USER DOCUMENTATION

INTRA**LATTICE**

CORE MODULES

v0.7.5

PREFACE

VERSION **0.7.5** — ALPHA

INTRA**LATTICE** is a C# plugin for Grasshopper, used to generate solid lattice structures within a design space. It was developed as an extensible, open-source alternative to current commercial solutions. As an ongoing project developed at McGill's Additive Design & Manufacturing Laboratory (ADML), it has been a valuable research tool, serving as a platform for breakthroughs in multi-scale design and optimization.

By providing a modular approach to lattice design, and giving you full access to the source, we hope to collectively explore lattice design at a deeper level, and consequently, engineer better products.

WEBSITE — http://intralattice.com

GITHUB — https://github.com/dnkrtz/intralattice/

DEVELOPER DOCS — http://intralattice.com/devdocs

THE MIT LICENSE

OPEN SOURCE

SUPPORT

If you have any issues, questions or feedback — please contact support@intralattice.com

LEAD DEVELOPERS

This project was started in the summer of 2014. Although the process was a collaborative effort, the contributions can be broadly categorized as follows:

Aidan Kurtz — Development of CORE modules, website and documentation.

Yunlong Tang — Development of OPTIMIZE modules (PhD research).

Prof. Fiona Zhao — Project supervisor and head of the research lab.

ACKNOWLEDGEMENTS

Many students/researchers have contributed to the project since its inception.

Marc Wang —

Ken Nsiempba —

Huiyuan Yang —

If you submit a pull request to the GitHub repository, and it is merged, your name and contribution will be listed. For more information on how to contribute, refer to the Developer Documentation.

The mesh generation methods used in this project are based on the Exoskeleton algorithms developed by David Stasiuk. And of course a big thanks to David Rutten for his work on Grasshopper.

TABLE OF CONTENTS

0.0 BACKGROUND

The freedom of form enabled by 3D printing has allowed engineers to integrate new orders of complexity into their designs. The goal of this research was to develop **a set of CAD tools for generating solid lattice structures** within a design space. The software would be used to:

- Reduce volume/weight while maintaining structural integrity.
- Increase surface area as a means of maximizing heat transfer.
- Generate porosity in bone scaffolds and implants.
- Serve as a platform for design optimization.

In doing so, it should always output a watertight mesh suited for 3D printing. The lack of flexibility of current software solutions was the motive for this project. We wanted to develop a flexible platform more conducive to research, which would allow us to explore and experiment with lattice design at a deeper level. The obvious first step was to decide in which environment we would develop our system. Rhinoceros is known to be very open ended, having its own engine for interpreting scripts (Python, C# and VB), and a powerful plugin SDK (RhinoCommon). Its Grasshopper addon is a visual programming tool widely used in architecture which provides an ideal interface for systematic design. In this visual interface, parameters and function components are combined sequentially to carry out the design of 3D models. By developing a set of custom components for Grasshopper, we could establish a modular workflow for generative lattice design.

That being said, if you are not familiar with Grasshopper, you are highly encouraged to have a look at the latest Grasshopper Primer, to bring you up to speed.

0.1 SYSTEM REQUIREMENTS

Operating System:	Windows 7 or 8 (64-bit recommended)
RAM:	8GB or more
Video Card:	OpenGL 2.0 capable video card
CPU:	No more than 63 CPU cores

0.2 INSTALLATION

The following **required software** should be installed on your system.

- **Rhinoceros 5**
- **Grasshopper**

Next, if you haven't yet, download the latest version of INTRA**LATTICE**

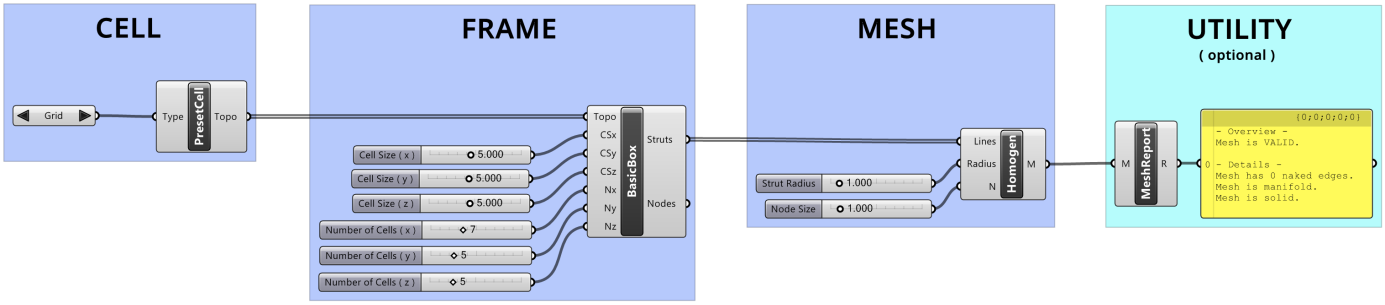
- **Intralattice**

To install, simply drag the 'IntraLattice.gha' file into your Grasshopper viewport. A new toolbar will appear.

Section 1

CORE FRAMEWORK

As mentioned previously, INTRALATTICE is a set of components for Grasshopper, a visual programming tool with an intuitive interface for creating algorithms that generate 3D models. This is particularly useful for designing cellular structures; a process that is highly parametric/systematic by nature. The core algorithm is split into modules

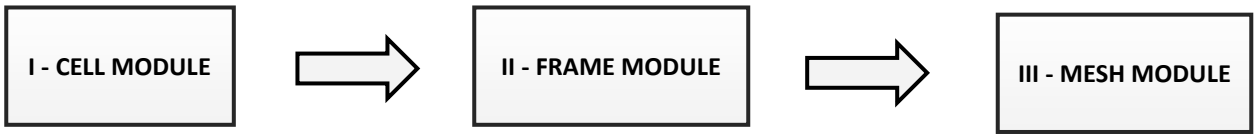


The benefits of defining a modular workflow include:

- **Flexibility** : each module can be changed independently.
- **Progressive computation** : you can preview each module before computing the next.

1.0 CORE MODULES

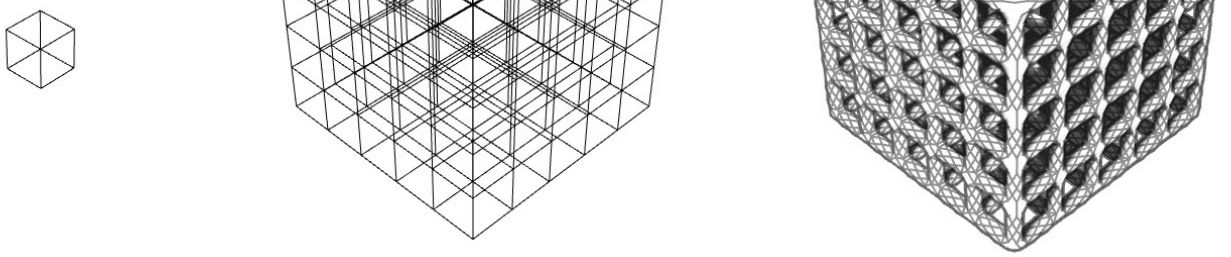
The core framework is shown below.



Each module serves a specific purpose, but this purpose can be accomplished in a variety of ways.

Also keep in mind that when we speak of curves, these may also be lines.

To illustrate the framework concept, consider the following simple example for a cubic design space.



Section 2

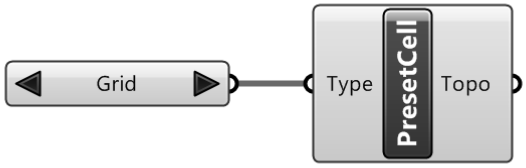
CELL MODULE

In our framework, the unit cell of the basis of the topology. At the moment,

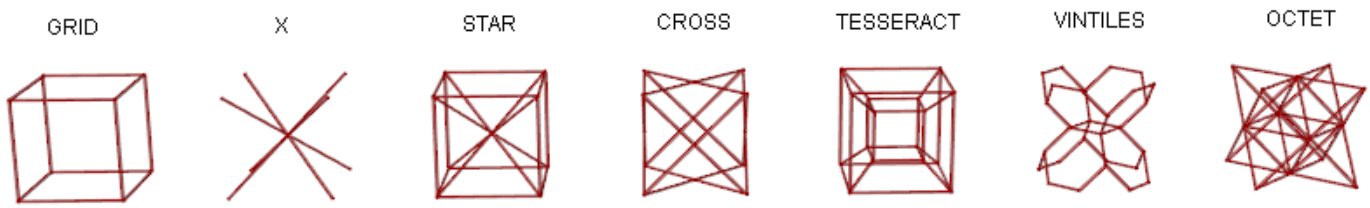
Generates the unit cell, as a list of lines. The **PresetCell** component provides a selection of built-in unit cells. Users may also define custom unit cells with the **CustomCell** component. This custom design can be imported from any CAD software, or defined directly in Grasshopper.

2.0 PRESET CELL

Built-in selection of unit cell topologies.

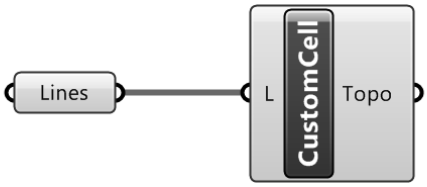


EXAMPLE OUTPUT



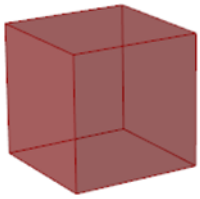
2.1 CUSTOM CELL

This component can be used to pre-process a custom unit cell. It will verify that the cell is valid, and return an error if it fails any of the validity tests.



2.1.0 What is a valid unit cell?

All unit cells generated by the PresetCell component are valid, and to that extent, the following information isn't useful. However, when designing custom unit cells, you need to keep in mind that the INTRALATTICE framework is built around a set of assumptions. The primordial assumption is that the cell has a cubic bounding box.

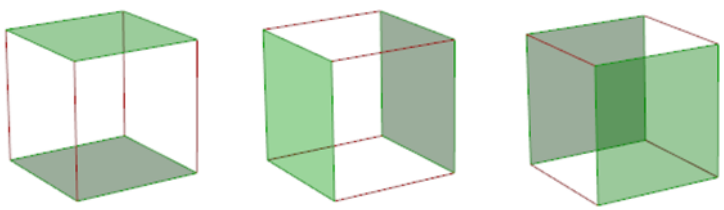


Requirement 1 : All struts are linear

The next two requirements are concerned with continuity within the lattice.

Requirement 2 : Each face of the bounding box of the cell has at least 1 node lying on it.

Requirement 3 : Opposing faces of the bounding box have the same nodes



2.1.0 Import a unit cell from any CAD software

Once you've created a unit cell as a set of lines, save it in a universal format (ex: _____).

1) **Open the file** in Rhinoceros.
   A) In Rhinoceros, go to "File -> Open".
   B) Select your file.

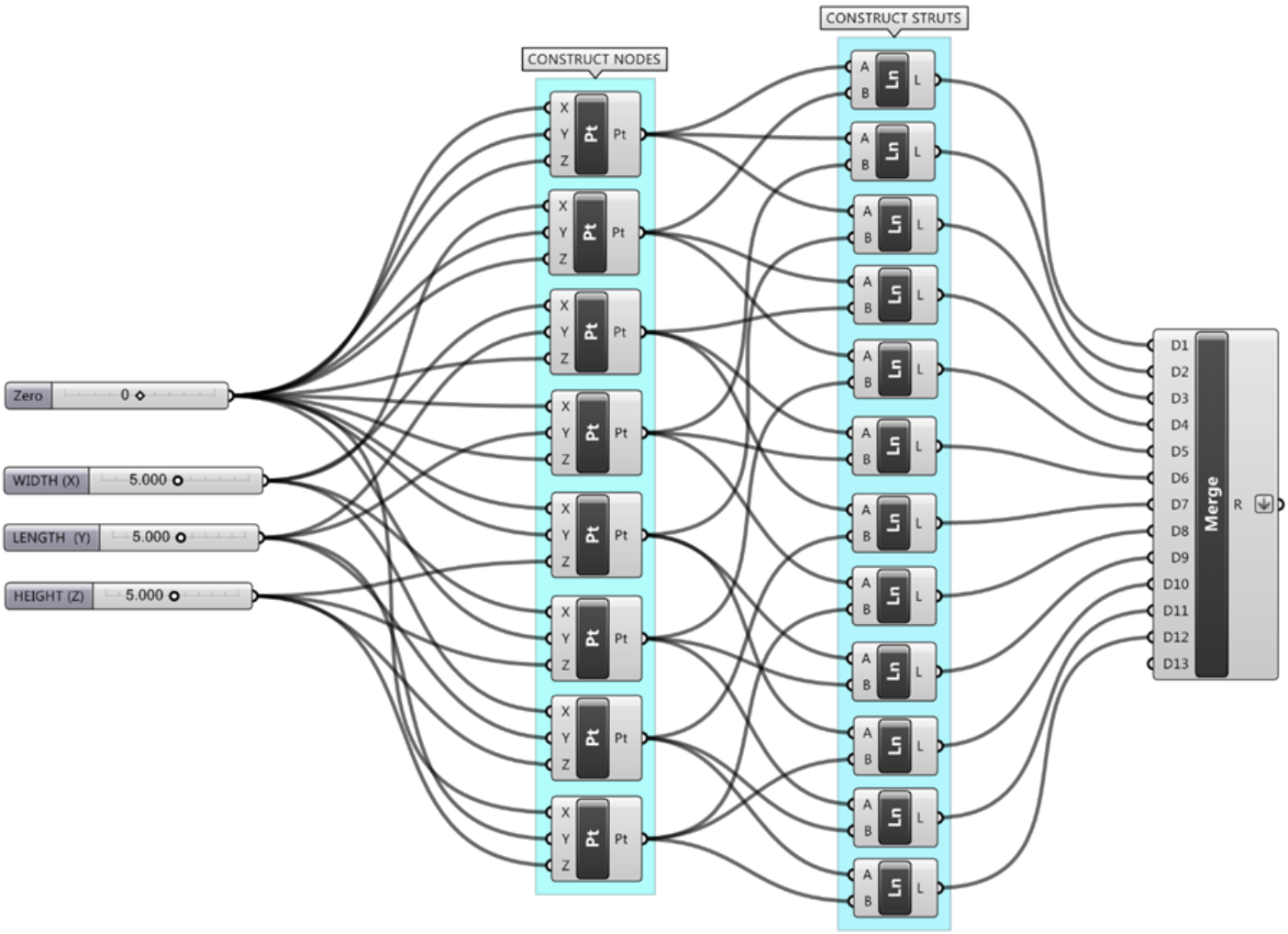
2) **Set the curves** in Grasshopper.
   A) In Rhinoceros, select all lines that make up your unit cell.
   B) In Grasshopper, right-click on the 'L' input, then choose "Set multiple curves".

3) **Internalise the data** on the CustomCell component.
   A) In Grasshopper, right-click on the 'L' input, then choose "Internalise data".
   The unit cell geometry is now being stored inside Grasshopper.

4) **Remove curves** from Rhinoceros.
   A) In Rhino3D, select all the lines and click Delete.

2.1.1 Create custom cell in Grasshopper

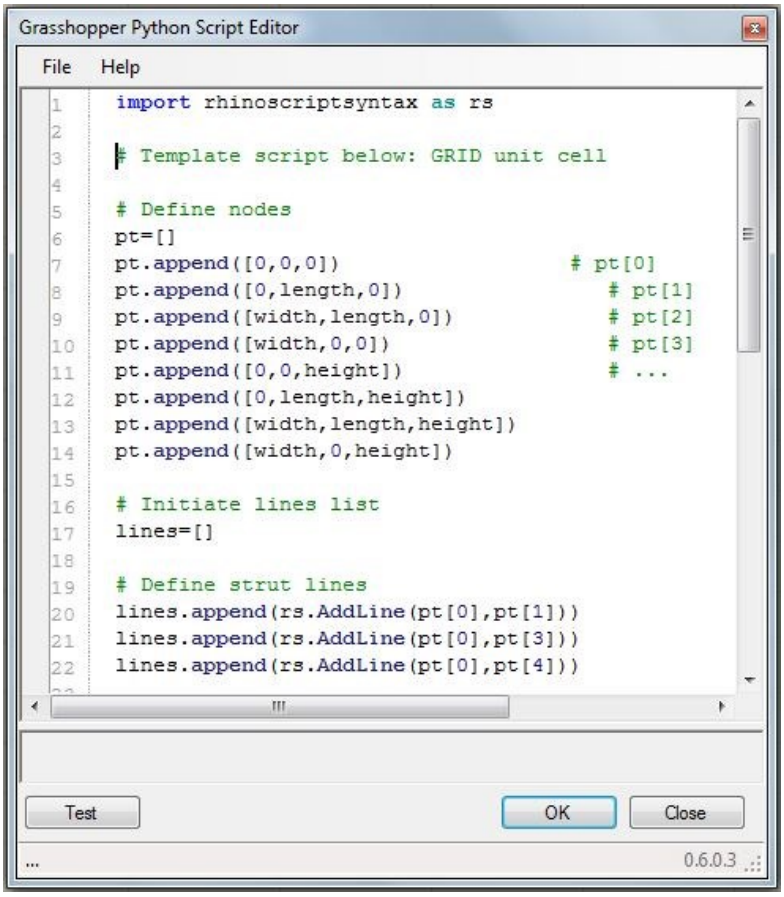
Another efficient way of designing unit cells is directly in Grasshopper. The figure below shows a simple example of how this could be done for the simple grid topology.



2.1.2 Create custom cell in C#/VB/Python script

Grasshopper has scripting capabilities for C# and VB.

Let’s consider an example with Python, for it’s relative ease-of-use.



Section 3

FRAME MODULE

The frame module generates a wireframe of the lattice structure, based on the unit cell generated by cell module, and a design space.

Generates the lattice wireframe, as a list of lines (and a data tree of the nodes). This wireframe is based on the unit cell and a design space. The **BasicBox** and **BasicCylinder** components generate lattices within a pre-defined design space; ideal for quickly testing new cell topologies, and 3D printing samples for testing. The **ConformSS**, **ConformSA** and **ConformSP** components generate lattices that conforms to a surface-based design space. The **UniformDS** component generates a trimmed uniform lattice within a mesh or brep design space.

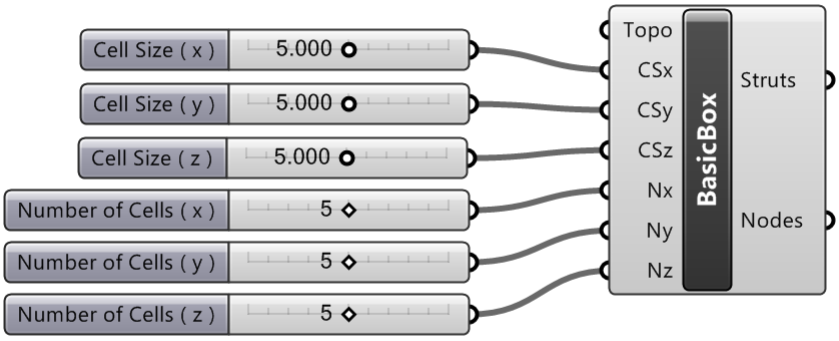
Note that our lattice structures generated are cellular structures by design, however, when pseudo-randomization is introduced, the cell boundaries dissolve (more on this later).

3.0 HOW TO DEFINE YOUR DESIGN SPACE

Important considerations: The frame will be thickened into a solid at the next step. When this happens, struts at the boundary will be thickened beyond the boundary. That being said, you may want to offset, or shrink your design space, to account for this. This is quite easy to do in Grasshopper, using any of the ‘Offset’ or ‘Scale’ components.

3.1 BASIC BOX

Description — Generates a lattice box.



INPUTS

Topo	List of lines coming from the unit cell module.
CSx, CSy, CSz	Size of unit cells in each of the xyz Cartesian coordinates .
Nx, Ny, Nz	Number of unit cells in each of the xyz Cartesian coordinates.

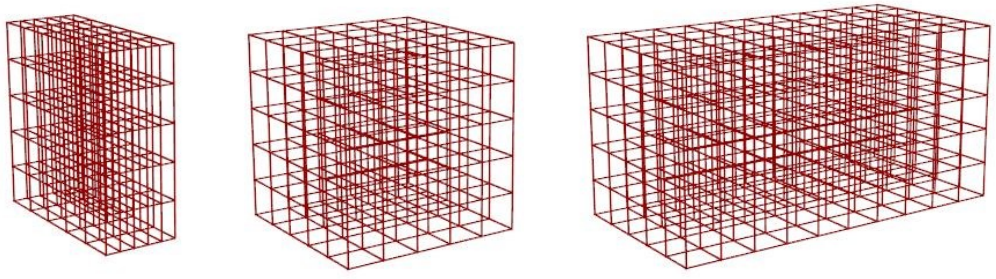
OUTPUTS

Struts	List of curves representing the lattice struts.
Nodes	Tree of points representing the lattice nodes.

EXAMPLES

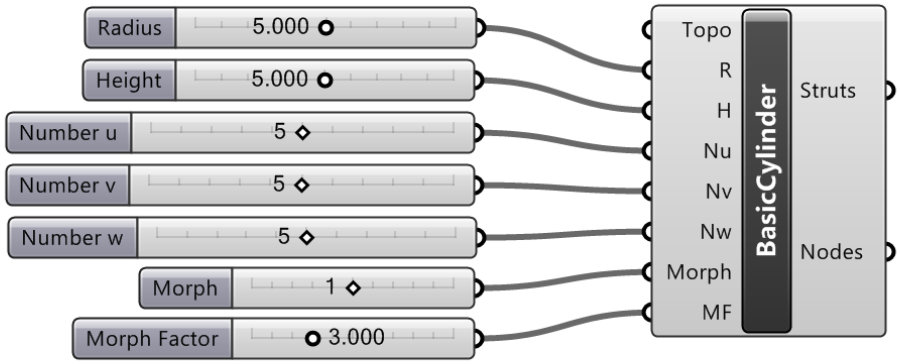
At this point, everything is pretty straight-forward. You can play with the size of unit cells, and also the number of cells in each direction.

Note that all examples in this documentation will use the grid unit cell topology.



3.2 BASIC CYLINDER

Description — Generates a lattice cylinder.



INPUTS

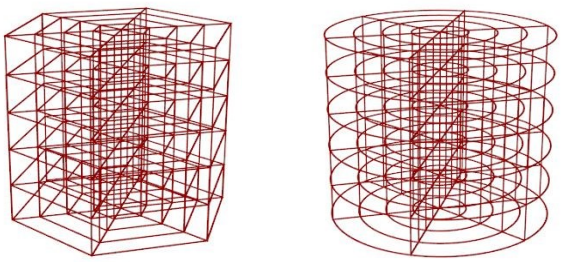
Topo	List of lines coming from the unit cell module.
R, H	Cylinder radius and height, respectively.
Nu, Nv, Nw	Number of unit cells in each of the UVW-map coordinates. (u—axial, v—theta, w—radial)
Morph	Strut morphing method (0—none, 1—Space morph, 2—Bezier morph)
MF	*Only used for Bezier morphing.* A contraction factor for the Bezier curve control points.

OUTPUTS

Struts	List of curves representing the lattice struts.
Nodes	Tree of points representing the lattice nodes.

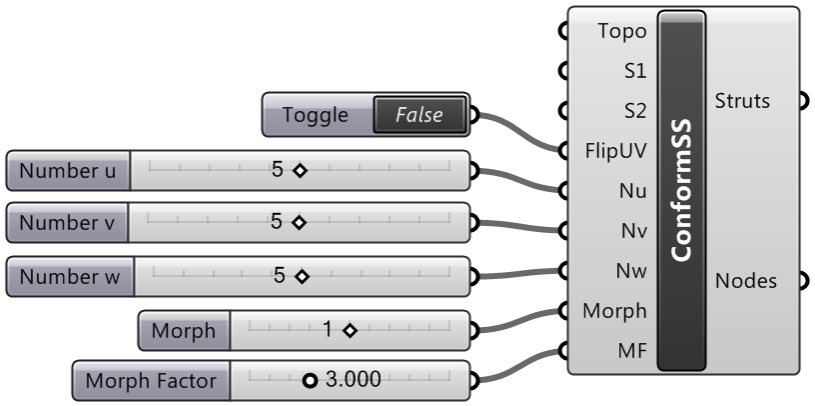
EXAMPLE

The example shown here illustrates **the effect of morphing**. While the nodes conform to the design space whether the structure is morphed or not, struts will not necessarily conform like we want them to. Using the Space morphing method discretizes the struts and maps an interpolated curve to the uvw-map. On the other hand, Bezier morphing computes derivatives for each node, then creates Bezier curves based on these derivatives.



3.2 CONFORM SURFACE-SURFACE

Description — Generates a conforming lattice between two surfaces.



INPUTS

Topo	List of lines coming from the unit cell module.
S1, S2	The surfaces that define the design space.
FlipUV	Sometimes required for surfaces with non-matching UV maps.
Nu, Nv, Nw	Number of unit cells in each of the UVW-map coordinates. (u—axial, v—theta, w—radial)
Morph	Strut morphing method (0—none, 1—space morph, 2—Bezier morph)
MF	*Only used for Bezier morphing.* A contraction factor for the Bezier curve control points.

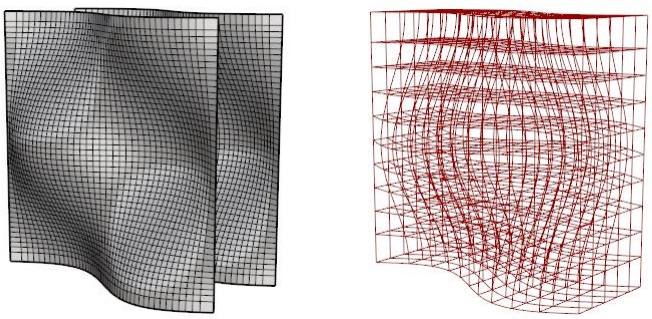
OUTPUTS

Struts	List of curves representing the lattice struts.
Nodes	Tree of points representing the lattice nodes.

EXAMPLE

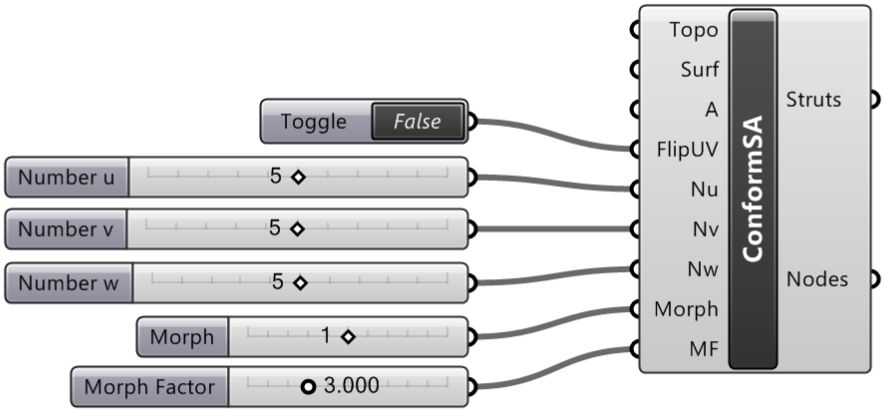
The lattice is conformed to the surfaces with the help of their UV maps, which are extended into a single UVW-map. By this token, you may input open or closed surfaces, such as spheres.

The example shows a pair of identical sinusoidal surfaces, and the imagined void between them is filled with lattice. Of course, your surfaces do not need to be identical.



3.3 CONFORM SURFACE-AXIS

Description — Generates a lattice cylinder.



INPUTS

Topo	List of lines coming from the unit cell module.
S1, A	The surface and axis that define the design space, respectively.
FlipUV	Sometimes required for surfaces with non-matching UV maps.
Nu, Nv, Nw	Number of unit cells in each of the UVW-map coordinates. (u—axial, v—theta, w—radial)
Morph	Strut morphing method (0—none, 1—space morph, 2—Bezier morph)
MF	*Only used for Bezier morphing.* A contraction factor for the Bezier curve control points.

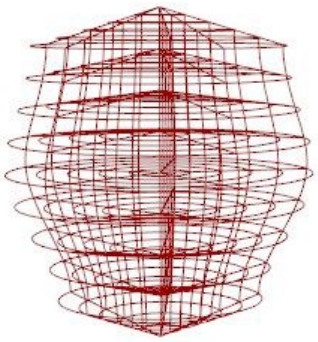
OUTPUTS

Struts	List of curves representing the lattice struts.
Nodes	Tree of points representing the lattice nodes.

EXAMPLE

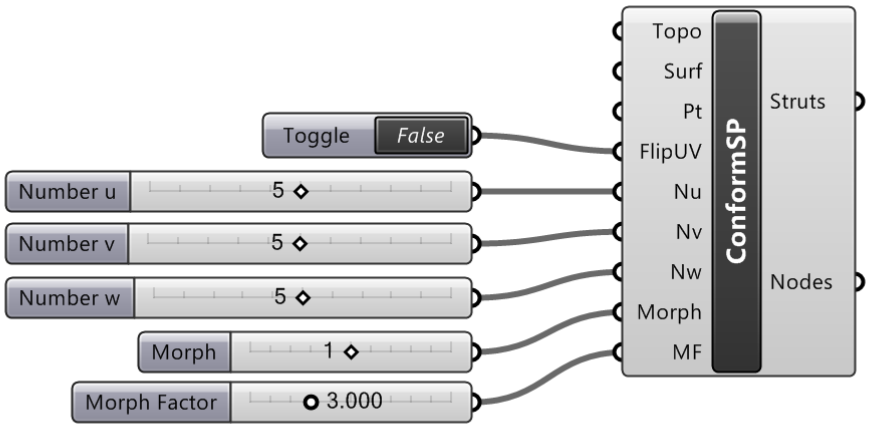
The lattice is conformed to the space between the axis and the surface by using the UV-map on the surface, and letting the axis be a U-map. The axis can be an open or closed curve. The surface can also be open or closed.

This example shows lattice being mapped to the space between a pseudo-cylindrical surface and an axis which is slightly longer than the surface.



3.4 CONFORM SURFACE-POINT

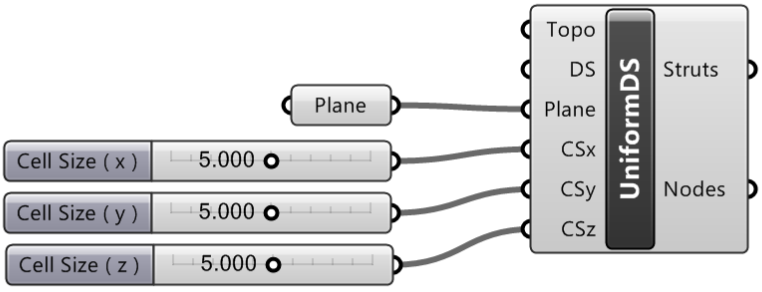
Description — Generates a lattice cylinder.



SOME EXAMPLES

3.5 UNIFORM DESIGN SPACE

Description — Generates a lattice cylinder.



Section 4

MESH MODULE

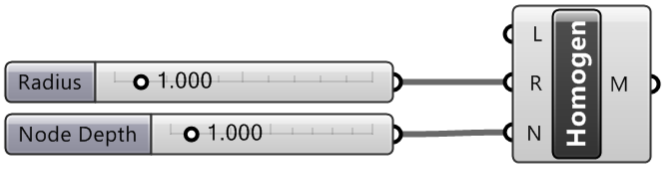
Generates solid mesh based on the lattice wireframe. The **Homogen** component generates a homogeneous mesh, meaning the strut radius is constant throughout the entire structure. The **HeterogenGradient** component generates a heterogeneous mesh, based on a set of predefined thickness gradients, but also lets the user define custom gradients mathematically. The **HeterogenCustom** component is used when the user wants to completely customize the thickness distribution.

The underlying mesh generation methods are based on Exoskeleton by David Stasiuk.

Note that they currently lack robustness

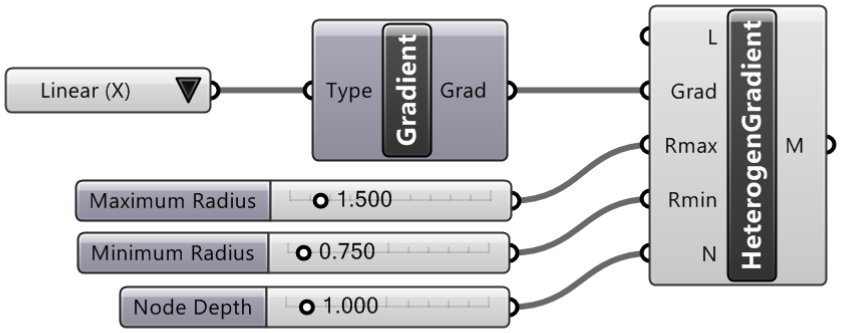
4.0 HOMOGENEOUS

Description — Generates a lattice cylinder.



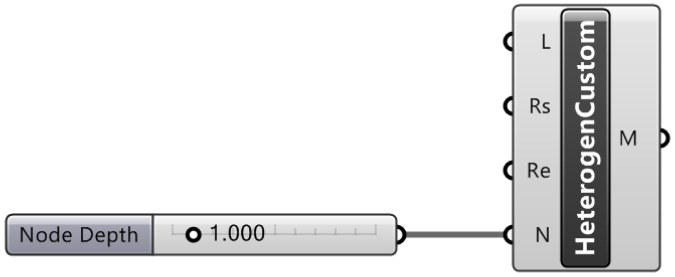
4.1 HETEROGENEOUS GRADIENT

Description — Generates a lattice cylinder.



4.2 HETEROGENEOUS CUSTOM

Description — Generates a lattice cylinder.



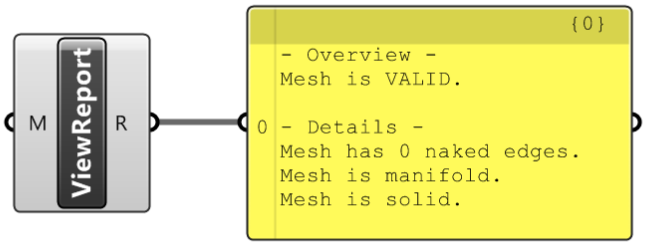
Section 5

UTILITY MODULE

Post-processing tools. The **MeshReport** component gives a comprehensive report of a mesh.
More coming soon...

5.0 VIEW REPORT

Description — Generates a lattice cylinder.



Section 6

CASE STUDIES

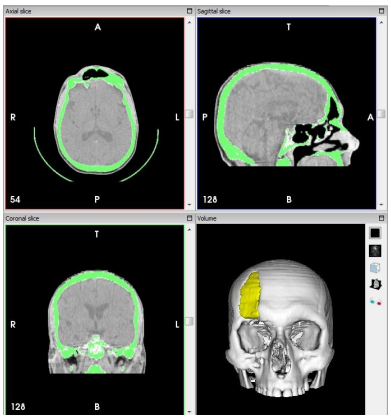
6.0 BONE GRAFTS

Bone tissue, unlike most other tissues, has the ability to regenerate when provided a space in which to grow. Bone grafting is a surgical procedure which aims to implant scaffold materials for reparative bone growth. The success of a bone graft relies on three biologic mechanisms: osteoconduction, osteoinduction and osteogenesis. Osteoconduction occurs when the graft serves as a scaffold upon which bone cells spread and form new bone. Naturally, the structural properties of this scaffold play an important role in this process.
Through the use of lattice structures, INTRA**LATTICE** allows us to manipulate the porosity and strength of the implant at various levels. First and foremost, defining the design space requires some form of medical imaging of the patient's fracture. Below, we will walkthrough the design process of a craniofacial bone graft implant.

I. CT Segmentation *(using InVesalius software)*

Based on a CT scan of the patients fractured skull, InVesalius reconstructed the set of medical images into a 3D representation. Then, by manually selecting pixels where cranial bone was missing, an implant space was defined (seen in yellow) and exported as a mesh (.stl).

For the record, this process was quite tedious and a more efficient approach is needed.



II. INTRA**LATTICE** Design Phase

The next step was to open the implant mesh in Rhino3D and select it as the design space in our Grasshopper algorithm. From here, experimenting with various lattice topologies and cell sizes allows us to manipulate the porosity and strength of the implant. In the example below, a grid-uniform-homogeneous lattice is generated:

I Fractured Skull → **II** Solid Implant → **III** Porous Implant

6.1 CELLULAR TIRE

Airless tires are of particular interest in offroad, military and extra-terrestrial vehicles. In general, the quality of such tires is largely based on resilience. However, since this property is hard to evaluate through FEA, extensive physical testing is required to validate the quality of a design. In this case study, we simply show prototypes which would be tested, the quality of these designs is unknown.

As usual, the first step was to define a design space, which in this case, was the space between the tire thread and the wheel rim.

Next, we set the various design parameters, such as unit cell type, conformal iteration values and strut radius, and the lattice is generated.

By changing the design parameters, we easily obtain very diverse structures. In the figure shown here, you can see, in order of appearance:

1. Bare design space
2. Grid conformal lattice
3. Octet conformal lattice
4. Vintiles conformal lattice