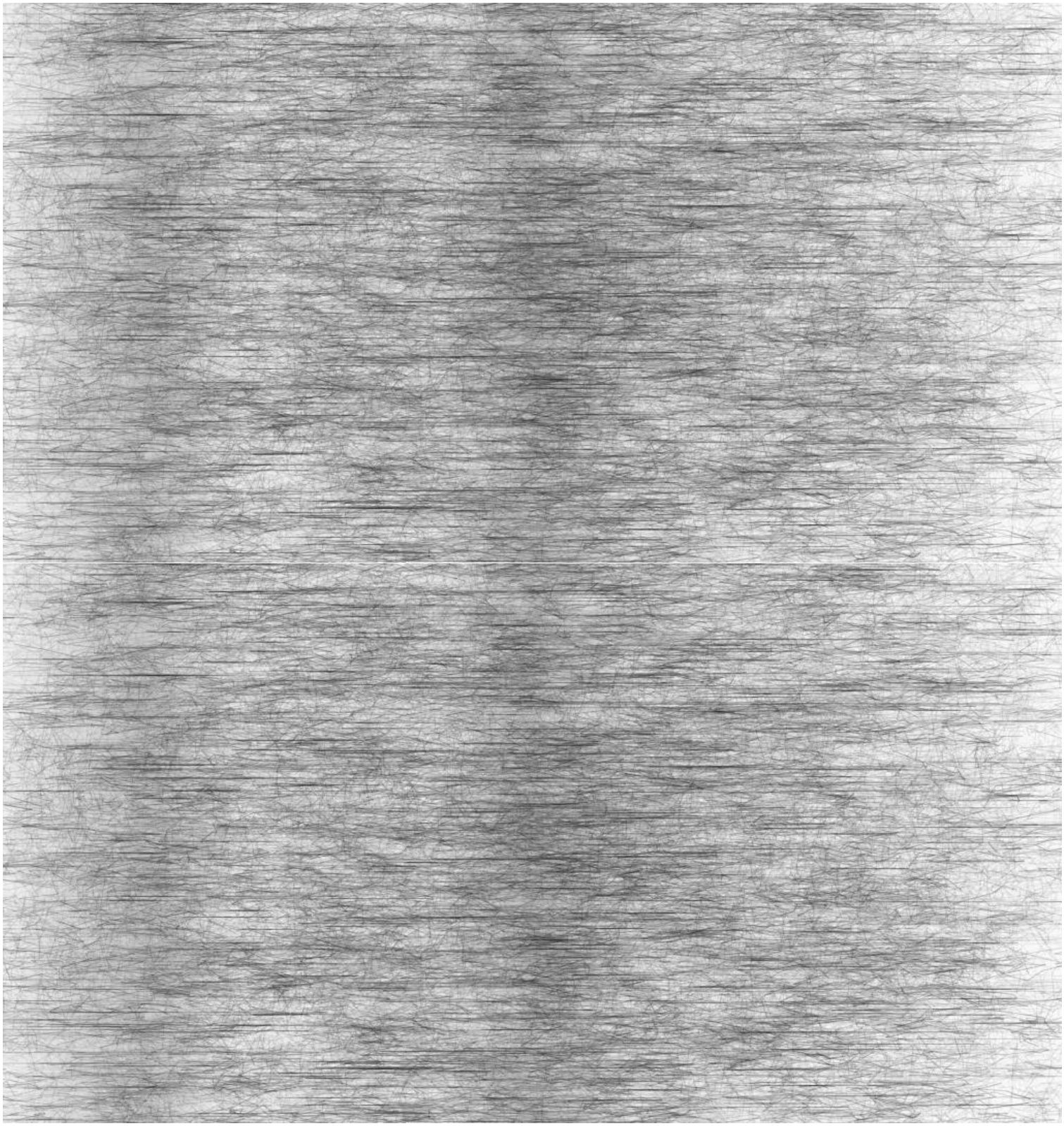


Millipede

Millipede March2014 release

www.sawapan.eu

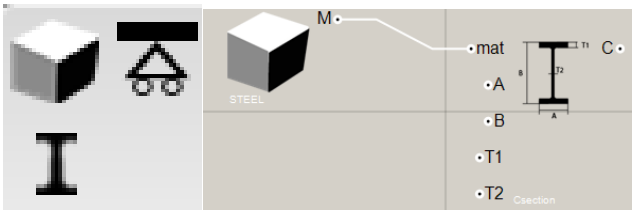
Millipede is a Grasshopper™ component focusing on the analysis and optimization of structures. In addition it exposes functionality that is relevant to the solution of many numerical and geometric problems.

α.Overview

At the core of this component is a library of very fast structural analysis algorithms for linear elastic systems. The library contains its own optimization algorithms based on topology optimization but due to its speed it can be used in combination with Galapagos for solving generic form finding problems [see included examples].

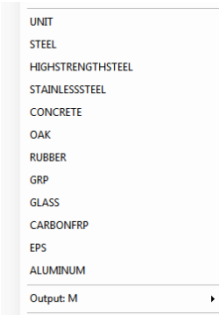
β. Components Overview

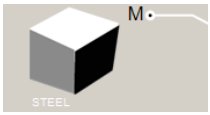
1. Stock Objects



Stock objects are shared by all analysis modules and are used in order to define materials, boundary conditions and cross sections.

a. 1. Material Definition

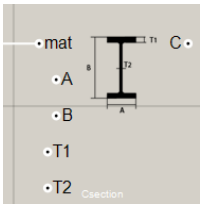




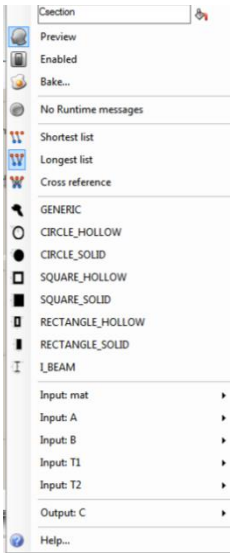
The material component allows you to select a stock material. This can be connected as an input to cross sections and shell elements in order to assign materials to the system.

You can right click on the material component to select one of a pre-set number of materials

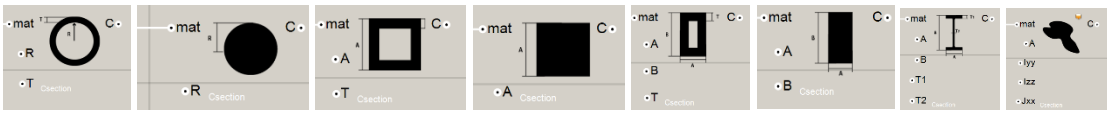
b. Cross Section Definition



The Cross Section component generates cross section definitions to be assigned to frame elements.



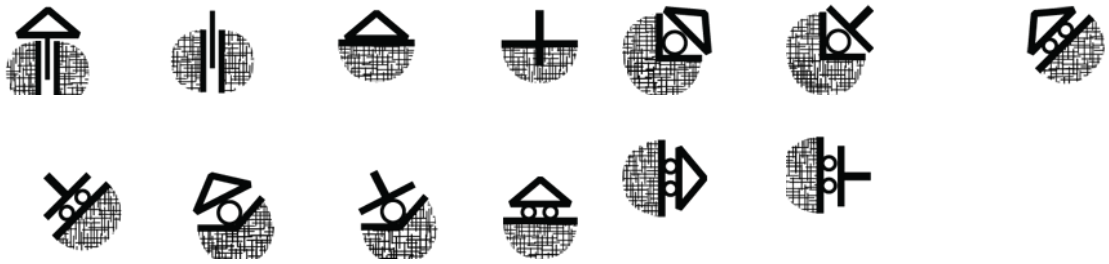
You can right click on the Cross Section component in order to choose a cross section type. Each time you change the cross section type the input parameters of the component are changed so as to reflect the different geometric properties of each section.



All types of cross sections receive a material  definition as their first input. The rest of the parameters are determine the dimensions according explained in each section type's diagram icon. All dimensions are in SI units [meters].

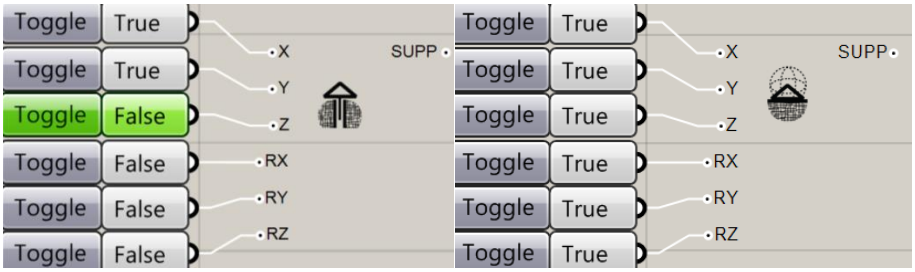
The special case of a generic cross section  allows one to input manually the Area, Moments of Inertia and Torsional constant [m^2 and m^4].

c. Boundary Condition Definition

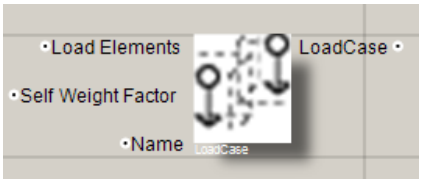


This component is used in order to define boundary conditions that fix different degrees of freedom of a given structure. The support component takes as inputs 6 Boolean values that determine which degrees of freedom are fixed at a point. XYZ represent translations along the corresponding axes and RX RY RZ represents the rotations around the corresponding axes.

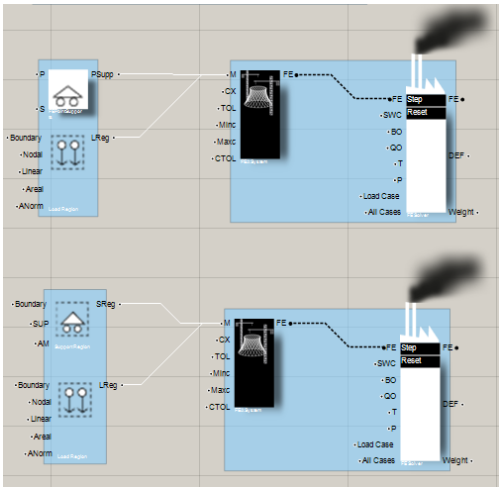
The component generates an integer which is a bitmask of the 6 on/off states [RZ RY RX TZ TY TX].



d. Load case definition 

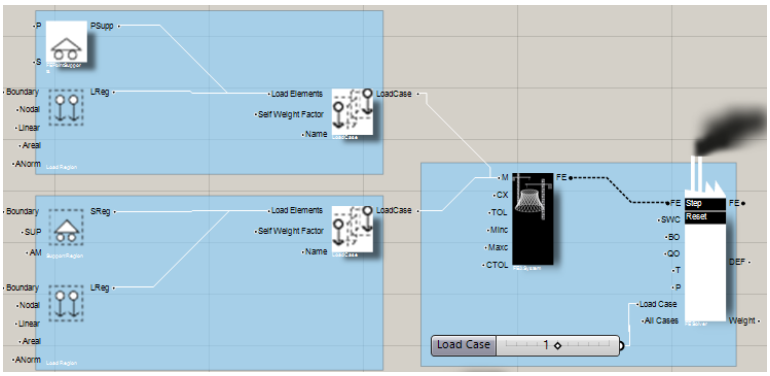


This component allows you to define a load case. Instead of connecting the loads and supports directly on the composition component  you first connect groups of loads and supports on loadcase and then connect those loadcases to the composition component. In this way you can select between different loadcases from the solver component. In addition this will allow you to run the optimization algorithms with multiple loadcases.



if you wanted to analyze the same structure with two different sets of supports and loads [e.g. under selfweight and then under selfweight + wind load] you would need duplicate composition and solver elements. In addition you wouldn't be able to combine the results from the two analyses for topology or sizing optimization.

Using the Load case component you would reconfigure the above in the following way:



The scrollbar connected to the solver allows you to select which loadcase is being analyzed. In addition any optimization can take into account all loadcases simultaneously if the "allCases" input is set to true.

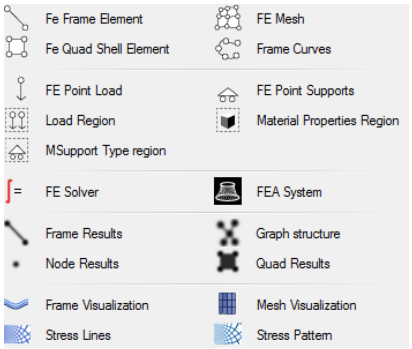
Inputs:

LoadElements [Generic]:  Any number of the outputs of these components are aggregated here.

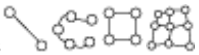
Self Weight Factor[Number:]This is the multiplication coefficient for the self-weight when this loadcase is used.

Name [string]: The name of the load case

2. Frame and Shell Element Analysis

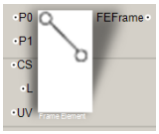


This toolbox contains the components necessary to build analyse and optimize structures composed of frame elements [beams, columns, trusses etc…] and shell elements.

e. Geometry constructors 

These elements apply structural properties to Rhino geometry in order to be passed as inputs in the structural system builder component.

1. Explicit Frame Element



This component creates a frame element connecting two points with a specified cross section, distributed linear load and cross section orientation.

Inputs:

P0 [Point3d]: The start point of the frame element

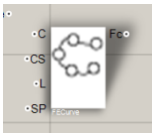
P1 [Point3d]: The end point of the frame element

CS [Cross Section]: The cross section, input from  component

L [Vector3d]: The distributed linear load along the element in global coordinates [N/m]

UV [Vector3d]: This is the vector that determines the orientation of the Y axis of the cross section along this element

2. Curve



This component assigns material properties to a curve object that will be converted into a series of frame elements during the system building phase.

Inputs:

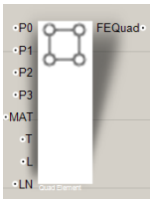
C [Curve]: The Curve geometry

CS [Cross Section]: The cross section, input from  component

L [Vector3d]: The distributed linear load along the element in global coordinates [N/m]

SP [Integer support code]: A support type  to be applied to all nodes that will be generated along this curve

3. Explicit Shell Quad Element



This component defined a single quad element for shell analysis

Inputs:

P0, P1, P2, P3 [Point3d]: The four corners given in counter clockwise order

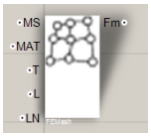
MAT [Material]: A material definition coming from component

T [Thickness]: The thickness of the shell element [m]

L [Vector3d]: Distributed area load in global coordinates [N/m2]

LN [Number]: Normal area load [N/m2]. Useful for defining pressure

4. Mesh to Shell



This component parses a Mesh and generates a quad shell element for each face in the mesh. If the mesh contains any triangular faces then the whole mesh is subdivided into quads before further processing. This may result in a tripling or quadrupling of the number of elements in the mesh.

!For best results make sure that the mesh does not contain elongated triangles or quads that are too twisted or faces that are too small. You can tweak the meshing parameters in rhino [especially the min/max edge lengths] before meshing any surface geometry.

Inputs

MS [Mesh]: The input mesh

MAT [Material]: A material definition coming from component

T [Thickness]: The thickness of the shell elements [m]

L [Vector3d]: Distributed area load in global coordinates [N/m2]

LN [Number]: Normal area load [N/m2]. Useful for defining pressure

f. Property modifiers 

These components assign properties at specific nodes in the system. Make sure that the points you pass to these components coincide with actual nodes in the system [end point of curves or frame element or vertices of meshes]

1. Applied point load

Applies a point load at a specific location in the structural system

Inputs:

P [Point3d]: Point of load application

L [Vector3d]: Load [N]

2. Point support type

This component applies a specific boundary condition at a location in space

Inputs:

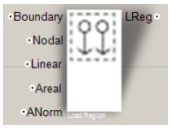
P [Point3d]: Location of support

S [Integer support code]: A support type  to be applied at the specified location

g. Region Property modifiers 

These modifiers apply a certain property within a region determined enclosed by a solid object [polysurface].

1. Region of Loads



Apply the specified types of loads to all elements that fall within the specified solid.

Inputs:

Boundary [Brep]: A closed solid geometry from rhino

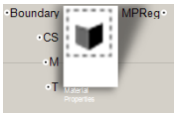
Nodal [Vector3d]: Point load to be applied to all nodes within boundary [N]

Linear [Vector3d]: Linear load to be applied to all frame elements within boundary [N/m]

Areal [Vector3d]: Area Load to be applied to all shell elements within boundary [N/m2]

ANorm [Number]: Normal Area Load to be applied to all shell elements within boundary [N/m2]

2. Region of Material properties



Apply the specified material properties to all elements that fall within the specified solid.

Inputs:

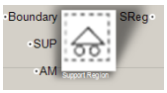
Boundary [Brep]: A closed solid geometry from rhino

CS [Cross Section]: The cross section, input from  component to be applied to all frame elements within boundary

M [Material]: A material definition coming from  component to be applied to all shell elements within boundary

T [Number]: A thickness to be applied to all shell elements within boundary

3. Region of Support



Apply the specified support types to all nodes that fall within the specified solid.

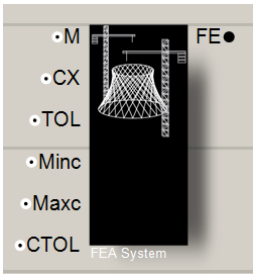
Inputs:

Boundary [Brep]: A closed solid geometry from rhino

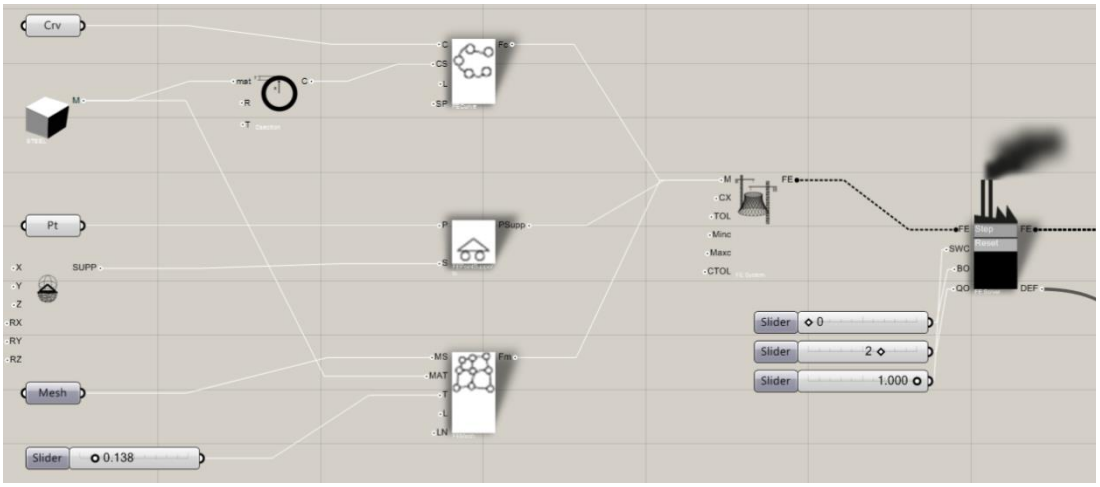
SUP [Integer support code]: A support type  to be applied to all nodes within boundary

AM [Boolean]: A value indicating whether support types should have an additive behaviour or just replace whatever support is already there.

h. System builder



This is the core component that aggregates all the information [geometry, properties etc…] and builds the actual analysis model but it does not initiate any analysis itself. For the actual analysis to happen you need to connect the output of this component to the input of the Solver component. The reason for this separation is that by separating the two components one can modify the analysis parameters without rebuilding all the geometry which is beneficial for certain optimization scenarios.



Inputs:

M [Generic]:  Any number of the outputs of these components are aggregated here.

The rest of the inputs determine how curves are going to be intersected and discretized in order to yield connected frame elements

CX [Boolean]: Intersect curves, if true all curve objects are intersected and split against each other so that nodes are inserted at the intersections

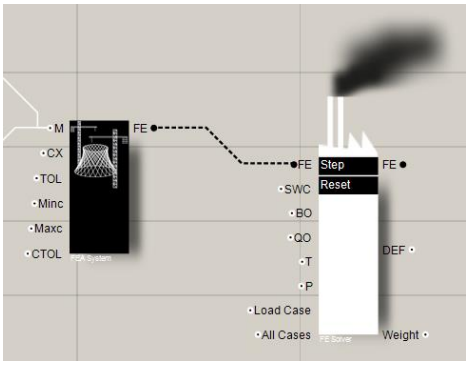
TOL [Number]: Tolerance for curve intersection and merging of points

Minc [Number]: Minimum allowed length for a frame element

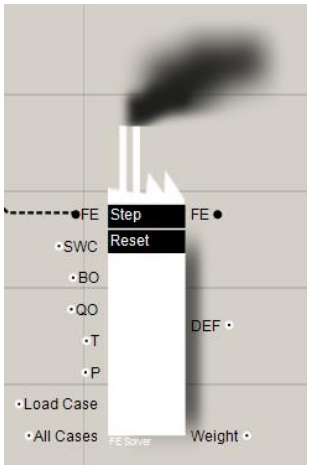
Maxc [Number]: Maximum allowed length for a frame element

CTOL [Number]: Tolerance that controls how close generated polylines approximate the original curves. Smaller values will result in more accurate representations at the expense of more frame elements and slower calculations.

i. Solver



The solver constitutes the slowest step in the process which involves the analysis of the finite element system and recovery of results. It must be connected FE->FE to a system builder component. In addition to its inputs it has two buttons that are used in optimization [when determining optimal sizes and thicknesses for frame and shell elements]



Inputs:

FE [StatFESystem]: The Finite element model coming from a system builder component

SWC [Number]: Self Weight Coefficient. It determines whether the self-weight of the structure is taken into account during the analysis and how much it will affect the final outcome. A value of 1.0 means that the self-weight is applied without modification, while a value of 0 means that it will not be taken into consideration. Values larger than 1.0 are also permitted.

BO [Integer]: A number determining how many iterations of Frame element size optimization should occur each time the user presses the “Step” button

QO [Integer]: A number determining how many iterations of Shell element thickness optimization should occur each time the user presses the “Step” button

T[Number]: The target percentage [a number between 0.0 and 1.0] for overall frame or shell mass to remain after optimization

P[Number]: A penalization number [see topology optimization components]. Usually 1.5 to 3.0

Load Case [Integerc]:This number determines the active loadcase to be analyzed or optimized.

All Cases [Boolean]: If true , during optimization all connected load cases will be taken into account.

Outputs:

FE [StatFESystem]: The analysed Finite Element system to be passed to other components that deal with visualization and recovery of specific results.

DEF [Number]: The maximum deflection in meters of the analysed structure. Useful for quick troubleshooting and getting a quick rough measure of system performance without doing detailed analysis of the results. If this number is too big there may be disconnected elements in the system and if it is zero you may haven't applied loads correctly.

Weight[Number]: The overall weight of the structure in Kilograms.

Controls:

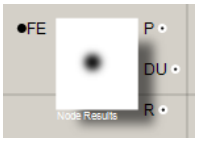
Step: When pressed the control carries BO steps of frame element size optimization and QO steps of shell element thickness optimization.

Reset: Reset all sizes and thickness to original [non optimized] state.

j. Results 

These components can be used to extract numerical results from the solver.

1. Nodal results



This component returns the locations, displacements and rotations for all nodes.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component 

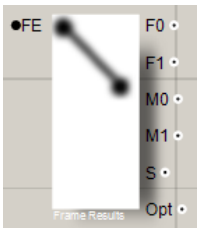
Outputs:

P [Point3d]: The locations of nodes

DU [Vector3d]: The displacements of all the nodes [m]

R [Vector3d]: The rotations around the XYZ axes for each node as a vector.

2. Frame results



This component generates the results for all frame elements [beams, column etc…] in the system.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component 

Outputs:

F0 [Vector3d]: The internal force at the start point in the local coordinate system. The X component is the axial force and the Y and Z components determine the shear force vector [N]

F1 [Vector3d]: The internal force at the end point in the local coordinate system [N]

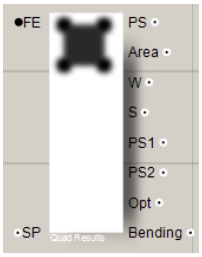
M0 [Vector3d]: The moment at the start point in local coordinate system. X component is torsion [due to rotation around the axis of the element] while the Y and Z determine the plane and magnitude of the bending moment [N.m]

M1 [Vector3d]: The moment at the end point in local coordinate system [N.m]

S [Number]: An estimate of the maximum normal stress [combined axial force and bending moment contributions] along the element [N/m2]

Opt [Number]: The optimization value. A number from 0.0 to 1.0 that determines the relative importance [or required size] of each frame element.

3. Shell Quad results



This component generates the results for all shell elements in the system.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component

SP [Number]: the layer at which principal stresses are calculated. This is a number form -1.0 [bottom layer] to +1.0 [top layer]. For shell elements stresses throughout each quad. The component calculates the stresses along a line that passes through the centre of each quad and is parallel to its normal. With this parameter you can choose whether the results returned are at the top or bottom layer of the shell. For example for a slab supported at two sides you should expect to see a lot of tension on the bottom layer and a lot of compression at the top.

Outputs:

PS [Plane]: A plane with its origin at the center of each quad and its axes aligned with the principal stress directions

Area [Number]: the area of each quad [m2]

W [Number]: the normal displacement at the center of each quad [m]

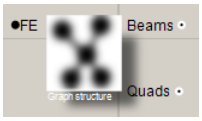
S [Number]: Estimated Maximum VonMises stress for all layers of each quad [N/m2]

PS1 [Number]: First principal stress. Positive values designate Tension and negative compression [N/m2].

PS2 [Number]: Second principal stress. Positive values designate Tension and negative compression [N/m2].

Opt [Number]: A number from 0.0 to 1.0 designating the relative strength of thickness multiplier during the optimization process.

4. Connectivity graph



This component generates two trees representing the connectivity graphs for frame elements and shell elements. Nodes are referenced by integer numbers that correspond to the ordering of nodes of the output of the node results component

Inputs:

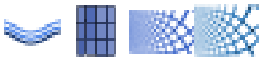
FE [StatFESystem]: The analysed Finite Element system coming from the solver component

Outputs:

Beams [Integer]: A tree structure with the same number of branches as there are frame elements in the system. Each branch contains two integer numbers designating the nodes connected by the corresponding element

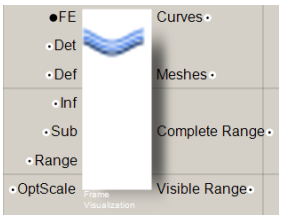
Quads [Integer]: A tree structure with the same number of branches as there are Quad shell elements in the system. Each branch contains four integer numbers designating the nodes connected by the corresponding element

k. Visualization



These components offer a variety of ways to visualize results from the analysis model

1. Frame Elements Visualization



This component offers a quick way to visualize the deformations in the frame elements of the system and see how different forces and stresses vary along the frame elements and around their cross sections.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component ∫=

Det [Boolean]: If true a detailed visualization of the deformations and results along each frame is generated. This involves sampling the element at intermediate points along its length generating more complex meshes.

Def [Number]: The deflection factor. You can use this number to exaggerate the calculated displacements [usually in the order of mm] in the structure so that you can observe the deformation from its original state to its equilibrium state. Usually you can set this number to something like 1.0/maximumdeflection to get a visible effect.

Inf [Number]: This factor determines the inflation of the cross section along each element and along the direction of maximum stress giving an immediate visual feedback about the distribution of stresses throughout each element as well as a suggestion for possible use of variable cross sections.

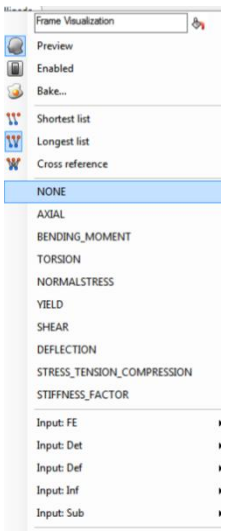
Sub [Number]: This number determines the length of the subdivisions of each element in detailed visualization mode. The smaller it is the more subdivisions will be generated at the expense of slower calculations and more complex meshes.

Range[Interval]: This input is optional. If it is set it determines the minimum and maximum values that will be used to determine the colours of the visualization. If it is not set the component will determine a range from the data, for example for stress analysis it will use the minim and maximum stress. this can be problematic when there are high stress concentrations.

OptScale[Boolean]: If true the cross sections of the elements are scaled to reflect the sizing optimization results [if used] instead of the original sizes]

Outputs:

Curves [Polyline]: A list of polylines representing the deflected shapes of the centrelines of all the frame elements.



Meshes [Mesh]: A mesh that contains the deflected and coloured shapes for all the frame elements in the system. To determine the colour mode right click on the component and select one of the available colour modes:

Complete Range[Interval]: The range of values for the chosen result

Visible Range[Interval]: the range of values that corresponds to the colour gradient of the mesh. In general this will be the same as the Compete range unless you explicitly set the Range input.

Options:

NONE: White meshes

AXIAL: Meshes are coloured red or cyan depending on whether the dominant effect is compression or tension

BENDING_MOMENT: Rainbow visualization of the distribution of bending moments

TORSION: Rainbow visualization of the distribution of Torsion

NORMAL_STRESS: Rainbow visualization of the distribution of NORMAL_STRESSES

YIELD: Ratio of stress to yield stress for specified material. Red means that the stress is over the yield stress threshold.

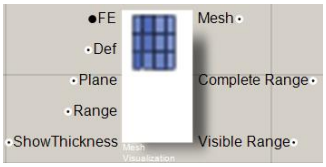
SHEAR: Rainbow visualization of the distribution of Shear forces

DEFLECTION: Rainbow visualization of the distribution of deflections

STRESS_TENSION_COMPRESSION: Detailed stress analysis throughout the cross section of each element revealing the distribution of tension and compression [cyan / red] as a result of a combination of axial forces and bending moment

STIFFNESS_FACTOR: This is the optimization calculated factor that determines the relative strength, size or importance of the frame element in the overall system

2. Shell Elements Visualization



This component generates meshes that visualize the distribution of forces and stresses over shell elements

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component ∫=

Def [Number]: The deflection factor. You can use this number to exaggerate the calculated displacements [usually in the order of mm] in the structure so that you can observe the deformation from its original state to its equilibrium state. Usually you can set this number to something like 1.0/maximumdeflection to get a visible effect.

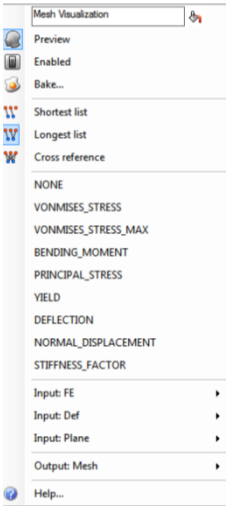
Plane [Number]: A number form -1.0 [bottom layer] to +1.0 [top layer] determining the layer throughout the thickness of each shell element at which the results are calculated [only affects stress results].

Range[Interval]: This input is optional. If it is set it determines the minimum and maximum values that will be used to determine the colours of the visualization. If it is not set the component will determine a range from the data, for example for stress analysis it will use the minim and maximum stress. this can be problematic when there are high stress concentrations.

ShowThickness[Boolean]: if true the mesh will have thickness reflecting the assigned thickness plus the effect of any sizing optimization

Outputs:

Mesh [Mesh]: A mesh that contains the deflected and coloured mesh for all the shell elements in the system. To determine the colour mode right click on the component and select one of the available colour modes:



Complete Range[Interval]: The range of values for the chosen result

Visible Range[Interval]: the range of values that corresponds to the colour gradient of the mesh. In general this will be the same as the Compete range unless you explicitly set the Range input.

Options:

NONE: White meshes

VONMISES_STRESS: Rainbow visualization of the vonmises stress at the specified layer

VONMISES_ STRESS_MAX: Maximum estimated vonmises stress [for all layers]

BENDING_MOMENT: Rainbow visualization of the maximum bending moment at the centre of each quad

PRINCIPAL_STRESS: Red/Cyan visualization depending on whether the dominant effect locally and at the specified layer is tension or compression

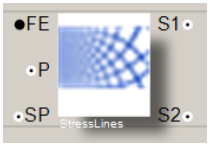
YIELD: Ratio of stress to yield stress for specified material. Red means that the stress is over the yield stress threshold.

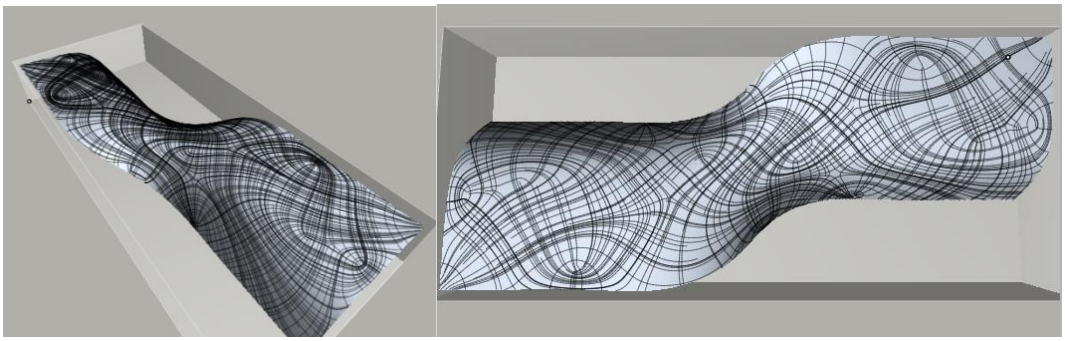
DEFLECTION: Rainbow visualization of the distribution of deflections

NORMAL_DISPLACEMNT: Rainbow visualization of normal displacement [out of plane] at the centre of each quad

STIFFNESS_FACTOR: This is the optimization calculated factor that determines the relative strength, density or thickness of the shell element

3. Stress lines





This component calculates the two principal stress lines emanating from a single point. Stress lines are curves that at each point are tangent to one of the principal stress directions.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component ∫=

P [Point3d]: A point that becomes the starting location for tracing stresslines

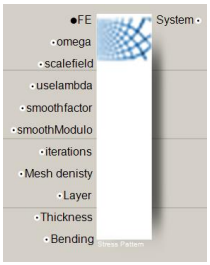
SP [Number]: A number from -1.0 to 1.0 that determines the layer at which the stress lines are traced

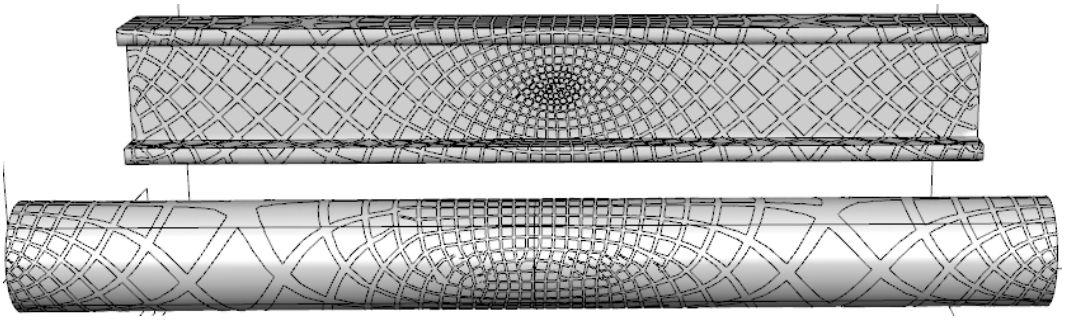
Outputs:

S1 [Polyline]: Principal stress line passing through point P along the 1st principal stress

S2 [Polyline]: Principal stress line passing through point P along the 2nd principal stress

4. Stress pattern





This component generates a pattern that follows the principal stress directions of the analysed meshes. Its output must be connected to one of the reparameterization group components.

Inputs:

FE [StatFESystem]: The analysed Finite Element system coming from the solver component ∫=

omega [Number]: A number that determines the scaling of the resulting pattern.

scalefield [Boolean]: if true the resulting pattern has variable scale allowing to better accommodate singularities without having to add bifurcation in order to account for stretching. In addition scaled patterns naturally reflect concentration of material near supports or highly stressed regions.

[ignore] uselambda[Boolean]: This option has only a marginal effect for highly anisotropic meshes

[ignore] smoothfactor[Number]:

[ignore] smoothModulo[Boolean]:

iterations[Integer]: The number of relaxation steps used for the parameterization. This can be 0 [faster calculation] but for certain difficult meshes might worth setting it to something like 500 or higher.

Mesh density[Integer]: An integer that determines the subdivisions of the resulting curve pattern. It only affects the Curve output component.

Layer[Number]: A number between -1.0 [bottom] and +1.0 [top] that selects a layer to extract the stresses from. For shell elements principal stress direction change throughout the thickness of the elements.

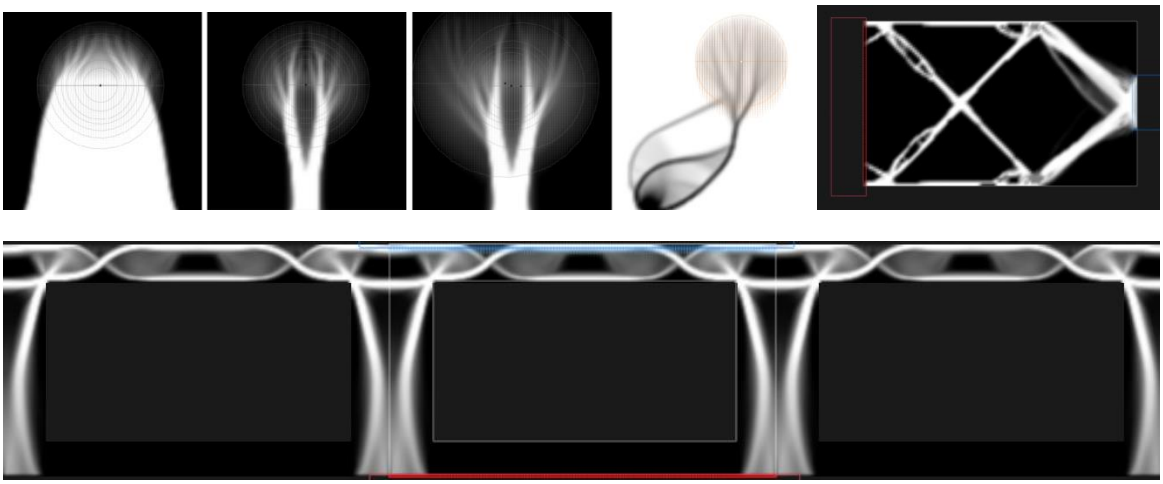
Thickness[Number]: Determines a thickness offset

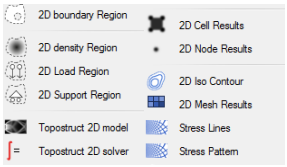
Bending[Boolean]: If true the pattern will follow the principal bending moment directions instead of the principal stress.

Outputs:

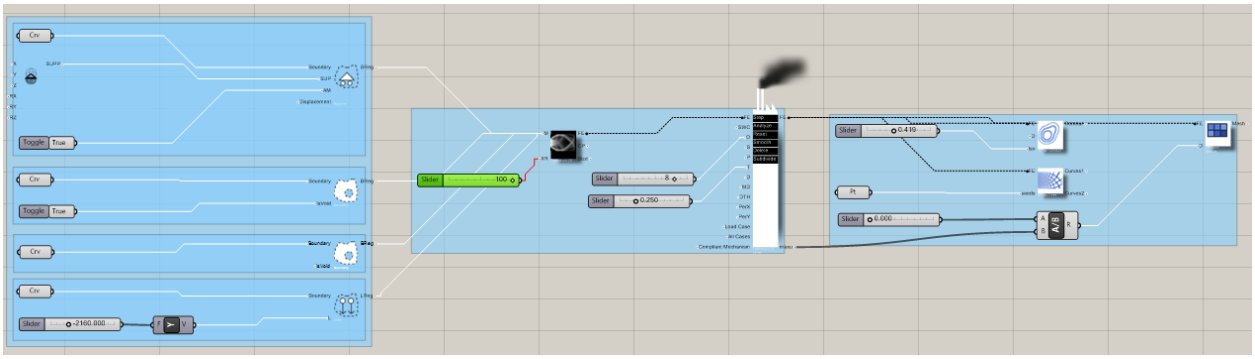
System [Generic]: The reparameterization system. You need to connect to one of the reparameterization components to see the results.

3. 2D analysis of plates with in plane forces and topology optimization





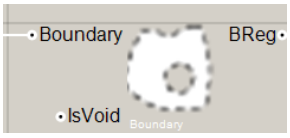
This is a toolbox for analysing and optimizing plates with in plane forces. The analysis is based on a "rasterization" of the problem domain producing a grid of square plate elements. The optimization is done using topology optimization methods.



Similarly to the previous section, a 2d system is constructed in 4 steps:

- Geometry, loads and boundary conditions definition
- System assembly
- Solver
- Results

1. Geometry constructors



This component uses closed curves to define the boundary [or holes] of the system

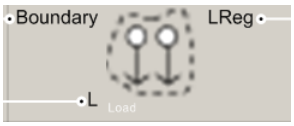
Inputs:

Boundary [Curve]: Closed planar curves to be converted to boundary elements determining the domain of the problem.

IsVoid [Boolean]: If true the boundary designates a hole otherwise it is part of the domain

m. Region Property modifiers 

1. Load Region



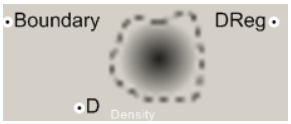
Designate a region where area loads are applied

Inputs:

Boundary [Curve]: Closed curve defining the region within which the loads are applied

L [Vector3d]: Load, only the X and Y components are used

2. Fixed Density Region



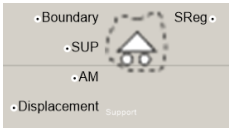
Designate a region where material density is constant [will remain unaffected during optimization]

Inputs:

Boundary [Curve]: Closed curve defining the region within which the density is fixed

D [Number]: Density as a number from 0.0 [no material] to 1.0 [full material]

3. Support Region



Designate a region where certain boundary conditions apply

Inputs:

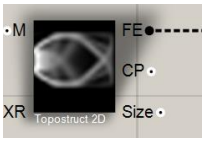
Boundary [Curve]: Closed curve defining the region within which to apply boundary conditions

SUP [Integer support code]: A support type  to be applied to all nodes within the boundary

AM [Boolean]: A value indicating whether support types should have an additive behaviour or just replace whatever support is already there.

Displacement [Vector]: using this input you can determine a fixed imposed displacement on the selected nodes.

n. **System builder**



This is the main model builder tool. It aggregates all the information from the previous components, rasterizes the domain and generates the finite element model.

Inputs:

M [ ]: Any number of regions with properties chosen form the topostruct 2D toolbar can be connected at this input.

XR [Integer]: The resolution along the x axis. This is the number of square plate elements to generate along the x axis of the bounding box of the input geometries. This number can dramatically increase the time and memory requirements of the solver. Values between 16 and 150 are generally good depending on the trade-off between speed and detail of results.

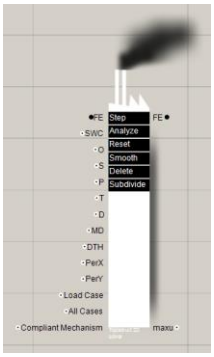
Outputs:

FE [Topostruct 2D model]: The analysis model

CP [list of Points]: A list of the centers of all the generated quad elements.

Size [Number]: The side length of each generated element

o. **Solver**



The solver is the core analysis component in this toolbox. It needs to be connected to a system builder component from which it acquires the finite element system.

Inputs:

FE [Topostruct 2D model]: A model provided from a system builder component

SWC [Number]: Self Weight Coefficient. A number that multiplies all the loads due to the self weight of the structure. For most optimization problems it should be set to 0 as it can induce a non-convergent feedback behavior in the optimization algorithm [as material gets redistributed the weight distribution changes as well which changes the optimal solution and so on…]. However usually it can generate interesting fine structures.

O [Integer]: Number of optimization iteration steps each time the user presses the “Step” button.

S [Number]: Smoothing factor applied at each iteration to avoid the generation of checker patterns. A number between 0.0 [no smoothing] and 1.0.

P [Number]: Penalization factor. The larger this number the fastest [fewer steps] the optimization will converge to a black and white state. However making it too high it can fail to redistribute material properly. A value from 1.5 to 3.0 is recommended.

T [Number]: Target Density percentage. A number from 0.0 to 1.0 designating the fraction of the overall material that should remain at the end of the optimization process. This is just a hint and not guaranteed to be achieved [you may want to tweak the magnitude of your applied loads relative to the scale of the system etc….]. It is also a percentage of the bounding box of the system rather than the material enclosed within the bounding regions [that may be curved or irregular].

D [list of Numbers]: A List of numbers, one for each quad element in the model that designates some initial material density distribution. These numbers should be in 1 to 1 correspondence with the points generated at the CP output of the  component.

MD [Number]: Minimum allowed density. The system should never have elements with density exactly 0.0 as that would make the analysis step fail. Instead the minimum density is capped at this value. 0.01 or 0.001 are generally good values.

DTH [Number]: Used in combination with the “Delete” button and defines the density value below which all quads are eliminated in order to speed up further steps.

PerX & PerY [Boolean]: These values determine whether the domain is assumed to be periodic in any direction. Useful when generating structural patterns [material design, or facades with periodic supports and loading conditions]. Setting any of these values to true will assume that the system exactly repeats on either side at the specified direction.

Load Case [Integerc]:This number determines the active loadcase to be analyzed or optimized.

All Cases [Boolean]: If true , during optimization all connected load cases will be taken into account.

Compliant Mechanism[Boolean]: If true the component will attempt to solve the compliant mechanism problem. For this to work you need to define the input and output actions as two separate loadcases connected to the  component.

Outputs:

FE [Topostruct 2D model]: The analysed Finite Element system to be passed to other components that deal with visualization and recovery of specific results.

maxu [Number]: The maximum deflection in meters of the analysed structure. Useful for quick troubleshooting and getting a quick rough measure of system performance without doing detailed analysis of the results. If this number is too big there may be disconnected elements in the system and if it is zero you may haven’t applied loads correctly.

Buttons:

Step: Each time you press this button O number of optimization iterations are executed

Analyze: Analyze the current model with its current density distribution without attempting any optimization

Reset: Reset the densities to either a homogeneous value or to the values provided at the D input discarding any optimization redistribution.

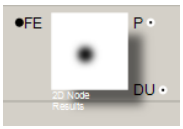
Smooth: Smooth the material distribution

Delete: Delete all elements that fall below the threshold defined by the DTH input. When you optimize you often see large parts of the structure discarded from the first few steps and which are unlikely to get any material assigned to them later. You can remove these element sin order to speed up calculations or increase the resolution of the remaining elements.

Subdivide: subdivide all elements in the system doubling the resolution [allowing you to see finer structures] and quadrupling the number of elements [and the time and memory required]

p. Results

1. Node results



This component returns the locations and displacements for all nodes.

Inputs:

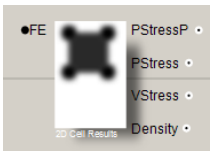
FE [Topostruct 2D model]: The analysed Finite Element system coming from the solver component

Outputs:

P [Point3d]: The locations of nodes

DU [Vector3d]: The displacements of all the nodes [m]

2. Plate element results



This component generates the results for all plate elements in the system.

Inputs:

FE [Topostruct 2D model]: The analysed Finite Element system coming from the solver component

Outputs:

PStressP [Plane]: A plane with its origin at the center of each quad and its axes aligned with the principal stress directions

PStress [Vector]: The X and Y components of these vectors hold the 1st and 2nd principal stresses [along the X and Y axes of the corresponding PStressP planes]. Positive values designate tension and negative compression

VStress [Number]: The von Mises stress for each quad. This is a measure of the overall stress at each point combining the values of both principal stresses.

Density [Number]: A number from 0.0 to 1.0 designating the redistributed material density at each quad. 1.0 means solid and 0.0 void.

q. Visualization

Visualization components help clarify the structural behaviour of the system

1. Contours

This component draws contours along lines of constant material density.

Inputs:

FE [Topostruct 2D model]: The analysed Finite Element system coming from the solver component

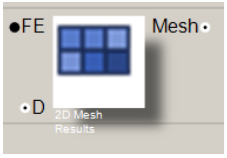
D [Number]: Deflection factor. This is used in order to exaggerate the deformation of the material at its equilibrium state. Useful making sure that all loads and supports are applied correctly and there aren't any disconnected components in the system.

Iso [Number]: contour value. This value is used in order to determine the density levels at which contours are drawn. For example a value of 0.5 means that a contour will be drawn that separates material densities less than 0.5 to material densities greater than 0.5.

Outputs:

Contour: A set of line segments forming a contour

2. Mesh visualization



This component visualizes the material distribution and stresses as a mesh

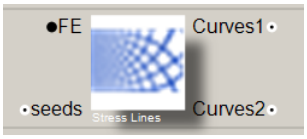
Inputs:

FE [Topostruct 2D model]: the analysed finite elements system coming from the solver component

D [Number]: Deflection factor. This is used in order to exaggerate the deformation of the material at its equilibrium state. Useful making sure that all loads and supports are applied correctly and there aren't any disconnected components in the system.

S [Boolean]: if this value is true then the mesh will be coloured according to the magnitude of stress, if it is false then the mess will display the material density as a grayscale gradient.

3. Stress lines



This component visualizes the stress lines in the analysed plate

Inputs:

FE [Topostruct 2D model]: the analysed finite elements system coming from the solver component

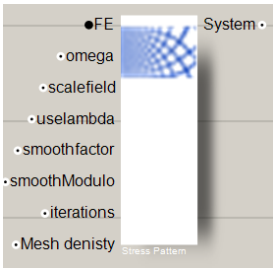
Seeds [Point]: Seed points from which stress lines are traced

Outputs:

S1 [Polyline]: Principal stress line passing through point P along the 1st principal stress

S2 [Polyline]: Principal stress line passing through point P along the 2nd principal stress

4. Stress pattern



This component generates a pattern that follows the principal stress directions of the analysed system. Its output must be connected to one of the reparameterization group components.

Inputs:

FE [StatFESystem]: The analysed topostruct2D system coming from the solver component ∫=

omega [Number]: A number that determines the scaling of the resulting pattern.

scalefield [Boolean]: if true the resulting pattern has variable scale allowing to better accommodate singularities without having to add bifurcation in order to account for stretching. In addition scaled patterns naturally reflect concentration of material near supports or highly stressed regions.

[ignore] uselambda[Boolean]: This option has only a marginal effect for highly anisotropic meshes

[ignore] smoothfactor[Number]:

[ignore] smoothModulo[Boolean]:

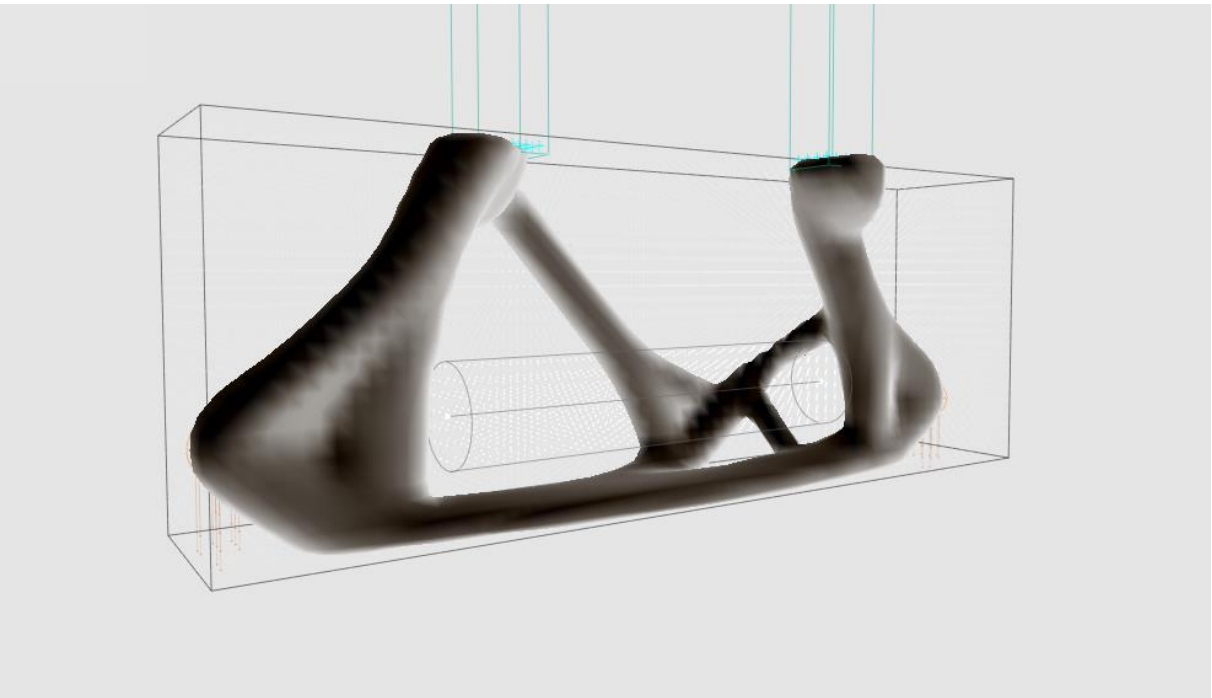
iterations[Integer]: The number of relaxation steps used for the parameterization. This can be 0 [faster calculation] but for certain difficult meshes might worth setting it to something like 500 or higher.

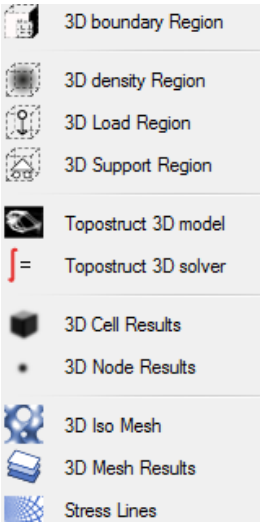
Mesh density[Integer]: An integer that determines the subdivisions of the resulting curve pattern. It only affects the Curve output component.

Outputs:

System [Generic]: The reparameterization system. You need to connect to one of the reparameterization components to see the results.

4. 3D volumetric analysis and topology optimization





This is a toolbox for analysing and optimizing volumetric models of variable material density. The analysis is based on a “voxelization” of the problem domain producing a three dimensional grid of cubic volume elements. The optimization is done using topology optimization methods.

r. Geometry constructors

1. 3D boundary Region

This component uses closed solids in order to define the boundaries [or holes] of the system’s domain.

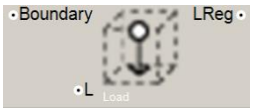
Inputs:

Boundary [Solid]: Closed solids to be converted to boundary elements determining the domain of the problem.

IsVoid [Boolean]: If true the boundary designates a hole otherwise it is part of the domain

s. Region Property modifiers 

1. Load Region



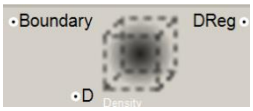
Designate a region where volume loads are applied

Inputs:

Boundary [Solid]: Closed solid defining the region within which the loads are applied

L [Vector3d]: Load in [N/m3]

2. Fixed Density Region



Designate a region where material density is constant [will remain unaffected during optimization]

Inputs:

Boundary [Solid]: Closed solid defining the region within which the density is fixed

D [Number]: Density as a number from 0.0 [no material] to 1.0 [full material]

3. Support Region

Designate a region where certain boundary conditions apply

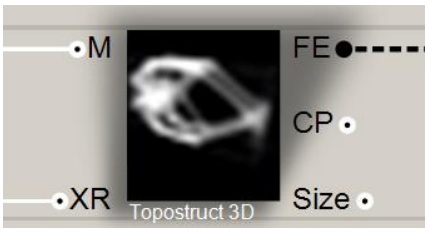
Inputs:

Boundary [Solid]: Closed solid defining the region within which to apply boundary conditions

SUP [Integer support code]: A support type  to be applied to all nodes within the boundary

AM [Boolean]: A value indicating whether support types should have an additive behaviour or just replace whatever support is already there.

t. System builder 



This is the main model builder tool. It aggregates all the information from the previous components, voxelizes the domain and generates the finite element model.

Inputs:

M []: Any number of regions with properties chosen form the topostruct 3D toolbar can be connected at this input.

XR [Integer]: The resolution along the x axis. This is the number of cubic volume elements to generate along the x axis of the bounding box of the input geometries. This number can dramatically increase the time and memory requirements of the solver. Values between 12 and 40 are generally good depending on the trade-off between speed and detail of results.

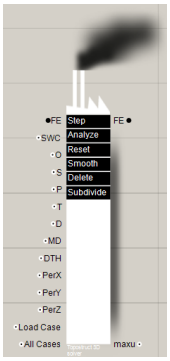
Outputs:

FE [Topostruct 3D model]: The analysis model

CP [list of Points]: A list of the centers of all the generated volume elements.

Size [Number]: The side length of each generated element

u. Solver ∫=



The solver is the core analysis component in this toolbox. It needs to be connected to a  system builder component from which it acquires the finite element system.

Inputs:

FE [Topostruct 3D model]: A model provided from a  system builder component

SWC [Number]: Self Weight Coefficient. A number that multiplies all the loads due to the self weight of the structure. For most optimization problems it should be set to 0 as it can induce a non-convergent feedback behavior in the optimization algorithm [as material gets redistributed the weight distribution changes as well which changes the optimal solution and so on…]. However usually it can generate interesting fine structures.

O [Integer]: Number of optimization iteration steps each time the user presses the “Step” button.

S [Number]: Smoothing factor applied at each iteration to avoid the generation of checker patterns. A number between 0.0 [no smoothing] and 1.0.

P [Number]: Penalization factor. The larger this number the fastest [fewer steps] the optimization will converge to a black and white state. However making it too high it can fail to redistribute material properly. A value from 1.5 to 3.0 is recommended.

T [Number]: Target Density percentage. A number from 0.0 to 1.0 designating the fraction of the overall material that should remain at the end of the optimization process. This is just a hint and not guaranteed to be achieved [you may want to tweak the magnitude of your applied loads relative to the scale of the system etc….]. It is also a percentage of the bounding box of the system rather than the material enclosed within the bounding regions [that may be curved or irregular].

D [list of Numbers]: A List of numbers, one for each volume element in the model that designates some initial material density distribution. These numbers should be in 1 to 1 correspondence with the points generated at the CP output of the  component.

MD [Number]: Minimum allowed density. The system should never have elements with density exactly 0.0 as that would make the analysis step fail. Instead the minimum density is capped at this value. 0.01 or 0.001 are generally good values.

DTH [Number]: Used in combination with the “Delete” button and defines the density value below which all quads are eliminated in order to speed up further steps.

PerX & PerY & PerZ [Boolean]: These values determine whether the domain is assumed to be periodic in any direction. Useful when generating structural.

Load Case [Integerc]:This number determines the active loadcase to be analyzed or optimized.

All Cases [Boolean]: If true , during optimization all connected load cases will be taken into account.

Outputs:

FE [Topostruct 3D model]: The analysed Finite Element system to be passed to other components that deal with visualization and recovery of specific results.

DEF [Number]: The maximum deflection in meters of the analysed structure. Useful for quick troubleshooting and getting a quick rough measure of system performance without doing detailed analysis of the results. If this number is too big there may be disconnected elements in the system and if it is zero you may haven’t applied loads correctly.

Buttons:

Step: Each time you press this button O number of optimization iterations are executed

Analyze: Analyze the current model with its current density distribution without attempting any optimization

Reset: Reset the densities to either a homogeneous value or to the values provided at the D input discarding any optimization redistribution.

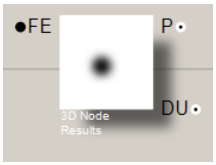
Smooth: Smooth the material distribution

Delete: Delete all elements that fall below the threshold defined by the DTH input. When you optimize you often see large parts of the structure discarded from the first few steps and which are unlikely to get any material assigned to them later. You can remove these elements in order to speed up calculations or increase the resolution of the remaining elements.

Subdivide: subdivide all elements in the system doubling the resolution [allowing you to see finer structures] and increases the number of elements by a factor of 8 [and the time and memory required]

v. Results

1. Node results



This component returns the locations and displacements for all nodes.

Inputs:

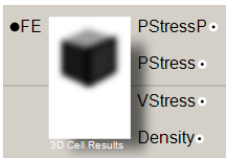
FE [Topostruct 3D model]: The analysed Finite Element system coming from the solver component

Outputs:

P [Point3d]: The locations of nodes

DU [Vector3d]: The displacements of all the nodes [m]

2. Volume element results



This component generates the results for all volume elements in the system.

Inputs:

FE [Topostruct 3D model]: The analysed Finite Element system coming from the solver component

Outputs:

PStressP [Plane]: A plane with its origin at the center of each voxel and its axes aligned with the three principal stress directions

PStress [Vector]: The X, Y and Z components of these vectors hold the 1st , 2nd and 3rd principal stresses [along the X, Y and Z axes of the corresponding PStressP planes]. Positive values designate tension and negative compression

VStress [Number]: The von Mises stress for each quad. This is a measure of the overall stress at each point combining the values of both principal stresses.

Density [Number]: A number from 0.0 to 1.0 designating the redistributed material density at each quad. 1.0 means solid and 0.0 void.

w. Visualization

Visualization components help clarify the structural behaviour of the system

1. Contours

This component draws contours along surfaces of constant material density.

Inputs:

FE [Topostruct 3D model]: The analysed Finite Element system coming from the solver component

D [Number]: Deflection factor. This is used in order to exaggerate the deformation of the material at its equilibrium state. Useful making sure that all loads and supports are applied correctly and there aren't any disconnected components in the system.

Iso [Number]: contour value. This value is used in order to determine the density levels at which contours are drawn. For example a value of 0.5 means that a contour will be drawn that separates material densities less than 0.5 to material densities greater than 0.5.

S [Boolean]: if this value is true then the mesh will be coloured according to the magnitude of stress.

Outputs:

Contour: A Mesh forming an iso surface through the material distribution

2. Mesh visualization

This component visualizes the material distribution and stresses as a series of horizontal meshes cutting through the volume of the domain.

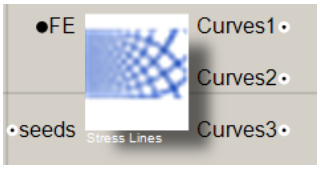
Inputs:

FE [Topostruct 3D model]: the analysed finite elements system coming from the solver component

D [Number]: Deflection factor. This is used in order to exaggerate the deformation of the material at its equilibrium state. Useful making sure that all loads and supports are applied correctly and there aren't any disconnected components in the system.

S [Boolean]: if this value is true then the mesh will be coloured according to the magnitude of stress, if it is false then the mess will display the material density as a grayscale gradient.

3. Stress lines



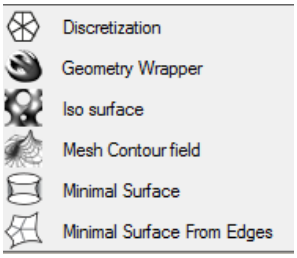
This component visualizes the stress lines in the analysed plate

Inputs:

FE [Topostruct 3D model]: the analysed finite elements system coming from the solver component

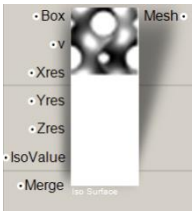
Seeds [Point]: Seed points from which stress lines are traced

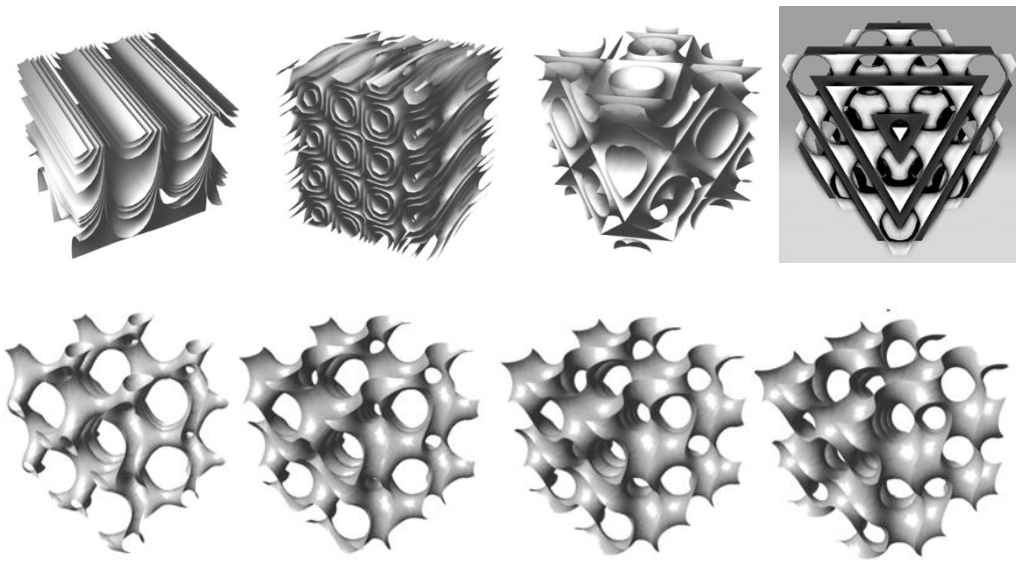
5. Geometry Utilities



These components are not directly related to the structural analysis module. However they contain functionality that is useful for the preparation of analysis models or the visualization of results.

x. Marching Cubes Iso surfaces 





This utility uses the marching cubes algorithms in order to extract iso-surfaces from a field of density values. These values must be defined on a three dimensional grid and can be generated through simple function [F(x,y,z)] or using the Geometry wrapper component below.

Inputs:

Box [Bounding Box]: The bounding box of the geometry

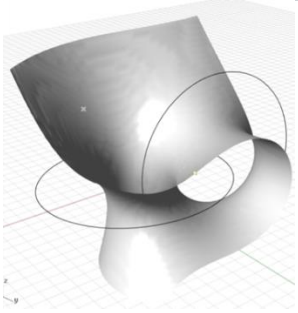
V [List of Numbers]: This list must contain Xres x Yres x Zres numbers designating the field values at each point of the notional grid inside Box. The values should be ordered in such a way so that x direction is traversed first then y and then the z layers.

Xres, Yres, Zres [Integer]: The resolution of the three dimensional grid.

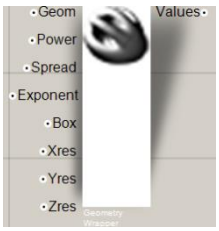
IsoValue [Number]: The threshold at which to cut an iso surface through the density field.

Merge [Boolean]: If true the resulting mesh will have its coinciding vertices fused and will look smoother [it does not alter the locations of vertices it just fuses coincident vertices so that normal and hence lighting calculations are continuous and not faceted]

y. Geometry Wrapper 



This component creates grids of values that can be used in conjunction with the iso surface component above  in order to generate smooth meshes that wrap around groups of any type of geometry [points, curves, solids]



Inputs:

Geom [Geometry]: Any number of points, curves and surfaces to compute a decaying density field around

Power [Number]: The strength of the field of each object. This is either a list of numbers [one for each geometric object] or a single value applied to all objects

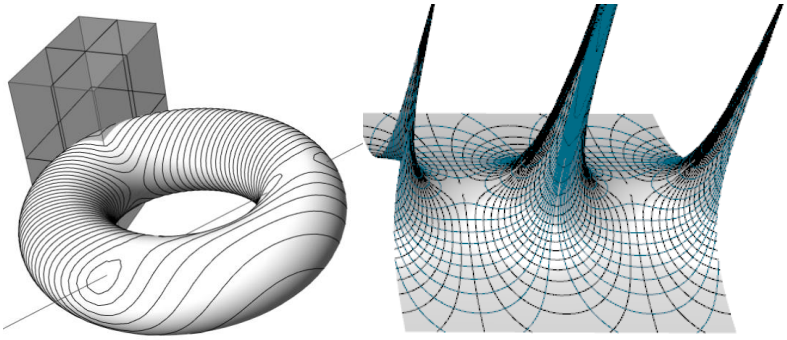
Spread [Number]: The spread of the field of each object. This is either a list of numbers [one for each geometric object] or a single value applied to all objects

Exponent [Number]: Not used

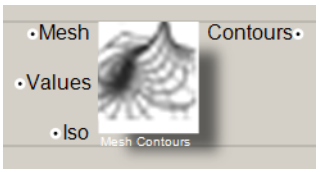
Box [Bounding box]: The bounding box within which the grid of values will be generated

Xres, Yres, Zres [Integer]: The resolution of the three dimensional grid.

z. Mesh contours



Given a mesh and a set of values defined at each vertex of the mesh, this component can extract contours from the density field of values.



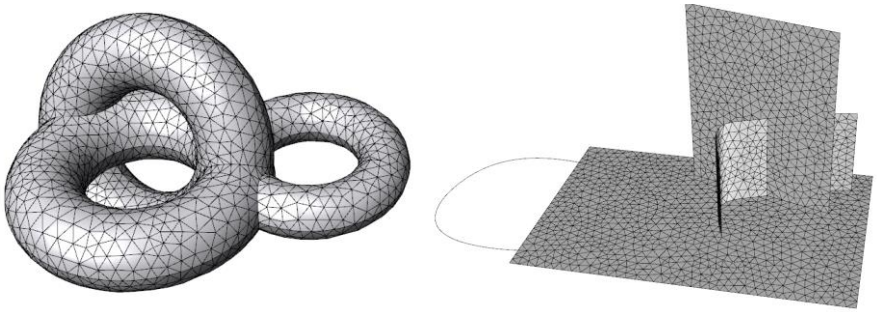
Inputs:

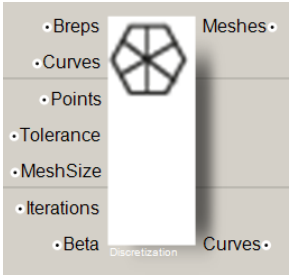
Mesh [Mesh]: A mesh over which the contours are distributed

Values [Number]: A list of numbers of equal length to the number of vertices in the mesh. This numbers determine the scalar field over the input mesh that determines the shape of contours.

Iso [Number]: the threshold at which a contour is generated

aa. Discretization [BETA]





This component can be used in order to triangulate complexes made of any number of solid, surfaces and curves, ensuring that the resulting meshes and polylines have common nodes along all intersections. It can handle a larger range of situations that the default rhino meshing cannot deal with. It will create slits on surfaces where a curve touches them or allow meshing of non-manifold connections [along edges where more than 2 surfaces intersect]. It will intersect and split all objects and create a complex of arbitrary topology. This component is still in development , may be slow and even crash so save before using and use with caution. Such meshing that ensures common vertices along contacts is important for most FE analysis applications.

Inputs:

Breps [Brep]: A list of polysurfaces

Curves [Curve]: A list of Curves.

Points [Point3d]: A list of points.

Tolerance[Number]: the intersection tolerance to be used during geometry splitting.

MeshSize[Number]: The target average edge length for the resulting meshes' triangles. Be careful with this value as small values can increase calculation time significantly

Iterations[Integer]: the number of relaxation iterations that is applied to meshes after discretization. This helps to further smooth out the differences in sizes between triangles

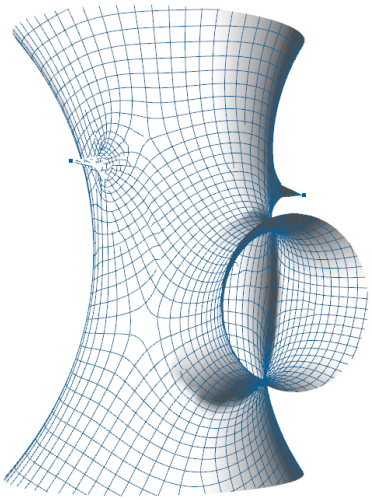
Beta[Number]: A factor that determines the scaling of the mesh. It works in connection to the Meshsize option. Usually values between 0.8 and 1.5 are good.

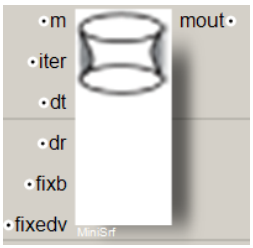
Outputs:

Meshes[Mesh]: A list of meshes corresponding to the initial geometry

Curves[Polyline]: A list of polylines that correspond to the discretized input curves

bb. Minimal surface 





This component will attempt to minimize the surface area of a mesh while preserving the locations of boundaries and select nodes. It uses a combination of a mean curvature flow with regular mesh relaxation.

Inputs:

m [Mesh]: The input mesh to be processed.

iter [Integer]: The number of iterations [for large meshes you might need to set it to several thousands.

dt [Number]: This number determines the strength of the mean curvature flow step. The larger it is the faster the surface will converge [with fewer iteration steps]. However when it gets too large the solution becomes unstable and the mesh crumbles. You might need to experiment with this number for different meshes and especially for different scales.

dr[Number]: This number determines the strength of the mesh relaxation step. This too has the same trade-offs as the dt parameter in terms of converges vs stability.

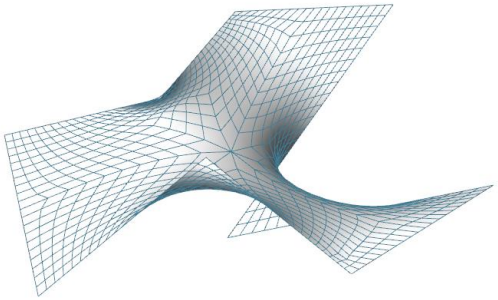
fixb[Boolean]: If true all boundary nodes are fixed during iterations.

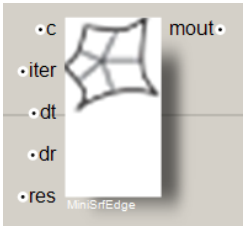
fixedv[Point3d]: A list of points to be held fixed during iterations. These points must coincide with vertices in the input mesh.

Outputs:

mout[Mesh]: The altered mesh

cc. Minimal Surface from Edges 





This component will attempt to create a minimal surface spanning an arbitrary boundary consisting of an arbitrary number of curves forming a closed loop. Therefore it can go around rhino's limitation of the 4-edge surface.

Inputs:

c [Curve]: A list of curves forming a closed loop [if your list contains polycurves or polylines it is better to explode them to separate lines and curves]

iter [Integer]: The number of iterations [for large meshes you might need to set it to several thousands.

dt [Number]: This number determines the strength of the mean curvature flow step. The larger it is the faster the surface will converge [with fewer iteration steps]. However when it gets too large the solution becomes unstable and the mesh crumbles. You might need to experiment with this number for different meshes and especially for different scales.

dr[Number]: This number determines the strength of the mesh relaxation step. This too has the same trade-offs as the dt parameter in terms of converges vs stability.

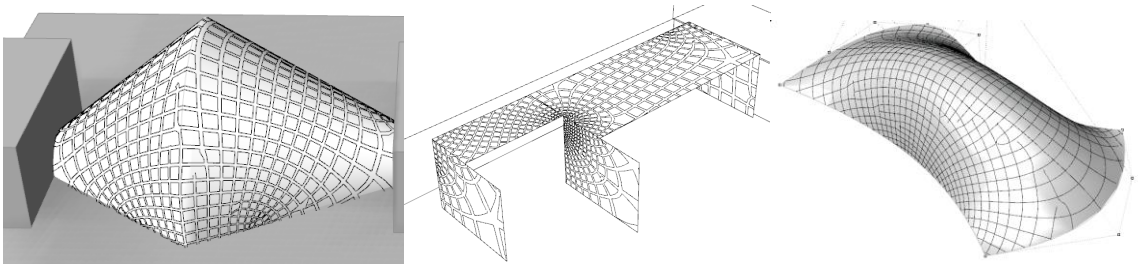
res[Integer]: The resolution of the mesh along each of the input curves.

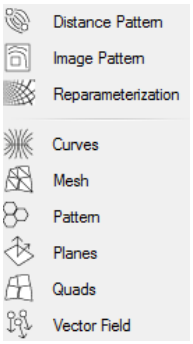
Outputs:

mout[Mesh]: The minimal surface spanning the input loop

6. Surface Reparameterization

This family of components implements the global periodic parameterization algorithm in order to create nearly conformal maps over meshes following some input vector field that determines directionality. It is used to create stress aligned or curvature aligned patterns over surfaces but it can also work with arbitrary vector fields [see examples].

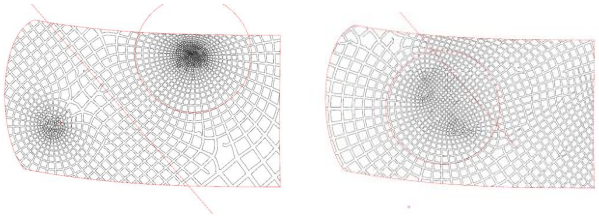


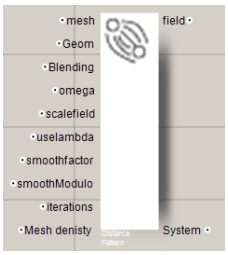


dd. Pattern Definitions

These components along with [stress pattern components in the FESystem and topostruct2D groups] define a reparameterization over an input mesh.

1. Distance Pattern





this component generates a parameterization controlled by a set of curves and points to which the pattern tries to align itself.

Inputs:

mesh [Mesh]: The mesh over which the new parameterization will be defined

Geom[Point3d, Curve]: A set of geometric objects around which the patterns will align

Blending[Number]: A number that determines the degree of blending between the different geometric attractors

omega [Number]: A number that determines the scaling of the resulting pattern.

scalefield [Boolean]: if true the resulting pattern has variable scale allowing to better accommodate singularities without having to add bifurcation in order to account for stretching. In addition scaled patterns naturally reflect concentration of material near supports or highly stressed regions.

[ignore] uselambda[Boolean]: This option has only a marginal effect for highly anisotropic meshes

[ignore] smoothfactor[Number]:

[ignore] smoothModulo[Boolean]:

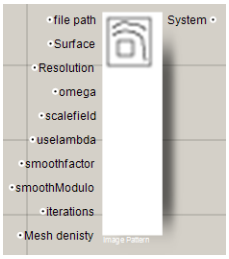
iterations[Integer]: The number of relaxation steps used for the parameterization. This can be 0 [faster calculation] but for certain difficult meshes might worth setting it to something like 500 or higher.

Mesh density[Integer]: An integer that determines the subdivisions of the resulting curve pattern. It only affects the Curve output component.

Outputs:

System [Generic]: The reparameterization system. You need to connect to one of the reparameterization components to see the results.

2. Image Pattern



With this component you can create parameterizations that conform to the gradient field defined by some input image which is mapped onto the geometry.

Inputs:

file path [string]: The file name of the image to be loaded

Surface[Surface]: A surface over which the imported image will be mapped

Resolution[Integer]: the resolution of the mesh to be constructed from the input image

omega [Number]: A number that determines the scaling of the resulting pattern.

scalefield [Boolean]: if true the resulting pattern has variable scale allowing to better accommodate singularities without having to add bifurcation in order to account for stretching. In addition scaled patterns naturally reflect concentration of material near supports or highly stressed regions.

[ignore] uselambda[Boolean]: This option has only a marginal effect for highly anisotropic meshes

[ignore] smoothfactor[Number]:

[ignore] smoothModulo[Boolean]:

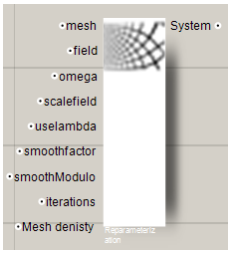
iterations[Integer]: The number of relaxation steps used for the parameterization. This can be 0 [faster calculation] but for certain difficult meshes might worth setting it to something like 500 or higher.

Mesh density[Integer]: An integer that determines the subdivisions of the resulting curve pattern. It only affects the Curve output component.

Outputs:

System [Generic]: The reparameterization system. You need to connect to one of the reparameterization components to see the results.

3. Reparameterization



This component allows you to compute the reparameterization of the input mesh using either its principal curvature directions or an arbitrary field

Inputs:

mesh [Mesh]: The mesh over which the new parameterization will be defined

field[Vector3d]: If this parameter is left empty then the component wil lcompute the principal curvature directions of the input mesh and use these to define the parameterization field. If you want to define your own alignment field then this input must be set to a list of vectors with the same number and order of elements as there are vertices in the input mesh.

omega [Number]: A number that determines the scaling of the resulting pattern.

scalefield [Boolean]: if true the resulting pattern has variable scale allowing to better accommodate singularities without having to add bifurcation in order to account for stretching. In addition scaled patterns naturally reflect concentration of material near supports or highly stressed regions.

[ignore] uselambda[Boolean]: This option has only a marginal effect for highly anisotropic meshes

[ignore] smoothfactor[Number]:

[ignore] smoothModulo[Boolean]:

iterations[Integer]: The number of relaxation steps used for the parameterization. This can be 0 [faster calculation] but for certain difficult meshes might worth setting it to something like 500 or higher.

Mesh density[Integer]: An integer that determines the subdivisions of the resulting curve pattern. It only affects the Curve output component.

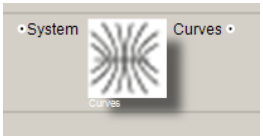
Outputs:

System [Generic]: The reparameterization system. You need to connect to one of the reparameterization components to see the results.

ee. Reparameterization Outputs

These components expose various ways to visualize and work with the new parameterization of a given mesh.

1. Curves



This component connects to any of the reparameterization component and extracts a network of curves aligned to the given parameterization. In effect these are the new UV curves of the input mesh.

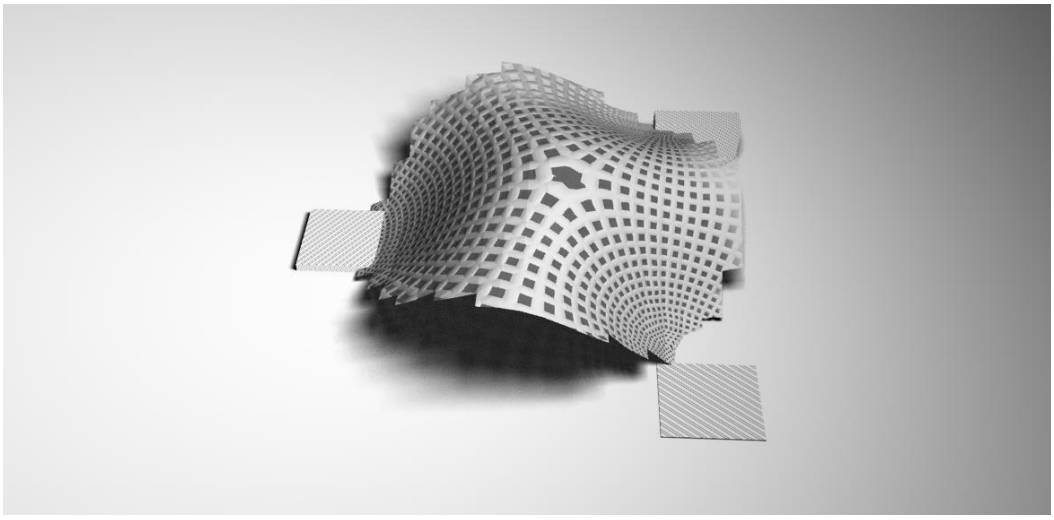
Inputs:

System [Generic]: any of the outputs of the following components

Outputs:

Curves [Curve]: A list of curves that form an orthogonal grid over the given mesh

2. Mesh



This component extracts the parameterized mesh. This mesh is identical to the mesh that was given to the parameterization algorithm; however the component attaches texture coordinates that reflect the new UV parameterization. You can use this mesh with texture mapping during rendering to map any image over the surface.

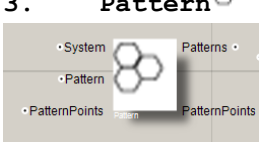
Inputs:

System [Generic]: any of the outputs of the following components

Outputs:

Mesh [Mesh]: A mesh with texture coordinates reflecting the parameterization

3. Pattern



With this component you can map arbitrary patterns [other than orthogonal grids] along the new parameterization. The Curves and points of the pattern must be 2d objects defined within the unit square [0,0] to [1,1] and they should possess symmetry with respect to 90 degrees rotations otherwise discontinuities will appear in the resulting patterning.

Inputs:

System [Generic]: any of the outputs of the following components 

Pattern[Line]: A list of lines defined within the unit square that will be tiled over the input mesh.

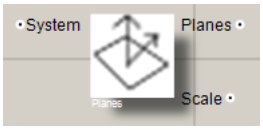
PatternPoints[Point3d]: a list of points to be mapped

Outputs:

Patterns [Line]: A list of the remapped lines

PatternPoints[Point3d]: A list of the remapped points

4. Planes 



This component extracts planes that are aligned to the new parameterization

Inputs:

System [Generic]: any of the outputs of the following components 

Outputs:

Planes[Plane]: A list of planes

Scale[Number]: A list of numbers that reflect the variable scaling of the parameterization. Each number corresponds to one Plane.

5. Quads 

Extract a mesh made of the quads that make up the regular parts of the new parameterization. This mesh will also have holes around singularities and along boundaries.

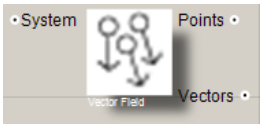
Inputs:

System [Generic]: any of the outputs of the following components 

Outputs:

Quadmesh[Mesh]: A mesh made of quads that align to the new parameterization

6. Vector Field



This component is useful for debugging and extracts the vector field that the algorithm used in order to calculate the new parameterization.

Inputs:

System [Generic]: any of the outputs of the following components

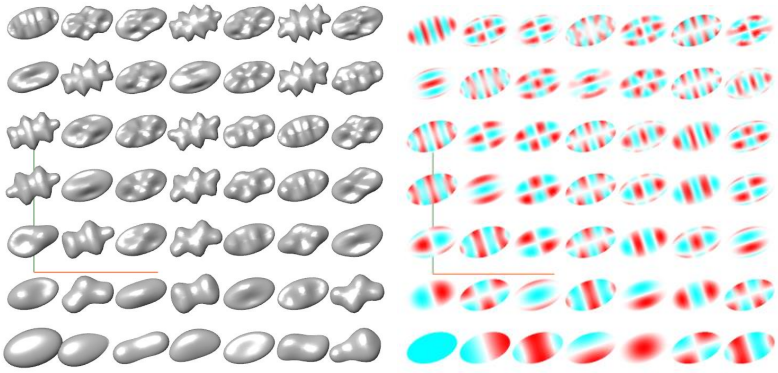
Outputs:

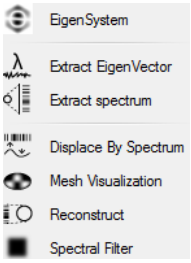
Points[Point3d]: A list of the points where the vectors are applied

Vectors[Vector3d]: A list of vectors that define the parameterization's vector field

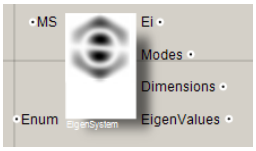
7. Eigenmode based analysis and synthesis of meshes

This group of components uses the Eigenmodes of the discrete Laplace Beltrami operator defined over some user defined meshes in order to analyse, modify and synthesize mesh based geometries.





ff. EigenSystem



This component takes as input a mesh and calculates the spectrum of its discrete Laplace Beltrami operator. The Laplace operator is a differential operator related to the Heat and the Wave equation. Its eigenfunctions correspond to the resonant vibration modes of membranes but can be generalized over any graph and geometric complex with a similar meaning. In addition they form a complete basis by which to analyse any function defined over a given geometric complex.

Inputs:

Mesh [Mesh]: The mesh to be analysed [this calculation can be heavy for meshes with a large number of faces]

Enum[Integer]: The number of Eigenmodes to compute. A mesh with N vertices will generate in general N eigenvectors with N components each. to reduce memory consumption we may want to limit the number of calculated eigenvectors to the first few. By setting this number to a value greater than 0 you can control how many modes are calculated. The default is to compute all the modes.

Outputs:

Ei[EigenSystem]: An Eigensystem object to be passed to the value extraction and visualization components

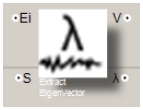
Modes[Integer]: The number of modes actually calculated

Dimensions[Integer]: the number of components per mode, this is equal to the number of vertices in the input mesh

EigenValues[Number]: A list of numbers with the eigenvalues of the operators sorted by magnitude. The Laplace operator is symmetric and its eigenvalues are all real.

gg. Numerical Value Extraction

1. Extract EigenVector



This component extracts an eigenvector from the eigensystem

Inputs:

Ei [Eigensystem]: The eigensystem input must be connected to the “Ei” output of the component

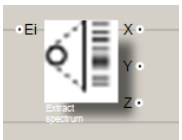
S[Integer]: The index of the eigenvector to extract

Outputs:

V[Number]: the eigenvector as a list of N numbers

λ[Number]: The corresponding eigenvalue to the extracted vector

2. Extract Spectrum



This component extracts the XYZ spectrums of the mesh. The X Y and Z coordinates of the vertices can be seen as three functions defined over the input surface. These functions can be “projected” over the eigenvectors yielding a spectrum for each coordinates. Filtering these spectrums we can modify the geometry of the original mesh.

Inputs:

Ei [Eigensystem]: The eigensystem input must be connected to the “Ei” output of the component

Outputs:

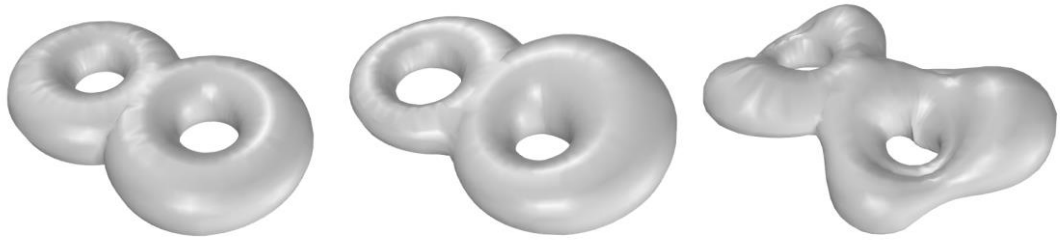
X[Number]: A list of numbers that corresponds to the spectrum of the x coordinates

Y[Number]: A list of numbers that corresponds to the spectrum of the y coordinates

Z[Number]: A list of numbers that corresponds to the spectrum of the z coordinates

hh. Mesh Manipulation

1. Displace by Spectrum



This component displaces a mesh along its eigenvectors creating wave like deformations.

Inputs:

Ei [Eigensystem]: The eigensystem input must be connected to the "Ei" output of the component

S[Number]: this is the list of numbers that determines the displacement spectrum. For example the list [0.1, 0.5, 0.2] will create a deformation where vertices move 0.1 along the first eigenmode, 0.5 against the second and so on… .

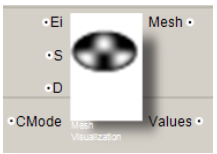
D[Number]: This is a multiplier that determines the strength of the overall displacement

Outputs:

Mesh[Mesh]: The deformed mesh

Values[Number]: The displacement values [one for each vertex]

2. Mesh Visualization



This component can be used in order to visualize the eigenmodes of the input geometry.

Inputs:

Ei [Eigensystem]: The eigensystem input must be connected to the "Ei" output of the component

S[Integer]: the index of the mode to visualize

D[Number]: The displacement of the resulting mesh along the eigenvector.

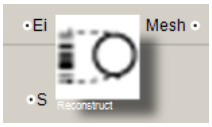
CMode[Integer]: The color mode to use [0->none, 1->RedBlue, 2->Grey, 3->GreyAbsoluteValue]

Outputs:

Mesh[Mesh]: The visualization mesh

Values[Number]: The components of the eigenmode.

3. Reconstruct



This component can reconstruct a mesh from a spectrum and an eigensystem. It is very similar to the displacement component wit the difference that the spectrum here expresses multipliers of the actual XYZ coordinates rather than displacement along the normals.

Inputs:

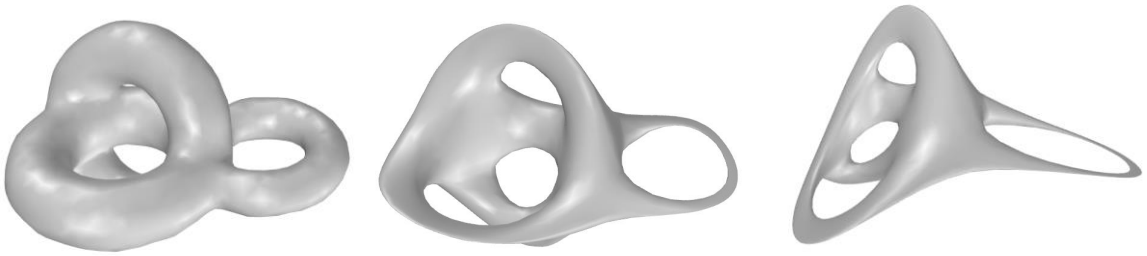
Ei [Eigensystem]: The eigensystem input must be connected to the “Ei” output of the component

S[Number]: A list of numbers corresponding to the XYZ spectrum multipliers

Outputs:

Mesh[Mesh]: The reconstructed mesh

4. Spectral filter



This component can be used as a low pass or high pass filter for geometry. By removing the high frequency eigenmodes the geometry is smoothed out.

Inputs:

Ei [Eigensystem]: The eigensystem input must be connected to the “Ei” output of the component

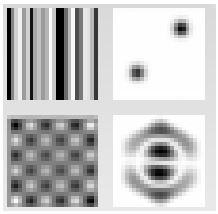
M0[Integer]: the lowest eigenmode to keep

M[Integer]: the highest eigenmode to keep

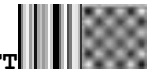
Outputs:

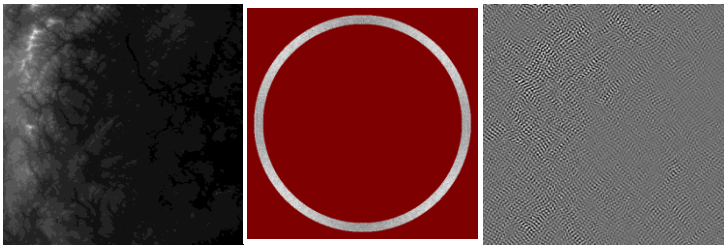
Mesh[Mesh]: The reconstructed mesh

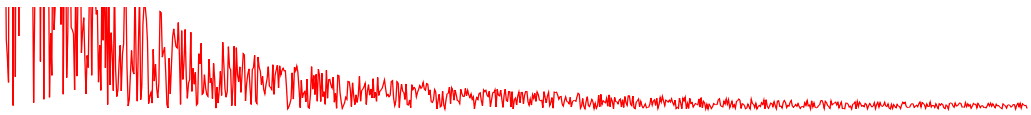
8. Numerical Utilities



These utilities are interfaces to the Intel Math Kernel Library exposing some standard numerical analysis functionality.

ii. FFT 





These two components can calculate the Fast Fourier transform of an input signal over 1 and two dimensions. For example in sound analysis the Fourier transform of the wave form yields the spectrum of frequencies. You can also apply the 2d transform on images do some filtering and invert the transform to isolate certain scales in the image.

1. 1D FFT

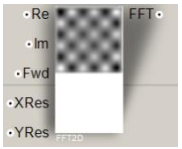
Inputs:

Re [Numbers] : A list of numbers forming the real part of the signal

Im [Numbers]: A list of numbers forming the imaginary part of the signal [if this parameter is not linked to anything then the signal is assumed to have only real values].

Fwd [Boolean]: If true compute the forward Fourier transform otherwise compute the inverse Fourier transform

2. 2D FFT



Inputs:

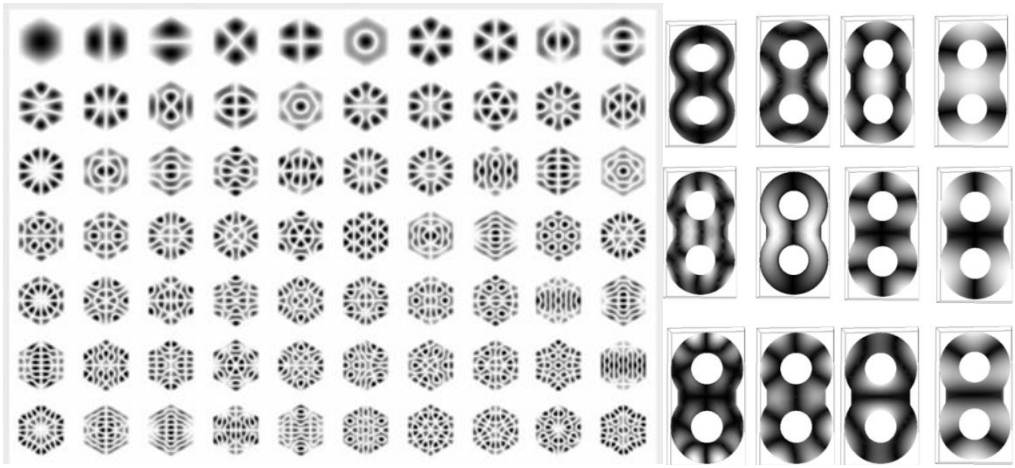
Re [Numbers] : A list of numbers forming the real part of the signal

Im [Numbers]: A list of numbers forming the imaginary part of the signal [if this parameter is not linked to anything then the signal is assumed to have only real values].

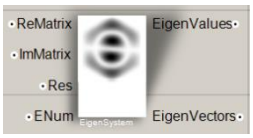
Fwd [Boolean]: If true compute the forward Fourier transform otherwise compute the inverse Fourier transform

XRes & YRes [Integer]: the dimensions of the two dimensional matrix that constitutes the input signal. The number of values in “Re” should be equal to XRes x YRes and the values should be in row major order.

jj. EigenSystems



This component computes the eigenvalues and eigenvectors of a real symmetric or Hermitian matrix



Inputs:

ReMatrix [Numbers]: A Res x Res list of numbers representing the real part of the matrix [it must represent a symmetric real matrix]

ImMatrix [Numbers]: If this component is defined then it should contain a Res x Res list of numbers representing the imaginary part of a matrix [assuming a Hermitian matrix]. If this parameter is left undefined then the system is assumed to be real.

Res [Integer]: The dimension of the matrix [number of rows]

ENum [Integer]: Number of eigenvalues and eigenvectors to compute. It can be any integer number between 0 and Res. 0 means compute all [Res] eigenvectors.

Outputs:

EigenValues [Numbers]: A list of numbers representing the eigenvalues of the system

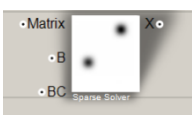
EigenVectors [Complex Numbers]: A tree with ENum branches and where each branch contains Res values corresponding to one eigenvector.

kk. Sparse Linear solver

This component is an interface to the PARDISO solver in the intel Math Kernel Library. This is a sparse linear system solver. A linear system of equations can be represented as:

A.X=B

Where A is a n x n symmetric matrix of coefficients, B is n long vector of values and X is an n long vector of unknowns. In many problems the matrix A can easily become very big. For example for a structural analysis problem with just 10 nodes assuming 6 degrees of freedom per node the matrix would be 60x60. However in many engineering applications the matrix A is sparse, which means that many of its entries are 0 and we don't have to waste a lot of memory [often more than your average desktop system can handle] storing all those entries. Here's where sparse solvers are useful as they operate on a representation of the matrix that only stores non zero values.



Inputs:

Matrix [Points]: A list of points representing entries to be added to the matrix. The X and Y coordinates represent the Row and column indices of the entry and the z coordinate is the actual value to be added to that entry.

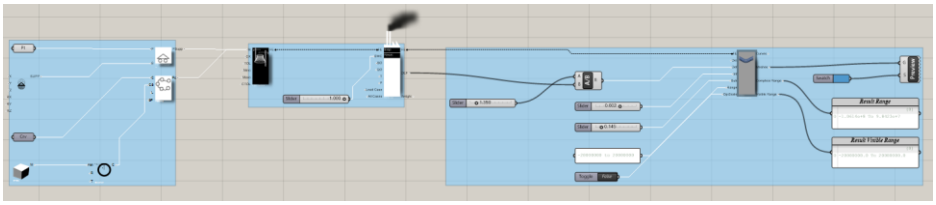
B [Numbers]: A List of numbers holding the right hand side of the system of equations. From the length of this list the algorithm determines the dimensions of the matrix.

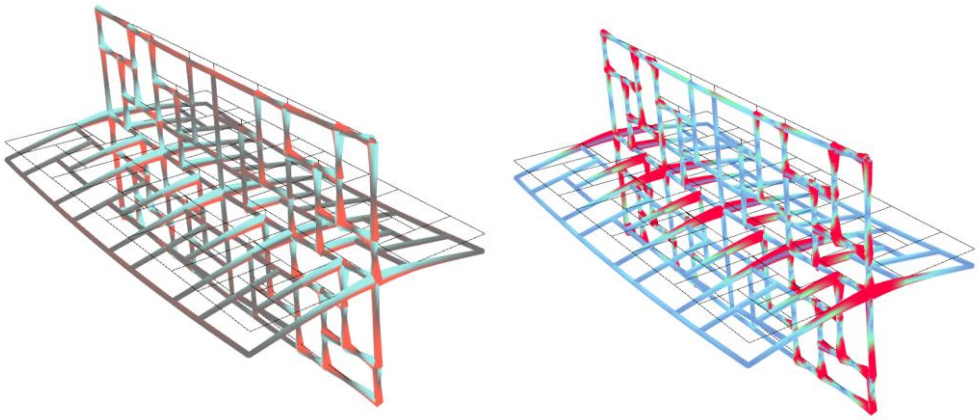
BC [Points]: A list of points representing boundary conditions [usually known X values that can be replaced and fixed]. The x coordinate of each point is the index of the value to fix and the y coordinate is the value to be fixed to.

γ. Samples

9. Basic Analysis

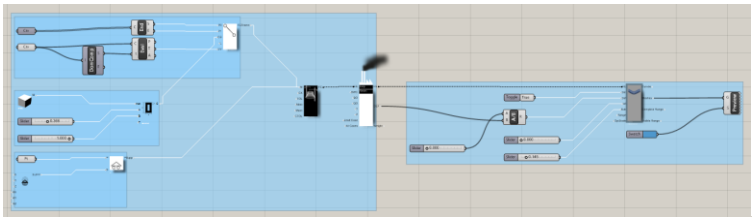
a. 000_simpleframes

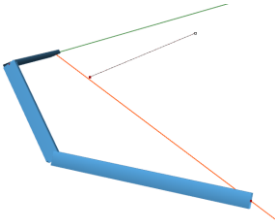




Using frame elements generated from a set of lines. You can set explicitly the numerical range of the results for the mesh visualization component.

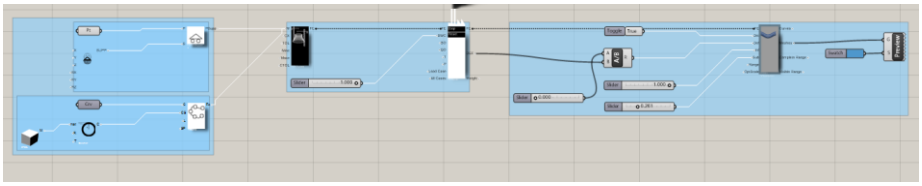
b. 000_simpleframes_upvector

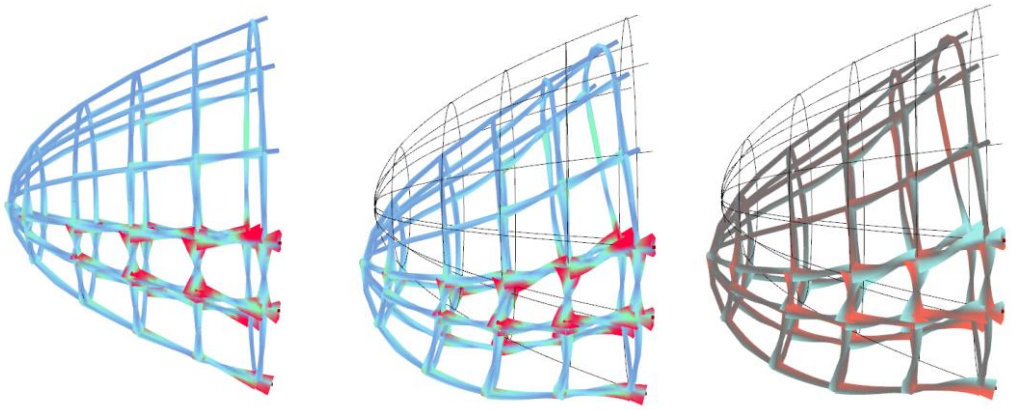




Use of vector hint that allows control over the orientation of applied cross-sections.

c. 001_Curvedframes

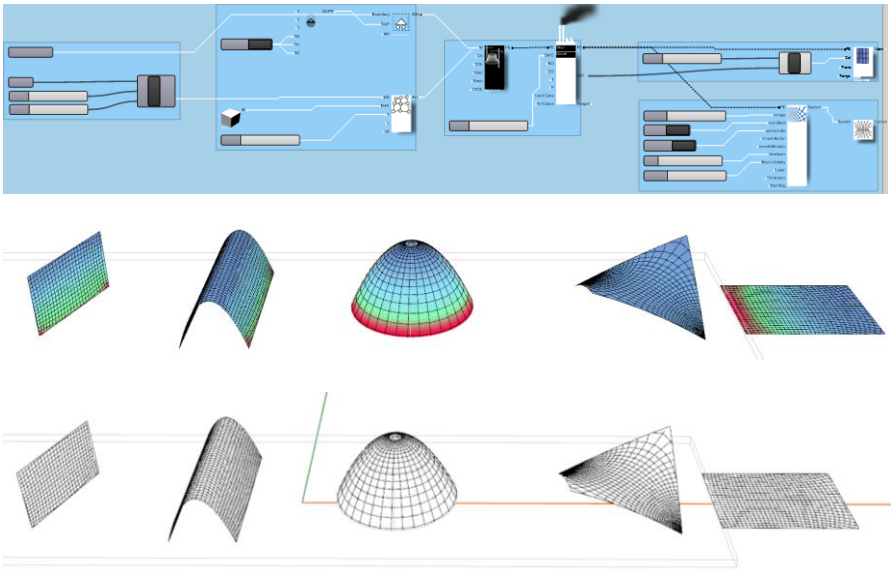




Using frame elements generated from a network of curves. Curves are automatically intersected and discretized by the FE system builder.

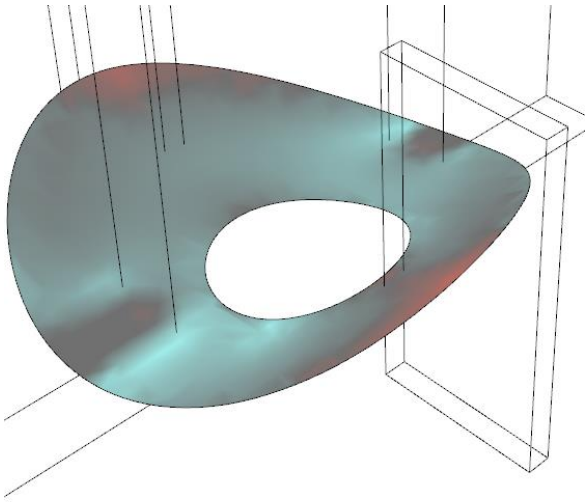
10. Shells

d. 00_SimpleShells



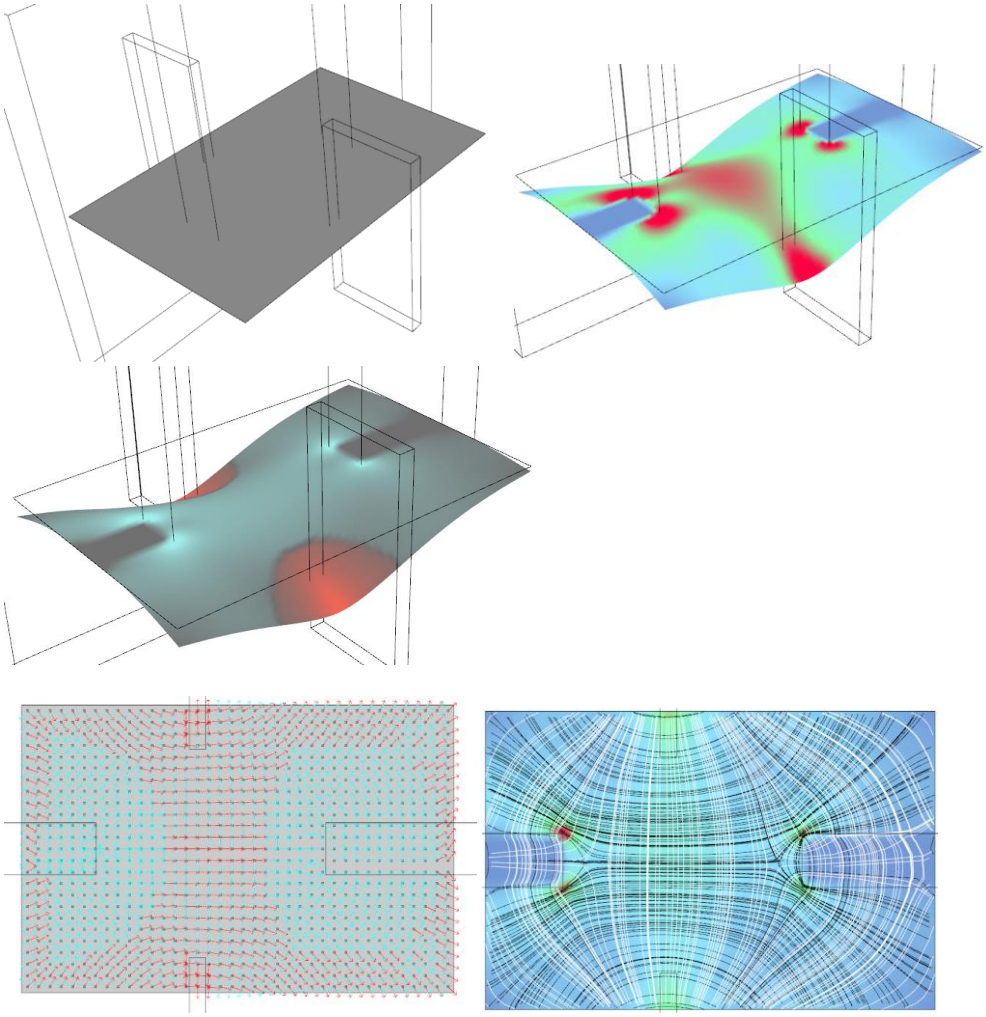
This GH file demonstrates basic shell analysis on a family of different surfaces based on fundamental shapes. Both stresses and principal stress direction patterns are displayed.

e. 002_ShellSurface.gh



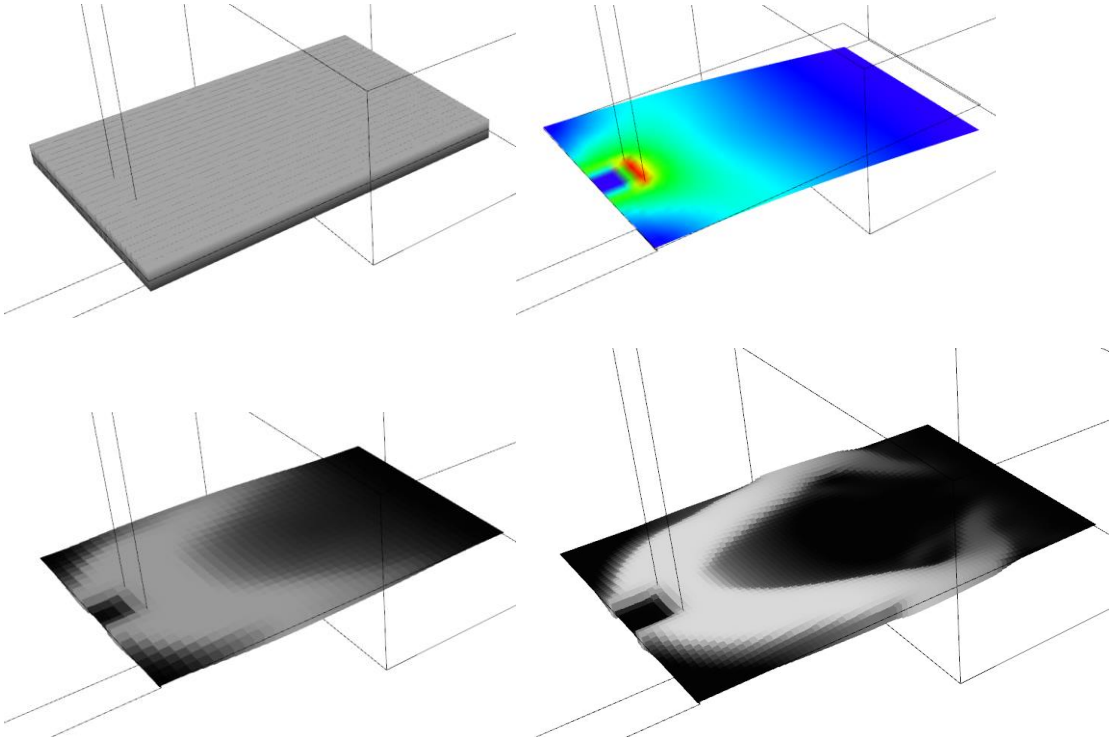
Using shell elements to analyse a surface.

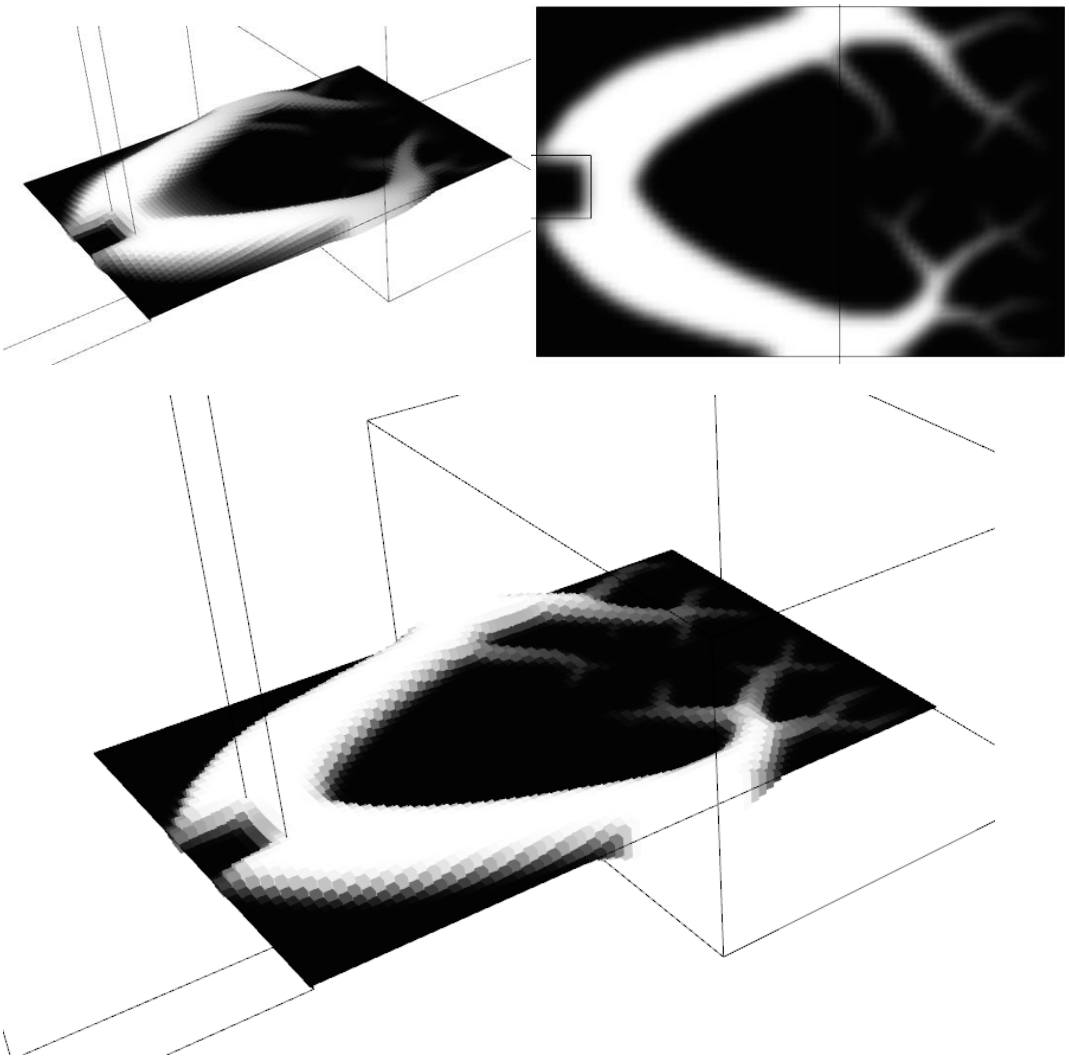
f. **003_slab.3dm**



Slab analysis with principal stress field visualization

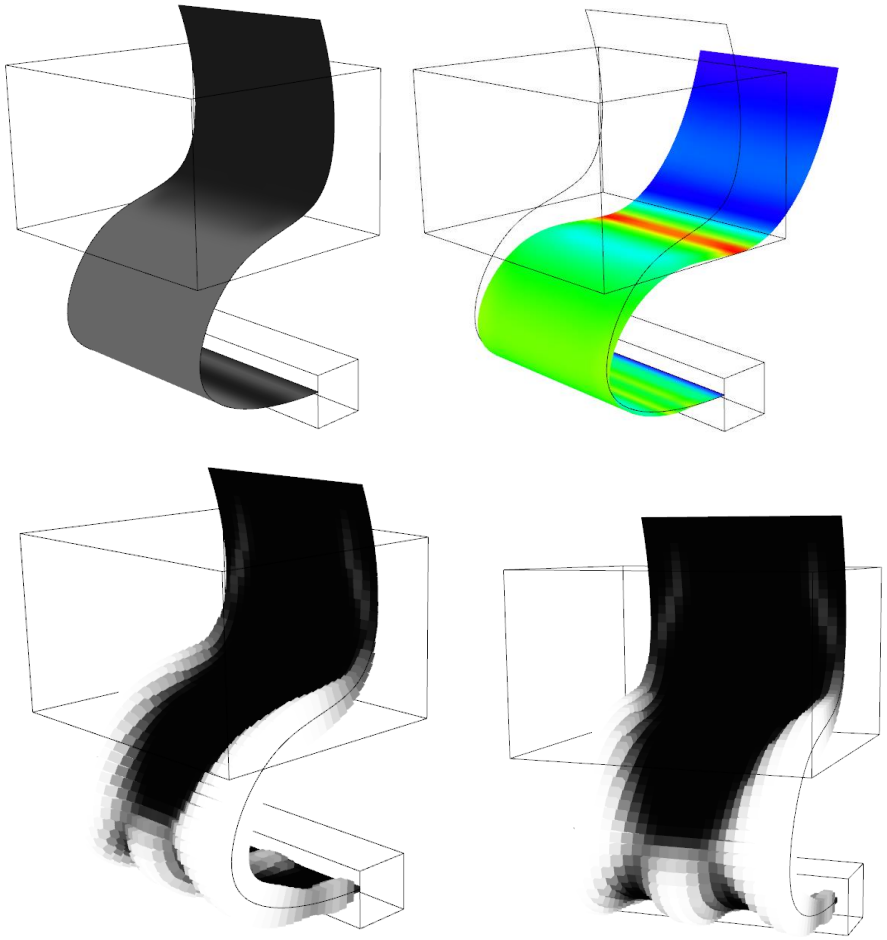
g. **004_leaf.gh**

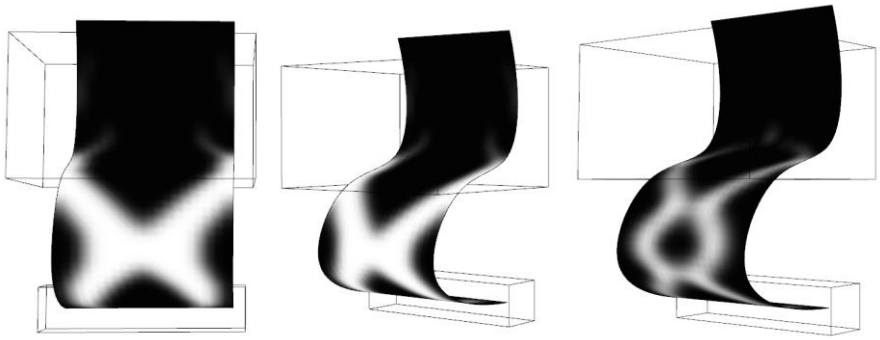




Cantilevering slab with thickness optimization. Branching structures emerge.

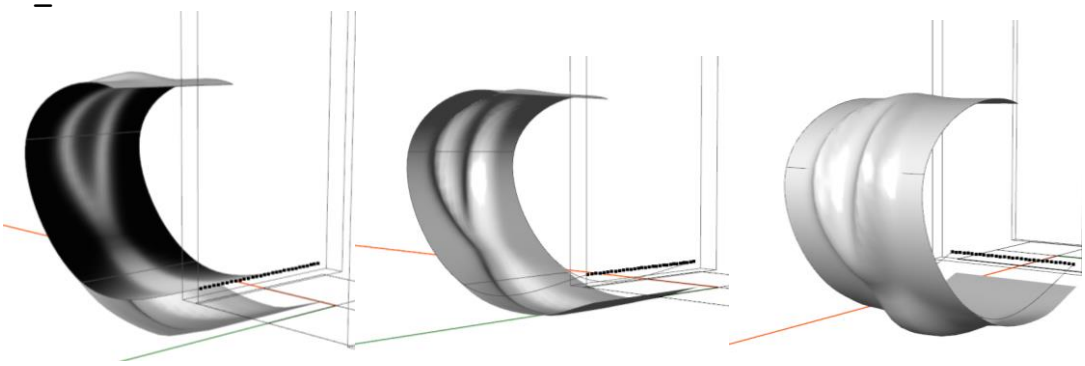
h. **005_chair.gh**





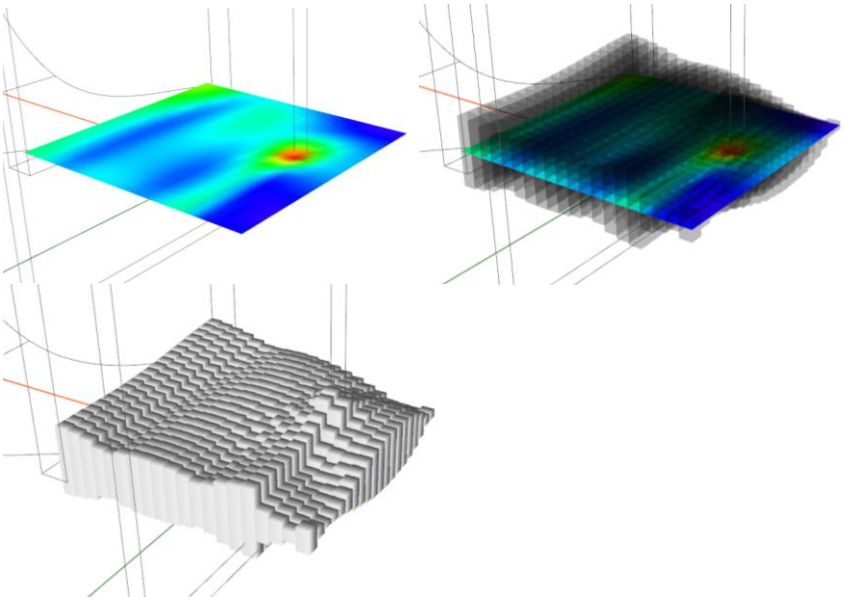
An s shaped chair and its thickness optimization diagram. Most material is concentrated on the bulging part that tends to fold on itself.

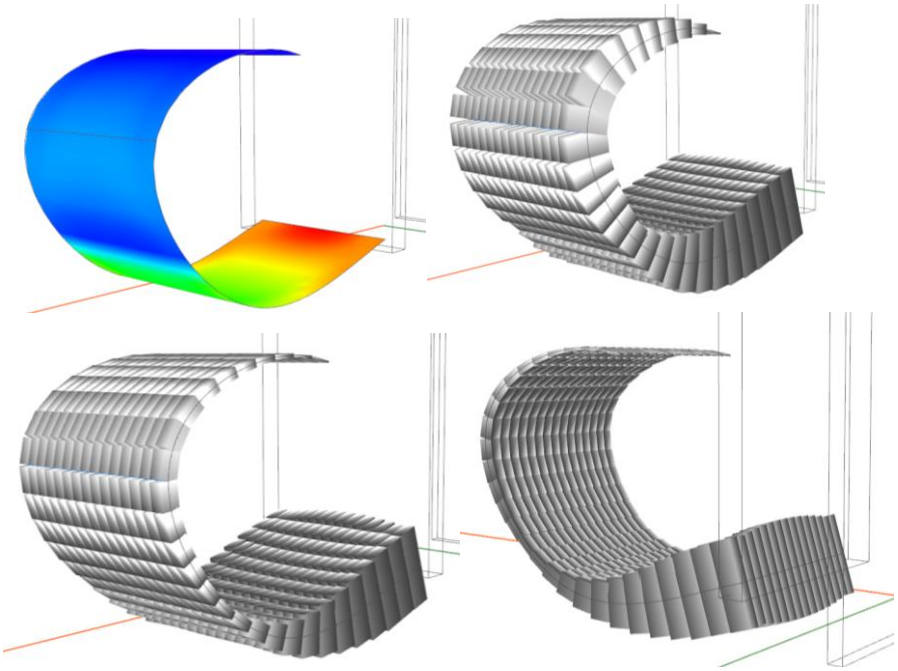
i. **06_ThicknessDistribution**



this example uses a topology optimization style method to change the thickness of the shell elements.

j. **05_ThicknessByStress**





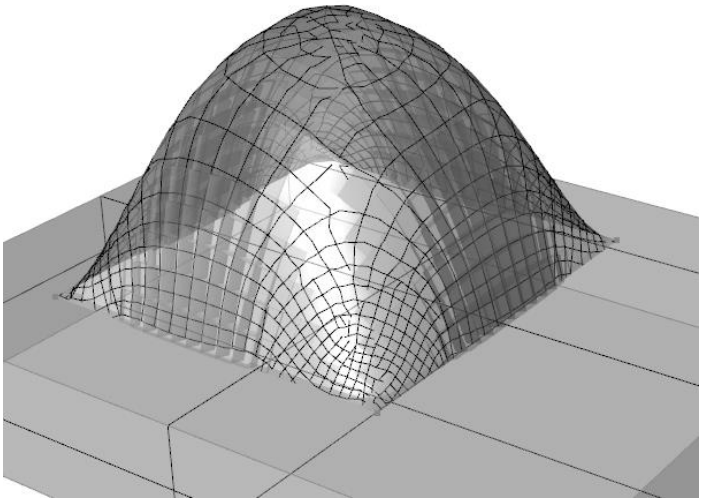
This example demonstrates the use of scripting for the modification of the thickness of a surface according to calculated stresses.

After the regular analysis is done, a c# component reads the stresses from the mesh quads and remaps them to a range of thicknesses which are then applied to the system and re-evaluated.

This method could be iterative and gradually redistribute thickness using multiple analysis steps. In this example we only see the first step [analysis->thickness->analysis]

In addition because stresses tend to have steep variations we remap the stresses to thickness by taking a fractional power [e.g. square root or less] of the stresses, thus creating a smoother distribution.

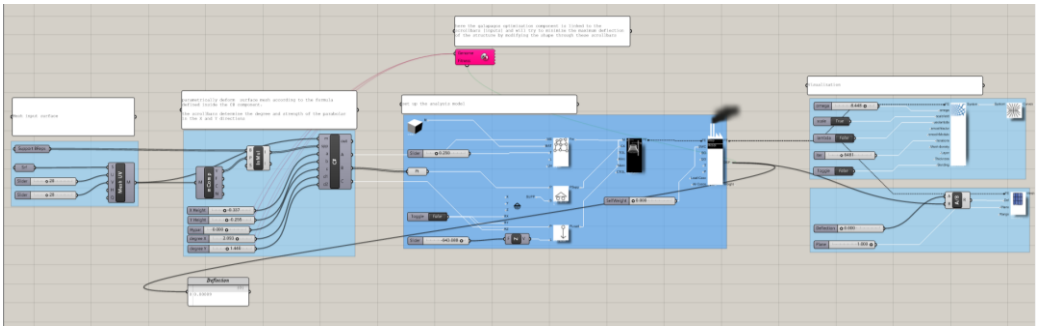
k. 025_readingresults.gh

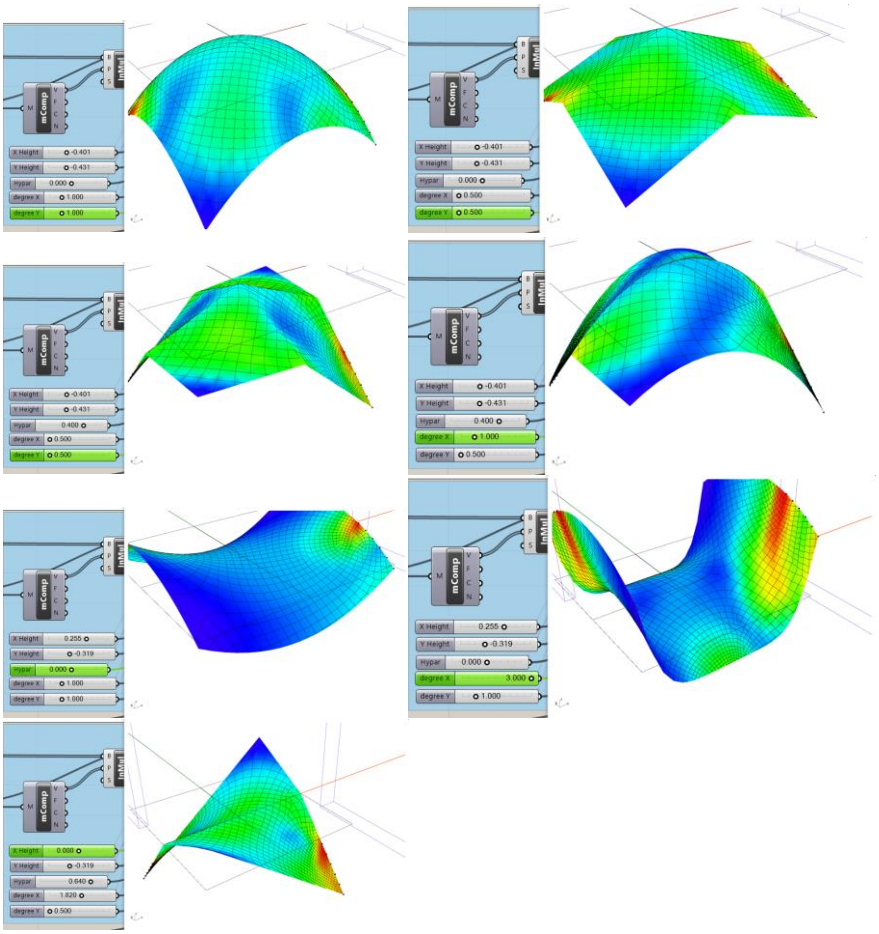


Shell analysis and stress lines.

11. Form Finding

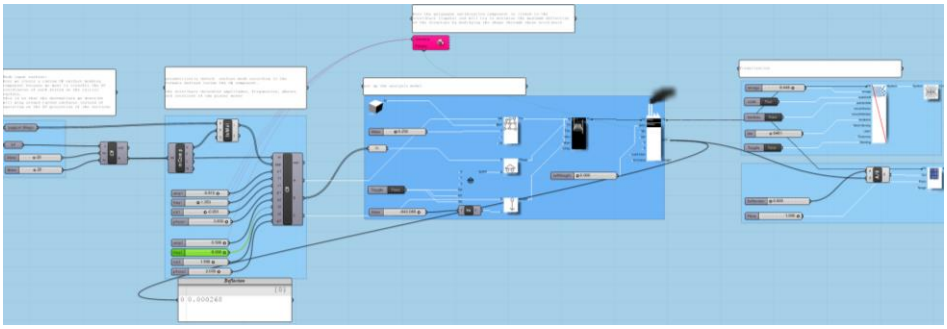
l. 01_ParabolicVaults

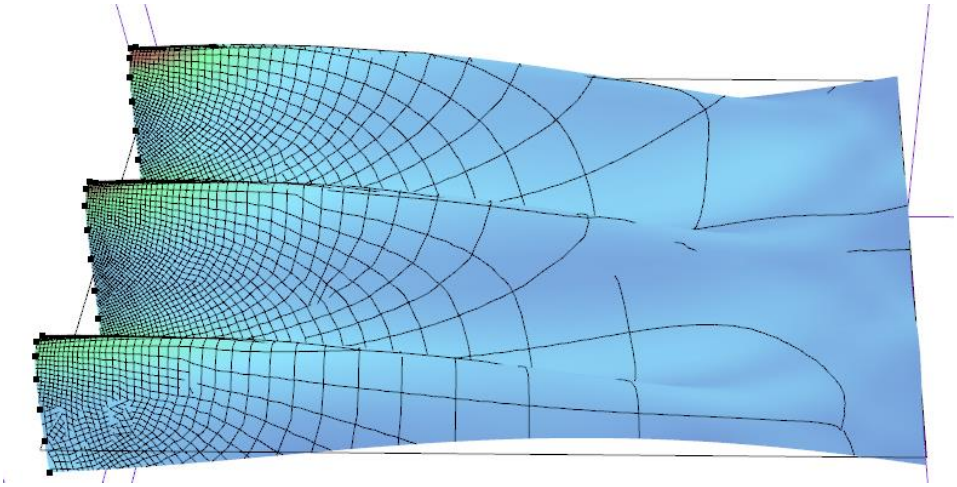


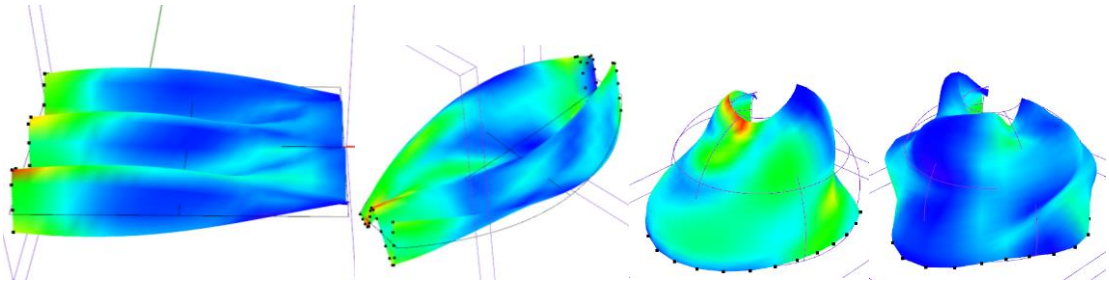


This example uses a simple script to generate parabolic and hyperbolic surfaces of varying degrees along the X and Y directions. With just 5 numbers you can explore a large family of surfaces and observe how the different shapes fair with the applied boundary conditions. You can also connect the parameters to the optimization module to search for optimal solutions.

m. 02_WaveFunctions

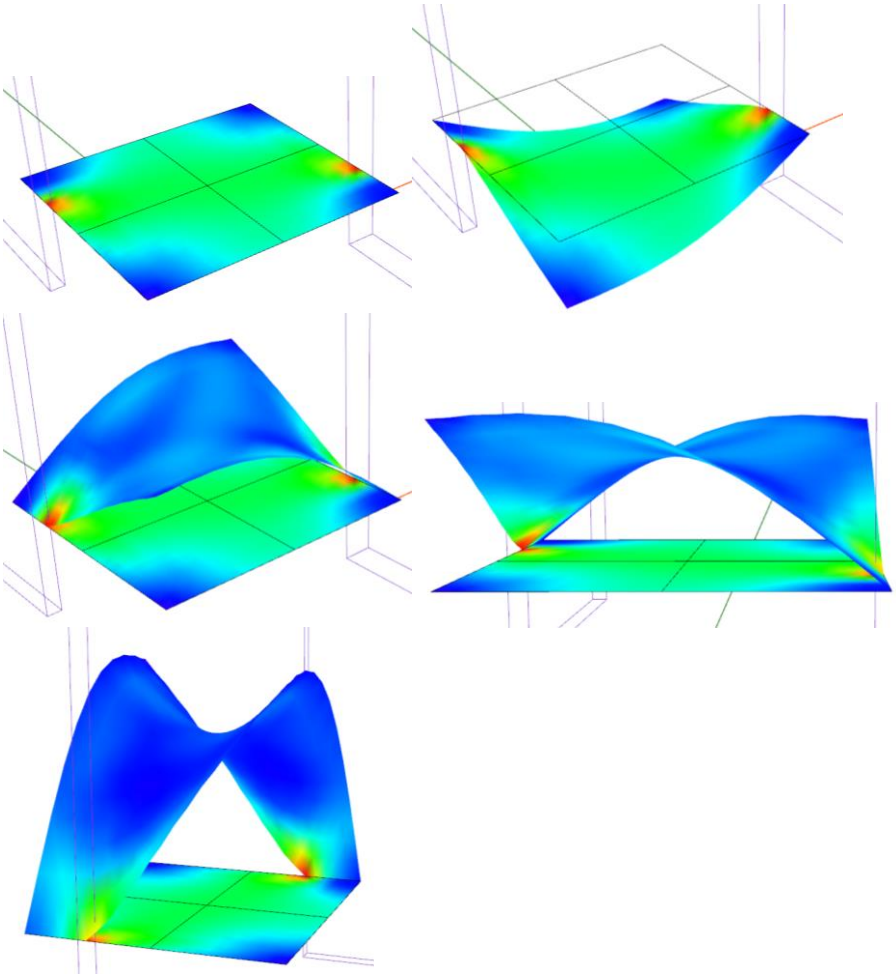






This example uses a script that superimposes two waves wrapping around any given surface. Wave like functions allow you to increase the structural depth and stiffness of the initial geometry by creating ridges. By controlling the amplitude frequency and phase of the waves you can explore a large design space with just a few parameters.

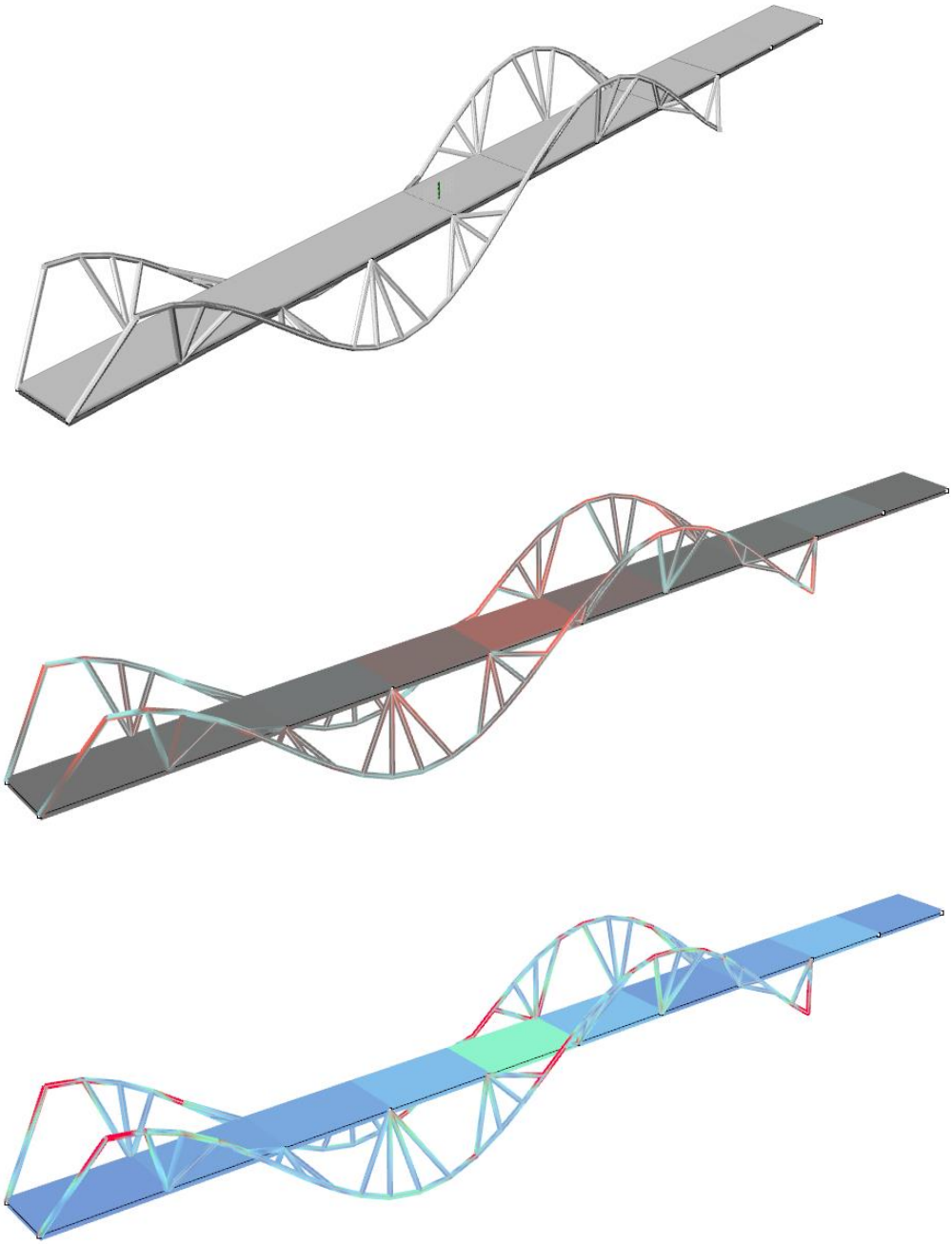
n. 03_InvertedDisplacement



This example demonstrates the use of scripting in order to extract the displacements at the nodes after the analysis step. This is done in order to invert the direction of these displacements and use them as shape

modifiers for the original geometry resulting in a vault structure. The analysis is repeated with the new shape and the results compared.

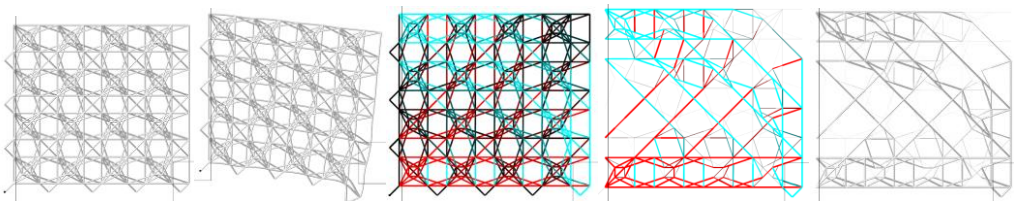
o. 020_bridge_02.gh

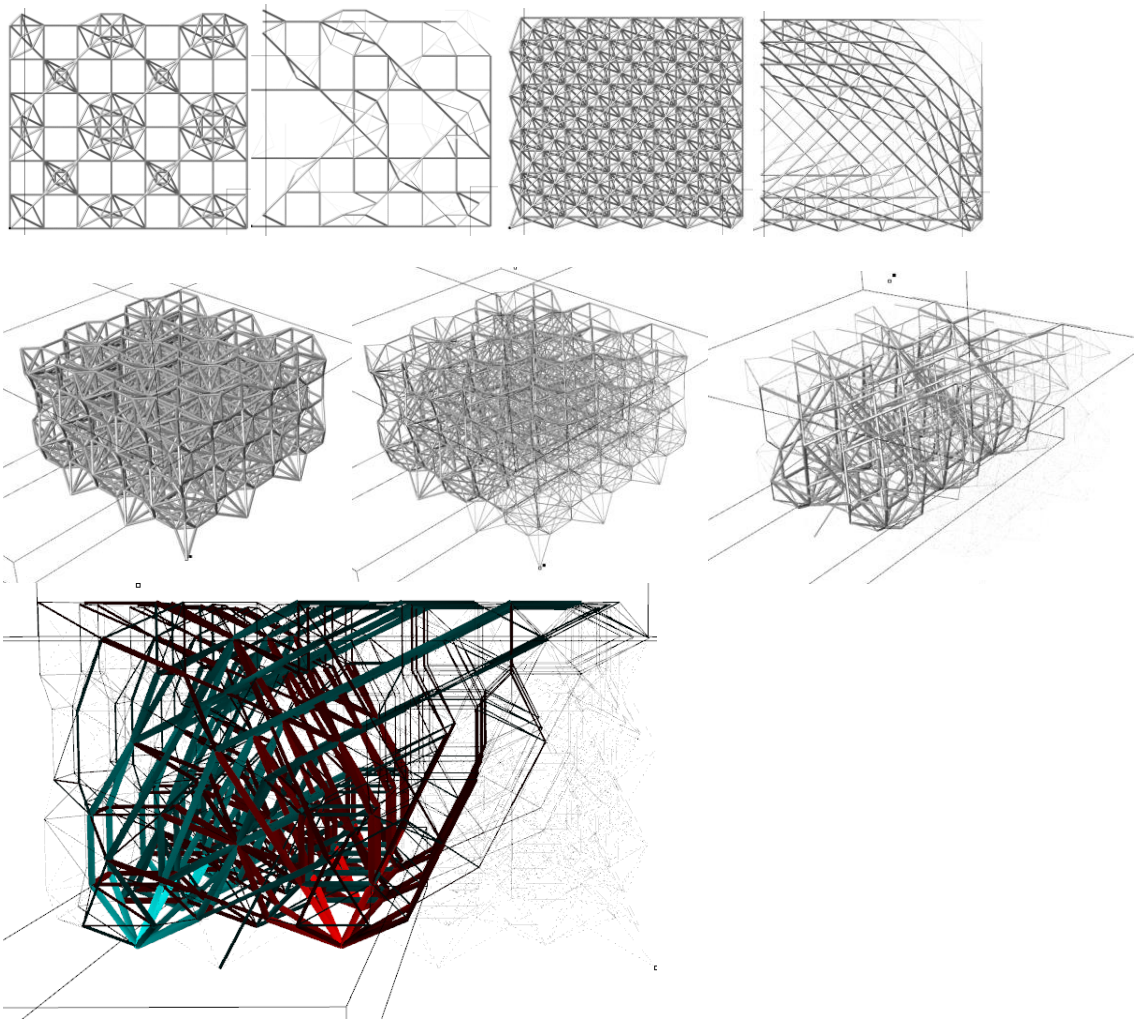


Use of the millipede structural analysis in conjunction with Galapagos for form finding.

12. Size Optimization

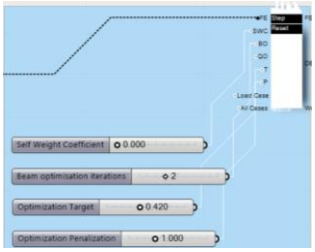
p. 06_patternSizeOptimization





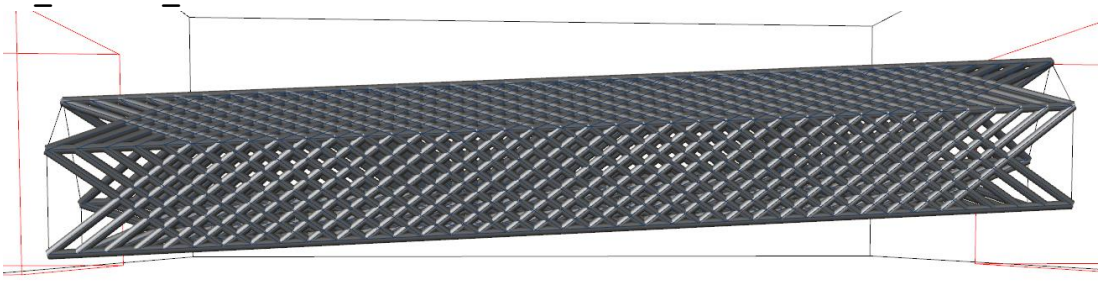
This and the next examples demonstrate the use of a sizing optimization algorithm for frame elements built into millipede. This algorithm iteratively reassigns thicknesses to the frame elements in effect converging to a system with fewer elements but minimum compliance.

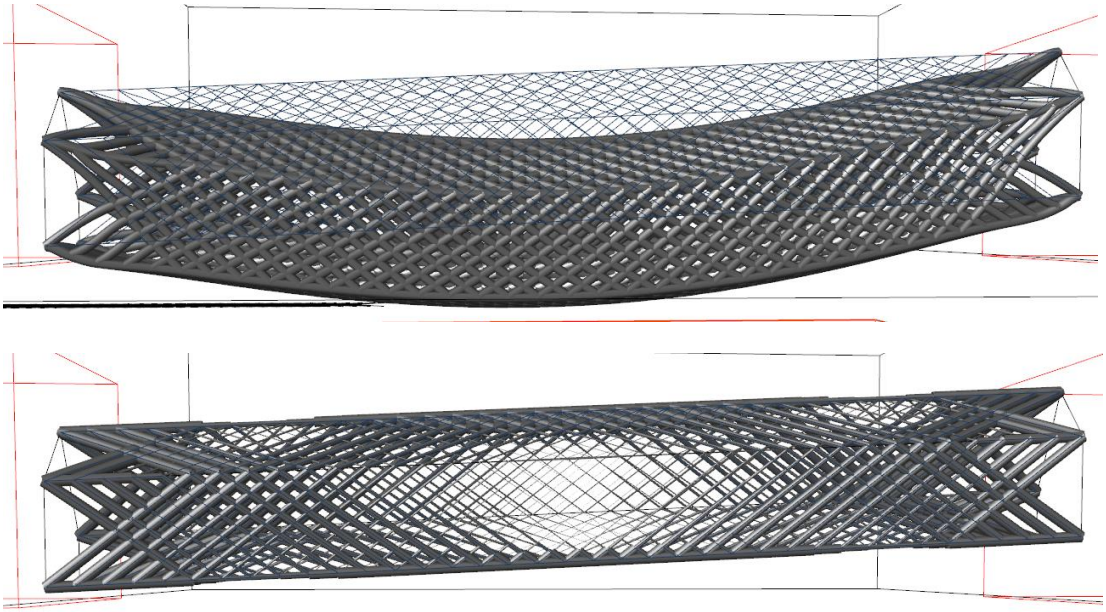
This file demonstrates the generation of dense patterns and the subsequent sizing optimization which leads to a reduction of redundancy in relation to the given load case.



Press repeatedly on the “Step” button to initiate optimization until you start seeing the sizes of the frame elements stabilizing.

q. 07_lattice_truss



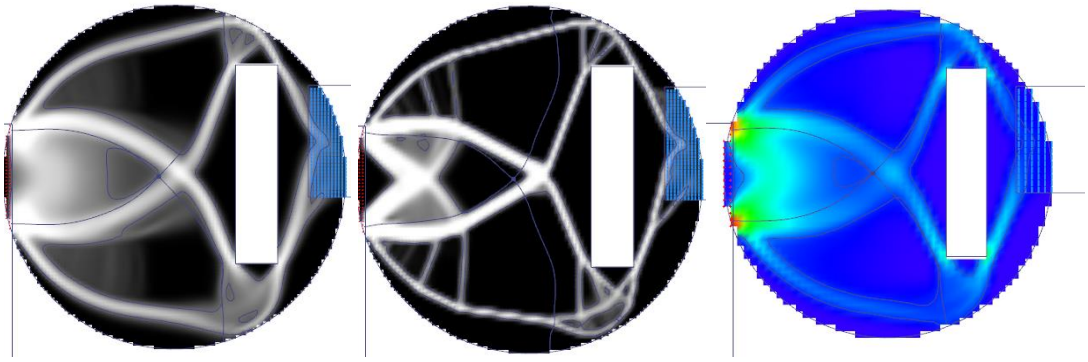


This example demonstrates how to use the size optimization in code. The relative thicknesses can be recovered as the StiffnessMultiplier value for each beam. This is a number between 0.0 and 1.0 designating the relative “importance” of the member for structural stability. Elements that have a near 0.0 value are not needed while elements with a value of 1.0 should be considered for reinforcement. After the analysis and optimization is finished the connected C# component selects all the elements that have a StiffnessMultiplier of 0.3 or more and outputs them as lines.

```
RStatSystem rs = ri as RStatSystem;


List<Line> slines = new List<Line>();

foreach (RStatBeam b0 in rs.Beams)
{

    if (b0.StiffnessMultiplier < 0.3) continue;

    slines.Add(new Line(b0.P0.ToPoint3d(), b0.P1.ToPoint3d()));
}

A = slines;
```

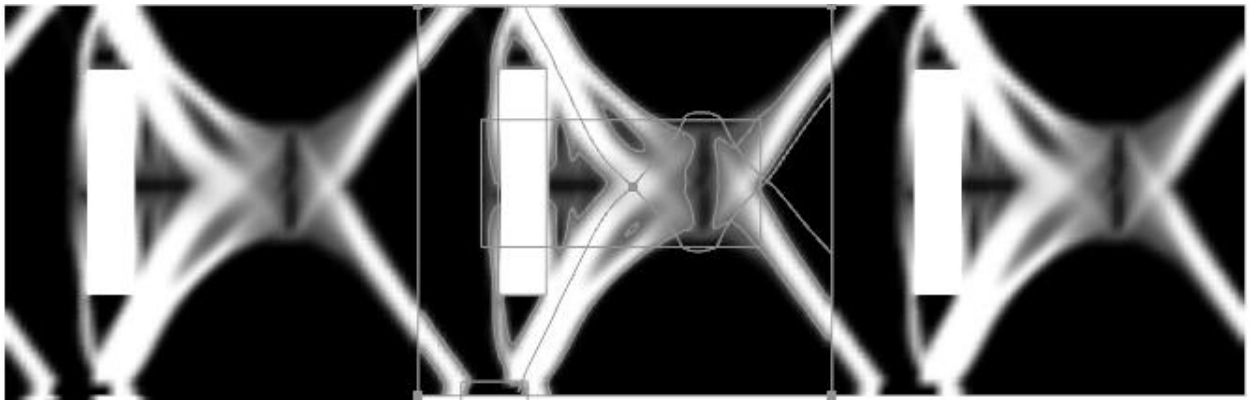
13. Topology Optimization 2D

r. 006_topostruct2d.gh



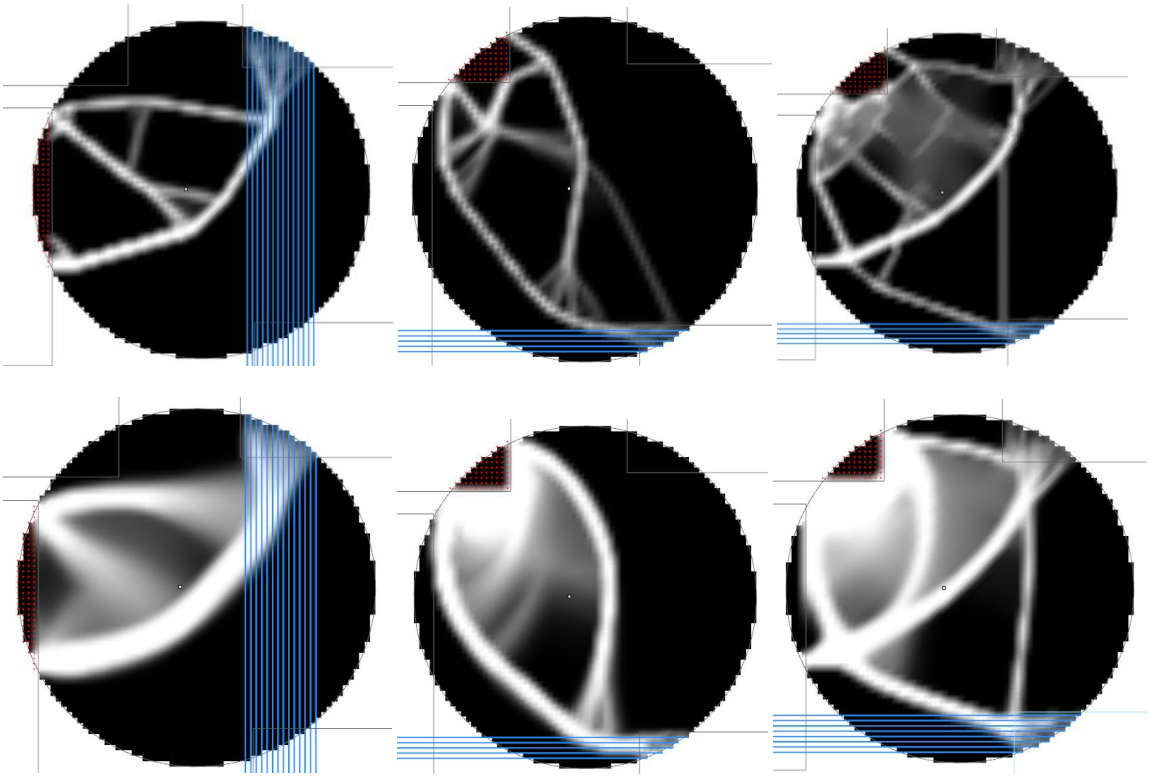
Simple two dimensional topology optimization with a circular boundary and an explicit rectangular void.

s. **007_topostruct2dPeriodic.gh**



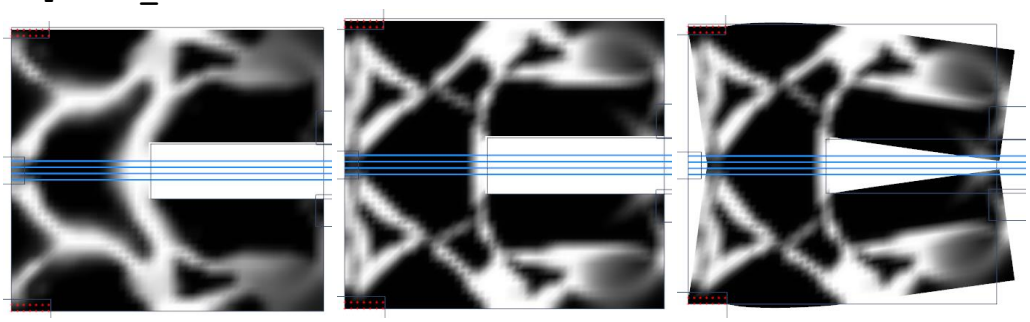
Periodic two dimensional topology optimization. By enabling the periodic option at the solver component the analysis considers the nodes at the boundary to be identical. this is useful for designing patterns in high resolution while analysing only a single module.

t. **b006_topostruct2dLoadCases**



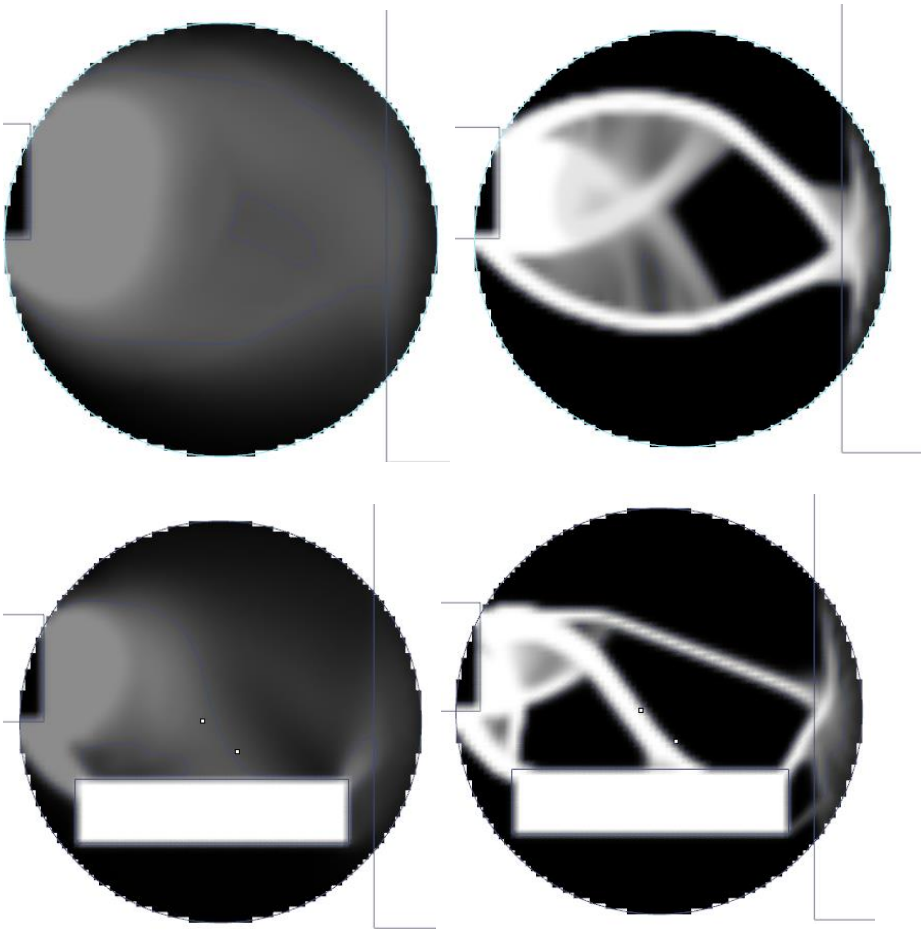
this example demonstrates the use of topology optimization when multiple load cases are considered. You can optimize against any of the load cases or against all.

u. **compliant_05**



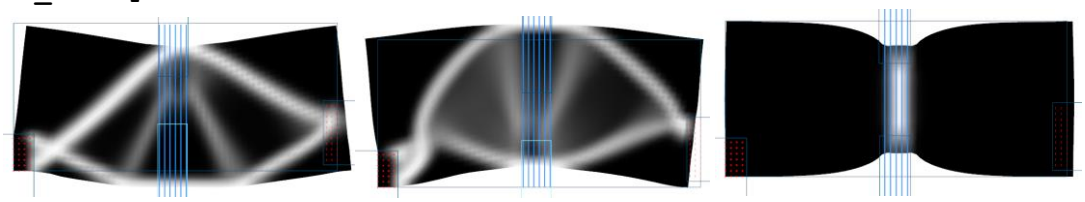
this example demonstrates the design of a compliant pincer mechanism using two loadcases for input and output actions.

v. **w3_explicitSolid**

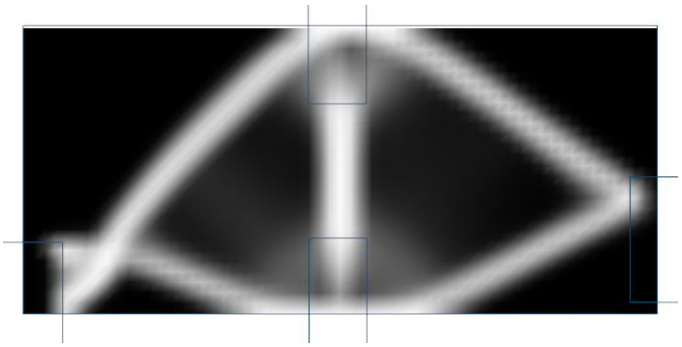


demonstration of use of explicitely defined solid region within optimization domain

w. **w5_multipleLoadCases**

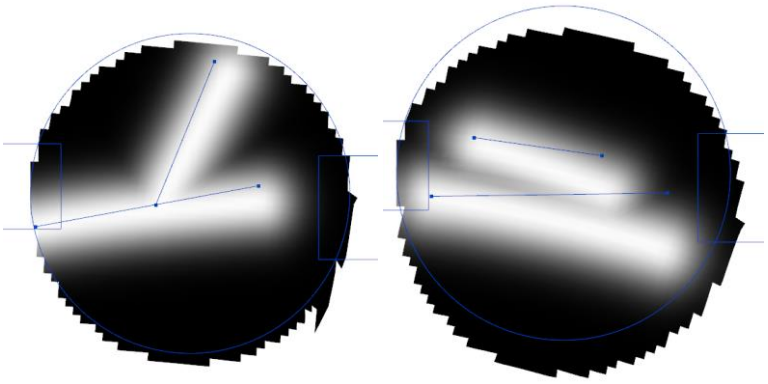


two loadcases applied separately [left and center] and simulaneosly [right]. Notice how they would cancel each other.



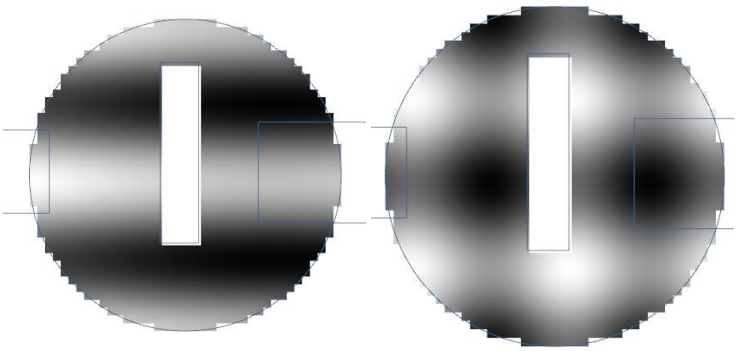
Optimal distribution against all load cases. This optimal is not the same as applying the two loadcases simultaneously [where they cancel each other], instead it is optimal with all loadcases separately. That means that it will perform well no matter what loadcase it is subjected to.

x. w6_pointDensity



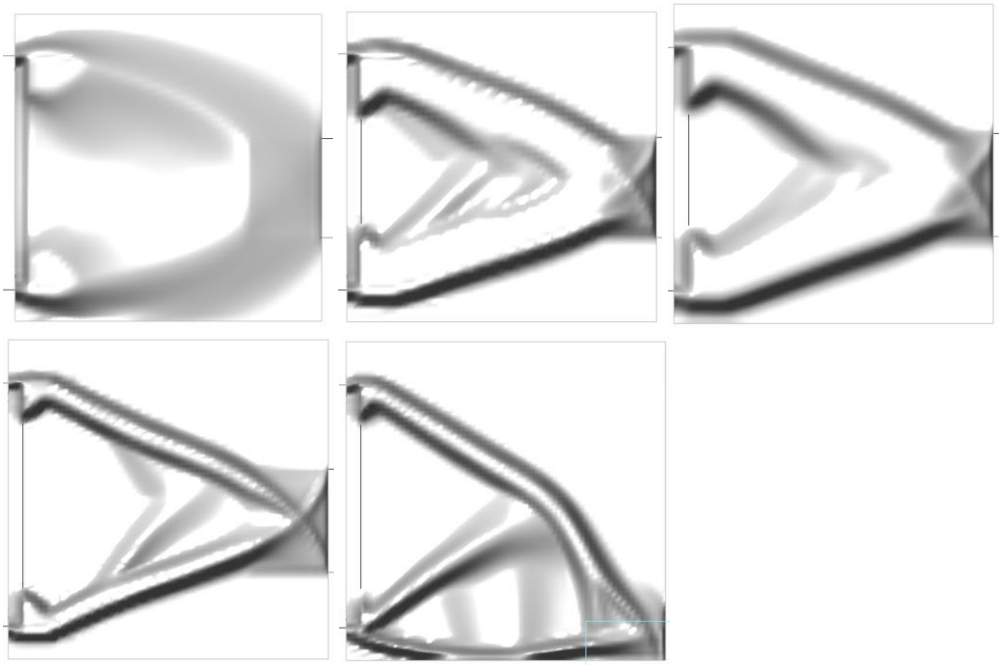
in this file we suspend the topology optimization module and explicitely define the density distribution by a smooth field around a couple of parametetrically defined lines. we can connect the coordinates fo the endpoints of the lines to galapagos and use it to search for optimal configurations.

y. w7_functionDensity



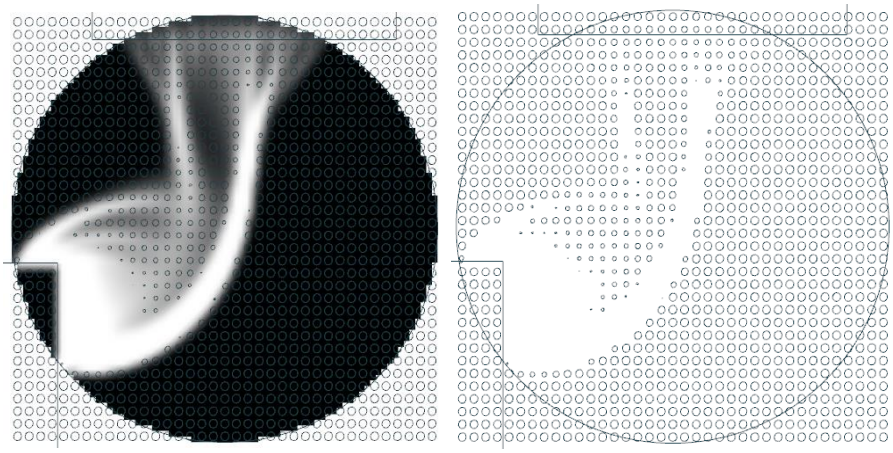
this example is similar to the previous one in that we explicitely control the distribution of density. However this time we use a formula with two parameters that we can connect to the optimization module.

z. w8_thickness



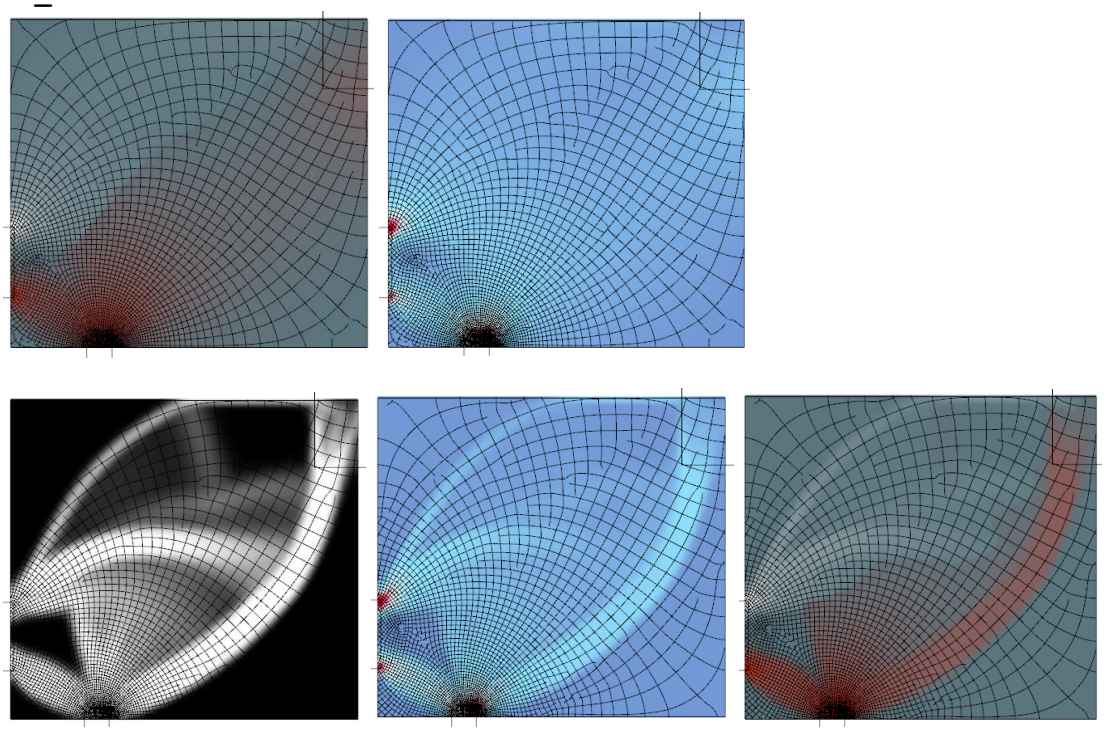
In this example the density field is translated into an offset surface with variable thickness.

aa. w9_raster



in this example the density distribution is translated into a perforation pattern. The pattern is determined by a small grayscale image allowing for quite complex halftoning.

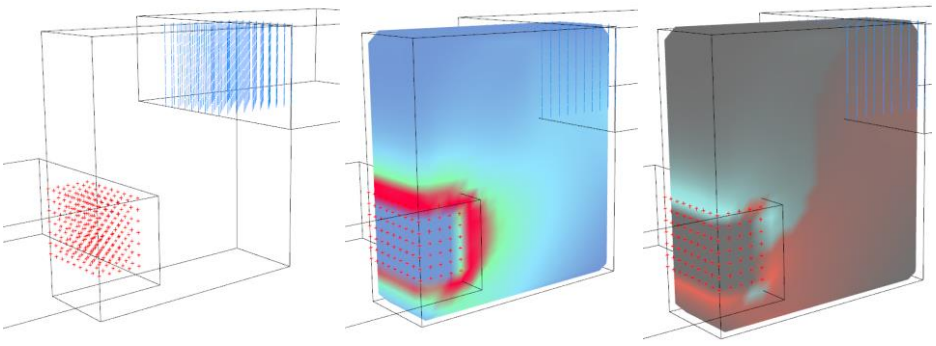
bb. w9_stresslines

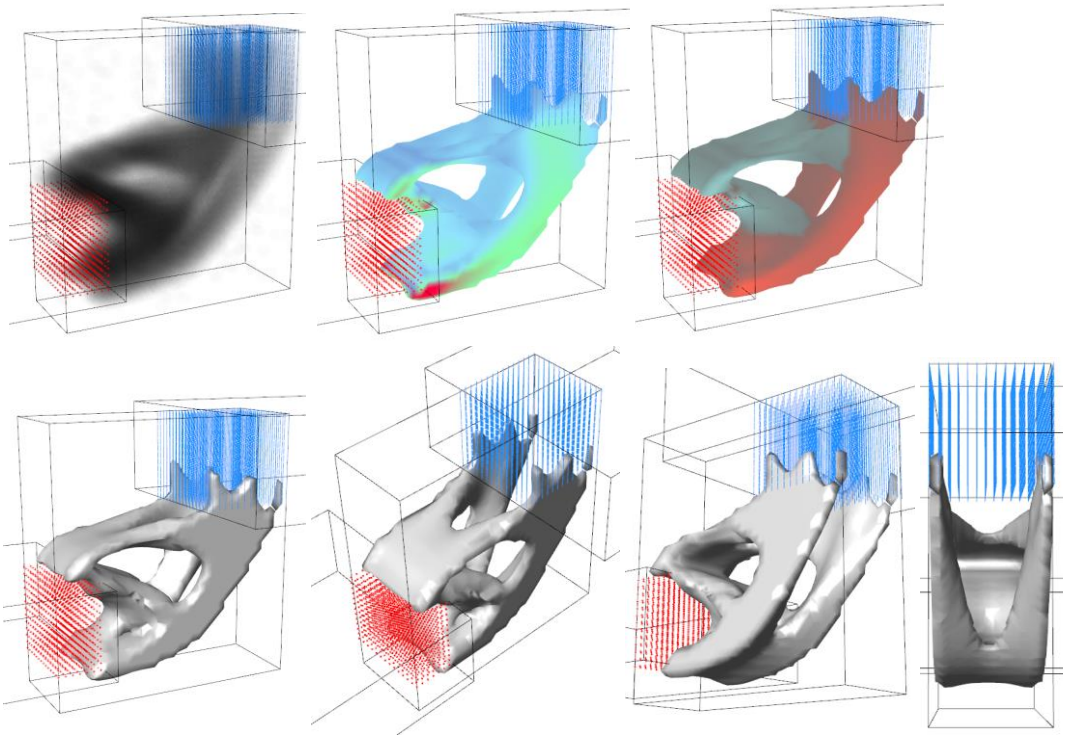


Stressline pattern extraction from plates. notice alignment of stress lines to optimal distribution.

14. Topology Optimization 3D

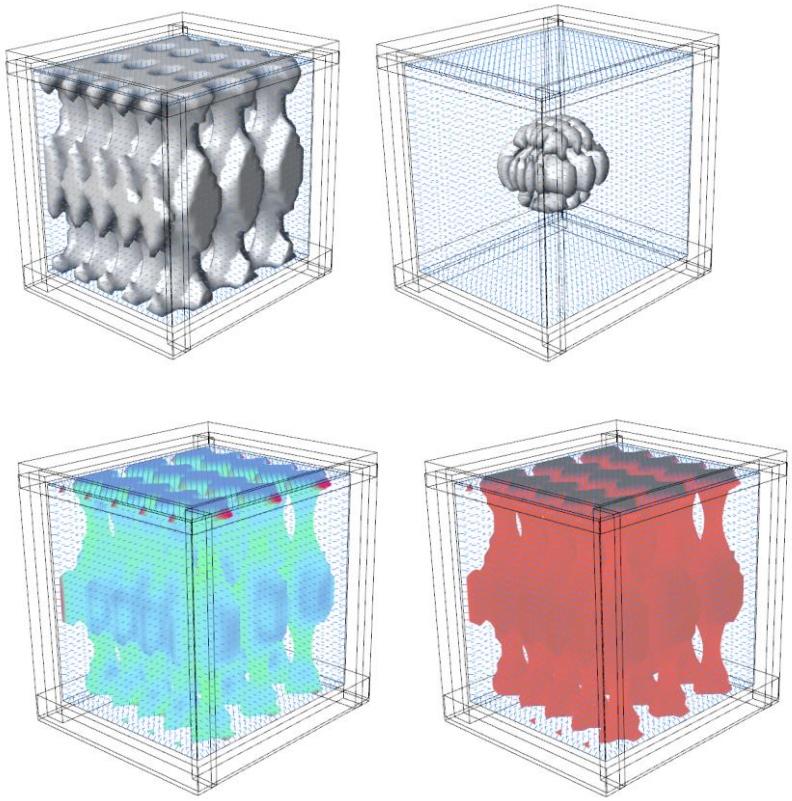
cc. 008_topostruct3d.gh





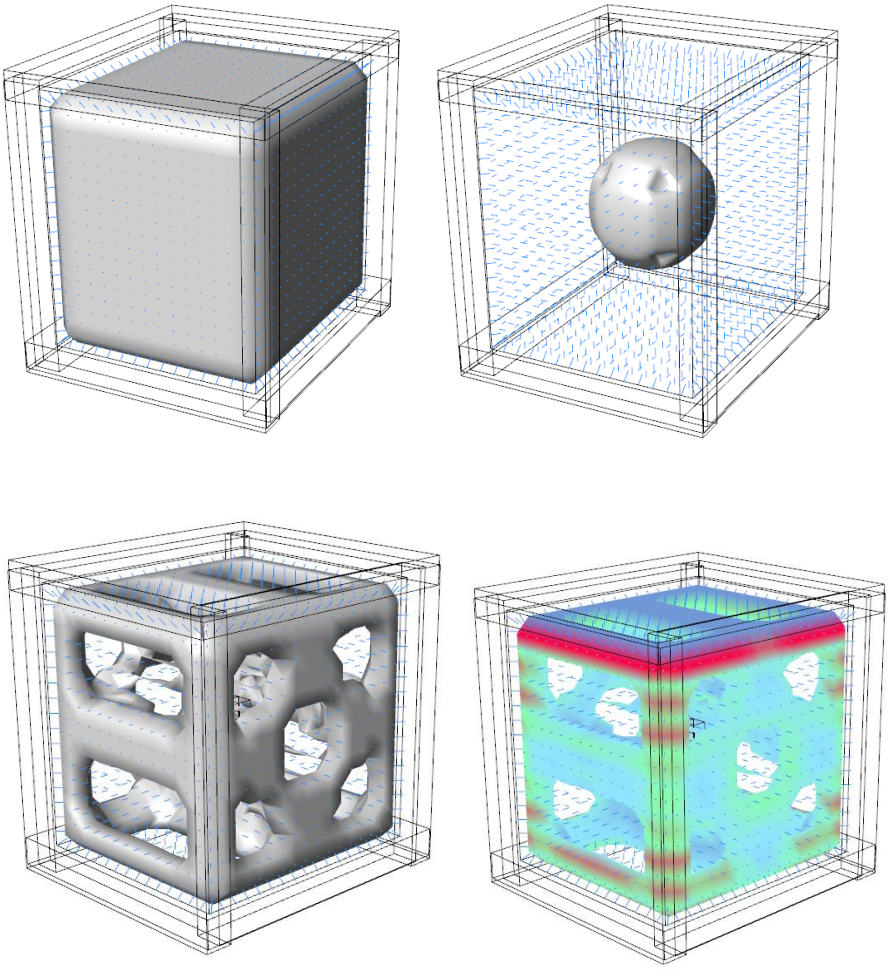
Basic three dimensional topology optimization.

dd. 009_CustomDensityFieldAnalysis.gh



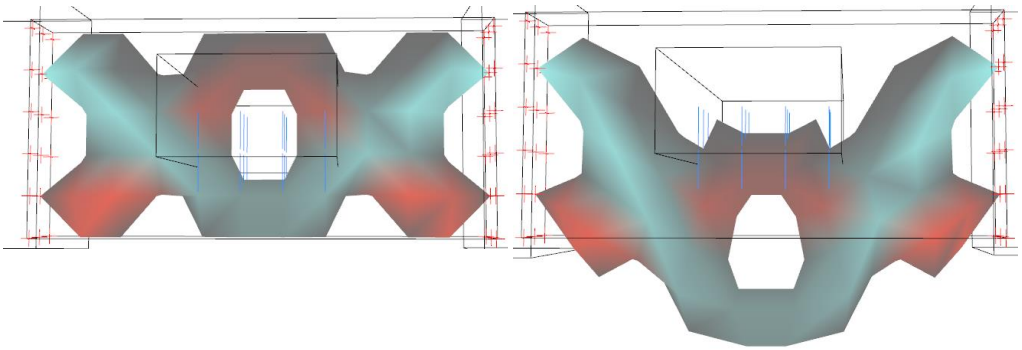
Analysis of inhomogeneous material distribution determined by a periodic density function. The six way loads apply an isotropic pressure to the cell.

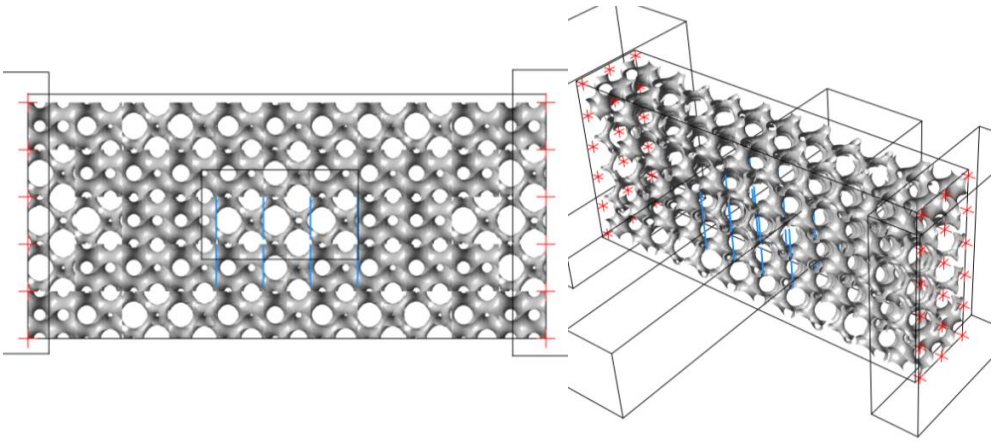
ee. 010_PressureCell.gh



Optimization of cell under omnidirectional pressure

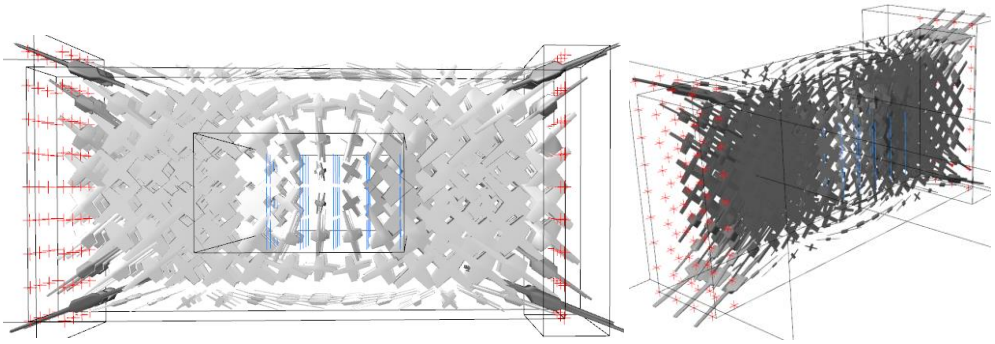
ff. 012_TopostructPorosity.gh





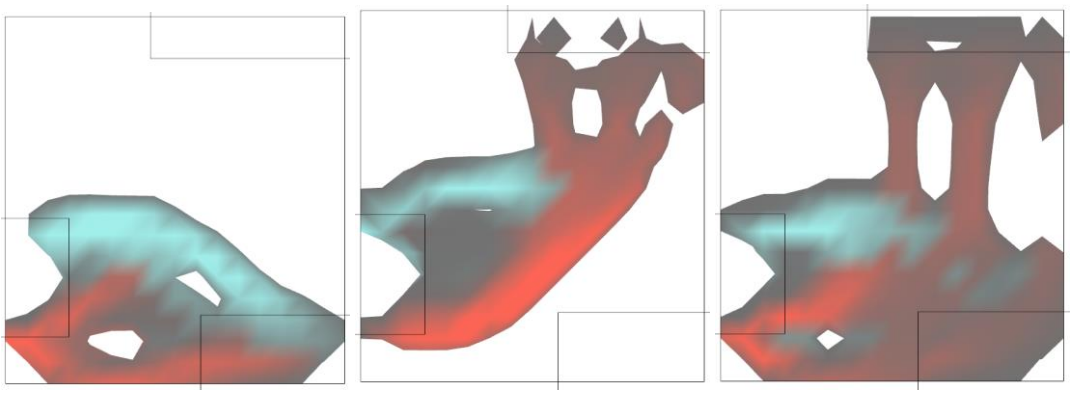
Use of iso surfaces of varying threshold to generate cells for variable porosity interpretation of topology optimization results.

gg. 013_stressAlign.gh



Visualization of principal stresses within a volume. This examples takes a simple mesh [a three dimensional cross] and maps to the principal stress tensor field.

hh. b008_topostruct3d

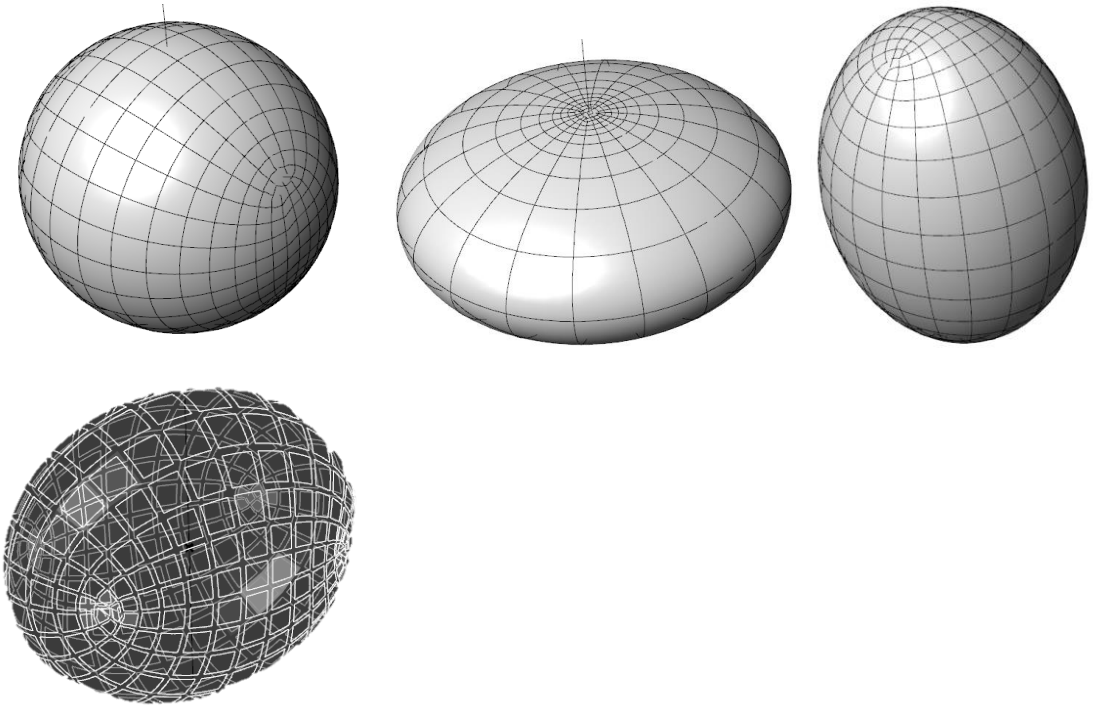


multiple loadcases in 3d topology optimization.

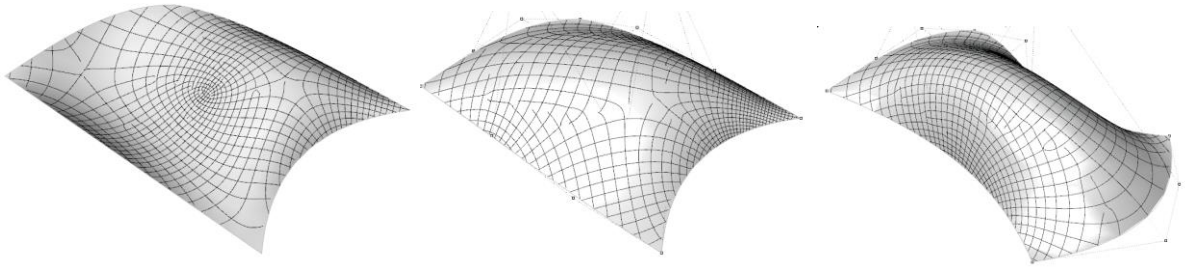
15. Surface Reparameterization

The following examples demonstrate the use of the reparameterization component. This component creates new UV coordinates for given meshes that follow some desired direction defined by a vector field. The component implements a variant of the Global periodic Parameterization algorithm developed by Nicolas Ray et.al at INRIA.

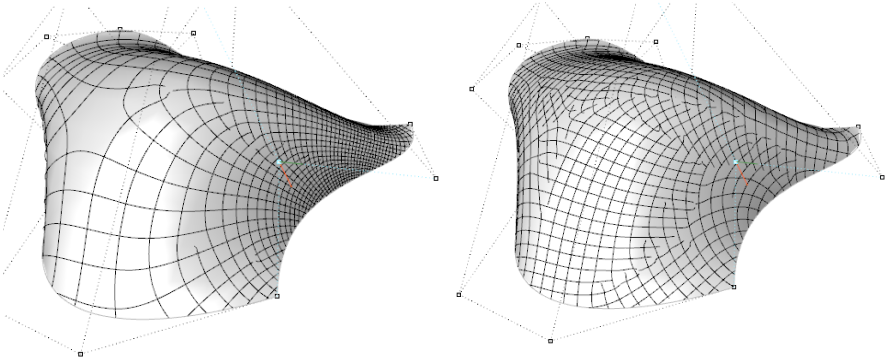
ii. workshop_curvature



curvature singularities change location as the shape of the ellipsoid

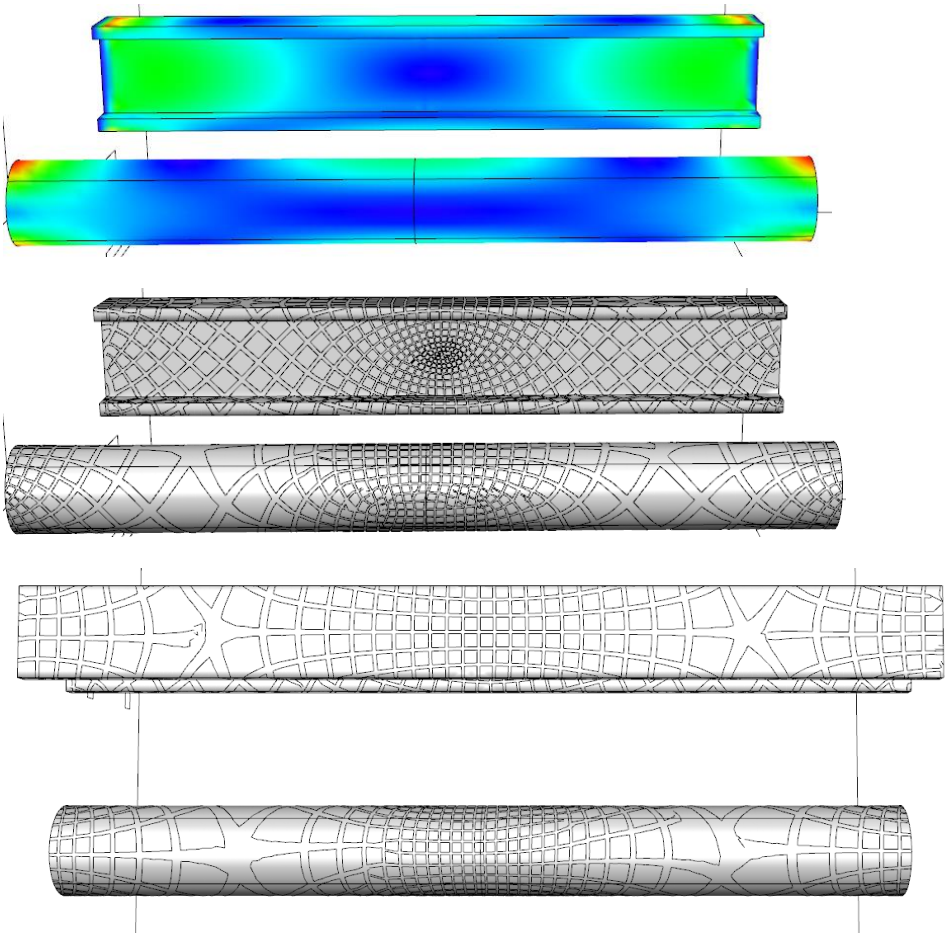


principal curvature aligned pattern over doubly curved surface



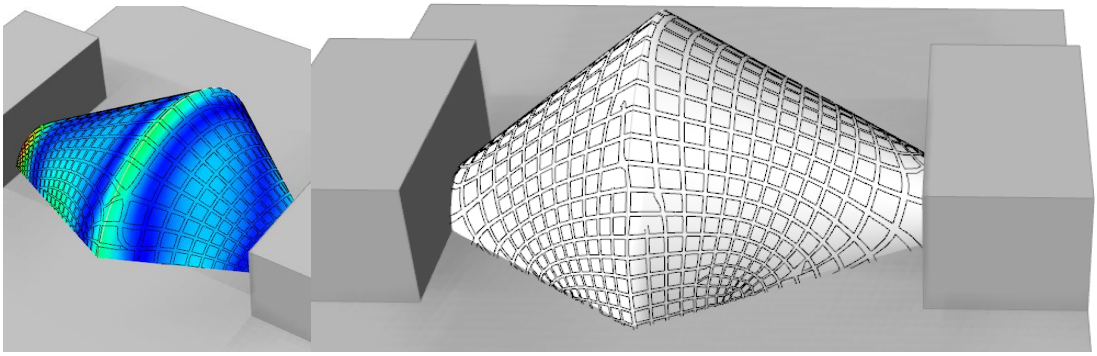
Same surface with principal curvature pattern, with scaling enabled [left] and disabled [right]

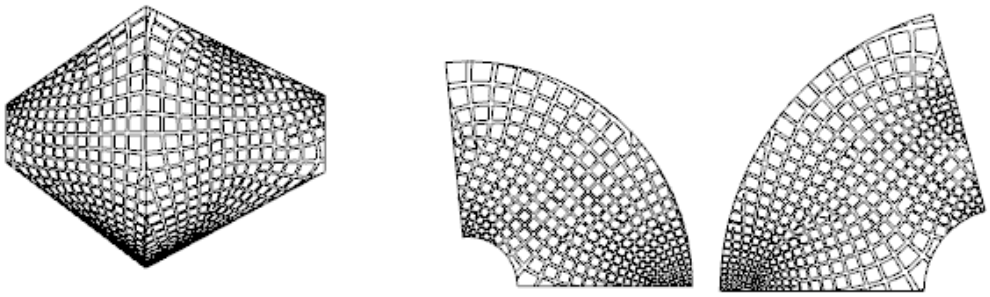
jj. workshop_stress_beams



Principal stress aligned pattern around beams modelled using shell elements

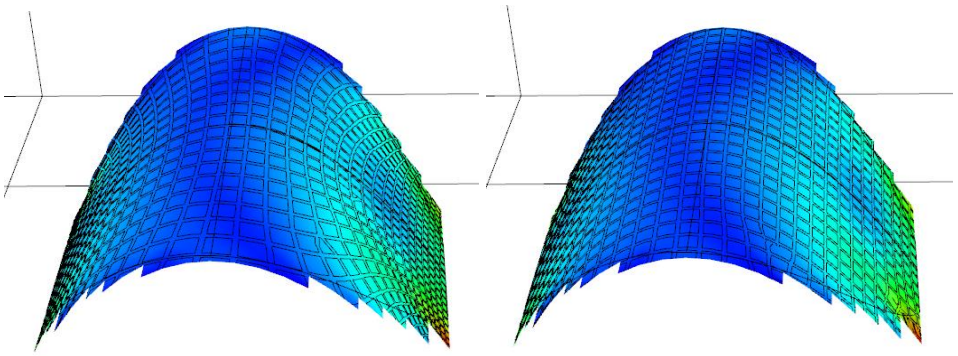
kk. **workshop_stress_curves**





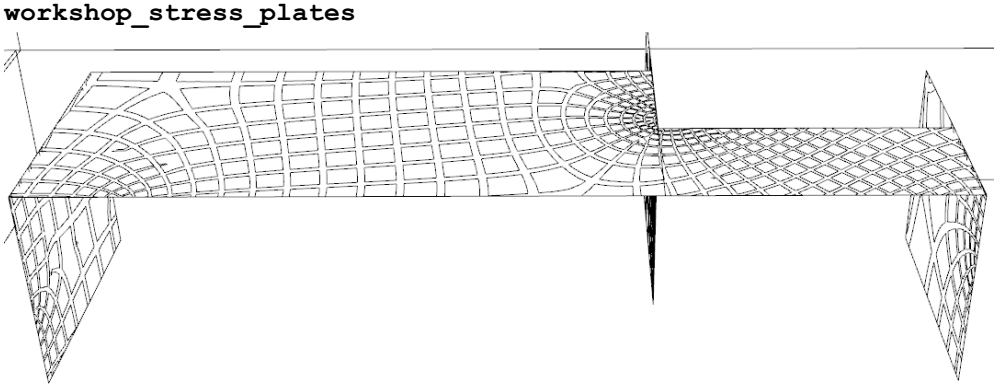
developable surface with stress aligned pattern and flattened pattern ready for laser cutting.

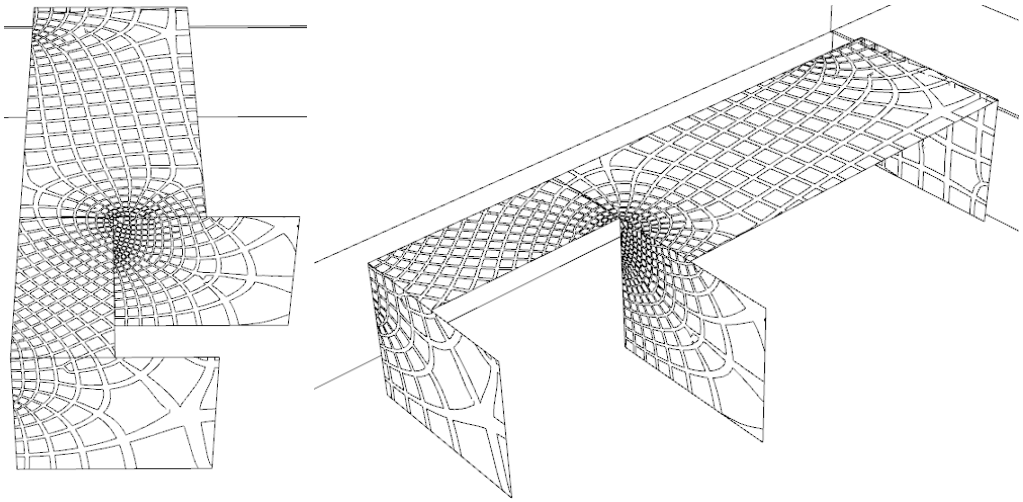
ll. **workshop_stress_curves_B**



pattern aligned to principal stresses [left] and bending moments [right]

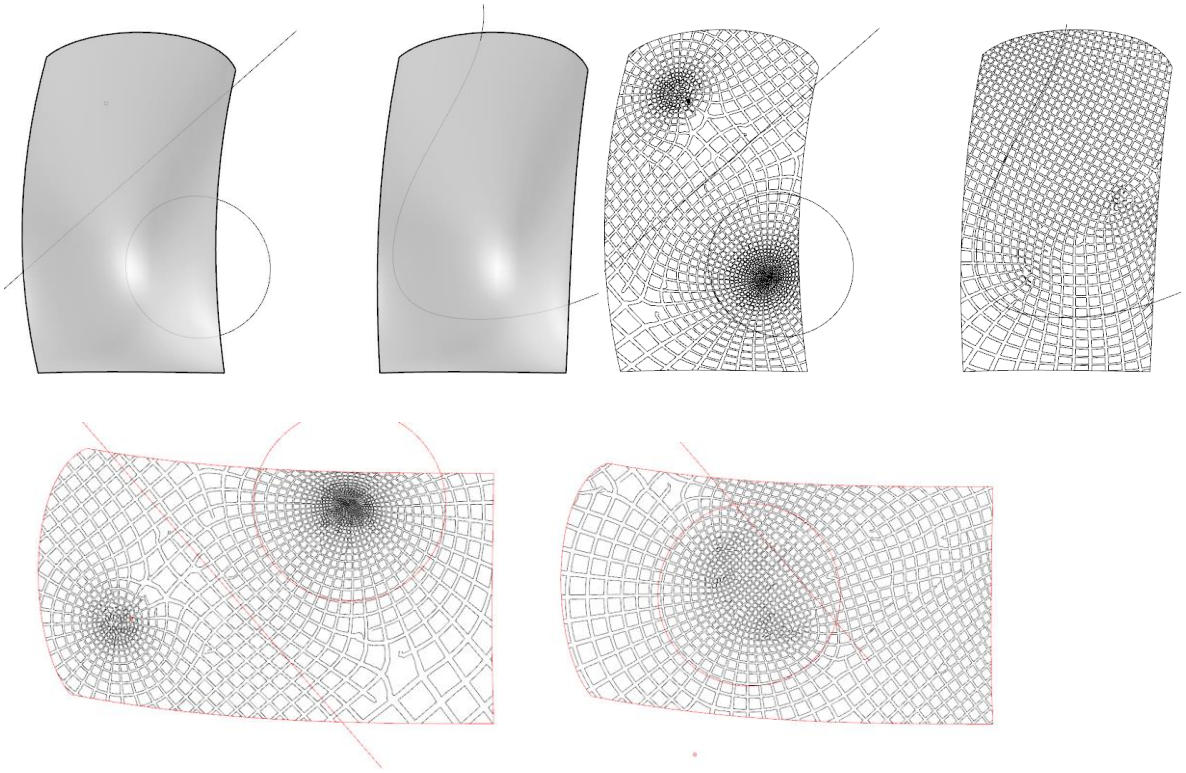
mm. **workshop_stress_plates**





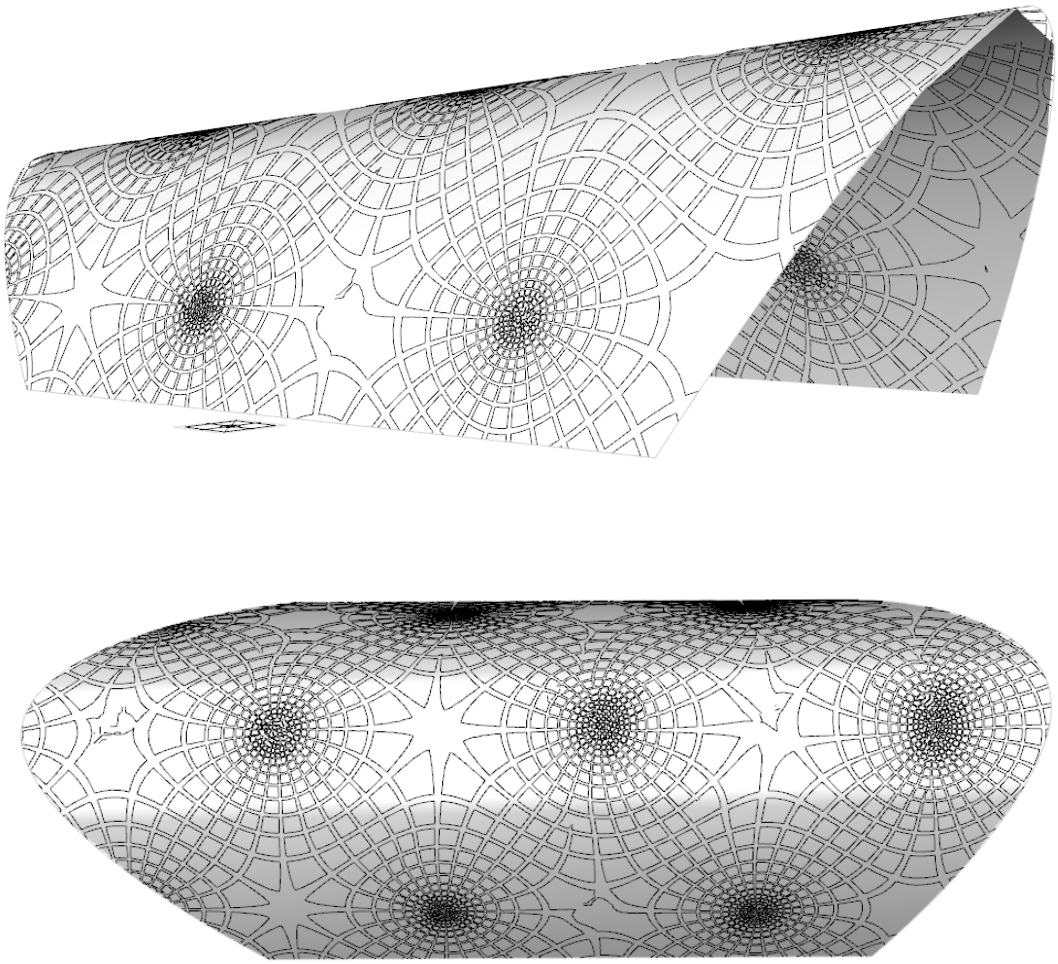
design of a bench with principal stress aligned patterns.

nn. workshop_tiling_attractors



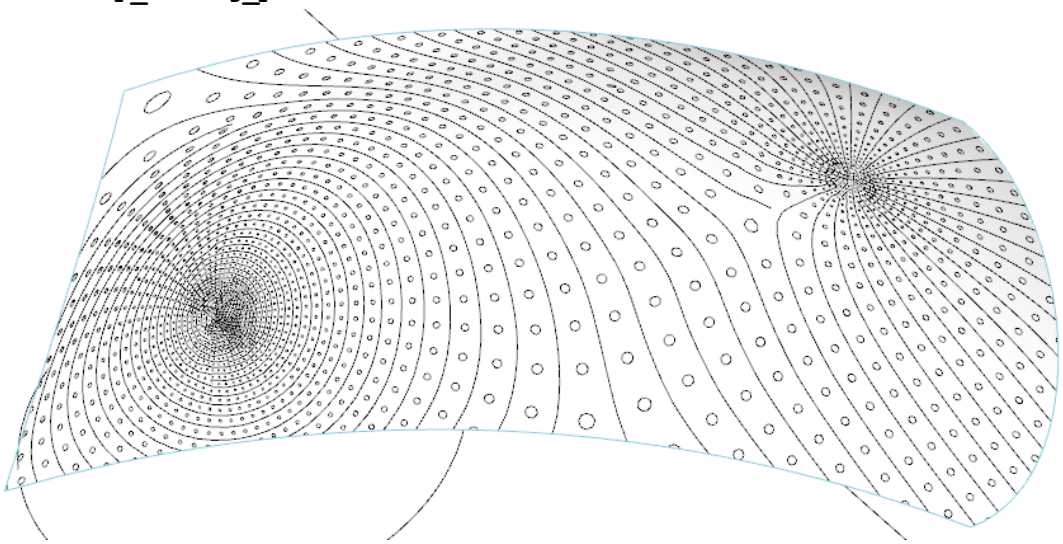
two surfaces are tiled using the reparameterization pattern generated to align with a vector field that wraps around control geometry [curves and points]

oo. workshop_tiling_function



use of a mathematical formula to control the vector field that guides the pattern.

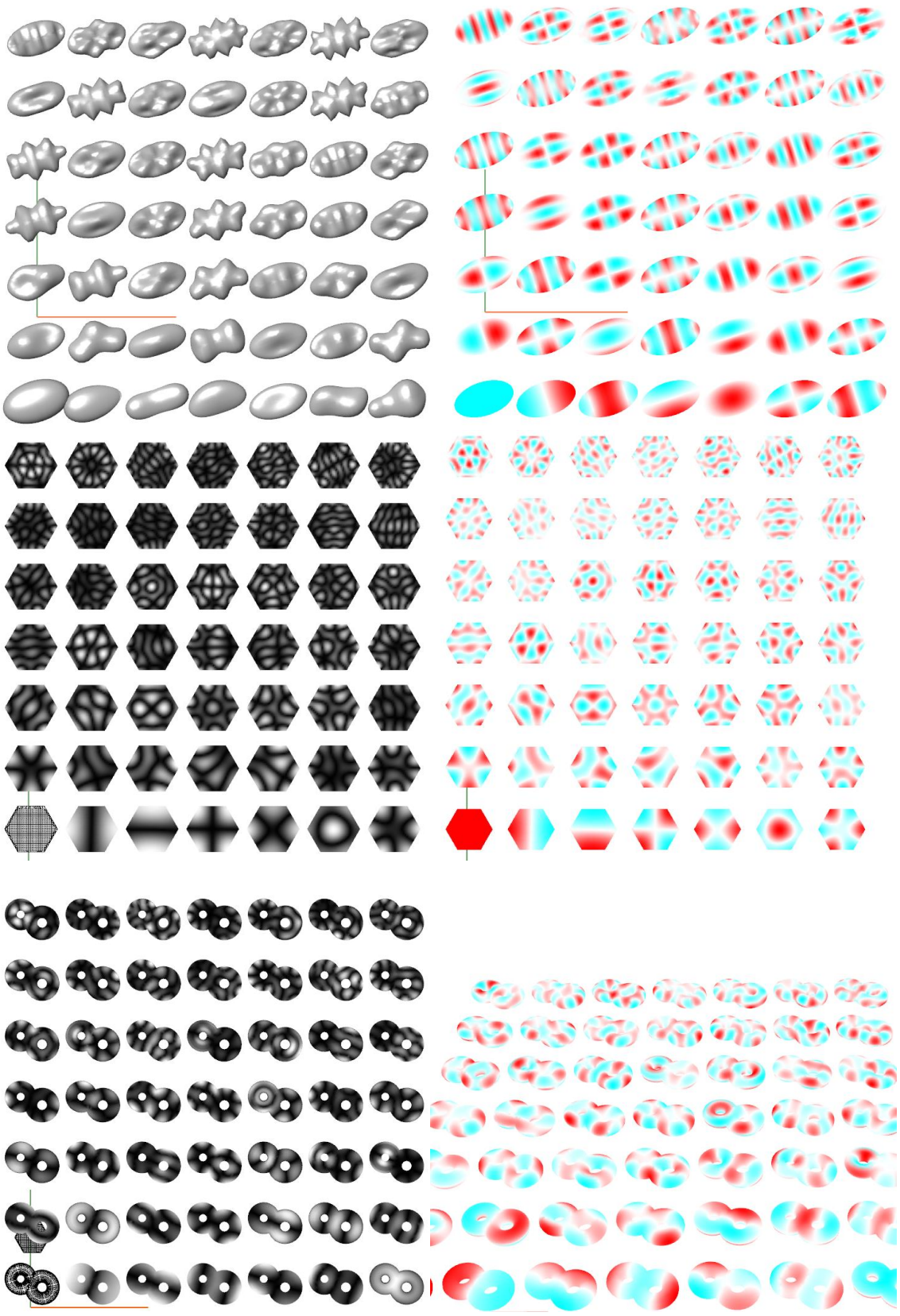
pp. workshop_tiling_pattern



after the parameterization has been calculated you can map any pattern over the input mesh

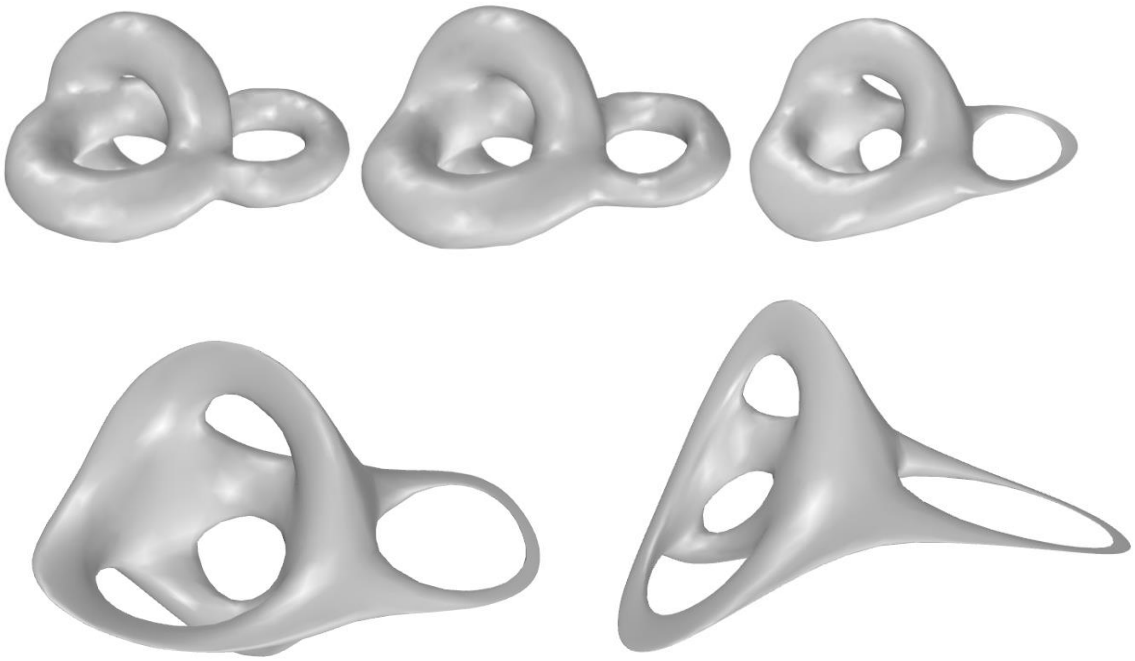
16. EigenMode Mesh Analysis

qq. 00_EigenModesVisualization



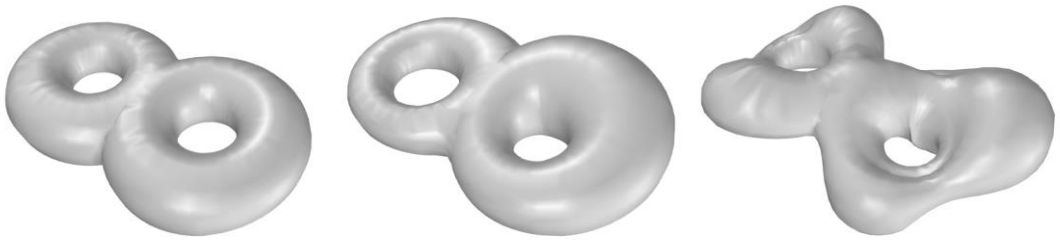
Using the eigenModes component visualize the first N eigenmodes of the Laplace Beltrami operator of different meshes.

rr. 01_EigenSmoothing



The smoothing component reconstructs a shape from its eigenmodes allowing you to filter out certain frequencies. It is equivalent to a low-pass or high-pass filter used in signal analysis. For example if you only keep the low frequencies [0 to say 15] you get a smoothing effect over the geometry.

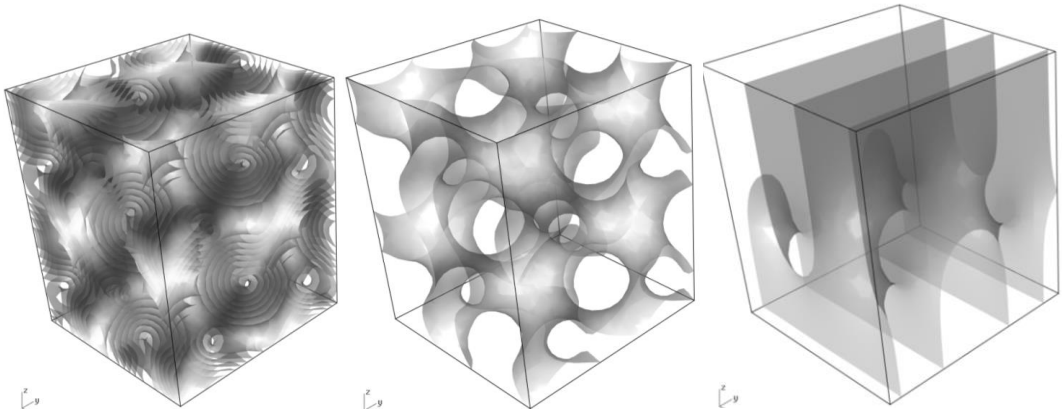
ss. 02_EigenDisplacement



We can use superposition of eigenmodes to define generalized deformation and displacement functions over any geometry. The scrollbars determine the strength of each frequency component in the final shape.

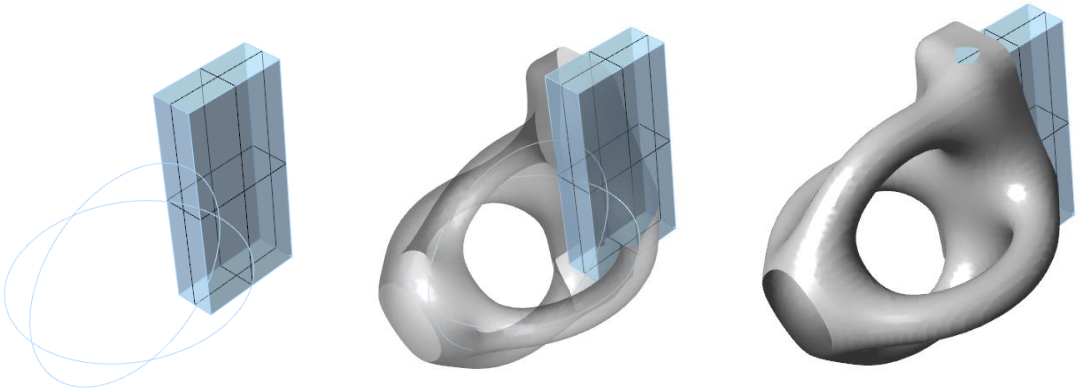
17. Geometry

tt. 011_isoSurfaces.gh



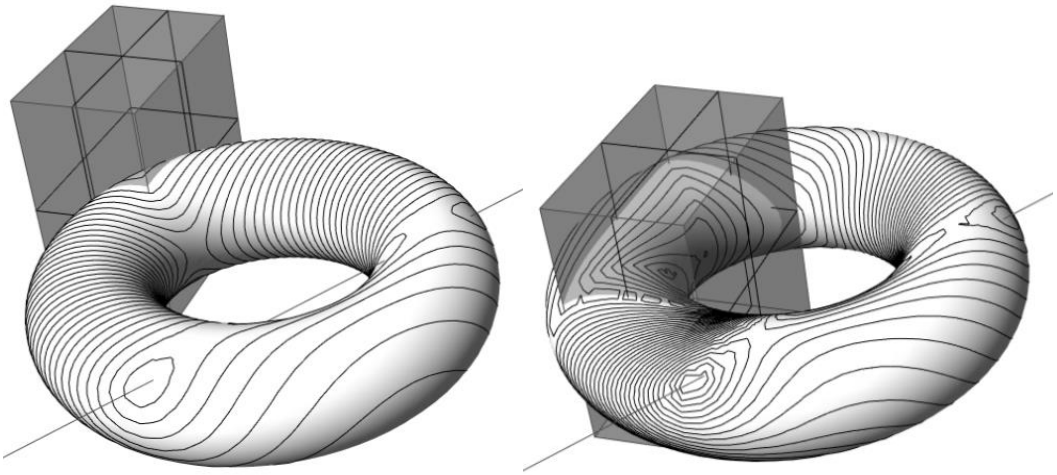
Generation of iso-surfaces defined in implicit form [F[x,y,z]=const].

uu. 014_object_contours.gh



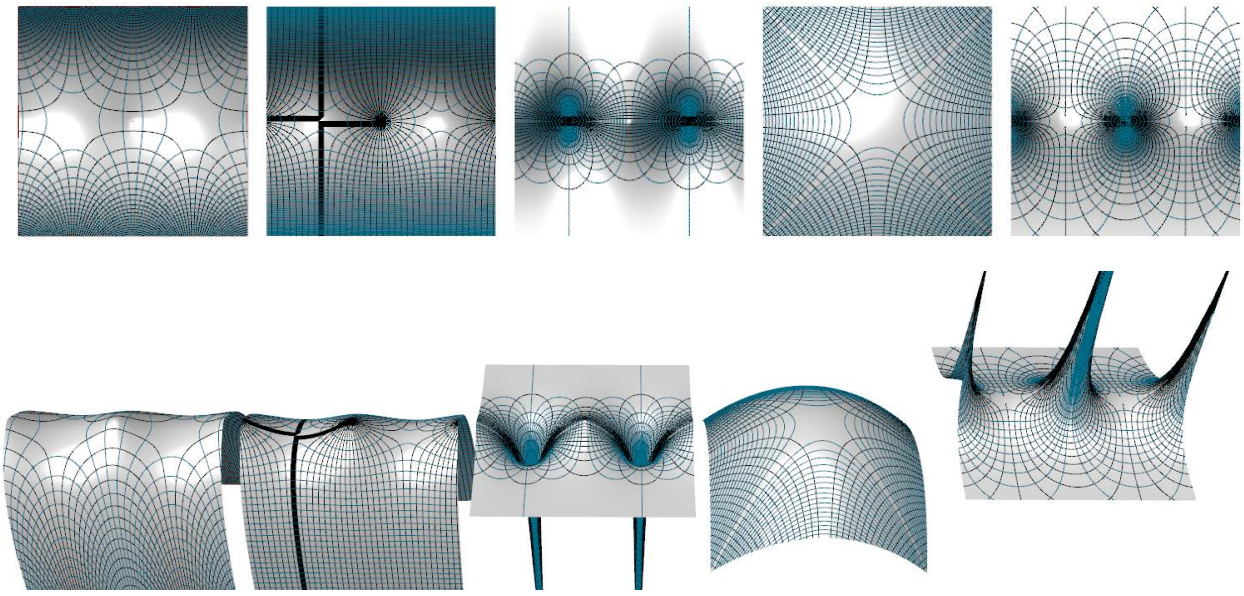
Using the object wrapper component you can generate meshes that smoothly wrap around different types of geometry.

vv. 019_contour_dist.gh



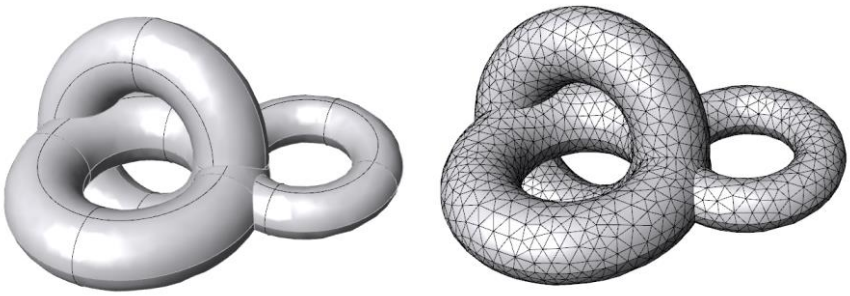
Use of the mesh contouring component for the generation of contours on a torus related to the distance from two objects [a box and a curve].

ww. ComplexContours.gh



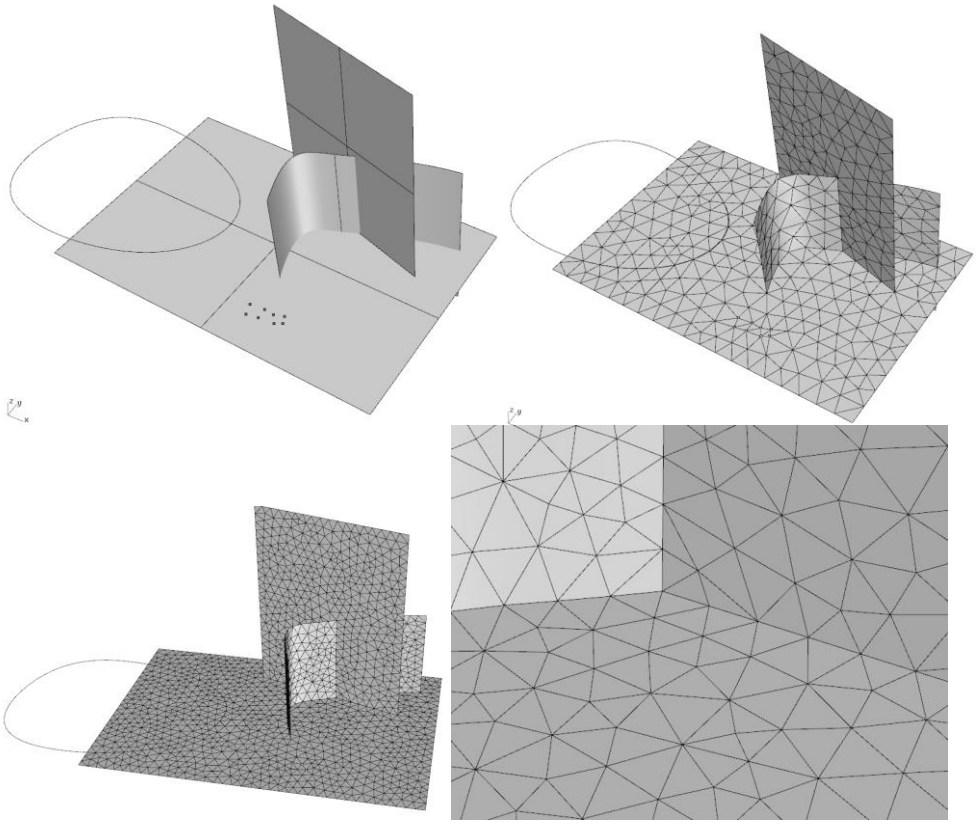
Demonstration of the use of the contour component for the generation of complex function visualizations.

xx. Discretization



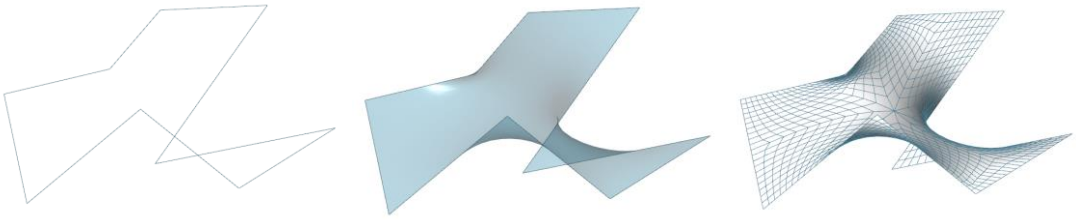
Brep triangulation using the Discretization component

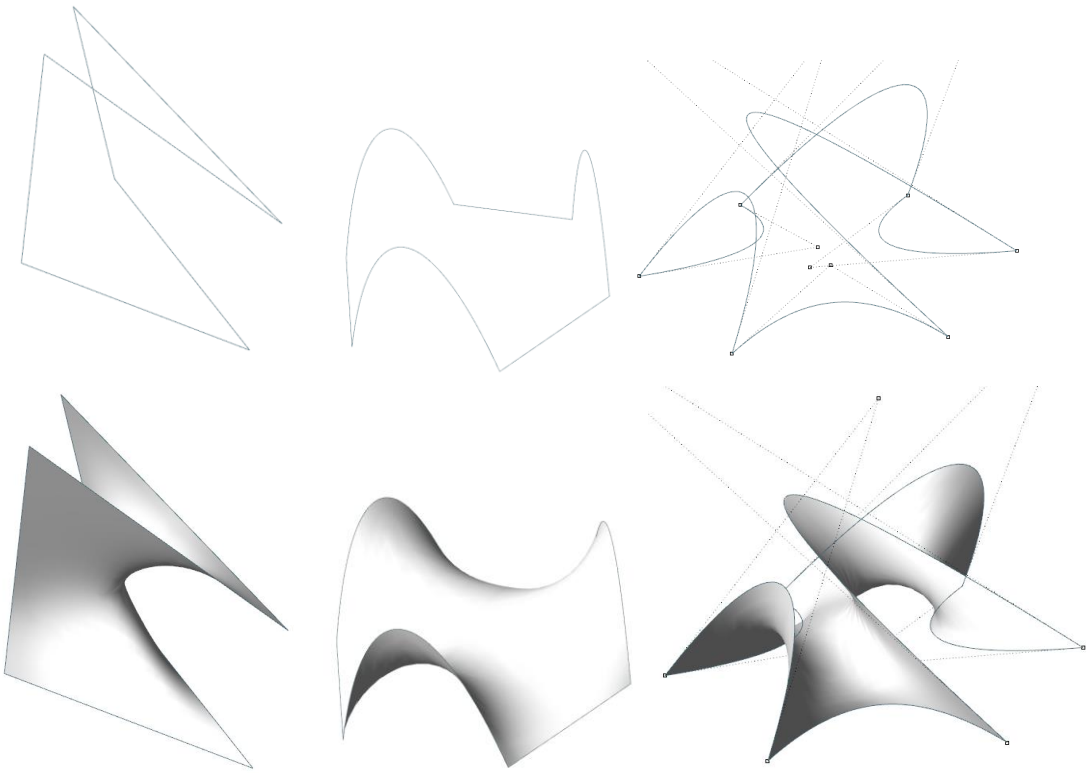
yy. DiscretizationB



Use of the Discretization component for the triangulation of a scene with non manifold connections. All surfaces, curves and points are intersected and taken into account during the triangulation ensuring common vertices along intersections for the resulting meshes. In this way you can create meshes for structural analysis with proper node connectivity.

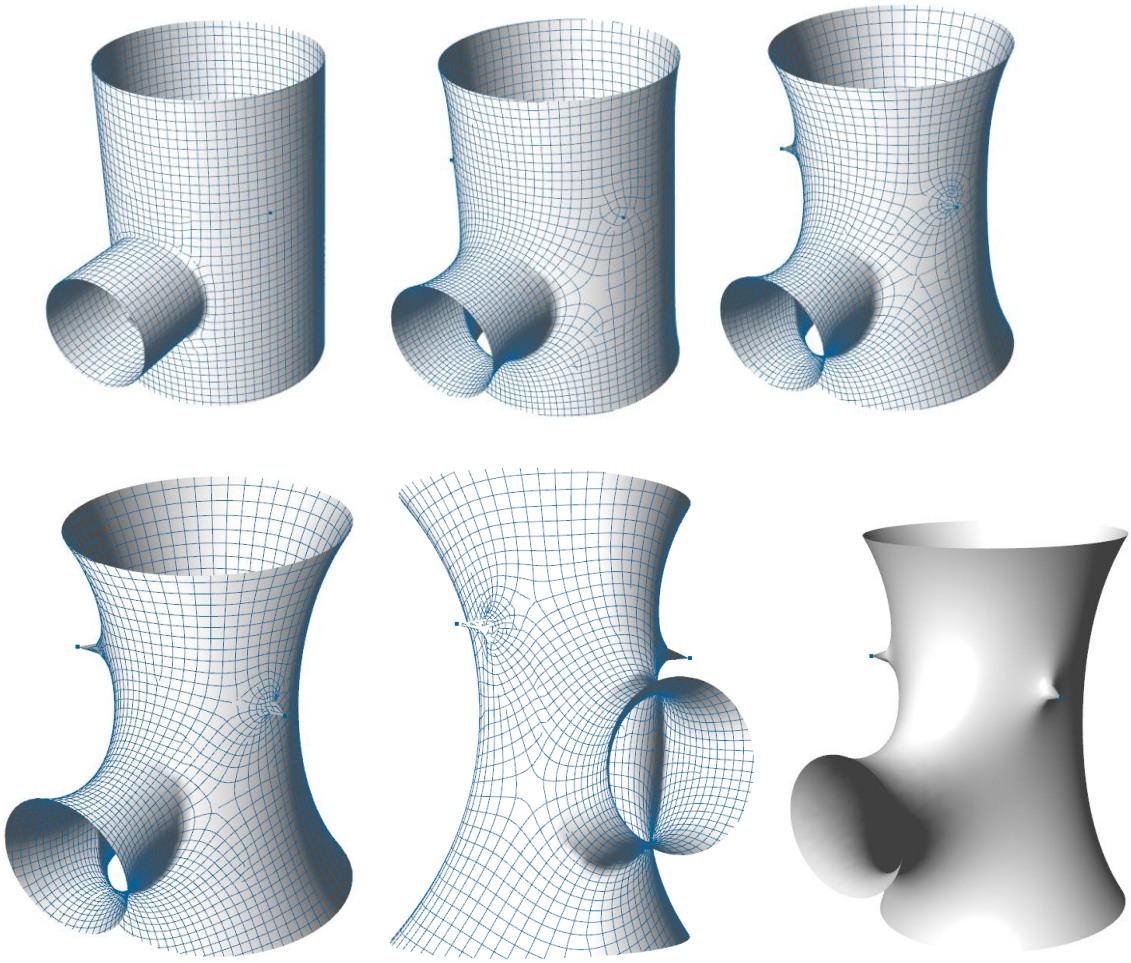
zz. MinimalSurfaceEdge





This component creates a minimal surface mesh spanning a loop made of any number of edges and of any geometry. It is useful when you want to fill in patches that have more than 4 boundary edges.

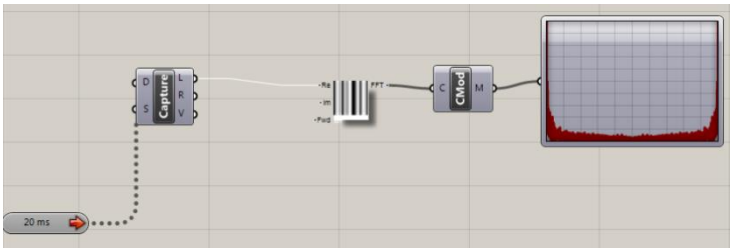
aaa. MinimalSurfaceFromMesh



the minimal surface mesh component iteratively applies mean curvature flow and relaxation on the input mesh while keeping the boundary and select vertices fixed, resulting in a minimal surface with the same topology as the given mesh.

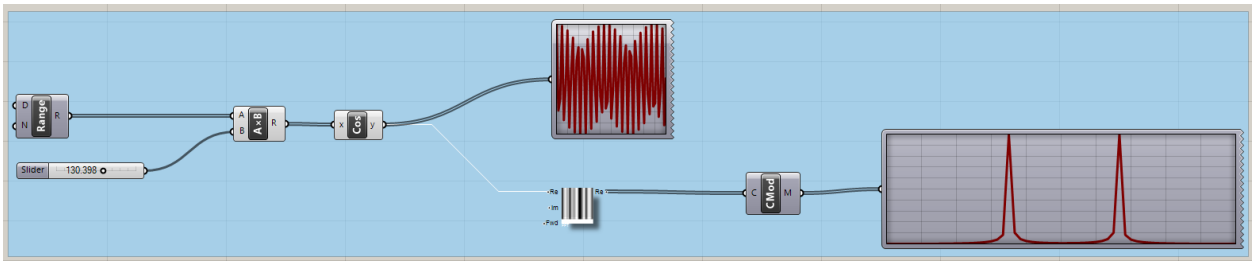
18. FFT

bbb. 016_FFTsound.gh



Use of the FFT component for the generation of the frequency spectrum of live sound [requires firefly].

ccc. 017_FFTSimple.gh



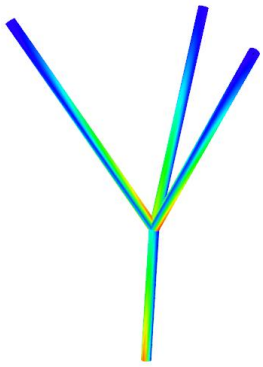
Use of FFT to analyse a sinusoidal wave form [notice the twin peaks at the frequency spectrum representing the pure tone]

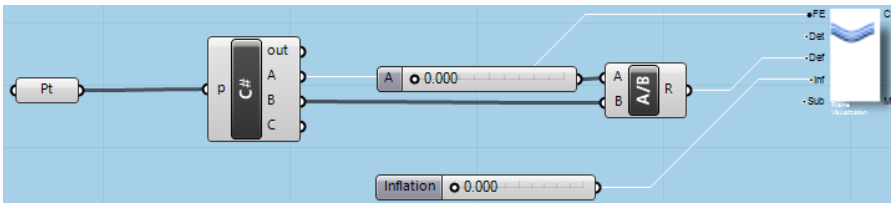
19. Programming with millipede

The following examples use millipede through C#.

Millipide is composed of several dll [dynamic Link Libraries] files. These files expose functions and objects that can be used in code to fine tune, create, analyze and optimize structural systems. You can reference these DLL file within grasshopper C# components or in standalone applications developed in visual studio. You may also use a mix of component based visual programming using the standard GH interface and C# coding where is needed, as the main millipede components output an RStatSystem object that contains all the information of the structure.

eee. 01_codingElements

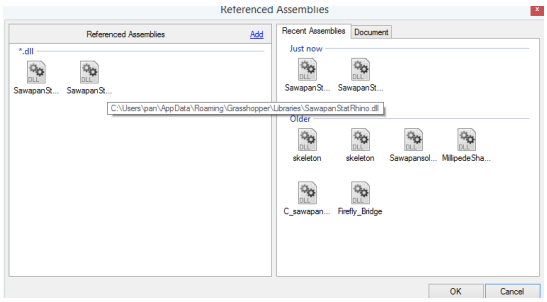




This example demonstrates how to build a structural model exclusively in code. Here is a step by step breakdown of the code and workflow:

1. Importing the assemblies

The first thing you need to do when accessing millipede through C# is to reference its DLL files so that your C# component can “see” the functionality inside millipede. To do so, right click on the C# component and choose “Manage Assemblies”:



In the window that opens select “Add” and browse to your GH components folder [where you installed millipede] and select SawapanStatica.dll and SawapanStatRhino.dll. Press “Ok”

After doing so, the C# component will be able to use all the functionality of millipede. In this example we are building a simple branching column with three branches. We take as input a single point that designates the root of the column and output the complete structural system object. We can then connect this output to the standard Millipede visualization components. In effect we collapsed all the tedious load , material, support and beams definitions to a few lines of code.

Here is the break-down of the code itself:

2. Create the system

First we create an RStatSystem object named rs. This object holds all the information regarding a structural system [members, supports, loads etc…] and functions that allow you to modify and interrogate it.

```
//create an FE system
RStatSystem rs = new RStatSystem();
```

3. Add Materials

We can add materials to the system as soon as we create it. You can add as many materials as you want and at any point of the program. A Material is added to the system using the AddMaterial function, passing the type of material and a name as a parameter. The name is arbitrary and is not really used internally.

```
//add a material to the system
StatMaterial m = rs.AddMaterial(MATERIALTYPES.STEEL, "steel1");
```

A StatMaterial object describes the numerical properties of a material. You can use any of the predefined types when adding a material [like MATERIALTYPES.STEEL etc…] but you can also create custom materials simply by changing its properties after you created it as a Generic type:

```
StatMaterial m2 = rs.AddMaterial(MATERIALTYPES.GENERIC, "m");
  m2.Em = 20000000.0;
  m2.Poisson = 0.2;
  m2.YieldStress = 8000000000.0;
  m2.Density = 100.0;
```

You can find these numbers for different materials from manufacturers and online databases.

Em: Modulus of elasticity [in N/m2]
Poisson: Poisson Ratio
YieldStress: Yield Stress [maximum stress allowed before the material goes into plastic permanent deformation] [in N/m2]

Density: Weight per cubic meter [Kg/m3]

Gm: Shear modulus [can be estimated by Em and Poisson if not given]

4. Define Cross-sections

When using frame elements we need to define cross-sections to assign to them. You can add as many cross sections as you want [even a different one for each frame element if you need to create a really fine-tuned system]. Here we are going to apply the same cross section to all elements. So we create a new Section using the material “m” we defined previously and set it to a circular hollow profile [there are other options that appear after you place the dot on s]

The initialization parameters for the Circular hollow profile are its inner and outer radius and the number of segments used for visualization. In general for large systems you should keep the number of segments low which will result in polygonal looking cross sections but will be in general faster.

```
//add a cross-section using material m
StatCrossSection s = rs.AddSection(m, "sec1");
//set the cros section shape to hollow circular, and set radii
s.CircHollow(0.03, 0.04, 16);
```

As in the case with material definitions, you can define arbitrary cross sections if you know their properties [area, Area moments of inertia etc…]. But for most practical purposes the provided cross section types should be sufficiently customizable.

5. Define some nodes

As we mentioned in class a finite element model is made of nodes will correspond to joints and hold information regarding position, forces, applied loads and supports. These nodes are subsequently connected by frame or surface elements that define the stiffness of the system.

Here we start by creating 5 points for our system and the corresponding nodes.

```
//create the 3 points of the system
Point3d p0 = p;
Point3d p1 = new Point3d(p.X, p.Y + 0.2, p.Z + 1.0);
Point3d p2 = new Point3d(p.X + 1.0, p.Y, p.Z + 2.5);
Point3d p3 = new Point3d(p.X - 1.0, p.Y, p.Z + 2.5);
Point3d p4 = new Point3d(p.X, p.Y + 1.0, p.Z + 2.5);

//create a node for each point [if two points coincide then you will get the same node for both of them]
RStatNode n0 = rs.AddNode(p);
RStatNode n1 = rs.AddNode(p1);
RStatNode n2 = rs.AddNode(p2);
RStatNode n3 = rs.AddNode(p3);
RStatNode n4 = rs.AddNode(p4);
```

6. Connecting the nodes

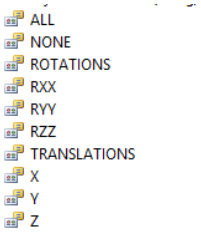
After we have defined some nodes we can connect them with various types of elements representing material objects. In our case we create 4 frame elements [referred to as beams here].

To add a frame element you call the AddBeam method of the RStatSystem object and pass the start and end nodes plus the cross section “s” defined above.

```
//add some frame elements connecting the previously created nodes
RStatBeam b0 = rs.AddBeam(n0, n1, s);
RStatBeam b1 = rs.AddBeam(n1, n2, s);
RStatBeam b2 = rs.AddBeam(n1, n3, s);
RStatBeam b3 = rs.AddBeam(n1, n4, s);
```

7. Adding Supports

Supports are defined as constrains applied to nodes. Here we just fix all translational and rotational degrees of freedom of the base node n0. You can change the support type at a node by assigning a different value to its SupportType member. The available support types are part of the BOUNDARYCONDITIONS enumerations. And include any combination of the following:



ALL: completely fixed support
NONE: completely free node
ROTATIONS: node can move but members around cannot rotate
RXX, RYY, RZZ: Node cannot rotate around specified axis [XX, YY or ZZ]
TRANSLATIONS: Pin Joint, node can rotate but not move
X, Y, Z: Node cannot move along specified axis.

You can combine more than one of these flags. For example to make a node that can slide along X but not Y or Z you need to combine the Y and Z supports as follows:

```
n0.SupportType = BOUNDARYCONDITIONS.Y | BOUNDARYCONDITIONS.Z;
```

Notice the use of the bitwise “OR” operator “|” that combines the two types of support.

In our example we simply fix all DOFs [degrees of freedom] for node 0:

```
n0.SupportType = BOUNDARYCONDITIONS.ALL;
```

8. Adding Loads

We can add applied loads to nodes by using their AddLoad method. The parameters of this method are the X,Y and Z components of the applied load in Newtons. Here we apply a load of -1000N equivalent to putting a 100Kg weight on its terminal branch point.

```
n2.AddLoad(0.0, 0.0, -1000.0);
n3.AddLoad(0.0, 0.0, -1000.0);
n4.AddLoad(0.0, 0.0, -1000.0);
```

In our example we also deactivate self-weight. The self-weight of the structure is the weight of its own components [beams, columns, slabs etc…]. The DeadLoadFactor is a property of the structural system and multiplies these weights before adding them as loads to the system. By setting this factor to 0 you practically instruct the analysis system to ignore its own weight. This is useful for certain optimization scenarios as well as when we want to look at different effects in isolation. In addition in some cases you might want to set this factor to more than 1.0 to exaggerate the effect of the self-weight on the structure as a safety factor.

```
//set the self weight factor to 0.0. Only applied loads will be taken into account
rs.DeadLoadFactor = 0.0;
```

9. Solve the system

Finally we are ready to actually analyze the system we built. By calling the SolveSystem method we instruct millipede to build the stiffness matrix and solve the large system of equations that determines the behavior of the structural system. This is the slowest step computationally. In general when you do iterative optimization this is the part where most of the time will be spent.

```
        //solve the system and recover displacements, forces etc....
        rs.SolveSystem();
```

10. Output something

Here we generate some output for the component. We can output the RStatSystem object itself which can become the input of the standard millipede components allowing you to mix code generated structures and UI manipualtions in the same workflow.

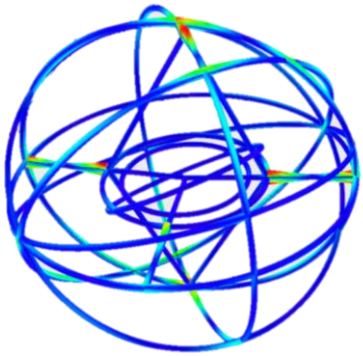
Here we also output the maximum deflection in the analyzed system plus the stress on the main vertical column element.

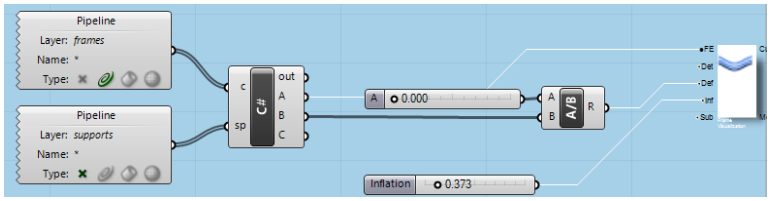
```
        //output the whole system [so that we can connect other millipede components
        A = rs;
        //output the Maximum deflection
        B = rs.MaximumDisplacement;
        //output the maximum normal stress along the first frame element
        C = b0.MaxStress;
```

11. Here is the complete code

```
private void RunScript(Point3d p, ref object A, ref object B, ref object C)
    {
        //create an FE system
        RStatSystem rs = new RStatSystem();

        //add a material to the system
        StatMaterial m = rs.AddMaterial(MATERIALTYPES.STEEL, "steel1");
        //add a cross-section using material m
        StatCrossSection s = rs.AddSection(m, "sec1");
        //set the cros section shape to hollow circular, and set radii
        s.CircHollow(0.03, 0.04, 16);

        //create the 3 points of the system
        Point3d p0 = p;
        Point3d p1 = new Point3d(p.X, p.Y + 0.2, p.Z + 1.0);
        Point3d p2 = new Point3d(p.X + 1.0, p.Y, p.Z + 2.5);
        Point3d p3 = new Point3d(p.X - 1.0, p.Y, p.Z + 2.5);
        Point3d p4 = new Point3d(p.X, p.Y + 1.0, p.Z + 2.5);

        //create a node for each point [if two points coincide then you will get the same node for both of them]
        RStatNode n0 = rs.AddNode(p);
        RStatNode n1 = rs.AddNode(p1);
        RStatNode n2 = rs.AddNode(p2);
        RStatNode n3 = rs.AddNode(p3);
        RStatNode n4 = rs.AddNode(p4);


        //add some frame elements connecting the previously created nodes
        RStatBeam b0 = rs.AddBeam(n0, n1, s);
        RStatBeam b1 = rs.AddBeam(n1, n2, s);
        RStatBeam b2 = rs.AddBeam(n1, n3, s);
        RStatBeam b3 = rs.AddBeam(n1, n4, s);

        n0.SupportType = BOUNDARYCONDITIONS.ALL;
        n2.AddLoad(0.0, 0.0, -1000.0);
        n3.AddLoad(0.0, 0.0, -1000.0);
        n4.AddLoad(0.0, 0.0, -1000.0);

        //set the self weight factor to 0.0. Only applied loads will be taken into account
        rs.DeadLoadFactor = 0.0;


        //solve the system and recover displacements, forces etc....
        rs.SolveSystem();


        //output the whole system [so that we can connect other millipede components
        A = rs;
        //output the Maximum deflection
        B = rs.MaximumDisplacement;
        //output the maximum normal stress along the first frame element
        C = b0.MaxStress;

    }
```

fff. 02_fromGeometry



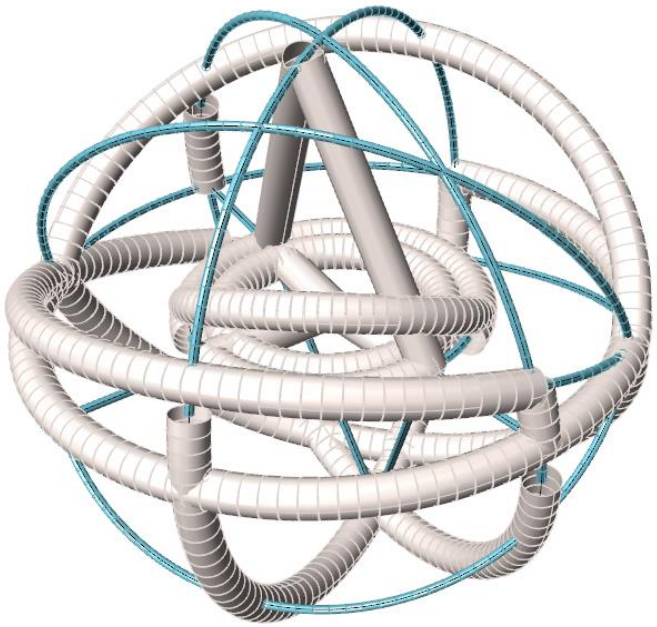


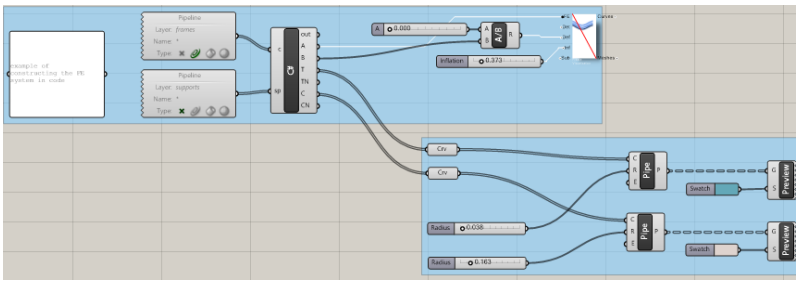
This example demonstrates how to use geometric input from rhino when building the FE model. It is nearly identical to the previous example but instead of explicitly defining the nodes and frame elements we use some handy utility functions that allow you to add whole lists of rhino curves as frame elements by assigning a cross section to them. The SplitAndAddCurvesAsBeams function will first intersect and split all curves against each other in order to guarantee the existence of node [joints] at intersection points. It will also break down curves elements into polylines made of linear finite elements. Therefore the beams in the structural system will not correspond to the input curves as there are more beams [linear elements] than curves in the end. However there are ways to refer back to the original curves if you need to modify the initial design as a response to analysis results.

The AddSupportAtPoints function will apply the requested support type to all nodes close to the points in the input list.

```
//create an FE system
            RStatSystem rs = new RStatSystem();

            //add a material to the system
            StatMaterial m = rs.AddMaterial(MATERIALTYPES.STEEL, "steel1");
            //add a cross-section using material m
            StatCrossSection s = rs.AddSection(m, "sec1");
            //set the cros section shape to hollow circular, and set radii
            s.CircHollow(0.03, 0.04, 16);


            rs.SplitAndAddCurvesAsBeams(c, s, 0.001, 0.1);

            rs.AddSupportAtPoints(sp, BOUNDARYCONDITIONS.ALL);



            rs.DeadLoadFactor = 1.0;


            //solve the system and recover displacements, forces etc....
            rs.SolveSystem();



            //output the whole system [so that we can connect other millipede components
            A = rs;
            //output the Maximum deflection
            B = rs.MaximumDisplacement;
```

ggg. 03_readingResults





This example demonstrates the extraction of results from the structural system after it has been analyzed. In this case we create two lists of lines, one that hold the geometry of all the frame element that are in tension and one that holds all the elements in compression. Visualizing these two groups of elements as members of different thickness provides a different reading of the differentiation of structural roles throughout the structure.

The code here is identical to the previous code up to the point of calling the SolveSystem method.

```
//create an FE system
RStatSystem rs = new RStatSystem();

//add a material to the system
StatMaterial m = rs.AddMaterial(MATERIALTYPES.STEEL, "steel1");
//add a cross-section using material m
StatCrossSection s = rs.AddSection(m, "sec1");
//set the cros section shape to hollow circular, and set radii
s.CircHollow(0.03, 0.04, 16);


rs.SplitAndAddCurvesAsBeams(c, s, 0.001, 0.1);

rs.AddSupportAtPoints(sp, BOUNDARYCONDITIONS.ALL);


rs.DeadLoadFactor = 1.0;

//solve the system and recover displacements, forces etc....
rs.SolveSystem();


//output the whole system [so that we can connect other millipede components
A = rs;
//output the Maximum deflection
B = rs.MaximumDisplacement;

List<Line> Tension = new List<Line>();
List<Line> Compression = new List<Line>();

List<double> TensionN = new List<double>();
List<double> CompressionN = new List<double>();

foreach (RStatBeam b in rs.Beams)
{
    if (b.ForceAtNode0.x < 0.0)
    {
        Tension.Add(new Line(b.P0.ToPoint3d(), b.P1.ToPoint3d()));
        TensionN.Add(b.ForceAtNode0.x);
    }
```

```
    else
    {
        Compression.Add(new Line(b.P0.ToPoint3d(), b.P1.ToPoint3d()));
        CompressionN.Add(b.ForceAtNode0.x);
    }
}

T = Tension;
TN = TensionN;
C = Compression;
CN = CompressionN;
```

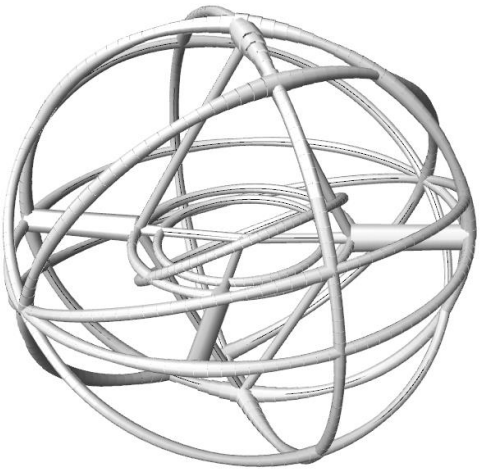
The final part of the code creates 2 lists of lines and 2 lists of numbers to hold the geometry of the elements and the corresponding axial force for each element separated in tension and compression elements.

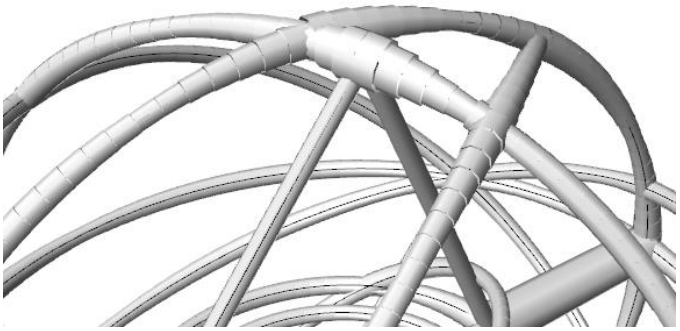
We loop through all the beam elements in the system and for each element b we check its axial force.

The internal forces of the element are computed at its endpoints and they correspond to the two vectors ForceAtNode0 and ForceAtNode1. So b.ForceAtNode0 is the internal force that emerges at the starting point of the frame element b. These forces are expressed in the local coordinate system of the beam with the X axis corresponding to the axis of the element. So the x component of the force is the axial and the y and z are the shear components.

The axial components of the start and end nodes of a frame element always have the same value but opposite signs. When the axial force of the first node is negative then we have tension [the force pulls the nodes apart] and when it is positive we have tension [the axial force pushes them together]

hhh. 04_reanalyize





In this example we analyze the structure twice. The first part is the usual analysis of the system, then we go through all the beams and we change the cross sections of each element depending on its stress. That creates a differentiated distribution of cross sections around the system stiffening areas of high stress. You can do all sorts of manipulation between analysis and reanalysis, even iteratively in order to optimize the distribution of material throughout the structure.

Here are the new steps added to the code:

Create a list of numbers to hold the stresses extracted from each beam element:

```
List<double> stresses = new List<double>();
foreach (RStatBeam b in rs.Beams)
{
    stresses.Add(b.MaxStress);
}
```

Sort the list so that we can find minimum , maximum and median stress[here we are only going to use the maximum]

```
stresses.Sort();
```

Extract the median and maximum stress from the list of stresses

```
double median = stresses[stresses.Count / 2];
double maximum = stresses[stresses.Count - 1];
```

Now step through each beam and create a new cross section for it whose radius is proportional to its stress. The beam with the highest stress will get a radius of 0.1m.

The member MaxStress for each beam is a number that represents an estimate of the maximum stress that appears somewhere along the beam elements. [For example for a horizontal beam in bending this maximum stress can be either at the endpoints or in the middle and will appear at the top or bottom of the cross section. In general stress varies from point to point throughout any element]

```
foreach (RStatBeam b in rs.Beams)
{
    double rad = 0.1 * b.MaxStress / maximum;
    StatCrossSection si = rs.AddSection(m, "sec2");
    si.CircHollow(rad, rad + 0.05, 9);

    b.CrossSection = si;
}
```

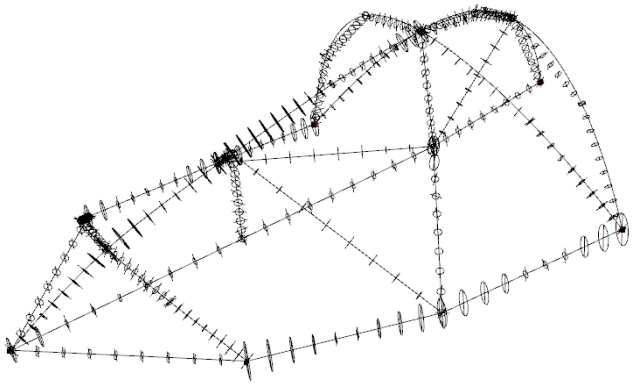
Reanalyze the system

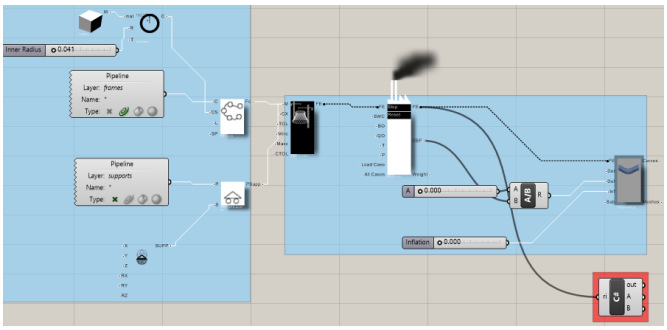
```
rs.SolveSystem();
```

1. Complete code

```
//create an FE system
RStatSystem rs = new RStatSystem();

//add a material to the system
StatMaterial m = rs.AddMaterial(MATERIALTYPES.STEEL, "steel1");
//add a cross-section using material m
StatCrossSection s = rs.AddSection(m, "sec1");
//set the cros section shape to hollow circular, and set radii
s.CircHollow(0.03, 0.04, 9);

rs.SplitAndAddCurvesAsBeams(c, s, 0.001, 0.1);
rs.AddSupportAtPoints(sp, BOUNDARYCONDITIONS.ALL);

rs.DeadLoadFactor = 1.0;

//solve the system and recover displacements, forces etc....
rs.SolveSystem();


List<double> stresses = new List<double>();
foreach (RStatBeam b in rs.Beams)
{
    stresses.Add(b.MaxStress);
}

stresses.Sort();

double median = stresses[stresses.Count / 2];
double maximum = stresses[stresses.Count - 1];

foreach (RStatBeam b in rs.Beams)
{
    double rad = 0.1 * b.MaxStress / maximum;
    StatCrossSection si = rs.AddSection(m, "sec2");
    si.CircHollow(rad, rad + 0.05, 9);

    b.CrossSection = si;
}

rs.SolveSystem();


//output the whole system [so that we can connect other millipede components
A = rs;
//output the Maximum deflection
B = rs.MaximumDisplacement;
```

iii. 05_detailedResults



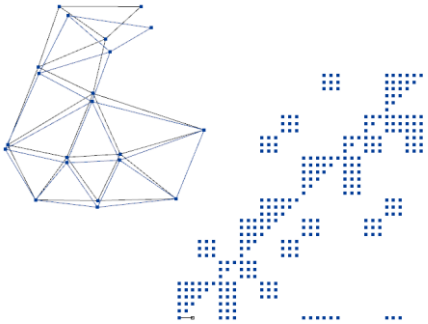


This example demonstrates two things. The first is the use of code to manipulate and extract results from a structural system that has been built using the standard UI of millipede. You just need to connect the FE output of any component to one of the inputs of your script and cast the incoming object to an RSTatSystem.

This is a slightly more advanced coding example where it steps along each member and computes the bending moment at each point in order to produce a variable orientation and thickness cross section.

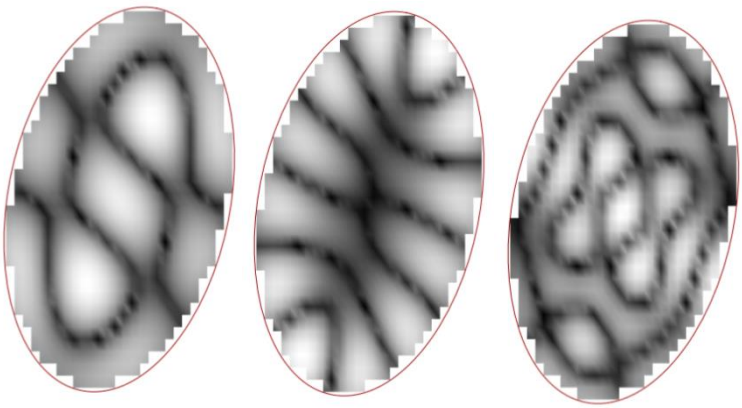
```
RStatSystem rs = ri as RStatSystem;

double maxmom = rs.Beams.Max(bx => bx.MaxBendingMoment);

StatBeamResults br = new StatBeamResults();

int num = 10;
List<Line> slines = new List<Line>();
List<Ellipse> el = new List<Ellipse>();

foreach (RStatBeam b0 in rs.Beams)
{
    for (int i = 0; i < num; ++i)
    {

        br.t = i / (double)(num - 1.0);
        br.yL = 0.0;
        br.zL = 0.0;

        b0.GetInterpolatedAxisResultsAt(rs.GravityDirection, rs.DeadLoadFactor, br);
        C_vector pt = b0.GetPointGlobal(br, 0.0);
        C_vector mo = new C_vector(0.0, -br.MomentL.z, br.MomentL.y);
        double mag = mo.Length;
        C_vector dm;
        b0.TransformVectorLtoW(mo, out dm);

        dm.Normalize();

        C_vector dz = C_vector.CrossProduct(dm, b0.Csys.Ex);

        Plane pl = new Plane(pt.ToPoint3d(), dm.ToVector3d(), dz.ToVector3d());

        double sz = 0.05 + 0.1 * mag / maxmom;

        C_vector pp0 = pt - dm * sz;
        C_vector pp1 = pt + dm * sz;

        if (mag > maxmom * 0.25)
            el.Add(new Ellipse(pl, sz, sz * 0.5));
        else
            el.Add(new Ellipse(pl, 0.03, 0.03));

        slines.Add(new Line(pp0.ToPoint3d(), pp1.ToPoint3d()));
    }
}

A = slines;
B = el;
```

jjj. 015_SparseSolver.gh



Use of the sparse linear system solver to simulate a basic truss [building the stiffness matrix etc…]. this scripting example requires a fundamental understanding of basic Finite element analysis.

kkk. 018_eigenfunctions.gh



Eigenfunctions of the graph Laplacian of a mesh. This scripting examples demonstrates how to manually build the discrete Laplacian matrix for a mesh [disregarding distortions and anisotropy which would require the more involved discrete Laplace Beltrami operator [see EigenModes component]]

δ. Disclaimer

Panagiotis Michalatos and Sawako Kaijima may terminate this Agreement immediately if Licensee breaches any provision. Upon notice of termination by Panagiotis Michalatos and Sawako Kaijima, all rights granted to Licensee under this Agreement will immediately terminate, and Licensee shall cease using the SOFTWARE and return or destroy all copies (and partial copies) of the SOFTWARE and documentation.

Disclaimer of warranty.

Licensee acknowledges that the Software is experimental. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS, Panagiotis Michalatos and Sawako Kaijima, AND CONTRIBUTORS [AS IS] AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Technical Support.

Panagiotis Michalatos and Sawako Kaijima have no obligation to furnish You with technical support.

No Other Obligations.

This Agreement creates no obligations on the part of Panagiotis Michalatos and Sawako Kaijima other than as specifically set forth herein.