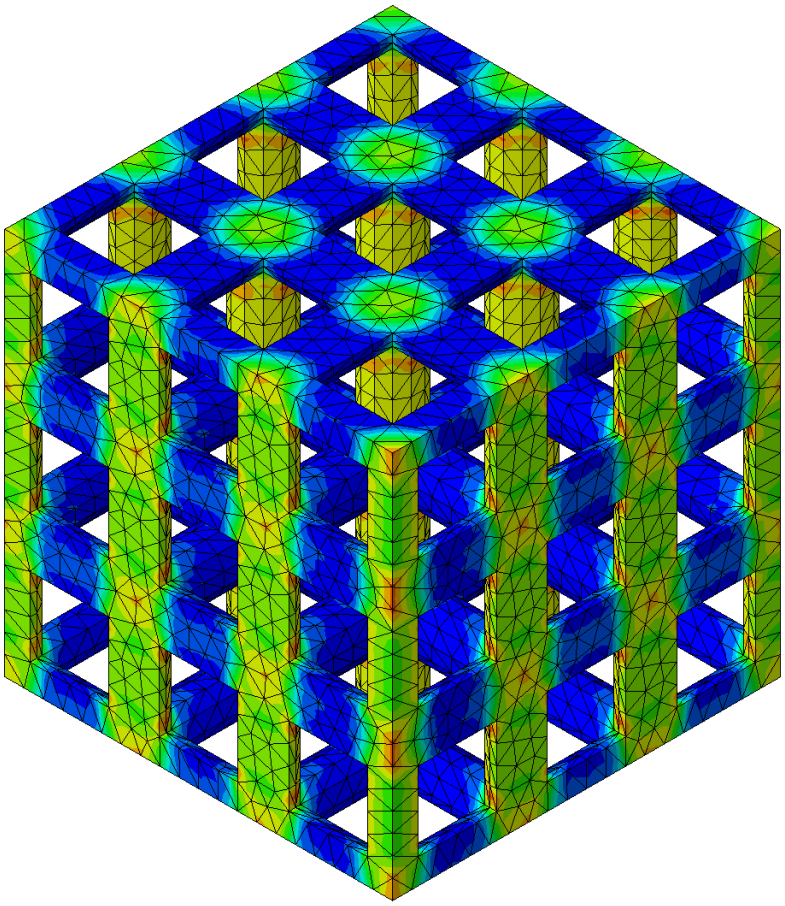
ETH zürich

ENGINEERING DESIGN AND COMPUTING



Computational Evaluation and Optimization of Mechanical and Form Properties of 3D-Printed Bone Tissue Engineering Scaffolds

Author: Max Engensperger

Supervised by: Dr. Paul Egan

Master's Thesis

FS 2016

ABSTRACT

This thesis is about computational mechanical simulation and optimization of scaffold designs for bone tissue engineering. The scaffolds consist of repeating unit cells which are made of struts creating a lattice structure with discrete elements. Several different unit cells are mechanically evaluated and compared via the finite element method. The simulation makes use of Euler-Bernoulli beam elements which represent the scaffold as a wire frame.
As design variables the strut thickness of each strut of a unit cell and the overall cell size of the unit cells are used. As optimization goal, the biomimetic approach is set, to match the compliance of the scaffold with human cortical bone.
It was determined that the wire frame approximation allows for fast mechanical evaluation of scaffolds, however leads to inaccuracy because of beam elements. The resulting scaffolds have matching mechanical properties with bone, but would have to be tested empirically in mechanical loading experiments to confirm this statement.
Design automation is achieved through the programming language Python, using the Python distribution Anaconda, and the evaluation is executed with the software Abaqus from Dassault Systèmes.
The code that was written for this thesis can be retrieved at
`https://github.com/Engensmax/Truss-Builder`.

ACKNOWLEDGEMENTS

I would like to thank Dr. Paul Egan for supporting this thesis from beginning to end.

My thanks to Cynthia Tan for handling the 3d-printing of some examples for me.

My thanks to Prof. Dr. Kristina Shea for the opportunity to write my master's thesis at the Engineering Design and Computing Lab (EDAC).

A thank you to the community of stack overflow for providing answers to my questions.

The following open-source software was directly used for this work:

PYTHON https://www.python.org/

ANACONDA https://www.continuum.io/

PYCHARM COMMUNITY EDITION https://www.jetbrains.com/pycharm/

SCILAB http://www.scilab.org/

VEUSZ http://home.gna.org/veusz/

CSV VIEWER http://www.csvviewer.com/

LATEX https://www.latex-project.org/

MIKTEX http://miktex.org/

TEXSTUDIO http://texstudio.org/

MINTED https://github.com/gpoore/minted

Additionally I would like to thank the people that pay taxes in Switzerland for coming up with a big chunk of the costs of my education. I hope to have it payed back someday.

I want to dedicate this thesis to all the people on the world that contribute to free software and free educational resources voluntarily.

My deepest regards to you.

For my personal well being throughout my bachelor's and master's studies I would like to thank the following people: Thierry Baasch, Damian Birchler, Remo Breitenmoser, Kerstin Cramer, Roman Ebneter, Arnold Engensperger, Denise Engensperger, Vanessa Engensperger, Valentin van Gemmeren, Sandro Kundert, Carine Neier, Tobias Peteler, Samuel Schafer, Franziska Schmidt, Sonja Segmüller, Rudolf Gerhard Stolz, Christoph Thormeyer, Thomas Witrahm, Jan Wolf, Sifu David Young, Christoph Zumbrunn

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ETHZ Eidgenössische Technische Hochschule Zürich — Swiss Federal Institute of Technology Zurich

EDAC Engineering Design and Computing Lab

TE Tissue Engineering

FEM Finite Element Method

FEA Finite Element Analysis

CAE Computer-aided Engineering

CAD Computer-aided Design

CAO Computer-aided Optimization

DOA Deterministic Optimization Algorithm

SOA Stochastic Optimization Algorithm

AM Additive Manufacturing

SLA StereoLithography

SLM Selective Laser Melting

SLS Selective Laser Sintering

CG Conjugate Gradient

BFGS Broyden-Fletcher-Goldfarb-Shanno

L-BFGS Limited Memory BFGS

L-BFGS-B Limited Memory BFGS with bounds

SLSQP Sequential Least Squares Quadratic Programming

HA Hydroxyapatite

TCP Tricalciumphosphate

PLA Polylactic Acid

1

INTRODUCTION

1.1 MOTIVATION

Bone is a complex adaptive material. It decomposes and rebuilds itself throughout its life. It is of great importance to understand the mechanisms that cause bone to regrow to be able to develop new medical treatments . As of today, regrowing bone still poses a major challenge for large bone defects where the bone would not regrow by itself. [1]
Bone Tissue Engineering (TE) is a steadily developing, yet not fully understood method to regenerate large bone defects [2]. Small Scaffolds with sizes on the scale of centimeters are inserted into the bone, bridging the gap and providing the bone cells a place to grow and proliferate [3].
The scaffolds require pores to provide surfaces for bone cells to initially seed and form tissue. An approach from classical engineering would be to generate scaffolds consisting of beams to create a framework. The advantage of that design is the ability to evaluate the mechanical properties faster within a Finite Element Method (FEM) simulation by applying the Euler-Bernoulli beam theory instead of other methods.
Additive Manufacturing (AM) offers a method to create small scaffolds with controlled microscale features. This opens a new design space in terms of producibility which is not offered by many other methods.
The mechanical properties of the scaffold may also influence the time it takes to regrow bone. Bone has an intrinsic mechanism to reinforce its structure where stress is applied; Bone's mechanical properties also vary a lot throughout the body and depend on the direction force is applied [4]. Therefore it could be helpful to generate scaffolds that have the same compliance that bone has to avoid stress shielding [1] and alleviate the regrowing phase for the body. By computationally optimizing the scaffold will also be possible to fit the mechanical properties of the individual bone.
This thesis shows an approach of automatically generating scaffolds as beam lattices with varying topology and beam thicknesses that are mechanically simulated and evaluated to fit certain target properties such as elastic modulus, shearing modulus, porosity and others to the properties of bone.

1 Stress shielding: loads are taken by a prosthesis and shielded from going to the bone [5]

2

STATE OF THE ART

2.1 BONE TISSUE ENGINEERING

Bone-TE has the goal to surgically repair or replace damaged bone by implanting cells, scaffolds, DNA, protein or protein fragments.[1] There are three widely accepted approaches [6],[7], [8]:

1. Implant a scaffold for cells to grow and proliferate
2. Extract cells and cultivate them in vitro to transplant them back into the patient.
3. Extract cells and cultivate them in a scaffold in vitro to transplant the cell-scaffold construct back into the patient.

The method introduced in this thesis is based on the first approach.

Main cases to use Bone-TE on a human are for spinal fusion[1] or for major fractures.
The developed scaffold should have a certain **stiffness** that is not too soft ,which could cause the structure to collapse, and not too hard, which could cause stress shielding that would slow down the rehabilitation process. One can argue that the target compliance of the meta material should be as close as possible to actual bone, also known as the biomimetic approach. The biomimetic approach requires a material with higher elastic modulus than polymers, reducing the number of material choices. On the other hand it is common to add a support structure to keep the bones and the scaffold in place during rehabilitation which takes a lot of load away from the scaffold, making its mechanical properties require a different tuning.
For Bone Tissue to regrow, a widely used method is to focus on the **pore size** of a potential scaffold. If the pore size of a scaffold is too big it will take a longer time for the bone tissue to grow. If the pore size is too small the scaffold will lack nutrient transport.
Coming along with pore size is **porosity**. Tendency wise one could say that the higher the porosity, the better the nutrient flow, but the lower the stiffness of the material. However, the interconnectivity of cells is a mechanic that is

1 Spinal fusion: A surgical technique to fuse two vertebrae. [9]

more complex and requires fluid dynamics simulations to provide a well researched statement about nutrient flow.
The most important factor is the biocompatibility. Most materials cause various foreign body reactions such as inflammation or even necrosis. This limits the material choices that can be made for engineered tissue. Another important quality of a bone-TE-scaffold is **absorbability**. Ideally the scaffold gets absorbed in the process leaving nothing but the regrown bone tissue. It is very difficult to achieve the right absorbability, because the scaffold must not be absorbed too fast nor too slow.
In summary taken from [10], desirable qualities of a bone-TE scaffold are the following

- Available to surgeon on short notice
- Promotes bone ingrowth
- Absorbs in predictable manner in concert with bone growth
- Does not induce soft tissue growth at bone/implant interface
- Adaptable to irregular wound site, malleable
- Average pore sizes approximately 200-400 μm
- Maximal bone growth through osteoinduction and/or osteoconduction
- No detrimental effects to surrounding tissue due to processing
- Correct mechanical and physical properties for application
- Sterilizable without loss of properties
- Good bony apposition
- Absorbable with biocompatible components

2.1.1 *Scaffold Designs*

Scaffold designs can be divided into two sections:
Discrete scaffolds that consist only of struts/beams/bars creating a lattice structure, and **continuous scaffolds** which are made by adding or subtracting volumetric shapes. While the discrete scaffolds offer a way for faster evaluation (see 2.2.3), the continuous scaffolds allow more intricate designs. A few designs from other research are shown in Appendix A.2 to give an impression of the vast amount of design possibilities.

2.1.2 *Material choices*

Important aspects regarding the material choice are **biocompatibility**, **absorbability** and **mechanical properties**. Research can be found about the following materials ([11], [12]):

Metals

The high stiffness of **Titanium** (Young's modulus of about 125 GPa) and its alloys allows very thin struts and high porosity while still achieving the stiffness of bone. The human body does not absorb those metals and the scaffold will stay inside the patient which can cause stress shielding [13]. A promising material choice may be **Magnesium** (Young's modulus of about 46 GPa), which is absorbable by corrosion [14]. While a scaffold made of an alloy is usually manufactured via the additive manufacturing method Selective Laser Sintering (SLS), metallic foams can be created by a powder metallurgical process [15].

Ceramics

Bone itself consists of at least 50% Hydroxyapatite (HA) [1] (ref Paper optimal design and fabrication...), which makes HA a logical choice for bone tissue engineering. Another ceramic that has been used is α-β-Tricalciumphosphate (TCP) (ref Paper and weber).

Polymers

Examples of biocompatible polymers [16] which can be manufactured additively are **Polycaprolactone** (PCL) [17], **Polylactic acid** (PLA) [18] or **Polypropylene fumarate/diethyl fumarate** (PPF/DEF) [19].

Others

Other approaches include the materials **bioglass biocomposites** [20], **nHAC powder/PLA composite** [18], [21]. Both materials are bioactive [2].

2 A bioactive material causes a biological response at the interface of the material which results in a bond between the tissue and the material [22]

2.2 MECHANICS

2.2.1 *Compliance*

The compliance offers a useful way to describe a structure's mechanical properties. It can be described by a matrix and contains the information that characterizes the linear elastic response of the structure to applied forces (see equation 1). From the compliance matrix properties such as Young's modulus, shearing modulus and Poisson's ratio can be calculated (see eq. 5). The compliance is independent of the outer form of a structure, which is ideal for describing a scaffold which might be repeated continuously. The compliance matrix also contains the information whether a material is anisotropic (eq. 2), orthotropic (eq. 4) or isotropic (eq. 3).

The relation between strain and stress in a material can be described by the compliance matrix.

$$\begin{pmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{pmatrix} = \epsilon = C * \sigma = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} \end{pmatrix} * \begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \tau_{23} \\ \tau_{31} \\ \tau_{12} \end{pmatrix} \tag{1}$$

Anisotropic Compliance is characterized by non-zero elements in the top right and bottom left quadrant of the compliance matrix. This means, that the material reacts with shearing displacements on compression or vice-versa.

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & c_{46} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & c_{56} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & c_{66} \end{pmatrix} \tag{2}$$

An Isotropic Compliance describes a material which is uniform in all directions.

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{11} & c_{12} & 0 & 0 & 0 \\ c_{12} & c_{12} & c_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{44} \end{pmatrix}; c_{44} = c_{11} - c_{12} \tag{3}$$

Orthotropic Compliance, where the matrix is symmetric and its top right and bottom left quadrant is zero as well as the bottom right quadrant is a diagonal matrix.

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & 0 & 0 & 0 \\ c_{21} & c_{22} & c_{23} & 0 & 0 & 0 \\ c_{31} & c_{32} & c_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{66} \end{pmatrix} \tag{4}$$

These equations show the connection between compliance and other material properties for an orthotropic material.

$$\begin{aligned} E_{11} &= \tfrac{1}{c_{11}} & E_{22} &= \tfrac{1}{c_{22}} & E_{33} &= \tfrac{1}{c_{33}} \\ \nu_{12} &= -\tfrac{c_{12}}{c_{11}} & \nu_{23} &= -\tfrac{c_{23}}{c_{22}} & \nu_{31} &= -\tfrac{c_{31}}{c_{33}} \\ \nu_{21} &= -\tfrac{c_{21}}{c_{22}} & \nu_{32} &= -\tfrac{c_{32}}{c_{33}} & \nu_{13} &= -\tfrac{c_{13}}{c_{11}} \\ G_{23} &= \tfrac{1}{c_{44}} & G_{31} &= \tfrac{1}{c_{55}} & G_{12} &= \tfrac{1}{c_{66}} \end{aligned} \tag{5}$$

E_{xx}: Elastic Modulus
ν_{xy}: Poisson's ratio
G_{xy}: Shearing Modulus

2.2.2 *Meta-material*

A meta-material is an arrangement of engineered elements, where the local properties differ from the global properties. This is used to create specific physical properties. One could also say, that meta-materials present the next level of structural organizational matter [23].
In this thesis, the generated scaffolds are regarded as meta-materials. The

meta-material does not take account of the intrinsic truss structure, which allows for describing the compliance (see 2.2.1) as if it were a genuine material. This is particularly useful for scaffold applications, since the outer measurements of the scaffold may change for the individual use, whereas the local structure stays the same.

2.2.3 *FEM*

The FEM (or Finite Element Analysis (FEA)) subdivides a larger problem of partial differential equations into numerical subproblems of sets of linear equations. For a mechanical analysis this means that the approach from continuum mechanics is subdivided into discrete elements. The sets of linear equations can be solved numerically by a computing system leading to an approximation of the solution. There are different FEM-solvers commercially available. For this thesis the ABAQUS-solver by SIMULIA was used.

2.3 OPTIMIZATION

Mathematical optimization (alternatively mathematical programming) is the method of searching for the best set of variables to minimize the value of a function. Usually one considers a problem as seen in equation 6. An algorithm is defined that evaluates the function for certain sets of variables and follows a strategy to find sets which minimize or maximize the value of the function (based on the function evaluations).
It is important to note, that there is a diversity of algorithm types that differ in their effectiveness in searching a problem space to find best sets in the shortest time for any given problem. The problem needs to be analyzed and a fitting algorithm has to be found. Optimization can be further divided into deterministic (2.3.1) and stochastic (2.3.2) methods.

$$
\begin{array}{lll}
\min\limits_{x} & f(x) = w_1 E_1(x) + w_2 E_2(x) + ... & \\
s.t. & g_1(x) & \leq \ 0 \\
 & g_2(x) & \leq \ 0 \\
 & \vdots & \\
 & h_1(x) & = \ 0 \\
 & h_2(x) & = \ 0 \\
 & \vdots &
\end{array}
\tag{6}
$$

$f(x)$:	Objective Function
$E_n(x)$:	Evaluation functions
w_n:	Weighting Factor for E_n
$g_n(x)$:	Inequality constraint
$h_n(x)$:	Equality Constraint

2.3.1 *Deterministic Optimization*

The main idea behind all deterministic methods is to search for the direction of steepest descent of the evaluation function and perform a line search into that direction until the set of variables for the smallest function evaluation is found. After that, this process is repeated until a stopping criteria is fulfilled. Deterministic Optimization Algorithm (DOA) search for the solution without the use of randomness. This means that if the initial set of variables is the same the final set of variables will be the same as well.
DOAs can be further classified by the amount of information they require to find the next search direction. The simplest DOAs sustain solely on the evaluation of the function itself (f.e. Downhill Simplex, Nelder-Mead, Powell, Response Surface). More extensive algorithms request the evaluation of the first derivatives with respect to the optimization variables, which is also called

$$H = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial^2 x_1{}^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial^2 x_2{}^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial^2 x_n{}^2} \end{pmatrix} \tag{8}$$

It is evident, that for a large set of variables the Hessian becomes a greater calculation effort.

the gradient of the function (equation 7), a vector that shows towards the direction of steepest descent (f.e. Cauchy, Fletcher-Reeves). The most expensive algorithms calculate the second derivatives, a matrix called the Hessian (equation 8) (f.e. Newton, Quasi-Newton, BFGS) [24].

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \tag{7}$$

2.3.1.1 *The BFGS-Algorithm*

The Broyden-Fletcher-Goldfarb-Shanno (BFGS)-algorithm is an approximate of the Newton's method. Instead of evaluating the Hessian matrix of second derivatives, an approximation of it is updated every step by using the gradient information. The BFGS-Method is one of the most popular members of this class [24]. Also quite common is the Limited Memory BFGS (L-BFGS)-method, which differs from BFGS by having a limited memory usage, and the Limited Memory BFGS with bounds (L-BFGS-B)-method, which additionally to limited memory makes use of bounding constraints. L-BFGS-B is implemented into the *optimize*-package of the *scipy*-package, which is a open-source software for mathematics, science and engineering with the programming language *python*.

2.3.2 *Stochastic Optimization*

In contrast to deterministic algorithms (2.3.1), a Stochastic Optimization Algorithm (SOA) does not always have the same outcome each time it is run. New sets of variables are determined with a random aspect. Rather than searching for a direction with a descent of the function value, SOAs define new sets of variables randomly. More complicated algorithms use the information they get to search in areas that are more likely to have better solutions.

A common group of SOAs are called evolutionary algorithms which are inspired by the evolution of biological organisms. These algorithms initially evaluate the objective function with random sets of variables. They then proceed to search for new solutions close to the sets which showed better results. Other prominent algorithms are Random Search, Stochastic Hill Climbing, Simulated Annealing.

3

GOALS

The objective of this thesis can be divided into 3 points:

MECHANICAL EVALUATION Computationally evaluate mechanical properties of a given scaffold

FORM EVALUATION Computationally evaluate form properties of a given scaffold

OPTIMIZATION Computationally optimize for target values respective to mechanical and form properties

3.1 MECHANICAL EVALUATION

The evaluation should be able to determine the reaction of the scaffold to compression and shearing loads. From the displacements of the framework caused by applied loads the compliance can be calculated to be able to fully describe the mechanical behaviour of the scaffold. The compliance matrix describes the mechanical properties of the meta material (see 2.2.1 and 2.2.2). By only using struts the whole scaffold can be evaluated in a FEM software with a wire frame that uses the Bernoulli beam theory to calculate the mechanical reaction to an applied load. The properties that should be evaluated are **Young's modulus**, **shearing modulus** and **Poisson's ratio** which can all be read out of the compliance.

3.2 FORM EVALUATION

An interesting form property can be described as the **porosity** of the meta material which is defined by the volume of the scaffold divided by the volume of the whole meta material. Some research refers to the **void ratio** which is described by equation 9.
Other form properties that are of interest would be the **surface area** and the **pore size** (As mentioned in 2.1).

$$e = \frac{\phi}{1 - \phi} \tag{9}$$

e: Void Ratio
ϕ: Porosity

3.3 OPTIMIZATION

Having the computer automatically evaluate the mechanic and form properties of a scaffold allows to implement an automated computational optimization. The goal would be to determine interesting solutions within an acceptable amount of time. Once adequate solutions were found for certain target values, the other properties can be further compared between the topologies.

Mechanical Properties	Form Properties
Young's Modulus	Volume
Shearing Modulus	Porosity/Void Ratio
Poisson's ratio	Surface Area
Compliance Matrix	Pore Size

Table 1.: Summarized properties that should be determined in the evaluation

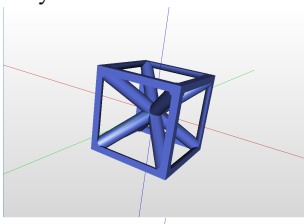
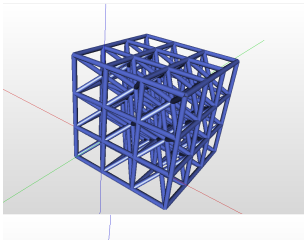
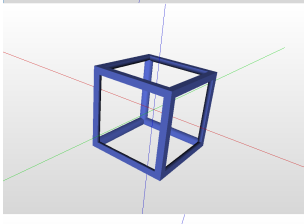
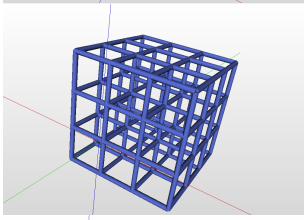
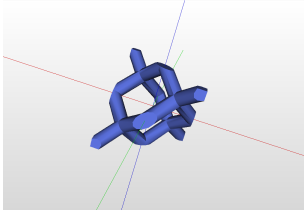
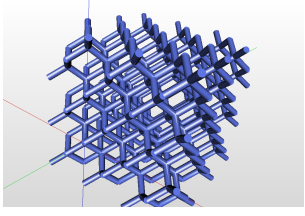
4

METHODS

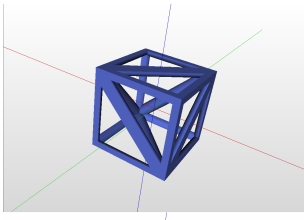
4.1 TOPOLOGIES

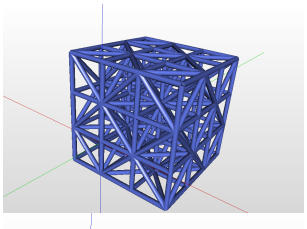
The unit cell topologies of the scaffolds are not automatically designed, but were developed by hand (see Table 2) and then automated to form a repeating solid structure, for form evaluation (3.2), and wire frame, for mechanical evaluation (3.1). Important factors for the design were inspiration by nature and geometry as well as comparability to other research. A general concern was to keep the topologies simple if possible to reduce the number of design variables required to tune each unit cell.
Because of the simplicity of **cubes** (see Table 2, Column 2) and its presence in other research ([30], [31], [32], [37], [38]) this model was chosen for comparison and validation of results.
As can be seen in the three by three by three lattices in Table 2, the unit cells are combined at their faces, which causes struts which are along the face of a unit cell to fuse with the strut from the unit cell next to it.

Table 2.: Topologies used in this thesis as a single cell and as a lattice consisting of three by three by three cells.

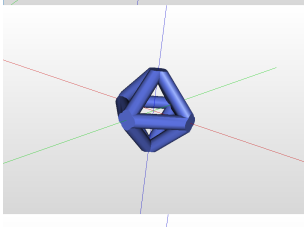
body centered cubes		
cubes		
diamonds		

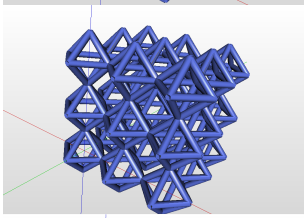
face diagonal cubes



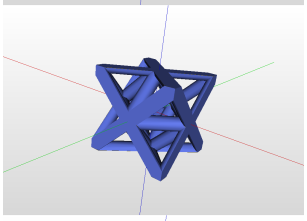


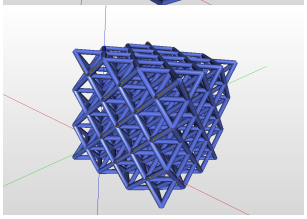
octahedra



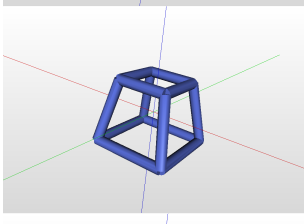


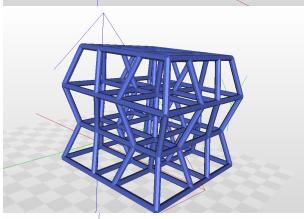
octets



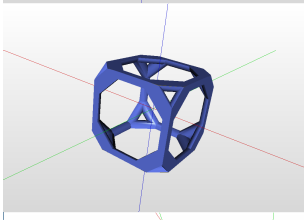


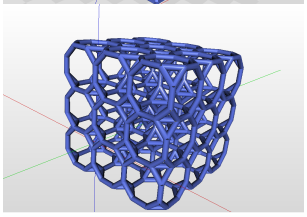
pyramids



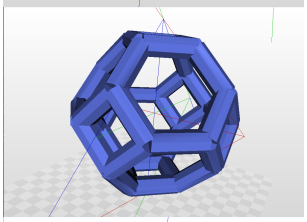


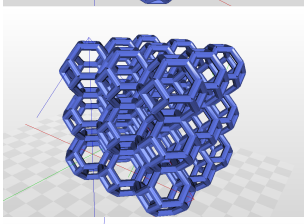
truncated cubes



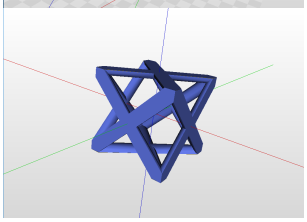


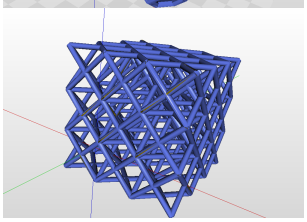
truncated octahedra





void octets





4.1.1 *Topology connections*

Three of the evaluated topologies, namely **truncated cubes** (Figure 1), **pyramids** (Figure 2), and **truncated octahedra** (Figure 3) allowed for a continuous change of the topology depending on a single variable which was called the cell ratio. An example of this ratio is the amount of truncation that is applied on the cubes topology. By reducing the truncation to zero, the cubes topology is created, and by maxing out the truncation, the original struts disappear resulting in the cuboctahedra topology (see Figure 1). In theory the cuboctahedra topology is equal to the octahedra topology in terms of strut arrangement. The cuboctahedra topology is the octahedra topology translated by half a cell size into every direction.

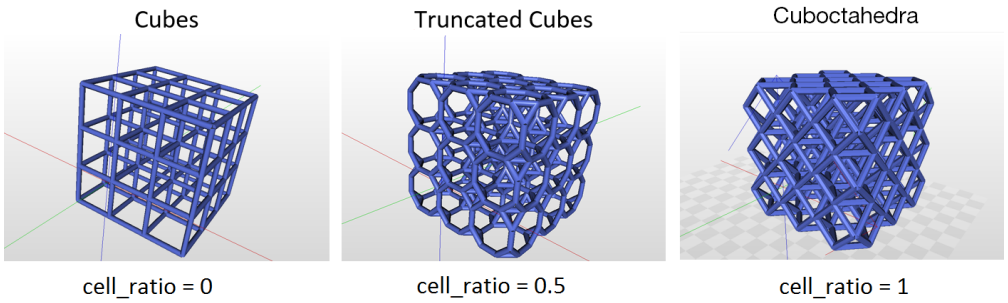


Figure 1.: This figure depicts the dependence of the truncated cubes topology to the cell ratio variable.

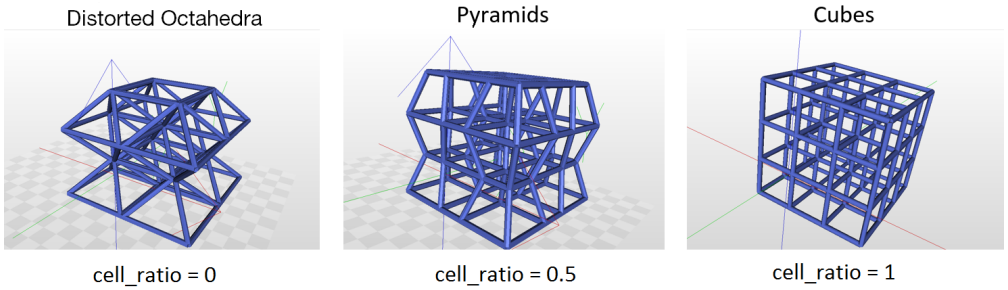


Figure 2.: This figure depicts the dependence of the pyramids topology to the cell ratio variable.

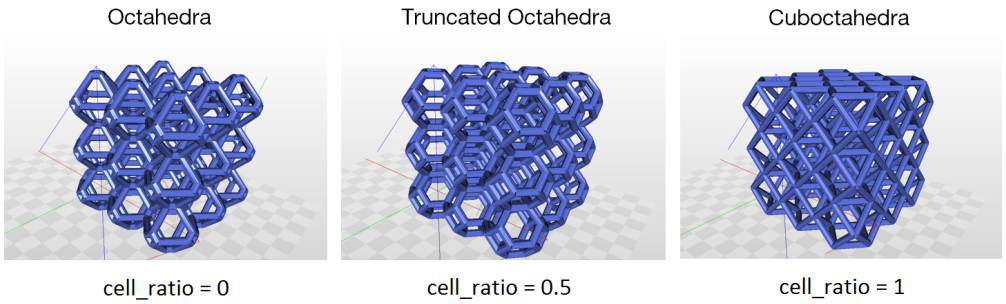


Figure 3.: This figure depicts the dependence of the truncated octahedra topology to the cell ratio variable. The octahedra topology on the right is shifted by half a cell in in comparison to the octahedra topology on the left

Another observation that was made is that the face diagonal cubes topology is the combination of the octet topology and the cubes topology, as is made clear in Figure 4.

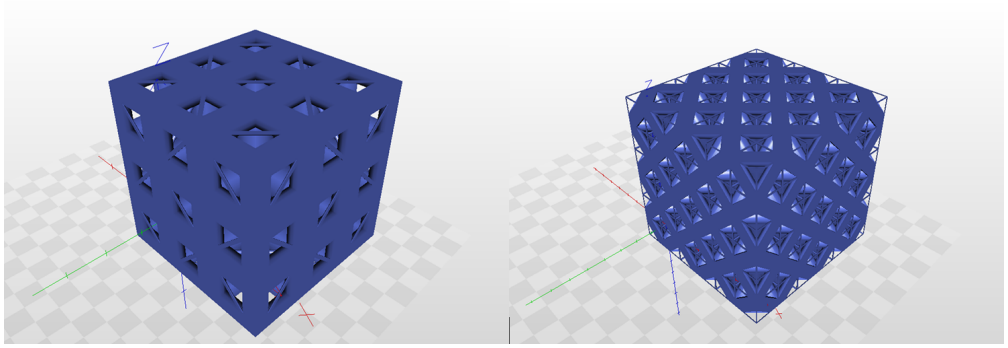


Figure 4.: On the left the face diagonal cubes topology is shown with very thin diagonal struts creating the cubes topology. On the right the same topology with doubled amount of cells per side is shown with very thin straight struts resulting in the octets topology.

4.2 FINITE ELEMENT METHOD

To evaluate the scaffold mechanically the FEM(see 2.2.3) was used. The mechanical behaviour of each strut of the scaffold is approximated with the euler bernoulli beam theorem. Quadratic shape functions were used while the element's length is set at a third of the single cell length (see Figure 5) in order to provide sufficient mechanical behavior. All evaluations are based on a linear elastic behaviour.

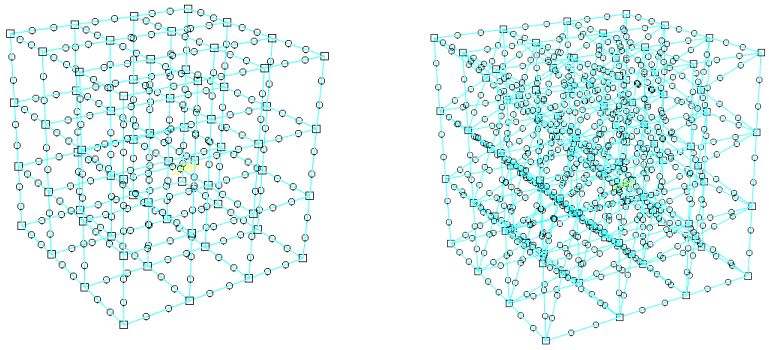
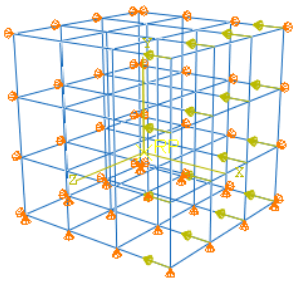
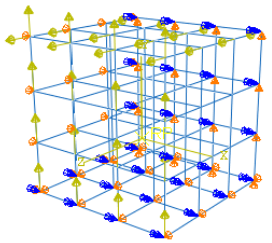
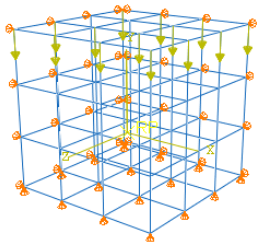
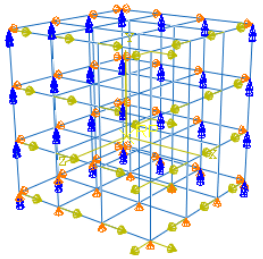
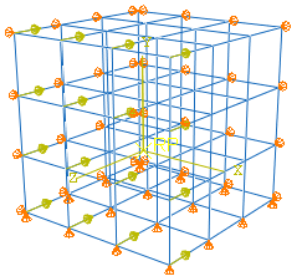
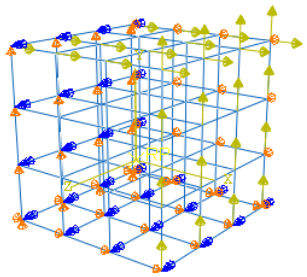


Figure 5.: FEM beam elements for three by three by three scaffolds with the **cubes** and **body centered cubes** topologies.

4.3 MECHANICAL ANALYSIS

To fully mechanically evaluate an orthotropic scaffold, six separate states are needed: 3 steps for determining the elastic moduli and 3 steps to determine the shearing moduli. In each state loads and fixations are applied to match a loading condition (see 3). The medians of the displacements on each side of the scaffold are read out to calculate the elements of the compliance (see 2.2.1) of the meta material.

Table 3.: The six states to calculate the compliance of a scaffold. The yellow arrows depict concentrated forces, the orange tips represent translational fixations and the blue symbols stand for rotational fixations

Sigma X		Tau YZ	
Sigma Y		Tau XZ	
Sigma Z		Tau XY	

$$\begin{pmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{pmatrix} = \epsilon = C * \sigma = \begin{pmatrix} * & c_{12} & * & * & * & * \\ * & c_{22} & * & * & * & * \\ * & c_{32} & * & * & * & * \\ * & c_{42} & * & * & * & * \\ * & c_{52} & * & * & * & * \\ * & c_{62} & * & * & * & * \end{pmatrix} * \begin{pmatrix} 0 \\ \sigma_{22} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{10}$$

$$d_{11}/l_{scaffold} = \epsilon_{11} = c_{12} * \sigma_{22} = c_{12} * \frac{F}{A_{scaffold}} \longleftrightarrow c_{12} = \frac{d_{11} * A_{scaffold}}{F * l_{scaffold}} \tag{11}$$

d_{11}:	displacement
$l_{scaffold}$:	length of the scaffold
F:	applied force
$A_{scaffold}$:	area of the scaffold

These equations show an example of the data that can be extracted from one simulation state (in this case **Sigma Y** and c_{12}). For each simulation state, six elements of the compliance matrix can be determined.

4.4 OPTIMIZATION

4.4.1 *Coupling of Design Variables*

To reduce the number of design variables and therefore reduce the search space, the thicknesses of certain struts were coupled. By decreasing the design variables, the optimization will find solutions exponentially faster. Nevertheless, there still need to be solutions that fit the optimization goal. The following rules were applied:

1. **The strut vis-à-vis** of the cell is coupled because it has to be equal if the design gets repeated continuously (as mentioned in 4.1).
2. All struts that face the **same direction** are coupled.
3. To always generate orthotropic materials **asymmetry is avoided**.

4.4.1.1 *Design Space Characterization*

Because of the way the design variables have been chosen it is to assume that the design space is convex. If the thickness of a strut that faces a certain direction increases, the whole stiffness of the meta material most likely increases as well in that same direction of the strut.
A deterministic algorithm, namely the BFGS-method (see 2.3.1.1) was used.

4.4.2 *Optimization Problems*

To show the capabilities of the method of this thesis, the following optimization goals are set:

MaxMin Optimization

Maximize for elastic modulus in z-direction and minimize for elastic modulus in x- and y-direction. The design variables are the thicknesses of the struts (see 5.2.1).

Biomimetic Optimization

Fit the elastic modulus properties to those of human cortical bone, for a scaffold made of titanium. The design variables are the thicknesses of the struts (see 5.2.2).

5

RESULTS

5.1 TOPOLOGIES

5.1.1 *Thickness Scaling*

In Figure 6 the evaluation for different topologies is shown with increasing thickness of the scaffold struts while keeping the cell size constant. Figure 7 zooms into the area with porosity of 60% to 90%, which is the interesting area to bone tissue engineering research for nutrient flow and space for cells to bind and proliferate on (see 2.1). Additionally, as will be mentioned in 6.1, the results are less accurate for lower porosities.

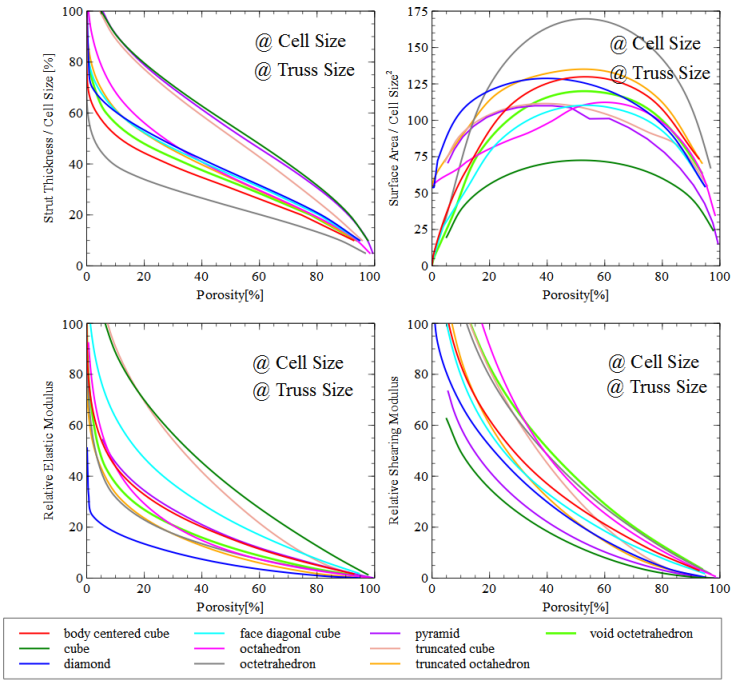


Figure 6.: Simulation results for different topologies in respect to porosity

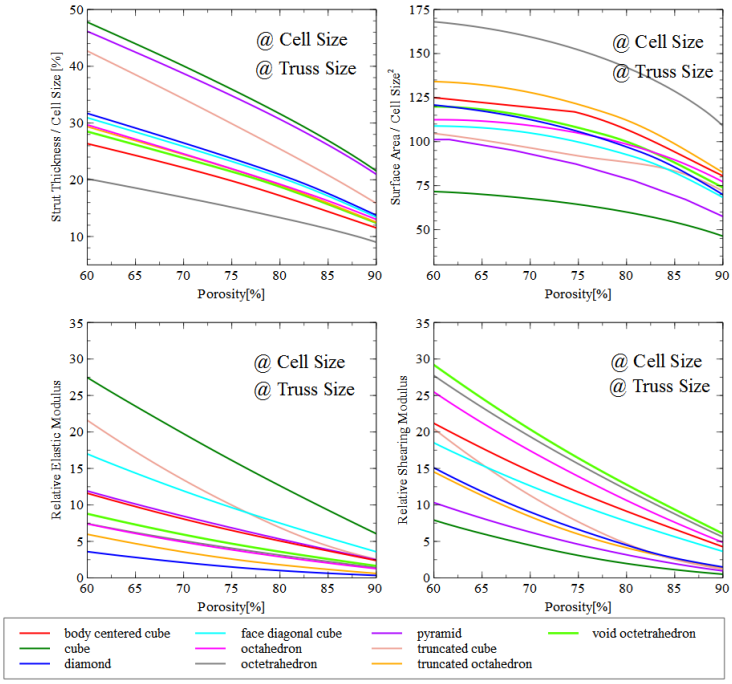


Figure 7.: Close-up of simulation results for different topologies in respect to porosity

Interpretation

As can be seen in other research ([26]), a distinction can be made between stretch-dominated and bend-dominated topologies. Stretch-dominated structures, such as cubes, body centered cubes, face diagonal cubes or octets, react to loads with axial stress in the struts, whereas bend-dominated structures, such as truncated octahedrons, octahedrons, diamonds or void octets, react with bending.
It can be seen in Figure 7 of section 5.1.1 that topologies that excel in compression generally perform bad in shearing and the other way around. One could say, that the no free lunch theorem applies here.
Some topologies perform worse than others in both cases. F.e. the diamond topology is outperformed by all topologies in compression and has a lower shearing modulus than most topologies. Because of the scope of this work no topology can really be excluded, since the mechanical properties might have to be low for some material choices.

5.1.2 *Cell Size Scaling*

In theory, the wanted surface area or pore size can be adjusted in the scaffold by scaling cell size and strut thickness proportionally while the other properties remain constant (see A.3). To compare this phenomenon, the topologies were evaluated at the same overall truss size, but with increasing number of cells and therefore decreasing cell size (see Figure 8). It is due to the inaccuracies that are mentioned in 6.1, that especially the elastic modulus deviates.

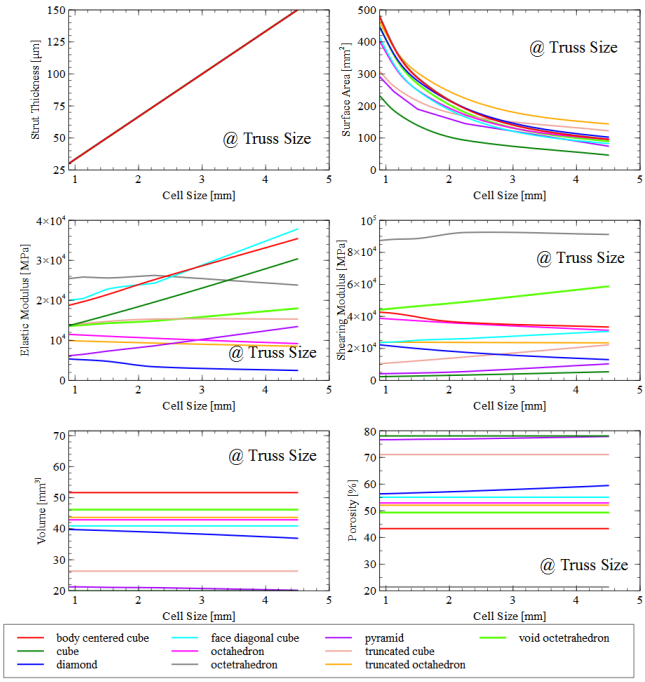


Figure 8.: Simulation results for different topologies while keeping the ratio between cell size and strut thickness constant.

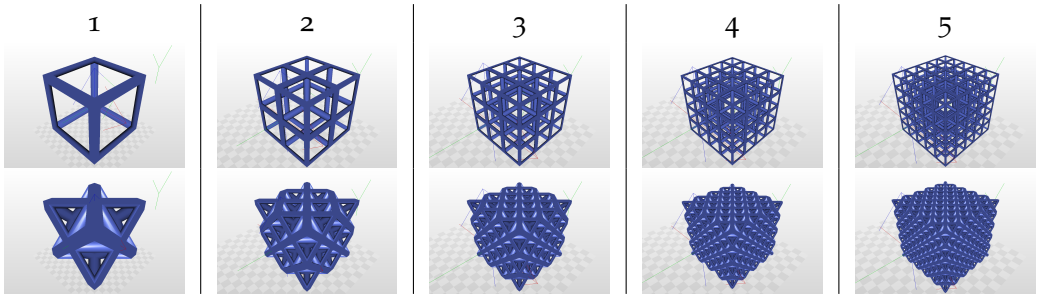


Table 4.: Examples of the cell size scaling, where the number of cells increases, but the overall truss size and porosity stays constant. This would allow for tuning the surface area and pore size of a given scaffold without changing its other properties.

5.1.3 *Ratio-changing Topologies*

As mentioned in 4.1.1, three topologies were further observed with a variable called the cell ratio that could change the topology of the cell f.e. the truncation of the cubes in the truncated cubes topology. In Figures 9, 10 and 11, the thickness scalings are shown with gradually increasing cell ratio.

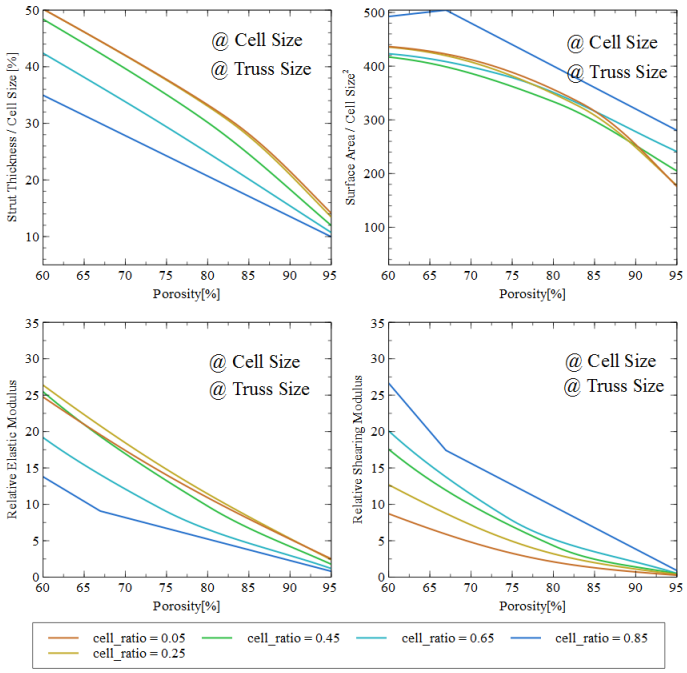


Figure 9.: Simulation results for different cell ratios for the truncated cubes topology.

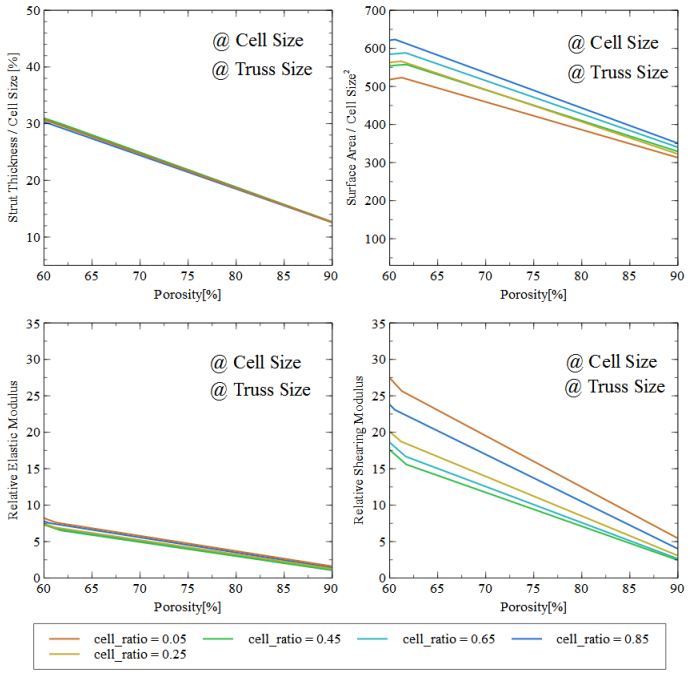


Figure 10.: Simulation results for different cell ratios for the truncated octahedra topology.

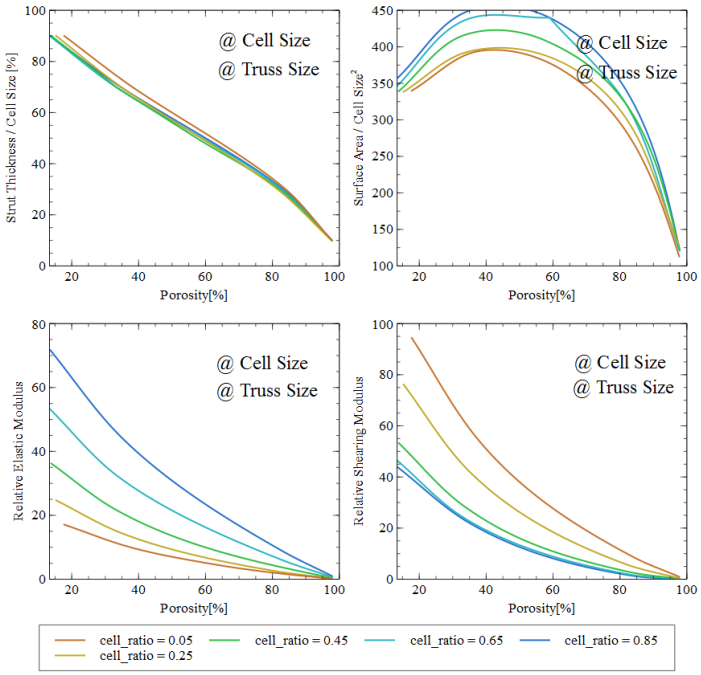


Figure 11.: Simulation results for different cell ratios for the pyramids topology.

Interpretation

These types of topologies offer a simple way of continuously varying the topology of a scaffold which might be an interesting optimization variable to use. Especially interesting are the pyramids and the truncated cubes topologies since their mechanical properties changes from high elastic modulus and low shearing modulus to low elastic modulus and high shearing modulus. The truncated octahedra and pyramids keep a nearly constant strut thickness over the cell ratio change at a given porosity, which might also be a desirable characteristic.

5.2 OPTIMIZATION

5.2.1 *Example of Minmaxing*

To test the capabilities of the presented method, an optimization goal was set that allowed for visual approving. Therefore the optimization goal was set to maximize stiffness into z-direction and minimize stiffness into x- and y-direction (see equation 12).

$$\begin{aligned} \min_{x} \quad & f(x) = E_x(x) + E_y(x) - E_z(x) \\ s.t. \quad & g_1(x) = x_1 \geq 0 \\ & g_2(x) = x_2 \geq 0 \\ & \vdots \end{aligned} \tag{12}$$

$f(x)$: Objective Function
$g_n(x)$: Inequality constraint

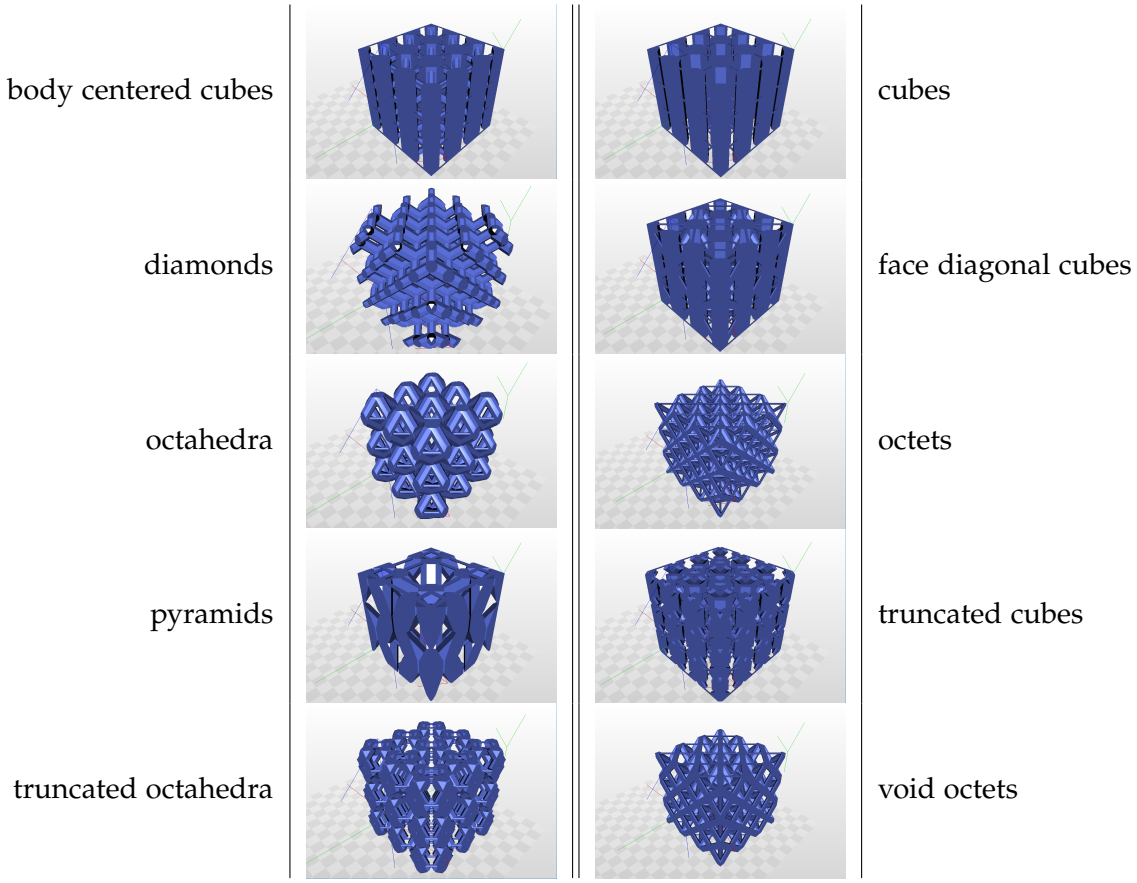


Table 5.: Optimization results for maximizing stiffness in z-axis and minimizing stiffness in x and y-axes.

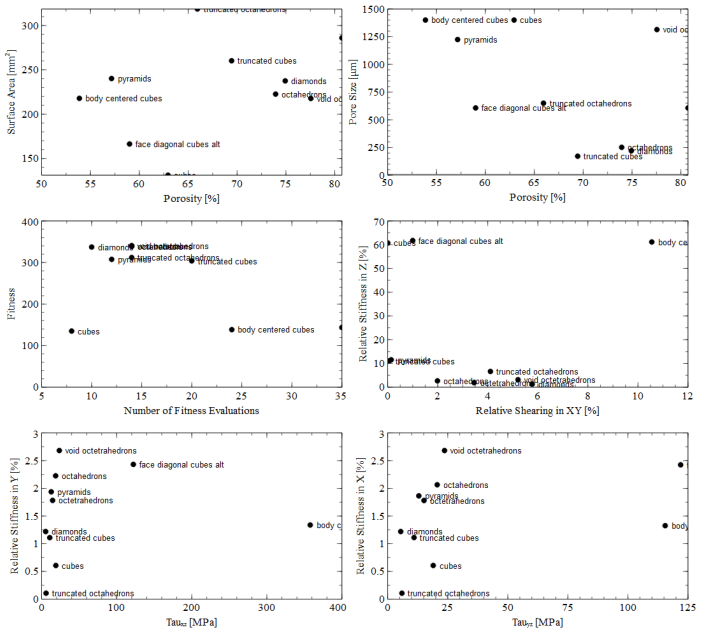


Figure 12.: Optimization results for the MinMax example with three by three by three cells.

Interpretation

As can be seen in Table 5, the optimization resulted in increasing the thickness of vertical struts and decreasing the thickness of the horizontal struts, as is expected. This task was especially well solved by the cubes, body centered cubes and face diagonal cubes topologies. In Figure 12 the properties of the resulted scaffolds for all topologies are depicted. This provides a multidimensional pareto front for interesting properties from which a topology can be chosen.

5.2.2 *Biomimetic Approach*

The optimization goal for the biomimetic approach was set to fit the elastic modulus of the meta-material to the elastic modulus of human cortical bone, namely 15 GPa into longitudinal and 12 GPa into transverse direction [25].

$$\begin{aligned} \min_{x} \quad & f(x) = \tfrac{1}{300}|E_x(x) - 12GPa| + \tfrac{1}{300}|E_y(x) - 12GPa| + \tfrac{1}{300}|E_z(x) - 15GPa| \\ s.t. \quad & g_1(x) = x_1 \geq 0 \\ & g_2(x) = x_2 \geq 0 \\ & \vdots \end{aligned} \tag{13}$$

$f(x)$: Objective Function
$g_n(x)$: Inequality constraint

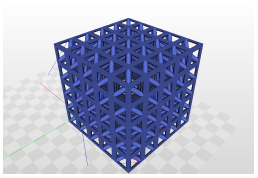
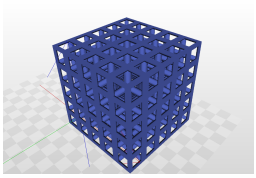
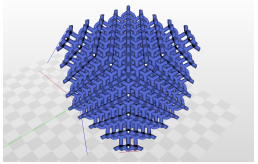
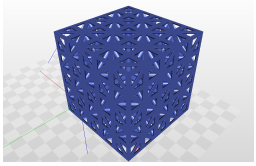
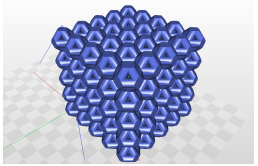
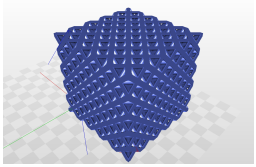
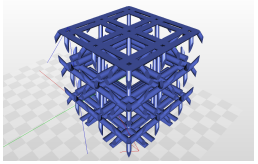
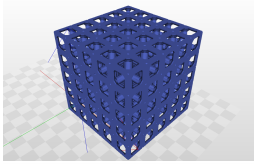
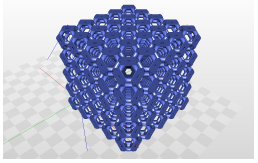
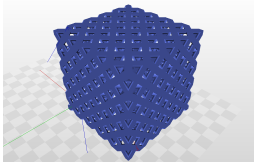
body centered cubes		 cubes
diamonds		 face diagonal cubes
octahedra		 octets
pyramids		 truncated cubes
truncated octahedra		 void octets

Table 6.: Optimization results for the biomimetic approach.

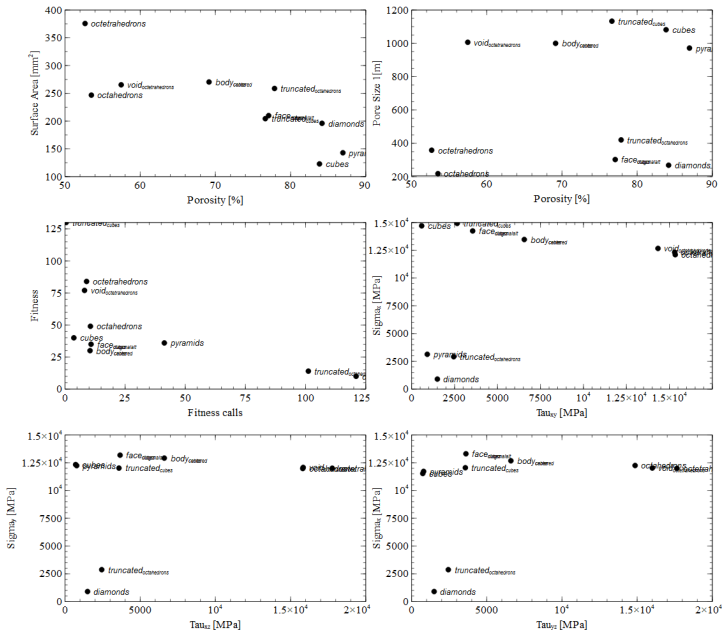


Figure 13.: Optimization results for the biomimetic approach with three by three by three cells.

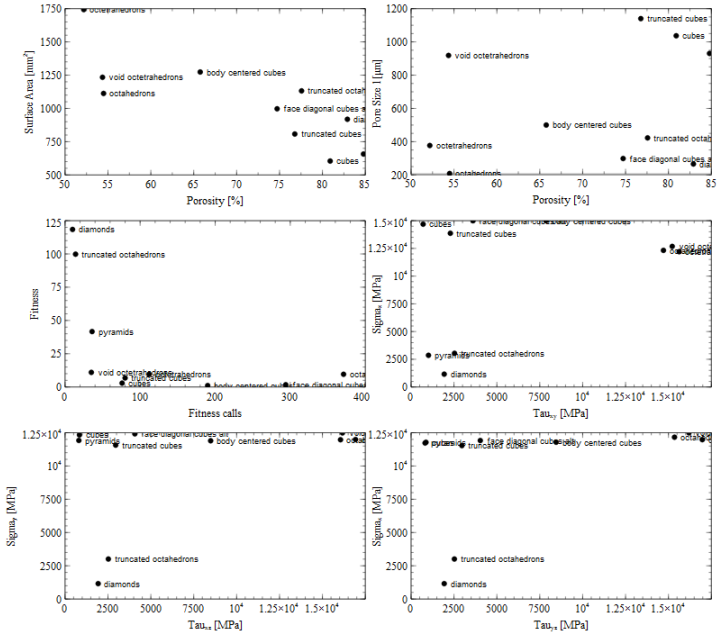


Figure 14.: Optimization results for the biomimetic approach with five by five by five cells.

Interpretation

In Figure 13 and 14 the properties of the resulted scaffolds for all topologies are depicted. As already mentioned in the interpretation of 5.2.1, this provides a multidimensional pareto front for interesting properties from which a topology can be chosen. The resulting scaffolds can be seen in Table 6.

6

DISCUSSION

6.1 VALIDITY OF RESULTS

By using a wire-frame model for the simulation (see 4.2), approximations are applied which deviate the results. The wire-frame simulation with beam elements is compared to an FEM-Model with solid (tetrahedral) elements to show the deviation.

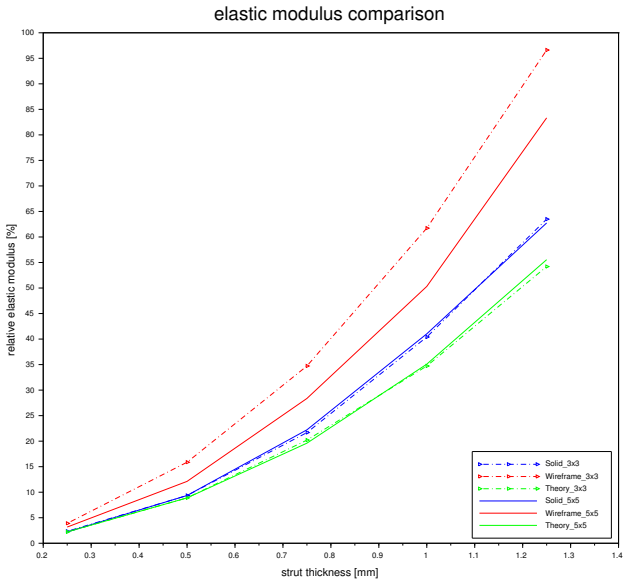


Figure 15.: In this graph the results of different methods to determine the elastic modulus of the meta material with the cubes topology are shown. The dotted lines were evaluated with a scaffold consisting of 3 by 3 by 3 cells. The filled lines show the results with scaffold consisting of 5 by 5 by 5 cells. The green lines show the elastic modulus of only vertical struts within the scaffold with regard of cutting off the border struts according to Figure 16. For the calculation see A.3. The elastic modulus for this theoretical calculation is too low since it does not take the transverse struts into account that would take part of the loading especially for lower porosities (see Figure 17).

In figure 15 it can be seen clearly that the elastic modulus of the scaffold is significantly higher in the wire-frame model than in the solid model.
One reason is that the wire-frame model does not take a cutoff of the border struts into account, which allows the scaffold to overhang and use more space than it should. As can be seen in Figure 16, this error is especially high for a small number of cells in a scaffold.

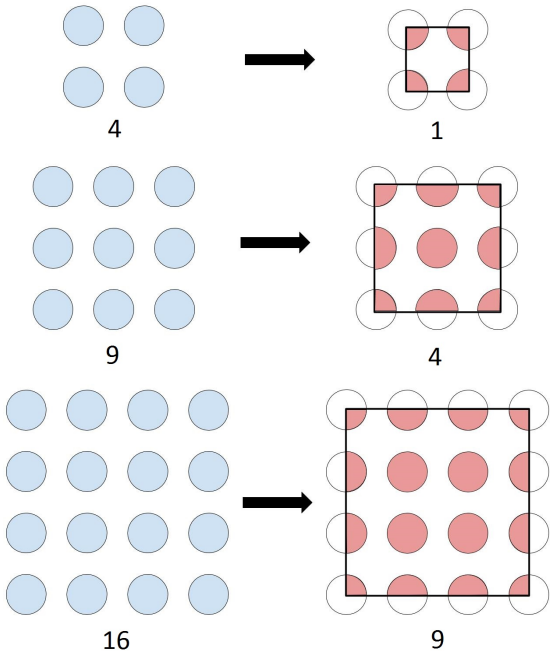


Figure 16.: The stiffness difference of the cubes topology between the cutoff model and the regular model is described by the factor $\frac{i^2}{(i+1)^2}$, where i is the number of cells in the scaffold.

The other reason for a deviation in the results is that beam elements can not fully represent the actual model. It can be seen in Figure 17 that the transverse struts of the solid elements absorb a part of the stress with increasing thickness while the beam elements that are perpendicular to the applied force remain unstressed. Especially for intersections where the angle between two struts is small, the wire-frame approach results in higher stiffnesses because it does not take the intersections between struts into account (see Figure 18). Overall it can be said, that the higher the number of cells and the smaller the strut thicknesses in relation to the cell size, the more accurate is the wire-frame model.

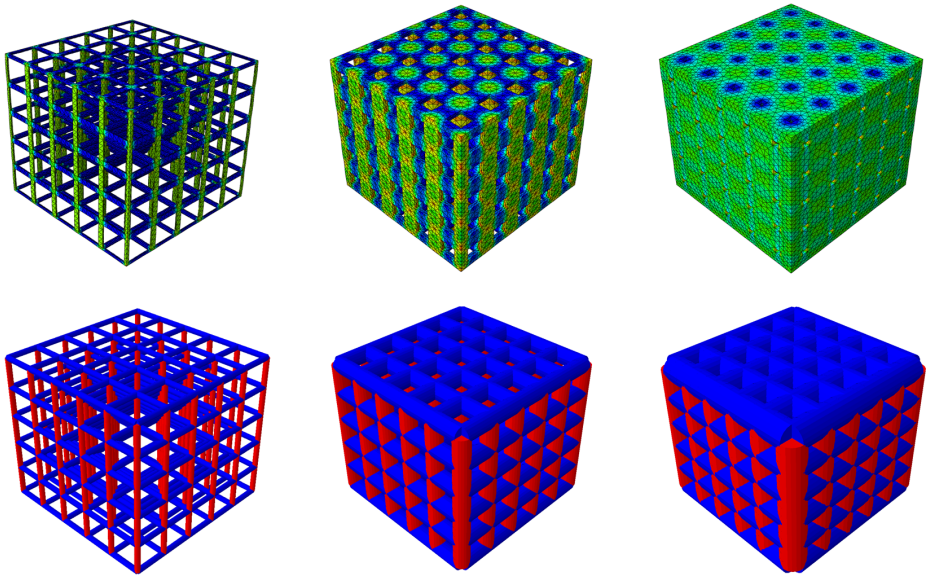


Figure 17.: Comparison of the simulation with solid and with beam elements for different strut thicknesses for the cubes topology. The thicker the struts, the more stress the transverse struts will carry. The wire-frame model does not take this into account.

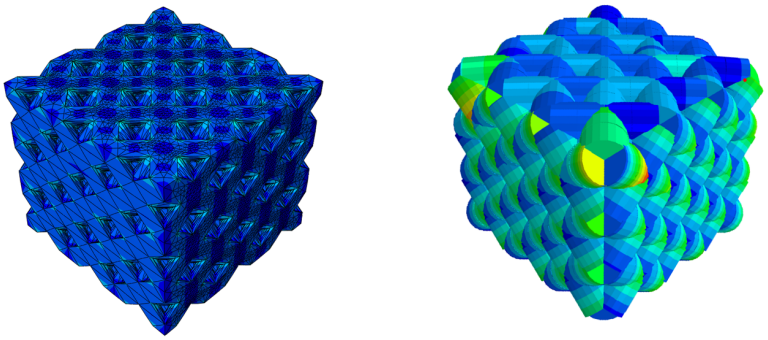


Figure 18.: Comparison of the simulation with solid and with beam elements for the octets topology with low porosity, causing the beam elements to overlap.

6.2 OPTIMIZATION PERFORMANCE

The L-BFGS-B-Algorithm proves to be quite proficient in finding decent results within a reasonable amount of time. The wire-frame model achieves fast results that are within an acceptable error for scaffolds with porosity higher than 60%. Despite the inaccuracy, the wire-frame model does show the correct tendencies for the optimization to step towards descent.

6.3 POSSIBLE FURTHER WORK

6.3.1 *Distortion of cells*

A rather simple method to have further variables to optimize would be to distort any given cell topology into the three main directions.

6.3.2 *Non-cubic cells*

The cell topologies in this thesis are all based on a cubic design to simplify the comparison between them. More complex designs could be evaluated that way. Although it has to be mentioned that if the given optimization goals can be achieved with the current design parameters there is no need for further complexity.

6.3.3 *Combine Cell Topologies*

The overall truss could consist of different topologies to create scaffolds with multiple pore sizes with different Bone-TE-objectives. The bigger pores could offer good interconnectivity to allow nutrient flow while the smaller pores would allow faster bone growth and proliferation.

6.3.4 *Topology Optimization by Bendsøe and Kikuchi*

By allowing struts to be removed once their thickness becomes too small, one could try an optimization similar to the method developed by bendsøe and kikuchi [27].

6.3.5 *Experimental comparison*

The developed scaffolds could be manufactured and tested for their mechanical properties. Additionally, in-vitro and eventually in-vivo experiments would show the actual performance of the scaffolds.

6.3.6 *Other scopes*

Other topics than TE could be investigated further with the presented method. For example, one could use this method on the design of the core of composite sandwich plates.

7

CONCLUSION

In this thesis a method has been introduced to evaluate truss-based scaffolds for their mechanical and volumetric properties. Multiple topologies have been introduced and compared for an example goal to fit the mechanical response of the scaffold to that of human cortical bone for bone-TE.

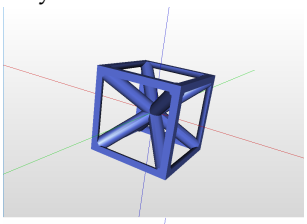
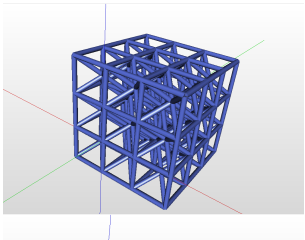
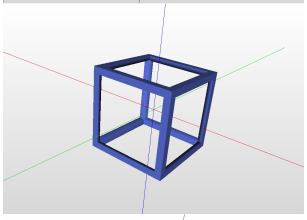
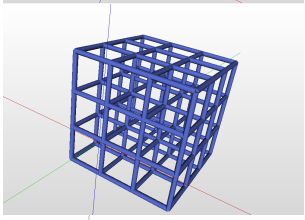
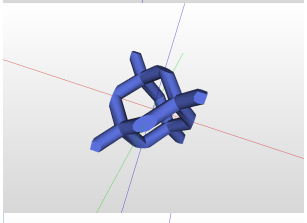
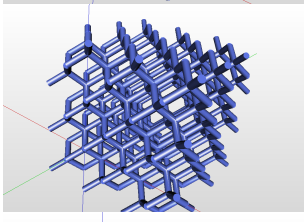
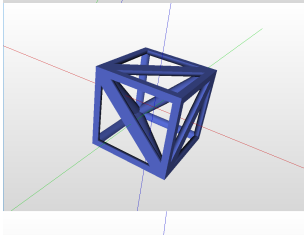
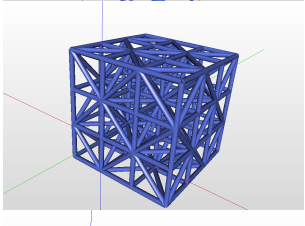
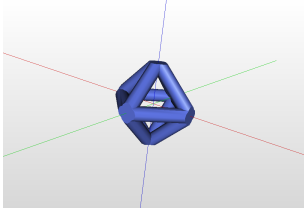
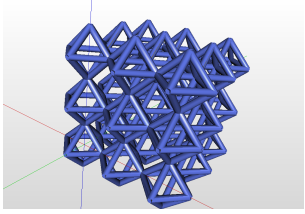
Appendices

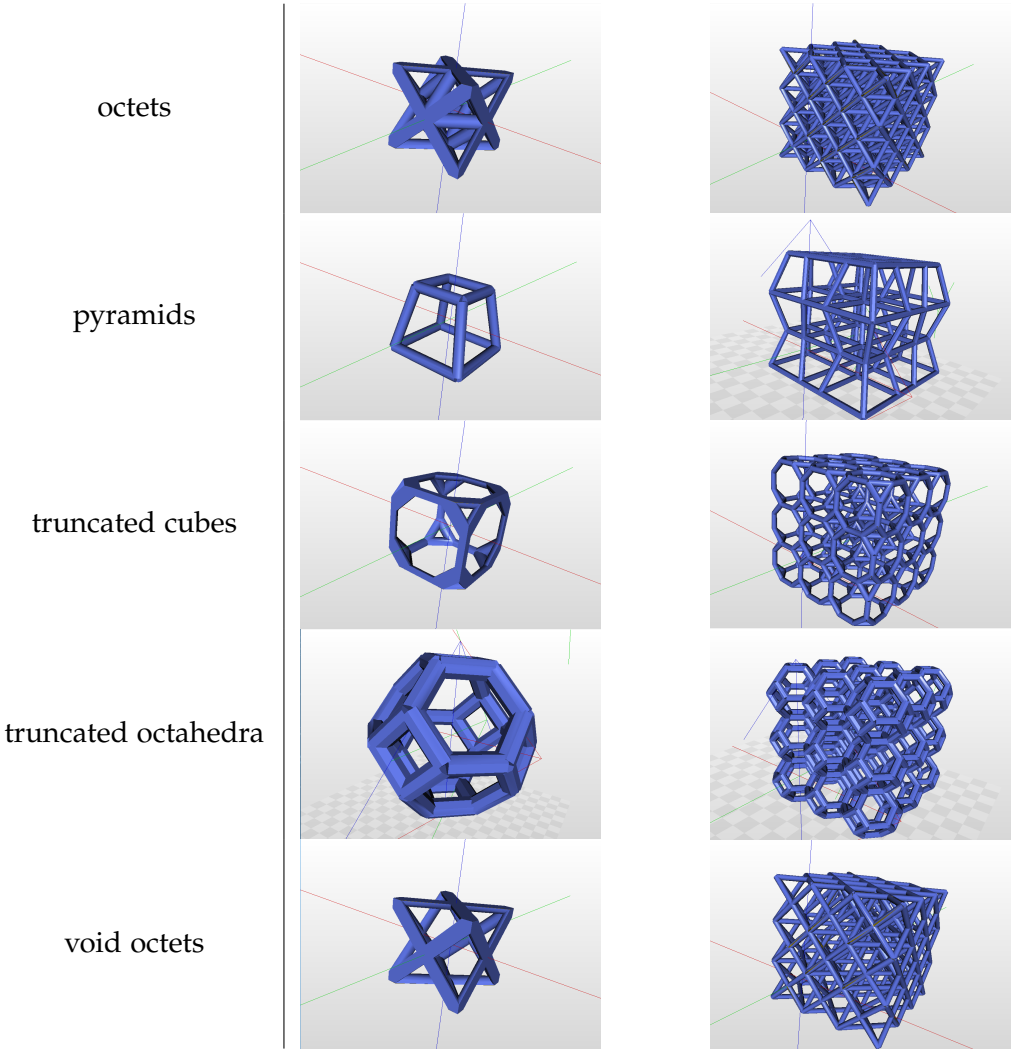
A

APPENDIX

A.1 TOPOLOGIES USED IN THIS THESIS

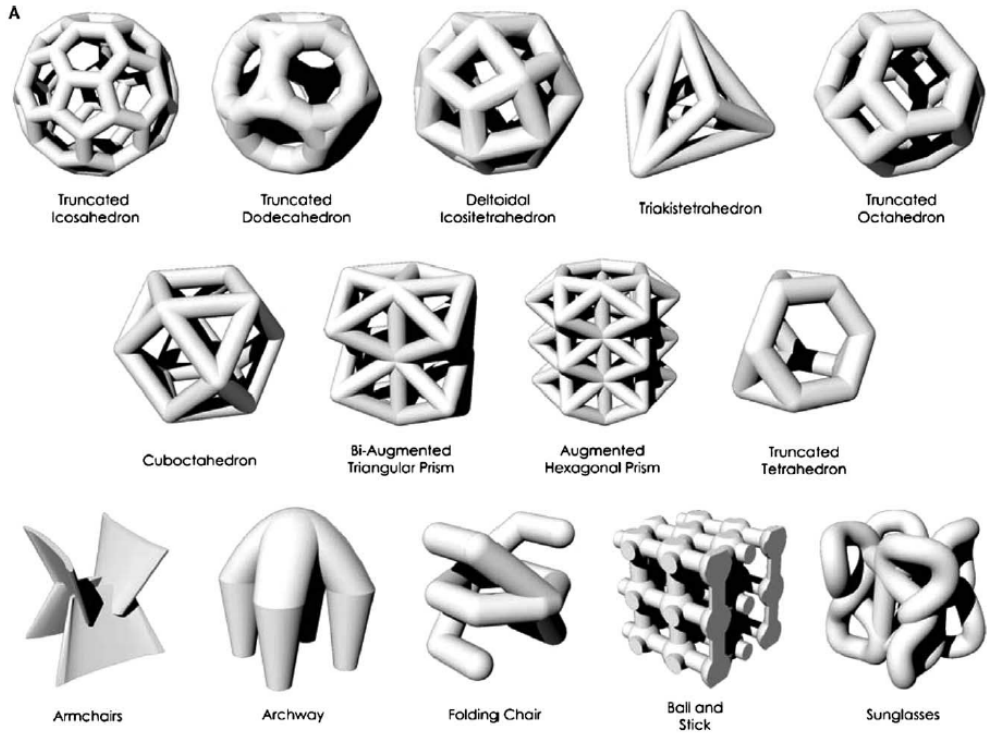
Table 7.: Topologies used in this thesis as a single cell and as a lattice consisting of three by three by three cells.

body centered cubes		
cubes		
diamonds		
face diagonal cubes		
octahedra		

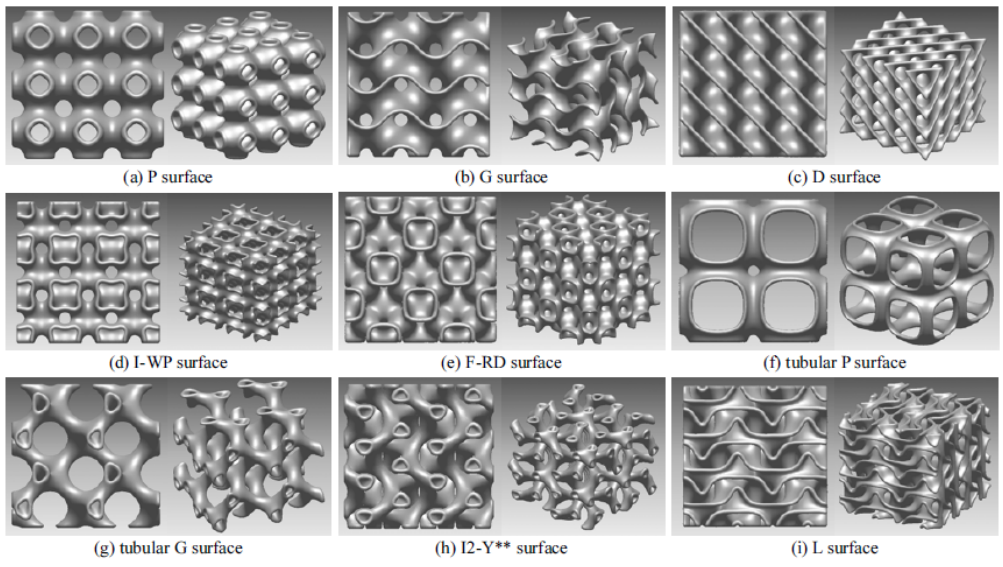

octets
pyramids
truncated cubes
truncated octahedra
void octets

A.2 TOPOLOGIES FROM OTHER RESEARCH

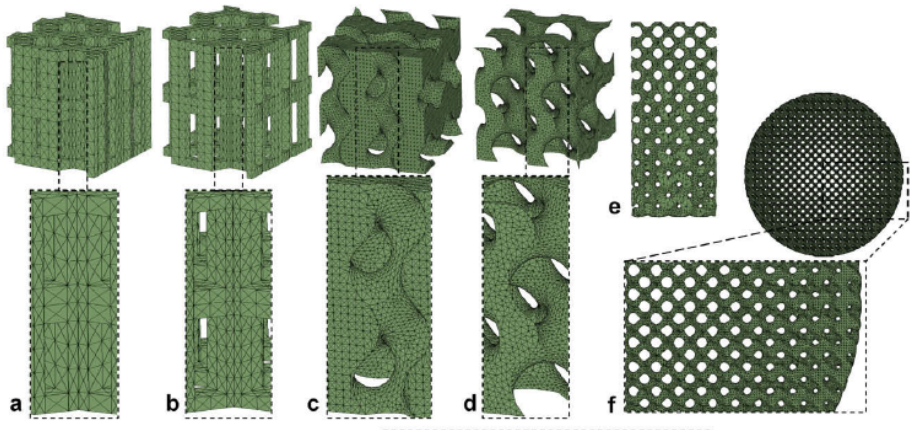
Table 8.: Topologies used in other research.



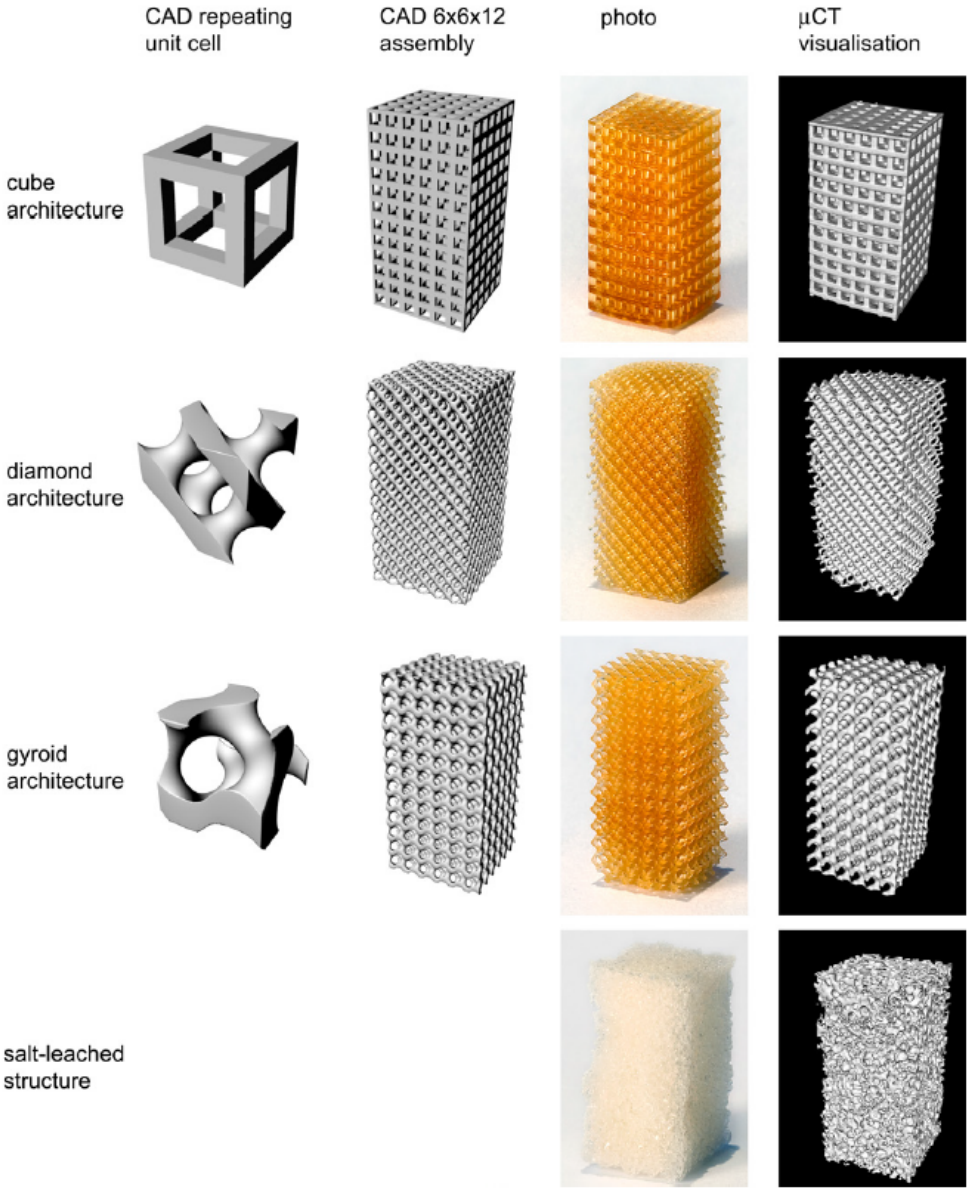
[28]



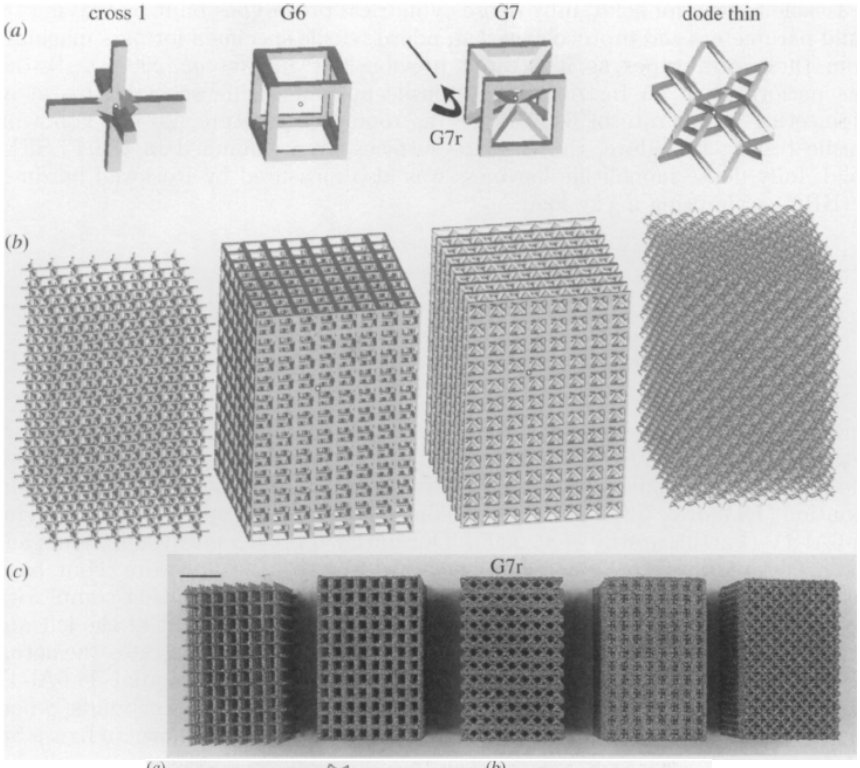
[41]


a
b
c
d
e
f

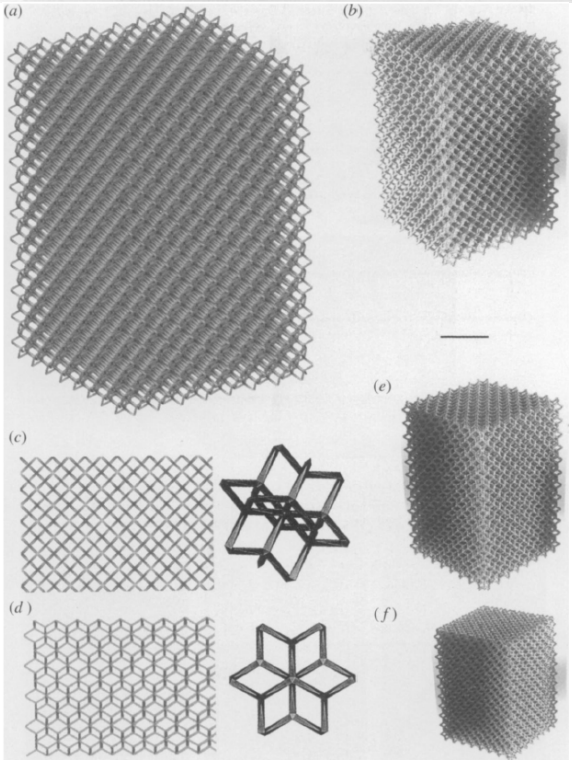
[29]


CAD repeating unit cell
CAD 6x6x12 assembly
photo
µCT visualisation
cube architecture
diamond architecture
gyroid architecture
salt-leached structure

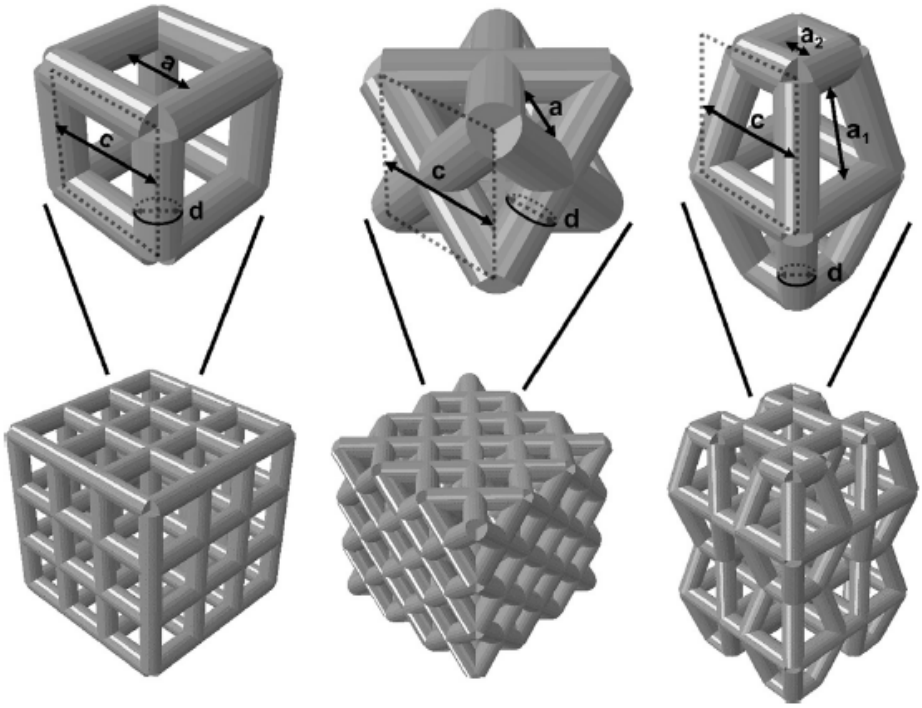
[30]



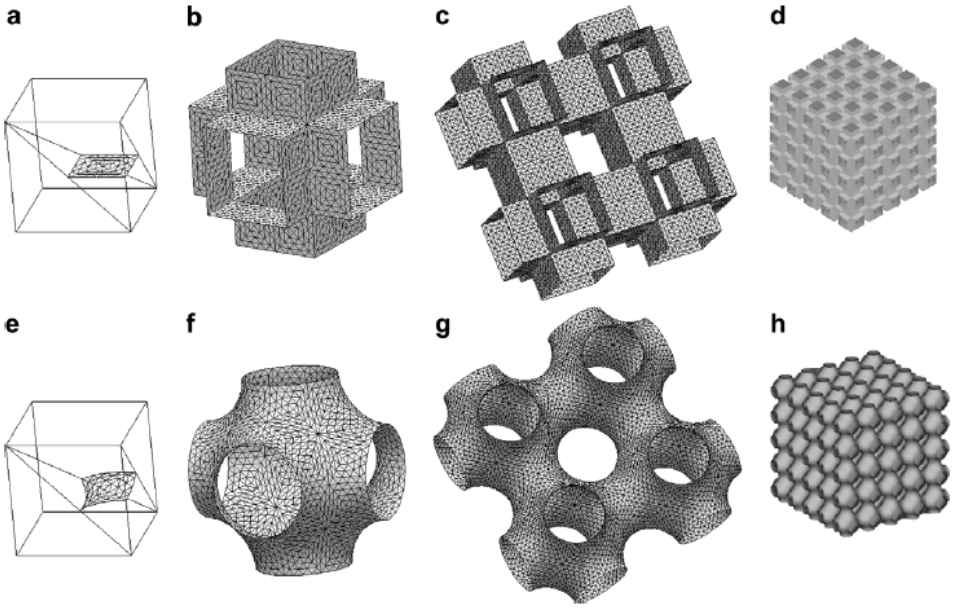
[31]



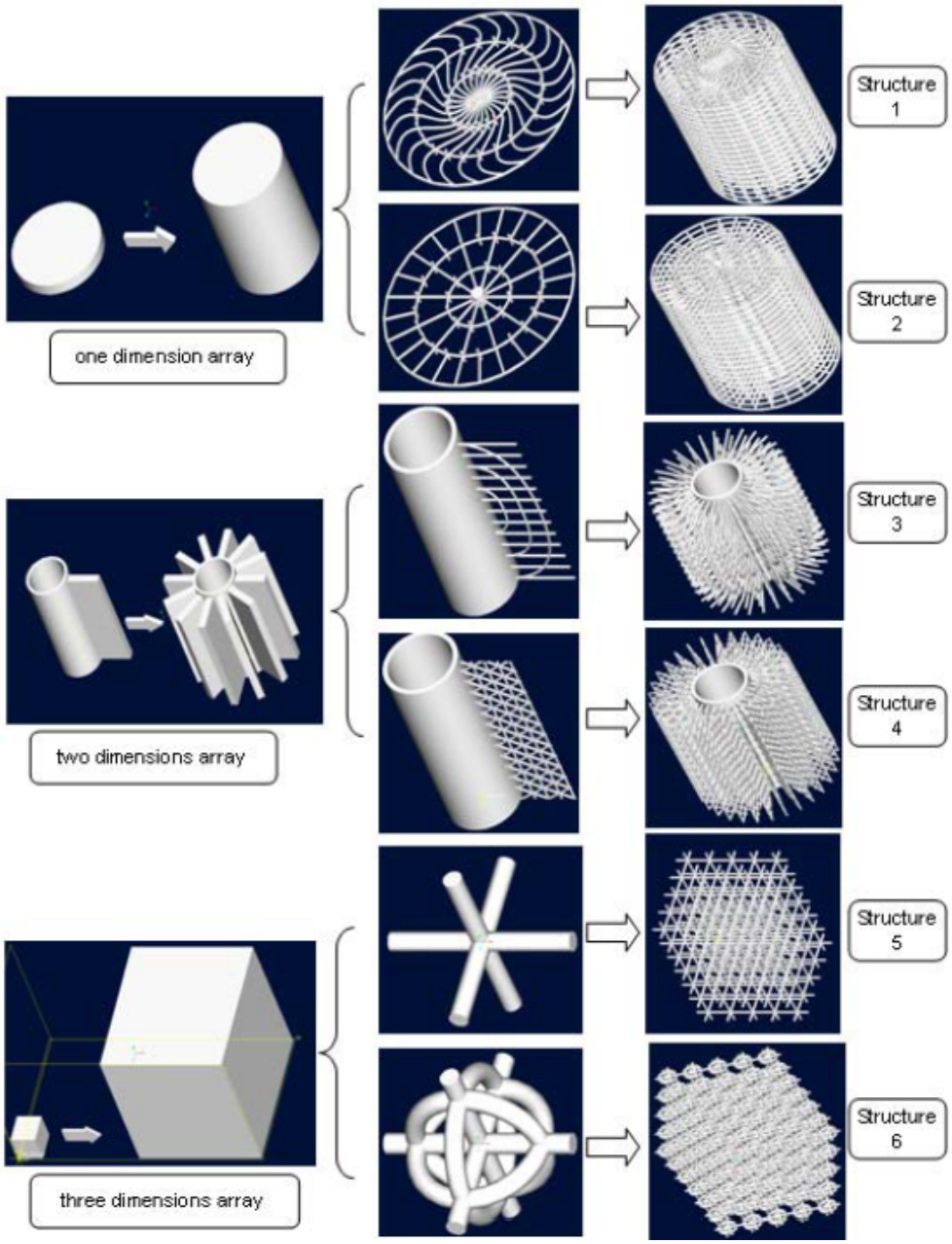
[31]



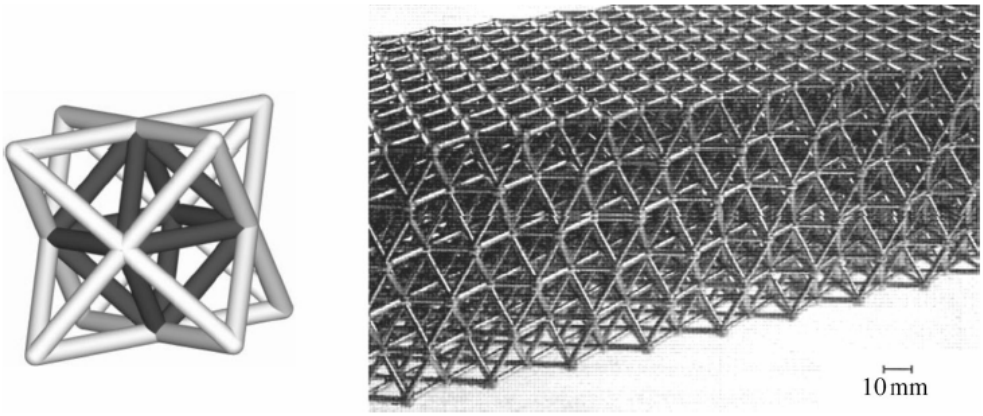
[32]



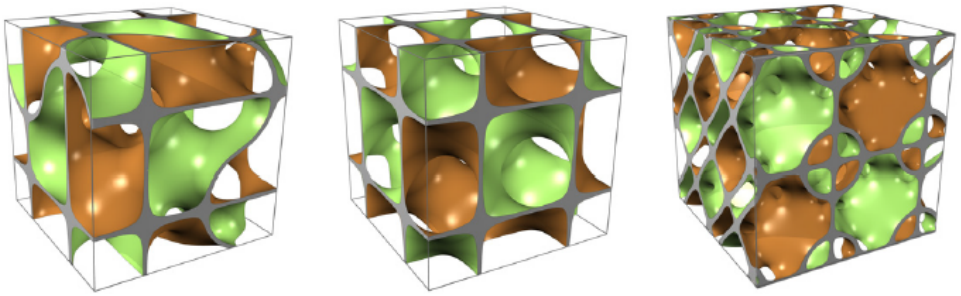
[33]

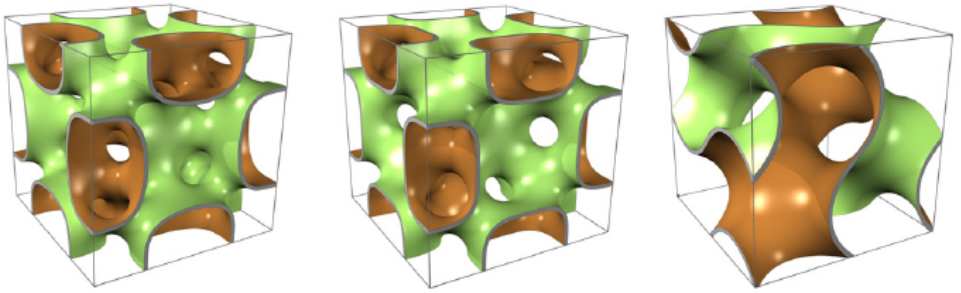

one dimension array
two dimensions array
three dimensions array
Structure 1
Structure 2
Structure 3
Structure 4
Structure 5
Structure 6

[34]

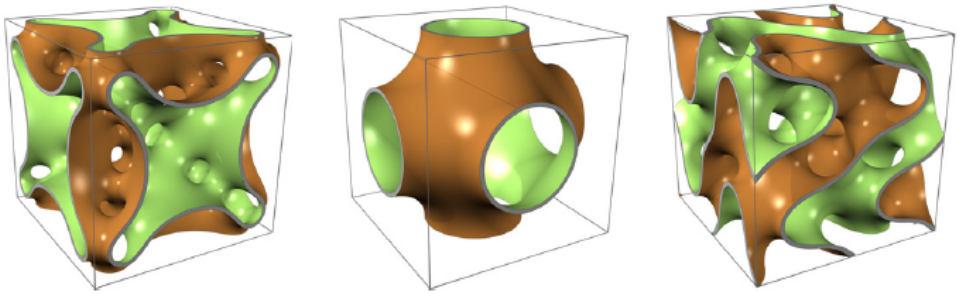


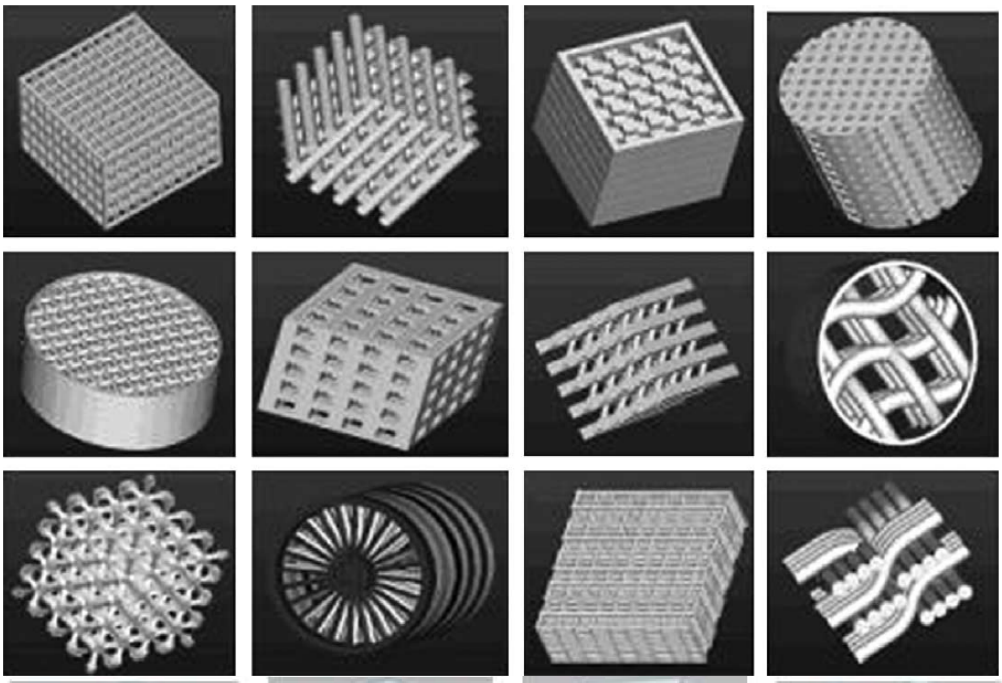
[35]



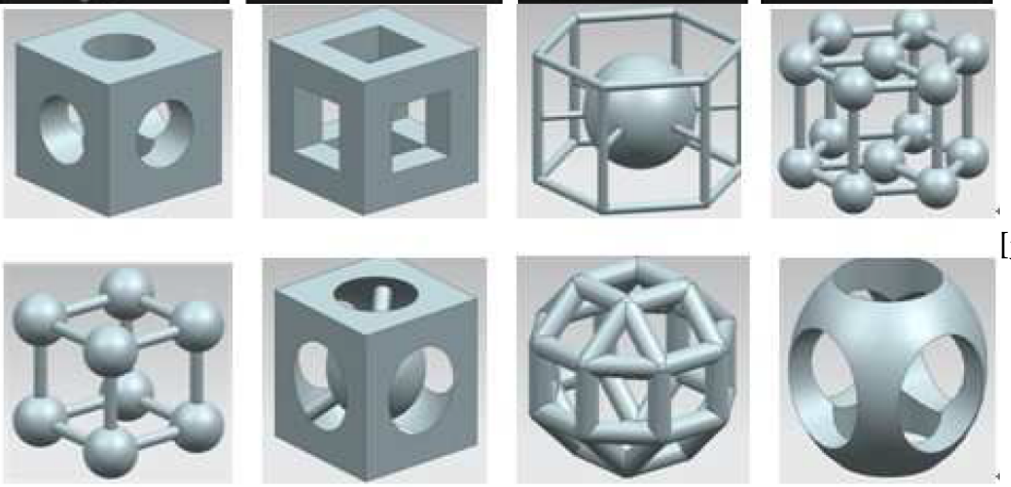


[36]





[37]



[38]

A.3 CALCULATION FOR THEORETICAL STIFFNESS OF META-MATERIAL

A.3.1 *SciLab Code*

```
clc
clear

strut_thickness = [0.25:0.25:1.25]
displacement_solid = [-3.8d-1 -9.5d-2 -4.1d-2 -2.2d-2 -1.4d-2] // -8.9d-3]
displacement_wireframe = [-5.7d-2 -1.4d-2 -6.4d-3 -3.6d-3 -2.3d-3]
displacement_theory = [-0.10 -0.025 -0.011 -0.0064 -0.0041]
displacement_solid = displacement_solid/4
// The force was set at 4N accidental in the simulation.
// Since its all linear elastic this is ok.

scf(0)
clf()
g = gcf()
g.figure_position=[200,200]
g.figure_size=[1000,1000]

subplot(221)
plot(strut_thickness, -displacement_solid/4.5, 'b-.>')
plot(strut_thickness, -displacement_wireframe/4.5, 'r-.>')
plot(strut_thickness, -displacement_theory/4.5, 'g-.>')
hl=legend(['Solid', 'Wireframe', 'Theory'],1)
title("strain 3x3x3", "font_size", 5)
xlabel('strut_thickness [mm]', "font_size", 3)
ylabel('strain [1]', "font_size", 3)

modulus_wireframe = -1*4.5./(4.5^2.*displacement_wireframe)
modulus_solid = -1*4.5./(4.5^2.*displacement_solid)
modulus_theory = -1*4.5./(4.5^2.*displacement_theory)

subplot(222)
plot(strut_thickness, modulus_solid, 'b-.>')
plot(strut_thickness, modulus_wireframe, 'r-.>')
plot(strut_thickness, modulus_theory, 'g-.>')
hl=legend(['Solid', 'Wireframe', 'Theory'],4)
title('elastic modulus 3x3x3', "font_size", 5)
xlabel('strut thickness [mm]', "font_size", 3)
ylabel('relative elastic modulus [%]', "font_size", 3)




strut_thickness5 = [0.25:0.25:1.25]
displacement_solid5 = [-2.4d-1 -5.7d-2 -2.4d-2 -1.3d-2 -0.85d-2] // -8.9d-3]
displacement_wireframe5 = [-4.2d-2 -1.1d-2 -4.7d-3 -2.65d-3 -1.6d-3]
displacement_theory5 = [-0.06 -0.015 -0.0068 -0.0038 -0.0024]
displacement_solid5 = displacement_solid5/4


subplot(223)
plot(strut_thickness5, -displacement_solid5/7.5, 'b-.>')
plot(strut_thickness5, -displacement_wireframe5/7.5, 'r-.>')
plot(strut_thickness5, -displacement_theory5/7.5, 'g-.>')
hl=legend(['Solid', 'Wireframe', 'Theory'],1)
title("strain 5x5x5", "font_size", 5)
xlabel('strut_thickness [mm]', "font_size", 3)
ylabel('strain [1]', "font_size", 3)

modulus_wireframe5 = -1*7.5./(7.5^2.*displacement_wireframe5)
modulus_solid5 = -1*7.5./(7.5^2.*displacement_solid5)
modulus_theory5 = -1*7.5./(7.5^2.*displacement_theory5)
subplot(224)
plot(strut_thickness5, modulus_solid5, 'b-.>')
plot(strut_thickness5, modulus_wireframe5, 'r-.>')
plot(strut_thickness5, modulus_theory5, 'g-.>')
hl=legend(['Solid', 'Wireframe', 'Theory'],4)
title('elastic modulus 5x5x5', "font_size", 5)
xlabel('strut thickness [mm]', "font_size", 3)
```

```
ylabel('relative elastic modulus [%]', "font_size", 3)

scf(1)
clf()
f = gcf()
f.figure_position=[1300,200]
f.figure_size=[1000,1000]
plot(strut_thickness, modulus_solid, 'b-.>')
plot(strut_thickness, modulus_wireframe, 'r-.>')
plot(strut_thickness, modulus_theory, 'g-.>')
plot(strut_thickness5, modulus_solid5, 'b')
plot(strut_thickness5, modulus_wireframe5, 'r')
plot(strut_thickness5, modulus_theory5, 'g')
hl=legend(['Solid_3x3', 'Wireframe_3x3', 'Theory_3x3', 'Solid_5x5',
'Wireframe_5x5', 'Theory_5x5'],4)
title('elastic modulus comparison', "font_size", 5)
xlabel('strut thickness [mm]', "font_size", 3)
ylabel('relative elastic modulus [%]', "font_size", 3)
```

A.4 CODE

A.4.1 *truss_builder.py*

```
from scipy import optimize
from matplotlib import pyplot
from evaluation import objective_function
import pickle
import os
import subprocess
pyplot.ion()


# STARTS OBJECTIVE FUNCTION DEPENDING ON THE VARIABLES TO OPTIMIZE
def optimizer(x, input1, input2):
    # Apparently the optimization toolbox from scipy turns scalar x inputs into lists with 1 element.
    if len(x) == 1:
        input1[options['optimization_variables']] = x[0]
    else:
        input1[options['optimization_variables']] = x
    return objective_function(input1, input2)


# PLOTS x VS y AND SAVES THE PLOT AS A PICTURE
def plot_output(pickle_file_path, x, y, saving_path):
    p_output = pickle.load(open(pickle_file_path, 'rb'))
    x_axis = list()
    y_axis = list()
    for results in p_output:
        x_axis.append(results[str(x)])
        y_axis.append(results[str(y)])
    pyplot.plot(x_axis, y_axis)
    pyplot.xlabel(x)
    pyplot.ylabel(y)
    pyplot.title(saving_path)
    pyplot.grid(True)
    pyplot.savefig(saving_path)
    subprocess.Popen(saving_path, shell=True)


# OPENS THE CSV
def open_csv(output_file_path):
    subprocess.Popen(str(output_file_path), shell=True)

inputs = dict()
bounds = dict()
options = dict()
########################################################################################################################
########################################################################################################################
#  ######################                        INPUT  FROM  USER                       ###################  #
########################################################################################################################
########################################################################################################################

# SMALL LIBRARIES
# Contains the number of different thicknesses on each cell:
truss_thicknesses_library = dict(cubes=3,
                                 inv_cubes=3,
                                 body_centered_cubes=4,
                                 truncated_cubes=9,
                                 face_diagonal_cubes=6,
                                 face_diagonal_cubes_alt=6,
                                 octetrahedrons=6,
                                 octahedrons=6,
                                 void_octetrahedrons=6,
                                 diamonds=4,
                                 templar_crosses=1,
                                 templar_alt_crosses=1,
                                 templar_alt2_crosses=1,
                                 pyramids=3,
                                 file_super_truss=1,
                                 tetroctas=20,
                                 truncated_octahedrons=6)
```

```
69  # Contains the number of different ratios on each cell (for topology optimization):
70  truss_ratio_library =       dict(cubes=0,
71                                   inv_cubes=0,
72                                   body_centered_cubes=0,
73                                   truncated_cubes=1,
74                                   face_diagonal_cubes=0,
75                                   face_diagonal_cubes_alt=0,
76                                   octetrahedrons=0,
77                                   octahedrons=0,
78                                   void_octetrahedrons=0,
79                                   diamonds=0,
80                                   templar_crosses=3,
81                                   templar_alt_crosses=3,
82                                   templar_alt2_crosses=3,
83                                   pyramids=1,
84                                   file_super_truss=0,
85                                   tetroctas=0,
86                                   truncated_octahedrons=1)
87  # Dictionary of materials: E_Modulus: [N/mm^2] = [MPa]
88  material_library =          dict(MED610=dict(name='MED610', E_Modulus=2.5e3, poisson_ratio=0.33),  # Med-grade Polymer
89                                   PLA=dict(name='PLA', E_Modulus=2.5e3, poisson_ratio=0.33),        # Poly-lactic acid
90                                   TCP=dict(name='TCP', E_Modulus=22e3, poisson_ratio=0.33),         # Tricalciumphosphate
91                                   HA=dict(name='HA', E_Modulus=6e3, poisson_ratio=0.33),            # Hydroxyapatite
92                                   Titanium=dict(name='Titanium', E_Modulus=116e3, poisson_ratio=0.32),
93                                   Relative=dict(name='Relative', E_Modulus=100, poisson_ratio=0.33)
94                                   )
95
96  ########################################################################################################################
97  # INPUTS
98  # Directory where abaqus loads and saves files. The directory will be created if it doesn't exist already.
99  # This generated data is rather large:
100 inputs['calculating_directory'] =  "C://abaqus_temp/"
101 # Directory where outputs such as csv and pickle get saved. The directory will be created if it doesn't exist already.
102 # This generated data is very small:
103 inputs['output_directory'] =       'C://Users/maxe/Dropbox/Master_Thesis/outputs/final/superfinal/hypothesis/'
104 # Affix to every file in the calculating directory:
105 inputs['job_name'] =               "temp_output"
106 # Name (and therefore topology) of the truss. Multiple names can be put in to iterate through all.
107 # For a single input, put it in brackets. For example: ['cubes']
108 # inputs['truss_names'] =             ['body_centered_cubes']  # , 'diamonds', 'truncated_cubes']
109 inputs['truss_names'] =             ['cubes', 'octetrahedrons']
110 # inputs['truss_names'] =             ['cubes', 'body_centered_cubes', 'truncated_cubes', 'face_diagonal_cubes_alt',
111 #                                      'octetrahedrons', 'octahedrons', 'void_octetrahedrons', 'diamonds', 'pyramids',
112 #                                      'truncated_octahedrons']
113
114 # Material. See material_library to add or see materials.
115 inputs['material'] =                            material_library['Relative']
116
117 # Cell topology
118 # Number of cells in one direction (Total number of cells is number_of_cells ** 2):
119 inputs['number_of_cells'] =                     5
120 # Length of one cell (Total size is cell_size * number_of_cells):
121 inputs['cell_size'] =                           1  # [mm]
122 # Minimal Thickness that can be used:
123 inputs['strut_min_thickness'] =                 0.1  # [mm]
124 # A list will be created inputs['strut_thickness_multiplicator'] that has the needed length depending on cell topoology
125 # This lists elements will all be inputs['strut_min_thickness'] * inputs['strut_thickness_multiplier']
126 inputs['strut_thickness_multiplier'] =          1
127 inputs['cell_ratio_multiplier'] =               0.5
128
129 ########################################################################################################################
130 ########################################################################################################################
131 # OPTIONS
132
133 # Creates the loading steps for the wireframe evaluation:
134 options['create_steps'] =                       True
135 # Submits the job and evaluates the created steps:
136 options['submit_job'] =                         True
137 # Generates the solid model of the truss and exports it into an stl-file. Needed to calculate volumetric outputs:
138 options['stl_generate'] =                       True
139 # Cuts the borders of the truss off:
140 options['cutoff'] =                             True
141 # Runs the program with the GUI (Generated User Interface). This blocks submitting of the job:
142 options['gui'] =                                False
143 # Overwrites the csv result_file instead of appending the new results
```

```
144  # The result gets saved in the output_directory
145  # If there hasn't been an output file yet, this needs to be True:
146  options['overwrite_csv'] =                          True
147  # Overwrites the pickled result file instead of appending the new results
148  # (The result gets saved in the output_directory
149  # If there hasn't been an output file yet, this needs to be True:
150  options['overwrite_pickle'] =                       True
151
152  # Viewers
153  # Opens the stl (STereoLithography) after completion:
154  options['stl_view'] =                               False
155  # Opens the odb (Output DataBase) after completion:
156  options['odb_view'] =                               False
157  # Opens the csv (Comma Separated Values) of the output after completion:
158  options['csv_view'] =                               False
159
160  # Cross Section Input
161  # Cross Section of the Struts for the Solid and Stl Model. Can be "square", "hexagon", "octagon" or "dodecagon":
162  options['strut_cross_section'] =                    'dodecagon'
163  # Version Input
164  # To determine Version: use windows command prompt: "abaqus cae nogui" and then ">>> version" or ">>> print(version)"
165  # If the student version is used add SE in the end. F.e. '6.14-2SE'
166  # This is used for defining the path of where to find the abaqus_plugin stlExport. See class Script : __init__()
167  options['abaqus_version'] =                         '6.14-1'
168
169  # File Path
170  # This is usually C:/SIMULIA/Abaqus/
171  options['abaqus_path'] =                            "C:/Program Files/Abaqus/"
172  ######################################################################################################################
173  # METHOD to run the engine. Possible entries are 'single_run', 'loop' or 'optimization'
174  options['method'] =                                 'loop'
175
176  # LOOP specific (applies for options['method'] == 'loop')
177  # Loop over first variable:
178  options['loop1_variable'] =                          'number_of_cells'
179  # List of all the values to evaluate
180  options['loop1_values'] =                            [a for a in range(1, 6)]
181
182
183  # Optional loop over second variable:
184  # Use 'None' if there is to be no loop
185  options['loop2_variable'] =                         'None'
186  # Use [0] if there is to be no loop
187  options['loop2_values'] =                           [0]
188
189  # OPTIMIZATION specific
190  #  Decides what to optimize for:
191  options['optimization_variables'] =                 'strut_thickness_multiplicator'
192  # Bounds of the optimization variables. This only applies to some algorithms.
193  options['bounds'] =                                 (1, 20)
194  # Define Fitness variables, their target value, their norming and their weighting.
195  # options['fitness_variables'] =                     {'Sigma_z': [30e3, 1e-2, 1]}
196  options['fitness_variables'] =                      {'Sigma_x': [12e3, 1e-2, 1],
197                                                       'Sigma_y': [12e3, 1e-2, 1],
198                                                       'Sigma_z': [15e3, 1e-2, 1]}
199  # Plot the fitness while optimizing:
200  options['plot_fitness'] =                           True
201  # Defines the algorithm used for the optimization:
202  options['algorithm'] =                              'L-BFGS-B'
203  # Possible algorithms:
204  # 'Nelder-Mead', 'Powell', 'CG', 'BFGS', 'Newton-CG', 'L-BFGS-B', 'TNC', 'COBYLA', 'SLSQP', 'dogleg', 'trust-ncg'
205  # Options for the optimization. See scipy optimization toolbox for further info:
206  options['options'] =                                {'disp': True, 'eps': 0.2, 'ftol': 0.2}
207  # Reruns the results with generating stl and writing all results into one csv:
208  options['run_results'] =                            True
209  ######################################################################################################################
210  # OUTPUT
211
212  # Decide what to save in the csv
213  options['output'] = dict()
214  # General:
215  options['output']['Step'] =                         True
216  options['output']['Truss_Name'] =                   True
217  options['output']['Fitness'] =                      True
218  # Geometric properties:
```

```
219 options['output']['Cell_size'] =                    True
220 options['output']['Strut_Thickness'] =              True  # =Strut_Thicknesses[0]
221 options['output']['Pore_size'] =                    True
222 # Mechanic properties:
223 options['output']["Young's Modulus"] =             True
224 options['output']['Shearing Modulus'] =            True
225 options['output']["Poisson's Ratio"] =             True
226 # Volumetric properties: (will only be calculated if the STL is generated)
227 options['output']['Volume'] =                      True
228 options['output']['Porosity'] =                    True
229 options['output']['Void_ratio'] =                  True
230 options['output']['Surface_Area'] =                True
231 # Optimization properties:
232 options['output']['X'] =                           True  # Strut_Thickness_Multiplicator
233
234 ##########################################################################################################################
235 ##########################################################################################################################
236 #  ####################                          NO MORE INPUT                          ######################  #
237 ##########################################################################################################################
238 ##########################################################################################################################
239 if options['gui']:
240     options['submit_job'] = False
241     print("Job will not be submitted if the Script is run with the GUI.")
242 if not options['stl_generate']:
243     options['stl_view'] = False
244     print("Stl cannot be viewed if it is not generated")
245 if not options['submit_job']:
246     options['odb_view'] = False
247     print("Odb cannot be viewed if the job is not submitted")
248
249 options['read_output'] = options['submit_job']
250
251 # Create folders if they don't exist yet.
252 if not os.path.exists(inputs['calculating_directory']):
253     os.mkdir(inputs['calculating_directory'])
254
255 if not os.path.exists(inputs['output_directory']):
256     os.mkdir(inputs['output_directory'])
257
258 if options['method'] == 'optimization':
259     # Initialize csv
260     file = open(inputs['output_directory'] + "optimization_results.csv", 'w')
261     file.write("x, fun, nit, success, nfev\n")
262     file.close()
263     # Initialize pickle file
264     pickle.dump(list(), open(inputs['output_directory'] + "pickled_results", 'wb'))
265
266 # Loop over the different truss topologies:
267 for name in inputs['truss_names']:
268     try:
269         inputs['truss_name'] = name
270
271         bounds[options['optimization_variables']] = [options['bounds']]
272
273         inputs['strut_thickness_multiplicator'] =         list()
274         # Multiplied by strut_min_thickness results in the actual thickness of the struts
275         bounds['strut_thickness_multiplicator'] =         list()
276         # Multiplied by strut_min_thickness results in the actual thickness of the struts bounds
277         for i in range(0, truss_thicknesses_library[name]):
278             inputs['strut_thickness_multiplicator'].append(inputs['strut_thickness_multiplier'])
279             bounds['strut_thickness_multiplicator'].append(options['bounds'])
280         # Best thicknesses for titanium (1.5mm cell_size)(0.1mm strut_min_thickness) (3x3x3)
281         # inputs['strut_thickness_multiplicator'] =         [4.71177218, 4.14965115, 4.80241845]
282
283         inputs['cell_ratio'] =                            list()
284         bounds['cell_ratio'] =                            list()
285         for j in range(0, truss_ratio_library[name]):
286             inputs['cell_ratio'].append(inputs['cell_ratio_multiplier'])
287             bounds['cell_ratio'].append(options['bounds'])
288         # Best ratio for templar_alt2_crosses:
289         # inputs['cell_ratio'] =                              [0.3, 0.8, 0.2]
290         # printable:
291         # inputs['cell_ratio'] =                              [0.5, 0.8, 0.35]
292
293         inputs['output_file'] = inputs['output_directory'] + 'optim_output_' + str(name)
```

```
294
295         if options['overwrite_csv']:
296             result_file = open(str(inputs['output_file']) + '.csv', 'w')
297             if options['output']['Step']:
298                 result_file.write('Step, ')
299             if options['output']['Truss_Name']:
300                 result_file.write('Truss Name, ')
301             if options['output']['Fitness']:
302                 result_file.write('Fitness, ')
303             if options['output']['Cell_size']:
304                 result_file.write('Cell Size [mm], ')
305             if options['output']['Strut_Thickness']:
306                 result_file.write('Strut Thickness [mu-m], ')
307             if options['output']['Pore_size']:
308                 result_file.write('Pore Size 1[mu-m], Pore Size 2[mu-m], Pore Size 3[mu-m], Pore Size 4[mu-m], ')
309             if options['output']["Young's Modulus"]:
310                 result_file.write('Sigma_z [MPa], Sigma_y [MPa], Sigma_x [MPa], ')
311             if options['output']['Shearing Modulus']:
312                 result_file.write('Tau_yz [MPa], Tau_xz [MPa], Tau_xy [MPa], ')
313             if options['output']["Poisson's Ratio"]:
314                 result_file.write('v21 [1], v31 [1], v32 [1], ')
315             if options['output']['Volume']:
316                 result_file.write('Volume [mm^3], ')
317             if options['output']['Porosity']:
318                 result_file.write('Porosity [%], ')
319             if options['output']['Void_ratio']:
320                 result_file.write('Void_ratio [1], ')
321             if options['output']['Surface_Area']:
322                 result_file.write('Surface_Area [mm^2], ')
323             if options['output']['X']:
324                 result_file.write('Strut_Thickness_Multiplicator [1], ')
325             result_file.write('\n')
326             result_file.close()
327
328         if options['run_results']:
329             result_file2 = open(inputs['output_directory'] + "best_results" + '.csv', 'w')
330             if options['output']['Step']:
331                 result_file2.write('Step, ')
332             if options['output']['Truss_Name']:
333                 result_file2.write('Truss Name, ')
334             if options['output']['Fitness']:
335                 result_file2.write('Fitness, ')
336             if options['output']['Cell_size']:
337                 result_file2.write('Cell Size [mm], ')
338             if options['output']['Strut_Thickness']:
339                 result_file2.write('Strut Thickness [mu-m], ')
340             if options['output']['Pore_size']:
341                 result_file2.write('Pore Size 1[mu-m], Pore Size 2[mu-m], Pore Size 3[mu-m], Pore Size 4[mu-m], ')
342             if options['output']["Young's Modulus"]:
343                 result_file2.write('Sigma_z [MPa], Sigma_y [MPa], Sigma_x [MPa], ')
344             if options['output']['Shearing Modulus']:
345                 result_file2.write('Tau_yz [MPa], Tau_xz [MPa], Tau_xy [MPa], ')
346             if options['output']["Poisson's Ratio"]:
347                 result_file2.write('v21 [1], v31 [1], v32 [1], ')
348             if options['output']['Volume']:
349                 result_file2.write('Volume [mm^3], ')
350             if options['output']['Porosity']:
351                 result_file2.write('Porosity [%], ')
352             if options['output']['Void_ratio']:
353                 result_file2.write('Void_ratio [1], ')
354             if options['output']['Surface_Area']:
355                 result_file2.write('Surface_Area [mm^2], ')
356             if options['output']['X']:
357                 result_file2.write('Strut_Thickness_Multiplicator [1], ')
358             result_file2.write('\n')
359             result_file2.close()
360
361         if options['overwrite_pickle']:
362             output_old = list()
363             file = open(inputs['output_file'] + "_pickle", 'wb')
364             pickle.dump(output_old, file)
365             file.close()
366
367         # Counts the number of function evaluations. Careful, this is global
368         universal_counter = 0
```

```
369         ########################################################################################################
370         # Calling the engine, depending on the chosen method
371         if options['method'] == 'single_run':
372             options['plot_fitness'] = False
373             objective_function(inputs=inputs, options=options)
374 
375         elif options['method'] == 'loop':
376 
377             options['plot_fitness'] = False
378             for loop1 in options['loop1_values']:
379                 for loop2 in options['loop2_values']:
380                     inputs[options['loop1_variable']] = loop1
381                     inputs[options['loop2_variable']] = loop2
382                     # SPECIAL RUN
383                     inputs['cell_size'] = 5 / inputs['number_of_cells']
384                     inputs['strut_min_thickness'] = inputs['cell_size'] / 5
385                     # END SPECIAL RUN
386                     objective_function(inputs=inputs, options=options)
387 
388         elif options['method'] == 'optimization':
389             print("Optimization Input:")
390             print("xo= " + str(inputs[options['optimization_variables']]))
391             print("inputs= " + str(inputs) + ", " + str(options))
392             print("options= " + str(options))
393             print("method= " + str(options['algorithm']))
394             print("bounds= " + str(bounds[options['optimization_variables']]))
395             print("options= " + str(options['options']))
396             print("########################################################################################################\n\n\n")
397 
398             result = optimize.minimize(optimizer,
399                                        x0=inputs[options['optimization_variables']],
400                                        args=(inputs,
401                                              options),
402                                        method=options['algorithm'],
403                                        bounds=bounds[options['optimization_variables']],
404                                        options=options['options'])
405 
406             # Save result as CSV:
407             file = open(inputs['output_directory'] + "optimization_results.csv", 'a')
408             key_list = ['x', 'fun', 'nit', 'success', 'nfev']
409             for key in key_list:
410                 file.write(str(result[key]) + ", ")
411             file.write("\n")
412             file.close()
413 
414             # Save result as pickled file:
415             result_pickle = {'inputs': inputs, 'options': options,
416                              'optimization': {'x': result['x'], 'nfev': result['nfev'],
417                                               'nit': result['nit'], 'success': result['success']}}
418             print(result_pickle['optimization'])
419             output_old = pickle.load(open(inputs['output_directory'] + "pickled_results", 'rb'))
420             output_old.append(result_pickle)
421             file = open(inputs['output_directory'] + "pickled_results", 'wb')
422             pickle.dump(output_old, file)
423             file.close()
424 
425             print("########################################################################################################\n\n\n")
426         else:
427             print("options['method'] does not contain a valid keyword. possible entries are: "
428                   "'single_run', 'loop' or 'optimization'.")
429         ########################################################################################################
430 
431         if options['csv_view']:
432             open_csv(str(inputs['output_file']) + '.csv')
433     except FileNotFoundError:
434         print("Abaqus faced an error.\nContinuing with the other topologies.")
435 
436 # Run the results with calculating volumetric properties
437 if options['method'] == 'optimization' and options['run_results']:
438     output = pickle.load(open(inputs['output_directory'] + "pickled_results", 'rb'))
439     print("Output: " + str(output))
440     for i in range(0, len(output)):
441         inputs = output[i]['inputs']
442         options = output[i]['options']
443         inputs[options['optimization_variables']] = output[i]['optimization']['x']
```

```
444         # inputs['number_of_cells'] = 3
445         inputs['output_file'] = inputs['output_directory'] + "best_results"
446         options['stl_generate'] = True
447         options['cutoff'] = True
448
449         # Initialize best_results_pickle
450         file = open(inputs['output_file'] + "_pickle", 'wb')
451         pickle.dump(list(), file)
452         file.close()
453
454         objective_function(inputs=inputs, options=options)
```

A.4.2 *evaluation.py*

```
"""
evaluation.py
This file contains the main steps of the program such as initiation of optimization values, initiation of
optimization and generation of the stl for the final solution.
Each solution is an object with the following properties:
Nodal points, start and finish of every strut, strut thickness of each strut, pore size, porosity, surface area
Each solution might have the following properties:
Material constants, Cell types (which are repeated through the truss), cell size, thickness of each strut of a cell
"""
import os
import subprocess
import time
import pickle
from library_truss import generate_truss
from Class_Script import Script
import statistics
from matplotlib import pyplot
import warnings
import matplotlib.cbook
import numpy

warnings.filterwarnings("ignore", category=matplotlib.cbook.mplDeprecation)


########################################################################################################################
# HEADER


# RUN THE ABAQUS SCRIPT VIA COMMAND LINE
def run(script, gui):
    print("Running Abaqus Script")
    t0 = time.time()
    os.chdir(str(script.filename[0]))

    if gui:
        subprocess.run('abaqus cae script=' + "".join(script.filename), shell=True)
    else:
        subprocess.run('abaqus cae noGUI=' + "".join(script.filename), shell=True)
    t1 = time.time()
    print("Time elapsed: " + str(t1 - t0) + " seconds\n")


# OPENS THE ABAQUS ODB VIEWER VIA COMMAND LINE
def odb_viewer(script):
    print("Starting Abaqus Viewer")
    subprocess.Popen('abaqus viewer database=' + script.filename[0] + script.filename[1], shell=True)


# OPENS THE STL FILE WITH SELECTED DEFAULT PROGRAM VIA COMMAND LINE
def stl_viewer(script):
    print("Starting STL Viewer")
    subprocess.Popen(script.filename[0] + script.filename[1] + ".stl", shell=True)


# READS IN THE DISPLACEMENTS OF THE SIDES OF THE CUBES FOR EACH STEP/LOADING CONDITION AND CALCULATES COMPLIANCE FROM IT
def read_results(script, x):
    def read_stress(result, name, plane, direction):
        list_of_coordinates = list()
        for point in result[name][plane]:
            list_of_coordinates.append(float(point[direction]))
        displacement = statistics.median(list_of_coordinates)
        # print("Average displacement " + name + ": " + str(round(displacement * 1e3, 3)) + " µm")
        young = abs(applied_force / ((script.truss.cell_size * script.truss.number_of_cells +
                                      script.truss.cells[0].strut_thicknesses[0]) * displacement))
        # print("Elastic Modulus in " + name + " step into " + plane +
        #       " direction: " + str(round(young / 1e6, 3)) + " MPa")
        return young

    def read_shearing(result, name, plane, direction):
        list_of_coordinates = list()
        for point in result[name][plane]:
```

```
72              list_of_coordinates.append(float(point[direction]))
73          displacement = statistics.median(list_of_coordinates)
74          # print("Average displacement " + name + ": " + str(round(displacement * 1e6, 3)) + " µm")
75          shear = abs(applied_force / ((script.truss.cell_size * script.truss.number_of_cells +
76                                        script.truss.cells[0].strut_thicknesses[0]) * displacement))
77          return shear
78
79      def read_displacement(result, step, plane, direction):
80          list_of_coordinates = list()
81          for point in result[step][plane]:
82              list_of_coordinates.append(float(point[direction]))
83          displacement = statistics.median(list_of_coordinates)
84          # print("Average displacement " + step + ": " + str(round(displacement * 1e6, 6)) + " µm")
85          return displacement
86
87      # IMPORT SOLUTION
88      with open(str(script.filename[0]) + str(script.filename[1]) + '_results', 'rb') as f:
89          u = pickle._Unpickler(f)
90          u.encoding = 'latin1'
91          results = u.load()
92          # results = pickle.load(f)
93      applied_force = 1  # [N]. This is also defined in Class_Script.py : def evaluate()
94
95      output = dict()
96      output['X'] = x
97      output['Cell_Size'] = script.truss.cell_size
98      output['Strut_Thickness'] = script.truss.cells[0].strut_thicknesses[0]
99      output['Truss_Name'] = script.truss.name
100     output['Pore_size'] = list()
101     for cell in script.truss.cells:
102         output['Pore_size'].append(cell.pore_size)
103
104     try:  # This outputs volumetrics such as porosity, which is only possible when the solid is generated in abaqus.
105         porosity = 1 - (results['Volume'] / ((script.truss.number_of_cells * script.truss.cell_size) ** 3))
106         print("Porosity: " + str(round(porosity * 1e2, 1)) + "%")
107         print("Void Ratio: " + str(round(porosity / (1 - porosity), 1)))
108         print("Pore Sizes: ")
109         for cells in script.truss.cells:
110             for pore in cells.pore_size:
111                 print(str(round(pore * 1e3, 3)) + " µm, ")
112         output['Volume'] = results['Volume']  # mm^3
113         output['Porosity'] = porosity
114         output['Void_ratio'] = porosity / (1 - porosity)
115         output['Surface_Area'] = results['Surface_Area']  # mm^2
116     except KeyError:
117         print("This Evaluation is done without calculating Porosity or Volume")
118     # Calculate Compliance Matrix
119     length_cube = (script.truss.cell_size * script.truss.number_of_cells + script.truss.cells[0].strut_thicknesses[0])
120
121     # First Quarter:
122     output['Sigma_x'] = read_stress(results, 'SIGMA_X', 'SIGMA_X_1', 0)
123     output['Sigma_y'] = read_stress(results, 'SIGMA_Y', 'SIGMA_Y_1', 1)
124     output['Sigma_z'] = read_stress(results, 'SIGMA_Z', 'SIGMA_Z_1', 2)
125     print("Elastic Modulus in X direction: " + str(round(output['Sigma_x'], 3)) + " MPa")
126     print("Elastic Modulus in Y direction: " + str(round(output['Sigma_y'], 3)) + " MPa")
127     print("Elastic Modulus in Z direction: " + str(round(output['Sigma_z'], 3)) + " MPa")
128
129     compliance_sigma = numpy.zeros([3, 3])
130     compliance_sigma[0, 0] = 1 / output['Sigma_x']
131     compliance_sigma[1, 0] = -(read_displacement(results, 'SIGMA_X', 'SIGMA_Y_1', 1) -
132                                read_displacement(results, 'SIGMA_X', 'SIGMA_Y_2_Y', 1)) * length_cube / applied_force
133     compliance_sigma[2, 0] = -(read_displacement(results, 'SIGMA_X', 'SIGMA_Z_1', 2) -
134                                read_displacement(results, 'SIGMA_X', 'SIGMA_Z_2_Z', 2)) * length_cube / applied_force
135     compliance_sigma[0, 1] = -(read_displacement(results, 'SIGMA_Y', 'SIGMA_X_1', 0) -
136                                read_displacement(results, 'SIGMA_Y', 'SIGMA_X_2_X', 0)) * length_cube / applied_force
137     compliance_sigma[1, 1] = 1 / output['Sigma_y']
138     compliance_sigma[2, 1] = -(read_displacement(results, 'SIGMA_Y', 'SIGMA_Z_1', 2) -
139                                read_displacement(results, 'SIGMA_Y', 'SIGMA_Z_2_Z', 2)) * length_cube / applied_force
140     compliance_sigma[0, 2] = -(read_displacement(results, 'SIGMA_Z', 'SIGMA_X_1', 0) -
141                                read_displacement(results, 'SIGMA_X', 'SIGMA_X_2_X', 0)) * length_cube / applied_force
142     compliance_sigma[1, 2] = -(read_displacement(results, 'SIGMA_Z', 'SIGMA_Y_1', 1) -
143                                read_displacement(results, 'SIGMA_X', 'SIGMA_Y_2_Y', 1)) * length_cube / applied_force
144     compliance_sigma[2, 2] = 1 / output['Sigma_z']
145
146     # Second Quarter:
```

```
147         compliance_coupling2 = numpy.zeros([3, 3])
148         compliance_coupling2[0, 0] = (read_displacement(results, 'TAU_YZ', 'SIGMA_X_1', 0) -
149                                       read_displacement(results, 'TAU_YZ', 'SIGMA_X_2_X', 0)) * length_cube / applied_force
150         compliance_coupling2[1, 0] = (read_displacement(results, 'TAU_YZ', 'SIGMA_Y_1', 1) -
151                                       read_displacement(results, 'TAU_YZ', 'SIGMA_Y_2_Y', 1)) * length_cube / applied_force
152         compliance_coupling2[2, 0] = (read_displacement(results, 'TAU_YZ', 'SIGMA_Z_1', 2) -
153                                       read_displacement(results, 'TAU_YZ', 'SIGMA_Z_2_Z', 2)) * length_cube / applied_force
154         compliance_coupling2[0, 1] = (read_displacement(results, 'TAU_XZ', 'SIGMA_X_1', 0) -
155                                       read_displacement(results, 'TAU_XZ', 'SIGMA_X_2_X', 0)) * length_cube / applied_force
156         compliance_coupling2[1, 1] = (read_displacement(results, 'TAU_XZ', 'SIGMA_Y_1', 1) -
157                                       read_displacement(results, 'TAU_XZ', 'SIGMA_Y_2_Y', 1)) * length_cube / applied_force
158         compliance_coupling2[2, 1] = (read_displacement(results, 'TAU_XZ', 'SIGMA_Z_1', 2) -
159                                       read_displacement(results, 'TAU_XZ', 'SIGMA_Z_2_Z', 2)) * length_cube / applied_force
160         compliance_coupling2[0, 2] = (read_displacement(results, 'TAU_XY', 'SIGMA_X_1', 0) -
161                                       read_displacement(results, 'TAU_XY', 'SIGMA_X_2_X', 0)) * length_cube / applied_force
162         compliance_coupling2[1, 2] = (read_displacement(results, 'TAU_XY', 'SIGMA_Y_1', 1) -
163                                       read_displacement(results, 'TAU_XY', 'SIGMA_Y_2_Y', 1)) * length_cube / applied_force
164         compliance_coupling2[2, 2] = (read_displacement(results, 'TAU_XY', 'SIGMA_Z_1', 2) -
165                                       read_displacement(results, 'TAU_XY', 'SIGMA_Z_2_Z', 2)) * length_cube / applied_force
166
167         # Third Quarter:
168         compliance_coupling3 = numpy.zeros([3, 3])
169         compliance_coupling3[0, 0] = (read_displacement(results, 'SIGMA_X', 'SIGMA_Z_1', 1) -
170                                       read_displacement(results, 'SIGMA_X', 'SIGMA_Z_2_Z', 1)) * length_cube / applied_force
171         compliance_coupling3[1, 0] = (read_displacement(results, 'SIGMA_X', 'SIGMA_X_1', 2) -
172                                       read_displacement(results, 'SIGMA_X', 'SIGMA_X_2_X', 2)) * length_cube / applied_force
173         compliance_coupling3[2, 0] = (read_displacement(results, 'SIGMA_X', 'SIGMA_Y_1', 0) -
174                                       read_displacement(results, 'SIGMA_X', 'SIGMA_Y_2_Y', 0)) * length_cube / applied_force
175         compliance_coupling3[0, 1] = (read_displacement(results, 'SIGMA_Y', 'SIGMA_Z_1', 1) -
176                                       read_displacement(results, 'SIGMA_Y', 'SIGMA_Z_2_Z', 1)) * length_cube / applied_force
177         compliance_coupling3[1, 1] = (read_displacement(results, 'SIGMA_Y', 'SIGMA_X_1', 2) -
178                                       read_displacement(results, 'SIGMA_Y', 'SIGMA_X_2_X', 2)) * length_cube / applied_force
179         compliance_coupling3[2, 1] = (read_displacement(results, 'SIGMA_Y', 'SIGMA_Y_1', 0) -
180                                       read_displacement(results, 'SIGMA_Y', 'SIGMA_Y_2_Y', 0)) * length_cube / applied_force
181         compliance_coupling3[0, 2] = (read_displacement(results, 'SIGMA_Z', 'SIGMA_Z_1', 1) -
182                                       read_displacement(results, 'SIGMA_Z', 'SIGMA_Z_2_Z', 1)) * length_cube / applied_force
183         compliance_coupling3[1, 2] = (read_displacement(results, 'SIGMA_Z', 'SIGMA_X_1', 2) -
184                                       read_displacement(results, 'SIGMA_Z', 'SIGMA_X_2_X', 2)) * length_cube / applied_force
185         compliance_coupling3[2, 2] = (read_displacement(results, 'SIGMA_Z', 'SIGMA_Y_1', 0) -
186                                       read_displacement(results, 'SIGMA_Z', 'SIGMA_Y_2_Y', 0)) * length_cube / applied_force
187
188         # Fourth Quarter:
189         output['Tau_yz'] = (read_shearing(results, 'TAU_YZ', 'SIGMA_Z_1', 1) +
190                             read_shearing(results, 'TAU_YZ', 'SIGMA_Y_1', 2)) / 2
191         output['Tau_xz'] = (read_shearing(results, 'TAU_XZ', 'SIGMA_X_1', 2) +
192                             read_shearing(results, 'TAU_XZ', 'SIGMA_Z_1', 0)) / 2
193         output['Tau_xy'] = (read_shearing(results, 'TAU_XY', 'SIGMA_Y_1', 0) +
194                             read_shearing(results, 'TAU_XY', 'SIGMA_X_1', 1)) / 2
195         print("Shearing Modulus in yz direction: " + str(round(output['Tau_yz'], 3)) + " MPa")
196         print("Shearing Modulus in xz direction: " + str(round(output['Tau_xz'], 3)) + " MPa")
197         print("Shearing Modulus in xy direction: " + str(round(output['Tau_xy'], 3)) + " MPa")
198
199         compliance_tau = numpy.zeros([3, 3])
200         compliance_tau[0, 0] = 1 / output['Tau_yz']
201         compliance_tau[1, 0] = (read_displacement(results, 'TAU_YZ', 'SIGMA_Z_1', 0) -
202                                 read_displacement(results, 'TAU_YZ', 'SIGMA_Z_2_Z', 0)) * length_cube / applied_force
203         compliance_tau[2, 0] = (read_displacement(results, 'TAU_YZ', 'SIGMA_Y_1', 0) -
204                                 read_displacement(results, 'TAU_YZ', 'SIGMA_Y_2_Y', 0)) * length_cube / applied_force
205         compliance_tau[0, 1] = (read_displacement(results, 'TAU_XZ', 'SIGMA_Z_1', 1) -
206                                 read_displacement(results, 'TAU_XZ', 'SIGMA_Z_2_Z', 1)) * length_cube / applied_force
207         compliance_tau[1, 1] = 1 / output['Tau_xz']
208         compliance_tau[2, 1] = (read_displacement(results, 'TAU_XZ', 'SIGMA_X_1', 1) -
209                                 read_displacement(results, 'TAU_XZ', 'SIGMA_X_2_X', 1)) * length_cube / applied_force
210         compliance_tau[0, 2] = (read_displacement(results, 'TAU_XY', 'SIGMA_Y_1', 2) -
211                                 read_displacement(results, 'TAU_XY', 'SIGMA_Y_2_Y', 2)) * length_cube / applied_force
212         compliance_tau[1, 2] = (read_displacement(results, 'TAU_XY', 'SIGMA_X_1', 2) -
213                                 read_displacement(results, 'TAU_XY', 'SIGMA_X_2_X', 2)) * length_cube / applied_force
214         compliance_tau[2, 2] = 1 / output['Tau_xy']
215
216         output['Compliance'] = numpy.zeros([6, 6])
217         output['Compliance'][0:3, 0:3] = compliance_sigma
218         output['Compliance'][0:3, 3:7] = compliance_coupling2
219         output['Compliance'][3:7, 0:3] = compliance_coupling3
220         output['Compliance'][3:7, 3:7] = compliance_tau
221
```

```
222         # POISSON'S RATIO
223         output['v21'] = -output['Compliance'][1, 0] / output['Compliance'][0, 0]
224         output['v31'] = -output['Compliance'][2, 0] / output['Compliance'][0, 0]
225         output['v32'] = -output['Compliance'][1, 2] / output['Compliance'][2, 2]
226         print("Poisson's ratio v21: " + str(round(output['v21'], 3)))
227         print("Poisson's ratio v31: " + str(round(output['v31'], 3)))
228         print("Poisson's ratio v32: " + str(round(output['v32'], 3)))
229 
230         print("Compliance Matrix [1/MPa]: ")
231         numpy.set_printoptions(precision=2, suppress=True)
232         print(numpy.multiply(output['Compliance'], 1e3))
233         return output
234 
235 
236     # APPENDS RESULTS TO A CSV FILE AND A SERIALIZED PICKLE FILE
237     def append_output_to_file(options, output, output_file):
238         result_file = open(output_file, 'a')
239         if options['output']['Step']:
240             result_file.write(str(output['Step']) + ", ")
241         if options['output']['Truss_Name']:
242             result_file.write(str(output['Truss_Name']) + ", ")
243         if options['output']['Fitness']:
244             result_file.write(str(round(output['Fitness'], 3)) + ", ")
245         if options['output']['Cell_size']:
246             result_file.write(str(round(output['Cell_Size'], 3)) + ", ")
247         if options['output']['Strut_Thickness']:
248             result_file.write(str(round(output['Strut_Thickness'] * 1e3, 3)) + ", ")
249         if options['output']['Pore_size']:
250             counter = 0
251             for cells in output['Pore_size']:
252                 for pore in cells:
253                     result_file.write(str(round(pore * 1e3, 3)) + ", ")
254                     counter += 1
255             for i in range(0, 4 - counter):
256                 result_file.write("None, ")
257         if options['output']["Young's Modulus"]:
258             result_file.write(str(round(output['Sigma_z'], 3)) + ", " +
259                               str(round(output['Sigma_y'], 3)) + ", " +
260                               str(round(output['Sigma_x'], 3)) + ", ")
261         if options['output']['Shearing Modulus']:
262             result_file.write(str(round(output['Tau_yz'], 3)) + ", " +
263                               str(round(output['Tau_xz'], 3)) + ", " +
264                               str(round(output['Tau_xy'], 3)) + ", ")
265         if options['output']["Poisson's Ratio"]:
266             result_file.write(str(round(output['v21'], 3)) + ", " +
267                               str(round(output['v31'], 3)) + ", " +
268                               str(round(output['v32'], 3)) + ", ")
269         try:
270             if options['output']['Volume']:
271                 result_file.write(str(round(output['Volume'], 3)) + ", ")
272             if options['output']['Porosity']:
273                 result_file.write(str(round(output['Porosity'] * 100, 3)) + ", ")
274             if options['output']['Void_ratio']:
275                 result_file.write(str(round(output['Void_ratio'] * 1, 3)) + ", ")
276             if options['output']['Surface_Area']:
277                 result_file.write(str(round(output['Surface_Area'], 3)) + ", ")
278         except KeyError:
279             if options['output']['Volume']:
280                 result_file.write("None, ")
281             if options['output']['Porosity']:
282                 result_file.write("None, ")
283             if options['output']['Void_ratio']:
284                 result_file.write("None, ")
285             if options['output']['Surface_Area']:
286                 result_file.write("None, ")
287         if options['output']['X']:
288             result_file.write(str(output['X']))
289         result_file.write('\n')
290         result_file.close()
291 
292         # # APPENDS OUTPUT TO THE PICKLED OUTPUT
293         unpickle = pickle._Unpickler(open(str(output_file[:len(output_file) - 4]) + "_pickle", 'rb'))
294         unpickle.encoding = 'latin1'
295         output_old = unpickle.load()
296         output_old.append(output)
```

```
297
298     file = open(str(output_file[:len(output_file) - 4]) + "_pickle", 'wb')
299     pickle.dump(output_old, file)
300     file.close()
301
302
303 # SAFES THE INPUT IN A SERIALIZED PICKLE FILE
304 def pickle_input(x, truss_name, input_file):
305     pickle_dump = list()
306     pickle_dump.append(x)
307     pickle_dump.append(truss_name)
308     file = open(str(input_file), 'wb')
309     pickle.dump(pickle_dump, file)
310     file.close()
311
312
313 universal_counter = 0
314
315
316 ######################################################################################################################
317 # BEGIN OF function
318
319
320 def objective_function(inputs, options):
321     global universal_counter
322     universal_counter += 1  # Careful this is global
323
324     # PRINT INFO FROM INPUT
325     print("Counter: " + str(universal_counter))
326     print("Truss: " + str(inputs['truss_name']))
327     print("Strut Minimal Thickness: " + str(inputs['strut_min_thickness']))
328     print("Strut Thickness Multiplier: " + str(inputs['strut_thickness_multiplicator']))
329     print("Number of Cells: " + str(inputs['number_of_cells']))
330     print("Cell Size: " + str(round(inputs['cell_size'], 3)) + " mm")
331     print("Total Size: " + str(round(inputs['number_of_cells'] * inputs['cell_size'], 3)) + " mm")
332     print("Cell Ratio: " + str(inputs['cell_ratio']) + "\n\n\n")
333
334     # MULTIPLY strut_thicknesses WITH min_thickness
335     thicknesses = list()
336     fitness = 0
337     for multiplicator in inputs['strut_thickness_multiplicator']:
338         if multiplicator > 0 and inputs['strut_min_thickness'] > 0:
339             thicknesses.append(inputs['strut_min_thickness'] * multiplicator)
340         else:    # Blocks negative values for strut thicknesses
341             thicknesses.append(abs(inputs['strut_min_thickness'] * multiplicator))
342             fitness += 1e9 * abs(multiplicator) * abs(inputs['strut_min_thickness'])
343     filename = list()
344     filename.append(inputs['calculating_directory'])
345     filename.append(inputs['job_name'] + str(universal_counter))
346     filename.append(".py")
347     # GENERATE TRUSS
348     truss = generate_truss(truss_name=inputs['truss_name'], affix="", cell_size=inputs['cell_size'],
349                            strut_thicknesses=thicknesses, number_of_cells=inputs['number_of_cells'],
350                            cell_ratio=inputs['cell_ratio'])
351
352     # GENERATE SCRIPT
353
354     # initialize.
355     model_script = Script(filename=filename, truss=truss, material=inputs['material'],
356                           abaqus_path=options['abaqus_path'], abaqus_version=options['abaqus_version'])
357     # generate wireframe and evaluate.
358     model_script.evaluate(create_steps=options['create_steps'], submit_job=options['submit_job'],
359                           read_output=options['read_output'], number_of_cells=inputs['number_of_cells'])
360     # generate solid and export to stl.
361     if options['stl_generate']:
362         model_script.generate_solid(strut_name=options['strut_cross_section'], cutoff=options['cutoff'],
363                                     number_of_cells=inputs['number_of_cells'])
364         model_script.export_stl()
365
366     # saves the results from the simulation in a serialized pickle file.
367     model_script.pickle_dump()
368
369     # RUN SCRIPT
370     run(script=model_script, gui=options['gui'])
371
```

```
372     # OPEN VIEWERS TO SEE RESULT
373     if options['stl_view']:
374         stl_viewer(model_script)
375     if options['odb_view']:
376         odb_viewer(model_script)
377 
378     # CALCULATE COMPLIANCE FROM DISPLACEMENTS FROM THE SIMULATION
379     if options['read_output']:
380         output = read_results(model_script, inputs['strut_thickness_multiplicator'])
381     else:
382         output = dict()
383 
384     # CALCULATE FITNESS
385     if options['method'] == 'optimization':
386         for variable in options['fitness_variables']:
387             fitness += (abs(output[variable] - options['fitness_variables'][variable][0]) *
388                         options['fitness_variables'][variable][1]) * options['fitness_variables'][variable][2]
389         fitness /= len(options['fitness_variables'])
390 
391     # SAFE OUTPUT INTO CSV FILE
392     if options['read_output']:
393         output['Step'] = universal_counter
394         output['Fitness'] = fitness
395         append_output_to_file(options, output, inputs['output_file'] + '.csv')
396 
397     # SAFE INPUT VALUES AS A SERIALIZED PICKLE FILE
398     pickle_input(inputs['strut_thickness_multiplicator'],
399                  inputs['truss_name'],
400                  str(inputs['calculating_directory']) +
401                  str(inputs['job_name']) +
402                  str(universal_counter) +
403                  "_input")
404 
405     # PLOT FITNESS
406     if options['plot_fitness']:
407         pyplot.figure(1)
408         pyplot.subplot(311)
409         pyplot.xlabel('Step')
410         pyplot.ylabel('Fitness')
411         pyplot.ylim((100, 2000))
412         pyplot.scatter(universal_counter, fitness)
413         pyplot.legend()
414         pyplot.subplot(312)
415         pyplot.xlabel('Step')
416         pyplot.ylabel('Fitness')
417         pyplot.ylim((20, 100))
418         pyplot.scatter(universal_counter, fitness)
419         pyplot.legend()
420         pyplot.subplot(313)
421         pyplot.xlabel('Step')
422         pyplot.ylabel('Fitness')
423         pyplot.ylim((-5, 20))
424         pyplot.scatter(universal_counter, fitness)
425         pyplot.legend()
426         pyplot.show()
427         pyplot.pause(0.00001)
428 
429     print("Fitness: " + str(fitness))
430     print("##########################################################################################################")
431     return fitness
```

A.4.3 *Class_Script.py*

```
"""
This File initiates the abaqus script to run for the optimization.
It consists of several functions that append text to a .py -file which is then run in abaqus.

The first block of functions creates a solid model which can be exported to stl.
The second block of functions creates a wireframe model and does an analysis of its properties.
The third block reads the output database that the simulation creates and returns it in a pickled file.

"""
import sys
import math
import numpy


########################################################################################################################
# FUNCTIONS FOR THE SOLID AND STL-FILE-GENERATION
# EXTRUDES A SIMPLE STRUT. POSSIBLE SECTIONS ARE SQUARE, HEXAGON, OCTAGON, DODECAGON
def generate_strut(filename, strut_name, affix, connection):
    file = open("".join(filename), "a")
    thickness = connection[2]
    begin_vector = numpy.asarray(connection[0])
    end_vector = numpy.asarray(connection[1])
    distance = numpy.subtract(end_vector, begin_vector)
    length = numpy.linalg.norm(distance)

    # GENERATE BASE SKETCH
    file.write(
        "s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',sheetSize=200.0)\n"
        "g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints\n"
        "s.setPrimaryObject(option=STANDALONE)\n")

    # SQUARE
    if strut_name == "square":
        file.write(
            "s.Line(point1=(" + str(-thickness / 2) + ", " + str(-thickness / 2) + "), point2=(" + str(thickness / 2) +
            ", " + str(-thickness / 2) + "))\n"
            "s.Line(point1=(" + str(thickness / 2) + ", " + str(-thickness / 2) + "), point2=(" + str(thickness / 2) +
            ", " + str(thickness / 2) + "))\n"
            "s.Line(point1=(" + str(thickness / 2) + ", " + str(thickness / 2) + "), point2=(" + str(-thickness / 2) +
            ", " + str(thickness / 2) + "))\n"
            "s.Line(point1=(" + str(-thickness / 2) + ", " + str(thickness / 2) + "), point2=(" + str(-thickness / 2) +
            ", " + str(-thickness / 2) + "))\n")

    # OCTAGON
    elif strut_name == "octagon":
        b = math.sqrt(2) / 2 * (1 - (math.sqrt(2) / (1 + math.sqrt(2)))) * thickness
        file.write(
            "s.Line(point1=(" + str(-thickness / 2 + b) + ", " + str(-thickness / 2) + "), point2=(" +
            str(thickness / 2 - b) + ", " + str(-thickness / 2) + "))\n"
            "s.Line(point1=(" + str(thickness / 2 - b) + ", " + str(-thickness / 2) + "), point2=(" +
            str(thickness / 2) + ", " + str(-thickness / 2 + b) + "))\n"
            "s.Line(point1=(" + str(thickness / 2) + ", " + str(-thickness / 2 + b) + "), point2=(" +
            str(thickness / 2) + ", " + str(thickness / 2 - b) + "))\n"
            "s.Line(point1=(" + str(thickness / 2) + ", " + str(thickness / 2 - b) + "), point2=(" +
            str(thickness / 2 - b) + ", " + str(thickness / 2) + "))\n"
            "s.Line(point1=(" + str(thickness / 2 - b) + ", " + str(thickness / 2) + "), point2=(" +
            str(-thickness / 2 + b) + ", " + str(thickness / 2) + "))\n"
            "s.Line(point1=(" + str(-thickness / 2 + b) + ", " + str(thickness / 2) + "), point2=(" +
            str(-thickness / 2) + ", " + str(thickness / 2 - b) + "))\n"
            "s.Line(point1=(" + str(-thickness / 2) + ", " + str(thickness / 2 - b) + "), point2=(" +
            str(-thickness / 2) + ", " + str(-thickness / 2 + b) + "))\n"
            "s.Line(point1=(" + str(-thickness / 2) + ", " + str(-thickness / 2 + b) + "), point2=(" +
            str(-thickness / 2 + b) + ", " + str(-thickness / 2) + "))\n")

    # DODECAGON
    elif strut_name == "dodecagon":
        for counter in range(0, 12):
            file.write("s.Line(point1=(" +
                       str(math.cos((15 + 30 * counter) / 180 * math.pi) * thickness / 2) + ", " +
                       str(math.sin((15 + 30 * counter) / 180 * math.pi) * thickness / 2) + "), point2=(" +
                       str(math.cos((15 + 30 * (counter + 1)) / 180 * math.pi) * thickness / 2) + ", " +
```

```
                        str(math.sin((15 + 30 * (counter + 1)) / 180 * math.pi) * thickness / 2) + "))\n")

    # HEXAGON
    elif strut_name == "hexagon":
        for counter in range(0, 6):
            file.write("s.Line(point1=(" +
                       str(math.cos((30 + 60 * counter) / 180 * math.pi) * thickness / 2) + ", " +
                       str(math.sin((30 + 60 * counter) / 180 * math.pi) * thickness / 2) + "), point2=(" +
                       str(math.cos((30 + 60 * (counter + 1)) / 180 * math.pi) * thickness / 2) + ", " +
                       str(math.sin((30 + 60 * (counter + 1)) / 180 * math.pi) * thickness / 2) + "))\n")

    else:
        sys.exit("ERROR: strut_name specified is not defined in the library")

    file.write("mdb.models['Model-1'].ConstrainedSketch(name='strut_sketch-" + str(strut_name) +
               str(affix) + "', objectToCopy=s)\n")

    # EXTRUDE BASE SKETCH
    file.write(
        "p = mdb.models['Model-1'].Part(name='strut_" + str(strut_name) + str(affix) +
        "', dimensionality=THREE_D, type=DEFORMABLE_BODY)\n"
        "p = mdb.models['Model-1'].parts['strut_" + str(strut_name) + str(affix) + "']\n"
        "p.BaseSolidExtrude(sketch=s, depth=" + str(length) + ")\n"
        "s.unsetPrimaryObject()\n")

    file.close()


# ADDS A SINGLE STRUT TO THE CELL STRUCTURE. THIS IS USED IN generate_cell()
def add_strut_to_cell(filename, strut_name, instance_name, strut_counter, connections):
    file = open("".join(filename), "a")
    begin_vector = numpy.asarray(connections[0])  # Determine the rotation and translation of the strut
    end_vector = numpy.asarray(connections[1])
    distance = numpy.subtract(end_vector, begin_vector)
    length = numpy.linalg.norm(distance)
    normal = numpy.cross(distance, [0, 0, 1])
    if numpy.linalg.norm(normal) == 0:
        normal = [1, 0, 0]
    angle = -abs(math.acos(numpy.dot([0, 0, 1], distance) / length)) / math.pi * 180
    # INCLUDE INTO ASSEMBLY
    file.write(
        "a = mdb.models['Model-1'].rootAssembly\n"

        "a.DatumCsysByDefault(CARTESIAN)\n"

        "p = mdb.models['Model-1'].parts['strut_" + str(strut_name) + str(strut_counter) + "']\n"

        "a.Instance(name='instance_" + str(instance_name) + "-" + str(strut_counter) + "', part=p, dependent=ON)\n"

        "a = mdb.models['Model-1'].rootAssembly\n"

        "a.rotate(instanceList=('instance_" + str(instance_name) + "-" + str(strut_counter) +
        "', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(" + str(normal[0]) + ", " +
        str(normal[1]) + ", " + str(normal[2]) + "), angle=" + str(angle) + ")\n"

        "a.translate(instanceList=('instance_" + str(instance_name) + "-" + str(strut_counter) + "', ), vector=(" +
        str(begin_vector[0]) + ", " +
        str(begin_vector[1]) + ", " +
        str(begin_vector[2]) + "))\n"
    )

    file.close()


# ADDS A SINGLE CELL TO THE TRUSS STRUCTURE. THIS IS USED IN generate_solid()
def add_cell_to_assembly(filename, cell_name, instance_name, cell_counter, truss_node_coordinates):
    file = open("".join(filename), "a")

    # INCLUDE INTO ASSEMBLY
    file.write(
        "a = mdb.models['Model-1'].rootAssembly\n"

        "a.DatumCsysByDefault(CARTESIAN)\n"

        "p = mdb.models['Model-1'].parts['" + str(cell_name) + "']\n"
```

```
147
148         "a.Instance(name='instance_" + str(instance_name) + "-" + str(cell_counter) + "', part=p, dependent=ON)\n"
149
150         "a = mdb.models['Model-1'].rootAssembly\n"
151
152         "a.translate(instanceList=('instance_" + str(instance_name) + "-" + str(cell_counter) + "', ), vector=(" +
153         str(truss_node_coordinates[0]) + ", " +
154         str(truss_node_coordinates[1]) + ", " +
155         str(truss_node_coordinates[2]) + "))\n")
156     file.close()
157
158
159 # MERGES ALL PARTS OF AN INSTANCE. THIS IS USED IN generate_cell() AND generate_solid()
160 def merge(filename, merge_name, instance_name, part_counter):
161     file = open("".join(filename), "a")
162     if part_counter > 1:
163         file.write("a.InstanceFromBooleanMerge(name='" + str(merge_name) + "', instances=(")
164         for counter in range(0, part_counter):
165             file.write("a.instances['instance_" + str(instance_name) + "-" + str(counter) + "'], ")
166         file.write("), originalInstances=SUPPRESS, domain=GEOMETRY)\n")
167         for counter in range(0, part_counter):
168             file.write("del a.features['instance_" + str(instance_name) + "-" + str(counter) + "']\n")
169     else:
170         file.write("mdb.models['Model-1'].rootAssembly.features.changeKey(fromName='instance_" + str(instance_name) +
171                    "-0', toName='" + str(merge_name) + "-1')\n")
172     file.close()
173
174
175 # GENERATES THE SOLID CELL
176 def generate_cell(filename, cell, strut_name):
177     strut_counter = 0
178     for connection in cell.connections:
179         generate_strut(filename, strut_name, strut_counter, connection)
180         add_strut_to_cell(filename, strut_name, cell.name, strut_counter, connection)
181         strut_counter += 1
182
183     merge(filename, cell.name, cell.name, strut_counter)  # "Instance with name 'cell.name'
184     delete_struts(filename, strut_name, strut_counter)
185
186
187 # DELETES ALL CELL INSTANCES AFTER MERGING
188 def delete_struts(filename, strut_name, number_of_struts):
189     for strut_counter in range(0, number_of_struts):
190         file = open("".join(filename), "a")
191         file.write("del mdb.models['Model-1'].parts['strut_" + str(strut_name) + str(strut_counter) + "']\n")
192         file.close()
193
194
195 def delete_instances(filename, objects):
196     file = open("".join(filename), "a")
197     for obj in objects:
198         file.write("del a.features['" + str(obj.name) + "-1']\n")
199
200     file.close()
201
202
203 # CUT OFF THE BORDERS OF THE TRUSS
204 def cut_off_borders(filename, cutoff_name, merge_name, truss, number_of_cells):
205     file = open("".join(filename), "a")
206     file.write("s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',sheetSize=200.0)\n"
207                "g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints\n"
208                "s.setPrimaryObject(option=STANDALONE)\n")
209     file.write("s.Line(point1=(" + str(-(number_of_cells + 4) * truss.cell_size / 2) + ", " +
210                str(-(number_of_cells + 4) * truss.cell_size / 2) + "), point2=(" +
211                str((number_of_cells + 4) * truss.cell_size / 2) + ", " +
212                str(-(number_of_cells + 4) * truss.cell_size / 2) + "))\n"
213                "s.Line(point1=(" + str((number_of_cells + 4) * truss.cell_size / 2) + ", " +
214                str(-(number_of_cells + 4) * truss.cell_size / 2) + "), point2=(" +
215                str((number_of_cells + 4) * truss.cell_size / 2) + ", " +
216                str((number_of_cells + 4) * truss.cell_size / 2) + "))\n"
217                "s.Line(point1=(" + str((number_of_cells + 4) * truss.cell_size / 2) + ", " +
218                str((number_of_cells + 4) * truss.cell_size / 2) + "), point2=(" +
219                str(-(number_of_cells + 4) * truss.cell_size / 2) + ", " +
220                str((number_of_cells + 4) * truss.cell_size / 2) + "))\n"
221                "s.Line(point1=(" + str(-(number_of_cells + 4) * truss.cell_size / 2) + ", " +
```

```
                str((number_of_cells + 4) * truss.cell_size / 2) + "), point2=(" +
                str(-(number_of_cells + 4) * truss.cell_size / 2) + ", " +
                str(-(number_of_cells + 4) * truss.cell_size / 2) + "))\n")
    file.write("mdb.models['Model-1'].ConstrainedSketch(name='cut_off', objectToCopy=s)\n")
    file.write(
        "p = mdb.models['Model-1'].Part(name='cut_off', dimensionality=THREE_D, type=DEFORMABLE_BODY)\n"
        "p = mdb.models['Model-1'].parts['cut_off']\n"
        "p.BaseSolidExtrude(sketch=s, depth=" + str(2 * truss.cell_size) + ")\n"
                                                                          "s.unsetPrimaryObject()\n")
    file.write(
        "a = mdb.models['Model-1'].rootAssembly\n"

        "a.DatumCsysByDefault(CARTESIAN)\n"

        "p = mdb.models['Model-1'].parts['cut_off']\n"

        "a.Instance(name='instance_cut_off1', part=p, dependent=ON)\n"
        "a.Instance(name='instance_cut_off2', part=p, dependent=ON)\n"
        "a.Instance(name='instance_cut_off3', part=p, dependent=ON)\n"
        "a.Instance(name='instance_cut_off4', part=p, dependent=ON)\n"
        "a.Instance(name='instance_cut_off5', part=p, dependent=ON)\n"
        "a.Instance(name='instance_cut_off6', part=p, dependent=ON)\n"

        "a.translate(instanceList=('instance_cut_off1', ), vector=(" +
        str(0) + ", " +
        str(0) + ", " +
        str((number_of_cells - 0.5) * truss.cell_size) + "))\n"
        "a.translate(instanceList=('instance_cut_off1', ), vector=(" + str(truss.cell_size) + "," +
        str(truss.cell_size) + ",0))\n"
        "a.rotate(instanceList=('instance_cut_off2', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(0,1,0), angle=180)\n"
        "a.translate(instanceList=('instance_cut_off2', ), vector=(" +
        str(0) + ", " +
        str(0) + ", " +
        str(-0.5 * truss.cell_size) + "))\n"
                                      "a.translate(instanceList=('instance_cut_off2', ), vector=(" +
        str(truss.cell_size) + "," + str(truss.cell_size) + ",0))\n"
        "a.rotate(instanceList=('instance_cut_off3', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(0,1,0), angle=-90)\n"
        "a.translate(instanceList=('instance_cut_off3', ), vector=(" +
        str(-0.5 * truss.cell_size) + ", " +
        str(0) + ", " +
        str(0) + "))\n"
                 "a.translate(instanceList=('instance_cut_off3', ), vector=(0, " + str(truss.cell_size) + "," +
        str(truss.cell_size) + "))\n"
        "a.rotate(instanceList=('instance_cut_off4', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(0,1,0), angle=90)\n"
        "a.translate(instanceList=('instance_cut_off4', ), vector=(" +
        str((number_of_cells - 0.5) * truss.cell_size) + ", " +
        str(0) + ", " +
        str(0) + "))\n"
                 "a.translate(instanceList=('instance_cut_off4', ), vector=(0, " + str(truss.cell_size) + "," +
        str(truss.cell_size) + "))\n"
        "a.rotate(instanceList=('instance_cut_off5', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(1,0,0), angle=90)\n"
        "a.translate(instanceList=('instance_cut_off5', ), vector=(" +
        str(0) + ", " +
        str(-0.5 * truss.cell_size) + ", " +
        str(0) + "))\n"
        "a.translate(instanceList=('instance_cut_off5', ), vector=(" + str(truss.cell_size) + ", 0, " +
        str(truss.cell_size) + "))\n"
        "a.rotate(instanceList=('instance_cut_off6', ), axisPoint=(0.0, 0.0, 0.0), axisDirection=(1,0,0), angle=-90)\n"
        "a.translate(instanceList=('instance_cut_off6', ), vector=(" +
        str(0) + ", " +
        str((number_of_cells - 0.5) * truss.cell_size) + ", " +
        str(0) + "))\n"
        "a.translate(instanceList=('instance_cut_off6', ), vector=(" + str(truss.cell_size) + ", 0, " +
        str(truss.cell_size) + "))\n"
    )

    file.write("a.InstanceFromBooleanCut(name='" + cutoff_name + "', "
               "instanceToBeCut=mdb.models['Model-1'].rootAssembly.instances['" + merge_name + "'], "
               "cuttingInstances=(a.instances['instance_cut_off1'], a.instances['instance_cut_off2'], "
               "a.instances['instance_cut_off3'], a.instances['instance_cut_off4'], a.instances['instance_cut_off5'], "
               "a.instances['instance_cut_off6'], ), originalInstances=DELETE)\n")
    file.close()


# MEASURE THE VOLUME AND SURFACE ARE OF THE TRUSS CREATED IN ABAQUS. ONLY WORKS IF THE BORDERS ARE CUT OFF
```

```
297 def get_area(script, filename, cutoff):
298     file = open("".join(filename), "a")
299     file.write("results['Volume'] = a.getVolume()\n"
300                "results['Surface_Area'] = a.getArea(a.instances['" + filename[1] +
301                "-1'].faces)\n"
302                "print('Results: ' + str(results['Surface_Area']))\n")
303     if cutoff:
304         if script.truss.name == "cubes" or script.truss.name == "body_centered_cubes" or \
305                 script.truss.name == "octetrahedrons" or script.truss.name == "face_diagonal_cubes_alt" or \
306                 script.truss.name == "void_octetrahedrons":
307             file.write("area_error=-6 * a.getArea(a.instances['" + filename[1] +
308                        "-1'].faces.findAt((" + str(script.truss.cells[0].strut_thicknesses[0] / 32 -
309                                                    script.truss.cell_size / 2) +
310                        ", " + str(-script.truss.cell_size / 2) + ", " +
311                        str(script.truss.cells[0].strut_thicknesses[0] / 32 - script.truss.cell_size / 2) + "),),))\n"
312                        "print('Area Error: ' + str(area_error))\n"
313                        "results['Surface_Area']+=area_error\n")
314         elif script.truss.name == "diamonds":
315             file.write("-12 * a.getArea(a.instances['" + filename[1] +
316                        "-1'].faces.findAt((" +
317                        str(script.truss.cells[0].strut_thicknesses[0] / 32 - script.truss.cell_size / 2) +
318                        ", " + str(-script.truss.cell_size / 2) + ", " +
319                        str(script.truss.cells[0].strut_thicknesses[0] / 32 - script.truss.cell_size / 2) + "),),))"
320                        )
321             file.write("-6 * a.getArea(a.instances['" + filename[1] +
322                        "-1'].faces.findAt((" + str(0) +
323                        ", " + str(-script.truss.cell_size / 2) + ", " + str(0) + "),),))\n"
324                        )
325         elif script.truss.name == "truncated_cubes":
326             file.write("-6 * a.getArea(a.instances['" + filename[1] +
327                        "-1'].faces.findAt((" +
328                        str(script.truss.cells[0].strut_thicknesses[0] / 32 - script.truss.cell_size / 2) +
329                        ", " + str(-script.truss.cell_size / 2) + ", " + str(0) + "),),))\n"
330                        )
331         else:
332             print("The Calculated Surface Area is too high for it does not substract the outside area of the " +
333                   str(script.truss.name) + " truss")
334     else:
335         print("No surface area correction because there is no cutoff")
336
337 ########################################################################################################################
338 # FUNCTIONS FOR THE FINITE ELEMENT ANALYSIS AND WIREFRAME GENERATION
339
340
341 # GENERATE THE WIREFRAME
342 def generate_wireframe(filename, truss):
343     file = open("".join(filename), "a")
344     file.write(
345         "p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D, type=DEFORMABLE_BODY)\n"
346         "p.ReferencePoint(point=(0.0, 0.0, 0.0))\n"
347         "p.WirePolyLine(points=(\n")
348     for nodes in truss.nodes:
349         for connections in nodes[3].connections:
350             file.write("                       ((" + str(connections[0][0] + nodes[0]) +
351                        ", " + str(connections[0][1] + nodes[1]) +
352                        ", " + str(connections[0][2] + nodes[2]) +
353                        "), (" + str(connections[1][0] + nodes[0]) +
354                        ", " + str(connections[1][1] + nodes[1]) +
355                        ", " + str(connections[1][2] + nodes[2]) +
356                        ")), \n")
357
358     file.write(
359         "), mergeType=IMPRINT, meshable=ON)\n"
360         "p = mdb.models['Model-1'].parts['Part-1']\n"
361         "e = p.edges\n"
362         "edges = e.getSequenceFromMask(mask=('[#7 ]', ), )\n"
363         "p.Set(edges=edges, name='Wire-" + str(truss.name) + "')\n"
364         "a = mdb.models['Model-1'].rootAssembly\n"
365         "a.DatumCsysByDefault(CARTESIAN)\n")
366     file.close()
367
368
369 # DEFINE THE MATERIAL PROPERTIES
370 def define_material(filename, material_name, young_modulus, poisson_ratio):
371     file = open("".join(filename), "a")
```

```
372         file.write("mdb.models['Model-1'].Material(name='" + str(material_name) +
373                    "', description='Automatically generated. See Class_Script')\n"
374                    "mdb.models['Model-1'].materials['" + str(material_name) + "'].Elastic(table=((" +
375                    str(young_modulus) + ", " + str(poisson_ratio) + "), ))\n")
376         file.close()
377
378
379     # ASSIGNS PROFILE AND MATERIAL
380     def assign_material_to_wire(filename, truss, material_name):
381         file = open("".join(filename), "a")
382
383         for cell in truss.cells:
384
385             list_of_nodes = list()
386             for node in truss.nodes:
387                 if cell.name == node[3].name:
388                     list_of_nodes.append(node)
389
390             counter = 0
391             for connection in cell.connections:
392                 file.write("mdb.models['Model-1'].CircularProfile(name='Profile-" + str(cell.name) + "-" + str(counter) +
393                            "', r=" + str(connection[2] / 2) + ")\n"
394                            "mdb.models['Model-1'].BeamSection(name='Section-" + str(cell.name) + "-" + str(counter) +
395                            "',integration=DURING_ANALYSIS, poissonRatio=0.0, profile='Profile-" + str(cell.name) + "-" +
396                            str(counter) + "',material='" + str(material_name) + "', temperatureVar=LINEAR, "
397                            "consistentMassMatrix=False)\n"
398                            "r = regionToolset.Region(edges=p.edges.findAt(")
399                 for connection_node in list_of_nodes:
400                     file.write("((" + str(1 / 2 * (connection[0][0] + connection[1][0]) + connection_node[0]) + ", " +
401                                str(1 / 2 * (connection[0][1] + connection[1][1]) + connection_node[1]) + ", " +
402                                str(1 / 2 * (connection[0][2] + connection[1][2]) + connection_node[2]) + ", " +
403                                "), ), ")
404                 file.write("))\n"
405                            "p.SectionAssignment(region=r, sectionName='Section-" + str(cell.name) + "-" + str(counter) +
406                            "', offset=0.0,"
407                            "offsetType=MIDDLE_SURFACE, offsetField='',"
408                            "thicknessAssignment=FROM_SECTION)\n"
409                            "p.assignBeamSectionOrientation(region=r, method=N1_COSINES, n1=(5,7,3))\n")
410                 counter += 1
411             del counter
412
413
414     # MESHES THE TRUSS
415     def mesh_part(filename, mesh_size):
416         file = open("".join(filename), "a")
417         file.write("a = mdb.models['Model-1'].rootAssembly\n"
418                    "p = mdb.models['Model-1'].parts['Part-1']\n"
419                    "a.Instance(name='Part-1-1', part=p, dependent=ON)\n"
420                    "p.seedPart(size=" + str(mesh_size) + ", deviationFactor=0.1, minSizeFactor=0.1)\n"
421                    "p.generateMesh()\n"
422                    "p.setElementType(regions=regionToolset.Region(p.edges), "
423                    "elemTypes=(mesh.ElemType(elemCode=B32,elemLibrary=STANDARD),))\n")  # B32
424         file.close()
425
426
427     # CREATES A STATIC STEP
428     def create_static_step(filename, step_name):
429         file = open("".join(filename), "a")
430         file.write("mdb.models['Model-1'].StaticStep(name='" + str(step_name) + "', previous='Initial')\n")
431         file.close()
432
433
434     # DEFINES LOADINGS ON THE TRUSS
435     def define_loadings(filename, step_name, affix, vertices, load_type, load_type_letter, force):
436         file = open("".join(filename), "a")
437         file.write("loading_points = a.instances['Part-1-1'].vertices.findAt(")
438         for vertex_coordinates in vertices:
439             file.write("((" + str(vertex_coordinates[0]) +
440                        ", " + str(vertex_coordinates[1]) +
441                        ", " + str(vertex_coordinates[2]) + "), ), ")
442         file.write(
443             ")\n"
444             "mdb.models['Model-1']." + str(load_type) + "(name='" + str(step_name + affix) + "_load', createStepName='" +
445             str(step_name) + "',region=a.Set(vertices=loading_points, name='" + str(step_name) + "_1" + str(affix) +
446             "'), c" + str(load_type_letter) + "1=" + str(force[0]) + ", c" + str(load_type_letter) + "2=" + str(force[1]) +
```

```
447             ", c" + str(load_type_letter) + "3=" + str(force[2]) +
448             ", distributionType=UNIFORM, field='', localCsys=None)\n")
449         file.close()
450         # ConcentratedForce     f
451         # Moment                m
452 
453 
454     # DEFINES BOUNDARY CONDITIONS OF THE MODEL
455     def define_boundary_conditions(filename, step_name, affix, vertices, direction):
456         file = open("".join(filename), "a")
457         condition_type = 'Displacement'
458         file.write("fixes = a.instances['Part-1-1'].vertices.findAt(")
459         for vertex_coordinates in vertices:
460             file.write("((" + str(vertex_coordinates[0]) +
461                        ", " + str(vertex_coordinates[1]) +
462                        ", " + str(vertex_coordinates[2]) + "), ), ")
463         file.write(")\n"
464                    "mdb.models['Model-1']." + str(condition_type) + "BC(name='" + str(step_name) + str(affix) +
465                    "_BC', createStepName='" + str(step_name) + "', region=a.Set(vertices=fixes, name='" +
466                    str(step_name) + "_2" + str(affix) + "'), localCsys=None, " + str(direction) + ")\n")
467         file.close()
468         # condition_types:
469         # Encastre
470         # Pinned
471         # Xsymm
472         # Ysymm
473         # Zsymm
474         # Xasymm
475         # Yasymm
476         # Zasymm
477         # Displacement
478         # direction:
479         # u1, u2, u3, ur1, ur2, ur3
480 
481 
482     # DEACTIVATES BOUNDARY CONDITIONS OF FORMER STEPS (THEY GET PROPAGATED BY ABAQUS DEFAULT)
483     def deactivate_boundary_conditions(filename, step_name, deactivate_step_name):
484         file = open("".join(filename), "a")
485         file.write("mdb.models['Model-1'].boundaryConditions['" + str(step_name) + "_BC'].deactivate('" +
486                    str(deactivate_step_name) + "')\n")
487         file.close()
488 
489 
490     # DEACTIVATES LOADINGS OF FORMER STEPS (THEY GET PROPAGATED BY ABAQUS DEFAULT)
491     def deactivate_loadings(filename, step_name, deactivate_step_name):
492         file = open("".join(filename), "a")
493         file.write("mdb.models['Model-1'].loads['" + str(step_name) + "_load'].deactivate('" +
494                    str(deactivate_step_name) + "')\n")
495         file.close()
496 
497 
498     # GET THE DISPLACEMENT OF REQUESTED FIELD OUTPUT
499     def request_field_output(filename, step_name, output_name):
500         file = open("".join(filename), "a")
501         file.write("mdb.models['Model-1'].FieldOutputRequest(name='" + str(output_name) + "', createStepName='" +
502                    str(step_name) + "', variables=('U','S', ))\n")
503         file.close()
504 
505 
506     # SUBMITS THE SIMULATION
507     def submit(filename, job_name):
508         file = open("".join(filename), "a")
509         file.write("mdb.Job(name='" + str(job_name) + "', model='Model-1', description='', type=ANALYSIS, "
510                    "atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90, "
511                    "memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True, "
512                    "explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF, "
513                    "modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='', scratch='', resultsFormat=ODB)\n"
514                    "mdb.jobs['" + str(job_name) + "'].submit(consistencyChecking=OFF)\n"
515                    "mdb.jobs['" + str(job_name) + "'].waitForCompletion()\n"
516                    )
517         file.close()
518 
519 
520     ########################################################################################################
521     # FUNCTIONS FOR THE OUTPUT EVALUATION OF THE SIMULATION
```

```
522
523 # READ THE DISPLACEMENTS OF ALL SIDES OF A CERTAIN STEP AND RETURNS THEM
524 def read_odb_step(filename, step_name, job_name):
525     file = open("".join(filename), "a")
526     file.write("myOdb = odbAccess.openOdb(path='" + str(job_name) + ".odb')\n")
527     # for step_name in step_name:
528     file.write("Step = myOdb.steps['" + str(step_name) + "']\n"
529                "displacement = Step.frames[1].fieldOutputs['U']\n"
530                "list_of_sides = ['SIGMA_X_1', 'SIGMA_X_2_X', 'SIGMA_Y_1', 'SIGMA_Y_2_Y', 'SIGMA_Z_1', 'SIGMA_Z_2_Z']\n"
531                "node_region=dict()\n"
532                "displacements = dict()\n"
533                "for sides in list_of_sides:\n"
534                "    node_region[sides]=myOdb.rootAssembly.nodeSets[sides]\n"
535                "    values = displacement.getSubset(region=node_region[sides]).values\n"
536                "    displacements[sides]=list()\n"
537                "    for value in values:\n"
538                "        displacements[sides].append(value.data)\n"
539                "results['" + step_name + "'] = displacements\n")
540     file.close()
541
542
543 ######################################################################################################################
544
545
546 class Script:
547     # INITIALIZE THE SCRIPT
548     def __init__(self, filename, truss, material, abaqus_path, abaqus_version):
549         self.filename = filename
550         self.truss = truss
551         self.material = material
552
553         # OPEN THE SCRIPT FILE
554         file = open("".join(self.filename), "w")
555         # WRITE INTO FILE
556         file.write("# " + "".join(self.filename) + "\n# This file is automatically generated to run in abaqus\n")
557         import datetime
558         file.write("# This file was generated on " + str(datetime.datetime.now()) +
559                    "\n# Max Engensperger, maxe@alumni.ethz.ch\n# Truss name: " + str(truss.name) + "\n# Cell_Size: " +
560                    str(truss.cell_size) + "\n# Number of Cells: " + str(truss.number_of_cells) + "\n")
561         file.write(
562             "from abaqus import *\n"
563             "from abaqusConstants import *\n"
564             "import __main__\n"
565             "import section\n"
566             "import regionToolset\n"
567             "import displayGroupMdbToolset as dgm\n"
568             "import part\n"
569             "import material\n"
570             "import assembly\n"
571             "import step\n"
572             "import interaction\n"
573             "import load\n"
574             "import mesh\n"
575             "import optimization\n"
576             "import job\n"
577             "import sketch\n"
578             "import visualization\n"
579             "import xyPlot\n"
580             "import displayGroupOdbToolset as dgo\n"
581             "import connectorBehavior\n"
582             "import sys\n"
583             "import odb\n"
584             "from odbAccess import *\n"
585             "import odbAccess\n"
586             "from odbMaterial import *\n"
587             "from odbSection import *\n"
588             "from abaqusConstants import *\n"
589             "from odbMaterial import *\n"
590             "from odbSection import *\n"
591             "import pickle\n"
592             "sys.path.insert(9, r'" + str(abaqus_path) + str(abaqus_version) +  # c:/SIMULIA/Abaqus
593             "/code/python2.7/lib/abaqus_plugins/stlExport')\n"
594             "import stlExport_kernel\n"
595             "Mdb()\n"
596             "session.viewports['Viewport: 1'].viewportAnnotationOptions.setValues(compassPrivilegedPlane=XYPLANE)\n"
```

```
            "session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=COORDINATE)\n"
            "results = dict()\n\n\n\n")
        file.close()

    # GENERATE STL FROM SOLID
    def generate_solid(self, strut_name, cutoff, number_of_cells):
        for cell_types in self.truss.cells:
            generate_cell(self.filename, cell_types, strut_name)
        cell_counter = 0
        for nodes in self.truss.nodes:
            add_cell_to_assembly(self.filename, nodes[3].name, self.filename[1], cell_counter, nodes)
            cell_counter += 1
        if cutoff:
            merge(self.filename, "merged", self.filename[1], cell_counter)
            cut_off_borders(self.filename, self.filename[1], "merged-1", self.truss, number_of_cells)
        else:
            merge(self.filename, self.filename[1], self.filename[1], cell_counter)
        delete_instances(self.filename, self.truss.cells)
        del cell_counter
        get_area(self, self.filename, cutoff)

    def export_stl(self):
        file = open("".join(self.filename), "a")
        file.write(
            "a = mdb.models['Model-1'].rootAssembly\n"
            "session.viewports['Viewport: 1'].setValues(displayedObject=a)\n"
            "session.viewports['Viewport: 1'].assemblyDisplay.setValues(optimizationTasks=OFF, "
            "geometricRestrictions=OFF, stopConditions=OFF)\n"
            "stlExport_kernel.STLExport(moduleName='Assembly', stlFileName='" +
            self.filename[0] + self.filename[1] + ".stl', stlFileType='BINARY')\n")
        file.close()

    # GENERATE FEA FROM WIREFRAME BEAM ELEMENTS
    def evaluate(self, create_steps, submit_job, read_output, number_of_cells):

        applied_force = 1  # [N]. This is also defined in evaluation.py : def read_results()

        generate_wireframe(self.filename, truss=self.truss)
        define_material(self.filename, material_name=self.material['name'], young_modulus=self.material['E_Modulus'],
                        poisson_ratio=self.material['poisson_ratio'])
        assign_material_to_wire(self.filename, self.truss, material_name=self.material['name'])  # "MED610"2.5e9, 0.33
        mesh_part(self.filename, mesh_size=self.truss.cell_size / 3)
        list_of_steps = list()

        # CREATE THE DIFFERENT STEPS/LOADING CONDITIONS
        if create_steps:

            # accuracy = 0.2 * self.truss.cell_size
            accuracy = 0
            if self.truss.name == "pyramids":
                accuracy = self.truss.cell_size * 0.55
            # SIGMA_Z
            step_name = "SIGMA_Z"
            create_static_step(self.filename, step_name=step_name)
            vertices = self.truss.find_points_in_plane(axis="z",
                                                       axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                                       accuracy=accuracy)
            define_loadings(self.filename, step_name=step_name, affix="", vertices=vertices,
                            load_type="ConcentratedForce", load_type_letter="f",
                            force=[0, 0, -applied_force / len(vertices)])

            define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
                                       vertices=self.truss.find_points_in_plane(axis="z",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u3=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
                                       vertices=self.truss.find_points_in_plane(axis="x",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u1=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
                                       vertices=self.truss.find_points_in_plane(axis="y",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
```

```
                            direction='u2=0')
    for step in list_of_steps:
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
        deactivate_loadings(self.filename, step_name=step_name, deactivate_step_name=step)
    request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_Z")
    list_of_steps.append(step_name)

    # SIGMA_Y

    step_name = "SIGMA_Y"
    create_static_step(self.filename, step_name=step_name)
    vertices = self.truss.find_points_in_plane(axis="y",
                                               axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                               accuracy=accuracy)
    define_loadings(self.filename, step_name=step_name, affix="", vertices=vertices,
                    load_type="ConcentratedForce", load_type_letter="f",
                    force=[0, -applied_force / len(vertices), 0])
    define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
                               vertices=self.truss.find_points_in_plane(axis="y",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u2=0')
    define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
                               vertices=self.truss.find_points_in_plane(axis="z",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u3=0')
    define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
                               vertices=self.truss.find_points_in_plane(axis="x",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u1=0')
    for step in list_of_steps:
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
        deactivate_loadings(self.filename, step_name=step_name, deactivate_step_name=step)
    request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_Y")
    list_of_steps.append(step_name)

    # SIGMA_X

    step_name = "SIGMA_X"
    create_static_step(self.filename, step_name=step_name)
    vertices = self.truss.find_points_in_plane(axis="x",
                                               axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                               accuracy=accuracy)
    define_loadings(self.filename, step_name=step_name, affix="", vertices=vertices,
                    load_type="ConcentratedForce", load_type_letter="f",
                    force=[-applied_force / len(vertices), 0, 0])
    define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
                               vertices=self.truss.find_points_in_plane(axis="x",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u1=0')
    define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
                               vertices=self.truss.find_points_in_plane(axis="y",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u2=0')
    define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
                               vertices=self.truss.find_points_in_plane(axis="z",
                                                                        axis_value=-0.5 * self.truss.cell_size,
                                                                        accuracy=accuracy),
                               direction='u3=0')
    for step in list_of_steps:
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
        deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
        deactivate_loadings(self.filename, step_name=step_name, deactivate_step_name=step)
    request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_X")
    list_of_steps.append(step_name)
```

```
            # TAU_XZ

            step_name = "TAU_XZ"
            create_static_step(self.filename, step_name=step_name)
            vertices = self.truss.find_points_in_plane(axis="z",
                                                       axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                                       accuracy=accuracy)
            define_loadings(self.filename, step_name=step_name, affix="_Z", vertices=vertices,
                            load_type="ConcentratedForce", load_type_letter="f",
                            force=[applied_force / len(vertices), 0, 0])
            vertices = self.truss.find_points_in_plane(axis="x",
                                                       axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                                       accuracy=accuracy)
            define_loadings(self.filename, step_name=step_name, affix="_X", vertices=vertices,
                            load_type="ConcentratedForce", load_type_letter="f",
                            force=[0, 0, applied_force / len(vertices)])
            define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
                                       vertices=self.truss.find_points_in_plane(axis="z",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u1=0, ur2=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
                                       vertices=self.truss.find_points_in_plane(axis="x",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u3=0, ur2=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
                                       vertices=self.truss.find_points_in_plane(axis="y",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u2=0')
            for step in list_of_steps:
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
                deactivate_loadings(self.filename, step_name=str(step_name) + "_X", deactivate_step_name=step)
                deactivate_loadings(self.filename, step_name=str(step_name) + "_Z", deactivate_step_name=step)
            request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_XZ")
            list_of_steps.append(step_name)

            # TAU_YZ

            step_name = "TAU_YZ"
            create_static_step(self.filename, step_name=step_name)
            vertices = self.truss.find_points_in_plane(axis="z",
                                                       axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                                       accuracy=accuracy)
            define_loadings(self.filename, step_name=step_name, affix="_Z", vertices=vertices,
                            load_type="ConcentratedForce", load_type_letter="f",
                            force=[0, applied_force / len(vertices), 0])
            vertices = self.truss.find_points_in_plane(axis="y",
                                                       axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
                                                       accuracy=accuracy)
            define_loadings(self.filename, step_name=step_name, affix="_Y", vertices=vertices,
                            load_type="ConcentratedForce", load_type_letter="f",
                            force=[0, 0, applied_force / len(vertices)])
            define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
                                       vertices=self.truss.find_points_in_plane(axis="z",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u2=0, ur1=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
                                       vertices=self.truss.find_points_in_plane(axis="y",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u3=0, ur1=0')
            define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
                                       vertices=self.truss.find_points_in_plane(axis="x",
                                                                                axis_value=-0.5 * self.truss.cell_size,
                                                                                accuracy=accuracy),
                                       direction='u1=0')
            for step in list_of_steps:
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
                deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
```

```
822                 deactivate_loadings(self.filename, step_name=str(step_name) + "_Z", deactivate_step_name=step)
823                 deactivate_loadings(self.filename, step_name=str(step_name) + "_Y", deactivate_step_name=step)
824             request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_YZ")
825             list_of_steps.append(step_name)
826 
827             # TAU_XY
828 
829             step_name = "TAU_XY"
830             create_static_step(self.filename, step_name=step_name)
831             vertices = self.truss.find_points_in_plane(axis="y",
832                                                        axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
833                                                        accuracy=accuracy)
834             define_loadings(self.filename, step_name=step_name, affix="_Y", vertices=vertices,
835                             load_type="ConcentratedForce", load_type_letter="f",
836                             force=[applied_force / len(vertices), 0, 0])
837             vertices = self.truss.find_points_in_plane(axis="x",
838                                                        axis_value=(number_of_cells - 0.5) * self.truss.cell_size,
839                                                        accuracy=accuracy)
840             define_loadings(self.filename, step_name=step_name, affix="_X", vertices=vertices,
841                             load_type="ConcentratedForce", load_type_letter="f",
842                             force=[0, applied_force / len(vertices), 0])
843             define_boundary_conditions(self.filename, step_name=step_name, affix='_Y',
844                                        vertices=self.truss.find_points_in_plane(axis="y",
845                                                                                 axis_value=-0.5 * self.truss.cell_size,
846                                                                                 accuracy=accuracy),
847                                        direction='u1=0, ur3=0')
848             define_boundary_conditions(self.filename, step_name=step_name, affix='_X',
849                                        vertices=self.truss.find_points_in_plane(axis="x",
850                                                                                 axis_value=-0.5 * self.truss.cell_size,
851                                                                                 accuracy=accuracy),
852                                        direction='u2=0, ur3=0')
853             define_boundary_conditions(self.filename, step_name=step_name, affix='_Z',
854                                        vertices=self.truss.find_points_in_plane(axis="z",
855                                                                                 axis_value=-0.5 * self.truss.cell_size,
856                                                                                 accuracy=accuracy),
857                                        direction='u3=0')
858             for step in list_of_steps:
859                 deactivate_boundary_conditions(self.filename, step_name=step_name + "_X", deactivate_step_name=step)
860                 deactivate_boundary_conditions(self.filename, step_name=step_name + "_Y", deactivate_step_name=step)
861                 deactivate_boundary_conditions(self.filename, step_name=step_name + "_Z", deactivate_step_name=step)
862                 deactivate_loadings(self.filename, step_name=str(step_name) + "_Y", deactivate_step_name=step)
863                 deactivate_loadings(self.filename, step_name=str(step_name) + "_X", deactivate_step_name=step)
864             request_field_output(self.filename, step_name=step_name, output_name="OUTPUT_XY")
865             list_of_steps.append(step_name)
866 
867         if submit_job:
868             submit(self.filename, job_name=self.filename[1])
869         if read_output:
870             read_odb_step(self.filename, 'SIGMA_Z', job_name=self.filename[1])
871             read_odb_step(self.filename, 'SIGMA_Y', job_name=self.filename[1])
872             read_odb_step(self.filename, 'SIGMA_X', job_name=self.filename[1])
873             read_odb_step(self.filename, 'TAU_YZ', job_name=self.filename[1])
874             read_odb_step(self.filename, 'TAU_XZ', job_name=self.filename[1])
875             read_odb_step(self.filename, 'TAU_XY', job_name=self.filename[1])
876 
877 # EXTRACTS THE RESULTS AS A DICTIONARY(PYTHON STRUCTURE) IN A BINARY PICKLE FILE
878     def pickle_dump(self):
879 
880         file = open("".join(self.filename), "a")
881         file.write("result_file = open('" + self.filename[0] + self.filename[1] + "_results', 'wb')\n"
882                    "pickle.dump(results, result_file)\n"
883                    "result_file.close()\n")
884         file.close()
```

A.4.4 *Class_Cell.py*

```
"""
This class defines the cell which will be inserted into the truss.
The object contains the needed nodes and the connections in between, which are dependent of the nodes to evade errors.
"""


class Cell:
    def __init__(self, name, pore_size, nodes, connections, strut_thicknesses):
        self.name = name                        # Name of the cell. F.e. "cube"
        self.pore_size = pore_size              # Should be determined once per cell-> Maybe determine roundness of pore
        self.nodes = nodes                      # Syntax: [["node1x", "node1y", node1z"], ...]
        self.connections = connections          # Syntax: [["node1", "node2", "thickness"], ...]
        self.strut_thicknesses = strut_thicknesses
```

A.4.5 *Class_Truss.py*

```
"""
This class defines the truss which will be generated with abaqus script.
The object contains the corner nodes for each cell, and the cell that is attached on every node.
"""


class Truss:
    def __init__(self, name, nodes, cells, cell_size, number_of_cells):
        self.name = name                    # Name of the cells. F.e. "cubetruss"
        self.nodes = nodes                  # Syntax: [["node1x", "node1y", "node1z", "self.cells[0]"], ...]
        self.cells = cells                  # Syntax: ["Cell1","Cell2",...]
        self.cell_size = cell_size          # Unit length of the smallest unit cell
        self.number_of_cells = number_of_cells

# FINDS THE POINTS IN A PLANE DEFINED BY THE NORMAL AXIS AND THE COORDINATE WHERE IT INTERSECTS WITH SAID AXIS
    def find_points_in_plane(self, axis, axis_value, accuracy):

        if axis == "x":
            array_position = 0
        if axis == "y":
            array_position = 1
        if axis == "z":
            array_position = 2

        list_of_points = list()

        for truss_nodes in self.nodes:
                for cell_nodes in truss_nodes[3].nodes:
                    node = [x + y for x, y in zip(truss_nodes, cell_nodes)]
                    if axis_value - accuracy <= node[array_position] <= \
                       axis_value + accuracy:
                        check = True
                        for point in list_of_points:
                            if abs(node[0] - point[0]) < 1e-3 and \
                               abs(node[1] - point[1]) < 1e-3 and \
                               abs(node[2] - point[2]) < 1e-3:
                                check = False
                        if check:
                            list_of_points.append(node)
        return list_of_points

# FINDS THE POINTS IN A SPACE DEFINED BY BOUNDARIES FOR EACH OF THE THREE COORDINATE DIRECTIONS
    def find_points_in_space(self, boundaries):

        list_of_points = list()

        for truss_nodes in self.nodes:
            for cell_nodes in truss_nodes[3].nodes:
                node = [x + y for x, y in zip(truss_nodes, cell_nodes)]
                if boundaries[0][0] <= node[0] <= boundaries[0][1]:
                    if boundaries[1][0] <= node[1] <= boundaries[1][1]:
                        if boundaries[2][0] <= node[2] <= boundaries[2][1]:
                            check = True
                            for point in list_of_points:
                                if abs(node[0] - point[0]) < 1e-3 and \
                                   abs(node[1] - point[1]) < 1e-3 and \
                                   abs(node[2] - point[2]) < 1e-3:
                                    check = False
                            if check:
                                list_of_points.append(node)

        return list_of_points
```

A.4.6 *library_cell.py*

```
"""
This file creates and returns the nodes of a cell and defines the connections between the nodes.
"""
import math
from Class_Cell import Cell
import csv_reader


def generate_cell(cell_name, affix, strut_thicknesses, cell_size, ratio):
    # NODES: Describe the nodes in cartesian coordinates.
    # ratio = 0.8
    # ratio = list()
    # ratio.append(0.3)   # 0-1    --> 0.3
    # ratio.append(0.4)   # 0-0.5  --> 0.8
    # ratio.append(0.05)  # 0-0.33 --> 0.15
    if type(ratio) == list:
        if len(ratio) == 1:
            ratio = ratio[0]
    pore_size = list()
    if cell_name == "truncated_cuboctahedron":
        sizing = 1 / (2 * (1 + 2 * math.sqrt(2)))
        a = sizing
        b = sizing * (1 + math.sqrt(2))
        c = sizing * (1 + 2 * math.sqrt(2))
        cell_node_coordinates = [
            [-a, -b, -c], [a, -b, -c], [b, -a, -c], [b, a, -c], [a, b, -c], [-a, b, -c], [-b, a, -c], [-b, -a, -c],
            [-a, -c, -b], [a, -c, -b], [c, -a, -b], [c, a, -b], [a, c, -b], [-a, c, -b], [-c, a, -b], [-c, -a, -b],
            [-b, -c, -a], [b, -c, -a], [c, -b, -a], [c, b, -a], [b, c, -a], [-b, c, -a], [-c, b, -a], [-c, -b, -a],
            [-b, -c, a], [b, -c, a], [c, -b, a], [c, b, a], [b, c, a], [-b, c, a], [-c, b, a], [-c, -b, a],
            [-a, -c, b], [a, -c, b], [c, -a, b], [c, a, b], [a, c, b], [-a, c, b], [-c, a, b], [-c, -a, b],
            [-a, -b, c], [a, -b, c], [b, -a, c], [b, a, c], [a, b, c], [-a, b, c], [-b, a, c], [-b, -a, c]]

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[0]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 4, strut_thicknesses[0]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[0]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 0, strut_thicknesses[0]],
                            [0, 8, strut_thicknesses[0]],
                            [1, 9, strut_thicknesses[0]],
                            [2, 10, strut_thicknesses[0]],
                            [3, 11, strut_thicknesses[0]],
                            [4, 12, strut_thicknesses[0]],
                            [5, 13, strut_thicknesses[0]],
                            [6, 14, strut_thicknesses[0]],
                            [7, 15, strut_thicknesses[0]],
                            [8, 9, strut_thicknesses[0]],
                            [10, 11, strut_thicknesses[0]],
                            [12, 13, strut_thicknesses[0]],
                            [14, 15, strut_thicknesses[0]],
                            [8, 16, strut_thicknesses[0]],
                            [9, 17, strut_thicknesses[0]],
                            [10, 18, strut_thicknesses[0]],
                            [11, 19, strut_thicknesses[0]],
                            [12, 20, strut_thicknesses[0]],
                            [13, 21, strut_thicknesses[0]],
                            [14, 22, strut_thicknesses[0]],
                            [15, 23, strut_thicknesses[0]],
                            [17, 18, strut_thicknesses[0]],
                            [19, 20, strut_thicknesses[0]],
                            [21, 22, strut_thicknesses[0]],
                            [23, 16, strut_thicknesses[0]],
                            [16, 24, strut_thicknesses[0]],
                            [17, 25, strut_thicknesses[0]],
                            [18, 26, strut_thicknesses[0]],
                            [19, 27, strut_thicknesses[0]],
                            [20, 28, strut_thicknesses[0]],
                            [21, 29, strut_thicknesses[0]],
```

```
                            [22, 30, strut_thicknesses[0]],
                            [23, 31, strut_thicknesses[0]],
                            [25, 26, strut_thicknesses[0]],
                            [27, 28, strut_thicknesses[0]],
                            [29, 30, strut_thicknesses[0]],
                            [31, 24, strut_thicknesses[0]],
                            [24, 32, strut_thicknesses[0]],
                            [25, 33, strut_thicknesses[0]],
                            [26, 34, strut_thicknesses[0]],
                            [27, 35, strut_thicknesses[0]],
                            [28, 36, strut_thicknesses[0]],
                            [29, 37, strut_thicknesses[0]],
                            [30, 38, strut_thicknesses[0]],
                            [31, 39, strut_thicknesses[0]],
                            [32, 33, strut_thicknesses[0]],
                            [34, 35, strut_thicknesses[0]],
                            [36, 37, strut_thicknesses[0]],
                            [38, 39, strut_thicknesses[0]],
                            [32, 40, strut_thicknesses[0]],
                            [33, 41, strut_thicknesses[0]],
                            [34, 42, strut_thicknesses[0]],
                            [35, 43, strut_thicknesses[0]],
                            [36, 44, strut_thicknesses[0]],
                            [37, 45, strut_thicknesses[0]],
                            [38, 46, strut_thicknesses[0]],
                            [39, 47, strut_thicknesses[0]],
                            [40, 41, strut_thicknesses[0]],
                            [41, 42, strut_thicknesses[0]],
                            [42, 43, strut_thicknesses[0]],
                            [43, 44, strut_thicknesses[0]],
                            [44, 45, strut_thicknesses[0]],
                            [45, 46, strut_thicknesses[0]],
                            [46, 47, strut_thicknesses[0]],
                            [47, 40, strut_thicknesses[0]]]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(2 * a)
        pore_size.append(math.sqrt(3) / 2 * a)
        pore_size.append((math.sqrt(2) + 1) * a)

    elif cell_name == "truncated_cube":

        tl = ratio / 2  # standing for truncated_length
        cell_node_coordinates = [[0, tl, 0], [tl, 0, 0], [1 - tl, 0, 0], [1, tl, 0], [1, 1 - tl, 0], [1 - tl, 1, 0],
                                 [tl, 1, 0], [0, 1 - tl, 0], [0, 0, tl], [1, 0, tl], [1, 1, tl], [0, 1, tl],
                                 [0, 0, 1 - tl], [1, 0, 1 - tl], [1, 1, 1 - tl], [0, 1, 1 - tl], [0, tl, 1], [tl, 0, 1],
                                 [1 - tl, 0, 1], [1, tl, 1], [1, 1 - tl, 1], [1 - tl, 1, 1], [tl, 1, 1], [0, 1 - tl, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[2]],
                            [3, 4, strut_thicknesses[3]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[1]],
                            [6, 7, strut_thicknesses[2]],
                            [7, 0, strut_thicknesses[3]],
                            [0, 8, strut_thicknesses[4]],
                            [8, 1, strut_thicknesses[5]],
                            [2, 9, strut_thicknesses[6]],
                            [9, 3, strut_thicknesses[4]],
                            [4, 10, strut_thicknesses[7]],
                            [10, 5, strut_thicknesses[6]],
                            [6, 11, strut_thicknesses[5]],
                            [11, 7, strut_thicknesses[7]],
                            [8, 12, strut_thicknesses[8]],
                            [9, 13, strut_thicknesses[8]],
                            [10, 14, strut_thicknesses[8]],
                            [11, 15, strut_thicknesses[8]],
                            [12, 16, strut_thicknesses[7]],
                            [12, 17, strut_thicknesses[6]],
                            [17, 18, strut_thicknesses[1]],
```

```
                            [18, 13, strut_thicknesses[5]],
                            [13, 19, strut_thicknesses[7]],
                            [19, 20, strut_thicknesses[3]],
                            [20, 14, strut_thicknesses[4]],
                            [14, 21, strut_thicknesses[5]],
                            [21, 22, strut_thicknesses[1]],
                            [22, 15, strut_thicknesses[6]],
                            [15, 23, strut_thicknesses[4]],
                            [23, 16, strut_thicknesses[3]],
                            [16, 17, strut_thicknesses[0]],
                            [18, 19, strut_thicknesses[2]],
                            [20, 21, strut_thicknesses[0]],
                            [22, 23, strut_thicknesses[2]]]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(2 / 3 * math.sqrt(2) * tl)
        if 1 - 2 * tl >= math.sqrt(2) * tl:
            pore_size.append(1)
        else:
            pore_size.append(math.sqrt(2) / 2 * (1 - tl))

    elif cell_name == "octetrahedron":
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [1 / 2, 1 / 2, 0],
                                 [1 / 2, 0, 1 / 2], [1, 1 / 2, 1 / 2], [1 / 2, 1, 1 / 2], [0, 1 / 2, 1 / 2],
                                 [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [1 / 2, 1 / 2, 1], ]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 4, strut_thicknesses[0]],
                            [1, 4, strut_thicknesses[1]],
                            [2, 4, strut_thicknesses[0]],
                            [3, 4, strut_thicknesses[1]],
                            [0, 5, strut_thicknesses[2]],
                            [1, 5, strut_thicknesses[3]],
                            [1, 6, strut_thicknesses[4]],
                            [2, 6, strut_thicknesses[5]],
                            [2, 7, strut_thicknesses[3]],
                            [3, 7, strut_thicknesses[2]],
                            [3, 8, strut_thicknesses[5]],
                            [0, 8, strut_thicknesses[4]],
                            [5, 9, strut_thicknesses[3]],
                            [5, 10, strut_thicknesses[2]],
                            [6, 10, strut_thicknesses[5]],
                            [6, 11, strut_thicknesses[4]],
                            [7, 11, strut_thicknesses[2]],
                            [7, 12, strut_thicknesses[3]],
                            [8, 12, strut_thicknesses[4]],
                            [8, 9, strut_thicknesses[5]],
                            [9, 13, strut_thicknesses[0]],
                            [10, 13, strut_thicknesses[1]],
                            [11, 13, strut_thicknesses[0]],
                            [12, 13, strut_thicknesses[1]],
                            [5, 6, strut_thicknesses[0]],
                            [6, 7, strut_thicknesses[1]],
                            [7, 8, strut_thicknesses[0]],
                            [8, 5, strut_thicknesses[1]],
                            [4, 5, strut_thicknesses[5]],
                            [4, 6, strut_thicknesses[2]],
                            [4, 7, strut_thicknesses[4]],
                            [4, 8, strut_thicknesses[3]],
                            [5, 13, strut_thicknesses[4]],
                            [6, 13, strut_thicknesses[3]],
                            [7, 13, strut_thicknesses[5]],
                            [8, 13, strut_thicknesses[2]],
                            ]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(math.sqrt(2) / 3)

    elif cell_name == "void_octetrahedron":
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [1 / 2, 1 / 2, 0],
                                 [1 / 2, 0, 1 / 2], [1, 1 / 2, 1 / 2], [1 / 2, 1, 1 / 2], [0, 1 / 2, 1 / 2],
                                 [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [1 / 2, 1 / 2, 1], ]
        for x in range(0, len(cell_node_coordinates)):
```

```
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 4, strut_thicknesses[0]],
                            [1, 4, strut_thicknesses[1]],
                            [2, 4, strut_thicknesses[0]],
                            [3, 4, strut_thicknesses[1]],
                            [0, 5, strut_thicknesses[2]],
                            [1, 5, strut_thicknesses[3]],
                            [1, 6, strut_thicknesses[4]],
                            [2, 6, strut_thicknesses[5]],
                            [2, 7, strut_thicknesses[3]],
                            [3, 7, strut_thicknesses[2]],
                            [3, 8, strut_thicknesses[5]],
                            [0, 8, strut_thicknesses[4]],
                            [5, 9, strut_thicknesses[3]],
                            [5, 10, strut_thicknesses[2]],
                            [6, 10, strut_thicknesses[5]],
                            [6, 11, strut_thicknesses[4]],
                            [7, 11, strut_thicknesses[2]],
                            [7, 12, strut_thicknesses[3]],
                            [8, 12, strut_thicknesses[4]],
                            [8, 9, strut_thicknesses[5]],
                            [9, 13, strut_thicknesses[0]],
                            [10, 13, strut_thicknesses[1]],
                            [11, 13, strut_thicknesses[0]],
                            [12, 13, strut_thicknesses[1]]]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(math.sqrt(2) / 3 * 2)

    elif cell_name == "diamond":
        cell_node_coordinates = [[0, 0, 0], [1 / 2, 1 / 2, 0], [1, 1, 0], [1 / 2, 0, 1 / 2], [1, 1 / 2, 1 / 2],
                                 [1 / 2, 1, 1 / 2], [0, 1 / 2, 1 / 2], [1, 0, 1], [1 / 2, 1 / 2, 1],
                                 [0, 1, 1], [1 / 4, 1 / 4, 1 / 4], [3 / 4, 3 / 4, 1 / 4], [3 / 4, 1 / 4, 3 / 4],
                                 [1 / 4, 3 / 4, 3 / 4], ]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        # node_connections = [
        #                     [0, 10, strut_multiplicator[0]],
        #                     [1, 10, strut_multiplicator[1]],
        #                     [1, 11, strut_multiplicator[2]],
        #                     [2, 11, strut_multiplicator[3]],
        #                     [3, 10, strut_multiplicator[4]],
        #                     [6, 10, strut_multiplicator[5]],
        #                     [4, 11, strut_multiplicator[6]],
        #                     [5, 11, strut_multiplicator[7]],
        #                     [3, 12, strut_multiplicator[8]],
        #                     [4, 12, strut_multiplicator[9]],
        #                     [7, 12, strut_multiplicator[10]],
        #                     [8, 12, strut_multiplicator[11]],
        #                     [5, 13, strut_multiplicator[12]],
        #                     [6, 13, strut_multiplicator[13]],
        #                     [8, 13, strut_multiplicator[14]],
        #                     [9, 13, strut_multiplicator[15]],
        #                     ]
        node_connections = [
            [0, 10, strut_thicknesses[0]],
            [1, 10, strut_thicknesses[1]],
            [1, 11, strut_thicknesses[0]],
            [2, 11, strut_thicknesses[1]],
            [3, 10, strut_thicknesses[2]],
            [6, 10, strut_thicknesses[3]],
            [4, 11, strut_thicknesses[2]],
            [5, 11, strut_thicknesses[3]],
            [3, 12, strut_thicknesses[0]],
            [4, 12, strut_thicknesses[1]],
            [7, 12, strut_thicknesses[2]],
            [8, 12, strut_thicknesses[3]],
            [5, 13, strut_thicknesses[1]],
            [6, 13, strut_thicknesses[0]],
            [8, 13, strut_thicknesses[2]],
            [9, 13, strut_thicknesses[3]],
```

```
        ]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(3 / 8)

    elif cell_name == "cube":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[1]],
                            [0, 4, strut_thicknesses[2]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[1]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[1]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 6, strut_thicknesses[2]],
                            [3, 7, strut_thicknesses[2]]]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(1)

    elif cell_name == "inv_cube":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1/2, 1/2, 1/2], [1/2, 1/2, 0], [1/2, 0, 1/2], [0, 1/2, 1/2], [1/2, 1, 1/2],
                                 [1, 1/2, 1/2], [1/2, 1/2, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [0, 2, strut_thicknesses[1]],
                            [0, 6, strut_thicknesses[0]],
                            [0, 4, strut_thicknesses[1]],
                            [0, 5, strut_thicknesses[2]],
                            [0, 3, strut_thicknesses[2]]]
        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(1)

    elif cell_name == "pyramid":
        # NODES: Describe the nodes in cartesian coordinates.
        p = ratio
        q = (1 - p) / 2
        cell_node_coordinates = [[-q, -q, 0], [1 + q, -q, 0], [1 + q, 1 + q, 0], [-q, 1 + q, 0],
                                 [q, q, 1], [1 - q, q, 1], [1 - q, 1 - q, 1], [q, 1 - q, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[1]],
                            [0, 4, strut_thicknesses[2]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[1]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[1]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 6, strut_thicknesses[2]],
                            [3, 7, strut_thicknesses[2]]]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        if (math.sqrt(1 / 4 + (1 - p / 2) ** 2) * math.sin(math.atan(2 / (2 - p)) - math.atan(1 / (2 - p)))) >= 1 / 2:
            pore_size.append(1)
```

```
372             else:
373                 pore_size.append(2 * (1 - p / 2) * math.tan(math.atan(1 / (1 - p)) / 2))
374 
375         elif cell_name == "pyramid_inv":
376             # NODES: Describe the nodes in cartesian coordinates.
377             p = ratio
378             q = (1 - p) / 2
379             cell_node_coordinates = [[q, q, 0], [1 - q, q, 0], [1 - q, 1 - q, 0], [q, 1 - q, 0],
380                                      [-q, -q, 1], [1 + q, -q, 1], [1 + q, 1 + q, 1], [-q, 1 + q, 1]]
381             for x in range(0, len(cell_node_coordinates)):
382                 for y in range(0, 3):
383                     cell_node_coordinates[x][y] -= 1 / 2
384 
385             # CONNECTIONS: Describes the connections via the nodes.
386             node_connections = [[0, 1, strut_thicknesses[0]],
387                                 [1, 2, strut_thicknesses[1]],
388                                 [2, 3, strut_thicknesses[0]],
389                                 [3, 0, strut_thicknesses[1]],
390                                 [0, 4, strut_thicknesses[2]],
391                                 [4, 5, strut_thicknesses[0]],
392                                 [5, 6, strut_thicknesses[1]],
393                                 [6, 7, strut_thicknesses[0]],
394                                 [7, 4, strut_thicknesses[1]],
395                                 [1, 5, strut_thicknesses[2]],
396                                 [2, 6, strut_thicknesses[2]],
397                                 [3, 7, strut_thicknesses[2]]]
398 
399             # PORE SIZE: Describes the pore size of the Cell in standard size 1.
400             # It is described by the diameter of the biggest fitting sphere in the cell
401             if (math.sqrt(1 / 4 + (1 - p / 2) ** 2) * math.sin(math.atan(2 / (2 - p)) - math.atan(1 / (2 - p)))) >= 1 / 2:
402                 pore_size.append(1)
403             else:
404                 pore_size.append(2 * (1 - p / 2) * math.tan(math.atan(1 / (1 - p)) / 2))
405 
406         elif cell_name == "pyramid_twist":
407             # NODES: Describe the nodes in cartesian coordinates.
408             p = ratio
409             q = (1 - p) / 2
410             cell_node_coordinates = [[-q, q, 0], [1 + q, q, 0], [1 + q, 1 - q, 0], [-q, 1 - q, 0],
411                                      [q, -q, 1], [1 - q, -q, 1], [1 - q, 1 + q, 1], [q, 1 + q, 1]]
412             for x in range(0, len(cell_node_coordinates)):
413                 for y in range(0, 3):
414                     cell_node_coordinates[x][y] -= 1 / 2
415 
416             # CONNECTIONS: Describes the connections via the nodes.
417             node_connections = [[0, 1, strut_thicknesses[0]],
418                                 [1, 2, strut_thicknesses[1]],
419                                 [2, 3, strut_thicknesses[0]],
420                                 [3, 0, strut_thicknesses[1]],
421                                 [0, 4, strut_thicknesses[2]],
422                                 [4, 5, strut_thicknesses[0]],
423                                 [5, 6, strut_thicknesses[1]],
424                                 [6, 7, strut_thicknesses[0]],
425                                 [7, 4, strut_thicknesses[1]],
426                                 [1, 5, strut_thicknesses[2]],
427                                 [2, 6, strut_thicknesses[2]],
428                                 [3, 7, strut_thicknesses[2]]]
429 
430             # PORE SIZE: Describes the pore size of the Cell in standard size 1.
431             # It is described by the diameter of the biggest fitting sphere in the cell
432             if (math.sqrt(1 / 4 + (1 - p / 2) ** 2) * math.sin(math.atan(2 / (2 - p)) - math.atan(1 / (2 - p)))) >= 1 / 2:
433                 pore_size.append(1)
434             else:
435                 pore_size.append(2 * (1 - p / 2) * math.tan(math.atan(1 / (1 - p)) / 2))
436 
437         elif cell_name == "pyramid_twist_inv":
438             # NODES: Describe the nodes in cartesian coordinates.
439             p = ratio
440             q = (1 - p) / 2
441             cell_node_coordinates = [[q, -q, 0], [1 - q, -q, 0], [1 - q, 1 + q, 0], [q, 1 + q, 0],
442                                      [-q, q, 1], [1 + q, q, 1], [1 + q, 1 - q, 1], [-q, 1 - q, 1]]
443             for x in range(0, len(cell_node_coordinates)):
444                 for y in range(0, 3):
445                     cell_node_coordinates[x][y] -= 1 / 2
446 
```

```
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[1]],
                            [0, 4, strut_thicknesses[2]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[1]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[1]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 6, strut_thicknesses[2]],
                            [3, 7, strut_thicknesses[2]]]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        if (math.sqrt(1 / 4 + (1 - p / 2) ** 2) * math.sin(math.atan(2 / (2 - p)) - math.atan(1 / (2 - p)))) >= 1 / 2:
            pore_size.append(1)
        else:
            pore_size.append(2 * (1 - p / 2) * math.tan(math.atan(1 / (1 - p)) / 2))

    elif cell_name == "face_diagonal_cube":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1]]
        # strut_multiplicator = 0.2   # unit meters
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [
            [0, 1, strut_thicknesses[0]],
            [1, 2, strut_thicknesses[1]],
            [2, 3, strut_thicknesses[0]],
            [3, 0, strut_thicknesses[1]],
            [0, 4, strut_thicknesses[2]],
            [4, 5, strut_thicknesses[0]],
            [5, 6, strut_thicknesses[1]],
            [6, 7, strut_thicknesses[0]],
            [7, 4, strut_thicknesses[1]],
            [1, 5, strut_thicknesses[2]],
            [2, 6, strut_thicknesses[2]],
            [3, 7, strut_thicknesses[2]],
            # [1, 3, strut_multiplicator[3]],
            # [1, 4, strut_multiplicator[4]],
            # [1, 6, strut_multiplicator[5]],
            # [3, 4, strut_multiplicator[6]],
            # [3, 6, strut_multiplicator[7]],
            # [4, 6, strut_multiplicator[8]],
            [1, 3, strut_thicknesses[3]],
            [1, 4, strut_thicknesses[4]],
            [1, 6, strut_thicknesses[5]],
            [3, 4, strut_thicknesses[5]],
            [3, 6, strut_thicknesses[4]],
            [4, 6, strut_thicknesses[3]],
        ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(math.sqrt(2) / 3)  # ONLY estimated, needs to be determined

    elif cell_name == "face_diagonal_cube_inv":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1]]
        # strut_multiplicator = 0.2   # unit meters
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [
            [0, 1, strut_thicknesses[0]],
            [1, 2, strut_thicknesses[1]],
            [2, 3, strut_thicknesses[0]],
            [3, 0, strut_thicknesses[1]],
```

```
            [0, 4, strut_thicknesses[2]],
            [4, 5, strut_thicknesses[0]],
            [5, 6, strut_thicknesses[1]],
            [6, 7, strut_thicknesses[0]],
            [6, 7, strut_thicknesses[0]],
            [7, 4, strut_thicknesses[1]],
            [1, 5, strut_thicknesses[2]],
            [2, 6, strut_thicknesses[2]],
            [3, 7, strut_thicknesses[2]],
            # [0, 2, strut_multiplicator[3]],
            # [0, 5, strut_multiplicator[4]],
            # [0, 7, strut_multiplicator[5]],
            # [2, 5, strut_multiplicator[6]],
            # [2, 7, strut_multiplicator[7]],
            # [5, 7, strut_multiplicator[8]],
            [0, 2, strut_thicknesses[3]],
            [0, 5, strut_thicknesses[4]],
            [0, 7, strut_thicknesses[5]],
            [2, 5, strut_thicknesses[5]],
            [2, 7, strut_thicknesses[4]],
            [5, 7, strut_thicknesses[3]],
        ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(math.sqrt(2) / 3)  # ONLY estimated, needs to be determined

    elif cell_name == "body_centered_cube":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 1],
                                 [1, 0, 1], [1, 1, 1], [0, 1, 1], [1 / 2, 1 / 2, 1 / 2]]
        # strut_multiplicator = 0.2   # unit meters
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[1]],
                            [0, 4, strut_thicknesses[2]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[1]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[1]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 6, strut_thicknesses[2]],
                            [3, 7, strut_thicknesses[2]],
                            [0, 8, strut_thicknesses[3]],
                            [1, 8, strut_thicknesses[3]],
                            [2, 8, strut_thicknesses[3]],
                            [3, 8, strut_thicknesses[3]],
                            [4, 8, strut_thicknesses[3]],
                            [5, 8, strut_thicknesses[3]],
                            [6, 8, strut_thicknesses[3]],
                            [7, 8, strut_thicknesses[3]],
                            # [0, 8, strut_thicknesses[3]],
                            # [1, 8, strut_thicknesses[4]],
                            # [2, 8, strut_thicknesses[5]],
                            # [3, 8, strut_thicknesses[6]],
                            # [4, 8, strut_thicknesses[5]],
                            # [5, 8, strut_thicknesses[6]],
                            # [6, 8, strut_thicknesses[3]],
                            # [7, 8, strut_thicknesses[4]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append = 2/3 * math.sqrt(2) / 2
        #  ONLY estimated, needs to be determined

    elif cell_name == "octahedron":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1 / 2, 1 / 2, 0], [1 / 2, 0, 1 / 2], [1, 1 / 2, 1 / 2],
```

```
                                [1 / 2, 1, 1 / 2], [0, 1 / 2, 1 / 2], [1 / 2, 1 / 2, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [0, 2, strut_thicknesses[1]],
                            [0, 3, strut_thicknesses[2]],
                            [0, 4, strut_thicknesses[3]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 5, strut_thicknesses[3]],
                            [3, 5, strut_thicknesses[0]],
                            [4, 5, strut_thicknesses[1]],
                            [1, 2, strut_thicknesses[4]],
                            [2, 3, strut_thicknesses[5]],
                            [3, 4, strut_thicknesses[4]],
                            [4, 1, strut_thicknesses[5]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(math.sqrt(2) / 3)

    elif cell_name == "truncated_octahedron":
        # NODES: Describe the nodes in cartesian coordinates.
        p = ratio / 2
        cell_node_coordinates = [[1 / 2 - p, 1 / 2, 0], [1 / 2, 1 / 2 - p, 0], [1 / 2 + p, 1 / 2, 0], [1 / 2,
                                                                                                         1 / 2 + p, 0],
                                 [0, 1 / 2, 1 / 2 - p], [1 / 2, 0, 1 / 2 - p], [1, 1 / 2, 1 / 2 - p], [1 / 2,
                                                                                                        1, 1 / 2 - p],
                                 [0, 1 / 2 + p, 1 / 2], [0, 1 / 2 - p, 1 / 2], [1 / 2 - p, 0, 1 / 2], [1 / 2 + p,
                                                                                                        0, 1 / 2],
                                 [1, 1 / 2 - p, 1 / 2], [1, 1 / 2 + p, 1 / 2], [1 / 2 + p, 1, 1 / 2], [1 / 2 - p,
                                                                                                        1, 1 / 2],
                                 [0, 1 / 2, 1 / 2 + p], [1 / 2, 0, 1 / 2 + p], [1, 1 / 2, 1 / 2 + p], [1 / 2,
                                                                                                        1, 1 / 2 + p],
                                 [1 / 2 - p, 1 / 2, 1], [1 / 2, 1 / 2 - p, 1], [1 / 2 + p, 1 / 2, 1], [1 / 2,
                                                                                                        1 / 2 + p, 1]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[0]],
                            [2, 3, strut_thicknesses[0]],
                            [0, 3, strut_thicknesses[0]],
                            [0, 4, strut_thicknesses[1]],
                            [1, 5, strut_thicknesses[2]],
                            [2, 6, strut_thicknesses[1]],
                            [3, 7, strut_thicknesses[2]],
                            [4, 8, strut_thicknesses[3]],
                            [4, 9, strut_thicknesses[3]],
                            [5, 10, strut_thicknesses[4]],
                            [5, 11, strut_thicknesses[4]],
                            [6, 12, strut_thicknesses[3]],
                            [6, 13, strut_thicknesses[3]],
                            [7, 14, strut_thicknesses[4]],
                            [7, 15, strut_thicknesses[4]],
                            [8, 16, strut_thicknesses[3]],
                            [9, 16, strut_thicknesses[3]],
                            [10, 17, strut_thicknesses[4]],
                            [11, 17, strut_thicknesses[4]],
                            [12, 18, strut_thicknesses[3]],
                            [13, 18, strut_thicknesses[3]],
                            [14, 19, strut_thicknesses[4]],
                            [15, 19, strut_thicknesses[4]],
                            [16, 20, strut_thicknesses[1]],
                            [17, 21, strut_thicknesses[2]],
                            [18, 22, strut_thicknesses[1]],
                            [19, 23, strut_thicknesses[2]],
                            [20, 21, strut_thicknesses[0]],
                            [21, 22, strut_thicknesses[0]],
                            [22, 23, strut_thicknesses[0]],
```

```
                            [20, 23, strut_thicknesses[0]],
                            [9, 10, strut_thicknesses[5]],
                            [11, 12, strut_thicknesses[5]],
                            [13, 14, strut_thicknesses[5]],
                            [15, 8, strut_thicknesses[5]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(ratio)

    elif cell_name == "templar_crosse":
        # p = ratio
        # p = 0.3  # 0-1
        # q = 0.4  # 0-0.5
        p = ratio[0]
        q = ratio[1] / 2
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1 / 2 - q, 1 / 2, 0], [1 / 2, 1 / 2 - q, 0], [1 / 2 + q, 1 / 2, 0],
                                 [1 / 2, 1 / 2 + q, 0],
                                 [0, 1 / 2, 1 / 2 - q], [0, 1 / 2 + q, 1 / 2], [0, 1 / 2, 1 / 2 + q],
                                 [0, 1 / 2 - q, 1 / 2],
                                 [1 / 2, 0, 1 / 2 - q], [1 / 2 - q, 0, 1 / 2], [1 / 2, 0, 1 / 2 + q],
                                 [1 / 2 + q, 0, 1 / 2],
                                 [1, 1 / 2, 1 / 2 - q], [1, 1 / 2 - q, 1 / 2], [1, 1 / 2, 1 / 2 + q],
                                 [1, 1 / 2 + q, 1 / 2],
                                 [1 / 2, 1, 1 / 2 - q], [1 / 2 + q, 1, 1 / 2], [1 / 2, 1, 1 / 2 + q],
                                 [1 / 2 - q, 1, 1 / 2],
                                 [1 / 2 - q, 1 / 2, 1], [1 / 2, 1 / 2 - q, 1], [1 / 2 + q, 1 / 2, 1],
                                 [1 / 2, 1 / 2 + q, 1],
                                 [1 / 2 * (1 - p), 1 / 2, 1 / 2 * (1 - p)], [1 / 2, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                                 [1 / 2 * (1 + p), 1 / 2, 1 / 2 * (1 - p)],
                                 [1 / 2, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],
                                 [1 / 2 * (1 - p), 1 / 2, 1 / 2 * (1 + p)], [1 / 2, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                                 [1 / 2 * (1 + p), 1 / 2, 1 / 2 * (1 + p)],
                                 [1 / 2, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],
                                 [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2], [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2],
                                 [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2],
                                 [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[0]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[0]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[0]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[0]],
                            [8, 9, strut_thicknesses[0]],
                            [9, 10, strut_thicknesses[0]],
                            [10, 11, strut_thicknesses[0]],
                            [11, 8, strut_thicknesses[0]],
                            [12, 13, strut_thicknesses[0]],
                            [13, 14, strut_thicknesses[0]],
                            [14, 15, strut_thicknesses[0]],
                            [15, 12, strut_thicknesses[0]],
                            [16, 17, strut_thicknesses[0]],
                            [17, 18, strut_thicknesses[0]],
                            [18, 19, strut_thicknesses[0]],
                            [19, 16, strut_thicknesses[0]],
                            [20, 21, strut_thicknesses[0]],
                            [21, 22, strut_thicknesses[0]],
                            [22, 23, strut_thicknesses[0]],
                            [23, 20, strut_thicknesses[0]],
                            [0, 24, strut_thicknesses[0]],
                            [1, 25, strut_thicknesses[0]],
                            [2, 26, strut_thicknesses[0]],
                            [3, 27, strut_thicknesses[0]],
                            [24, 4, strut_thicknesses[0]],
                            [25, 8, strut_thicknesses[0]],
                            [26, 12, strut_thicknesses[0]],
```

```
                            [27, 16, strut_thicknesses[0]],
                            [6, 28, strut_thicknesses[0]],
                            [10, 29, strut_thicknesses[0]],
                            [14, 30, strut_thicknesses[0]],
                            [18, 31, strut_thicknesses[0]],
                            [28, 20, strut_thicknesses[0]],
                            [29, 21, strut_thicknesses[0]],
                            [30, 22, strut_thicknesses[0]],
                            [31, 23, strut_thicknesses[0]],
                            [7, 32, strut_thicknesses[0]],
                            [32, 9, strut_thicknesses[0]],
                            [11, 33, strut_thicknesses[0]],
                            [33, 13, strut_thicknesses[0]],
                            [15, 34, strut_thicknesses[0]],
                            [34, 17, strut_thicknesses[0]],
                            [19, 35, strut_thicknesses[0]],
                            [35, 5, strut_thicknesses[0]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(0)

    elif cell_name == "templar_alt_crosse":
        # p = ratio
        # p = 0.3   # 0-1
        # q = 0.4   # 0-0.5
        # r = 0.05  # 0-0.33
        p = ratio[0]
        q = ratio[1] / 2
        r = ratio[2] / 2
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1 / 2 - q, 1 / 2, 0], [1 / 2, 1 / 2 - q, 0], [1 / 2 + q, 1 / 2, 0],
                                 [1 / 2, 1 / 2 + q, 0],  # 0-3
                                 [0, 1 / 2, 1 / 2 - q], [0, 1 / 2 + q, 1 / 2], [0, 1 / 2, 1 / 2 + q],
                                 [0, 1 / 2 - q, 1 / 2],  # 4-7
                                 [1 / 2, 0, 1 / 2 - q], [1 / 2 - q, 0, 1 / 2], [1 / 2, 0, 1 / 2 + q],
                                 [1 / 2 + q, 0, 1 / 2],  # 8-11
                                 [1, 1 / 2, 1 / 2 - q], [1, 1 / 2 - q, 1 / 2], [1, 1 / 2, 1 / 2 + q],
                                 [1, 1 / 2 + q, 1 / 2],  # 12-15
                                 [1 / 2, 1, 1 / 2 - q], [1 / 2 + q, 1, 1 / 2], [1 / 2, 1, 1 / 2 + q],
                                 [1 / 2 - q, 1, 1 / 2],  # 16-19
                                 [1 / 2 - q, 1 / 2, 1], [1 / 2, 1 / 2 - q, 1], [1 / 2 + q, 1 / 2, 1],
                                 [1 / 2, 1 / 2 + q, 1],  # 20-23
                                 ##############################################################################################
                                 [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 - p)],
                                 [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                                 [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 - p)],
                                 [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 24-27
                                 [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 + p)],
                                 [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                                 [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 + p)],
                                 [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 28-31
                                 [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 + r],
                                 [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 + r],
                                 [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 + r],
                                 [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 + r],  # 32-35
                                 ##############################################################################################
                                 [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 - p)],
                                 [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                                 [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 - p)],
                                 [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 36-39
                                 [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 + p)],
                                 [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                                 [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 + p)],
                                 [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 40-43
                                 [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 - r],
                                 [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 - r],
                                 [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 - r],
                                 [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 - r]]  # 44-47
        #                        ##############################################################################################
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2

```

```
822         # CONNECTIONS: Describes the connections via the nodes.
823         node_connections = [[0, 1, strut_thicknesses[0]],
824                             [1, 2, strut_thicknesses[0]],
825                             [2, 3, strut_thicknesses[0]],
826                             [3, 0, strut_thicknesses[0]],
827                             [4, 5, strut_thicknesses[0]],
828                             [5, 6, strut_thicknesses[0]],
829                             [6, 7, strut_thicknesses[0]],
830                             [7, 4, strut_thicknesses[0]],
831                             [8, 9, strut_thicknesses[0]],
832                             [9, 10, strut_thicknesses[0]],
833                             [10, 11, strut_thicknesses[0]],
834                             [11, 8, strut_thicknesses[0]],
835                             [12, 13, strut_thicknesses[0]],
836                             [13, 14, strut_thicknesses[0]],
837                             [14, 15, strut_thicknesses[0]],
838                             [15, 12, strut_thicknesses[0]],
839                             [16, 17, strut_thicknesses[0]],
840                             [17, 18, strut_thicknesses[0]],
841                             [18, 19, strut_thicknesses[0]],
842                             [19, 16, strut_thicknesses[0]],
843                             [20, 21, strut_thicknesses[0]],
844                             [21, 22, strut_thicknesses[0]],
845                             [22, 23, strut_thicknesses[0]],
846                             [23, 20, strut_thicknesses[0]],
847                             ##############################
848                             [0, 24, strut_thicknesses[0]],
849                             [1, 25, strut_thicknesses[0]],
850                             [2, 26, strut_thicknesses[0]],
851                             [3, 27, strut_thicknesses[0]],
852 
853                             [24, 36, strut_thicknesses[0]],
854                             [25, 37, strut_thicknesses[0]],
855                             [26, 38, strut_thicknesses[0]],
856                             [27, 39, strut_thicknesses[0]],
857 
858                             [36, 4, strut_thicknesses[0]],
859                             [37, 8, strut_thicknesses[0]],
860                             [38, 12, strut_thicknesses[0]],
861                             [39, 16, strut_thicknesses[0]],
862                             ##############################
863                             [6, 40, strut_thicknesses[0]],
864                             [10, 41, strut_thicknesses[0]],
865                             [14, 42, strut_thicknesses[0]],
866                             [18, 43, strut_thicknesses[0]],
867 
868                             [28, 40, strut_thicknesses[0]],
869                             [29, 41, strut_thicknesses[0]],
870                             [30, 42, strut_thicknesses[0]],
871                             [31, 43, strut_thicknesses[0]],
872 
873                             [28, 20, strut_thicknesses[0]],
874                             [29, 21, strut_thicknesses[0]],
875                             [30, 22, strut_thicknesses[0]],
876                             [31, 23, strut_thicknesses[0]],
877                             ##############################
878                             [7, 44, strut_thicknesses[0]],
879                             [44, 32, strut_thicknesses[0]],
880                             [32, 9, strut_thicknesses[0]],
881 
882                             [11, 45, strut_thicknesses[0]],
883                             [45, 33, strut_thicknesses[0]],
884                             [33, 13, strut_thicknesses[0]],
885 
886                             [15, 46, strut_thicknesses[0]],
887                             [46, 34, strut_thicknesses[0]],
888                             [34, 17, strut_thicknesses[0]],
889 
890                             [19, 47, strut_thicknesses[0]],
891                             [47, 35, strut_thicknesses[0]],
892                             [35, 5, strut_thicknesses[0]],
893                             ]
894 
895         # PORE SIZE: Describes the pore size of the Cell in standard size 1.
896         # It is described by the diameter of the biggest fitting sphere in the cell
```

```
            pore_size.append(0)

        elif cell_name == "templar_alt2_cross":
            # p = ratio
            # p = 0.3   # 0-1
            # q = 0.4   # 0-0.5
            # r = 0.05  # 0-0.33
            p = ratio[0]
            q = ratio[1] / 2
            r = ratio[2] / 3
            # NODES: Describe the nodes in cartesian coordinates.
            cell_node_coordinates = [[1 / 2 - q, 1 / 2 - r, 0], [1 / 2 + r, 1 / 2 - q, 0],
                                     [1 / 2 + q, 1 / 2 + r, 0], [1 / 2 - r, 1 / 2 + q, 0],  # 0-3
                                     [0, 1 / 2 + r, 1 / 2 - q], [0, 1 / 2 + q, 1 / 2 + r],
                                     [0, 1 / 2 - r, 1 / 2 + q], [0, 1 / 2 - q, 1 / 2 - r],  # 4-7
                                     [1 / 2 - r, 0, 1 / 2 - q], [1 / 2 - q, 0, 1 / 2 + r],
                                     [1 / 2 + r, 0, 1 / 2 + q], [1 / 2 + q, 0, 1 / 2 - r],  # 8-11
                                     [1, 1 / 2 - r, 1 / 2 - q], [1, 1 / 2 - q, 1 / 2 + r],
                                     [1, 1 / 2 + r, 1 / 2 + q], [1, 1 / 2 + q, 1 / 2 - r],  # 12-15
                                     [1 / 2 + r, 1, 1 / 2 - q], [1 / 2 + q, 1, 1 / 2 + r],
                                     [1 / 2 - r, 1, 1 / 2 + q], [1 / 2 - q, 1, 1 / 2 - r],  # 16-19
                                     [1 / 2 - q, 1 / 2 + r, 1], [1 / 2 - r, 1 / 2 - q, 1],
                                     [1 / 2 + q, 1 / 2 - r, 1], [1 / 2 + r, 1 / 2 + q, 1],  # 20-23
                                     ##########################################################################################
                                     [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 - p)],
                                     [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                                     [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 - p)],
                                     [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 24-27
                                     [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 + p)],
                                     [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                                     [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 + p)],
                                     [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 28-31
                                     [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 + r],
                                     [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 + r],
                                     [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 + r],
                                     [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 + r],  # 32-35
                                     ##########################################################################################
                                     [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 - p)],
                                     [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                                     [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 - p)],
                                     [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 36-39
                                     [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 + p)],
                                     [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                                     [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 + p)],
                                     [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 40-43
                                     [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 - r],
                                     [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 - r],
                                     [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 - r],
                                     [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 - r]]  # 44-47
            #                        ##########################################################################################
            for x in range(0, len(cell_node_coordinates)):
                for y in range(0, 3):
                    cell_node_coordinates[x][y] -= 1 / 2

            # CONNECTIONS: Describes the connections via the nodes.
            node_connections = [[0, 1, strut_thicknesses[0]],
                                [1, 2, strut_thicknesses[0]],
                                [2, 3, strut_thicknesses[0]],
                                [3, 0, strut_thicknesses[0]],
                                [4, 5, strut_thicknesses[0]],
                                [5, 6, strut_thicknesses[0]],
                                [6, 7, strut_thicknesses[0]],
                                [7, 4, strut_thicknesses[0]],
                                [8, 9, strut_thicknesses[0]],
                                [9, 10, strut_thicknesses[0]],
                                [10, 11, strut_thicknesses[0]],
                                [11, 8, strut_thicknesses[0]],
                                [12, 13, strut_thicknesses[0]],
                                [13, 14, strut_thicknesses[0]],
                                [14, 15, strut_thicknesses[0]],
                                [15, 12, strut_thicknesses[0]],
                                [16, 17, strut_thicknesses[0]],
                                [17, 18, strut_thicknesses[0]],
                                [18, 19, strut_thicknesses[0]],
                                [19, 16, strut_thicknesses[0]],
```

```
                            [20, 21, strut_thicknesses[0]],
                            [21, 22, strut_thicknesses[0]],
                            [22, 23, strut_thicknesses[0]],
                            [23, 20, strut_thicknesses[0]],
                            ##############################
                            [0, 24, strut_thicknesses[0]],
                            [1, 25, strut_thicknesses[0]],
                            [2, 26, strut_thicknesses[0]],
                            [3, 27, strut_thicknesses[0]],

                            [24, 36, strut_thicknesses[0]],
                            [25, 37, strut_thicknesses[0]],
                            [26, 38, strut_thicknesses[0]],
                            [27, 39, strut_thicknesses[0]],

                            [36, 4, strut_thicknesses[0]],
                            [37, 8, strut_thicknesses[0]],
                            [38, 12, strut_thicknesses[0]],
                            [39, 16, strut_thicknesses[0]],
                            ##############################
                            [6, 40, strut_thicknesses[0]],
                            [10, 41, strut_thicknesses[0]],
                            [14, 42, strut_thicknesses[0]],
                            [18, 43, strut_thicknesses[0]],

                            [28, 40, strut_thicknesses[0]],
                            [29, 41, strut_thicknesses[0]],
                            [30, 42, strut_thicknesses[0]],
                            [31, 43, strut_thicknesses[0]],

                            [28, 20, strut_thicknesses[0]],
                            [29, 21, strut_thicknesses[0]],
                            [30, 22, strut_thicknesses[0]],
                            [31, 23, strut_thicknesses[0]],
                            ##############################
                            [7, 44, strut_thicknesses[0]],
                            [44, 32, strut_thicknesses[0]],
                            [32, 9, strut_thicknesses[0]],

                            [11, 45, strut_thicknesses[0]],
                            [45, 33, strut_thicknesses[0]],
                            [33, 13, strut_thicknesses[0]],

                            [15, 46, strut_thicknesses[0]],
                            [46, 34, strut_thicknesses[0]],
                            [34, 17, strut_thicknesses[0]],

                            [19, 47, strut_thicknesses[0]],
                            [47, 35, strut_thicknesses[0]],
                            [35, 5, strut_thicknesses[0]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(0)

    elif cell_name == "templar_alt2_cross_inv":
        # p = ratio
        # p = 0.3   # 0-1
        # q = 0.4   # 0-0.5
        # r = 0.05  # 0-0.33
        p = ratio[0]
        q = ratio[1] / 2
        r = ratio[2] / 3
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1 / 2 - q, 1 / 2 - r, 0], [1 / 2 + r, 1 / 2 - q, 0],
                                 [1 / 2 + q, 1 / 2 + r, 0], [1 / 2 - r, 1 / 2 + q, 0],  # 0-3
                                 [0, 1 / 2 + r, 1 / 2 - q], [0, 1 / 2 + q, 1 / 2 + r],
                                 [0, 1 / 2 - r, 1 / 2 + q], [0, 1 / 2 - q, 1 / 2 - r],  # 4-7
                                 [1 / 2 - r, 0, 1 / 2 - q], [1 / 2 - q, 0, 1 / 2 + r],
                                 [1 / 2 + r, 0, 1 / 2 + q], [1 / 2 + q, 0, 1 / 2 - r],  # 8-11
                                 [1, 1 / 2 - r, 1 / 2 - q], [1, 1 / 2 - q, 1 / 2 + r],
                                 [1, 1 / 2 + r, 1 / 2 + q], [1, 1 / 2 + q, 1 / 2 - r],  # 12-15
                                 [1 / 2 + r, 1, 1 / 2 - q], [1 / 2 + q, 1, 1 / 2 + r],
                                 [1 / 2 - r, 1, 1 / 2 + q], [1 / 2 - q, 1, 1 / 2 - r],  # 16-19
```

```
                            [1 / 2 - q, 1 / 2 + r, 1], [1 / 2 - r, 1 / 2 - q, 1],
                            [1 / 2 + q, 1 / 2 - r, 1], [1 / 2 + r, 1 / 2 + q, 1],  # 20-23
                            ##############################################################################
                            [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 - p)],
                            [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                            [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 - p)],
                            [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 24-27
                            [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 + p)],
                            [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                            [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 + p)],
                            [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 28-31
                            [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 + r],
                            [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 + r],
                            [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 + r],
                            [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 + r],  # 32-35
                            ##############################################################################
                            [1 / 2 * (1 - p), 1 / 2 + r, 1 / 2 * (1 - p)],
                            [1 / 2 - r, 1 / 2 * (1 - p), 1 / 2 * (1 - p)],
                            [1 / 2 * (1 + p), 1 / 2 - r, 1 / 2 * (1 - p)],
                            [1 / 2 + r, 1 / 2 * (1 + p), 1 / 2 * (1 - p)],  # 36-39
                            [1 / 2 * (1 - p), 1 / 2 - r, 1 / 2 * (1 + p)],
                            [1 / 2 + r, 1 / 2 * (1 - p), 1 / 2 * (1 + p)],
                            [1 / 2 * (1 + p), 1 / 2 + r, 1 / 2 * (1 + p)],
                            [1 / 2 - r, 1 / 2 * (1 + p), 1 / 2 * (1 + p)],  # 40-43
                            [1 / 2 * (1 - p), 1 / 2 * (1 - p), 1 / 2 - r],
                            [1 / 2 * (1 + p), 1 / 2 * (1 - p), 1 / 2 - r],
                            [1 / 2 * (1 + p), 1 / 2 * (1 + p), 1 / 2 - r],
                            [1 / 2 * (1 - p), 1 / 2 * (1 + p), 1 / 2 - r]]  # 44-47
        #                   ##############################################################################
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
            cell_node_coordinates[x][2] = -cell_node_coordinates[x][2]

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[0]],
                            [2, 3, strut_thicknesses[0]],
                            [3, 0, strut_thicknesses[0]],
                            [4, 5, strut_thicknesses[0]],
                            [5, 6, strut_thicknesses[0]],
                            [6, 7, strut_thicknesses[0]],
                            [7, 4, strut_thicknesses[0]],
                            [8, 9, strut_thicknesses[0]],
                            [9, 10, strut_thicknesses[0]],
                            [10, 11, strut_thicknesses[0]],
                            [11, 8, strut_thicknesses[0]],
                            [12, 13, strut_thicknesses[0]],
                            [13, 14, strut_thicknesses[0]],
                            [14, 15, strut_thicknesses[0]],
                            [15, 12, strut_thicknesses[0]],
                            [16, 17, strut_thicknesses[0]],
                            [17, 18, strut_thicknesses[0]],
                            [18, 19, strut_thicknesses[0]],
                            [19, 16, strut_thicknesses[0]],
                            [20, 21, strut_thicknesses[0]],
                            [21, 22, strut_thicknesses[0]],
                            [22, 23, strut_thicknesses[0]],
                            [23, 20, strut_thicknesses[0]],
                            ##############################
                            [0, 24, strut_thicknesses[0]],
                            [1, 25, strut_thicknesses[0]],
                            [2, 26, strut_thicknesses[0]],
                            [3, 27, strut_thicknesses[0]],

                            [24, 36, strut_thicknesses[0]],
                            [25, 37, strut_thicknesses[0]],
                            [26, 38, strut_thicknesses[0]],
                            [27, 39, strut_thicknesses[0]],

                            [36, 4, strut_thicknesses[0]],
                            [37, 8, strut_thicknesses[0]],
                            [38, 12, strut_thicknesses[0]],
                            [39, 16, strut_thicknesses[0]],
                            ##############################
```

```
                            [6, 40, strut_thicknesses[0]],
                            [10, 41, strut_thicknesses[0]],
                            [14, 42, strut_thicknesses[0]],
                            [18, 43, strut_thicknesses[0]],

                            [28, 40, strut_thicknesses[0]],
                            [29, 41, strut_thicknesses[0]],
                            [30, 42, strut_thicknesses[0]],
                            [31, 43, strut_thicknesses[0]],

                            [28, 20, strut_thicknesses[0]],
                            [29, 21, strut_thicknesses[0]],
                            [30, 22, strut_thicknesses[0]],
                            [31, 23, strut_thicknesses[0]],
                            ##############################
                            [7, 44, strut_thicknesses[0]],
                            [44, 32, strut_thicknesses[0]],
                            [32, 9, strut_thicknesses[0]],

                            [11, 45, strut_thicknesses[0]],
                            [45, 33, strut_thicknesses[0]],
                            [33, 13, strut_thicknesses[0]],

                            [15, 46, strut_thicknesses[0]],
                            [46, 34, strut_thicknesses[0]],
                            [34, 17, strut_thicknesses[0]],

                            [19, 47, strut_thicknesses[0]],
                            [47, 35, strut_thicknesses[0]],
                            [35, 5, strut_thicknesses[0]],
                            ]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(0)
    elif cell_name == "tetrocta":  # These are basically just octahedrons cut on a different plane. A.k.a.
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[1 / 2, 0, 0], [1, 1 / 2, 0], [1 / 2, 1, 0], [0, 1 / 2, 0],
                                 [0, 0, 1 / 2], [1, 0, 1 / 2], [1, 1, 1 / 2], [0, 1, 1 / 2],
                                 [1 / 2, 0, 1], [1, 1 / 2, 1], [1 / 2, 1, 1], [0, 1 / 2, 1],
                                 [1 / 2, 1 / 2, 1 / 2]]
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 3):
                cell_node_coordinates[x][y] -= 1 / 2
        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1, strut_thicknesses[0]],
                            [1, 2, strut_thicknesses[1]],
                            [2, 3, strut_thicknesses[2]],
                            [3, 0, strut_thicknesses[3]],
                            [8, 9, strut_thicknesses[0]],
                            [9, 10, strut_thicknesses[1]],
                            [10, 11, strut_thicknesses[2]],
                            [11, 8, strut_thicknesses[3]],
                            [0, 5, strut_thicknesses[4]],
                            [1, 5, strut_thicknesses[5]],
                            [1, 6, strut_thicknesses[6]],
                            [2, 6, strut_thicknesses[7]],
                            [2, 7, strut_thicknesses[8]],
                            [3, 7, strut_thicknesses[9]],
                            [3, 4, strut_thicknesses[10]],
                            [0, 4, strut_thicknesses[11]],
                            [4, 8, strut_thicknesses[12]],
                            [5, 8, strut_thicknesses[13]],
                            [5, 9, strut_thicknesses[14]],
                            [6, 9, strut_thicknesses[15]],
                            [6, 10, strut_thicknesses[16]],
                            [7, 10, strut_thicknesses[17]],
                            [7, 11, strut_thicknesses[18]],
                            [4, 11, strut_thicknesses[19]]]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(1)

    elif cell_name == "file_super_truss":
```

```
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = csv_reader.csv_read_nodes("C:\\Users/Maxe-PC2/Desktop/code_and_csv/"
                                                          "1nC[10]tL[1]tG[1]uL[1000]di[200]/nodes.csv", 0)
        # for x in range(0, len(cell_node_coordinates)):
        #     for y in range(0, 3):
        #         cell_node_coordinates[x][y] -= 1/2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = csv_reader.csv_read_mems("C:\\Users/Maxe-PC2/Desktop/code_and_csv/"
                                                    "1nC[10]tL[1]tG[1]uL[1000]di[200]/mems.csv", 0)

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(0)

    elif cell_name == "square":
        # NODES: Describe the nodes in cartesian coordinates.
        cell_node_coordinates = [[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0]]
        # strut_multiplicator = 0.2   # unit meters
        for x in range(0, len(cell_node_coordinates)):
            for y in range(0, 2):
                cell_node_coordinates[x][y] -= 1 / 2

        # CONNECTIONS: Describes the connections via the nodes.
        node_connections = [[0, 1],
                            [1, 2],
                            [2, 3],
                            [3, 0]]

        # PORE SIZE: Describes the pore size of the Cell in standard size 1.
        # It is described by the diameter of the biggest fitting sphere in the cell
        pore_size.append(0)

    else:
        node_connections = [None, None]
        cell_node_coordinates = None
        pore_size = None
        exit("ERROR: cell_name specified is not defined in the library")
    # DEFINE the strut thickness of each strut

    if len(node_connections[1]) < 3:
        counter = 0
        for struts in node_connections:
            struts.append(strut_thicknesses[counter])
            counter += 1
        del counter

    # SIZING change to cell size
    for x in range(0, len(cell_node_coordinates)):
        for y in range(0, 3):
            cell_node_coordinates[x][y] *= cell_size
    for counter in range(0, len(pore_size)):
        pore_size[counter] *= cell_size
        pore_size[counter] -= strut_thicknesses[0]
        if pore_size[counter] < 0:
            pore_size[counter] = 0
    # GENERATE beginning and ending coordinates of each connection in cartesian coordinates.
    cell_node_connections = list()
    for connection_points in node_connections:
        cell_node_connections.append([cell_node_coordinates[connection_points[0]],
                                      cell_node_coordinates[connection_points[1]],
                                      connection_points[2]])

    cell = Cell(name=(cell_name + affix), pore_size=pore_size,
                nodes=cell_node_coordinates, connections=cell_node_connections, strut_thicknesses=strut_thicknesses)
    return cell
```

A.4.7 *library_truss.py*

```
"""
This file creates and returns the nodes of a truss and defines the containing cells.
"""
from Class_Truss import Truss
from library_cell import generate_cell


def generate_truss(truss_name, affix, cell_size, strut_thicknesses, number_of_cells, cell_ratio):
    # NODES: Describe the nodes and their affiliated cells in cartesian coordinates.
    cells = list()
    nodes = list()
    if (truss_name == "cubes" or truss_name == "inv_cubes" or truss_name == "body_centered_cubes" or
        truss_name == "octahedrons" or
       truss_name == "truncated_cubes" or truss_name == "diamonds" or truss_name == "varying_truncated_cubes" or
       truss_name == "face_diagonal_cubes" or truss_name == "octetrahedrons" or truss_name == "void_octetrahedrons" or
       truss_name == "templar_crosses" or truss_name == "templar_alt_crosses" or truss_name == "tetroctas" or
       truss_name == "truncated_octahedrons"):
        cells.append(generate_cell(cell_name=truss_name[:len(truss_name) - 1], affix="",
                                   strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
        for x in range(0, number_of_cells):
            for y in range(0, number_of_cells):
                for z in range(0, number_of_cells):
                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[0]])
    elif truss_name == "face_diagonal_cubes_alt":
        if number_of_cells == 1:
            cells.append(generate_cell(cell_name="face_diagonal_cube", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            nodes.append([0, 0, 0, cells[0]])
        else:
            cells.append(generate_cell(cell_name="face_diagonal_cube", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            cells.append(generate_cell(cell_name="face_diagonal_cube_inv", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))

            for x in range(0, number_of_cells):
                for y in range(0, number_of_cells):
                    for z in range(0, number_of_cells):
                        if (x + y + z) % 2 == 0:
                            nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[0]])
                        else:
                            nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[1]])
    elif truss_name == "templar_alt2_crosses":
        cells.append(generate_cell(cell_name="templar_alt2_cross", affix="",
                                   strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
        cells.append(generate_cell(cell_name="templar_alt2_cross_inv", affix="",
                                   strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))

        for x in range(0, number_of_cells):
            for y in range(0, number_of_cells):
                for z in range(0, number_of_cells):
                    if (x + y + z) % 2 == 0:
                        nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[0]])
                    else:
                        nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[1]])
    elif truss_name == "pyramids":
        if number_of_cells == 1:
            cells.append(generate_cell(cell_name="pyramid", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            nodes.append([0, 0, 0, cells[0]])
        else:
            cells.append(generate_cell(cell_name="pyramid", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            cells.append(generate_cell(cell_name="pyramid_inv", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            cells.append(generate_cell(cell_name="pyramid_twist", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            cells.append(generate_cell(cell_name="pyramid_twist_inv", affix="",
                                       strut_thicknesses=strut_thicknesses, cell_size=cell_size, ratio=cell_ratio))
            for x in range(0, number_of_cells):
                for y in range(0, number_of_cells):
                    for z in range(0, number_of_cells):
```

```
                        if z % 2 == 0:
                            if y % 2 == 0:
                                if x % 2 == 0:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[0]])
                                else:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[3]])
                            else:
                                if x % 2 == 0:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[2]])
                                else:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[1]])
                        else:
                            if y % 2 == 0:
                                if x % 2 == 0:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[1]])
                                else:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[2]])
                            else:
                                if x % 2 == 0:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[3]])
                                else:
                                    nodes.append([x * cell_size, y * cell_size, z * cell_size, cells[0]])
    elif truss_name == "file_super_truss":
        cells.append(generate_cell(cell_name="file_super_truss", affix="",
                                   strut_thicknesses=[1], cell_size=1, ratio=cell_ratio))
        nodes.append([0, 0, 0, cells[0]])

    else:
        print("Possible Names are: \n"
              "cubes\nbody_centered_cubes\noctahedrons\ntruncated_cubes\npyramids\ndiamonds\nvarying_truncated_cubes\n"
              "face_diagonal_cubes\nface_diagonal_cubes_alt\noctetrahedrons\nvoid_octetrahedrons\ntemplar_crosses\n"
              "templar_alt_crosses\nfile_super_truss\ntruncated_octahedrons\ntetroctas")
        nodes = None
        cells = None
        exit("ERROR: truss_name specified is not defined in the library")

    truss = Truss(name=(truss_name + affix),
                  nodes=nodes,
                  cells=cells,
                  cell_size=cell_size,
                  number_of_cells=number_of_cells)
    return truss
```

A.4.8 *csv_reader.py*

```python
import csv


def csv_read_mems(path, multiplicator):
    connections = [[]]
    connections.pop(0)
    i = 0
    # C:\\Users / Maxe - PC2 / Desktop / code_and_csv / mems.csv
    with open(path, newline='') as csvfile:
        conn_reader = csv.reader(csvfile, delimiter=',')
        for row in conn_reader:
            i += 1
            # row2 = [map(int, x) for x in row[2]]
            # if(i>2): connections.append([row2, row[3], row[13]])
            # row2 = [map(int, x) for x in row[2]]
            # if(i>2): connections.append([list(map(int, row[2])), row[3], row[13]])
            if i > 2:
                connections.append([int(row[2]) - 1, int(row[3]) - 1, float(row[13]) * 1e-6])
            # if(i>2): connections.append([list(map(int, row[2])), row[3], row[13]])
            # print(row[0]+"\t"+row[2]+"\t"+row[3])
    return connections


def csv_read_nodes(path, multiplicator):
    coordinates = [[]]
    coordinates.pop(0)
    i = 0
    # "C:\\Users/Maxe-PC2/Desktop/code_and_csv/nodes.csv"
    with open(path, newline='') as csvfile:
        coord_reader = csv.reader(csvfile, delimiter=',')
        for row in coord_reader:
            i += 1
            if i > 2:
                coordinates.append([float(row[3]) * 1e-3, float(row[4]) * 1e-3, float(row[5]) * 1e-3])

    return coordinates


def csv_read_list(path):

    values = [[]]
    values.pop(0)
    i = 0
    # "C:\\Users/Maxe-PC2/Desktop/code_and_csv/nodes.csv"
    with open(path, newline='') as csvfile:
        coord_reader = csv.reader(csvfile, delimiter=',')
        for row in coord_reader:
            value_row = list()
            for value in row:
                value_row.append(float(value))
            values.append(value_row)

    return values
```

BIBLIOGRAPHY

[1] X. Wang, J.s. Nyman, X. Dong, H. Leng, and M. Reyes. Fundamental biomechanics in bone tissue engineering. *Synthesis Lectures on Tissue Engineering*, 2(1):1–225, 2010. doi: 10.2200/s00246ed1v01y200912tis004.

[2] Robert E. Guldberg and Angel O. Duty. Design parameters for engineering bone regeneration. *Functional Tissue Engineering*, page 146–161. doi: 10.1007/0-387-21547-6_12.

[3] J. C. Reichert and D. W. Hutmacher. Bone tissue engineering. *Tissue Engineering*, page 431–456, 2010. doi: 10.1007/978-3-642-02824-3_21.

[4] Susmita Bose, Sahar Vahabzadeh, and Amit Bandyopadhyay. Bone tissue engineering using 3d printing. *Materials Today*, 16(12):496–504, 2013. doi: 10.1016/j.mattod.2013.11.017.

[5] M.i.z. Ridzwan ., Solehuddin Shuib ., A.y. Hassan ., A.a. Shokri ., and M.n. Mohamad Ibrahim . Problem of stress shielding and improvement to the hip implant designs: A review. *Journal of Medical Sciences(Faisalabad) J. of Medical Sciences*, 7(3):460–467, Jan 2007. doi: 10.3923/jms.2007.460.467.

[6] F Chen, X Feng, W Wu, H Ouyang, Z Gao, X Cheng, R Hou, and T Mao. Segmental bone tissue engineering by seeding osteoblast precursor cells into titanium mesh–coral composite scaffolds. *International journal of oral and maxillofacial surgery*, 36(9):822–827, 2007.

[7] Jan-Thorsten Schantz, Dietmar Werner Hutmacher, Christopher Xu Fu Lam, Maik Brinkmann, Kit Mui Wong, Thiam Chye Lim, Ning Chou, Robert Erling Guldberg, and Swee Hin Teoh. Repair of calvarial defects with custofmised tissue-engineered bone grafts ii. evaluation of cellular efficiency and efficacy in vivo. *Tissue engineering*, 9(4, Supplement 1): 127–139, 2003.

[8] Mary S Tyler and David P McCobb. The genesis of membrane bone in the embryonic chick maxilla: epithelial-mesenchymal tissue recombination studies. *Development*, 56(1):269–281, 1980.

[9] M.c. Kruyt, S.m. Van Gaalen, F.c. Oner, A.j. Verbout, J.d. De Bruijn, and W.j.a. Dhert. Bone tissue engineering and spinal fusion: the potential of hybrid constructs by combining osteoprogenitor cells and scaffolds. *Biomaterials*, 25(9):1463–1473, 2004. doi: 10.1016/s0142-9612(03)00490-3.

[10] Karen J.l Burg, Scott Porter, and James F Kellam. Biomaterial developments for bone tissue engineering. *Biomaterials*, 21(23):2347–2359, 2000. doi: 10.1016/s0142-9612(00)00102-2.

[11] V Karageorgiou and D Kaplan. Porosity of 3d biomaterial scaffolds and osteogenesis. *Biomaterials*, 26(27):5474–5491, 2005. doi: 10.1016/j.biomaterials.2005.02.002.

[12] Marc Bohner. Resorbable biomaterials as bone graft substitutes. *Materials Today*, 13(1-2):24–30, 2010. doi: 10.1016/s1369-7021(10)70014-6.

[13] Paul A. Banaszkiewicz. Porous-coated hip replacement. the factors governing bone ingrowth, stress shielding, and clinical results. *Classic Papers in Orthopaedics*, page 51–55, Jul 2013. doi: 10.1007/978-1-4471-5451-8_12.

[14] Mark P. Staiger, Alexis M. Pietak, Jerawala Huadmai, and George Dias. Magnesium and its alloys as orthopedic biomaterials: A review. *Biomaterials*, 27(9):1728–1734, 2006. doi: 10.1016/j.biomaterials.2005.10.003.

[15] C. E. Wen, Y. Yamada, K. Shimojima, Y. Chino, H. Hosokawa, and M. Mabuchi. Novel titanium foam for bone tissue engineering. *Journal of Materials Research J. Mater. Res.*, 17(10):2633–2639, 2002. doi: 10.1557/jmr.2002.0382.

[16] John C Middleton and Arthur J Tipton. Synthetic biodegradable polymers as orthopedic devices. *Biomaterials*, 21(23):2335–2346, 2000. doi: 10.1016/s0142-9612(00)00101-0.

[17] Jessica M. Williams, Adebisi Adewunmi, Rachel M. Schek, Colleen L. Flanagan, Paul H. Krebsbach, Stephen E. Feinberg, Scott J. Hollister, and Suman Das. Bone tissue engineering using polycaprolactone scaffolds fabricated via selective laser sintering. *Biomaterials*, 26(23):4817–4827, 2005. doi: 10.1016/j.biomaterials.2004.11.057.

[18] K. Rezwan, Q.z. Chen, J.j. Blaker, and Aldo Roberto Boccaccini. Biodegradable and bioactive porous polymer/inorganic composite scaffolds for bone tissue engineering. *Biomaterials*, 27(18):3413–3431, 2006. doi: 10.1016/j.biomaterials.2006.01.039.

[19] Malcolm N. Cooke, John P. Fisher, David Dean, Clare Rimnac, and Antonios G. Mikos. Use of stereolithography to manufacture critical-sized 3d biodegradable scaffolds for bone ingrowth. *Journal of Biomedical Materials Research J. Biomed. Mater. Res.*, 64B(2):65–69, Jun 2003. doi: 10.1002/jbm.b.10485.

[20] Qizhi Chen, Chenghao Zhu, and George A Thouas. Progress and challenges in biomaterials used for bone tissue engineering: bioactive glasses and elastomeric composites. *Prog Biomater Progress in Biomaterials*, 1(1):2, 2012. doi: 10.1186/2194-0517-1-2.

[21] S. S. Liao, F. Z. Cui, W. Zhang, and Q. L. Feng. Hierarchically biomimetic bone scaffold materials: Nano-ha/collagen/pla composite. *Journal of Biomedical Materials Research J. Biomed. Mater. Res.*, 69B(2):158–165, 2004. doi: 10.1002/jbm.b.20035.

[22] Wanpeng Cao and Larry L. Hench. Bioactive materials. *Ceramics International*, 22(6):493–507, 1996. doi: 10.1016/0272-8842(95)00126-3.

[23] The virtual institute for artificial electromagnetic materials and metamaterials. `http://www.metamorphose-vi.org/index.php/metamaterials`. Accessed: September 2nd, 2016.

[24] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2006.

[25] Liliana Polo-Corrales, Magda Latorre-Esteves, and Jaime E. Ramirez-Vick. Scaffold design for bone regeneration. *Journal of Nanoscience and Nanotechnology j. nanosci. nanotech.*, 14(1):15–56, Jan 2014. doi: 10.1166/jnn.2014.9127.

[26] X. Zheng, H. Lee, T. H. Weisgraber, M. Shusteff, J. Deotte, E. B. Duoss, J. D. Kuntz, M. M. Biener, Q. Ge, J. A. Jackson, and et al. Ultralight, ultrastiff mechanical metamaterials. *Science*, 344(6190):1373–1377, 2014. doi: 10.1126/science.1252291.

[27] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2):197–224, 1988. doi: 10.1016/0045-7825(88)90086-2.

[28] B. S. Bucklen, W.a. Wettergreen, E. Yuksel, and M.a.k. Liebschner. Bone-derived cad library for assembly of scaffolds in computer-aided tissue engineering. *Virtual and Physical Prototyping*, 3(1):13–23, 2008. doi: 10.1080/17452750801911352.

[29] Andy L. Olivares, Èlia Marsal, Josep A. Planell, and Damien Lacroix. Finite element study of scaffold architecture design and culture conditions for tissue engineering. *Biomaterials*, 30(30):6142–6149, 2009. doi: 10.1016/j.biomaterials.2009.07.041.

[30] Ferry P.w. Melchels, Katia Bertoldi, Ruggero Gabbrielli, Aldrik H. Velders, Jan Feijen, and Dirk W. Grijpma. Mathematically defined tissue engineering scaffold architectures prepared by stereolithography. *Biomaterials*, 31(27):6909–6916, 2010. doi: 10.1016/j.biomaterials.2010.05.068.

[31] L. E. Murr, S. M. Gaytan, F. Medina, H. Lopez, E. Martinez, B. I. Machado, D. H. Hernandez, L. Martinez, M. I. Lopez, R. B. Wicker, and et al. Next-generation biomedical implants using additive manufacturing of complex, cellular and functional mesh arrays. *Philosophical Transactions of the*

Royal Society A: Mathematical, Physical and Engineering Sciences, 368(1917): 1999–2032, 2010. doi: 10.1098/rsta.2010.0010.

[32] Jan Wieding, Andreas Wolf, and Rainer Bader. Numerical optimization of open-porous bone scaffold structures to match the elastic properties of human cortical bone. *Journal of the Mechanical Behavior of Biomedical Materials*, 37:56–68, 2014. doi: 10.1016/j.jmbbm.2014.05.002.

[33] S Rajagopalan and R Robb. Schwarz meets schwann: Design and fabrication of biomorphic and durataxic tissue engineering scaffolds. *Medical Image Analysis*, 10(5):693–712, 2006. doi: 10.1016/j.media.2006.06.001.

[34] Zhongzhong Chen, Xilan Feng, and Zhiqiang Jiang. Structural design and optimization of interior scaffolds in artificial bone. *2007 1st International Conference on Bioinformatics and Biomedical Engineering*, 2007. doi: 10.1109/icbbe.2007.88.

[35] M.f Ashby. The properties of foams and lattices. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 364 (1838):15–30, 2006. doi: 10.1098/rsta.2005.1678.

[36] Sebastian C. Kapfer, Stephen T. Hyde, Klaus Mecke, Christoph H. Arns, and Gerd E. Schröder-Turk. Minimal surface scaffold designs for tissue engineering. *Biomaterials*, 32(29):6875–6882, 2011. doi: 10.1016/j.biomaterials.2011.06.012.

[37] W. Sun, B. Starly, J. Nam, and A. Darling. Bio-cad modeling and its applications in computer-aided tissue engineering. *Computer-Aided Design*, 37(11):1097–1114, 2005. doi: 10.1016/j.cad.2005.02.002.

[38] Qing-Xi Hu, Yi Qian, Yuan Yao, Jun Guo, and Yongwei Yu. A computer designing approach for generation of controllable porosity in the bone scaffold. *2009 IEEE 10th International Conference on Computer-Aided Industrial Design and Conceptual Design*, 2009. doi: 10.1109/caidcd.2009.5375116.

[39] Aldo R Boccaccini and Jonny J Blaker. Bioactive composite materials for tissue engineering scaffolds. *Expert Review of Medical Devices*, 2(3): 303–317, 2005. doi: 10.1586/17434440.2.3.303.

[40] Dietmar W. Hutmacher. Scaffolds in tissue engineering bone and cartilage. *Biomaterials*, 21(24):2529–2543, 2000. doi: 10.1016/s0142-9612(00)00121-6.

[41] Dong-Jin Yoo. Computer-aided porous scaffold design for tissue engineering using triply periodic minimal surfaces. *International Journal of Precision Engineering and Manufacturing Int. J. Precis. Eng. Manuf.*, 12(1): 61–71, 2011. doi: 10.1007/s12541-011-0008-9.

[42] Y. Shikinami, K. Okazaki, M. Saito, M. Okuno, S. Hasegawa, J. Tamura, S. Fujibayashi, and T. Nakamura. Bioactive and bioresorbable cellular cubic-composite scaffolds for use in bone reconstruction. *Journal of The Royal Society Interface*, 3(11):805–821, 2006. doi: 10.1098/rsif.2006.0144.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.