchart, rather than typing textual code. Although it can be used to solve a wide range of problems, one of its main uses in architecture is the production of generative designs.

REQUIREMENTS To be able to follow the course, the student must already know how to use the Rhino 3D software. Students bringing their own laptops must be running Windows XP or higher. Windows 8 or 7 are recommended. It's not recommended that Windows is virtualized over another operating system (use Boot Camp rather than Parallels on MacOS).

Students must have the following software installed in their computer, preferably the English version:

Rhinoceros 5.0

Grasshopper 0.9.0064

Weaverbird 0.7.50.0

MeshEditTools 1.0.0.9

HOW TO USE THE .GH FILES THAT COME WITH THE WORKBOOK

Every exercise has a corresponding .gh file that you can open in Grasshopper. The geometric data referenced in these files has been internalized. This means that there is no need to open a .3dm Rhino file for them to work. To be able to change the geometry in Rhino, the internalized data has to be baked and referenced into the same parameter.

If you want to find out where a component is located in the tool bar, press and hold the control and alt keys while left clicking on the component.

INTERFACE

The Grasshopper interface consists of the following elements:

1 Menus: Similar to other software, contain options for operations like saving files, copy, paste, interface toggles and so on. The far right side of the menu allows you to switch between the opened files.

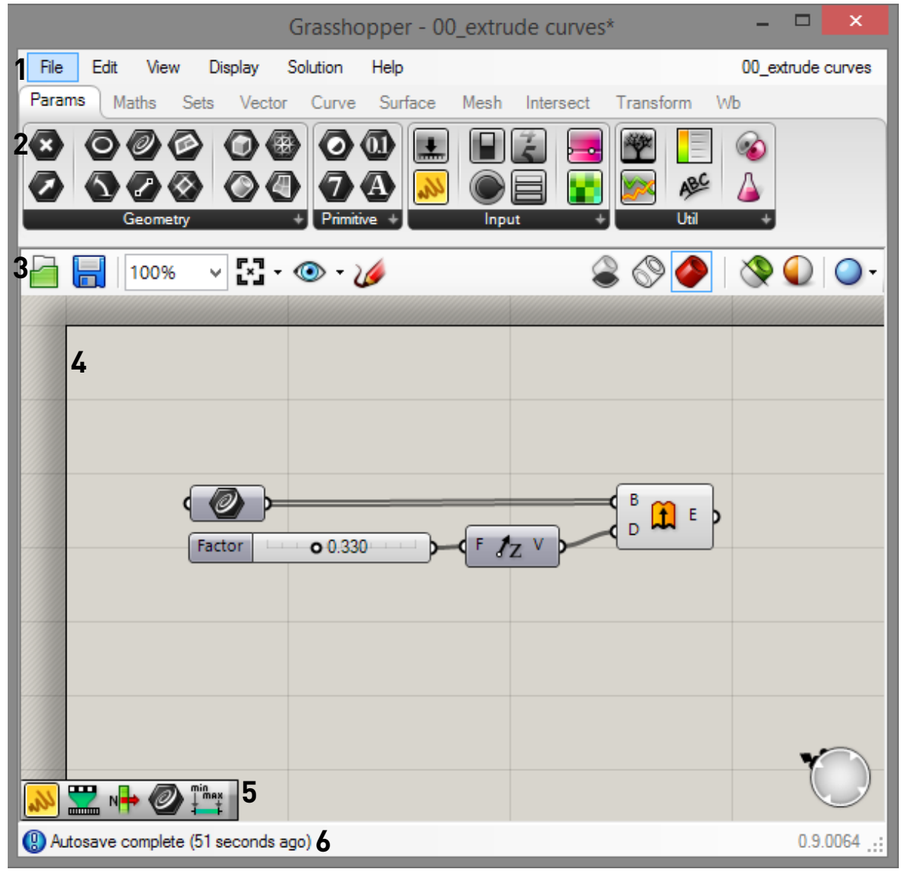
2 Component tabs: Contain all the elements that can be placed on the canvas to generate a solution.

3 Canvas tool-bar: Contain options to navigate through the canvas and to change how geometry is displayed in the view-point.

4 Canvas: The canvas is an infinitive two dimensional plane where you design your algorithm by linking elements from the components tab.

5 Canvas widgets: Expose information or adds functionality to the canvas. These can be enabled or disabled under the display menu.

6 Status bar: Display messages, for example the time it took a file to be processed. It also displays the version number.



TERMINOLOGY

Definition: Grasshopper files are called definitions.

Parameter: A parameter is any type of data. Parameter capsules contain information but will not modify it. This information might be stored inside the parameter capsule, it might come from another parameter or it might be referenced from geometry modeled in Rhino. In the canvas you might find parameters as separate capsules or as the inputs and outputs of a component.

Components: They combine the data connected to their inputs to generate new data. The results will be exposed on their outputs, to be used as input into another component.

Interface objects: Allow the user to input information directly into the canvas in an intuitive way using the mouse or keyboard.

Wires: They pass information from one component or parameter to another. Information always flows from left to right, from outputs to inputs.

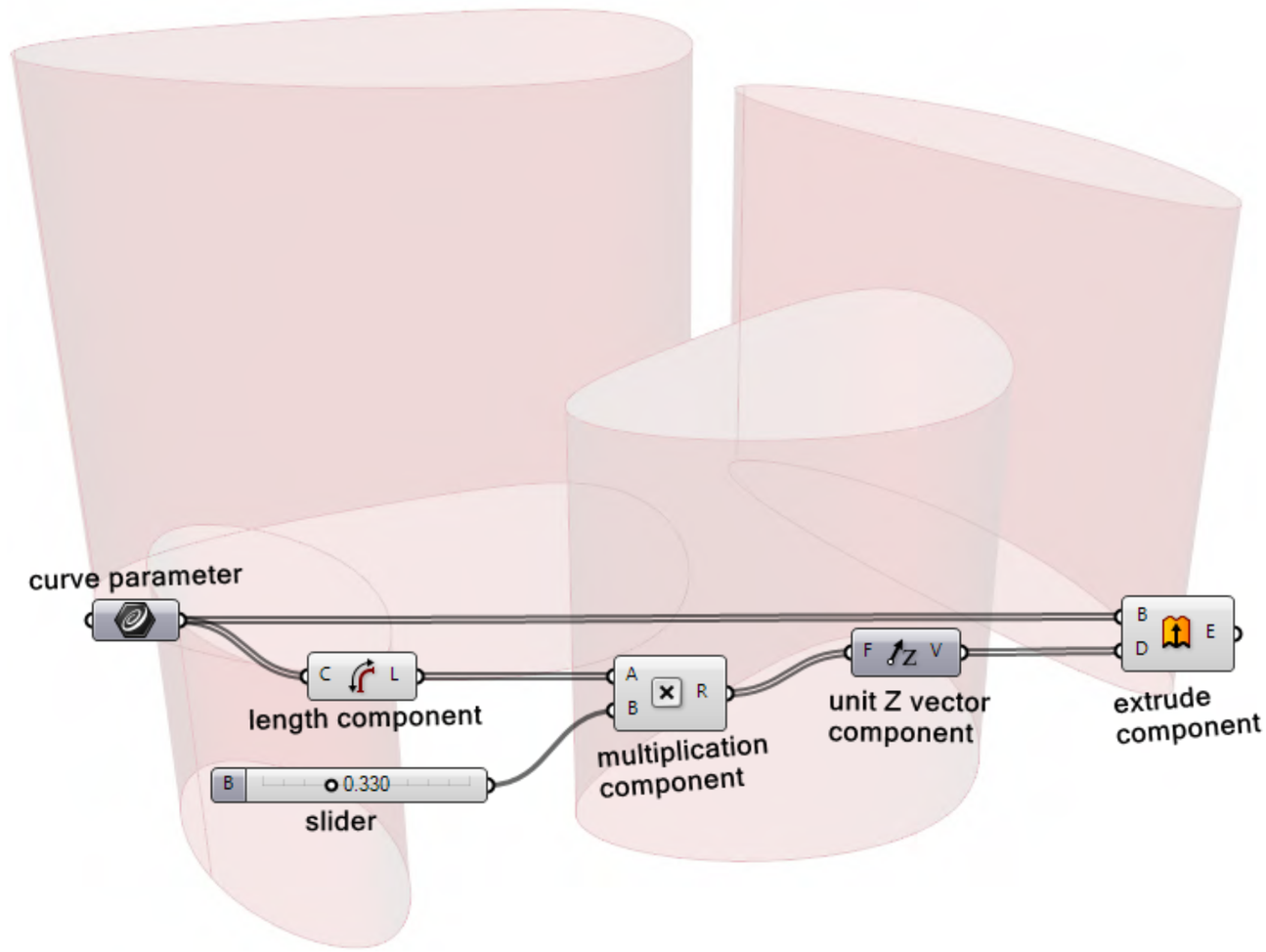
EXERCISES

EXERCISE 00 - INTRODUCTION

FILE: 00_extrude curves.gh

This exercise serves as a basic introduction to Grasshopper and its interface. A set of curves are extruded an amount that is proportional to the length of each curve.

The **curve** parameter references a curve drawn in Rhino. The **length** component calculates the length of a curve. The **multiplication** component multiplies the length of each curve by a given number. This number is given by a slider. A slider is an interface object that allows the user to quickly change a number by dragging the value with the mouse. The **unit Z vector** component creates a new vector that points vertically and has a magnitude of one unit. The previously calculated value is connected to it becoming the magnitude of the vector. In Cartesian coordinates vectors have the form of {x,y,z}. Since this vector is completely vertical, the output of this component will be {0,0,F}, where F is the previously calculated number. Finally, the **extrude** component extrudes the curves using the vectors for the direction and distance.



EXERCISES

EXERCISE 01 - GRID OF CIRCLES

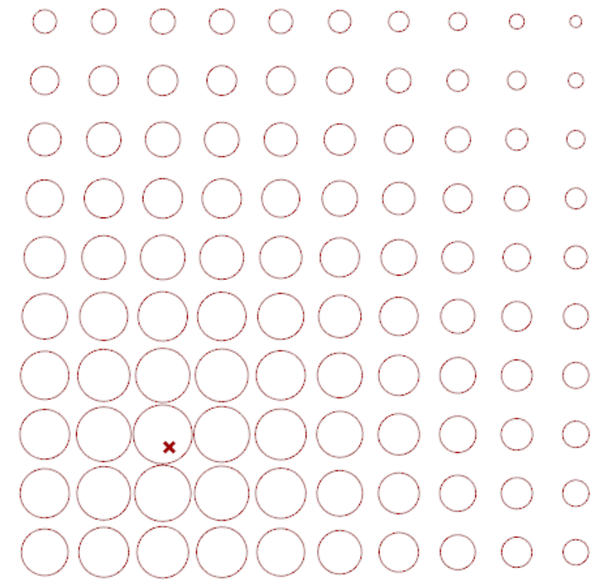
01.A - CREATE AN ATTRACTOR

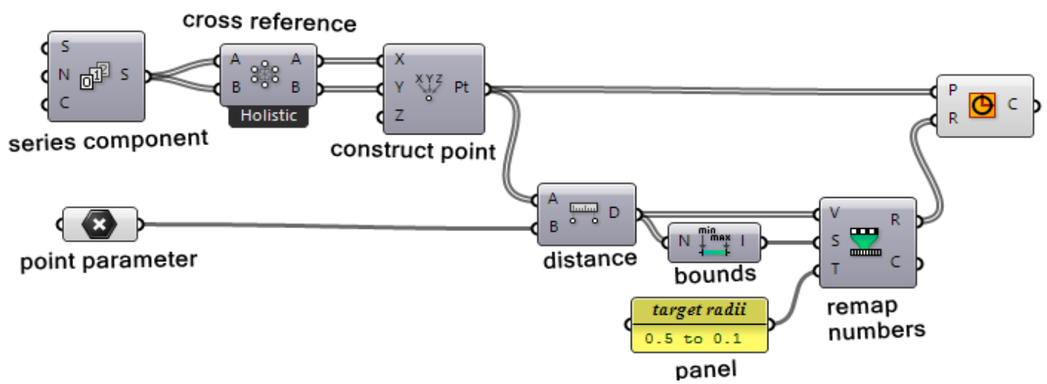
FILE: 01A_grid of circles - attractor.gh

This definition creates a grid of circles with different radii depending on the distance to a given point. The circles closest to the point have the largest radius, getting smaller the further away they are.

The **series** component creates a list of numbers. By default it generates 10 numbers that range from 0 to 10. The **cross reference** component rearranges and duplicates the elements of this list in a way that when the outputs are connected to the same component, all values are matched with each other. If this component is not used a diagonal line, rather than a grid, is generated. The **construct point** component creates a point out of three numbers (the x, y and z coordinates). The cross referenced list of numbers will generate a grid of points. The **distance** component measures the distance between the grid of points and a point parameter that contains a referenced point. The **bounds** component returns a domain that ranges from the smallest value of the input list to the largest value.

To input the target domain a **panel** object is used. A domain can be defined inside a panel by using the to keyword between two numbers. The **remap numbers** component scales a list of numbers from a source domain to new domain. This is used to convert the distance numbers into the radii of the circles. In this case the new domain is [0.5 to 0.1]. This means that the smallest distance becomes 0.5 while the largest distance becomes 0.1. All other distances will be scaled proportionally to a number in between 0.5 to 0.1. Finally, the **circle** component creates the circles given the grid of points and the corresponding radii. This component requires a plane rather than a point to place a circle. When a point is connected to a plane parameter, an XY plane and centered on the input point is created. The radius of the circle is set by a number parameter.





EXERCISES

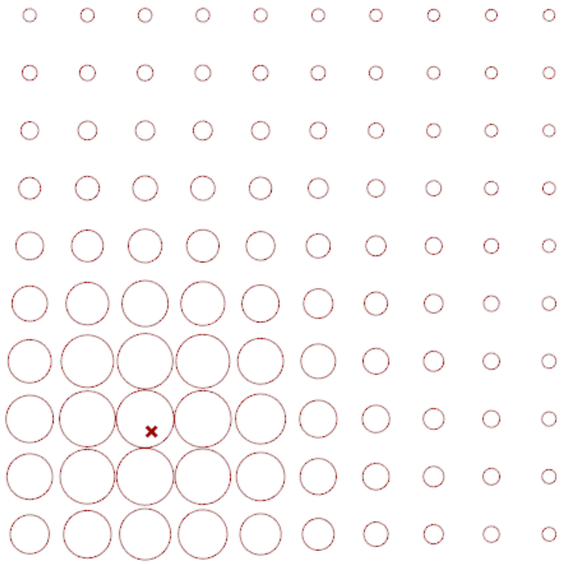
EXERCISE 01 - GRID OF CIRCLES

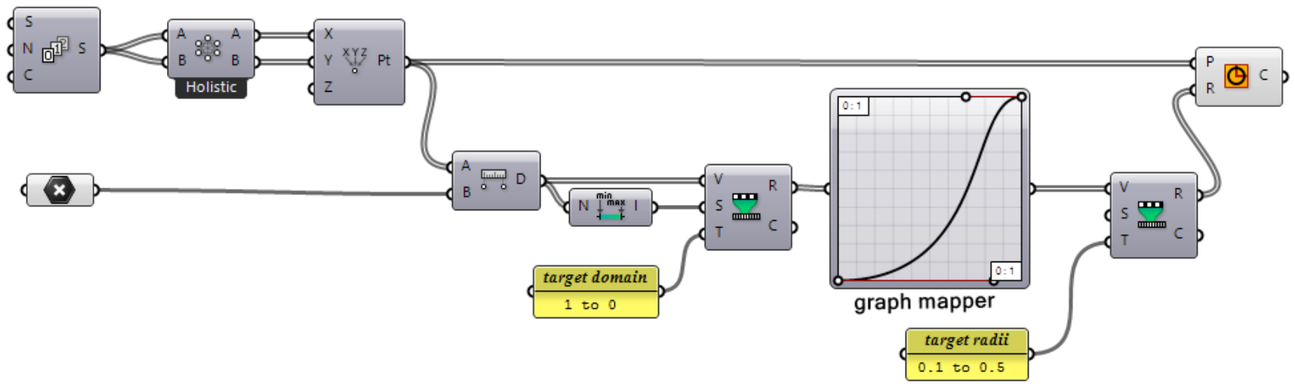
01.B - CHANGING THE FALLOFF

FILE: 01B_grid of circles - graph mapper.gh

This exercise serves as a basic introduction to Grasshopper and its interface. A set of curves are extruded an amount that is proportional to the length of each curve.

The first **remap numbers** component normalizes the distances so that they range from 0 to 1. This is done because the graph mapper component range is set from 0 to 1 by default. It's also practical to normalize data to this range when we only care about the relationship between the values rather than the nominal values. The values are also inverted (the domain is set from 1 to 0) so that the largest value becomes the smallest and vice-versa. The **graph mapper** component modifies the input values by placing them in the x axis of the graph, intersecting them with the graph curve and using its position in the y axis as the new value. This is similar as how curve adjustment works on a graphics editor like Photoshop. Right clicking on the component allows the selection of different graph types. In this case the bezier graph is selected. Finally, another **remap** component scales the normalized values to the range numbers that can be used as the radii of the circles [0.1 to 0.5].





EXERCISES

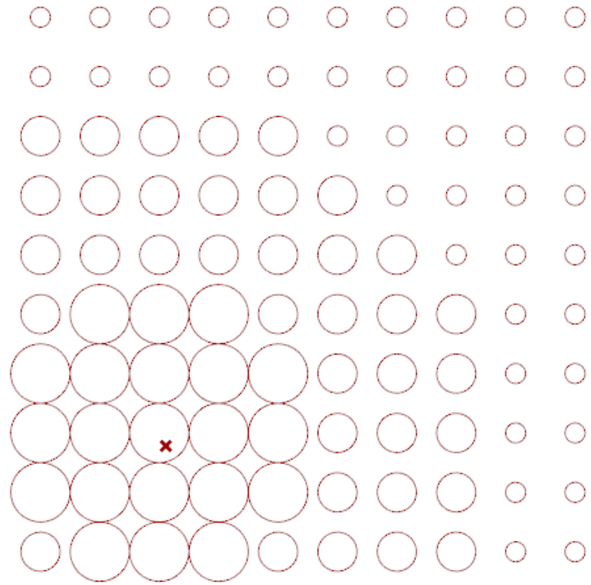
EXERCISE 01 - GRID OF CIRCLES

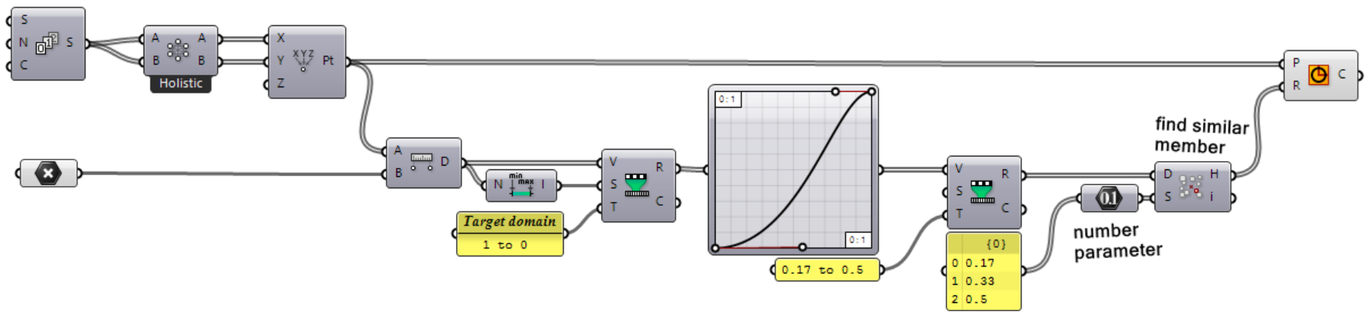
01.C - FIXED NUMBER OF RADII

FILE: 01C_grid of circles - fixed number of radii.gh

In the previous exercise, the radii are calculated directly from the distance to the referenced point, this means that there are as many different radii as there are different distances. It's common to want limit the different available shape sizes to just a few.

A **panel** object is used to manually add a list of predefined radii. A panel can be used to define a list of data by right clicking on it and deselecting multiline data. The **find similar member** component will output the closets value of the predefined list of radii for each of the radii calculated in the previous exercise. Panels generate text data that will automatically be converted to **number data** when connected to an input expecting numbers. The find similar member component can work with other data types besides numbers, so a number parameter must be connected to the panel to force a conversion to numbers.





EXERCISES

EXERCISE 01 - GRID OF CIRCLES

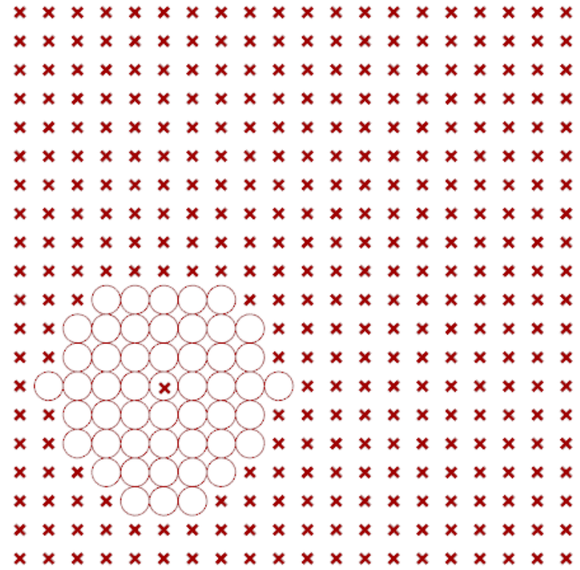
01.D - DISPLAY THE 50 CLOSEST CIRCLES

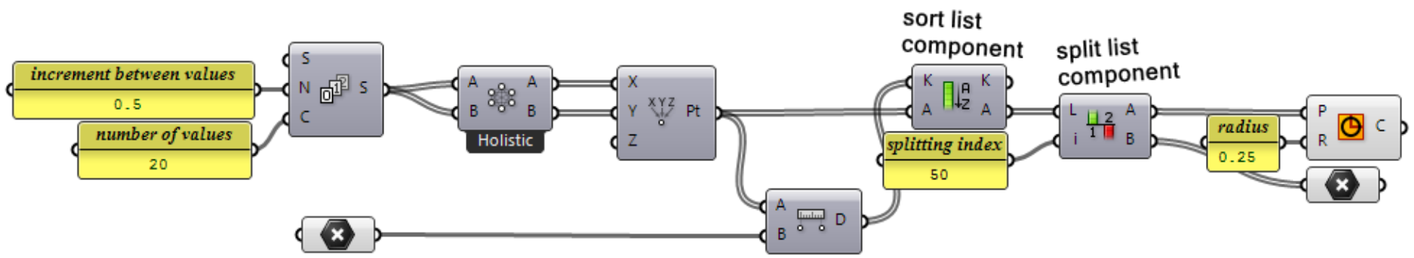
FILE: 01D_grid of circles - sort list.gh

In the following exercises, the radii of the circles stay constant. The distance of each circle to the referenced point is used to filter some of them out depending on the logic used. In this example only the closest 50 circles will be displayed and the rest of them will be filtered out.

The **sort** component sorts a list of numbers connected to the K input. Any other list connected to the A input will be sorted in the same way that the K list has been sorted. Connecting the list that contains the grid of points to the A input will sort those points from the closest one to the one furthest away. The **split list** component splits the sorted list into two different outputs. The i input corresponds to the index number of the first element of the second list. The 50th closest point has an index number of 49 so the splitting index number is set to 50.

The **circle** component is only connected to the first list so that only the closest 50 circles are displayed.





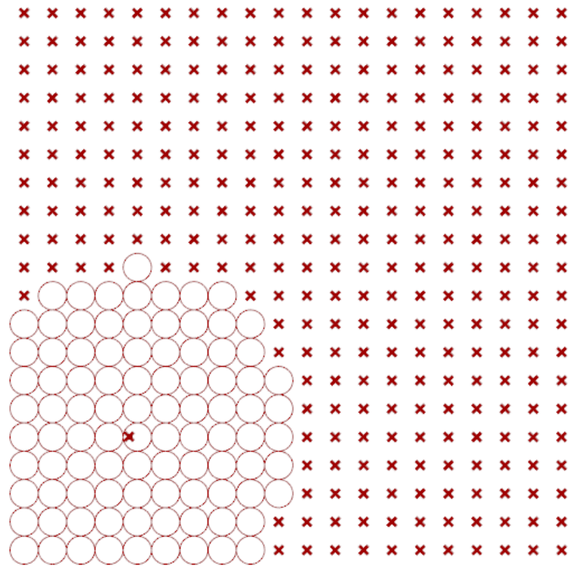
EXERCISES

EXERCISE 01 - GRID OF CIRCLES

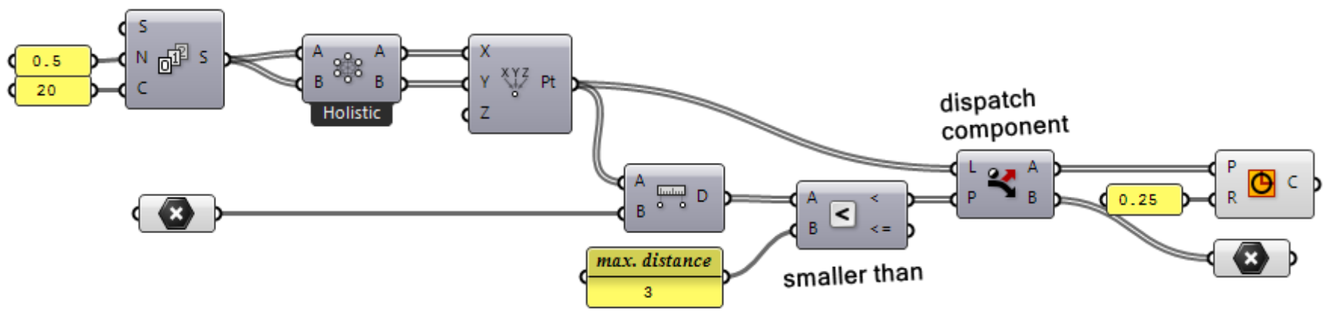
01.E - DISPLAY THE CIRCLES THAT FALL INSIDE A GIVEN DISTANCE

FILE: 01E_grid of circles - cull distance.gh

In this exercise, some circles are filtered out if they are further away than a given distance. Depending on the distance more or less circles will be filtered out.



A **smaller than** component is used to compare the list of distances to a given number. If the distance is smaller, the smaller than component retruns true, if it's larger it retruns false. The **dispatch** component outputs the elements of the L input to the A output when the corresponding P value is true or to the B output when it's false. In this case the points closer than the maximum distance value will be placed on the A output. Circles are created using the points from the A output.



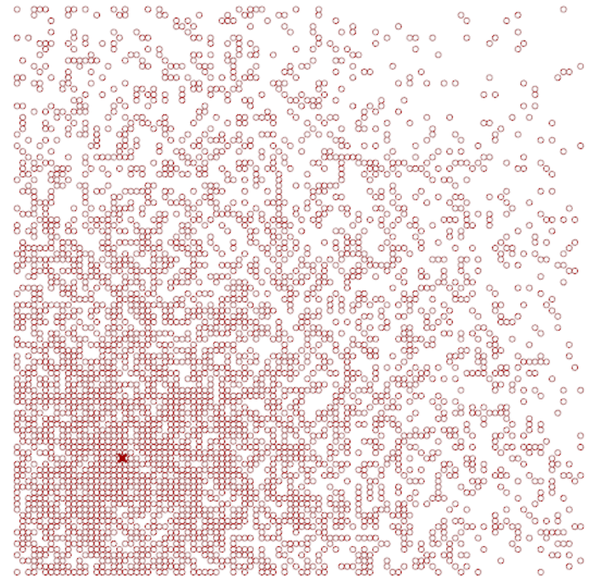
EXERCISES

EXERCISE 01 - GRID OF CIRCLES

01.F - CULL CIRCLES GRADUALLY

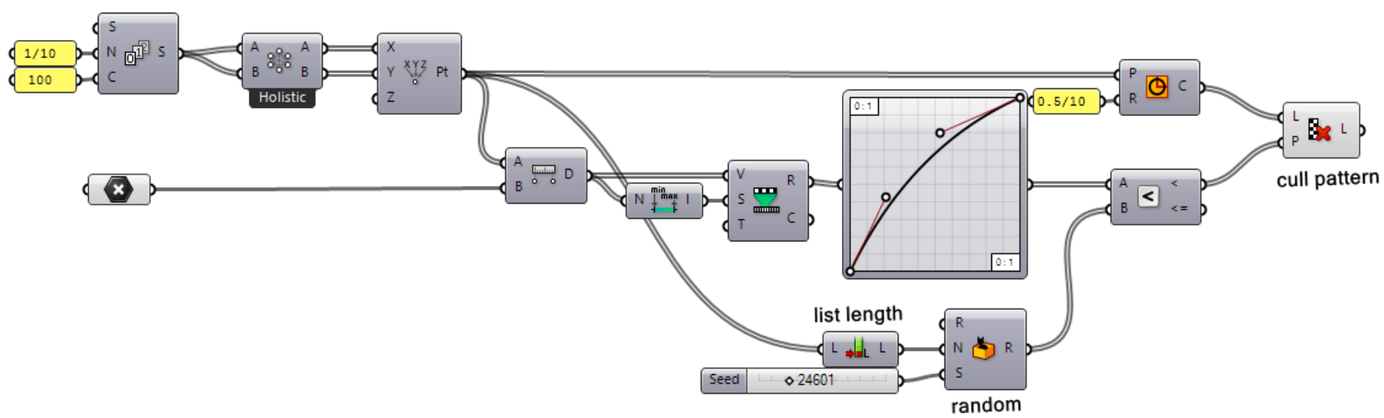
FILE: 01F_grid of circles - density.gh

In this exercise, points are filtered out gradually over distance. The further away, the more points will be filtered out.



The distance values are normalized to the [0 to 1] range as in the 1B exercise. Optionally these values can be modified using a **graph mapper**. For each point, a random number is generated that ranges from 0 to 1. These numbers are created using the **random** component. The R input is a domain where all random numbers will fall inside, by default it's set to the [0 to 1] range. The N input is the number of random values to generate. A **list length** component is used to calculate the total number of points in the list. For every seed value, a different set of random numbers is generated. The seed value can be set to any integer. The normalized distances are compared to the random numbers using the smaller than component.

When the distance is smaller than its corresponding random number, it will return true. These are the circles that will be kept, the rest will be filtered out. The smaller the distance, the greater the probably that the random number will be larger. The **cull pattern** component works similar to the dispatch component but will only output the elements connected to the L input when its corresponding value in the P input is true.



EXERCISES

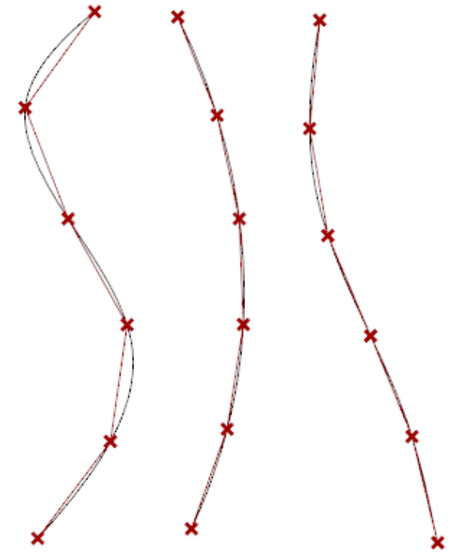
EXERCISE 02 - DIVIDE CURVES

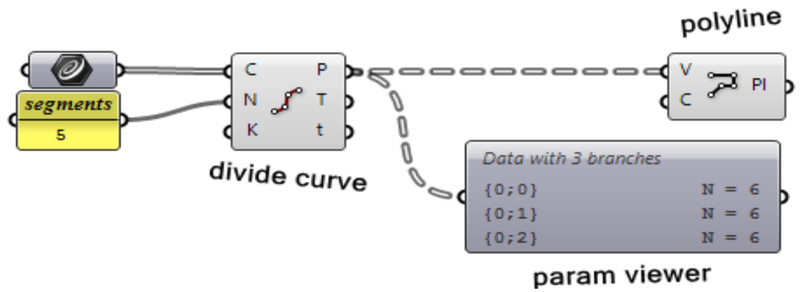
02.A - CREATE A POLYLINE THROUGH THE CURVES

FILE: 02A_divide curves - fit polyline.gh

This exercise introduces the concept of data trees. Three different curves are divided into a three sets of points, then a polyline is created for each set of points.

The **divide curve** component places a set of points approximately evenly spaced over a curve. The input is the number of segments to divide the curve into. If the curve is open, the number of points will be one more than the defined number of segments. Since for every input (every curve) a list of elements (a set of points) is created, the component creates a sublist for each list of points. These are called branches. This type of data structure is called a data tree. The **polyline** component will grab a whole list of points and create a single polyline out of it. Since there are three branches, it creates three different polylines. Components with inputs that have the **as list** text on their description will use the whole list of data every time the component is run, rather than running for each individual element.





EXERCISES

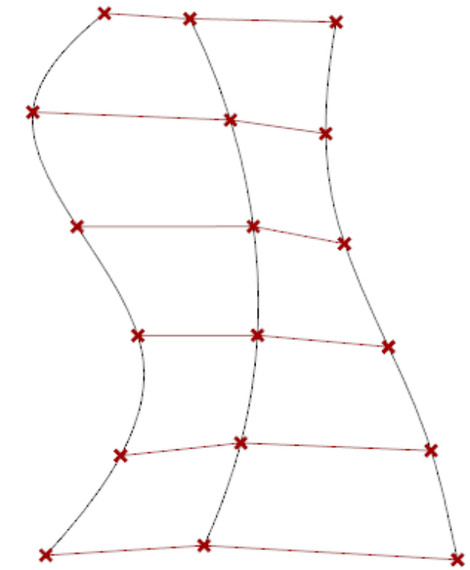
EXERCISE 02 - DIVIDE CURVES

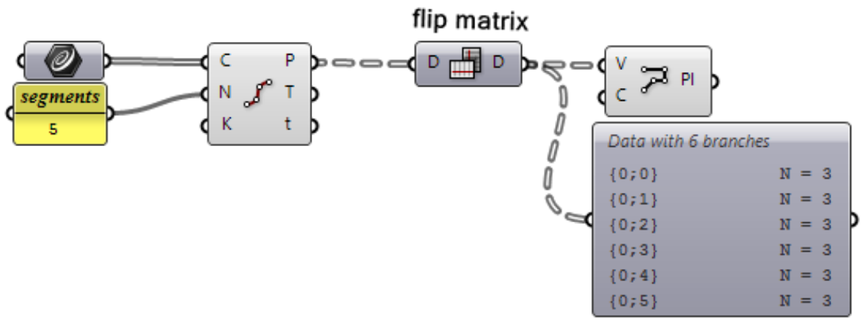
02.B - CREATE POLYLINES PERPENDICULAR TO THE CURVES

FILE: 02B_divide curves - flip matrix.gh

This exercise shows a simple way to change the structure of a data tree. The three branches of points are transposed so that one polyline connects the first point of the first curve, the first point of the second curve and the first point of the third curve. A second polyline will connect the second point of the first curve, the second point of the second curve and the second point of the third curve, and so on.

> As before, the **divide curve** component creates three branches, one for each curve, each containing six points. The **flip matrix** component transposes the data. Elements that share the same index value are placed on the same branch creating six branches of three points each.





INDEX

EXERCISES

EXERCISE 03 - CREATE A GRID ON A SURFACE

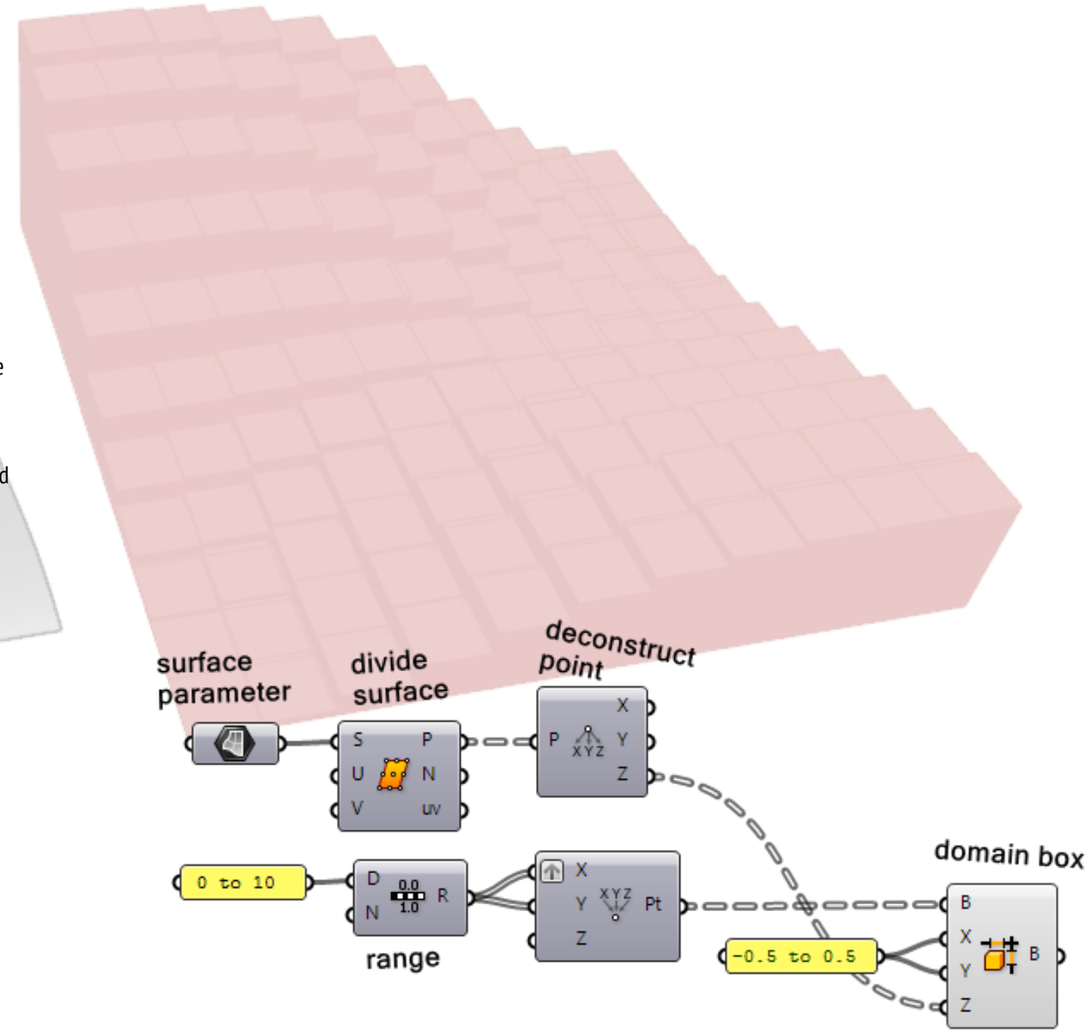
03.A - CREATE A GRID OF BOXES

FILE: 03A_ grid on surface - boxes.gh

A doubly curved surface modeled in Rhino is used to control the height of a grid of boxes.

The **surface** parameter is used to reference a surface modeled in Rhino. The **divide surface** component creates a grid of points over the surface (similar to the divide curve component). Each row of points is placed into a different branch. The **range** component divides a domain into a set of numbers. The **construct point** component is used to create a grid of points on the XY plane. The X input is grafted (right click on the input, select **graft**) so that each element is placed into an individual branch. This creates one branch for each row so that the data structure matches the grid of points over the surface. The **deconstruct** component separates in different outputs the numbers that make the coordinates of each point. The Z value of each point will be used as the height of each box.

The **domain box** component creates a box given a plane and three domains, one for each axis. Domains allow to offset the box from the plane so it does not need to be centered on it. In this case the bottom face of the box will be placed on the plane and will grow vertically. To achieve this, a domain ranging from 0 to the Z value extracted in the previous step is connected to the Z input. When connecting a number to a domain, a domain that ranges from 0 to the number is created automatically. The box sides on the in the X and Y directions measure 1 unit and are centered on the plane. To achieve this a domain that ranges from -0.5 to 0.5 (as a magnitude of 1 unit) is connected to the X and Y inputs.



INDEX

EXERCISES

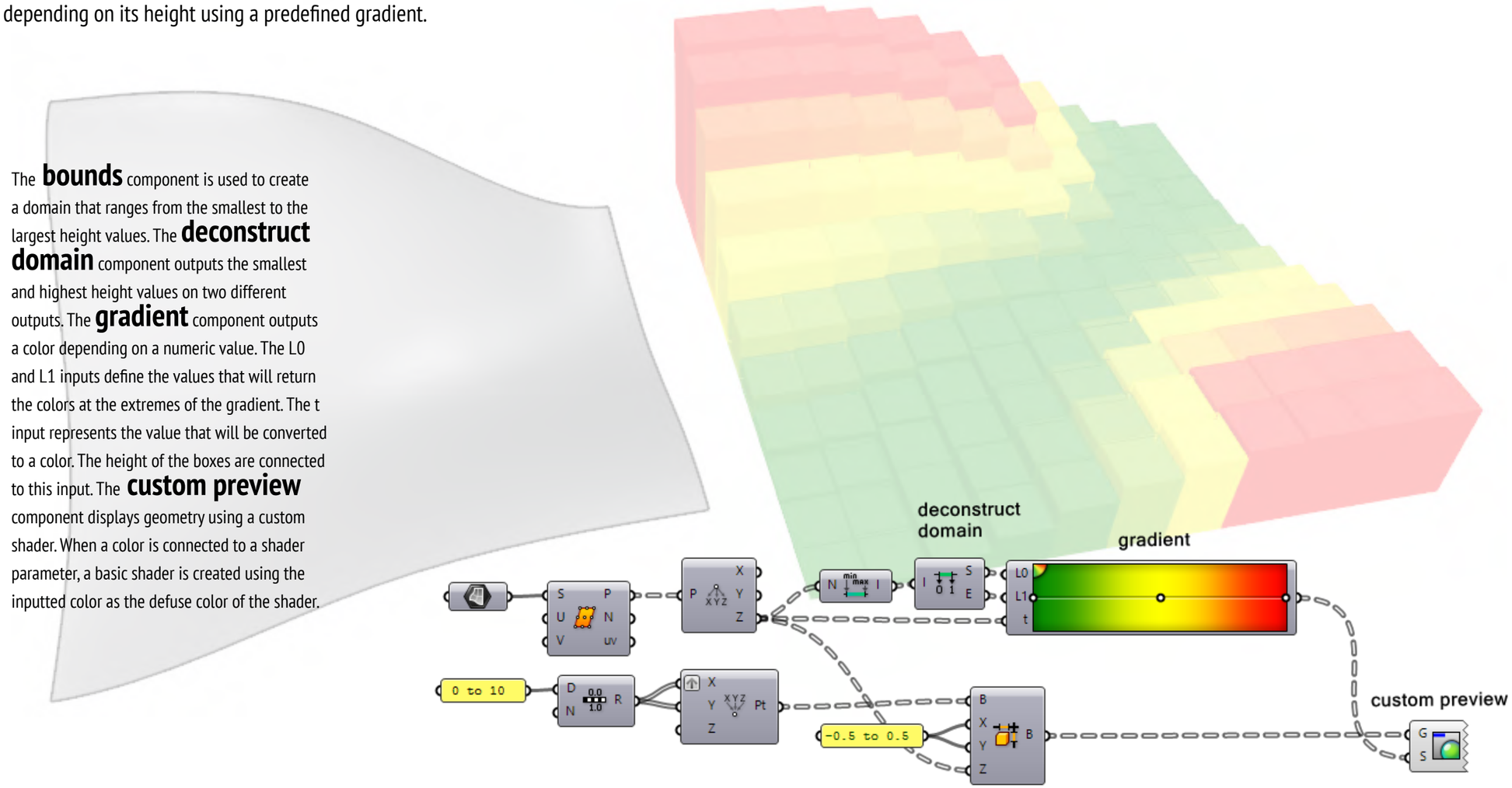
EXERCISE 03 - CREATE A GRID ON A SURFACE

03.B - COLOR THE BOXES

FILE: 03B_ grid on surface - colors.gh

Each box is colored depending on its height using a predefined gradient.

The **bounds** component is used to create a domain that ranges from the smallest to the largest height values. The **deconstruct domain** component outputs the smallest and highest height values on two different outputs. The **gradient** component outputs a color depending on a numeric value. The L0 and L1 inputs define the values that will return the colors at the extremes of the gradient. The t input represents the value that will be converted to a color. The height of the boxes are connected to this input. The **custom preview** component displays geometry using a custom shader. When a color is connected to a shader parameter, a basic shader is created using the inputted color as the defuse color of the shader.



INDEX

EXERCISES

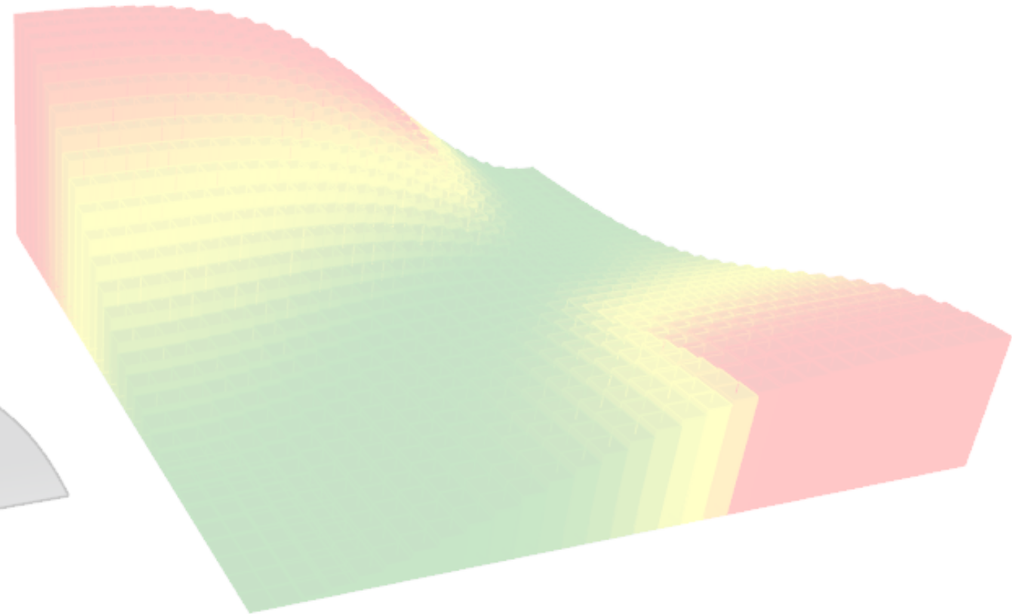
EXERCISE 03 - CREATE A GRID ON A SURFACE

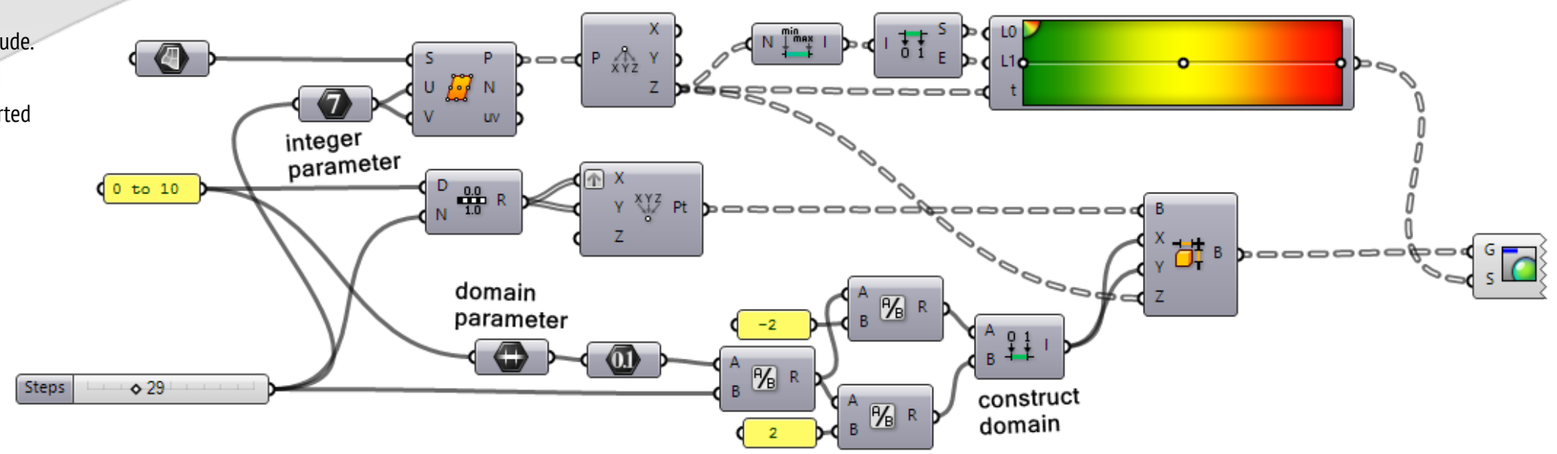
03.C - ESTABLISH NUMERICAL RELATIONSHIPS

FILE: 03C_ grid on surface - relationships.gh

To change the number of boxes there are three different inputs that have to be changed. A fourth input has to be changed in order to keep the boxes to not overlap or leave gaps. This exercises establishes relationships to all these inputs so that changing a single value on a slider will set the right values to all of these inputs.

A **slider** is used to set the number of points on both axis of the grid. Since both axis are the same, the U and V inputs of the divide surface component are going to be the same number. An integer parameter is connected to the **slider** and then to both of these inputs in order to tidy up the definition. The same slider is connected to the range component that creates the planar grid of the same size. To figure out the size of the box, the size of the grid has to be divided by the number of segments. The size of the grid is defined by a domain. A number parameter connected to a domain will return its magnitude. Since the panel where the domain is defined outputs text, first it has to properly be converted to a domain using a domain parameter.





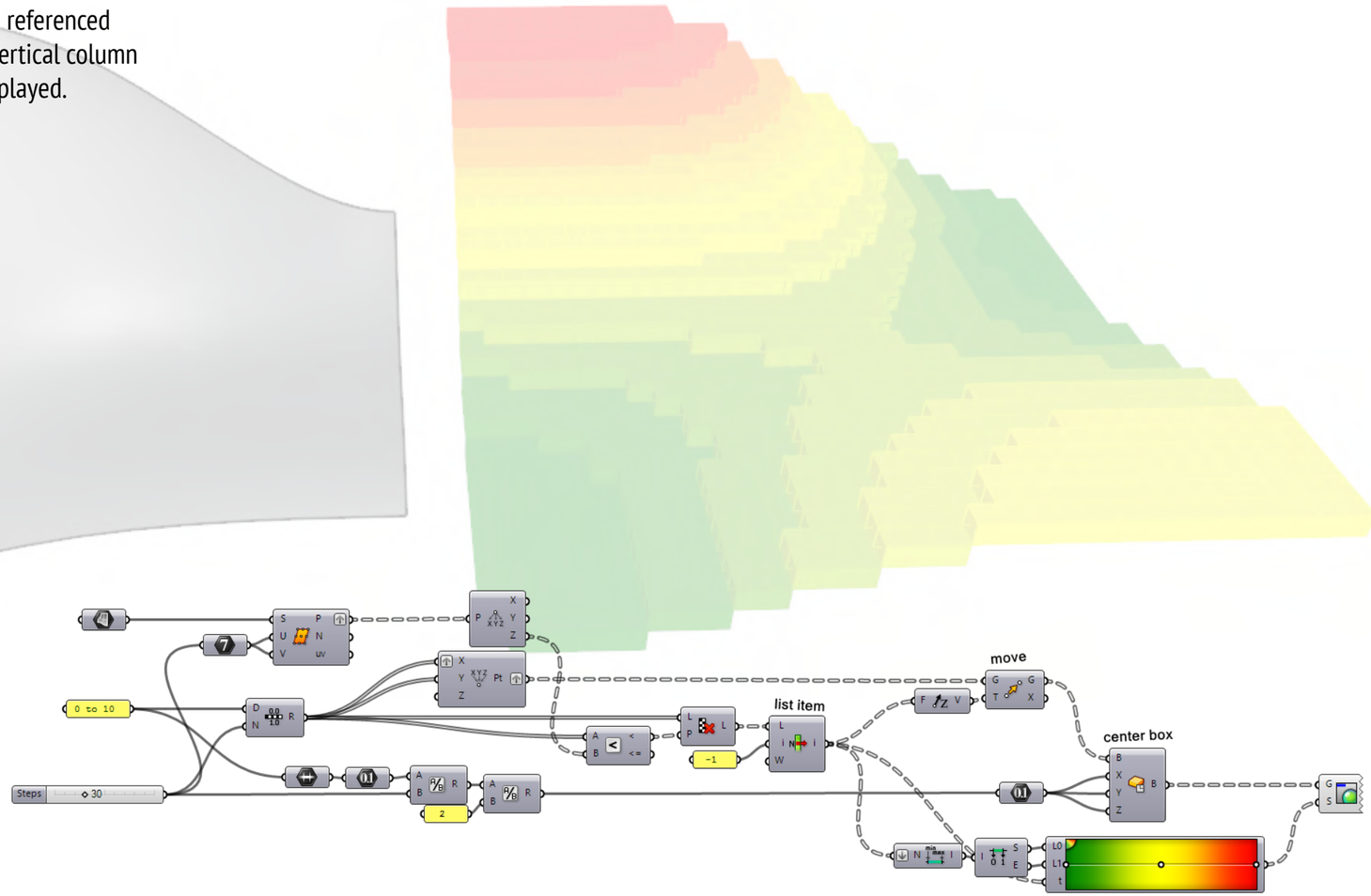
EXERCISES

EXERCISE 03 - CREATE A GRID ON A SURFACE

03.D - FILTER A GRID OF CUBES

FILE: 03D_ grid on surface - cubes.gh

In this exercise, rather than setting an exact size to each box, the referenced surface is used to filter boxes from a 3D grid of cubes. For each vertical column of cubes only the last cube that lies under the surface will be displayed.



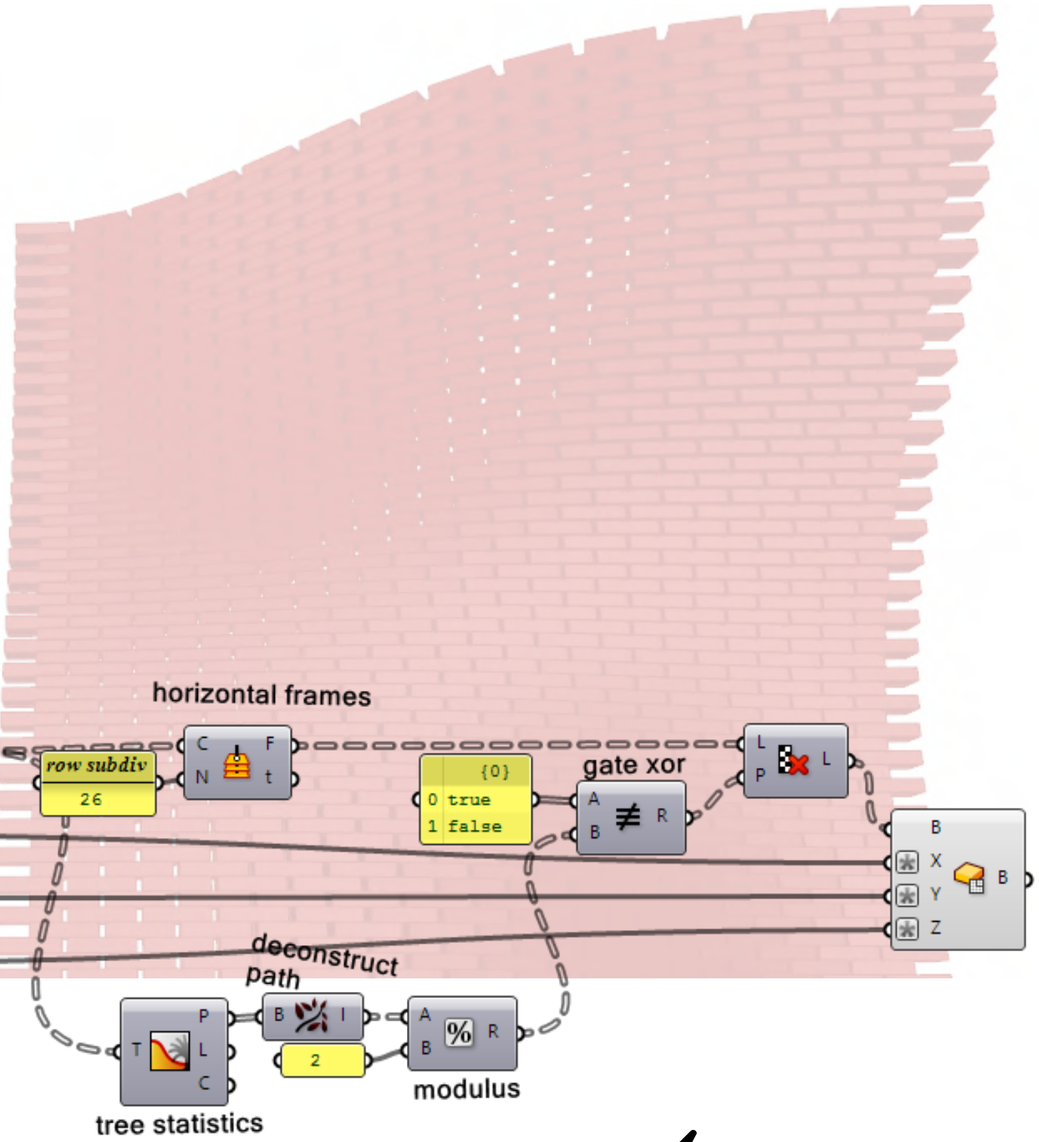
Both grids are **grafted** (right click on the input and select graft). This means that each point is placed in an individual branch. This way each vertical column of the 3D grid will be on a different branch. The height values from the surface grid of points are compared to the numbers generated by the **range** component using the smaller than component. The same values from range component are used because the 3D grid has the same size in all dimensions. For each vertical column, the **cull pattern** component keeps only the height values that lie under the surface. The **list item** component extracts a single element from a list. In this case it extracts the last element of the list of height values. This is done using the -1 value as the index value, since they wrap around the list.The move component is used to create new points at the extracted number. The **center box** component is used to create cubes on these points. Center box requires a plane and three numbers that correspond to the three axis. These numbers are the distance from the plane to the one of the faces of each side, this means that the calculated size of the box has to be divided by two first.

EXERCISES

EXERCISE 04 - BRICKS ON SURFACE

FILE: 04_ brickwork - stretcher bond.gh

In this exercise a brick wall is created following the stretcher bond over a doubly curved surface modeled in Rhino.



The **contours** component intersects the surface with parallel planes creating curves. These curves are the axis of each of the rows of bricks. The normal of the intersection planes is left to its default value, the world Z axis. A branch is created for each intersection plane. Since there are no holes on concave sides on the surface only one curve is created on each branch. The **horizontal frames** component divides each curve into a set of planes parallel to the XY plane that follow the curve. These planes are used to place the bricks. To create the bond pattern some of the planes have to be filtered out. In this case the odd rows will have the odd planes contained on them filtered out and vice-versa. To find out which row is odd or even, first the tree statistic component extracts the names of the branches, these are called paths. The paths can be decomposed into integers with the deconstruct path component. The **modulus** component returns the remainder of the division between the integer and 2. It will return 0 on the even paths and 1 on the odd paths.

The **gate XOR** component produces a exclusive disjunction between the result of the modulus component and a list of two values (true, false).The integers from the modulus component are converted 1 to true and 0 to false when connected to a boolean input. This returns “true, false” on the even rows and “false, true” on the odd rows. The **cull pattern** component filters out the planes using the pattern achieved in the previous step. The **center box** component crates the bricks on the left over planes.

INDEX

EXERCISES

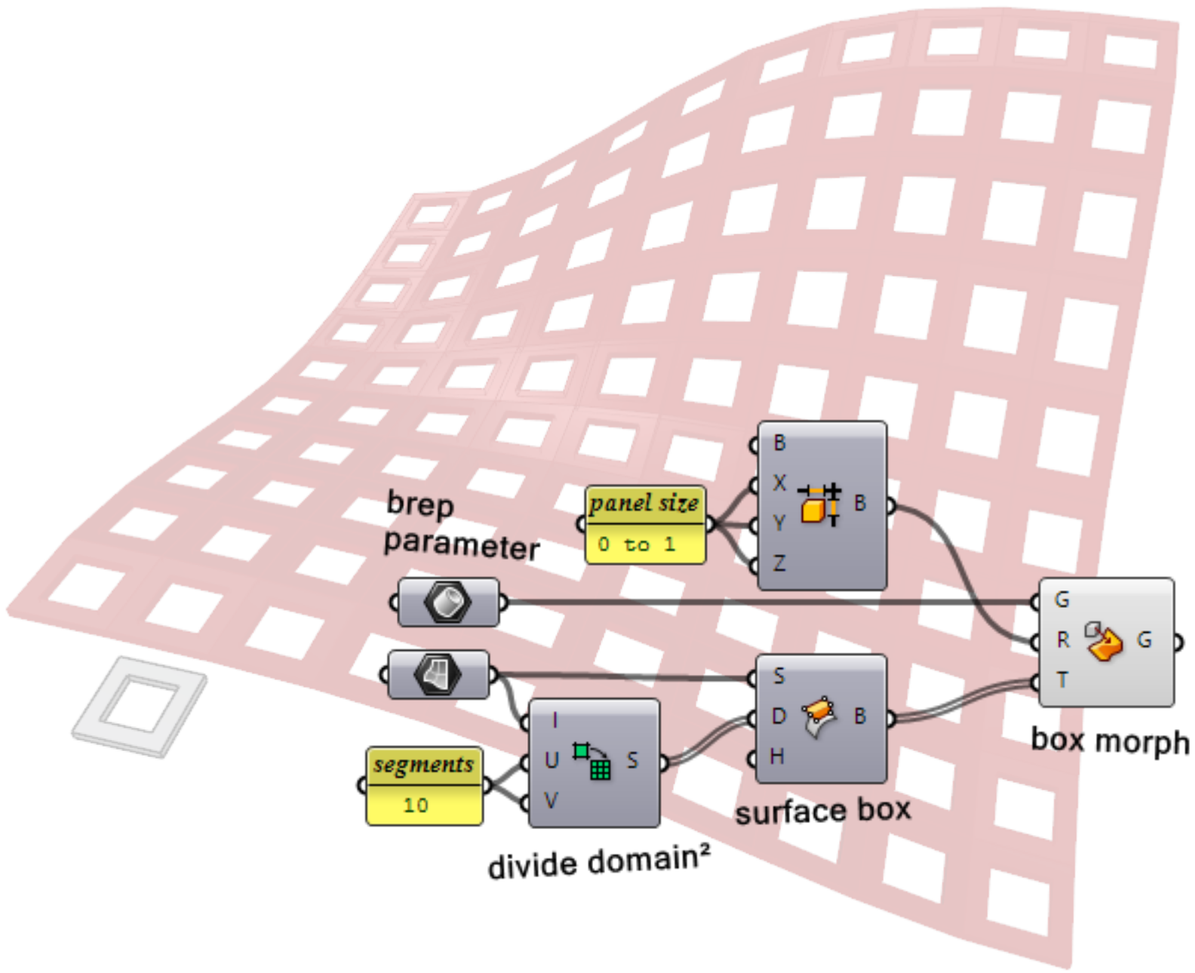
EXERCISE 05 - CREATE PANELS ON A SURFACE

05.A - BASE PANEL MODELED IN RHINO

FILE: 05A_ panel surface - from brep.gh

In this exercise a surface modeled in Rhino is panelized using a base shape. Each panel is morphed to follow the surface curvature making a continuous form with no gaps or overlaps between the panels.

The **BREP** parameter references the base panel shape modeled in Rhino as a polysurface. The **divide domain²** component divides the UV domain space of the surface into a set of sub domains. The **surface box** component creates a twisted box on each of the sub domains on the surface. These boxes have their 4 vertical edges perpendicular to the surface. The **interval box** component creates a bounding box around the base panel brep. The **box morph** component morphs the panel from the interval box into each of the twisted boxes.



EXERCISES

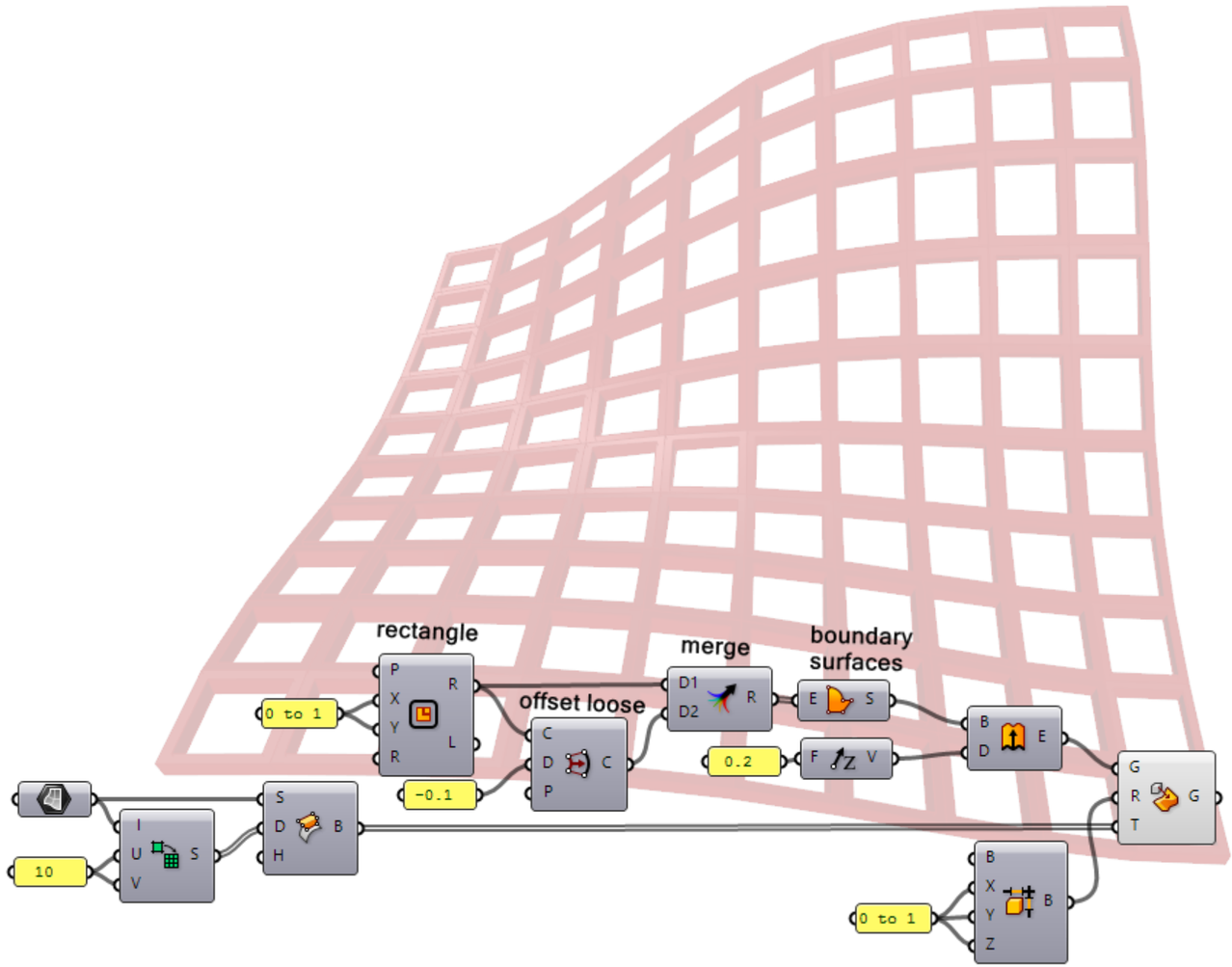
EXERCISE 05 - CREATE PANELS ON A SURFACE

05.B - BASE PANEL MODELED IN GRASSHOPPER I

FILE: 05B_ panel surface - from gh 1.gh

Instead of modeling the base panel shape in Rhino, in this exercise the panel is created using Grasshopper components.

A **rectangle** component is used to create the outer perimeter of the base panel. An **offset loose** component is used to create the interior hole of the panel. These two curves are placed in a single list using the merge component. The **boundary surfaces** component creates a planar surface out of these curves. If one closed curve is inside another it will create a single surface with a hole in it (a trimmed surface). The **extrude component** adds thickness to create the final shape of the base panel.



EXERCISES

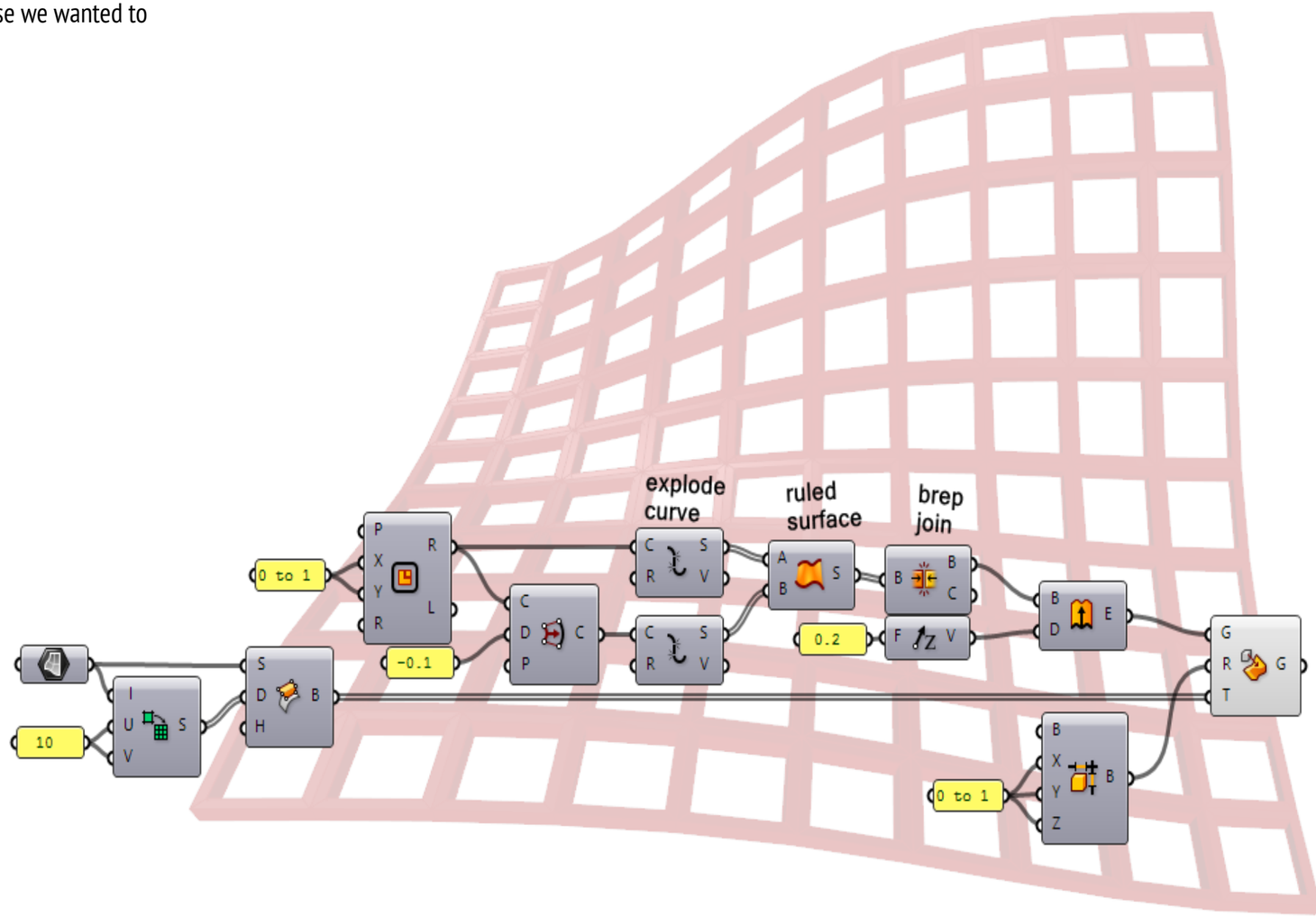
EXERCISE 05 - CREATE PANELS ON A SURFACE

05.C - BASE PANEL MODELED IN GRASSHOPPER II

FILE: 05C_ panel surface - from gh 2.gh

Rather than using a single trimmed planar surface to create the panel, an alternative is to use four untrimmed surfaces. This is useful in case we wanted to extract a clean mesh from the geometry.

Two **explode curve** components are used to split the curves that make the hole and the outer boundary into its four edges.
A surface is created for each pair of edges using the **ruled surface** component.
The **join brep** component joins the four surfaces into a single brep with four faces.



INDEX

EXERCISES

EXERCISE 05 - CREATE PANELS ON A SURFACE

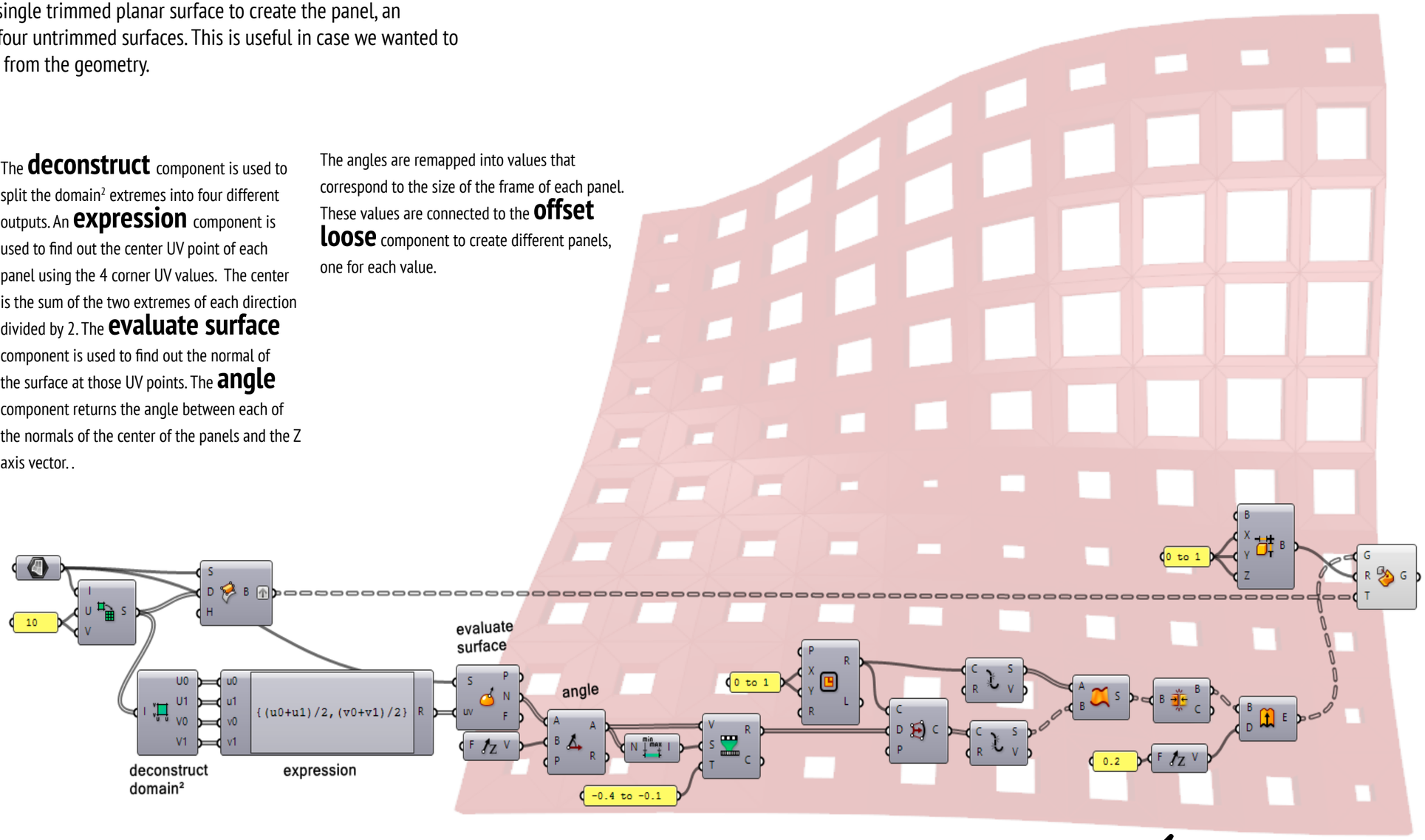
05.D - CHANGING OPENING SIZE DEPENDING ON A VARIABLE

FILE: 05D_ panel surface - different sizes.gh

Rather than using a single trimmed planar surface to create the panel, an alternative is to use four untrimmed surfaces. This is useful in case we wanted to extract a clean mesh from the geometry.

The **deconstruct** component is used to split the domain² extremes into four different outputs. An **expression** component is used to find out the center UV point of each panel using the 4 corner UV values. The center is the sum of the two extremes of each direction divided by 2. The **evaluate surface** component is used to find out the normal of the surface at those UV points. The **angle** component returns the angle between each of the normals of the center of the panels and the Z axis vector. .

The angles are remapped into values that correspond to the size of the frame of each panel. These values are connected to the **offset loose** component to create different panels, one for each value.



INDEX

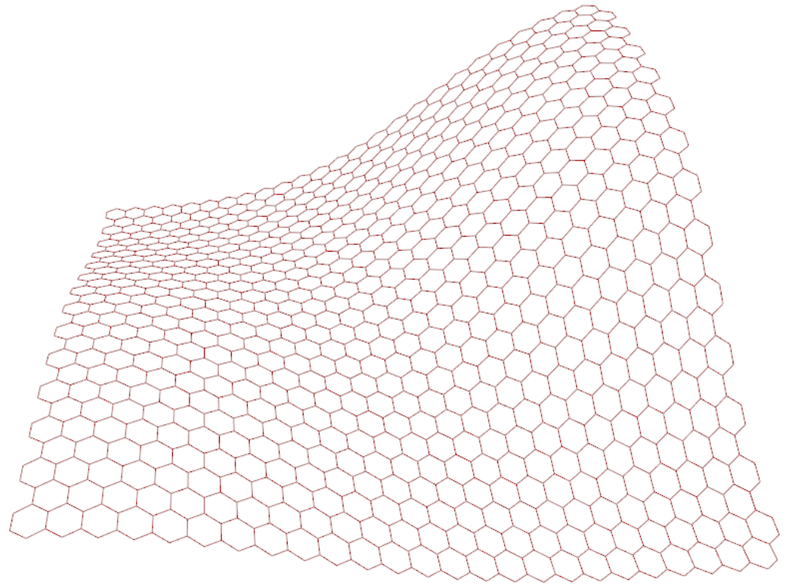
EXERCISES

EXERCISE 06 - CUSTOM GRIDS ON SURFACE

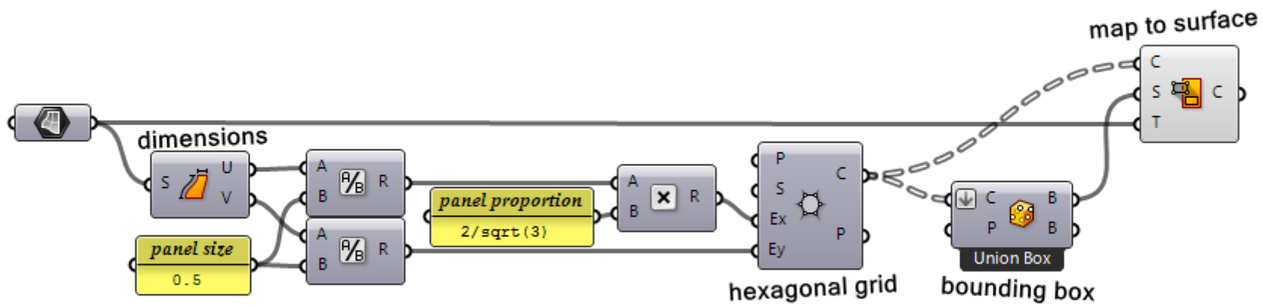
06.A - HEXAGONAL PANELS

FILE: 06A_ custom grid on surface - hexagons.gh

In this exercise, the surface is paneled using a grid of hexagonal curves. Rather than defining the number of rows and columns, an approximate panel size will be set and the rows and columns will be calculated from it.



The **dimension** component calculates the approximate size of the surface on its U and V directions. Both dimensions are divided by the panel size to calculate the necessary rows and columns. Since the hexagonal panels are not square, one of the dimensions is multiplied by its width/height ratio. The **hexagonal** component creates a flat 2D grid of hexagons as a set of polylines. The **bounding box** component creates a box that is the size of the hexagonal grid. The union box option must be selected (right click on the component, select union box) to create a single bounding box for all items. Since each row is in a different branch, the C input has to be flattened (right click on the input and select flatten). The **map to surface** component maps the flat grid of hexagons to the surface. The source surface will be a surface that is the size of the flat grid of hexagons. Connecting the flat bounding box to the target surface parameter will create a surface with the X and Y dimensions of the box.



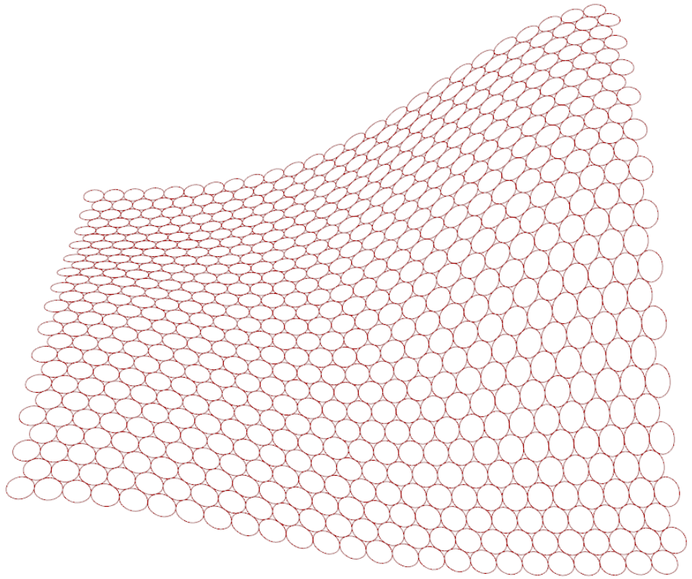
EXERCISES

EXERCISE 06 - CUSTOM GRIDS ON SURFACE

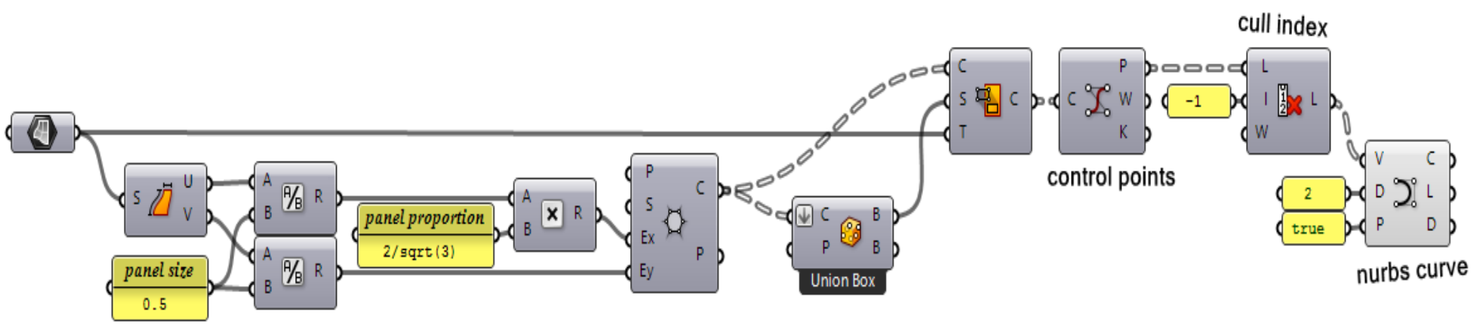
06.B - CIRCULAR GRID

FILE: 06B_ custom grid on surface - circles.gh

The best way to pack identical circles is to use a hexagonal pattern. In this exercise the hexagons will be converted to circle-like curves by using its vertices as control points for a degree 2 NURBS curve.



The **control points** component extracts the list of points that make the hexagonal polyline. Since the hexagonal polylines are closed, the last point is the same as the first one. A **cull index** component is used to remove the last point. The **NURBS curve** component creates a degree 2 NURBS curve out of this list of points. The curves will have a shape that approximates a circle.



EXERCISES

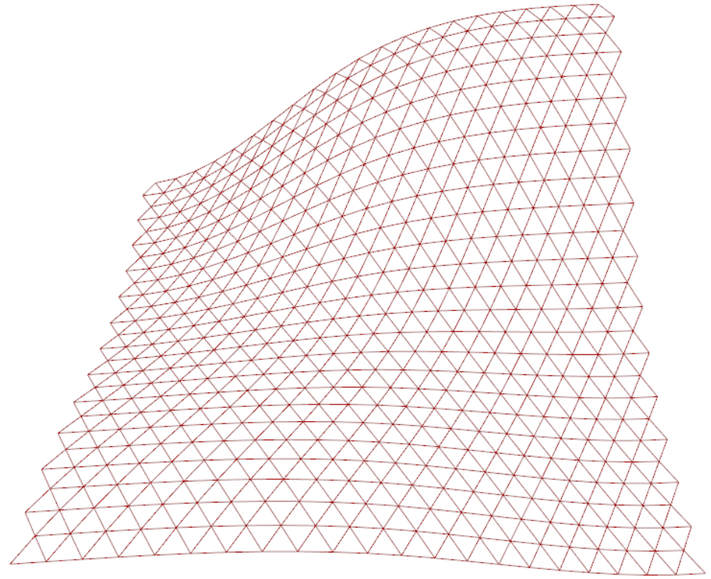
EXERCISE 06 - CUSTOM GRIDS ON SURFACE

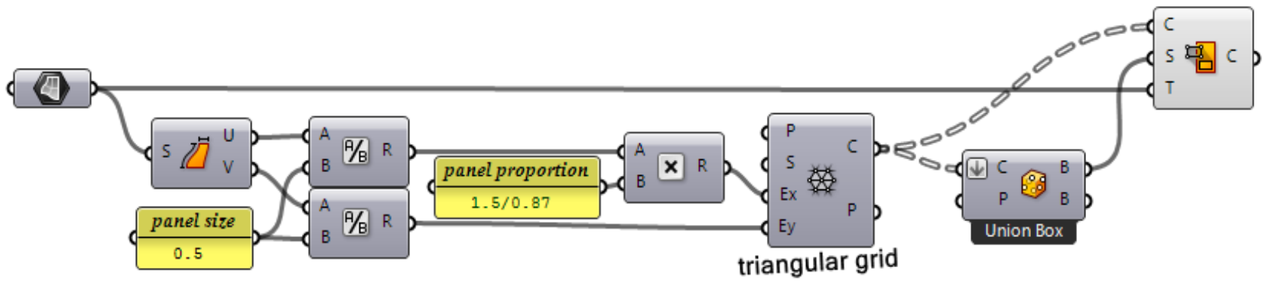
06.C - TRIANGULAR GRID

FILE: 06C_ custom grid on surface - triangles.gh

In this exercise a grid of equilateral triangles are mapped to the referenced surface. An advantage of using triangles is that they always remain flat. Note that after mapping to a doubly curved surface the triangles are not perfectly equilateral anymore.

The definition is almost identical to the hexagonal grid but in this case a **triangular grid** component is used. The panel proportion is different and has to be adjusted.





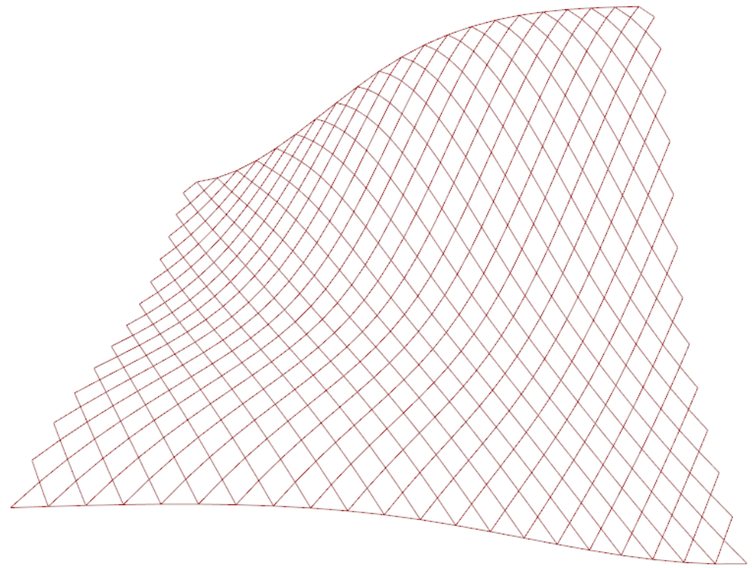
EXERCISES

EXERCISE 06 - CUSTOM GRIDS ON SURFACE

06.D - DIAMOND GRID

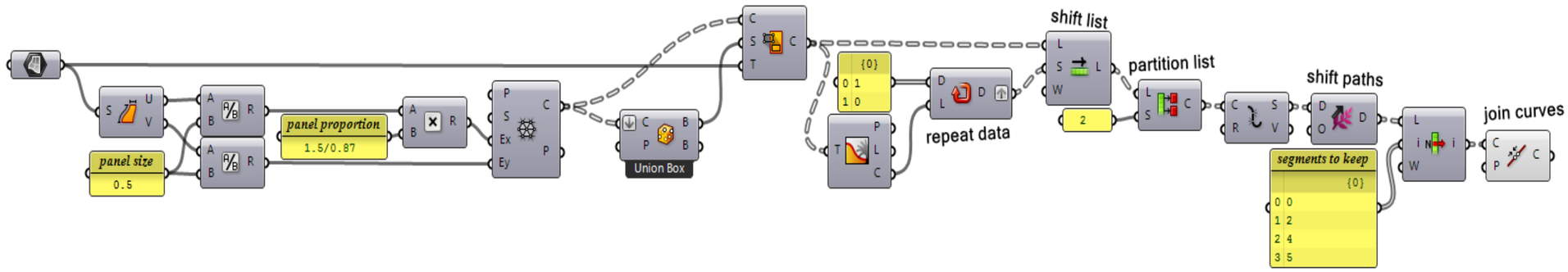
FILE: 06D_ custom grid on surface - diamonds.gh

The triangles are grouped in pairs to create a diamond grid.



The **repeat data** component is used to repeat a [1,0] pattern until there are as many elements as rows. The **tree statistics** component returns the number of branches of a data tree, which in this case is the number of rows. The list of numbers is grafted so that the data structure matches the triangular grid. These numbers are connected to a **shift list** component to shift only the odd rows. The last panels of these rows become the first and all the others move one position down the list.

The **partition list** component places every two triangles of each row into an individual branch. The triangles are exploded into their three segments. The **shift paths** component moves the segments one path down, placing all the segments of the same pair of triangles in the same branch. The segments that make the outer boundary of the diamonds are selected using the **list item** component. The segments are joined together into a closed polyline using the **join curves** component.



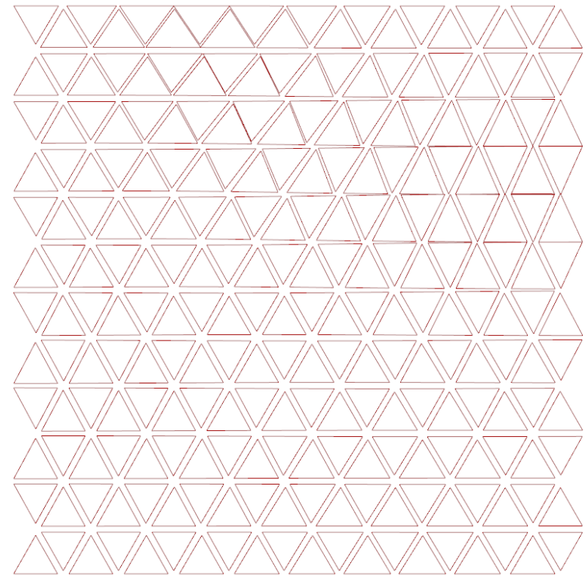
EXERCISES

EXERCISE 06 - CUSTOM GRIDS ON SURFACE

06.E - NESTING

FILE: 06E_ custom grid on surface - nesting.gh

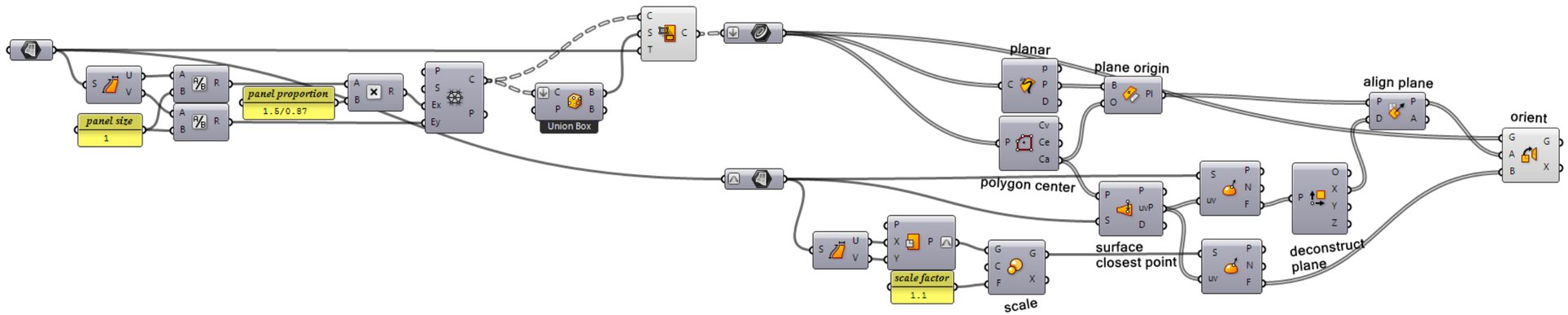
In this exercise, the triangular panels are laid out in a horizontal plane. This is useful in fabrication, for example, to create a physical model using a laser cutter.



The data-tree for the triangular panels is flattened and the surfaces are **reparameterized** so the surface domains range from 0 to 1 (right click on the parameter and select reparameterize). This is done using external parameters. Two sets of planes have to be created to orient the 3d geometry to the XY plane. One that matches the position and orientation of the 3D panels and one for their final position in the XY plane. The **planar** component is used to find out the plane of each panel.

The **polygon center** component is used to find out the center point of each panel. The **plane origin** component moves the curve plane to the center of the panel. The **surface closest point** component together with the **evaluate surface** component create a plane on the surface for each panel. This plane is used to align the previously created plane so that they all point in the direction of the surface.

The **deconstruct plane** component is used to extract the X vector of the surface plane in order to align the curve plane to it. The **scale** component scales the horizontal surface so that the panels don't overlap. A list of planes are created on the horizontal surface using the same UV values from the 3D surface. The **orient** component places the panels on the horizontal planes.



INDEX

EXERCISES

EXERCISE 06 - CUSTOM GRIDS ON SURFACE

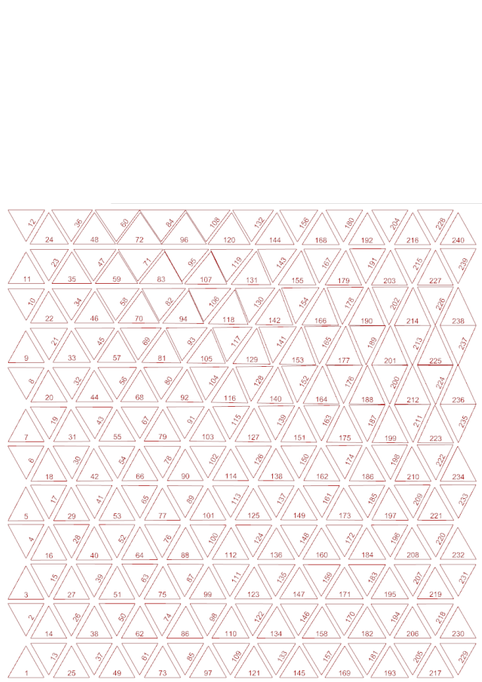
06.F - NUMBERING THE PANELS

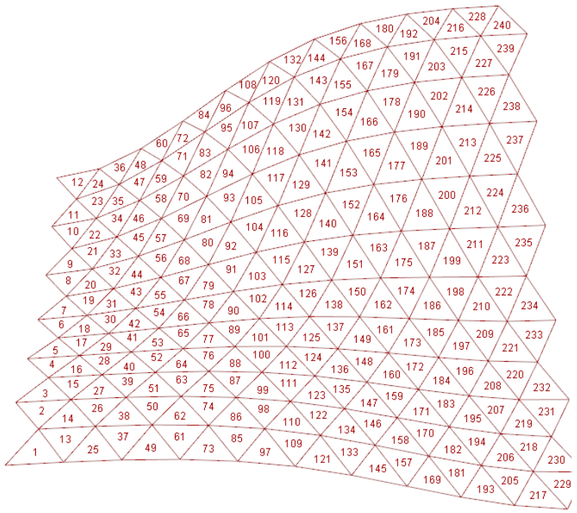
FILE: 06F_ custom grid on surface - text.gh

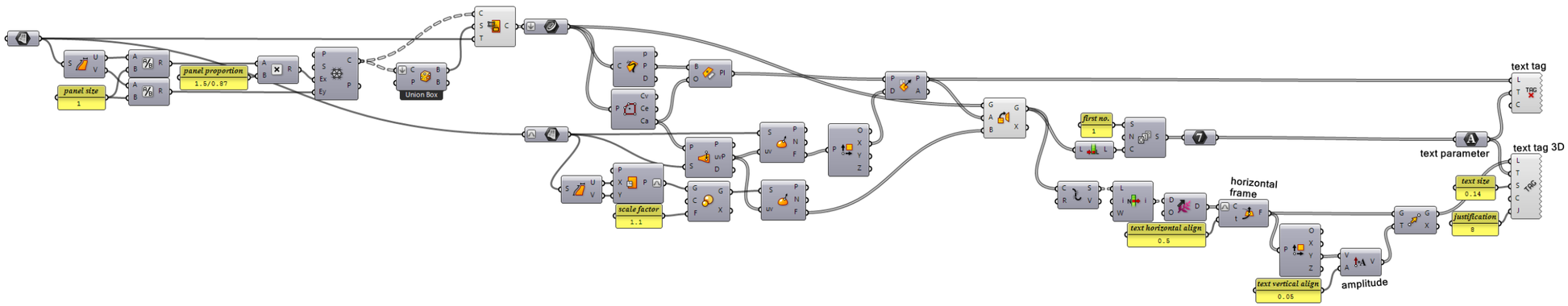
To be able to tell which 2D panel corresponds to which 3D panel they have to be numbered and annotated in the document.

A list of numbers is created, one for each panel, using the **series** component and the **list length** component. The first number of the series is changed to start at 1. Since there is no need for decimal places, the numbers are changed to integers using an **integer** parameter. The **text tag** component creates a text dot object using the center points of the 3D panels and the list of numbers as the text content. To place text over the 2D panels a **text tag 3D** component is used. This text is orientable, a plane is used for its orientation. The 2D panels are exploded into segments and the first one is extracted.

The **horizontal frame** component is used to create a horizontal plane in the middle of the curve with its X axis oriented to it. The plane is moved slightly toward the inside of the panel. To achieve this the plane is moved on its Y axis. The **amplitude** component is used to change the magnitude of the Y axis to the amount to be moved. The **text tag 3D** component uses the same list of numbers as the text tag component as the text content and places them in the horizontal planes created in the previous step.







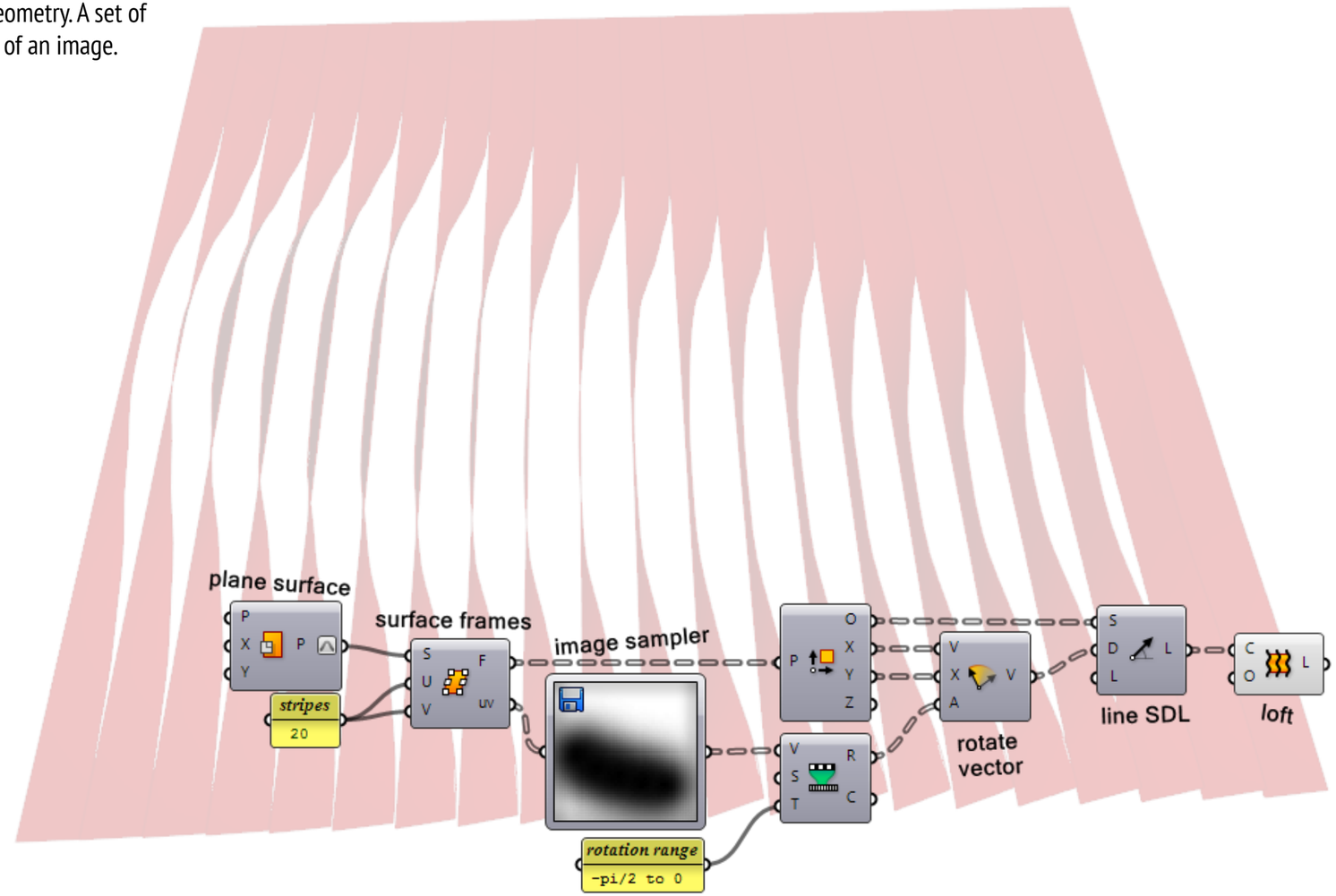
INDEX

EXERCISES

EXERCISE 07 - ROTATE STRIPS USING AN IMAGE

FILE: 07_ image sampler - strips.gh

In this exercise an image is used to drive the shape of some geometry. A set of strips are folded on certain parts depending on the brightness of an image.



A **plane surface** component is used to create a planar surface using a plane and two domains. The surface is reparameterized. The **surface frames** component is similar to the **divide surface** component, but creates planes tangent to the surface at each point. Each row is placed in a different branch. The **image sampler** returns color information given a set of normalized (that range from 0 to 1) points. The UV points from the surface frame component are used to return brightness values. The settings have to be changed to only output brightness values. It returns 1 for white pixels, 0 for black pixels and numbers in between for gray pixels. These numbers are remapped to become the amount of radians that each stripe will rotate at a given point. The **rotate vector** component rotates the X vector of each plane using the Y vector as the axis and the remapped values as the rotation angle. The **line SDL** component creates a line using the plane origin as the start point and the rotated vector as the line direction. The **loft** component creates a single surface for each row by lofting the lines together.

EXERCISES

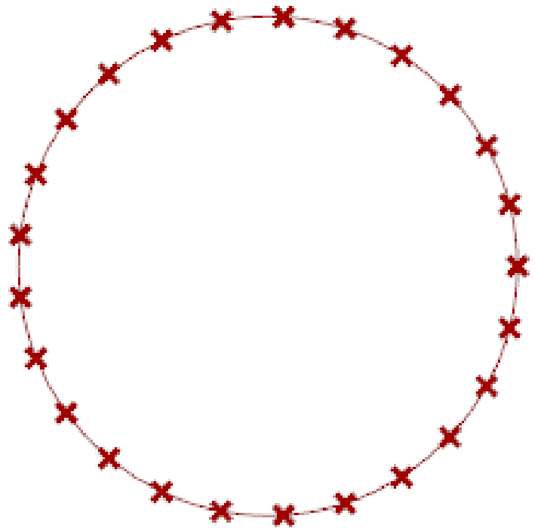
EXERCISE 08 - PARAMETRIC ECUATIONS

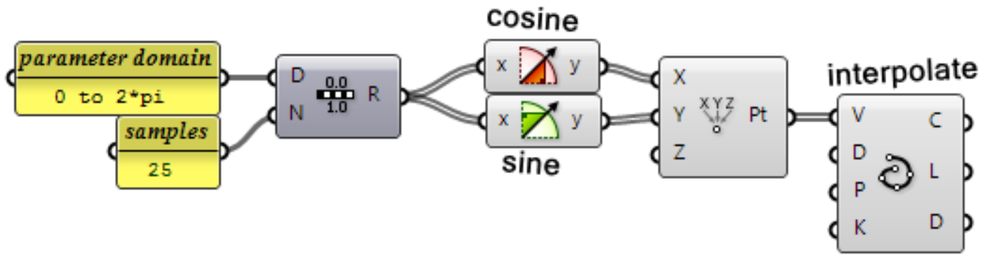
08.A - CIRCLE

FILE: 08A_ parametric equations - circle.gh

The coordinates for some curves and surfaces can be obtained from a set of equations. These contain variables called parameters. Curves contain one parameter (t), while surfaces contain two parameters (u, v). This exercise will generate a circle from a set of equations

A circle can be parameterized as {cos(t), sin(t)} for t = 0 to 2 x PI. The range component is used to generate a series of sample values for the curve parameter. **Sine**, **cosine** and the **construct point** components are used to create a set of sample points. An **interpolate curve** component is used to create a smooth curve that goes through the set of points..





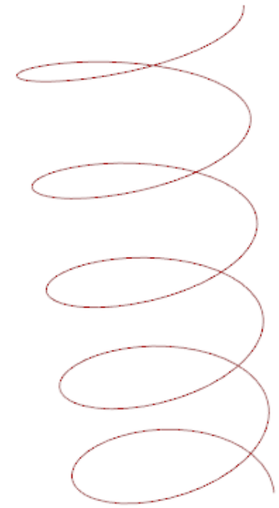
EXERCISES

EXERCISE 08 - PARAMETRIC ECUATIONS

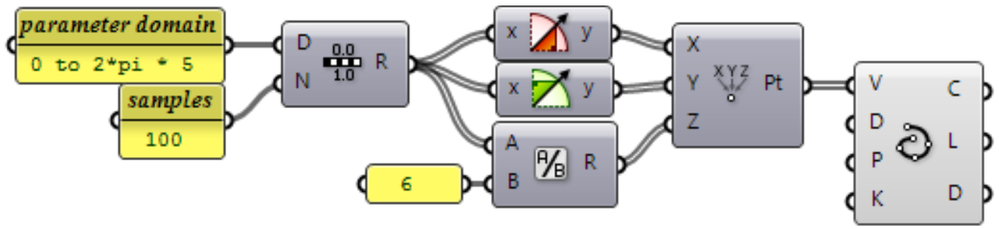
08.B - HELIX

FILE: 08B_ parametric equations - helix.gh

The equations to create a circle are modified in order to create a helix.



The parameter **domain** is modified so that the helix revolves 5 times rather than 1. The parameter is used as the **z value** so that the height of each sample point increases linearly. The parameter is divided by a number to reduce the **slope** of the helix.



INDEX

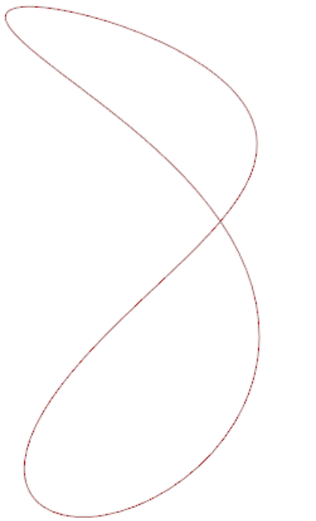
EXERCISES

EXERCISE 08 - PARAMETRIC ECUATIONS

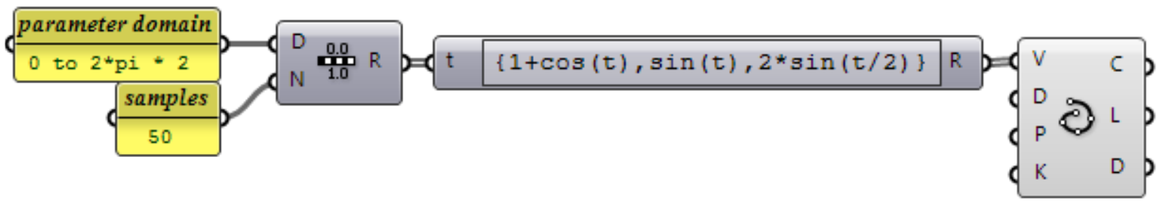
08.C - VIVIANI'S CURVE

FILE: 08C_ parametric equations - vivianis curve.gh

The Viviani's curve is the curve generated by the intersection between a sphere and a cylinder that has half the diameter of the sphere and is tangent to it.



Since the formula is a bit longer, rather than using multiple components, a single **expression** component is used to create the sample points.



EXERCISES

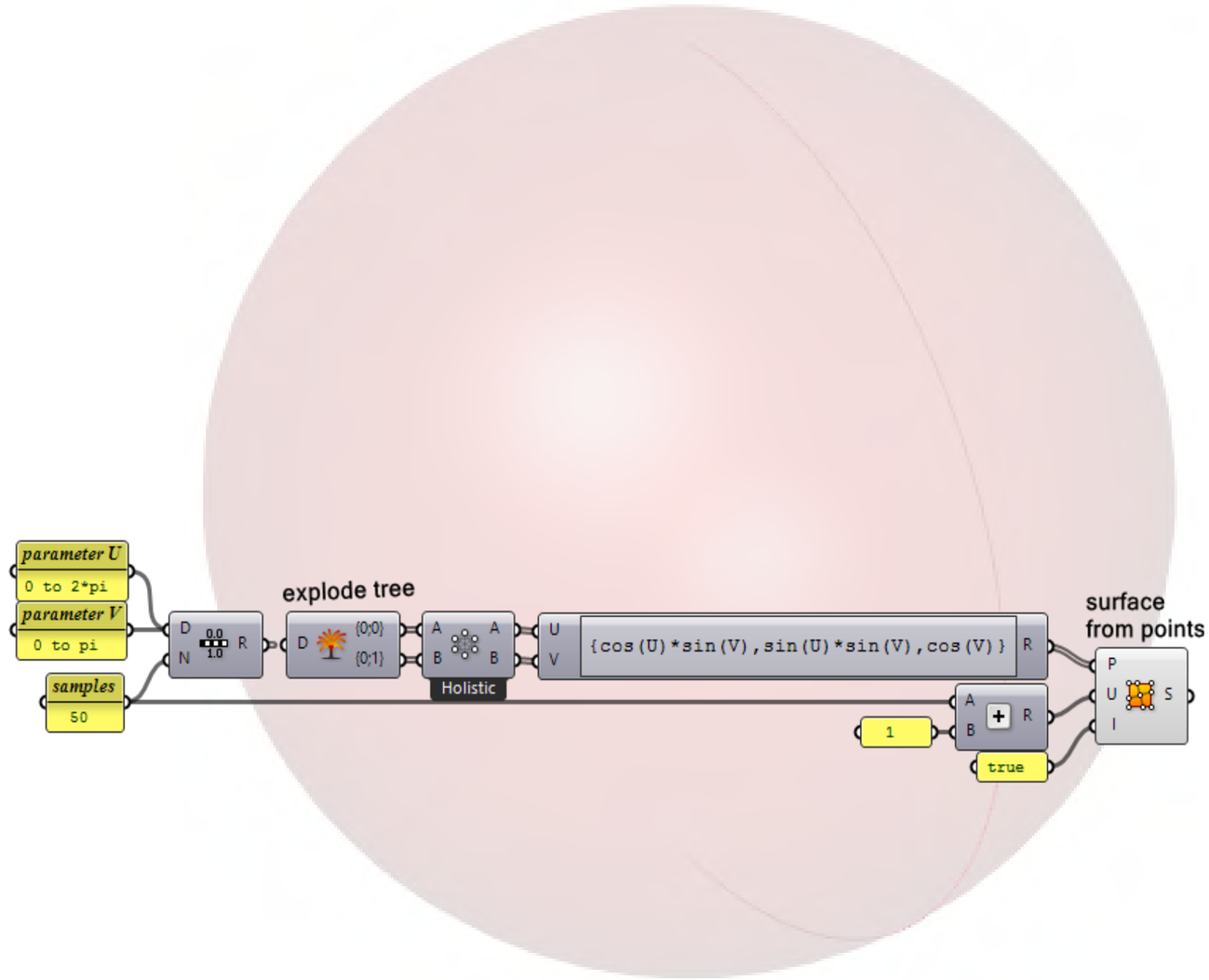
EXERCISE 08 - PARAMETRIC ECUATIONS

08.D - SPHERE

FILE: 08D_ parametric equations - sphere.gh

Parametric formulas for surfaces contain two parameters (u, v). The sample points will form a grid rather than a row. This exercise generates a sphere using parametric formulas.

Two sets of parameter sample values are generated using the **range** component, one for each surface direction. The **explode tree** component is used to separate the sample values in two different outputs. The **cross reference** component is used to modify the lists of sample points so that all values match each other creating a grid. The **expression** component is used as in the previous exercises. The **surface from points** component fits a NURBS surface through a set of sample points. These points have to be ordered to form a grid. The I input is set to true so that the surfaces goes through the grid, if set to false the grid will be used as control points.



EXERCISES

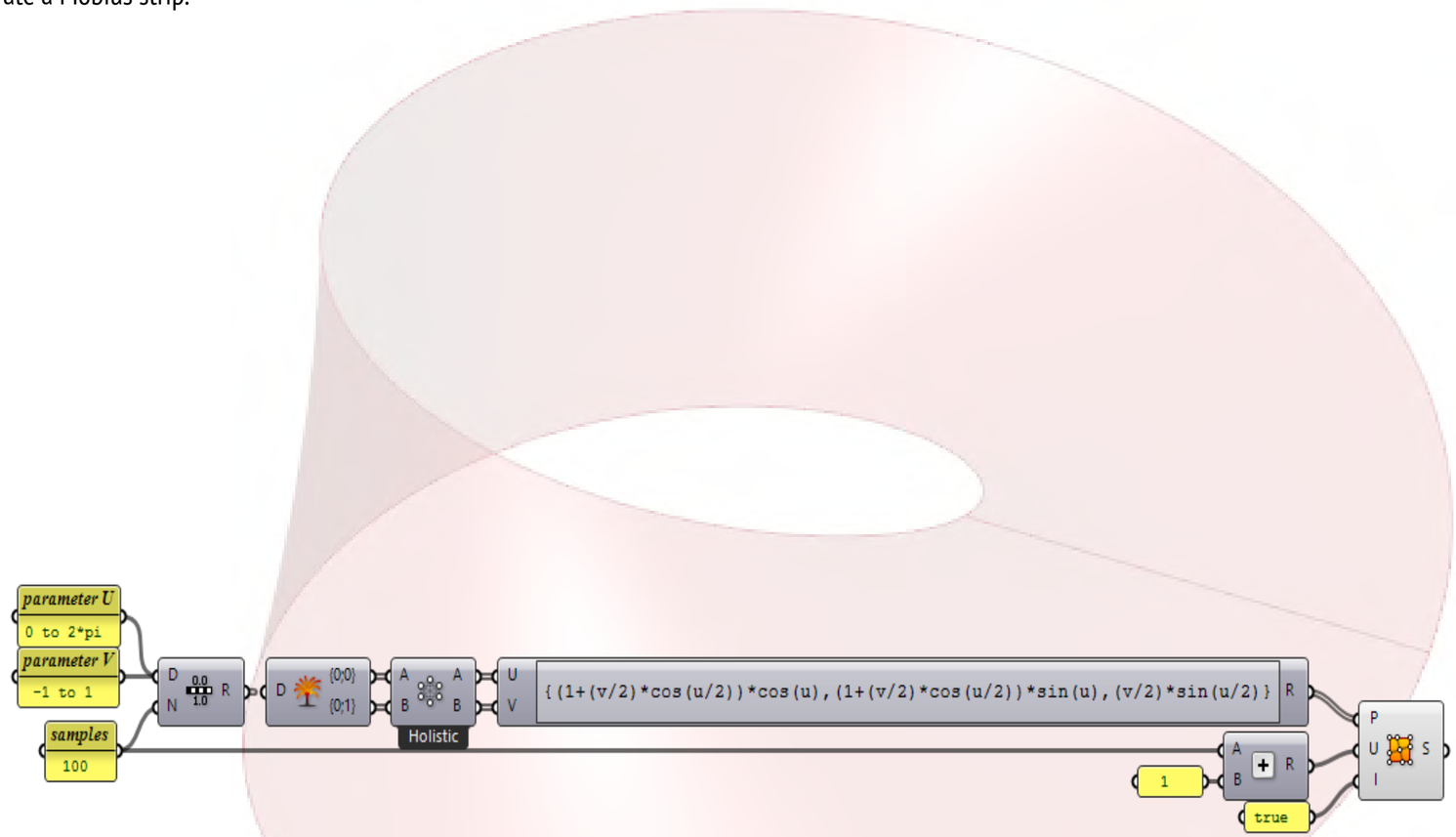
EXERCISE 08 - PARAMETRIC ECUATIONS

08.E - MOBIUS STRIP

FILE: 08E_ parametric equations - mobius strip.gh

In this exercise a set of parametric equations are used to generate a Möbius strip.

> The definition works just as the previous exercise, but with a different set of equations and parameter domains.



EXERCISES

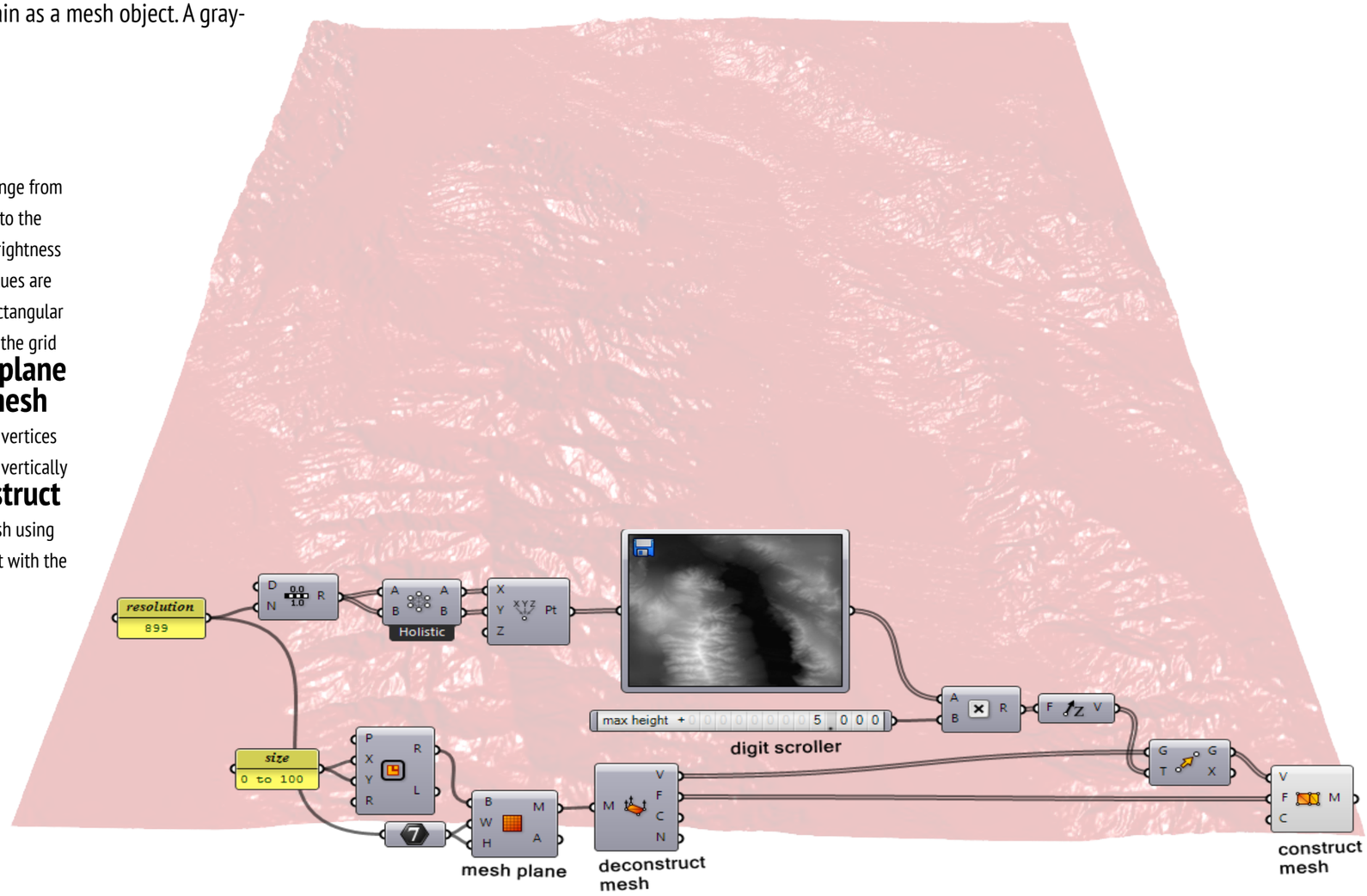
EXERCISE 09 - MESH TERRAIN

09.A - HEIGHTMAP

FILE: 09A_ mesh terrain - heightmap.gh

The following exercise generates the shape of a terrain as a mesh object. A gray-scale image is used for the elevation data.

A grid of points with coordinates that range from 0 to 1 is generated. These point are fed to the **image sampler** component, brightness values are outputted. The brightness values are multiplied by the maximum height. A rectangular mesh with the same number of faces as the grid of points is created using the **mesh plane** component. The **deconstruct mesh** component is used to retrieve the mesh vertices and faces. The mesh vertices are moved vertically using the brightness values. The **construct mesh** component creates a new mesh using the same faces as the previous mesh but with the new vertices.



INDEX

EXERCISES

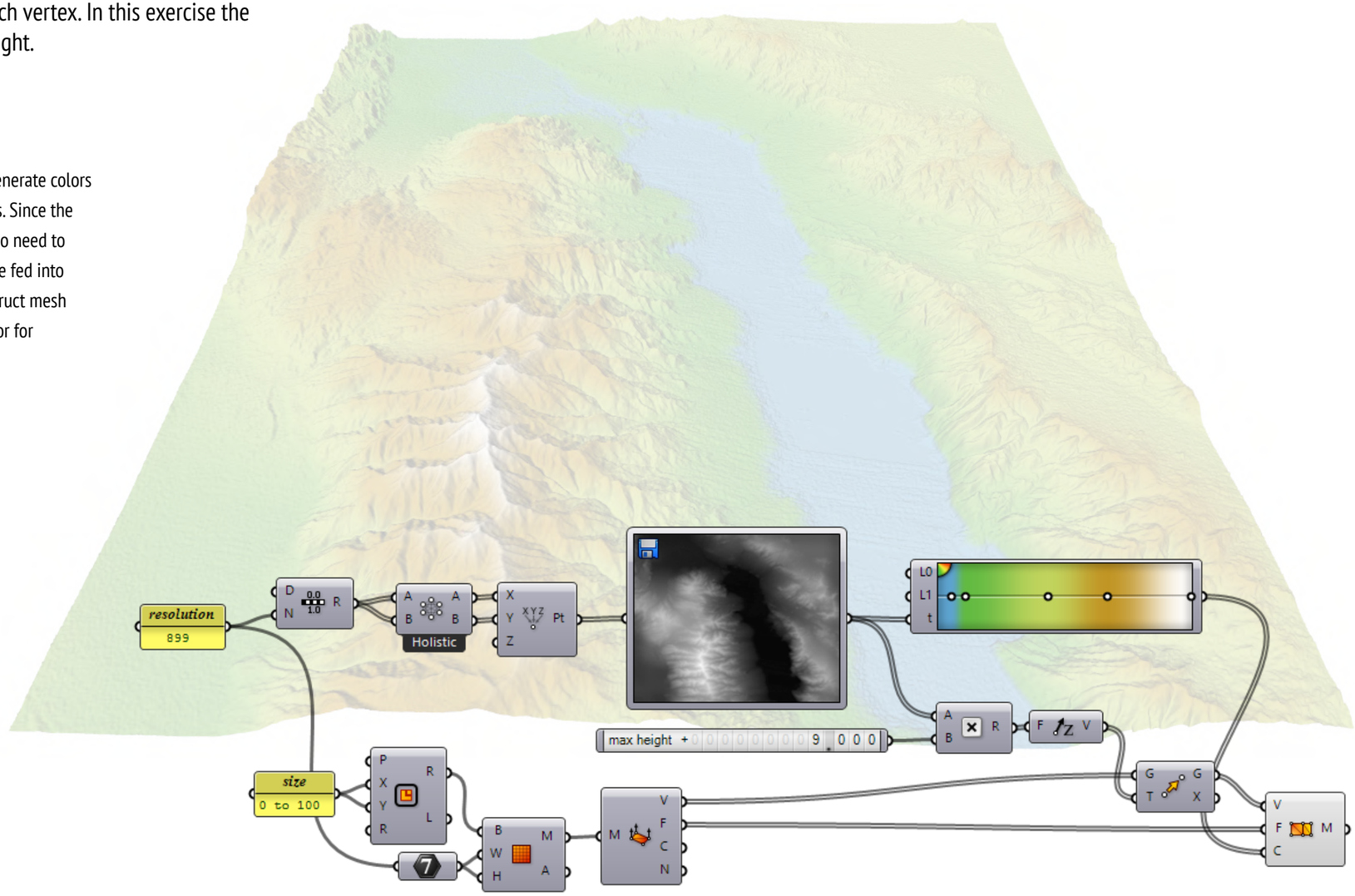
EXERCISE 09 - MESH TERRAIN

09.B - COLOR BY HEIGHT

FILE: 09B_ mesh terrain - color by height.gh

Meshes can be colored by assigning a color to each vertex. In this exercise the mesh vertices are colored depending on their height.

A gradient component is used to generate colors given the **brightness** values. Since the values range from 0 to 1, there is no need to change the extremes. The colors are fed into the **color input** of the construct mesh component. There must be one color for each **vertex**.



EXERCISES

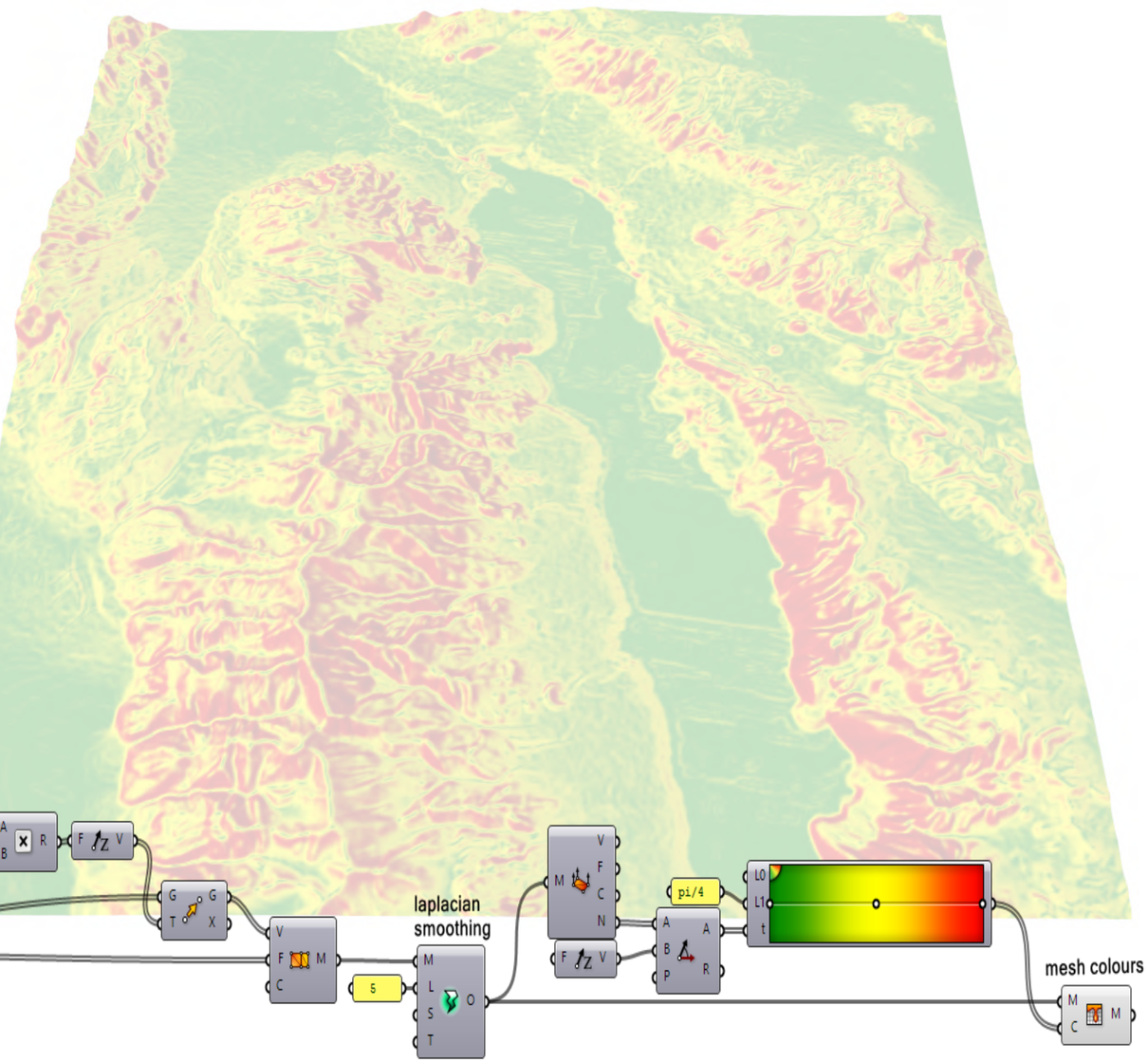
EXERCISE 09 - MESH TERRAIN

09.C - COLOR BY SLOPE

FILE: 09C_ mesh terrain - color by slope.gh

In this exercise the mesh vertices are colored depending on its slope. The slope can be found by measuring the angle between the vertex normal and the Z vector.

The mesh is smoothed using the **Laplacian smoothing** component to reduce noise. The **deconstruct mesh** component is used to retrieve the vertex normals. The **angle** component returns the angle between the vertex normals and the Z vector. A **gradient** component generates colors from the angle values. The **mesh colors** component assigns a color to each vertex.



INDEX

EXERCISES

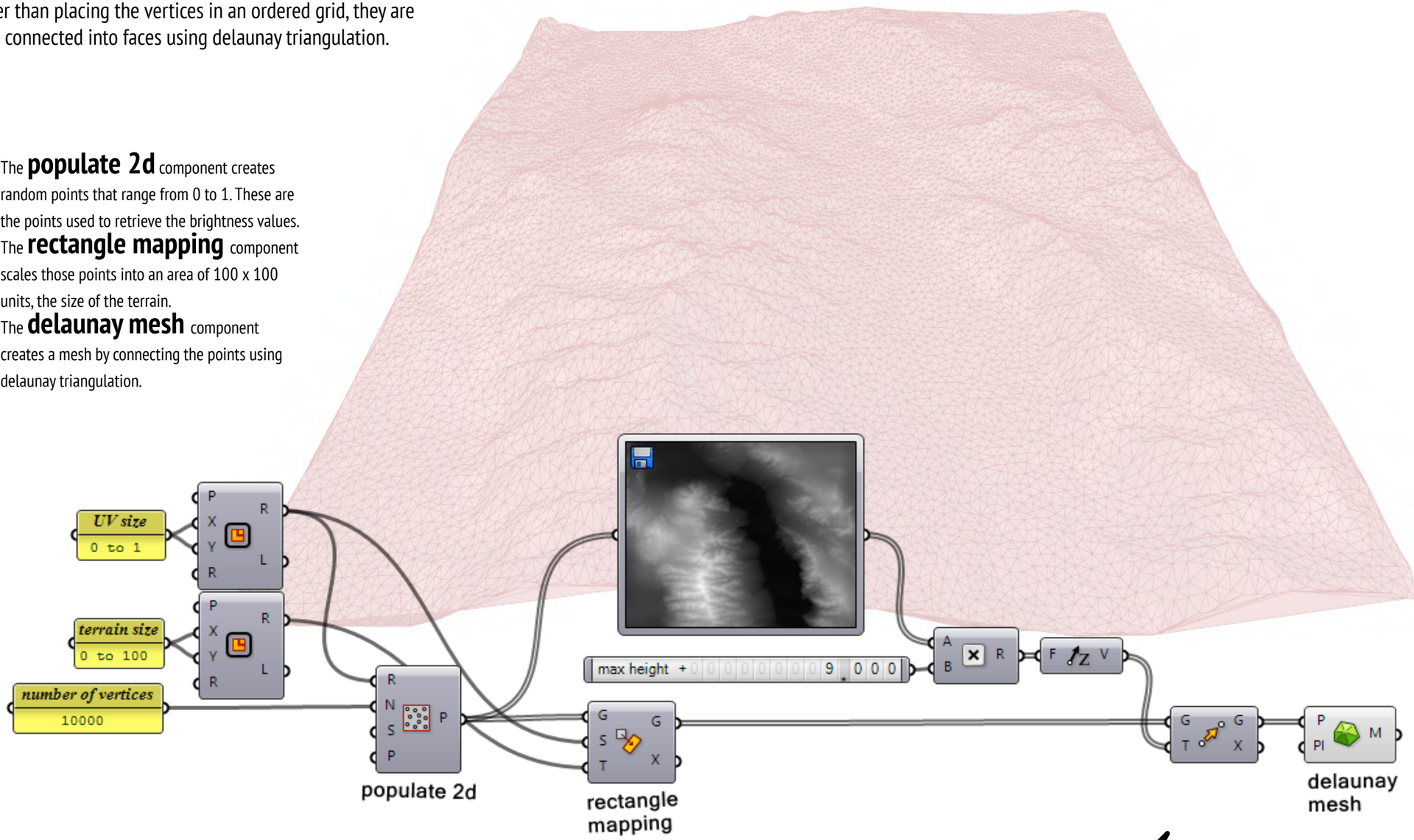
EXERCISE 09 - MESH TERRAIN

09.D - DELAUNAY TRIANGULATION

FILE: 09D_ mesh terrain - delaunay.gh

In this exercise, rather than placing the vertices in an ordered grid, they are placed randomly and connected into faces using delaunay triangulation.

The **populate 2d** component creates random points that range from 0 to 1. These are the points used to retrieve the brightness values.
The **rectangle mapping** component scales those points into an area of 100 x 100 units, the size of the terrain.
The **delaunay mesh** component creates a mesh by connecting the points using delaunay triangulation.



INDEX

EXERCISES

EXERCISE 10 - MESH SUBDIVISION

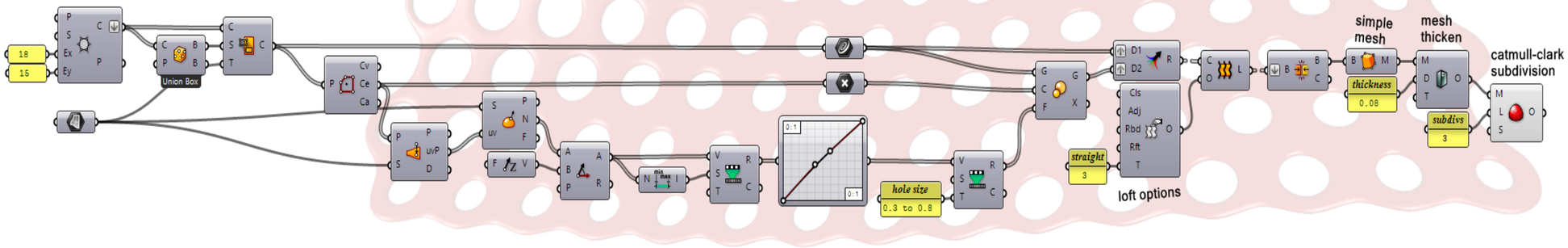
10.A - HOLES ON SURFACE

FILE: 10A_ mesh subdiv - holes.gh

In this exercise a hexagonal grid is used over a surface to create holes of different sizes depending on its slope. The hexagons are smoothed using Catmull-Clark mesh subdivision turning them into circular holes.

The **map to surface** component is used to map a hexagonal grid to the surface.
To calculate the slope of each hexagon, its center is projected to the surface retrieving its corresponding UV point. The UV point is used to find the normal to the surface. The angle between the normal and the Z vector is the slope. **Remap** and **graph mapper** components are used to transform the angles into scale values. The **scale** component is used to scale the hexagons given its slope. The hexagons are lofted together using the **loft** component. The loft type is set to straight.

All resulting lofts are joined into a single brep using the **brep join** component. Since all the faces of the brep are untrimmed and have 4 edges, the **simple mesh** component can be used to transform it to a clean mesh. Thickness is added to the mesh using the **mesh thicken** component. Finally, the **Catmull-Clark** component is used to subdivide the mesh and smooth the shape into circular holes.



INDEX

EXERCISES

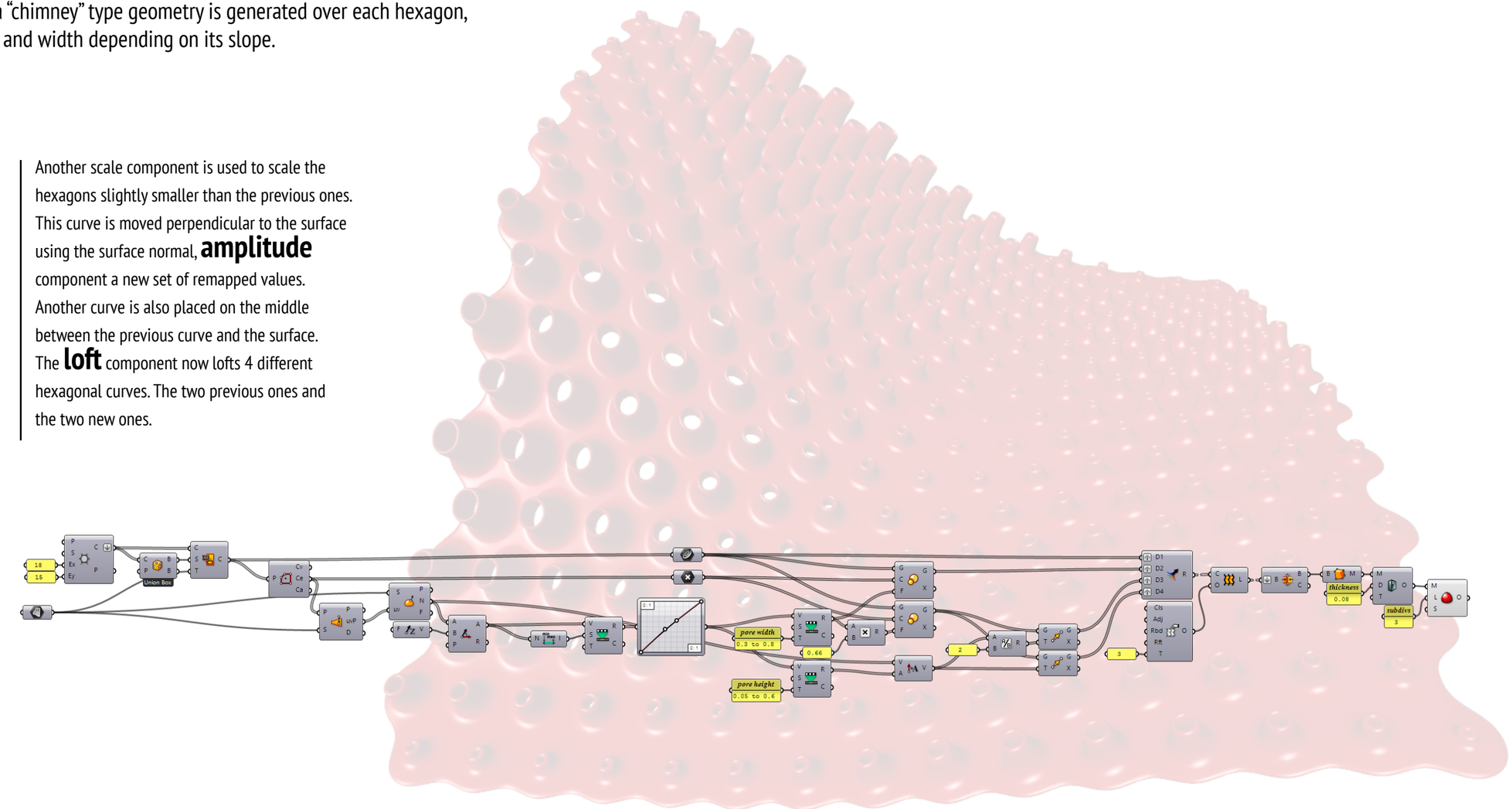
EXERCISE 10 - MESH SUBDIVISION

10.B - POROUS SURFACE

FILE: 10B_ mesh subdiv -pores.gh

In this exercise a "chimney" type geometry is generated over each hexagon, changing height and width depending on its slope.

Another scale component is used to scale the hexagons slightly smaller than the previous ones. This curve is moved perpendicular to the surface using the surface normal, **amplitude** component a new set of remapped values. Another curve is also placed on the middle between the previous curve and the surface. The **loft** component now lofts 4 different hexagonal curves. The two previous ones and the two new ones.



EXERCISES

EXERCISE 10 - MESH SUBDIVISION

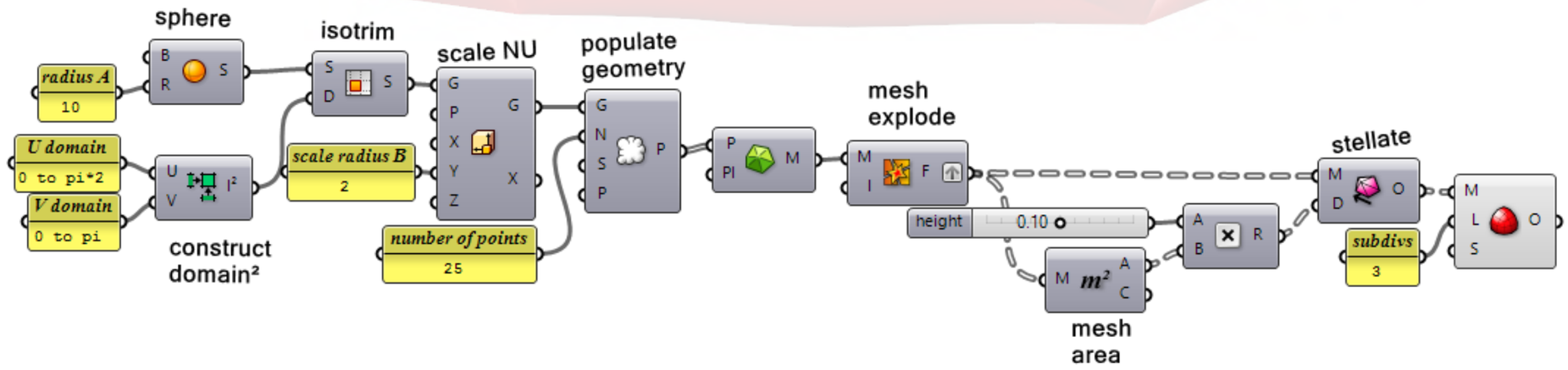
10.C - PNEUMATIC TRIANGLES

FILE: 10C_ mesh subdiv - inflatable.gh

In this exercise a series of triangles, placed randomly over the top half an ellipsoid, are inflated.

A sphere is created using the **sphere** component. The **isotrim** component trims the bottom half of the sphere. The scale NU component scales the sphere only on the Y axis creating an ellipsoid. The **populate geometry** component places random points over the surface. The **delaunay mesh** component connects the random points creating a triangular mesh. The explode mesh component separates each triangle into an individual mesh.

The **mesh area** component is used to calculate the area of each triangle. The area is used as a variable in the inflation distance so that larger triangles will be inflated more. **The stellate** component creates a mesh pyramid on each triangle using height calculated in the previous step. The **Catmull-Clark** component smooths out each pyramid to create a shape that resembles an inflated object.



INDEX

EXERCISES

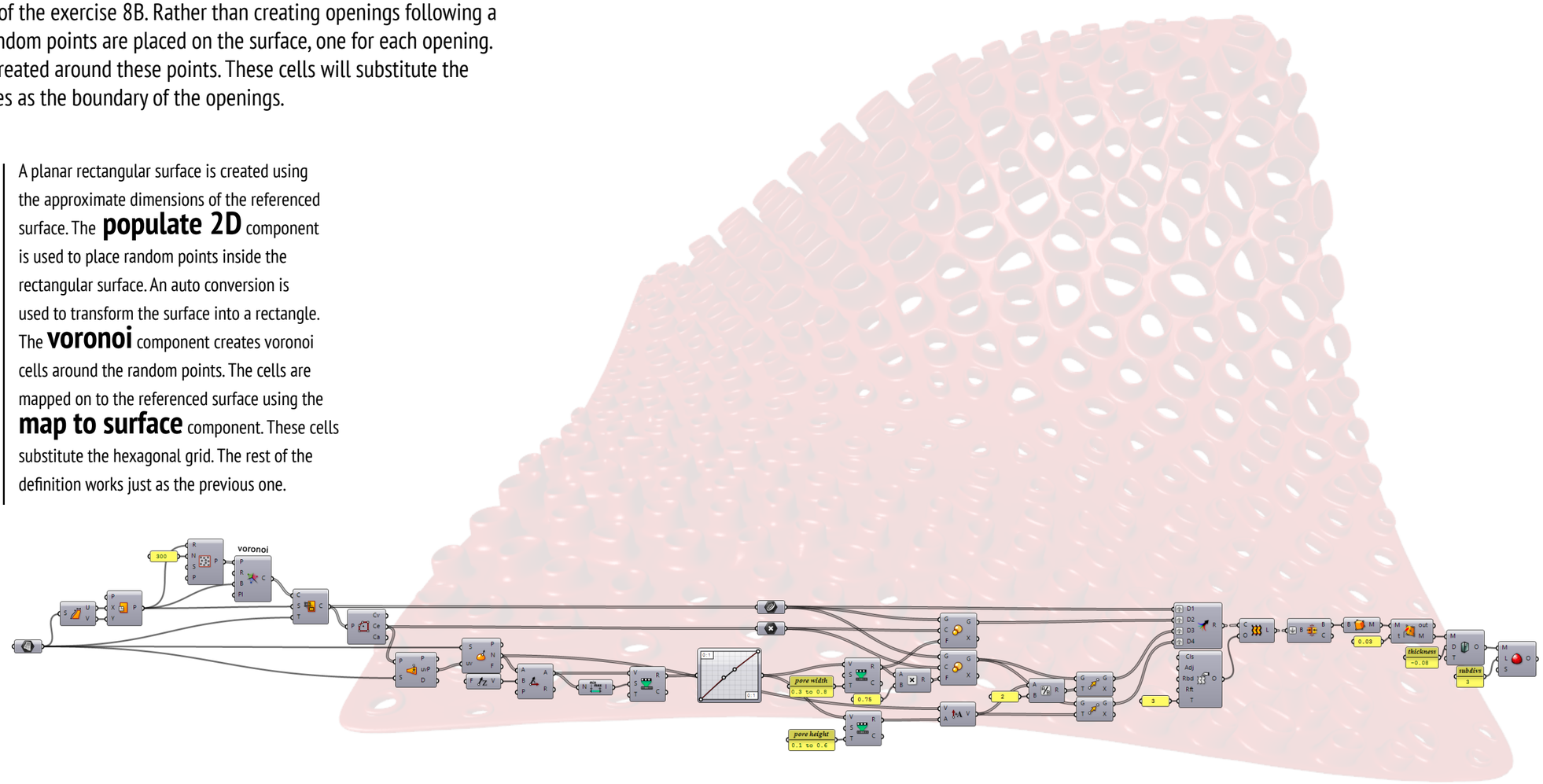
EXERCISE 10 - MESH SUBDIVISION

10.D - VORONOI CELLS

FILE: 10D_ mesh subdiv - voronoi.gh

This is a variation of the exercise 8B. Rather than creating openings following a hexagonal grid, random points are placed on the surface, one for each opening. Voronoi cells are created around these points. These cells will substitute the hexagonal polylines as the boundary of the openings.

A planar rectangular surface is created using the approximate dimensions of the referenced surface. The **populate 2D** component is used to place random points inside the rectangular surface. An auto conversion is used to transform the surface into a rectangle. The **voronoi** component creates voronoi cells around the random points. The cells are mapped on to the referenced surface using the **map to surface** component. These cells substitute the hexagonal grid. The rest of the definition works just as the previous one.



INDEX

TOOLS

PHON

Title **Grasshopper**

Typeface
Heads **Dinpro**
Body **Pt Sans Narrow**

Date **2013/2014**

INDEX