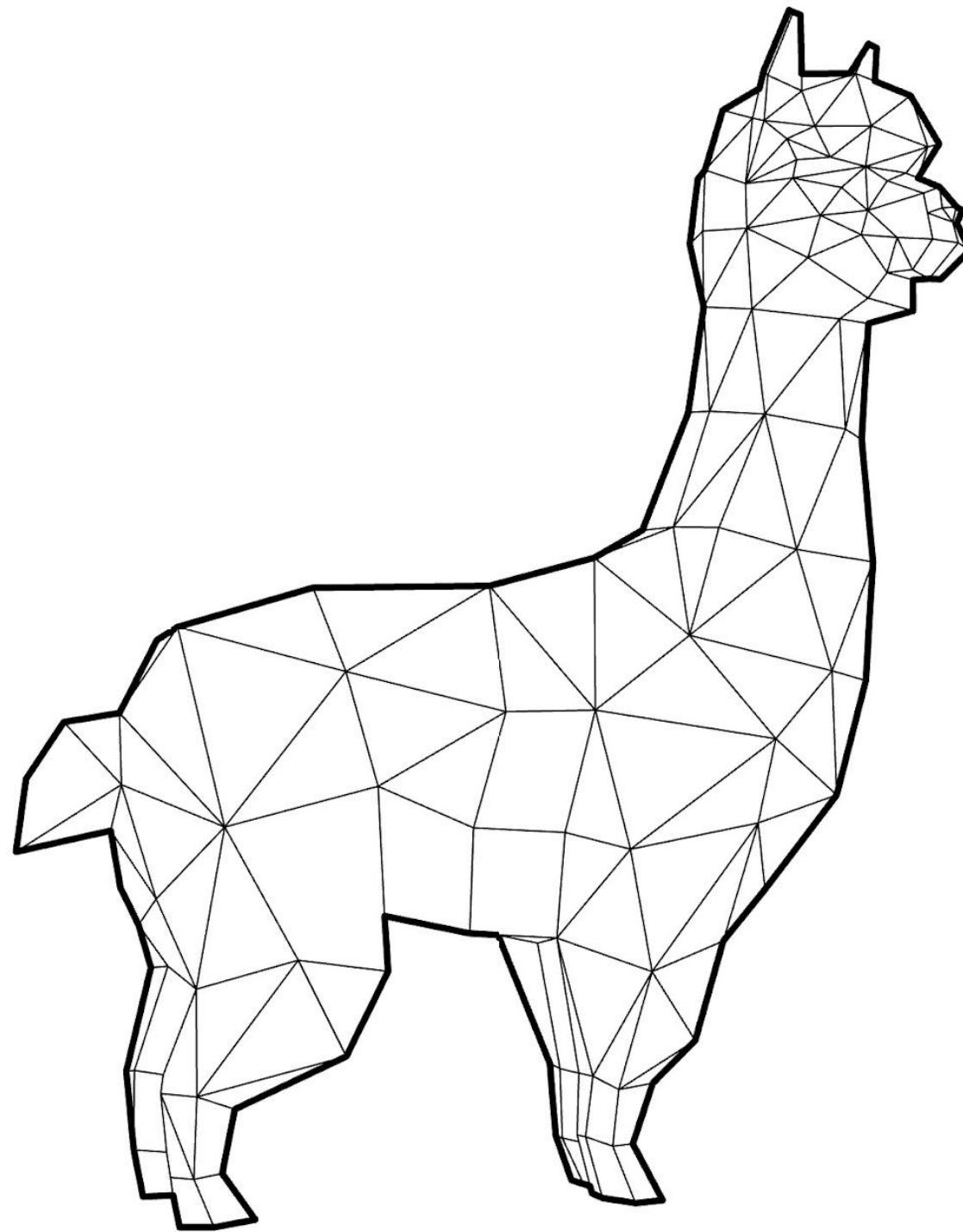
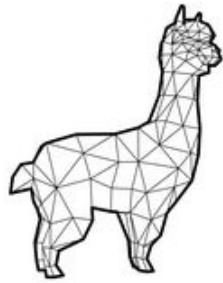


Alpaca4d



A revolutionary Parametric FEA software



Introduction

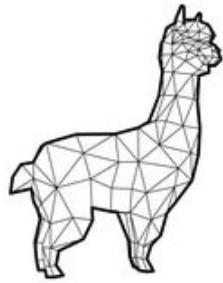
Alpaca4d is a Grasshopper plugin which has been developed on top of **OpenSees**.

The library is mostly used by researchers and academia because of the not user-friendly behaviour even if the math behind the library is highly sophisticated.

The main idea of **Alpaca4d** is to provide an efficient and easy way to use OpenSees without writing any line of code.

The belief is to bring more users to perform structural analyses with Opensees in a parametric environment such as Grasshopper.





Introduction

<https://opensees.berkeley.edu>

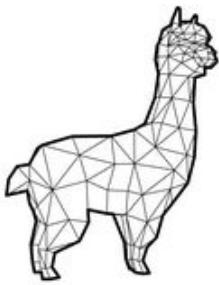
i.e. Catenary Cable

Command_Manual

This command is used to construct a catenary cable element object.

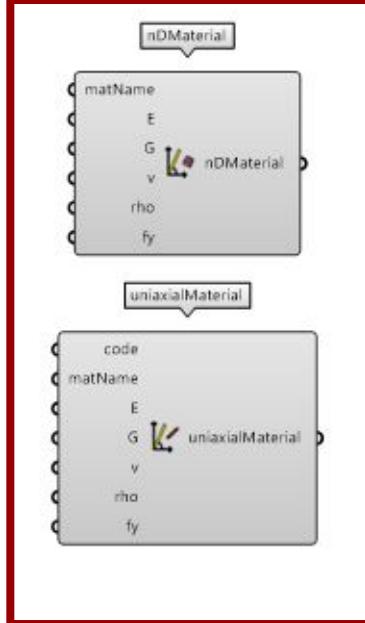
```
element CatenaryCable $tag $iNode $jNode $weight $E $A $L0 $alpha $temperature_change $rho $errorTol $Nsubsteps $massType
```

\$eleTag	unique element object tag
\$iNode \$jNode	end nodes (3 dof per node)
\$E	elastic modulus of the cable material
\$A	cross-sectional area of element
\$L0	unstretched length of the cable
\$alpha	coefficient of thermal expansion
\$temperature_change	temperature change for the element
\$rho	mass per unit length
\$errortol	allowed tolerance for within-element equilibrium (Newton-Raphson iterations)
\$Nsubsteps	number of within-element substeps into which equilibrium iterations are subdivided (not number of steps to convergence)
\$massType	Mass matrix model to use (\$massType = 0 lumped mass matrix, \$massType = 1 rigid-body mass matrix (in development))

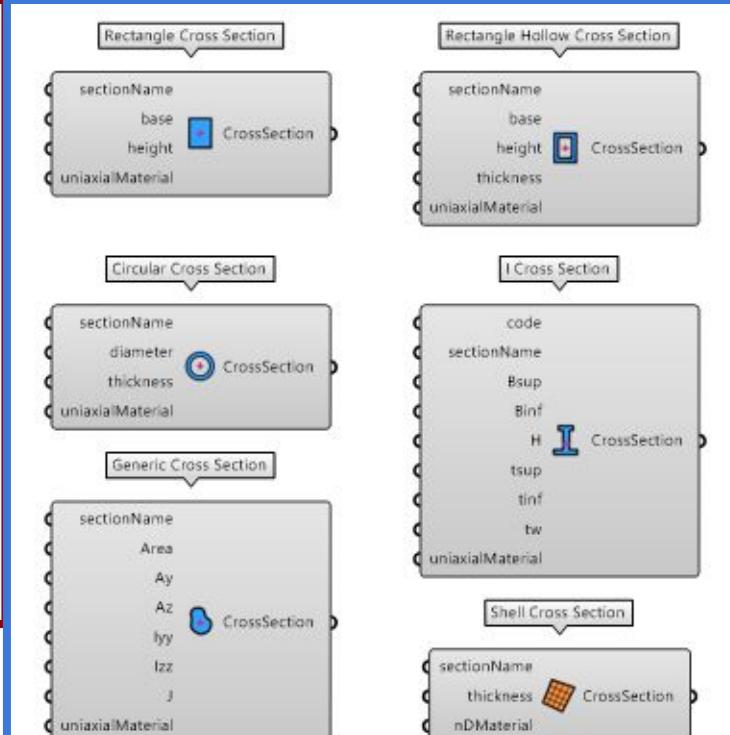


Plug-in

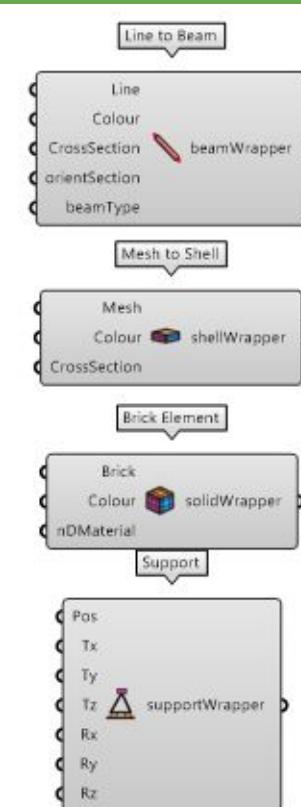
Materials



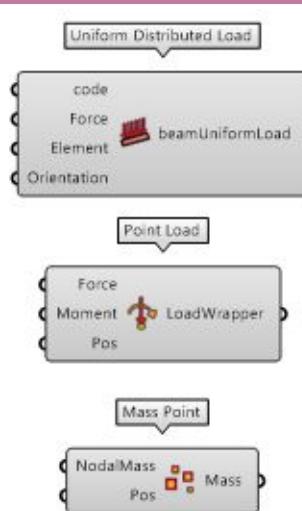
Cross Sections



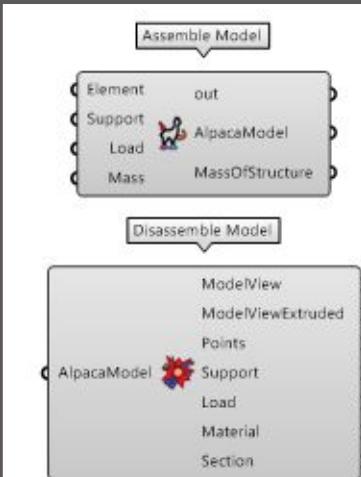
Elements



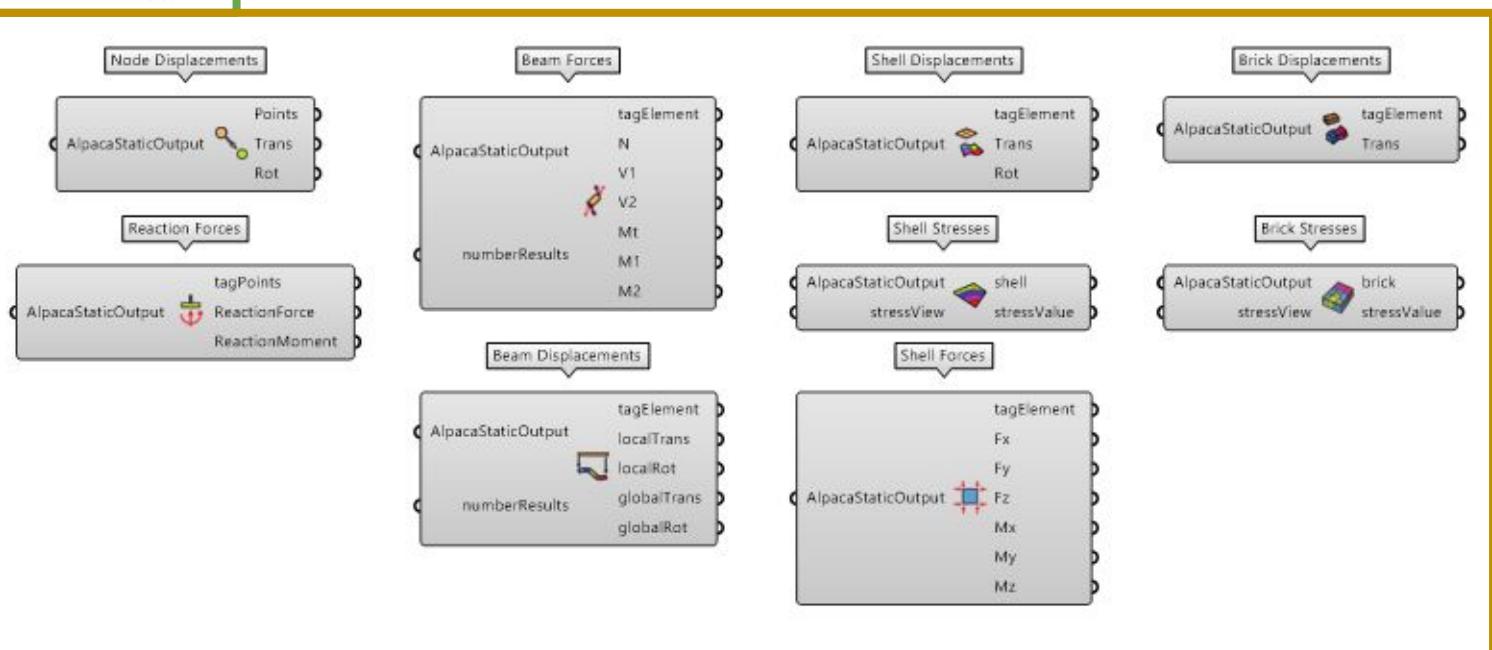
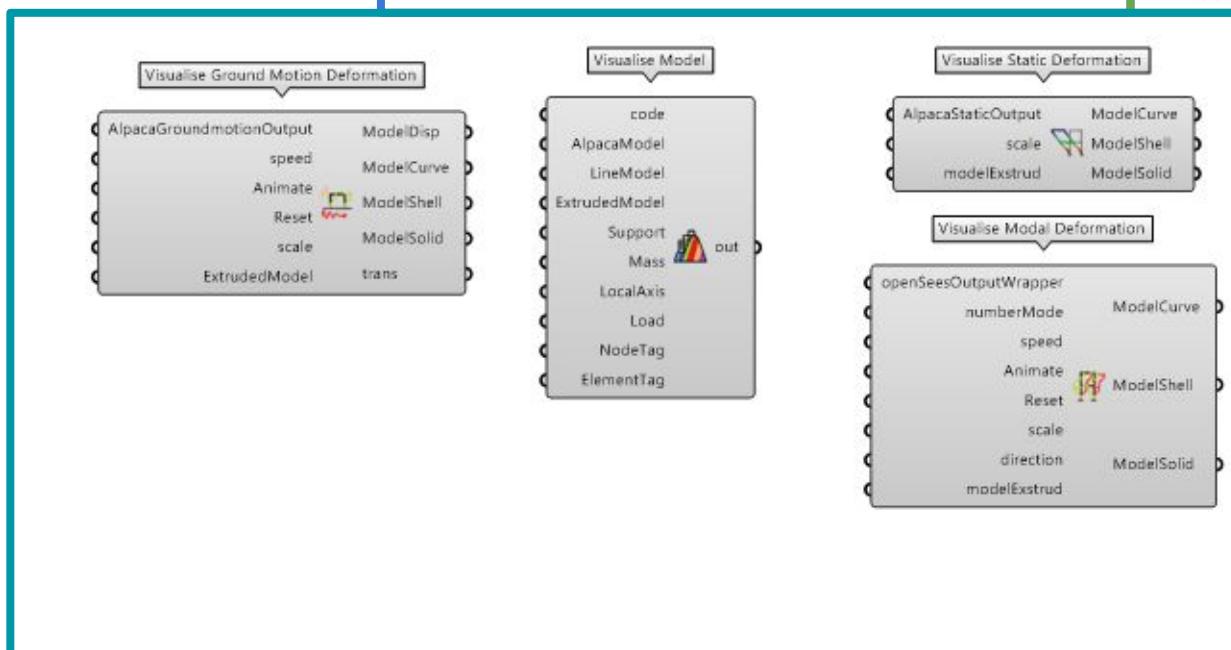
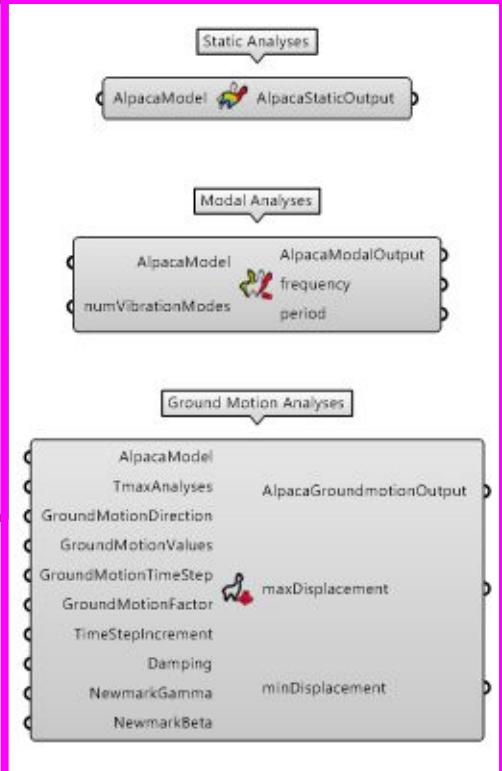
Loads

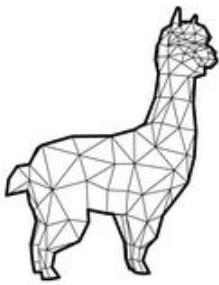


Assemble/ Disassemble



Analysis

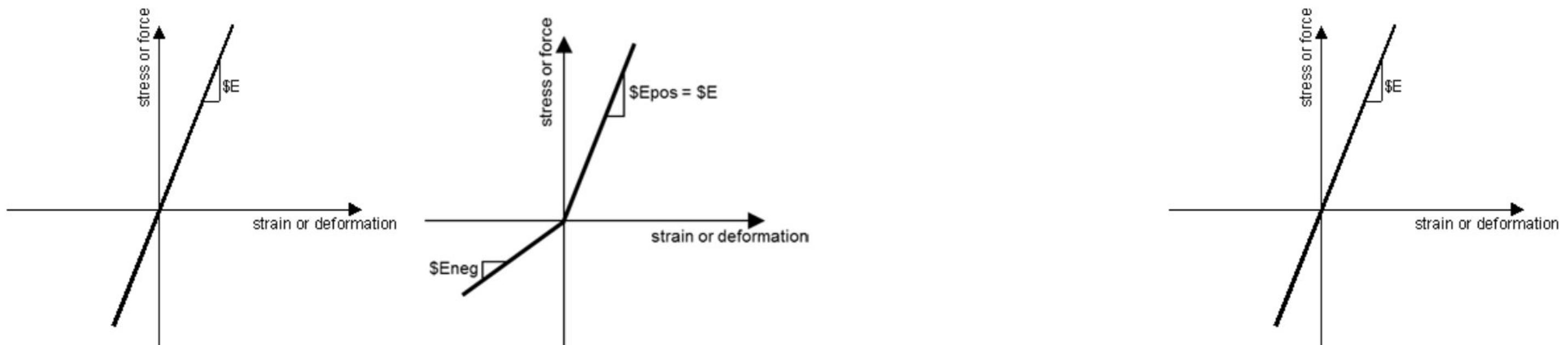
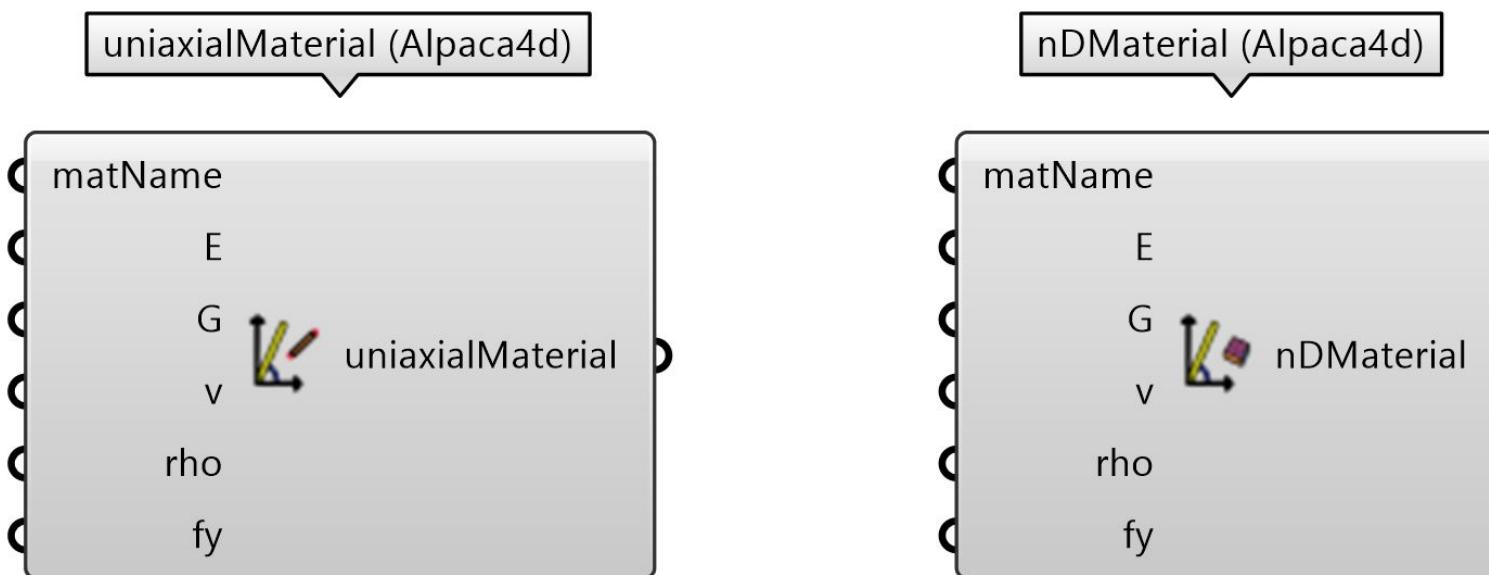


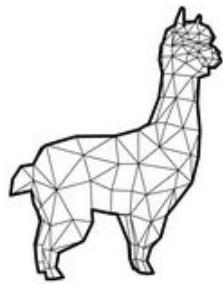


Materials

UniaxialMaterial is used to construct a material object which represents uniaxial stress-strain relationships. The implemented relationship is Linear Elastic.

NDMaterial is used to construct a material object which represents the stress-strain relationship at the gauss-point of a continuum element. The behaviour is an Elastic Isotropic.



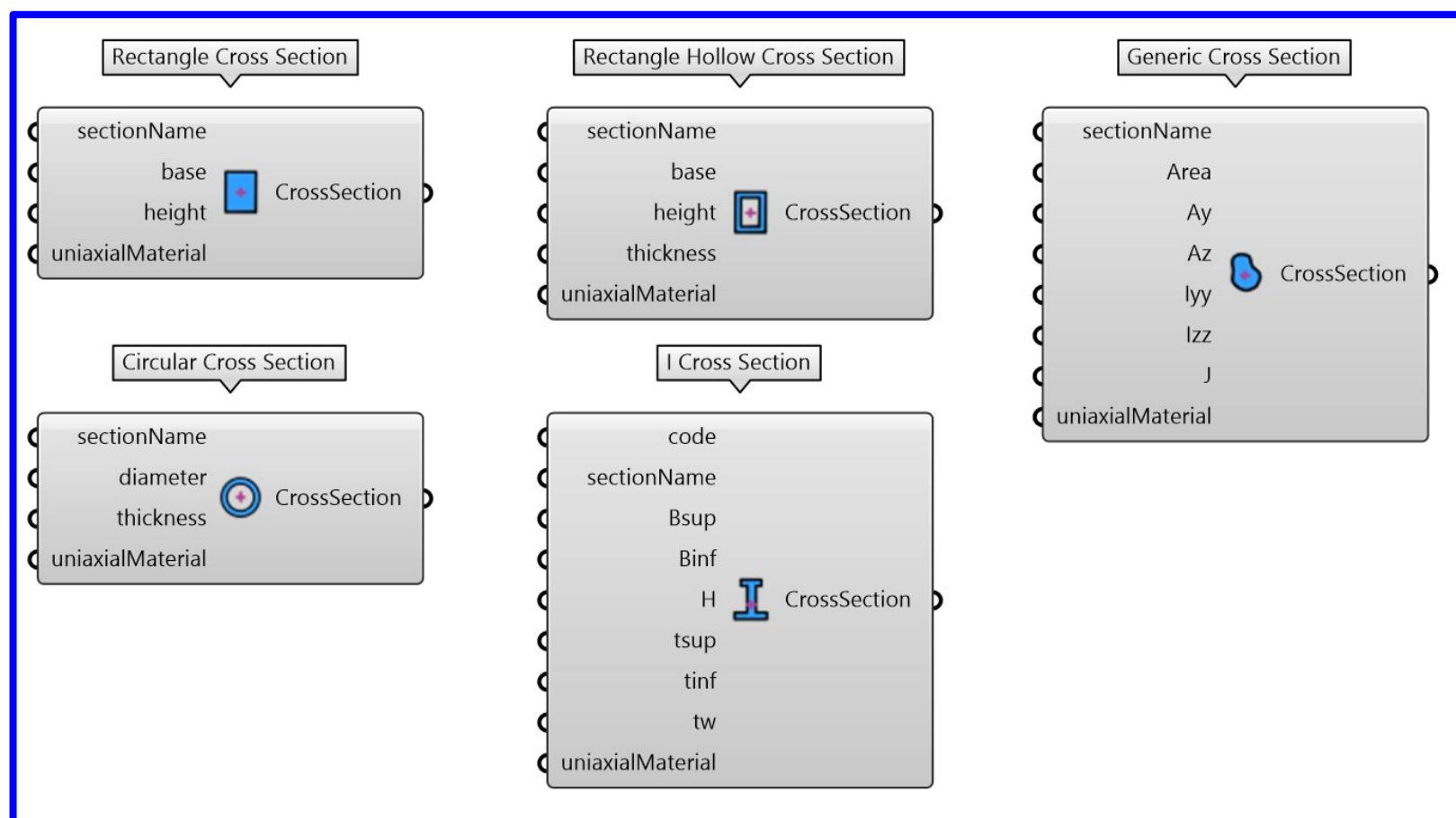


Cross Sections

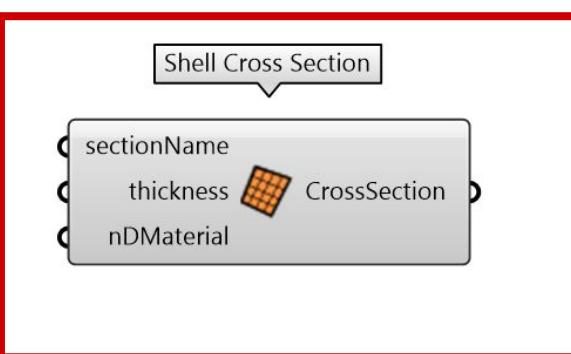
Solid rectangle section, Circular Hollow Section, Rectangular Hollow Section, I-Section and Generic Cross Section are provided to the users to assign mechanical properties to a beam element.

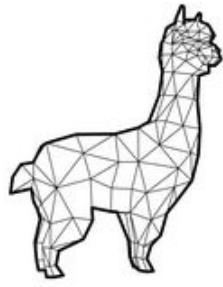
Constant thickness section has been implemented for shell elements.

Beam Cross Sections



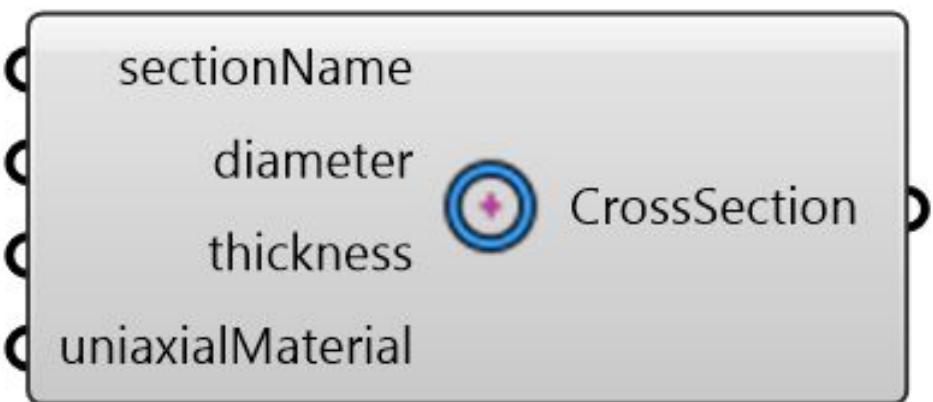
Shell Cross Section





Cross Sections

Circular Cross Section

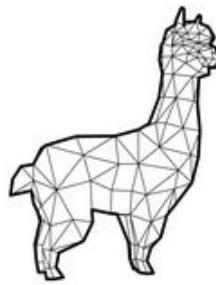


```
import math
import Grasshopper as gh

def CircleCrossSection(sectionName, diameter, thickness, uniaxialMaterial):
    sectionName = sectionName
    shape = "circular"
    diameter = diameter / 1000           # Input value in mm ----> Output m
    thickness = thickness / 1000          # Input value in mm ----> Output m
    if thickness == 0 or thickness == diameter / 2:
        Area = math.pow(diameter,2)/4 * math.pi
        Ay = Area * 0.9
        Az = Area * 0.9
        Iyy = math.pow(diameter,4)/64 * math.pi
        Izz = Iyy
        J = pow(diameter,4)/32 * math.pi
    elif thickness < diameter/2 and thickness >= 0:
        Area = (math.pow(diameter,2)-math.pow((diameter-2*thickness),2))/4 * math.pi
        Ay = Area * 0.9
        Az = Area * 0.9
        Iyy = (math.pow(diameter,4)-math.pow((diameter-2*thickness),4))/64 * math.pi
        Izz = Iyy
        J = (math.pow(diameter,4)-math.pow((diameter-2*thickness),4))/32 * math.pi
    else:
        msg = "Incorrect values. Thickness has to be greater than D/2 and greater than 0"
        ghenv.Component.AddRuntimeMessage(gh.Kernel.GH_RuntimeMessageLevel.Error, msg)

    material = uniaxialMaterial

    return [[ Area, Ay, Az, Iyy, Izz, J, material, [shape, diameter, thickness]], sectionName ]
```



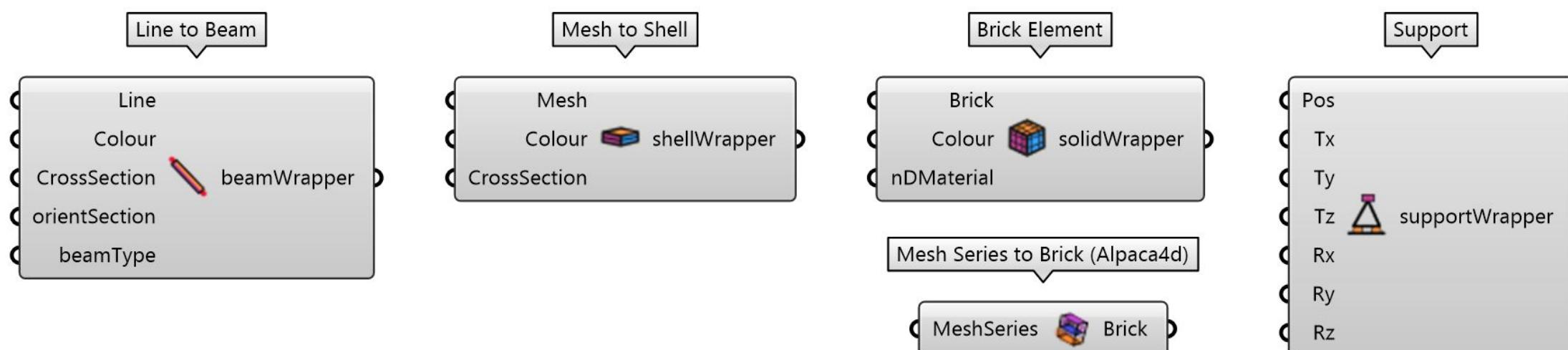
Elements

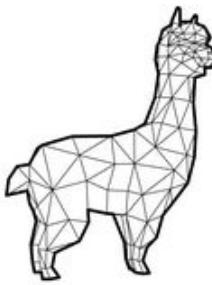
The finite elements implemented are **Elastic Timoshenko Beam Column Element**, **ShellMITC4-ShellDKGT**, **bbarBrick**.

Line to Beam component converts a line geometry to a *Timoshenko* Beam.

Mesh to Shell component converts a mesh geometry to a **Shell Element**. Shell elements are used to model structures in which one dimension, the thickness, is significantly smaller than the other dimensions. Triangular faces will be converted to a **ShellDKGT** formulation and the Quad faces to a **ShellMITC4** formulation.

Brick Element component converts an hexahedral element into a **bbarBrick**. A hexahedron is any polyhedron with six faces. It is generally a difficult task to convert a generic solid into a set of hexahedron and **MeshSeriesToBrick** component might help to construct some simple polyhedron.





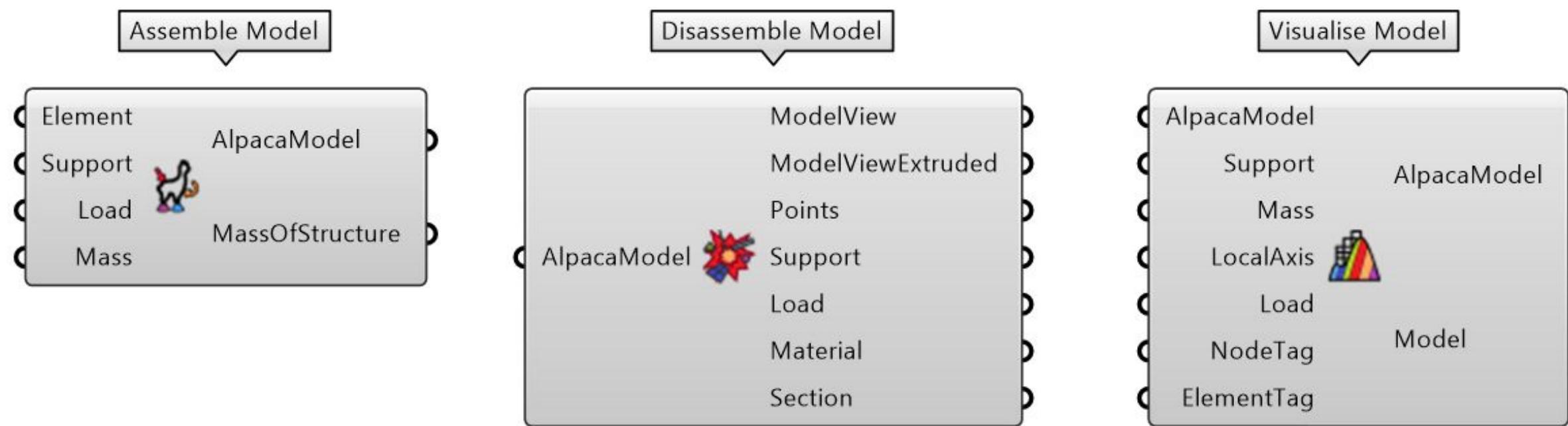
Assemble

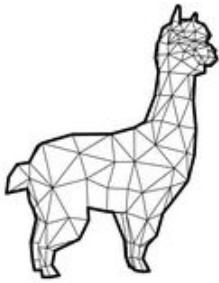
Element and Support are the minimum input that the user should provide to perform an analysis.

Assemble Model builds a text file with the necessary information to be sent to OpenSees solver. The mass of the structure is automatically calculated from the assemble component.

Disassemble Model retrieves some text information to double check that the model has been assembled correctly.

Visualise Model is a graphical representation of the structure. The model output is either an extruded model or lines model.





Assemble

```
"""Generate a text file model to be sent to OpenSees
Inputs:
    Element: Structural element.
    Support: Support element.
    Load: Load element.
    Mass: Mass point.
Output:
    AlpacaModel: Assembled Alpaca model.
    MassOfStructure: Total mass of the structure [kg].
"""

import sys
import clr
import os

import Rhino.Geometry as rg
import Grasshopper as gh

def Assemble(Element, Support, Load, Mass):
    points = []
    startPointList = []
    endPointList = []
    geomTransf = []
    matWrapper = []
    secTagWrapper = []

    for element in Element:
        #print(element[0])
        if (element[1] == "ElasticTimoshenkoBeam") or (element[1] == "Truss"): # element[1] retrieve the type of the beam
            startPoint = element[0].PointAt(element[0].Domain[0])
            endPoint = element[0].PointAt(element[0].Domain[1])
            points.append(startPoint)
            points.append(endPoint)
            geomTransf.append(element[0])
            matWrapper.append([element[2][0], element[2][0][1:]]) # to be careful because we are assigning "uniaxial" inside the solver. We need to find a clever way to assigning outside
        elif (element[1] == "ShellMITC4") or (element[1] == "shellDNGT"):
            mesh = element[0]
            vertices = mesh.Vertices
            for i in range(vertices.Count):
                points.append(rg.Point3d.FromPoint3f(vertices[i]))
            matWrapper.append([element[2][0], element[2][0][1:]])
            secTagWrapper.append([element[2][0], element[2][0][1:]])
        elif (element[1] == "bboxBrick") or (element[1] == "FourNodeTetrahedron"):
            mesh = element[0]
            vertices = mesh.Vertices
            for i in range(vertices.Count):
                #points.append( rg.Point3d.FromPoint3f(vertices[i]) )
                points.append( rg.Point3d( vertices[i] ) )
            matWrapper.append([element[2][0], element[2][0][1:]]))

    # create MatTag
    # use dictionary to delete duplicate
    matNameDict = dict(matWrapper)

    matNameList = []
    i = 1

    for key, value in matNameDict.items():
        temp = [key,i,value]
        matNameList.append(temp)
        i += 1

    openSeesMatTag = []
    for item in matNameList:
        openSeesMatTag.append([ item[0], item[1:] ])

    openSeesMatTag = openSeesMatTag
    matNameDict = dict(openSeesMatTag)

    # create SecTag
    # use dictionary to delete duplicate
    secTagDict = dict(secTagWrapper)
    secTagList = []
    i = 1

    for key, value in secTagDict.items():
        temp = [key,i,value]
        secTagList.append(temp)
        i += 1

    openSeesSecTag = []
    for item in secTagList:
        openSeesSecTag.append([ item[0], item[1:] ])

    openSeesSecTag = openSeesSecTag
    secTagDict = dict(openSeesSecTag)

    #print(secTagDict)

    # create GeomTransf
    geomTransf = [row[0] for row in geomTransf]
    geomTransfList = list(dict.fromkeys(geomTransf))
    geomTransfDict = { geomTransfList[i] : i+1 for i in range(len(geomTransfList)) }

    geomTag = geomTransfDict.values() # elemento a dx (tag)
    geomVec = geomTransfDict.keys() # elemento a sx (vettore)

    GeomTransf = []
    for i in range(0, len(geomTag)):
        GeomTransf.append( [ geomTag[i], list(rg.Vector3d(geomVec[i])) ] )

    GeomTransf = GeomTransf

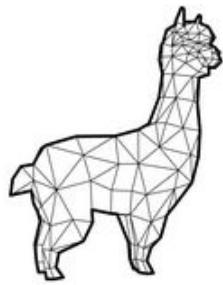
    oPoints = rg.Point3d.CullDuplicates(points, 0.01) # Collection of all the points of our geometry
    cloudPoints = rg.PointCloud(oPoints) # Convert to PointCloud to use ClosestPoint Method

    ## FOR NODE ##
    openSeesNode = []

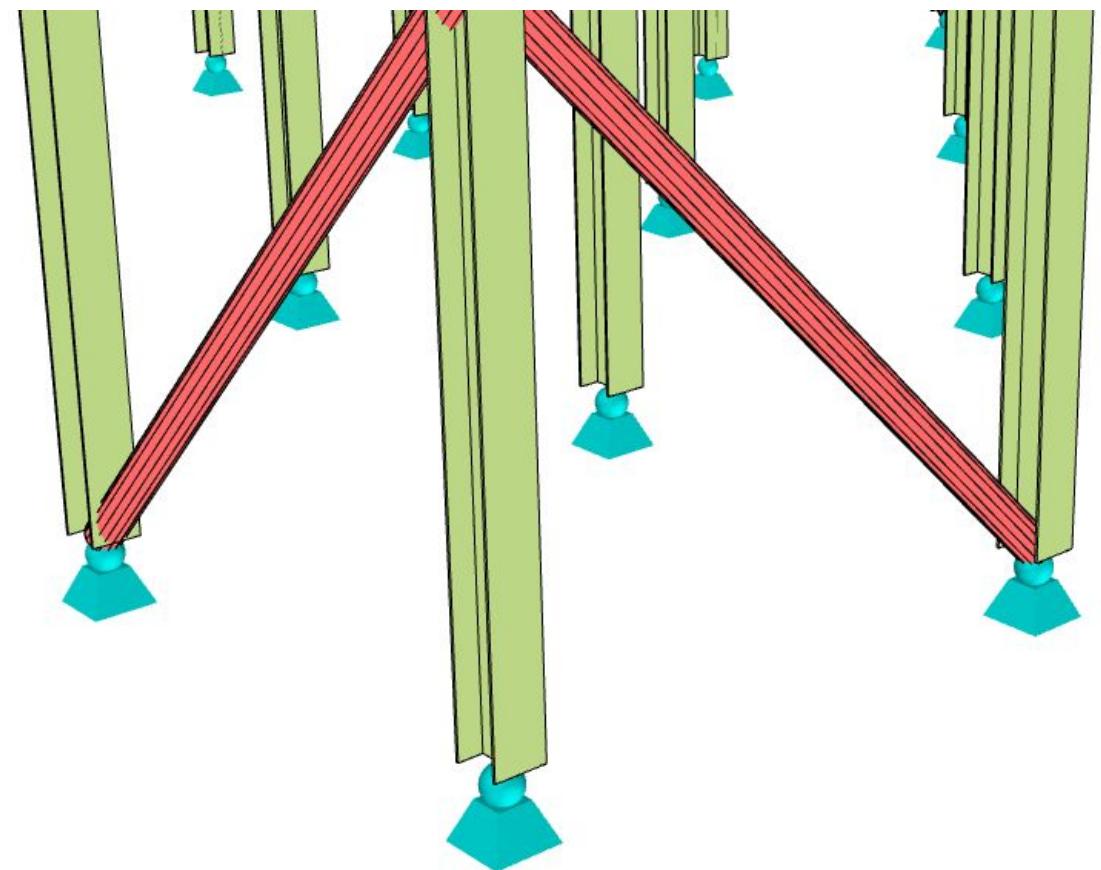
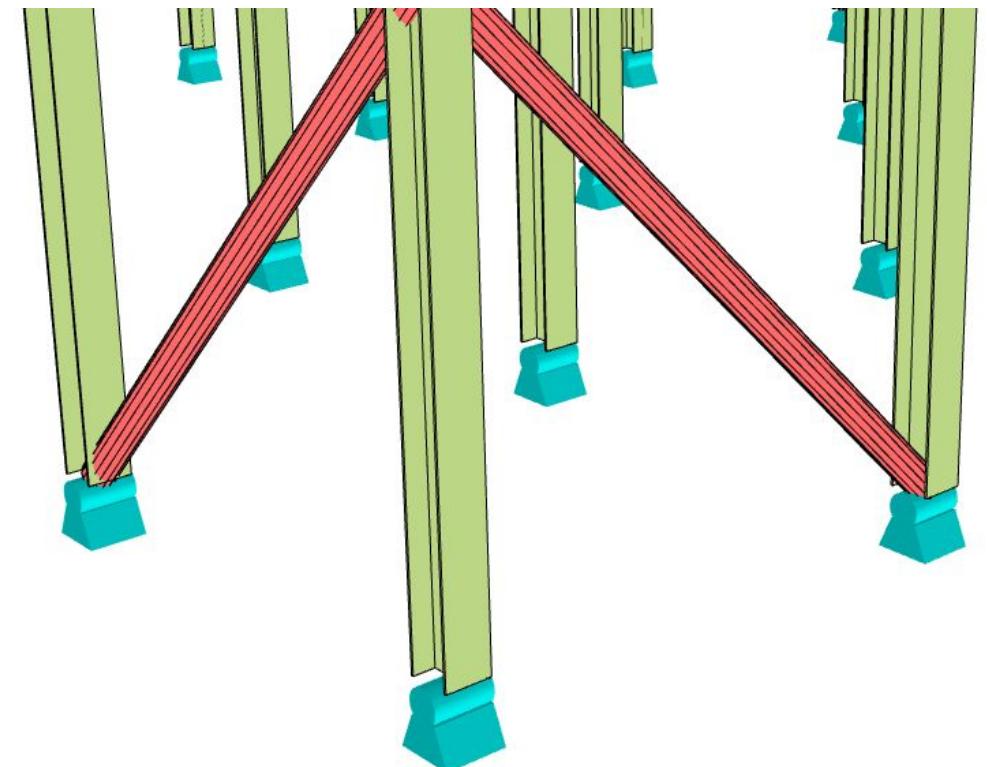
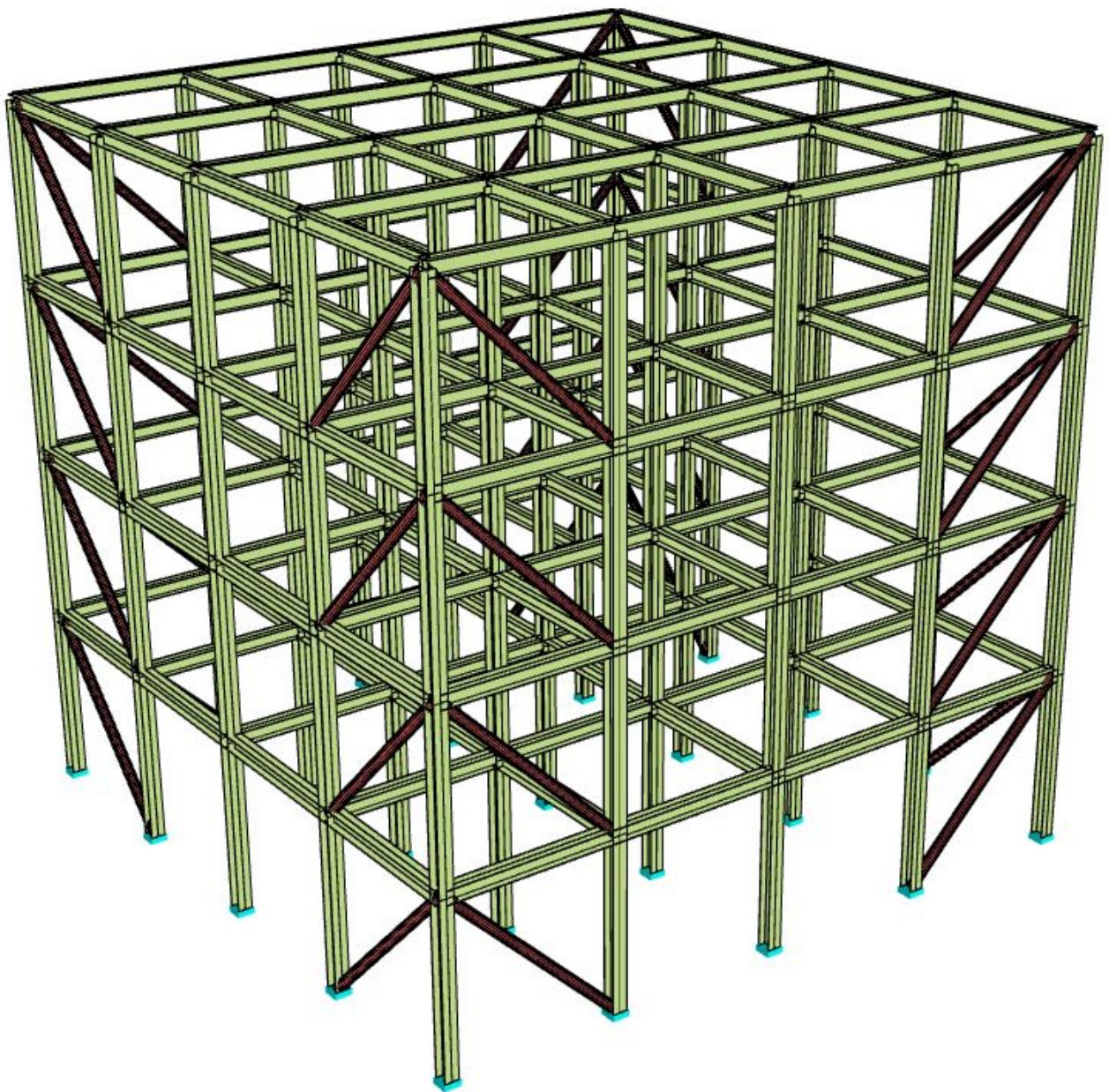
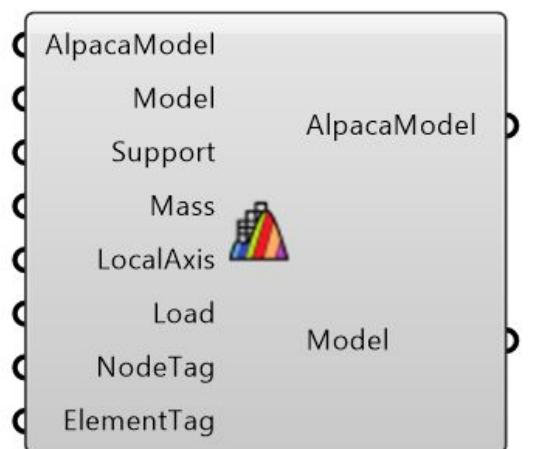
    for nodeTag, node in enumerate(oPoints):
        openSeesNode.append( [nodeTag, node.X, node.Y, node.Z] )

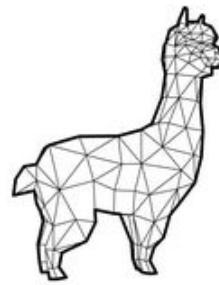
    openSeesNode = openSeesNode

    ## FOR ELEMENT ##
    openSeesBeam = []
    openSeesShell = []
    openSeesSolid = []
    MassOfStructure = 0
    for eleTag, element in enumerate(Element):
        if (element[1] == "ElasticTimoshenkoBeam") or (element[1] == "Truss"): # element[1] retrieve the type of the beam
```



Visualise



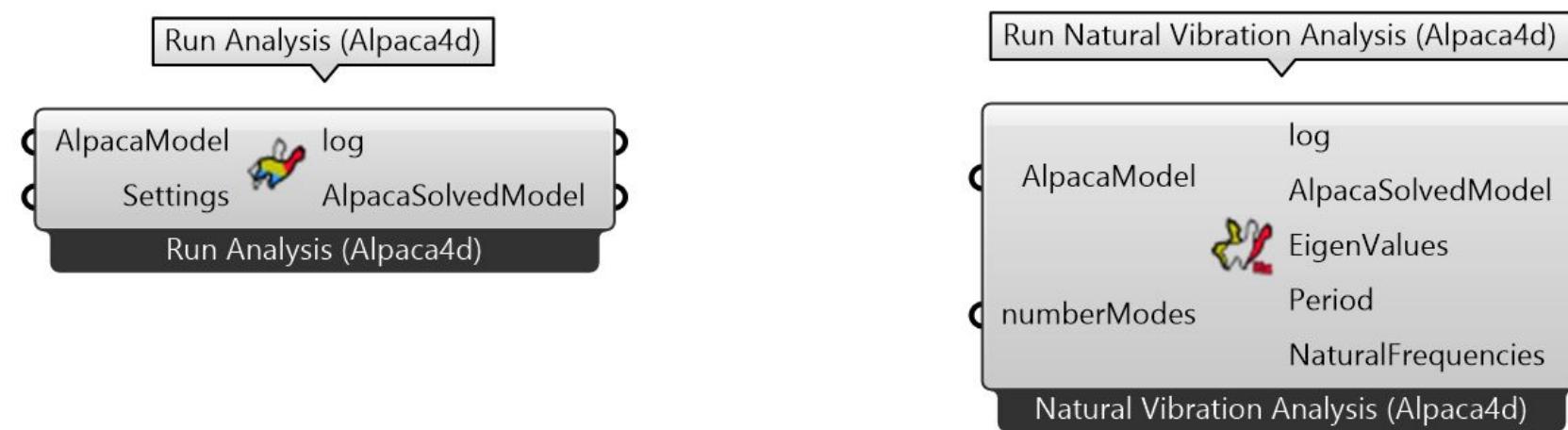


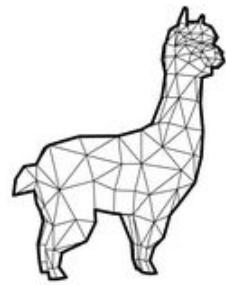
Analysis

Alpaca 4d can perform Static Analysis, Modal analysis, Ground Motion Analysis and 3NDF Analysis (WIP-Brick Elements).

Static Analysis computes the response of the structure.

Modal Analysis computes the Modal shapes of the structure and returns the period and frequency of that mode.

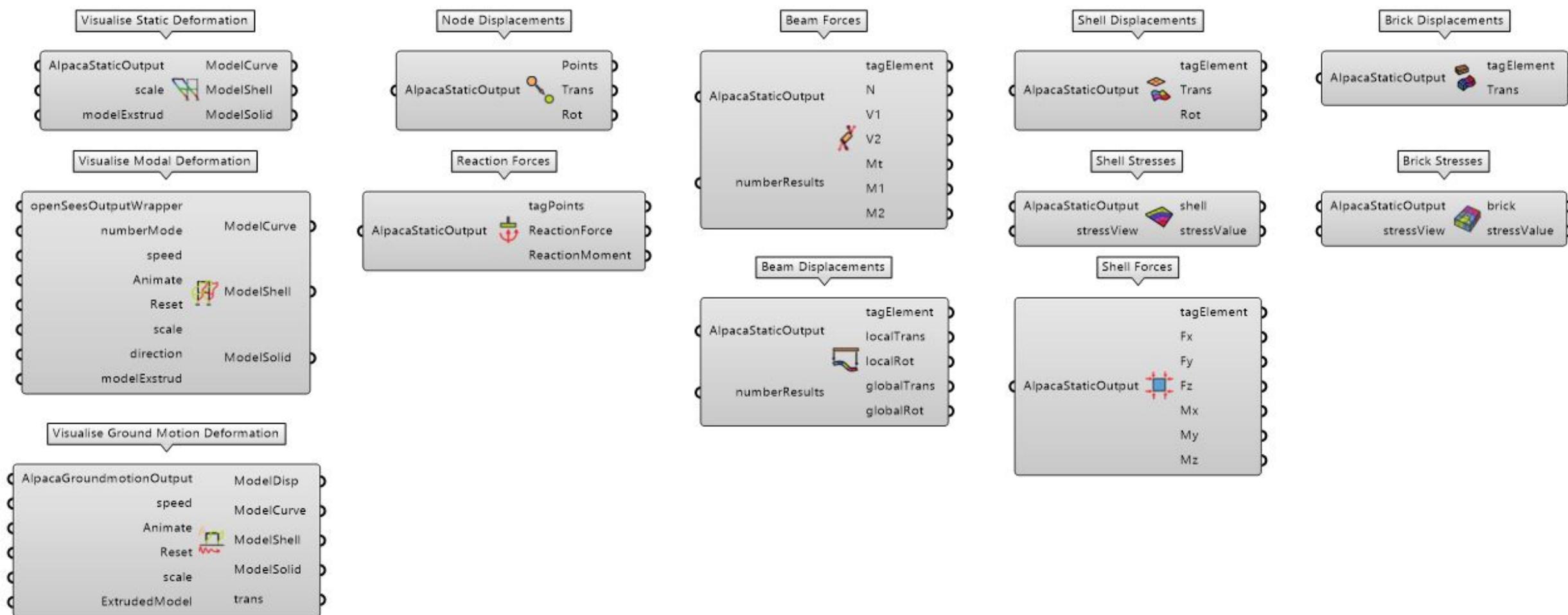


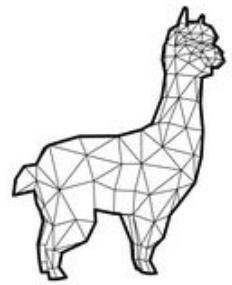


Numerical Computation and Visualization

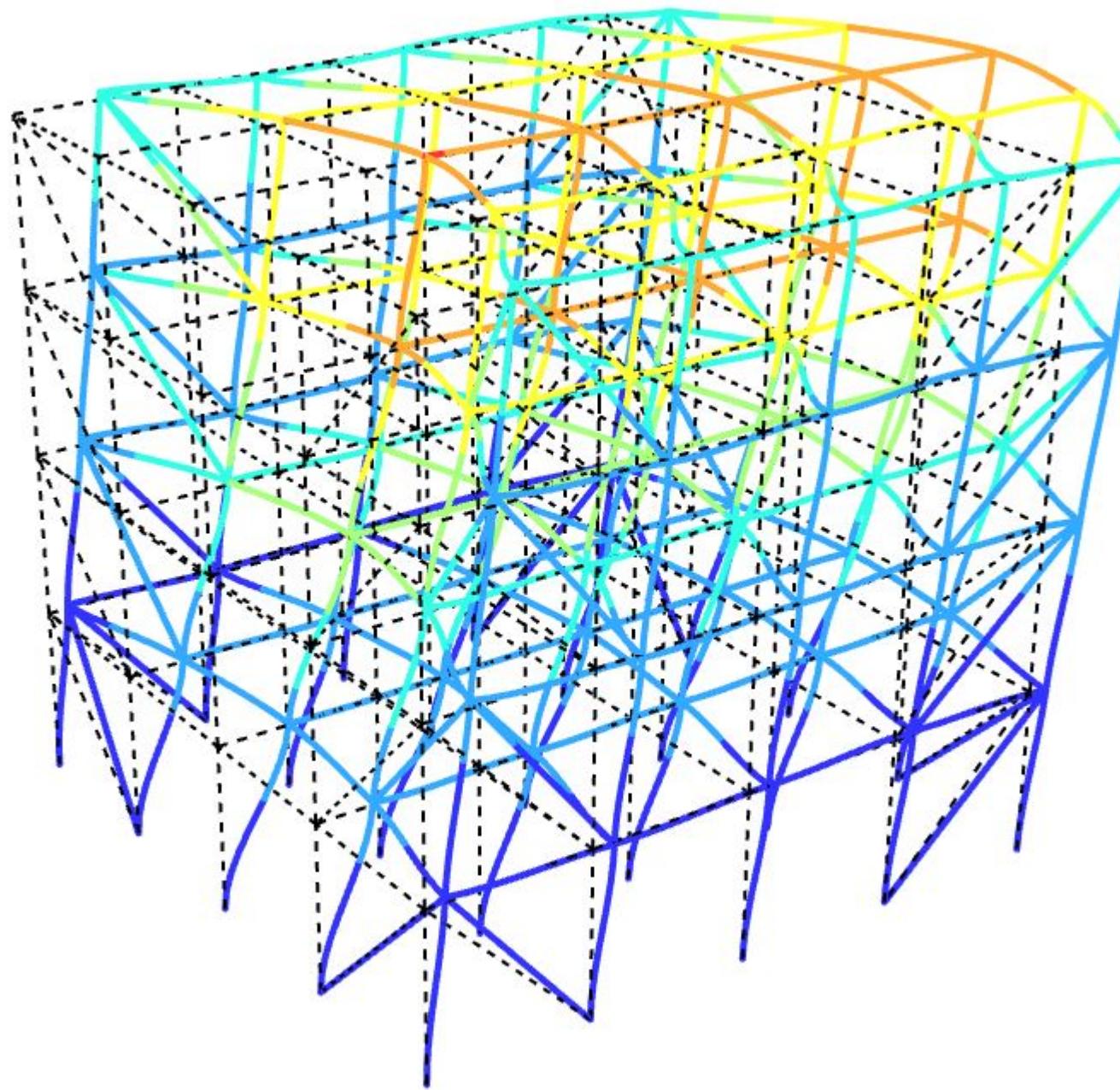
The static analysis returns the numerical outputs to study the behaviour of the structure. It is possible to query results for Point, Beam, Shell and Brick elements.

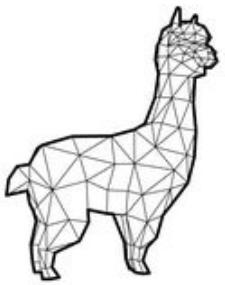
The visualization tool has been implemented for all types of analyses.



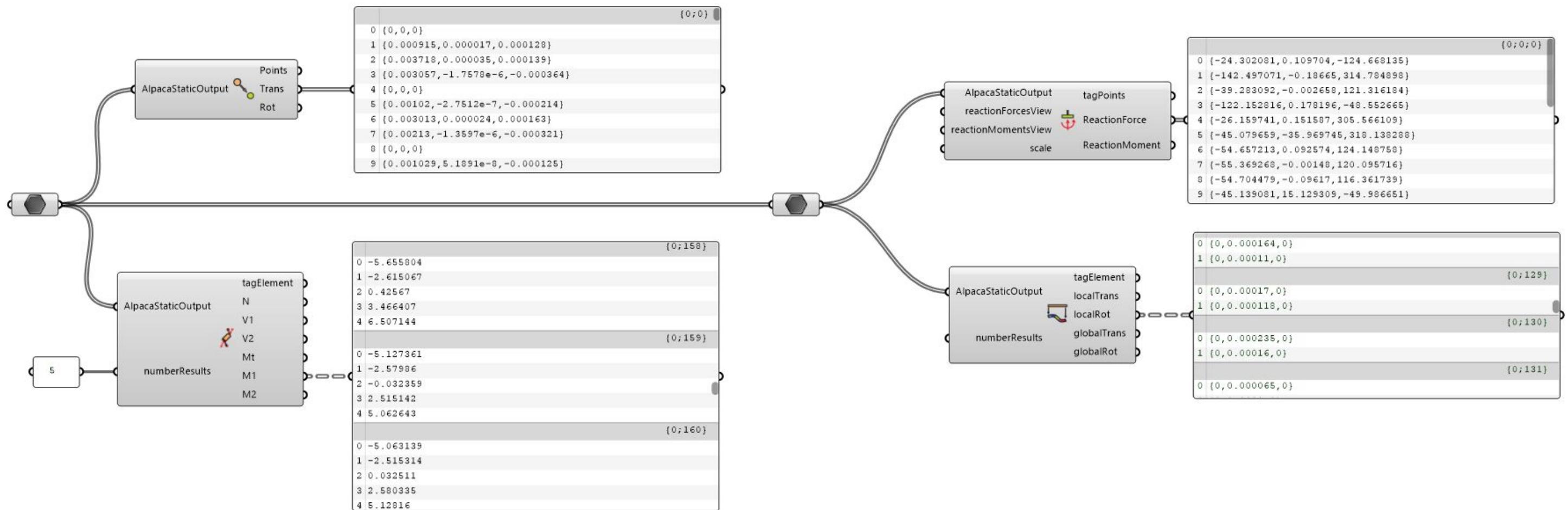


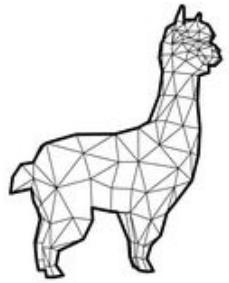
Numerical Computation and Visualization





Numerical Computation and Visualization





Pro

Open-Source code in C++

Complex Analysis:

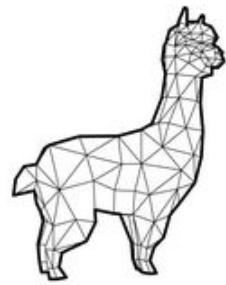
- Linear
- Non linear
- Static
- Dynamic

In the process of validation

User in Academia (i.e. UCL)

Cons

Slow (for now)



Computing Time

Number of points: 450
Number of floor: 10
Computing time: 0.36 s



Number of points: 900
Number of floor: 20
Computing time: 0.55 s

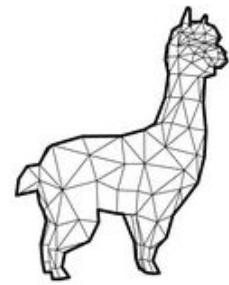


Number of points: 1800
Number of floor: 40
Computing time: 1.08 s

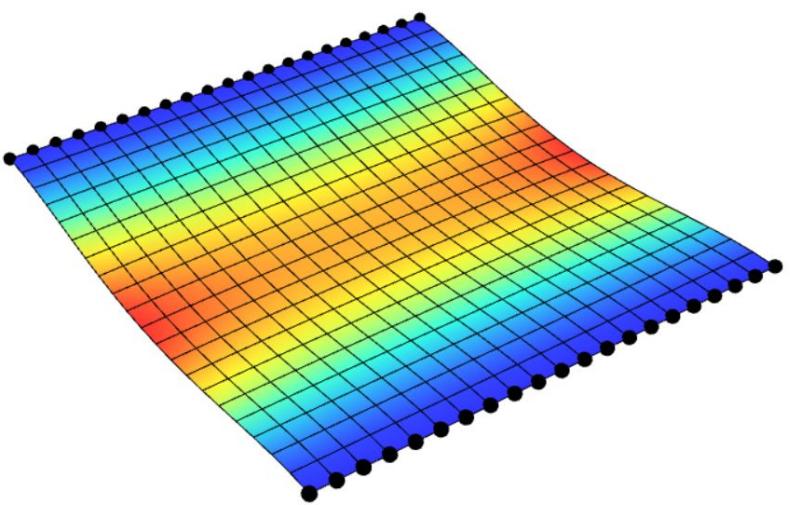


Number of points: 3600
Number of floor: 80
Computing time: 1.95 s

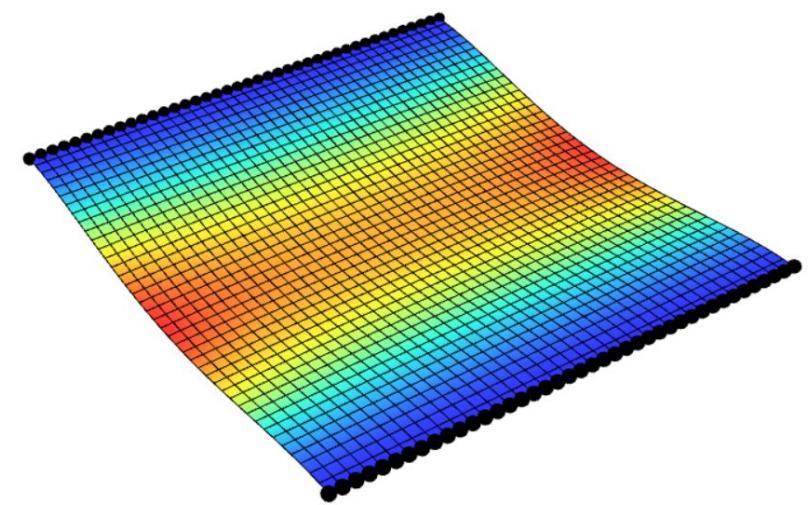




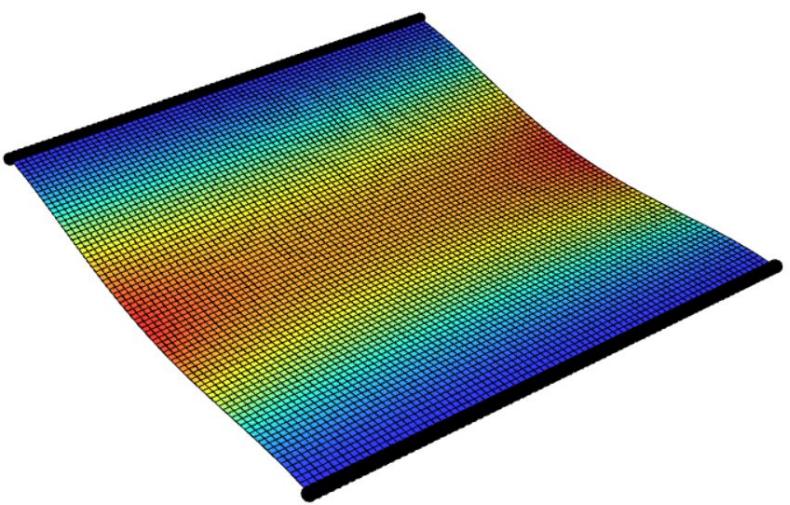
Computing Time



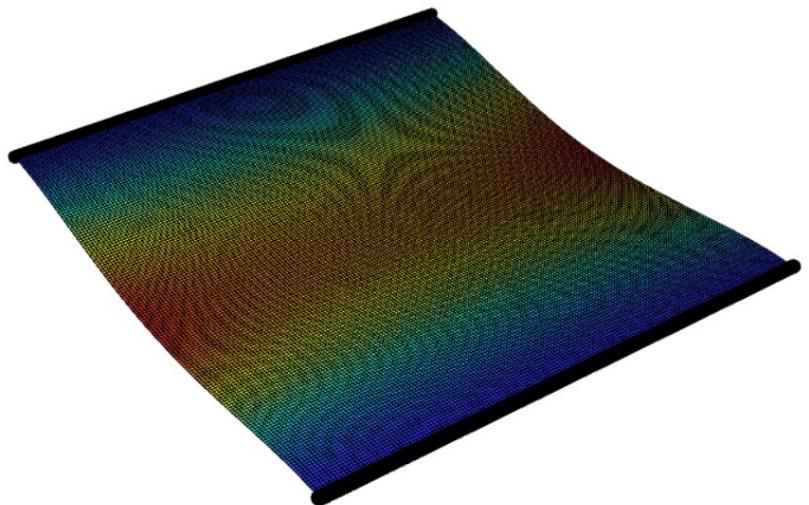
Number of points: 441
Computing time: 0.35 s



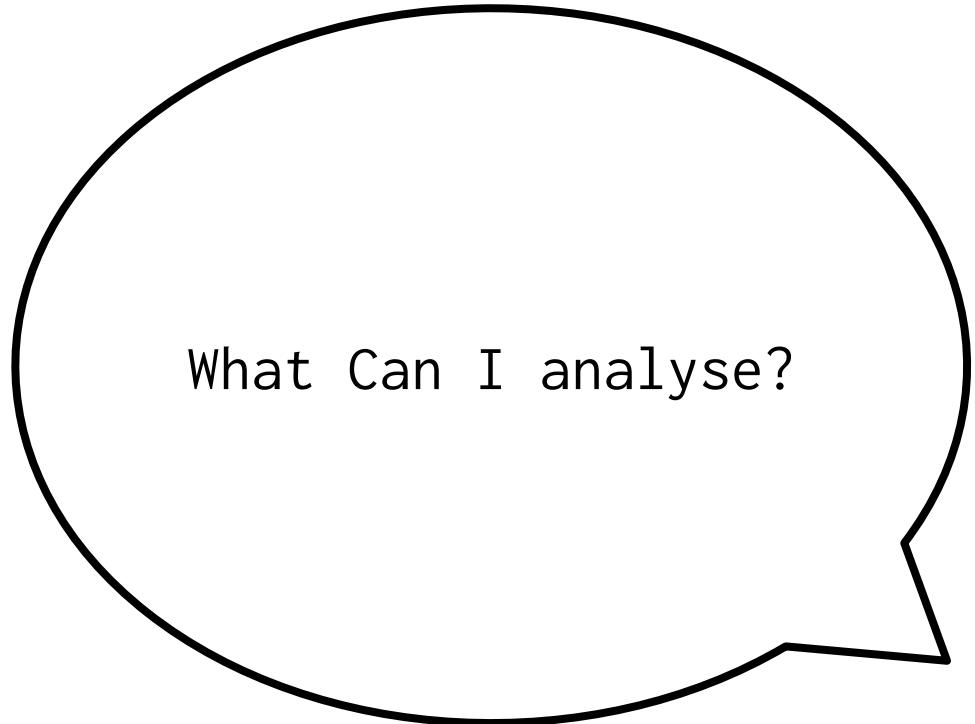
Number of points: 1681
Computing time: 1.3 s



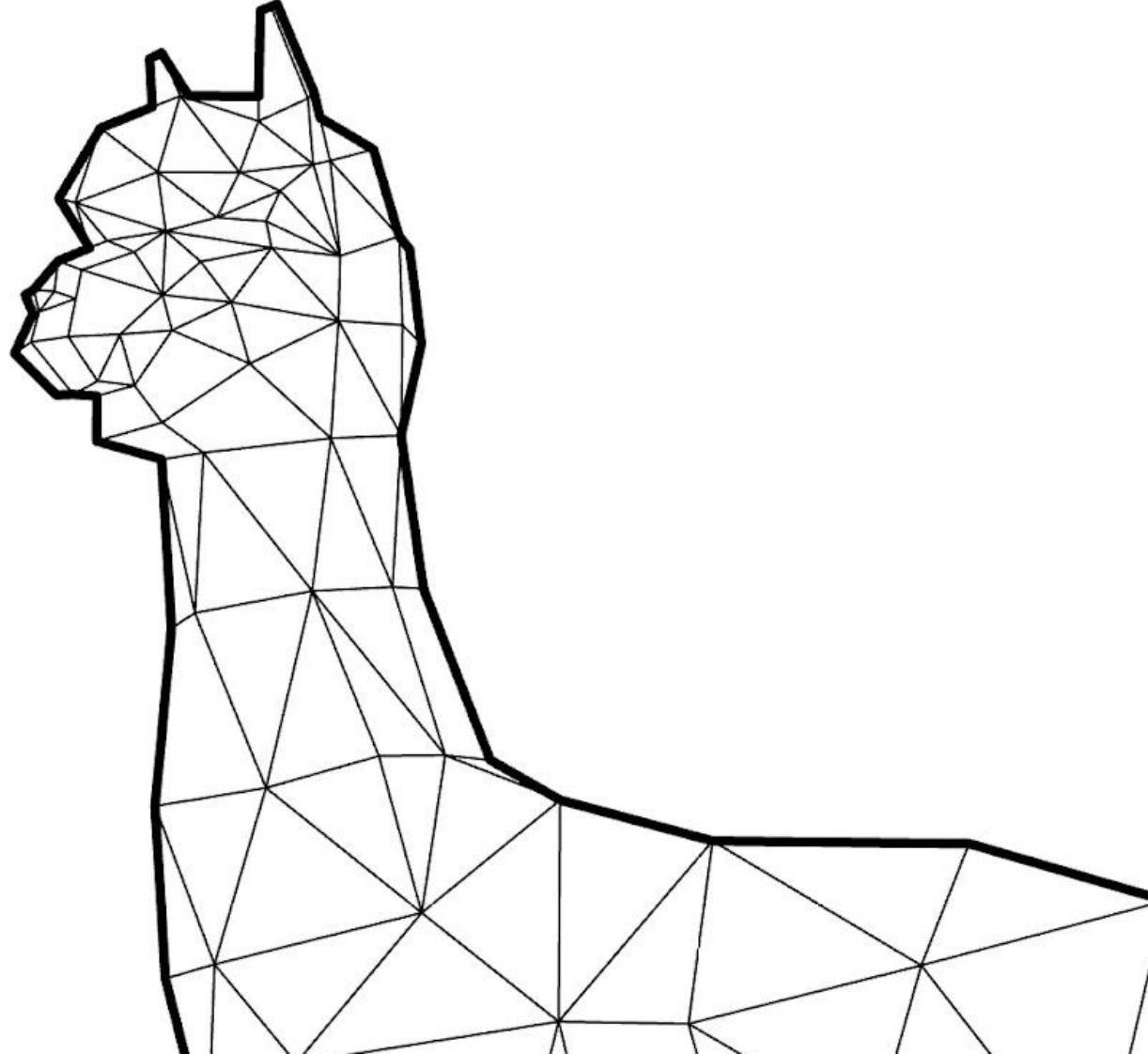
Number of points: 6561
Computing time: 13.0 s

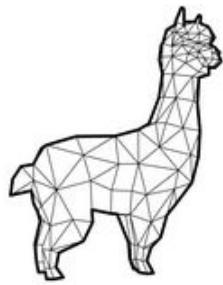


Number of points: 25921
Computing time: 3.7 min

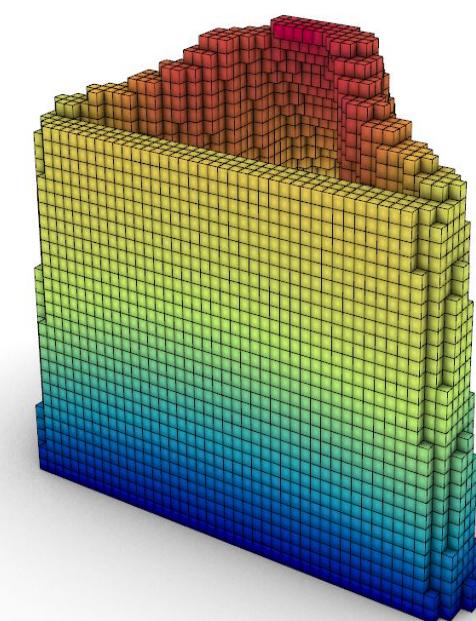
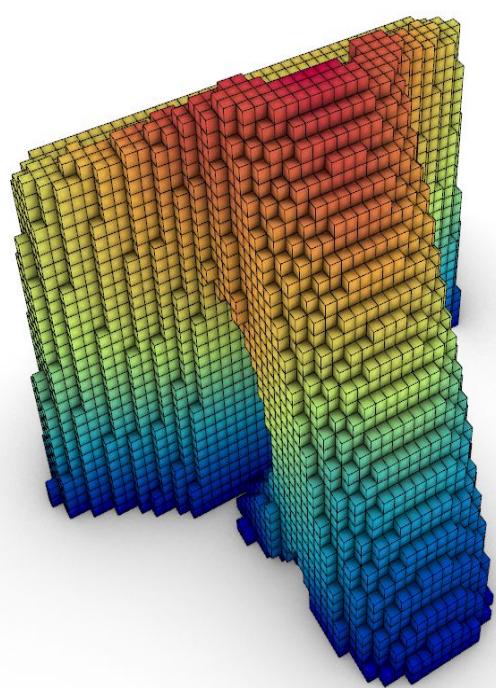
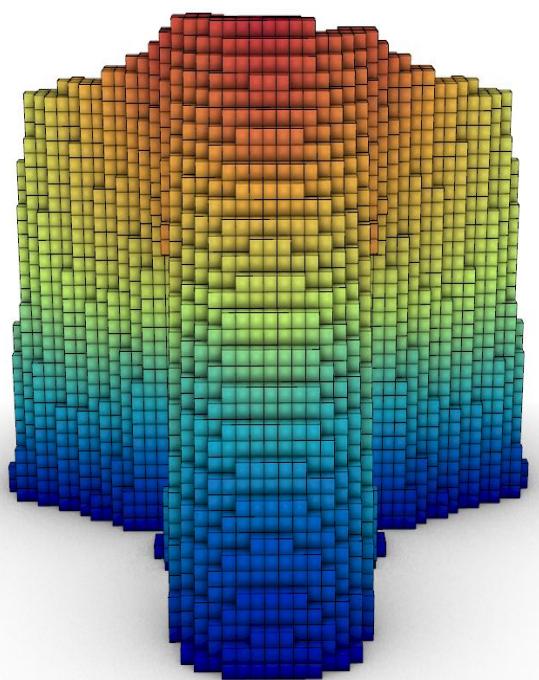
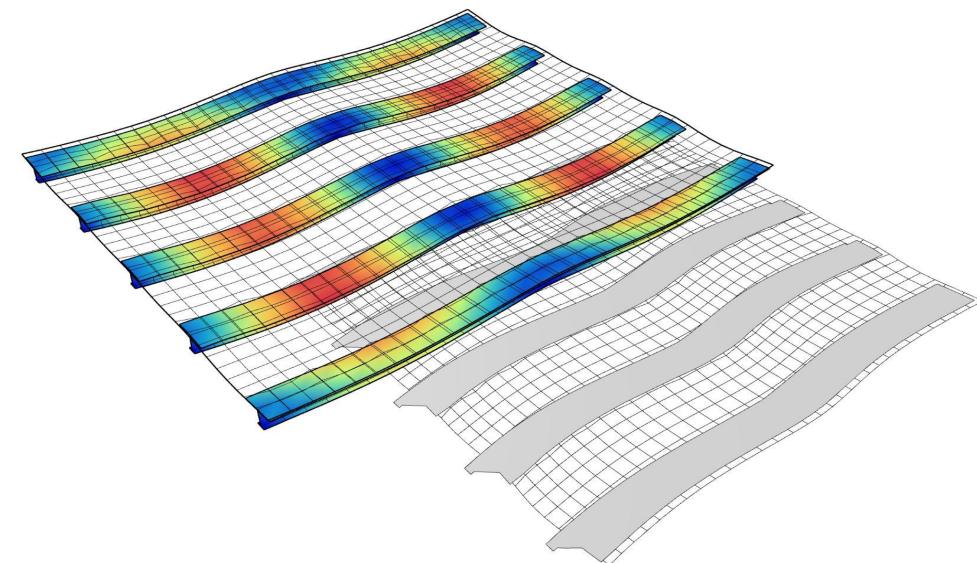
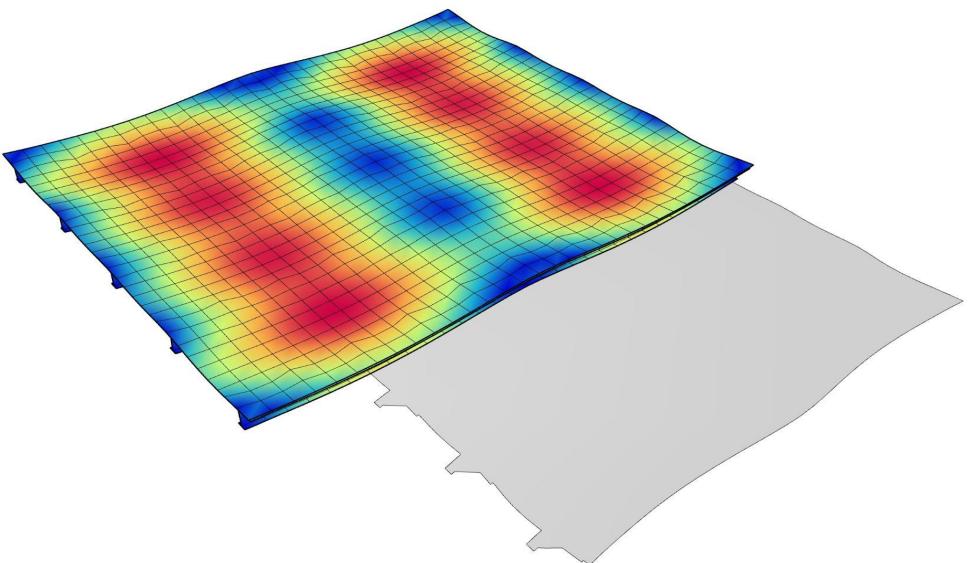


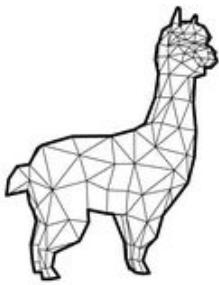
What Can I analyse?



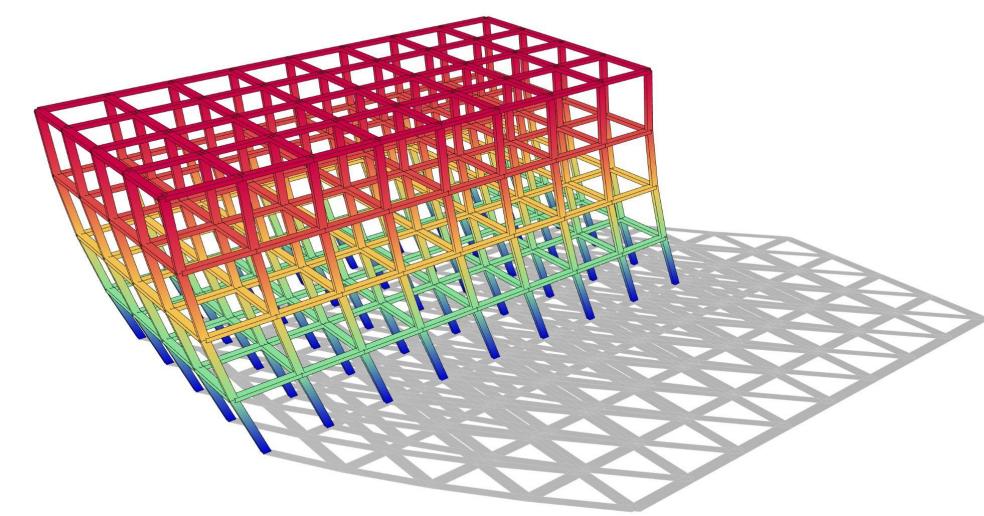
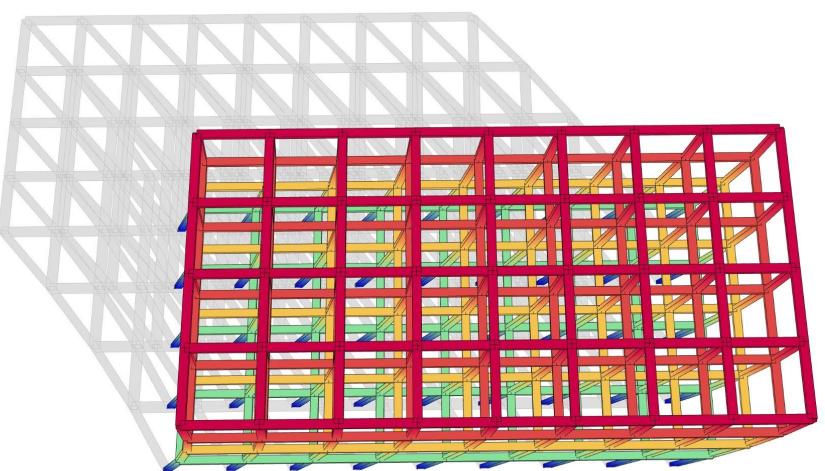
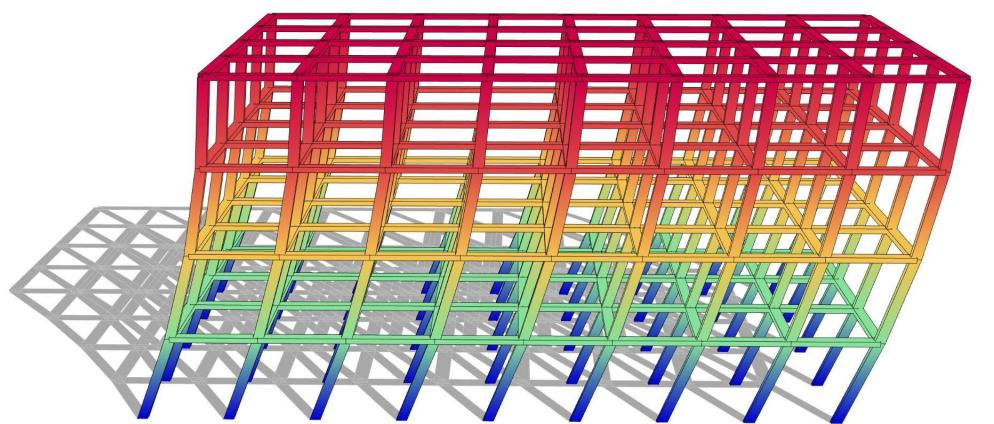
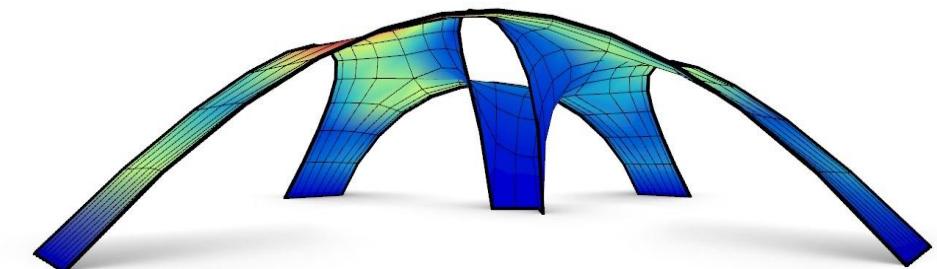
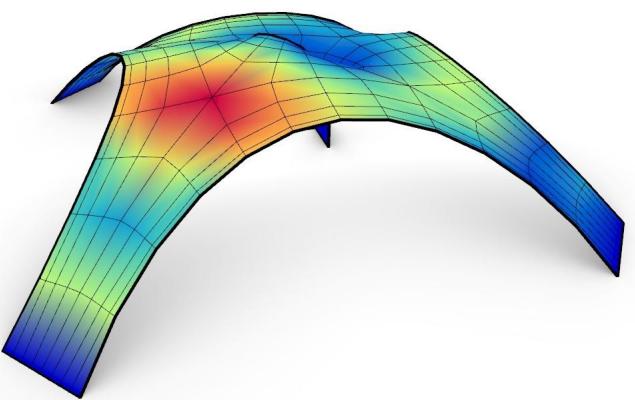
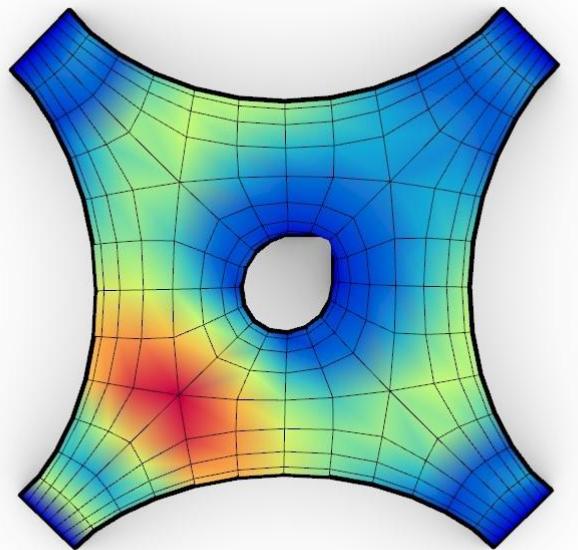


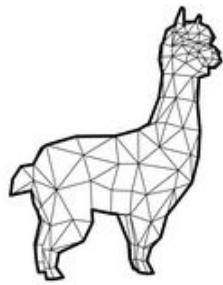
Examples



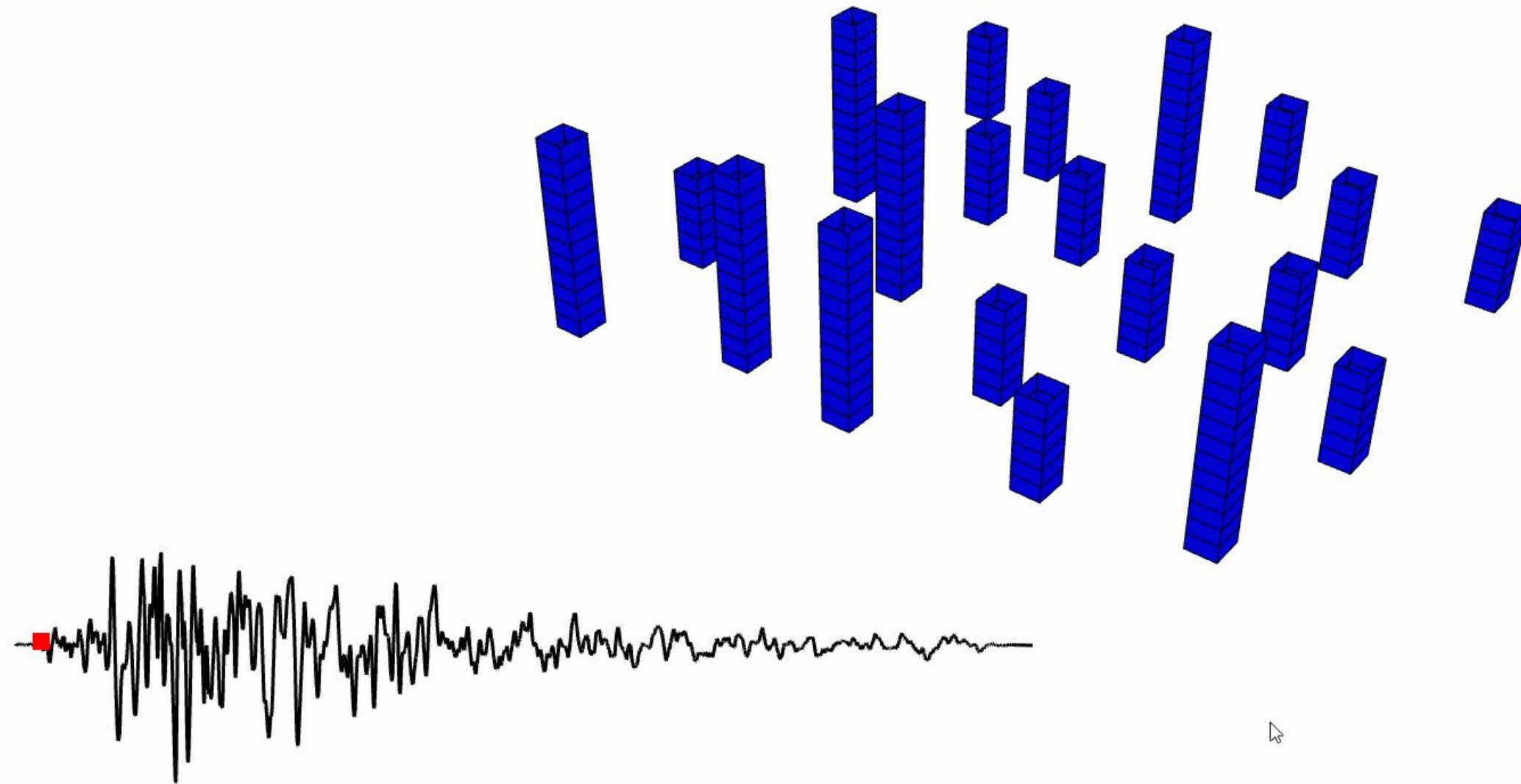


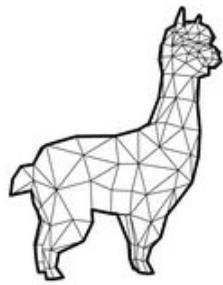
Examples



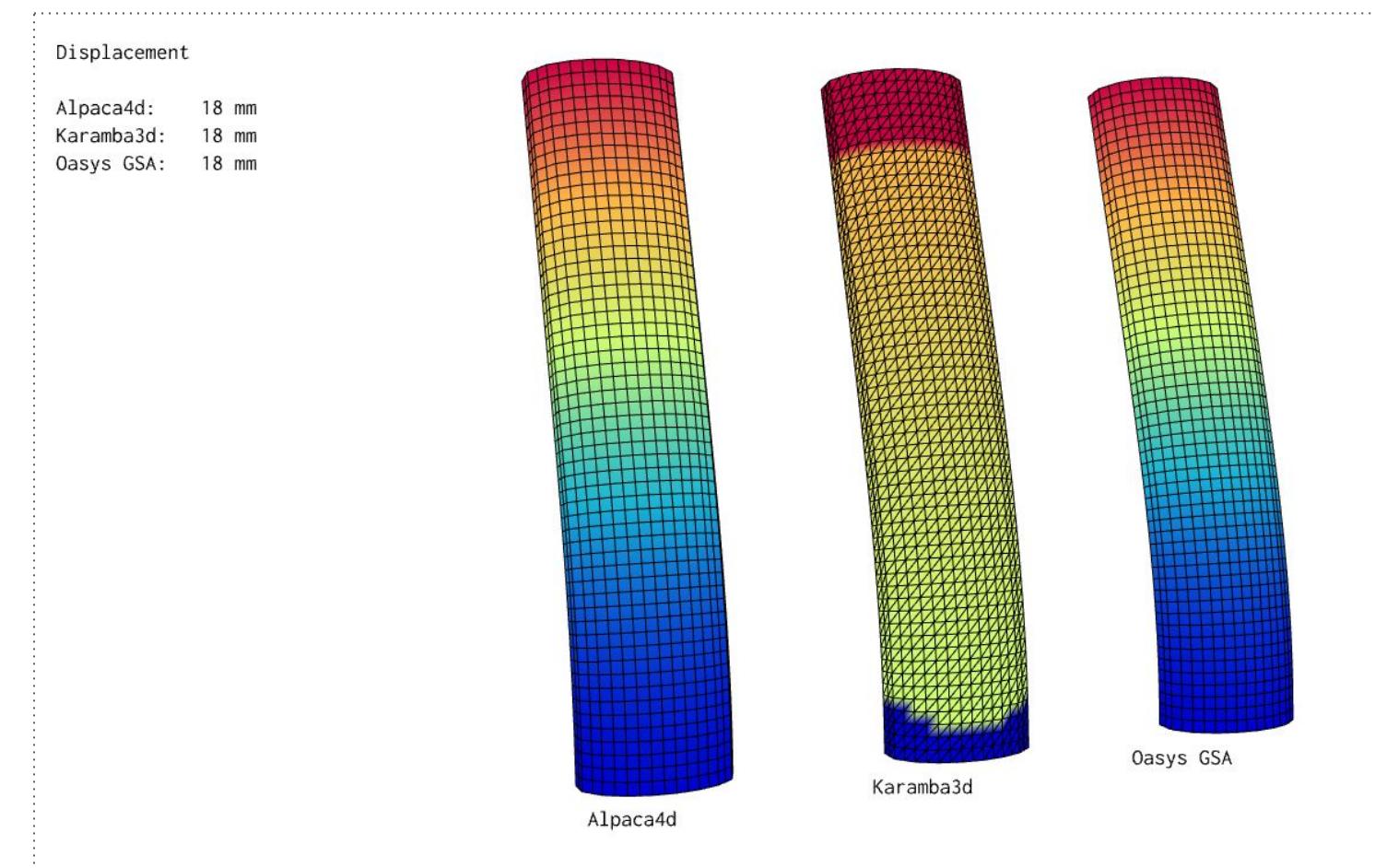
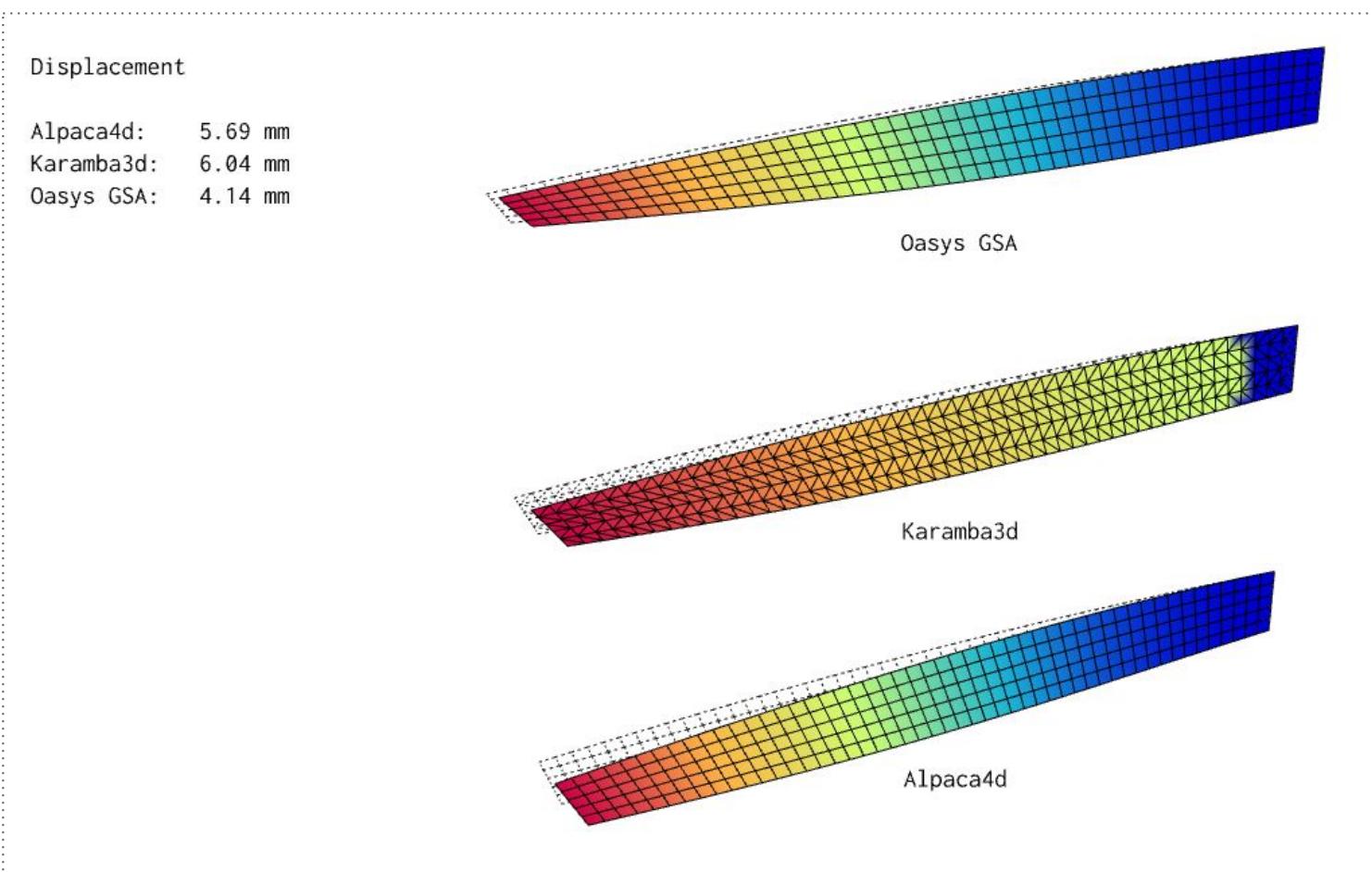


Examples

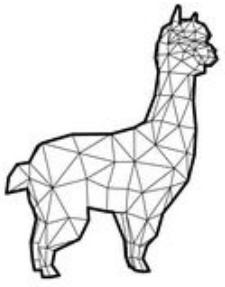




Benchmark



Analytical solution = 5.68 mm



WIP

Technical aspect

- Adding Elements type
- Adding Load type
- Acquiring data for Machine Learning purpose
- Outputting Stress Utilisation for elements

Environmental aspect

- Embodied Carbon tool

Belief

- Better Design for a better world



alpaca4d.github.io



[Alpaca4d](https://www.instagram.com/Alpaca4d)