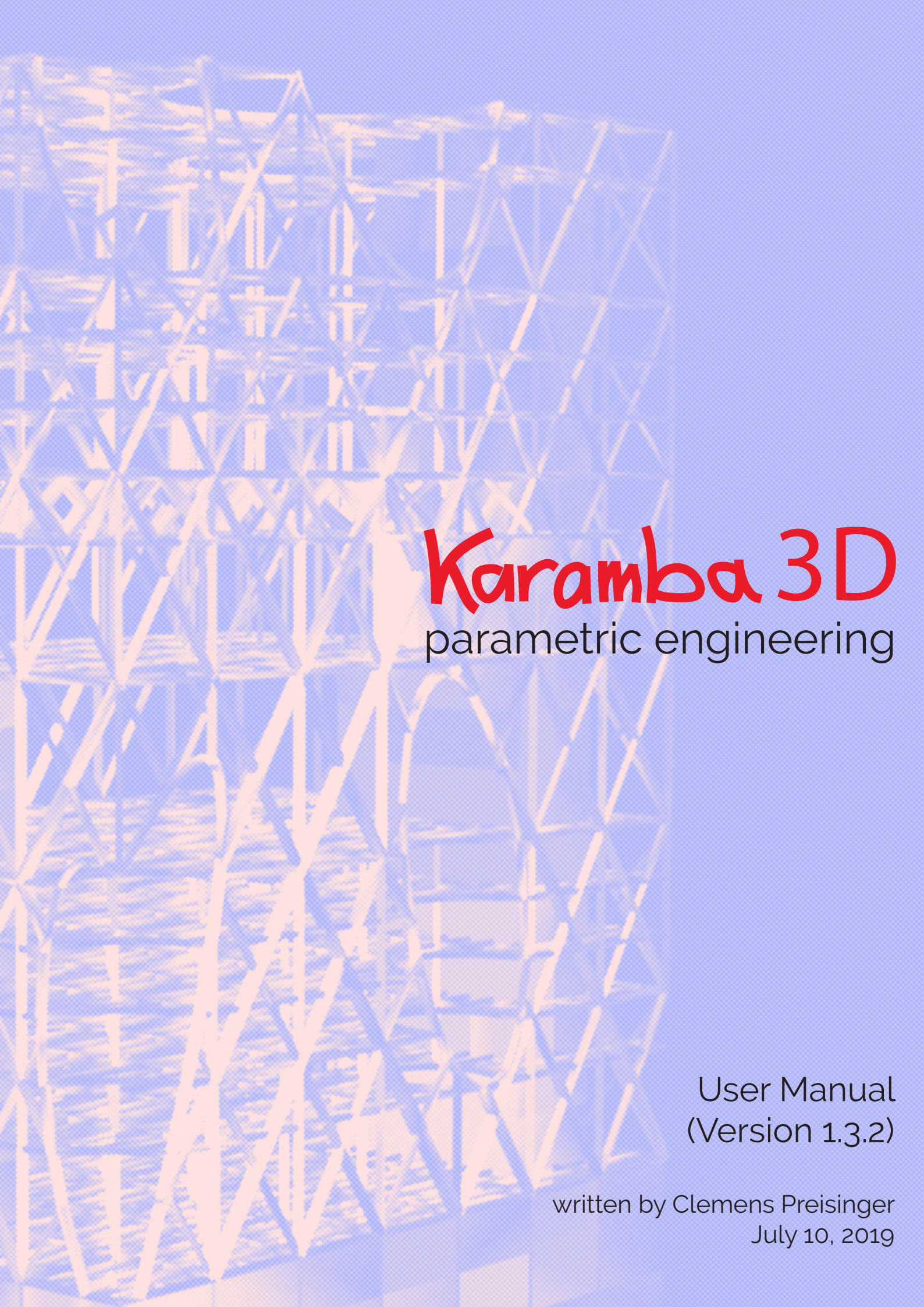

Karamba 3D

parametric engineering

User Manual
(Version 1.3.2)

written by Clemens Preisinger
July 10, 2019

Contents

Chapter 1

Introduction

Karamba3D is a Finite Element program like many others. However it has advantages over these programs in several important respects: It is easy to use for non-experts, has been tailored to the needs of architects and engineers in the early design phase, works interactively and costs slightly less than the rest.
Karamba3D is fully embedded in the parametric environment of Grasshopper which is a plug-in for the 3d modeling tool Rhinoceros. This makes it easy to combine parameterized geometric models, finite element calculations and optimization algorithms like Octopus or Galapagos.
Besides the free and full version for non-commercial use only, there exists also a pro-version of Karamba3D for commercial use. Table 1.1 lists their main features. Those parts of this manual that apply to the full-version only, are either blue or have blue section headings.

Table 1.1 Variants of Karamba3D

Karamba3D version	beam elements	shell elements	other features
free	unlimited	unlimited	limited
full	≤ 20	≤ 50	unlimited
pro	unlimited	unlimited	unlimited
pro-student	unlimited	unlimited	unlimited

1.1 Citing Karamba3D

In case you use Karamba3D for your scientific work, please cite the following paper:

Preisinger, C. (2013), *Linking Structure and Parametric Geometry*. Archit Design, 83: 110-113. doi: 10.1002/ad.1564.

1.2 Disclaimer

Although being tested thoroughly Karamba3D probably contains errors – therefore no guarantee can be given that Karamba3D computes correct results. Use of Karamba3D is entirely at your own risk. Please read the license agreement that comes with Karamba3D in case of further questions.

Chapter 2

Getting started

If all goes well during installation you will notice upon starting Grasshopper (GH) that there is a new category called Karamba3D on the component panel. It consists of roughly ten subsections (see figure 2.1). In case you do not see any icons select “Draw All Components” in Grasshoppers “View”-menu. The installation can be tested by placing a Karamba3D “License”-component on the canvas: it should not issue a warning or error. If not, see section 4.3.1 for how to solve that issue. On Apple machines make sure to have Microsofts .NET Framework 4.0 installed in case of Rhino5 and version 4.5 in case of Rhino6.

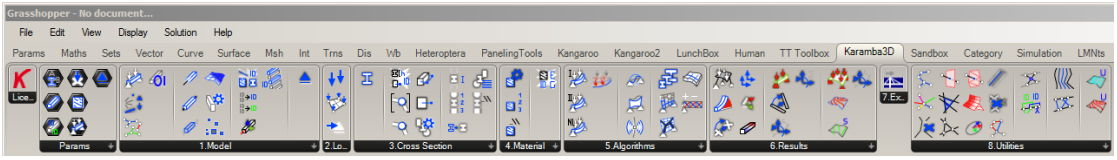

Figure 2.1 Category “Karamba3D” on the component panel

These are the subsections which show up in the Karamba3D category:

- “License”: The “License”-component contained in here delivers information regarding the current type of license and how to get a pro-version of Karamba3D (see fig. **??**).
- “Params”: containers for Karamba3D objects like beams, loads, models, . . .
- “1.Model”: lets you create a basic models with default settings for cross sections and materials
- “2.Load”: components for defining external forces
- “3.Cross Section”: contains components to create and select cross sections for elements.
- “4.Materials”: components for the definition and selection of materials
- “5.Algorithms”: components for analyzing the structural model
- “6.Results”: for the retrieval of calculation results
- “7.Export”: for exporting Karamba3D-models to RStab or Robot via DStV-file.

- "8.Utilities": contains some extra geometric functionality that makes it easier to handle and optimize models.

The colors of Karamba3D's icons have a special meaning: black or white designates the entity or entities on which a component acts. Products of components get referenced by a blue symbol.

2.1 Karamba3D Entities

Grasshopper (GH) is an object oriented, visual scripting environment. It provides items like points, curves, surfaces, ... for geometric computing. The full range of geometric items can be inspected in the subcategory "Geometry" of the toolbar section "Params". Karamba3D adds seven entities for building structural models:

- "Model": contains all the information related to a structure.
- "Element": can be a beam, truss, shell or springs.
- "Element Set": groups together elements in a given order, makes them accessible under a common name.
- "Joint": defines the connectivity between neighboring elements.
- "Load": external action which is imposed on the structure.
- "Cross-section": defines a structural element's geometry in section.
- "Material": provides information regarding the physical behavior of what a cross section is made of.
- "Support": defines how a structure connects to the ground.

Karamba3D objects behave like GH entities.

- They can be stored in containers (see the "Params" subcategory of the "Karamba3D" toolbar).
- When converted into text by plugging them into a panel they provide textual output regarding their fundamental properties.

In order to build a structural model not all of the above entities need to be present. Karamba3D assumes default settings for materials and cross sections:

- If no material is given, Karamba3D chooses steel (S235 according to EC3 with $f_{yk} = 23.5\,\mathrm{kN/cm^2}$) for all cross sections.
- For beams the default is a circular hollow cross sections (CHS) with an outer diameter of $114.4\,\mathrm{mm}$ and a wall thickness of $4\,\mathrm{mm}$. The default thickness of shells amounts to $10\,\mathrm{mm}$.

2.2 Setting up a Structural Analysis

2.2.1 Define the Model Elements

Straight lines form the basis of beam, truss and spring elements. Shells and slabs are based on GH meshes. Fig. 2.2 shows a definition which defines a single beam, assembles a model and displays it. The "LineToBeam"-component takes a "Line"-object as input and creates a beam element from it. Karamba3D assumes all geometry input to be in m. Assigning names to elements can be a great help in case of working with large, complex structures. In fig. 2.2 the name "bart" is assigned to the new beam. Element identifiers need not be unique. This allows to use them for grouping elements. The "Assemble"-component gathers the information in a model. When plugged into a panel, a model displays its basic features: "c.Length" stands for characteristic length and is calculated as the diagonal of the bounding box of the structure. In case of the beam there are two nodes which define one element. In the absence of material definitions Karamba3D automatically generates one default material. This is applied to the default beam cross section. In case of the presence of shells a second default cross section would show up. The model contains no loads, there is however a default load-case. A load-case corresponds to a scenario where a group of external actions impact a model. Think of e.g. wind which can hit a structure from several directions but not from all directions at one. Each wind direction would correspond to a load-case.

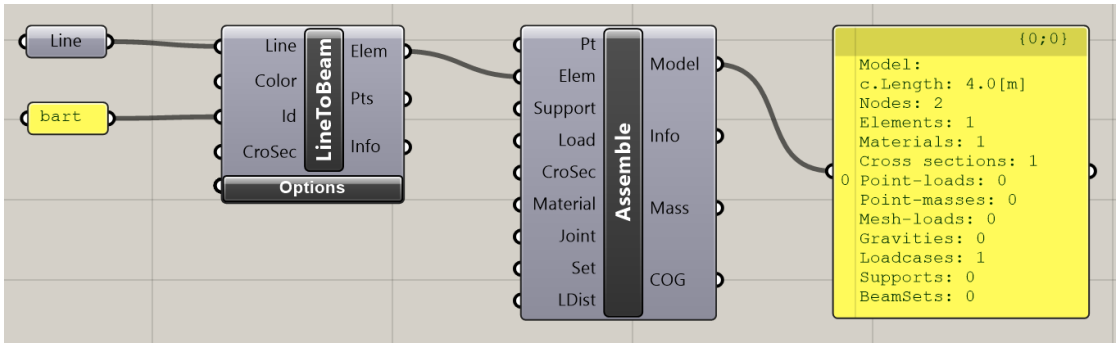

Figure 2.2 A structural model with geometry only.

2.2.2 View the Model

There are three components for controlling how a model is displayed in the Rhino viewport:

1. "ModelView": sets the general display properties. These are stored in the model and stay valid until overwritten by a downstream "ModelView"-component.
2. "BeamView": lets you control the display properties specific for beams. Renders e.g. the cross section as a mesh.
3. "ShellView": Contains the display-settings for shells.

Fig. 2.3 shows hot the "ModelView"- and "BeamView"-component team up for rendering the beam-model. In order to unfold the viewing-components click on the black section headings on the compo-

nents. In the “Tags” section of “ModelView” the “Elements” checkbox is on by default. This enables the display of the elements middle axis. When activated, the “Nodetags”-option makes the node numbers show up. The nodes in a Karamba3D model are numbered starting with zero. The same applies to model elements. Showing their identifiers via “Element Ids” is sometimes more useful.
In case of more than one element the “Assemble”-component rigidly connects them if their nodes coincide. If working with imprecise geometry this can give unexpected results: Although two elements may appear connected, there could ne in fact inaccuracies in the node positions. A gap in a model usually has a large impact on its physical behavior. Enabling the display of node indexes can help to find such gaps, since there will be two node numbers in one place.

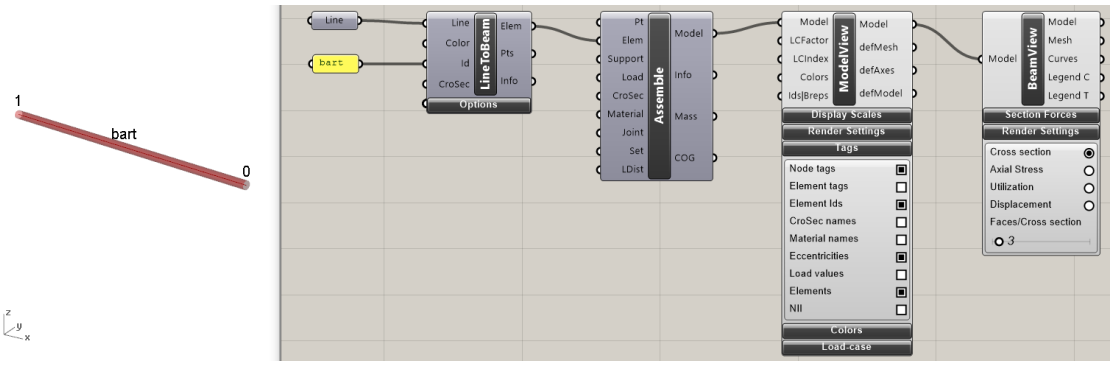

Figure 2.3 “ModelView”- and “BeamView”-components are used to display a model.

2.2.3 Add Supports

Supports define how a structure connects to the ground. They suppress translations or rotations at nodes. An activated button appears black and means either zero translation (T) in the direction of the global x-, y- or z-axis or zero rotation (R) about the corresponding global axis (or local axis if you input a plane). A node index or position can be used to specify the location of a support. Supply a plane as input for specifying locally oriented support conditions.
Green arrows symbolize translational supports, purple circles stand in for rotational supports (see fig 2.4).

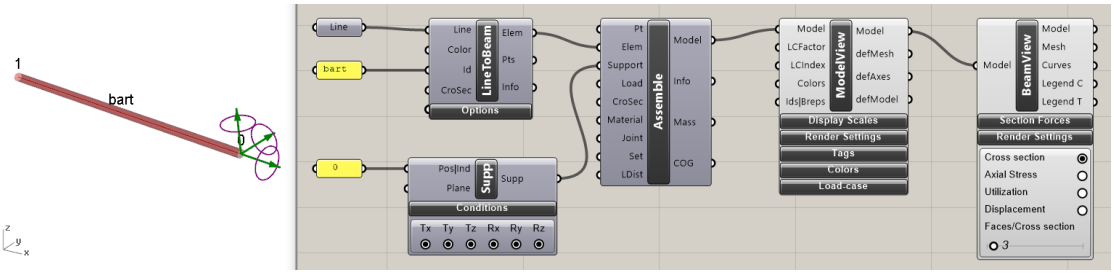

Figure 2.4 Supports specify how a structure interacts with the ground.

For static analysis, a structure needs to be supported in such a way that it can not fly around freely. In three dimensional space, a rigid body has six modes of movement or degrees of freedom (DOFs): three translations and three rotations (see fig. 2.5). Thus there need to be at least six support conditions in a structural model to fix it. When a model or parts of it are moveable, the "Analyze"-component either refuses to calculate or returns huge displacements. Should you encounter a problem like this plug your model into the "Eigenmodes"-component. It can detect the rigid body movements which cause the problem.

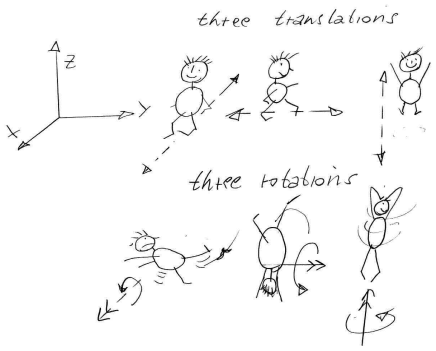

Figure 2.5 A body in space has six degrees of freedom (DOFs).

2.2.4 Define Loads

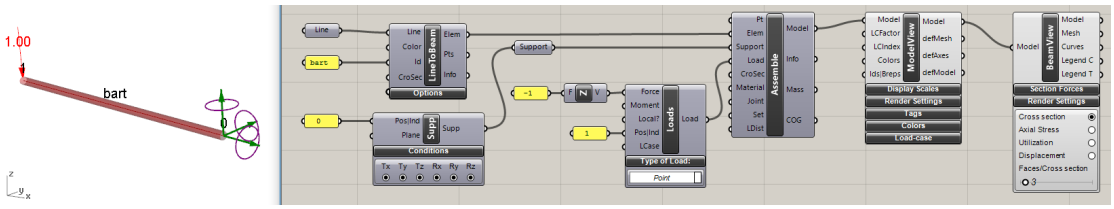

Figure 2.6 A cantilever with a point-load at its tip.

In Fig. 2.6 a point-load of 1kN is added at the tip of the cantilever beam. A Vector at the input-plug "force" specifies direction and magnitude of the load: since the global Z-axis points upwards a load acting downwards has a negative z-component.
The input-plug "LCase" can be used to set the number of the load case in which the load acts. This allows different load scenarios (e.g. wind from different directions) to be created.
The dropdown list at the bottom of the "Loads"-component lets one choose between different types of loads as shown in fig. 2.7. Gravity loads (1) act on the whole structure. The location of point loads (2) can be specified by node index or position. Beam loads (3) act on elements given by element identifiers. Distributed loads on arbitrary meshes (4) get reduced to approximately statically equivalent node and beam loads.

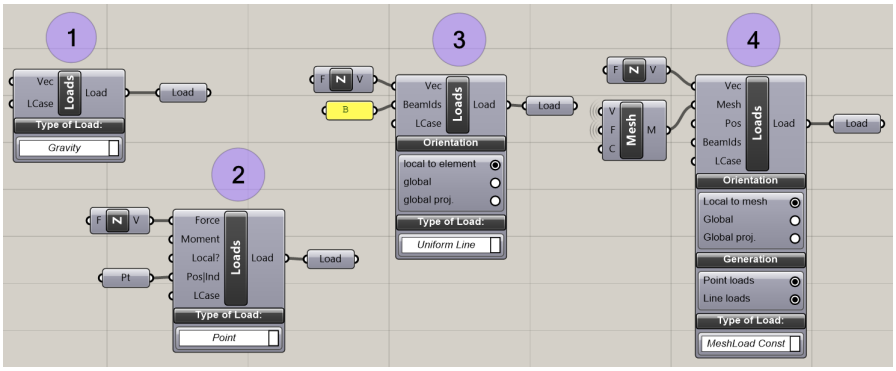

Figure 2.7 definitions of gravity load (1), point load (2), uniformly distributed load on a beam (3) and distributed load on a mesh (4)

The directions of gravity and point-loads refer to the global coordinate system. The direction vector of beam- and mesh-loads can be specified relative to the global or local (relating to the element or mesh) coordinate system.

2.2.5 Choose an Algorithm

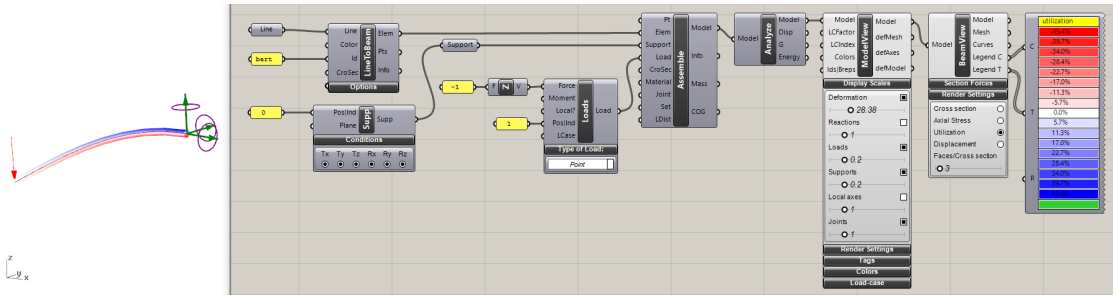

Figure 2.8 Deflection and stress-wise utilization of a cantilever beam with a point-load at its tip.

Karamba3D offers various options of analyzing a structural model. The “Analyze” component (see fig. 2.8) calculates the deformation and stresses of a model under external loads. The “Deformation”-slider in the submenu “Display Scales” of the “ModelView”-component allows you to scale the graphical output of the displacements. The default magnification factor is 50. In case the numeric range of the “Deformation”-slider does not fit it can be adapted. A double-click on the knob of the slider invokes a window for adapting the slider settings.
In order to get the numbers which correspond to the colors of the utilization output a “Legend”-component is used in fig 2.8. Dividing the normal stress in a point of the cantilever by the strength of its material gives the stress-wise utilization output of the “BeamView”-component. Negative values correspond to compression, positive values to tension. Stress-wise utilization can be misleading: Slender beams under axial compression buckle and thus collapse before the compressive stresses

reach the material strength. The “Utilization”-component includes stability and should be used in such cases.

2.2.6 Provide Cross Sections – or Use the Default

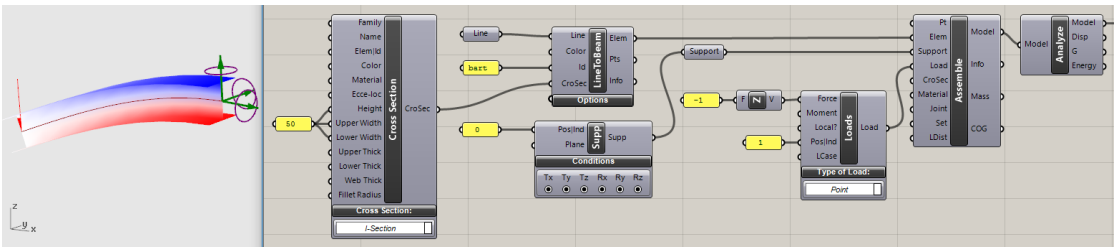

Figure 2.9 Definition of a cross section.

Fig. 2.9 shows how to attach a custom cross section to an element. It is an I-profile with a height and width of 50cm. The physical unit of any input- or output-value is mentioned in the help text which pops up when the mouse pointer hovers over the corresponding plug.
There are to options for attaching cross sections to elements:

- Directly at the component where the element is created as shown in fig. 2.9.
- Via element names (“B” and “S” in fig. 2.10) or regular expression by plugging cross sections into the “Assemble”-component.

Direct assignment overrides definition via the “Assemble”-component.

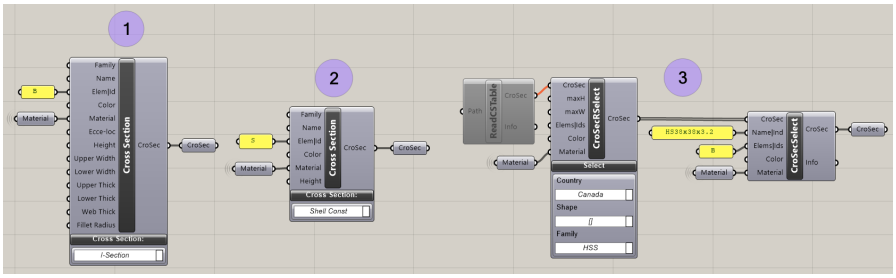

Figure 2.10 left: definition of a beam cross section (1); Middle: definition of a shell cross section (2); Right: selection of a cross section from the default cross section library. (3)

Arbitrary I-, hollow box, filled trapezoid and hollow circular cross sections can be defined for beams. Alternatively Karamba3D lets the user select a predefined standard cross section. In case of shells, it is possible to attach a different cross sections to each element.
The Karamba3D cross sections are available as multi-components: they can be accessed via the single component “Cross Sections”. The drop down menu lets one select the cross section type.

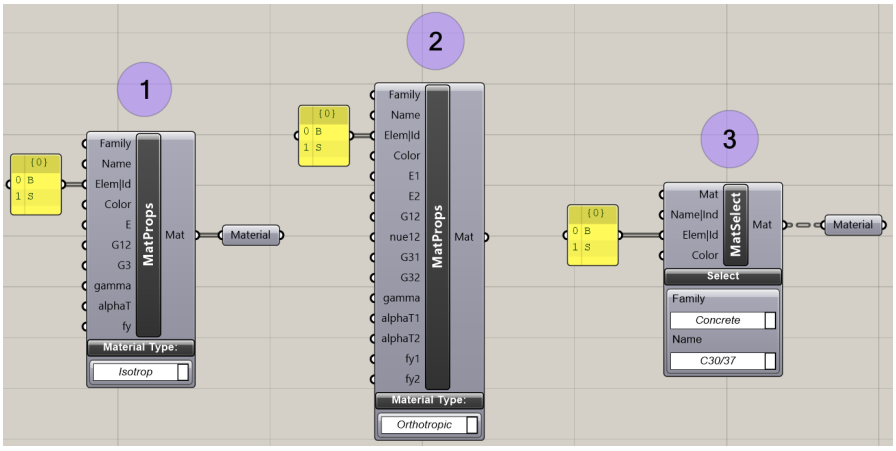

Figure 2.11 left: definition of an isotropic custom material (1). Middle: definition of an orthotropic custom material (2). Right: selection of a material from the material library (3)

2.2.7 Specify Materials – or Use the Default

Materials can be either defined by manually setting their mechanic properties or by selection from a library of predefined materials (see fig. 2.11). Materials attach to cross sections. There are two options for assigning materials:

- Direct input at the “Cross Section”-component like in fig. 2.12.
- Assignment via the “Assemble”-component. The “Elems|Id” input-plug specifies the names of the elements to which the material shall be attached. Alternatively a regular expression can be used to select elements. Leaving “Elems|Id” empty sets the material for all elements. Materials are not attached to elements directly but to the element’s cross section. In order to avoid side effects on other elements, cross sections get copied before a new material is assigned.

A material definition via the cross section overrides assignment via “Assemble”-component. Fig. 2.11 shows

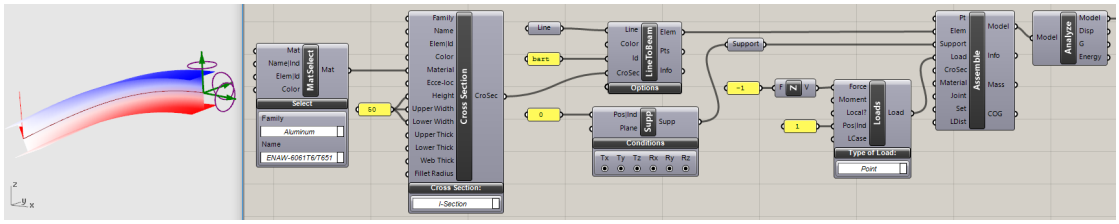

Figure 2.12 Definition of a material.

2.2.8 Retrieve Results

Visualization

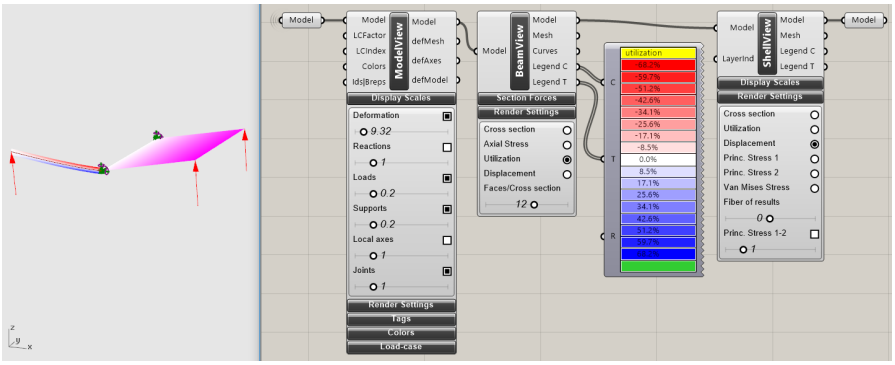

Figure 2.13 There are three components for visualizing the model: “ModelView”, “BeamView” and “ShellView”

Karamba3D offers three components for visualizing a structural model (see fig. 2.13):

1. “ModelView”: Sets the basic visualization properties like scaling factor of displacements, sizes of symbols, number of displayed load case, . . .
2. “BeamView”: visualizes beams
3. “ShellView”: visualizes shells

Each of these components contains submenus which can be expanded by clicking on the black caption bar. The numerical range of sliders can be set by double-clicking on their black knob. Visualization properties stick to the model and stay valid until they get overruled by another downstream visualization component.

Results

Structural response properties can be used to inform the model and e.g. optimize it. Fig. 2.14 shows some of the available options.

Disassembling a Structural Model

In order to modify a model or retrieve e.g. cross section types after cross section optimization, Karamba3D lets you disassemble models, elements, cross sections and materials. The components “Disassemble Model”, “Disassemble Element”, “Disassemble Cross Section” and “Disassemble Material” let you dissect a structural model in great detail.

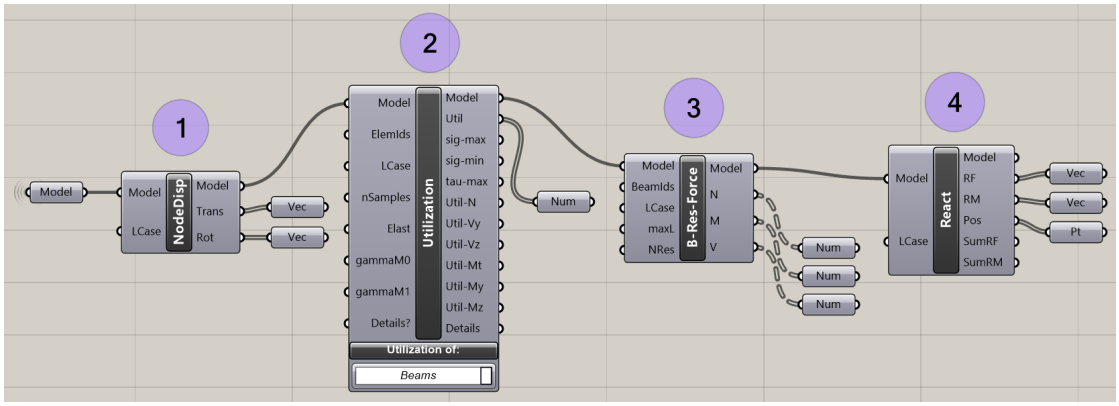

Figure 2.14 Retrieval of numerical results: nodal displacements (1), level of material utilization (2), resultant cross section forces (3) and reaction forces (4).

2.3 Physical Units

On installing Karamba3D one can specify the family of physical units to be used for input and results. The default option is metric (e.g. meters, centimeters, degree Celsius, Newtons, . . .) but Karamba3D can also deal with Imperial units (e.g. feet, inch, degree Fahrenheit, kiloponds, . . .).
The set of units to be used can be changed any time by editing the "karamba.ini" file.
Depending on the family of units Karamba3D interprets geometric input either as meters or feet. The kind of physical units that components expect to receive shows up in the tool-tip which appears when the mouse pointer hovers over an input-plug.
Changing the type of physical units during the creation of a GH definition may give rise to problems: The help text of Grasshopper components does not change dynamically. Switching from SI to Imperial Units leaves the help text of those components already placed on the canvas unaltered. The interpretation of the input values however changes. Opening a GH definition with a Karamba3D versions with differently set physical units entails the same problem.
Karamba3D comes with databases for predefined cross sections and materials. The properties there are given in SI units. The same applies to physical constants (e.g. "gravity") defined in the "karamba.ini"-file.
Throughout the rest of this manual SI units will be used exclusively in order to ensure good readability. When specific differences exist between using Karamba3D with Imperial Units and SI units, this will be mentioned in the text.

2.4 Quick Component Reference

2.4.1 License

License: Returns the program version, license information and can be used to manage the license file.

2.4.2 Params

Karamba3D introduces seven new classes for defining structural models and corresponding containers:


Cross-section: Container for cross section objects


Element: Container for finite elements


Element Set: Container for ordered groups of elements


Joint: Container for connectivity conditions between elements


Load: Container for load objects


Material: Container for materials


Model: Container for models


Support: Container for supports

2.4.3 Model

This subcategory contains components for assembling a model, converting geometry into finite elements and defining support conditions.


Assemble Model: Creates a finite element model by collecting given entities (points, beams, shells, supports, loads, cross sections, materials, . . .).


Disassemble Model: Decomposes a model into its components.


Modify Model: Changes the nodal positions.


Connected Parts: Returns groups of interconnected lines of the model.


Activate Element: Activates the elements of a model according to the activation list. Uses the soft kill approach for inactive elements.


Line to Beam: Creates beams with default properties from given lines. Lines that meet at a common point are by default rigidly connected with each other. Karamba assumes input to be in meter or feet.


Connectivity to Beam: Creates beams with default properties from given connectivity diagram.


Index to Beam: Creates beams with default properties from given node indexes.


Mesh to Shell: Creates shells with default properties from given meshes. Quad faces are split to triangles.

Modify Element: Multi-component for modifying elements. Works either directly on an element or indirectly as an autonomous agent:

-  **Modify Beam (default):** Modifies beams only.
-  **Modify Shell:** Modifies shells only.

Point-Mass: Attaches a point mass to a node of given index or position. Does not result in additional weight, only translational inertia.

Disassemble Element: Decomposes elements into their components.

Make Beam-Set: Puts beams designated by their beam identifier into a group.

Orientate Beam: Sets the local Z-axis of beams according to a given vector and adds a rotation angle DAlpha $[deg]$ about the longitudinal axis. Flips beam direction according to a given x-vector.

Select Beam: Selects beams according to a given identifier and puts all incoming beams in two groups: selected or rejected. The identifier may be the element index, name or a regular expression.

Support: Creates supports at nodes of given node-indexes or node-coordinates. Lets you select translations/rotations which should be zero and the support orientation with respect to the global coordinate system.

2.4.4 Load

The component in this subcategory let one define and manipulate external actions which impact a structure.

Loads: Multi-component for defining loads:

-  **Gravity (default):** Creates gravity from a specified direction vector for given load-cases.
-  **Point-Load:** Creates point loads at points of given index or position.
-  **Imperfection-Load:** Defines imperfections for beams under normal forces N^{II}.
-  **Initial Strain-Load:** Sets initial axial strains on beams.
-  **Temperature-Load:** Imposes a temperature difference on an element with respect to its initial temperature at construction.
-  **Line-Load on Element:** Creates a uniformly distributed load on a beam.

-  **MeshLoad Const:** Creates approximately equivalent point- and line-loads from a const surface load on a mesh. The constant surface load is defined by one vector.
-  **MeshLoad Var:** Creates approximately equivalent point- and line-loads from a variable surface load on a mesh. The variable surface load is defined by one vector for each mesh face. The longest list principle applies when the mesh-faces outnumber the load-vectors.

 **Disassemble Mesh Load:** Splits a mesh-load into corresponding line- and point-loads.

 **Prescribed Displacement:** Prescribes displacements at nodes of given node-indexes or node-coordinates. Select translations or rotations which should be prescribed. For load-cases with no displacements prescribed this will create a support.

2.4.5 Cross Section

 **Cross Sections:** Multi-component for creating cross sections:

-  **Box-Profile (default):** Creates rectangular, trapezoid and triangular hollow cross sections.
-  **Circular Hollow Profile:** Creates circular hollow cross sections.
-  **I-Profile:** Creates I-shaped cross sections.
-  **Shell Const:** Lets you set the height and material of a shell with constant cross section.
-  **Shell Var:** Lets you set the height and material of each face of a shell.
-  **ShellRC Std Const:** A standard reinforced concrete cross section consists of four layers of orthogonal reinforcement. This component allows to define such a cross section which is constant throughout a shell.
-  **ShellRC Std Var:** Same as above, lets one set the reinforced concrete cross section properties for each shell face separately.
-  **Spring-Cross Section:** Defines the spring stiffness of an element.
-  **Trapezoid-Profile:** Creates filled rectangular, trapezoid and triangular cross sections.

 **Disassemble Cross Section:** Retrieves properties of a cross section.

 **Beam-Joint Agent:** Crawls around in the model and adds joints to beams on the basis of geometric relations. Is of type cross section.

Beam-Joints: Adds hinges at the end-points of beams. Is of type cross sections.

Eccentricity on Beam: Sets the eccentricity of a cross section relative to the element axis in global coordinates.

Eccentricity on Cross Section: Sets the eccentricity of a cross section relative to the element axis in local beam coordinates.

Modify Cross Sections: Multi-component for modifying cross sections. Works either directly on a cross section object or indirectly as an autonomous agent:

-  **Modify Beam Cross Sections (default):** Modifies beam cross sections only.
-  **Modify Shell Cross Sections:** Modifies shell cross sections only.

Cross Section Range Selector: Lets you select cross sections by country, shape, family or maximum depth or width.

Cross Section Matcher: Returns for a cross section the best fitting cross section contained in a given list. The matched cross section is equal or better in all mechanical aspects at minimum weight.

Cross Section Selector: Lets you select cross sections by name, regular expression or index from a list of cross sections.

Generate Cross Section Table: Converts a list of cross sections into a string which can be streamed as a csv-file and used as a cross section table.

Read Cross Section Table from File: Reads cross section data from a csv-file.

2.4.6 Material

Material Properties: Sets the characteristic parameters of an isotropic or orthotropic material.

Material Selection: Lets you select a material by name, regular expression or index from a list of materials.

Read Material Table from File: Reads a list of materials from a table given in csv-format.

Disassemble Material: Outputs the physical properties of a material.

2.4.7 Algorithms

AnalyzeThI: Calculates the deflections of a given model using first order theory.

AnalyzeThII: Calculates the deflections of a given model including the effect of axial or in-plane forces.

Analyze Nonlinear WIP: Handles calculations involving large deformations. Is work in progress: the speed of convergence will be improved in future releases. Currently it works best for beams, but can also handle shell structures.

Large Deformation Analysis: Does an incremental geometrically non-linear analysis for loads in load case zero. Return displacements only, no stresses of cross section forces.

Buckling Modes: Calculates the buckling-modes and buckling load-factors of the given model under normal forces N^{II}.

Eigen Modes: Calculates the eigen-modes of the given model according to the special eigen-value problem.

Natural Vibrations: Calculates the natural vibrations of the given model.

Optimize Cross Section: Iteratively selects optimum cross sections for beams, trusses and shells.

BESO for Beams: Optimizes the topology of beams in a a structure by using Bi-directional Evolutionary Structural Optimization.

BESO for Shells: Optimizes the topology of shells in a a structure by using Bi-directional Evolutionary Structural Optimization.

Optimize Reinforcement: Performs reinforcement design for shells. It uses linear elastic cross section forces and the assumption of zero tensile concrete strength for determining reinforcement quantities

Tension/Compression Eliminator: Removes beams or trusses under axial tension or compression. By default compression members will be removed.

2.4.8 Results

Model View: Lets you inspect the general properties of the model.

Deformation-Energy: Retrieves deformation energies of the elements of the model.

Nodal Displacements: Returns nodal displacements: translations/rotations in global x-, y-, and z-direction; rotations about global x-, y- and z-axis.

Principal Strains Approximation: Approximates the principal strain directions from the model deformation at arbitrary points.

Reaction Forces: Returns reaction forces and moments at supports.

Utilization of Elements: Multi-component that returns the utilization of elements. "1" means 100 %:

- 

  Utilization of Beams (default): The utilization of beams is calculated according to EC3 (see section A.4.6).

- 

  Utilization of Shells: Returns the maximum Van Mises stress in each sub-element of the shell.

Beam View: Lets you inspect beam properties: section forces, cross sections, displacement, utilization and stresses. Is to be plugged into the definition after the ModelView-component.

Beam Displacements: Returns displacements along elements: translations/rotations in global x-, y-, and z-direction; rotations about global x-, y- and z-axis.

Section Forces: Retrieves section forces along beams and trusses.

Resultant Section Forces: Retrieves resultant section forces of beams.

Shell View: Lets you inspect shell properties: displacement, utilization, principal stresses and Van Mises stress. Is to be plugged into the definition after the ModelView-component.

Shell Line Results: Multi-component for generating line results on shells:

-  **Force Flow Lines on Shells (default):** Computes flow lines for forces in given direction at user defined positions.
-  **Isolines on shells:** Creates lines that connect points of same value for selected shell results (e.g. principal stresses, displacement, utilization, cross section thickness) at user defined positions. Also returns values and can thus be used for probing the shell state.
-  **Principal Moment Lines on Shells:** Returns the principal moment lines that originate from user defined points on shells.
-  **Principal Stress Directions on Shells:** Outputs the principal stress directions in the center of each shell element.

Shell Vector Results: Multi-component for generating vector results in each element of a shell:

-  **Principal Forces on Shells (default):** Outputs the first and second principal normal forces and moments in the center of each shell element as vectors.
-  **Principal Stresses on Shells:** Outputs the values of first and second principal stress on a given layer in the center of each shell element.

Shell forces: Outputs the values of first and second principal normal forces and moments in the center of each shell element.

2.4.9 Export

Export Model to RStab: Exports a model to RStab5, RStab6, RStab7, RStab8 or Robot by creating a DStV-file.

.

2.4.10 Utilities


Nearest Neighbors: Connects each node of one set to a given number of nearest neighbor nodes or neighbors within a specified distance of another set.


Multi-dimensional Nearest Neighbors: Performs a multidimensional nearest neighbor search on a two sets of vectors.


Cull Curves: Inputs a data tree of straight lines and thins them out so that no lines in different branches are closer than a given limit distance.


Detect Collisions: Counts the number of intersections between the model and a given mesh.


Get Cells from Lines: Creates closed cells from a graph and vertices on a user supplied plane.


Line-Line Intersection: Intersects given lines and returns resulting end-points and pieces.


Line-Mesh Intersection: Returns the points where given lines intersect given meshes.


Mesh Breps: Takes multiple breps and generates a unified mesh from them. The algorithm takes account of common edges and insertion points. This lets one define positions for supports or point-loads on shells.


Principal States Transformation: Transforms given principal vectors of stresses, moments or in-plane forces to an arbitrary direction.


Remove Duplicate Lines: Eliminates identical lines.


Remove Duplicate Points: Eliminates identical points.


Element Felting: Felts elements of a model by connecting them at their mutual closest points.

Mapper: Applies mappings (like Simple Stitch) to a model.


Interpolate Shapes: Interpolates between a base geometry (0.0) and given shape(s) (1.0).


Simple Stitch: Multi-component for defining modes of connection between sets of beams:

- 
Simple Stitch (default): Connects beam sets by a preset number of elements.
- 
Stacked Stitch: Connects beam sets by a preset number of elements that do not intersect each other.
- 
Proximity Stitch: Connects beam sets by a preset number of elements whose maximum inclination can be controlled via min/max offset-limits from their starting point.

User Iso-Lines: Creates iso-lines on a model based on user supplied nodal values.

User Stream-Lines: Creates stream-lines on a model based on user supplied vectors at the nodes.

Chapter 3

In-depth Component Reference

3.1 Model

The subsection “1.Model” of Karamba3D contains components for handling the basic aspects of a statical model.

3.1.1 Assemble Model

In order to calculate the behavior of a real world structure one needs to define its geometry, loads and supports. The component “Assemble” gathers all the necessary information and creates a statical model from it (see figure 3.1).

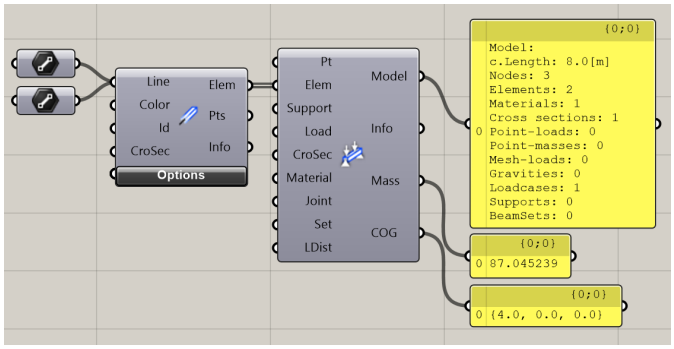

Figure 3.1 The “Assemble”-component gathers data and creates a model from it.

In case that some beams were defined by node indexes then these will refer to the list of points given at the “Pt”-input-plug: the first node in the list has index zero in the model, the next one index one, and so on. The “Pt”-input can also be used to give the model nodes a specific order.
The value at the input-plug “LDist” defines the distance of points below which they will be merged to one. This helps in dealing with inaccurate geometry. The limit distance default value is 5 mm. Snapping together of nodes does not apply to points given via the “Pt”-input-plug. This can be used for defining zero length springs between.
By default, elements with coincident nodes get rigidly connected. Sometimes however it is necessary to introduce a spring element of zero length in order to get more control over how elements interact

– think e.g. of the bolt which connects the two pieces of a scissor mechanism. In such a case one can provide duplicate points via the “Pt”-input. Elements, which connect to these points do so in alternating fashion: the first element in the model connects to the first duplicate node, the elements after that to the second, and so on. The actual connection between the elements can be made via a spring with zero length as shown in fig. 3.2. The local axes of zero length spring elements correspond to the global coordinate system.

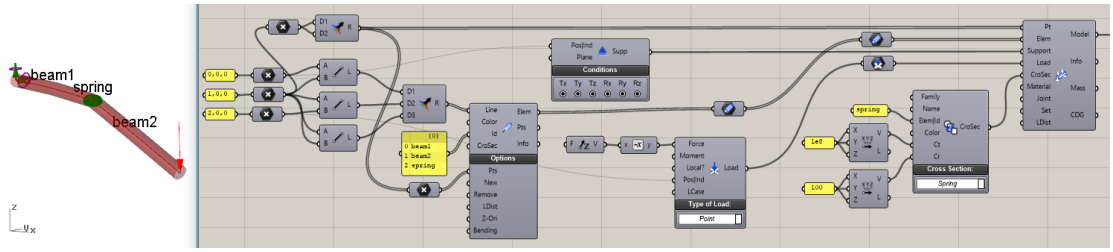

Figure 3.2 In order to define zero length elements provide duplicate points at the “Pt”-input of the “Assemble”- and “LineToBeam”-component. Elements attach to these nodes in alternating fashion.

Cross sections of elements and materials can be defined either upon creating an element or at the “Assemble”-component. The letter option overrides the former and assigns cross sections and materials via element identifiers. Using regular expressions for selecting identifiers of elements provides a flexible means of attaching cross sections and materials to different parts of a model.
The output-plug “Mass” renders the mass of the structure in kilogram, “COG” the position of its center of gravity. When being plugged into a panel the model prints basic information about itself: number of nodes, elements, and so on. At the start of the list the characteristic length of the model is given, which is calculated as the distance between opposing corners of its bounding box.

3.1.2 Disassemble Model

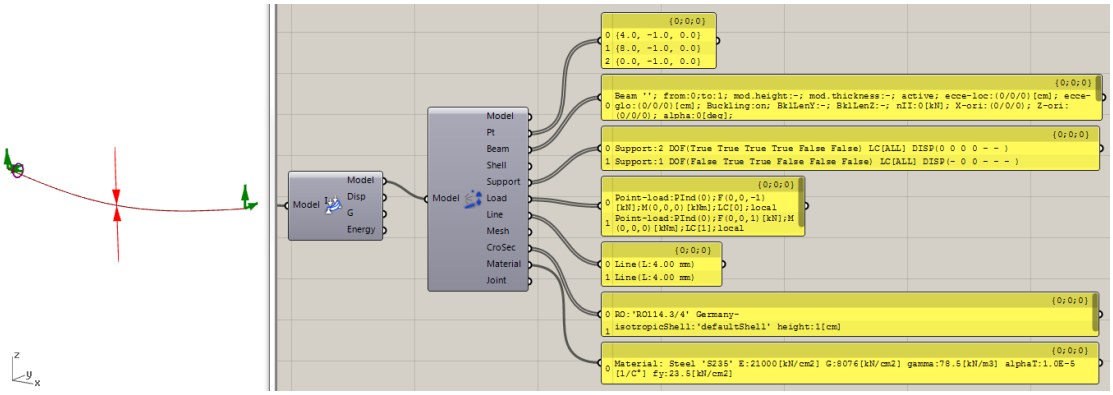

Figure 3.3 Model is decomposed into its components.

It is sometimes necessary to take apart existing models in order to

1. reassemble them in different configurations,
2. retrieve the results of e.g. a cross section optimization.

The “DisassembleModel”-component can be used for decomposing a structural model into its components (see figure 3.3). Resulting loads, supports and elements reference the nodes they connect to by position – regardless whether they were initially defined using coordinates or node-indexes. This allows to reuse parts of an old model and reassemble them in a new model where the node indexes have changed. At the “CroSec”- and “Material”-output only those cross sections and materials show up which were directly fed into the “Assemble”-component. In order to get all cross sections, it is necessary to disassemble the model elements. The cross section materials result from disassembling the cross sections.

3.1.3 Modify Model

In case one wants to change the nodal positions of an existing model the “DisassembleModel”-component comes in handy. The “Pt”-input expects the list of new points to be used as the models new nodal positions (see fig. 3.4).

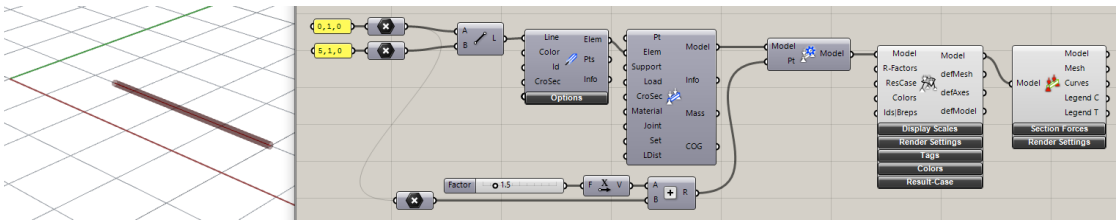

Figure 3.4 The “DisassembleModel”-component changes the nodal positions of model.

3.1.4 Connected Parts

When creating a model based on unprecise geometry or using a generative process, elements may not be connected to each other as desired. The “Connected Parts”-component (see fig. 3.5) takes a model as input and determines its connected parts. It considers beams and trusses only. Connected groups get listed in a data tree in descending order of group size.

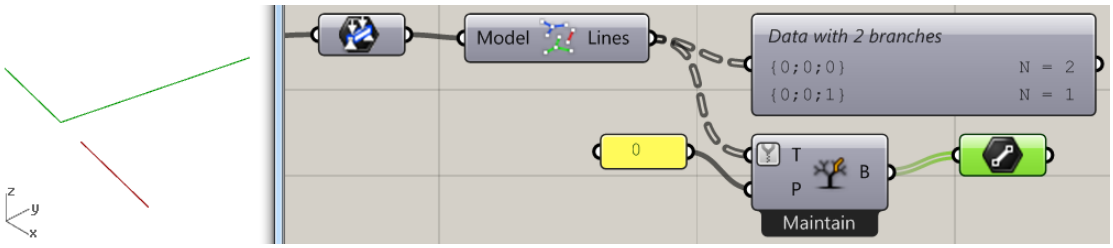

Figure 3.5 The “Connected Parts”-component groups beams into sets of elements that have at least on node in common each.

3.1.5 Activate Element

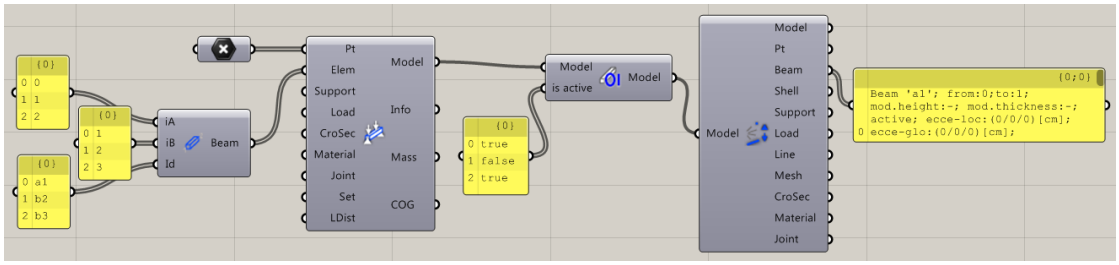

Figure 3.6 Setting the activation state of all elements of a model with a list of boolean values.

The activation state of an element can be controlled with the "Activate Element"-component (see fig. 3.6). This component expects a model and a list of boolean values as input. The list of true/false values will be mapped to the activation status of the elements in the model. "True" corresponds to active, "False" to inactive. Section 3.5.9 shows, how the "Activate Element"-component enables one to view the solution history of the iterative "BESO for Beams"-algorithm.
Karamba3D sets elements inactive by giving them a very weak material with zero weight.

3.1.6 Line to Beam

Figure 3.7 shows how the "LineToBeam"-component takes two lines as input, finds out how they connect and outputs beams as well as a set of unique points which are their end-points. Lines with end-points of identical index get automatically removed. Unless lines get removed, there is a one to one correspondence between the list of input lines and output beams.
The "LineToBeam"-component accepts only straight lines as geometric input. Therefore poly-lines and the like need to be exploded into segments first. All coordinates are in meters (or feet in case of Imperial Units).
The "Color"-input lets one define a color for the rendering of elements. In order to display the colors activate the "Elements"-button in submenu "Colors" of the "ModelView"-component. In addition the option "Cross section" of the submenu "Render Settings" of the "BeamView"-component needs to be on.

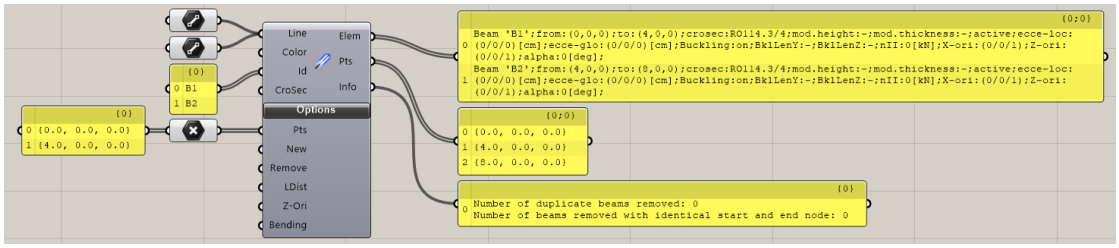

Figure 3.7 The "LineToBeam"-component that turns two lines into beams

Elements can be given non-unique names via the "Id"-input . It takes a list of strings as identifiers for beams. The default value is an empty string. Each beam has a name by default: its zero based index

in the model. Identifiers provide a useful means to group the beams in order to modify or display them.
Cross sections can be attached to elements with the “CroSec”-input. Cross section definitions via the “Assemble”-component override these settings.
A click on the “Options” submenu heading reveals additional input-options of the “LineToBeam”-component:

- “Pts”: The order in which points appear in the output node-list is random by default. However it is sometimes advantageous to identify certain points by their list index in order to put loads on them or to define supports. This can be achieved by feeding a list of coordinates into the “Points”-plug. They will be placed at the beginning of the output nodes-list. So in order that the end-points of the structure in figure 3.7 have index 0 and 1 it is necessary to input a list of points with coordinates (0/0/0) and (8/0/0).
- “New”: If this plug has the value “False” only those lines will be added to the structure that start and end at one of the points given in the input points-list.
- “Remove”: If this option has the value “True” the LineToBeam-component checks for lines that lie on each other and merges such duplicates into one. This prevents an error that is hard to detect by visual inspection alone: Two lines on the same spot mean double member stiffness in the statical model. Alternatively apply the “Remove Duplicate Lines”-component from the karamba3D utilities section on the list of incoming lines. This assures a one-to-one correspondence between lines and elements.
- “LDist”: Sets the limit distance for two points to be merged into one. Points supplied via lines count as identical if their distance is less than that given in “LDist”. The default value of “LDist” is 5 mm. The snapping of nodes does not apply to points supplied via the “Pt”-input-plug. The mechanism for attaching duplicate nodes to elements is identical to that used by the “Assemble”-component (see section 3.1.1).
- “Z-Ori”: The default orientation of beams and trusses is described in section 3.1.14. The “Z-Ori”-input lets one define a non-standard direction for the local Z-axis.
- “Bending”: Allows to switch off the bending stiffness of beams. For details see section 3.1.10.

Beams that meet at a common point are by default connected rigidly in the statical model like they were welded together. See section 3.3.6 on how to define joints at the end of beams. The “Info” output-plug informs about the number of removed nodes and beams.
In order to be of immediate use, beams come with a number of default properties. They can be seen in the top right string-output of figure 3.7: “active” means that a beam will be included in the statical model. The default cross section is a circular hollow profile of diameter 114 mm with a wall-thickness of 4 mm. The default material is steel of grade “S235” according to Eurocode 3.

3.1.7 Connectivity to Beam

In Grasshopper meshing algorithms can generate topological connectivity diagrams. With the help of the “Connectivity to Beam”-component these may be directly converted to beam-structures (see figure 3.8).

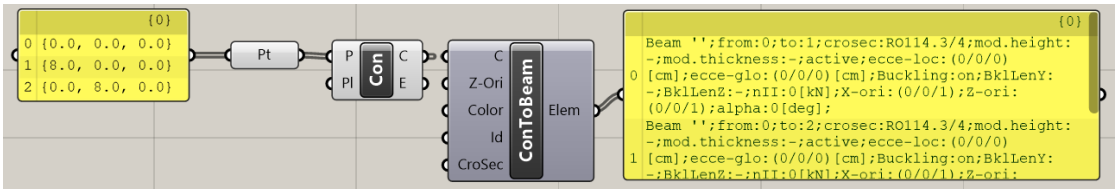

Figure 3.8 The "Connectivity to Beam"-component turns connectivity diagrams into sets of beams.

The input-plugs "Z-Ori", "Color", "Id" and "CroSec" have the same meaning as for the "LineToBeam"-component (see 3.1.6).

3.1.8 Index to Beam

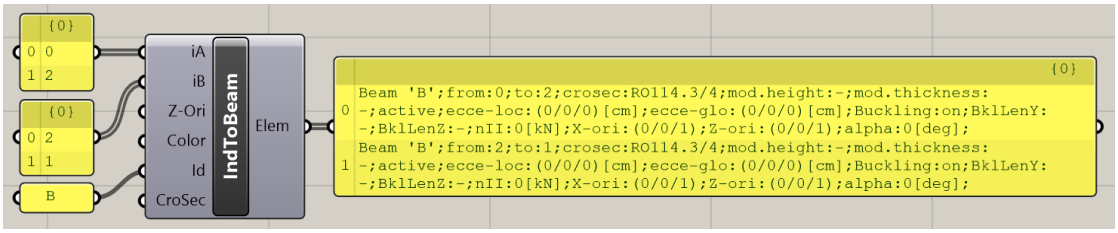

Figure 3.9 The "IndexToBeam"-component lets you directly define the connectivity information of beams.

Sometimes the initial geometry is already given as a set of points and two lists of node-indexes with one entry for each start- and end-point of beams respectively. In such a case it would be cumbersome to convert this information into geometric entities only for feeding it into the "LineToBeam"-component which reverses the previous step. The "IndexToBeam"-component (see figure 3.9) accepts a pair of lists of node-indexes and produces beams with default properties from it. This speeds up model generation considerably for there is no need to compare nodes for coincident coordinates. The "IndToBeam"-component makes it possible to define elements with zero length. This proves useful in case you want to connect elements that touch each other but should not be rigidly connected (think of a scissor – see section 3.3.3 about springs).
The input-plugs "Z-Ori", "Color", "Id" and "CroSec" have the same meaning as for the "LineToBeam"-component (see 3.1.6).

3.1.9 Mesh to Shell

The "MeshToShell"-component takes a triangle or quad mesh and turns it into a group of shell elements (see fig. 3.10). Quads get automatically decomposed to triangles. Shell patches are rigidly connected when some of their nodes have the same index.
Colors can be attached to shells via the "Color"-input. In order to enable the display of shell element colors, activate "Elements" in submenu "Colors" of the "ModelView"-component. In addition "Cross section" needs to be selected in the submenu "Render Settings" of the "ShellView"-component.

Each patch of shells can be given an identifier via input “Id” for later reference when attaching custom material or cross section properties. By default shells have a thickness of $1[cm]$ and steel as their material. Use the “CroSec”-input to change that. Clicking on the “Options” submenu header further unfolds the component: The “Pts”- and “LDist”-input serve the same purpose as in the “LineToBeam”-component – see sec. 3.1.6. Additionally mesh faces with an area smaller than $LDist^2 \cdot 0.1$ get automatically removed
The shell elements used in Karamba3D resemble the TRIC-element devised by Argyris and coworkers (see [1], [2] for details). They are facetted (i.e. flat) elements. Karamba3D neglects transverse shear deformation in case of shell elements.

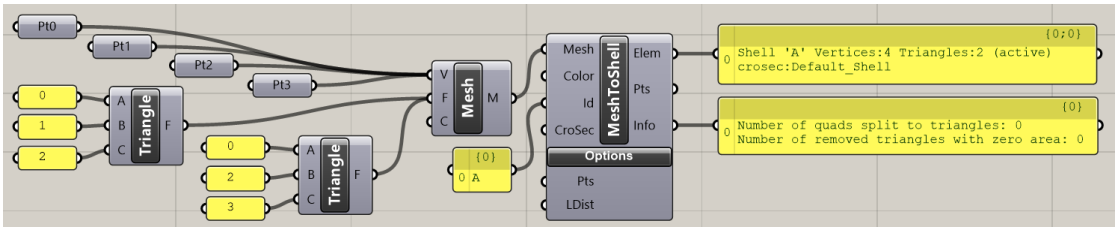

Figure 3.10 The “MeshToShell”-component turns meshes into shells

3.1.10 Modify Element

“Modify Element” is a multi-component which can be applied to shell-, beam- and truss elements. Use the drop-down list at the bottom of the component to select the type.
By default Karamba3D assumes the cross-section of beams to be steel tubes with a diameter of 114 mm and a wall-thickness of 4 mm. Use the “ModifyElement”-component with “Element Type” set to “Beam” to set the beam properties according to your choice. Figure 3.11 shows how this can be done. There are two variants for using the “Modify Element”-component:

1. Insert it in front of the “Assemble”-component and let element objects flow through it (see e.g. the modification of beams in fig. 3.11). By default the “ModifyElement”-component leaves all incoming elements unchanged. Several “ModifyBeam”-components may act consecutively on the same beam.

2. Create a stand-alone element-agent that can be fed into the “Elem”-input of the “Assemble”-component. The input-plug “ShellId” or “BeamId” let you select the elements to be modified. Use regular expressions to specify groups of elements.

Modify Beam

These element properties can be modified:
Activation status of beams
When input “Active” is set to false the corresponding beam is excluded from further calculations until “Active” is reset to true. See section 3.1.5 for an alternative way of setting a beams activation state.
Bending stiffness

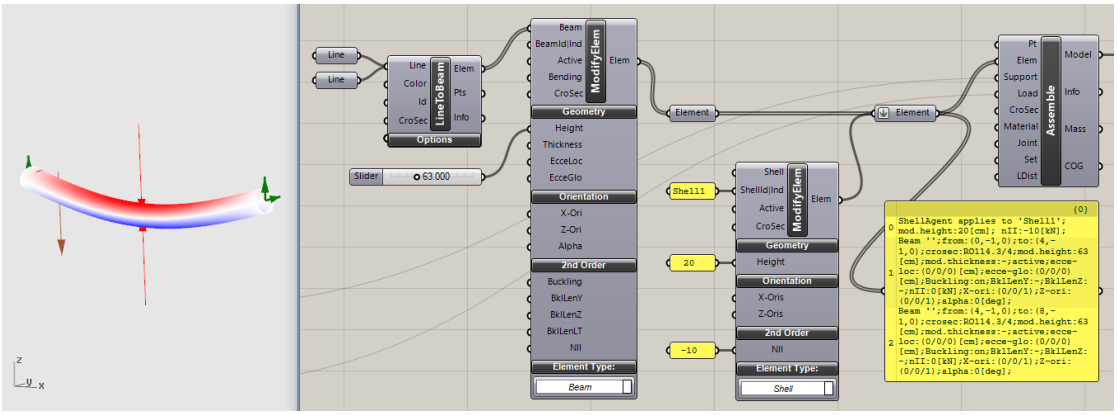

Figure 3.11 Modification of the default element properties.

Beams resist normal forces and bending moments. Setting the "Bending"-input of the "ModifyElement"-component to "False" disables the bending stiffness and turns the corresponding beam into a truss. There exist reasons that motivate such a step:

- Connections between beams that reliably transfer bending and normal force are commonly more expensive than those that carry normal force only. The design of connections heavily depends on the kind of material used: rigid bending connections in wood are harder to achieve than in steel. Yet rigid connections add stiffness to a structure and reduce its deflection. Therefore you are always on the safe side if you use truss elements instead of beams.
- For slender beams i.e. beams with small diameter compared to their length the effect of bending stiffness is negligible compared to axial stiffness. Just think of a thin wire that is easy to bend but hard to tear by pulling.
- Abandoning bending stiffness reduces computation time by more than half for each node with only trusses attached.
- Karamba3D bases deflection calculations on the initial, undeformed geometry. Some structures like ropes are form-active. This means that when a rope spans between two points the deformed geometry together with the axial forces in the rope provide for equilibrium. This effect is not taken into account in Karamba3D first order theory (Th.I.) calculations. In such a case only the bending stiffness of the rope (which is very small) keeps it from deflecting indefinitely. One way to circumvent this lies in using a truss instead of a beam-element when doing first order analysis. The second possibility would be to reduce the specific weight of the rope to zero (see further below). The third possibility would be to start from a slightly deformed rope geometry and apply the external loads in small steps where the initial geometry of each step results from the deformed geometry of the previous one (see section 3.5.4).

Trusses only take axial forces. Therefore they do not prevent the nodes they are connected to from rotating. In case that only trusses attach to a node, Karamba3D automatically removes its rotational degrees of freedom. Otherwise the node could freely rotate which is a problem in static calculations.

As soon as one beam with bending enabled connects to a node the node has rotational degrees of freedom. Bear this in mind when the "Analysis"-component turns red and reports a kinematic system. Transferring only axial forces means that a truss reduces a nodes movability in one direction. A node that is not attached to a support has three translational degrees of freedom. Thus there must be three truss elements that do not lie in one plane for a node to be fixed in space.

Height and Thickness of Cross-sections

"Height" – which in case of circular tubes is equivalent to the outer diameter D – and wall-thickness of a cross-section influence a beams axial and bending stiffness. Karamba3D expects both input values to be given in centimeter. The cross-section area is linear in both diameter and thickness whereas the moment of inertia grows linearly with thickness and depends on D^3 for e.g. full rectangular sections and on D^2 for e.g. I-profiles and box sections. So in case of insufficient bending stiffness it is much more effective to increase a beams height (or diameter) than increasing its wall thickness.

Local and Global Eccentricity of the Beam Axis

The input-plugs "EcceLoc" and "EcceGlo" serve to set the eccentricity of the beam-axis with respect to the connection line between its endpoints. Both expect a three dimensional vector. "EcceLoc" refers the eccentricity to the local, "EcceGlo" to the global coordinate system. Eccentricities of beams can also be defined via the "Eccentricity on Beam"-component (see sec. 3.3.7).

Orientation of the Beam

Lets you define the orientation of a beam. Works analogously to the orientate-beam-component (see 3.1.14).

Buckling property for cross section optimization

Buckling can be turned off for cross section optimization. This lets you simulate pre-tensioned, slender elements without having to really pretension them. The necessary pretension force is roughly the negative value of the largest compressive axial normal force of all load cases.

Buckling length in local beam directions

For doing cross section optimization it is necessary to know a beam's buckling length. Karamba3D approximates it using the algorithm described in section 3.5.8. For cases of system buckling this approximation does not lie on the safe side. The input-plugs "BklLenY", "BklLenZ" and "BklLenLT" allow to specify the buckling length of a beam for its local Y- and Z- axis respectively as well as for lateral torsional buckling. When specified, these values override those from the buckling length calculation of Karamba3D. The value "lg" sets the distance of transverse loads from the center of shear of the cross section. It defaults to zero. Positive values mean that the loads point towards the shear center and thus act destabilizing for lateral torsional buckling. The property "lg" influences the beams utilization with respect to lateral torsional buckling according to Eurocode 3.

Second order theory normal force N^{II}

Axial normal forces influence the stiffness of a beam in second order theory (Th.II) calculations. If compressive they lower, in case of tension they increase its bending stiffness. Think of a guitar string which vibrates at a higher frequency (i.e. is stiffer) under increased tension. In Karamba3D the normal force which impacts stiffness (N^{II}) is independent from the normal force which actually causes stresses in the cross section (N). This enables one to superimpose second order theory results on the safe side by choosing N^{II} as the largest compressive force N of each beam.

Modify Shell

Height

Sets a uniform cross section height throughout the shell.

Second order theory normal force N^{II}

As for beams N^{II} for shells specifies the in-plane normal force which impact stiffness in case of second order theory calculations. It is a force per unit of length assumed to be of same magnitude in all directions.

3.1.11 Point-Mass

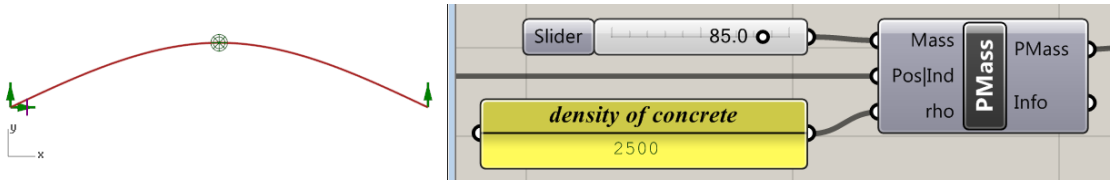

Figure 3.12 Vibration mode of beam with point mass in the middle.

Karamba3D is capable of calculating the natural vibration modes and frequencies of structures (see sec. 3.5.7). For results to match reality the inertia properties of a structure need to be modeled correctly. Masses of elements (e.g. beams, trusses, shells) are automatically taken care of. All other items need to be included via point-masses. Be aware of the fact that masses defined with the "Point-Mass"-component do not have a weight but inertia only! Thus they effect only the calculation of natural frequencies. The "Point-Mass" component expects a mass in kg at its input-plug "Mass" (see fig. 3.12). Nodes where masses shall sit can be identified by supplying node indexes or positions (just like for point-loads). Point masses get displayed as green spheres. Their diameters result from the volume calculated as mass divided by density. The latter defaults to 7850 $\mathrm{kg/m^3}$ (steel) and can be provided at the input-plug-"rho".

3.1.12 Disassemble Element

When interested in the information contained in a beam or shell element feed it into the "DisassembleElement"-component (see fig. 3.13). The component contains several subsections which can be unfolded by clicking on the dark menu header.

3.1.13 Make Beam-Set

The "Make Beam-Set"-component provides a practical way for grouping different elements under one identifier (see fig. 3.14). Beam-sets need not be disjoint. The "Beam Id"-plug expects a list of strings with beam-identifiers, beam indexes, other beam-set-identifiers or a regular expression. Regular expressions have "&" as their first character by definition. "Set Id" expects a string which serves as identifier of the new set of beams.

The group of beams defined by a set can be used for defining geometric mappings. In this context a beam-set represents a polygon of straight segments. The order of the elements in the set is defined by the order in which they were entered into the set. Such polygons can be split at an arbitrary position (see e.g. section 3.8.15). "MinSLen" (minimum segment length) lets you set the minimum

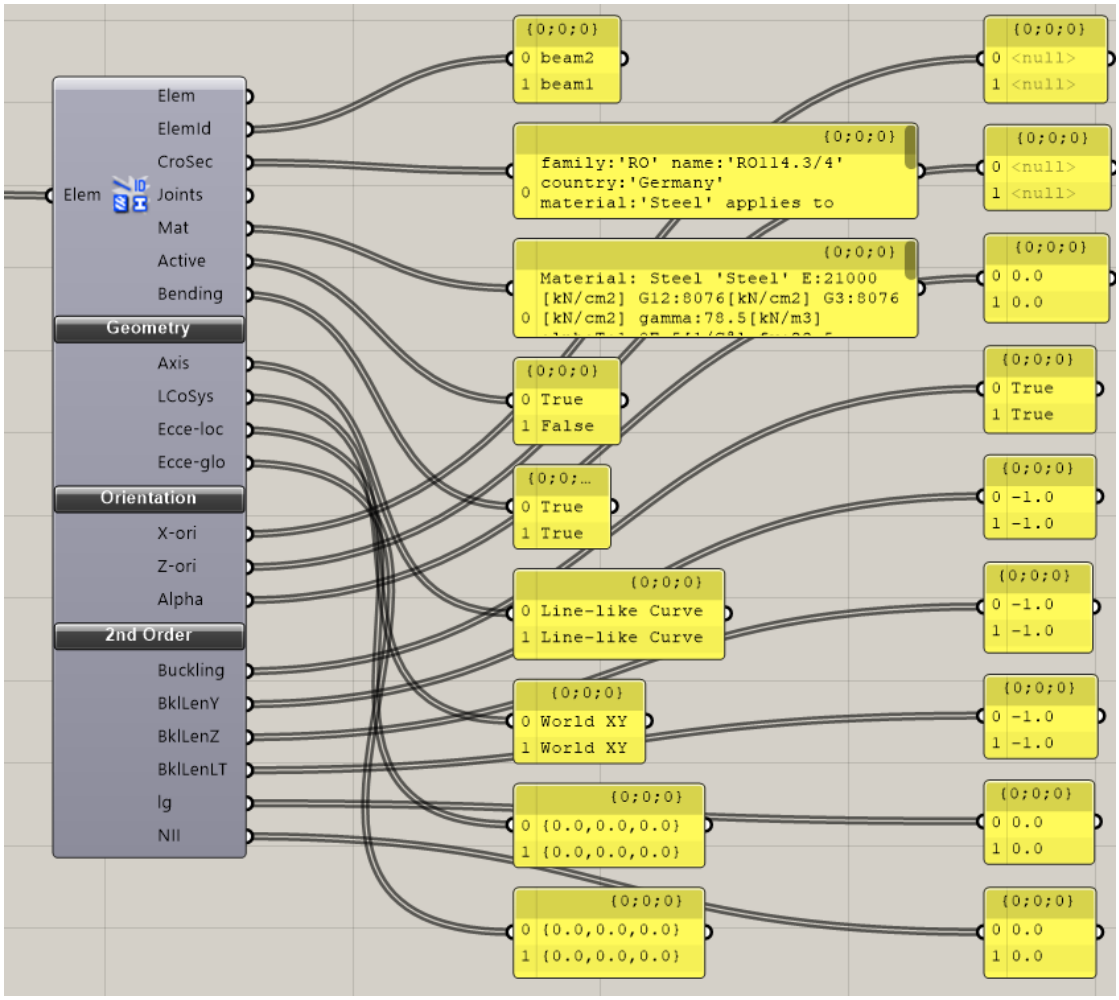

Figure 3.13 A beam decomposed into its individual parts.

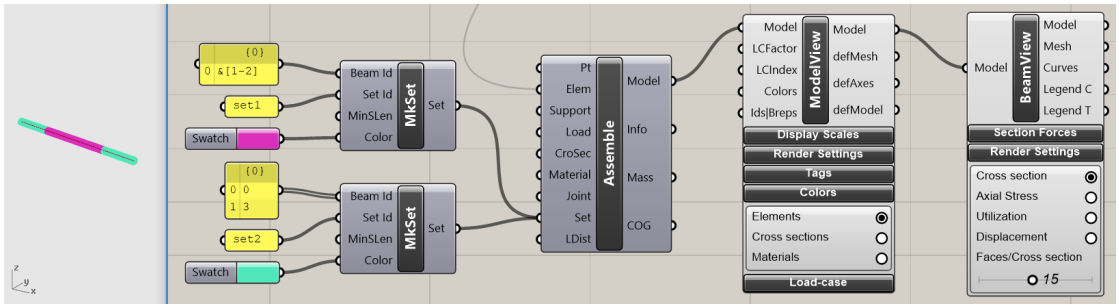

Figure 3.14 Beam-sets can be used to group beams.

length which may result from such a split. In case of potentially smaller segments the intersection point snaps to its nearest neighbor.
In order to group a structure visually, beam-sets can be given different colors. These colors show when "Cross section" is enabled in the "BeamView"component's "Render Settings" (see section 3.6.7) and the option "Elements" in the submenu "Colors" of the "ModelView"-component is on.
The identifier of a beam-set can be used anywhere instead of a beam identifier. In order to be registered with the model, beam-sets need to be fed into the "Set" input-plug of the "Assemble"-component.

3.1.14 Orientate Element

"Orientate Element" is a multi-component where the drop-down list under "Element Type" lets one select between beams and shells.

Orientate Beam

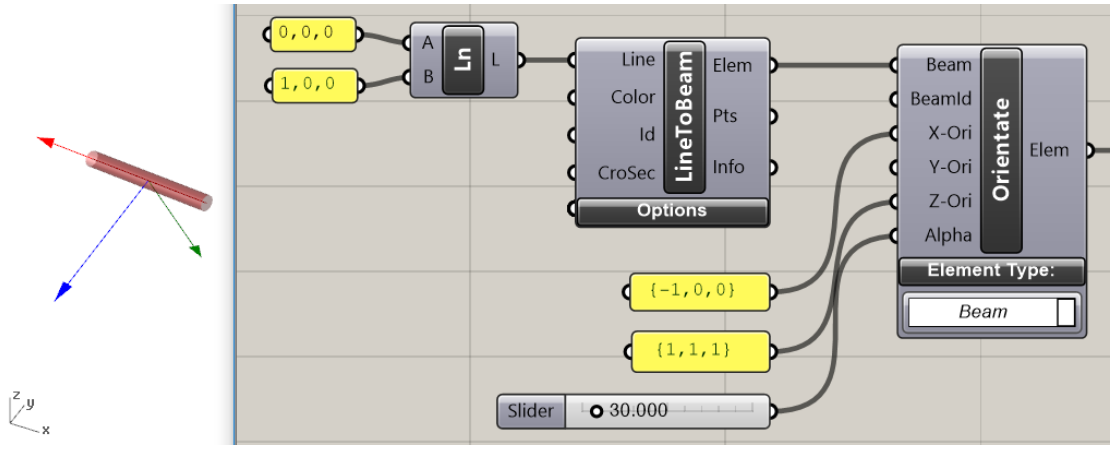

Figure 3.15 The orientation of the local beam coordinate system can be controlled with the "OrientateBeam"-component.

In Karamba3D the default orientation of the local coordinate system of a beam or truss follows these conventions:

- The local X-axis (of red color) is the beam axis and points from starting-node to end-node.
- The local Y-axis (green) is at right angle to the local X-axis and parallel to the global XY-plane. This specifies the local Y-axis uniquely unless the local X-axis is perpendicular to the XY-plane. If this is the case, the local Y-axis is chosen parallel to the global Y-axis. The default criteria for verticality is, that the z-component of the unit vector in axial direction is larger or equal to 0.999 999 995. This value can be changed in the karamba.ini-file via the "limit_parallel" property.
- The local Z-axis (blue) follows from the local X- and Y-axis so that the three of them form a right-handed coordinate system.

The local coordinate system affects the direction of locally defined loads and the orientation of the element's cross section. Use the "Orientate Beam" component to set the local coordinate system (see fig. 3.15):

- The input plug "X-axis" accepts a vector. The local X-axis will be oriented in such a way that its angle with the given vector is less than 90 degree. This allows to give a consistent orientation to a group of beams.
- The local Y-axis lies in the plane which is defined by the local X-axis and the vector plugged into the "Y-axis"-input. If the Y-axis is parallel to the beam axis it is not applicable to the element.
- If no vector is supplied at the "Y-axis"-input or the given Y-axis is not applicable, then the local Z-axis of the beam lies in the plane which is defined by the local X-axis and the vector plugged into the "Z-axis"-input.
- "Alpha" represents an additional rotation angle (in degree) of the local Z-axis about the local X-axis.

In order to control the orientation of a beam, the "Orientate Beam"-component can be applied in two ways:

1. "Flow-through": Plug it in between a "LineToBeam"- and an "Assemble"-component. The changes will be applied to all beam elements which pass through it. In case of shell elements the output is "Null".
2. "Agent". Specify beams by identifier via input "BeamId" and plug the resulting beam-agent directly into the "Elem"-input of the "Assemble"-component. This method allows to harness the power of regular expressions for selecting elements (see section 3.1.15).

Orientate Shell

For shells the default orientation of their local coordinate systems can be seen in fig. 3.16. The following convention applies: The local x-axis is parallel to the global x-direction unless the element normal is parallel to the global x-direction. In that case the local x-axis points in the global y-direction. The local z-axis is always perpendicular to the shell element and its orientation depends on the order of the vertices of the underlying mesh face: If the z-axis points towards ones nose, the order of the face vertices is counter-clockwise[1]

[1] See fig. 6.17 in [12] for an unforgettable way of remembering the right-hand rule of rotation.

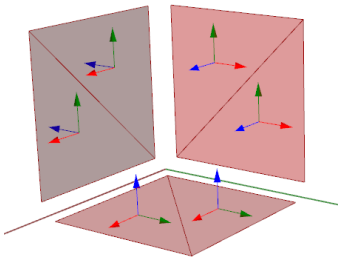

Figure 3.16 Default orientation of the local shell coordinate systems.

The “orientate Shell”-component lets one control local x- and z- directions of the elements which make up a shell: “X-oris” and “Z-Oris” inputs expect lists of direction vectors, one for each mesh face. In case the number of vectors does not match the number of faces the longest list principle applies. Infeasible directions (e.g. a prescribed z-vector which lies in the plane of an element) get ignored. Regarding the application of the “orientate Shell”-component the same two options (“flow-through” or “agent”) exist as for the “Orientate Beam”-component.

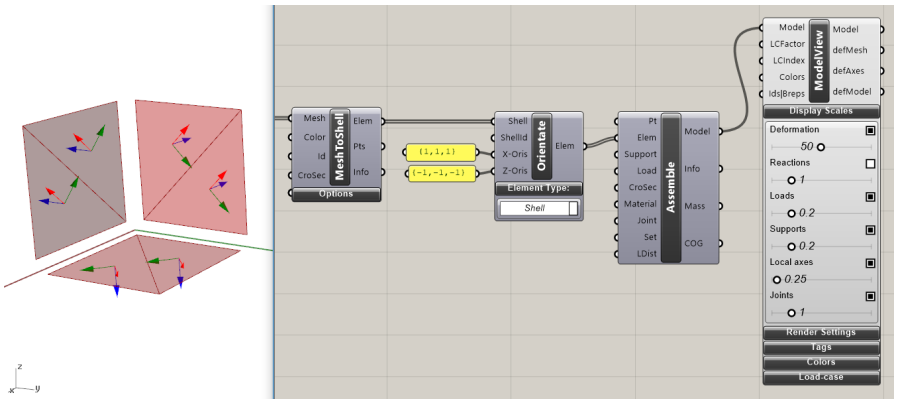

Figure 3.17 Default orientation of the local shell coordinate systems.

3.1.15 Select Beam

All structural elements can be given identifiers, i.e. names. These names need not be unique: Two elements can have the same name without Karamba3D complaining. By default element has a default identifier: its index. This is the reason why it is not allowed to have an integer number as an element identifier. Figure 3.18 shows how a list of elements can be split into two data trees using their identifiers. The “Select Beam”-component expects a list of elements in “Elems” as well as a list of identifiers or regular expressions in “Id”. Regular expressions need to be prefixed by a “&”. They represent a very mighty selection tool. In fig. 3.18 one can see three use-cases :

- “&.[1-2]”: a “.” matches any character; “[1-2]” matches one character in the range of “1” to “2”. This is equivalent to “[12]”.

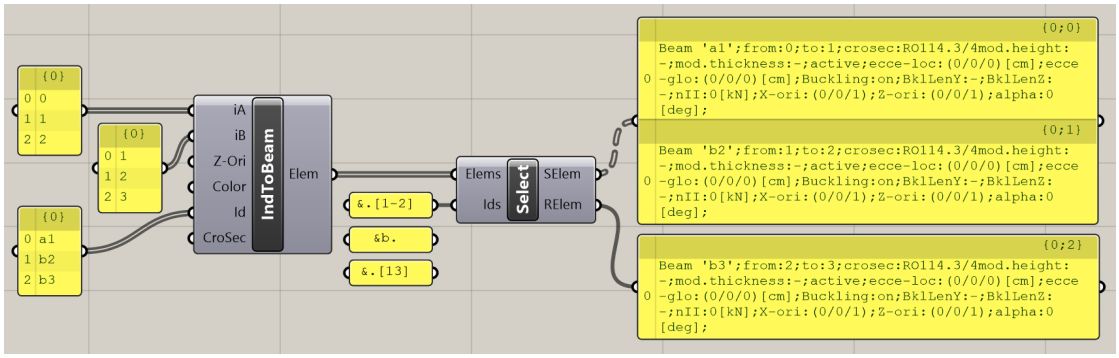

Figure 3.18 Elements can be selected by using their identifiers.

- "&b.": matches any identifier that starts with "b" followed by an arbitrary character.
- "&.[13]": matches any identifier that starts with an arbitrary character followed either by "1" or "3".

There are two output-plugs on the "Select Beam"-component: "SElem" renders the selected elements which match the selection criteria, "RElem" returns the rest. The entries of the "SElem" and "RElem" output data remember their spot in the original list of elements. Joining them results in the original order of elements.

3.1.16 Support

Without supports a structure would have the potential to freely move around in space. This is not desirable in case of most buildings. Thus there should always be enough supports so that the structure to be calculated can not move without deforming i.e. exhibits no rigid body modes.
When defining the supports for a structure one has to bear in mind, that in three dimensional space a body has six degrees of freedom (DOFs): three translations and three rotations (see figure 3.19). The structure must be supported in such a way that none of these is possible without invoking a reaction force at one of the supports. Otherwise Karamba3D will either refuse to calculate the deflected state or render very large displacements. Sometimes you get results from moveable structures although you should not: The reason for this lies in the limited accuracy of computer-calculations which leads to round-off errors. Sometimes one is tempted to think that if there act no forces in one direction – consider e.g. a plane truss – then there is no need for corresponding supports. That is wrong: What counts is the possibility of a displacement.
Errors in defining support conditions are easy to detect with Karamba3D: In section 3.5.6 it is shown how to calculate the Eigen-modes of a structure. This kind of calculation works even in cases of moveable structures: rigid body modes – if present – correspond to the first few eigen-modes.
Figure 3.20 shows a simply supported beam. The "Support"-component takes as input either the index[3] or the coordinates of the point (or a list with indexes or positions of points) to which it applies.

[3] In order to find out the index of a specific node, enable the node-tag checkbox in the "ModelView"-component. See section 3.1.6 on how to predefine the index of specific nodes

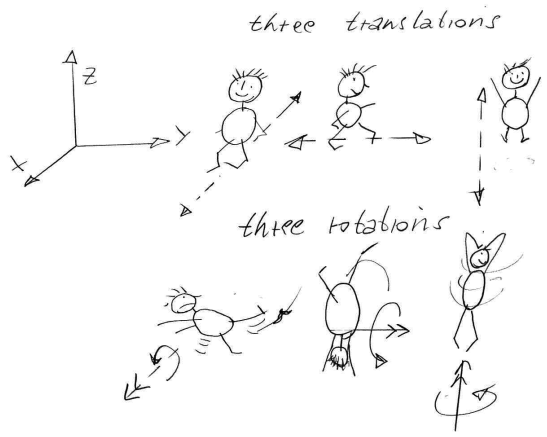

Figure 3.19 Metaphor for the six degrees of freedom of a body in three-dimensional space.

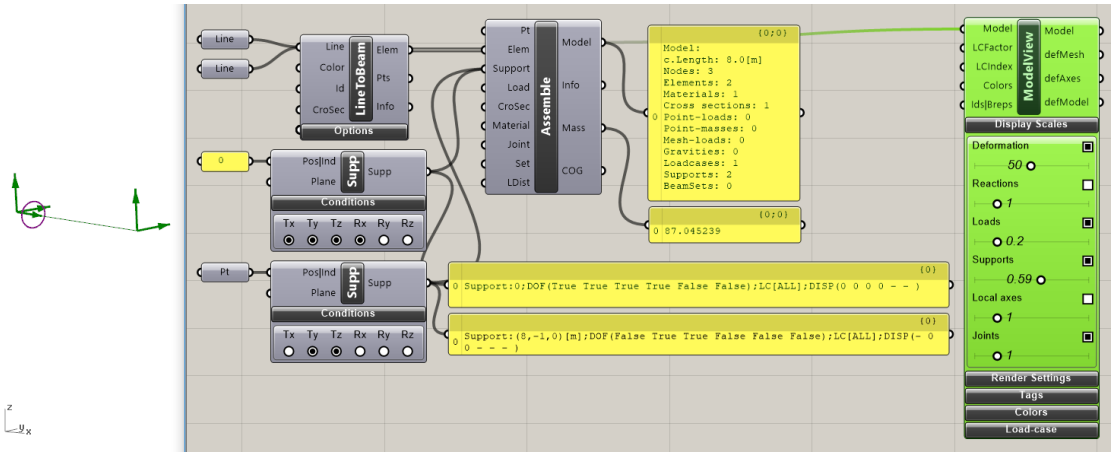

Figure 3.20 Define the position of supports by node-index or position.

By default the coordinate system for defining support conditions is the global one. This can be changed by defining a plane and feeding it into the "Plane"-input plug of the "Support"-component. Six small circles on the component indicate the type of fixation: The first three correspond to translations in global x, y and z-direction, the last stand for rotations about the global x,y and z-axis. Filled circles indicate fixation which means that the corresponding degree of freedom is zero. The state of each circle can be changed by clicking on it. The string output of the component lists node-index or nodal coordinate, an array of six binaries corresponding to its six degrees of freedom and the number of load-case to which it applies. Supports apply to all load cases by default.

Supports cause reaction forces. These can be visualized by activating "Reactions" in the "Display Scales" section of the "ModelView"-component (see section 3.6.1). They show as arrows with numbers in green – representing forces – and purple – representing moments. The numbers either mean kN in case of forces or kNm when depicting moments. The orientation of the moment arrows corresponds to the screw-driver convention: They rotate about the axis of the arrow anti-clockwise when

looked at in such a way that the arrow head points towards the observer[2].

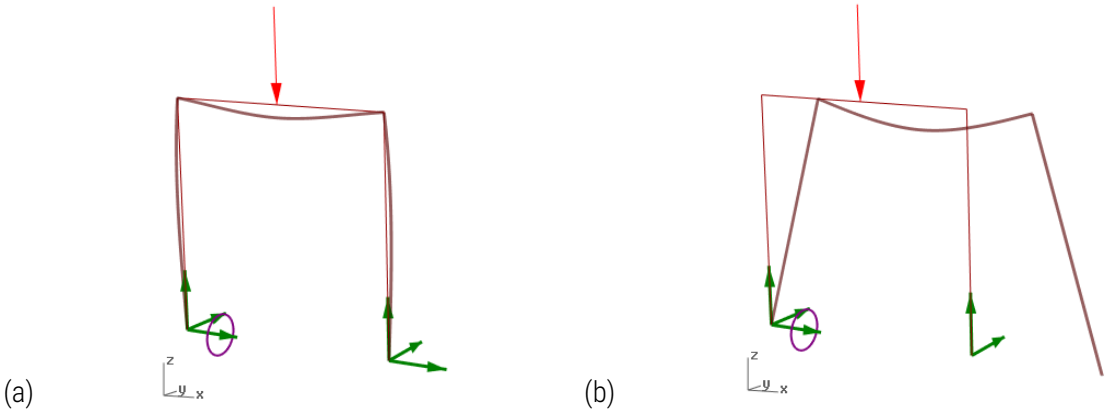

Figure 3.21 Influence of support conditions – undeformed and deflected geometry. *Left:*All translations fixed at supports. *Right:* One support moveable in horizontal direction.

From the support-conditions in figure 3.20 one can see that the structure is a simply supported beam: green arrows symbolize locked displacements in the corresponding direction. The translational movements of the left node are completely fixed. At the right side two supports in y- and z-direction suffice to block translational movements of the beam as well as rotations about the global y- and z-axis. The only degree of freedom left is rotation of the beam about its longitudinal axis. Therefore it has to be blocked at one of the nodes. In this case it is the left node where a purple circle indicates the rotational support.
The displacement boundary conditions may influence the structural response significantly. Figure 3.21 shows an example for this: When calculating e.g. the deflection of a chair, support its legs in such a way that no excessive constraints exist in horizontal direction – otherwise you underestimate its deformation. The more supports one applies the stiffer the structure and the smaller the deflection under given loads. In order to arrive at realistic results introduce supports only when they reliably exist.
By default the size of the support symbols is set to approximately 1.5 m. The slider with the heading "Support" on the "ModelView"-component lets you scale the size of the support symbols. Double click on the knob of the slider in order to set the value range.

3.2 Load

3.2.1 Loads

Currently Karamba3d supports these types of loads: gravity-, point-, imperfection-, pretension-, temperature-loads, constant mesh-loads, variable mesh-loads and prescribed displacements at supports. An arbitrary number of point-, mesh-, etc.-loads and one gravity-load may be combined to form a load-case of which again an arbitrary number may exist. Figure 3.22 shows the definition of loads with the help of the "Loads" multi-component. On the bottom of the "ModelView"-component (see section 3.6.1) there is a drop-down-list (unfold it by clicking on the "Result-case Selection"-menu

[2] See fig. 6.17 in [12] for an unforgettable way of remembering the right-hand rule of rotation.

header) which can be used to select single load-cases for display. Select "–all–" in order to view all existing load-definitions of all load-cases simultaneously. Use the force-slider to scale the size of the load-symbols (double-clicking on its knob lets you change the value range and its current value).

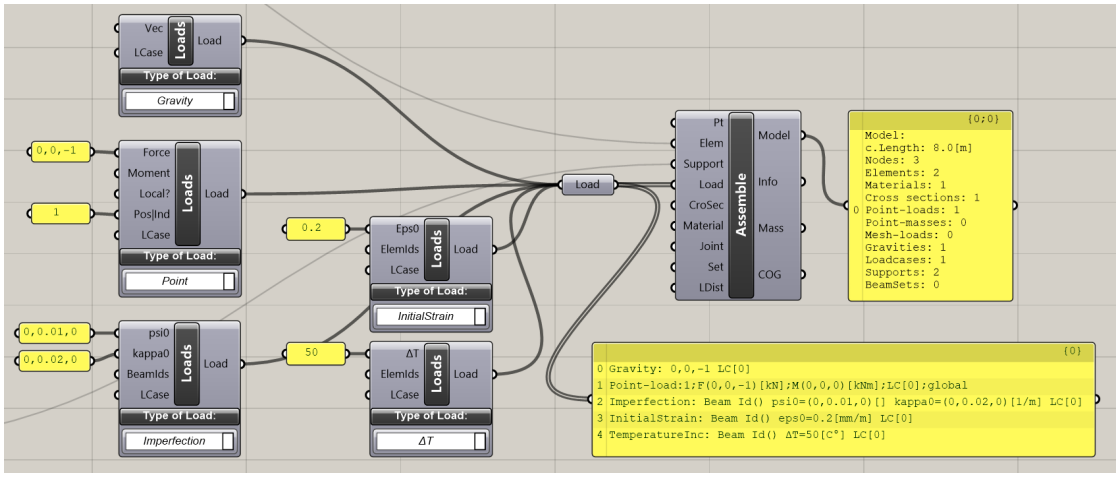

Figure 3.22 Simply supported beam with five loads.

Gravity

It is the default setting when you place a "Loads"-component on the canvas.
Each load case may contain zero or one definition for the vector of gravity. In this way one can e.g. simulate the effect of an earthquake by applying a certain amount of gravity in horizontal direction. For Vienna, which has medium earthquake loads, this amounts to approximately $14\,\%$ of gravity that a building has to sustain in horizontal direction. In areas with severe earthquake loads this can rise to $100\,\%$ (this however also depends on the stiffness properties of the structure and underlying soil). Gravity applies to all active elements in the statical model for which the specific weight gamma (see section 3.4.1) is not zero. The gravity vector defines the direction in which gravity shall act. A vector of length one corresponds to gravity as encountered on earth.
When working in SI-units Karamba3D assumes a value of $10\,\mathrm{m/s^2}$ for the acceleration of gravity. In case of Imperial Units $g = 9.806\,635\,2\,\mathrm{m/s^2}$ is used. Otherwise the conversion from pound mass to pound force does not work. The value of g can be set in the "karamba.ini"-file, which resides in the "Karamba3D" installation folder.

Point-Load

The component "Point-Load" lets you define loads on points. These get attached to their points either by node-index[6] or coordinate. Feed a corresponding list of items into the "Pos|Ind"-plug (quite analogous to the "Support"-component). Point-loads can be either a forces (kN) or moments (kNm).

[6] In order to find out the index of a specific node enable the "node tag"-checkbox in the "ModelView"-component. See section 3.1.6 on how to predefine the index of specific nodes or node-position

Feed a force- or moment-vector into the “Force” or “Moment” input-plug. Its components define the force or moment in global x-, y- and z-direction.
When set to “True” the boolean input “Local?” makes loads and moments follow the nodal rotations in large displacement calculations (see section 3.5.4).
Plugging a point-load into a panel component gives the following information: Node-index where the load gets applied or position, force-vector, moment vector, the number of load case to which it belongs and whether the load is tied to the nodal coordinate system.
By default point loads will be put into load case zero. Any positive number fed into the “LCase”-plug defines the load case to which the corresponding load will be attributed. A value of -1 signals that the load acts in all existing load cases.
For more information on loads and some typical values see section A.4.2.

Imperfection-Load

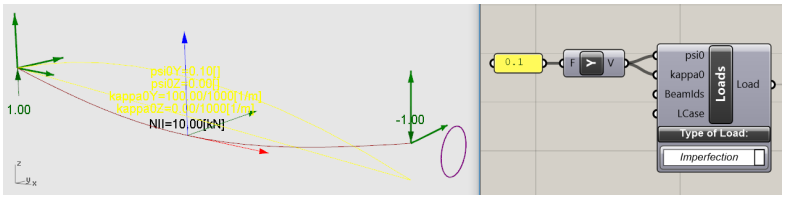

Figure 3.23 Displacements and reaction forces of an initially straight beam with a second order theory normal force of $N^{II} = 10\,\mathrm{kN}$, an initial inclination of $0.1\,\mathrm{rad}$ about the local y-axis and an initial curvature of $0.1\,\mathrm{rad/m}$.

There exists no such thing as an ideally straight column positioned perfectly vertical. The deviation of a real column from its ideal counterpart is called imperfection. This term comprises geometric and material imperfections.
The “Imperfection” variant of the “Loads” multi-component allows to specify geometric imperfections (see fig. 3.23). “psi0” takes the vector of the initial inclination of the beam axis about the axes of the local element coordinate system in radians. With “kappa0” one can specify the initial curvature. A positive component of curvature means that the rotation of the middle axis about the corresponding local coordinate axis increases when moving in longitudinal beam direction. Small inclinations and curvatures are assumed.
Imperfection loads do not add directly to the beam displacements. They act indirectly and only in the presence of a normal force N^{II}. An initial inclination ψ_0 causes transverse loads $\psi_0 \cdot N^{II}$ at the elements endpoints. An initial curvature κ_0 results in a uniformly distributed line load of magnitude $\kappa_0 \cdot N^{II}$ and transverse forces at the elements endpoints that make the overall resultant force zero. For details see e.g. [10].

Initial Strain-Load

Karamba3D lets you define axial initial strains. Fig. 3.24 shows a beam with both ends fixed, subject to a positive initial constant strain and curvature. The unit of dimension of the pretension which gets fed into the “Eps0” plug is $\mathrm{mm/m}$. A positive value means that the element gets longer.

Applying initial strain to an element is not the same as applying a pair of opposite forces or moments at its endpoints: In case of initial strain, the axial force in the element depends on its boundary conditions: If the structure to which it connects is very stiff then the resulting axial force will be $N = -\epsilon_0 \cdot A \cdot E$. In figure 3.24 the supports are rigid, the elements cross section $A = 25\ \mathrm{cm}^2$, Young's Modulus $E = 21\,000\ \mathrm{kN/cm}^2$ and $\epsilon_0 = 0.00015$. This results in an axial force of $N = -78.75\ \mathrm{kN}$ and shows up as horizontal support reactions. When the rest of the structure does not resist, then a pretension-load merely results in lengthening or shortening the corresponding element.
The "Kappa0"-input is a vector of curvature values with respect to the local element axes. A positive component value signifies an anti-clockwise rotation about the corresponding axis.
The input plug "ElemIds" defines the elements where the load acts and "LCase" the load-case.

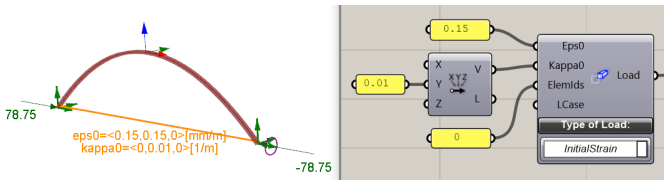

Figure 3.24 Member under initial strains fixed at both ends and resulting support reactions.

Temperature-Load

The definition of temperature loads works analogously to defining pretension loads (see sec. 3.2.1). The coefficient of thermal expansion (see section 3.4.1) characterizes the response of a material to temperature changes.

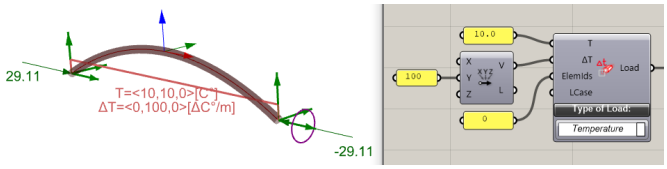

Figure 3.25 Temperature load on a member which is fixed at both ends.

Line-Load on Element

Figure 3.26 shows a tilted structure consisting of three beams under the action of a uniformly distributed load at elements "0" and "2". The load acts parallel to the beams local z-axis. The components of the load vector are assumed to be given in kilo Newton per meter $\mathrm{kN/m}$. The input-plug "BeamIds" receives a list of the identifier of the beams on which the load shall act. See section 3.1.6 for how to attach identifiers to beams. By default beams are named after their index in the FE-model. There are three options for the orientation of the load: "local to element", "global" and "global proj.". Their meaning corresponds to the options available for mesh-loads (see fig. 3.29). The input-plug "LCase" which designates the load case defaults to "0".

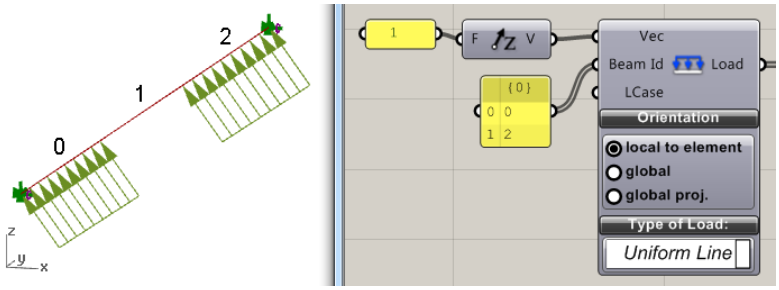

Figure 3.26 Line loads on a structure consisting of three beam elements defined in local beam coordinate systems.

Mesh-Load: Const and Variable

Mesh

The "MeshLoad"-component can be used to transform surface loads into equivalent node- or element-loads. This lets you define life-loads on floor slabs, moving loads on bridges (see example "Bridge.ghx" in the examples collection on the Karamba3D web-site), snow on roofs, wind-pressure on a facade, etc.. The mesh where the load is applied and the underlying structure do not need to be connected. It needs to be fed into the "Mesh"-input-plug.

Vec

There exist two types of mesh-loads:

1. "MeshLoad Const": for loads which are constant throughout the mesh.
2. "MeshLoad Var": lets one set specific load-values for each face of the mesh.

These two variants differ with respect to the data-structure expected at the input "Vec" and "Vecs" respectively: Either a single vector for specifying a constant load or a list of vectors. In the latter case the list items are applied to the mesh-faces based on the longest list principle. In what follows the "MeshLoad Const"-variant will be depicted but everything mentioned there applies to "MeshLoad Var" also.

Figure 3.27 left side shows a simply supported beam and a mesh which consists of two rectangular faces. Each face covers one half of the beam and has a width of $2\,\mathrm{m}$ perpendicular to the beam axis. With a distributed load of $1\,\mathrm{kN/m^2}$ in negative global Z-direction a uniformly distributed load of $2\,\mathrm{kN/m}$ results.

Pos

In order to define structure nodes where equivalent point-loads may be generated, plug a list of their coordinates into the "Pos"-plug. These need to correspond to existing nodes – otherwise the Assembly-component turns red. Offending nodes will be listed in its run-time error message. By default all points of the structure are included. Uncheck "Point loads" to avoid point-loads.

BeamIds

With the input-plug "BeamIds", groups of elements can be specified on which equivalent loads shall be generated. By default all beams of the model are included. In case no beam loads shall be included uncheck the "Line loads" button on the "Generation" submenu.

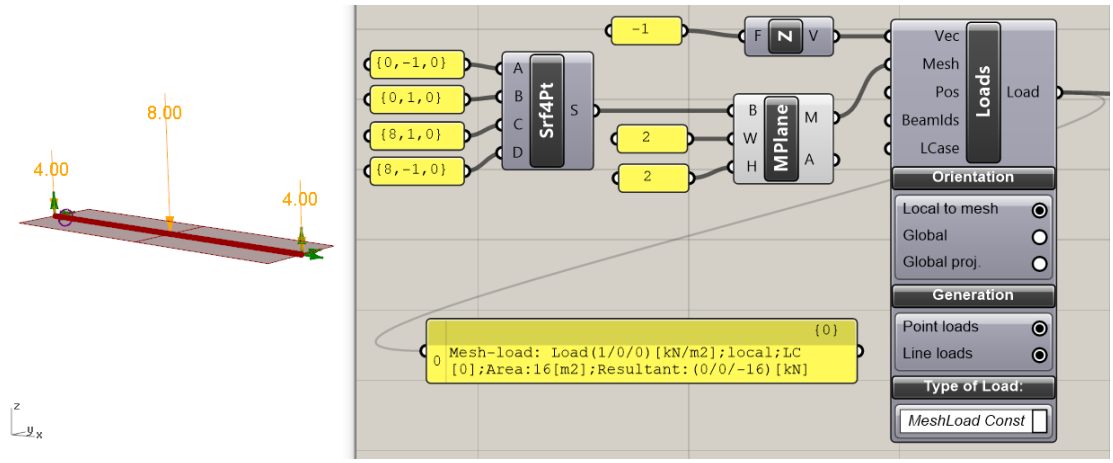

Figure 3.27 Simply supported beam loaded with line loads that approximate a given, evenly distributed surface load on a mesh.

The procedure for calculating nodal loads and uniformly distributed beam loads from surface loads consists of the following steps: First Karamba3D calculates the resultant load on each face of the given mesh. Then the resultant load of each face gets evenly distributed among its three or four vertices.

The second step consists of distributing the vertex-loads among the nodes of the structure. In order to arrive at beam loads additional helper-nodes along their axes get generated. The mutual distance of those is chosen equal to a third of the mean edge length of the given mesh.

Each mesh vertex transfers its load to the nearest node. In case that there are several nodes within a radius of less than "LDist" as set at the Assemble-component (see section 3.1.1) the vertex load gets evenly distributed among them. The loads received by the helper-nodes along beam axes get summed up and divided by the element length. This results in the approximately equivalent uniformly distributed load which is placed on the element. From the procedure described, one can see that a crude mesh may lead to a locally incorrect distribution of loads. In the system shown in fig. 3.27 the points closest to the vertices are the element's end-points. Therefore the helper nodes along the beam-axis do not receive a share in the mesh-load and thus no line-load results.

Fig. 3.28 shows a similar setting as in fig. 3.27. The difference lies in the refined mesh with more vertices along the beam axis. Now loads from the mesh vertices get distributed also to the helper nodes along the element axis. This leads to the generation of a uniform line-load.

LCase

Set the "LCase"-input to the index of the load case in which the surface load shall act. Indexing of load-cases starts with zero, "-1" is short for all load cases.

Orientation

The right side of figure 3.28 shows what data the "MeshLoad const"-component collects: The input-plug "Vec" expects a vector which specifies the surface load. Its physical unit is kilo Newton per square meter $\mathrm{kN/m^2}$. The orientation of the load-vector depends on the checkbox selected under "Orientation" (see also figure 3.29):

- "local to mesh": The convention for local coordinate systems for local loads corresponds to

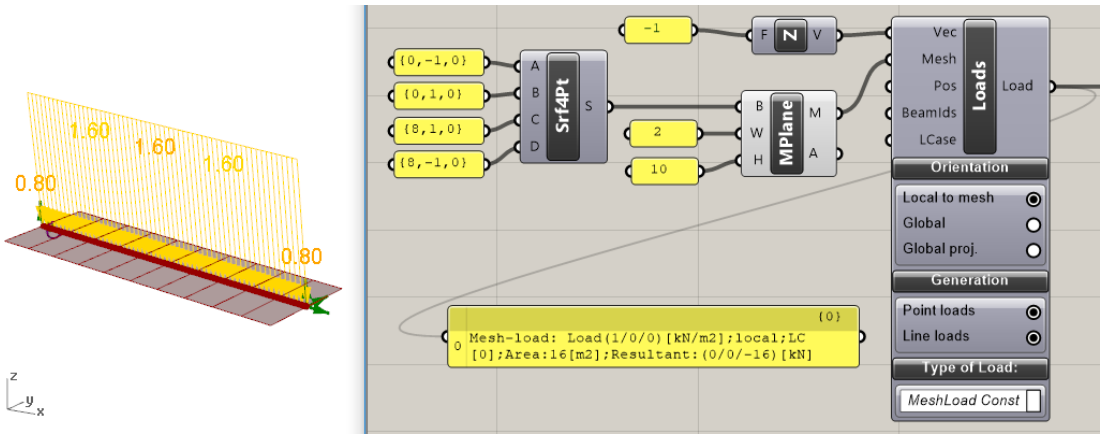

Figure 3.28 Simply supported beam loaded with point loads (dark orange) that approximate a given, evenly distributed surface load on a mesh.

that given in section 3.1.14: The local x-axis is parallel to the global x-direction unless the mesh-face normal is parallel to the global x-direction. In that case the local x-axis points in the global y-direction. The local z-axis is always perpendicular to the mesh-face and its orientation depends on the order of the vertices: If the z-axis points towards ones nose, the order of the face vertices is counter-clockwise.

Z-component of the force vector is at right angle to the mesh-face; the Y-component acts horizontally if the mesh-face X-axis is not parallel to the global Z-axis. Otherwise the Y-component of the force is parallel to the global Y-axis. This means a surface load with components only in Z-direction acts like wind pressure.

- "global": The force-vector is oriented according to the global coordinate system. This makes the surface load behave like additional weight on the mesh plane.
- "global proj.": The force-vector is oriented according to the global coordinate system. The corresponding surface load is distributed on the area that results from projecting the mesh-faces to global coordinate planes. In such a way the action of snow load can be simulated.

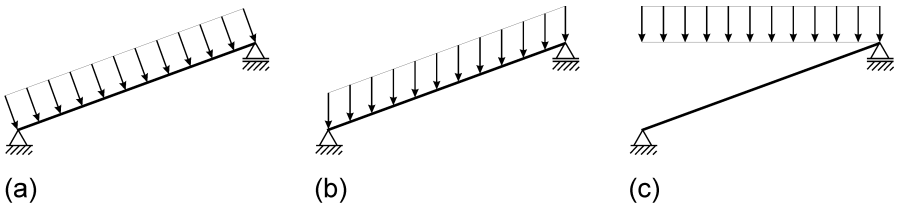

Figure 3.29 Orientation of loads on mesh: (a) local; (b) global; (c) global projected to global plane.

Generation

By default, the "MeshLoad const"-component creates point- and line-loads. The radio-buttons in the submenu "Generation" can be used to disable the first or the latter.

3.2.2 Disassemble Mesh Load

The procedure for distributing mesh-loads on a structure can be computationally heavy. The "Disassemble Mesh Load"-component lets one freeze a mesh-load. It returns the point- and element-loads which were originally generated for reuse with another structure (see fig. 3.30). One has to make sure that the parts of the geometry on which the original mesh-load acted did not change too much. Point-loads are defined using their position, element-loads refer to their elements via element-index. In order to reuse the latter, the corresponding element-indexes of the new and old model need to match.

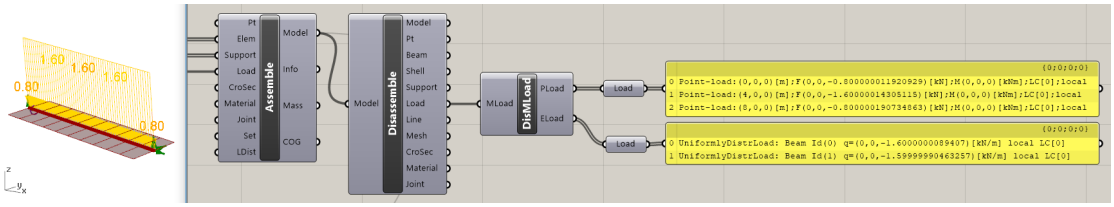

Figure 3.30 The "Disassemble Mesh Load"-component splits mesh-loads into point- and element-loads for further reuse.

3.2.3 Prescribed displacements

Supports as described in section 3.1.16 are a special case of displacement[4] boundary conditions: They set the corresponding degree of freedom of a node to zero. The more general "Prescribed Displacement"-component lets you preset arbitrary displacements at nodes. Figure 3.31 shows a beam with prescribed, clockwise rotations at both end-points.

The "PreDisp"-component resembles the "Support"-component to a large degree: Nodes where displacement conditions apply can be selected via node-index[5] or nodal coordinates. The "Plane"-plug can be used to define an arbitrarily oriented coordinate system for the application of support conditions.

Input-plug "LCase" lets you set the index of the load-case in which displacements shall have a specified value. The default value is "-1" which means that the displacement condition is in place for all load-cases. It is not possible to have displacement boundary conditions active in one load-case and completely disabled in others: For load-cases not mentioned in "LCase" the "PreDisp"-component will act like a simple support with fixed degrees of freedom equal to zero.

The "Trans"- and "Rot"-input-plugs expect vectors. They define nodal translations and rotations either in global coordinates or in the coordinate system defined by the plane fed into the "Plane"-input plug. Translations are to be given in meter (or feet), rotations in degree. The X-component of the rotation vector describes a rotation about the coordinate systems X-axis. A positive value means that the node rotates counter-clockwise if the X-axis points towards you. Analog definitions apply to rotations about the Y- and Z-axis. Karamba3D is partly based on the assumption of small deflections. Thus be aware that large prescribed displacements and rotations give rise to incorrect results in case of

[4] The term "displacement" as used throughout this manual includes translations and rotations.

[5] In order to find out the index of a specific node enable the "node tag"-checkbox in the "ModelView"-component. See section 3.1.6 on how to predefine the index of specific nodes

Figure 3.31 Left: Deflection of a beam under predefined displacements at its end-supports; Right: "PreDisp"-component for setting displacement condition at left support.

geometric linear calculations. For approximating effects due to large displacements see e.g. section 3.5.4.
Displacements can only be prescribed if the corresponding displacement degree of freedom is removed from the statical system. This means you have to activate the corresponding button in the Conditions-section of the "PreDisp"-component. The first three buttons stand for translations the last three for rotations. Only those components of the "Trans"- and "Rot"-vectors take effect which correspond to activated supports.

3.3 Cross Section

Karamba3D offers cross section definitions for beams, shells and springs. They can be generated with the "Cross Sections" multi-component. Use the drop-down list on the bottom to chose the cross section type.
The dimensions of each cross section may be defined manually or by reference to a list of cross sections (see section 3.3.10).

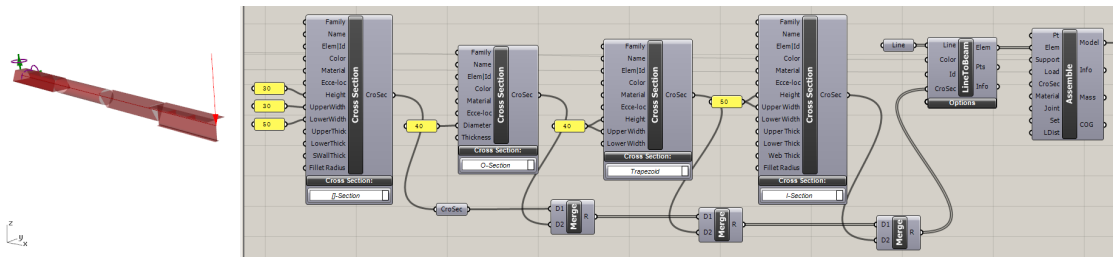

Figure 3.32 Cantilever with four different kinds of cross section.

Cross sections can be plugged directly into the components for creating elements ("LineToBeam", "MeshToShell", ...). Alternatively when fed into an "Assemble"-component (see fig. 3.32) they act on the elements whose identifiers match the string given via "Elem|Id". In case an element is provided at the "Elem|Id"-input, its identifier is used for attaching the cross section to elements. A cross section added via the "Assemble"-component overrides a cross section provided directly at an element-creation-component.
The indirect cross section specification through the "Assemble"-component has the advantage that elements can be specified using regular expressions. Upon assembly all element identifier are com-

pared to the “Elem|Id” entry of a cross section. In case of a match the cross section is attached to the element. An empty string – which is the default value – signifies that the cross section shall be applied to all elements. If two cross sections refer to the same element then that which gets processed later by the assemble-component wins. It makes no sense to attribute beam cross sections to shells and vice versa – Karamba3D ignores any such attempts.

3.3.1 Beam Cross Sections

Karamba3D offers five basic types of beam cross section:

- circular tube – the default
- hollow box section
- filled trapezoid section
- I-profile

Fig. 3.32 shows a cantilever with cross section properties defined directly at the “LineToBeam”-component. Without eccentricities defined, the beam axis always coincides with the centroid a cross section. Changing e.g the upper flange width of an I-section therefore results in a slight movement of the whole section in the local Z-direction. In case the position of e.g. the upper side of a cross section needs to be fixed, specify an eccentricity. This can be done either via a specific component (see section 3.3.7) or through the input-plug “Ecce-loc”. Provide a vector there in order to move the cross sections relative to the beam axis. The given eccentricity is relative to the local coordinate system of the beam. The resulting position of the centroid can be retrieved from the “Disassemble Cross Section”-component (see section 3.3.4).
Apart from the input-plugs that define the cross section geometry, the “Elem|Id”- and the “Ecce-loc”-input there are:

- “Family”: Each cross section belongs to a family. When doing cross section optimization (see section 3.5.8), Karamba3D selects only profiles that belong to the same family as the original section. Families can be composed of arbitrary section types.
- “Name”: The identifier of a cross section – need not be unique. Enable “CroSec names” in ModelViews “RenderSettings”-submenu in order to view them.
- “Color”: Lets one define a color for a cross section. In order to see it enable “Cross sections” in submenu “Colors” of the “ModelView”-component and activate “CroSec section” in submenu “Render Settings” of the “BeamView”-component.
- “Material”: Sets the material of the cross section. Indirect material assignments via the “Assemble”-component override direct definition of the cross section material.

3.3.2 Shell Cross Sections

In Karamba3D there are four different kinds of shell cross sections:

1. “Shell Const”: For shells with constant thickness and material over all elements

2. “Shell Var”: Lets one specify the thickness and material of each element of the shell individually
3. “ShellRC Std Const”: This allows to specify a standard (Std) reinforced concrete(RC) cross section which is constant over the shell.
4. “ShellRC Std Var”: The same as above but lets one choose the reinforced concrete properties differently for each element

Constant and Variable Shell Cross Sections

The components “Shell Const” and “Shell Var” only differ in the data structures expected at the inputs “Material(s)” and “Height(s)”. In case of “Shell Const” these are data items. The “Shell Var”-variant expects two lists. The descriptions below refer to the “Shell Var”-component.
Fig. 3.33 shows a shell consisting of two elements. Triangular meshes form the basis for defining a shell geometry (see section 3.1.9) and specify the sequence of faces (i.e. shell elements). The list of element thicknesses in fig. 3.33 corresponds to that order. Be aware of the fact that meshes containing quads will be automatically triangulated. In case that there are more mesh faces than thickness specifications, the last item (6 cm in this case) acts as the default value. The same holds for the supplied list of materials. Make sure to graft the “Materials”- and “Heights”-input when you want to define a list of shell cross sections. Otherwise one cross section results where one would expect several. For the “Shell Const” definition no data tree manipulation is necessary in such a case.

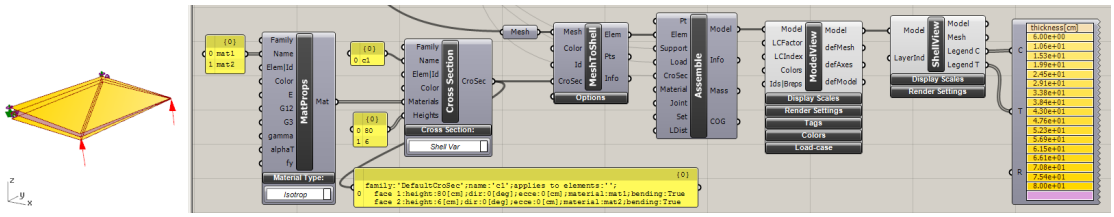

Figure 3.33 Shell made up of two elements with different thicknesses.

When rendering the shell cross sections (see fig. 3.33) thicknesses get linearly interpolated between the nodes. The cross section height at each node results from the mean thickness of shell elements attached to it.
The input-plugs “Family”, “Name”, “Color” and “Materials” have the same meaning as described in section 3.3.1.

Constant and Variable Reinforced Concrete Shell Cross Sections

The design of reinforced concrete cross sections in Karamba3D is based on linear elastic cross section forces. The “Optimize Reinforcement”-component takes these and computes the necessary reinforcement assuming cracked concrete cross sections. Thus defining reinforced concrete cross sections does not alter the mechanical behavior of the structure. They rather serve as input to the reinforcement design procedure.
Similar to shell cross sections there exist two variants of components for reinforced cross sections:

1. “ShellRC Std Const”: For shells with constant height, material and reinforcement. It saves the user thoughts about data trees.
2. “ShellRC Std Var”: This component allows to specify different heights, materials and reinforcement for each element of a shell mesh.

Further below variant two will be explained. The “ShellRC Std Const”-component works similar to the variable-variant. The only difference are the data structures expected at the inputs.

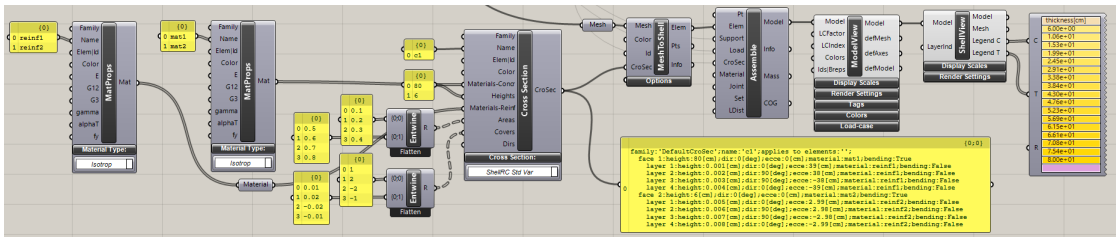

Figure 3.34 Shell made up of two elements with different thicknesses, materials and reinforcement definitions.

Fig. 3.34 shows the definition for a reinforced concrete shell with two elements. The geometry corresponds to that of fig. 3.33. A standard reinforced concrete cross sections consists of five layers: Layer zero is the concrete cross sections. The layers one to four correspond to reinforcement. The top layer (with respect to where the local z-axis points to) comes first, the bottom layer last. Their orientation with respect to layer zero is 0°, 90°, 90° and 0°.
Besides the standard inputs of cross sections (“Family”, “Name”, “Elem|Id” and “Color”) the “ShellRC Std Var”-component offers these.

- “Materials-Concr”: Expects a list of materials to be used for the concrete cross section. The items of this list get mapped to the shell elements according to the longest list principle. “C30/37” according to Eurocode 2 represents the default concrete.
- “Heights”: Heights of the concrete cross sections for each element. The longest list principle applies. The default height is 20 cm.
- “Materials-Reinf”: List of materials to be used as reinforcement for each element – by default “BSt 500” according to Eurocode 2 with a characteristic strength of $50\,\mathrm{kN/cm^2}$. Again the longest list principle applies.
- “Areas”: Expects a data-tree with a maximum of four entries per branch. The values define the minimum reinforcement for each layer. The physical unit is cm. Thus the areas of the reinforcement bars need to be divided by their mutual distance in order to arrive at an equivalent plate thickness. The layer thicknesses default to 0 cm.
- “Covers”: input here a data-tree with four values per branch. These specify the position of the reinforcement layers with respect to the upper and lower side of the concrete cross sections. Positive values give the distance from the upper, negative values the distance from the lower side towards the interior. Without any input the covers default to 3.5 cm, 4.5 cm, −4.5 cm and −3.5 cm.

- "Dirs": Reinforcement layers can be given an angle with respect to the local shell coordinate system. A positive value rotates in anti-clockwise direction about the local z-axis. A value of zero – which is the default – aligns the first and last layer with the local x-axis. The angle of rotation can be specified for each shell element individually.

With the input-plug "LayerInd" of the "ShellView"-component one can select specific layers for visual inspection and results retrieval.

3.3.3 Spring Cross Sections

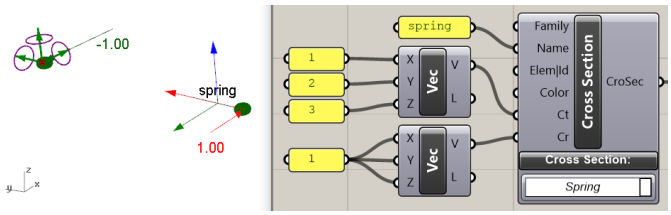

Figure 3.35 Spring fixed at one end and loaded by a point load on the other.

Springs allow you to directly define the stiffness relation between two nodes via spring constants. Each node has six degrees of freedom (DOFs): three translations and three rotations. Using the "Cross Sections" multi-component with "Cross Section" set to "Spring" lets one couple these DOFs by means of six spring-constants. A relative movement $u_{i,rel}$ between two nodes thus leads to a spring force $F_i = c_i \cdot u_{i,rel}$. In this equation $u_{i,rel}$ stands for a relative translation or rotation in any of the three possible directions x, y, z, c_i is the spring stiffness. In Karamba3D the latter has the meaning of kilo Newton per meter $\mathsf{kN/m}$ in case of translations and kilo Newton meter per radiant $\mathsf{kNm/rad}$ in case of rotations. The input-plugs "Ct" and "Cr" expect to receive vectors with translational and rotational stiffness constants respectively. Their orientation corresponds to the local beam coordinate system to which they apply. In case of zero-length springs this defaults to the global coordinate system but can be changed with the "OrientateBeam"-component.
In case one wants to realize a rigid connection between two nodes the question arises as to which spring stiffness should be selected. A value too high makes the global stiffness matrix badly conditioned an can lead to a numerically singular stiffness matrix. A value too low results in unwanted relative displacements. So you have to find out by trial and error which value gives acceptable results. Figure 3.35 shows a peculiarity one has to take into account when using springs: They are unaware of the relative position of their endpoints. This is why the load on the right end of the spring does not evoke a moment at the left, fixed end of the spring.

3.3.4 Disassemble Cross Section

In some cases (e.g. after optimizing cross sections) it may be necessary to retrieve the properties of a cross section. Use the "Disassemble Cross Section"-component for that (see fig. 3.36). Unfold the component sub-sections by clicking on the dark section headers.

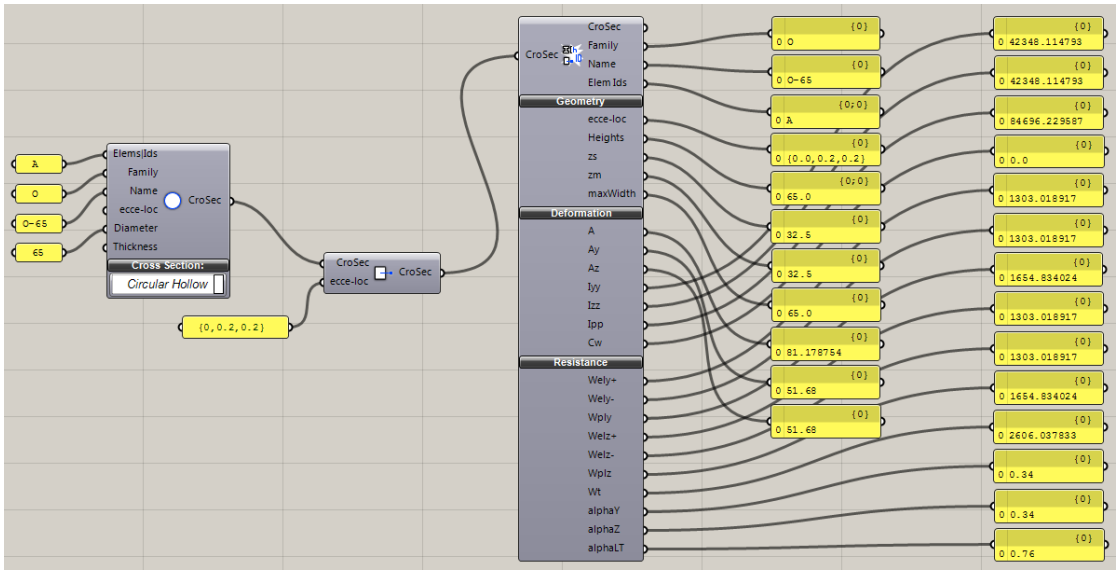

Figure 3.36 Properties of a given cross section can be retrieved via the "Disassemble Cross Section"-component.

3.3.5 Beam-Joint Agent

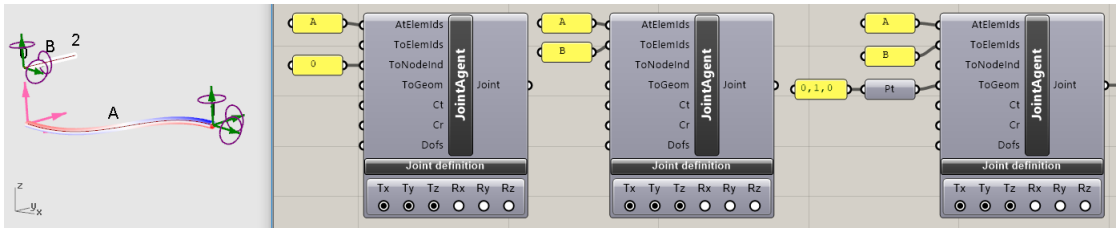

Figure 3.37 Three different but equivalent possibilities for defining a hinge based on geometric relations using a "Beam-Joint Agent"-component.

The "Beam-Joint Agent" creates hinges on beams based on geometric relations. Fig. 3.37 shows three different but equivalent possibilities for defining a joint. The element or the group of elements where the joint(s) shall be placed is set by providing a list element identifiers at the "AtElemsIds" input-plug. Upon assembly the beam-joint agent crawls around in the model and places hinges when one of the following conditions apply:

- The node on the at-element connects to an element whose identifier is listed in the "ToElemIds" input.
- The node on the at-element connects to a node which has a number listed in the "ToNodeInd" input.

- The node on the at-element lies on one of the geometric items supplied in “ToGeom”. This can be points, curves, planes, breps or meshes. The tolerance for two geometric items touching in space is “LDist” as defined on model assembly (see 3.1.1).

The meaning of “Ct”, “Cr” and “Dofs” is analogous to that of the Beam-Joints-component featured in section 3.3.6.

3.3.6 Beam-Joints

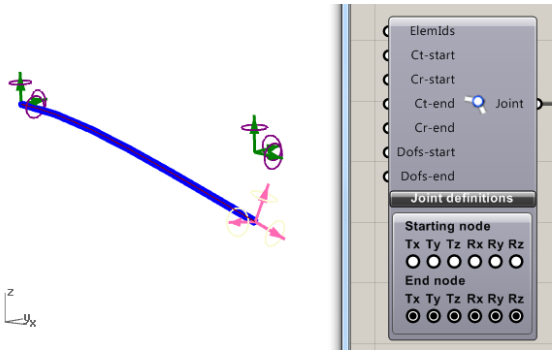

Figure 3.38 Beam under dead weight, fixed at both supports with a fully disconnected joint at one end resulting in a cantilever.

A structure usually consists of a large number of load bearing elements that need to be joined together. When rigidly connected, such a joint has to transfer three section forces (one axial force, two shear forces) and three moments (one torsional and two bending moments). Depending on the type of material such full connections are sometimes (e.g. for wood) hard to achieve, costly and bulky. A solution to this problem consists in introducing hinges.
Figure 3.38 shows a beam under dead weight with fully fixed boundary conditions at both end-points. At the right end the joint (which is in fact no joint any more) completely dissociates the beam from the support there. The result is a cantilever.
The symbols for joints resemble that for supports: pink arrows represent translational joints, white circles symbolize moment hinges. In Karamba3D joints are realized by inserting a spring between the endpoint of a beam and the node to which it connects. This necessitates sufficient support conditions at the actual nodes to prevent them from freely moving around. See for example the right node in fig. 3.38 which has to be fully fixed – otherwise the system would be kinematic.
The “Beam-Joint”-component allows to define hinges at a beam’s starting- and end-node. A list of beam-identifiers lets you select the beams where the joint definition shall apply. Filled circles mean that the corresponding degrees of freedom represent joints. “T” stands for translation, “R” for rotation. Feed the resulting cross-section into the “Joint”-plug of the “Assemble”-component. The orientation of the axes of the joints corresponds to the local coordinate system of the beam they apply to.
Sometimes the stiffness of connections lies between fully fixed and zero. With the input-plugs “Ct-start” and “Cr-start” it is possible to set the stiffness of the hinge in translation ($\mathrm{kN/m}$) and rotation ($\mathrm{kNm/rad}$) respectively at the start of the element. “Ct-end” and “Cr-end” provide the same functionality for the end-point.

In order to make the definition of hinges accessible to optimization the input-plugs “Dofs-start” and “Dofs-end” can be used to set hinges at the beams endpoints with a list of numbers. Integers in the range from 0 to 5 signify degrees of freedom to be released in addition to those specified manually with the radio-buttons.

3.3.7 Eccentricity on Beam, Eccentricity on Cross Section

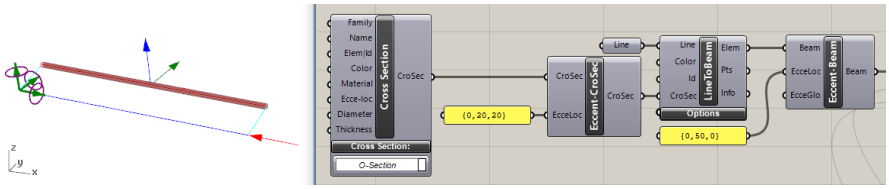

Figure 3.39 Beam positioned eccentrically with respect to the connection line of its two end-nodes.

Cross section forces of beam and truss elements relate to the line that connects the cross section centroids. When a cross section changes, chances are high that also the position of its centroid shifts. In case of elements predominantly loaded by bending moments, such a shift can normally be neglected. In the presence of normal forces however – e.g. when considering columns – changes in the centroid position lead to additional bending moments that may be decisive for a members cross section design.
In Karamba3D there exist two components that can be used to take care of eccentricities (see fig. 3.39): One works on beams, the other on cross sections. When both variants of definition coincide for an element, then they get additively combined. This enables one to define families of cross sections of different size with e.g. the position of their upper sides at one level.
The definition of a local eccentricity for cross sections with a “Eccent-CroSec”-component is straight forward: The “EcceLoc”-input plug expects a vector that defines the offset with respect to the local beam axes. Values are expected in centimeters. “x” represents the longitudinal beam axis, “y” is horizontal, “z” points vertically upwards. Cross sections with eccentricities can be stored in cross section tables using the “Generate Cross Section Table”-component and thus be made reusable in other projects.
The “Eccent-Beam”-component has one additional input-plug as compared to the cross section variant: “EcceGlo” lets one define beam eccentricities (cm) with respect to the global coordinate system.

3.3.8 Modify Cross Section

In Karamba3D cross section properties fall into four categories:

- “General”: parameter which set the name, family, color, material and element identifiers.
- “Geometry”: properties that determine the cross section size and eccentricity
- “Deformation”: these parameters influence the elastic behavior of a structure
- “Resistance”: properties which are used by cross section deign procedures in order to determine the load-bearing capabilities.

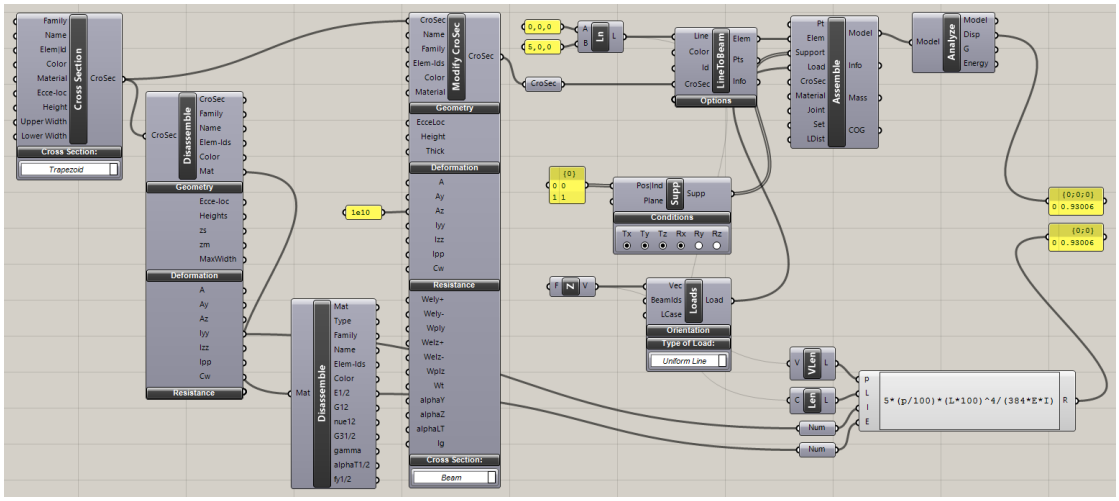

Figure 3.40 A "Modify CroSec"-component can be used to imposes shear rigidity in local z-direction on a cross section. Now the calculated maximum displacement coincides with the result of the formula without shear effects.

The "Modify Cross Section"-component allows to change these properties. Two operation modes exist for this component:

Flow through: When a Cross section is provided as input, the result on the left side is the same cross section by default. Only those properties get changed, for which values are supplied as input.

Agent: The cross sections which shall be modified can be selected via the "Elem-Ids"-input. It is possible to apply regular expressions. The resulting cross section agent is of type "Cross Section" and gets active when being plugged into an "Assemble"-component.

Fig. 3.40 shows the definition of a simply supported beam under uniform load. Textbook formulas for calculating the maximum displacement of such a system usually neglect the influence of shear-deformations. In order to make a cross section nearly rigid in shear the "Modify Cross Section"-component is used to set the shear area A_z to a very large value.
In case the height or thickness of a cross section is changed along with deformation- or resistance parameters, then evaluation proceeds from top to bottom: First, all parameters get updated according to the new cross section dimensions, these may then be overwritten by new values for deformation of resistance properties. The drop-down list at the bottom of the component allows to switch between beam- and shell-cross sections.

3.3.9 Cross Section Range Selector

The cross section library that comes with Karamba3D contains roughly 22 000 profiles. In order to reduce the amount of information the list can be shortened by applying selection criteria on it using the "Cross Section Range Select" component (see fig. 3.41). The input-plugs "maxH" and "maxW" let you limit the list according to maximum cross section height and width. The submenu which unfolds

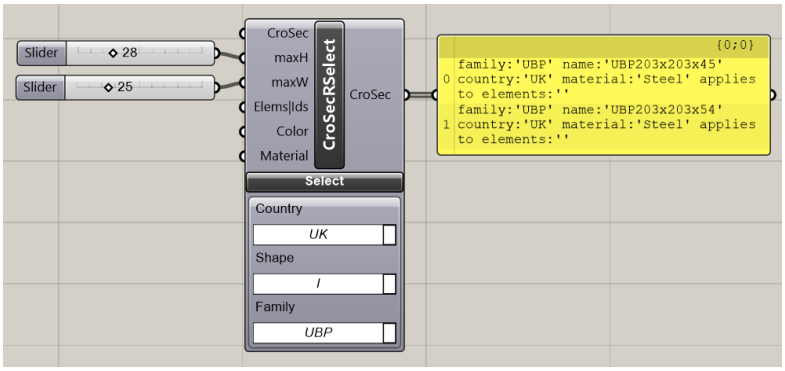

Figure 3.41 Selection of a range of cross sections from among a given list.

when clicking on the black "select"-bar offers further options for narrowing the search: country of origin, general shape and family name.
In case one does not supply a list of cross sections at the "CroSec" input-plug, the cross section table that comes with Karamba3D is used by default.

3.3.10 Cross Section Selector

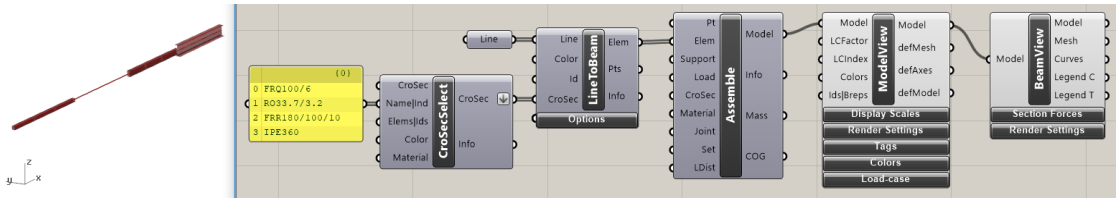

Figure 3.42 Cantilever with four different kinds of cross section taken from the standard cross section table.

The component "CroSecSelect" deals with selecting cross sections by name or index from a list of cross sections. Provide the name(s) or index(es) of desired cross sections in the "Name|Ind"-plug. Cross section names are not case sensitive. All characters coming after "#" count as remark. It is possible to use regular expressions for selection (these start with "&"). Cross section names are case sensitive. List indexes start from zero.
"CroSecSelect" lets you specify beams via the "Elems|Ids"-plug which shall be assigned a specific cross section. The "Assemble"-component sets the cross-sections of elements accordingly. Alternatively, cross sections can be directly plugged into the element-creation-components.
In case one does not supply a list of cross sections at the "CroSec"-input-plug, the cross section table that comes with Karamba is used by default.

3.3.11 Cross Section Matcher

Use the "Cross Section Matcher"-component in case you want to find the first profile from a given list that provides equal or higher resistance compared to a given custom profile (see fig. 3.43). The

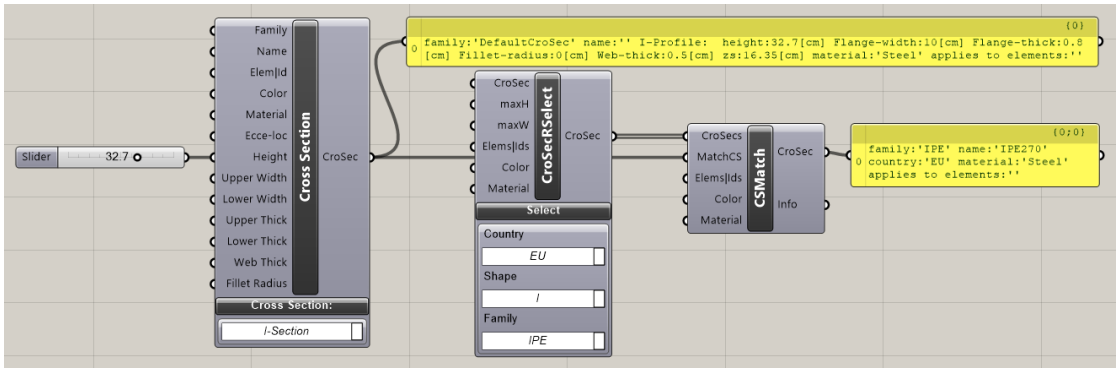

Figure 3.43 The “Cross Section Matcher”-component returning a standard profile for a custom profile.

“CSMatch”-component takes a cross section and a list of cross sections as input. Traversing the list starting from the first element it proceeds until an appropriate profile is found which is returned as the result.

3.3.12 Generate Cross Section Table

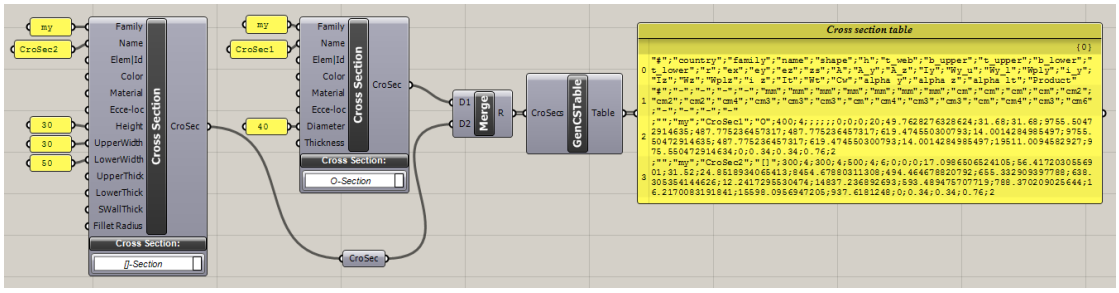

Figure 3.44 Transformation of a list of cross sections to a cross section table.

An entry in a cross section table consists of a row which contains:

- “country”: country of origin
- “family”: name of the group to which the cross section belongs (see section 3.3.1)
- “name”: name of the specific cross section (see section 3.3.1)
- a “shape” field which defines the basic cross section type:
  - “I”: I-section
  - “[]”: hollow box section
  - “V”: trapezoid, filled section
  - “O”: circular tube

  - "S": spring
  - "Sh": shell

- geometric properties which are used for drawing the cross section
- area, moments of inertia, etc. that define the cross sections mechanical behavior. Can be independently defined from the cross section geometry

A "#" in the first column means that the corresponding row serves as a comment.
The "GenCSTable"-component takes a cross section (or a list of cross sections) as input and returns the equivalent table of data as a string. The physical units used for output are always metric. When plugged into a panel the information can be streamed to a file which then constitutes a valid cross section table. Karamba3D reads the data of cross section tables only once. So in order that changes in a table take effect, restart Grasshopper.

Read Cross Section Table from File

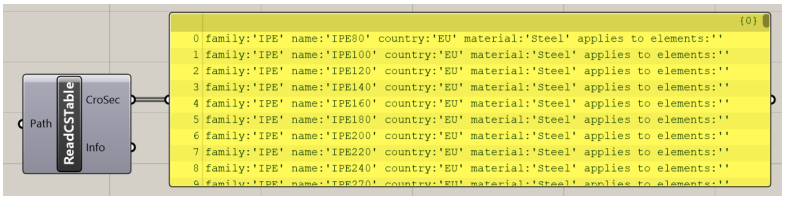

Figure 3.45 List of cross sections generated from the standard cross section table.

Predefined cross sections stored in a csv-database can be used to generate lists of cross sections via the "ReadCSTable"-component (see fig. 3.45). It works along the same lines as the "ReadMatTable" (see section 3.4.3) component. When given no path to a valid csv-table "ReadCSTable" uses the cross section table that comes with Karamba3D and is situated in ".../Grasshopper/Libraries/Karamba/CrossSectionValues.bin". This table contains definitions for a range of standard steel profiles. Depending on the given file extension the data is expected to either be in binary format (".bin") or comma separated values (".csv"). The former has the advantage of fast processing, the latter can be viewed and extended using a text editor or OpenOffice. In csv-files "#" is used to mark the rest of a line as comment. The physical units are always assumed to be metric – irrespective of the user settings at installation. In case of an entry in a csv-file in the first column which is not a "#", the cross section properties get calculated based on the geometric dimensions of the cross section. In case of a deviation between the given and the calculated values of more than 10 % a warning is output at the "Info"-plug.
When opening the Karamba3D installation folder (double-click on the Karamba3D desktop icon for that) you will find three differently named cross section tables: "CrossSectionValues.bin" and "CrossSectionValues_sortedForHeight.bin" contain cross sections sorted according to increasing height. In "CrossSectionValues_sortedForWeight.bin" the area and thus weight per unit of length determines a cross sections relative position within a family. When doing cross section optimization (see section 3.5.8) those two sorting options lead to different results. Depending on external requirements they result in structures of minimum cross section height or structural weight.

3.4 Material

There are two ways for defining materials in Karamba3D: Either select a material by name from a list of materials (see section 3.4.2) or set mechanical material properties manually (see below).
The Appendix (see section A.4.1) contains additional information on mechanical properties of materials.
Materials constitute a property of cross sections. There are two ways of attaching materials to cross sections:

1. In order to directly assign a material to a cross section, plug it into the corresponding cross section creation component. This is overridden any indirect material definitions via the "Assemble" component as described below.

2. Alternatively materials (like cross sections) may be plugged into the "Assemble" component. They know about the elements (or element sets) they apply to by their "Elems|Ids" property: This is a list of strings containing element identifiers (see 3.1.6) or regular expressions that match a group of element identifiers (element-ids). Upon assembly each element-id is compared to all "Elems|Ids" entries of a material. In case they match the material is attached to the element. An empty string – which is the default value – signifies that the material shall be applied to all elements.

3.4.1 Material Properties

The component "MatProps" lets one directly define isotropic and orthotropic materials. Use the drop-down menu at the bottom of the component to chose between ortho- and isotropic materials.

Isotropic Material Properties

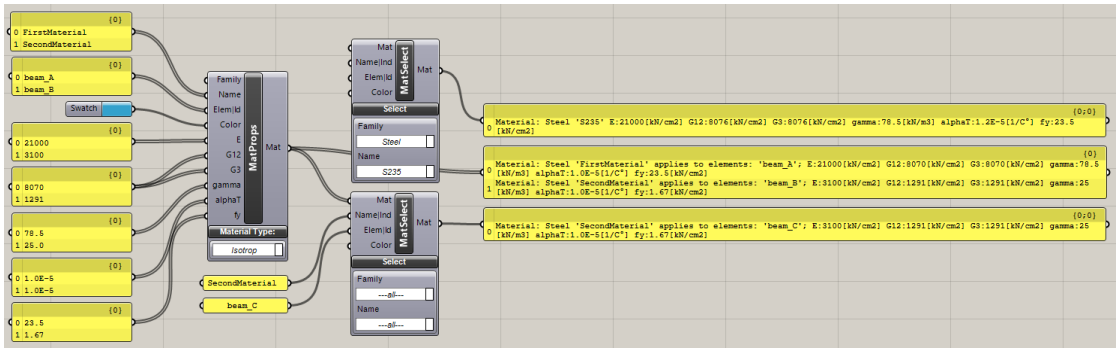

Figure 3.46 The definition of the properties of two isotropic materials via the "Material Properties" component, selection of the second material from the resulting list (mid, bottom) or selection from the default material table (mid, top).

Material isotropy means that the material's behavior does not change with direction. Karamba3D uses the following parameters to characterize an isotropic material (see fig. 3.46):

Family: Family name of the material (e.g. "steel"); is used for selecting materials from a list.

Name: Name of the material (e.g. "S235"); serves as identification when selecting materials from a list.

Elem|Id: An element with an identifier, a string containing an identifier or a regular expression that depicts the elements that shall have the specified material

Color: Color of the material. In order to see it enable "Materials" in submenu "Colors" of the "ModelView"-component, then enable "Cross section" in submenu "Render Settings" of the "BeamView"- and/or "ShellView"-component.

E: Young's Modulus ($\mathsf{kN/cm^2}$): characterizes the stiffness of the material.

G12: In-plane shear modulus ($\mathsf{kN/cm^2}$): In case of isotropic materials the following constraint applies: $E/3 < G12 < E/2$. in case this condition is not fulfilled, the structure may show strange behavior.

G13: Transverse shear modulus ($\mathsf{kN/cm^2}$): Is the same as $G12$ in case of isotropic materials like e.g. steel. This value can be chosen independently from E. In case of e.g. wood, the value may be much smaller than $G12$.

gamma: Specific weight ($\mathsf{kN/m^3}$)

alphaT: Coefficient of thermal expansion ($\mathsf{1/°C}$)

fy: Yield stress $\mathsf{kN/cm^2}$ - the material strength.

The yield stress characterizes the strength of a material. The utilization of cross sections as displayed by the "BeamView"-component (see section 3.6.7) is the ratio of actual stress and yield stress. In case of shells, utilization is determined as the ratio of Van Mises Stress (as computed from the stresses in the shell) and yield stress (see section 3.6.11). Cross section optimization (see section 3.5.8) also makes use of the materials yield stress.
In case of temperature changes materials expand or shorten. "alphaT" sets the increase of strain per degree Celsius of an unrestrained element. For steel the value is $1.0E-5$ ($1.0E-5 = 1.0 \cdot 10^{-5} = 0.00001$). Therefore an unrestrained steel rod of length $\mathsf{10\,m}$ lengthens by $\mathsf{1\,mm}$ under an increase of temperature of $\mathsf{10\,°C}$. "alphaT" enters calculations when temperature loads are present.

Orthotropic Material Properties

Material orthotropy means that the material's behavior changes with direction. The material properties in two orthogonal directions fully characterize any orthotropic material. In Karamba3D orthotropic materials take effect only in shells. When supplied to beams, the material properties in the first direction are applied. For shells the first material direction corresponds to the local x-axis. See section 3.1.14 on how to set user defined local coordinate systems on shells.
In fig. 3.47 an orthotropic material gets defined using a "Material Property"-component. Besides "Family", "Name", "Elem|Id" and "Color" it expects the following input:

E1: Young's Modulus in first direction ($\mathsf{kN/cm^2}$)

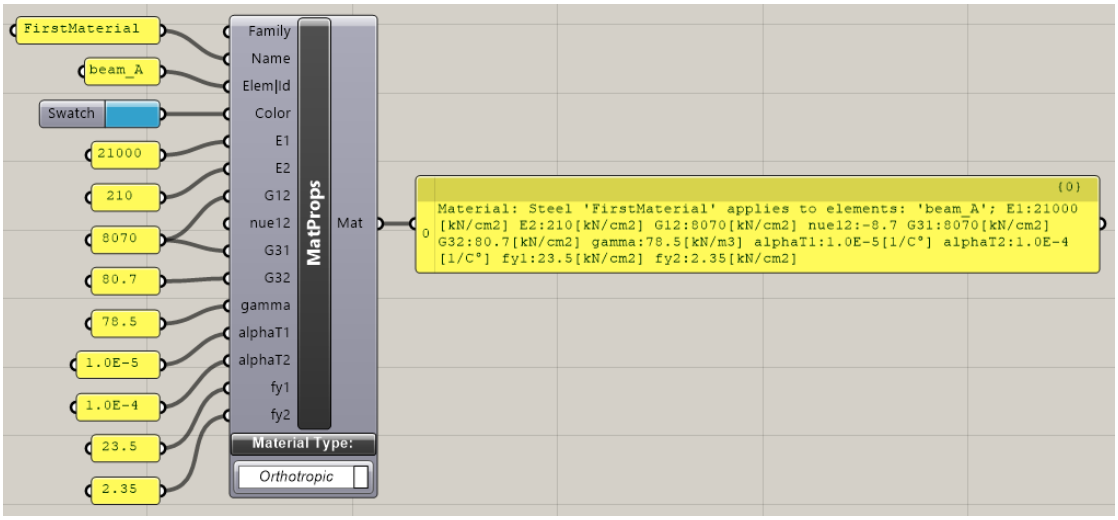

Figure 3.47 The definition of properties of an orthotropic material via the "Material Properties" component.

E2: Young's Modulus in second direction ($\mathrm{kN/cm^2}$)

G12: In-plane shear modulus ($\mathrm{kN/cm^2}$): The value of G_{12} is liable to a constraint which is further depicted below.

nue12: ν_{12} is the in-plane lateral contraction coefficient (also called Poisson's ratio): In case $\nu_{12} = -1$ (the default) the approximate formula of Huber [8] is applied to calculate ν_{12} from E_1, E_2 and G_{12}:

$$\nu_{12} = \frac{E_1}{2 \cdot G_{12}} - \sqrt{\frac{E_1}{E_2}}$$

For ν_{12} the constraint $|\nu_{12}| < \sqrt{\frac{E_1}{E_2}}$ applies. In case of $\nu_{12} = -1$ this limits the possible range of value of G_{12}.

G31: Transverse shear modulus in the first direction ($\mathrm{kN/cm^2}$)

G32: Transverse shear modulus the in second direction ($\mathrm{kN/cm^2}$)

gamma: Specific weight ($\mathrm{kN/m^3}$)

alphaT1: Coefficient of thermal expansion in the first direction ($1/°\mathrm{C}$)

alphaT2: Coefficient of thermal expansion in the second direction ($1/°\mathrm{C}$)

fy1: Yield stress in the first direction $\mathrm{kN/cm^2}$

fy2: Yield stress in the second direction $\mathrm{kN/cm^2}$ is not used at the moment.

3.4.2 Material Selection

The “Material Selection”-component in the menu subsection “Material” lets you select a material by family, name or index from a given list of materials (see fig. 3.46 or fig. 3.49). Input the list of materials via the plug “Mat”. In case no list of materials is supplied, the material table that comes with Karamba3D is used.
The input-plug “Name|Ind” expects either the zero-based list index of the selected material or its name. The names of materials are not case sensitive. A “#” in a material name means that the rest of the line is a comment. “&” starts a regular expression – in that case material names are case sensitive.
For quick access materials may be selected via the drop down lists “Family” and “Name”, which unfold when clicking on the components “Select” bar. These two entries serve as additional criteria which act on the list of materials selected through the “Name|Ind” input.
The inputs “Elem|Id” and “Color” have the same meaning as in the “Material Properties”-component (see section 3.4.1). Any element identifiers already present in a material get overwritten by the values input via “Elem|Id”. Without a color supplied at input “Color” the original material color persists.

3.4.3 Read Material Table from File

	A	B	C	D	E	F	G	H	I	J	K
1	#				E	G_in-plane	G_transverse	gamma	alphaT	fyk	color
2	#	family	name	type	[kN/cm2]	[kN/cm2]	[kN/cm2]	[kN/m3]	[m/(mC°)]	[kN/cm2]	
3		Steel	Steel	iso	21000	8076	8076	78.5	1.20E-05	23.5	Blue
4		Steel	S235	iso	21000	8076	8076	78.5	1.20E-05	23.5	Blue
5		Steel	S275	iso	21000	8076	8076	78.5	1.20E-05	27.5	Blue
6		Steel	S355	iso	21000	8076	8076	78.5	1.20E-05	36.	Blue
7		Steel	St37-2	iso	21000	8076	8076	78.5	1.20E-05	24.	Blue

Figure 3.48 Partial view of the default data base of materials. SI units are used irrespective of user settings. Automatic conversion ensures compatibility with Imperial units.

Karamba3D comes with a table of predefined materials. The csv-file “Materialproperties.csv” resides in the Karamba3D installation-folder. By default the “ReadMatTable”-component takes this file and creates a list of materials from it. These are available at the output-plug “Material”. The data-base currently holds properties for “steel”, “concrete”, “wood” and “aluminum”. There exist different types of steel, concrete etc.. The generic term “concrete” will result in the selection of an everyday type of concrete - a C25/30 according to Eurocode 2. More specific descriptions may be given: Have a look at the data-base in order to get an overview. Material properties specified via table are assumed to be in SI units. They get automatically converted when used in the context of Imperial units.
Fig. 3.48 shows examples of how to define isotropic materials via table. In case of orthotropic materials, the item in column “D” needs to be set to something different from “iso”. The order of orthotropic material parameters which follow from column “E” onward correspond to that of the “Material Property”-component: E_1, E_2, G_{12}, ν_{12}, G_{31}, G_{32}, γ, α_{T1}, α_{T2}, f_{y1}, f_{y2} and “Color”.
The extension .csv stands for “comma separated value”. The file can be opened with any text editor and contains the table entries separated by semicolons. It is preferable however to use OpenOffice or Excel (both can read and write csv-files): They render the data neatly formatted (see fig. 3.48). Make sure to have a “.” and not a “,” set as your decimal separator. In some countries “.” is used to separate thousands which then needs to be adapted as well. The setting may be changed under Windows via

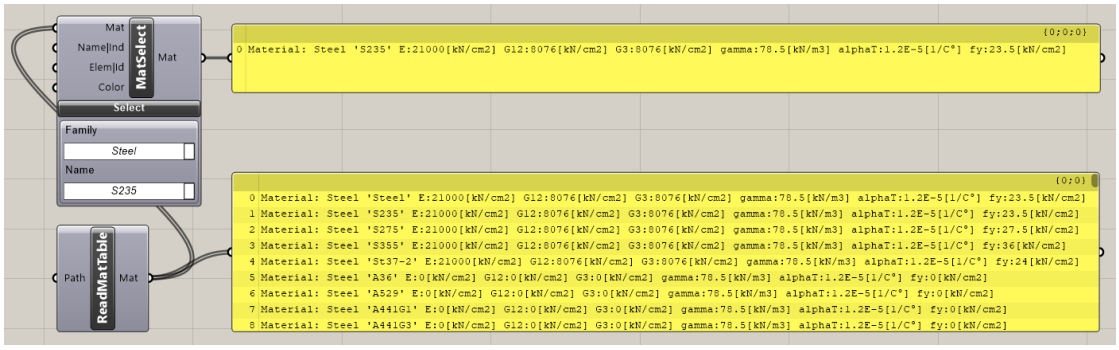

Figure 3.49 List of materials resulting from the "ReadMatTable"-component reading the default data base of materials. Selection of the default "Steel" via "MatSelect".

"regional settings" in "system settings". All lines in the table that start with "#" are comments. Feel free to define your own materials.
The file path to the materials data-base can be changed in two ways: first right-click on the component and hit "Select file path to material definitions" in the context menu that pops up. Second plug a panel with a file path into "Path". Relative paths are relative to the directory where your definition lies.

3.4.4 Disassemble Material

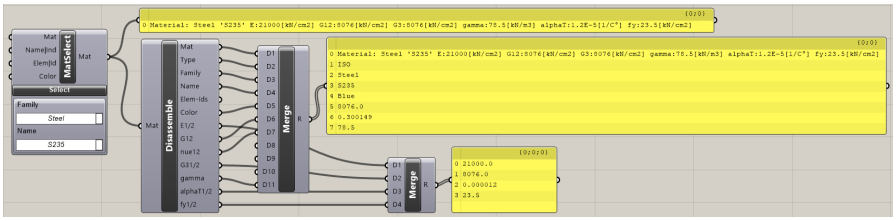

Figure 3.50 The "Disassemble Material"-component gives access to all material properties.

In case one wants to retrieve the parameters which define a material, the "Disassemble Material"-component does the job (see fig. 3.50). It can be applied to isotropic and orthotropic materials alike. In order to be valid for both types of materials, the outputs "E1/2", "G31/2", "alphaT1/2" and "fy1/2" return lists of numbers instead of single items. For isotropic materials these lists contain one member only. In case of orthotropic materials two numbers are present, corresponding to the first and second material direction respectively.

3.5 Algorithms

3.5.1 Analyze

With geometry, supports and loads defined, the statical model is ready for processing. The “Analyze”-component computes the deflection for each load case and adds this information to the model.
The algorithm behind the “Analyze”-component neglects the change of length in axial or in-plane direction which accompanies lateral deformations. This is justified in case of displacements which are small with respect to the dimensions of a beam of shell. For dealing with situations where this condition does not hold, geometric non-linear calculations need to be used (see sections 3.5.3 and 3.5.4).
In case of the presence of second order normal forces (N^{II}, see below) their influence on structural stiffness is taken into account. Those N^{II}-forces do not get updated by the “Analyze”-component. Use the “AnalyzeThII” for that.

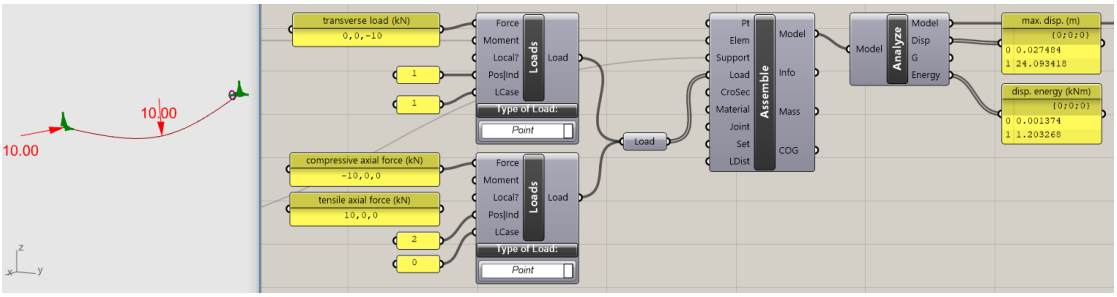

Figure 3.51 Deflection of simply supported beam under single load in mid-span and axial, compressive load.

Figure 3.51 shows a deflected beam with two load-cases. An axial load acts in load-case zero, a transverse load in mid-span in load-case one.
The analysis component not only computes the model deflections but also outputs the maximum nodal displacement (in centimeter), the maximum total force of gravity (in kilo Newton, if gravity is set) and the structures internal deformation energy of each load case - see section 3.6.2 for details on work and energy.
These values can be used to rank structures in the course of a structural optimization procedure: the more efficient a structure the smaller the maximum deflection, the amount of material used and the value of the internal elastic energy. Real structures are designed in such a way that their deflection does not impair their usability. See section A.4.3 for further details. Maximum deflection and elastic energy both provide a benchmark for structural stiffness, yet from different points of view: The value of elastic energy allows to judge a structure as a whole; The maximum displacement returns a local peak value.
In order to view the deflected model use the “ModelView”-component (see section 3.6.1) and select the desired load case in the menu “Result Case”.
Looking at figure 3.51 one immediately notices that only beam center axes are shown. In order to see beams or shells in a rendered view add a “BeamView”- or “ShellView”-component after the “ModelView”. See sections 3.6.7 and 3.6.11 for details.

3.5.2 AnalyzeThII

Axial forces in beams and in-plane forces in shells influence the structures stiffness. Compressive forces decrease a structure's stiffness, tensile forces increase it. The influence of compressive forces on displacements and cross section forces may be neglected as long as their absolute value is less than $10\,\%$ of the buckling load.

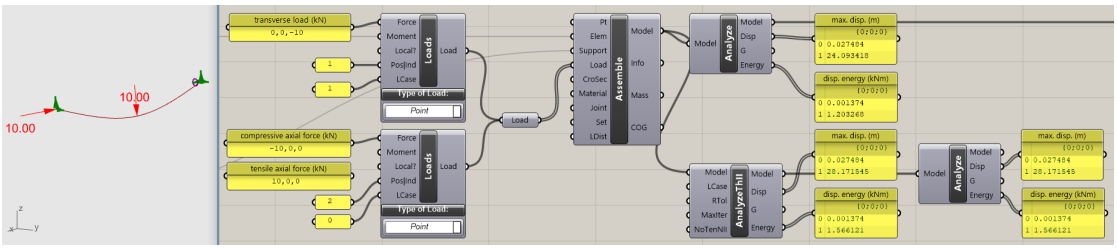

Figure 3.52 Deflection of simply supported beam under single load in mid-span and axial compressive load. Comparison of first and second order theory results.

In Karamba3D distinction is made between normal forces N which cause stresses in the members and normal forces N^{II} which result in second order effects (see also [10]). At first sight this concept seems weird. How can there be two kinds of normal forces in the same beam? Well, in reality there can't. In a computer program it is no problem: stresses get calculated as $\sigma = N/A$ and N^{II} is used for determining second order effects only. The advantage is, that in the presence of several load-cases one can chose for each element the largest compressive force as N^{II}. This gives a lower limit for the structures stiffness. A reevaluation of the load-cases using these N^{II} values leads to a structural response which is too soft. However the different load-cases may then be safely superimposed.
Use the "AnalyzeThII"-component for automatically determining the normal forces N^{II} from cross section forces N_x. N^{II} influences a structures stiffness which in turn impacts the distribution of cross section forces N_x. Thus an iterative procedure with repeated updates of N^{II}-forces needs to be applied.
Fig. 3.52 shows the same system as in fig. 3.51. This time with results according to first and second order theory. When comparing the transverse deflections in load-case two one can see that the maximum deflection increased from $0.24[m]$ to $0.28[m]$ due to the effect of the axial compressive load.
The "AnalyzeThII"-component features the following input-plugs:

Model: Model to be considered.

LC: Number of load-case from which to take the normal force N^{II} which cause second order theory effects. If set to -1 (the default) the minimum normal force of all load-cases is considered.

RTol: The determination of N^{II} is an iterative process. The value of "RTol" is the upper limit of displacement increments from one iteration to the next.

MaxIter: Supply here the maximum number of iterations for determining N^{II}. The default is 50. In case "RTol" can not be reached within the preset number of iterations the component turns orange.

NoTenNII: Tension forces increase the stiffness of a statical system. Setting "NoTenNII" to "True" limits N^{II} to non-positive values.

The normal forces N^{II} get attached to the model and will be considered in all further analysis steps. They impact the results of the "Analyze"-, "Buckling Modes"-, "Natural Vibrations"- and "Optimize Cross Sections"-components. For imperfection loads N^{II}-forces have a direct impact on the applied loads.
Use the "NII" button in submenu "Tags" of the "ModelView"-component to display N^{II}-forces.

3.5.3 Analyze Nonlinear WIP

Linear structural behavior means that if one changes the external loads by a factor f also the physical response quantities (displacements, cross section forces, stresses, ...) change by that factor. This has the pleasant effect, that the impact of different loads can be superimposed. Thus it is not necessary to recalculate the model for each possible combination of external loads. For real structures the assumption of linear behavior is an approximation – a good one in many cases. There are two mayor sources of non-linearity:

- Physical non-linearity: Comes into play when materials leave the linear elastic range (e.g. concrete that cracks, steel that yields, ...)
- Geometric non-linearity: Takes effect when
  - lateral displacements get so large, that their effect on the axial (in case of e.g. beams) or in-plane (think of shells) deformation can not be neglected any more,
  - a nodal rotation α reaches such a value, that the difference between α and $\tan(\alpha)$ gains importance.

The "Analyze Nonlinear WIP"-component lets one deal with geometric non-linearity. It is work-in-progress. This means that especially for shells the algorithms may not converge within acceptable time for some structures. If however a result is returned, then it should be sound.
With the "Analyze Nonlinear WIP"-component one can chose from three variants of iterative solution algorithms. Each of these has different benefits and liabilities which will be explained below. The algorithms are based on the assumption of small strains, but allow arbitrarily large displacements.
The target of all three algorithms is to find a displacement state, where the external loads and the internal forces are in equilibrium. Starting from a known initial displacement state, one has to guess how the structure deforms under the given loads. This guess leads to a second displacement state where the internal and external forces usually do not match. The remaining imbalance forms the basis of a next prediction regarding the change of displacements and so on. Equilibrium is reached when the residual-force or change of displacements falls below a given threshold. The three algorithms offered by the "Analyze Nonlinear WIP"-component differ in how they predict the displacement increments.

Dynamic Relaxation

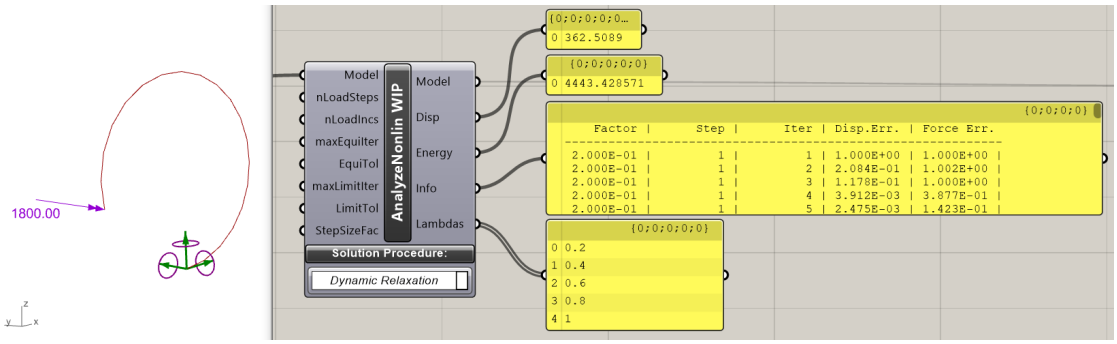

Figure 3.53 large displacement of cantilever beam with moment-load at its tip computed using the dynamic relaxation method option of the "Analyze Nonlinear WIP"-component.

Fig. 3.53 shows a cantilever beam with a bending moment load about the local y-axis at its tip. It consists of 20 beam elements. For calculating its response the "DynamicRelaxation"-option is used. This algorithm predicts the next move of a structure based on the direction of the residual forces acting on each node. It is a robust procedure which converges to equilibrium quite reliably but sometimes needs a large number of iterations to do so. This component offers the following input-plugs:

Model: Structure to be analyzed.

nLoadSteps: Number of load-cases which shall act as load-steps. The default is 1. When setting it to e.g. 2 it means that the algorithm starts with finding equilibrium for load-case 0. After that, the loads of load-case 0 remain in place and the loads of load-case 1 are added in order to arrive at the final stage. This allows to model a loading history. The remaining load-cases get added to the final stage separately and under the assumption of small displacements. In the case of an active bending structure the first load-cases serve as those which cause the deformed structure, whereas the rest of the load-cases constitute additional actions on the deformed configuration like wind- or live-load. The scaling factor for displacements in the "Display Scales" submenu of the "ModelView"-component acts only on the loading-steps for which small displacements are assumed. The large deformation share of the total displacements does not get scaled and is displayed in real size.

nLoadIncs: Number of increments per load-case (the default is 5). External loads get divided into several steps. In case of structures with nearly linear behavior, the number of increments can be set to a small number. For highly non-linear problems a larger value can be advantageous. The smaller the load-increments, the easier it is then for the algorithm to find equilibrium. The number of iterations usually decreases with increasing number of load-steps (and thus decreasing step size). The overall performance can however suffer if the number of load-steps is set to a number which is too high for the given type of structural behavior.

maxEquilter: Sets the maximum number of equilibrium iterations per load-increment and thus sets a limit on computation time. It defaults to 200.

EquiTol: Tolerance for the iterative change of residual forces and displacements relative to their incremental change in the current load-step. The default value is $1E-7$.

maxLimitIter: The range of problems which can be tackled using the dynamic relaxation (DR) algorithm as implemented in Karamba3D is limited to stable structures. In case of phenomena like buckling or snap-through, equilibrium states may exist beyond the point of initial instability. They are however hard to reach due to their often large distance from the last known stable configuration. In such a case the DR-algorithm does not converge to an equilibrium state within the maximum number of equilibrium iterations. It then tries to close in on the point of assumed instability by halving the load-increment which lead to divergence. By proceeding in this manner, the so called limit load can be determined with arbitrary precision. "maxLimitIter" sets an upper limit on the number of limit-load-iterations which is equal to 200 by default. Sadly, divergence can also be caused by numerical problems in the algorithm. Thus the limit-load-factor as determined by the "Analyze Nonlinear WIP"-component constitutes only a lower limit estimation.

LimitTol: Sets the minimum load-increment for calculating the limit-load.

StepSizeFac: A factor for scaling the predicted displacement increments of the DR-algorithm.

During a non-linear calculation lots of things can happen. In order to get an idea about why and where something went wrong, the DR variant of the "DynamicRelaxation"-option produces the following output:

Model: Structure with calculated displacements, stresses and internal forces.

Disp: Maximum displacement reached in centimeter.

Energy: Deformation energy stored in the structure in kN m.

Info: Details regarding the solution process. It outputs five columns of text:

- "Factor": The factor of the loads of the current load-step.
- "Step": The current load-step
- "Iter": Counts the number of iterations for the current load-increment.
- "Disp.Err": Outputs the ratio of the sum of iterative changes of the nodal displacements with respect to the change of displacements of the first iteration in the current load-increment
- "Force.Err": Outputs the ratio of the sum of iterative changes of the residual forces with respect to the current load-increment.

Lambdas: Informs about the load-factors for which equilibrium could be reached.

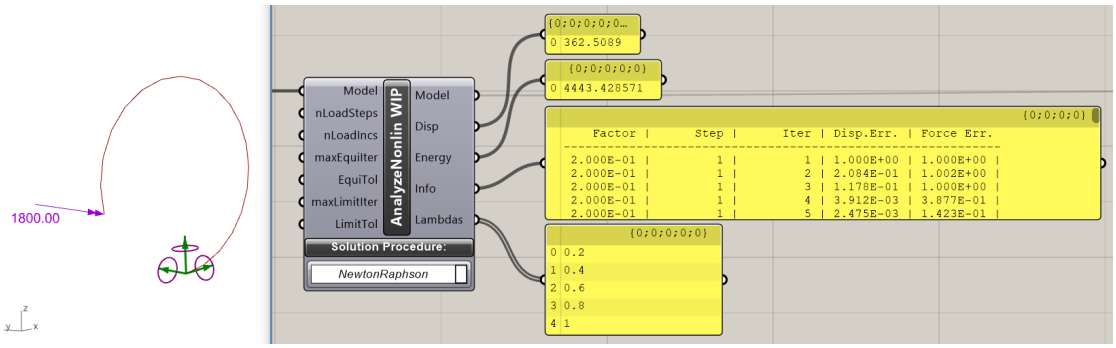

Figure 3.54 large displacement of cantilever beam with moment-load at its tip computed using the Newton-Raphson method option of the "Analyze Nonlinear WIP"-component.

Newton-Raphson Method

In practice, dynamic relaxation(DR) procedures are used for highly non-linear problems like numerical crash-tests of cars, bolts being shot into a wall, ... The reason is, that implementing non-linear effects as DR code is relatively easy. This ease of implementation comes at the cost of high computational effort: Many iterations are necessary to reach equilibrium with acceptable accuracy. The way out of this is to invest more effort in a better prediction of the displacement increments. In DR-methods the residual forces at the nodes form the basis of predicting the next position of a node. Methods like the Newton-Raphson- or Arc-Length-method use a stiffness matrix for producing displacement predictions. There the computational cost per iteration is higher, but the number of iterations can be made much smaller as compared to DR-methods. With a consistent stiffness matrix quadratic convergence can be achieved under optimal conditions. This means that for the iterative displacement- and force-errors the number of zeros after the decimal separator doubles in each iteration. For the "Analyze Nonlinear WIP"-component this is not yet the case and one reason for the "work in progress"-label. Details on the Newton-Raphson- or Arc-Length methods can be found in [6] on page 102 ff. and 214 ff..
Fig. 3.54 shows the same cantilever beam as before, this time analyzed with the "NewtonRaphson"-option. The Newton-Rahpson variant of the "Analyze Nonlinear WIP"-component comes with nearly the same input- and output-plugs as the DR-version. The only difference is the missing "StepSizeFac"-input. Since Newton-Raphson procedures have the same limitation with respect to unstable structures as DR-methods, an interval halving strategy for closing in on limit-points is applied as before.

Arc-Length Method

For many structures reaching a first point of instability is not yet the end of the story. Especially thin plate and shell structures show large load bearing reserves when considering their post-buckling behavior. The Arc-Length-method can be used for these kinds of situations. Fig. 3.55 shows the calculation of a truss structure which snaps through from an unstable state to a stable post-buckling configuration.
The first two inputs of the "Arclength"-component have the same meaning as before. Here a description of how the rest of the input-plugs controls the solution process:

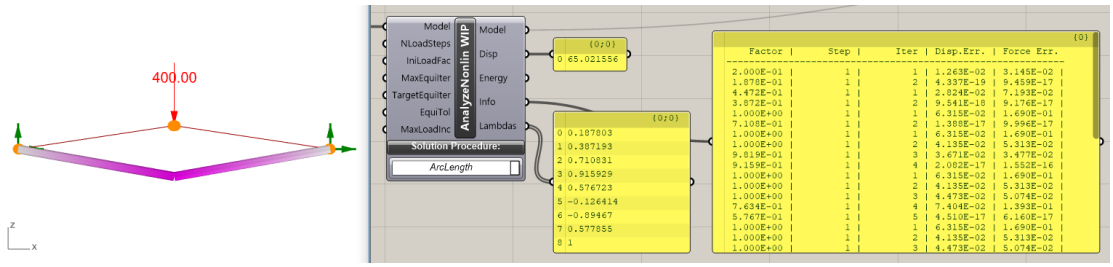

Figure 3.55 snap through of a structure consisting of two truss elements with default cross sections and materials using the Arc-Length method option of the "Analyze Nonlinear WIP"-component.

IniLoadFac: The displacement of the structure under the external load multiplied by "IniLoadFac" serves as a first estimate for the target deformation increments.

MaxEquilter: The maximum number of iterations per load increment.

TargetEquilter: Sets the number of increments to be used in each increment. Is used to scale the load-increments acordingly.

EquiTol: The tolerance for out of balance forces and displacement changes from one iteration to the next.

MaxLoadInc: Maximum number of load iterations.

3.5.4 Analyze Large Deformation

Before the advent of digital modeling people like Heinz Isler, Antoni Gaudi or Sergio Musmeci helped themselves with physical models for generating curved geometries. A popular method was to use the shape of meshes or elastic membranes hanging from supports.

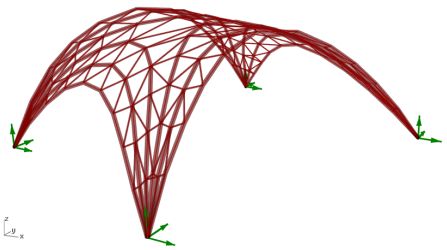

Figure 3.56 Structure resulting from large deflection analysis with the "LaDeform"-component.

In Karamba3D the behavior of hanging models can be simulated with the help of the "Analyze Large Deformation"-component. Figure 3.56 shows a geometry derived from an initially flat mesh under evenly distributed point-loads. The algorithm behind the "Analyze Large Deformation"-component handles geometric non-linearity by an incremental approach only: All external loads get applied in steps. After each step the model geometry updates to the deflected state. The more and the smaller

the steps, the better the approximation of geometric non-linearity. This purely incremental method however incurs an unavoidable drift from the exact solution. For form-finding this error is negligible in most cases. The methods available under the "Analyze Nonlinear WIP"-component (see section 3.5.3) do not suffer from this lack of accuracy, since they apply an incremental-iterative approach. Yet they normally require more computational effort to arrive at a similar shape as the algorithm behind the "Analyze Large Deformation"-component.

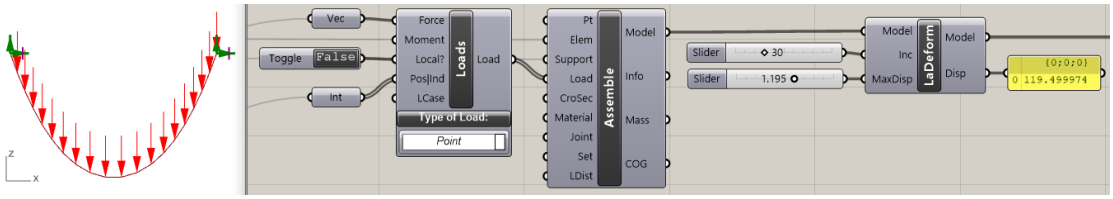

Figure 3.57 Catenary resulting from point loads that do not change their direction when displaced.

Figure 3.57 shows a simply supported beam under the action of uniformly distributed point loads. Due to its slenderness axial stiffness by far outweighs bending stiffness. Thus the deflected shape corresponds to a rope under self weight.
The "LaDeform" component has four input-plugs:

Model: Structure to be deformed. "LaDeform" uses load-case 0 for calculating the deflected shape.

Inc: Number of increments for applying the loads.

MaxDisp: Maximum displacement to be reached in meter. When supplied with a value, the incremental deflection in each step is scaled to $MaxDisp/Inc$. This enables Karamba3D to handle problems with overly large deflections at the beginning of the incremental procedure. Think of an initially straight rope: Due to its negligible bending stiffness it tends to deform tremendously in the first loading step.

With no value supplied in "MaxDisp" external loads get incremented proportionally in each step. Aside from cases like mentioned above this results in an approximation of the structures real deflections under the given loads.

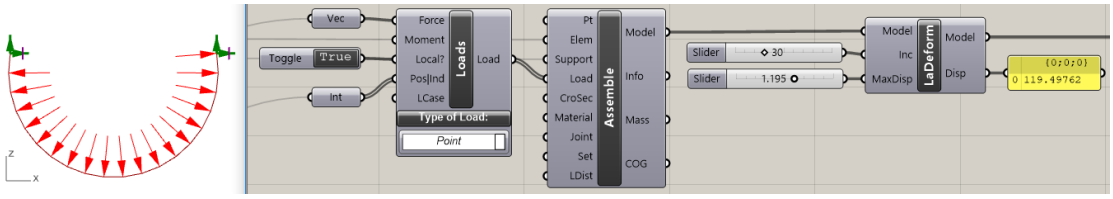

Figure 3.58 Pneumatic form resulting from point loads that rotate along with the points they apply to.

In fig. 3.57 the point loads are defined with respect to the global coordinate system: The input-plug "Local?" at the point-load component is set to "False". Fig. 3.58 shows what happens if one

changes that property to "True": The point-loads co-rotate with the points they apply to. This leads to a pneumatic shape. The same happens for locally defined line-loads.
The two output plugs of the "LaDeform"-component supply the deflected model and the maximum deflection reached in the calculation.
The local coordinate system of each element gets updated along with its positions. By default an elements local Y-axis is taken parallel to the global X-Y-plane. If an element reaches a vertical position however, its default coordinate system flips – the Y-axis is then taken parallel to the global Y-axis. This may lead to unwanted results when using line-loads which flip along with the local coordinate system. It is possible to avoid this by defining local axes via the "OrientateBeam"-component.
The deflected model contains no information regarding internal forces or stresses. The reason for this is that, owing to the purely incremental approach, these properties would be utterly inaccurate.

3.5.5 Buckling Modes

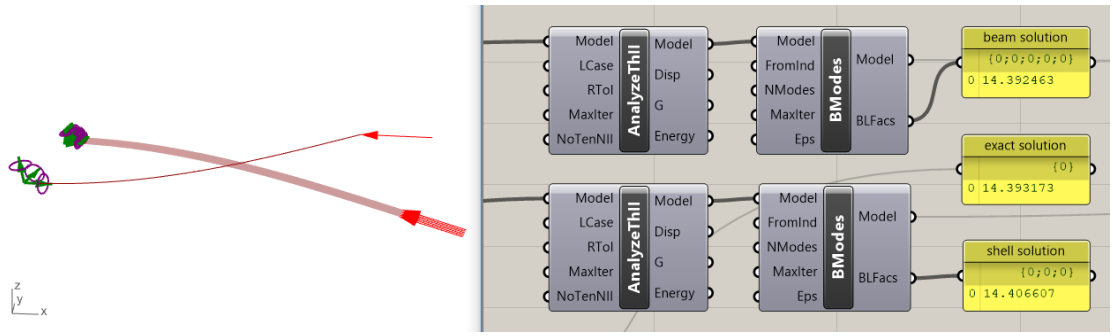

Figure 3.59 Shape and load-factors of the first buckling mode of a cantilever analyzed as a beam and shell.

Axial forces in beams and trusses as well as in-plane forces in shells change the elements response under transverse load. Tension makes them stiffer, compression has a softening effect.
Slender columns or thin shells may fail due to buckling before the stresses in the cross section reach the material strength. Stability analysis therefore plays an important role in structural design.
When doing cross section optimization with the "Optimize Cross Section"-component, the design formulas applied take account of buckling, based on the buckling length of the members. By default local buckling of individual elements is assumed. So-called global buckling occurs if a structural sub-system consisting of several elements (like e.g. a truss) loses stability. Global buckling can be checked with the "Buckling Modes"-component (see fig. 3.59).
The "Buckling Modes"-component expects these input parameters:

Model: Structure with second order normal forces N^{II} defined. These forces can either be taken from a second order theory calculation (like in fig. 3.59) or specified via a "Modify Element"-component.

FromInd: Index of the first buckling mode to be determined. The default is 1. This is also normally the only buckling shape of interest, since it corresponds to the mode of failure.

NModes: Number of buckling modes to be calculated. The default is 1.

MaxIter: The determination of the buckling modes is an iterative procedure. "MaxIter" sets the maximum number of iterations.

Eps: : Represents the convergence criteria. For convergence the iterative change of the norm of the displacements needs to fall below that value.

The model which comes out on the right side lists the computed buckling-modes as result-cases. The buckling shapes get scaled, so that their largest displacement component has the value 1. "BLFacs" returns the buckling load factors which are assumed to be non-negative. When multiplied with those factors the current normal forces N^{II} would lead to an unstable structure. The buckling load factors are listed in ascending order. The calculation of buckling factors assumes small deflections up to the point of instability. This may not always be the case.

3.5.6 Eigen Modes

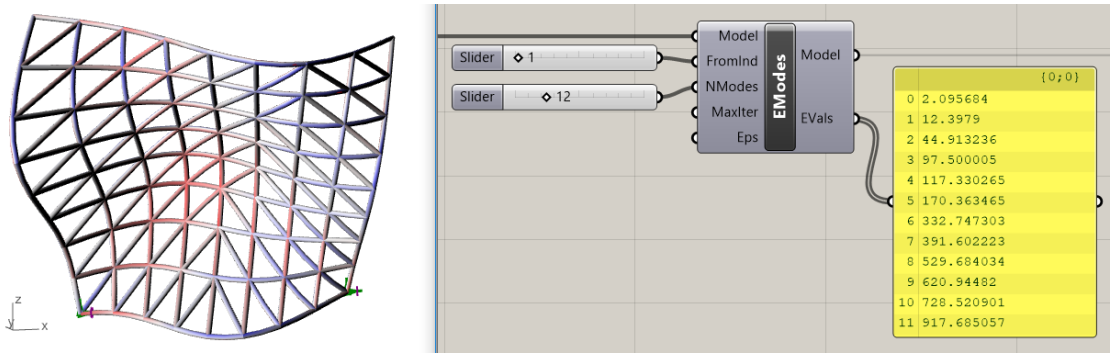

Figure 3.60 Left: 14^{th} eigen-mode with strain display enabled. Right: EigenMode-component in action.

Karamba3D's "EigenMode"-component allows to calculate eigen-modes and corresponding eigen-values of structures (see figure 3.60).
The input parameters are a model, the index of the first eigen-mode to be computed and the number of desired eigen-modes. The model which comes out on the right side lists the computed eigen-modes as result-cases. Thus they can be superimposed using the "ModelView"-component for form-finding or structural optimization. All loads which were defined on the input model get discarded. The determination of eigen-shapes can take some while in case of large structures or many modes to be calculated. Grasshopper has no "Cancel"-button. Therefore you should save your model before activating the component.
The number of different eigen-modes in a structure equals the number of degrees of freedom. In case of beams there are six degrees of freedom per node, with only trusses attached, a node possesses three degrees of freedom. Figure 3.61 shows the first nine eigen-modes of a triangular beam mesh that is fixed at its lower corners. In the upper left corner of figure 3.61 one sees the undeformed shape. The higher the index of an eigen-mode the more folds it exhibits.
The eigen-values represent a measure for the resistance of a structure against being deformed to the corresponding eigen-form. Values of zero or nearly zero signal rigid body modes. In case that

the “Analyze”- or “AnalyzeThII”-components complain about a kinematic structure the eigen-forms can be used to detect those kinematic modes.
Again the displacements of the eigen-modes get scaled such that the largest displacement-component corresponds to 1.

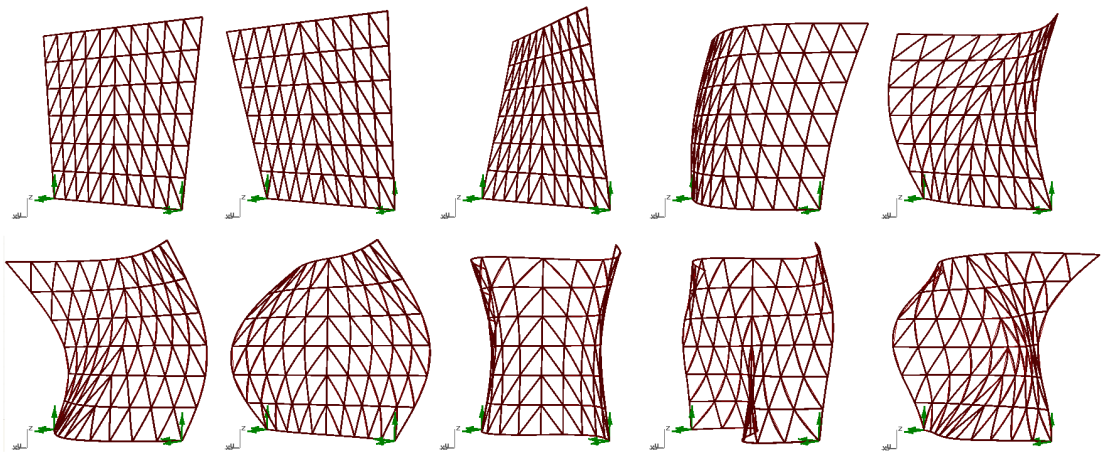

Figure 3.61 Undeformed geometry (upper left corner) and the first nine eigen-modes of the structure.

3.5.7 Natural Vibrations

In case you want to know how and at which frequency a structure vibrates use the “NaturalVibrations”-component (see fig. 3.62).
The mass of beams and trusses enters the calculation of natural vibrations with the values derived from their material weight. Karamba3D uses consistent mass matrixes for beam elements. For truss and shell elements a lumped approach is applied.
At nodes additional masses (see sec. 3.12) can be defined to simulate the effect of e.g. concrete slabs (these normally make up the majority of mass in high-rises) in an approximate manner. These masses are assumed to have translational inertia only.
Karamba3D scales the resulting vibration modes $\vec{v_i}$ in such a way that their largest component is 1. They get attached to a model as result-cases which can be viewed via a “ModelView”-component. The calculation of modal mass and participation factors are based on the modal displacements as scaled in the above described manner.

3.5.8 Optimize Cross Section

Use the “Optimize Cross Section”-component for the automatic selection of the most appropriate cross sections for beams and shells. It takes into account the cross sections load bearing capacity and optionally limits the maximum deflection of the structure.
Figure 3.63 shows a typical set-up. The initial structure consisted of I-sections of type HEA100 which have a height and width of 100 mm. They could not sustain the given load: The resulting bending stresses would lie way beyond the yield stress of the assumed material which is steel S235 with $f_y = 23.5\,\mathrm{kN/cm^2}$.

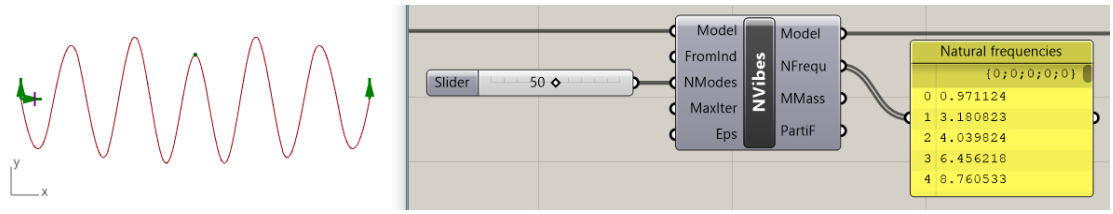

Figure 3.62 Simply supported steel beam IPE100 of length $10\,\mathrm{m}$ with a point-mass at mid-span in its 24^{th} natural vibration mode.

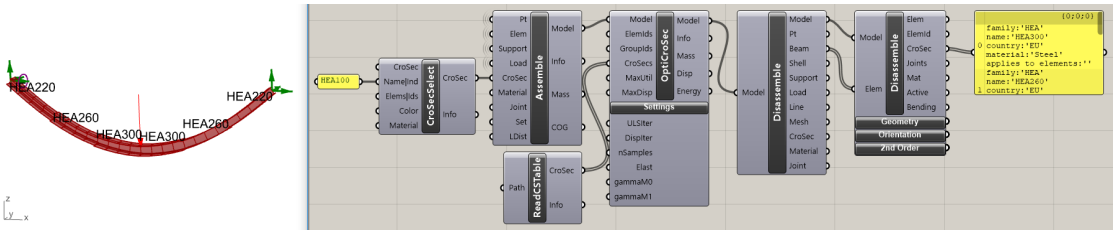

Figure 3.63 Cross section optimization with the "OptiCroSec"-component on a simply supported beam.

First the "OptiCroSec"-component determines the cross section of each element in such a way that their load-bearing capacity is sufficient for all load-cases. In order to achieve this, Karamba3D uses the following procedure:

1. Determination of section forces at "nSamples" points along all beams using the initial cross section
2. For each element or given set of elements: selection of the first sufficient entry from the family to which each cross section belongs
3. If no changes were necessary in step two or the maximum number of design iterations is reached, the algorithm stops. Otherwise it returns to step one using the cross sections selected in step two.

In statically indeterminate structures the section forces depend on the stiffness (i.e. cross section and materials) of the members. This necessitates the iterative procedure described above.
In fig. 3.64 one can see the example of a cantilever idealized with shell elements: The optimization results in thicker shell elements at the top and bottom edge of the built-in side. The "CroSecs"-input of the "OptiCroSec"-component consists of a family of constant shell cross sections.
For shells the mechanical utilization is calculated as the maximum Van Mises stress in a point divided by the material strength. For cross section optimization of shells the same procedure applies as for beams. Starting with the first item of a cross section family the algorithm tests all members and stops when a cross section is encountered for which the utilization is less than a pre-set value which is 1 by default. This corresponds to $100\,\%$.
After ensuring safety against structural failure a second, optional step follows where Karamba3D tries to reach a user supplied maximum deflection. Behind the scenes Karamba3D iteratively adapts

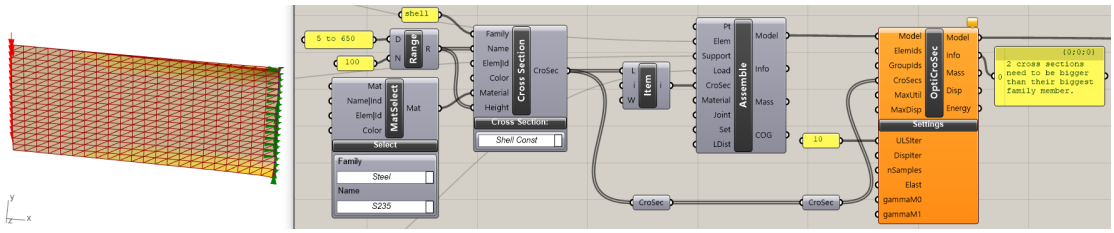

Figure 3.64 Cross section optimization with the OptiCroSec-component on a cantilever discretized with shell elements.

temporarily the strength of the materials. This may lead to uneconomic results in case of structures where the maximum displacement occurs in a small region, whereas the rest of the structure shows a much smaller deformation. In order that the iterative adaption for the maximum displacement works, the number of design iterations should be chosen appropriately – five is normally sufficient.
Building codes prescribe different levels of safety against reaching maximum displacement and load bearing limits. When using external loads at ultimate limit state level one should keep in mind that this is approximately 1.4 times the loads used to check maximum displacement requirements. Thus one way of designing structures in Karamba3D is to limit material utilization to $1/1.4 \approx 0.7$ under characteristic loads and use the resulting displacements directly for usability design.
When the given loads surpass the load bearing capacity of the biggest cross section available in a cross section family, Karamba3D issues a warning via the "Info" output-plug (see fig. 3.64).
There is no guarantee, that the iteration procedure for finding the optimal cross sections eventually converges. One can check the results via the "Utilization of Elements"-component. It applies the same procedure as the "OptiCroSec"-component for assessing elements according to Eurocode 3 [5] and consider the load-bearing capacity of the whole cross section. The utilization-output of the "ModelView"-component only shows the ratio between the stress in a point of a cross section and the material strength there. Effects like buckling under compression are not considered. This is why the utilization as displayed by the "ModelView"-component may deviate from the results of the "Utilization of Elements"-component.
Due to the lower-bound theorem of plasticity, the structure will be sufficient for the given loads at any iteration step – although some elements may show over-utilization – provided that the material is sufficiently plastic (like e.g. steel). With increasing number of iterations the statical system tends to become more and more statically determinate.
The profile selection procedure assumes that the cross sections of a family are ordered: starting with your most favorite and descending to the least desired cross section. In the cross section table "CrossSectionValues.bin" that comes with Karamba3D all families are ranked according to their height. The cross section with the smallest height comes first, the one with the largest height last. When using cross section area as sorting criteria, structures of minimum weight (and thus approximately cost) result. See 3.3.12 for how to switch between minimum height and minimum weight design. Ordering the profiles by area may lead to structures where the cross section heights vary significantly from one beam to the next.
In order to check whether a given beam cross section is sufficient, Karamba3D applies a procedure for steel beams according to Eurocode 3 (EN 1993-1-1) (see [5] for details). The interaction values for the cross section forces k_{yy}, k_{yz} and so on get calculated according to EN 1993-1-1 appendix B.

The values C_{my} and C_{mz} are limited to a minimum of 0.9. The design procedure takes account of normal force, biaxial bending, torsion and shear force. For more details see section A.4.6 and the master thesis of Jukka Mäenpää [9]. It is possible to switch off the influence of buckling for single members or set user defined values for the buckling length (see section 3.1.10).
The adverse effect of compressive normal forces in a beam can be taken into account globally (see section 3.5.5) or locally on the level of individual members. The procedure applied in Karamba3D for cross section optimization works on member level. A crucial precondition for this method to deliver useful results is the determination of a realistic buckling length l_b of an element. For this the following simplification – which is not always on the safe side – is applied: Starting from the endpoints of an element, proceeding to its neighbors, the first nodes are tracked that connect to more than two elements. The buckling length is determined as the distance between these two nodes. It lies on the safe side in case of endpoints held by the rest of the structure against translation. When beams are members of a larger part that buckles (e.g. a girder of a truss) then the applied determination of buckling length produces unsafe results! One should always check this by calculating the global buckling modes (see section 3.5.5). In case of a free end the buckling length is doubled. Compressive normal forces in slender beams reduce their allowable maximum stress below the yield limit. Visualizing the level of utilization with the "ModelView"-component will then show values below 100 $\%$ in the compressive range.
The design procedure applied in Karamba3D takes lateral torsional buckling into account. An elements lateral torsional buckling length is calculated in the same way as for conventional buckling. The buckling length for lateral torsional buckling can be set manually via the property "BklLenLT" of the "Modify Beam"-component.
In the course of cross section optimization Karamba3D checks the cross sections for local buckling and issues a warning if necessary. The check for local buckling uses the classification of cross sections into classes 1 to 4 according to EN 1993-1-1. Class 4 cross sections are susceptible to local buckling.
During the optimization of cross sections normal forces N^{II} are not updated. In order to include second order theory effects either set N^{II} manually or use "AnalysisThII" (see section 3.5.2) to determine N^{II} iteratively.
The OptiCroSec-component provides the following set of input-plugs:

Model: Structure to be optimized

ElemIds: Identifiers of elements that should be optimized. If not specified, optimization is carried out for the entire model.

GroupIds: List of identifiers of groups of elements that take part in cross section design and shall have uniform cross section. One can use the names of element sets and regular expressions for defining groups..

CroSecs: Cross section-list that contains families of cross sections ordered from most favorite to least desired. Family membership of cross sections is given via their "family" property.

MaxUtil: Target value of the element utilization where 1.0 means full utilization - the default. In some situations (e.g. early stage design) loads or geometry can not be fully specified. Then it makes sense to keep some structural reserves for later phases by setting this value to less than 1.0. When working with characteristic loads this value should be less than 0.7.

MaxDisp: For usability of a structure it is necessary to put a limit on its maximum deflection. This can be done using the "MaxDisp"-plug. By default its value is -1 which means that the maximum deflection is not considered for cross section design. When working with design loads keep in mind that those are roughly a factor of "1.4" above the level to be considered for usability.

In order to see all input-plugs click on the "Settings"-button to unfold the rest of the component:

ULSIter: Maximum number of design iterations for sufficient load bearing capacity in the ultimate limit state (ULS). The default value is five.

DispIter: Maximum number of iterations used to reach the maximum displacement criteria in case there is one. The design iterations for maximum displacement come after those for load bearing capacity.

nSamples: Number of points along beams at which their utilization is determined. The default is three.

Elast: If set to "True" (the default) cross section design is done within the elastic range. This means that under given loads the maximum resulting stress in a cross section has to lie below the yield stress f_y of the material. In case of materials with high ductility (like steel) the plastic capacity of cross sections can be exploited. Depending on the cross section shape the plastic capacity is $10\,\%$ to $20\,\%$ higher than the elastic capacity. Set "Elast" to "False" in order to activate plastic cross section design. When enabling plastic cross section design do not be surprised that the "ModelView" reports utilization-levels beyond $100\,\%$. The reason is that Karamba3D assumes linear elastic material behavior.

gammeM0: Material safety factor according to EN 1993-1-1 in case that failure is not initiated by buckling. Its default value is 1.0. In some European countries this factor lies above 1.0.

gammeM1: Material safety factor according to EN 1993-1-1 in case that buckling initiates failure. The default value again lies at 1.0 - may be specified differently in your national application document of EN 1993-1-1.

On the output side the "Model"-plug renders the structure with optimized cross sections. Check the "Info"-plug in order to see whether any problems occurred during optimization. The "Mass"-output informs you about the overall mass of the optimized structure. "Disp"- and "Energy"-plugs return the maximum displacement and internal energy of the structure after the last cross section design iteration.
The aim of the design procedure applied in Karamba3D is to render plausible cross section choices. Be aware that it builds upon assumptions like the correct determination of buckling lengths.

3.5.9 BESO for Beams

Evolutionary structural optimization (ESO) constitutes a method of topology optimization which was pioneered by Y.M. Xie and G.P. Steven. The underlying principle is simple: One starts from a given volume made up from structural elements on predefined supports and with preset loads acting on it.

Calculating the structural response will show that there are regions which carry more of the external load than others. Now one removes a number of those elements of the structure that are least strained and thus least effective. Again the response of the now thinned out model is determined, under-utilized elements removed and so on. This iterative procedure stops when a target volume or number of remaining structural elements is reached.
The above algorithm can be viewed as a way of tracing the internal force flow through a structure and removing those elements that do not form part of it. Fig. 3.65 shows a cantilever after applying the "BESO for Beams"-component on it. The algorithm works on beam and truss elements only. For shells a separate component exists (see section 3.5.10).

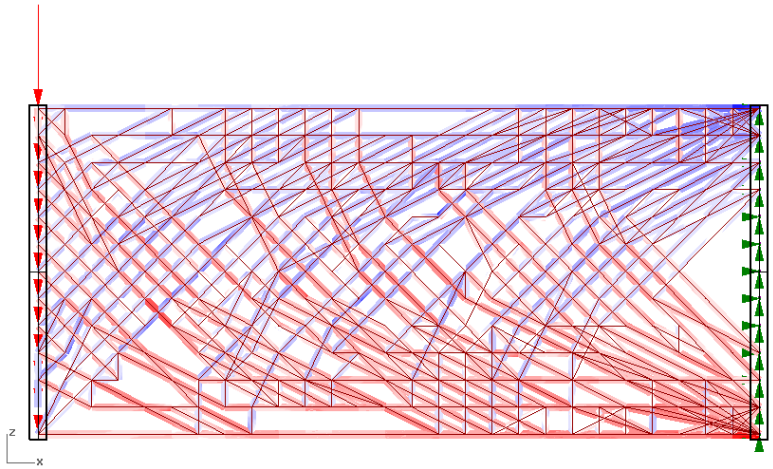

Figure 3.65 Cantilever with initially regular mesh after application of the "BESO for Beams"-component.

Figure 3.66 shows the "BESO for Beams"-component at work. On the left side one can see the initial geometry which is a triangular mesh derived from a surface. There exist two load cases with loads acting in the plane of the structure in horizontal and vertical direction respectively. Three corner nodes of the structure are held fixed. The right picture shows the optimized structure reduced to 45 % of its initial mass in the course of 20 design iterations.
Here the description of the input parameters:

Model: Receives the model to be processed.

ElemIds: There are two alternatives concerning this input parameter:

- No input: The whole of the structure will be included in the optimization procedure.
- The input consists of a list of strings: All elements whose identifiers match take part.

LCases: List of load cases to be considered. Zero is the index of the first load case. Considering the total effect of several load cases amounts to adding up their individual influences on an element.

TargetRatio: Ratio of the target mass of beam- or truss elements to the initial mass of beam-s/trusses in a structure. When determining the initial mass all beam- or truss- elements of

the structure – irrespective of state of activation – count. In the target structure only active elements contribute to its mass. This enables one to apply BESO-components in series. Depending on the activation status of the model elements applying "BESO for Beams" will lead to an increase or decrease in the number of active elements. The activation status of individual elements can be set by means of the "ModifyBeam"- and "ActivateModel"-components.

MaxChangeIter: Number of iterations within which the target mass of the structure should be reached. If the number of iterations is selected too low then it may occur that single beams get disconnected from the main structure and they seem to fly. The reason for this lies in the fact that Karamba3D applies a so called soft-kill approach for thinning out the structure: Elements are not removed but simply given small stiffness values. This ensures that structural response can be calculated under all circumstances.

MaxConvIter: Maximum number of additional iterations for convergence after the structures mass has reached its target value using "MaxChangeIter" iterations.

GroupIds: Expects a list of strings. Elements that match a given list entry take part in the optimization and belong to one group. They get collectively activated or deactivated during force path finding. A structure may consist of active and non-active elements. The initial state of a group is determined by the state of the majority of its elements. Groups need not be disjoint.

By clicking on the "Settings" bar you can unfold the following input-plugs:

Factors for weighting forces/moments: The "BESO for Beams"-component lets you select weighting factors for the different force and bending components in an element. The weight of an element is determined on the basis of the density of deformation energy induced by individual cross section force components. Multiplication by the corresponding user given weighting factor and adding up the component contributions results in the element weight. The weight of groups results from the average of their members. These are the available weighting factors:

- "WTension": factor for axial tension force
- "WCompr.": factor for axial compression force
- "WShear": factor for resultant shear force
- "WMoment": factor for resultant moments

BESOFac: Say in each iteration step there needs to be a mass of $n[kg]$ removed in order to meet the structures target mass in the given $MaxChangeIter$ number of iterations. With $BESOFac = m$ there will be $(m+1) \cdot n$ active elements moved to the pool of inactive elements. An evaluation of the structures response follows. In a second step $m \cdot n$ members get flipped from inactive to active so that the balance is right again. This adds a bi-directional component to the process which often leads to improved results.

MinDist: In some cases one wishes to limit the number of elements that get added or removed in a certain area. "MinDist" lets you select the minimum distance in meter between the endpoints of elements that may be changed in one iteration.

WLimit: At the end of the BESO-process it often occurs that a small fraction of the elements is much less utilized than the average. "WLimit" lets you remove those elements whose weight is below "WLimit" times the average weight of elements.

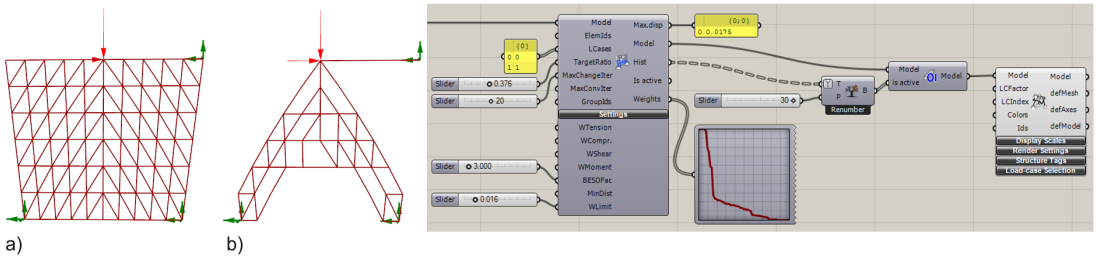

Figure 3.66 Triangular mesh of beams before (a) and after (b) applying the "BESO for Beams"-component.

On the right side of the "BESOBeam"-component these output-plugs exist:

Max.disp: maximum displacement of the resulting model from among all load cases.

Model: structure with element activation according to the force path found.

Hist: a data tree which contains for each iteration step a list of boolean values that signify whether an element in active (true) or inactive (false). The boolean values map directly on the model elements. Using a "Tree Branch"-component with a slider connected to a "Activate Model"-component (see section 3.1.5) lets you inspect the history of the BESO-process (see fig. 3.66).

Is active: renders a list of "True"/"False" values – one for each element. "True" signals that the corresponding element is part of the final structure (i.e. active). Otherwise it contains a "False" entry.

Weights: List of element or group weights in ascending order in the final structure. This can be used as a qualitative check of the result: The more evenly distributed the weights, the better utilized the structure. There will always be force concentrations around supports and external loads which show up as sharp peaks. A good way of visualization is to use a "Quick Graph"-component (see fig. 3.66).

3.5.10 BESO for Shells

An in-depth description of the BESO for shells algorithm used in Karamba3D can be found in [7]. Fig. 3.67 shows an example which starts off from a rectangular wall which supports two point-loads at each of its upper corners. The result of the BESO procedure is the X-shaped structure shown on the left side.
These are the main parameters that control the optimization process:

Model: Model to be optimized.

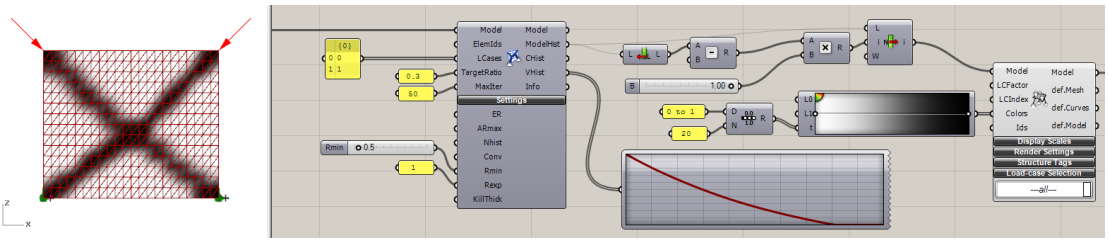

Figure 3.67 Bidirectional Evolutionary Structural Optimization (BESO) for a rectangular plate under two corner loads.

ElemIds: List of identifiers of shells that take part in the optimization. In case of an empty list (the default) all shells are included.

LCases: List of load cases to be considered. Zero is the index of the first load case. Considering the total effect of several load cases amounts to adding up their individual influences on an element.

TargetRatio: Ratio of the target mass to the initial mass of the shells in a structure. When determining the initial mass all shell elements of the structure – irrespective of state of activation – count. In the target structure only active elements contribute to its mass. This enables one to apply BESO-components in series.

MaxIter: Maximum number of iterations.

Under the submenu "Settings" these additional options can be used to further customize the optimization procedure:

ER: Is short for evolutionary ratio and defines the ratio between the volumes V_i and V_{i+1} of the optimized structure in two consecutive steps: $V_{i+1} = V_i \cdot (1 \pm ER)$. The sign of "ER" depends on whether elements shall be added or removed. In case that $ER < 0$ – which is the default – "ER" is set automatically: $ER = (1 - TargetRatio)/MaxIter + AR_{max}/2$. In case that "ER" is too small, the target mass of the optimized structure can not be reached within "MaxIter" steps.

ARmax: The ratio between maximum number of elements to be added per step and all shell elements.

Nhist: Number of iterations between those steps which are used for calculating the convergence criteria.

Conv: Relative change of the mass between two iterations N_{hist} cycles apart below which convergence is assumed.

Rmin: In order to avoid the formation of checkerboard patterns a filter scheme is used for calculating the fitness of individual elements (see [7], section 3.3.2). R_{min} defines the radius of influence

in meters for determining the element sensitivity. It is thus important to choose this value according to the mean element size. If $R_{min} < 0$ (the default) then R_{min} is set equal to the characteristic element length which is calculated as $(totalArea/numberOfElements)^{0.5} \cdot 2$.

Rexp: Determines how the strain energy at nodes within the distance R_{min} of the element center is weighted for calculating an elements sensitivity. The weight is determined as $w = (R_{ij}/\sum R_{ij})^{R_{exp}}$. Here $R_{ij} = R_{min} - R$ with R being the distance between a sample node and the center of the element. $\sum R_{ij}$ is the sum of the center distances of all nodes closer than R_{min} to the element center.

KillThick: The BESO for shell procedure makes use of a so called "soft kill"-approach. Instead of removing elements from the model they are made very soft by reducing their thickness. With the input-plug "KillThick" a value other than the default 0.000 01 m can be selected.

The output-plugs of the "BESOShell"-component return the following data:

Model: Model which results from the BESO optimization.

ModelHist: List of intermediate models – one for each iteration step of the BESO procedure.

CHist: History of the volume weighted compliance of the structure which drives the BESO procedure. When fed into a "Quick Graph" component one can check whether the BESO procedure converged: at the end the chart should be horizontal. If that is not the case try a smaller "ER"-value.

VHist: List of values which chart the development of the volume of the shells to be optimized.

Info: Returns information regarding the solution process in case something goes wrong.

3.5.11 Optimize Reinforcement

The "Optimize Reinforement"-component calculates reinforcement quantities for arbitrary shells. The algorithm is based on the sandwich model approach of Marti (see [6] or [4]). Each load-case is considered separately and the maximum reinforcement of all load-cases is chosen.

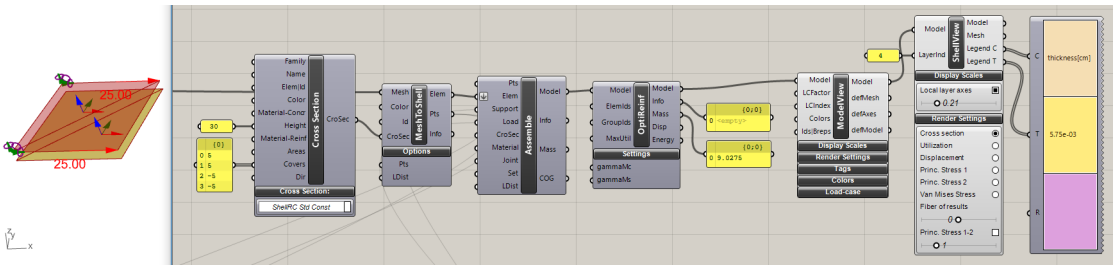

Figure 3.68 Calculation of reinforcement quantities for a plate of 1 m by 1 m and an in-plane tensile load of 50 kN.

Fig. 3.68 shows the a rectangular plate of size 1 m by 1 m with a uniform tensile line-load of 50 kN/m which translates to two point-loads of 25 kN each. The first step consists of defining a reinforced

concrete cross section via a “Cross Section”-component (see section 3.3.2). The definition of reinforcement layers does not impact the displacements or cross section forces of the model. It merely forms the basis for calculating reinforcement quantities using linear elastic cross section forces. For reinforcement design it is assumed that the material in layer zero (usually concrete) has no tensile and infinite compressive strength. Since the latter is a simplification, one should assure a sufficient height of the concrete cross section by using the “Optimize Cross Section”-component first.
The input-plugs of the “Optimize Reinforement”-component are similar to those of the “Optimize Cross Section”-component:

Model: Structure for doing reinforcement design.

ElemIds: Identifiers of elements for which reinforcement quantities should be calculated. If not specified, optimization is carried out for all reinforced concrete cross sections.

GroupdIds: List of identifiers of groups of elements that take part in reinforced cross section design and shall have uniform reinforcement.

MaxUtil: Target value of the reinforcement utilization where 1.0 means full utilization - the default.

Under the submenu “Settings” reside two plugs which let you specify the partial safety factors for concrete (“gammaMc”) and steel (“gammaMs”). The former exists for future use, since currently concrete is assumed to be infinitely strong in compression. The latter is set to 1.15 which constitutes the standard value according to Eurocode 2 (see [3]). The output of “Optimize Reinforement” comprises these plugs:

Model: Structure with optimized reinforcement.

Info: Warnings regarding the reinforcement design process.

Mass: Total mass of reinforcement after optimization in kilograms.

Disp: Maximum displacement of the structure for each load-case.

Energy: Elastic deformation energy of the structure for each load-case.

The “ShellView”-component lets one display the thickness of the reinforcement layers and the stresses there (see fig. 3.68). The input-plug “LayerInd” sets the index of the layer to be displayed. The concrete cross section has index zero, index one corresponds to the top, index four to the bottom-most reinforcement layer. The Z-direction of the local coordinate system points to the top of the cross section. The entry “Local layer axes” under “Display Scales” in the component “ModelView” lets one enable, disable and scale the arrows of the local coordinate system.
In the example of fig. 3.68 the default reinforcement material BSt500 with a characteristic yield strength of $f_{y,k} = 50\,\mathrm{kN/cm^2}$ leads to a necessary amount of reinforcement of $a_s = \frac{50\,\mathrm{kN}\cdot 1.15}{2\cdot 50\,\mathrm{kN/cm^2}} = 0.575\,\mathrm{cm^2/m}$. This is equivalent to a layer thickness of $0.005\,75\,\mathrm{cm}$. The “2” in the denominator results from the fact that there are two reinforcement layers (of index one and four) which point in the direction of the tensile force.

3.5.12 Tension/Compression Eliminator

The “Tension/Compression Eliminator”-component removes elements from a model based on the sign of their axial force.

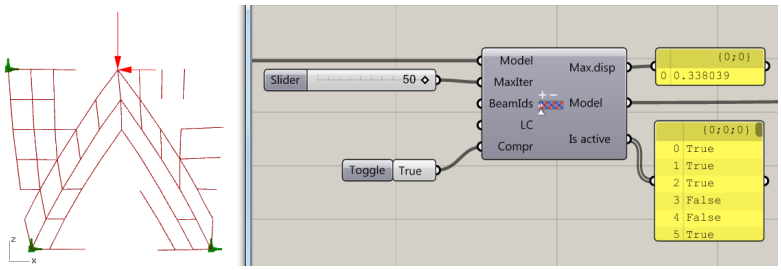

Figure 3.69 The “Tension/Compression Eliminator”-component.

These are the available input parameters:

Model: Structure to be processed.

MaxIter: The removal of tensile or compressive elements works in an iterative fashion. The procedure stops either when no changes occur from one step to another or if the maximum number of iterations “MaxIter” is reached.

BeamInds: Indexes of the elements that may be removed in the course of the procedure. By default the whole structure is included.

LC: You can specify a special load case to consider. The default is 0.

Compr: If “True”, then only members under compression will be kept. Otherwise only members under tension will survive. This value is “False” by default.

Elements selected for removal are assigned a negligible stiffness (i.e. a soft-kill approach is used).

3.6 Results

The results category consists of three sections. The first contains components that apply to a structure in general. Components of the second and third category apply specifically to beams and shells respectively.

3.6.1 ModelView

The “ModelView”-component of the “Results” subsection controls the general display properties of the statical model (see figure 3.70). More specific visual properties that relate to beam and shell elements can be defined with the “BeamView” and “ShellView”-component. The viewing options get stored in the model. Settings of view-components thus stick with the model and remain valid further down the data-stream until changed by another view-component.

When adding a “ModelView” to the definition it is sometimes a good idea to turn off the preview of all other components so that they do not interfere. Clicking on the black menu headings unfolds the “ModelView”-component and unveils widgets for tuning the model display. Each of these will be explained further below. The range and current value of the sliders may be set by double-clicking on their knob.

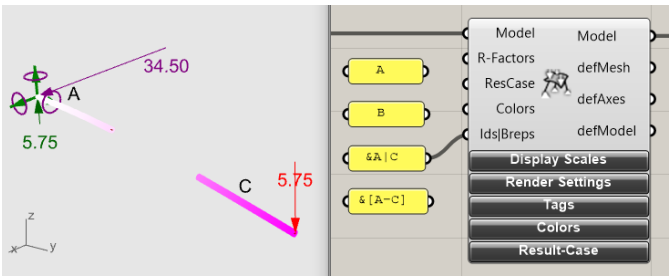

Figure 3.70 Partial view of a model.

The “ModelView”-component features five plugs on its left side:

Model: Expects the model to be displayed.

R-Factors: There exist two options for scaling the deflection output. First there is a slider entitled “Deformation” in the menu “Display Scales” that lets you do quick fine-tuning on the visual output (see below). Second option: the input-plug “R-Factors” which accepts a list of numbers that “ModelView” uses to scale the displacement-output. Its default value is 1.0. Each item in the list applies to a result-case. If the number of items in this list and the number of result-cases do not match then the last number item is copied until there is a one to one correspondence.

This option for scaling displacements can be used in the course of form-finding operations: The “def.Model”-plug at the right side of the “ModelView” (see below) outputs the model with displaced geometry which can be used for further processing. Selecting item “–all–” on the drop-down-list for the selected load case results in a superposition of all load cases with their corresponding scaling factor.

ResCase: Lets one select the visible result-case. The value in “ResCase” will be added to the result-case selected in the drop-down-list of “ModelView”. “—all—” at the drop-down-list and “ResCase” set to 0 result in the first result-case to be displayed. If the resulting number is larger than the number of available result-cases the “ModelView” turns red. If the resulting value is smaller than 0 (the default) all result-cases are superimposed. The possibility of using a number-slider for selecting load-cases makes life easier in case that there are many of them.

Colors: Color plots for e.g. stresses use a color spectrum from blue to white to red by default. One can customize the color range by handing over a list of RGB-values to the “Colors”-plug. There have to be at least four colors given. The first color is used for values below, the last color for values above the current number range. The remaining colors get distributed over the number range (see fig. 3.71). The colors are centered on zero if zero is part of the number range. Otherwise the colors spread evenly between lower and upper numerical limit. In case

you want to change the coloring defaults, set them in the “karamba.ini”-file. There it is also possible to switch off the centering around zero by setting “center_color_range_on_zero” to false.

Ids|Breps: This plug lets one select those parts of a model which shall be displayed. It expects a list of strings or Breps. The default value is an empty string which means that all of the model shall be visible. As one can see in fig. 3.70 it is possible to input regular expressions. These must start with the character “&” and adhere to the conventions for regular expressions as used in C#. The identifier of each element of the model is compared to each item of the given string list. In case a list entry matches the element identifier the element will be displayed. Fig. 3.70 contains four examples of “Id” lists: The first would limit visibility to element “A”, the second to element “B”. The third is a regular expression which matches elements “A” or “C”. The fourth matches elements “A” to “C”. Alternatively one can plug closed Breps into the “Ids|Breps”-plug. In that case only those elements get displayed which lie inside one of the volumes with at least one of their end-nodes.

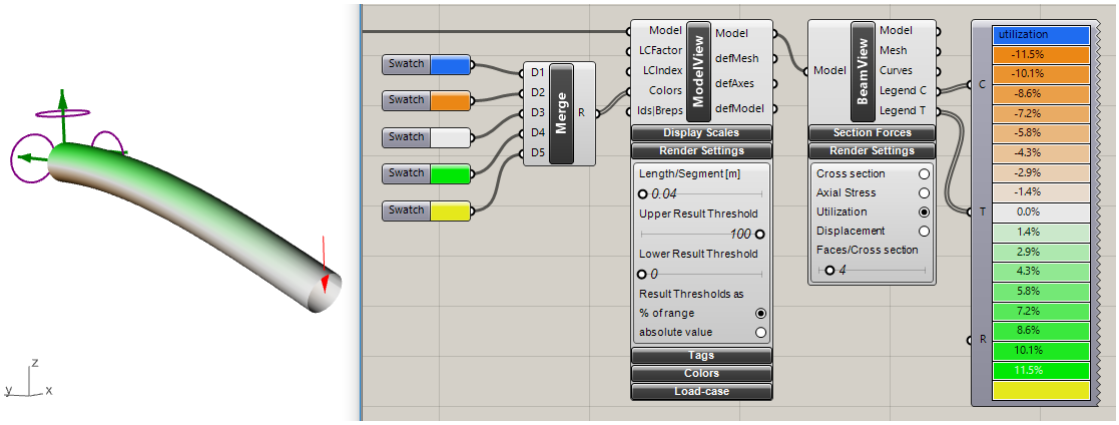

Figure 3.71 Color plot of strains with custom color range.

There are five output plugs on the ModelView-component:

- “Model” Is the model which was fed in on the left side with viewing options attached.
- From the “defMesh” output-plug you can get the mesh of the shells and beam cross sections of the deformed model for further processing. It is a list of meshes with each item corresponding to one shell or beam.
- The “defAxes” plug delivers the axes of the beams of the deformed structure as interpolated 3^{rd} degree nurb-splines. Use the Length/Subdivision slider to set the number of interpolation points.
- “defModel”: When there are results available from a statical calculation deflections are scaled and added to the node coordinates of the original model so that it contains the deformed geometry.

The “Display Scales”-submenu

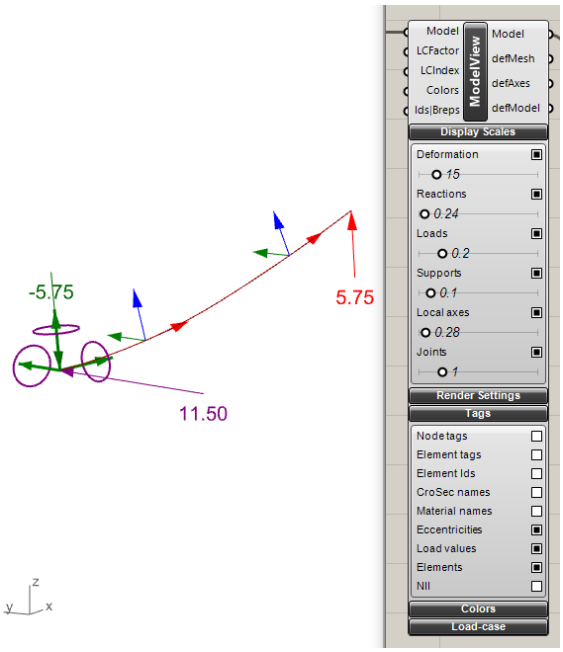

Figure 3.72 Local axes of cantilever composed of two beam elements, reaction force and moment at support.

The “Display Scales”-submenu contains check boxes and sliders to enable/disable and scale displacements, reaction forces at supports, load-symbols, support-symbols, local coordinate systems and symbols for joints at the endpoints of elements. The displacement scale influences the display and the output at the model-plug. It has no effect on stresses, strains, etc.. The colors of the local coordinate axes red, green, blue symbolize the local X-, Y-, and Z-axis.

The “Render Settings”-submenu

The slider entitled “Length/Segment[m]” lets one control the distance at which beam results (displacements, forces, moments, etc.) are plotted (see 3.6.7). It also sets the number of control points that are used for the “defAxes”-output and for displaying.
In some cases the color display of results gets distorted by the presence of stress concentrations or utilization peeks. They make much of the structure look unstrained with some small patches of color where the peeks are. The “Upper Result Threshold”- and “Lower Result Threshold”-sliders let you eliminate these extreme values. In case of the “Upper Result Threshold”-slider a value of x% sets the upper boundary value of the color range in such a way that x% of the actual value range is below. For the lower threshold it is vice versa. Values in the model beyond the given thresholds are given special colors to make them easily recognizable.
By default the result threshold values given above refer to the value range in percent. Sometimes it turns out to be practical to prescribe absolute values as thresholds (e.g. the yield stress of a material). The radio button group “Result Threshold as” can be used to switch between relative and absolute thresholds.

Limiting the value range of utilization values can be confusing: If the result thresholds are given in percent, then setting the lower threshold to zero and the upper to 100 displays the full range of utilization values. If the result thresholds are given as absolute values then a lower threshold of -100 and an upper threshold of 100 limit the color range to the areas where the material resistance is sufficient.

The "Tags"-submenu

The "Structure Tags" menu contains checkboxes for adding visual information to parts of the model:

Node tags: attaches the node-indexes nodes.

Element tags: attaches element-indexes elements.

Element Ids: displays the element identifiers.

Elements: If enabled the "defAxes" output-plug emits the axis of the deformed elements as lines and shows them on the Rhino-canvas.

CroSec names: displays the name of the cross-section of each element

Material names: displays the name of the material of each element

Eccentricities: visualizes beam eccentricities as blue lines at the end-points if active.

Load values: adds the numerical values of loads or point masses to the corresponding sysmbols.

NII: prints the value of second order theory normal forces N^{II} for all element where it is not equal to zero. For the meaning of N^{II} see 3.5.2.

The "Result-Case"-submenu

The "Result-Case" menu contains a drop-down list from which one can choose the result-case which should be displayed. When used on a model with loads, the result-cases are equivalent to load-cases. After a buckling-modes, eigen-modes or natural frequency calculation the result-cases contain buckling-, eigen- or natural modes respectively.
By default it is set to "—all—" which means that the results of all result-cases are superimposed. Define displacement-factors by feeding a corresponding list of numbers into the "R-Factor" input-plug. The "Result-Case"-selection sets the result-case to be queried for some shell results-components placed further downstream (e.g. "Force Flow Lines on Shells", "Principal Stress Lines on Shells", ...).

3.6.2 Deformation-Energy

In mechanics, energy is equal to force times displacement parallel to its direction. Think of a rubber band: If you stretch it, you do work on it. This work gets stored inside the rubber and can be transformed into other kinds of energy. You may for example launch a small toy airplane with it: Then the elastic energy in the rubber gets transformed into kinetic energy. When stretching an elastic material the force to be applied at the beginning is zero and then grows proportionally to the stiffness and the increase of length of the material. The mechanical work is equal to the area beneath the curve

that results from drawing the magnitude of the applied force over its corresponding displacement. In case of linear elastic materials this gives a rectangular triangle with the final displacement forming one leg and the final force being its other leg. From this one can see, that for equal final forces the elastic energy stored in a material decreases with decreasing displacements which corresponds to increasing stiffness.

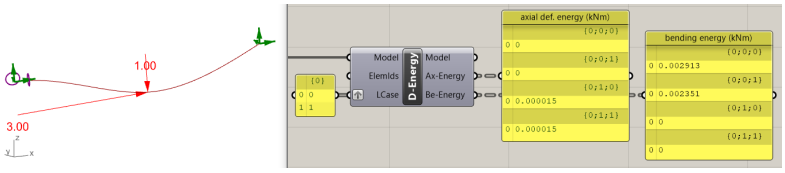

Figure 3.73 Simply supported beam under axial and transversal point-load: List of axial deformation energy and bending energy for each element.

Via the input "ElemIds" one can supply identifiers of those parts for which the deformation energy shall be calculated. An empty list means that all elements are considered. The "LCase"-input lets one select the load-case for which results shall be retrieved. The structure of the data trees returned from the "D-Energy"-component (see fig. 3.73) contains one branch per element. In case of shells the branches contain the axial or bending energy of each element of the shell.

3.6.3 Nodal Displacements

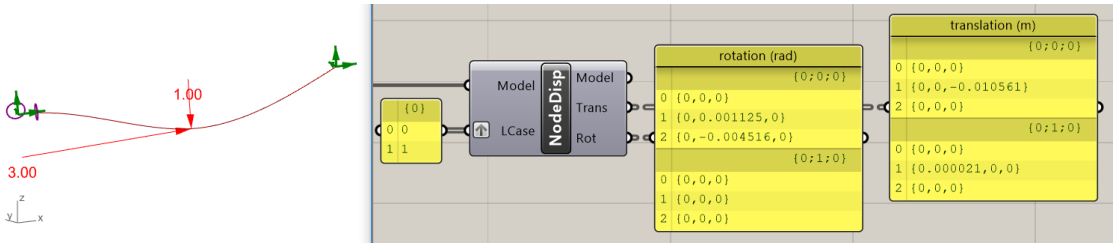

Figure 3.74 Simply supported beam under axial and transverse point-load: List of nodal displacements: vectors with translations and rotations for each node and load case.

The "NodeDisp"-component lists the displacements of each node for the load cases specified via the "LCase"-input. Two lists consisting of vectors make up its output at the plugs "Trans" and "Rot". For each node there is a vector which contains the three nodal translations or rotations (see fig. 3.74) respectively. The vectors refer to the global coordinate system. Their units are meter or radiant. A positive rotation say about the global X-axis means that the node rotates counter clockwise for someone who looks at the origin of the coordinate system with the X-axis pointing towards him or her.

3.6.4 Principal Strains Approximation

Karamba3D includes shell elements from which principal stress lines can be retrieved (see sec. 3.6.12). In case of single layer grid shells made up of beams the "Approximate Principal Strains"-

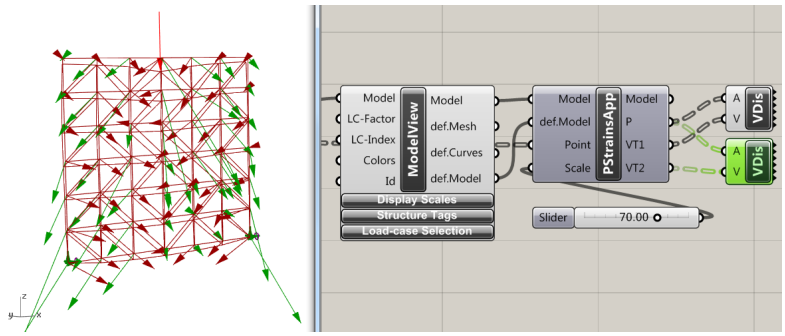

Figure 3.75 Approximation of principal strains in a simply supported slab simulated with beam elements under a point-load. Irregularity of principal strain directions is due to the irregularity of the element grid.

component can be used to determine the approximate principal strain directions of such structures (see fig. 3.75). It works on arbitrary sets of deformed points.
The calculation of principal strains is based on the assumption of a continua. When applied to nodes connected with linear elements the result can thus only result in a qualitative picture – therefore the term "Approximate".
The "Approximate Principal Strains"-component expects as input a reference model (input-plug "Model") and the same model in a deformed configuration (input-plug "def.Model"). The deformed model can be the output of a "ModelView"-component. Hand over a list of points to the input-plug "Point" where principal strain directions shall be computed. For each point in this list the following two steps are applied: First those three nodes of the reference model that do not lie on a line and have minimum distance to the given point are determined. Second the strains in the sides of the thus found triangle determine the principal strain directions – plane stress is assumed. The conversion of first (output-plug "VT1") and second principal strains (output-plug "VT2") to vectors occurs in such a way that they align with the average displacement of the triangle that defines the corresponding strain-state. The size of the vectors emanating from "VT1" and "VT2" can be scaled by providing a factor in the input-plug "Scale".
The principal strains are tangents to the principal stress lines of a structure. Use e.g. Daniel Hambleton's "SPM Vector Components" (see `http://www.grasshopper3d.com/group/spmvectorcomponents`) to retrieve these lines from the strain-vector-field.

3.6.5 Reaction Forces

The "Reaction Forces"-component gives access to the reaction forces and moments at supports. It expects a model at its input-plug and returns via "RF" and "RM" a list containing reaction forces in kN and reaction moments in kN m as three dimensional vectors. The input-plug "LCase" sets the load-case index for which the results shall be output. All support reactions are ordered in such a way that the indexes of the nodes they attach to form an ascending sequence. In case of locally oriented supports, reaction forces refer to the local coordinate system. "Pos" returns the position of the supports in the same order as the results. "SumRF" and "SumRM" offer the possibility to check the resultant reaction moments and forces of each load-case.

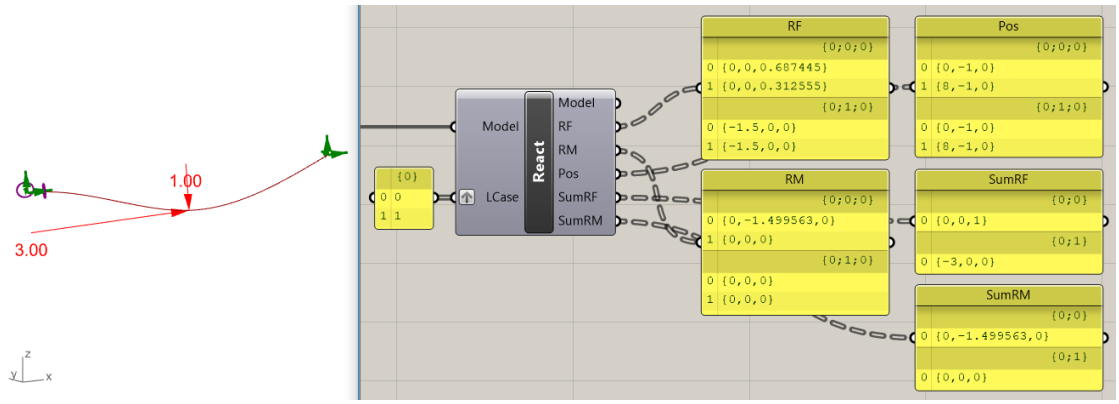

Figure 3.76 Beam under axial and transverse point-load: Reaction forces and moments for both load cases.

3.6.6 Utilization of Elements

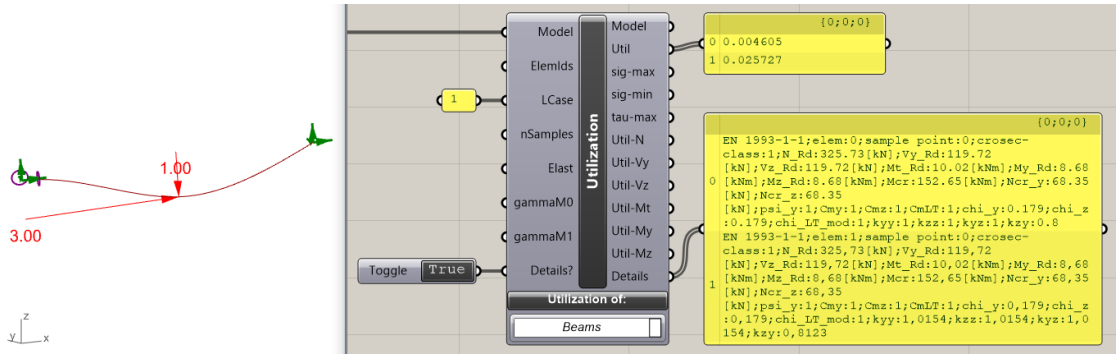

Figure 3.77 Beam under axial and transverse point-load: Utilization of the cross sections of the elements.

Use the “Utilization of Elements”-component in order to get the level of utilization for each element. It comes as a multi-component where the drop-down list on the bottom decides whether the utilization of shell of beam elements shall be returned. With beam-utilization selected, the utilization output of shell patches will be output as zero - and vice versa. This serves to maintain the one to one relationship between elements and results.
The input-plug “Model” expects an analyzed model. With “ElemIds” it is possible to limit the range of elements which shall be considered. By default the component returns results for all elements. The “LCase” selects the load-case to be used for calculating the utilization. By default it is set to “-1” which means that the maximum utilization of all load-cases will be returned.

Utilization of Beams

Fig. 3.77 shows the utilization component for beams. In case of shells, 0 is output as utilization. The meaning of the input-plugs “nSamples”, “Elast”, “gammaM0” and “gammaM1” exactly corresponds to that of the “Optimize Cross Section” (see 3.5.8). The algorithm for determining an elements utilization

is the same as that underlying the cross section optimization procedure. Set the input-plug “Details?” to “true” in order to get intermediate values of the utilization calculation at the output-plug “Details”. For large structures the generation of the detailed output may take some time.
Utilization numbers for beams rendered by this component (output-plug “Util”) and the “ModelView” show differences – especially for compressive axial forces: The “ModelView”-component returns the ratio of stress to strength stress as the level of utilization, whereas the “Utilization of Elements”-component also includes buckling. See for example the two utilization entries on the in fig. 3.77: The second load case (i.e. number “1”) is made up of an axial load acting in the middle of the beam. As both ends are axially fixed, one beam is in tension, one in compression. The absolute value of the normal force in both elements is the same. Yet the beam under compression has a utilization of 0.26, the one under tension only 0.05. “1” means 100 %.
The output-plugs “sig-max” and “sig-min” return the minimum and maximum stress in each beam.
In order to diagnose the reason why a specific beam shows over-utilization the output-plugs “Util-N”, “Util-Vy”, “Util-Vz”, “Util-Mt”, “Util-My” and “Util-Mz” return the contribution of each cross section force component to the overall utilization. When enabled via “Details?” the output-plug “Details” renders a detailed account of intermediate values used for the calculation of the element’s utilization according to EN 1993-1-1 [5].

Utilization of Shells

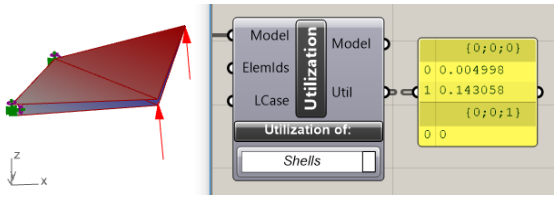

Figure 3.78 Utilization of a shell consisting of two elements.

The utilization calculated for shells (see fig. 3.78) is the ratio between yield stress and Van Mises stress in each element of the shell. The output-plug “Util” lists the utilization of each element of the shell in the same order as the mesh-faces are listed in the mesh which underlies the shell geometry. In case of beams or trusses, 0 is output as utilization.

3.6.7 BeamView

The “BeamView” component controls the display options related to beams and trusses (see fig. 3.79). This concerns the rendering of cross section forces, resultant displacements, utilization of material and axial stress.

The “Render Settings”-submenu

When activated, “Cross section”, “Displacement”, “Utilization” and “Axial Stress” result in a rendered view of the model. Utilization is calculated as the ratio between the normal stress at a point and the strength of the corresponding material. Shear and buckling are not considered.

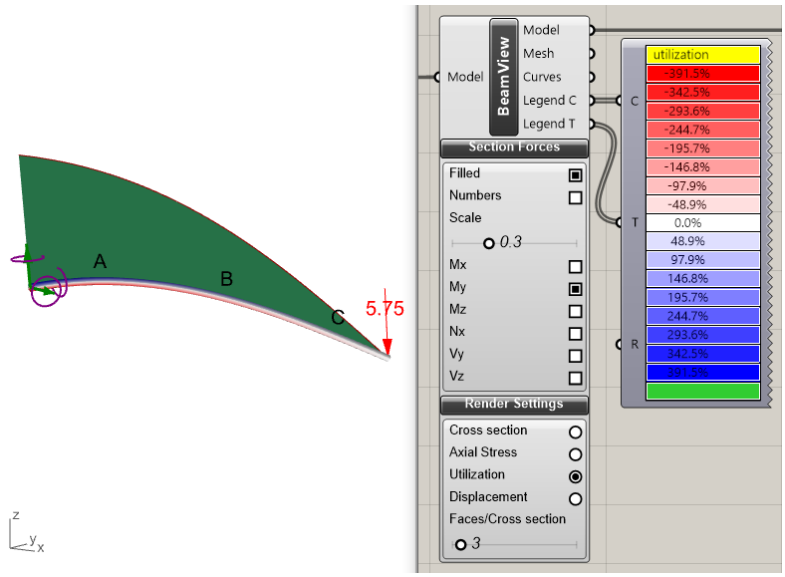

Figure 3.79 Display of utilization and bending moments of cantilever beam.

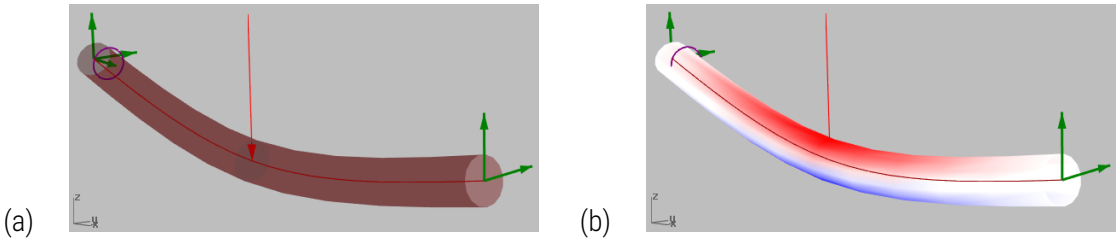

Figure 3.80 Rendered images of the beam. *Left:* "Cross section"-option enabled. *Right:* "Axial Stress" enabled.

The color range of the results starts at the minimum value and stretches to the maximum. You can define individual color ranges for all result quantities in the "karamba.ini"-file. A "Legend"-component lets you inspect the meaning of the colors.

The mesh of the rendered image is available at the "Mesh"-output of the "BeamView"-component. Two sliders control the mesh-size of the rendered beams: First "Length/Segment" of "ModelView" determines the size of sections along the middle axis of the beams. Second "Faces/Cross section" of "BeamView" controls the number of faces per cross-section. For rendering circular hollow cross sections the number of "Faces/Cross section" is multiplied by six in order to get a smooth visual result.

It is instructive to see which parts of a beam are under tension or compression. Activate the "Stress"-checkbox in menu "Render Settings" in order to display the stresses in longitudinal beam direction. Red (like brick) means compression, blue (like steel) tension. In some models there may exist small regions with high stresses with the rest of the structure having comparatively low stress levels. This results in a stress rendering that is predominantly white and not very informative. With the sliders for "Upper Result Threshold" and "Lower Result Threshold" of the "ModelView" you can set the range of the color-scale. Result values beyond the upper limit appear yellow, below the lower threshold green

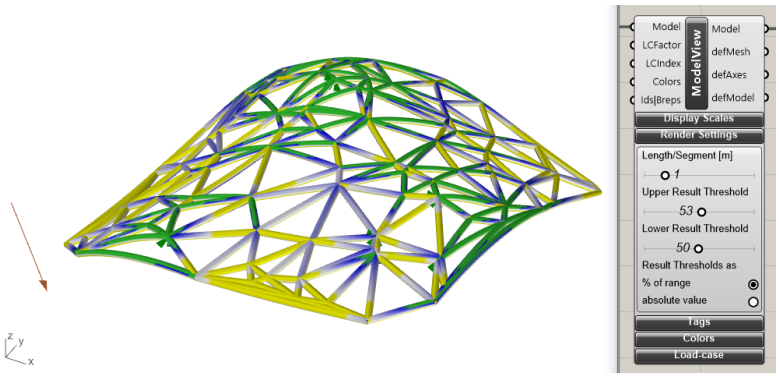

Figure 3.81 Mesh of beams under dead weight with upper and lower results threshold set to 53 % and 50 % respectively.

(see figure 3.81).

Display of cross section forces and moments

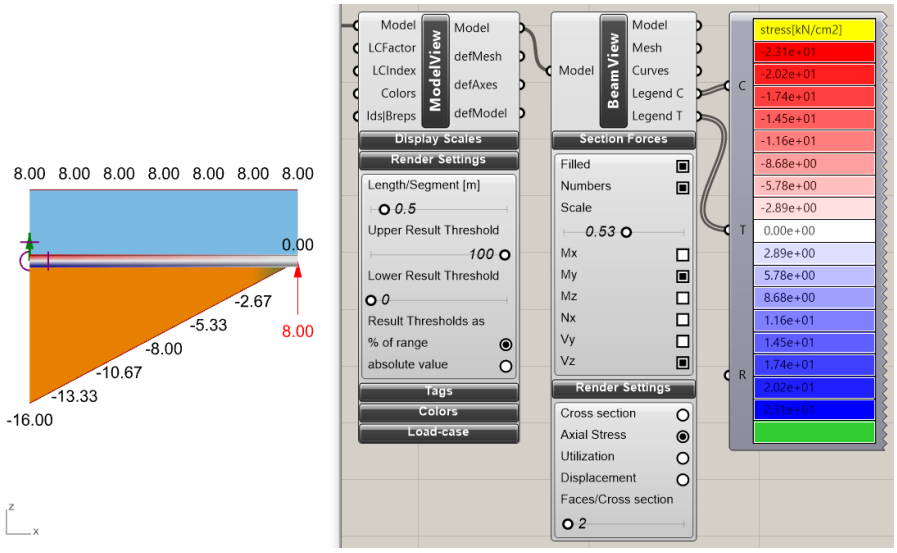

Figure 3.82 Moment M_y (orange) about the local beam Y-Axis and shear force V_z (blue) in local Z-direction.

The “Section Forces” sub-menu lets you plot section forces and moments as curves, meshes and with or without values attached. All generated curves and meshes get appended to the “BeamView” “Curves” and “Mesh” output. The graphical representation is oriented according to the local coordinate axes of the beam and takes the deflected geometry as its base. The index of bending moments indicates the local axis about which they rotate, for shear forces it is the direction in which they act (see also fig. 3.85). Customize the mesh-colors via “karamba.ini”. The slider “Length/Subdivision”

in sub-menu “Render Settings” of the “ModelView”-component controls the number of interpolation points.

3.6.8 Beam Displacements

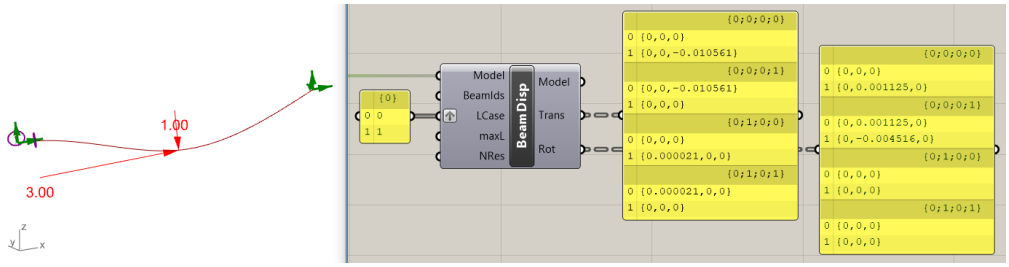

Figure 3.83 Beam consisting of two elements under axial and transverse point-load: List of displacements along the axis: three components of translations and rotations for each section and load case.

In case you want to know how displacements change over the length of a beam use the “Beam Displacements”-component (see fig. 3.83). The “BeamIDs”, “LCase”, “maxL” and “NRes” input-plugs work analogously to those of the “Section Forces”-component (see section 3.6.9).

3.6.9 Beam Forces

Sometimes it is desirable to have section forces and moments represented as components in the direction of the local axes of the cross section instead of resultant values. Use the “Beam Forces”-component in such a case. Its output is similarly structured as that of the “Beam Resultant Forces”-component, but its output plugs comprise the force components in local directions (see fig. 3.84). “LCase” and “BeamIds” input can be used to limit the results to a specific load-case of a subset of the beams in the model – the default value of “−1” outputs results for all load-cases. An empty string at the “BeamIDs”-input stands for all elements. The input parameters “maxL” and “NRes” determine the number of results along the beam axis. “maxL” can be used to control the maximum distance between results. A negative value for “maxL” means that there is no maximum distance condition. “NRes” sets the number of results along the beams. The beams endpoints are automatically included in the output.

3.6.10 Resultant Section Forces

The “Beam Resultant Forces”-component retrieves axial forces “N”, resultant bending moments “M” and shear forces “V” for all beams and load cases. See fig. 3.85 for the definition of “N”, “V” and “M”. The sequence of element results corresponds to the sequence of beams. Thus the data can be used for cross section design of radially symmetric elements.
Figure 3.86 shows a beam with two load cases presented in one picture. The beam consists of two elements and has a total length of eight meters. In load case zero a vertical force of magnitude 1 kN acts vertically downwards in the middle of the beam. Load case one consists of a point-load of 3 kN directed parallel to the undeformed beam axis. The results at the output-plugs “N” and “M”

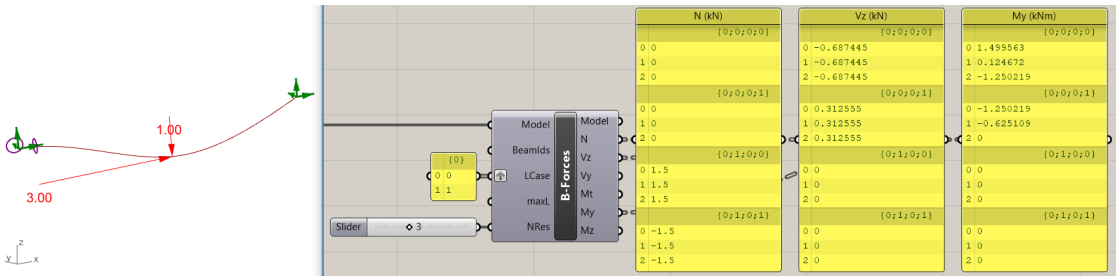

Figure 3.84 Simply supported beam under axial and transverse point-load: List of normal forces, shear forces and moments for all elements and all load cases along an the elements.

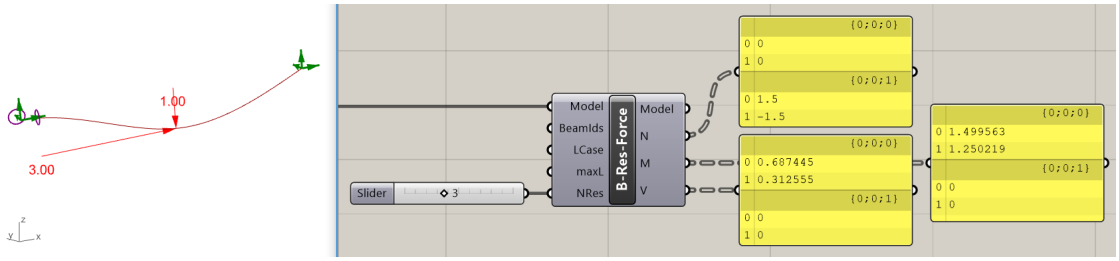

Figure 3.85 Normal force "N", shear force "V" and resultant moment "M" at a cross section with local coordinate axes XYZ. Force and bending moment components are positive in the direction of the local coordinate axes.

in fig. 3.86 are trees that hold the beams normal force and resultant bending moment. Each branch corresponds to one load-case. If the input-plug "LCase" has a value other than the default of "-1" the output in "N", "M", and "V" is limited to the load-case with the corresponding index. With "BeamIds" the result output may be confined to a subset of the beams in the model.
Tensile normal forces come out positive, compressive normal forces have negative sign. The resultant moment yields always positive values as it is the length of the resultant moment vector in the plane of the cross section.

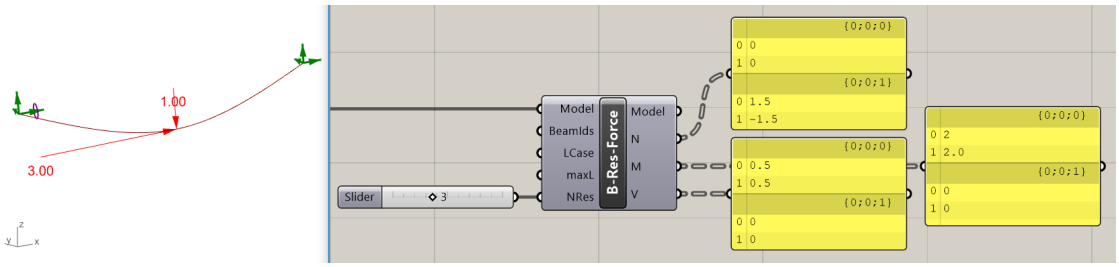

Figure 3.86 Beam under axial and transverse point-load: List of normal forces, shear forces and moments for all elements and all load cases.

The input-plug "NRes" sets the number of equidistant points along the beam axis where resultant

forces are calculated in order to determine the maximum values for output. In case of zero gravity and in the absence of uniform beam loads the maximum values of M and N occur at the endpoints. Otherwise these maxima may lie inside the elements. The default value of "NRes" is three which means that values are checked at the beams end-points and in the middle.
As "M" is always rendered positive the maximum along an element is unambiguously given. Under gravity normal forces in a beam may change sign. In such a case Karamba3D returns that "N" which gives the maximum absolute value.
Fig. 3.86 shows the results of a simply supported beam consisting of two elements under two load-cases: In load case zero both elements return zero normal force because there acts no external axial load. The maximum moment of both elements is $2\,\mathrm{kN\,m}$. For a simply supported beam under a mid-point transverse load the maximum moment occurs in the middle and turns out to be $M = F \cdot L/4 = 1\,\mathrm{kN} \cdot 8\,\mathrm{m}/4 = 2\,\mathrm{kN\,m}$.
The axial force of $3\,\mathrm{kN}$ in load case one flows to equal parts into both axial supports. It causes tension ($1.5\,\mathrm{kN}$) in the left element and compression ($-1.5\,\mathrm{kN}$) in the right one.

3.6.11 ShellView

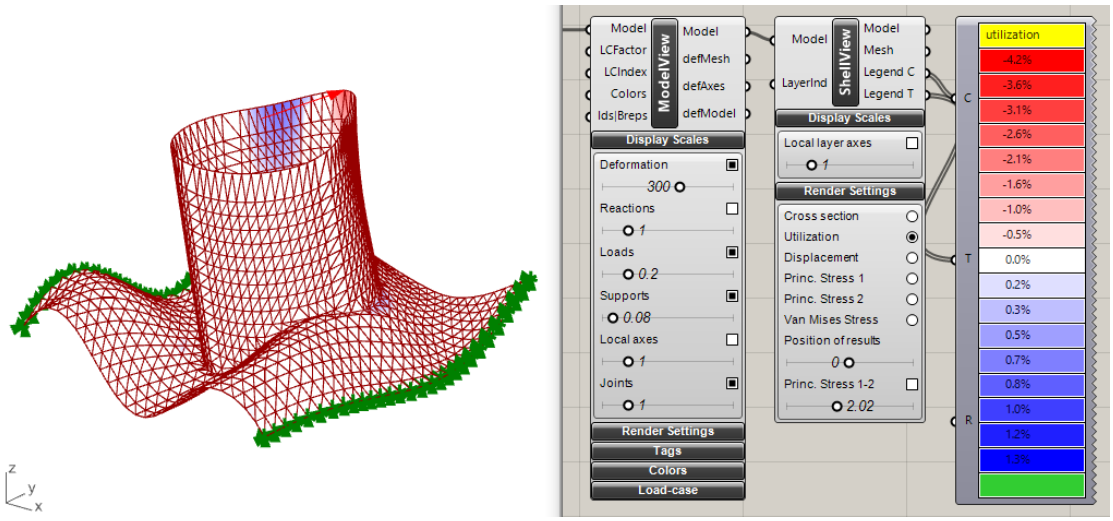

Figure 3.87 Utilization on a deformed shell.

The "ShellView"-component works like the "BeamView"-component (see sec. 3.6.7 and controls the display of shell results. Figure 3.87 shows the resultant displacement of a shell.
Shell cross sections may consist of several layers (see section 3.3.2). The layer with index "0" spans the whole cross section height by default, the other layers can be used to e.g. specify reinforcement layers. With the "LayerInd"-input, which defaults to "0", one can specify the cross section layer to be displayed. In order to set the location inside a layer one has to specify a fiber: "1" corresponds to the upper boundary, "-1" to the lower boundary of a layer. The local z-axis points to the upper boundary. Since the shell layers may have arbitrary orientations, it is often useful to display their local coordinate systems. This can be done with the "Local layer axes"-radio button under the "Display Scales"-sub-menu. The slider there allows to adapt the size of the coordinate system arrows.

Under the submenu “Render Settings” you can choose from these rendering options – the always refer to the currently set layer:

Cross section: Shows the upper and lower surface of the current layer and adds them to the output at the “Mesh”-output plug.

Displacement: Colors the shell according to the resultant displacement.

Utilization: Renders the material utilization. The utilization is calculated as the ratio between the material strength and the maximum Van Mises stress. A negative sign results if the negative value of the second principal stress is larger than the first principal stress. The output comprises two meshes symbolizing the utilization values on the top- and bottommost layer. The Van Mises Yield criterion is not applicable to brittle materials like concrete.

Princ. Stress 1: Visualizes the resultant value of the first principal stress in the current fiber of the current layer. The fiber can be set with the “Fiber of results” slider (see below).

Princ. Stress 2: Displays the resultant value of the second principal stress in the current fiber.

Van Mises Stress: Renders the Van Mises Stress in the current layer and position.

Position of Results: Sets the position on the current shell layer on which results are calculated. A value of 1 corresponds to the upper, -1 to the lower boundary of the current layer.

Princ. Stress 1-2: Enables or disables and scales the vector display of principal stresses in the current fibre of the current layer. Use the “Result Threshold” sliders in the “Display Scales”-menu of the “ModelView”-component to thin them out if necessary.

3.6.12 Line Results on Shells

Line Results on Shells

This component constitutes a multi-component which lets you generate force-flow-lines, iso-lines, principal moment- and stress-lines.

Force Flow Lines on Shells

Force flow (FF) lines or load pathes (as they are also sometimes called) illustrate the load distribution in structures [11]). There is a loose analogy between those force flow (FF) lines and streamlines in hydromechanics: The law of conservation of mass in hydromechanics is matched by the static conditions of equilibrium in a specified direction. If there are two FF-lines the resultant force between those in a predefined direction stays constant. Consider e.g. the cantilever in fig. 3.88 for which the force flow in horizontal direction is described by the red lines. At the supports the force flow lines run nearly horizontal at the upper and lower side where the normal stresses from the supports reach their maximum and thus dominate the resultant force. They gradually curve down to the neutral axis where the shear stresses constitute the only contribution to horizontal forces.
Aside from resulting in nice line drawings those force flow lines can be practical as well [11]:

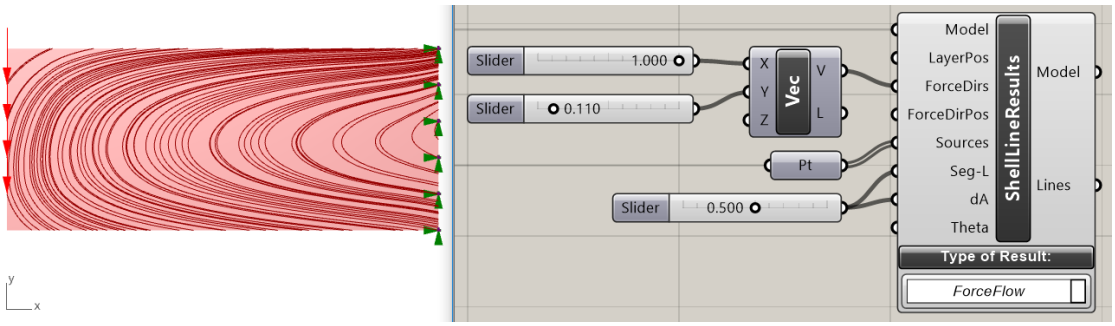

Figure 3.88 Cantilever consisting of triangular shell elements: Flow lines of force in horizontal direction.

- FF-lines form eddies in ineffective (with respect to the given force direction) parts of a structure or reverse their direction there.
- In case you want to strengthen a structure with linear elements (e.g. fibres) align them with FF-lines to get the most effective layout.

FF-lines are not the same as principal stress lines because the latter lack the property of constant force between adjacent lines.
The "Shell Force Flow Lines"-component lets you create force flow lines in arbitrary points of shells (see fig. 3.88). The load-case considered is that defined in the nearest upstream "ModelView"-component.
There exist seven input plugs:

Model: The model from which you want to create FF-lines. By default the results of all load-cases get superimposed with factor "1". Use a "ModelView"-component to select specific load-cases or to impose load-factors other than "1".

Layer: In case of bending, the stress state of shells and therefore the FF-lines change over the cross section height. A value of "-1" denotes the lower "1" the upper shell surface and "0" the middle layer. The default value in "0".

ForceDirs: Expects a vector or list of vectors that defines the direction of force. This direction gets projected on each element in order to define the local force flow directions. Elements perpendicular to the "ForceDir"-vector are skipped. Multiple such directions can be defined for different regions.

ForceDirPos: For each vector in "ForceDirs" a position can be defined. The force direction at an arbitrary point on the shell corresponds to the "ForceDir"-vector with the closest "ForceDirPos".

Source: Defines points on the shell where FF-lines shall originate. You can feed points on or near the shell into this plug. It is also possible to use lines that intersect the shell. In case of multiple intersections there will be the same number of FF-lines.

Seg-L: Intended length of the segments of the resulting FF-lines. Is 0.5 m by default. A negative value means that only $INT(abs(Seg - L))$ line segments will be drawn.

dA: This parameter sets the accuracy with which the FF-lines get determined: It is the maximum differential angle between to adjacent pieces of a FF-line. If this criteria results in pieces of length smaller than "Seg-L" then they will be joined before sent to the output-plug "Line". By default this value is set to 5°.

theta: Here you can define an angle between the FF-lines and those lines output at the "Line"-output plug. The angle is in degree and defaults to zero.

The output of the "ShellFFlow"-component consists of lines arranged in a data tree. The right-most dimension contains the branches of each flow-path: In case of a e.g. a plane there are two branches that originate from the given intersection point. In case of T-like shell topologies this number can grow to three and larger.

Isolines on Shells

The "Isolines on Shells"-component lets you do two things: First draw contour lines on shells that connect points of equal principal stresses, principal moments, utilization, resultant displacements or shell thickness (see fig. 3.89). Second query results in arbitrary points of the shell.

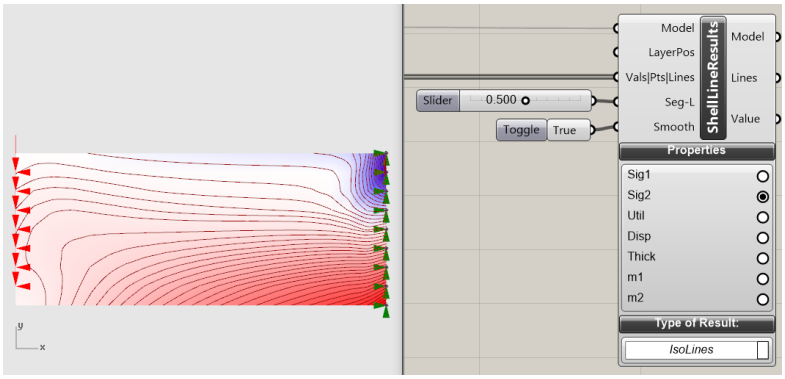

Figure 3.89 Lines of equal second principal stress on a cantilever.

The input-plugs "Model", "Layer" and "Seg-L" have the same meaning as for the "Force Flow Lines on Shells"-component (see sec. 3.6.12). In terms of placing iso-lines on the structure the input "Vals|Pts|Lines" offers the following options:

Vals: In case a list of numbers is supplied, iso-lines at these levels will be created. See the context help of the input-plug for the physical unit to use. One can e.g. use the "Legend T"-output of the "ShellView"-component after removing the first and last item from the list.

Pts: Iso-lines will start at the closest projection of the given points on the shell.

Lines: The intersection points of the given lines and the shells serve as seeds of iso-lines

The load-case to examine as well as load-case factors can be set with a "ModelView"-component plugged into the definition ahead of the "Isolines on Shells"-component. By default all load-cases get superimposed using unit load-factors.

Isolines are straight lines within each shell element. This may result in slightly rugged poly-lines. Set the "Smooth"-input plug to "True" in order to flatten them out. The "Line"-output-plug will then return splines instead of lists of line-like curves. They result from using the calculated iso-points as control-points. For curved shell geometries this has the disadvantage that those splines no longer stay exactly on the shell surface. This may give you a hard time trying to intersect different groups of such lines.
In the "property"-submenu you can select the result-value to be displayed: first or second principal stress ("Sig1", "Sig2"), first or second principal bending moments ("m1", "m2"), utilization ("Util")), resultant displacement ("Disp") or shell thickness ("Thick").
The "Lines"-output data-structure corresponds to that of the "Force Flow Lines on Shells"-component. Each number in the output-plug "Value" corresponds to one piece of isoline from the "Lines"-output.

Principal Moment Lines on Shells

Works like the "Principal Stress Lines on Shells" component (see section 3.6.12). Instead of principal stress lines it returns principal moment lines.

Principal Stress Lines on Shells

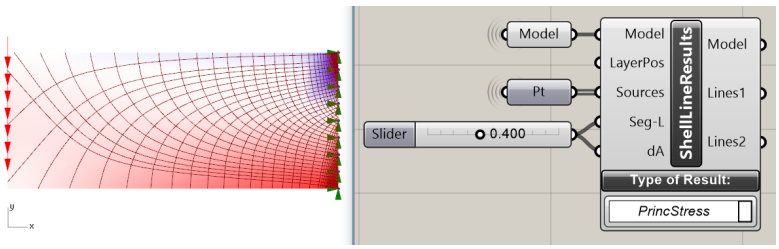

Figure 3.90 Principal stress lines: they are tangent to the first and second principal stress direction. The coloring reflects the level of material utilization.

Principal stress (PS) lines are tangent to the principal stress directions (see fig. 3.90). In the case of a cantilever they either run parallel or at right angle to the free boundaries. In the middle where normal stresses due to bending vanish, first and second principal stress lines intersect the middle axis at 45°.
The meaning of the input-plugs of the "Principal Stress Lines on Shells"-component correspond to that of the "Force Flow Lines on Shells"-component (see sec. 3.6.12 for details). On the output side "Lines1" and "Lines2" hold the first and second principal stress lines in data trees: the right-most dimension holds a list of lines that represent a part of a PS-line. There are usually two parts per line that start off to either side of the starting point. In case of more complicated topologies there can be more than two parts. These parts populate the second dimension from the right.

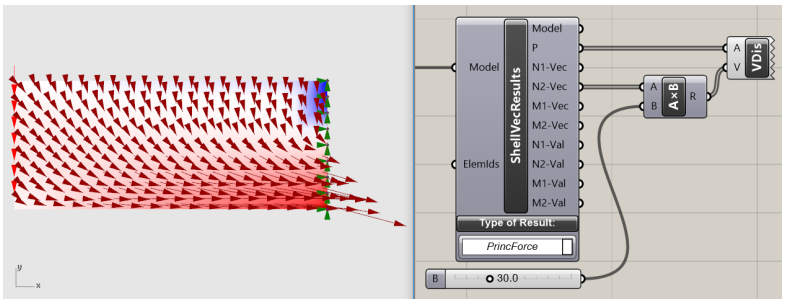

Figure 3.91 Cantilever analyzed as shell structure: directions of second principal normal forces at element centers.

3.6.13 Result Vectors on Shells

Principal Force Directions on Shells

The "Principal Force Directions on Shells"-component lets you retrieve principal normal forces and moments at element centers (see fig. 3.91). All shells of a model get considered by default. Use the input-plug "ElemIds" to select a subset. Results refer to the selected load-case at the nearest upstream "ModelView"-component.
The order of all result lists corresponds to the order of faces in the mesh used to generate the shell. Output-plug "P" lists the coordinates of the element centers where the normal forces and bending moments were calculated. "N1-Vec", "N2-Vec", "M1-Vec" and "M2-Vec" deliver the first and second principal normal force and moment directions as vectors. Their length corresponds to the absolute value of the corresponding quantity in kilo Newton per meter. The output plugs "N1-Val", "N2-Val", "M1-Val" and "M2-Val" return the values of the principal normal forces and moments.

Principal Stress Directions on Shells

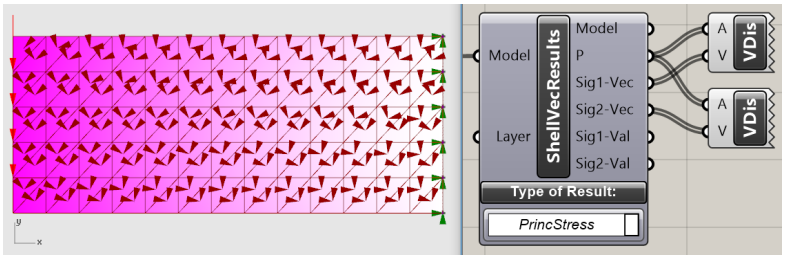

Figure 3.92 Triangular mesh of shell elements and principal stress directions at their centers. Colors indicate the resultant displacement.

This components provides the same results as the "Princ. Stress 1-2" option of the "ShellView"-component (see fig. 3.92). The output-plug "P" renders the positions of the elements centers. "Sig1-Vec" and "Sig2-Vec" return vectors for the first and second principal stresses there. "Sig1-Val" and "Sig2-Val" output the values of the first and second principal stresses. Thin out results by setting

“Result Threshold” of “ModelView” (needs to be upstream of the data-flow) to a value of less than 100 % . Like for isoline and force-flow lines a specific load case or superimposition of load cases can be set via “ModelView”. By default all load-case results get added up using unit load factors.

3.6.14 Shell Forces

With the “Shell Forces”-component one can retrieve principal or local shell cross section forces. Use the drop-down-list at the bottom of the component to switch between “Principal” and “Local”.

Principal Shell Forces

Sometimes it is of interest to know the value of principal normal forces and moments on shells. In such cases the “Shell Forces”-component for principal forces comes in handy. It lists the results in the same order as the “Principal Force Directions on Shells”-component. Thus the element centers returned there can be used in combination with the numeric results.
Distributed normal forces are negative in case of compression. Positive bending moments result in tension on the upper side of a shell. The upper side of a shell element is defined by a positive value of the local Z-axis. When in doubt about the orientation of your shell elements enable the preview of local element axes in the “ModelView”-component (see section 3.6.1). The output-plugs “vx” and “vy” return transverse shear forces along lines parallel to the shells local x- and y-axis.

Local Shell Forces

When setting the result type to “Local” the shell cross section forces in local coordinate directions will be output. See section 3.1.14 on how to change the local coordinate systems of shells.

3.7 Export

Karamba3D is not meant to be a substitute for a full blown structural engineering finite element software package. Instead it aims at providing flexibility in testing different structural designs that the more traditional FE-applications lack. We therefore started to implement interfaces to those traditional civil engineering packages. At the moment Karamba3D supports data exchange with RStab5, RStab6, RStab7, RStab8 and Robot. With GeometryGym (see `https://geometrygym.wordpress.com/downloads/` it is possible to export Karamba3D model data to IFC.

3.7.1 Export Model to DStV

Communication between Karamba3D and RStab or Robot works via “DStV”-file which is a “STEP”-derivative.
In order to create an exchange file place a “Export Model to DStV”-component on you definition and feed it with a model via the “Model”-plug. The “Path”-plug lets you chose a name for the exchange-file. RStab names cross sections and materials depending on the selected language. Therefore set RStab to “English” before importing Karamba3D DStV-files.
The output-plug “DStV” returns the export-file as a list of strings which can be viewed with a “Panel”-component. Check the output of the “Info”-plug to see whether Karamba3D encountered problems during export.

The different versions of RStab and Robot interpret some aspects of DStV data differently. For example distributed, projected loads will not be imported correctly in all versions of RStab. Therefore you should always compare the structural response calculated with the exported model in RStab or Robot to that obtained with Karamba3D.
In DStV-files the export of standard cross sections works via their name. RStab and Robot show slight differences in naming them: for example a "HEAA100" in Karamba3D is the same as a "HE100AA" in Robot and a "HE100A" in RStab. In order to get the standard cross sections right one needs to convert their names in the exported file. Karamba3D applies the conversion tables "CrossSectionNameAliases_Robot.csv" and "CrossSectionNameAliases_RStab.csv" for Robot and RStab respectively. These can be found in the Karamba3D installation folder [3]. One can open them with either a text editor or a spreadsheet program. In case the first row contains a "#" the rest is a remark. The second row contains the cross section name as used in Karamba3D, the third the alias, the fourth a user defined name or nothing. Currently the name conversion tables which come with Karamba3D do cover many but not all cross section names. In case there is no alias for a cross section name it gets exported as a non-standard cross section.

3.8 Utilities

3.8.1 Mesh Breps [4]

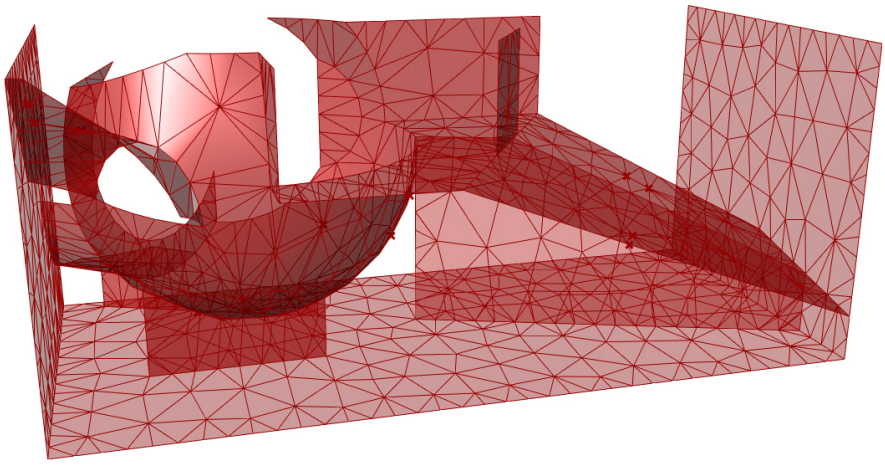

Figure 3.93 Unified mesh generated from Breps using the "MeshBreps"-component; created by Moritz Heimrath.

In Karamba3D the geometry of shells is represented by meshes. Each mesh face corresponds to a constant strain finite element. The element nodes determine its connectivity to the other parts of the structure. Thus point-loads, supports, point-masses and the like can only be attached to mesh vertices. When two meshes have common boundaries they need to have identical vertices there in order to be structurally connected.

[3] double-click on the Karamba3D desktop icon to get there.
[4] The "Mesh Breps" component was programmed by Moritz Heimrath.

The “MeshBreps”-component ensures the connectedness of meshes generated from multiple breps. It also allows to define points on those breps where mesh vertices shall result. Fig. 3.93 shows the unified mesh based on four breps and one predefined point.

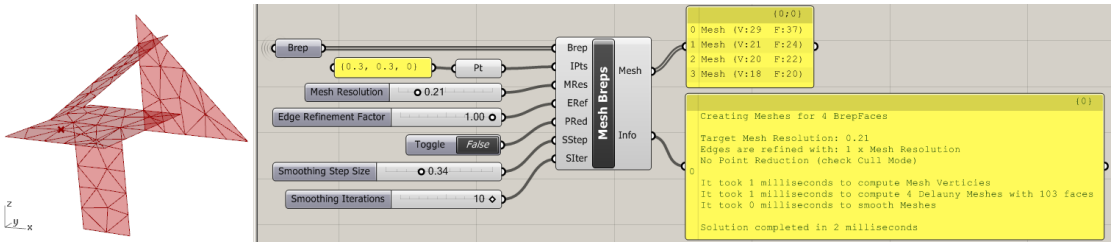

Figure 3.94 In- and output of the “MeshBreps”-component; created by Moritz Heimrath.

These input-plugs control mesh generation (see fig. 3.94):

Brep: List of breps to be joined and meshed.

IPts: Points on breps or at their boundaries where mesh vertices shall be generated. If a point’s shortest distance to the nearest brep exceeds 0.000 01 m it will be discarded.

MRes: Target size of mesh faces in meter.

Edge Refinement Factor: Multiplication factor for “MRes” that determines the target edge length of faces at brep-boundaries.

Point Reduction: If “True” vertices in overly dense areas get culled. Such regions may result from a distorted UV-space in the underlying brep. Points closer than half of “MRes” to their next neighbor get removed.

SStep and SIter: These two parameters let you control mesh relaxation. Triangular finite shell elements give better results when having sides of approximately equal length. The smoothing algorithm tries to improve the mesh in that direction: It moves each vertex towards the center of gravity of those vertices it connects to. During mesh relaxation vertices always remain on the brep they belong to.

The output of the “MeshBreps”-component consists of meshes with identical vertices at common boundaries. The “Info” output-plug provides information regarding the meshing process.

3.8.2 Closest Points

Assume you have two sets of points: say “P” and “C”. Further imagine you want to have a network that connects each point of the first set with a predefined number of its nearest neighbors in the second set (input “N”) or to points in set two that lie within a given distance “LDist”. In that case the “Nearest Neighbor”-component will be the right choice (see fig. 3.95). It outputs at “P” a data tree with branches containing the points of set “C” which are closest to the “P”-points. The order of branches corresponds to the points in “P”. The output “i” returns the indexes of the nodes in “C”, and “D” the distances.

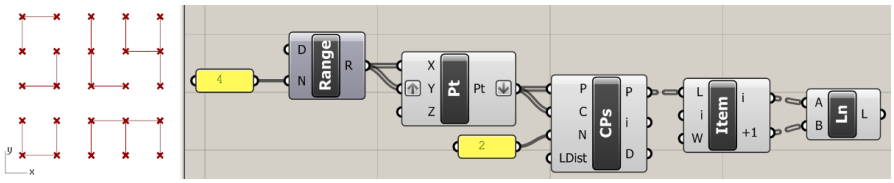

Figure 3.95 A grid of points where each point connects to one or two of its nearest neighbors.

3.8.3 Closest Points Multi-dimensional

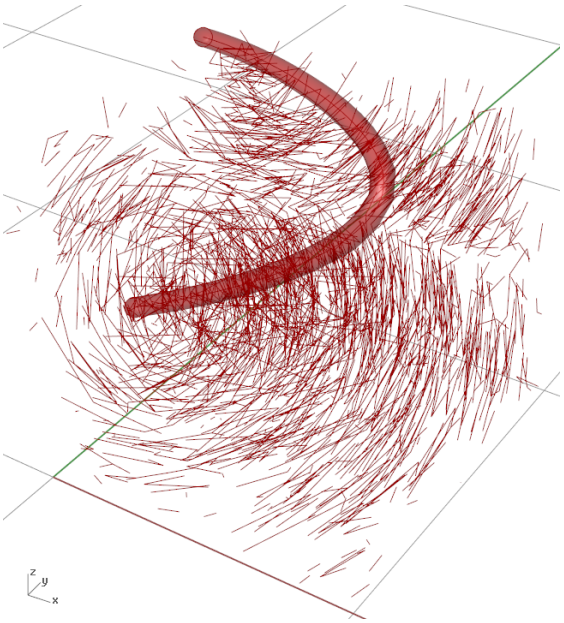

Figure 3.96 Random points in a unit volume connected to their nearest neighbor in a 5-D setting

A nearest neighbor search can be generalized to any number of dimensions. Use the "Closest Point Multi-Dimensional"-component in case of more than three. Fig. 3.96 shows an example with five dimensions: Dimensions one to three represent space. Dimension four is the shortest distance between each point and the thick red guide line. The curve parameter of the guide line at the point where it meets the line of shortest distance acts as fifth dimension. Each of the randomly generated points is connected with its nearest neighbor. One can see from fig. 3.96 that the resulting line segments align to the guide curve – in some way.
There are three input-plugs on the component:

P: Expects a data tree, where each branch contains n values which are the coordinates of the points. The "P"-input specifies points where nearest neighbor connections can start.

C: Expects the same sort of input as "P". It contains the points where nearest neighbor connections can end.

N: number of nearest neighbor connections to be generated for each point in "P"

The output "i" of the "Multi-dimensional Nearest Neighbors"-component is a connectivity diagram.

3.8.4 Cull Curves

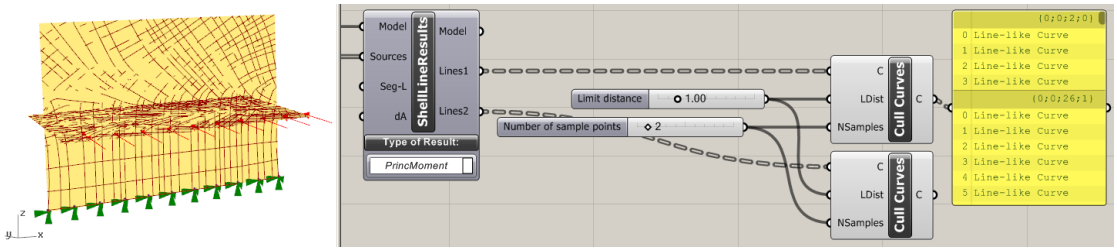

Figure 3.97 A data tree of line segments thinned out based on their mutual distance.

Components like "Line Results on Shells" may result in a dense pattern of lines. The "Cull Curves"-component lets you thin out the line segments based on their mutual distance. The input at "C" expects a data tree, where each branch contains curve segments which make up a stream line. "LDist" sets the minimum distance of curve segments from different branches below which a curve segment gets removed. Proximity is determined on the basis of equidistant control points along the curve segments. By default three points are used per segment. The number of points can be set via the "NSamples"-input.

3.8.5 Detect Collisions

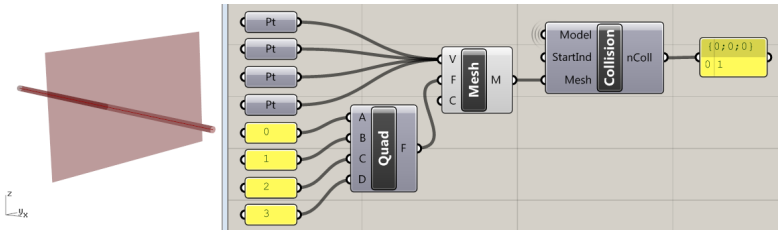

Figure 3.98 Karamba3D model collides with mesh.

In connection with the optimization of structures there is often the need to define volumes where there should be no structure (e.g. windows, rooms). One approach is to choose the parametrization in such a way, that the volumes that should stay free can not be crossed by structure. This is often rather hard to achieve. Another method consists of using penalty functions: The parametrization does not preclude "wrong" solutions, but makes them perform worse than the "right" ones. This is where the "Detect Collision"-component comes into play. It takes a model and a mesh as input and determines the number of collisions between model and mesh. The result is returned in "nColl" (see fig. 3.98). The input-plug "StartInd" accepts the index of the element with which the collision detection should start. By default all elements of the model are included.

3.8.6 Get Cells from Lines

This component takes a connectivity diagram and corresponding nodes and determines all closed cells. Input plug “Pl” can be used to supply planes that are approximately parallel to the cells. The result is a connectivity diagram of the closed cells.

3.8.7 Line-Line Intersection

This component takes a list of lines as input and mutually tests them for intersection. Output-plug “IP” delivers a list of intersection points. “LSS” returns all the line-segments including those which result from mutual intersection. There is a one-to-one correspondence between the list of input lines and the data-tree branches with line-segments at “LSS”. The input-plug “LTol” sets the tolerance length in meter for finding intersections between lines that do not actually intersect. “LTol” is also the minimum length of segments at the ends of lines (e.g. at T-intersections).

3.8.8 Principal States Transformation

With the “Principal States Transformation”-component one can define a tensor via first and second principal vectors “Vec1” and “Vec2” and get the components for a given direction “DirX”. This allows to e.g. calculate the bending moments or in-plane forces of a shell for a section perpendicular to “DirX”. “N” specifies the direction normal of a plane onto which “Vec1” and “Vec2” get projected before performing the transformation.

3.8.9 Remove Duplicate Lines

When you have a list of lines that you suspect of containing duplicate lines then send it through this component and out comes a purified list of ones of a kind. The input-plug “LDist” determines the limit distance for nodes to be considered as identical. Lines of length less than “LDist” will be discarded. Lines that overlap only partly are not detected.

3.8.10 Remove Duplicate Points

Does essentially the same as the component described above – only with points instead of lines.

3.8.11 Simplify Model

Simplifies a model by straightening the connecting elements between nodes that connect to more than two neighbor nodes.

3.8.12 Element Felting

Sometimes one has several (potentially) structural elements neatly positioned in space but no connections between them. The “Element Felting”-component helps out in such situations by generating connections between neighboring elements (see fig. 3.99). The components behavior can be controlled with these input-plugs:

Model: Model to be dealt with

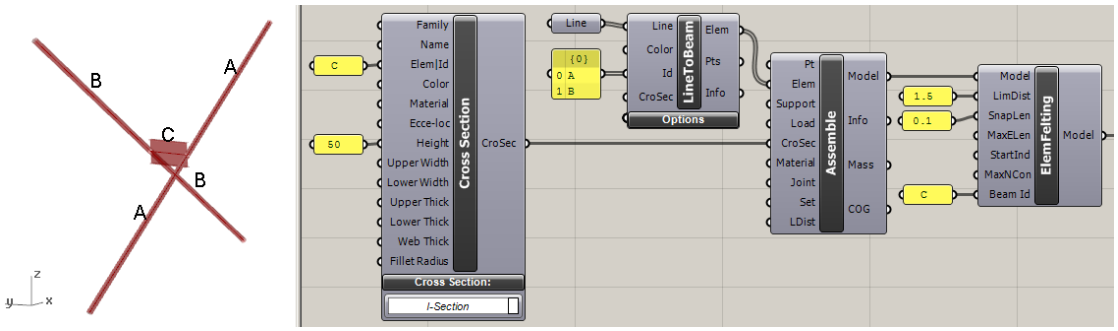

Figure 3.99 The elements "A" and "B" of the original model are connected resulting in the additional element "C".

LimDist: The "Element Felting"-component calculates the shortest distance between each pair of elements in the model. If this distance is less than "LimDist" meters a connection will be generated.

SnapLen: In case that a connection is to be generated the participating elements need to be divided and a connection element introduced. If any of the thus arising elements has a length of less than "SnapLen" meter then the element will be removed and its endpoints snap to the older point of the two.

MaxELen: You can set here a length limit for elements that shall take part in the felting-process. All element longer than the value of "MaxELen" meter will be ignored.

StartInd: Lets you limit the felting process to elements with an index larger than or equal "StartInd".

MaxNCon: This sets the maximum number of new connections to be generated. If this value is reached then felting simply stops.

Beam Id: The beam identifier provided here will be attributed to the connections generated by the component. Cross sections, materials and eccentricities previously defined for this beam identifier apply to these. In case no identifier is given neighboring elements snap to the point in the middle of their shortest connection line.

The felting algorithm proceeds from the first element to the last, always testing against all currently existing elements. Therefore newly generated connection elements may be the source of further connections.

3.8.13 Mapper

A "Mapper" is a component that takes a Karamba3D-model and modifies it according to some generic rules defined by mappings, based on parameters supplied by the user. It acts directly on the model, so the process of transferring Grasshopper-geometry to a Karamba3D-model is dispensed with. The resulting gain of speed can be important when running optimization tasks with e.g. Galapagos.
Fig. 3.100 shows a definition where a mapper applies a mapping called "Simple Stitch" on a given model which originally consists of two elements: "A" and "B". The input-plug "Params" receives two

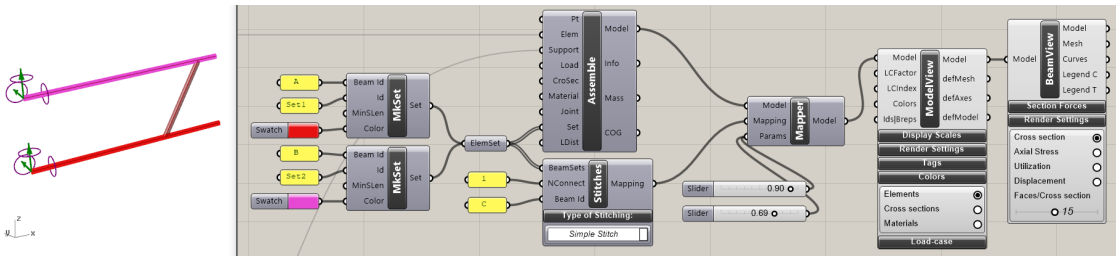

Figure 3.100 The "Mapper"-component applies mappings to a given model. In this case there is one mapping that connects two beam-sets with elements whose position is controlled by the parameters given to the mapper.

parameters. In the context of the "Simple Stitch"-mapping these parameters give the relative position on the two beam-sets "A" and "B" where a connection "C" shall be introduced. So a mapping encapsulates an operation and the mapper activates it [10].
Currently Karamba3D offers mappings which mainly deal with connecting existing beam-sets by variants of an operation termed "stitching". The notion comes from the analogy to joining together pieces of cloth. These mappings will be explained further below. They are rooted in the research project "Algorithmic Generation of Complex Space Frames" which was conducted at the University of Applied Arts Vienna.

3.8.14 Interpolate Shape

The "Interpolate Shape"-component allows one to span a design-space with basic means: One Karamba3D-model, which is directly fed into the "Mapper"-component acts as the origin of that space; an arbitrary number of other models – the input of the "Interpolate Shape"-component – define a coordinate axis each. Linear interpolation takes place between the model at the origin and those defining the axes. A parameter value of 0.0 corresponds to the origin, 1.0 to the model defining the corresponding axis.

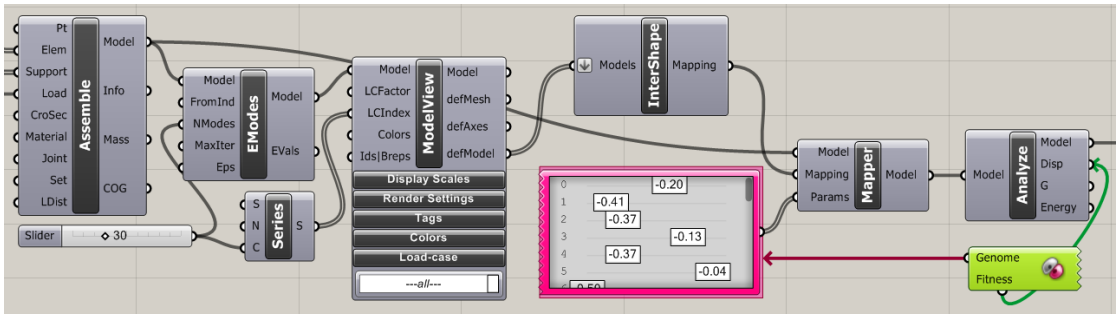

Figure 3.101 Definition for optimizing the shape of a simply supported beam under mid-span single load.

Fig. 3.101 shows a definition where the first 30 eigen-forms (the thin red lines in 3.102) of a simply supported beam serve as the shape-dimensions of the design space.

[10] In other words: a mapping is a functor.

Galapagos is used to determine the position in that design-space which results in minimum deflection under a single load at mid-span. It is clear (think of a hanging model) that the optimum shape has a sharp kink under the load and is otherwise straight.
Fig. 3.102 shows the result of the optimization run which resembles the ideal shape to a large degree. A sharper bend underneath the load could be achieved by including more shape-dimensions in the design space.

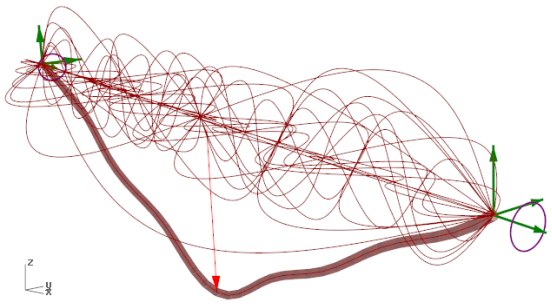

Figure 3.102 Result of shape optimization (thick red line) for a simply supported beam under mid-span single load using the first 30 eigen-forms – the thin red lines – as axes of the design space.

3.8.15 Connecting Beams with Stitches

The "Stitches"-multi-component has three states which correspond to three different connection types. Select the concrete type via the drop-down list at the bottom of the component.

Proximity Stitch

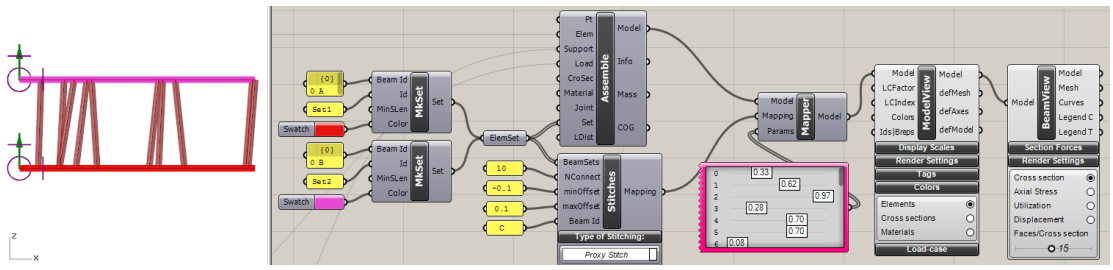

Figure 3.103 "Proximity Stitch"-mapping with the same set-up as in fig. 3.100 but ten random connections instead of two.

The "Proximity Stitch" is a tamed "Simple Stitch" (see sec. 3.8.15): In case of n beam-sets a tuple of n parameters describes one connection. All parameters are in the range $[0, 1]$. The first value p_1 sets the relative location l_1 on the first beam-set. All following parameters p_n relate to the restricted interval $[l_{n-1} - minOffset, l_{n-1} + maxOffset]$. Here "minOffset" and "maxOffset" can be defined by the user. The narrower the interval they define, the more regular the structure.

Simple Stitch

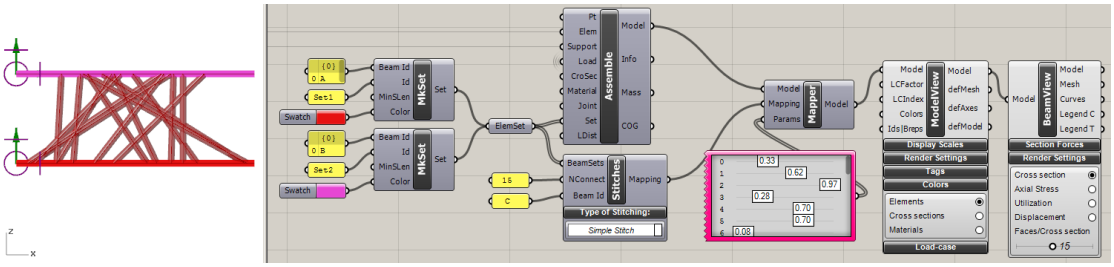

Figure 3.104 "Simple Stitch"-mapping with the same set-up as in fig. 3.100 but 15 random connections instead of two.

A "Simple Stitch"-mapping connects two or more beam-sets with truss or beam-elements. It is available via the "Stitches"-multi-component. The input-plug "BeamSets" expects a list of beam-sets which get connected in the order as they are listed. Double entries of sets are no problem. Via "NConnect" one sets the number of connections. There needs to be one parameter per beam-set and connection for specifying the mapping. The numerical range of parameters should be zero to one: 0 is the starting position of the beam-set, 1 its end. In case you fail to provide the mapper with a sufficient number of parameters it will turn red. Read its error message in order to see how many parameters are needed. The input plug "Beam Id" can be used to define the name of the connection elements. Fig. 3.104 shows a structure with 15 connections resulting from 30 randomly selected parameters.
This "simple"-variant of stitches is also the most versatile one: It gives you great freedom in generating connection patterns by defining the way how a set of parameters is mapped to the set of values that are fed into the "Simple Stitch". The varieties "Proximity Stitch" and "Stacked Stitch" are limiting the scope of possible patters. This leads to faster convergence in case of optimization with e.g. Galapagos and spares you scripting effort but lacks the full flexibility of the "Simple Stitch".

Stacked Stitch

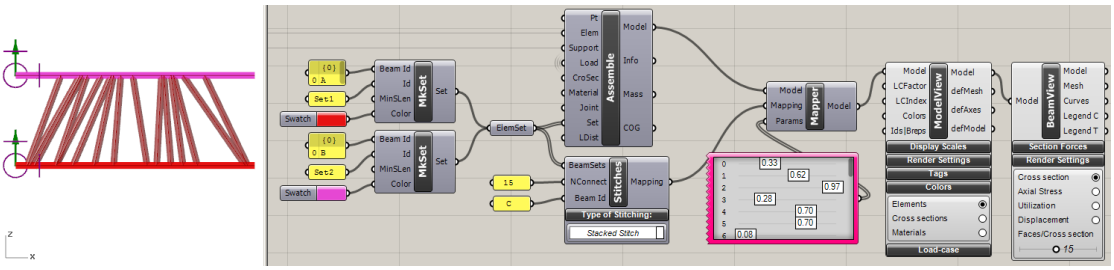

Figure 3.105 "Stacked Stitch"-mapping with the same set-up as in fig. 3.100 but fifteen random connections instead of two.

A "Stacked Stich"-component works along the same lines as a "Simple Stitch". The difference is, that it maps the given parameters to a geometry in such a way, that no two connection elements

cross each other (see fig. 3.105). The input-plug “unevenness” can be used to fine-tune the average irregularity of the result. Zero means totally even: The connection elements are placed at equal distance along the beam-sets (in terms of the corresponding beam-set parameter). The larger the value in “unevenness” the more irregular the layout of connection elements.

3.8.16 User Iso-Lines and Stream-Lines

The components “User Iso-Lines” and “User Stream-Lines” let you draw iso-lines and stream-lines on arbitrary meshes by defining values and vectors at their vertices.

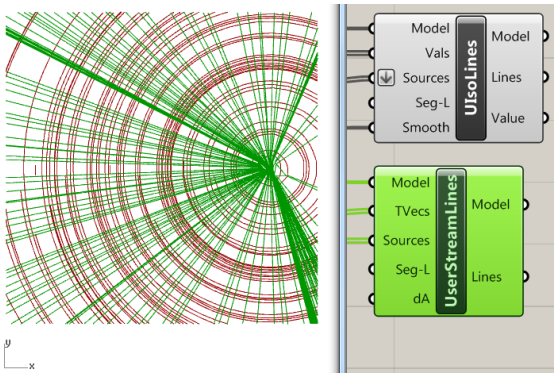

Figure 3.106 User defined Iso-lines (red) and stream-lines (green) on a rectangular shell patch.

The definition “User_Iso_Lines.gh” can be found in the Karamba3D installation folder [5] under “..-Examples/TestExamples/”. It uses a function which when given a point returns the distance from and direction to a predefined source point (see fig. 3.106).
Both components “User Iso-Lines” and “User Stream Lines” work similar to the “IsoLines”- (see section 3.6.12) and “Principal Stress Lines on Shells”-components (see section 3.6.12) respectively. The only difference lies in the fact that for each node of the model (which includes also nodes that only connect to beams) a value (input-plug “Vals”) or vector (input-plug “TVecs”) tangent to the flow needs to be supplied.

[5] Double-click on the Karamba3D desktop icon to get there.

Chapter 4

Trouble shooting

4.1 Do not panic

Do not panic in case some Karamba3D-components turn red. Read the error message. It usually contains information that helps you further. A very efficient technique for tracking errors is to divide and conquer:

1. Split the model in half.
2. Check both parts.
3. Scrutinize the part that does not work.
4. See whether you can find an error in it.
5. If not, take that part as your new model and proceed to point 1.

4.2 Karamba3D does not work for unknown reason

This is the recommended procedure:

1. If a component turns red, read its runtime-error-message.
2. In case that more than one item is plugged into an input, check the incoming data via a panel component.
3. Sometimes flattening the input data helps: The path of input-lists must be consistent. For diagnosis plug them into a Panel-component which will show the path and dimensionality of the data. Another method is to enable “Draw Fancy Wires” in Grasshopper’s View menu: Differently outlined connection lines signify different dimensionality of the data that flows through them.
4. If no results show, check whether preview is enabled on the “ModelView”-, “BeamView”- or “ShellView”-component.
5. If the “Analyze”-component reports a kinematic structure do the following:

- Check the supports for forgotten support conditions.
- Start to fix all degrees of freedom on your supports until the Analyze-component reacts. In case you define a hinge next to a pinned support it could be that the node at the support gets kinematic.
- Introduce additional supports.
- Plug the model into the EigenModes-component. The first eigen-modes will be the rigid body modes you forgot to fix. Save your model before doing that: Large models can take a long time to calculate.
- If the first few eigen-modes seemingly show an un-deflected structure there might be beams in the system that rotate about their longitudinal axis. Enable "Local Axes" in the "ModelView"-component and move the slider for scaling the deformation in order to check this.
- Turn trusses into beams by activating their bending-stiffness (set "Bending" to "True"). Be aware of the fact that a node has to be fixed by at least three trusses that do not lie in one plane.
- Remember that trusses have no torsional or bending stiffness and thus can not serve to fix the corresponding rotations on a beam that attaches to the same node.
- Check whether an element has zero area, height or Young's Modulus.

4.3 Miscellaneous Problems

4.3.1 "fem.karambaPINVOKE"-exception

On some computers the analysis component of Karamba3D refuses to work and throws a "feb.karambaPINVOKE" exception. This may be due to left-overs from previous Karamba3D installations which were not removed properly during the installation procedure. In such a case proceed as follows:

- Uninstall Karamba3D completely via settings/Software/...
- Make sure that everything was removed:
  - Remove "karamba.dll" and "libiomp5md.dll" from the windows folder if they still exist.
  - Search your hard-disk for "karamba.dll"-, "karamba.gha"- and "karambaCommon.gha"-files and remove all occurrences by hand.
- reinstall Karamba3D.

If this does not help do the following:

- Check whether the "karamba.dll"-file in the "Windows"-folder is blocked: right-click on the file and select "Properties" then security.
- Make sure that you installed a Karamba3D version with the correct bitness: "Karamba3D (64bit)" can be used together with "Rhinoceros 5 (64bit)"; "Karamba3D (32bit)" with "Rhinoceros 5". Be aware of the fact that both versions of Rhino 5 get installed. "Rhino 6" and "Rhino 7" come as 64bit-applications only.

This is plan “b” if the above does not help:

- Start Grasshopper
- Type “GrasshopperDeveloperSettings” in the Rhino Window and hit “ENTER”
- Toggle the status of the “Memoryload *.GHA assemblies using COFF byte arrays” option
- Restart Rhino

Plan “c” is to post a help request to the Karamba3D group at `http://www.grasshopper3d.com/group/karamba`.

4.3.2 The “StackedStitch”-components renders structures with overlapping diagonals

Beam-sets have an orientation. You probably use beam-sets with opposing directions.

4.3.3 Karamba3D does not work after reinstalling Grasshoper

Upon installing Grasshopper some files of the Karamba3D package may have been erased. Try to reinstall Karamba3D.

4.3.4 Karamba3D does not appear nor any of its components seem to be installed

In case of multiple versions of Rhino on your machine make sure that you installed Karamba3D to that right versions. Karamba3D can be installed for “Rhinoceros 5 (64-bit)”, “Rhino 6” and “Rhino 7” in parallel. In that case there will be three “Karamba3D” entries in the list of installed applications.

4.3.5 Karamba3D seems to get stuck while calculating a model

Depending on your computer (CPU, size of internal memory) Karamba3D can handle models in the order of 10 000 elements efficiently. If overlong computation times occur check the number of models you actually calculate. Having the path structures of the input-data wrong may lead to multiple models. In such cases flattening or simplifying the input-data helps.

4.3.6 Predefined displacements take no effect

Check whether you disabled the correct degrees of freedom in the “Conditions” section of the “PreDisp”-component.

4.3.7 The “ModelView”-component consistently displays all load cases simultaneously

If the “ModelView”-component does not seem to react to selections done with the drop-down-list for load cases, check the value in the “ResCase”-input plug. Remember that its value is added to the result-case index selected on the drop-down-list. If the sum is negative all load cases will be displayed.

4.3.8 The “View”-components do not show rendered meshes (stress, strain,. . .), supports, etc.

Check whether “Shaded Preview” is enabled in Grasshoppers Solution menu.

4.3.9 The “ModelView”-component does not display any tags

Check whether your Rhino background color is black. Some types of tags are printed in black and do not show on a black canvas. You can change the text color in the “karamba.ini”-file (see section A.2).

4.3.10 Circular cross sections show up as flat stripes when rendered

Set the “Faces/Cross section” slider of the “ModelView”-component to a value larger than two, such that the displayed result sufficiently corresponds to your idea of roundness.

4.3.11 Icons in “Karamba3D”-toolbar do not show up

Sometimes it happens that Karamba3D’s component panels do not display any component icons. Select menu item “View/Show all components” in order to make them show up.

4.3.12 Error messages upon loading definitions saved with outdated Karamba3D versions

When loading definitions based on outdated Karamba3D version a pop-up window will inform you that “IO generated x messages,. . . ”. Normally this can be ignored. It may happen however that very old Karamba3D components do not load. In this case put their current versions in place. Deprecated components have an “old” sign painted over their icon when you zoom in on them.

4.3.13 Component in old definition reports a run-time error

On some components the order of input-plugs changed over time (e.g. the “Assemble”-component). They will turn red when loaded and the runtime error message will state that one object can not be cast to some other object. In this case replace the old component with a new one and reattach the input-plugs accordingly.

4.3.14 The “Optimize Cross Section”-component does not work

Make sure that the beams cross sections you intend to optimize belong to the same family as those you want them to be selected from.

4.3.15 The “Optimize Cross Section”-component returns wrong results

Increase the value at the “ULSIter”- and “DispIter”-input-plug. The cross section optimization algorithm is an iterative procedure. In case you stop too early – having limited the maximum number of iterations to a small value – the algorithm has no chance to converge and thus returns seemingly wrong results. Always check the “Info”-output of the component for information on the solution procedure.

4.3.16 Other problems

In case you encounter any further problems please do not hesitate to contact us at `info@karamba3d.com` or via the Karamba group at `http://www.grasshopper3d.com/group/karamba`.

4.4 How to install Karamba3D for Rhino5, Rhino6 and Rhino7 in parallel

You can install the Rhino5 64-bit version of Karamba3D alongside the the Rhino 6 version. Simply run the two installers one after the other.
It is however not possible to mix the Rhino5 32-bit version of Karamba3D with any of the others.

Appendix A

Appendix

A.1 How to obtain a pro- or pro-student-license

In order to obtain a pro- or pro-student-license go to `http://www.karamba3d.com`. The license agreement and a table of features can be found at `https://www.karamba3d.com/buy/features/`. Details regarding stand-alone-licenses and network-licenses can be found here: `https://www.karamba3d.com/support/standalone-licenses/` and here: `https://www.karamba3d.com/support/network-licenses/`.

A.2 Installation

These are the prerequisites for installing Karamba3D:

- **Rhino 5.0**, **Rhino 6.0** or **Rhino 7.0**
- **Grasshopper**
  version 1.3.2 of Karamba3D was tested on GH 1.0.0007.

In case you do not possess Rhino, download a fully featured, free trial version from `http://www.rhino3d.com/download.html`. Grasshopper for Rhino5 is free and can be found at `http://www.grasshopper3d.com/`. It is already integrated in Rhino6 and Rhino7.

A.2.1 Normal Installation

For installing Karamba3D the usual way download one of the installers and double-click on the msi-file. Make sure to choose the right bitness (32 or 64) of the installer when working with Rhino5. The 64-bit version of Rhino5 is entitled "Rhinoceros 5 (64-bit)". Rhino6 and Rhino7 come as a 64-bit applications only.
The installation procedure lets you set the physical units used for calculation. By default Karamba3D assumes input to be in SI units (e.g. meters for point coordinates). You can switch to Imperial units either on installation or later on by editing the "karamba.ini" file. Coordinates will then be interpreted to be in "feet", force in "kips", material strength in "ksi" and so on.
The following files and folder will be copied to your machine:

- "karamba.dll" and "libiomp5md.dll" to `C:\Windows`.

- "karambaCommon.dll", "karamba.gha" and the Karamba-folder to the "Plug-ins"-folder of Rhino. The Karamba-folder contains material- and cross section libraries, examples, the karamba.ini-file and the license-folder.

This installs Karamba3D for all users by default. In order to install Karamba3D without the installer simply copy the above files and the Karamba-folder from one machine to the next.
Unless deselected, the installer places a Karamba3D-icon on the desktop. Double-click on it to open the Karamba-folder. Karamba3D can be installed in parallel for Rhino5(64bit), Rhino6 and Rhino7.

A.2.2 Silent Installation

In order to install Karamba3D on remote machines "msiexec.exe" provides options to circumvent the graphical user-interface of the installer. Start the Windows CommandPrompt, navigate to where the Karamba3D-installer lies and type e.g.

```
msiexec.exe /i karamba3d_1_3_2_RH7.msi /passive
   ADDLOCAL=FullFeatures,SIUnits,LicensePublicKey,Tables,Examples
```

in one line to install the full version of Karamba3D ("FullFeatures") with SI-Units ("SIUnits"), cross section and material tables ("Tables") and the example definitions ("Examples"). In order to get the free version and Imperial units substitute "FullFeatures" with "FreeFeatures" and "SIUnits" with "IMPUnits".
The static license can be supplied without graphical user interface (GUI) by renaming the license-file to "licensePRO.lic" and copying it to the "License"-folder under `...\Rhino\Plug-ins\Karamba\`.

A.3 What's new in version 1.3.2. with respect to version 1.3.1

A.3.1 New features which change the behavior of older definitions

- The definition of cross sections and materials via the "Assemble"-component has now priority over definition at components which generate elements.

A.3.2 New features in Karamba3D 1.3.2

- Karamba3D can be used as a scripting library independently from Grasshopper. It now comes with its own geometry classes.
- A project for running automatic Karamba3D tests under NUnit has been published. See `https://github.com/karamba3d/K3D_tests`.
- The cross section library has been extended considerably and comprises standard steel sections from the EU, UK, US, Russia, Japan, India, Canada, Germany, Australia, China, South Africa, Brazil and Korea.
- It is possible to define initial-curvature and non-uniform temperature loads on beams and shells.
- There is a Karamba3D version for Rhino7.

- A model's nodal positions can be changed via the "ModifyModel"-component. This is necessary since the "DisassembleModel"-component outputs elements with nodal coordinates attached.
- "Joints" have now their own class and container.
- All known bugs of Karamba3D 1.3.1. were fixed.

For the list of changes of all versions of Karamba3D see `https://www.grasshopper3d.com/group/karamba3d/page/new-features-and-bug-fixes`

A.3.3 Known limitations

- The contribution of locally defined loads to the system stiffness is not considered in second order analysis or the calculation of buckling load factors.

A.3.4 Third party products

Jon Mirtschin implemented an exporter from Karamba3D to IFC. This tool is part of his "GeometryGym" suit of Grasshopper plug-ins. For details see `http://geometrygym.blogspot.co.at/`.

A.4 Background information

A.4.1 Basic Properties of Materials

Material Stiffness

The stiffness i.e. resistance of a material against deformation is characterized by its Young's Modulus or modulus of elasticity "E". The higher its value the stiffer the material. Table A.1 lists E-values for some popular building materials.

type of material	E $\mathrm{kN/cm^2}$
steel	21000
aluminum	7000
reinforced concrete	3000
glass fiber	7000
wood (spruce)	1000

Table A.1 Young's Modulus of materials

For composite materials – like in the case of rods made from glass fiber and epoxy – it is necessary to defer a mean value for E using material tests. Karamba3D expects the input for E to be in kilo Newton per square centimeter ($\mathrm{kN/cm^2}$).
If one stretches a piece of material it not only gets longer but also thinner: it contracts laterally. In case of steel for example lateral strain amounts to 30 % of the longitudinal strain. In case of beams with a large ratio of cross section height to span this effect influences the displacement response.

In common beam structures however this effect is of minor importance. The shear modulus "G" describes material behavior in this respect.

Specific Weight

The value of "gamma" is expected to be in kilo Newton per cubic meter ($\mathrm{kN/m^3}$). This is a force per unit of volume. Due to Earths gravitational acceleration ($a = g = 9.81\,\mathrm{kg\,m/s^2}$) and according to Newtons law ($f = m \cdot a$) a mass m of one kilogram acts downwards with a force of $f = 9.81\,\mathrm{N}$. For calculating deflections of structures the assumption of $f = 10\,\mathrm{N}$ is accurate enough. If you want a more precise value change the entry "gravity" in the "karamba.ini"-File. In case of Imperial Units the exact value for "gravity" is automatically set – otherwise the conversion from lbm to lbf does not work properly.
Table A.2 gives specific weights of a number of typical building materials. The weight of materials only takes effect if gravity is added to a load case (see section 3.2.1).

Theoretical Background of Stiffness, Stress and Strain

As mentioned in section 3.5.1 strain is the quotient between the increase of length of a piece of material when loaded and its initial length. Usually one uses the Greek letter ε for strains.
Stress is force per unit of area. From the stress in a beam cross-section one can calculate the normal force that it withstands by adding up (integrating) the product of area and stress in each point of the cross-section. Stress is normally symbolized by the Greek letter σ. Linear elastic materials show a linear dependence between stress and strain. The relation is called Hooke's Law and looks like this:

$$\sigma = E \cdot \varepsilon$$

"E" stands for Young's Modulus which depends on the material and depicts its stiffness. Hooke's law expresses the fact that the more you deform something the more force you have to apply.

A.4.2 Additional Information on Loads

Karamba3D expects all force-definitions to be in kilo Newton (kN). On earth the mass of 100 kg corresponds to a weight force of roughly 1 kN. The exact number would be roughly 0.981 kN but 1 kN is normally accurate enough. Table A.2 contains the specific weight of some everyday materials. Rules of thumb numbers for loads can be found in table A.3. Do not take these values too literally. For example snow loads vary strongly depending on the geographical situation.
Loads acting along lines or on a specified area can be approximated by point-loads. All you need to do is estimate the area or length of influence for each node and multiply it with the given load value. The Mesh-Load-component (see section 3.2.1) automates this task for surface loads.

A.4.3 Tips for Designing Statically Feasible Structures

Karamba3D can be used to analyze the response of structures of any scale. When using the "Analysis"-component for assessing the structural behavior be aware of two preconditions: First, deflections are small as compared to the size of the structure. Second, materials do behave in a linear elastic manner – i.e. a certain increase of deformation is always coupled to the same increase of load. Real materials behave differently: they weaken at some point and break eventually.

type of material	kN/m^3
reinforced concrete	25.0
glass	25.0
steel	78.5
aluminum	27.0
fir wood	3.2
snow loose	1.2
snow wet	9.0
water	10.0

Table A.2 Specific weights of some building materials

loads	
type	kN/m^2
life load in dwellings	3.0
life load in offices	4.0
snow on horizontal plane	1.0
cars on parking lot (no trucks)	2.5
trucks on bridge	16.7

Table A.3 Loads for typical scenarios

If you want to calculate structures with large deflections you have to increase the load in several steps and update the deflected geometry. This can be done with the "Large Deformation Analysis"-component (see section 3.5.4) or the component for geometrically non-linear analysis "AnalyzeNonlin WIP".

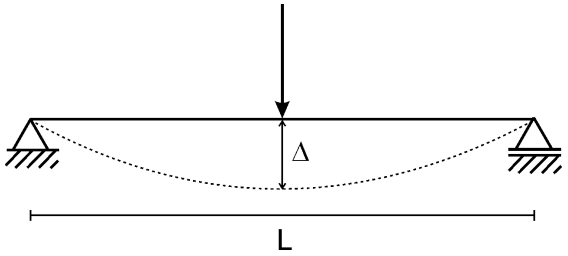

Figure A.1 Simply supported beam.

For typical engineering structures the assumptions mentioned above suffice for an initial design. In order to get meaningful cross section dimensions limit the maximum deflection of the structure. Figure A.1 shows a simply supported beam of length L with maximum deflection Δ under a single force at mid-span. The maximum deflection of a building should be such that people using it do not

start to feel uneasy. As a rough rule of thumb try to limit it to $\Delta \leq L/300$. If your structure is more like a cantilever $\Delta \leq L/150$ will do. This can normally be achieved by increasing the size of the cross-sections. If deflection is dominated by bending (like in figure A.1) it is much more efficient to increase the height of the cross-section than its area (see section 3.1.10). Make sure to include all significant loads (dead weight, live load, wind, ...) when checking the allowable maximum deflection. For a first design however it will be sufficient to take a multiple of the dead-weight (e.g. with a factor of 1.5). This can be done in Karamba3D by giving the vector of gravity a length of 1.5.
In case of structures dominated by bending, collapse is preceded by large deflections (see for example the video of the collapse of the Tacoma-Narrows bridge at `http://www.youtube.com/watch?v=3mclp9QmCGs`. So limiting deflection automatically leads to a safe design. If however compressive forces initiate failure, collapse may occur without prior warning. The phenomenon is called buckling. Using the "Analysis"-component it makes no difference whether an axially loaded beam resists compressive or tensile loads: it either gets longer or shorter and the absolute value of its change of length is the same. In real structures the more slender a beam the less compressive force it takes to buckle it. An extreme example would be a rope. In case buckling might occur, use the "AnalysisThII"-component which takes into account the destabilizing effect of compressive axial forces. The "Buckling Modes"-component lets you then compute the first buckling load-factor. This is the factor with which the external loads need to be multiplied for initiating linear buckling.

A.4.4 Hints on Reducing Computation Time

Karamba3D spends most of its time solving for the deflections of a model. The time needed depends on the number of degrees of freedom n of the statical system and how many connections exist between the nodes. In the theoretical case that each node is connected to all others computation-time grows with n^3. If each node is connected to n_{neigh} others and the overall structure has a main axis along which it is oriented (i.e. there are no connections between distant nodes), then computational effort increases approximately with $0.5 \cdot n \cdot n_{neigh}^2$. Karamba3D makes use of multiple processors so having more than one saves time. Using trusses instead of beams more than halves computation time.
When doing optimization with e.g. Galapagos the continuous display updates slow down things considerably. For better performance disable the preview of all components or minimize the Rhino window.
The rendering of results as meshes can be sped up considerably by joining the meshes before displaying them.

A.4.5 Natural Vibrations, Eigen Modes and Buckling

The Eigen-modes of a structure describe the shapes to which it can be deformed most easily in ascending order. The first mode is the one which can be achieved most easily. The higher the mode number the more force has to be applied. Due to this the "Eigen Modes"-component can be used to detect kinematic modes.
An Eigen-mode $\vec{x}$ is the solution to the matrix-equation $\underset{\sim}{C} \cdot \vec{x} = \lambda \cdot \vec{x}$ which is called the special eigen-value problem. Where $\underset{\sim}{C}$ is a matrix, $\vec{x}$ a vector and λ a scalar (that is a number) called eigen-value.
The whole thing does not necessarily involve statical structures. Eigen-modes and eigen-values are

intrinsic properties of a matrix. When applied to structures then $\underset{\sim}{C}$ stands for the stiffness-matrix whose number of rows and columns corresponds to the number of degrees of freedom of the statical system. $\vec{x}$ is an eigen-mode as can be computed with Karamba3D.
Vibration modes $\vec{x}$ of structures result from the solution of a general Eigenvalue problem. This has the form $\underset{\sim}{C} \cdot \vec{x} = \omega^2 \cdot \underset{\sim}{M} \cdot \vec{x}$. In a structural context $\underset{\sim}{M}$ is the mass-matrix which represents the effect of inertia. The scalar ω can be used to compute the eigen-frequency f of the dynamic system from the equation $f = \omega/2\pi$. In the context of structural dynamics eigen-modes are also called normal-modes or vibration-modes.
The "Buckling Modes"-component calculates the factor with which the normal forces N^{II} need to be multiplied in order to cause structural instability. The buckling factors are the eigenvalues of the general Eigenvalue problem $\underset{\sim}{C} \cdot \vec{x} + \lambda^2 \cdot \underset{\sim}{C_G} \cdot \vec{x} = 0$. Here $\underset{\sim}{C}$ is the elastic stiffness matrix and $\underset{\sim}{C_G}$ the geometric stiffness matrix. The latter captures the influence of normal forces N^{II} on a structure's deformation response.

A.4.6 Approach Used for Cross Section Optimization

Karamba3D does cross section design by going through the list of cross sections in a group of cross sections called a family. It starts at the first entry and proceeds to the next until a cross section is found that is sufficient for the given cross section forces of an element.
The calculation of the utilization of a beam is done using the procedure described in EN 1993-1-1 i.e. Eurocode (EC) 3. It takes account of buckling and lateral torsional buckling. The superposition of different cross section forces works according to procedure 2 (see annex B of EC 3). The values C_{my}, C_{mz} and C_{mLT} are limited to 0.9. The "Utilization"-component's "Details"-output returns a listing of intermediate values from the calculation according to EC3 (see section 3.6.6). For a detailed comparison of the steel design procedures implemented in Karamba3D and Dlubals RFEM see [9].

Bibliography

[1] J. H. Argyris, L. Tenek, and L. Olofsson. Tric: a simple but sophisticated 3-node triangular element based on 6 rigid.body and 12 straining modes for fast computational simulations of arbitrary isotropic and laminated composite shells. *Comput. Methods Appl. Mech. Engrg.*, 145: 11–85, 1997.

[2] J.H. Argyris, M. Papadrakakis, C. Apostolopoulou, and S. Koutsourelakis. The tric shell element: theoretical and numerical investigation. *Comput. Methods Appl. Mech. Engrg.*, 182:217–245, 2000.

[3] Andrew W Beeby and RS Narayanan. *Designers' Guide to EN 1992-1-1 and EN 1992-1-2. Eurocode 2: Design of Concrete Structures: General Rules and Rules for Buildings and Structural Fire Design*. Thomas Telford, 2005.

[4] Johan Blaauwendraad. *Plates and FEM*. Springer, 2012.

[5] BSI. Bs en 1993-1-1: 2005: Eurocode 3. design of steel structures. general rules and rules for buildings., 2005.

[6] Fédération Internationale du Béton. Practitioners guide to finite element modelling of reinforced concrete structures. *State-of-Art Report*, 2008. URL `https://www.istructe.org/fibuk/files/fib_bull45_nmg.pdf`.

[7] X. Huang and M. Xie. *Evolutionary Topology Optimization of Continuum Structures: Methods and Applications*. Wiley, 2010.

[8] MT Huber. The theory of crosswise reinforced ferroconcrete slabs and its application to various important constructional problems involving rectangular slabs. *Der Bauingenieur*, 4(12):354–360, 1923.

[9] Mäenpää Jukka. Algorithm-aided structural engineering of steel-framed warehouse. Master's thesis, Tampere University of Technology, 2018. URL `https://dspace.cc.tut.fi/dpub/handle/123456789/25580`.

[10] Rubin H. Schneider K.-J. *Baustatik Theorie I. und II. Ordnung*. Werner-Verlag, 1996.

[11] H. Moldenhauer. Die visualisierung des kraftflusses in stahlbaukonstruktionen. *Stahlbau, Ernst & Sohn Verlag für Architektur und technische Wissenschaften GmbH & Co. KG, Berlin*, 81:32–40, 2012.

[12] Robert Woodbury. *Elements of Parametric Design*. Taylor & Francis Ltd, 2010. ISBN 0415779871. URL `https://www.ebook.de/de/product/10781735/robert_simon_fraser_university_canada_woodbury_elements_of_parametric_design.html`.