

Topology Optimization of Masonry Structures

Kaan Akbaba

Topology Optimization of Masonry Structures

by

Kaan Akbaba

to obtain the degree of Master of Science
at the Delft University of Technology,
at the faculty of Architecture and the Built Environment (Building Technology)
&
at the faculty of Civil Engineering and Geosciences (Structural Engineering)

Student number: 4655923

Project duration: November, 2021 – January, 2023

Thesis committee: Dr. P. Nourian 1st mentor Building Technology (November, 2021 – December, 2022),
Prof. dr. ir. arch. I. S. Sariyildiz 1st mentor Building Technology (December, 2022 – January, 2023),
Dr. A. A. Mehrotra 1st mentor Structural Engineering,
Dr. ir. F. A. Veer 2nd mentor Building Technology & Structural Engineering

Contents

1	Introduction	3
1.1	Background	3
1.2	Problem statement	4
1.3	Research objectives	4
1.4	Research questions	5
1.5	Relevance	5
2	Literature Review & Methodology	7
2.1	Literature review	7
2.1.1	Generation of geometry	7
2.1.2	Structural analysis	7
2.1.3	Topology optimization	8
2.2	Methodology	9
3	Model & Tool Development	11
3.1	Generation of initial geometry and topology.	11
3.1.1	Shape	11
3.1.2	Abstraction	14
3.1.2.1	A single element	14
3.1.2.2	Interactions between elements	14
3.1.3	Generating structures	16
3.1.3.1	Cubic sphere	16
3.1.3.2	Rhombic dodecahedron	20
3.1.3.3	Voids.	24
3.2	Structural Analysis	28
3.2.1	Time-explicit analysis using global damping	28
3.2.1.1	Stiffness matrix	28
3.2.1.2	Mass matrix	38
3.2.1.3	Damping.	41
3.2.1.4	Solving the equation of motion	42
3.2.2	Time-explicit analysis using local damping	43
3.2.3	Static analysis	44
3.2.4	Results	45
3.2.4.1	Time-explicit analyses	45
3.2.4.2	Static analysis.	49
3.3	Optimization	51
3.3.1	Objective function	51
3.3.2	Constraints	52
3.3.2.1	Volume	52
3.3.2.2	Roof	53
3.3.3	Filtering	55
3.3.4	Results	56
3.3.4.1	Time-explicit optimization.	56
3.3.4.2	Static analysis.	58
4	Verification	67
4.1	Time-explicit optimizations	68
4.2	Static optimizations.	71
4.3	Physical model	74

5 Application	75
5.1 Current implementation	75
5.1.1 Optimization by parts	75
5.1.2 Optimization of modular structures	77
5.2 Future visions.	78
5.2.1 Structures with custom tessellations.	78
5.2.2 Aid in restoration projects	79
6 Conclusions	81
Bibliography	83

Abstract

Designing structures that are more sustainable is a relevant topic within the construction industry. By choosing materials that have a low embodied energy value and optimizing the structures that can be constructed by these materials, one could potentially minimize the economical and environmental footprint of a structural design.

As burnt clay bricks have a relatively low embodied energy value, are relatively cheap as a construction material and are relatively durable, it is interesting to investigate the optimization of masonry structures. To achieve this, use is made of topology optimization. To accurately optimize the topology of masonry structures however, this optimization must be performed based on the results of a discrete element analysis. This thesis presents several methods to set up such a model based on masonry structures of arbitrary size and lay-out, departing from the smallest scale: the individual brick.

First, a method is developed to create arbitrary shapes for bricks. An algorithm is developed to parametrically create structures for two distinct shapes. A procedure to abstract these structures and translate their geometrical representation to a simplified numerical model is then presented. Several methods for structural analyses are detailed and their results are evaluated and compared. The results of these analyses are used to optimize the topology of the initial structure by means of the Method of Moving Asymptotes. The resulting structures are then verified using 3DEC. Finally, some applications of the developed method are presented along with future visions.

Introduction

In this first chapter mainly the background of this thesis is given. The reason why topology optimization for masonry structures is researched is discussed in detail along with relevant problems and objectives. From these a research question is then formulated that is attempted to be answered by means of this thesis. Finally, some relevant potential applications from a societal point-of-view are mentioned.

1.1. Background

Designing buildings or structures that are as sustainable as possible is important for the development of the built environment while minimally impacting the environment. As such, one could realistically wish for inherently sustainable designs that make use of as little active measures as possible to keep them operable. Such active measures include (but are not limited to) cooling, heating, ventilation, or structural maintenance. As this thesis is written to obtain a Master's degree for both Building Technology (BT) and Structural Engineering (SE), it focuses on improving the sustainability of designs by considering its structure.

The sustainability of a design based on its structure is a multifaceted problem where not only the locations, connections, or dimensions of structural members are relevant, but also which materials are used. Choosing a construction material that has the least environmental impact can be done based on the amount of energy necessary to produce and transport it. This energy is also called the embodied energy of that material. One example of a construction material with a relatively low embodied energy value is burnt clay bricks (Reddy, n.d.). Burnt clay bricks have an embodied energy value for its production of $2550 \frac{\text{MJ}}{\text{m}^3}$, which is around 60% lower than for Portland cement (which is used to produce concrete) and 95% lower than for steel. The embodied energy from transportation of burnt clay bricks is however twice as large compared to both Portland cement and steel. This means that to minimize the amount of embodied energy, one could seek to construct using burnt clay bricks that are produced locally.

Apart from the fact that burnt clay bricks have a relatively low embodied energy, it is also relatively cheap to construct using burnt clay bricks. When comparing the construction costs of a two-storey semi-detached dwelling realized with unreinforced masonry or reinforced masonry to the cost of its construction realized with reinforced concrete, a cost-reduction of 24% and 16% was obtained respectively (Marques & Lourenço, 2014). In addition to its cost-effectiveness, masonry structures are known to be durable and enjoying a relatively long lifespan (Ghiassi & Lourenço, 2018).

1.2. Problem statement

If one chooses to design a structure using masonry elements, the next step would be to make a structural design so that these elements are used as efficiently as possible. This is so that the least amount of material is used, while the structural integrity is maintained. If one's goal is to design sustainable structures, this is especially important. One tool that can be used to aid a designer with designing efficient structural designs is topology optimization.

Structural topology optimization is about changing the topology of a structure in such a way to obtain the best structural performance with a specified amount of material. The other way around, it could be regarded as achieving a certain structural performance (for instance stiffness or strength in a specific member or system) with the least amount of material used. Such optimizations usually employ the finite element method (FEM) to perform structural analyses. Optimized structures can save on material costs, which is interesting not only from an economical point-of-view, but also environmentally, as less material needed results in less embodied energy for a structure.

In order to optimize masonry structures, it is important to accurately model their structural behaviour. Analysing a masonry structure accurately can be done by using a finite element model with joints or interface elements, or by using the discrete element method (or distinct element method). The term discrete elements is commonly used for numerical models to analyse the mechanical behaviour of systems consisting of multiple bodies, blocks, or particles. This is related to masonry as masonry structures can be idealized as a discontinuum, modelling the mechanical behavior of the elements and the interaction between them separately (Lemos, 2007). Algorithms to optimize three-dimensional structures using the discrete element method (DEM) however, do not exist, and joint or interface elements in FEM must be manually defined, which makes it a tedious process for large structures. An algorithm to optimize three-dimensional structures using the discrete element method must thus be developed to optimize masonry structures.

1.3. Research objectives

The main objective of this thesis is to develop a method to optimize three-dimensional masonry structures using the discrete element method. To achieve this, some sub-objectives are also defined that must be achieved in the process.

The first sub-objective is to define the shape of the masonry elements and how they relate to each other. In other words, a starting topology must be defined. Apart from this, a method to generate numerical data of this structure that will be used to perform structural analyses with, must be developed. A constraint that is put on the structures, is that they must consist of exactly one type of element. This is partly to simplify the production process for the resulting structures, and partly to provide a general method to generate structures with arbitrary dimensions.

The starting topology of the masonry structure should be used to perform structural analyses using the discrete element method. In order to achieve this, a discrete element model must be implemented. This will finally result in output values that can be used to optimize the structural topology.

The final sub-objective is to optimize the starting topology using the output of the performed structural analysis. This means that a method to perform this optimization must be chosen as well as what output values will be used.

1.4. Research questions

This thesis aims to answer the following, rather general, main research question:

How can an arbitrary three-dimensional masonry structure be optimized using the discrete element method?

This question is kept quite general as infinitely many methods can be formulated to optimize masonry structures, but as there is no precedent in current literature, it is deemed important that some formulation be provided and tested.

Some relevant sub-questions to be answered during this research include:

- How are elements shaped?
- How is the starting geometry (before optimization) generated?
- How is the structural behaviour modelled?
- Which value should be optimized?
- How will this value be optimized?
- What optimization constraints should be used?
- Are the optimized structures feasible?
- How can this algorithm be applied?

The following chapter presents the results of the conducted literature review and provides an overview of the methodology followed to develop the optimization algorithm.

Afterwards the implemented method is presented in detail. This is done mostly by means of mathematical expressions and pseudo-codes. The results are illustrated and an overview will be provided of the values used to arrive at these results.

The resulting structures are then analysed using established DEM software. By using third-party software, and comparing its results with the results of the developed method, it is aimed to verify the developed method and provide insight on what should be altered in potential future developments.

As this thesis is written to obtain a Master's degree for both BT and SE, it should be clearly stated which part of it is relevant for which study programme. For BT mainly chapters 2, 3.1, 3.3 and 5 are relevant, while chapters 2, 3.2, 4 are relevant for SE.

1.5. Relevance

Apart from the fact that structurally optimized masonry structures could result in more sustainable structures, some other relevant uses of the final method could be mentioned.

One use could be, for instance, in the renovation or restoration of existing masonry structures. These structures can be reinforced by using the results from a topology optimization. Topology optimization could also indicate which parts of existing structures are more important for the structural integrity and may thus aid with managing restoration projects.

The resulting data may also be used during the construction of masonry structures where use is made of robotics (J. Kim et al., 2015). Robots may assist in placing masonry elements in difficult to reach locations or when a high degree of precision is necessary.

The method that is developed in this thesis does not have to be limited to only bricks. As long as the structure is constructed using distinct elements, this topology optimization method may prove useful. The designer may thus be able to run optimizations using a variety of different material parameters.

2

Literature Review & Methodology

This chapter provides an overview of the results of a preliminary literature review. Based on the results of this review a methodology is outlined which will be presented in further detail.

2.1. Literature review

Before any model or method was developed a literature review was conducted to gain insight into existing methods. This study can be organized into three parts, corresponding to the sub-objectives of this thesis, as presented in chapter 1.

2.1.1. Generation of geometry

As stated previously only one type of element is used in the structures presented in this thesis. This means that the shape of these elements must be space-filling in order to generate structures where the elements fit together without leaving any gaps. Inchbald, 1996 provides details about five of such shapes, but it becomes apparent that such polyhedrons may have rather complicated or impractical shapes. Loeb, 1991 gives a thorough description of space-filling shapes. Some examples showcased are the truncated octahedron, rhombic dodecahedron, and the cube. Thus, some simpler shapes are also space-filling.

Loeb, 1991 also gives a description of how complex geometrical shapes can be described or constructed by using more elementary shapes like cubes or tetrahedrons. More details about the geometrical and physical properties of tetrahedrons are given by Burkardt, 2010 and Tonon, 2004.

2.1.2. Structural analysis

According to Lemos, 2007 the discrete element method is an approach to accurately model interactions within masonry structures. This method dates back as far as 1979 (Cundall & Strack, 1979). According to Luding, 2008 there exist two approaches for the discrete element method. One approach is the so-called soft particle molecular dynamics method where an equation of motion is solved for a system of interacting particles. Another approach is the hard-sphere, event-driven method. The soft-particle approach has many applications, while the event-driven method is often used for rigid interactions, like collisions. A method used to model collisions does not appear appealing to model buildings, which should normally display only small displacements. It is for this reason that the former approach will be adopted in this thesis.

Jing and Stephansson, 2007 gives a detailed description of the definitions used in two-dimensional and three-dimensional applications of the discrete element method. Additionally, they show how discrete element models tend to behave. While the examples shown are for granular materials like soil, it is expected that the described model can be used to accurately describe the behaviour of masonry structures. Here, it is assumed that a masonry structure is simply an upscaled granular model.

Frick et al., 2016 provides a data structure for discrete element interfaces. In their method they model interfaces as either points, lines or areas shared between elements, depending on their locations and orientations. This is quite a general formulation of interfaces, which is useful for imperfect discrete element assemblies. As the starting topology will be generated as a perfectly ordered lattice however, such an approach is an unnecessary generalization which will likely slow down the algorithm without much benefit. Spring-like interfaces between elements that essentially lump the interface to a single point as described in Jing and Stephansson, 2007 seem to be a more appealing implementation in this case.

Rojek, 2018 describes some linear contact models between elements. Two models described by them are the unilateral model where no tensile forces are allowed, and the bilateral model or adhesive model, where tensile forces are allowed between elements. The former model is adopted in this thesis, as it is assumed to be reasonable that no significant tensile forces can appear between masonry elements, even with mortar between elements. This assumption can be backed by experimental data that shows that the initial tensile strain limit of concrete is just 0.01%, which is 10 times smaller than its initial compressive strain limit (Ng et al., 2010). If however, another adhesive material is used that can withstand significant tensile stresses, this assumption may not hold true anymore.

2.1.3. Topology optimization

According to Bendsoe and Sigmund, 2003 there are three categories of structural optimization, namely: size optimization, shape optimization, and topology optimization (figure 2.1). Within topology optimization the aim is to parameterize the stiffness components of a given design domain (for instance the beam shown in figure 2.1) and iteratively change these parameters such that the compliance of the structural system is minimized (or its stiffness is maximized) (Bendsoe, 1989). A further addition to this formulation is given by Bendsoe and Sigmund, 1999 with the addition of penalization of intermediate parameter values. This method is called the solid isotropic material with penalization model (SIMP).

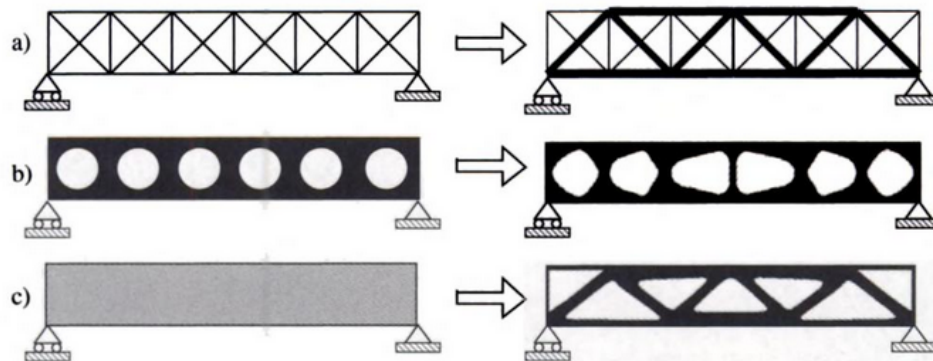


Figure 2.1: three categories of structural optimization. a) sizing optimization of a truss structure, b) shape optimization and c) topology optimization (Bendsoe & Sigmund, 2003).

O'Shaughnessy et al., 2022 describes a method for the topology optimization of two-dimensional discrete element structures, where the elements are modelled by means of circular particles connected by springs. In this method, rather than minimizing the structural compliance, the work done on the system is minimized, while the internal energy is maximized. This formulation is also adopted in this thesis, as the formulation of the structural behaviour developed in this thesis is similar to the one described in O'Shaughnessy et al., 2022.

Usually in topology optimization, a volume constraint is used to ensure that the optimizer does not yield structures that consist of too little (or no) material or too much (Bendsoe, 1989). Such a constraint also provides some control over the output, as one can simply change the parameter for this constraint to obtain a thinner or larger structure.

Topology optimization algorithms often employ filters to ensure mesh-independence (Pedersen et al., 2006). Such filters, along with their radii, will affect the resulting optimal structure. Thus, it is important to have insight into the effects of using mesh-independence filters on the optimization process.

There are several methods to calculate the next iteration of stiffness parameters (better known as *design variables*) in topology optimization problems. Some of these methods are: the optimality criteria method (OCM) (Bendsøe, 1995), the generalized optimality criteria method (GOCM) (N. H. Kim et al., 2021), the method of moving asymptotes (MMA) (Svanberg, 1987), and the sequential linear programming method (SLQM) (Dunning & Kim, 2015). As it is not within the scope of this thesis to discuss these methods in-depth, a method to perform structural topology optimizations will be chosen mainly based on its availability.

2.2. Methodology

Based on the research objectives of chapter 1 and the literature review of section 2.1, a methodology can be outlined.

First, some geometrical shapes are assembled using simple shapes. From these shapes a data structure is set up to represent them numerically. This representation is meant to simplify the computational model, and to translate the geometries to data that can be used for structural analyses. An algorithm is set up to not only set up a lattice of arbitrary size consisting of the defined shapes, but also to compute their physical properties.

Then, a method to perform structural analyses on the generated lattices is outlined. The results of these analyses are shown and evaluated.

Afterwards, the topology optimization algorithm is detailed. Using the results of the structural analyses optimizations are performed. Again, these results are shown and evaluated.

At the end, the results of this algorithm are verified using the commercial DEM software package 3DEC. This program is chosen due to its availability and can be interchanged by any DEM software.

The entire methodology followed in this thesis is visualized in figure 2.2. Here the three columns represent the main parts of the development of the model as described above. One interesting characteristic of these parts is that they are somewhat independent. For instance, a completely different approach could be taken to generate the geometry without significant change in the way how the structural analysis is performed or how the topology is optimized.

The developed algorithm is written using Python. As such, the MMA implementation of Deetman, 2021 is used to perform the topology optimizations. Again, this choice is made due to the availability of the method, which means that any of the aforementioned solution procedures could be a viable choice. Visualizations are made by using Rhinoceros 3D with its built-in Grasshopper plug-in. The complete implementation is made publicly available at <https://gitlab.com/te872/todes>.

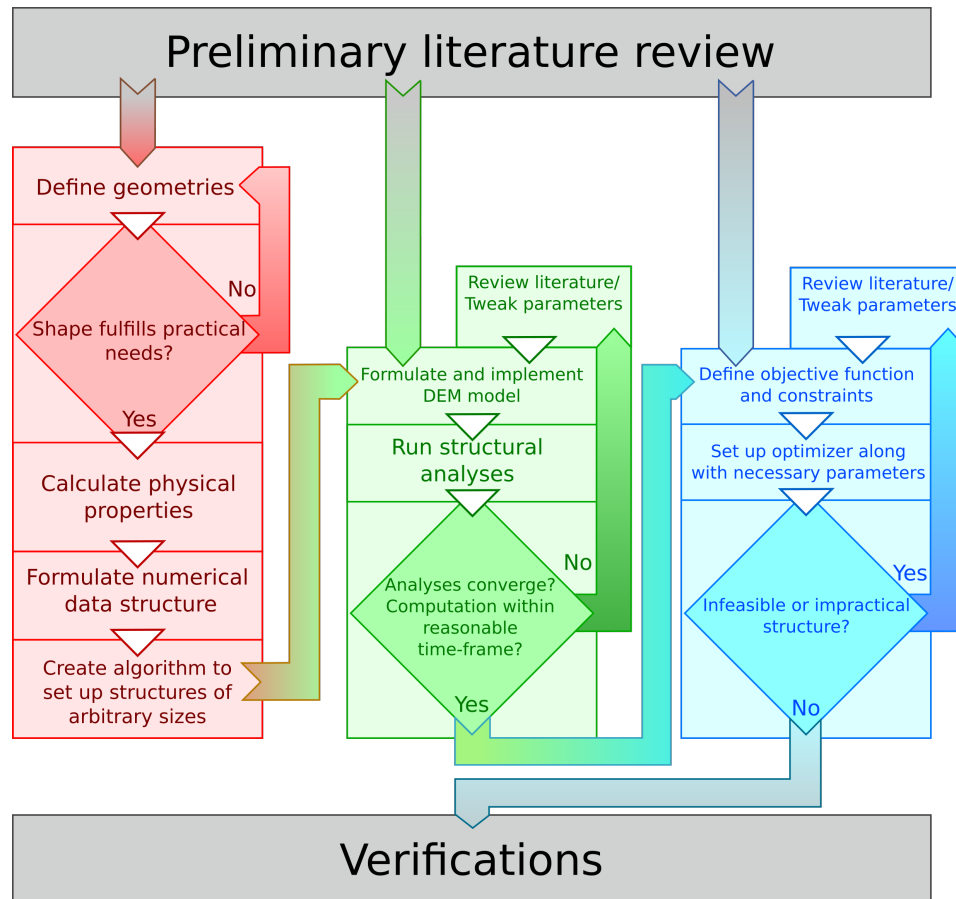


Figure 2.2: flowchart illustrating the methodology of this thesis

3

Model & Tool Development

3.1. Generation of initial geometry and topology

The first step in order to optimize masonry structures, is to define the shape of the individual elements. This chapter details the procedure to create arbitrary assemblies of space-filling elements.

3.1.1. Shape

To find suitable shapes to perform the topology optimization on, a set of rules is defined. This ruleset is meant to objectively select elements. The rules are as follows:

1. The selected shape should be space-filling. This means that a shape must be able to fill an indefinite space when repeated in a certain pattern. It is important to note that no gaps are allowed to exist in this space. The most simple example of such a shape is a cube, as it is able to fill a space when it is repeated in a regular pattern. There are however, many more shapes that are space-filling.
2. Shapes must be able to be assembled from smaller cubes or tetrahedrons. These smaller shapes will be called *elementary shapes* henceforth. The difference between elementary shapes and elements should be noted: an element, which has an arbitrary shape, consists out of multiple elementary shapes, which are either cubes or tetrahedrons in this case.

This rule is set to exclude curved shapes and thus narrow down the available options. It also allows for the same shape to be assembled from smaller elementary shapes, which can be seen as increasing its resolution (figure 3.1).

3. Elementary shapes (including elementary shapes with an increased resolution) will not be considered as elements, as one aim of this research is to explore different shapes and the potentially different results they could yield. Opting for elementary shapes as the elements may cut this search short.
4. The final rule is to choose a shape that constitutes the least amount of elementary shapes. This is to prevent shapes from being too complex and thus being more computationally expensive.

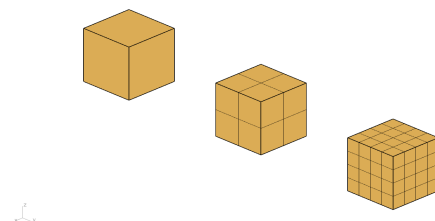


Figure 3.1: cubes with increasing resolution

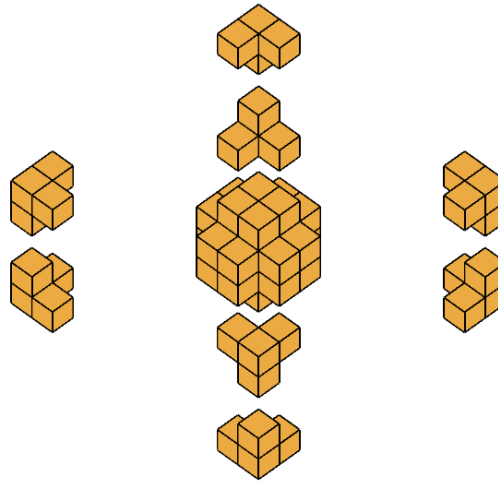
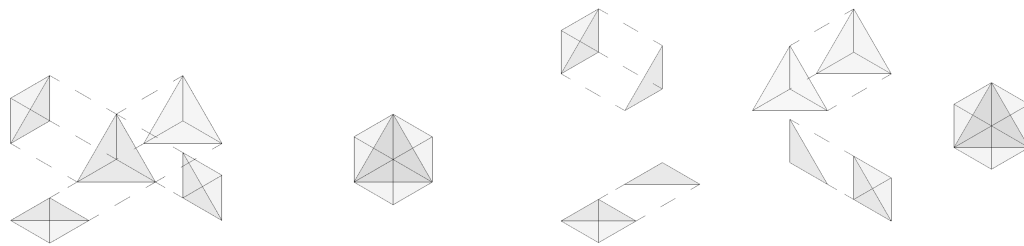


Figure 3.2: cubic spheres truncated at cubic boundaries



(a) splitting a cube into 5 tetrahedrons

(b) further splitting of the inner tetrahedron

Figure 3.3: splitting process of a cube into tetrahedrons

With these rules in place, it is time to look at different assemblies of elementary shapes. It was decided to choose one element assembled from only cubic elements, and one that is assembled from solely tetrahedral elements.

For the cube-based element it was first considered to make a star-shaped element, with a central cube and cubes attached to all of its sides. The main reason such a shape was regarded, is because such an element will have interactions with elements above and below it, which are themselves offset in x- and y-directions. Considering that the optimized structures will most likely resemble arches or domes, such a property is desirable. This shape however, is not space-filling, and thus it is modified slightly. The resulting shape slightly resembles a sphere and is space-filling (figure 3.2). This shape satisfies rule 1 to 3 and after further modifications, it was decided that no other shape satisfies these rules with less elementary shapes. As such, it was decided that rule 4 is also satisfied, but the existence of a less complex shape is not ruled out. This shape is called a *cubic sphere* (CS) from this point onward, as it resembles a sphere and it consists solely of cubes.

The process to find the tetrahedron-based element is a more lengthy one. First, a cube is divided into five tetrahedrons. Four of these tetrahedrons are identical and make up the outer part of the cube. The tetrahedron at the center of the cube is again split into four identical tetrahedrons (figure 3.3). This results in eight tetrahedrons, of which two sets of four identical tetrahedrons can be made (namely four outer and four inner tetrahedrons). An interesting property of this tessellation is that it is chiral, meaning that the tessellated cube has a mirror image that is not equal to itself (figure 3.4). Making use of this chirality, one can alternate between the chiral tessellations. From here, tetrahedrons are removed one by one until a certain shape is acquired, as if it is chiselled out. A small collection of shapes found in this fashion can be seen in figure 3.5. The shape that satisfies all the rules is found to be the rhombic dodecahedron (RD) (figure 3.6). This polyhedron again has the desirable characteristic that it connects to neighbouring elements at an offset.

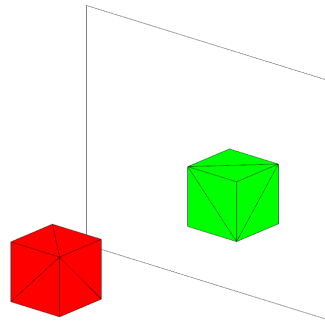
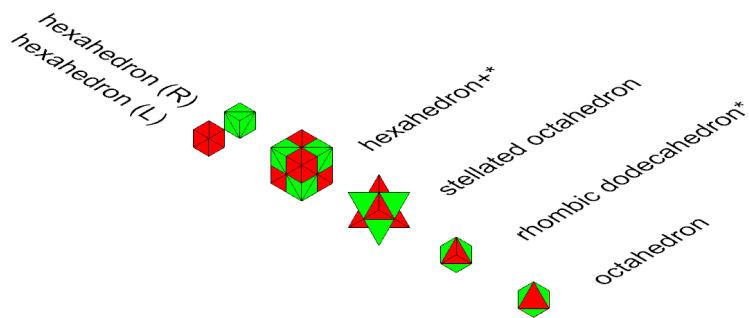
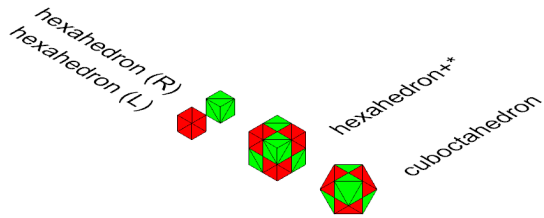


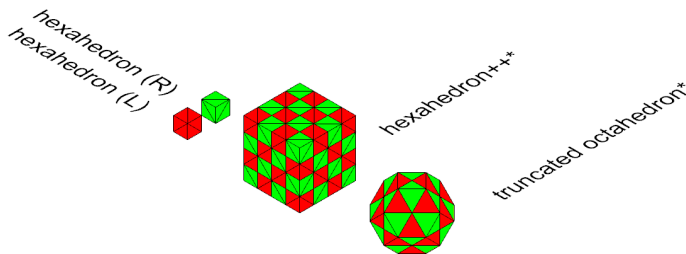
Figure 3.4: chirality of the cube consisting of tetrahedrons



(a) 2x2 tetrahedral cubes (chirality: left)



(b) 2x2 tetrahedral cubes (chirality: right)



(c) 4x4 tetrahedral cubes (chirality: right)

Figure 3.5: different shapes found using tetrahedrons

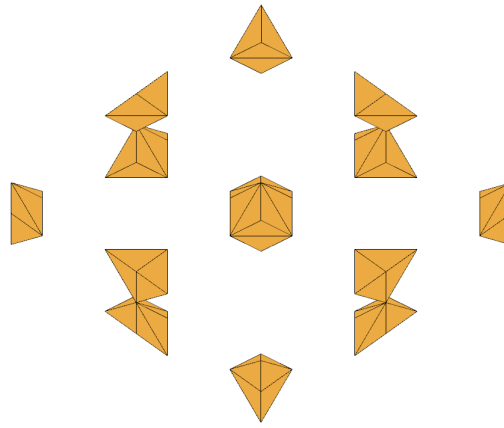


Figure 3.6: rhombic dodecahedrons truncated at cubic boundaries

3.1.2. Abstraction

Now that two basic shapes have been found, it is important to reformulate their geometrical representation in a more abstract sense. This results in a simplified numerical model, as ultimately abstracted connections between elements will be modelled instead of their physical contact surfaces. Because the elements consist of multiple elementary shapes, one could define internal interactions in addition to interaction between different elements. These two types of interactions are detailed separately in the following sections.

3.1.2.1. A single element

As stated earlier, an element consists of multiple elementary shapes. To simplify the problem and make the final analysis less computationally expensive, another way of representing shapes is needed. It is chosen to let go of the concept of geometry in its entirety; shapes will be represented by graphs. In order to do so accurately and consistently, a few rules have to be set up again:

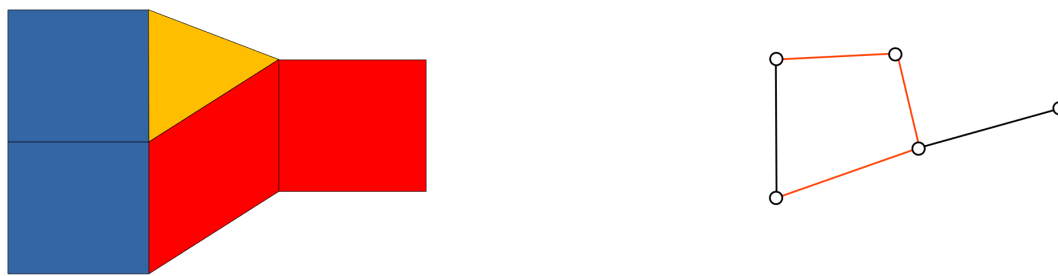
1. A node will represent an elementary shape, and its location will be the volumetric centroid of the shape it represents.
2. A connection will be represented by a connection or path between two nodes. A connection will be made between elementary shapes if they belong to the same element and if they share a face.

With these two rules, the previously found shapes can be translated from a geometric representation to an abstracted graphical representation (figure 3.7a). An important point to make is that no stresses or strains will be computed to reduce the complexity of the final model. In other words, the connections between elementary shapes of one element are fully rigid connections.

3.1.2.2. Interactions between elements

The interactions between elements and how they are abstracted is one of the fundamental choices made in this model. In line with the rules described in section 3.1.2.1 connections between elements will only be made between nodes if the shapes they represent share a face. An important difference however, is that these connections are not rigid. These connections behave like springs and are illustrated in figure 3.7b.

A connection between two elements represents multiple springs. An overview of different springs is shown in figure 3.8. A detailed description on how these springs are actually modelled is given in section 3.2.



(a) geometry before abstraction (shapes belonging to different elements have different colours)

(b) graph after abstraction

Figure 3.7: visualization of the geometrical abstraction rules

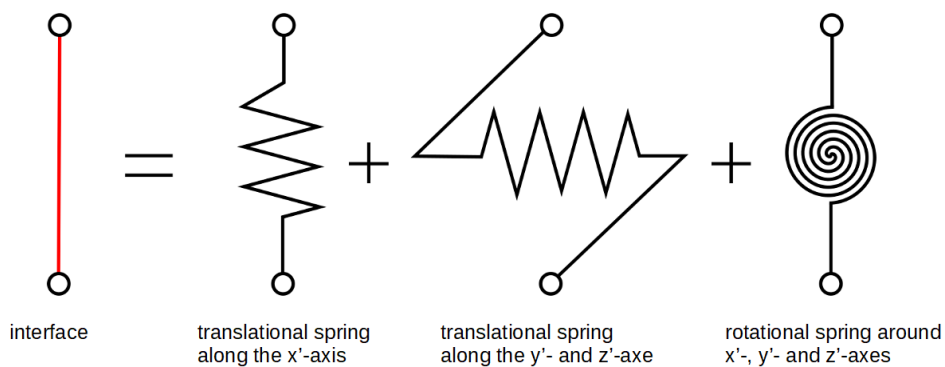


Figure 3.8: the spring components that are represented by an interface connection

3.1.3. Generating structures

Having defined an outline for the abstraction of shapes as graphs, it is now necessary to create some kind of data structure to set up a graph (lattice or structural system) consisting of these sub-graphs (elements). The essential properties of a graph are its nodes and how these are connected. It is thus necessary to generate this data for arbitrary structure sizes. Due to the fundamental differences in their shapes and where their elementary shapes are exactly located, both shapes will be dealt with separately.

3.1.3.1. Cubic sphere

To formulate an algorithm to generate lattices consisting of cubic spheres, one could depart from the $4 \times 4 \times 4$ cube as shown in figure 3.1. Comparing this with figure 3.2 it can be seen that every cubic sphere can be bounded by a $4 \times 4 \times 4$ cube. The only piece of information necessary to now generate a lattice, is the location of each of these bounding cubes. The location is chosen to be the smallest elementary cube index of the bounding cube (figure 3.9). Note that this means the location of a cubic sphere is defined by a cube which is not actually inside it.

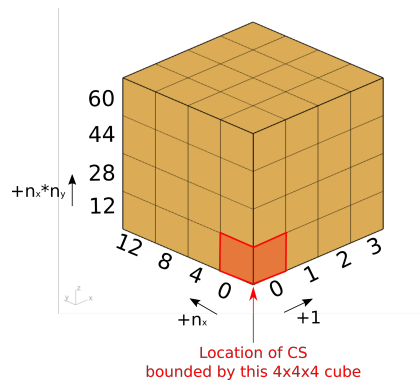
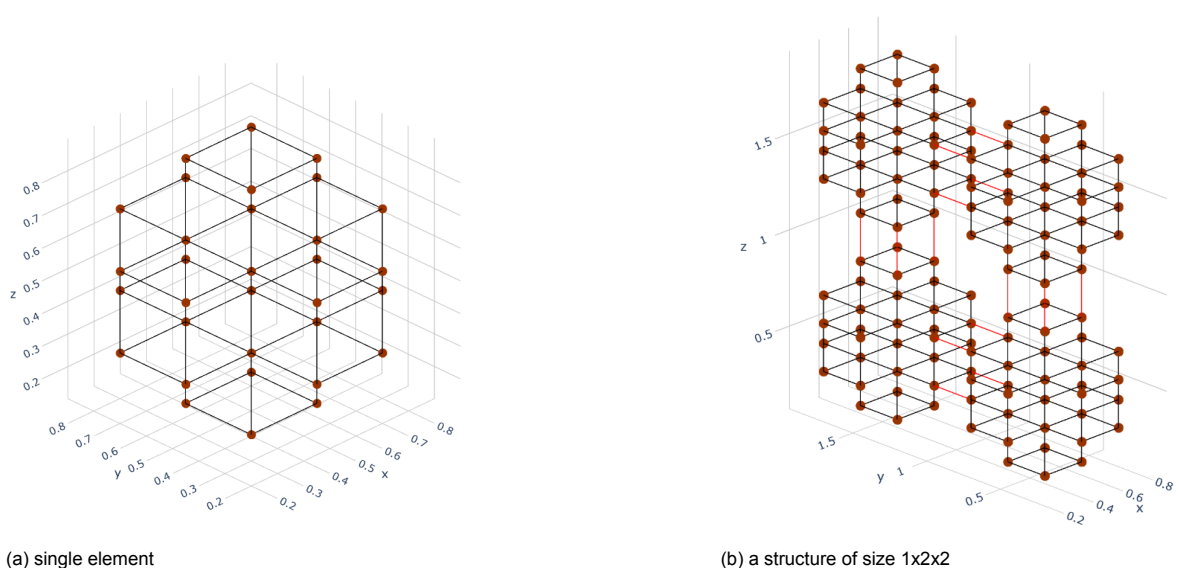


Figure 3.9: location of a CS inside a $4 \times 4 \times 4$ along with the indexing convention of elementary cubes

Using the indexing convention of figure 3.9 a set of indices could be defined that refer to all the elementary cubes that describe a CS. Afterwards, using the rules described in section 3.1.2.1, a set of edges can also be defined. As this is quite straightforward for a voxelated grid, it will not be detailed here which exact indices are included in these sets. The resulting nodal dual graphs are shown in figure 3.10. These graphs are generated with use of the Plotly library for Python.



(a) single element

(b) a structure of size $1 \times 2 \times 2$

Figure 3.10: nodal dual graphs for CS-based structures

One could now define an algorithm to set the locations of each cubic sphere for a given structure size (the number of these 4x4x4 cubes in each direction) (algorithm 1).

Algorithm 1: Algorithm for computing the locations of on-grid CS elements

Data: Structure size N_x , N_y and N_z
Result: Locations of CS elements loc_{CS}

$$N_{elem,x} = 4N_x$$

$$N_{elem,y} = 4N_y$$

```

for  $z$  in  $range(N_z)$  do
  | for  $y$  in  $range(N_y)$  do
  | | for  $x$  in  $range(N_x)$  do
  | | |  $loc_{CS}.append(4 * (x + y * N_{elem,x} + z * (N_{elem,x} * N_{elem,y})))$ 
  | | end
  | end
end

```

However, not all cubic spheres are on an orthogonal grid. There are so-called off-grid cubic spheres (figure 3.11) that should be added to this algorithm (algorithm 2).

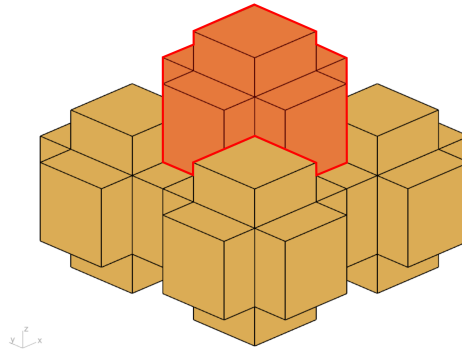


Figure 3.11: off-grid cubic sphere

Algorithm 2: Algorithm for computing the locations of off-grid CS elements

Data: Structure size N_x , N_y and N_z
Result: Locations of CS elements loc_{CS}

$$N_{elem,x} = 4N_x$$

$$N_{elem,y} = 4N_y$$

```

for  $z$  in  $range(N_z - 1)$  do
  | for  $y$  in  $range(N_y - 1)$  do
  | | for  $x$  in  $range(N_x - 1)$  do
  | | |  $loc_{CS}.append((4x + 2) + (4y + 2) * N_{elem,x} + (4z + 2) * (N_{elem,x} * N_{elem,y}))$ 
  | | end
  | end
end

```

Applying algorithms 1 and 2 results in a particular element indexing that is illustrated in figure 3.12b. Figure 3.12b also shows which specific elements connect to which elements as a dual graph of elements. Note that this dual graph is fundamentally different than the nodal dual graphs shown in figure 3.10.

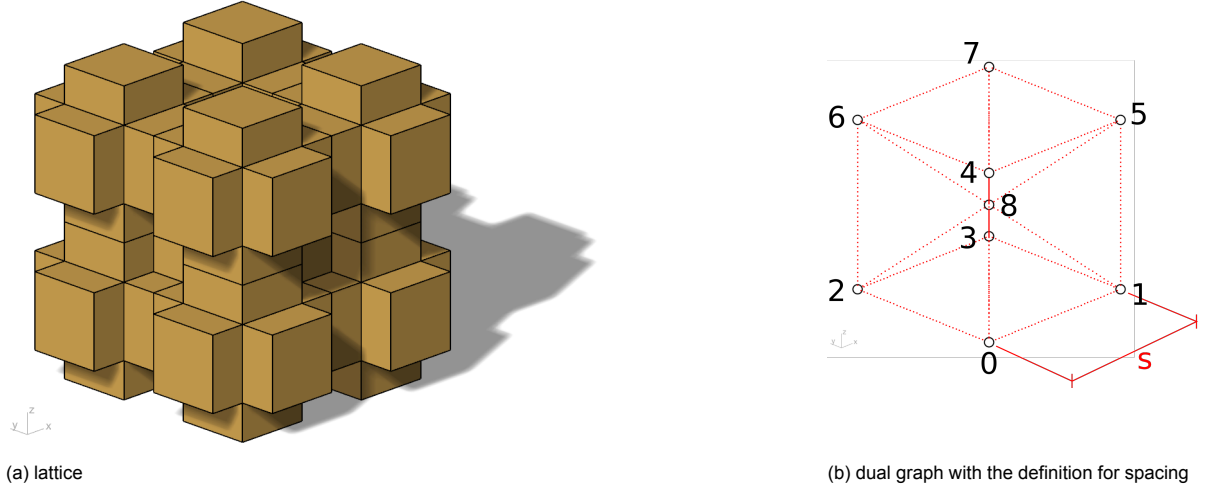


Figure 3.12: resulting lattice for a structure of size 2x2x2 using cubic spheres with element indexing shown

Having obtained the locations of all CS elements, one should then find which nodes (representing centroids of elementary cubes) are connected between elements. As it is not straight-forward which elements are in contact with which elements, this is not done on element-basis, but on node-basis. Algorithm 3 gives an outline on how this is achieved.

It is important to note that interfaces are not simply saved as a tuple of the two nodes they connect. Instead, an interface class is used wherein the direction, (original) length and center point of the interfaces is stored. This information proves to be necessary in the structural analysis.

Figure 3.12b shows how the spacing between elements is defined. A default value of $s = 1\text{ m}$ is taken. The length of an interface can then be computed as

$$l_{CS,interface} = \frac{s}{4} = 0.25\text{ m} \quad (3.1)$$

Here the spacing is divided by four, as that is the number of cubes within 'one spacing'. So the distance between cube centroids, and thus the length of an interface connecting the two, is computed by dividing the spacing by four. The area of an interface can be computed as

$$A_{CS,interface} = \left(\frac{s}{4}\right)^2 = 0.0625\text{ m}^2 \quad (3.2)$$

The volume of a CS can be calculated by first computing the volume of an elementary cube and then multiplying that with the number of elementary cubes inside a CS (which is 32):

$$V_{CS} = 32 \cdot \left(\frac{s}{4}\right)^3 = 32 \cdot 0.015625 = 0.5\text{ m}^3 \quad (3.3)$$

Then the mass of a CS can be computed as:

$$M_{CS} = \rho V_{CS} = 1000\text{ kg} \quad (3.4)$$

where a mass density of $\rho = 2000 \frac{\text{kg}}{\text{m}^3}$ is taken. This value is based on the densities of brick clay given in Carmichael, 2017, which range from $\rho = 1200 \frac{\text{kg}}{\text{m}^3}$ to $\rho = 2750 \frac{\text{kg}}{\text{m}^3}$.

These values are of importance to determine the stiffness and mass components of each element, as will be detailed in section 3.2.

Algorithm 3: Algorithm to initiate interfaces between CS elements**Data:** Structure size N_x , N_y and N_z ; List of nodes nl ; List of elements el **Result:** List of interfaces il $N_{elem,x} = 4N_x$ $N_{elem,y} = 4N_y$ $N_{elem,z} = 4N_z$

// the following is a list of number of nodes per row, layer and complete structure

 $N_{dims} = [N_{elem,x}, (N_{elem,x} * N_{elem,y}), (N_{elem,x} * N_{elem,y} * N_{elem,z})]$ **for** $element \in el$ **do** **for** $node \in element.nodes.items()$ **do** $index = node[0]$ $n1 = node[1]$ **for** $n \in range(3)$ **do**

// the following is a list of offsets to find the neighbouring node in x-, y- and z-directions

 $offset = [1, N_{elem,x}, N_{elem,x} * N_{elem,y}]$ // check if neighbouring node is not within the same row (for $n=0$) or layer ($n=1$) or within the structure ($n=2$), and continue to next n if so **if** $(index + offset[n]) \% N_{dims}[n] < index \% N_{dims}[n]$ **then** | **continue** **end** $n2 = nl[index + offset[n]]$

// node values refer back to the elements they belong in, so simply check if they are unequal to find out if they belong to different elements. Also check if node represents an element (and not empty space)

if $n1 \neq n2$ & $n2.type == Element$ **then** | $il.append(interface((n1, n2))$ **end** **end** **end****end**

3.1.3.2. Rhombic dodecahedron

To derive an algorithm which generates a lattice of RD elements, first the locations of the centroids of each elementary tetrahedron needs to be computed. This is not as straightforward as with elementary cubes. To find the location of the centroid of an arbitrary tetrahedron, one should compute the average of the node coordinates (Burkardt, 2010):

$$\mathbf{g}_{tetrahedron} = \frac{\mathbf{p}_1 + \mathbf{p}_2 + \mathbf{p}_3 + \mathbf{p}_4}{4} \quad (3.5)$$

where \mathbf{p}_i is a vector containing the spatial coordinates of vertex i of a tetrahedron.

As shown in figure 3.4 and 3.5a, a RD element consists of eight smaller cubes (which will be called *subcubes*) which have alternating chiralities. Applying eq. (3.5) on the eight tetrahedrons for a left chiral subcube (shown in red in figure 3.4) yields the following eight nodes:

$$\begin{aligned} \mathbf{n}_0 = s \begin{Bmatrix} 0.125 \\ 0.125 \\ 0.125 \end{Bmatrix}, \quad \mathbf{n}_1 = s \begin{Bmatrix} 0.1875 \\ 0.1875 \\ 0.1875 \end{Bmatrix}, \quad \mathbf{n}_2 = s \begin{Bmatrix} 0.375 \\ 0.375 \\ 0.125 \end{Bmatrix}, \quad \mathbf{n}_3 = s \begin{Bmatrix} 0.3125 \\ 0.3125 \\ 0.1875 \end{Bmatrix} \\ \mathbf{n}_4 = s \begin{Bmatrix} 0.375 \\ 0.125 \\ 0.375 \end{Bmatrix}, \quad \mathbf{n}_5 = s \begin{Bmatrix} 0.3125 \\ 0.1875 \\ 0.3125 \end{Bmatrix}, \quad \mathbf{n}_6 = s \begin{Bmatrix} 0.125 \\ 0.375 \\ 0.375 \end{Bmatrix}, \quad \mathbf{n}_7 = s \begin{Bmatrix} 0.1875 \\ 0.3125 \\ 0.3125 \end{Bmatrix} \end{aligned} \quad (3.6)$$

and for a right chiral subcube (shown in green in figure 3.4):

$$\begin{aligned} \mathbf{n}_0 = s \begin{Bmatrix} 0.375 \\ 0.125 \\ 0.125 \end{Bmatrix}, \quad \mathbf{n}_1 = s \begin{Bmatrix} 0.3125 \\ 0.1875 \\ 0.1875 \end{Bmatrix}, \quad \mathbf{n}_2 = s \begin{Bmatrix} 0.125 \\ 0.375 \\ 0.125 \end{Bmatrix}, \quad \mathbf{n}_3 = s \begin{Bmatrix} 0.1875 \\ 0.3125 \\ 0.1875 \end{Bmatrix} \\ \mathbf{n}_4 = s \begin{Bmatrix} 0.125 \\ 0.125 \\ 0.375 \end{Bmatrix}, \quad \mathbf{n}_5 = s \begin{Bmatrix} 0.1875 \\ 0.1875 \\ 0.3125 \end{Bmatrix}, \quad \mathbf{n}_6 = s \begin{Bmatrix} 0.375 \\ 0.375 \\ 0.375 \end{Bmatrix}, \quad \mathbf{n}_7 = s \begin{Bmatrix} 0.3125 \\ 0.3125 \\ 0.3125 \end{Bmatrix} \end{aligned} \quad (3.7)$$

Figure 3.13 shows the indexing convention used for the subcubes. It appears that from each subcube only two nodes are necessary to describe a RD as a graph. This means that in total 16 nodes (representing 16 tetrahedrons) make up a RD. The location of each RD is described by a right chiral (green) subcube, meaning that each RD is bounded by the set of subcubes shown in figure 3.13. Table 3.1 shows which nodes from eqs (3.6) and (3.7) belong to the RD that is bounded by this set of subcubes.

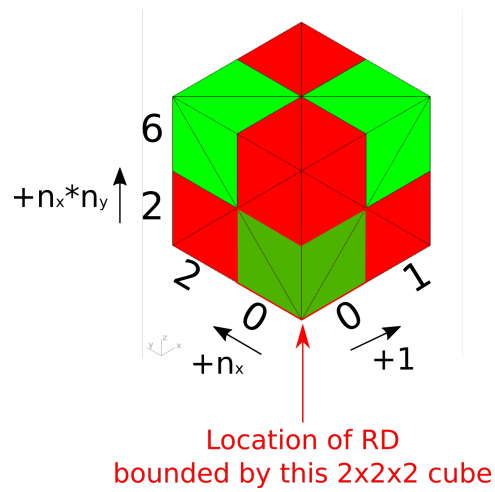


Figure 3.13: location of a RD inside a 2x2x2 cube along with the indexing convention of the subcubes

subcube index	chirality	nodes
0	R	n_6, n_7
1	L	n_6, n_7
2	L	n_4, n_5
3	R	n_4, n_5
4	L	n_2, n_3
5	R	n_2, n_3
6	R	n_0, n_1
7	L	n_0, n_1

Table 3.1: table showing which nodes from each subcube are necessary to describe a RD element

As for the edges to describe which tetrahedrons share faces, one could obtain these as tuples by using algorithm 4.

Algorithm 4: Algorithm to get a list of edges of a RD element

```

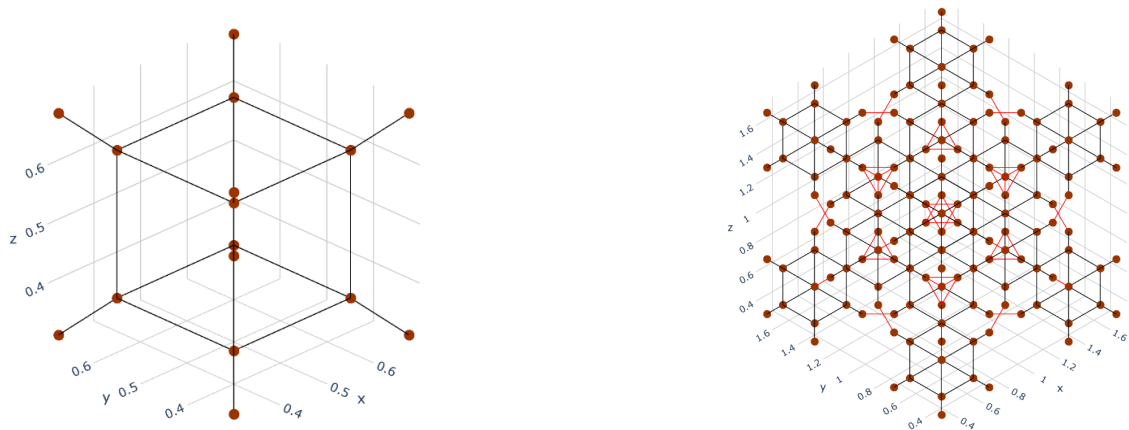
Data: List of subcubes sl
Result: List of edges edl

for i, s  $\in$  enumerate(sl) do
    // the following three checks are to generate edges between
    // subcubes
    if  $i\%2 == 0$  then
        | edl.append(s[6 - 2floor(i/2)], sl[i+1][6 - 2floor(i/2)])
    end
    if  $i\%4 < 2$  then
        | edl.append(s[6 - 2floor(i/2)], sl[i+2][6 - 2floor(i/2)])
    end
    if  $i < 4$  then
        | edl.append(s[6 - 2floor(i/2)], sl[i+4][6 - 2floor(i/2)])
    end

    // the following edge is inside the subcube
    edl.append(s[6 - 2floor(i/2)], s[7 - 2floor(i/2)])
end

```

One could then slightly adjust the checks and indexing of this algorithm to allow for generation inside a larger graph that represent a greater structural system. Another option is to first generate and then translate the node and edge indexes from a sub-graph (representing a RD element) to the larger graph. The resulting nodal dual graph for RD structures is shown in figure 3.14.



(a) single element

(b) a structure of size 2x2x2

Figure 3.14: nodal dual graphs for RD-based structures

The next step is to define an algorithm to find the location of each RD when a certain structure size is given. The structure size is given as the amount of 2x2x2 subcube sets as shown in figure 3.13 in a certain direction. If the structure size were given in the amount in of subcubes, no elements would be generated for a structure size of 1x1x1. A detailed outline on how to find the locations of the RD elements is given in algorithm 5.

The final step is to generate interfaces between RDs. Because of the way the data is structured, it is surprisingly simple to achieve this (algorithm 6).

Algorithm 5: Algorithm to get the locations of all RD elements

Data: Structure size N_x , N_y and N_z
Result: Locations of RD elements loc_{RD}

```
// get the amount of subcubes in all directions
 $N_{sc,x} = 2N_x$ 
 $N_{sc,y} = 2N_y$ 
 $N_{sc,z} = 2N_z$ 

 $chir = True$ 

for  $z \in range(N_{sc,z} - 1)$  do
  for  $y \in range(N_{sc,y} - 1)$  do
    for  $x \in range(N_{sc,x} - 1)$  do
      if  $chir$  then
        |  $loc_{RD}.append(x + y * N_{sc,x} + z * (N_{sc,x} * N_{sc,y}))$ 
      end
      |  $chir = not chir$ 
    end
  end
end
end
```

Algorithm 6: Algorithm to initiate interfaces between RD elements

Data: Structure size N_x , N_y and N_z ; List of subcubes sl
Result: List of interfaces il

```
 $N_{sc,x} = 2N_x$ 
 $N_{sc,y} = 2N_y$ 
 $N_{sc,z} = 2N_z$ 

for  $z \in range(N_{sc,z})$  do
  for  $y \in range(N_{sc,y})$  do
    for  $x \in range(N_{sc,x})$  do
       $s = sl[x + y * N_{sc,y} + z * (N_{sc,x} * N_{sc,y})]$ 

      // iterate from 1 to (not including) 7 taking steps of 2
      for  $i \in range(1, 7, 2)$  do
         $n1 = s[i]$ 

        for  $j \in range(i, 9, 2)$  do
           $n2 = s[j]$ 

          if  $n1.type == Element \& n2.type == Element$  then
            |  $il.append(interface((n1, n2))$ 
          end
        end
      end
    end
  end
end
end
```

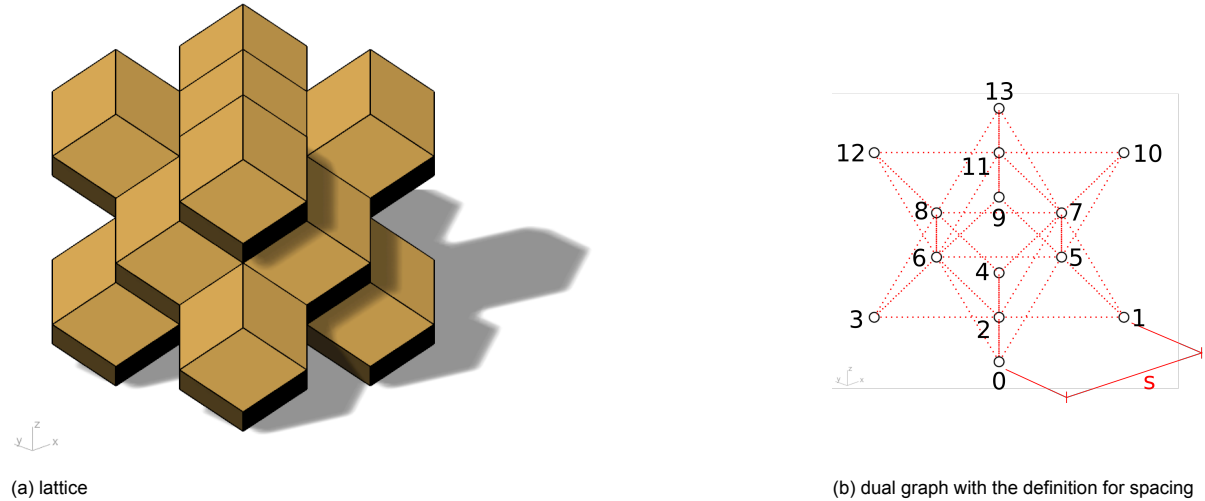


Figure 3.15: resulting lattice for a structure of size 2x2x2 using rhombic dodecahedrons with element indexing shown

Figure 3.15 shows a 2x2x2 RD-based structure geometrically and its dual graph. Again, note the difference between this dual graph and the nodal dual graph. It is noted that for the same structural dimensions, a RD-based structure consists of more elements than a CS-based structure (cf. figures 3.12 and 3.15).

Like the CS elements, the only unknown to generate geometries is the spacing between the elements. A standard value for the spacing is taken as $s = 1\text{m}$. With this value the length and area of an interface can be calculated. The length is obtained as:

$$l_{RD,interface} = \sqrt{(0.125s)^2 + (0.125s)^2} \approx 0.1768\text{ m} \quad (3.8)$$

One should note that the interface is not rhombic in shape as figure 3.15a could suggest. Figure 3.6 shows that all interfaces represent triangles, all of which being equal. By calculating the base length and height of a triangle, the following value is obtained for its area:

$$A_{RD,interface} = \frac{1}{2} \left(\frac{s}{2} \sqrt{2} \cdot \frac{s}{4} \right) = \frac{1}{16} s^2 \sqrt{2} \approx 0.0884\text{ m}^2 \quad (3.9)$$

To obtain the volume of a tetrahedron the following should be computed (Burkardt, 2010):

$$V_{tetrahedron} = \pm \begin{vmatrix} p_{1[1]} & p_{1[2]} & p_{1[3]} & 1 \\ p_{2[1]} & p_{2[2]} & p_{2[3]} & 1 \\ p_{3[1]} & p_{3[2]} & p_{3[3]} & 1 \\ p_{4[1]} & p_{4[2]} & p_{4[3]} & 1 \end{vmatrix} \quad (3.10)$$

It is necessary to specify the absolute value in eq. (3.10), as the sign of the volume is dependent on how the vertices are ordered. Doing this for all tetrahedrons yields the following volume for a single RD: $V_{RD} = 0.25s^3 = 0.25\text{ m}^3$. With the volume known and assuming a uniform mass density, one could simply calculate the mass of a RD as follows:

$$M_{RD} = \rho V_{RD} = 500\text{ kg} \quad (3.11)$$

where a mass density of $\rho = 2000 \frac{\text{kg}}{\text{m}^3}$ is again taken.

3.1.3.3. Voids

It should be possible for the user to define voids within a structure. Such voids may represent minimal spaces necessary for rooms or corridors. Voids can intuitively be given as ranges of x-, y- and

z-coordinates. Within those ranges no elements can be generated. To keep computation relatively simple, it is chosen to check whether the centroid of an element is within the specified domain. If that is the case, the element is not generated. An algorithm is outlined for both CS (algorithm 7) and RD elements (algorithm 8). Note that generating an element is in principle saving the elements data (such as location, volume, nodes) to memory and setting the nodes that belong to it to the right values (i.e. setting the nodes as references to the element that they are part of).

Some generated structures consisting using CS and RD elements are shown in figure 3.16. An indefinite number of voids can be defined, which means that there is a large amount of freedom when defining an initial structural topology in this regard.

Algorithm 7: Algorithm to check if current CS is within a void

Data: Structure size N_x , N_y and N_z ; Spacing s ; List of voids $Vlist$

```

for  $z \in \text{range}(N_z)$  do
  for  $y \in \text{range}(N_y)$  do
    for  $x \in \text{range}(N_x)$  do
      InVoid = False

      for  $\text{void} \in Vlist$  do
         $\text{locx} = 0.5s + x * s$ 
         $\text{locy} = 0.5s + y * s$ 
         $\text{locz} = 0.5s + z * s$ 

        // check if the x-, y-, z-coordinates are within the domain
        // of the current void, and set InVoid to True if so
        if  $\text{locx} > \text{void}[0][0] \ \& \ \text{locx} < \text{void}[0][1] \ \& \ \text{locy} > \text{void}[1][0] \ \& \ \text{locy} <$ 
            $\text{void}[1][1] \ \& \ \text{locz} > \text{void}[2][0] \ \& \ \text{locz} < \text{void}[2][1]$  then
          | InVoid = True
        end
      end
    end

    if InVoid then
      | continue
    end

    GenerateElement()
  end
end
end

```

Algorithm 8: Algorithm to check if current RD is within a void**Data:** Structure size N_x , N_y , and N_z ; Spacing s ; List of voids $Vlist$

```

// get the amount of subcubes in all directions
 $N_{sc,x} = 2N_x$ 
 $N_{sc,y} = 2N_y$ 
 $N_{sc,z} = 2N_z$ 

 $chir = True$ 

for  $z \in range(N_{sc,z} - 1)$  do
  for  $y \in range(N_{sc,y} - 1)$  do
    for  $x \in range(N_{sc,x} - 1)$  do
      if  $chir$  then
         $InVoid = False$ 

        for  $void \in Vlist$  do
           $locx = 0.5s + 0.5x * s$ 
           $locy = 0.5s + 0.5y * s$ 
           $locz = 0.5s + 0.5z * s$ 

          // check if the x-, y-, z-coordinates are within the
          // domain of the current void, and set  $InVoid$  to True if
          // so
          if  $locx > void[0][0] \ \& \ locx < void[0][1] \ \& \ locy > void[1][0] \ \& \ locy <$ 
           $void[1][1] \ \& \ locz > void[2][0] \ \& \ locz < void[2][1]$  then
             $InVoid = True$ 
          end
        end

        if  $InVoid$  then
           $continue$ 
        end

         $GenerateElement()$ 
      end
       $chir = not chir$ 
    end
  end
end

```

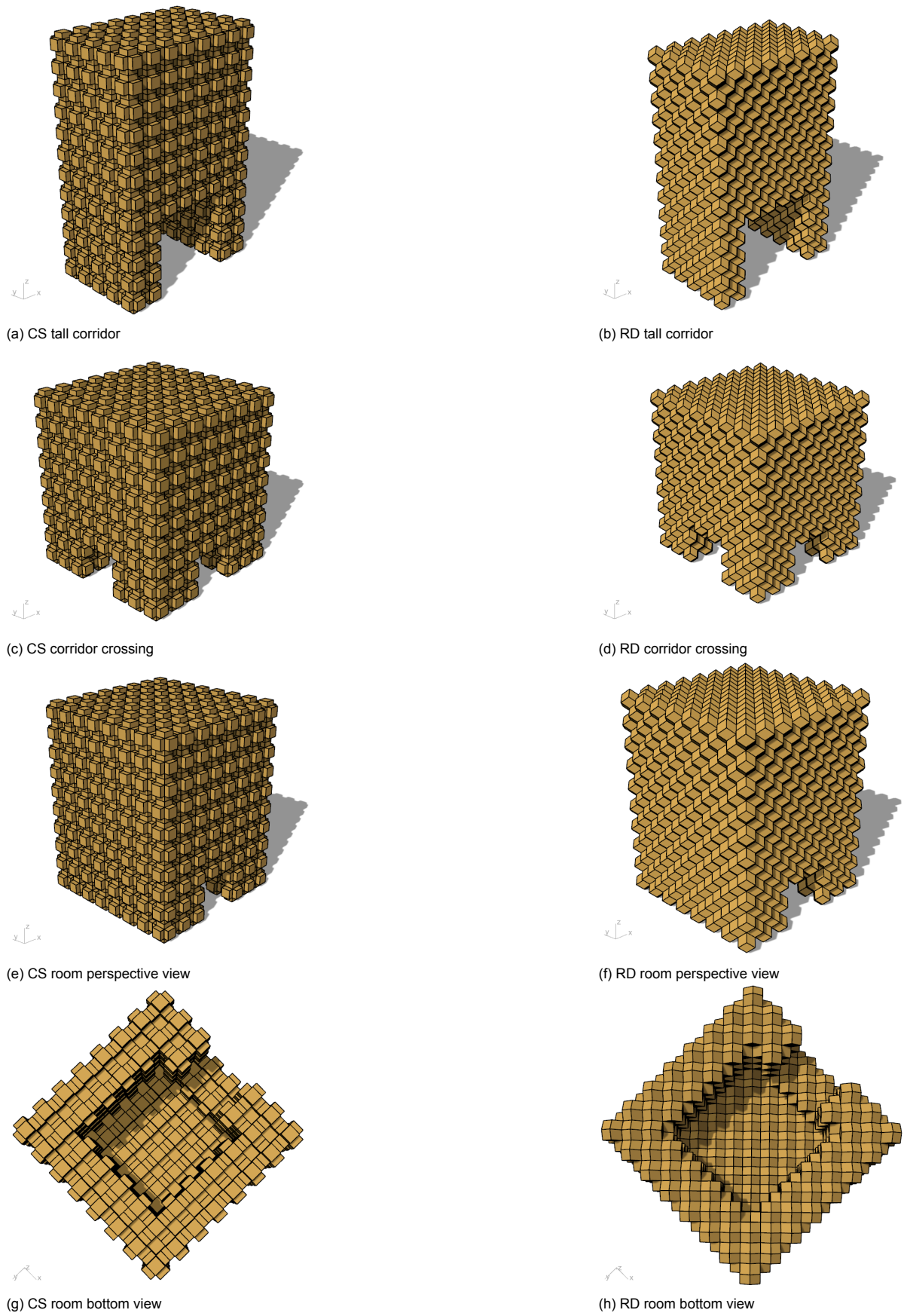


Figure 3.16: results of geometry generation

3.2. Structural Analysis

Now that it is possible to generate arbitrary masonry structures, it is necessary to model the mechanical behaviour of these structures in order to optimize them. In the following sections an outline is given for three different analysis methods, namely: a time-explicit analysis method using global damping, a time-explicit analysis method using local damping, and a static analysis method. These methods have some overlap, which means that the values used in these methods are mostly the same, but the way of computing outcomes is different. The results of each model are then shown, and notable differences are showcased.

A small note on notations should be made before any model is explained. Subscripts as names for vectors and matrices are written bold without brackets: \mathbf{A}_{ij} , while indices are written non-bold between square brackets: $\mathbf{A}_{[ij]}$

3.2.1. Time-explicit analysis using global damping

This model includes time-explicit calculations to obtain acceleration and subsequently compute displacements and velocities for the next time-step. Essentially, the model is described by the following equation of motion Friswell and Mottershead, n.d., chapter 2:

$$\hat{\mathbf{M}}^{(t)} \hat{\mathbf{u}}^{(t)} + \mathbf{C}^{(t)} \hat{\mathbf{u}}^{(t)} + \hat{\mathbf{K}}^{(t)} \hat{\mathbf{u}}^{(t)} = \hat{\mathbf{f}}^{(t)} \quad (3.12)$$

where:

$\hat{\mathbf{M}}^{(t)}$: system mass matrix at time t

$\hat{\mathbf{K}}^{(t)}$: system stiffness matrix at time t

$\mathbf{C}^{(t)}$: global damping coefficient at time t

$\hat{\mathbf{f}}^{(t)}$: system force vector time t

$\hat{\mathbf{u}}^{(t)}$: system displacement vector at time t

$\hat{\mathbf{u}}^{(t)}$: system velocity vector at time t

$\hat{\mathbf{u}}^{(t)}$: system acceleration vector at time t

The following sections will provide a detailed outline on how all the components of this equation are computed.

3.2.1.1. Stiffness matrix

The computation of the stiffness matrix is inspired by Jing and Stephansson, 2007, chapter 11. The interfaces between elements as described in section 3.1.2.2 are used to set up this matrix. As a first step, one should compute the stiffness values of the springs that are represented by the interfaces. This is done using Hooke's Law, which states the following:

$$\sigma_n = E\epsilon_n \quad (3.13)$$

where:

E : the Young's Modulus of an element (Pa)

ϵ_n : the strain in the normal direction of an interface

σ_n : the stress in the normal direction of an interface (Pa)

The spring stiffness values are computed slightly differently for dry connections and mortar connections.

For dry connections, the deformation of the elements is in essence lumped in the interfaces. As it was stated in section 3.1.2, the connections within an element are considered to be rigid, while the interfaces are modelled as springs. This means that the stiffness components of the edges between

nodes that represent interfaces can be computed by eq. (3.13). Slightly rewriting this expression enables one to find an expression for the stiffness component which is normal to an element's face k_n :

$$\begin{aligned}\Delta\sigma_n &= E\Delta\epsilon_n \\ \frac{\Delta F_n}{A} &= E\frac{\Delta l}{l_0} \\ \frac{\Delta F_n}{\Delta l} &= \frac{EA}{l_0} = k_n\end{aligned}\quad (3.14)$$

where:

A : the area that is represented by a spring. Values are calculated by eqs (3.2) and (3.9) (m²)

ΔF_n : force change in the normal direction of an interface (N)

Δl : spring (or interface) length change (m)

l_0 : the original spring (or interface) length. Values are calculated by eqs (3.1) and (3.8) (m)

To find a value for the shear spring stiffness value k_s , the relatively simple approach followed by Červenka et al., 2000 is used. This results in:

$$G = \frac{E}{2(1 + \nu)} \quad (3.15)$$

$$k_s = \frac{GA}{l_0} \quad (3.16)$$

where:

ν : the Poisson's ratio of an element

Note the resemblance between eqs (3.14) and (3.16).

For the normal stiffness of mortar connections Hooke's law is again used, along with the notion of springs in series. Springs connected in series may not necessarily have the same elongation, but they have the same applied force. If this were not the case, there would be a force imbalance between springs. With this principle, the following can be derived:

$$\begin{aligned}\Delta F_n &= k_n(\Delta l + \Delta t) \\ \frac{1}{\Delta F_n} &= \frac{1}{k_n(\Delta l + \Delta t)} \\ \frac{(\Delta l + \Delta t)}{\Delta F_n} &= \frac{\Delta l}{\Delta F_n} + \frac{\Delta t}{\Delta F_n} = \frac{1}{k_n} = \frac{1}{k_e} + \frac{1}{k_m} \\ \frac{1}{k_n} &= \frac{l_0 - t_0}{E_e A} + \frac{t_0}{E_m A} \\ \frac{1}{k_n} &= \frac{E_m A(l_0 - t_0)}{E_e E_m A^2} + \frac{E_e A t_0}{E_e E_m A^2} = \frac{E_m(l_0 - t_0) + E_e t_0}{E_e E_m A} \\ k_n &= \frac{E_e E_m A}{E_m(l_0 - t_0) + E_e t_0}\end{aligned}\quad (3.17)$$

where:

E_e : the Young's Modulus of an element (Pa)

E_m : the Young's Modulus of the mortar (Pa)

Δt : mortar thickness change (m)

t_0 : the original mortar thickness (m)

k_s is calculated using eq. (3.16). Some values are different however:

$$G_m = \frac{E_m}{2(1 + \nu)} \quad (3.18)$$

$$k_s = \frac{G_m A}{t_0} \quad (3.19)$$

For both dry connections and mortar connections, some sort of non-linear material behaviour should be defined. It is obvious that once contact between elements is broken, no forces can act between them anymore. The same can be stated about mortar connections; once the mortar cracks no more forces are possible (at least in tension). One way this can be modelled is by setting a limit to the strain of a spring. In this model it is assumed that springs have non-zero stiffness components only when:

$$-90\% \leq \frac{\Delta l}{l_0} \leq 0\% \quad (3.20)$$

This essentially means that after the spring has elongated more than 0% or compressed more than 90% of its original length, it will be no longer considered as having any stiffness. In this way, crushing and loss of contact (or cracking) is modelled. Even though the initial tensile strain limit of concrete is 0.01% and the initial compressive strain limit is 0.1% (Ng et al., 2010), it is chosen to fully neglect tension of mortar connections while disregarding crushing for the most part. This allows the user to set a lower Young's Modulus to model different materials or to increase the time-step size (section 3.2.1.4). Figure 3.17 shows the constitutive relation between strains (or relative elongation) and the normal stress which originates from eq. (3.14) and the area of the interface. It should be noted that whenever the elongation of an interface returns within these bounds later in the analysis, the stiffness components are again reintroduced. The analyses carried out in subsequent sections make use of dry connection types, but the final implementation includes both options.

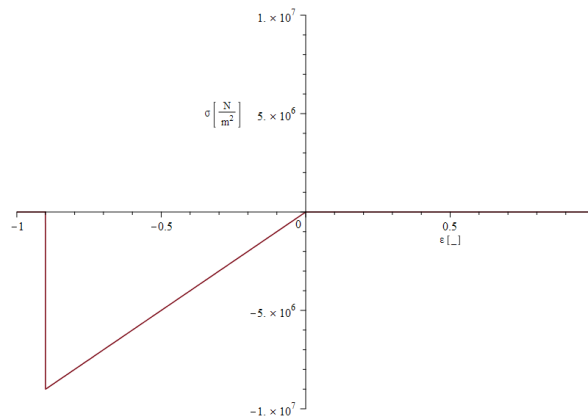


Figure 3.17: material behaviour for each interface spring along its axis with $E = 10 \cdot 10^6$ Pa

The stiffness components derived above can be used to define the relation between the degrees of freedom of each element and the forces that act on it. The following stiffness matrix follows from Jing and Stephansson, 2007:

$$\bar{\mathbf{K}}'_{ii,mn} = \begin{bmatrix} k_n & 0 & 0 & 0 & 0 & 0 \\ 0 & k_s & 0 & 0 & 0 & 0 \\ 0 & 0 & k_s & 0 & 0 & 0 \\ 0 & 0 & 0 & k_s r_{mn,i,x'} r_{mn,i,x'} & 0 & 0 \\ 0 & 0 & 0 & 0 & k_s r_{mn,i,y'} r_{mn,i,y'} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_s r_{mn,i,z'} r_{mn,i,z'} \end{bmatrix} \quad (3.21)$$

$$\mathbf{K}'_{ij,mn} = \begin{bmatrix} \bar{\mathbf{K}}'_{ii,mn} & -\bar{\mathbf{K}}'_{ij,mn} \\ -\bar{\mathbf{K}}'_{ij,mn} & \bar{\mathbf{K}}'_{jj,mn} \end{bmatrix} \quad (3.22)$$

where:

$r_{mn,i,x'}$: the position along the local x-direction of the center of interface mn from the centroid of element i (m)

This matrix relates the degrees of freedom of an element to the external forces acting on it, in the following manner:

$$\underbrace{\begin{Bmatrix} f'_i \\ f'_j \end{Bmatrix}}_{\text{force vector } \mathbf{f}'_{ij}} = \mathbf{K}'_{ij,mn} \underbrace{\begin{Bmatrix} u'_i \\ u'_j \end{Bmatrix}}_{\text{displacement vector } \mathbf{u}'_{ij}} \quad (3.23)$$

where:

$$\mathbf{u}'_i = (u_{i,x'}, u_{i,y'}, u_{i,z'}, \phi_{i,x'}, \phi_{i,y'}, \phi_{i,z'})^T$$

$$\mathbf{f}'_i = (f_{i,x'}, f_{i,y'}, f_{i,z'}, m_{i,x'}, m_{i,y'}, m_{i,z'})^T$$

$u_{i,x'}$: translation of element i in the local x-direction (m)

$\phi_{i,x'}$: rotation of element i around the local x-axis (rad)

$f_{i,x'}$: external force acting on element i in the local x-direction (N)

$m_{i,x'}$: external moment acting on element i around the local x-axis (Nm)

Note that eq. (3.23) does not include third or fourth order tensors, but uses the previous equations to describe first and second order tensors in a compact manner.

Force components in force vector \mathbf{f}'_{ij} describe external forces acting on elements. This includes self-weight and user-defined forces, which can be defined by slightly modifying algorithm 7 for CS structures or algorithm 8 for RD structures to apply forces on elements within spatial bounds. In the same way, displacements in displacement vector \mathbf{u}'_{ij} can be set to zero for constrained degrees of freedom.

It is also an important point to mention that according to Hibbeler, 2004, chapter 20, rotations cannot be stored as a vector, as the order of applying the rotations has an impact on the resulting total rotation so: $\phi_{x,i} + \phi_{y,i} \neq \phi_{y,i} + \phi_{x,i}$. For infinitesimal rotations, the order of rotations does not matter and the rotations may be stored as a vector: $d\phi_{x,i} + d\phi_{y,i} = d\phi_{y,i} + d\phi_{x,i}$. This leads to an assumption that the rotations in the outlined discrete element model should be small around at least two axes, say $|\phi_{k,i}| < 0.1$. Thus when one observes large values for rotations of an element around multiple axes, the resulting rotations should be regarded as erroneous. Ultimately, it is not expected that these

incorrect values will affect the results of the optimization, as elements having large displacements, be it translations or rotations, will likely be removed from the structure.

Using the stiffness matrix as defined in eqs (3.21) and (3.22) prevents the occurrence of toppling of any element due to insufficient supporting when only self-weight is applied. This is due to the uncoupled nature of this matrix: a force in any direction can never result in a moment (and subsequent rotation) of an element.

To solve this problem, a new stiffness matrix is defined by making use of the displacement method. In this method a small positive displacement is applied and the resulting force/moment balance equations are solved. It is important to note that the displacements that are applied are small enough so that the resulting forces from springs that are perpendicular to the direction of the displacement are insignificantly small and will thus be neglected in the force balance equations (figures 3.18a, 3.18b and 3.18c).

As an example one column of the stiffness matrix between two elements for one interface will be derived. Suppose elements i and j are connected by an interface at node m in element i and at node n in element j . The physical interface between the elements will be somewhere between nodes m and n . For simplicity, it is assumed it is exactly mid-way between the nodes at point p'_{mn} . The distance between the point of contact and the center-of-mass of elements is calculated as:

$$\mathbf{r}'_{mn,i} = \mathbf{p}'_{mn} - \mathbf{p}'_i \quad (3.24)$$

Note that eq. (3.24) makes use of local coordinates aligned to the interface direction. Now a small displacement $du_{i,x'}$ is applied for element i (figure 3.18d). The resulting force balances are:

$$\begin{aligned} df_{sp,i,x'} + f_{i,x'} &= -k_n du_{i,x'} + f_{i,x'} = 0 \\ df_{sp,i,y'} + f_{i,y'} &= 0 + f_{i,y'} = 0 \\ df_{sp,i,z'} + f_{i,z'} &= 0 + f_{i,z'} = 0 \end{aligned} \quad (3.25)$$

To calculate the resulting moments on element i , use is made of the following definition (Hibbeler, 2004, chapter 17):

$$\mathbf{m} = \mathbf{r} \times \mathbf{f} \quad (3.26)$$

Evaluating this cross product for this case yields:

$$\mathbf{r}'_{mn,i} \times \begin{Bmatrix} -k_n du_{i,x'} \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -k_n du_{i,x'} r_{mn,i,z'} \\ k_n du_{i,x'} r_{mn,i,y'} \end{Bmatrix} \quad (3.27)$$

Now the moment balances for element i can be formulated as:

$$\begin{aligned} dm_{sp,i,x'} + m_{i,x'} &= 0 + m_{i,x'} = 0 \\ dm_{sp,i,y'} + m_{i,y'} &= -k_n du_{i,x'} r_{mn,i,z'} + m_{i,y'} = 0 \\ dm_{sp,i,z'} + m_{i,z'} &= k_n du_{i,x'} r_{mn,i,y'} + m_{i,z'} = 0 \end{aligned} \quad (3.28)$$

When the same procedure is followed for element j and it is recognized that the force is now pointed in the positive local-x direction, the following relations are obtained:

$$\begin{aligned} df_{sp,j,x'} + f_{j,x'} &= k_n du_{i,x'} + f_{j,x'} = 0 \\ df_{sp,j,y'} + f_{j,y'} &= 0 + f_{j,y'} = 0 \\ df_{sp,j,z'} + f_{j,z'} &= 0 + f_{j,z'} = 0 \\ dm_{sp,j,x'} + m_{j,x'} &= 0 + m_{j,x'} = 0 \\ dm_{sp,j,y'} + m_{j,y'} &= k_n du_{i,x'} r_{mn,j,z'} + m_{j,y'} = 0 \\ dm_{sp,j,z'} + m_{j,z'} &= -k_n du_{i,x'} r_{mn,j,y'} + m_{j,z'} = 0 \end{aligned} \quad (3.29)$$

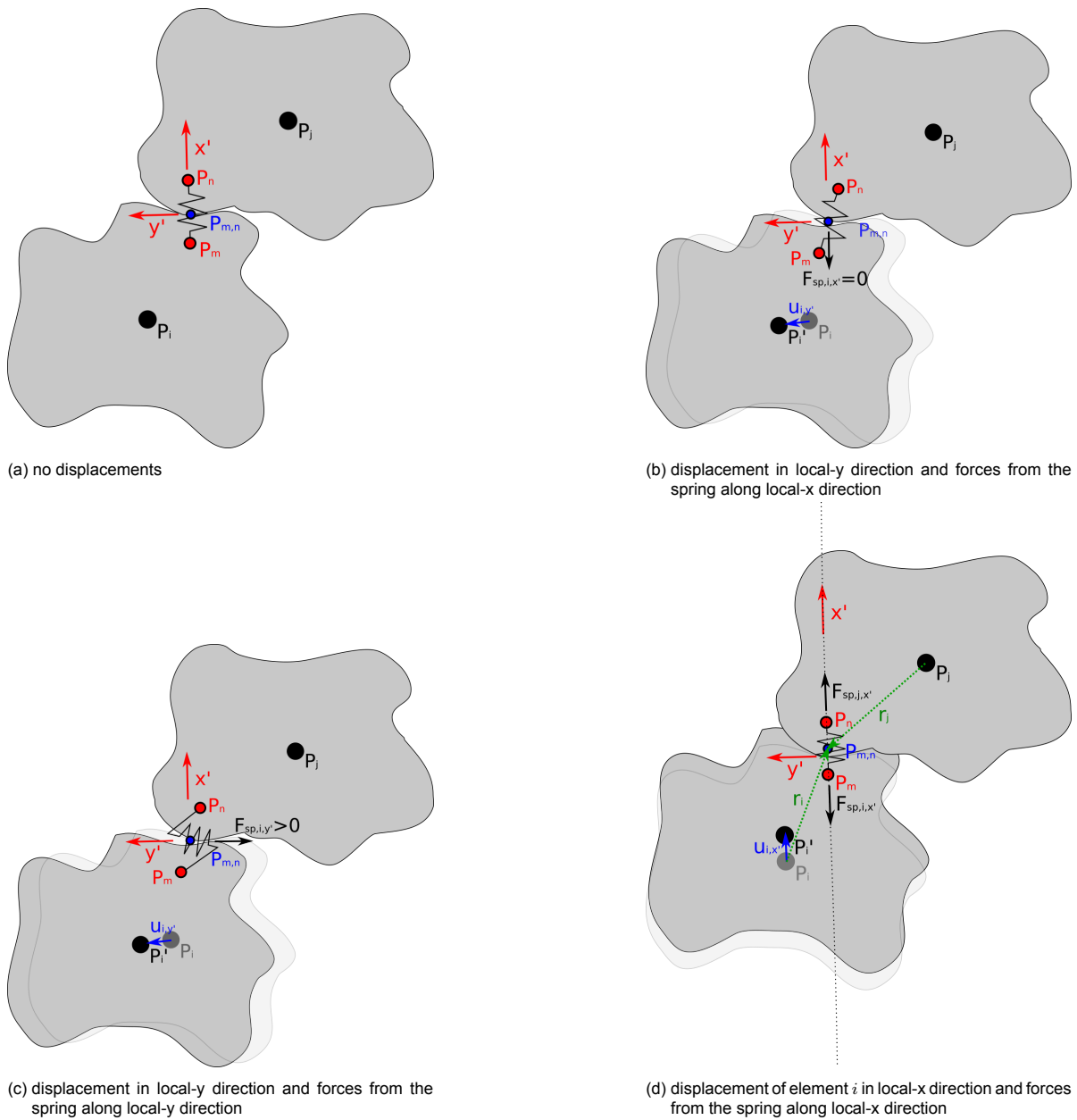


Figure 3.18: general representation of element *i* and element *j* with nodes *m* and *n* connected by an interface in a local coordinate system

When this procedure is repeated for all degrees of freedoms of the two elements, the following stiffness matrix is obtained for that interface:

$$\bar{K}'_{ii,mn} = \begin{bmatrix} k_n & 0 & 0 & 0 & k_n r_{mn,i,z'} & -k_n r_{mn,i,y'} \\ 0 & k_s & 0 & -k_s r_{mn,i,z'} & 0 & k_s r_{mn,i,x'} \\ 0 & 0 & k_s & k_s r_{mn,i,y'} & -k_s r_{mn,i,x'} & 0 \\ 0 & -k_s r_{mn,i,z'} & k_s r_{mn,i,y'} & k_1 & k_4 & k_5 \\ k_n r_{mn,i,z'} & 0 & -k_s r_{mn,i,x'} & k_4 & k_2 & k_6 \\ -k_n r_{mn,i,y'} & k_s r_{mn,i,x'} & 0 & k_5 & k_6 & k_3 \end{bmatrix}$$

with:

$$\begin{aligned} k_1 &= k_s r_{mn,i,y'}^2 + k_s r_{mn,i,z'}^2 \\ k_2 &= k_s r_{mn,i,x'}^2 + k_n r_{mn,i,z'}^2 \\ k_3 &= k_s r_{mn,i,y'}^2 + k_n r_{mn,i,y'}^2 \\ k_4 &= -k_s r_{mn,i,x'} r_{mn,i,y'} \\ k_5 &= -k_s r_{mn,i,x'} r_{mn,i,z'} \\ k_6 &= -k_s r_{mn,i,y'} r_{mn,i,z'} \end{aligned} \quad (3.30)$$

$$\bar{K}'_{ij,mn} = \begin{bmatrix} -k_n & 0 & 0 & 0 & -k_n r_{mn,j,z'} & k_n r_{mn,j,y'} \\ 0 & -k_s & 0 & k_s r_{mn,j,z'} & 0 & -k_s r_{mn,j,x'} \\ 0 & 0 & -k_s & -k_s r_{mn,j,y'} & k_s r_{mn,j,x'} & 0 \\ 0 & k_s r_{mn,i,z'} & -k_s r_{mn,i,y'} & k_7 & k_{10} & k_{11} \\ -k_n r_{mn,i,z'} & 0 & k_s r_{mn,i,x'} & k_{12} & k_8 & k_{13} \\ k_n r_{mn,i,y'} & -k_s r_{mn,i,x'} & 0 & k_{14} & k_{15} & k_9 \end{bmatrix}$$

with:

$$\begin{aligned} k_7 &= -k_s r_{mn,i,y'} r_{mn,j,y'} - k_s r_{mn,i,z'} r_{mn,j,z'} \\ k_8 &= -k_s r_{mn,i,x'} r_{mn,j,x'} - k_n r_{mn,i,z'} r_{mn,j,z'} \\ k_9 &= -k_s r_{mn,i,x'} r_{mn,j,x'} - k_n r_{mn,i,y'} r_{mn,j,y'} \\ k_{10} &= k_s r_{mn,i,y'} r_{mn,j,x'} \\ k_{11} &= k_s r_{mn,i,z'} r_{mn,j,x'} \\ k_{12} &= k_s r_{mn,i,x'} r_{mn,j,y'} \\ k_{13} &= k_s r_{mn,i,z'} r_{mn,j,y'} \\ k_{14} &= k_s r_{mn,i,x'} r_{mn,j,z'} \\ k_{15} &= k_s r_{mn,i,y'} r_{mn,j,z'} \end{aligned} \quad (3.31)$$

$$\bar{K}'_{jj,mn} = \begin{bmatrix} k_n & 0 & 0 & 0 & k_n r_{mn,j,z'} & -k_n r_{mn,j,y'} \\ 0 & k_s & 0 & -k_s r_{mn,j,z'} & 0 & k_s r_{mn,j,x'} \\ 0 & 0 & k_s & k_s r_{mn,j,y'} & -k_s r_{mn,j,x'} & 0 \\ 0 & -k_s r_{mn,j,z'} & k_s r_{mn,j,y'} & k_{16} & k_{19} & k_{20} \\ k_n r_{mn,j,z'} & 0 & -k_s r_{mn,j,x'} & k_{19} & k_{17} & k_{21} \\ -k_n r_{mn,j,y'} & k_s r_{mn,j,x'} & 0 & k_{20} & k_{21} & k_{18} \end{bmatrix}$$

with:

$$\begin{aligned} k_{16} &= k_s r_{mn,j,y'}^2 + k_s r_{mn,j,z'}^2 \\ k_{17} &= k_s r_{mn,j,x'}^2 + k_n r_{mn,j,z'}^2 \\ k_{18} &= k_s r_{mn,j,y'}^2 + k_n r_{mn,j,y'}^2 \\ k_{19} &= -k_s r_{mn,j,x'} r_{mn,j,y'} \\ k_{20} &= -k_s r_{mn,j,x'} r_{mn,j,z'} \\ k_{21} &= -k_s r_{mn,j,y'} r_{mn,j,z'} \end{aligned} \quad (3.32)$$

$$K'_{ij,mn} = \begin{bmatrix} \bar{K}'_{ii,mn} & \bar{K}'_{ij,mn} \\ \bar{K}'_{ij,mn}^T & \bar{K}'_{jj,mn} \end{bmatrix} \quad (3.33)$$

To transform the tensors of eq. (3.23) from a local coordinate system to a global coordinate system, a transformation matrix is necessary. Jing and Stephansson, 2007 presents the following transformation matrix:

$$T_{\text{partial}} = \begin{bmatrix} \cos(x'_x) & \cos(x'_y) & \cos(x'_z) \\ \cos(y'_x) & \cos(y'_y) & \cos(y'_z) \\ \cos(z'_x) & \cos(z'_y) & \cos(z'_z) \end{bmatrix} \quad (3.34)$$

$$T = \begin{bmatrix} T_{\text{partial}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & T_{\text{partial}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & T_{\text{partial}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & T_{\text{partial}} \end{bmatrix} \quad (3.35)$$

where:

x'_y : the angle between the local x-axis and the global y-axis

The transformation matrix described in (3.35) is orthogonal. The following thus holds:

$$T^{-1} = T^T \quad (3.36)$$

An arbitrary vector of length 12 is now transformed from local to global coordinates as follows:

$$\mathbf{a} = T^T \mathbf{a}' \quad (3.37)$$

and likewise:

$$\mathbf{a}' = T \mathbf{a} \quad (3.38)$$

For matrices a different approach is necessary. As an example, relation (3.23) is considered. In the following example the local stiffness matrix K' is transformed into global stiffness matrix K using eqs (3.37) and (3.38):

$$\begin{aligned} \mathbf{f} &= \mathbf{K}\mathbf{u} = \mathbf{T}^T(\mathbf{K}'\mathbf{u}') \\ \mathbf{K}\mathbf{u} &= \mathbf{T}^T(\mathbf{K}'\mathbf{T}\mathbf{u}) \\ \mathbf{K} &= \mathbf{T}^T\mathbf{K}'\mathbf{T} \end{aligned} \quad (3.39)$$

In the same way, the following can be derived for the reverse case:

$$\mathbf{K}' = \mathbf{T}\mathbf{K}\mathbf{T}^T \quad (3.40)$$

A significant simplification can be made with the transformation matrix T . When this matrix is applied to rotate the stiffness matrix of an arbitrary interface from a local to the global coordinate system, the cosines of the angles between the axes of these two coordinate systems should be calculated. The definition of the dot product is given as:

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \quad (3.41)$$

If all vectors have a norm of 1, the denominator of eq. (3.41) cancels out. So to efficiently calculate the cosine of an angle between two axes, unit vectors are needed in the directions of these axes:

$$\cos(y'_x) = \mathbf{e}'_y \cdot \mathbf{e}_x \quad (3.42)$$

The definitions for the unit vectors for the global coordinate system are straight-forward:

$$\begin{aligned} \mathbf{e}_x &= (1, 0, 0)^T \\ \mathbf{e}_y &= (0, 1, 0)^T \\ \mathbf{e}_z &= (0, 0, 1)^T \end{aligned} \quad (3.43)$$

For the local axes it will be enforced that the local x-axis always coincides with the direction of the spring. The other two axes are two vectors that are perpendicular to the local x-axis while also perpendicular to each other. Algorithm 9 shows how these axes are computed.

Algorithm 9: Algorithm for computing local axes of interface

Data: Location vectors P_1 and P_2
Result: Local axes as unit vectors e_x , e_y and e_z
 $x = P_2 - P_1$
 $e_x = x / \text{norm}(x)$
 $y = \text{cross}(e_x, \text{vector}([1, 0, 0]))$
if $\text{norm}(y) == 0.0$ **then**
 | $y = \text{cross}(e_x, \text{vector}([0, 1, 0]))$
end
 $e_y = y / \text{norm}(y)$
 $e_z = \text{cross}(e_x, e_y)$

With these axes defined and making use of eq. (3.42), eq. (3.34) can be simplified as follows:

$$\mathbf{T}_{\text{partial}} = \begin{bmatrix} \mathbf{e}'_{x[1]} & \mathbf{e}'_{x[2]} & \mathbf{e}'_{x[3]} \\ \mathbf{e}'_{y[1]} & \mathbf{e}'_{y[2]} & \mathbf{e}'_{y[3]} \\ \mathbf{e}'_{z[1]} & \mathbf{e}'_{z[2]} & \mathbf{e}'_{z[3]} \end{bmatrix} \quad (3.44)$$

Until now, the stiffness matrices only described the relation between forces and displacements between two elements. The final model however, will utilize just one stiffness matrix. This matrix describes the relation between all the forces acting on the structural system and all its degrees of freedom (displacements). As each element has six degrees of freedom (translation along three axes and rotation around three axes) the final system matrix will have a size of $(6 \cdot n_{el}) \times (6 \cdot n_{el})$, where n_{el} denotes the number of elements in the structural system.

The assembly process of the system stiffness matrix will be described with the help of an example. Assume that elements i and j are connected to each other according to the rules described in section 3.1.2. Each of these elements have six degrees of freedom. Element i has degrees of freedom $6 \cdot (i - 1) + 1$ to $6 \cdot (i - 1) + 6$. Assume now that a global stiffness matrix contains the stiffness parameters between i and j . As can be seen in eq. (3.23), the local (and global) stiffness matrix describes different influences two elements have on each other. Such influences can be described as follows:

- The first six rows by first six columns describe the relation between the displacements of element i and the forces acting on element i .
- The first six rows by last six columns describe the relation between the displacements of element j and the forces acting on element i .
- The last six rows by first six columns describe the relation between the displacements of element i and the forces acting on element j .
- The last six rows by last six columns describe the relation between the displacements of element j and the forces acting on element j .

Essentially this means that the stiffness matrix between two elements can be divided into four parts. Each part is a 6x6 matrix relating the displacements of an element to forces on itself or on an element it is connected to. This property is used in the assembly of the system stiffness matrix by adding the values of this small 6x6 matrix to it. One could determine to which row should be added by determining which force the stiffness parameter is related to. Assume that a stiffness parameter is related to a force with index 13, then this value would be added to the thirteenth row of the system stiffness matrix. The same logic should be followed to determine to which column a value should be added, except that the index of the related degree of freedom is regarded instead of the applied force. In this way the precise indices can be determined to add to in the system stiffness matrix. When this procedure is followed for every global element stiffness matrix, one ends up with the final system stiffness matrix that relates all external forces within the structural system to its degrees of freedom. This is the same procedure that is followed in finite element implementations, and one such implementation can be found in Nikishkov, 2004, chapter 5 and is illustrated below.

$$\begin{bmatrix}
 \ddots & & & & & \\
 \vdots & & & & & \\
 \dots & \mathbf{K}_{ii} & \dots & \mathbf{K}_{ij} & \dots & \\
 \vdots & & & & & \\
 \dots & (\mathbf{K}_{ij})^T & \dots & \mathbf{K}_{jj} & \dots & \\
 \vdots & & & & & \\
 \ddots & & & & &
 \end{bmatrix}
 \begin{Bmatrix}
 \vdots \\
 \vdots \\
 \mathbf{u}_i \\
 \vdots \\
 \mathbf{u}_j \\
 \vdots
 \end{Bmatrix}
 =
 \begin{Bmatrix}
 \vdots \\
 \vdots \\
 \mathbf{f}_i \\
 \vdots \\
 \mathbf{f}_j \\
 \vdots
 \end{Bmatrix}$$

3.2.1.2. Mass matrix

As a discrete element analysis is a time-dependent analysis, a mass matrix needs to be assembled in addition to a stiffness matrix. Hibbeler, 2004, chapter 13 states that the relation between a translational acceleration of a body and the forces acting on it are simply described as:

$$\mathbf{f}_i = M_i \mathbf{a}_i \quad (3.45)$$

where:

\mathbf{f}_i : a vector containing the force components acting on element i

\mathbf{a}_i : a vector containing the acceleration components of element i

M_i : mass of element i (kg)

Which is better known as Newton's second law. The same also applies for rotations, but more attention should be given to the term that describes rotational inertia. Hibbeler, 2004, chapter 17 describes the relation between angular acceleration and moments as:

$$\mathbf{m}_i = I_i \boldsymbol{\alpha}_i \quad (3.46)$$

where:

\mathbf{m}_i : a vector containing the moment components acting on element i

$\boldsymbol{\alpha}_i$: a vector containing the angular acceleration components of element i

I_i : inertia tensor of element i

I_i is defined as:

$$\mathbf{I}_i = \begin{bmatrix} I_{xx,i} & -I_{xy,i} & -I_{xz,i} \\ -I_{yx,i} & I_{yy,i} & -I_{zz,i} \\ -I_{zx,i} & -I_{zy,i} & I_{zz,i} \end{bmatrix} \quad (3.47)$$

When the point at which the inertia tensor (3.47) should be calculated coincides with the center of mass of a body b the terms described in the inertia tensor are defined as follows:

$$\begin{aligned} I_{xx,b} &= \int_{M_b} r_x^2 dM_b = \int_{M_b} (y^2 + z^2) dM_b \\ I_{yy,b} &= \int_{M_b} r_y^2 dM_b = \int_{M_b} (x^2 + z^2) dM_b \\ I_{zz,b} &= \int_{M_b} r_z^2 dM_b = \int_{M_b} (x^2 + y^2) dM_b \\ I_{xy,b} &= I_{yx,b} = \int_{M_b} xy dM_b \\ I_{yz,b} &= I_{zy,b} = \int_{M_b} yz dM_b \\ I_{xz,b} &= I_{zx,b} = \int_{M_b} xz dM_b \end{aligned} \quad (3.48)$$

It will not always be the case however, that the the point at which the inertia tensor should be calculated coincides with the center of mass of a body. This is the case when an element consists of smaller elementary shapes. To calculate the inertia tensor for the element, first the inertia tensors of the elementary shapes should be calculated *at the center of mass of the larger element* and summed

up afterwards. This is why one should make use of the parallel axis and parallel plane theorems to calculate these values around an arbitrary point. Applying these theorems yields the following relations:

$$\begin{aligned}
 I_{xx,b,p} &= I_{xx,b} + M_b(y_b^2 + z_b^2) \\
 I_{yy,b,p} &= I_{yy,b} + M_b(x_b^2 + z_b^2) \\
 I_{zz,b,p} &= I_{zz,b} + M_b(x_b^2 + y_b^2) \\
 I_{xy,b,p} &= I_{xy,b,p} = I_{xy,b} + M_b y_b x_b \\
 I_{yz,b,p} &= I_{zy,b,p} = I_{yz,b} + M_b y_b z_b \\
 I_{xz,b,p} &= I_{zx,b,p} = I_{xz,b} + M_b x_b z_b
 \end{aligned} \tag{3.49}$$

The computation of the values of the inertia tensor can be significantly simplified. Instead of letting the algorithm solve integrals over 3-dimensional geometries, it is more efficient to implement the integrated expression, which would be dependent on some set of variables. The 3D integral for an arbitrary cube with side length s and uniform mass density ρ , can be analytically expressed by first expanding the definition of infinitesimal mass dM :

$$dM = \rho dV = \rho dx dy dz \tag{3.50}$$

Inserting eq. (3.50) into eq. (3.48) allows one to solve the 3D integral:

$$\begin{aligned}
 I_{xx,cube} &= \int_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} \int_{y=-\frac{1}{2}s}^{y=\frac{1}{2}s} \int_{x=-\frac{1}{2}s}^{x=\frac{1}{2}s} (y^2 + z^2) \rho dx dy dz \\
 &= \left| \int_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} \int_{y=-\frac{1}{2}s}^{y=\frac{1}{2}s} x(y^2 + z^2) \rho dy dz \right|_{x=-\frac{1}{2}s}^{x=\frac{1}{2}s} = \int_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} \int_{y=-\frac{1}{2}s}^{y=\frac{1}{2}s} s(y^2 + z^2) \rho dy dz \\
 &= \left| \int_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} s \left(\frac{1}{3} y^3 + yz^2 \right) \rho dz \right|_{y=-\frac{1}{2}s}^{y=\frac{1}{2}s} = \int_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} \left(\frac{1}{12} s^4 + s^2 z^2 \right) \rho dz \\
 &= \left| \left(\frac{1}{12} s^4 z + \frac{1}{3} s^2 z^3 \right) \rho \right|_{z=-\frac{1}{2}s}^{z=\frac{1}{2}s} = \left(\frac{1}{12} s^5 + \frac{1}{12} s^5 \right) \rho \\
 &= \frac{1}{6} s^5 \rho
 \end{aligned} \tag{3.51}$$

The mass of a cube with uniform mass density can be expressed as:

$$M_{cube} = \rho s^3 \tag{3.52}$$

Inserting eq. (3.52) into eq. (3.51) yields:

$$I_{xx,cube} = \frac{1}{6} M_{cube} s^2 \tag{3.53}$$

Note that such a cube has all its faces perpendicular to coordinate axes and the origin of the coordinate system at its centroid. This cube is identical in all orthogonal directions and as such, the following could be stated (Hibbeler, 2004, chapter 21):

$$\mathbf{I}_{cube} = \begin{bmatrix} I_{cube} & 0 & 0 \\ 0 & I_{cube} & 0 \\ 0 & 0 & I_{cube} \end{bmatrix} \tag{3.54}$$

where $I_{cube} = I_{xx,cube} = I_{yy,cube} = I_{zz,cube}$.

The process of computing the inertia tensor for an arbitrary tetrahedron is more lengthy. When the mass of a tetrahedron is known, alongside the positions of its vertices, one could compute the inertia tensor around an arbitrary point in space p as follows (Tonon, 2004):

$$\mathbf{I}_{tetrahedron,p} = \begin{bmatrix} a & -b' & -c' \\ -b' & b & -a' \\ -c' & -a' & c \end{bmatrix} \quad (3.55)$$

$$\begin{aligned} a &= \frac{M_{tetrahedron}}{60} \sum_{i=1}^4 \sum_{j=1}^4 (\mathbf{y}_{[i]} \mathbf{y}_{[j]} + \mathbf{z}_{[i]} \mathbf{z}_{[j]}) \\ b &= \frac{M_{tetrahedron}}{60} \sum_{i=1}^4 \sum_{j=1}^4 (\mathbf{x}_{[i]} \mathbf{x}_{[j]} + \mathbf{z}_{[i]} \mathbf{z}_{[j]}) \\ c &= \frac{M_{tetrahedron}}{60} \sum_{i=1}^4 \sum_{j=1}^4 (\mathbf{x}_{[i]} \mathbf{x}_{[j]} + \mathbf{y}_{[i]} \mathbf{y}_{[j]}) \\ a' &= \frac{M_{tetrahedron}}{120} \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{y}_{[i]} \mathbf{z}_{[j]} + \sum_{i=1}^4 \mathbf{y}_{[i]} \mathbf{z}_{[i]} \right) \\ b' &= \frac{M_{tetrahedron}}{120} \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{x}_{[i]} \mathbf{z}_{[j]} + \sum_{i=1}^4 \mathbf{x}_{[i]} \mathbf{z}_{[i]} \right) \\ c' &= \frac{M_{tetrahedron}}{120} \left(\sum_{i=1}^4 \sum_{j=1}^4 \mathbf{x}_{[i]} \mathbf{y}_{[j]} + \sum_{i=1}^4 \mathbf{x}_{[i]} \mathbf{y}_{[i]} \right) \end{aligned} \quad (3.56)$$

As stated earlier, these expressions can be used to find the inertia tensor for all the elementary shapes of an element at the element's center of mass. After adding these values up, it would yield the inertia tensor of the element at its center of mass. Using eqs (3.45), (3.46) and (3.47), one could write the mass matrix for element i as:

$$\underbrace{\begin{Bmatrix} \mathbf{f}_i \\ \mathbf{m}_i \end{Bmatrix}}_{\text{element force vector } \mathbf{f}_{ele,i}} = \underbrace{\begin{bmatrix} M_i & 0 & 0 & 0 & 0 & 0 \\ 0 & M_i & 0 & 0 & 0 & 0 \\ 0 & 0 & M_i & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx,i} & -I_{xy,i} & -I_{xz,i} \\ 0 & 0 & 0 & -I_{yx,i} & I_{yy,i} & -I_{zz,i} \\ 0 & 0 & 0 & -I_{zx,i} & -I_{zy,i} & I_{zz,i} \end{bmatrix}}_{\text{element mass matrix } M_i} \underbrace{\begin{Bmatrix} \mathbf{a}_i \\ \boldsymbol{\alpha}_i \end{Bmatrix}}_{\text{element acceleration vector } \dot{\mathbf{u}}_{ele,i}} \quad (3.57)$$

An approach one could take to compute the mass matrix is to calculate the mass matrix for a unit element ($s = 1$ m) and only use the spacing value s to calculate the correct mass matrices. Doing this, the following matrices are obtained ($\rho = 2000 \frac{\text{kg}}{\text{m}^3}$):

$$M_{CS,i} = s^3 \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05208333s^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05208333s^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05208333s^2 \end{bmatrix} \quad (3.58)$$

for CS elements, and

$$M_{RD,i} = s^3 \begin{bmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.00295139s^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.00295139s^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.00295139s^2 \end{bmatrix} \quad (3.59)$$

for RD elements.

This matrix can be used to assemble a system mass matrix in the same way the system stiffness matrix was assembled. In this case, it is less complicated, as a mass matrix is just related to one element. The system mass matrix is assembled as follows:

$$\hat{M} = \begin{bmatrix} M_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & M_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & M_n \end{bmatrix} \quad (3.60)$$

3.2.1.3. Damping

For damping, use is made of *adaptive global damping* as described by Cundall, 1982. This type of damping distributes the damping across all elements and is dependent on the change of kinetic energy of the system. It can be calculated as follows:

$$C = \beta \frac{\dot{\mathcal{E}}_k}{6n_{el}} \quad (3.61)$$

where:

- $\dot{\mathcal{E}}_k$: change of kinetic energy of the system
- β : parameter to scale the global damping
- n_{el} : the number of elements in the structure

One setback of adaptive global damping is that it distributes viscous damping equally to all elements, while it may be possible that some elements are more strongly damped than others.

According to Jain, 2010, chapter 4, the kinetic energy of a multi-body system can be computed as follows:

$$\mathcal{E}_k^{(t)} = \frac{1}{2} \left(\hat{\mathbf{u}}^{(t)} \right)^T \hat{\mathbf{M}}^{(t)} \hat{\mathbf{u}}^{(t)} \quad (3.62)$$

The rate of change of the kinetic energy of a system is calculated by means of the following expression:

$$\dot{\mathcal{E}}_k^{(t)} \begin{cases} 0, & \text{if } t = 0 \\ \frac{\mathcal{E}_k^{(t)} - \mathcal{E}_k^{(t-\Delta t)}}{\Delta t}, & \text{otherwise} \end{cases} \quad (3.63)$$

3.2.1.4. Solving the equation of motion

Now that all the components of the system have been defined, it is time to use the equation of motion as defined in eq. (3.12) and solve for the accelerations. To achieve this, the following assumptions are made:

$$\begin{aligned} \hat{\mathbf{u}}_{[i]}^{(0)} &= 0 \\ \hat{\mathbf{a}}_{[i]}^{(0)} &= 0 \end{aligned} \quad (3.64)$$

The above expressions hold true for all t for degrees of freedom that are constrained. By making use of eq. (3.64) the only unknown is the vector containing the acceleration values for each degree of freedom. With this in mind, eq. (3.12) can be rewritten to solve for $\hat{\mathbf{u}}$:

$$\hat{\mathbf{u}}^{(t)} = \hat{\mathbf{M}}^{(t)-1} \left(\hat{\mathbf{f}} - \mathbf{C}^{(t)} \hat{\mathbf{u}}^{(t)} - \hat{\mathbf{K}}^{(t)} \hat{\mathbf{u}}^{(t)} \right) \quad (3.65)$$

For velocities half time intervals are computed, to make use of a midpoint integration scheme (Sluys & de Borst, 2001, chapter 4):

$$\begin{aligned} \hat{\mathbf{u}}_{[i]}^{(\frac{1}{2}\Delta t)} &= \hat{\mathbf{u}}_{[i]}^{(0)} + \frac{1}{2}\Delta t \hat{\mathbf{a}}_{[i]}^{(0)} \\ \hat{\mathbf{u}}_{[i]}^{(\Delta t)} &= \hat{\mathbf{u}}_{[i]}^{(0)} + \Delta t \hat{\mathbf{a}}_{[i]}^{(\frac{1}{2}\Delta t)} \end{aligned} \quad (3.66)$$

and for subsequent time-steps:

$$\begin{aligned} \hat{\mathbf{u}}_{[i]}^{(t+\frac{1}{2}\Delta t)} &= \hat{\mathbf{u}}_{[i]}^{(t-\frac{1}{2}\Delta t)} + \Delta t \hat{\mathbf{a}}_{[i]}^{(t)} \\ \hat{\mathbf{a}}_{[i]}^{(t)} &= \hat{\mathbf{u}}_{[i]}^{(t+\frac{1}{2}\Delta t)} - \hat{\mathbf{u}}_{[i]}^{(t-\frac{1}{2}\Delta t)} \\ \hat{\mathbf{u}}_{[i]}^{(t+\Delta t)} &= \hat{\mathbf{u}}_{[i]}^{(t)} + \Delta t \hat{\mathbf{a}}_{[i]}^{(t+\frac{1}{2}\Delta t)} \end{aligned} \quad (3.67)$$

The only question now is what time-step value Δt should be used. According to Jing and Stephansson, 2007, chapter 11, the following time-step value is needed to prevent instabilities:

$$\Delta t = 0.8 \sqrt{\frac{\text{diag} \left(\hat{\mathbf{M}}^{(t)} \right)_{min}}{\text{diag} \left(\hat{\mathbf{K}}^{(t)} \right)_{max}}} \quad (3.68)$$

To shorten analyses, a lower Young's Modulus can be chosen. Such an approach is also taken by Dell'Endice et al., 2021. According to Lommen et al., 2014, for bulk compression a lower Young's Modulus may be taken unless the stiffness components becomes so small that elements are not able to counteract the forces applied to them, resulting in elements falling through each other.

It is now possible to iteratively compute the displacement and velocity values for subsequent time-steps. One should however define a certain state where equilibrium is reached (or more precisely, where an equilibrium-state is approximated closely enough). To achieve this, a criterion should be set up in order to stop the analysis after an equilibrium state has been reached. This can be done by checking the out-of-balance forces. The analysis is terminated when the out-of-balance forces are

below a certain threshold. Out-of-balance forces for a static-equilibrium can be computed as follows (Sluys & de Borst, 2001, chapter 4):

$$\Delta \hat{\mathbf{f}}^{(t)} = \hat{\mathbf{f}} - \hat{\mathbf{K}}^{(t)} \hat{\mathbf{u}}^{(t)} = \hat{\mathbf{f}} - \hat{\mathbf{f}}_{int}^{(t)} \quad (3.69)$$

Some further additions should be made however, in order to ensure that convergence will be reached. It is possible for elements to not have any interfaces due to the limits from eq. (3.20). Certain degrees of freedom thus may not have any stiffness components at all related to them. This could result in large out-of-balance forces without any possibility to counteract them. The same holds for applied displacements with constrained degrees of freedom. As these displacements will always be zero, the force related to those degrees of freedom will not be balanced by stiffness components. It could also be possible that a number of elements depart from the system together. In this case there will be non-zero stiffness components for degrees of freedom related to any interface they may share. It cannot be expected that any equilibrium will be reached if none of the drifting elements are supported. Because of this, also a limit will be imposed on the magnitude of velocities. Eq. (3.69) will thus be rewritten as:

$$\Delta \hat{\mathbf{f}}_{[i]}^{(t)} = \begin{cases} 0, & \text{if } \hat{\mathbf{f}}_{int[i]}^{(t)} = 0 \vee \hat{\mathbf{u}}_{[i]}^{(t)} > \dot{u}_{max} \\ (\hat{\mathbf{f}} - \hat{\mathbf{f}}_{int})_{[i]}^{(t)}, & \text{otherwise} \end{cases} \quad (3.70)$$

Convergence is reached when:

$$\|\Delta \hat{\mathbf{f}}^{(t)}\|_2 < 10^{-2} \|\Delta \hat{\mathbf{f}}^{(\Delta t)}\|_2 \quad (3.71)$$

This value for the convergence criterion is chosen as it proved to be loose enough to let optimizations not take a long time (within an hour for structures consisting of more than 100 elements).

3.2.2. Time-explicit analysis using local damping

Another way to calculate displacements and velocities for each time-step is to use local damping. This type of damping avoids assigning a global damping value that is equal for all elements, ensuring that energy is always dissipated (Cundall, 1987). In this method the equation of motion (3.12) is not used. Instead the following expressions are used to calculate velocity values for the next time-step (note that accelerations are not calculated using this method):

$$\begin{aligned} \dot{\mathbf{u}}^{(t+\frac{1}{2}\Delta t)} &= \dot{\mathbf{u}}^{(t-\frac{1}{2}\Delta t)} + \Delta t \hat{\mathbf{M}}^{-1} \left(\hat{\mathbf{f}}_{int}^{(t)} - \hat{\mathbf{f}}_d^{(t)} \right) \\ \hat{\mathbf{f}}_d^{(t)} &= \alpha |\vec{\hat{\mathbf{f}}}_{int}^{(t)}| \odot \text{sgn} \left(\dot{\mathbf{u}}^{(t-\frac{1}{2}\Delta t)} \right) \end{aligned} \quad (3.72)$$

where:

$|\vec{\hat{\mathbf{f}}}_{int}^{(t)}|$: a vector containing the absolute values of the internal force vector

α : parameter to scale the local damping

\odot : Hadamard product (element-wise multiplication)

$$\text{sgn} \left(\dot{\mathbf{u}}_{[i]}^{(t-\frac{1}{2}\Delta t)} \right) = \begin{cases} 1, & \text{if } \dot{\mathbf{u}}_{[i]}^{(t-\frac{1}{2}\Delta t)} > 0 \\ -1, & \text{if } \dot{\mathbf{u}}_{[i]}^{(t-\frac{1}{2}\Delta t)} < 0 \\ 0 & \text{otherwise} \end{cases}$$

When the velocity at a half time-step is known, the displacements for the next time-step can be calculated if the velocity is assumed be constant within a time-step:

$$\mathbf{u}^{(t+\Delta t)} = \mathbf{u}^{(t)} + \Delta t \dot{\mathbf{u}}^{(t+\frac{1}{2}\Delta t)} \quad (3.73)$$

For the first time-step the following boundary conditions are taken:

$$\begin{aligned} \mathbf{u}^{(0)} &= 0 \\ \dot{\mathbf{u}}^{(-\frac{1}{2}\Delta t)} &= 0 \end{aligned} \quad (3.74)$$

Together with the definitions in section 3.2.1, the above equations completely describe the procedure followed during a time-explicit analysis using local damping.

3.2.3. Static analysis

To allow for faster analyses where a convergence criterion does not have to be satisfied, a static analysis is also implemented. In such an analysis the equation of motion (3.12) is simplified by neglecting all the terms that are time-dependent. The following relation is thus obtained:

$$\hat{\mathbf{K}}\hat{\mathbf{u}} = \hat{\mathbf{f}} \quad (3.75)$$

Choosing this analysis procedure has significant implications for the material-behaviour. In section 3.2.1.1 a non-linear constitutive relation was proposed between the elongation of the interface and the normal force (or normal stress multiplied by the area the interface represents) it exerts. As the goal of this simplified analysis is to have a non-iterative procedure, such non-linearities cannot be considered in this model.

The solution procedure is quite simple: The system stiffness matrix is assembled as outlined in section 3.2.1.1 and the external force vector containing the self-weight of the elements and user-defined applied forces is defined. Applied constraints are considered by removing corresponding rows and columns from the system stiffness matrix, and corresponding rows from the displacement and external force vector. To calculate the displacements of the free degree of freedoms the following needs to be solved:

$$\hat{\mathbf{u}}_f = \hat{\mathbf{K}}_f^{-1} \hat{\mathbf{f}}_f \quad (3.76)$$

It is best to store the system stiffness matrix as a sparse matrix as most of its entries are zero (figure 3.19). This not only saves memory, but can also speed up solving for the displacements in eq. (3.76). Saving a sparse matrix and solving the subsequent system is done with help of the SciPy library for Python.

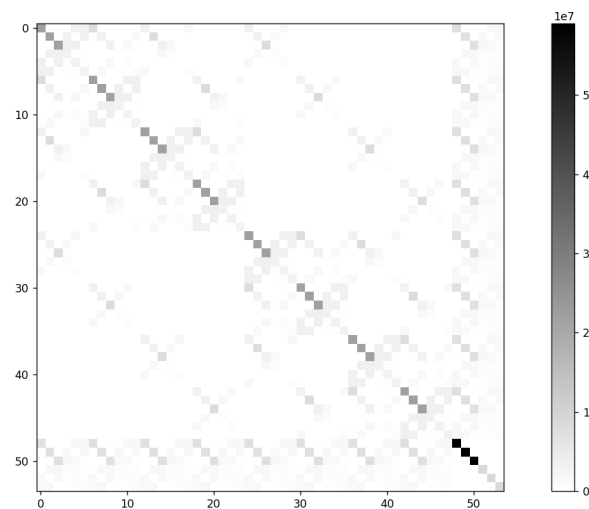


Figure 3.19: the absolute values of the system stiffness matrix for a 2x2x2 CS structure

3.2.4. Results

In this section some results will be shown of the discrete elements analyses and the behaviour of the models will be evaluated.

3.2.4.1. Time-explicit analyses

Convergence analysis

To evaluate the convergence behaviour of the dynamic models using global damping and local damping, analyses were carried out using both models and the results were visualized. Figures 3.20 and 3.21 show the results of analyses at different time-steps for analyses using global damping. Table 3.2 gives an overview of the values used. For the time-explicit analyses the only applied force is the self-weight of the elements. The lower corners of the structure are completely fixed (no translations nor rotations are possible). The displacement vector norm is computed by calculating the Euclidean norm of the translations for each element. Note that the magnitude of rotations is not visualized.

It is important to note the extremely large displacements at figures 3.20c and 3.21c. These displacements happen suddenly and suggest the model has an instability issue. It should also be noted that these analyses did not convergence and were instead terminated because all non-supported elements had velocities exceeding the maximum velocity value \dot{u}_{max} . The values of \dot{u}_{max} were taken to be very large to record the unstable behaviour of the model, as it would be terminate before instability arises otherwise.

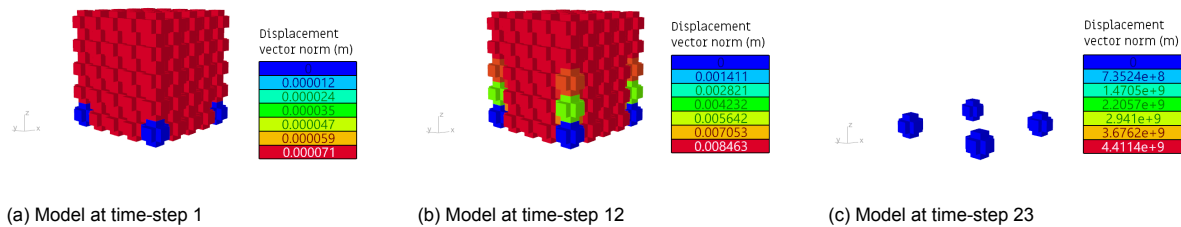


Figure 3.20: a CS-based structure of size 5x5x5 at different time-steps using global damping

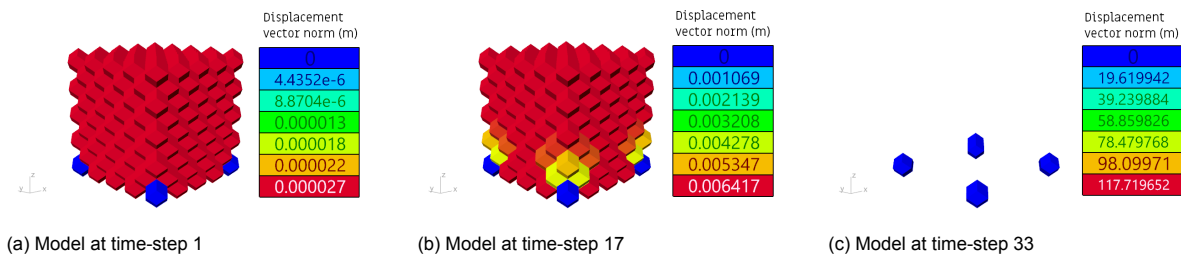


Figure 3.21: a RD-based structure of size 5x5x5 at different time-steps using global damping

shape	E (Pa)	ν	β	\dot{u}_{max} ($\frac{m}{s}$)
CS	$1000 \cdot 10^6$	0.25	100	100
RD	$1000 \cdot 10^6$	0.25	100	100

Table 3.2: table showing the values used during the time-explicit analyses with global damping

The same analyses are also carried out using the locally damped time-explicit model. The results are shown in figures 3.22 and 3.23. The used values are listed in table 3.3.

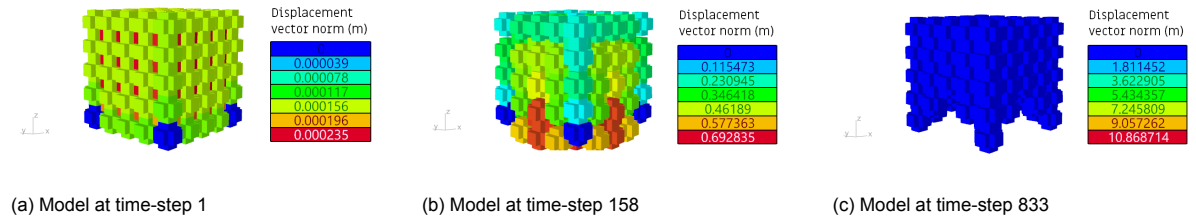


Figure 3.22: a CS-based structure of size 5x5x5 at different time-steps using local damping

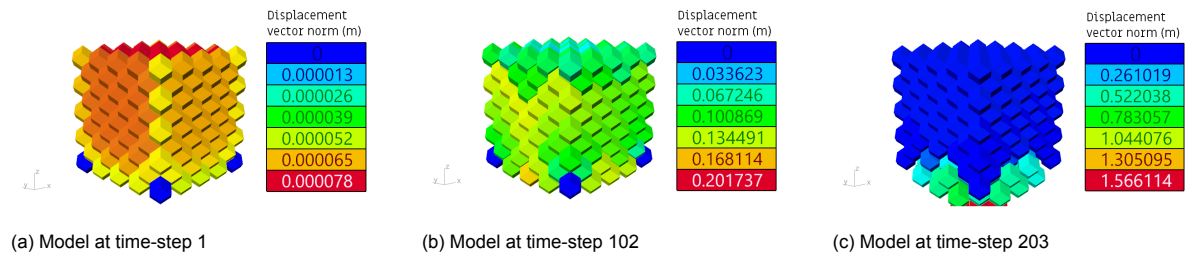


Figure 3.23: a RD-based structure of size 5x5x5 at different time-steps using local damping

shape	E (Pa)	ν	α	\dot{u}_{max} ($\frac{m}{s}$)
CS	$1000 \cdot 10^6$	0.25	0.8	1
RD	$1000 \cdot 10^6$	0.25	0.8	0.5

Table 3.3: table showing the values used during the time-explicit analyses with local damping

It is clear that the time-explicit analyses using local damping are more robust than those using global damping. This can be seen by the fact that the analyses using local damping have completed more time-steps, do not show extremely large displacements and have been terminated by reaching convergence. The convergence behaviour for both the CS-based and RD-based structures is shown in figure 3.24.

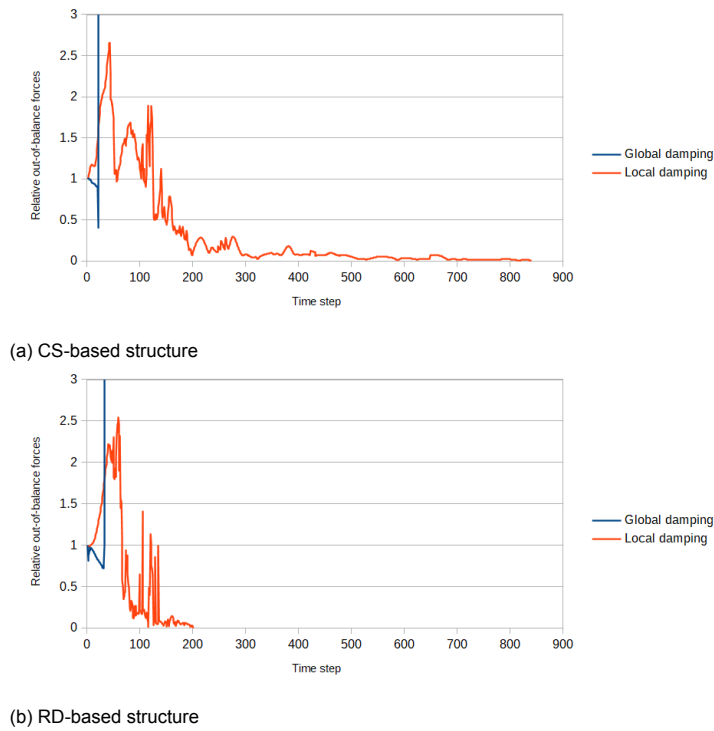


Figure 3.24: the convergence behaviour of analyses for CS-based and RD-based structures using global or local damping

Sliding with externally applied moments

To test whether user-defined applied forces and the non-linear interface behaviour were working correctly, a moment and a horizontal force were applied on an element on top of a fully constrained element. The applied force is in positive y-direction, while the moment is applied in positive z-direction (figure 3.25). Use is made of local damping to ensure stability. Values used in this model are listed in table 3.4. Figure 3.26 shows that the observed displacements are in the expected directions. Figure 3.27 clearly shows how the interfaces stop being considered in the analysis as their lengths exceed their elongation limit. It can be seen that the top element first slowly slides, and after the interfaces elongates outside the bounds specified in eq. (3.20), it is observed that the sliding and rotation speed up considerably.

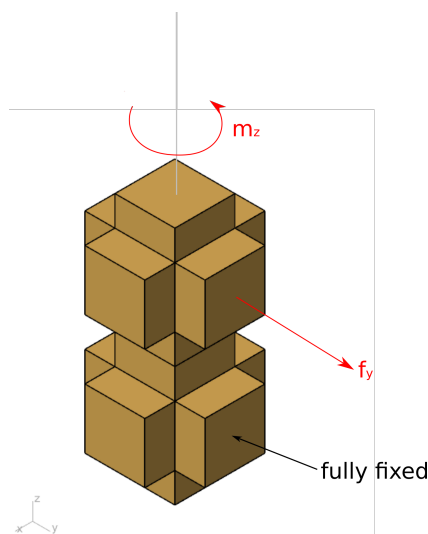


Figure 3.25: sign convention and setup of the sliding model

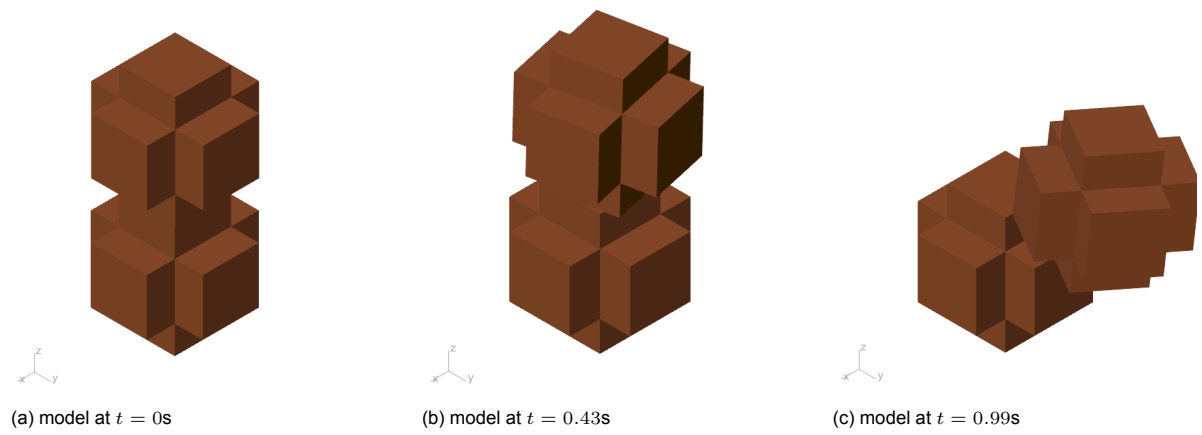


Figure 3.26: images from the sliding model to check whether applied forces and interfaces are working

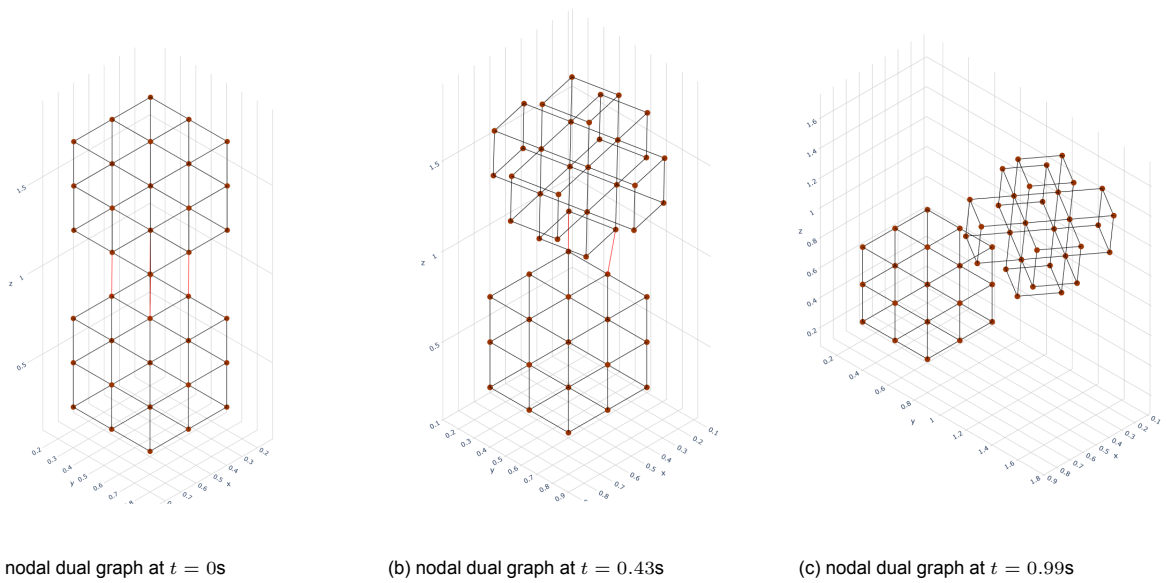


Figure 3.27: nodal dual graphs of the sliding model

E (Pa)	ν	α	f_y (N)	m_z (Nm)
$1000 \cdot 10^6$	0.25	0.8	$30 \cdot 10^3$	10^5

Table 3.4: table showing the values used for the sliding model

3.2.4.2. Static analysis

On the same 5x5x5 structures shown in the previous section, static analyses were performed as well. These analyses only consider the self-weight of the elements. The results of these analyses are shown in figures 3.28 and 3.29. For both analyses a Young's Modulus of $1000 \cdot 10^6$ Pa and Poisson's ratio of 0.25 was taken. These analyses both took no more than 10 seconds to complete, while it took the time-explicit analysis with local damping around 10 minutes to reach convergence. This clearly shows the advantage of the static analysis.

Comparing the results of figures 3.28 and 3.29 with figures 3.22c and 3.23c however, shows a disadvantage of using this analysis procedure. As no non-linear material behaviour is modelled with a static analysis, there is no way to model unsupported elements falling out of the structure. In other words, the static analysis allows for significant tensile forces while this should not be the case between masonry elements.

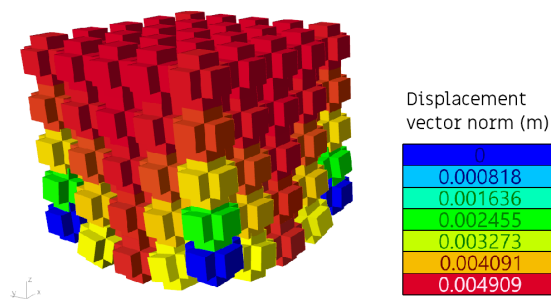


Figure 3.28: result of the static analysis on a 5x5x5 CS-based structure (for visualization the displacements are magnified by a factor 100)

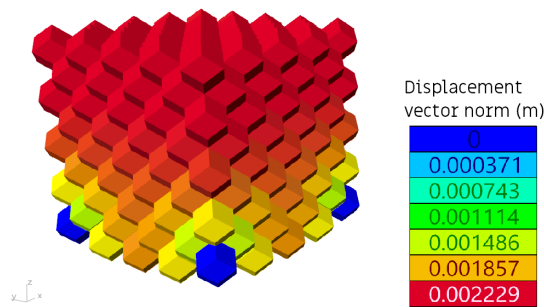


Figure 3.29: result of the static analysis on a 5x5x5 RD-based structure (for visualization the displacements are magnified by a factor 100)

The results of structural analyses of a large 20x5x5 structure are shown in figures 3.30 and 3.31. Again, this highlights the strength of the static solution method, as both analyses took less than a minute, while a structure of this size may not reach convergence within a reasonable amount of time using a time-explicit method. This is especially the case if one considers that ultimately a multitude of analyses should be carried out for a structural topology optimization

Comparing figures 3.30 and 3.31 with each other, an interesting difference between CS-based and RD-based structures becomes apparent. For the CS-based structure the displacement differential along the top of the structure is more significant than for the RD-based structure. This suggests that displacements spread out more to elements on the same layer in RD-based structures than it does in CS-based structures.

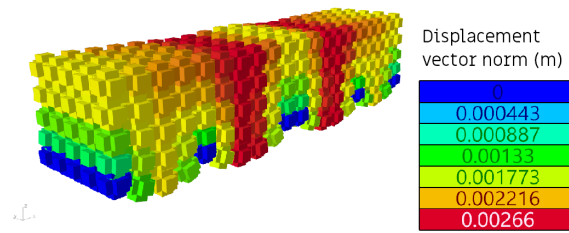


Figure 3.30: result of the static analysis on a 20x5x5 CS-based structure (for visualization the displacements are magnified by a factor 500)

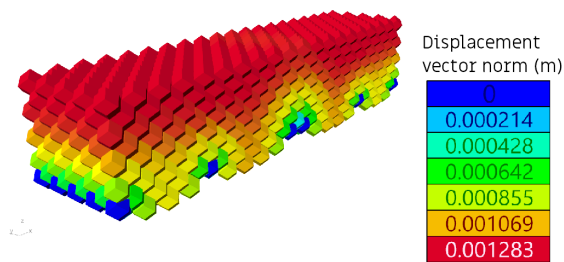


Figure 3.31: result of the static analysis on a 20x5x5 RD-based structure (for visualization the displacements are magnified by a factor 300)

3.3. Optimization

Now that a discrete element model is formulated and implemented, it is time to outline the details of the optimization algorithm. The optimizer used is an algorithm called the Method of Moving Asymptotes (MMA) (Svanberg, 1987). The Python implementation of this algorithm is developed by Deetman, 2021.

3.3.1. Objective function

The objective function forms the core of the optimization algorithm. It is the function that is attempted to be minimized by the optimizer. The procedure taken by O'Shaughnessy et al., 2022 will be mostly followed. The objective function is given as follows:

$$\min(c(\chi)) = \mu_0 \cdot (\hat{\mathbf{u}}^T \hat{\mathbf{f}} - \hat{\mathbf{u}}^T \hat{\mathbf{K}} \hat{\mathbf{u}}) \quad (3.77)$$

where:

χ : vector containing design variables for the topology optimization, $0.001 \leq \chi_{[i]} \leq 1$

μ_0 : parameter to scale the objective function value

In essence, eq. (3.77) is nothing else than a work-balance equation for work done by external and internal forces. It should be noted however, that not a balance of work is attempted to be obtained, but rather a maximization of strain energy while the external work is minimized. In the case of the time-explicit models, elements that drift away or fall from the analysed structure will have a relatively large external work value compared to their strain energy. It is thus likely that these elements with significant displacements will be removed from the model.

Before optimization, the dependence of eq. (3.77) on the design variables χ should be defined in more detail. In topology optimization, applied forces are not dependent on any design variables. If the contrary were the case, the topology of a structure would simply be optimized by removing elements with applied forces. There is however, a dependency of the body forces (or self-weight in this case) on the design variables (Bruyneel & Duysinx, 2001). The system force vector can be written as:

$$\hat{\mathbf{f}} = \hat{\mathbf{f}}_{applied} + \hat{\mathbf{f}}_{body} \odot \chi \quad (3.78)$$

For the case that self-weight is considered, the body forces can simply be computed as follows:

$$\hat{\mathbf{f}}_{body[i]} = \nabla \hat{\mathbf{f}}_{[i]} = -M_i g \quad (3.79)$$

where:

g : acceleration due to gravity, $g = 9.81 \frac{\text{m}}{\text{s}^2}$

Eq. (3.79) shows that the body forces equal the gradient of the applied forces on the system (dependent on the design variables). Note that the self-weight is added only to the degrees of freedom that describe translation in the global z-direction.

To illustrate the dependency of the system strain energy on χ , use is made of the local stiffness matrix as defined in eq. (3.23). It can be observed that this stiffness matrix defines the behaviour of spring-like interfaces between two elements. It is for that reason that the following definition is used, similarly to O'Shaughnessy et al., 2022:

$$\mathcal{K}'_{ij} = \chi_{[i]}^p \chi_{[j]}^p \mathbf{K}'_{ij} \quad (3.80)$$

This expression makes use of the Solid Isotropic Materials with Penalization (SIMP) method (Rozvany, 2000). The parameter p is the penalization parameter used to penalize intermediate values of χ and possibly speed up convergence of the optimizer.

Note that it is in fact \mathcal{K}'_{ij} that will be transformed by a transformation matrix and assembled in a system stiffness matrix instead of \mathbf{K}'_{ij} , during the optimization.

Apart from defining the objective function, for the MMA algorithm it is also necessary to calculate the gradient of the objective function. This gradient can be expressed as follows:

$$\nabla c(\boldsymbol{\chi}) = \mu_0 \cdot \left(\hat{\mathbf{u}}^T \hat{\mathbf{f}}_{body} - \hat{\mathbf{u}}^T \nabla \hat{\mathbf{K}} \hat{\mathbf{u}} \right) \quad (3.81)$$

$$\frac{\partial \mathcal{K}'_{ij}}{\partial \chi_{[i]}} = p \chi_{[i]}^{p-1} \chi_{[j]}^p \mathbf{K}'_{ij} \quad (3.82)$$

When all i components of eq. (3.82) are computed, they should be assembled in a system stiffness matrix as usual to compute $\hat{\mathbf{u}}^T \nabla \hat{\mathbf{K}} \hat{\mathbf{u}}$. This means that for every iteration during the optimization, the system stiffness matrix should be assembled twice; once to obtain the system stiffness matrix to run the structural analysis and once to calculate the gradient of the internal work dependent on the design variables.

3.3.2. Constraints

Eq. (3.77) cannot be used on its own to optimize a structural topology. One should define constraints to be sure that the optimizer does not return a solution that does not satisfy practical needs for instance. For the optimization of discrete element structures, two constraints are defined, which will be explained in detail in the following sections.

3.3.2.1. Volume

The most basic constraint is the constraint put on the volume of the total structure. This constraint can be written as follows (O'Shaughnessy et al., 2022):

$$\mu_1 \cdot \left(\frac{V(\boldsymbol{\chi})}{V_0} - f \right) = \mu_1 \cdot \left(\frac{\sum_{i=1}^{n_{el}} \chi_{[i]}}{n_{el}} - f \right) = 0 \quad (3.83)$$

where:

V_0 : the full 'volume' of the system if all design variables are set to 1, this is equal to the number of elements in the structure

f : the target mean value of $\boldsymbol{\chi}$, $0 < f < 1$

μ_1 : parameter to scale the volume constraint value

A certain mean value for the design variables relative to the full volume is thus defined. This constraint prevents the optimization algorithm from finding solutions with either too small or too large values for $\boldsymbol{\chi}$. The solver also needs the gradient of this constraint to satisfy it. This gradient is computed as:

$$\mu_1 \cdot \frac{\partial}{\partial \boldsymbol{\chi}} \left(\frac{V(\boldsymbol{\chi})}{V_0} - f \right) = \frac{\mu_1}{V_0} \mathbf{1} \quad (3.84)$$

3.3.2.2. Roof

Another constraint that proved to be necessary, is the constraint to have an element above each point in the system. This essentially means that every part of the structure needs to be covered by a roof. When this constraint is not in place, the solver tends to favour elements around the supports. This results in the absence of a spanning structure and the formation of columns (figure 3.32).

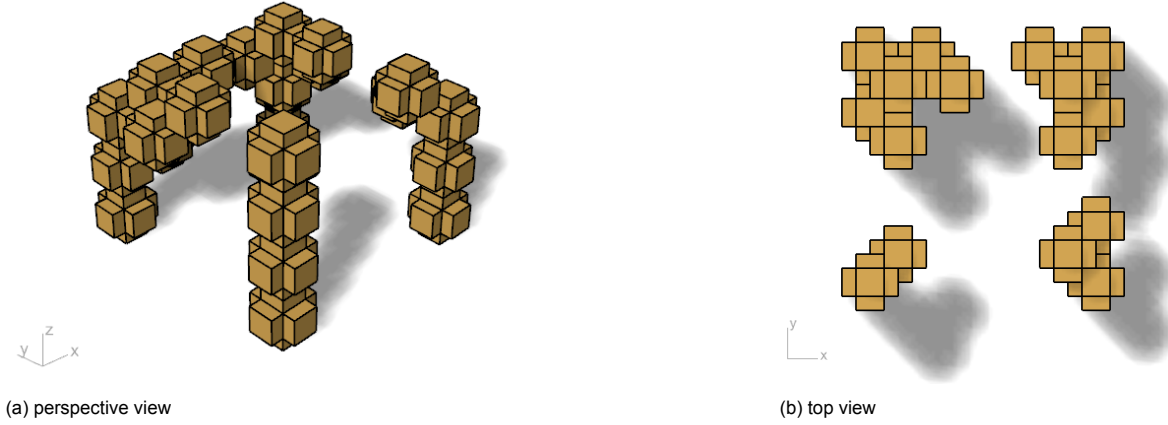


Figure 3.32: optimized lattice structure with corners supported, without roof constraint

To alleviate this problem, one could check whether there is an element present along the z-axis at each point. The following constraint is proposed:

$$g(\chi) \begin{cases} \mu_2 \cdot \left(\left(1 - \frac{1}{q+1}\right) - \chi_{[i]} + \frac{1}{q+1} \chi_{[i]}^{q+1} + \sum_{K \neq i} \chi_{[i]} \chi_{[K]}^q \right), & \text{if } i \in K \\ 0, & \text{otherwise} \end{cases} \quad (3.85)$$

$$\frac{\partial g(\chi)}{\partial \chi_{[i]}} \begin{cases} \min \left(0, \mu_2 \cdot \left(-1 + \sum_K \chi_{[K]}^q \right) \right), & \text{if } i \in K \\ 0, & \text{otherwise} \end{cases} \quad (3.86)$$

where:

q : parameter to penalize intermediate values within the roof constraint, $q = 3$ seemed to work well

μ_2 : parameter to scale the roof constraint value

These expressions can best be explained by using an example:

Suppose a column K with three elements, namely: i , j , and k . Plotting eqs (3.85) and (3.86) for different design values for these elements yields the graphs shown in figure 3.33 and figure 3.34. The solver attempts to find the value for $\chi_{[i]}$ that results in eq. (3.85) being zero (or as close as possible to zero). It achieves this by making use of derivative (3.86). The reason why the minimum value between 0 and the derivative of eq. (3.85) is computed, is because the constraint should not be to have exactly one element as a roof, but to have *at least one element* as a roof. It is not an issue if at one point there are multiple elements in a column. Therefore, the problem should not be overconstrained by trying to strictly minimize eq. (3.85) (which is done when exactly one element in a column has a design value of 1, and all the others 0).

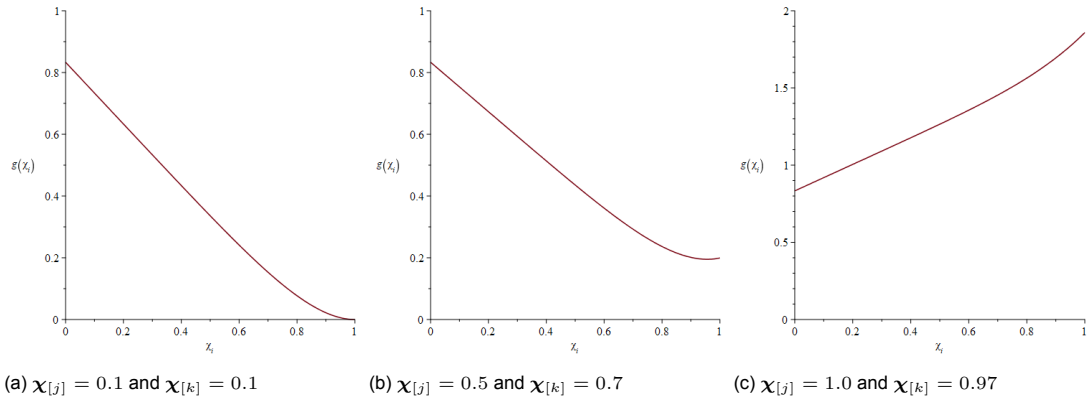


Figure 3.33: values for roof constraint (3.85) depending on $\chi_{[i]}$ for different values of $\chi_{[j]}$ and $\chi_{[k]}$ ($q = 5$)

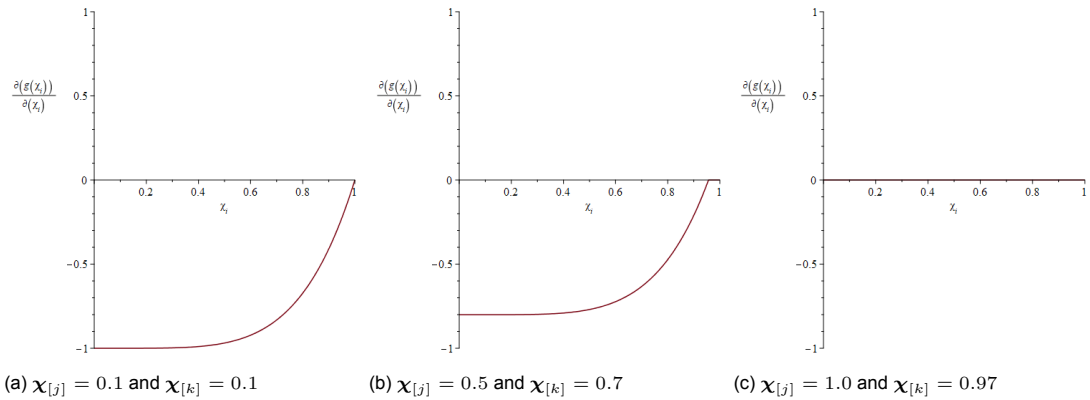


Figure 3.34: values for derivative (3.86) depending on $\chi_{[i]}$ for different values of $\chi_{[j]}$ and $\chi_{[k]}$ ($q = 5$)

3.3.3. Filtering

In topology optimization there exists a common phenomenon called the *checkerboard pattern* (Koga et al., 2013). This is the result of small elements in a mesh allowing the solver to create a porous structure, which is in essence a structurally sound solution, but not necessarily the way conventional structures are built. The same structure using a larger mesh size may not show the same result as the optimizer may not be able to create this porous structure without sacrificing the structural integrity (figure 3.35).

To solve the problem of mesh-dependence, filters are commonly used. Filters relate the design variables of neighbouring elements to each other so they are not only dependent on the behaviour of the model (Pedersen et al., 2006). Such a filter is also incorporated in the Python MMA implementation of Deetman, 2021. In this case, use is made of sensitivity filtering which relates the sensitivities of one element (so the gradients shown in the previous sections) to the sensitivities of elements within a certain radius r . This radius is a user defined value and may influence the output topology (figure 3.36)



Figure 3.35: topology optimization results for different mesh sizes without using any filtering (Talischi et al., 2008)

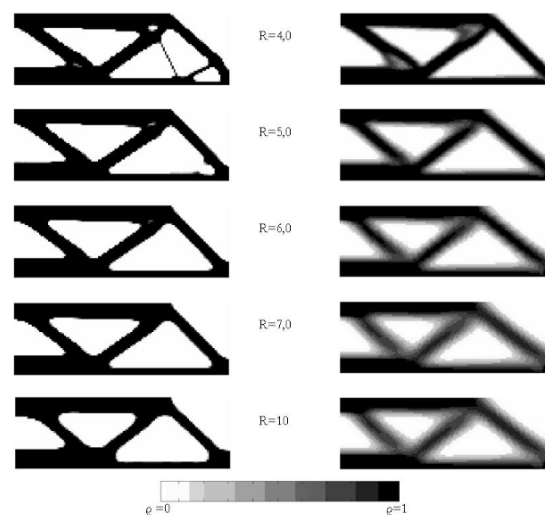


Figure 3.36: different output topologies for different values of the filter radius r using two different filters (Pedersen et al., 2006)

3.3.4. Results

Optimizations were carried out using time-explicit local damping and static analyses. The results of these analyses are shown in the following paragraphs.

3.3.4.1. Time-explicit optimization

Using the time-explicit analysis the following initial topologies were optimized:

- A 5x2x5 structure where two opposite sides are fully supported at the bottom (arch)
- A 5x5x4 structure (for CS) and 5x5x5 structure (for RD) where all bottom corners are fully supported (cross vault)

Here a penalization value of $p = 2$ is taken, as this value seemed to result in feasible optimizations within a reasonable time-frame. The structures that are optimized are chosen to be rather small in size, which is to limit the run-time. The values used for each analysis are listed in table 3.5 along with the run-time to acquire the results shown in figures 3.37 and 3.38.

shape	structure	E (Pa)	ν	r (m)	μ_0	μ_1	μ_2	f	run-time (min)
CS	arch	$5 \cdot 10^6$	0.25	0.8	1.0	1.0	10.0	0.35	15
RD	arch	$5 \cdot 10^6$	0.25	0.8	1.0	1.0	10.0	0.35	13
CS	cross-vault	$5 \cdot 10^6$	0.25	0.8	1.0	1.0	10.0	0.35	31
RD	cross-vault	$5 \cdot 10^6$	0.25	0.8	1.0	1.0	10.0	0.35	57

Table 3.5: table showing the values used during the time-explicit optimizations

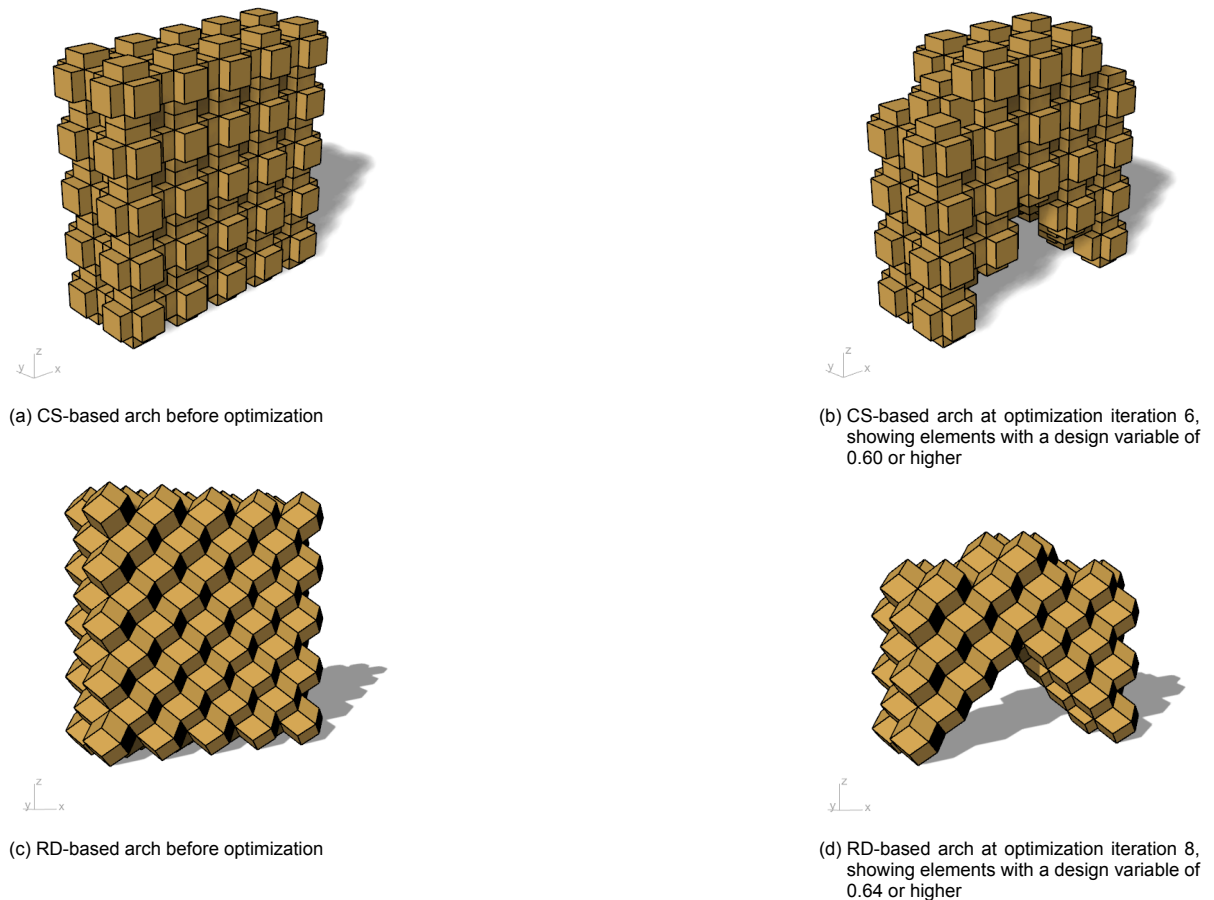
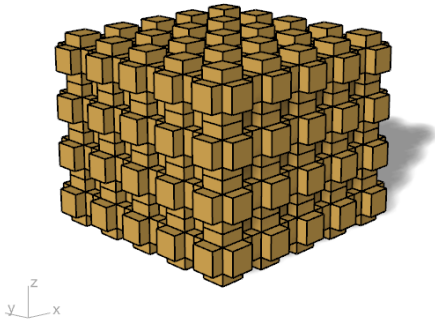
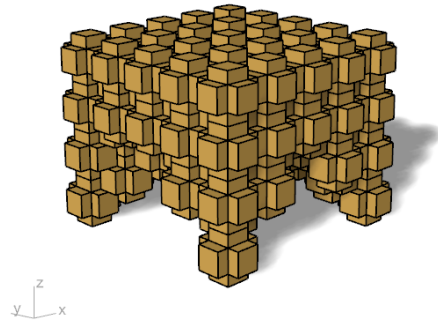


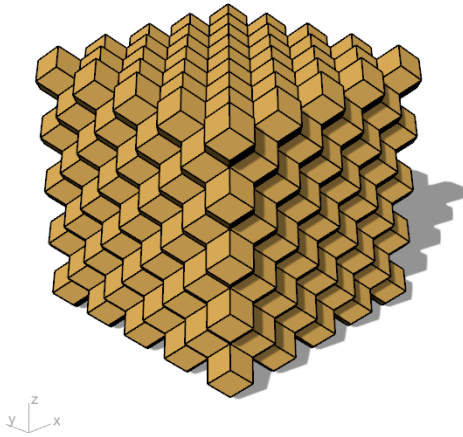
Figure 3.37: arch optimization



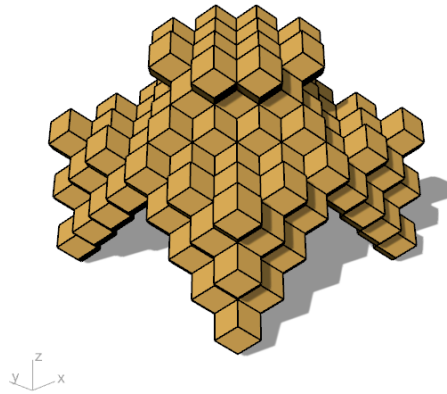
(a) CS-based cross vault before optimization



(b) CS-based cross vault at optimization iteration 7, showing elements with a design variable of 0.30 or higher



(c) RD-based cross vault before optimization



(d) RD-based cross vault at optimization iteration 17, showing elements with a design variable of 0.53 or higher

Figure 3.38: cross vault optimization

3.3.4.2. Static analysis

Static optimizations were performed on a multitude of different initial topologies. For these optimizations a penalization of $p = 3$ is used, as it resulted in faster convergence. The following optimizations are shown here:

- A 10x2x10 structure supported at the bottom corners (deep beam)
- A 8x15x7 structure supported along the y-directions on the bottom at opposing ends (long corridor)
- A 8x5x12 structure supported at the bottom at with a void of 4x5x4 (tall corridor)
- A 8x8x10 structure supported at the bottom with multiple voids (room)
- A 8x8x9 structure supported at the bottom with multiple voids (crossing)

An overview of the values used per analysis is given in table 3.6. The results are illustrated in figures 3.41, 3.42, 3.43, 3.44 and 3.45 for both CS and RD-based structures. It becomes apparent that these structures are significantly larger than the structures optimized using the time-explicit analysis. Optimizations of RD structures took significantly longer than CS structures of the same size.

One should note that during the static optimizations a downward external is introduced on the top-most elements. Without this applied force, the optimizer yielded nonsensical results (figure 3.39). It is schematically shown how these forces are exactly defined in figure 3.40.

It is also noted that to acquire the RD structures, a smaller cut-off value was used than for CS structures. Setting a higher cut-off value for RD structures resulted in very slender (and sometimes even floating) structures. This could be an inherent difference between these types of elements, and perhaps this can be counteracted by changing the input parameters in some way.

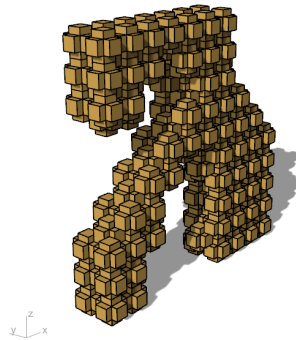


Figure 3.39: resulting 'optimal structure' using static analysis without externally applied forces

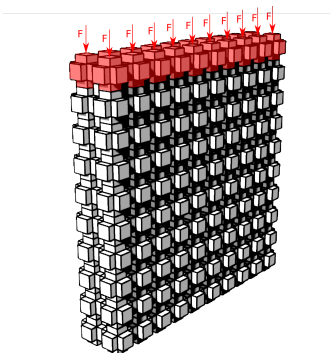
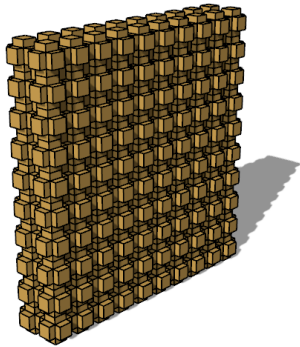


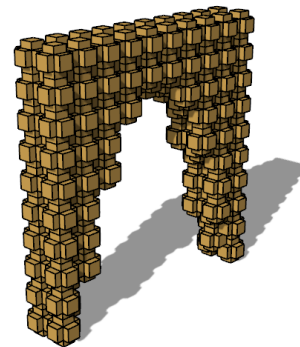
Figure 3.40: definition of the applied downward force F per element

shape	structure	E (Pa)	ν	F (N)	r (m)	μ_0	μ_1	μ_2	f	run-time (min)
CS	deep beam	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	5.0	1.0	0.4	2
RD	deep beam	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	100.0	1.0	0.4	25
CS	long corridor	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	5.0	1.0	0.4	22
RD	long corridor	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	100.0	1.0	0.4	184
CS	tall corridor	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	3.0	1.0	0.4	9
RD	tall corridor	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	100.0	1.0	0.4	107
CS	room	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	3.0	1.0	0.4	37
RD	room	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	5.0	1.0	0.7	214
CS	crossing	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	2.0	1.0	0.4	30
RD	crossing	$2000 \cdot 10^6$	0.25	100000	1.6	1.0	5.0	1.0	0.7	162

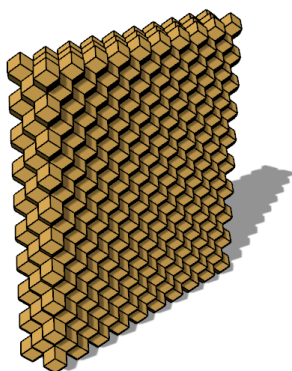
Table 3.6: table showing the values used during the static optimizations



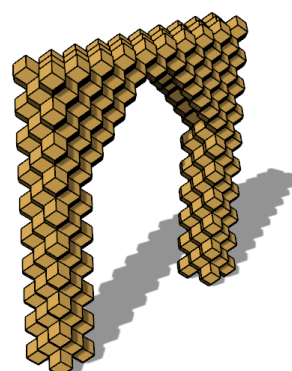
(a) CS-based deep beam before optimization



(b) CS-based deep beam at optimization iteration 35, showing elements with a design variable of 0.85 or higher



(c) RD-based deep beam before optimization



(d) RD-based deep beam at optimization iteration 48, showing elements with a design variable of 0.20 or higher

Figure 3.41: deep beam optimization

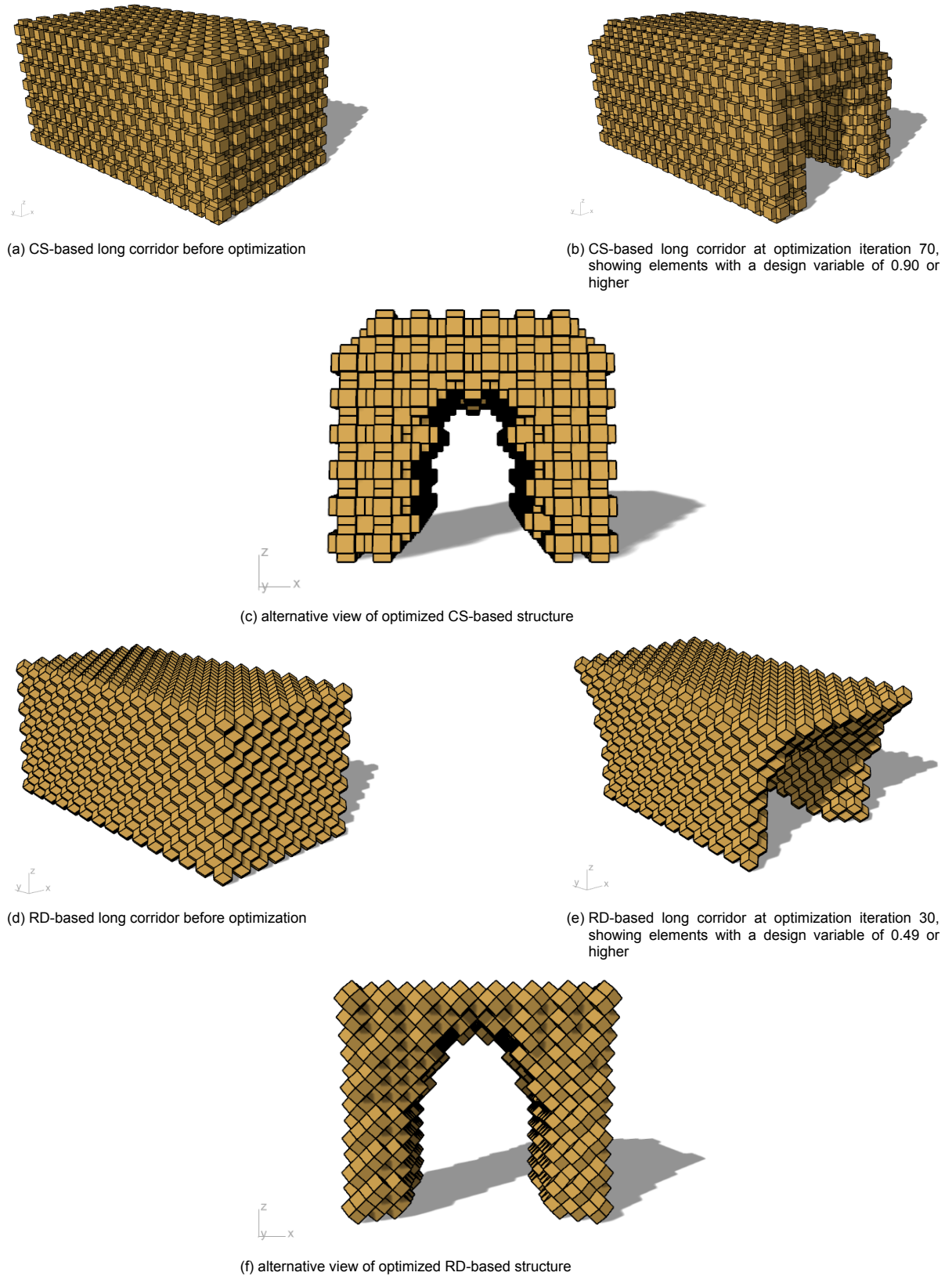
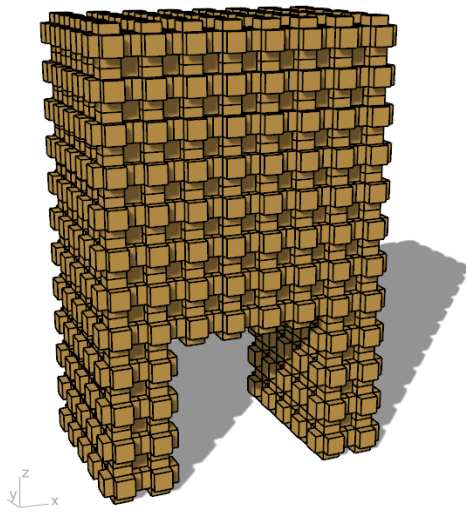
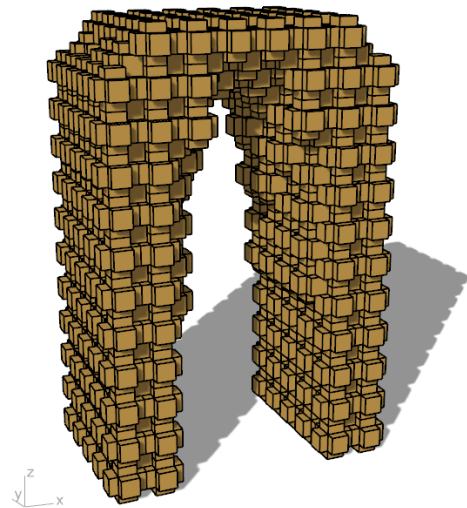


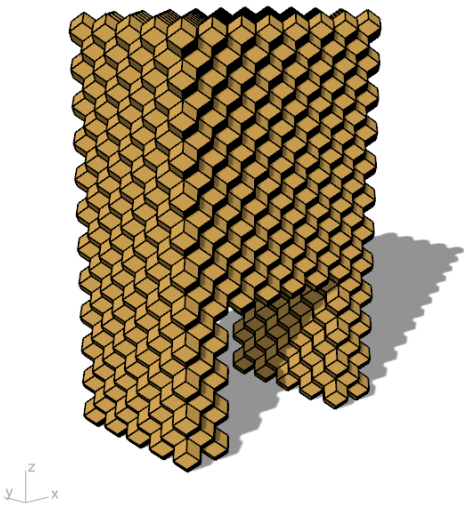
Figure 3.42: long corridor optimization



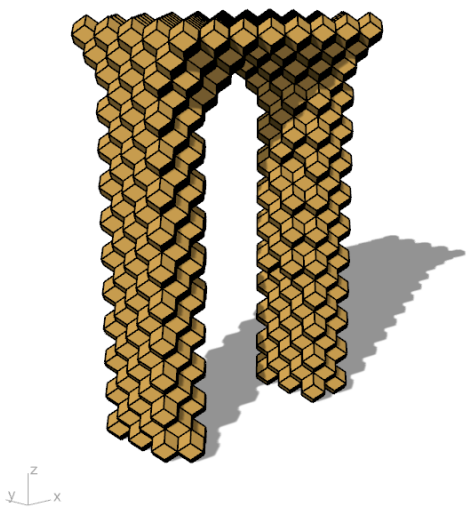
(a) CS-based tall corridor before optimization



(b) CS-based tall corridor at optimization iteration 72, showing elements with a design variable of 0.90 or higher



(c) RD-based tall corridor before optimization



(d) RD-based tall corridor at optimization iteration 100, showing elements with a design variable of 0.40 or higher

Figure 3.43: tall corridor optimization

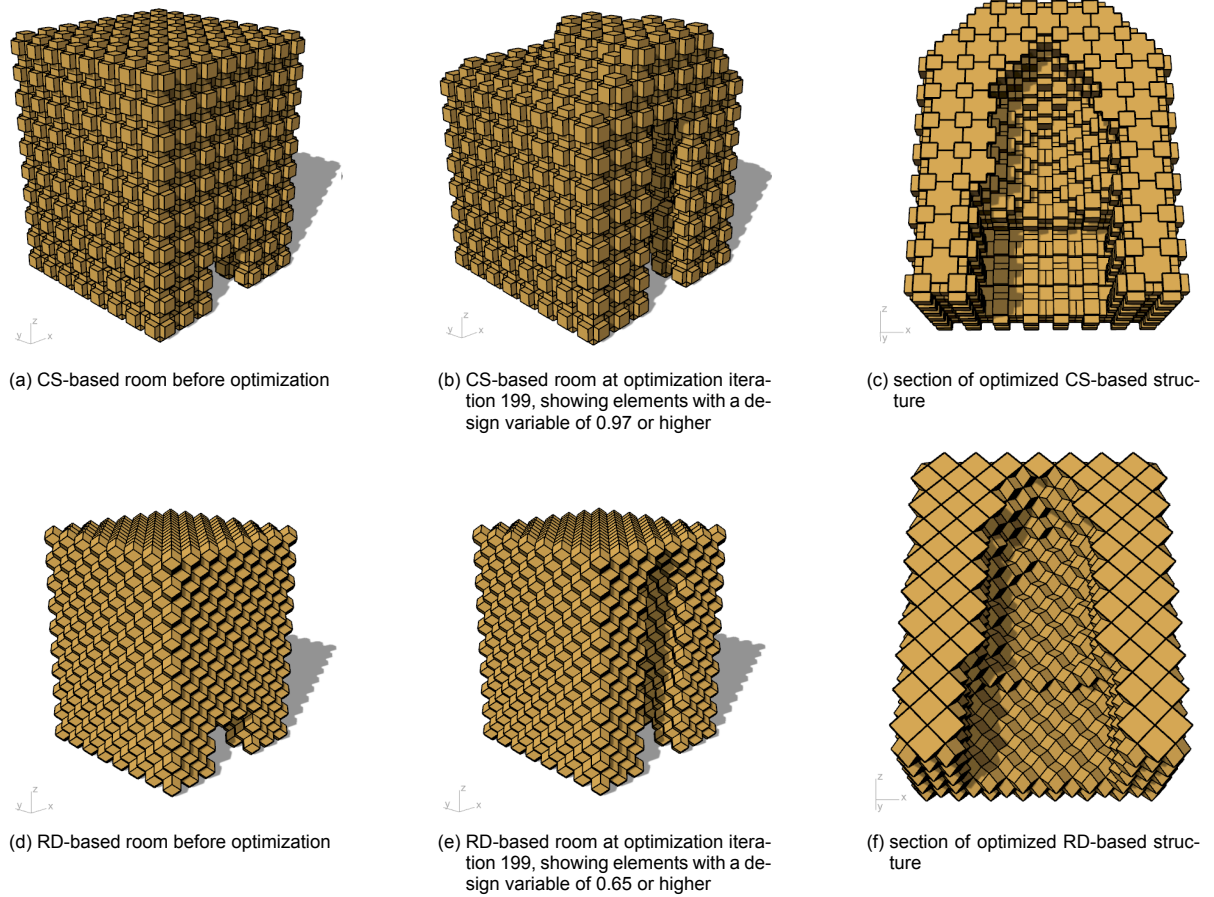


Figure 3.44: room optimization

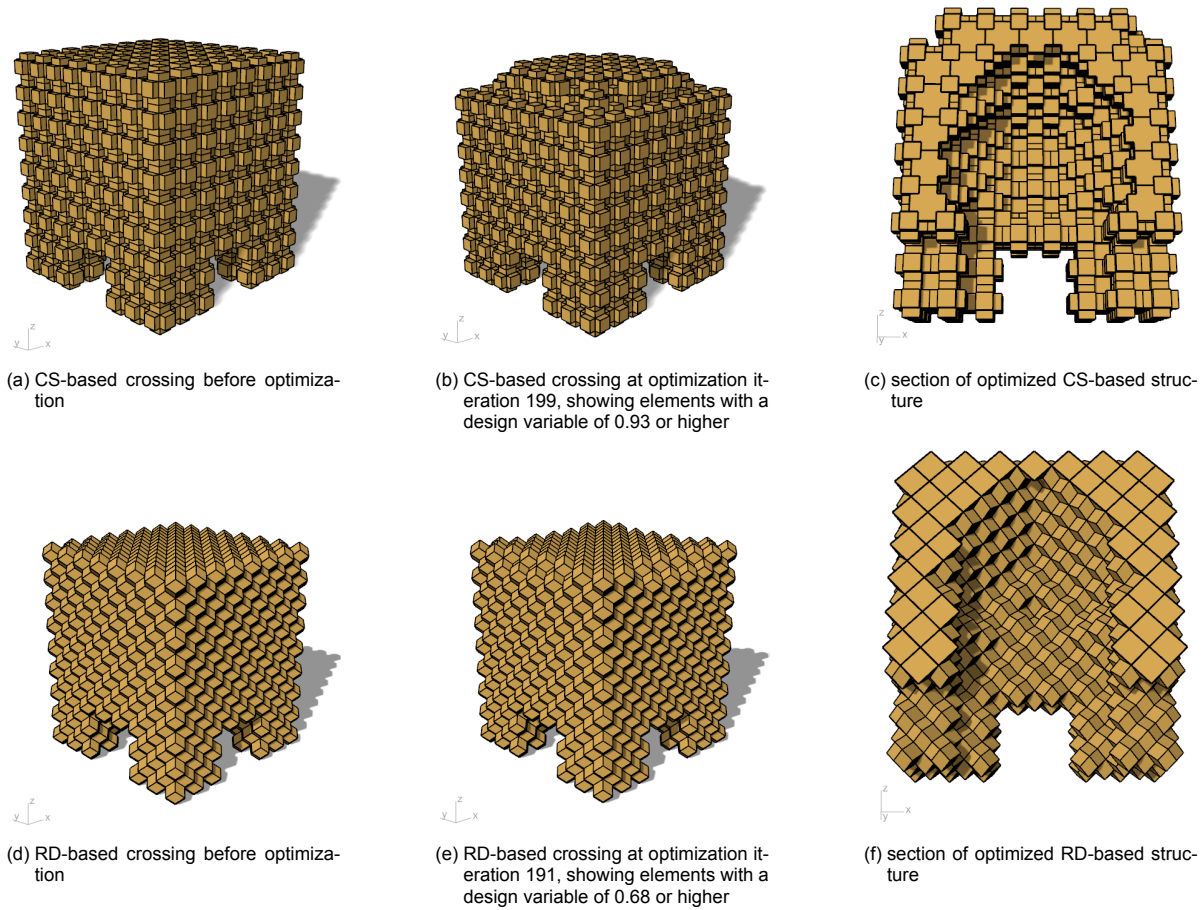


Figure 3.45: crossing optimization

When performing a topology optimization using the static analysis on smaller structures, like the ones shown in the optimization using the time-explicit analysis, very different results are obtained (figure 3.46). When using the static analysis method to optimize smaller structures, the supports tend to be removed, which may result in unrealistic structures. It is concluded that optimizations using the time-explicit method are better suited for smaller structures, while the static method works better for somewhat larger structures.

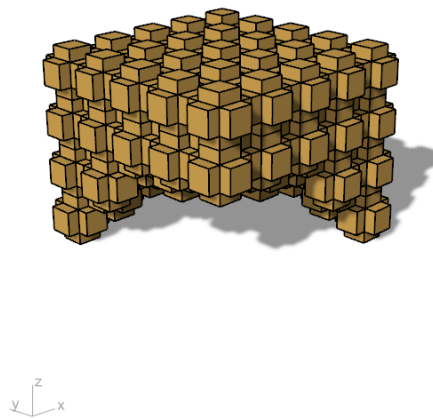
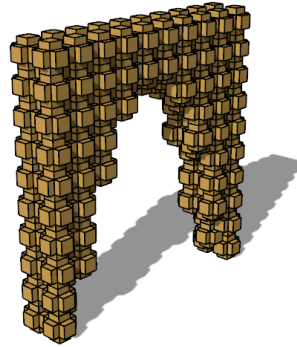
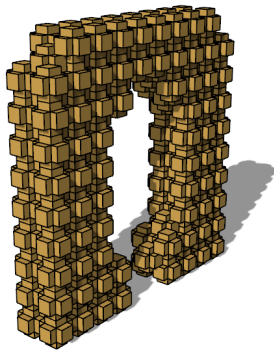
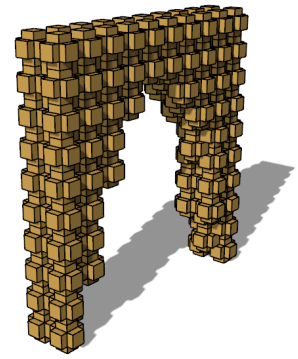
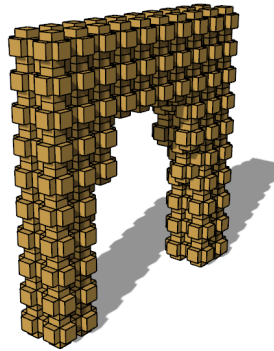
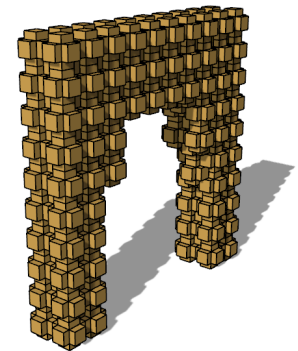


Figure 3.46: optimized 'cross-vault' using the static analyses method

Some further optimizations were done to observe the effects of different parameters. These variation studies were performed on the CS deep beam structure. Figure 3.47 shows the results of these optimizations.



(a) original

(b) $E = 1000 \cdot 10^6 \text{ Pa}$ (c) $E = 4000 \cdot 10^6 \text{ Pa}$ (d) $\nu = 0.1$ (e) $\nu = 0.4$

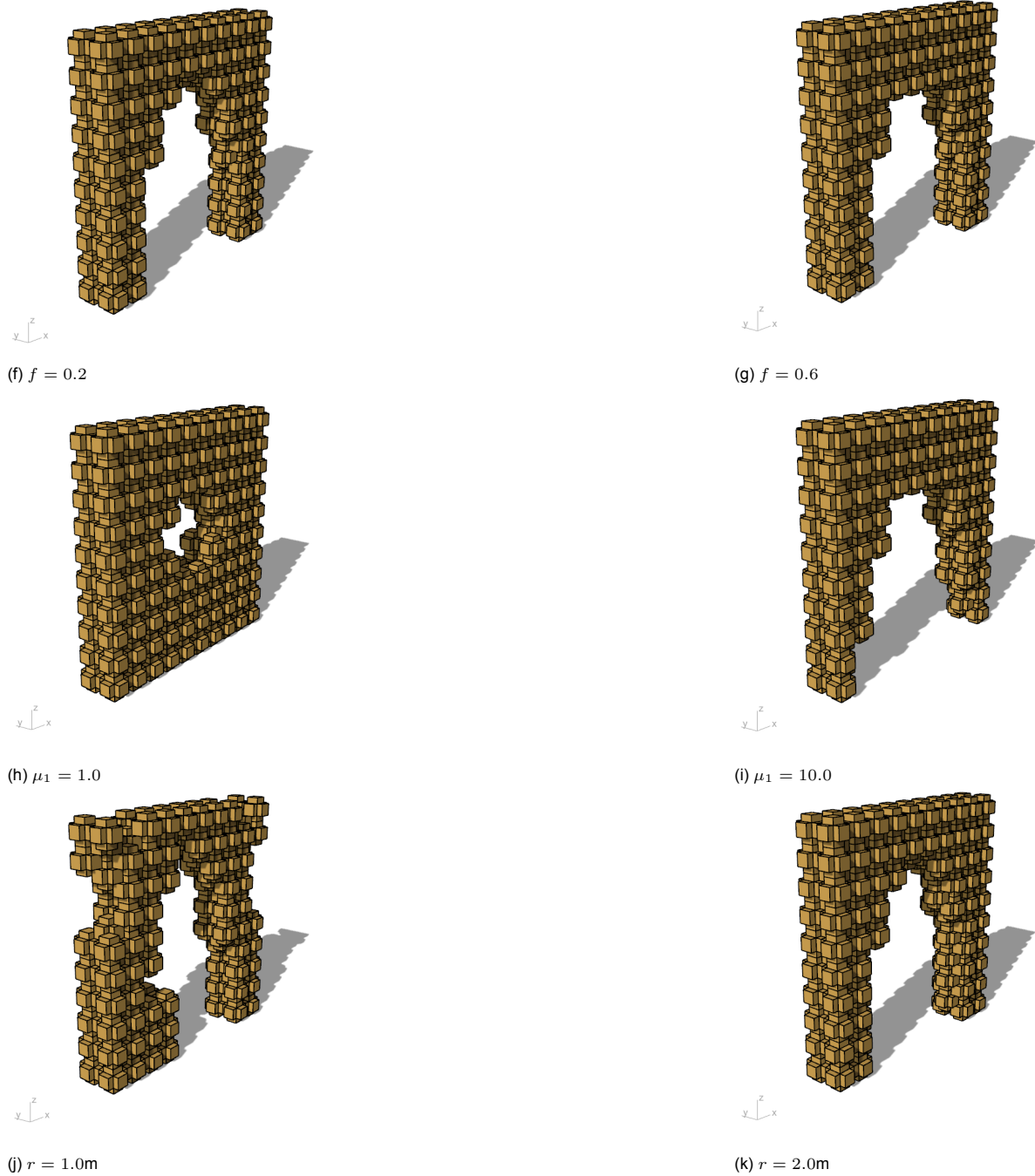


Figure 3.47: variations on the CS deep beam

It appears that the parameters affecting the optimization results most are the Young's Modulus E , the volume constraint multiplier μ_1 and the filtering radius r . When too low values are taken for these parameters, the optimizer may yield unrealistic structures. Especially when optimizing for materials with a lower Young's Modulus this could pose limitations, as the other parameters are non-physical and are only used by the optimizer.

4

Verification

To test whether the optimized structures are feasible, their structural behaviour is verified using 3DEC, a commercial software package based on the discrete element method. In the following sections the verifications of both the locally damped time-explicit and static optimizations are shown.

The code to run the verifications is mostly the same for all the analyses. The only changes are the model names and in some cases different material properties. The used code is shown below:

```
1  model new
2  model large-strain on
3
4  block generate from-vrml filename '[MODEL NAME].wrl'
5  block apply velocity (0.0 0.0 0.0) range position-z 0.0 0.6
6
7  model gravity 9.81
8  block face triangulate radial-8
9  block contact generate-subcontacts
10 block contact prop stiffness-norm=[k_n] stiffness-shear=[k_s] friction=80
11 block contact material-table default prop stiffness-norm=[k_n] stiffness-shear=[k_s] fric
    =80
12 block property density 2000
13
14 plot s
15 block mech damp local 0.80
16 model solve
```

The friction angle is taken to be 80° to neglect slipping, as this is not considered in the developed model either.

4.1. Time-explicit optimizations

To verify the optimizations using the locally damped time-explicit model, first the displacements of the optimized structures are checked (figure 4.1).

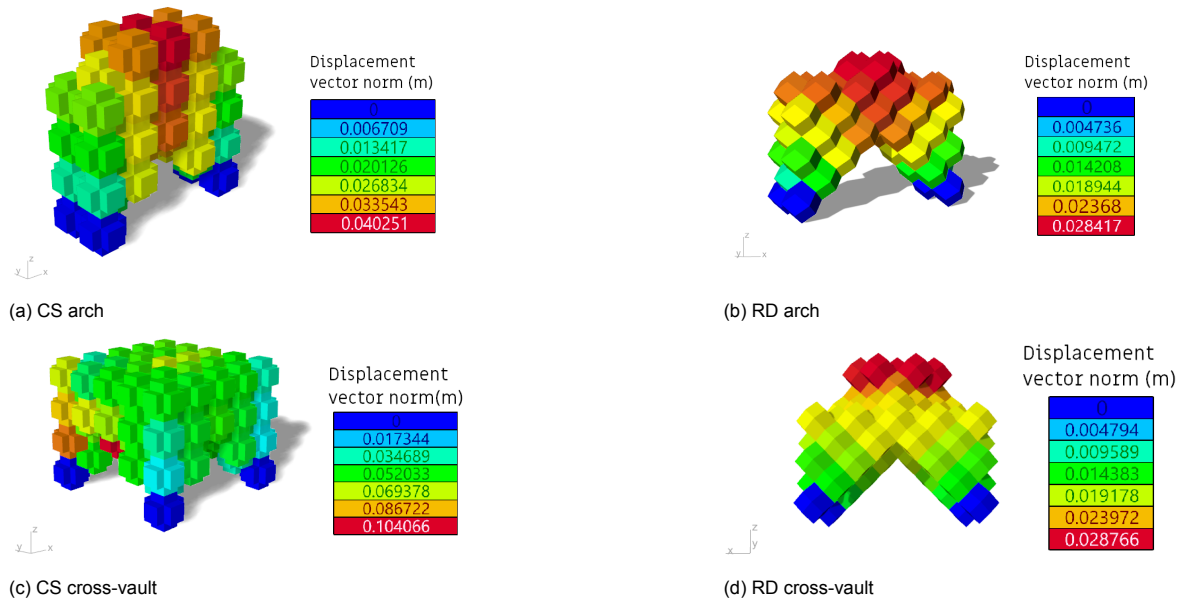


Figure 4.1: displacements of the different time-explicitly optimized structures

To calculate the correct values to use for k_n and k_s within 3DEC, it should be noted that 3DEC makes use of these stiffness values defined per area. These values are multiplied by the area that is represented by nodes during the analysis (figure 4.2). Making use of the values for the undeformed spring length as calculated in eqs (3.1) and (3.8) the following stiffness values are obtained using eqs (3.14) and (3.16) ($s = 1 \text{ m}$, $E = 5 \cdot 10^6 \text{ Pa}$, $\nu = 0.25$):

$$k_{n,CS} = \frac{5 \cdot 10^6}{0.25s} = 20 \cdot 10^6 \frac{\text{Pa}}{\text{m}}$$

$$k_{s,CS} = \frac{5 \cdot 10^6}{2(1+0.25)} = 8 \cdot 10^6 \frac{\text{Pa}}{\text{m}}$$

$$k_{n,RD} = \frac{5 \cdot 10^6}{0.1768s} \approx 28 \cdot 10^6 \frac{\text{Pa}}{\text{m}}$$

$$k_{s,RD} = \frac{5 \cdot 10^6}{2(1+0.25)} \approx 11 \cdot 10^6 \frac{\text{Pa}}{\text{m}}$$

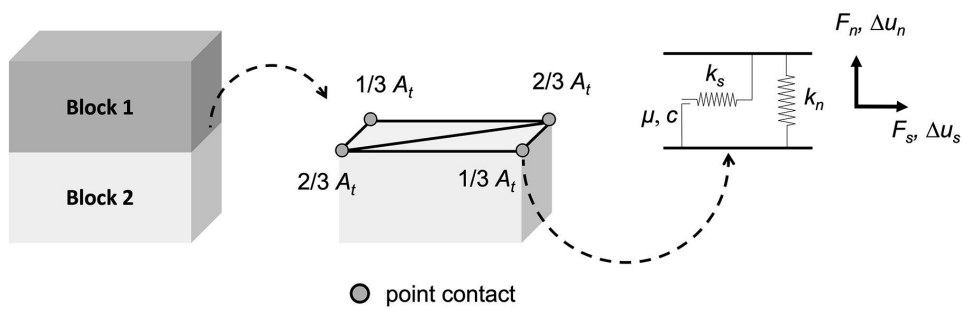


Figure 4.2: mechanical representation of contact between blocks as implemented in 3DEC (Colombo et al., 2022)

Using these values for their respective structures however, results in elements falling through other elements for CS structures, indicating that the used stiffness values are too low (figure 4.3). It should be noted that this behaviour was not observed in RD structures with the same stiffness values.

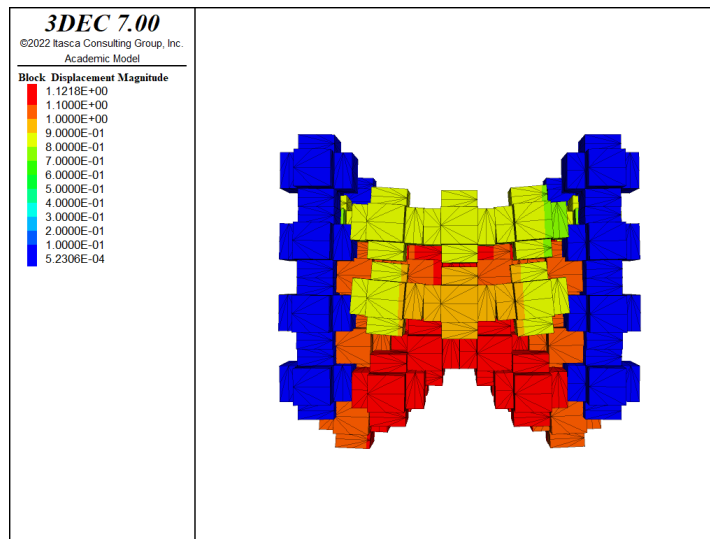


Figure 4.3: elements falling through each other resulting from too low stiffness components

To be sure whether this is not an issue originating from importing the geometry, the optimized CS structures are imported in a different manner. The CS elements are imported to 3DEC as seven separate rectangular prisms which are then joined within 3DEC to form a CS (figure 4.4). This process does slow down the analysis somewhat, as first these blocks are joined together one by one before any structural analysis is performed. Performing a structural analysis in this way on CS structures however, does lead to converging structures where no blocks fall through each other using the aforementioned stiffness values. The results for both CS and RD structures are shown in figure 4.5.

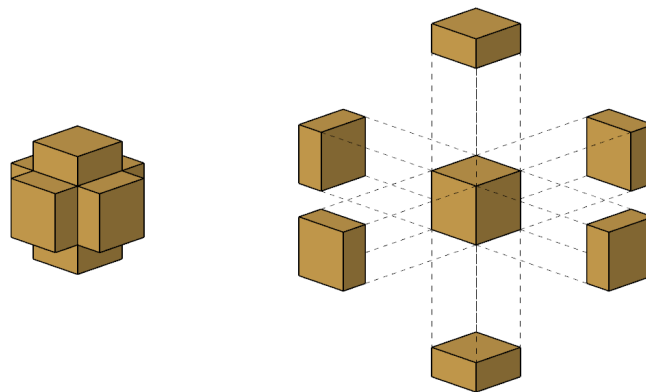


Figure 4.4: the division of CS elements into seven rectangular prisms that are joined together in 3DEC

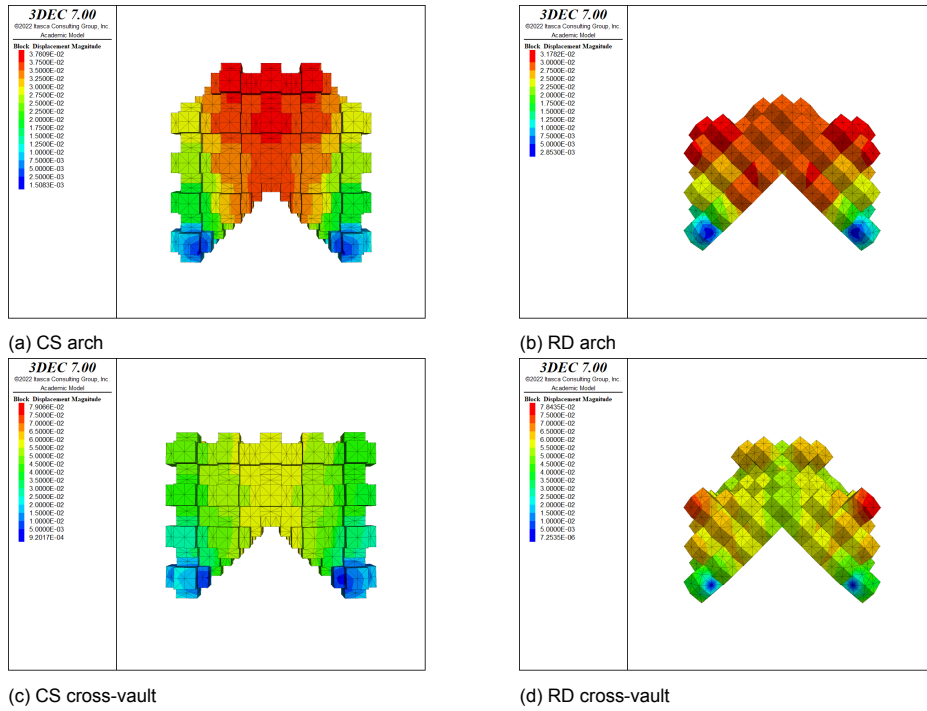


Figure 4.5: displacements of the different time-explicitly optimized structures from 3DEC

Comparing the displacements for each respective structure, it becomes apparent that there is a significant difference in the values of both models. One reason for this difference, could be that the stiffness values are reduced during the optimization as a result of the design variables (eq. (3.80)). The developed model is not implemented to perform structural analyses using the unpenalized stiffness matrix for optimized structures. To get around this issue and have some kind of comparison between the two models, an optimization is carried out with the volume constraint set to $f = 1.0$ and defining voids and supports in such a way that the optimized structure is inputted directly. The displacements shown after the first iteration of the optimization would then be the displacements for the optimized structure with unpenalized stiffness values. As this is quite cumbersome to input for every optimized structure, only the case of the CS arch is considered (figure 4.6). The maximum displacement shown here has a relative difference of about 6% compared to the maximum displacement of the CS arch analysed using 3DEC. It is assumed that the same similarities hold for the other structures shown in figure 4.5.

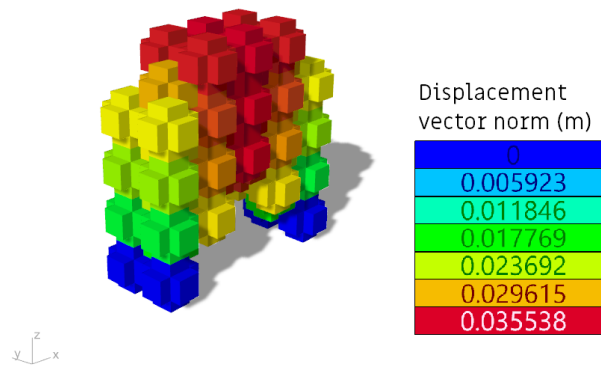


Figure 4.6: analysis of the CS arch with unpenalized stiffness values

4.2. Static optimizations

All the optimizations from section 3.3 based on the static analyses are also verified using 3DEC. These analyses were carried out using the following stiffness values ($s = 1 \text{ m}$, $E = 2 \cdot 10^9 \text{ Pa}$, $\nu = 0.25$):

$$k_{n,CS} = \frac{2 \cdot 10^9}{0.25s} = 8 \cdot 10^9 \frac{\text{Pa}}{\text{m}}$$

$$k_{s,CS} = \frac{2 \cdot 10^9}{0.25s} = 3.2 \cdot 10^9 \frac{\text{Pa}}{\text{m}}$$

$$k_{n,RD} = \frac{2 \cdot 10^9}{0.1768s} \approx 11.3 \cdot 10^9 \frac{\text{Pa}}{\text{m}}$$

$$k_{s,RD} = \frac{2 \cdot 10^9}{0.1768s} \approx 4.52 \cdot 10^9 \frac{\text{Pa}}{\text{m}}$$

Some of these structures showed signs of local failures. In the optimized structures some elements are not properly supported and fall downward (figure 4.3).

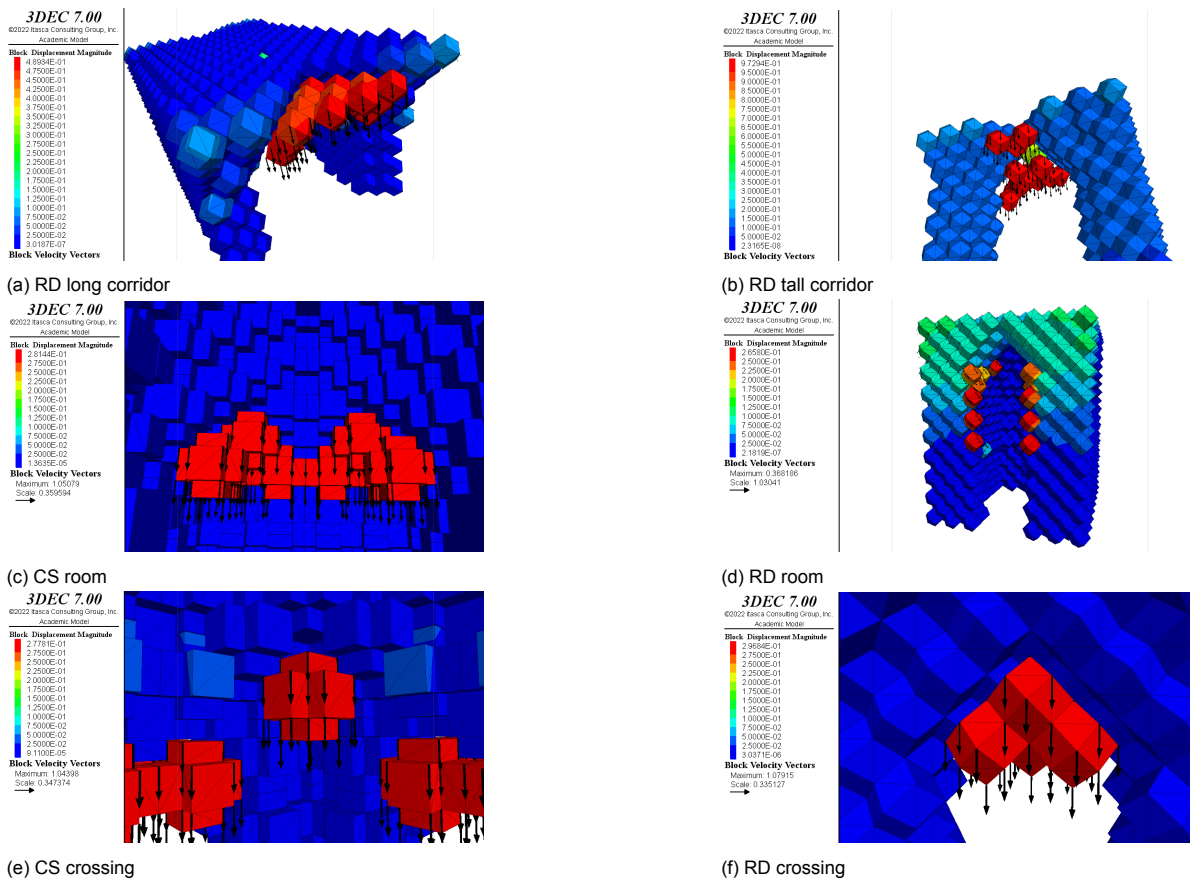
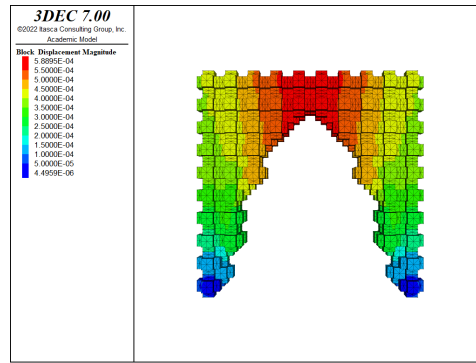
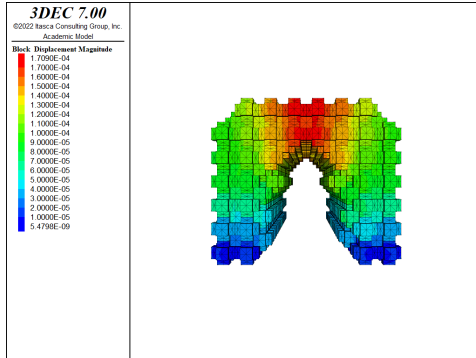


Figure 4.7: close-ups of some failures

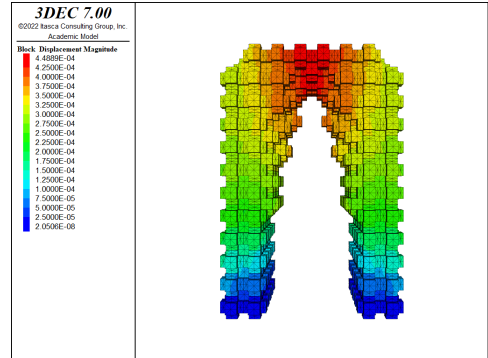
After removing the falling elements, the analyses converged without any issues. The results are shown in figures 4.8 and 4.9. Table 4.1 shows the maximum displacement values for the statically optimized structures using the method developed in chapter 3. Again, there is a significant difference between the maximum displacement values from 3DEC and the developed model. In the same way as outlined before, the CS deep beam is reanalysed in the optimizer with unpenalized stiffness values and, to keep the comparison fair, using a linear elastic analysis in 3DEC (figure 4.10). Comparing these maximum displacement values, a relative difference of about 5% is calculated. Again, the assumption is made that the same similarities hold for the other structures.



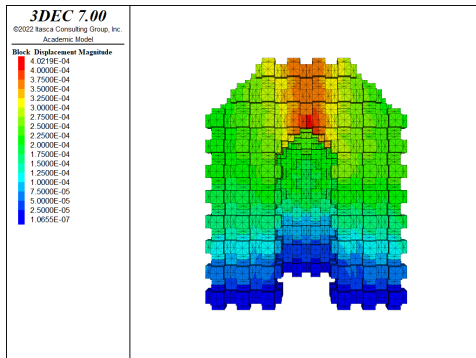
(a) deep beam



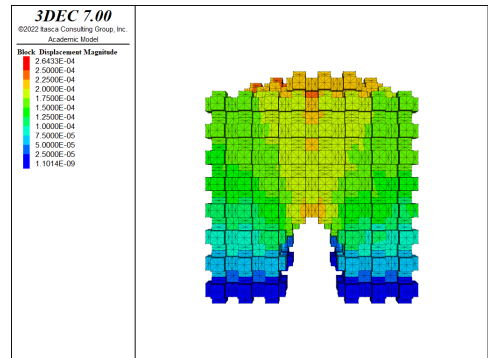
(b) long corridor



(c) tall corridor



(d) room



(e) crossing

Figure 4.8: 3DEC analyses on statically optimized CS structures after removing falling elements

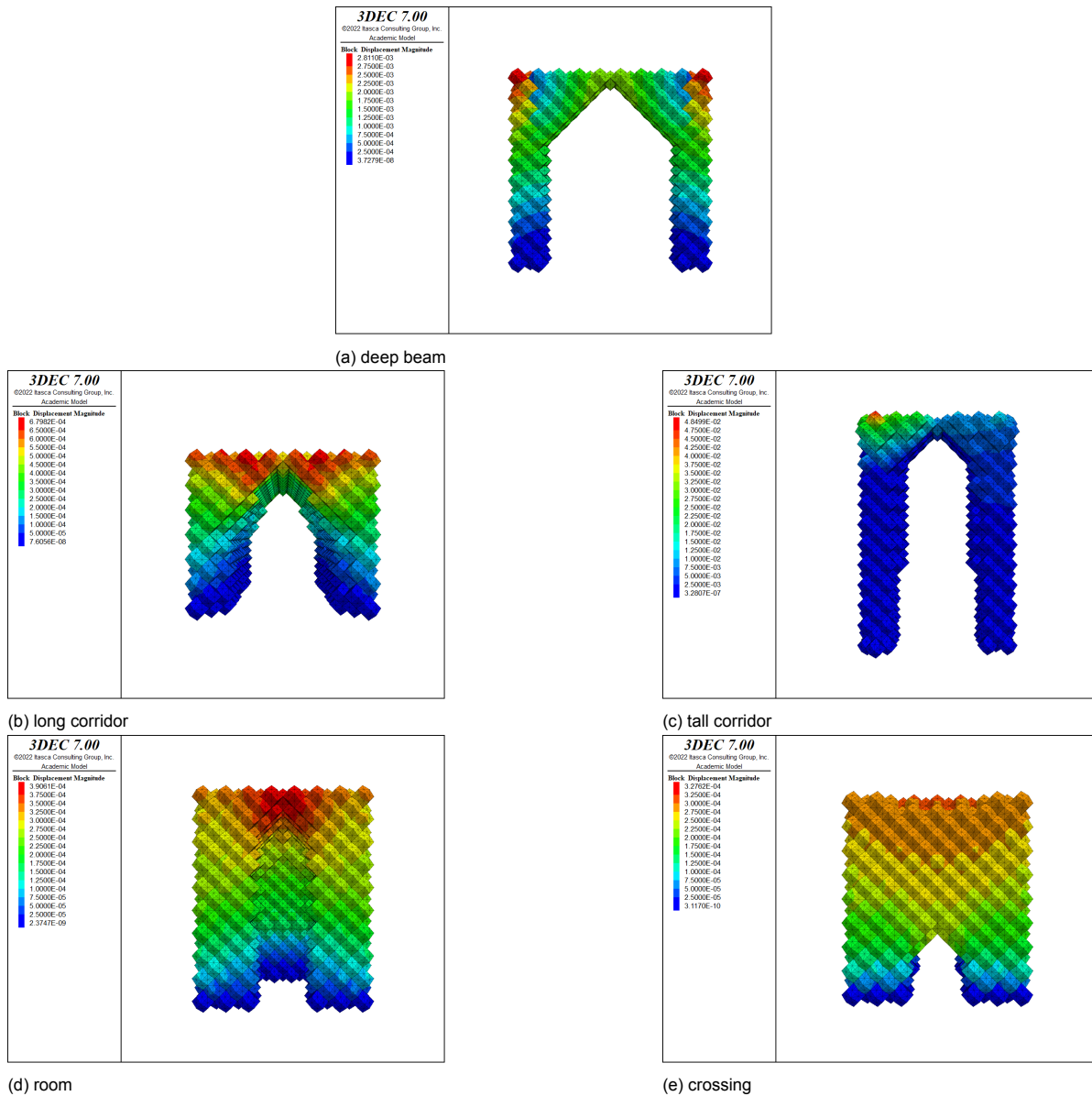


Figure 4.9: 3DEC analyses on statically optimized RD structures after removing falling elements

structure	maximum displacement CS (m)	maximum displacement RD (m)
deep beam	$1.912 \cdot 10^{-3}$	$2.052 \cdot 10^{-2}$
long corridor	$5.35 \cdot 10^{-4}$	$4.634 \cdot 10^{-3}$
tall corridor	$1.208 \cdot 10^{-3}$	$1.163 \cdot 10^{-2}$
room	$7.93 \cdot 10^{-4}$	$1.074 \cdot 10^{-3}$
crossing	$8.12 \cdot 10^{-4}$	$1.191 \cdot 10^{-3}$

Table 4.1: table showing the maximum displacement values for the statically optimized structures

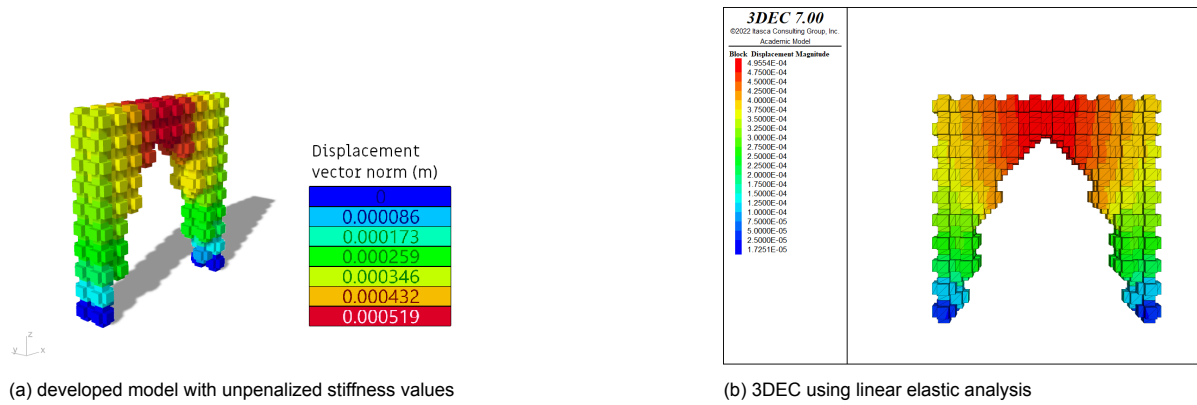


Figure 4.10: analyses of the CS deep beam

4.3. Physical model

As an extra verification, a physical model is made out of 3D-printed cubic spheres. It proved difficult to make a small-scale model however. The tolerances of the elements are too small to fit well, and even when the elements fit they are easily dislodged. With the help of glue (analogous to mortar) the CS arch model is recreated and this model is shown in figure 4.11.

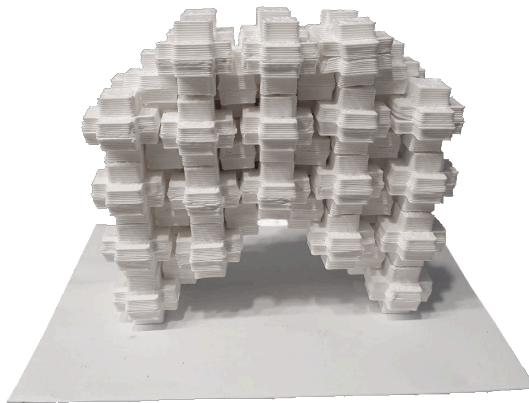


Figure 4.11: physical model of the CS arch

To conclude this chapter, it can be stated that the resulting optimized structures can be verified conclusively using 3DEC. The same stiffness values can be used, but one should note that 3DEC uses a different unit for stiffnesses ($\frac{\text{Pa}}{\text{m}}$ instead of Pa). All structures converged in 3DEC without any issues, except for statically optimized structures where some unsupported elements were present. After removing the unsupported elements, these structures also converged without any issues. Comparisons of maximum displacement values resulting from 3DEC and the developed model are difficult to carry out, as the developed model runs analyses with penalized stiffness values. Inputting the optimized structure with a volume constraint of $f = 1.0$ to the optimizer enables one to make this comparison with 3DEC, but this is quite a cumbersome procedure. A small-scale physical model to act as further verification proved to be difficult to accurately produce and assemble.

5

Application

In this chapter some possible applications of the current implementation of the topology optimization tool using the discrete element method are shown. In addition to that, some ideas are presented for future developments.

5.1. Current implementation

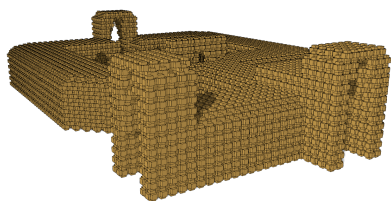
5.1.1. Optimization by parts

The presented optimization procedure can be used as a design tool based on structural principles. As it is not possible to input a large-scale structure and optimize it within a reasonable time-frame, the user could to optimize the structure by parts.

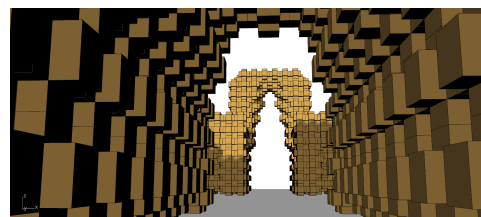
One could use the structures optimized in chapter 3.3 and put those together to create a design. This is done below as an example, with the addition of two more optimized structures. An overview of which structures and how many of them are used, is given in table 5.1. Some impressions from the final design are shown in figure 5.1. Different designs can be created by optimizing more structures or arranging a given set of structures in a different way.



(a)



(b)



(c)

Figure 5.1: resulting design from optimization by parts

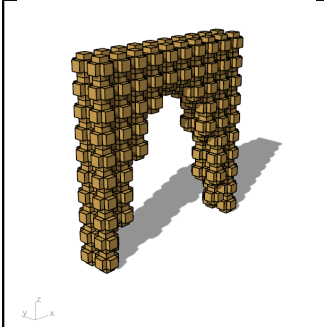
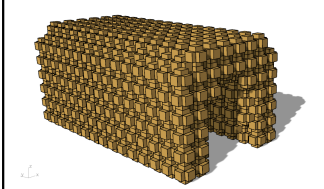
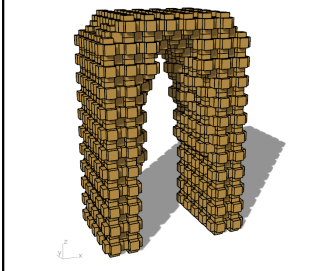
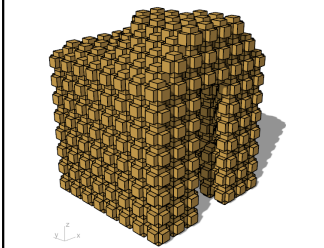
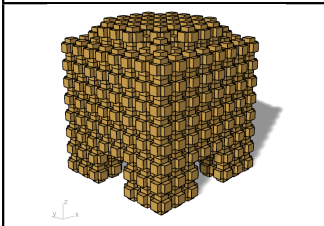
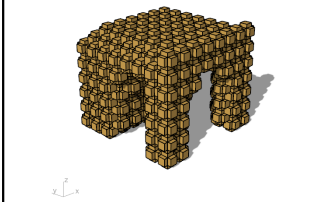
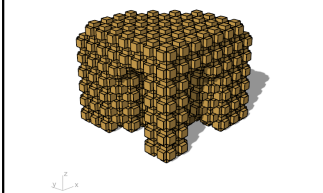
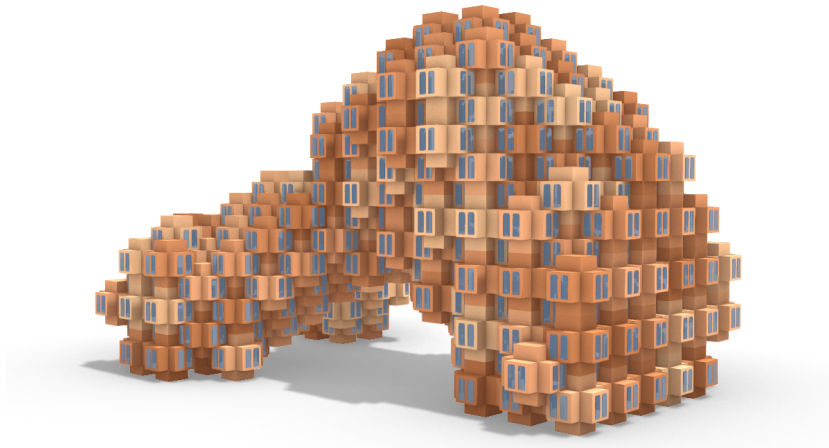
	deep beam	1
	long corridor	2
	tall corridor	3
	room	1
	crossing	1
	colonnade straight	16
	colonnade bend	4

Table 5.1: table showing the values used during the static optimizations

5.1.2. Optimization of modular structures

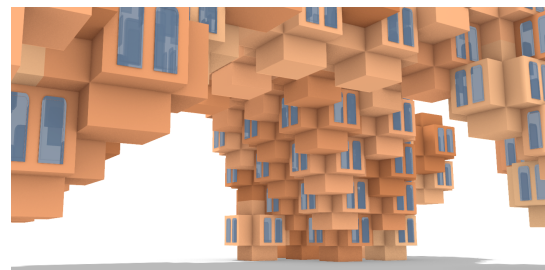
It is possible to regard the elements as building modules or rooms instead of masonry elements. This enables the user to create an initial design of a large-scale structure in a relatively short amount of time. Figure 5.2 shows a result of this approach and is reminiscent of The 5 Farming Bridges, a design of bridges consisting of modular housing units by Belgian architect Vincent Callebaut (figure 5.3) or Habitat 67, a housing complex designed by Israeli-Canadian architect Moshe Safdie (figure 5.4). It should be noted that this approach does not model any structural response from within these modular units; they are modelled as rigid, solid blocks. As such, the feasibility of these structures should be verified with more detailed models.



(a)



(b)



(c)

Figure 5.2: optimization of a modular structure



Figure 5.3: The 5 Farming Bridges (Vincent Callebaut Architectures. (n.d.). The 5 farming bridges. https://vincent.callebaut.org/object/171023_mosul/mosul/projects)



Figure 5.4: Habitat 67 (E. Blue. (n.d.). Habitat 67. <https://www.mtl.org/en/experience/revolutionary-montreal-icon-habitat-67>)

5.2. Future visions

In this section some ideas for future applications are presented. It should be noted that these ideas are not practical or possible to carry out using the current implementation. Instead, these applications are intended to provide some direction towards which the current tool can be developed to expand on its applications within the construction industry.

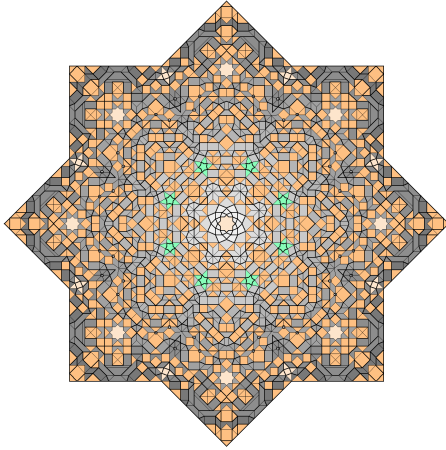
5.2.1. Structures with custom tessellations

The initiation of the geometry outlined in section 3.1 is limiting in the sense that only structures can be optimized that consist of only one type of elements. The optimization of CS and RD structures has been presented in detail in chapter 3. It is however, not unconventional to construct structures that consist of multiple types of elements, having either multiple material properties or shapes.

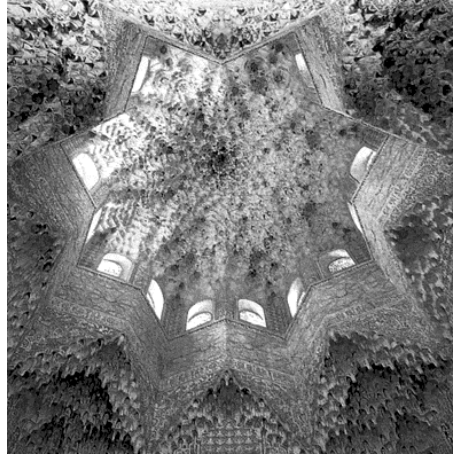
The optimization of multi-material structures can be implemented by defining multiple constitutive models and material properties, possibly along with an additional optimization constraint to control the ratio of different materials used in an optimized structure.

Adding the possibility to model different tessellations can be done in two ways. One could either implement different repeating patterns as was shown in section 3.1, in which the only user inputs are how many times the structure repeats in each direction and the physical size of each repetition. This method however, requires one to implement every pattern separately and severely limits the user's freedom when it comes to providing the optimizer with different tessellations, as only predefined tessellations can be analysed and optimized. A more functional procedure would be to implement the option

to provide a custom tessellation as meshes (a collection of nodes, edges and faces) that represent the individual elements (or elementary shapes). Then this tessellation can be abstracted, analysed and optimized the same way as outlined before. This provides the user the ability to optimize different tessellations of the same structure and serves as an additional degree of freedom in the design process. It is expected that this additional feature may provide the optimizer the possibility to output for example a muqarnas, which is a type of ornamental vaulting reminiscent of fractals (figure 5.5).



(a) drawing

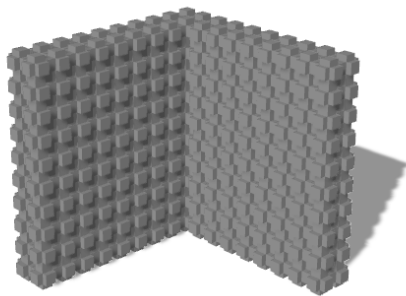


(b) photograph

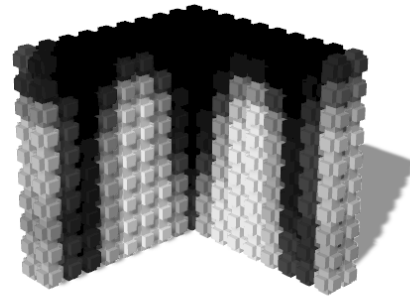
Figure 5.5: a muqarnas in Alhambra, Granada (S. Takahashi. (n.d.). Sala de los abencerrajes. alhambra palase, granada. spain. 1354-91. <http://www.shiro1000.jp/muqarnas/data/034/34.htm>)

5.2.2. Aid in restoration projects

Another future application that could be of interest is using the optimizer as an aid during renovation or restoration projects of existing masonry structures. If the geometry of these structures is imported, abstracted, analysed and optimized, one could get an overview of which parts of the structure contribute more to its structural integrity than others (figure 5.6). Perhaps this knowledge could prove useful to safely and efficiently intervene in existing masonry structures. One approach that could possibly be taken is to regard elements with a low design value (close to 0) as being of lesser importance for the structural integrity than elements with a high design value (close to 1). Whether this is truly the case should be verified using additional structural analyses; for instance by removing elements with a low design value and running a structural analysis on the resulting structure, as is done in chapter 4. Afterwards a suitable intervention strategy can be proposed, for example how to support the structure during reconstruction or which elements should be reinforced or replaced to prolong the lifespan of the structure as efficiently as possible.



(a) initial structure



(b) visualization of design variables (white=0, black=1)

Figure 5.6: topology optimization of two masonry walls, which could be part of an existing structure. In this case the lower-most elements are fixed, while the top-most elements are subjected to a downward force

6

Conclusions

This thesis has outlined an approach to optimize the topology of masonry (or more generally, discrete) structures. It has detailed the generation of the starting topology, how a structural analysis can be performed on this and how the results of these analyses can be used to optimize the topology.

For the structural analyses it can be concluded that two methods outlined in this thesis are viable options to analyse masonry structures. The locally damped time-explicit method is better suited for optimizations of small structures due to the time it takes to convergence during the structural analysis. Static analyses do not need to converge and are thus faster, but they cannot include the effect of non-linear material behaviour without making use of iterative procedures, which defeats its purpose. Time-explicit analyses using global-damping have shown to be unstable, and are thus not suitable for the structural analyses shown in this thesis.

Results of different optimizations are shown with different geometrical properties and boundary conditions. In addition to this, results from optimizations while changing just one parameter are shown. These results have indicated that mainly the Young's Modulus of the elements, the volume constraint weight and the filtering radius affect the final outcome of the optimization. The Young's Modulus seems to be the only physical quantity affecting the outcome of the optimizations. To be sure of the truth of this statement, more research on the effects of different model quantities should be conducted. For now it could be said that structures consisting of elements made out of different materials (e.g., wooden or concrete blocks) can be optimized using the outlined approach, as long as the Young's Modulus is not low enough for the optimizer to include elements connected in tension.

One issue that should be pointed out with the volume constraint used in the optimization algorithm, is that it does not seem to be strictly enforced during the optimizations. This means that the user has less control over the final volume of the optimized structure. As mentioned above, the volume is dependent on other parameters, the effects of which on the volume may not necessarily be predictable.

The results from a Python implementation are verified using 3DEC. The structural integrity of the optimizations seems to be ensured if slipping is neglected. Comparisons between 3DEC and the developed model are somewhat difficult to carry out, as the developed model runs the structural analyses using penalized stiffness matrices. A method is outlined that makes these comparisons possible, and the results of both models are similar for the cases considered. To achieve more certainty in the structural performance of the optimized structures one could physically construct these. Small-scale models do not seem to be a practical and reliable method to do this, so one could verify the resulting structures by constructing them in full-scale. This is not done here as this goes beyond the scope of this thesis.

The current model can be used to create architectural designs either by parts or as modular units within a reasonable time-frame. The outlined method thus provides the user the possibility to optimize indefinitely many structures defined by their dimensions, voids, material properties, boundary conditions and optimization parameters. These optimizations result in a final design which is based on its structural performance. This is in line with the the aim of this thesis, which is to design sustainable

structures not only by choosing construction materials that have low embodied energy, are durable and relatively cheap, but also by optimizing the structures that contain these materials to further reduce their economical and ecological footprint. Whether the resulting structures actually have a lower embodied energy value or a longer lifespan or whether they are cheaper than similar structures using more conventional structural materials like concrete or steel requires further investigation.

Bibliography

- Bendsøe, M. P. (1989). Optimal shape design as a material distribution problem. *Structural Optimization*, 1(4), 193–202. <https://doi.org/10.1007/bf01650949>
- Bendsøe, M. P. (1995). *Optimization of structural topology, shape, and material* (Vol. 414). Springer.
- Bendsøe, M. P., & Sigmund, O. (1999). Material interpolation schemes in topology optimization. *Archive of applied mechanics*, 69(9), 635–654.
- Bendsoe, M. P., & Sigmund, O. (2003). *Topology optimization: Theory, methods, and applications*. Springer Science & Business Media.
- Bruyneel, M., & Duysinx, P. (2001). Topology optimization with self-weight loading:(un-expected) problems and solutions. *2nd Max Planck Workshop on Engineering Design Optimization*.
- Burkardt, J. (2010). Computational geometry lab: Tetrahedrons.
- Carmichael, R. S. (2017). *Practical handbook of physical properties of rocks and minerals (1988)*. CRC Press. <https://doi.org/10.1201/9780203710968>
- Červenka, V., Jendele, L., & Červenka, J. (2000). Atena program documentation–part 1. *Cervenka Consulting sro*.
- Colombo, C., Savalle, N., Mehrotra, A., Funari, M. F., & Lourenço, P. B. (2022). Experimental, numerical and analytical investigations of masonry corners: Influence of the horizontal pseudo-static load orientation. *Construction and Building Materials*, 344, 127969. <https://doi.org/10.1016/j.conbuildmat.2022.127969>
- Cundall, P. A. (1982). Adaptive density-scaling for time-explicit calculations.
- Cundall, P. A. (1987). Distinct element models of rock and soil structure. *Analytical and computational methods in engineering rock mechanics*, 129–163.
- Cundall, P. A., & Strack, O. D. (1979). A discrete numerical model for granular assemblies. *geotechnique*, 29(1), 47–65.
- Deetman, A. (2021). Gcmmma-mma-python.
- Dell'Endice, A., Iannuzzo, A., Mele, T. V., & Block, P. (2021). Influence of settlements and geometrical imperfections on the internal stress state of masonry structures. *12th International Conference on Structural Analysis of Historical Constructions*. <https://doi.org/10.23967/sahc.2021.131>
- Dunning, P. D., & Kim, H. A. (2015). Introducing the sequential linear programming level-set method for topology optimization. *Structural and Multidisciplinary Optimization*, 51(3), 631–643.
- E. Blue. (n.d.). Habitat 67. <https://www.mtl.org/en/experience/revolutionary-montreal-icon-habitat-67>
- Frick, U., Mele, T. V., & Block, P. (2016). Data management and modelling of complex interfaces in imperfect discrete-element assemblies. *Proceedings of IASS annual symposia*, 2016(17), 1–9.
- Friswell, M., & Mottershead, J. (n.d.). Finite element model updating in structural dynamics.
- Ghiassi, B., & Lourenço, P. B. (2018). *Long-term performance and durability of masonry structures: Degradation mechanisms, health monitoring and service life design*. Woodhead Publishing.
- Hibbeler, R. C. (2004). *Engineering mechanics: Dynamics*. Pearson Educación.
- Inchbald, G. (1996). Five space-filling polyhedra. *The Mathematical Gazette*, 80(489), 466–475. <https://doi.org/10.2307/3618509>
- Jain, A. (2010). *Robot and multibody dynamics: Analysis and algorithms*. Springer Science & Business Media.
- Jing, L., & Stephansson, O. (2007). *Fundamentals of discrete element methods for rock engineering: Theory and applications*. Elsevier.
- Kim, J., You, S., Lee, S., Kamat, V., & Robert, L. (2015). Evaluation of human robot collaboration in masonry work using immersive virtual environments.
- Kim, N. H., Dong, T., Weinberg, D., & Dalidd, J. (2021). Generalized optimality criteria method for topology optimization. *Applied Sciences*, 11(7), 3175.
- Koga, J., Koga, J., & Homma, S. (2013). Checkerboard problem to topology optimization of continuum structures. *ArXiv*, [abs/1309.5677](https://arxiv.org/abs/1309.5677).

- Lemos, J. V. (2007). Discrete element modeling of masonry structures. *International Journal of Architectural Heritage*, 1(2), 190–213. <https://doi.org/10.1080/15583050601176868>
- Loeb, A. L. (1991). *Space structures*. Birkhäuser Boston. <https://doi.org/10.1007/978-1-4612-0437-4>
- Lommen, S., Schott, D., & Lodewijks, G. (2014). DEM speedup: Stiffness effects on behavior of bulk material. *Particuology*, 12, 107–112. <https://doi.org/10.1016/j.partic.2013.03.006>
- Luding, S. (2008). Introduction to discrete element methods. *European Journal of Environmental and Civil Engineering*, 12(7-8), 785–826. <https://doi.org/10.1080/19648189.2008.9693050>
- Marques, R., & Lourenço, P. B. (2014). Unreinforced and confined masonry buildings in seismic regions: Validation of macro-element models and cost analysis. *Engineering Structures*, 64, 52–67. <https://doi.org/10.1016/j.engstruct.2014.01.014>
- Ng, P. L., Lam, J. Y., & Kwan, A. K. (2010). Tension stiffening in concrete beams. part 1: Fe analysis. *Proceedings of the Institution of Civil Engineers-Structures and Buildings*, 163(1), 19–28.
- Nikishkov, G. (2004). Introduction to the finite element method. *University of Aizu*, 1–70.
- O’Shaughnessy, C., Masoero, E., & Gosling, P. D. (2022). Topology optimization using the discrete element method. part 1: Methodology, validation, and geometric nonlinearity. *Meccanica*, 57(6), 1213–1231. <https://doi.org/10.1007/s11012-022-01493-w>
- Pedersen, C. G., Lund, J. J., Damkilde, L., & Kristensen, A. S. A. (2006). Topology optimization-improved checker-board filtering with sharp contours. *Proceedings of the 19th Nordic Seminar on Computational Mechanics*, 182–185.
- Reddy, V. (n.d.). Bv and jagadish, ks (2003) embodied energy of common and alternative building materials and technologies. *Energy and Buildings*, 35(2), 129–137.
- Rojek, J. (2018). Contact modeling in the discrete element method. In *Contact modeling for solids and particles* (pp. 177–228). Springer International Publishing. https://doi.org/10.1007/978-3-319-90155-8_4
- Rozvany, G. (2000). The SIMP method in topology optimization - theoretical background, advantages and new applications. *8th Symposium on Multidisciplinary Analysis and Optimization*. <https://doi.org/10.2514/6.2000-4738>
- S. Takahashi. (n.d.). Sala de los abencerrajes. alhambra palase, granada. spain. 1354-91. <http://www.shiro1000.jp/muqarnas/data/034/34.htm>
- Sluys, L., & de Borst, R. (2001). Computational methods in non-linear solid mechanics. *TU Delft*, 21, 76.
- Svanberg, K. (1987). The method of moving asymptotes—a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2), 359–373.
- Talisch, C., Paulino, G. H., Le, C. H., Paulino, G. H., Pindera, M.-J., Dodds, R. H., Rochinha, F. A., Dave, E., & Chen, L. (2008). Topology optimization using wachspress-type interpolation with hexagonal elements. *AIP Conference Proceedings*. <https://doi.org/10.1063/1.2896796>
- Tonon, F. (2004). Explicit exact formulas for the 3-d tetrahedron inertia tensor in terms of its vertex coordinates. *Journal of Mathematics and Statistics*, 1(1), 8–11.
- Vincent Callebaut Architectures. (n.d.). The 5 farming bridges. https://vincent.callebaut.org/object/171023_mosul/mosul/projects