Peregrine User Manual

v7.1

Linwei He, Thomas Pritchard, Jack Maggs,
Matthew Gilbert and Hongjia Lu

limitstate

Page intentionally blank

Contents

List of Figures

1 Introduction

1.1 About Peregrine

Peregrine is a powerful structural layout optimization plugin for Grasshopper, the algorithmic modelling environment built into the Rhino 3D modelling software. Peregrine can be used to identify efficient (minimum volume) layouts of elements forming a truss, for a given design domain and set of loads, supports and material properties. It includes a range of tools that enable the practicality and efficiency of generated designs to be balanced.

Peregrine was originally developed in association with the University of Sheffield, Arup and other partners as part of the UK government-funded BUILD-OPT research project. This collaborative research project involved the Universities of Sheffield, Bath and Edinburgh, AECOM, Arup, BuroHappold, Expedition Engineering, IStructE, Ramboll and the Steel Construction Institute.

Example files can be accessed from:

- The Peregrine Examples desktop shortcut (installation option)
- Start > Peregrine > Example Files
- In the Peregrine folder within Rhino > Plug-ins

For more information, visit limitstate.com/peregrine

To purchase a license, visit shop.limitstate.com/peregrine

1.2 About LimitState

LimitState Ltd was spun out from the University of Sheffield in 2006 to develop and market cutting edge ultimate analysis and design software for engineering professionals. Peregrine takes advantage of LimitState:FORM technology, with the aim of making this useful for building optimization and related problems. This is one of a number of LimitState products, with applications in the structural, geotechnical and mechanical engineering sectors. LimitState aims to be a world leading supplier of computational limit analysis and design software. The company maintains close links with the University of Sheffield, enabling us to draw on and rapidly implement the latest innovations.

For more information, visit limitstate.com/our-company

1.3 Disclaimer

Peregrine undergoes a wide array of tests before release. Notwithstanding this, bugs may exist and therefore no guarantee can be given that the results generated are correct. Use of

Peregrine is therefore at the user's own risk. Please refer to the license agreement for further details.

1.4 Referencing

To reference this document, please use:

He L, Pritchard T, Maggs J, Gilbert M and Lu H (2021) *Peregrine User Manual*, LimitState Limited, Sheffield.

2 Installation & Licensing

2.1 Prerequisites

These are the prerequisites for installing Peregrine:

- Rhino 7 SR36 or above (Peregrine Rhino 7 Installer)
- Rhino 8 SR7 or above (Peregrine Rhino 8 Installer)

If a copy of Rhino is not already installed, it can be downloaded from the Rhino3D website (rhino3d.com); a fully functional trial version is available. Grasshopper is integrated into Rhino 6 and above.

2.2 Installation

Installation of Peregrine is a straightforward process:

> Download the Peregrine installer from www.food4rhino.com/en/app/peregrine

> Double-click on the executable to start the installation process

> Follow the instructions provided by the installer

2.3 Licensing

Following installation of Peregrine, a new category called Peregrine should be visible as a tab on the component panel (Figure 1) when starting Grasshopper (GH).

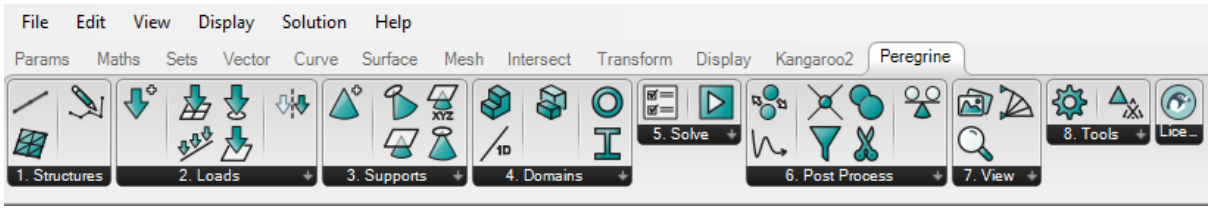

Figure 1 – The Peregrine tab on the Grasshopper component panel (ribbon)

If no icons are visible under the Peregrine tab, select “Draw All Components” in the Grasshopper “View” menu.

Detailed instructions for installing and licensing Peregrine are provided in the following section.

2.3.1 The License Component

The Peregrine License component can be found at the right of the Peregrine toolbar (Figure 1). After dragging on to the canvas, right-clicking the component name provides the user with the ability to select between a number of different licensing systems (or Types), as seen in Figure 2:

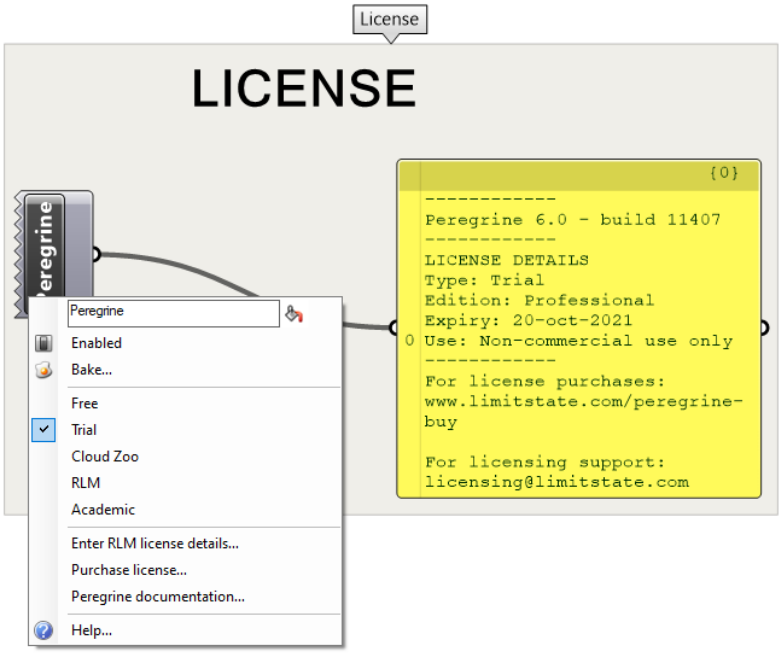

Figure 2 – Licensing options in the Peregrine License dialog

Connecting the output of the License component to a Panel will display information relating to the Peregrine version and currently active license:

- Version and build number
- Type (relating to the method being used to license the plug-in)
- Edition (relating to the functionality that is available)
- Use (relating to the type of usage that the license is valid for e.g. ‘non-commercial use only’ or ‘all’)

To change the type of license that is active, open the context menu and select the desired option. Note that this might cause some components in an active document to display error messages relating to e.g. functionality that is not allowed. If this relates to the previous license, right-clicking in free space on the canvas and selecting ***Recompute*** should update the document. Similarly, you can also remove and re-connect a wire to the component in question.

2.3.2 License Types

2.3.2.1 Free

A ***Free*** license provides access to the most popular functionality of the plug-in, but restricts the number of nodes that can be specified in the problem to 200 as well as a number of other limitations (see www.limitstate.com/peregrine-buy for more details).

If no other valid license is found, the plug-in will run with a ***Free*** license by default.

If the user has attempted to specify a different license type, but the plug-in cannot find a valid license of that kind, a ***Free*** license will be used, so that the basic functionality remains available. However, the ***License*** component will warn that the requested license type is not active, so that appropriate measures can be taken.

2.3.2.2 Cloud Zoo

Cloud Zoo licensing (or Zoo for short) is the most common method of licensing the plug-in. It makes use of the McNeel Rhino Cloud Zoo infrastructure.

Switching to a ***Cloud Zoo*** license will query your Rhino installation and/or your Rhinoceros User Account to determine if you have a valid Peregrine license. If you have not yet activated a Peregrine license key on ***Cloud Zoo***, you will need to obtain one from LimitState (see shop.limitstate.com/peregrine). Adding the license to your ***Cloud Zoo*** account is then a matter of logging in and following the "Add License" procedure. For more information on this, please refer to limitstate.com/peregrine-cloud-zoo-licensing.

You can check if you have access to a Cloud Zoo license from within Rhino, by going to:

File > Properties > Licenses

This will also indicate how long the license will remain active should your machine be taken offline.

2.3.2.3 RLM

Reprise License Manager (***RLM***) is an alternative to using Cloud Zoo. It requires either:

1. A standalone license (.lic) file on the host machine, or
2. A connection to a networked license server machine with the RLM software installed and a valid network license file.

If a valid ***RLM*** license is not available, selecting the ***RLM*** option in the context menu will bring up a dialog (Figure 3) that allows the user to either:

- Enter an ***RLM*** license activation key, or
- Enter a network location or a local license path

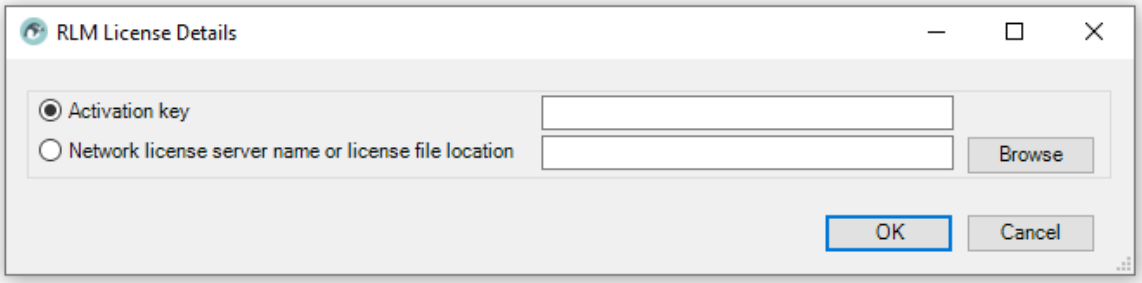

Figure 3 – The Peregrine RLM license details dialog

To obtain a ***RLM*** license of any kind, contact LimitState. Instructions for installing and maintaining a ***RLM*** network license server are available in the Network Licensing Guide.

2.3.2.4 *Academic*

Academic licenses are made available to users on academic networks, wishing to use Peregrine only for research and teaching purposes. The system checks to see if the requesting machine is on a recognized academic network and, if it is, serves a 30 day license, so that the user can work offline.

To check if your institution is currently recognized, on a machine that is connected to the organization's network, visit www.limitstate.com/academic-licensing and use the "Check Academic License Availability" feature. Note that some institutions use a "split tunneling" VPN system, which will require calls to http://acad.limitstate.com to be specifically set to use the IP of the institution.

If your institution is not recognized by our systems, please get in touch to request that it is added.

2.4 License Editions

The availability of certain Peregrine functionality is controlled by the license edition. More information on the features available for each license edition, visit our website: www.limitstate.com/peregrine-buy

Functionality that is restricted by license edition is referred to herein using red boxes. In the plug-in, error messages on a component will warn the user when a feature is not available, for example as shown in Figure 4:

The editions available are:

2.4.1 Trial

Upon first installation, the software will issue a ***Trial*** license with a duration of 14 days. This provides access to the functions associated with a ***Professional*** license. After the 14 days have

elapsed, the software will revert to a ***Free*** license, unless another license type has been provided.

Note that the ***Trial*** license is node-locked and only available for an initial 14 day period following installation. Reinstallation of the plugin will not reset the license and altering the system clock will prevent the software from working and is likely to have unwanted repercussions for other applications.

2.4.2 Free

As detailed in Section 2.3.2.1, a ***Free*** license provides access to core functionality, but with some restrictions. If the free license has expired, please check the [Peregrine page on Food4Rhino] for an updated installer or licensing instructions.

2.4.3 Standard

A ***Standard*** license Provides access to the most popular functionality of the plug-in and does not restrict the number of nodes and load cases available to use.

2.4.4 Professional

A ***Professional*** license provides access to all popular and advanced functionality of the plug-in.

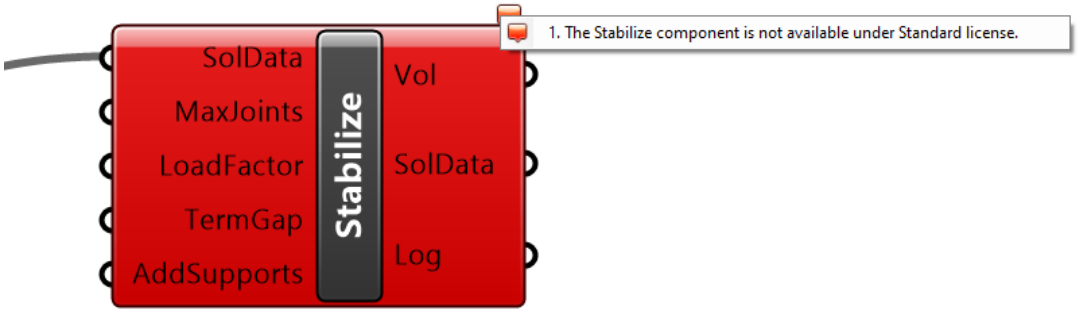

Figure 4 – Error message in the Stabilize component due to license restrictions

3 Quickstart

3.1 Basic Optimization Requirements

In order to undertake an optimization in Peregrine, a number of model features need to exist:

1. A **design domain**, within which the optimized structure will be situated.
2. **Material properties**, such that the optimum structure can be correctly dimensioned.
3. **Support conditions**, to provide sufficient restraint.
4. One or more **loads** for the optimized structure to resist.

The components within Peregrine allow these features to be quickly defined and used as the basis of an optimization problem. Further functionality then allows the refinement and editing of the output from the optimization, in order to assist the user in realizing a structure that is potentially more practically useful for their needs.

3.2 Example Files Location

Peregrine is installed with a number of example files for you to use and gain an understanding of the software. By default, these are located in the Rhino installation directory, under:

> *<Rhino>\Plug-ins\Peregrine\examples*

Files used in the Quickstart are available within the '*Free* 'subdirectory. Other example files are arranged into subdirectories according to the license type required to use the functionality being demonstrated.

Example files can be accessed from:

- The Peregrine Examples desktop shortcut (installation option)
- Start > Peregrine > Example Files
- In the Peregrine folder within Rhino > Plug-ins

3.3 Starting from a Grasshopper Defined Geometry

This example builds the design domain for use in the optimization using a geometry defined entirely within the Grasshopper environment.

> Open the example file *Simple_2D.gh* in Grasshopper.

3.3.1 Design Domain

3.3.1.1 Geometry

In the Rhino environment, a cantilever problem with a rectangular design domain will have been generated (but no optimized solution exists as yet). The cantilever is supported on the left edge and loaded on the right (Figure 5).

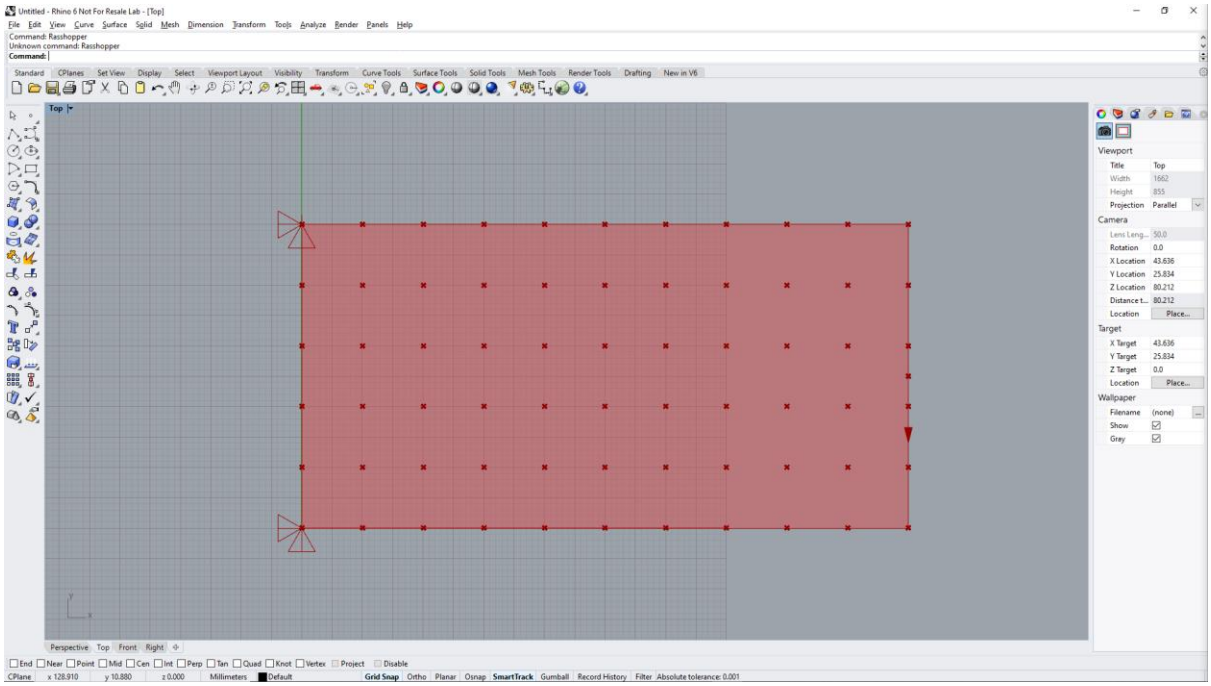

Figure 5 – Design Domain – Geometry Defined within Grasshopper

We shall more closely examine the way in which the domain was generated.

Within Grasshopper, zoom into the area containing the Geometry component (top left, Figure 6)

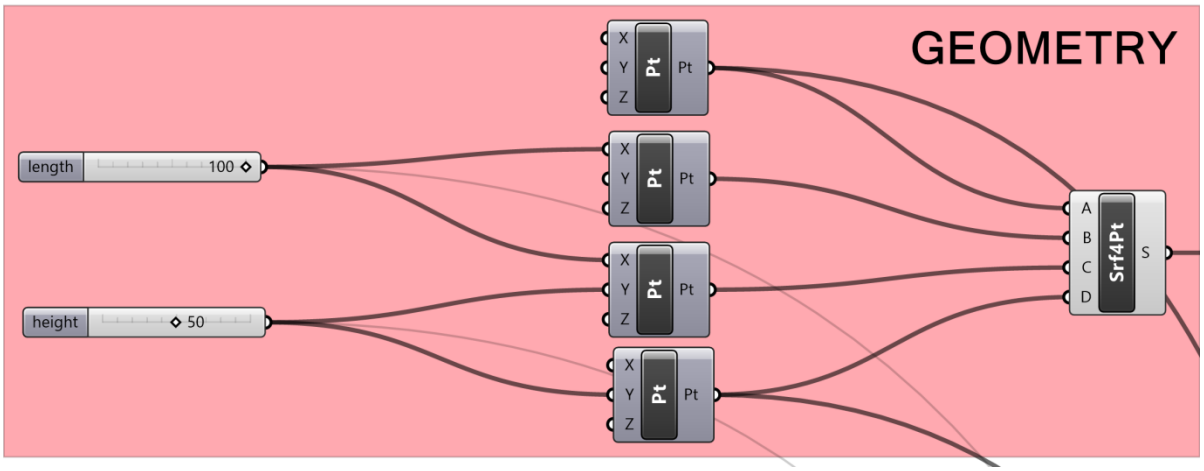

Figure 6 – Definition of the Design Domain within Grasshopper

You will see that four points (***Vector*** > ***Construct Point***) have been specified. Each point is defined by an X and Y co-ordinate (Z is omitted in this instance) and these are controlled by ***length*** and ***height*** sliders. The co-ordinates are then combined as the input to define a ***4 Point Surface*** (accessed from the ***Freeform*** group in the ***Surface Component*** panel). In this way, the size of the design domain can be altered by adjusting the sliders alone.

3.3.1.2 *Material Properties*

Tracking the output (***S***) from the ***4 Point Surface*** (***Srf4Pt***) it can be seen that the resultant ***Surface*** is used as The ***Mesh*** input into the ***Domain***. Other inputs include the ***Material Properties*** (***Mat***).

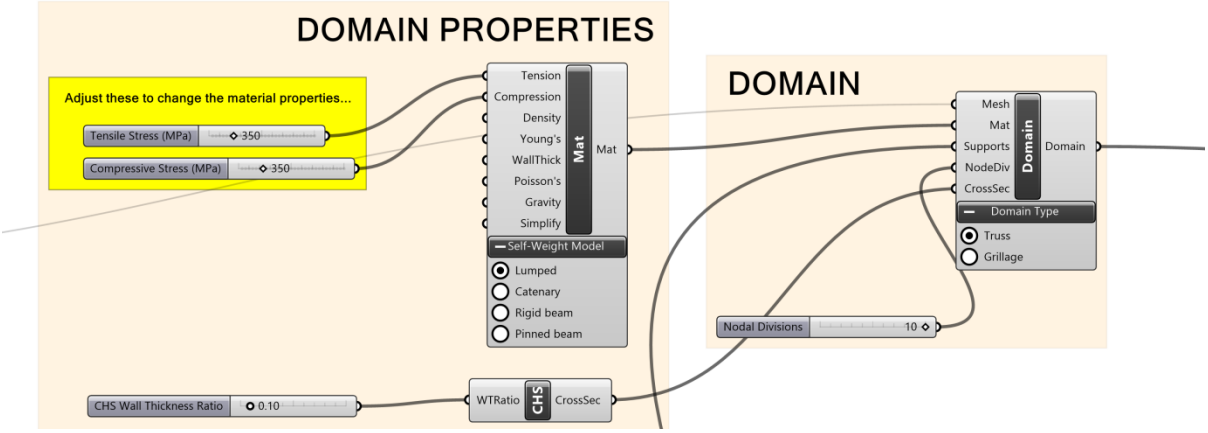

Figure 7 – Specification of the Design Domain

The Material properties and how they affect the results of the optimization are discussed in detail in the Component Reference section (Section 5).

3.3.1.3 *Supports*

Another input into the ***Domain*** is the ***Supports***. In this particular problem, the supported locations correspond to the two points that define the left boundary of the Design Domain [(0, 0) and (0, 50)]. As such, the input to each ***Point Support*** (***PointS***) is the output from one of these original ***Point*** (***Pt***) components.

The remaining variable inputs are Boolean operators representing the restraint provided by that support for each of the Cartesian directions. By default, restraint is assumed to be set to *True* and therefore, as there is no additional input to change this state, the two ***Point Supports*** are assumed fully fixed. The output from the ***Supports*** component is then used as input into the ***Supports*** channel of the ***Domain*** component.

3.3.1.4 *Nodal Divisions*

The final input into the ***Domain*** component is a slider that outlines the number of nodes used in the optimization problem – represented by the number of equally spaced divisions (***NodeDiv***) into which the Domain is divided (see Section 5.4.1.4).

3.3.2 Loading

Loading in Peregrine can be applied at a ***Point*** (***PointL***), across a ***Surface*** (***SurfaceL***) or along a ***Line*** (***LineL***). Individual loads, or groups of loads, can be made into separate ***Load Cases*** for input into the ***LCase*** component.

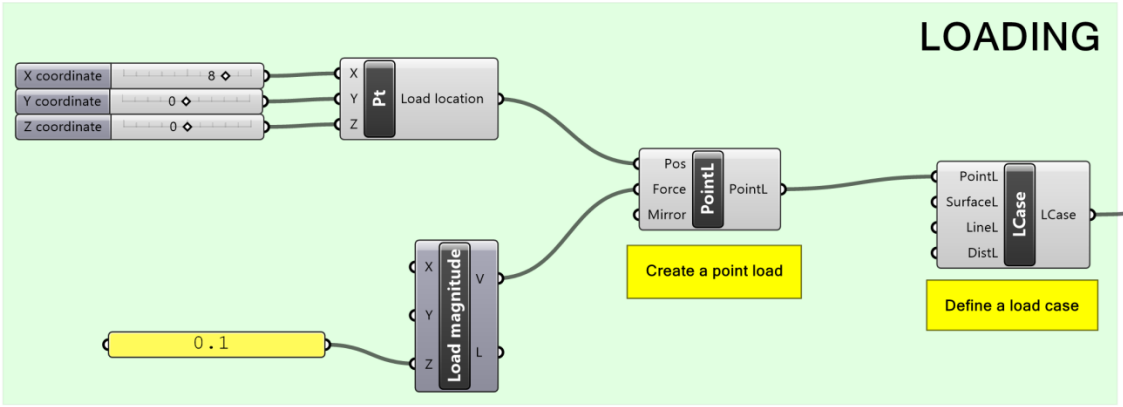

Figure 8 – A Point Load added as a Load Case

The Point Load (***PointL***) accepts a ***Position*** (***Pos***) and a ***Magnitude*** (***Mag***). In this problem, a ***PointL*** of unit magnitude is applied in the downwards (negative) Y direction at mid-height on the right edge of the design domain. The controls for this are situated in the "Point load" group and take further input from the ***Geometry*** settings.

3.3.3 Solving the Optimization Problem

All the necessary aspects of the problem are now present. We have defined:

- Design domain geometry
- Loading
- Support
- Material properties

The listed features are combined as input into the ***Problem Specification*** (***ProbSpec***) component (as the ***Domain*** and ***Load Case***). The output from this component is then used as the ***ProbSpec*** input into the ***Layout Optimization (LO)*** component (Figure 9). It is this latter component which controls the optimization. A toggle is added to the ***Enable*** input, which can then be used as a switch to turn on (and off) the optimizer kernel. Other inputs (e.g. ***JointCost***) are optional controls on the problem setup and are explained later in this document.

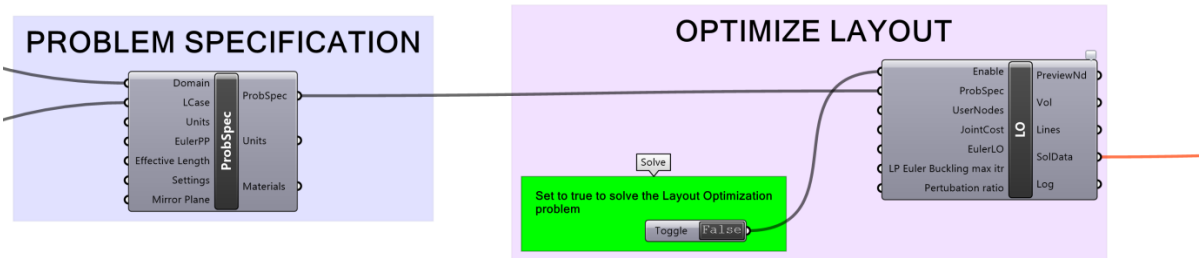

Figure 9 – The Problem Specification and Layout Optimization components

> To solve the optimization problem, toggle the "Enable" switch.

Once the problem has been solved (which should take less than a second), the optimized truss layout can be displayed in Rhino. This occurs by connecting the ***Solution Data*** (***SolData***) output from the ***LO*** component to the ***SolData*** input of the ***View*** component (Figure 10):

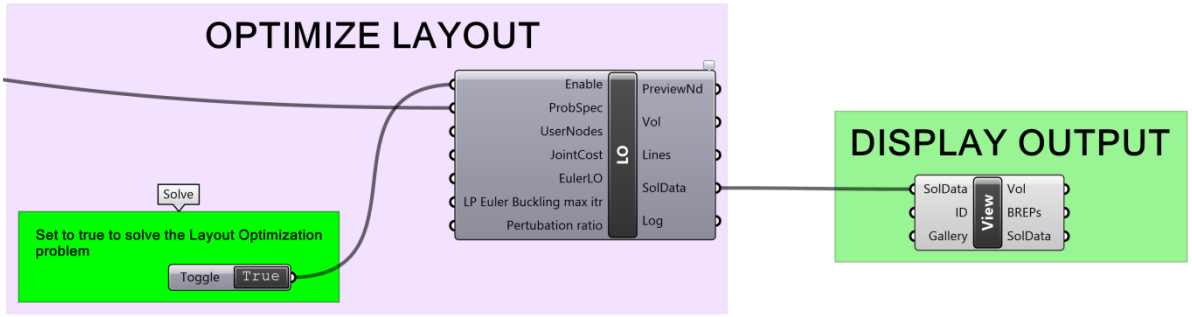

Figure 10 – Vizualizing a solution

It should be observed that the connector turns from dark orange to gray, to signify that a valid solution is available and is being displayed. The resultant structural geometry can be highlighted in Rhino by selecting the ***View*** component in Grasshopper (Figure 11). It is possible to determine the numerical solution (volume of the truss) by hovering over the ***Vol*** output of either the ***Layout Optimization*** or ***View*** components.

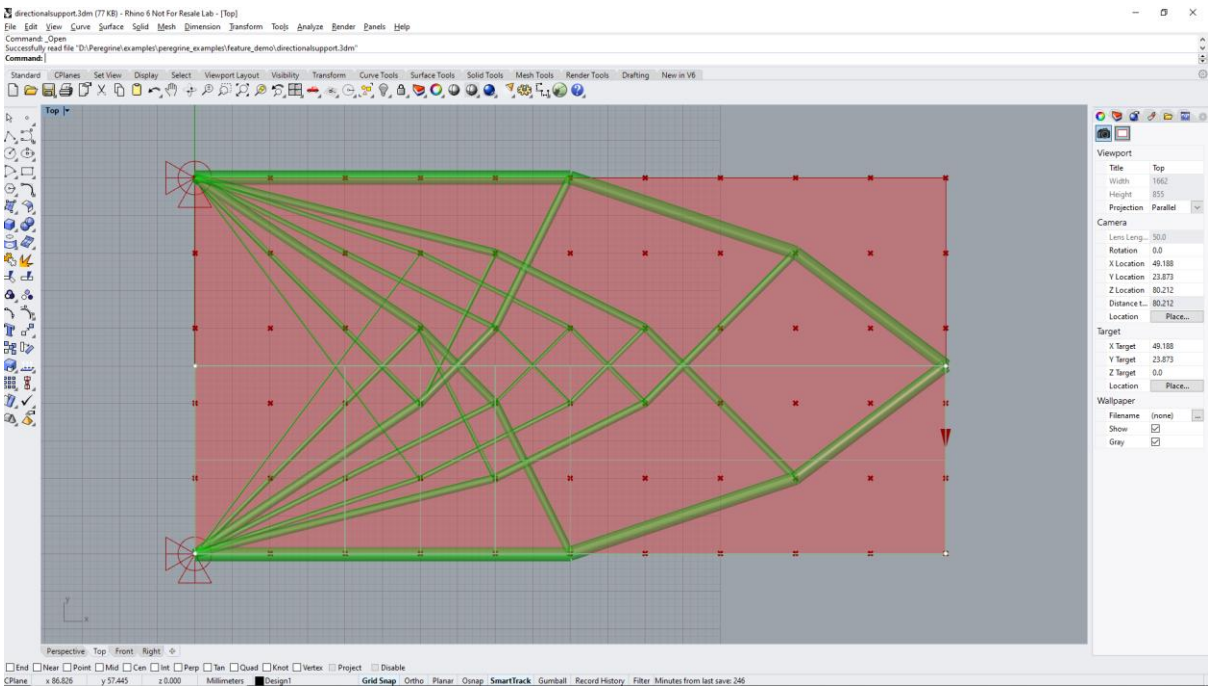

Figure 11 – Optimized solution (displayed in Rhino)

3.4 Starting from a Rhino Defined Geometry

In addition to defining a design domain geometry using Grasshopper inputs, one can also utilize geometries that have been defined within the Rhino environment itself. This is a good method by which complex shapes can be incorporated into the optimization.

> In Rhino, open the *DirectionalSupport.3dm* file

The geometry shown in Figure 12 will be loaded:

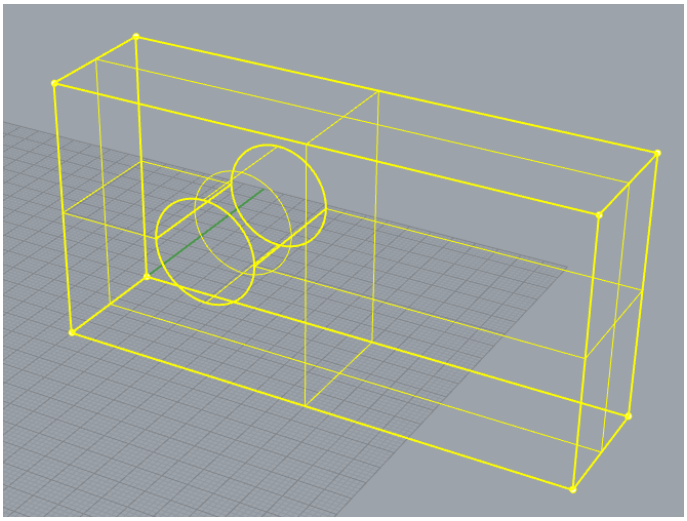

Figure 12 – A Design Domain geometry defined in Rhino

> In Grasshopper, open the *DirectionalSupport.gh* file

The problem will solve automatically and the resulting optimized structure shown in Rhino (Figure 13):

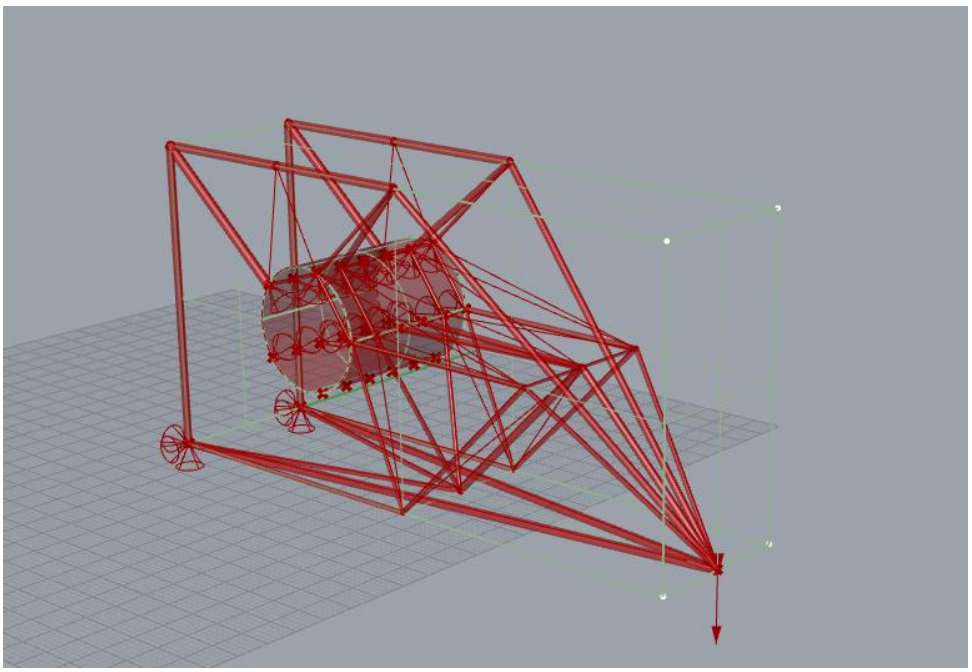

Figure 13 – Optimized Structure using Rhino defined Domain

> In Grasshopper, explore the components to get an overview of how the problem is constructed

The geometry (***Mesh***) for the ***Design Domain*** can be seen to originate from a ***Brep*** component. This, in-turn, references the geometry that is present in Rhino.

> Right-click the ***Brep*** component and select "*Clear values*"

We have just disassociated the geometry from the design domain. Notice that the Peregrine workflow now contains a number of components showing warnings (orange). To associate the Brep geometry once more:

> Right-click the ***Brep*** component and select "*Set Multiple Breps*"

> In Rhino, right-click and draw a box around the entire geometry (the Breps will turn yellow)

> Press *Enter*

The geometry will now be associated with the Peregrine problem again. The optimization will run and a solution will be presented.

3.5 Combining components for advanced optimization

Once familiar with the process of optimizing structures using Peregrine, it is possible to develop complex optimization problems and solve them with relative ease. For example, one can optimize grillages for a number of floors of a building and size the columns at the same time. This is explored in *Building_Commercial_Residential_Tower.gh* (Figure 14):

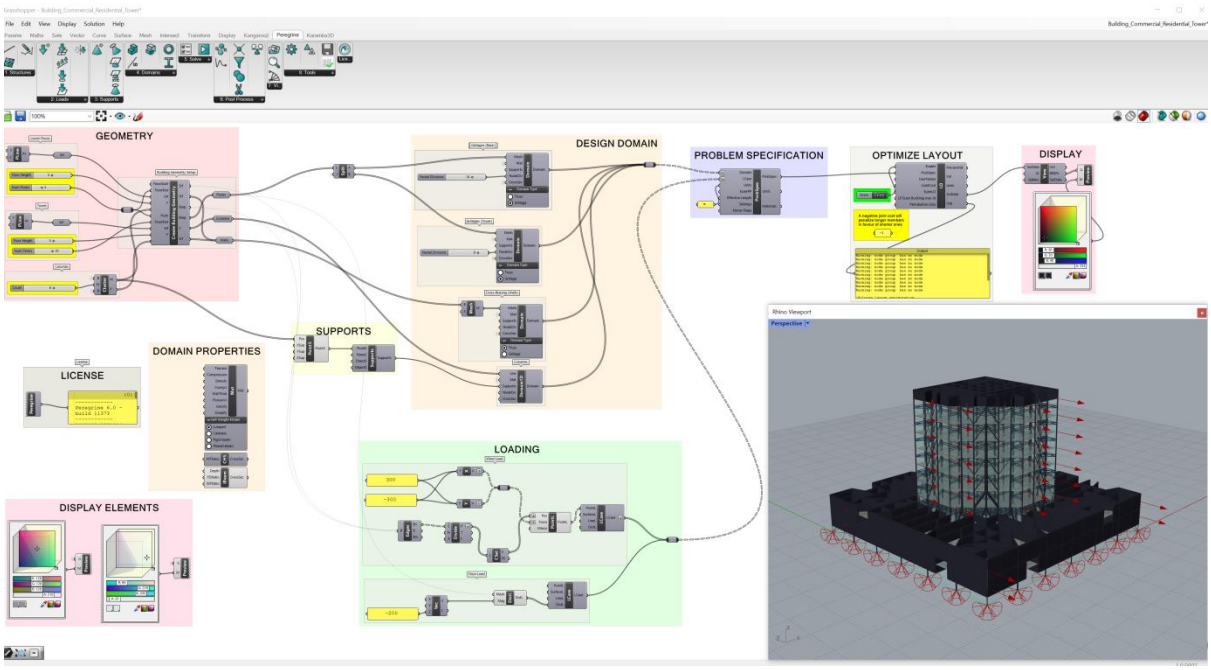

Figure 14 – Optimization of a complex building (grillages and column sizing)

3.6 Units

In Peregrine, all purely geometrical quantities (length, area, volume, position) adopt the same unit system as that used by the current Peregrine file. The active setting can be queried in the ***Units*** output of the Peregrine ***ProbSpec*** component and, in cases where the system being used is not that which is desired, the units can be changed by entering e.g. "mm" or "m" into the ***Units*** input (Figure 15):

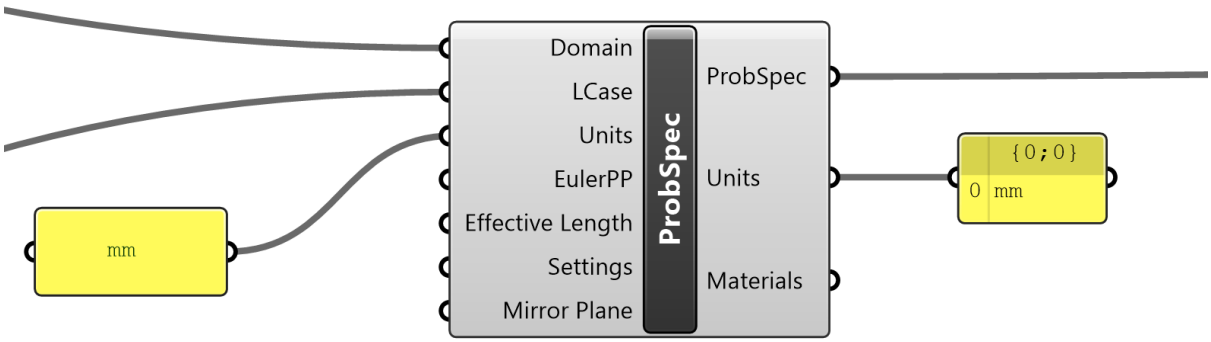

Figure 15 – The unit system can be defined or queried in the ProbSpec component

All other quantities (force, pressure, material properties) use metric units (e.g. kN), and will need to be specified as such. The units of a particular quantity can be seen by hovering over the component input/output that uses that quantity, as shown in Figure 16:

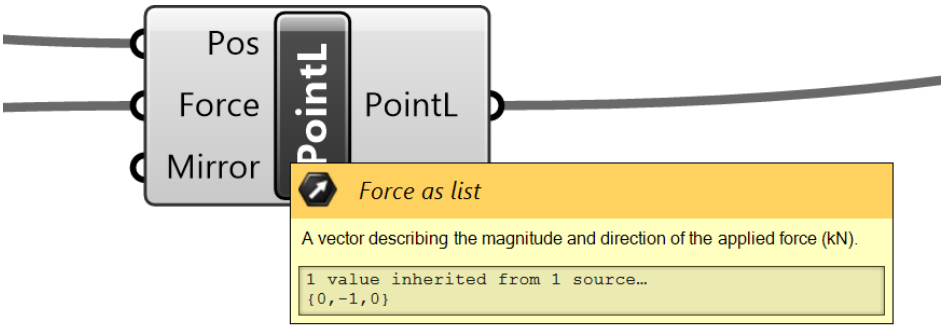

Figure 16 – Tooltip showing the units of the Layout Optimization volume output

4 Basic Concepts

4.1 Workflow

Peregrine is designed to work in a logical left-to-right manner across the Component Panel.

The following are the subsections that appear in the Peregrine category:

- **Structure** – define a starting topology of the user's own choosing
- **Load** – specify externally applied loads and combine these into load cases

- **Support** – specify a set of support conditions for the problem
- **Domain** – define the geometric and material properties of the volume within which the optimized structure must exist
- **Solve** – combine the design domain and load case(s) into a problem and optimize
- **Post Process** – take the results from the optimization and modify them as needed
- **View** – Visualize the optimized solution(s) in Rhino
- **Tool** – Useful tools for working with Peregrine output
- **About** – About Peregrine

5 Component Reference

5.1 Structures

5.1.1 Line Structure (LineStruct)

5.1.1.1 Icon

Figure 17 – The Line Structure (LineStruct) icon

5.1.1.2 Component

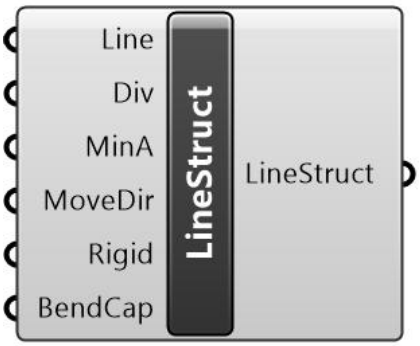

Figure 18 – The Line Structure (LineStruct) component

5.1.1.3 Description

A line or line-based structure. An object for loading or to represent a pre-existing structural component that can be utilized within the optimization process.

5.1.1.4 Summary

The ***Line Structure*** (***LineStruct***) component accepts a range of inputs and constructs a line or line structure from these, for use within the optimization (e.g. as an element for loading, or to represent a pre-existing structural component or frame that can be utilized within the optimization process).

The ***Line*** input accepts a ***Line*** (***Ln***) element or elements (double-click the grasshopper canvas and type "line" to bring up a list of line components – select the option titled "Line" with the "create a line between two points" tooltip).

Divisions (***Div***) is an integer specifying how many equally sized divisions the line(s) should be divided into for the purpose of e.g. applying load.

Min Area (***MinA***) is a floating point number describing the minimum cross-sectional area to ascribe to the elements. By default this value is set to 0.0.

Move Direction (***MoveDir***) is a vector describing the direction (if any) that the line element can move in. By default this value is set to (0.0, 0.0, 0.0) (no movement).

If ***Rigid*** is set to *True*, the line element ignores the stress constraints defined in the ***Mat*** component. By default this value is set to *False*.

Bending Capacity (***BendCap***) accepts values of 0.0 or 1.0, as well as intermediate values. When set to 0.0 (default) it is assumed that the line elements are rigid and cannot bend. When the value is set to 1.0, the bending capacity of each line is finite, calculated based on the available cross-section, taking due account of the presence of axial load. Note that due to the relative inefficiency of bending as a load transmission mechanism, the finite bending capacity may not be utilized in the optimal structure. By default this value is set to 0.0.

For an example of a Peregrine problem containing a ***LineStruct*** component, run the *Structure_LineStruct.gh* file.

Note that problems containing multiple materials cannot be used in conjunction with a user-defined line structure. If this attempted, a warning message will be displayed.

5.1.1.5 Input / Output

Input / Output	Description	Default Value
Line (in)	Line or line based structure.	N/A
Div (in)	Specifies how many equally sized divisions the line(s) should be divided into for the purpose of e.g. applying load.	10

Input / Output	Description	Default Value
MinA (in)	A minimum cross-sectional area to ascribe to the member(s).	0.0
MoveDir (in)	Describes the direction (if any) that the line element can move in.	(0.0, 0.0, 0.0)
Rigid (in)	When True, ignores any limiting stress constraints defined in the Material component.	False
BendCap (in)	Specify a bending capacity from 0.0 (no bending capacity) to 1.0 (full bending capacity).	0.0
LineStruct (out)	A line or line-based structure. An object for loading or to represent a pre-existing structural component that can be utilized within the optimization process.	N/A

5.1.2 Surface Structure (SurfStruct)

5.1.2.1 Icon

Figure 19 – The Surface Structure (SurfStruct) icon

5.1.2.2 Component

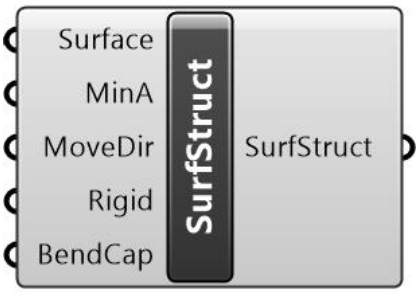

Figure 20 – The Surface Structure (SurfStruct) component

5.1.2.3 Description

A surface structure described by a mesh and having properties used in structural analysis. An object on to which loads are applied or to use as support.

5.1.2.4 Summary

The ***Surface Structure*** (***SurfStruct***) component is very similar in operation to the ***LineStruct*** component. It accepts a range of input and constructs a surface structure from these, for use within the optimization (e.g. as an object for loading).

The ***Surface*** input accepts a ***Meshed Surface or Mesh*** (***M***) object.

Min Area (***MinA***) is a floating point number describing the minimum cross-sectional area to ascribe to the elements of the mesh. By default this value is set to 0.0.

Move Direction (***MoveDir***) is a vector describing the direction (if any) that the mesh element can move in.

If ***Rigid*** is set to *True*, the surface ignores the stress constraints defined in the ***Mat*** component.

Bending Capacity (***BendCap***) accepts values of 0.0 or 1.0, as well as intermediate values. When set to 0.0 (default) it is assumed that the surface elements are rigid and cannot bend. When the value is set to 1.0, the bending capacity of each element on the surface is finite, calculated based on the available cross-section, taking due account of the presence of axial load. Note that due to the relative inefficiency of bending as a load transmission mechanism, the finite bending capacity may not be utilized in the optimal structure.

For an example of a Peregrine problem containing a ***SurfStruct***, run the *Structure_SurfStruct.gh* file.

5.1.2.5 Input / Output

Input / Output	Description	Default Value
Mesh (in)	Meshable surface(s) for load application. Note that loading is distributed at the corner points of any input geometry (i.e. use meshes for best coverage).	N/A
MinA (in)	A minimum cross-sectional area to ascribe to the member(s)	0.0
MoveDir (in)	Describes the direction (if any) that the surface can move in.	(0.0, 0.0, 0.0)
Rigid (in)	When True, ignores any limiting stress constraints defined in the Material component.	False
BendCap (in)	Specify a bending capacity from 0.0 (no bending capacity) to 1.0 (full bending capacity).	0.0

Input / Output	Description	Default Value
SurfStruct (out)	A surface structure described by a mesh and having properties used in structural analysis. An object on to which loads are applied or to use as support.	N/A

5.1.3 Define Topology (DefTop)

5.1.3.1 Icon

Figure 21 – The Define Topology icon

5.1.3.2 Component

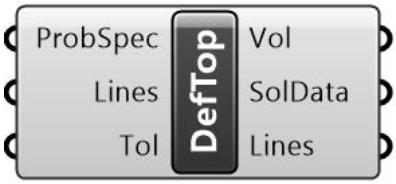

Figure 22 - The Define Topology component

5.1.3.3 Description

Define a Rhino structure and determine its efficiency. Post-optimization steps can also be performed on the output to ascertain whether the layout can be improved.

5.1.3.4 Summary

The ***Define Topology*** component allows the user to define a 2D structure within Rhino and to determine its efficiency, as well as to perform post-optimization steps to ascertain whether the layout can be improved upon.

The component accepts a ***Problem Specification*** (***ProbSpec***) and a topology as a series of ***Lines*** (or Polylines). The ***Lines*** can e.g. be imported via the use of a ***Curve*** (***Crv***) component into which the topology is passed through the use of the "*Set multiple curves*" command.

Note that, as part of the functionality, the input topology is validated. If the topology is found to be unstable, or if it contains parts that lie outside the design domain, a preliminary Geometry Optimization is undertaken and will be pushed to grid or vertex of domain if it's close enough. This distance is defined by the ***Tolerance (Tol)*** which is the fraction of the longest

diagonal of the domain below which member end points may be moved. By default this value is set to 0.01.

The output consists of:

A ***Volume*** (***Vol***), which is the volume of the solution structure.

The ***Solution Data*** (***SolData***) output packages together all the required data relating to the optimization problem and optimal (least volume) structure in a form that can be used as input into the ***Post-Process*** stages or displayed within the Rhino environment.

Lines, which is a list of all the elements in the structure.

For an example of a Peregrine problem containing a ***Define Topology*** component, run the *Structure_ConstructTopology.gh* file.

5.1.3.5 Input / Output

Input / Output	Description	Default Value
ProbSpec (in)	A combination of one or more Domain(s) and Load Case(s) for input into the Layout Optimization (LO).	N/A
Lines (in)	A user-defined structural topology formed from lines or polylines.	N/A
Tol (in)	Measure of the proximity of member end points to the edge of the design domain. Above this value, the joint will be pushed back inside. The longest diagonal of the model bounding box is determined and the proximity is the percentage of this measure which is used as the "search area" for domain violation.	0.01
Vol (out)	Volume of the solution structure (in document units).	N/A
SolData (out)	Optimization solution data.	N/A
Lines (out)	Output topology as line elements. Checked and fixed to ensure nodal equilibrium and domain compliance.	N/A

5.2 Loads

5.2.1 Load Case (LCase)

5.2.1.1 Icon

Figure 23 – The Load Case icon

5.2.1.2 Component

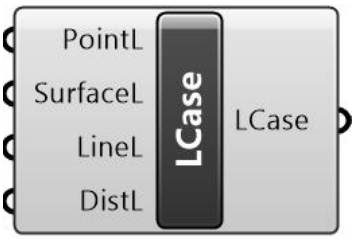

Figure 24 – The Load Case component

5.2.1.3 Description

A load or group of loads to be applied to the structure as an independent case.

Note:
- Specification of ***multiple load cases*** requires a ***Standard*** or ***Professional*** license.

5.2.1.4 Summary

The ***Load Case*** (***LCase***) component accepts one or more loads (***PointL***, ***SurfaceL***, ***LineL***, ***DistL***). It outputs a ***Load Case***. Several Load Cases can be combined and used as input into the ***Problem Specification*** (***ProbSpec***) component.

For an example of a Peregrine problem containing multiple load cases, run the *ProbSpec_MultipleLoadCases2.gh* file.

5.2.1.5 Input / Output

Input / Output	Description	Default Value
PointL (in)	A load applied at a specified point.	N/A

Input / Output	Description	Default Value
SurfaceL (in)	Loading applied to a meshable surface structure.	N/A
LineL (in)	Loading applied over the length of a line.	N/A
DistL (in)	Define a distributed load (for use with Grillages).	N/A
Load Case (out)	A load or group of loads to be applied to the structure as an independent case.	N/A

5.2.2 Point Load (PointL)

5.2.2.1 Icon

Figure 25 – The Point Load (PointL) icon

5.2.2.2 Component

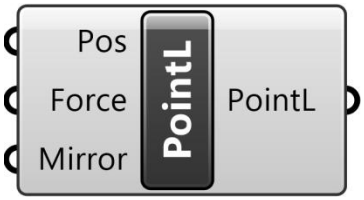

Figure 26 – The Point Load (PointL) component

5.2.2.3 Description

A load applied at a specified point.

5.2.2.4 Summary

The ***PointL*** component is used as input into a ***Load Case***. It accepts {xyz} coordinates in Cartesian space (***Pos***) and a magnitude (***Mag***) in kN. The output is a Point Load (***PointL***).

For an example of a Peregrine problem containing a point load, run the *Simple2D.gh* file.

5.2.2.5 Input / Output

Input / Output	Description	Default Value
Pos (in)	A location in Cartesian space, defined using a **Construct Point**	N/A
Mag (in)	A vector describing the magnitude and direction of the applied force (kN).	(0.0, 0.0, 0.0)
Mirror (in)	Apply a mirror of this load simultaneously? Requires a mirror plane to be defined in ***ProbSpec***.	0 (no mirror)
PointL (out)	A load applied at a specified point.	N/A

5.2.3 Line Load (LineL)

5.2.3.1 Icon

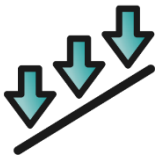

Figure 27 – The Line Load (LineL) icon

5.2.3.2 Component

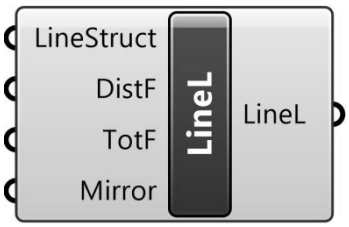

Figure 28 – The Line Load (LineL) component

5.2.3.3 Description

Loading applied over the length of a line.

5.2.3.4 Summary

The **Line Load** (***LineL***) component accepts a **LineStruct** as the definition of its geometry and either a force in kN per unit length (***DistF***) or a total force in kN (***TotF***) to define the magnitude of the load imparted. The output is a loaded line (***LineL***).

For an example of a Peregrine problem containing a line load, run the *Structure_LineStruct.gh* file.

5.2.3.5 Input / Output

Input / Output	Description	Default Value
LineStruct (in)	A line or line-based structure. An object for loading or to represent a pre-existing structural component that can be utilized within the optimization process.	N/A
DistF (in)	Force per unit length of line (kN/document unit).	(0.0, 0.0, 0.0)
TotF (in)	Total force applied to the line (kN) (will be uniformly distributed).	(0.0, 0.0, 0.0)
Mirror (in)	Apply a mirror of this load simultaneously? Requires a mirror plane to be defined in ***ProbSpec***.	0 (no mirror)
LineL (out)	Loading applied over the length of a line.	N/A

5.2.4 Surface Load (SurfaceL)

5.2.4.1 Icon

Figure 29 – The Surface Load (SurfaceL) icon

5.2.4.2 Component

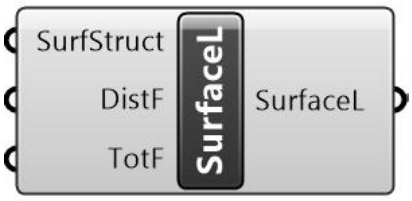

Figure 30 – The Surface Load (SurfaceL) component

5.2.4.3 Description

Loading applied to a meshable surface structure.

5.2.4.4 Summary

The ***SurfaceL*** component is used as input into a ***Load Case***. It accepts a ***Surface Structure*** (***SurfStruct***) and either a either a force in kN per unit area (***DistF***) or a total force in kN (***TotF***) to define the magnitude of the load imparted. The output is a ***Surface Load*** (***SurfaceL***).

For an example of a Peregrine problem containing a surface load, run the *Structure_SurfStruct.gh* file.

5.2.4.5 Input / Output

Input / Output	Description	Default Value
SurfStruct (in)	A surface structure described by a mesh and having properties used in structural analysis. An object on to which loads are applied or to use as support.	N/A
DistF (in)	Force per unit area of surface (kN/document area unit).	0.0
TotF (in)	Total force (kN) applied over the surface (distributed evenly).	0.0
LineL (out)	Loading applied to a meshable surface structure.	N/A

5.2.5 Distributed Load (DistL)

5.2.5.1 Icon

Figure 31 – The Distributed Load (DistL) icon

5.2.5.2 Component

Figure 32 – The Distributed Load (DistL) component

5.2.5.3 Description

Define a distributed load (for use with Grillages).

5.2.5.4 Summary

The ***DistL*** component divides a given pressure between the nodes that are generated for a given 2D domain. It accepts a ***Mesh*** as the definition of the area over which to apply the loading and a total magnitude (***Mag)*** as the pressure in kN per unit area. The output is a loaded area (***DistL***) that can be used as input for a ***Load Case***.

Note - in order to properly function, the input ***Mesh*** must correspond exactly to a mesh that is used to define a 2D domain within the problem.

For an example of a Peregrine problem containing a distributed load, run the *Grillage_DistributedLoad.gh* file. Note that analysis of problems involving Grillages requires a Professional license.

5.2.5.5 Input / Output

Input / Output	Description	Default Value
Mesh (in)	A mesh defining the region over which the load is applied. Note that this must match an existing mesh in the Design Domain	N/A
Mag (in)	The load to apply over the mesh (kN/document area unit)	0.0
DistL (out)	Define a distributed load (for use with Grillages).	N/A

5.2.6 Mirror

5.2.6.1 Icon

Figure 33 – The Mirror icon

5.2.6.2 Component

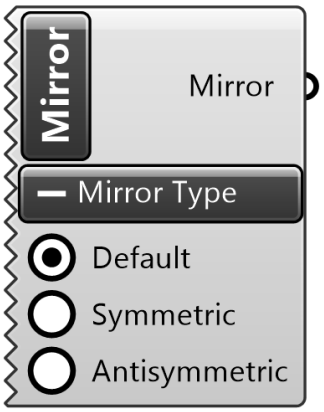

Figure 34 – The Mirror component

5.2.6.3 Description

Mirror point or line loads.

5.2.6.4 Summary

The mirror component is used to set the load mirroring behaviour when a mirror plane has been defined in the ***Problem Specification***. See section 6.1.4 for more detail.

For an example of a Peregrine problem containing a mirror, run the *Structures_L_Truss.gh* file.

5.2.6.5 Input / Output

Input / Output	Description	Default Value
Mirror Type (in)	Sets the behaviour of the load in the presence of a mirror plane: • *Default*: the load is applied only to one side of the design domain (but the solution is symmetrical) • *Symmetric*: the load is applied simultaneously on both sides of the mirroring plane, with the direction of the load reflected • *Antisymmetric*: the load is applied simultaneously on both sides of the mirroring plane	Default
Mirror (out)	Behavior of a point or line load if a mirror plane is defined.	N/A

5.3 Supports

5.3.1 Supports (Supports)

5.3.1.1 Icon

Figure 35 – The Supports icon

5.3.1.2 Component

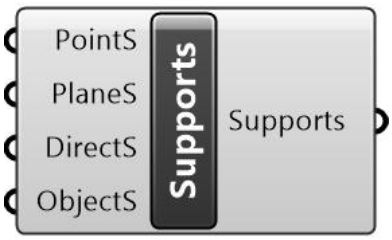

Figure 36 – The Supports component

5.3.1.3 Description

A set of one or more Support definitions.

5.3.1.4 Summary

The ***Supports*** component accepts one or more support objects (***PointS***, ***PlaneS***, ***DirectionalS*** or ***ObjectS***), combines them and outputs them as ***Supports*** that can then be used as input into the ***Design Domain*** (***Domain***).

Refer to the individual support type descriptions for example files.

5.3.1.5 Input / Output

Input / Output	Description	Default Value
PointS (in)	Restrict translation at specified point(s) in one or more Cartesian directions.	N/A
PlaneS (in)	Provide support across a planar surface or face.	N/A
DirectS (in)	Restrict translation at specified point(s) along a specified vector.	N/A
ObjectS (in)	Support provided by a specified object	N/A

Input / Output	Description	Default Value
Supports (out)	A set of one or more Support definitions.	N/A

5.3.2 Point Support (PointS)

5.3.2.1 Icon

Figure 37 – The Point Support (PointS) icon

5.3.2.2 Component

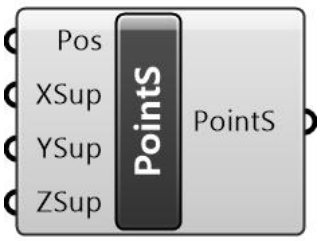

Figure 38 – The Point Support (PointS) component

5.3.2.3 Description

Restrict translation at specified point(s) in one or more Cartesian directions.

5.3.2.4 Summary

The ***PointS*** component is used as input into ***Supports***. It accepts a ***Construct Point*** (***Pt***) for its position and a toggle (Boolean operator) to specify whether support is provided at the location in the X, Y or Z direction. By default, all three directions are provided with support (*True*), thus the user only need add extra controls if they wish to allow freedom of movement in one of the directions. The output is a ***Point Support*** (***PointS***).

For an example of a Peregrine problem containing a line load, run the *Simple2D.gh* file.

5.3.2.5 Input / Output

Input / Output	Description	Default Value
Pos (in)	A location in Cartesian space, defined using a **Construct Point**	N/A

Input / Output	Description	Default Value
XSup (in)	Provide restraint against movement in the X direction.	True
YSup (in)	Provide restraint against movement in the Y direction.	True
ZSup (in)	Provide restraint against movement in the Z direction.	True
PointS (out)	Restrict translation at specified point(s) in one or more Cartesian directions.	N/A

5.3.3 Directional Support (DirectionalS)

5.3.3.1 Icon

Figure 39 – The Directional Support (DirectionS) icon

5.3.3.2 Component

Figure 40 – The Directional Support (DirectionS) component

5.3.3.3 Description

Restrict translation at specified point(s) along a specified vector.

5.3.3.4 Summary

The ***DirectionalS*** component is used to specify support positions with restraint in a particular direction. It is used as input into the ***Supports*** component. It accepts a list of ***Vertices*** that describe the support ***Positions*** (***Pos***) and a similarly sized list of ***Vectors*** to describe the ***Direction*** (***Dir***) afforded to those positions. Using this component, it is possible to e.g. specify support to the vertices of a pre-defined mesh, in a direction normal to the mesh.

For an example of a Peregrine problem containing a directional support, run the *DirectionalSupport.gh* file.

5.3.3.5 Input / Output

Input / Output	Description	Default Value
Pos (in)	Point at which support is to be provided.	N/A
Dir (in)	Direction along which support is to be provided.	N/A
DirectS (out)	Restrict translation at specified point(s) along a specified vector.	N/A

5.3.4 Plane Support (PlaneS)

5.3.4.1 Icon

Figure 41 – The Plane Support (PlaneS) icon

5.3.4.2 Component

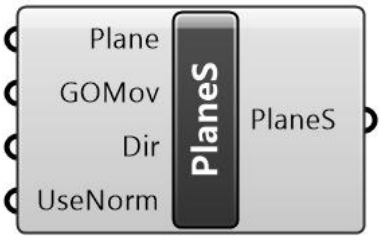

Figure 42 – The Plane Support (PlaneS) component

5.3.4.3 Description

Provide support across a planar surface or face.

5.3.4.4 Summary

The ***PlaneS*** component is used to specify plane or face support surfaces to be used as input into ***Supports***. It accepts a ***Plane*** (e.g. ***XY***) and a number of options to specify how the nodes on the surface behave.

The ***Move*** (***GOMov***) input is a Boolean toggle (*False* by default) to specify whether the nodes that lie on the surface of the supported plane (i.e. the points of support) are allowed to move around that plane during any Geometry Optimization stage(s).

The ***Support Direction(s)*** (***Dir***) input is a vector specifying the direction of support at the plane.

The ***Use Plane Normal Support*** (***UseNorm***) input is a Boolean toggle (*False* by default) to specify whether support is provided to the plane in the normal (perpendicular) direction.

The output is a ***Plane Support*** (***PlaneS***).

For an example of a Peregrine problem containing a plane support, run the *SolutionViewer_Symmetry1.gh* file.

5.3.4.5 Input / Output

Input / Output	Description	Default Value
Plane (in)	Planar surface to which support is to be provided.	N/A
GOMov (in)	Allow nodes to move on the plane during Geometry Optimization.	False
Dir (in)	A vector specifying the direction of support at the plane.	N/A
UseNorm (in)	Specify whether support is provided to the plane in the normal (perpendicular) direction.	False
PlaneS (out)	Provide support across a planar surface or face.	N/A

5.3.5 Plane SupportXYZ (PlaneSXYZ)

5.3.5.1 Icon

Figure 43 – The Plane SupportXYZ (PlaneSXYZ) icon

5.3.5.2 Component

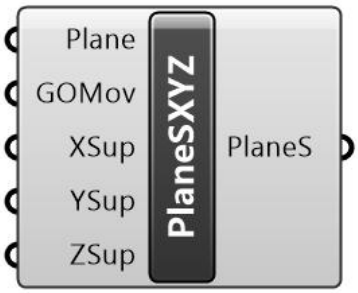

Figure 44 – The Plane SupportXYZ (PlaneSXYZ) component

5.3.5.3 Description

Provide support across a planar surface or face.

5.3.5.4 Summary

The ***PlaneSXYZ*** component is similar in nature to the ***PlaneS*** component. It is used to specify plane or face support surfaces with restraint in one or more of the {xyz} directions, to be used as input into ***Supports***. It accepts a ***Plane*** and a number of options to specify how the nodes on the surface behave.

The ***Move*** (***GOMov***) input is a Boolean toggle (*False* by default) to specify whether the nodes that lie on the surface of the supported plane (i.e. the points of support) are allowed to move around that plane during any Geometry Optimization stage(s).

The ***X***, ***Y*** and ***Z*** inputs are Boolean toggles (*True* by default) to specify whether restraint is provided in the direction specified.

The output is a ***Plane Support XYZ*** (***PlaneSXYZ***).

For an example of a Peregrine problem containing a Plane XYZ support, run the *ProbSpec_MultipleLoadCases1.gh* file.

5.3.5.5 Input / Output

Input / Output	Description	Default Value
Plane (in)	Planar surface to which support is to be provided.	N/A
GOMov (in)	Allow nodes to move on the plane during Geometry Optimization.	False
XSup (in)	Provide restraint against movement in the X direction.	True

Input / Output	Description	Default Value
YSup (in)	Provide restraint against movement in the Y direction.	True
ZSup (in)	Provide restraint against movement in the Z direction.	True
PlaneS (out)	Provide support across a planar surface or face.	N/A

5.4 Domains

5.4.1 Design Domain (Domain)

5.4.1.1 Icon

Figure 45 – The Design Domain (Domain) icon

5.4.1.2 Component

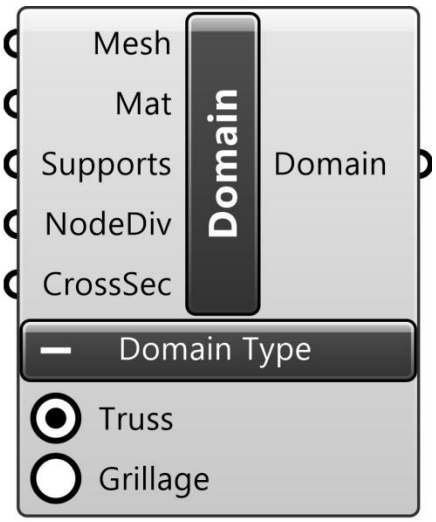

Figure 46 – The Design Domain (Domain) component

5.4.1.3 Description

Design domain described by a triangulated mesh or lines.

Notes:

- The number of ***Divisions*** is limited in ***Free*** mode such that only problems that are 200 nodes or fewer are solvable.
- Specification of ***Grillage*** domains cases requires a ***Professional*** license.
- Specification of ***Line*** domain cases requires a ***Professional*** license.
- Specification of ***multiple design domains*** requires a ***Professional*** license.

5.4.1.4 Summary

The ***Domain*** component is used to collate the *Geometry*, *Material* and *Support* data for entry into the ***Problem Specification*** (***ProbSpec***). It is also used to specify the ***Domain type*** as either a ***Truss*** or a ***Grillage***.

A ***Truss*** comprises an arrangement of triangulated elements in either a 2D or 3D environment and which works primarily in tension and compression (Figure 47), whereas a ***Grillage*** comprises a series of interconnected elements over a 2D planar surface with a common structural depth and which works in bending (Figure 48).

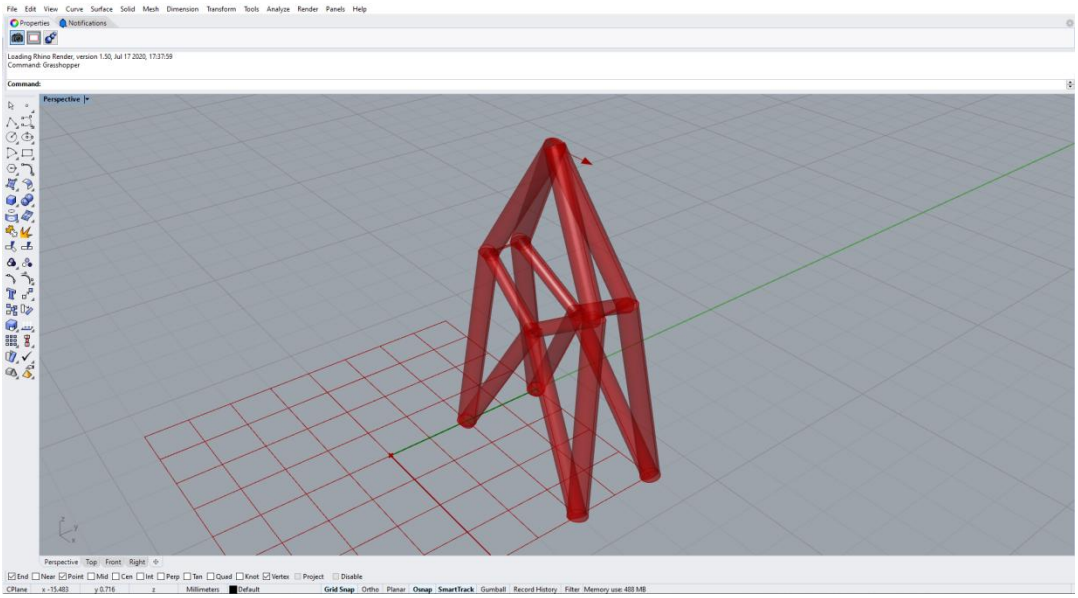

Figure 47 – A (simple) optimized truss structure

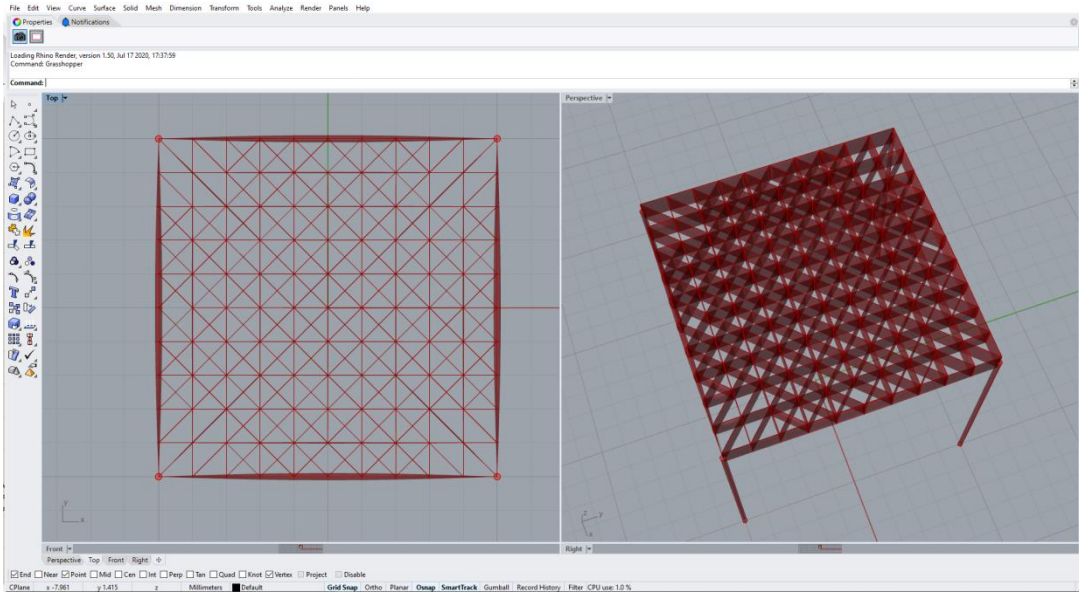

Figure 48 – An example of an optimized grillage structure (supported on four columns)

The ***Domain Mesh (Mesh)*** input defines the geometry of the Design Domain. It takes a triangulated mesh, which can come from a variety of sources. When the ***Domain type*** is set to be a ***Grillage***, the input mesh must be 2D.

Multiple domains can be specified in a problem. These can be of differing types and each can contain a different material etc. However, all shared geometries between any two domains (i.e. interfaces) must be entire faces, entire edges or singular points. Partial overlaps, whether in 2D or 3D, are not permitted (see Figure 49):

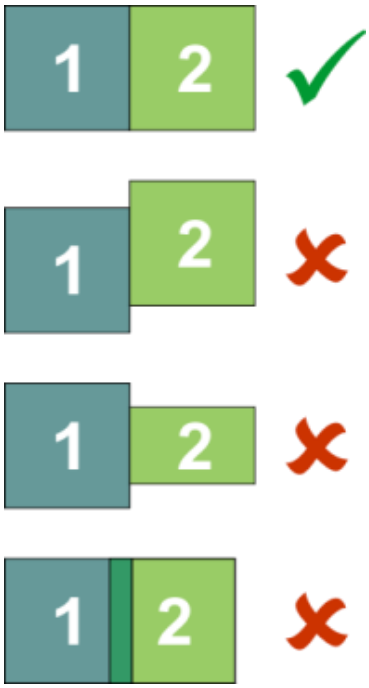

Figure 49 – Allowable mesh interfacing configurations

The ***Material*** (***Mat***) input accepts a user-defined set of material properties from a ***Mat*** component.

The ***Depth*** (***Depth***) is an input that is only associated with the ***Grillage*** type. It specifies the structural depth of all grillage elements within the ***Domain***. Note that grillages are able to transmit infinite forces in-plane (i.e. they are assumed to be rigid in-plane).

The ***Supports*** input accepts a collection of supports from a ***Supports*** component.

The ***Nodal Division*** (***NodeDiv***) input accepts a number and defines the density of nodes assigned to the problem. To do this,, first a local coordinate system is established:

- A face on the ***Domain*** is selected as the base for the local coordinate system. If there are parallel faces, the largest face that is parallel with some other face is selected. If none of the faces are parallel, the largest overall face is selected.
- The normal to the selected face is assigned as the local "Z" axis.
- The first edge on the face is assigned as the local "X" direction.
- The local "Y" direction is determined.

Secondly, the bounding box of this face is determined, based on the local coordinate system.

Lastly, the longest edge of the bounding box is divided into the specified number of nodal divisions to set the nodal spacing for the domain.

Note that many post-processing steps use the ***NodeDiv*** setting to determine suitable limits on e.g. the movement of nodes during geometry optimization. Therefore, a suitable value should be provided, even if user-defined nodes are being provided for the main optimization.

The output is a ***Domain***.

For an example of a Peregrine problem containing a basic ***Truss*** domain component, run the *Simple2D.gh* file.

For an example of a Peregrine problem containing a basic ***Grillage*** domain component, run the *Grillage_SimpleSquareOn8Columns.gh* file.

5.4.1.5 Input / Output

Input / Output	Description	Default Value
Mesh (in)	A triangulated mesh defining the geometry of the Design Domain. When the Domain type is set to be a Grillage, the input mesh must be 2D.	N/A
Mat (in)	The properties of the material used in this Domain.	N/A

Input / Output	Description	Default Value
Supports (in)	A set of one or more support definitions.	True
NodeDiv (in)	Defines the density of auto-generated nodes assigned to the Domain. Selects the largest face of the Domain and uses the bounding box to set up a regular Cartesian nodal grid from this.	1
CrossSec (in)	Defines cross section properties.	Default ***CHS*** / ***I-beam*** component properties
Domain (out)	Design domain described by a triangulated mesh.	N/A

5.4.1.6 Cross section types

Note that multiple section types can be supplied to the domain ***CrossSec*** input for a single problem. For example, a domain that uses both catenary and beam materials may use both CHS and I-Beam members, and so requires both a ***CHS Cross section*** and ***I-Beam cross section*** to be input into the domain.

5.4.1.7 Compatibility with post-processing components

Some post-processing functionality is incompatible / has limited compatibility with problems that include grillages. In most cases the grillage domain itself will simply be omitted from the post-processing steps. However, the ***Stabilize*** component will not work with any problems containing an active grillage domain.

The following table outlines the behaviour experienced for each potential scenario:

Component	Component works on problem containing grillage domains?	Component applied to grillage members?
Create Crossovers	Yes	Yes
Filter	Yes	Yes
Geometry Optimization (GO)	Yes	No
Simplify (all modes)	Yes	No

Component	Component works on problem containing grillage domains?	Component applied to grillage members?
Merge Joints	Yes	No
Reduce Complexity	Yes	No
Stabilize	No	N/A

5.4.2 Design Domain 1D (Domain 1D)

5.4.2.1 Icon

Figure 50 – The Design domain 1D (Domain 1D) icon

5.4.2.2 Component

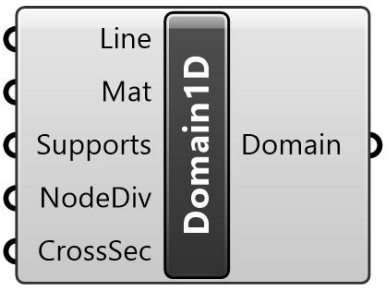

Figure 51 – The Design domain 1D (Domain 1D) component

5.4.2.3 Description

Define area(s) of design domain where a 1D (line) element can exist. This is e.g. useful for specifying the locations of potential columns in a building optimization.

5.4.2.4 Summary

The ***Design domain 1D*** (***Domain 1D***) component is used to define one or more specified areas of design domain where a 1D (line) element can exist, e.g. it is useful for specifying the locations of potential columns in a building optimization.

The output is a ***Design Domain*** (***Domain***), which can then form part of the input into the ***Problem Specification*** (***ProbSpec***) component.

For an example of a Peregrine problem containing a ***Domain 1D*** component, run the *Grillage_SimpleSquareOn8Columns.gh* file.

5.4.2.5 Input / Output

Input / Output	Description	Default Value
Line (in)	Line(s) defining the geometry of the design Domain.	N/A
Mat (in)	The properties of the material used in this Domain.	True
Supports (in)	A set of one or more support definitions.	True
NodeDiv (in)	Defines the density of auto-generated nodes assigned to the Domain. Selects the largest face of the Domain and uses the bounding box to set up a regular Cartesian nodal grid from this.	True
CrossSec (in)	Defines cross section properties.	Default ***CHS*** / ***I-beam*** component properties
Domain (out)	Design domain described by a line.	N/A

5.4.3 Material (Mat)

5.4.3.1 Icon

Figure 52 – The Material (Mat) icon

5.4.3.2 Component

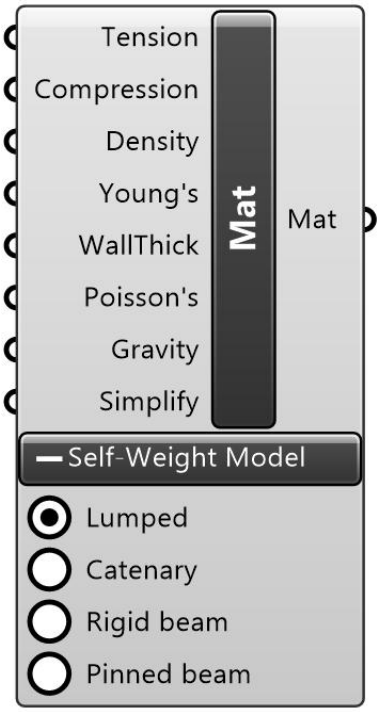

Figure 53 – The Material (Mat) component

5.4.3.3 Description

The properties of the material used in this Domain.

5.4.3.4 Summary

The ***Material*** (***Mat***) component is used to define material properties for use in a design domain. These are used by the optimization and analysis algorithms.

The ***Tension*** and ***Compression*** inputs accept values for the limiting stresses of the material (MPa), while the ***Density*** input accepts a value in kg/m^3. Young's modulus (***Young's***) and Poisson's ratio (***Poisson's***) are elastic material properties that are used in e.g. Euler buckling calculations and post-processing functions, but which do not form part of the "normal" layout optimization (LO) process.

The Wall Thickness input (***WallThick***) is a deprecated parameter that has been retained for file compatibility. Wherever possible, users should ensure this is set to "-1" and use the ***WTRatio*** input to a ***CHS*** component instead. A warning will be shown on the ***Material*** component when this is not the case.

Gravity accepts a vector to describe the body force accelerations (m/s^2) in the Cartesian directions. For example, a ***Vec*** of (0.0, 0.0, -9.81) applies structural self-weight in the downwards Z direction.

The ***Simplify*** input is a Boolean that tells the ***Simplify*** component (if present) whether members of this material should count towards the complexity metrics when undertaking a simplification action. By default, this is set to true.

The ***Self-Weight Model*** radio buttons indicate the way in which the self-weight of members comprising of this material will be calculated:

- ***Lumped***: assumes equal distribution of mass between member end points (neglects bending);
- ***Catenary***: assumes only axial stress only under a combination of axial load and self-weight (compressive forces lead to arching, tensile lead to sagging). Useful for modelling cable members;
- ***Rigid beam***: assumes moment resistance at joints (potentially leading to lower volume structures, but longer computation times); useful in multi-load case problems;
- ***Pinned beam:*** assumes NO moment resistance at joints.

Members that use the ***Catenary*** model will assume the form of an equally stressed catenary. This is a shape which is defined by the existence of solely axial stresses under the combined effects of axial load and self-weight. The equal stress catenary has a curved centerline, forming a shallow arch when designed to resist compressive forces, or a sagging curvature to resist tensile forces. The curvature of the centerline depends only on the material's permitted stress and unit weight, and can therefore be efficiently implemented in the layout optimization (LO) procedure. In Peregrine, where two limiting stresses are defined in the ***Mat*** component (***Compression*** and ***Tension***), the larger of the two will be assumed and a warning shown to indicate that this is the case.

Members that use a ***Beam*** representation are assumed to be prismatic, with geometry as outlined by the ***Cross-section*** component. The self-weight of these members is applied uniformly along their length, generating bending moments and shear forces which must be resisted. The joints between these members may be rigid (***Rigid beam***), in which case moment equilibrium is considered. Alternatively, the joints may be set to be pinned (***Pinned beam***), in which case the bending moment at the ends of each member will be fixed at zero.

For more information, the reader is referred to Fairclough et al., 2018.

The output is a ***Material*** (***Mat***), which is then used as an input for a ***Design Domain*** (***Domain***) component. Multiple materials using different self-weight models may be supplied to the ***Domain***, in which case members of both will be considered. Note that if a **Lumped** or ***Catenary*** material is used, a ***CHS*** component should be supplied to the ***Domain*** to set the cross section properties of members that use this material. Conversely, if a ***Rigid beam*** or ***Pinned beam*** material is used, an ***I-Beam*** component should be supplied. Peregrine will warn on the ***Domain*** component if this has not been done.

For an example of a simple Peregrine problem containing a ***Material*** component, run the *Simple2D.gh* file.

For an example of a more complex Peregrine problem containing a ***Material*** component with Catenary self-weight, run the *Bridge_HalfSpanArray.gh* file.

5.4.3.5 Input / Output

Input / Output	Description	Default Value
Tension (in)	Limiting (maximum) tensile stress (MPa).	350.0
Compression (in)	Limiting (maximum) compressive stress (MPa).	350.0
Density (in)	Material density (kg/m^3).	7800.0
Young's (in)	Young's modulus (GPa).	210.0
WallThick (In)	Ratio of wall thickness to member radius (>0.0 to 1.0, or -1). Note that this input is deprecated and wall thickness ratio should be set on the ***CHS Cross-Section*** (***CHS***) component instead. To prevent this input causing a warning to be shown on the component, set the value to -1.	-1
Poisson's (In)	Poisson's ratio.	0.3
Gravity (in)	Body force acceleration (m/s^2) in the Cartesian directions. For example (0, 0, -9.81) applies structural self-weight in the downwards Z direction.	(0.0, 0.0, 0.0)
Simplify (in)	Boolean to specify whether members of this material should be considered as contributing to structural complexity when undertaking a Simplify action.	True
Self-Weight Model (in)	Radio buttons indicating the way in which the self-weight of members comprising of this material will be calculated (lumped, catenary, pinned beam or rigid beam).	Lumped
Mat (out)	The properties of the material used in this Domain.	N/A

5.4.4 CHS Cross-Section (CHS)

5.4.4.1 Icon

Figure 54 – The CHS Cross-Section (CHS) icon

5.4.4.2 Component

Figure 55 – The CHS Cross-Section (CHS) component

5.4.4.3 Description

Defines cross-section properties for circular hollow sections (CHS).

5.4.4.4 Summary

The ***CHS Cross-Section*** (***CHS***) component is used to define the properties of all CHS members used in the design domain to which it is supplied.

Wall Thickness Ratio (***WTRatio)*** specifies the ratio of wall thickness to the outer member radius. Note that the magnitude of the ***WTRatio*** parameter allows users to specify whether e.g. the members are CHS or CSS as well as influencing Euler buckling calculations.

For an example of a Peregrine problem containing a ***CHS*** component, run the *Structures_L_Truss.gh* file.

5.4.4.5 Input / Output

Input / Output	**Description**	**Default Value**
WTRatio (in)	Ratio of wall thickness to member radius (0.0 < value < 1.0). Default = 1.0. Note that assigning a value of 1.0 will set the section to be solid.	1.0
CrossSec (Out)	CHS Cross section for input into a ***Domain*** component.	N/A

5.4.5 I-Beam Cross-Section (IBeam)

5.4.5.1 Icon

Figure 56 – The I-Beam Cross-Section (IBeam) icon

5.4.5.2 Component

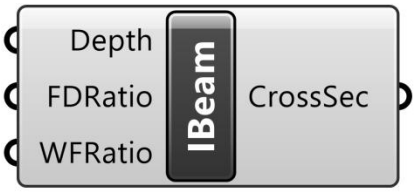

Figure 57 – The I-Beam (IBeam) component

5.4.5.3 Description

Defines cross-section properties for I-beams.

5.4.5.4 Summary

The ***I-Beam Cross-Section*** (***IBeam***) component is used to define the properties of all I-beam members used in the design domain to which it is supplied. An I-beam member is specified by three constants and two variable areas, one at each endpoint.

The depth, flange thickness, and web thickness, shown (see Figure 58), are constant for all I-beam member cross-sections of a given domain, and it is these which are defined by the ***IBeam*** component. Figure 59 shows a plan view of an arbitrary grillage member.

The beam web is considered to have zero area, such that the area of a member at some point along its length is controlled entirely by the flange width at that point, which varies linearly between the values at each endpoint. These endpoint widths are found from the corresponding areas determined by the solver.

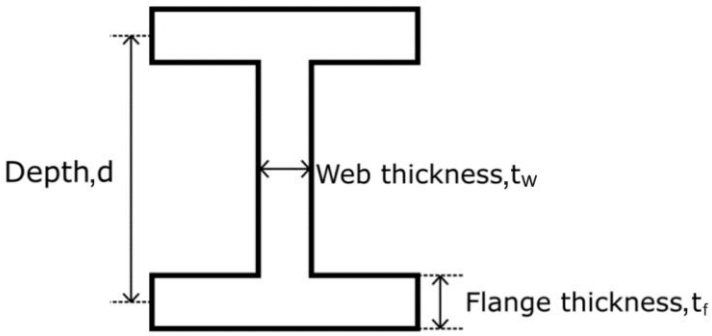

Figure 58 – Cross section of an I-Beam member

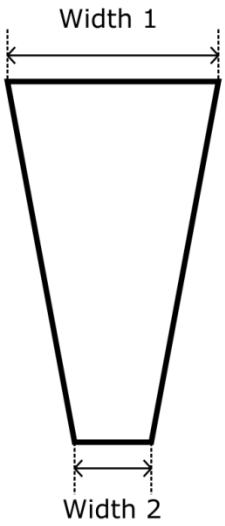

Figure 59 – I-Beam plan view

For an example of a Peregrine problem containing an ***IBeam*** component, run the *Building_Commercial_Residential_Tower.gh* file.

5.4.5.5 Input / Output

Input / Output	**Description**	**Default Value**
Depth (in)	The structural depth (*d*) of all I-beam elements within the Domain, measured between the neutral axes of the two flanges. Note that grillages are able to transmit infinite forces in-plane (i.e. they are assumed to be rigid in-plane).	5.0
FDRatio (in)	= t_f/d. Ratio of flange thickness to I-beam ***Depth***.	0.1
WFRatio (in)	= t_w/t_f. Ratio of web thickness to flange thickness.	0.625

Input / Output	Description	Default Value
CrossSec (Out)	Calculated I-beam cross sections, for input into a ***Domain*** component.	N/A

5.5 Solve

5.5.1 Problem Specification (ProbSpec)

5.5.1.1 Icon

Figure 60 – The Problem Specification (ProbSpec) icon

5.5.1.2 Component

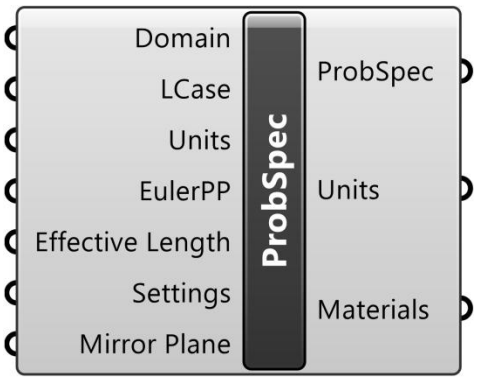

Figure 61 – The Problem Specification (ProbSpec) component

5.5.1.3 Description

A combination of one or more Domains and Load Cases for input into a Layout Optimization (LO).

5.5.1.4 Summary

The ***Problem Specification*** (***ProbSpec***) component combines one or more ***Domain(s)*** and the applied ***Loading*** for input into the ***Layout Optimization*** component. The ***Units*** of the problem are also input here. The output is a ***Problem*** (***ProbSpec***), the ***Units*** of the problem, and a list of the materials in the problem together with their properties.

For an example of a Peregrine problem containing the ***ProbSpec*** component, run the *Simple2D.gh* file.

Note:
- Use of the Euler buckling (PP) functionality requires a ***Standard*** or ***Professional*** license

5.5.1.5 Input / Output

Input / Output	Description	Default Value
Domain (in)	Design domain(s).	N/A
LCase (in)	A load or group of loads to be applied to the structure as an independent case.	N/A
Units (in)	Defines the units of length used for geometrical measures defined in Grasshopper. If nothing is specified, the Rhino units of length are assumed by default. Permissible values: mm, cm, m, km, in, ft Note that other units used by Peregrine and incorporating length (allowable stress, pressure etc.) are not modified when changing this input and may need to be updated manually.	mm
EulerPP (in)	Consider Euler buckling as part of the post processing stage by increasing the cross-sectional area of any element that does not meet the required Euler buckling capacity. Note that this differs to Euler buckling when considered as part of the layout optimization stage (see Section 5.5.2.4).	False
Effective Length (in)	Multiplier on the length of the member required to cause it to buckle as 'pinned – pinned'.	1.0
Settings (in)	General settings applied in this problem.	N/A
Mirror Plane (in)	Defines the plane at which mirror symmetry is applied. Note that the surface normal vector must be set such that the defined supports and loading lie on the plane or to the rear (inward). See section 6.1.4 for more details.	N/A
ProbSpec (out)	Problem specification for use in ***LO*** or ***ConstructTopology***.	N/A

Input / Output	Description	Default Value
Units (out)	Units used in Grasshopper.	N/A
Materials (out)	Permitted materials in each domain.	N/A

5.5.2 Layout Optimization (LO)

5.5.2.1 Icon

Figure 62 – The Layout Optimization icon

5.5.2.2 Component

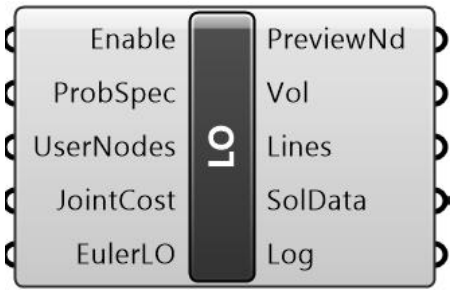

Figure 63 – The Layout Optimization component

5.5.2.3 Description

Solve layout optimization problem (see Section 6.1.1).

5.5.2.4 Summary

The ***Layout Optimization*** component takes the problem input, conducts an optimization and presents the output in a number of formats.

Optimization is toggled on or off using the ***Enable*** function. When enabled, the ***Layout Optimization*** takes the ***Problem Specification*** (***ProbSpec***) and performs a linear programming layout optimization on it (see Section 6.1.1 for more details).

While the nodal discretization within the problem is generally defined as part of setting up the ***Design Domain***, one can override this in the ***Layout Optimization*** by providing a list of nodes at

specific positions via the ***User Nodes*** (***UserNodes***) input. For an example of a Peregrine problem containing ***User Nodes***, run the *LO_UserNodes.gh* file.

By specifying a ***Joint Cost*** (***JointCost***), the solver will add a penalty to elements in the optimum layout in the form of a fictional additional length (*JC*) on top of the actual element length (*L*). This can help to generate simpler layouts with reduced complexity (as fewer elements mean a lower total penalization). However, the volume will be increased as a result. By specifying a number of ***Joint Costs*** as input, a number of different solutions can be found.

In truss cases where *JC* exceeds the length of an element, the value of *JC* is capped at 0.9*L* and a warning shown on the ***LO*** component. Without this cap, the problem would become unbounded and fail to reach a valid solution.

More about ***Joint Costs*** in the layout optimization formulation is provided in Section 6.1.3. For an example of a Peregrine problem containing ***Joint Costs***, run the *LO_JointCost.gh* file.

The output from the ***Layout Optimization*** component relates to the problem and optimum ***Solution*** (***Sol***):

With Euler buckling layout optimization (***EulerLO***) enabled, the solver will attempt to ensure stability against Euler buckling as part of the optimization process. Note that this requires an appropriate ***Effective Length*** to be specified in the ***Problem Specification*** (***ProbSpec***).

The Euler buckling process makes use of a heuristic approach and, as such, is not guaranteed to be successful in all situations. Additionally, the ***EulerLO*** functionality contains an ***EulerPP*** step at the end of each iteration (see Section 5.5.1.4). As such, it is anticipated that, when ***EulerLO*** is enabled, the solution will be the same irrespective of whether ***EulerPP*** is also enabled. A full outline of expected outcome, based on the two settings, is provided below:

EulerLO	**EulerPP**	**Behaviour**
✘ False	✘ False	Euler buckling is not considered as part of the analysis.
✓ True	✘ False	Euler buckling is considered during the layout optimization (LO) stage. At each iteration, elements are identified that do not meet the necessary Euler buckling capacity and their cross-sectional areas are increased to meet that required resist buckling. Subsequent iterations will then decide whether to retain these elements or substitute a different arrangement of elements instead. Resizing requires that an ***EulerPP*** check is undertaken. This is done automatically, irrespective of the separate setting for ***EulerPP*** in the ***ProbSpec*** component. Therefore, when ***EulerLO*** is enabled, the solution will be the same irrespective of whether ***EulerPP*** is also enabled.

EulerLO	EulerPP	Behaviour
✘ False	✓ True	An optimum topology is identified by the layout optimization stage and the elements in this structure are checked for their Euler buckling capacity. Where the check is failed, elements are re-sized to just resist bucking.
✓ True	✓ True	Euler buckling is considered during the LO stage and again as a final check on the optimised structure. When ***EulerLO*** is enabled, the solution will be the same irrespective of whether ***EulerPP*** is also enabled.

Note:
- Use of the Euler buckling (LO) functionality requires a ***Standard*** or ***Professional*** license
- ***EulerLO*** is not compatible with problems containing catenary materials or grillages

An example comparing Euler buckling as post-process and in-optimization steps is provided in the Euler_buckling.gh file.

The ***Volume*** (***Vol***) output supplies the volume (mm^3) of the optimum structure at the end of the optimization stage.

The ***Topology*** output delivers a representation of the optimum structure as line elements.

The ***Solution*** (***Sol***) output packages together all the required data relating to the optimization problem and optimal, least volume, structure in a form that can be used as input into the post processing stages or displayed within the Rhino environment.

The ***Solution Log*** (***Log***) output can be connected to e.g. a ***Panel*** in order to display information relating to the ***Layout Optimization*** iterations (Iteration Number, Nodes, Members and Volume).

For an example of a Peregrine problem containing a ***Layout Optimization*** component, run the *Simple2D.gh* file.

5.5.2.5 Input / Output

Input / Output	Description	Default Value
Enable (in)	Enable this component.	False
ProbSpec (in)	A combination of one or more **Domain**(s) and **Load Case**(s) for input into the **Layout Optimization** (LO).	N/A

Input / Output	Description	Default Value
UserNodes (in)	A set of user-defined nodal positions to use as potential member end points during optimization. Note that the ***NodeDiv*** input for ***Domain*** and ***Domain1D*** will not be used for the purposes of structural layout, but may still be used when calculating other analysis parameters (e.g. tolerances).	N/A
JointCost (in)	A Boolean to specify whether the point is restrained against movement in the Z direction object.	0.0
EulerLO (in)	Consider Euler buckling as part of the layout optimization. Slender members will be identified and suppressed during optimization.	False
PreviewNd (out)	Preview the nodes that are used in the layout optimization.	N/A
Vol (out)	Volume of the solution structure.	N/A
Lines (out)	The optimum structure represented by line elements.	N/A
SolData (out)	Optimization solution data.	N/A
Log (out)	Solution log.	N/A

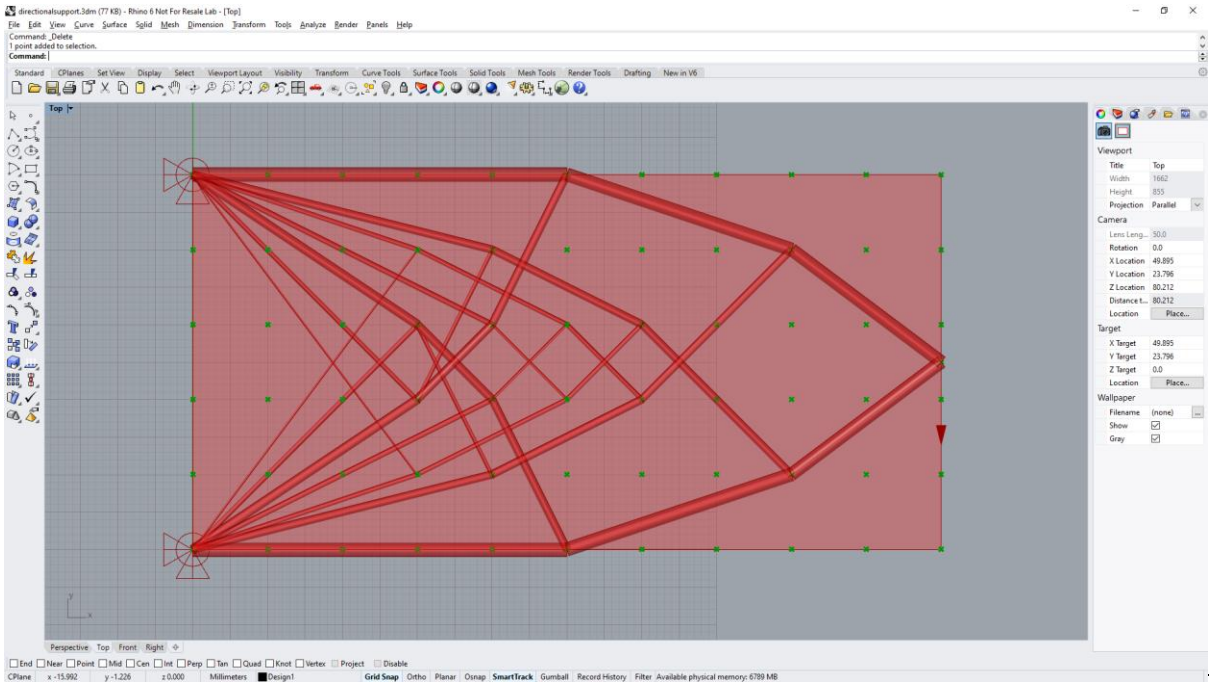

Figure 64 – The Simple2D problem in Rhino. Nodes previewed in green

5.6 Post Process

5.6.1 Geometry Optimization (GO)

5.6.1.1 Icon

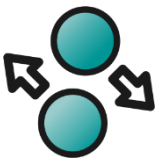

Figure 65 – The Geometry Optimization (GO) icon

5.6.1.2 Component

Figure 66 – The Geometry Optimization (GO) component

5.6.1.3 Description

Undertakes a Geometry Optimization (see Section 6.1.2).

5.6.1.4 Summary

The ***Geometry Optimization*** (***GO***) component undertakes a geometry optimization on the output from a previous optimization or simplify operation (***Sol***) by moving the positions of joints to more favorable positions (i.e. positions that will lead to a lower structural volume). As part of this process, the structure may be automatically converted from a pin-jointed representation (no moment resistance at joints) to a frame representation (moment resistance at joints). See (He, L et al., 2018) for the background theory and (Fairclough et al., 2021) for a discussion of the heuristics involved in the algorithm.

The ***Maximum Number of Input Nodes Allowed*** (***Limit***) is an integer describing the upper limit on the number of nodes that are allowed in a structure that is to undergo ***Geometry Optimization***. If the input ***Solution*** (***Sol***) contains more nodes than this value, a warning will be shown. By default, the limit is set to 100.

The ***Maximum Number of Iterations*** (***Iter***) is an integer that sets the number of geometry optimization iterations that are allowed to take place before the solution is returned. If convergence occurs before this value is hit, the geometry optimization will be curtailed. By default, the limit is set to 5.

The ***Volume*** (***Vol***) is a measure of the volume of the structure post-geometry-optimization.

The ***Solution*** (***Sol***) output packages together all the required data relating to the optimization problem and optimal structure in a form that can be used as input into further post processing stages or displayed within the Rhino environment using the ***SolutionViewer***.

The ***Solution Log*** (***Log***) can be connected to e.g. a ***Panel*** in order to display information relating to the ***LO*** iterations (Iteration Number, Nodes, Members and Volume).

For an example of a Peregrine problem containing ***GO*** component, run the *PostProcess_GO.gh* file.

Note that ***Geometry optimization*** will not be applied to members that belong to a grillage domain, see section 5.4.1.6 for details.

5.6.1.5 Input / Output

Input / Output	**Description**	**Default Value**
SolData (in)	Optimization solution data.	N/A
MaxJoints (in)	The maximum number of joints allowed in the input structure.	100

Input / Output	Description	Default Value
MaxIters (in)	The maximum number of geometry optimization iterations.	5
Vol (out)	Volume of the solution structure.	True
SolData (out)	Geometry optimization solution data.	N/A
Log (out)	Solution log.	N/A

5.6.1.6 Example

An example of the ***Geometry Optimization*** process is shown in Figure 67:

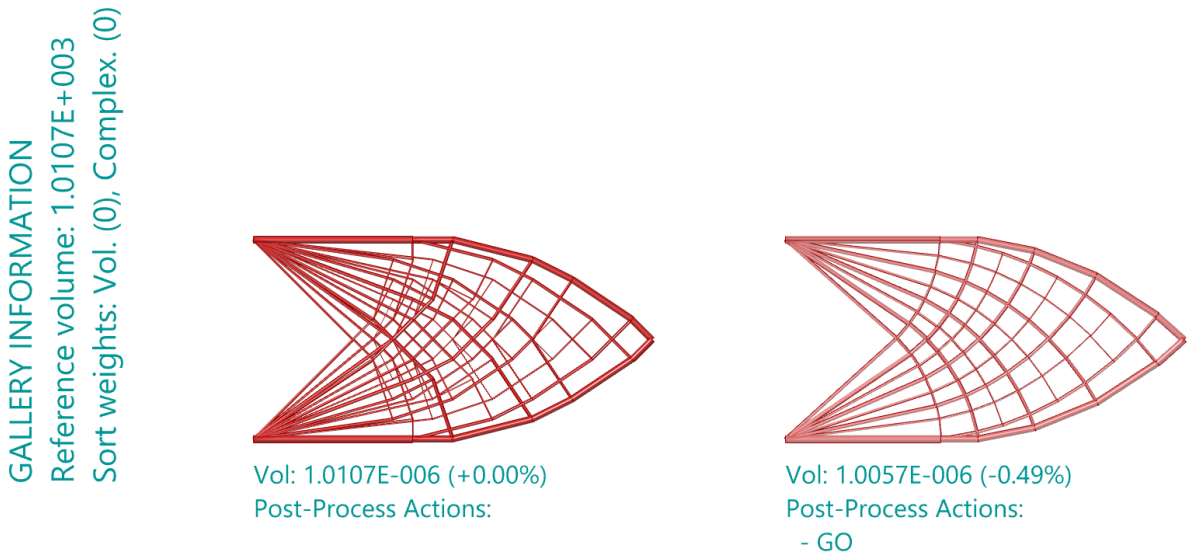

Figure 67 – The GO component used to reduce the complexity of a solution

5.6.2 Simplify (Simplify)

5.6.2.1 Icon

Figure 68 – The Simplification icon

5.6.2.2 Component

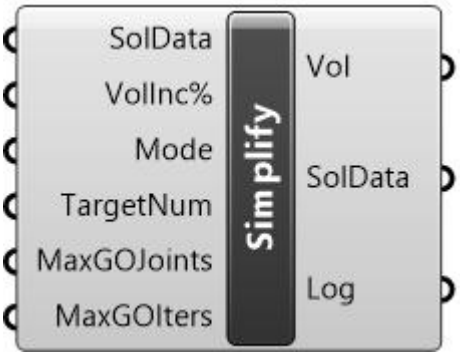

Figure 69 – The Simplify component

5.6.2.3 Description

Reduce the number of members that connect at joints using a variety of smooth Heaviside functions (see Section 6.1.3 for more information).

5.6.2.4 Summary

The ***Simplify*** component takes the output (***Sol***) from an optimization and attempts to reduce the number of elements that connect at the joints. The algorithm is summarized in the appendix (see 6.1.3). The background theory is described in more detail in He, L et al., (2018) and the additional heuristics involved are discussed in Fairclough et al., (2021).

The user specifies a permissible ***Volume Increase*** (***VolInc%***) in the volume of the structure that can be traded for this reduced complexity. A smooth Heaviside formulation is used to e.g., reduce the number of elements, while still possessing a volume that is below the permissible value.

Additionally, the user may specify the ***Simplification Mode*** (***Mode***) with an integer, used to select from one of three options that define the method used to simplify the input structure. These are:

1. ***All members*** – seeks to reduce the total number of members in the structure while keeping the volume within the specified percentage of volume increase (***VolInc%***)

2. ***Joint complexity*** – examines each joint within the optimum structure and attempts to reduce the number of elements meeting towards the specified value (***Num***), while keeping the volume within the specified percentage of volume increase (***VolInc%***)
3. ***Most complex joint*** - examines the most complex joint within the optimum structure and attempts to reduce the number of elements meeting at it towards the specified value (***Num***), while keeping the volume within the specified percentage of volume increase (***VolInc%***)

The ***Target # members*** (***TargetNum***) input.

The ***Volume*** (***Vol***) is a measure of the volume of the structure post-simplification.

The ***Solution*** (***Sol***) output packages together all the required data relating to the optimization problem and optimal structure in a form that can be used as input into further post processing stages or displayed within the Rhino environment using the ***Solution Viewer***.

The ***Solution Log*** (***Log***) can be connected to e.g. a ***Panel*** in order to display information relating to the ***LO*** iterations (Iteration Number, Nodes, Members and Volume).

Note that, if a simplification step is unable to complete, or has no effect on the solution, a warning message will be given by the component to alert the user to this fact. The output from the component will be the same as the input (allowing downstream actions to still occur).

Note that simplification will not be applied to members that belong to a grillage domain, see section 5.4.1.6 for details.

For an example of a Peregrine problem containing a ***Simplify*** component, run the *Structures_L_Truss.gh* file.

5.6.2.5 Input / Output

Input / Output	Description	Default Value
SolData (in)	Optimization solution data.	N/A
VolInc% (in)	The volume of the structure (as a %) that can be traded for reduced complexity.	5
Mode (in)	"The type of simplification to perform: 1) Reduce # members overall 2) Reduce # members at all joints 3) Reduce # members at the most complex joint(s). In each case, the simplification will try to keep the volume within the specified allowable increase (VolInc%). In modes 2 and 3, joint(s) will considered 'complex' if they contain more than TargetNum converging members.	1

Input / Output	Description	Default Value
TargetNum (in)	In applicable simplification modes, joint(s) will considered 'complex' if they contain more than TargetNum converging members.	4
MaxGOJoints (in)	The maximum number of joints that the SolData solution can contain. Above this number, a geometry optimization (GO) will not be attempted at the end of the simplification.	100
MaxGOIters (in)	The maximum number of geometry optimization iterations to undertake.	5
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Log (out)	Solution log.	N/A

5.6.2.6 Example

Please note that the ***Simplify*** component can sometimes produce truss configurations that are asymmetrical, despite having a symmetrical problem definition. This can generally be addressed by passing the output through a ***Geometry Optimization (GO)*** component and increasing the ***Iteration*** (***Iter***) count as necessary. For an example of a problem containing a ***Simplify*** component, see the *PostProcess_SimplificationJointComplexity.gh* file.

5.6.2.7 Example

An example of the ***Simplify*** process is shown in Figure 70:

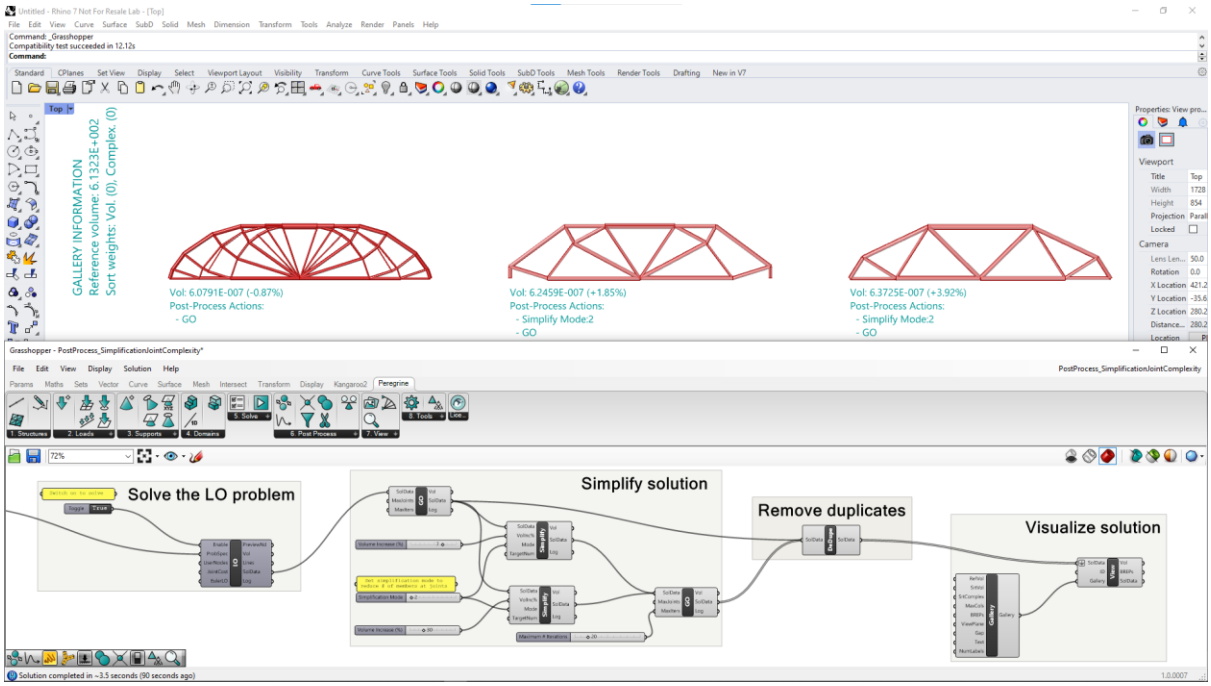

Figure 70 – Simplification of a truss structure

5.6.3 Crossovers (Crossovers)

5.6.3.1 Icon

Figure 71 – The Crossovers icon

5.6.3.2 Component

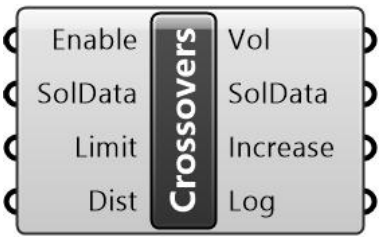

Figure 72 – The Crossovers component

5.6.3.3 Description

Create joints at locations where elements cross, or come close to crossing.

5.6.3.4 Summary

Occasionally, generally as a result of geometry optimization, elements can be found that overlap, but do not have a node at the overlap location. The ***Crossovers (Crossover)*** component takes the output (***Sol***) from an optimization and introduces nodes where these elements cross each other.

The user specifies a permissible ***Volume Increase Limit*** (***Limit***) (default of 100%). Above this value, the crossovers will not be generated.

The ***Crossover Distance Ratio*** (***Dist***) is a measure of the proximity of members below which a crossover will be created. The longest diagonal of the model bounding box is determined and the ratio value is the percentage of this measure which is used as the "search area" for crossovers. For example, in a 3D model with a 100mm bounding box, a value of "*Dist=1*" will ensure that elements with center lines that pass within 1mm of each other are checked and a crossover node added if required.

The ***Enable*** input is connected to a toggle, to allow the selective enabling and disabling of this component functionality. If changes are to be made to the ***Limit*** or ***Dist*** inputs, it is recommended that the component is left disabled while this is done.

The ***Volume*** (***Vol***) is a measure of the volume of the structure post-modification.

The ***Solution*** (***Sol***) output packages together all the required data relating to the optimization problem and optimal structure in a form that can be used as input into further post processing stages or displayed within the Rhino environment using the ***SolutionViewer***. It is generally recommended that a ***GO*** step is undertaken after crossovers are created.

The ***Volume Increase*** (***Increase***) output is a measure of the increase in structural volume encountered as a result of the crossovers functionality being undertaken.

The ***Solution Log*** (***Log***) can be connected to e.g. a ***Panel*** in order to display information relating to the component actions.

For an example of a Peregrine problem containing the ***Crossovers*** component, run the *PostProcess_Crossover.gh* file.

Note that ***Crossovers*** component is able to create crossovers between members that belong to a grillage domain, see section 5.4.1.6 for details.

5.6.3.5 Example

An example of the ***Crossovers*** process is shown in Figure 73:

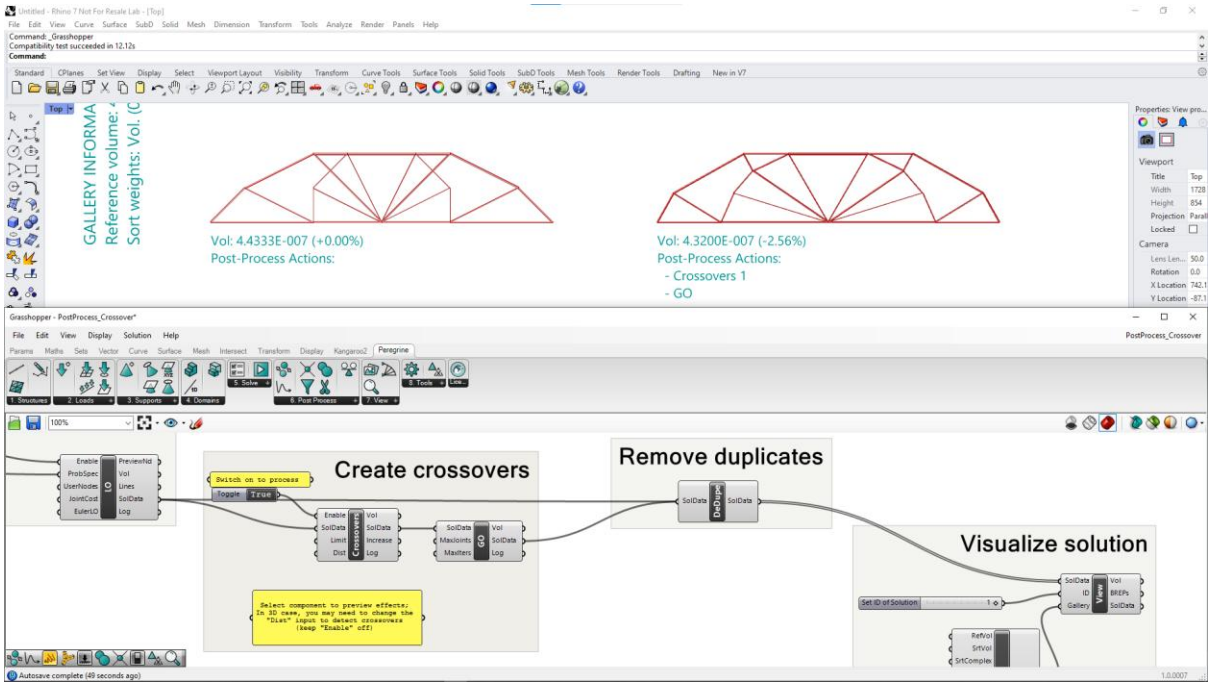

Figure 73 – An example of the output of the crossovers functionality

Here, the leftmost solution is the result of a normal ***LO*** step. ***Crossovers*** are created in the center image (note that the volume remains unchanged) and the resulting structure passed to the ***GO*** component, which moves joints to more optimal positions.

5.6.3.6 Input / Output

Input / Output	**Description**	**Default Value**
Enable (in)	Enable this component.	False
SolData (in)	Optimization solution data.	N/A
Limit (in)	Limit in solution volume percentage increase. Above this value, the solution offered by this component will be rejected.	100.0
Dist (in)	Measure of the proximity of members, below which a crossover will be created. The longest diagonal of the model bounding box is determined and the proximity is the percentage of this measure which is used as the "search area" for crossovers.	1
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Increase (out	Volume increase (%).	N/A

Input / Output	Description	Default Value
Log (out)	Solution log.	N/A

5.6.4 Filter (Filter)

5.6.4.1 Icon

Figure 74 – The Filter icon

5.6.4.2 Component

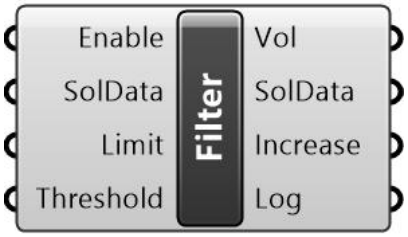

Figure 75 – The Filter component

5.6.4.3 Description

Removes the elements with a cross-sectional area below a specified Filter Threshold %. Note that this process may reposition joints to restore equilibrium.

5.6.4.4 Summary

The ***Filter*** functionality takes a Solution (***Sol***) from an optimization or post-process step and removes the elements with a cross-sectional area below a specified ***Filter Threshold %*** (***Threshold***).

The value of the ***Threshold*** is a user-specified percentage of the cross-sectional area of the largest element that exists in the solution. The component will attempt to remove elements with cross-sections below that which this relates to.

By removing elements, the equilibrium of forces at nodes will naturally be compromised to some extent. The Filter function will therefore attempt to also restore equilibrium where necessary, using a geometry optimization step.

The ***Volume Increase Limit*** (***Limit***) is the percentage increase in structural volume that is allowed to occur as a result of the ***Filter*** functionality. By default, the limit is set to 100.0.

The ***Enable*** input is connected to a toggle, to allow the selective enabling and disabling of this component functionality. If changes are to be made to the ***Limit*** or ***Threshold*** inputs, it is recommended that the component is left disabled while this is done.

For an example of a Peregrine problem containing ***Filter*** component, run the *PostProcess_Filter.gh* file.

Note that the filter component is able to act on members in grillage domains, see section 5.4.1.6 for details.

5.6.4.5 Input / Output

Input / Output	Description	Default Value
Enable (in)	Enable this component.	False
SolData (in)	Optimization solution data.	N/A
Limit (in)	Limit in solution volume percentage increase. Above this value, the solution offered by this component will be rejected.	100.0
Threshold (in)	A percentage of the cross-sectional area of the largest element that exists in the solution. Members with areas below this will be removed where possible.	5
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Increase (out	Volume increase (%).	N/A
Log (out)	Solution log.	N/A

5.6.4.6 Example

An example of the results of a ***Filter*** stage is shown in Figure 76:

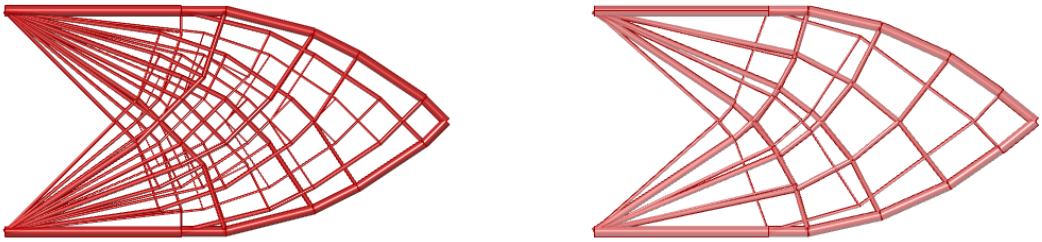

Figure 76 – Filtering the output from Layout Optimization stage (L = original, R = filtered)

On the right is the original solution. On the left is a filtered version, where a filter ***Threshold*** of 3% has been specified and a ***Geometry Optimization*** stage carried out after the filter. It can be seen that the resulting structure is both simpler and lower in volume.

5.6.5 Merge Joints (Merge)

5.6.5.1 Icon

Figure 77 – The Merge Joints icon

5.6.5.2 Component

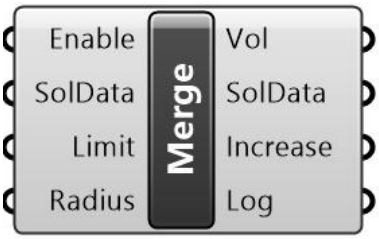

Figure 78 – The Merge Joints component

5.6.5.3 Description

Merge joints within the structure that lie within a prescribed distance of each other.

5.6.5.4 Summary

The ***Merge Joints (Merge)*** functionality accepts a Solution (***Sol***) from an optimization or post-process step and combines nodes (joints) within the structure that lie within a prescribed ***Merge Radius Ratio (Radius)***.

The value of the ***Radius*** (default = 5) is a measure of the proximity of nodes, below which a merge will be attempted. The longest diagonal of the model bounding box is determined and the ratio value is the percentage of this measure which is used as the "search area". For example, in a model with a 100mm bounding box, a value of "*Radius = 5*" will ensure that nodes which lie within 5mm of each other are checked and an attempt made to merge them.

By removing nodes and elements, the equilibrium of forces may be compromised to some extent. The ***Merge*** function will therefore attempt to also restore equilibrium where necessary, using a supplementary geometry optimization step.

The ***Volume Increase Limit*** (***Limit***) is the percentage increase in structural volume that is allowed to occur as a result of the ***Filter*** functionality. By default, the limit is set to 100.0.

The ***Enable*** input is connected to a toggle, to allow the selective enabling and disabling of this component functionality. If changes are to be made to the ***Limit*** or ***Radius*** inputs, it is recommended that the component is left disabled while this is done.

For an example of a Peregrine problem containing ***Merge*** component, run the *PostProcess_MergeNodes.gh* file.

Note that nodes that belong to a grillage domain will not be merged, see section 5.4.1.6 for details.

5.6.5.5 Input / Output

Input / Output	Description	Default Value
Enable (in)	Enable this component.	False
SolData (in)	Optimization solution data.	N/A
Limit (in)	Limit in solution volume percentage increase. Above this value, the solution offered by this component will be rejected.	100.0
Radius (in)	The distance between joints, below which a merge will be attempted. The longest diagonal of the model bounding box is determined and the Merge Radius is the percentage of this measure to be used as the "search area".	5.0

Input / Output	Description	Default Value
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Increase (out	Volume increase (%).	N/A
Log (out)	Solution log.	N/A

5.6.5.6 Example

An example of the results of a ***Merge*** stage is shown in Figure 79:

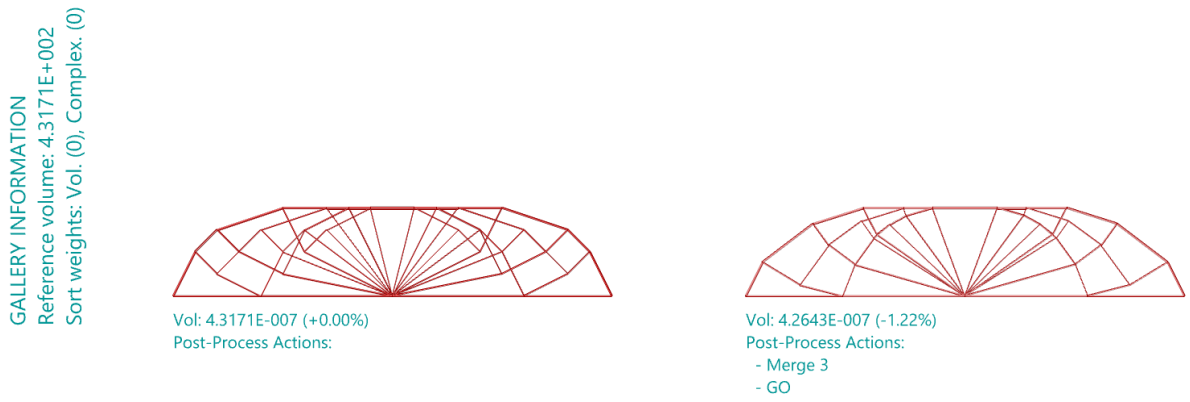

Figure 79 – A node Merge (L = original solution, R = node merge)

5.6.6 Reduce Complexity (Complexity)

5.6.6.1 Icon

Figure 80 – The Reduce Complexity icon

5.6.6.2 Component

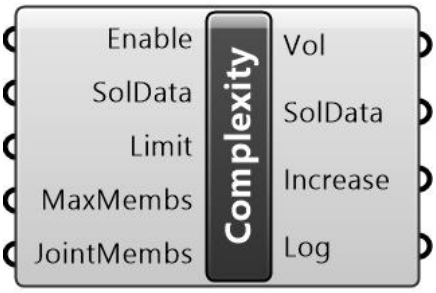

Figure 81 – The Reduce Complexity component

5.6.6.3 Description

Attempts to reduce the number of members according to a specified directive on the method used and within an allowable increase in structural volume.

Note:

- This feature is now deprecated. Please use the ***Simplify*** component instead (see Section 5.6.2).

5.6.6.4 Summary

The ***Reduce Complexity (Complexity)*** component accepts a Solution (***Sol***) from an optimization or post-process step and attempts to reduce the number of elements according to a user-specified directive on the method used, plus an allowable increase in structural volume.

Note that in general the ***Simplify*** component, which is based on a different numerical approach, is likely to produce superior results and is therefore recommended for most situations.

The value of the ***Maximum number of members at joint (BarsAtJoint)*** (default = 8) defines the greatest number of members that can join at any one location (joint). The ***Complexity*** functionality will then attempt to identify the minimum volume structure that conforms to this requirement. By removing nodes and elements, the equilibrium of forces may be compromised to some extent. The ***Complexity*** function will therefore attempt to also restore equilibrium where necessary, using a supplementary geometry optimization step.

The ***Volume Increase Limit*** (***Limit***) is the percentage increase in structural volume that is allowed to occur as a result of the operation of the ***Complexity*** functionality. By default, the limit is set to 100.0.

The ***Enable*** input is connected to a toggle, to allow the selective enabling and disabling of this component functionality. If changes are to be made to the ***MaxBars.*** or ***BarsAtJoint*** inputs, it is recommended that the component is left disabled while this is done.

The output is a new optimal ***Solution*** (***Sol***) along with data relating to its generation.

For an example of a Peregrine problem containing the ***Complexity*** component, run the *PostProcess_Complexity.gh* file.

Note that members that belong to a grillage domain will not be modified, see section 5.4.1.6 for details.

5.6.6.5 Input / Output

Input / Output	Description	Default Value
Enable (in)	Enable this component.	False
SolData (in)	Optimization solution data.	N/A
Limit (in)	Limit in solution volume percentage increase. Above this value, the solution offered by this component will be rejected.	100.0
MaxMembs (in)	Maximum number of members in the optimum structure.	100.0
JointMembs (in)	Maximum number of members that can meet at a joint.	8
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Increase (out	Volume increase (%).	N/A
Log (out)	Solution log.	N/A

5.6.6.6 Example

An example of the results of a ***Reduce Complexity*** stage is shown in Figure 82.

On the right is the original solution. On the left is a modified version, where a complexity threshold of 6 members per node has been specified and a ***GO*** stage carried out after the filter. It can be seen that the resulting structure is simpler but has a comparable volume.

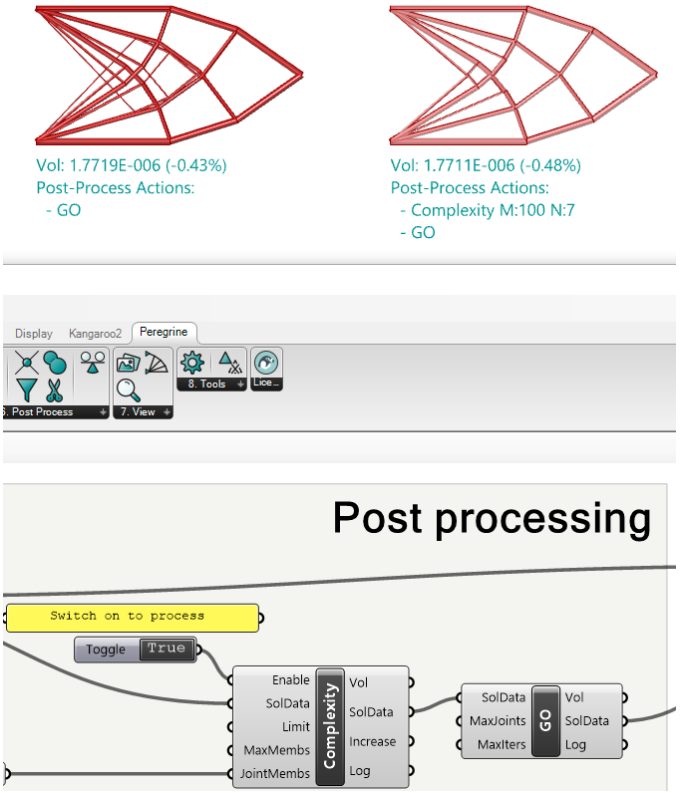

Figure 82 – Reducing the complexity of the solution (left = original, right = refined)

5.6.7 Stabilize (Stabilize)

5.6.7.1 Icon

Figure 83 – The Stabilize icon

5.6.7.2 Component

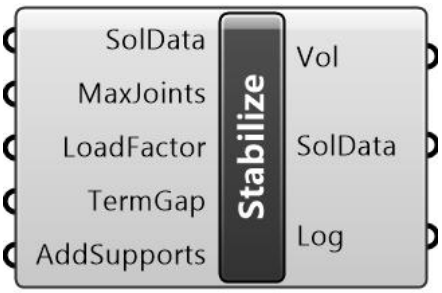

Figure 84 – The Stabilize component

5.6.7.3 Description

Takes a solution or constructed topology and attempts to ensure that the structure is linearly stable under the specified loads. Note that this step should generally be the last in the optimization workflow, as other components may introduce a state of unstable equilibrium once more.

Note:
- Use of the Stabilize component requires a ***Professional*** license.

5.6.7.4 Summary

The ***Stabilize (Stabilize)*** component accepts a Solution (***Sol***) from an optimization, constructed topology or post-process step and, using a semi-definite programming approach, attempts to ensure that the structure is linearly stable under the specified loads.

If the calculated load factor is less than a user-specified target value (***Load factor***, default value = 1), corresponding buckling modes will be analyzed to identify the most critical members. These members will then be re-sized or additional elements and previously eliminated support points added, to suppress buckling modes.

The ***Termination gap*** (***TermGap***) input is a measure on the percentage error. When the stabilization routine gets to (or below) this value, it will be allowed to terminate. A default of zero (no difference between completely stabilized structure and current solution) is used, but a higher magnitude could be allocated and therefore result in a faster calculation.

Enabling the ***Add Supports*** (***AddSupports***) input allows the stabilization routine to once again consider areas of the geometry that were specified as supports, but that were not 'active' in the previous optimal solution. These may provide restraint for subsequently added stabilizing elements.

Note that this step should generally be the last in the optimization workflow, as other components may introduce a state of unstable equilibrium once more, thus negating the effect of the stabilization.

As the calculations involved in ensuring stability may require a reasonable amount of computational effort, the value of ***Limit*** (default = 50) is used to act as a threshold on the maximum solution size that the component will accept. If the number of nodes in the problem (note, not the number of joints in the structure) that is passed to the component exceeds the allocated value, a warning is issued ("*Layout contains too many nodes that may lead to slow progress, please increase node limit")* and the stabilization is not undertaken. The user can choose to increase the value as necessary, on the understanding that this will require more resources.

The output is a new optimal ***Solution*** (***Sol***) along with data relating to its generation.

For an example of a Peregrine problem containing the ***Stabilize*** component, run the *PostProcess_GlobalStability.gh* file.

Note that the ***Stabilize*** functionality is incompatible with problems that contain a grillage domain, see section 5.4.1.6 for details.

5.6.7.5 Input / Output

Input / Output	Description	Default Value
SolData (in)	Optimization solution data.	N/A
MaxJoints (in)	Maximum number of joints that can exist in the input structure. Use to suppress stabilization of very complex structures, which may require longer solution times.	50
LoadFactor (in)	If the calculated stability load factor is lower than this value, stabilization measures are attempted (adding members or increasing the cross-section of existing parts).	1.0
TermGap (in)	A measure on the percentage error of the solution. When the stabilization routine gets to (or below) this value, it will be allowed to terminate.	5.0
AddSupports (in)	Allows the stabilization routine to consider areas of the geometry that were specified as supports, but that were not 'active' in the previous optimal solution. These may provide restraint for subsequently added stabilizing elements.	True
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Log (out)	Solution log.	N/A

5.7 View

5.7.1 Gallery Settings (Gallery)

5.7.1.1 Icon

Figure 85 - The Gallery Settings (Gallery) icon

5.7.1.2 Component

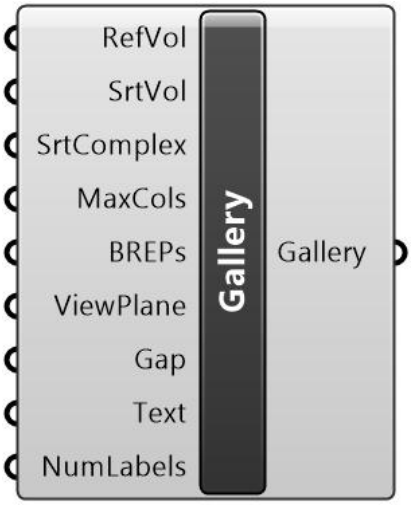

Figure 86 – The Gallery Settings (Gallery) component

5.7.1.3 Description

Settings for controlling the display of solutions as a gallery.

5.7.1.4 Summary

The ***Gallery*** component is used to control the display of multiple solutions in the ***View Solution*** (***View***) component. It accepts a ***Reference Volume*** (***RefVol***), against which the optimized volumes can be compared. By default, this volume is set to zero and no comparison is undertaken. The ***Gallery*** output contains the display settings to be used in the ***View Solution.***

For an example of a Peregrine problem containing the ***Gallery*** component, run the *LO_JointCost.gh* file.

5.7.1.5 Input / Output

Input / Output	Description	Default Value
RefVol (in)	Reference volume to compare solutions against (using document units).	0.0
SrtVol (in)	Higher numbers mean that volume is weighted higher when sorting solutions in the gallery. ***SrtVol*** and ***SrtComplex*** must both be above 1 to have an effect.	0.0
SrtComplex (in)	Higher numbers mean that structural complexity is weighted higher when sorting solutions in the gallery. ***SrtVol*** and ***SrtComplex*** must both be above 1 to have an effect.	0.0
MaxCols (in)	Max number of columns of solutions displayed in the gallery.	3
BREPs (in)	Display solutions as BREPs.	True
ViewPlane (in)	Select a view plane: (0) XY plane (1) XZ plane (2) YZ plane	0 (XY)
Gap (in)	Adjusts the gap between solutions.	1.5
Text (in)	Adjusts the scale of the caption text in the gallery.	1.0
NumLabels (in)	Set the number of solver output labels to display as captions under each solution.	2
Gallery (out)	Settings for controlling the display of solutions as a gallery.	N/A

5.7.1.6 Example

An example of the ***Gallery*** output is presented in Figure 87:

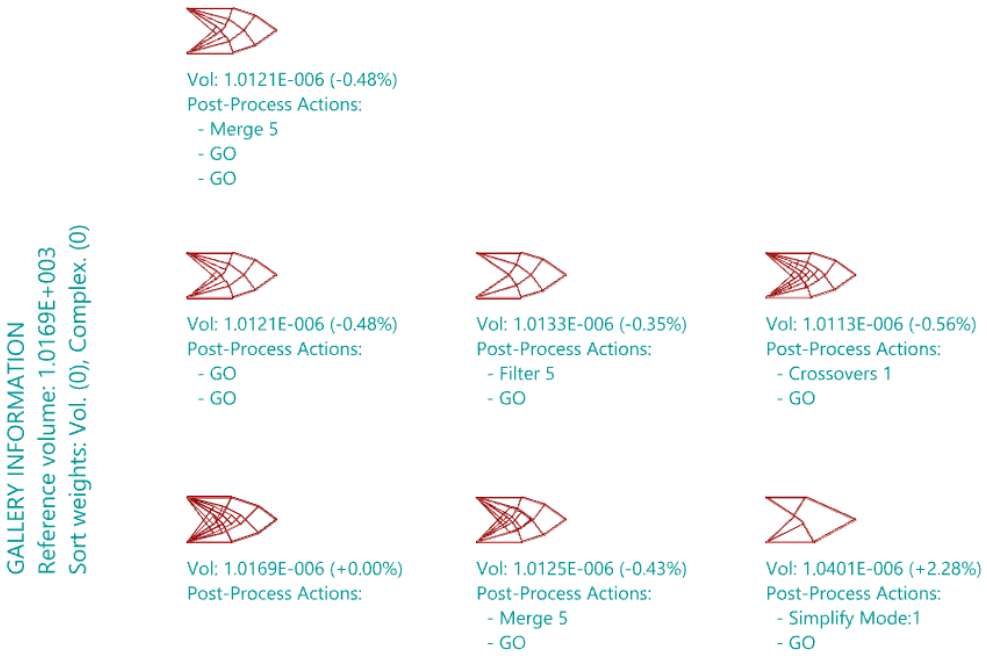

Figure 87 – An Example of Gallery output

5.7.2 Solution Details (Details)

5.7.2.1 Icon

Figure 88 – The Solution Details icon

5.7.2.2 Component

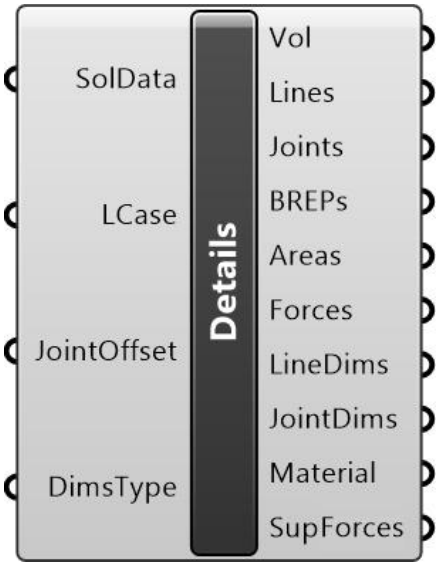

Figure 89 – The Solution Details (Details) component

5.7.2.3 Description

Outputs a range of information relating to the optimum structure for the specified Load Case. Accepts a Solution (***SolData***) from an optimization or post-process step and an integer specifying the Load Case.

5.7.2.4 Summary

The ***Solution Details (Details)*** component accepts a Solution (***Sol***) from an optimization or post-process step and an integer representing the ***Load Case*** that the user wishes to determine the details for.

The output from the component comprises a range of information relating to the optimum structure for the specified ***Load Case.***

Output from this component may be used to generate a single shell representation of the structure using the Grasshopper ***MultiPipe*** component (with Rhino version 7.2 or later).

5.7.2.5 Input / Output

Input / Output	Description	Default Value
SolData (in)	Optimization solution data.	N/A
LCase (in)	A load or group of loads to be applied to the structure as an independent case.	N/A

Input / Output	Description	Default Value
JointOffset (in)	Optionally include additional joints along members, each offset as a fraction of the member length from the member endpoints. If 0.0 then no additional joints are added. (Added to support compatibility with the MultiPipe component).	0.0
DimsType (in)	Member radius returned: 1: Outer radius (solid section or CHS) 2: Mid-surface radius (CHS) 3: Inner radius (CHS)	1
Vol (out)	Volume of the structure.	N/A
Lines (out)	The optimum structure represented by line elements.	N/A
Joints (out)	A list of coordinates describing the location of joints in the optimum structure.	N/A
BREPs (out)	The optimum structure represented by BREPs.	N/A
Areas (out)	A list of member cross-sectional areas (using document unit).	N/A
Forces (out)	A list of member axial forces (tension positive).	N/A
LineDims (out)	A list of key member radii of type determined by the DimsType input (using document unit).	N/A
JointDims (out)	The radius of the largest member connected to a given joint (using document unit).	N/A
Material (out)	Outputs, for each member, a grasshopper path structure indicating the index of the domain and material in the format: { Domain index, Material Index}.	N/A
SupForces (out)	Outputs, for each support point, the co-ordinate and forces experienced as part of the optimal solution (using document units)	N/A

5.7.2.6 Use with the Grasshopper MultiPipe component

An alternative to the ***Solution Viewer*** is to use the Grasshopper ***MultiPipe*** component, available in Rhino 7.2 or later, which generates a single shell representation of the structure, with smooth transitions between members at joints.

The ***MultiPipe*** component receives a set of curves as an input, with each curve being used to generate a member. Nodes are generated at points where two or more curves meet, and each node is assigned a radius that matches that of the closest point in a list of points passed via the ***SizePoints*** input. The radius of a pipe generated between nodes will smoothly transition between the radii of those nodes. The radius of each point in ***SizePoints*** is specified by the values in the ***NodeSize*** input.

When the ***Solution Details*** component is used with the ***MultiPipe*** component, the central portions of members can be sized correctly if intermediate joints (nodes) are created at positions:

$$\alpha L$$

$$1 - \alpha L$$

Where the member length is L and α is the offset from each endpoint as a fraction of the length of the member, specified by the ***JointOffset*** input of ***Solution Details.*** Note the following cases:

1. $(0 < \alpha < 0.5)$
   Each member is split at two intermediate locations. The surface segment generated between the two new joints created will be nominally circular, with a radius that corresponds to that of the member in the Peregrine solution. The surface will smoothly transition from the original joints (which may have a larger radius) to the new joints.
2. $(\alpha = 0.5)$
   Each member is split at its midpoint. The surface will smoothly transition from the original joints (which may have a larger radius) to the new midpoint joint.
3. $(\alpha = 0)$
   Members are not split and no additional joints are created. This is the default value for ***JointOffset.*** This setting should not be used with ***MultiPipe*** as the members will not be sized correctly.

Before connecting the **Solution Details (Details)** component with ***MultiPipe***, first specify a suitable ***JointOffset*** in the ***Solution Details*** component. Then connect the ***Lines, Joints,*** and ***JointDims*** outputs of ***Solution Details*** to the ***Curves, SizePoints,*** and ***NodeSize*** inputs of the ***MultiPipe*** component respectively**.** The ***EndOffset*** input of the ***MultiPipe*** component should typically be as small as possible but nonzero, but may be increased if the user finds that some pipes are bending too sharply at joints.

It is also possible to generate a hollow pipe structure using the ***MultiPipe*** component, by creating two ***MultiPipe*** structures using the inner and outer CHS radii of the members, converting these to watertight meshes and then subtracting the smaller mesh from the larger one. For an example of a Peregrine problem that is used to generate a ***MultiPipe*** structure, run the *HalfWheelMultipipe.gh* file.

5.7.3 View Solution (View)

5.7.3.1 Icon

Figure 90 – The View Solution icon

5.7.3.2 Component

Figure 91 – The View Solution component

5.7.3.3 Description

Display the optimum structure(s) within the Rhino environment. The output can also be passed to other components for further processing.

5.7.3.4 Summary

The ***View Solution*** component accepts a Solution (***Sol***) from an optimization or post-process step, an optional ***ID*** (to highlight a particular solution) and a set of ***Gallery*** parameters which outline how the solution(s) are displayed in Rhino.

The output from the component can be passed to other components for further processing. However, the central function of the component is to display the structure(s) within the Rhino environment.

With respect to the output:

The ***Volume*** (***Vol***) is the volume(s) of the structures passed into the component.

The ***Bars*** are a list of the elements in the various layouts, as BREPs.

The ***Solution*** (***Sol***) is data relating to the problem setup and optimum structures.

For an example of a Peregrine problem containing the ***View Solution*** component, run the *LO_JointCost.gh* file.

5.7.3.5 Input / Output

Input / Output	Description	Default Value
SolData (in)	Optimization solution data.	N/A
ID (in)	Set the ID of the solution to highlight in the gallery output.	0
Vol (out)	Volume of the solution structure.	N/A
SolData (out)	Optimization solution data.	N/A
Log (out)	Solution log.	N/A

5.8 Tools

5.8.1 Settings (Settings)

5.8.1.1 Icon

Figure 92 – The Settings icon

5.8.1.2 Component

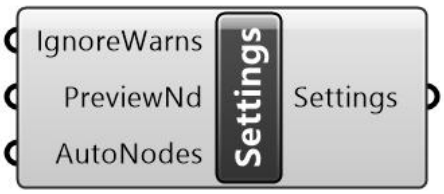

Figure 93 – The Settings component

5.8.1.3 Description

Adjust the Settings used in the Problem Specification.

5.8.1.4 Summary

The ***Settings*** component provides control over a number of LO solve related settings and outputs them in a form suitable for input into the ***ProbSpec*** component.

Occasionally Peregrine will issue warnings before attempting to solve a problem (e.g. relating to the solution time, if it is likely to be particularly long). These messages can be suppressed by setting the ***IgnoreWarns*** input to true.

The ***Preview Nodes (PreviewNd)*** input can be used to toggle display of the nodes generated by the ***Layout Optimization (LO)*** component.

For an example of a Peregrine problem containing the ***Settings***, component, run the *LO_UserNodes.gh* file

Note that, for files generated prior to Peregrine v5.0, the default ***AutoNodes*** setting may cause solutions to differ from the original. To rectify this:

- Use the "***(3) None***" setting for instances where ***NodeDiv*** = 0
- Use the "***(2) Surface***" setting for instances where ***NodeDiv*** > 0

5.8.1.5 Input / Output

Input / Output	Description	Default Value
IgnoreWarns (in)	Ignore any warnings (e.g. slow progress) that are generated on solve.	False
PreviewNd (in)	Preview the nodes used in the Layout Optimization (LO).	False
AutoNodes (in)	Controls how auto-generated nodes are considered alongside UserNodes, when these have also been defined. Default = 1. (1) All: Generate and consider auto-nodes on and throughout the domain (2) Surface: Generate and consider auto-nodes only on the surfaces of the domain (boundaries if 2D) (3) None: Do not generate or consider auto-nodes Note that, in all cases, supported and loaded nodes will be considered, irrespective of whether they are auto- or user- generated.	1
Settings (out)	Settings used in the problem specification.	N/A

5.8.2 Remove Duplicated Solutions (DeDupe)

5.8.2.1 Icon

Figure 94 – The Remove Duplicated Solutions (DeDupe) icon

5.8.2.2 Component

Figure 95 – The Remove Duplicated Solutions (DeDupe) component

5.8.2.3 Description

Accepts a series of solutions (***SolData***) and removes repeated instances (i.e. those with the same layout and optimal volume as an existing solution). This is especially useful when e.g. comparing the effect of various levels of joint cost penalization.

5.8.2.4 Summary

The ***Remove Duplicated Solutions (DeDupe)*** component accepts a series of Solutions (***Sol***) and removes repeated instances (i.e. those with the same layout and optimal volume as an existing one). This is especially useful when e.g. comparing the effect of various levels of joint cost penalization.

For an example of a Peregrine problem containing ***Remove Duplicated Solutions***, run the *LO_JointCost.gh* file.

5.8.2.5 Input / Output

Input / Output	**Description**	**Default Value**
SolData (in)	Optimization solution data.	N/A
SolData (out)	Optimization solution data, with duplicates filtered out.	N/A

5.9 License

5.9.1 Peregrine

5.9.1.1 Icon

Figure 96 – The License icon

5.9.1.2 Component

Figure 97 – The License component

Right-clicking on the Peregrine License component will bring up the context menu, as seen in Figure 98:

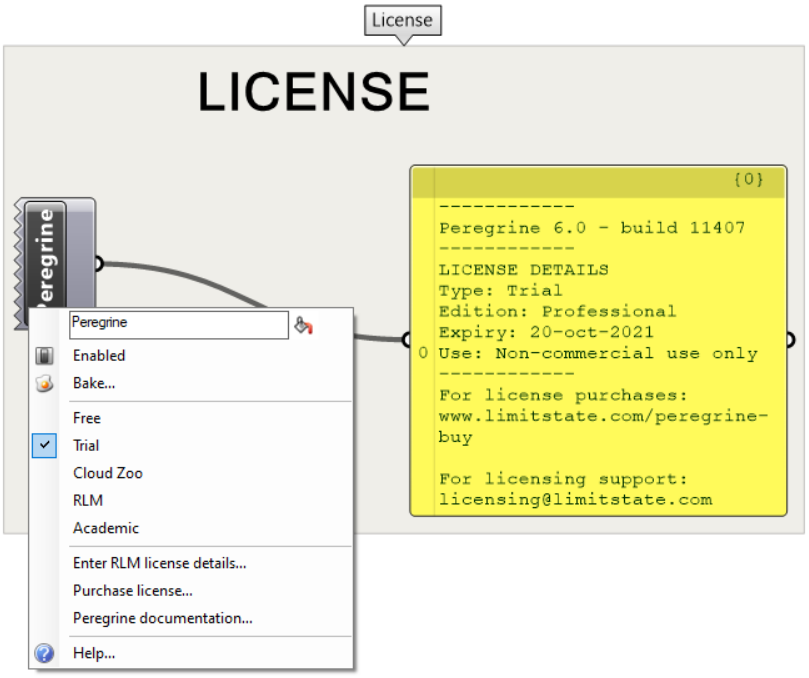

Figure 98 – The Peregrine License component and licensing context menu

5.9.1.3 Description

Provides information on the active license and means to switch to a different license when required.

5.9.1.4 Summary

The ***License*** component is used to change the license type, and includes information relating to the specific version of the software, as well as links to pertinent websites. There are five types of license available:

1. Trial
2. Free
3. Cloud Zoo
4. RLM
5. Academic

Links to purchase a license and view online documentation are also present. For further details on the different options available, please refer to Section 2.3.

5.9.1.5 Input / Output

Input / Output	**Description**	**Default Value**
Log (out)	Outputs the results of the license check	N/A

6 Appendix

6.1 Theory

6.1.1 Layout Optimization

The ***LO*** component undertakes a layout optimization of the problem. Layout optimization is a numerical approach utilized to design (near-) optimal truss structures. It directly identifies the optimal connectivity of nodes in a design domain.

A standard layout optimization has four steps:

1. Firstly, a design domain is specified along with load and support conditions;
2. Secondly, this design domain is discretized using nodes;
3. Thirdly, potential members are generated by connecting nodes, forming a ‘ground structure’;

4. Finally, the most efficient arrangement of members (which will be a small subset of the 'ground structure') is identified by solving a linear programming (LP) problem.

An example of a layout optimization is provided in Figure 99. Here, the design space is shown in (a), while the optimized structure is shown in (b).

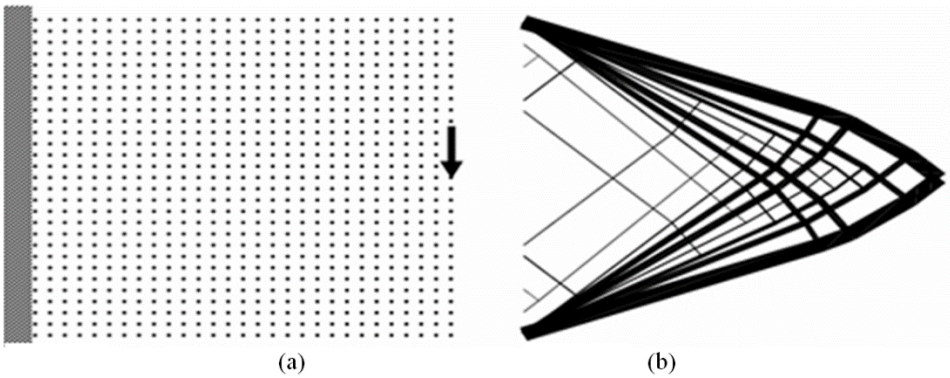

Figure 99 – Example layout optimization

It should be noted that a 'ground structure' normally contains a very large number of potential truss layouts, permitting highly efficient, though not necessarily practical, solutions to be found.

6.1.1.1 Formulation

The LP formulation for layout optimization was first proposed by Dorn et al., (1964) and more recently was made efficient for single and multiple load cases problems respectively by Gilbert and Tyas (2003) and Pritchard et al., (2005). The formulation used here is:

$$\min V = \mathbf{l}^{\mathrm{T}}\mathbf{a}$$

Subject to:

$$\text{for all } \alpha \in \mathbb{F} \begin{cases} \mathbf{B}\mathbf{q}^{\alpha} = \mathbf{f}^{\alpha} \\ -\sigma^{-}\mathbf{a} \leq \mathbf{q}^{\alpha} \leq \sigma^{+}\mathbf{a} \end{cases}$$
$$\mathbf{a} \geq 0$$

where V is structural volume, $\mathbf{l}$ and $\mathbf{a}$ are vectors of member lengths and areas, respectively. $\mathbf{B}$ is an equilibrium matrix comprising direction cosines; $\mathbf{q}$ is a vector containing the internal bar forces and $\mathbf{f}$ is a vector containing the external forces. Also σ^{+} and σ^{-} are limiting tensile and compressive stresses respectively, $\mathbb{F} = \{1, 2, \ldots, p\}$ is a load case set, where α is the load case identifier and p represents the total number of load cases.

6.1.1.2 Adaptive Solution Scheme

To improve computational efficiency of layout optimization, Gilbert and Tyas (2003) proposed an adaptive solution scheme. It starts with a 'ground structure' containing only a few potential

members, and then gradually adds new members into the 'reduced' optimization problem. It is important to note that the adaptive solution scheme has no impact on the structural efficiency of the outcome truss layout, i.e., it is mathematically guaranteed that the structural volume obtained is the same as that derived using the fully connected formulation. For more information on this, the reader is referred to the original paper. A Python implementation of this method is also freely available; see He et al., (2019).

6.1.1.3 Layout Extraction

The raw solution derived from layout optimization may contain a considerable number of thin members. Most of these are not structurally important and can be deleted from the design. For this reason, a threshold 'filter' number is used to select only the important members from the raw solution. This 'filter' number will be validated so the resulting layout is as efficient as the raw solution.

6.1.2 Geometry Optimization

The ***Geometry Optimization*** component seeks to both simplify and improve on the solution that it is passed. It does this by undertaking a geometry optimization (GO). Here, the joint positions of the structure as provided are moved to reduce the volume of the structure. As part of this process, it is possible that joints may be merged.

When considering the geometry optimization process in isolation, for a problem involving a single load case, without self-weight, the optimization problem is written as:

$$\min_{x,y,a,q} V = \mathbf{l}(\mathbf{x},\mathbf{y})^{\mathrm{T}}\mathbf{a}$$

Subject to:

$$\mathbf{B}_{(\mathbf{x},\mathbf{y})}\mathbf{q} = \mathbf{f}$$

$$-\sigma^{-}\mathbf{a} \leq \mathbf{q} \leq \sigma^{+}\mathbf{a}$$

$$\mathbf{a} \geq 0$$

Where **l** is a vector containing the lengths of the truss bars. The optimization variables in this case are *x*, *y*, *a* and *q*. It is evident that the objective function and equality constraint are both now non-linear (c.f. the layout optimization problem). Without loss of generality, the GO can be categorized as a non-linear, non-convex optimization problem.

For more information on the GO problem and the practicalities involved in its implementation, the reader is referred to He and Gilbert (2015).

6.1.3 Heaviside Simplification

The ***Simplify*** component is used to simplify a solution by removing members, at the cost of some volume increase. It leverages a smooth approximation to the Heaviside function, shown below, to minimize the area of as many members as possible while maintaining a valid solution.

$$H(x) = \coth(\mu)\tanh(\mu x)$$

This is shown graphically in Figure 100 for a range of μ values:

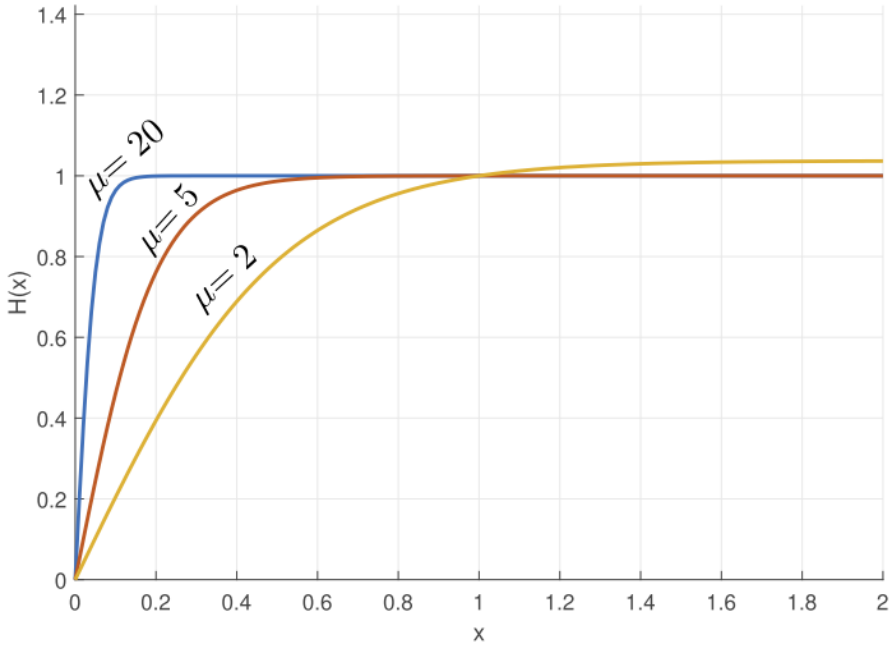

Figure 100 Heaviside smoothing for a range of μ values (after Fairclough et al., 2021).

It does this by reformulating the geometry optimization problem; a maximum acceptable volume increase for the simplification process (ε) is specified and this is added as a constraint to the optimization problem, and the objective function becomes one of minimizing the sum of the Heaviside function of member areas. Thus, the objectives and constraints become:

$$\min_{x,y,a,q} \Phi_M = \sum_{i=1}^{m} H(a_i/a_{ref})$$

Subject to:

$$\mathbf{B}_{(\mathbf{x},\mathbf{y})}\mathbf{q} = \mathbf{f}$$

$$-\sigma^{-}\mathbf{a} \leq \mathbf{q} \leq \sigma^{+}\mathbf{a}$$

$$\tilde{V} \leq (1 + \varepsilon)V_{ref}$$

$$\mathbf{a} \geq 0$$

The smooth Heaviside function decreases sharply as x approaches 0, and this gradient increases with μ. Consequently, the optimization problem will typically decrease the areas of a few members while leaving the remainder largely unchanged. Also note that, with increasing μ, the problem will favor more sharply reducing the area of smaller numbers of members.

Simplification iteratively solves this optimization problem for a series of increasing μ values, the output layout from a given optimization used as the input layout for the next. Afterwards, the algorithm attempts to filter all members with area below some threshold value, checks if the new layout is valid, and if not, filters again with a smaller threshold, repeating until a valid solution is found.

A limitation of this method is seen with problems that have small members which are crucial to the stability of the structure. In such cases, filtering will fail for all threshold area values larger than that of these members.

Another point to note is that a valid solution to a Heaviside simplification problem has not been optimized for minimum volume, and for this reason, it is advisable to run a ***geometry optimization*** step on the output layout of a ***Simplification*** optimization problem.

For more information, see He, L et al., (2018) and Fairclough et al., (2021).

6.1.4 Mirroring

6.1.4.1 Background

In many real-world applications, it is useful to be able to guarantee that the final structure is symmetric. In Peregrine, this is made possible by its mirroring functionality. Point and line loads, as well as supports of all types, can be mirrored.

In some cases, the design problem is inherently symmetric, so there is no reason to consider layouts which do not exhibit symmetry. In other cases, it may simply be advantageous to produce a solution which is symmetric. Possible reasons include:

- *Design requirements*
- *Aesthetics*: symmetric designs often look more visually appealing
- *Confidence*: a designed part may appear more rational to an engineer if it is symmetric. A symmetric design may inspire greater confidence even if it makes no difference from the point of view of analysis
- *Manufacture*: if two components are performing a similar task (e.g. on the left and right side of an assembly), there is no reason to design separate parts. Having fewer different types of component may make it easier to manufacture and catalogue the individual components and to assemble the final structure.

In addition to being able to guarantee a symmetric design, mirroring is also beneficial from the point of view of computational efficiency. When performing layout optimization the size of the problem can be greatly reduced. This can greatly reduce calculation times and computational memory requirements. In some cases, entire load cases can be removed from consideration as they are implied by existing load cases.

6.1.4.2 Implementation

Peregrine offers three types of mirroring:

- ***Default***: load is applied only to one side of the design domain (Figure 101).
- ***Symmetric***: load is applied simultaneously on both sides of the mirroring plane, with the direction of the load reflected (Figure 102).
- ***Antisymmetric***: load is applied simultaneously on both sides of the mirroring plane (Figure 103).

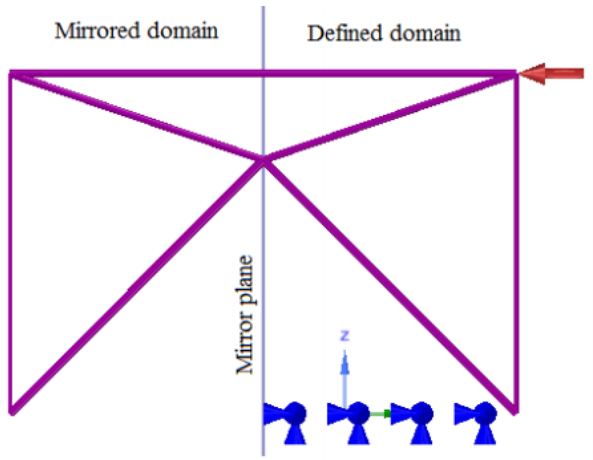

Figure 101 – Optimization using "Default" mirroring properties

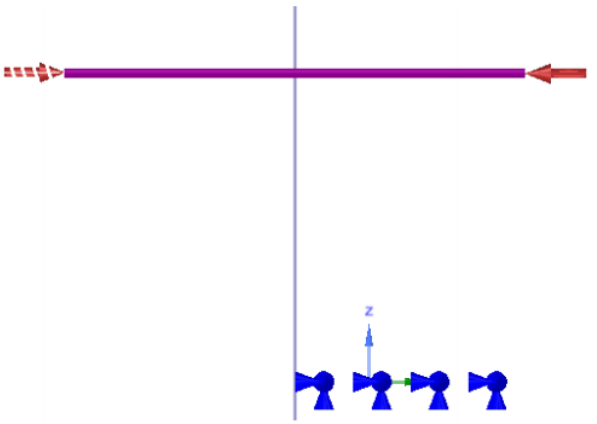

Figure 102 – Optimization using "Symmetric" mirroring properties

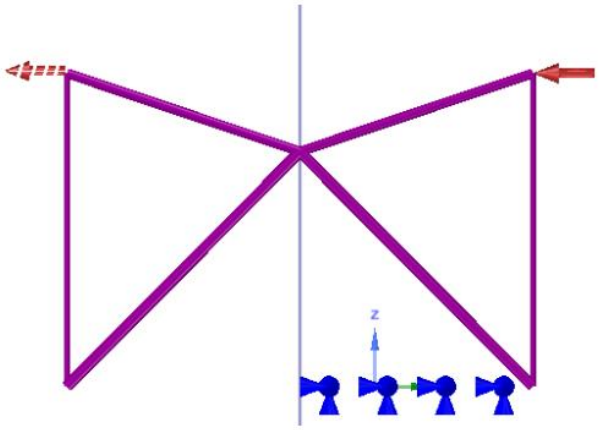

Figure 103 - Optimization using "Antisymmetric" mirroring properties

In order to use mirroring, the user must first specify a mirror plane as input to the ***ProbSpec*** component. It's important that the surface normal vector of this plane is set such that the defined supports and loading lie on the plane or to the rear (inward). If it is found to be facing in the wrong direction, the normal of a plane can be flipped using the Grasshopper ***Flip Plane*** (***PFlip***) component. The direction of the normal can be queried by combining ***Vector*** (***Vec***) and ***Vector Display*** (***VDis***) components as shown in Figure 104:

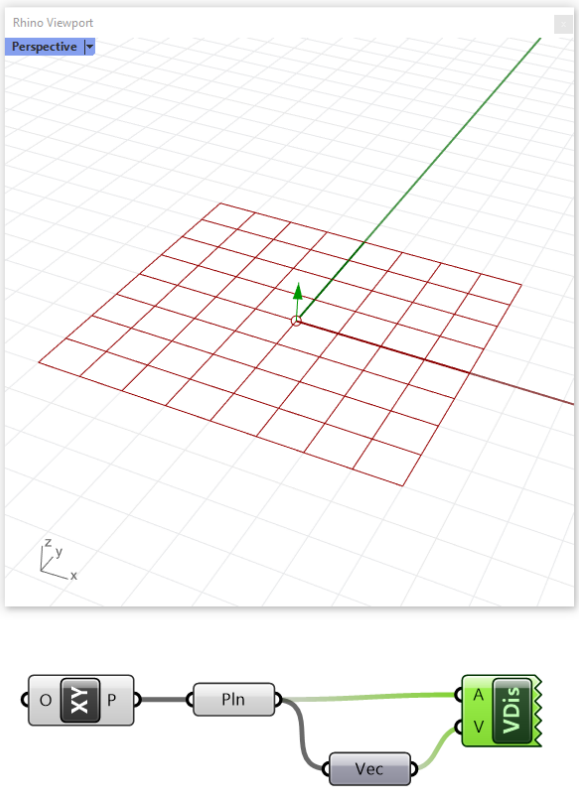

Figure 104 - Querying the normal direction of a plane using Vec and VDis components

When mirroring is implemented, nodes are arranged symmetrically so that any node is either on the mirroring plane or has a twin on the other side of the plane (Figure 105):

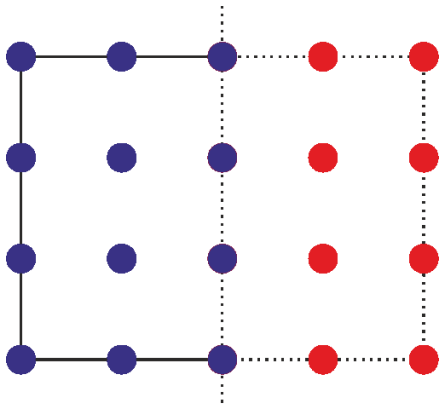

Figure 105 – Master nodes (blue) and their mirror nodes (red)

In Figure 105, master nodes are shown in blue and twin nodes in red. When the software analyses potential structures, it considers all potential connections between all master nodes. For each such master connection, there is a corresponding twin connection on the other side of the mirroring plane. As such, only one half of the design domain needs to be defined.

In the context of layout optimization, mirroring is simply the requirement that each master connection must have the same area as its twin. This condition ensures that the structure is symmetric.

Without the mirroring requirement, there is no guarantee that the design calculated by layout optimization will be symmetric. It is therefore advisable to use mirroring for problems which are symmetric.

It is important to point out that for a symmetric problem, the mirroring requirement (i.e. that each member must have the same area as its twin) does not impact on the efficiency of the design. In other words, optimal designs produced with and without mirroring would require the same overall volume of material.

6.1.4.3 Layout mirroring

Given a symmetry plane with unit normal $\hat{n}$, let $\overrightarrow{p_0}$ be an arbitrary point on the plane and $\overrightarrow{p_i}$ be a node in the problem specification. In layout mirroring, for every node $\overrightarrow{p_i}$, a symmetry node $\overrightarrow{p'_i}$ will be generated about the mirror plane such that

$$\overrightarrow{p'_i} = \overrightarrow{p_i} - 2d\hat{n}$$

Where $d = (\overrightarrow{p_i} - \overrightarrow{p_0}) \cdot \hat{n}$ is the distance of the point $\overrightarrow{p_i}$ from the mirror plane.

Members generated by solving the optimization problem will also be mirrored. The area a_i of the i[th] member in a domain defined in the problem specification will match the area a'_i of the mirrored member:

$$a_i = a'_i$$

Note that mirroring generates a symmetrical structure irrespective of load and support conditions, such that forces in mirrored members are not required to be the same. Consequently, a mirrored structure represents the minimum volume *symmetrical* solution for a particular problem, and not necessarily the minimum volume solution.

6.1.4.4 Load mirroring

There are two types of load mirroring: symmetry and antisymmetry. In both cases, the mirrored load has the same loading position and the same magnitude. Their directions are different. Given a load $\vec{f}$, its symmetrical load $\overrightarrow{f'_S}$ is calculated using:

$$\overrightarrow{f'_S} = \vec{f} - 2\left(\vec{f} \cdot \hat{n}\right)\hat{n}$$

Where $\hat{n}$ is the unit normal of the mirror plane. Its antisymmetrical load $\overrightarrow{f'_{AS}}$ is obtained using:

$$\overrightarrow{f'_{AS}} = -\overrightarrow{f'_S}$$

6.1.4.5 Support mirroring

Supports are automatically mirrored in the mirror plane.

6.1.5 Example Problem

The following example considers both geometry optimization (GO) and joint length rationalization of a problem. It is taken from He and Gilbert (2015).

The example was first considered by Hemp (1974). The problem involves application of a point load at mid-height between two pinned supports. Hemp quoted the analytical volume to be 4.34P L/ σ , but Lewinski (2005) repeated the calculations using greater precision to obtain a more accurate solution, 4.32168P L/ σ .

A sample layout optimization solution and corresponding rationalized (***GO***) solutions are also shown in Figure 106. It is evident that both rationalization techniques allow simplified solutions to be obtained when compared to the original layout optimization solution (a). However, whereas the volume associated with the solution obtained using joint length rationalization (b) is 1.49 % above the exact value, the solution obtained using geometry optimization rationalization (c) is only 0.23 % above the exact value, a significant improvement on the original layout optimization error of 0.75 %.

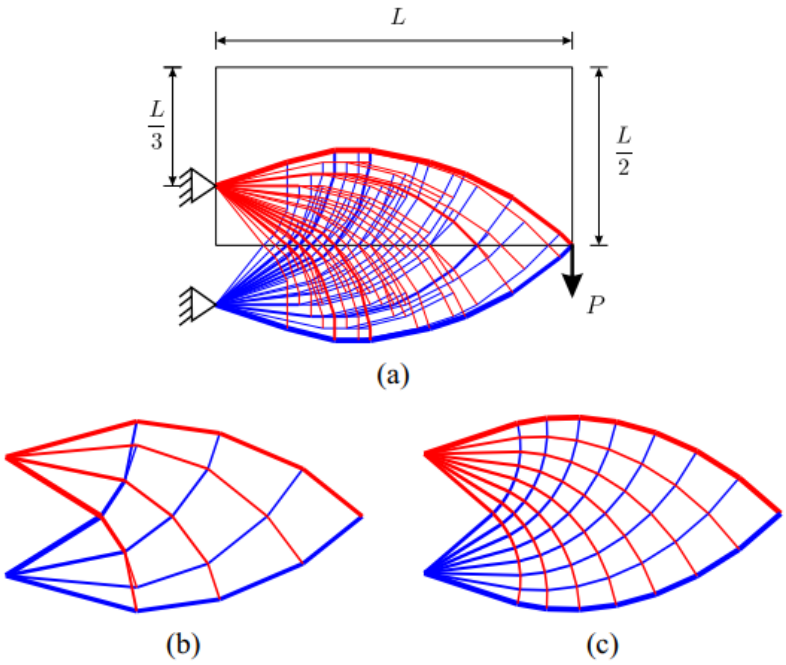

Figure 106 – Rationalization of cantilever truss using joint costs and GO (b) and GO (c)

6.2 Known Issues

- It is not possible to prescribe limits on structural deflection. To address this, stress limits can be scaled down to compensate.

7 References

Dorn W (1964). Automatic design of optimal structures. *Journal de Mecanique*, 3:25–52.

Gilbert M and Tyas A (2003), Layout optimization of large-scale pin-jointed frames. *Engineering Computations*, 20(8):1044-1064.

He L and Gilbert M (2015), Rationalization of trusses generated via layout optimization. *Structural and Multidisciplinary Optimization*, 52(4):677-694.

He, L., Gilbert, M., Shepherd, P., Ye, J., Koronaki, A., Fairclough, H., Davison, B., Tyas, A., Gondzio, J., & Weldeyesus, A. (2018). In: Mueller C, Adriaenssens S (eds) Creativity in Structural Design: A new conceptual design optimization tool for frame structures: Proceedings of the IASS Symposium 2018.

Fairclough HE, Gilbert M, Pichugin AV, Tyas A, Firth I (2018), Theoretically optimal forms for very long-span bridges under gravity loading. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences , 474:20170726.

Fairclough, H.E., He, L., Pritchard, T.J. et al., (2021). LayOpt: an educational web-app for truss layout optimization. Struct Multidisc Optim 64, 2805–2823

He, L., Gilbert, M. Rationalization of trusses generated via layout optimization. Struct Multidisc Optim 52, 677–694 (2015). https://doi.org/10.1007/s00158-015-1260-x

He L, Gilbert M and Song X (2019), A Python script for adaptive layout optimization of trusses. Structural and Multidisciplinary Optimization, 60(2):835–847.

Hemp WS (1974), Michell frameworks for uniform load between fixed supports. *Engineering Optimization* 1:61–69.

Lewinski T (2005) Personal communication.

Parkes EW (1975), Joints in optimum frameworks. *International Journal of Solids and Structures*, 11(9):1017–1022.

Pritchard T, Gilbert M and Tyas A (2005) Plastic layout optimization of large-scale frameworks subject to multiple load cases, member self-weight and with joint length penalties. *Proc of 6th World Congresses of Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brazil.

The Innovation Centre
217 Portobello
Sheffield, S1 4DP
United Kingdom

limitstate.com/peregrine
info@limitstate.com
+44 (0) 114 224 2240